Research in Industrial Projects for Students

Sponsor

Analog Devices Inc.

Final Report

Development of system identification tools for precise control of motors

Student Members

Shreya Jha*, *Georgia Institute of Technology*,
sjha75@gatech.edu

Shaymaa Mahmoud*, *American University in Cairo*

Michelle Bang*, *Oregon State University*

Emre Isik*, *University of Cambridge*

Academic Mentor

Chunyang Liao, liaochunyang@math.ucla.edu

Sponsoring Mentor

Srilakshmi Pattabiraman, Srilakshmi.Pattabiraman@analog.com

Date: August 21, 2024

This project was jointly supported by Analog Devices Inc. and NSF Grant DMS 1925919.
All authors contributed equally to this work.

Abstract

Motor control is incredibly important in various applications, ranging from robotics to the automobile industry. Ensuring precision and accuracy of motors requires accurate system identification tools. System identification is the process of modelling the transfer function, which characterizes the dynamic behavior of a system and its response to input signals. The Vector Fitting (VF) Algorithm, an algorithm that iteratively adjusts the poles and residues of an approximated transfer function, is a prominent technique used in this regard, known to work well with exact data. However, when data is noisy, as is often the case in practical scenarios, VF runs into convergence and poor prediction issues. To address the prediction issues, we explore the use of neural networks as an alternative approach for approximating the target transfer function. Specifically, we investigate two types of neural networks: the widely used ReLU neural network and a rational neural network, which uses rational functions as a non-linear activation function. The motivation for using two different architectures is to compare the performance of a standard and a more complex activation function in handling noise in the data. We run numerical experiments to verify the effectiveness of these neural network models in approximating transfer functions. Our results show that the rational neural network provides a powerful tool for analyzing and reconstructing the transfer function mathematically, which is not possible with a standard deep neural network. Additionally, the rational neural network has fewer parameters to train and achieves similar or higher accuracy scores to the ReLU deep neural network, despite being mathematically less complex. This combination of interpretability and efficiency makes the rational neural network a possible candidate for future studies on system identification tools.

Acknowledgments

We would like to express our sincere gratitude to our industry sponsor, Analog Devices Inc, our mentor, Srilakshmi Pattabiraman, and the Trinamic team for their generous support in providing the motors that enabled us to work with real data and their guidance throughout the process. Their valuable feedback and resources significantly enhanced our understanding of the material and contributed to the success of our project.

We would also like to thank the Institute of Pure and Applied Mathematics for hosting the Research in Industrial Projects for Students (RIPS) program, which provided us with the unique opportunity to engage in this research. Their commitment to fostering collaboration between students and industry is invaluable.

Additionally, we would like to thank our academic mentor, Chunyang Liao, for his guidance, advice, and innovative ideas throughout this project. His expertise and encouragement have been instrumental in shaping our research journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This project is proposed by Analog Devices, Inc. (ADI), which is a global semiconductor leader bridging the physical and digital worlds to enable breakthroughs at the Intelligent Edge. ADI combines analog, digital, and software technologies into solutions that help drive advancements in digitized factories, mobility, digital healthcare, combating climate change, and reliably connect humans and the world.

1.1 The Proposed Problem

Electric motors have many industrial applications such as assembly lines, electric vehicles, drones, 3D printers and robotic arms. In many of these industrial applications, precision in motors' position and trajectory control is crucial to perform complicated tasks reliably and efficiently. One important step to improving accuracy in motor control is through constructing the input-output relationship of the system. Having an accurate transfer function that describes this relationship means that the output of a system can be controlled precisely with a given input.

The conventional method of computing the transfer function of a system is by solving the underlying physics equations. However, as the complexity of a system increases, computing the transfer function through complex analysis techniques for a physical state becomes a laborious exercise which can also get computationally heavy.

The alternative method is based on data-driven transfer function approximation. In simple terms, subjecting a component to sufficiently diverse set of input signals and analyzing the corresponding outputs, we can sometimes reconstruct the system's input-output relationship. Vector Fitting (VF) is a powerful data-driven algorithm which is provably guaranteed to model the transfer function of a system exactly given that the data is clean, linear and error-free. With noise, VF will try to model the noise in the data, which reduces the accuracy of the resulting model. In practice, one rarely encounters a true linear system, and data is never noise-free.

One of our goals of this project is to design a data-driven model to approximate the transfer functions. We look at three main transfer functions in this project: current open loop, current closed loop, and velocity loop, which are depicted in Figure 1.1, 1.2, and 1.3, respectively. The theoretical transfer functions, which are assumed to be the true target, are derived from the physics based first-principles method from the stepper motor outputs.

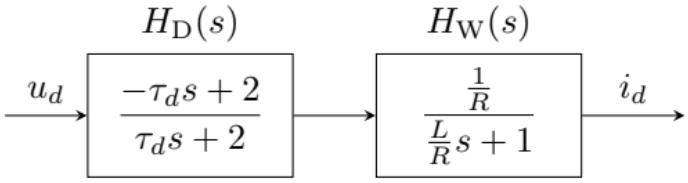


Figure 1.1: Current open loop

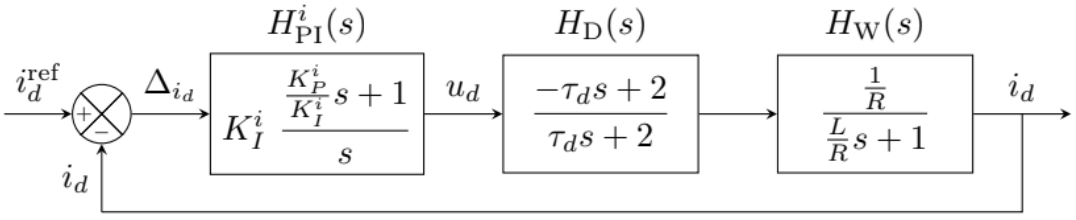


Figure 1.2: Current closed loop

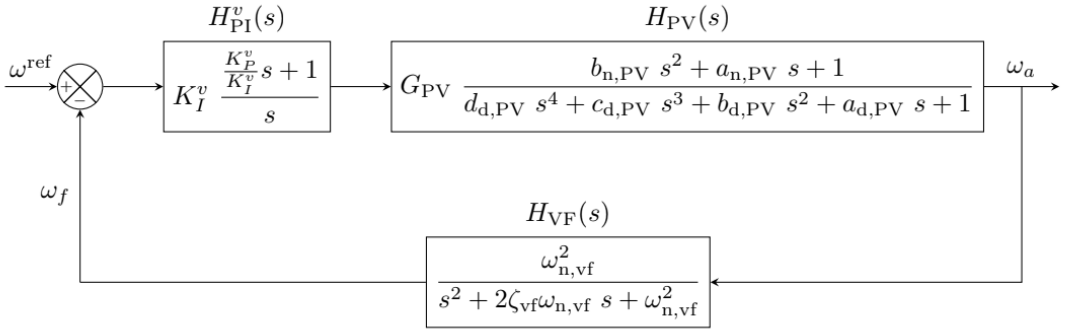


Figure 1.3: Velocity loop

The transfer functions for the three different loops are defined as follows. Note that the order represents the degree of the rational function.

1. Current open loop (order 2):
$$H_D(s) \cdot H_W(s)$$

2. Current closed loop (order 3):
$$\frac{H^i_{PI}(s) \cdot H_D(s) \cdot H_W(s)}{1 + H^i_{PI}(s) \cdot H_D(s) \cdot H_W(s)}$$

3. Velocity loop (order 7):
$$\frac{H^v_{PI}(s) \cdot H_{PV}(s)}{1 + H^v_{PI}(s) \cdot H_{PV}(s) \cdot H_{VF}(s)}$$

For the current loop, $H_D(s)$, $H_W(s)$, and $H_{PI}^i(s)$ are the transfer functions for delay in the power stage, winding in the motor, and Proportional Integral (PI) controller, respectively. For the velocity loop, $H_{PI}^v(s)$, $H_{PV}(s)$, and $H_{VF}(s)$ are the transfer functions for the PI controller, plant and sensor, respectively.

1.2 Our Team's Approach

A neural network is a machine learning process that has been found to be robust to noise. Thus, we looked into three different neural network architectures as an alternative system identification tool to VF.

The first neural network we looked into incorporates the Rectified Linear Unit, ReLU, activation function as defined below.

$$\text{ReLU}(x) = \max(0, x)$$

ReLU is a commonly used activation function in neural networks as it is computationally fast to evaluate [1]. Compared to the other two neural networks, the ReLU neural network is composed of multiple layers and nodes, thus, it can be classified under deep neural networks.

The next two architectures have a rational function as its activation function. A rational function is defined in the following form:

$$\text{Rational}(x) = \frac{\sum_{i=0}^{P} a_i x^i}{\sum_{j=0}^{Q} b_j x^j}$$

where P is the order of the numerator polynomial, Q is the order of the denominator polynomial, and $\{a_i\}_{i=0}^{P}$ and $\{b_j\}_{j=0}^{Q}$ are trainable coefficients. In theory, given an error threshold ϵ, rational neural networks require fewer parameter for training compared to ReLU networks [1].

The first rational neural network architecture we proposed has a singular layer and multiple nodes based off the the order, also known as the degree, of the transfer function we were trying to approximate. The complete structure of the rational architecture 1 can be found in Figure 3.2.

The second rational neural network architecture we proposed has a singular layer and a singular node based off the order of the transfer function we were trying to approximate. The complete structure of the rational architecture 1 can be found in Figure 3.3.

In all three structures, our inputs are complex frequencies and the outputs are the evaluated transfer function values, which are also complex. While there has been previous research on complex ReLU neural networks [9], [4], no articles could be found on the applications of complex rational neural networks. Thus, our approach brings in new insight into the applicability of rational neural networks when working with complex values.

1.3 Overview of the Report

Our report is organized as follows. In Chapter 2, we introduce the VF algorithm and test the algorithm with real, simulated and synthetic data. Then, we dive into neural networks and discuss our proposed models using both ReLU and rational activation functions in Chapter 3. We conclude our report in Chapter 4 by summarizing our overall results and recommendations for future directions.

Chapter 2

Vector Fitting and its Stress Regions

In this chapter, we will do the following. In section 2.1, we cover the VF algorithm. In section 2.2, we examine the results of VF on synthetic data. In section 2.3, we describe the metric used to evaluate VF performance. In section 2.4, we further explore VF performance on simulated data. In section 2.5, we cover VF trained on real data collected from motors provided by Analog Devices, Inc.

2.1 The Vector Fitting Algorithm

Transfer functions (TFs) are a vital aspect of this project, and are central to analyzing and designing linear time-invariant systems, which are common in a variety of engineering disciplines, including control systems and signal processing. A linear time-invariant (LTI) system can be written as

$$y(t) = \int_{-\infty}^{\infty} h(\alpha)x(t-\alpha)\,d\alpha = \int_{-\infty}^{\infty} x(\alpha)h(t-\alpha)\,d\alpha. \tag{2.1}$$

where $x(t)$ is the input signal to the system and $y(t)$ is the output. The integral operation in equation (2.1) is called the convolution of impulse response h and the input signal x. Taking Laplace transform for both sides of equation (2.1) leads to

$$Y(s) = H(s)X(s) \Longrightarrow H(s) = \frac{Y(s)}{X(s)} \tag{2.2}$$

where $Y(s)$ is the Laplace transform of the output signal, $X(s)$ is the Laplace transform of the input signal and $H(s)$ is the transfer function of LTI system defined in the s-domain. Notice that the transfer function $H(s)$ can be written as a ratio of $Y(s)$ and $X(s)$, see equation (2.2), it is natural to use a rational function to approximate the true transfer function. Mathematically, a rational function takes the form:

$$\hat{H}(s) = \frac{n(s)}{d(s)} = \frac{\sum_{n=0}^{\bar{n}} a_n s^n}{\sum_{n=0}^{\bar{n}} b_n s^n}$$

where numerator $n(s)$ and denominator $d(s)$ are polynomials of s. Moreover, rational function is a universal approximator, i.e. it can approximate any continuous function. The coefficients are trainable parameters. Thus, approximating the true transfer functions

involves finding the most accurate coefficients. Algorithms, such as the Sanathanan-Koerner algorithm, that aim to fit data to the $\hat{H}$ equation directly suffer from numerical due to the ill-conditioning of the s^n terms. VF solves this problem as it replaces the numerator and denominator of $\hat{H}(s)$ with partial fractions [10], and thus approximator $\hat{H}(s)$ can be written as

$$\hat{H}^{(i)}(s) = r_0 + \sum_{n=1}^{\bar{n}} \frac{r_n^{(i)}}{s - p_n^{(i-1)}}$$

where $\{p_n\}_{n=1}^{\bar{n}}$ are poles and $\{r_n\}_{n=1}^{\bar{n}}$ are residues.

When solving for the poles and residues, the VF algorithm begins by making an initial guess of the transfer function's poles and fixes the unknown coefficients in the denominator. Through iterations, it adjusts the poles and zeros to minimize the difference between the actual output data and output. Linear least squares regression is used to approximate the system's transfer function. In short, the VF algorithm first fixes the coefficients in the denominator and then minimizes the coefficients in the numerator at each iteration.

For noise-free data, VF works efficiently with great accuracy. Researchers found that the model convergence happened within 4 or 5 iterations with a model error below 10^{-8} [10].

2.1.1 Implementation of VF in Python

The VF algorithm was implemented in Python by using the reference code provided in [10] as a reference.

2.2 Testing VF on Synthetic Data

This section will contain the results of VF tested on synthetically generated data. There were three different transfer functions that we wanted VF to approximate:

- First order TF : $H_W(s)$
- Open loop including controller TF : $H_{PI}(s) \cdot H_D(s) \cdot H_W(s)$
- Closed loop TF : $H_{CL}(s) = \frac{H_{OL}(s)}{1+H_{OL}(s)}$

These transfer functions are generated by directly computing the products of each component.

The Bode Plots for each of these transfer functions were as follows:

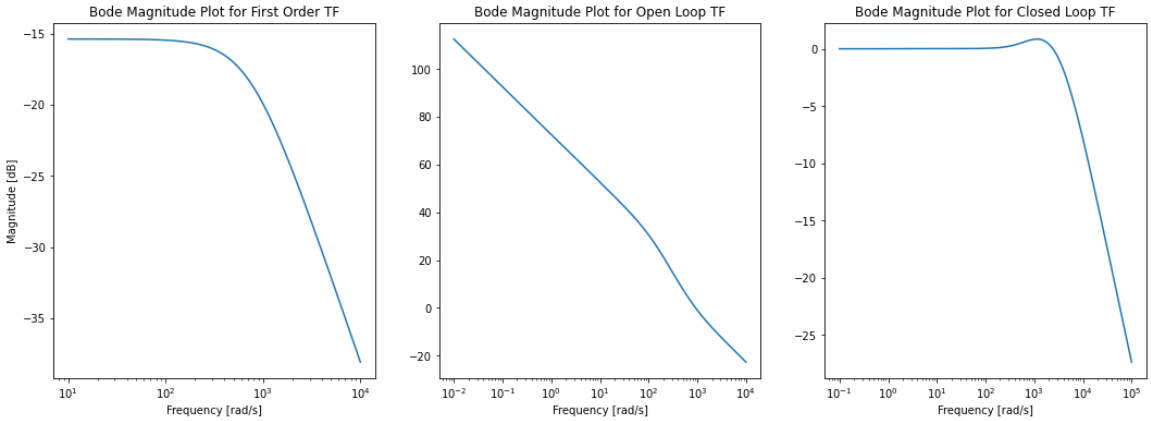


Figure 2.1: Magnitude bode plots of the first order, open loop, and closed loop transfer functions.

We choose the set of frequencies and responses to train the VF algorithm. For the first order transfer function, we let the training frequencies be 100 points between 10^1 and 10^4 in the log scale. For the open loop transfer function, we let the training frequencies be 100 points between 10^{-2} and 10^4. For the closed loop transfer function, we let the training frequencies be 100 points between 10^{-1} and 10^4, spaced apart logarithmically. In the next two subsections, we test the VF algorithm on clean synthetic data, meaning the accurate values of the transfer function at certain frequencies, and noisy synthetic data, respectively.

2.2.1 Clean Data

In this section, we tested the performance of VF on clean training data which contains no noise or error, see Figure 2.2. With clean training data, VF algorithm perfectly captures all true transfer functions, as shown in the plots.

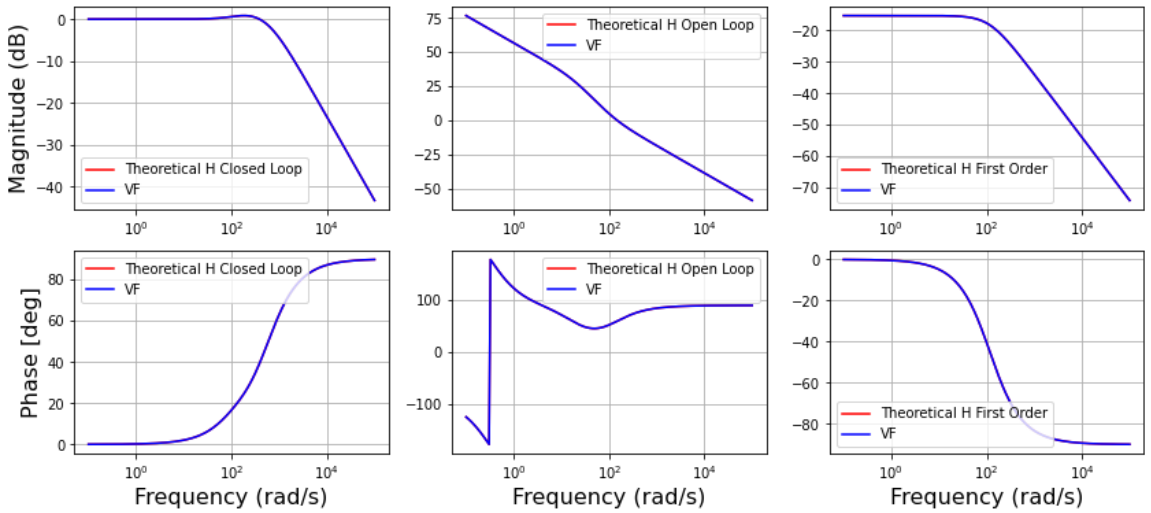


Figure 2.2: Magnitude and phase bode plots of closed loop, open loop, and first order transfer functions, with the VF approximation.

2.2.2 Noisy Data

The VF algorithm does not work well if the training samples are noisy. In this section, we want to verify this phenomenon. We added random Gaussian noise to the frequency responses depending on a given Signal to Noise Ratio (SNR). Given the input frequency responses, noise was added as followed:

$$\text{SNR} = 10\log(\frac{P_{\text{signal}}}{P_{\text{noise}}}) \tag{2.3}$$

$$P_{\text{noise}} = \frac{P_{\text{signal}}}{10^{\frac{\text{SNR}}{10}}} \tag{2.4}$$

Here, P_{signal} is the magnitude of H at a given complex frequency. Noise is added to both the magnitude and the phase of H for each frequency.

P_{noise} represents the variance of the Gaussian distribution that was drawn from, with mean 0, and added to the frequency responses. SNR is another input representing the level of noise.

For each transfer function, we added three different levels of noise to the data: SNR of 30 dB, 20 dB, and 10 dB. The smaller the SNR value, the larger the noise level. VF was then trained on this noisy data, and tested on another set of frequencies. The model was then plotted with the ground truth model for comparison. Figures 2.3, 2.4, and 2.5 illustrate the performance of VF when trained on data of noise level 10 dB SNR. See the appendix for the other noise levels. As depicted in the plots, when noise is introduced, VF no longer captures the true transfer function.

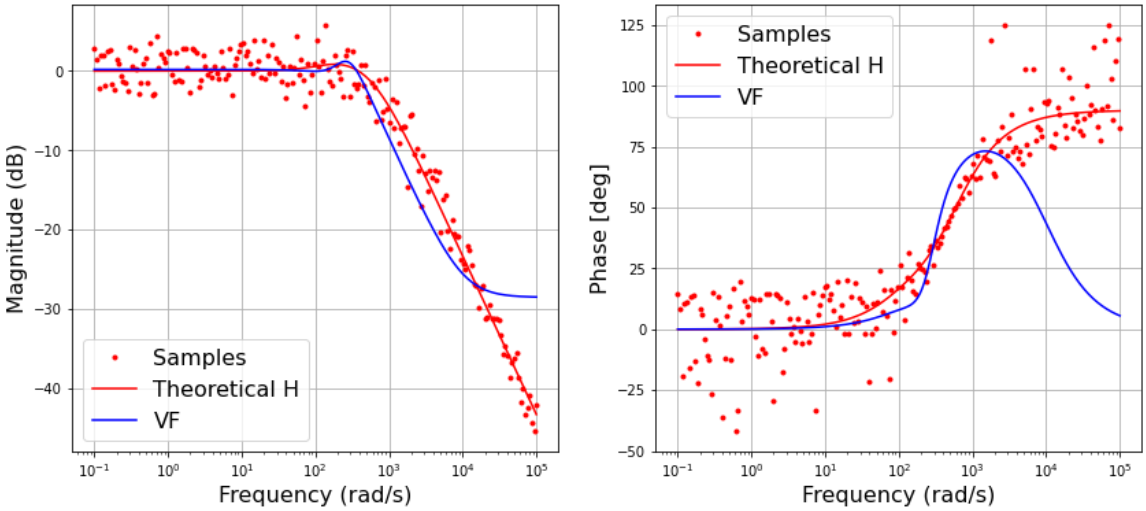


Figure 2.3: Closed Loop transfer function bode plot with VF model trained on 10 SNR data.

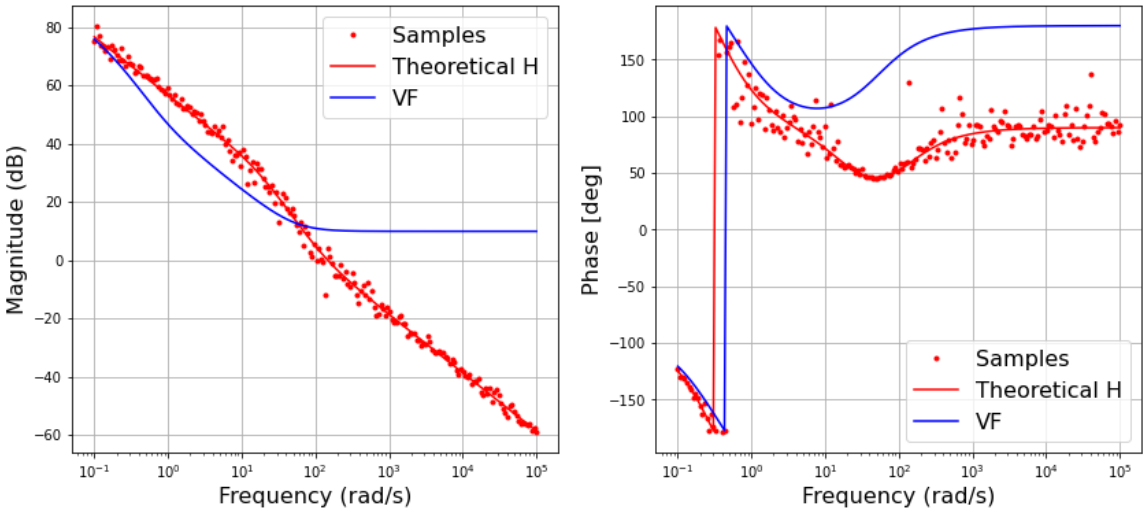


Figure 2.4: Open Loop transfer function bode plot with VF model trained on 10 SNR data.

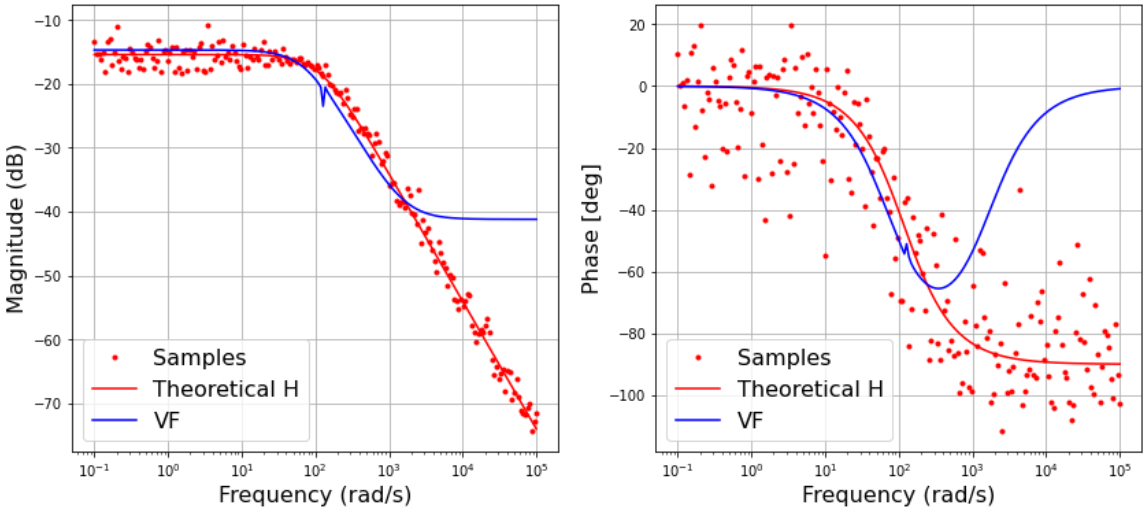


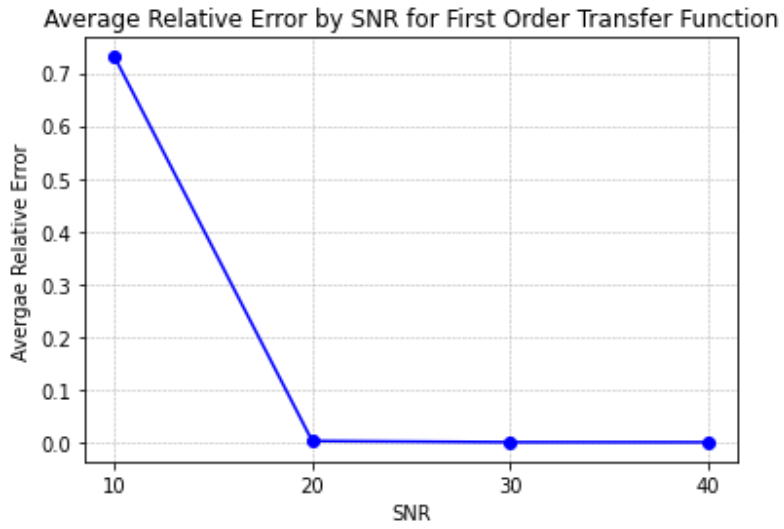
Figure 2.5: First order transfer function bode plot with VF model trained on 10 SNR data.

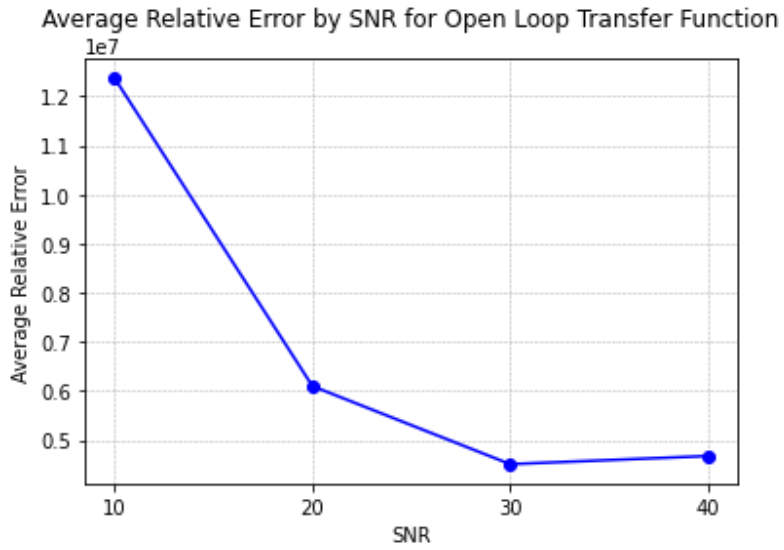
2.3 Metric Development

In order to evaluate the performance of VF algorithm, in general every transfer function approximation method, we looked into average relative error, which is defined as

$$\frac{1}{K}\sum_{i=1}^{K}\frac{|\hat{H}(s_i)-H(s_i)|^2}{|H(s_i)|^2},$$

where $H(s_i)$ and $\hat{H}(s_i)$ are the true function value and the approximated function value at input s_i. We have chosen to evaluate the performance of the VF model, as well as the neural network models, since this metric incorporates scale and is not sensitive to larger magnitudes. In Figure 2.6, we vary the noise levels to see the average relative error of the VF algorithm. We test three different transfer function. In each example, we observe that higher levels of noise result in worse performances by the VF algorithm.





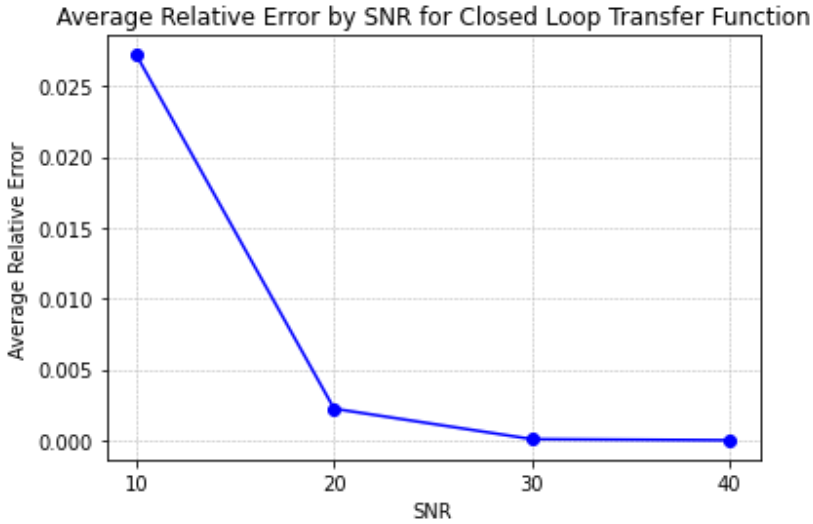


Figure 2.6: Average relative error by SNR for a first order TF, an open loop TF, and a closed loop TF.

2.4 Testing Vector Fitting on Simulated Data

In this section, we present the numerical experiments of testing the VF algorithm on simulated data. We looked at the three main transfer functions: current open loop, current closed loop, and velocity loop, which were depicted in Figure 1.1, 1.2, and 1.3, respectively.

Simulated data was generated using the *lsim* function in MATLAB. The *lsim* function takes in a transfer function, input signals, and time samples and returns the simulated response data. Compared to synthetic data, which was generated in the frequency domain directly, simulated data was generated in the time domain to mimic the real data conditions.

As VF takes in inputs from the frequency domain, a Discrete Fourier Transform (DFT) was performed on the dataset. We use the built-in fft function in the SciPy library in Python, which implements a fast algorithm to compute Discrete Fourier Transform [2]. Thus, the inputs were sine waves that had full periods captured in exactly 512 sample points with a sampling rate of $4 \cdot 10^{-5}$ seconds. A full period was necessary for Fast Fourier Transform (FFT) to accurately convert the time domain signals into the appropriate frequency domain equivalent. The general equation for a DFT is as follows

$$y[k] = \sum_{n=0}^{N-1} e^{-2\pi j \frac{kn}{N}} x[n].$$

To control the period of our data, we used the following equation:

$$\omega = \frac{25000n}{512}$$

Here, ω represents the frequency of the wave and n is an integer. The amplitudes of the waves were precisely chosen so that it would be around 90 percent of the voltage and current threshold. Therefore, the general steps of the VF pipeline is as follows.

1. Generate simulated data using the *lsim* function in MATLAB.
2. Convert time-domain signals into frequency-domain using the FFT function in Python.
3. Input the frequency-domain signals into VF algorithm implemented in Python.

The frequencies of the sine waves ranged from as small as 48 Hz to as large as 10000 Hz. Due to the Nyquist–Shannon sampling theorem, which states that frequencies larger than X Hz for a sampling rate of $\frac{1}{2X}$ are prone to data distortion, frequencies larger than 12500 Hz were not looked at [7].

We also look at noisy simulated data. Noise was added point-wise to only the output response of the simulation. The noise added was randomly drawn from a normal distribution. The method for adding noise was similar to the method described in equation 2.4.

2.4.1 Current Open Loop

In this subsection, we tested the current open loop. We considered the clean simulated data case as well as the noisy data case. In each figure, the red curve and the blue curve represented the theoretical transfer function and its approximation using the VF algorithm, respectively. We also included training samples (in black) and real points (red dots) in each plot. From the tables, we documented the poles convergence error for each iteration and the final model error. The first average relative error represent the error tested between the same sample points and real points. The second average relative error represent the error tested between randomly selected points and the theoretical model.

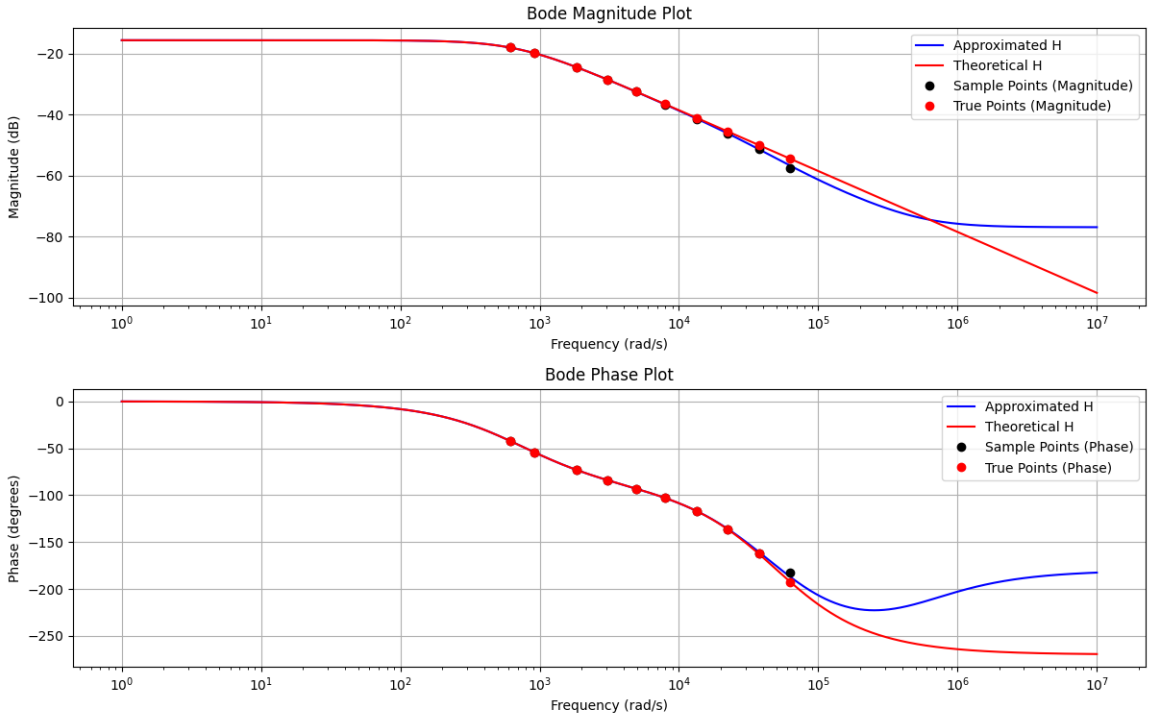
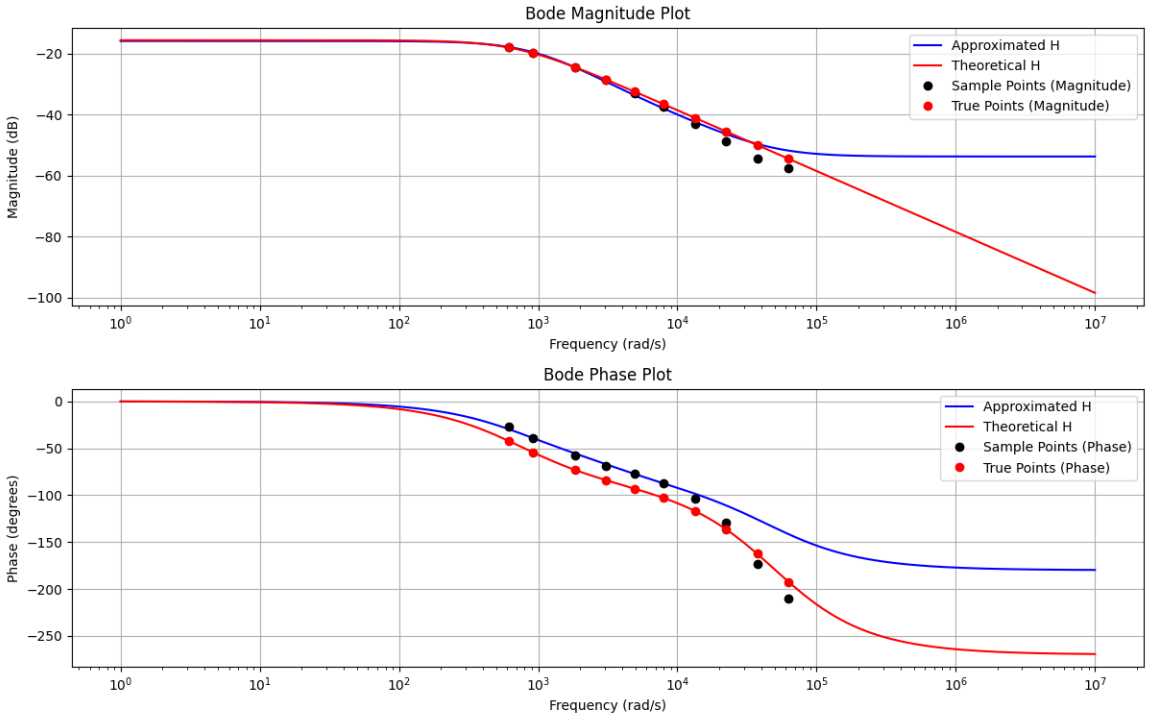


Figure 2.7: Bode plot for open loop model trained on 10 simulated sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.22e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (8.25e-02)
Final Model Error	6.60e-05
Average Relative Error	8.79e-03
Average Relative Error (random points)	3.09e-3

Table 2.1: Table of VF outputs for open loop model trained on 10 simulated sine waves.

The 10 frequencies inputted into the VF algorithm were those for Test 1 noted in Table A.1. From the plots, we observe that the VF model matches up with the theoretical TF for smaller frequencies and a visible difference is present for the larger two frequencies. Two iterations were needed for model convergence.



Iteration n.1	
Convergence Test (poles estimation)	failed (1.11e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (9.15e-01)
Final Model Error	2.61e-03
Average Relative Error	2.32e-01
Average Relative Error (random points)	9.59e-02

Table 2.2: Table of VF outputs for open loop model trained on 10 simulated sine waves not in steady state.

The 10 frequencies inputted into the VF algorithm were those for Test 1 noted in Table A.1. The main difference between our first plots and the second plots is that the first 512 points of the simulation were used for FFT for the second plot and the last 512 points of the simulation were used for the first plot. The data used here was not in a steady state, resulting in numerical differences between the true points and the sample points from the simulation. The plots clearly demonstrated this difference as we can see that a phase shift of at least 20 degrees is present. Two iterations were needed for model convergence, but the average relative error is about 10 times worse compared to the first test.

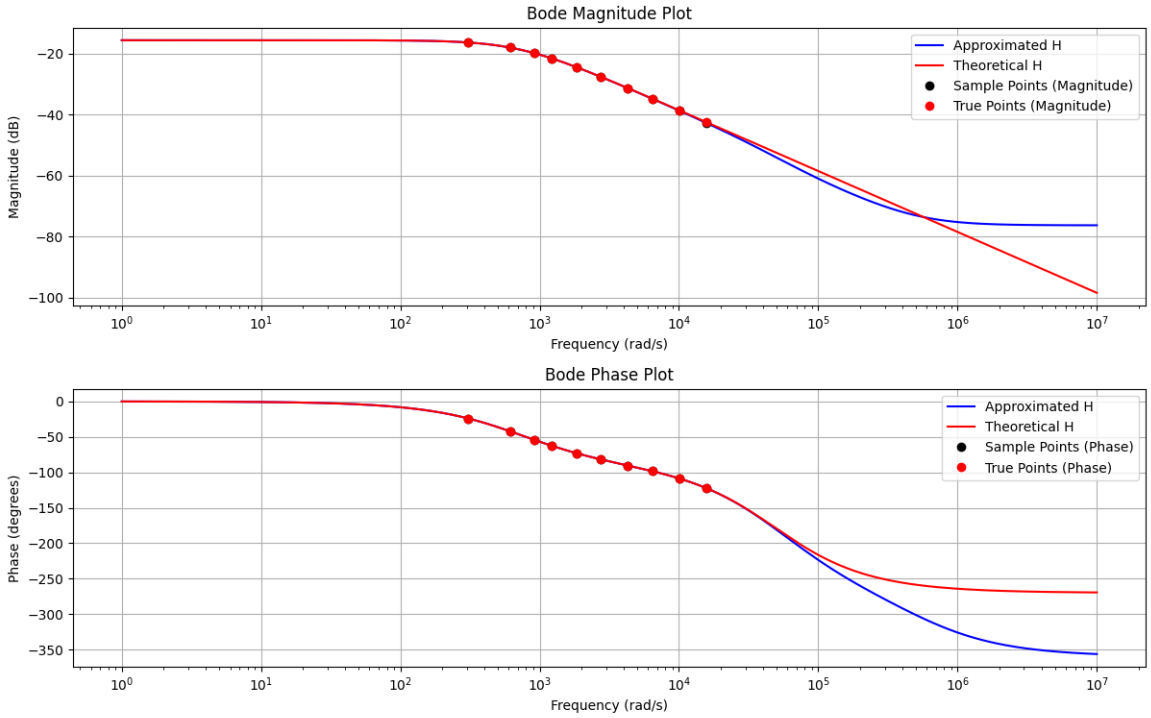


Figure 2.8: Bode plot for open loop model trained on 10 simulated sine waves with frequencies ranging from 48 Hz to 2490 Hz.

Iteration n.1	
Convergence Test (poles estimation)	failed (2.20e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (1.52e-03)
Final Model Error	1.86e-06
Average Relative Error	1.16e-04
Average Relative Error (random points)	6.16e-05

Table 2.3: Table of VF outputs for open loop model trained on 10 simulated sine waves with frequencies ranging from 48 Hz to 2490 Hz.

A smaller range of frequencies were fed into the VF algorithm. The 10 frequencies inputted into the VF algorithm were those for Test 2 noted in Table A.1. Two iterations were needed for model convergence, and the final model error was 10 times smaller than the final model error for the VF model that was trained with a larger range of frequencies. From the plots, we observed that all the sample points followed the true points very closely with no identifiable differences.

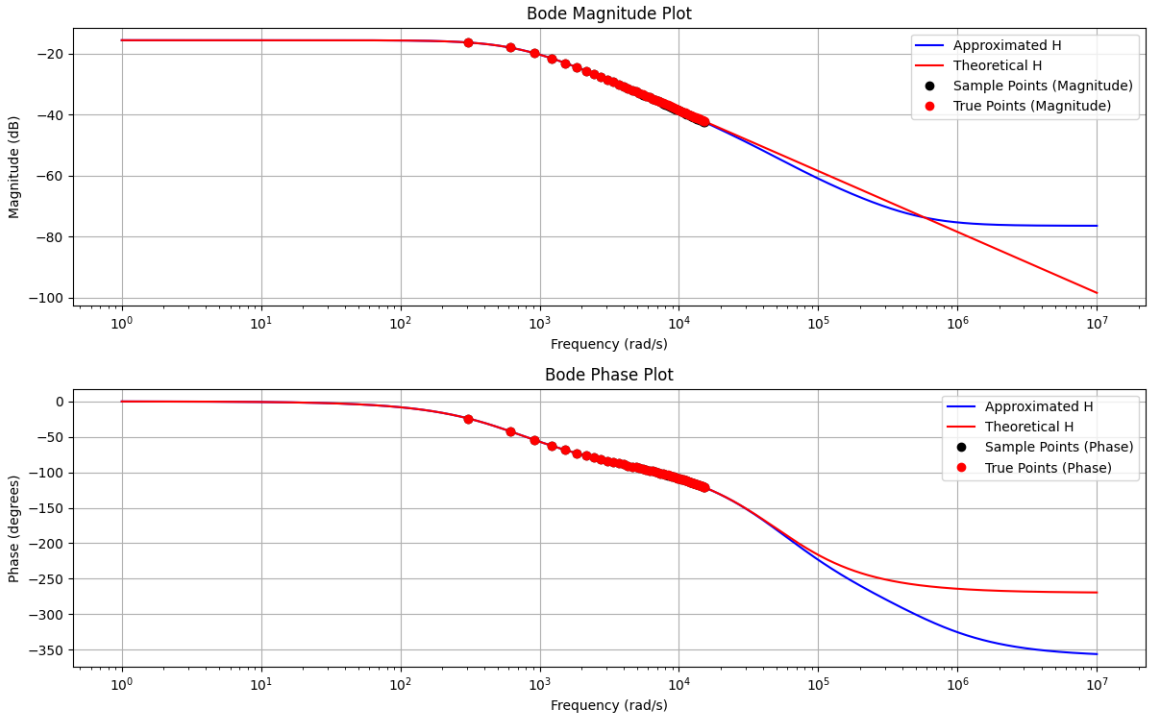


Figure 2.9: Bode plot for open loop model trained on 50 simulated sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (3.17e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (3.04e-04)
Final Model Error	2.53e-06
Average Relative Error	1.92e-04
Average Relative Error (random points)	7.54e-05

Table 2.4: Table of VF outputs for open loop model trained on 50 simulated sine waves.

The 50 frequencies inputted into the VF algorithm were those for Test 3 noted in Table A.1. Two iterations were needed for model convergence. From the plots, we observed that VF was approximating the theoretical H very closely within the range of 10 and 10^5 rad/s.

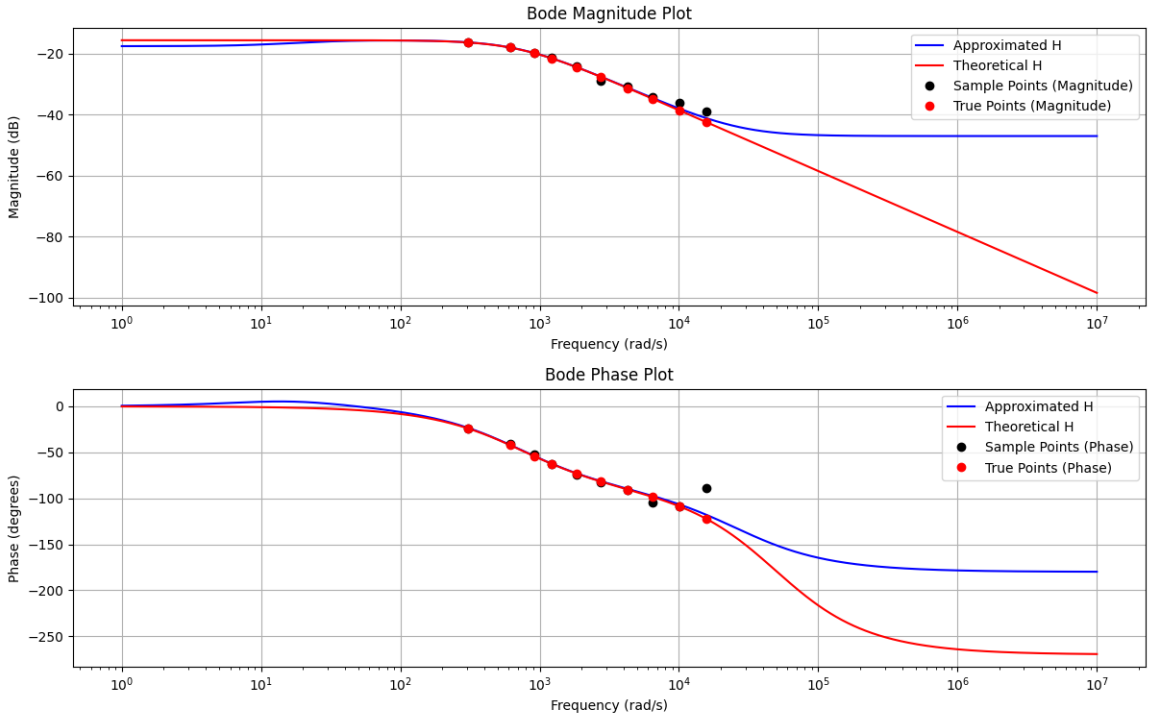


Figure 2.10: Bode plot for open loop model trained on 10 simulated sine waves with a noise level of 10 SNR.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.03e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (8.37e-01)
Final Model Error	3.33e-03
Average Relative Error	4.12e-03
Average Relative Error (random points)	4.80e-03

Table 2.5: Table of VF outputs for open loop model trained on 10 simulated sine waves with a noise level of 10 SNR.

The 10 frequencies inputted into the VF algorithm were those for Test 2 noted in Table A.1. Two iterations were needed for model convergence, however, the average relative error was about 67 times worse than the results for its no noise equivalent. From the plots, we observed that noise was more evident for the larger frequencies, and the overall VF model diverges from the theoretical model for both the smaller and larger frequency ranges.

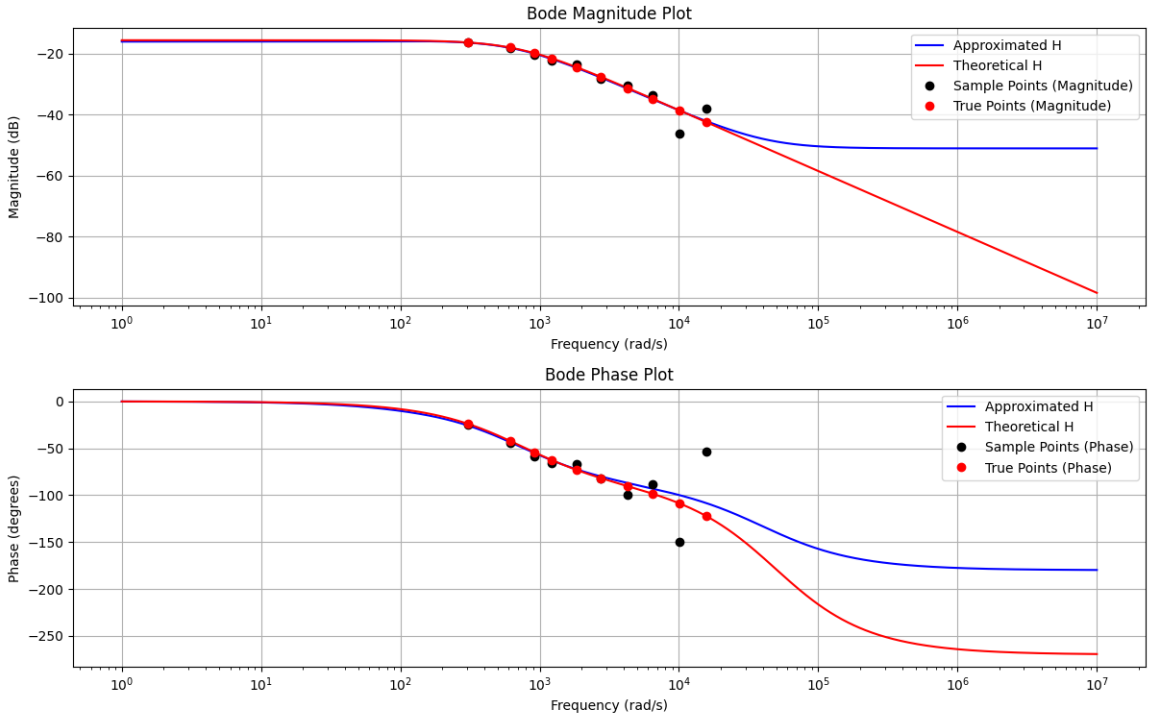


Figure 2.11: Bode plot for open loop model trained on 10 simulated sine waves with a noise level of 5 SNR.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.09e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (7.94e-01)
Final Model Error	6.69e-03
Average Relative Error	1.03e-02
Average Relative Error (random points)	7.38e-03

Table 2.6: Table of VF outputs for open loop model trained on 10 simulated sine waves with a noise level of 5 SNR.

The 10 frequencies inputted into the VF algorithm were those for Test 2 noted in Table A.1. Two iterations were needed for model convergence, however, the average relative error was about 117 times worse than the results for its no noise equivalent. With a smaller SNR ratio, we observed from the plots that the sample points were deviating substantially from the true points compared to the previous plot with a SNR ratio of 10.

2.4.2 Current Closed Loop

In this subsection, we document the results of the closed open loop.

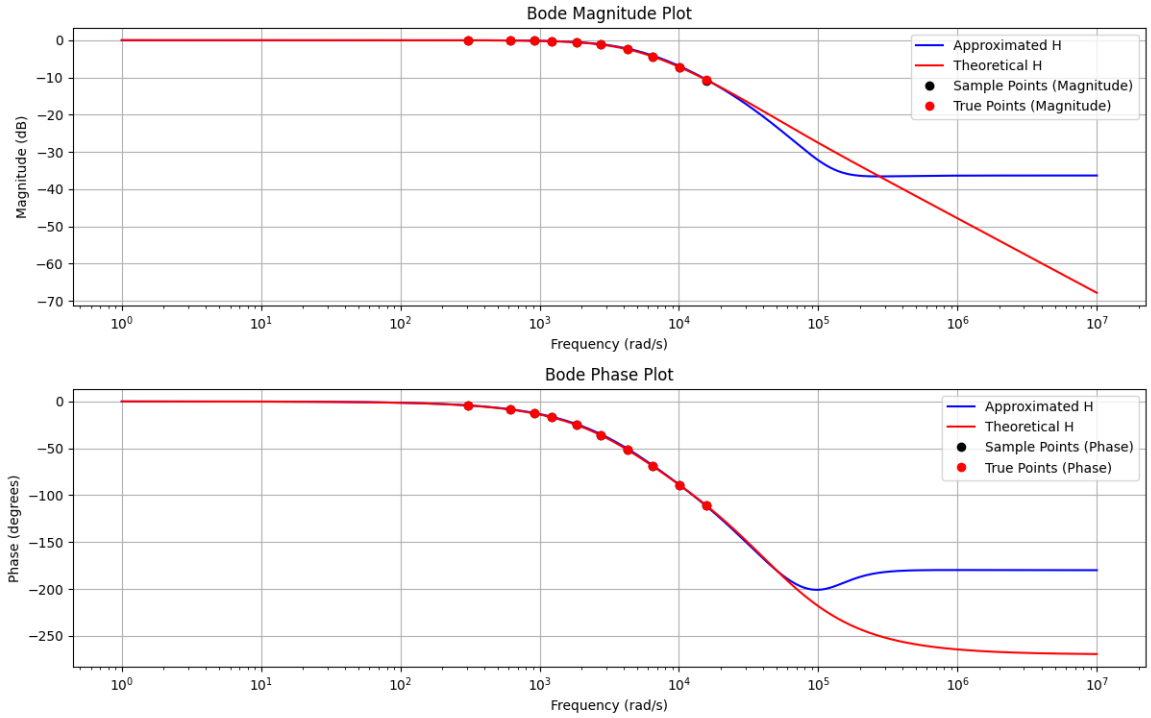


Figure 2.12: Bode plot for closed loop model trained on 10 simulated sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.46e+02)
Iteration n.2	
Convergence Test (poles estimation)	passed (2.10e-03)
Final Model Error	2.36e-07
Average Relative Error	1.20e-04
Average Relative Error (random points)	6.15e-05

Table 2.7: Table of VF outputs for closed loop model trained on 10 simulated sine waves.

The 10 frequencies inputted into the VF algorithm were those for Test 2 noted in Table A.1. Two iterations were needed for model convergence. From the plots, we observed that the VF model matched the theoretical model very closely and the sample points lied on the theoretical model.

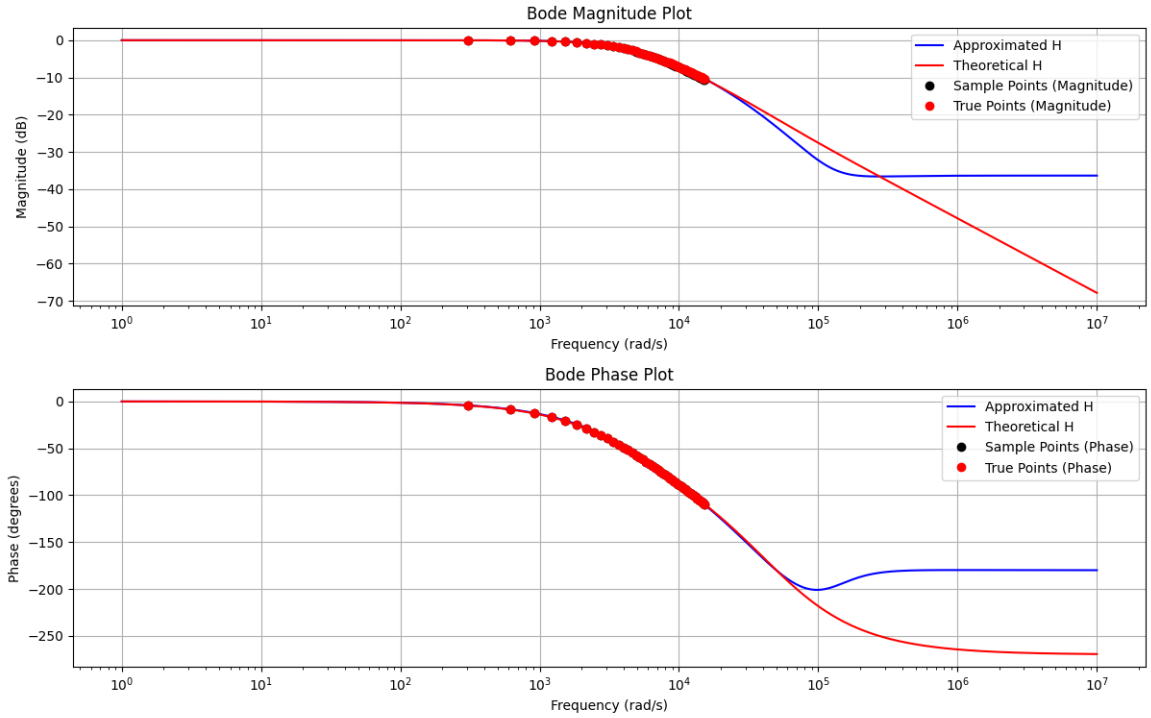


Figure 2.13: Bode plot for closed loop model trained on 50 simulated sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (7.44e+01)
Iteration n.2	
Convergence Test (poles estimation)	passed (3.96e-03)
Final Model Error	3.45e-07
Average Relative Error	1.95e-04
Average Relative Error (random points)	6.14e-05

Table 2.8: Table of VF outputs for closed loop model trained on 50 simulated sine waves.

The 50 frequencies inputted into the VF algorithm were those for Test 3 noted in Table A.1. Two iterations were needed for model convergence. From the plots, we observed that the VF model matched the theoretical model almost exactly in terms of both magnitude and phase.

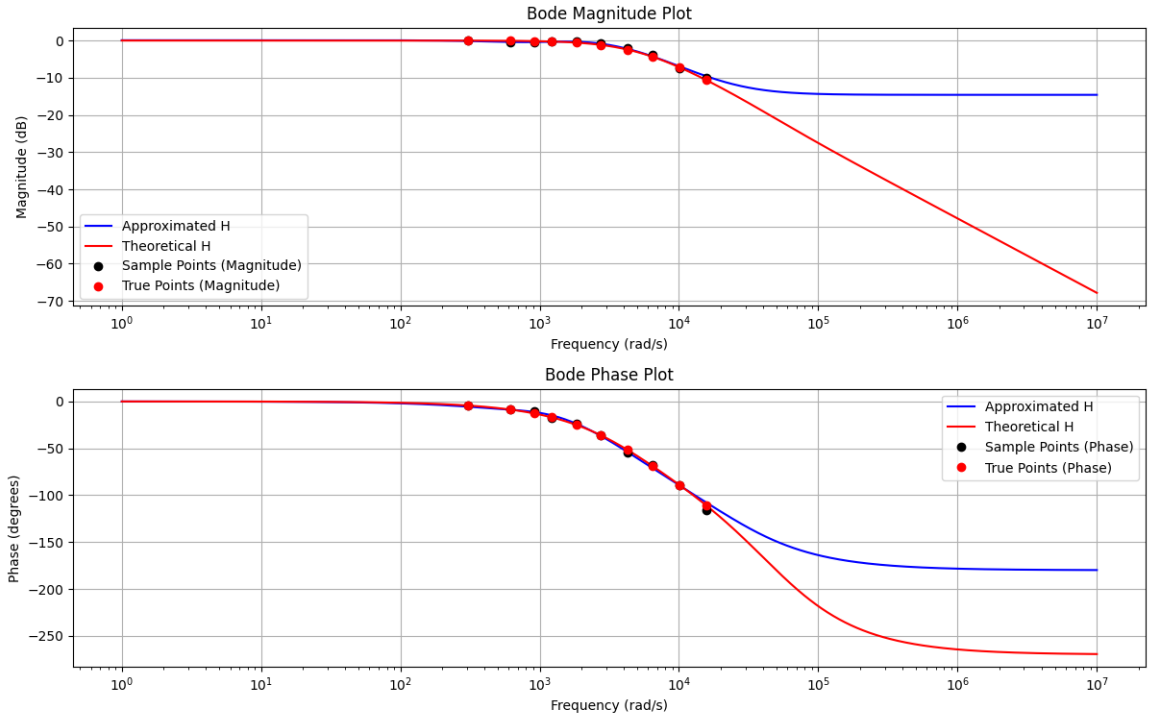


Figure 2.14: Bode plot for closed loop model trained on 10 simulated sine waves with a noise level of 10 SNR.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.11e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (4.42e-01)
Final Model Error	2.85e-02
Average Relative Error	3.25e-03
Average Relative Error (random points)	2.41e-03

Table 2.9: Table of VF outputs for closed loop model trained on 10 simulated sine waves with a noise level of 10 SNR.

The 10 frequencies inputted into the VF algorithm were those for Test 2 noted in Table A.1. Two iterations were needed for model convergence, however, the average relative error was about 33 times worse than the results for its no noise equivalent. From the plots, we observed that the sample points were no longer directly in line with our true points.

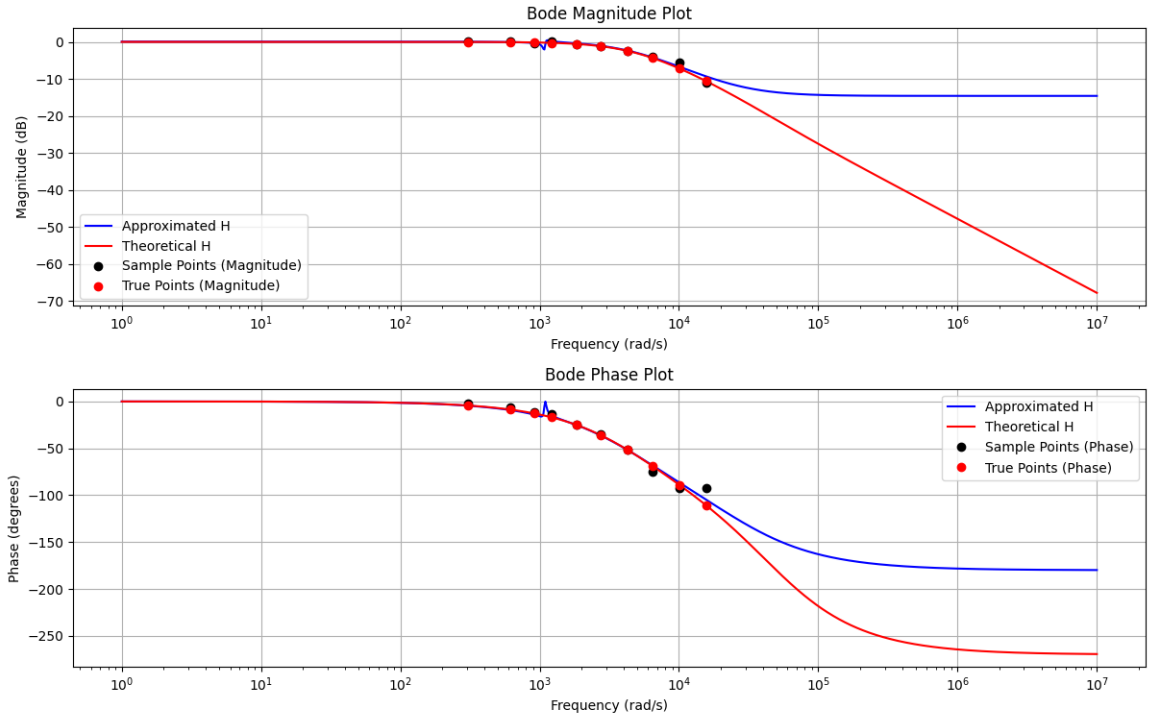


Figure 2.15: Bode plot for closed loop model trained on 10 simulated sine waves with a noise level of 5 SNR.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.08e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (8.54e-01)
Final Model Error	5.59e-02
Average Relative Error	4.98e-03
Average Relative Error (random points)	3.53e-03

Table 2.10: Table of VF outputs for closed loop model trained on 10 simulated sine waves with a noise level of 5 SNR.

The 10 frequencies inputted into the VF algorithm were those for Test 2 noted in Table A.1. Two iterations were needed for model convergence, however, the average relative error was about 50 times worse than the results for its no noise equivalent. Similar to before, we observed from our plots that the sample points were drifting away from the true points. In both the magnitude and phase plot, we saw a small peak around 10^3 rad/s.

2.4.3 Velocity Loop

In this subsection, we include results on the velocity loop.

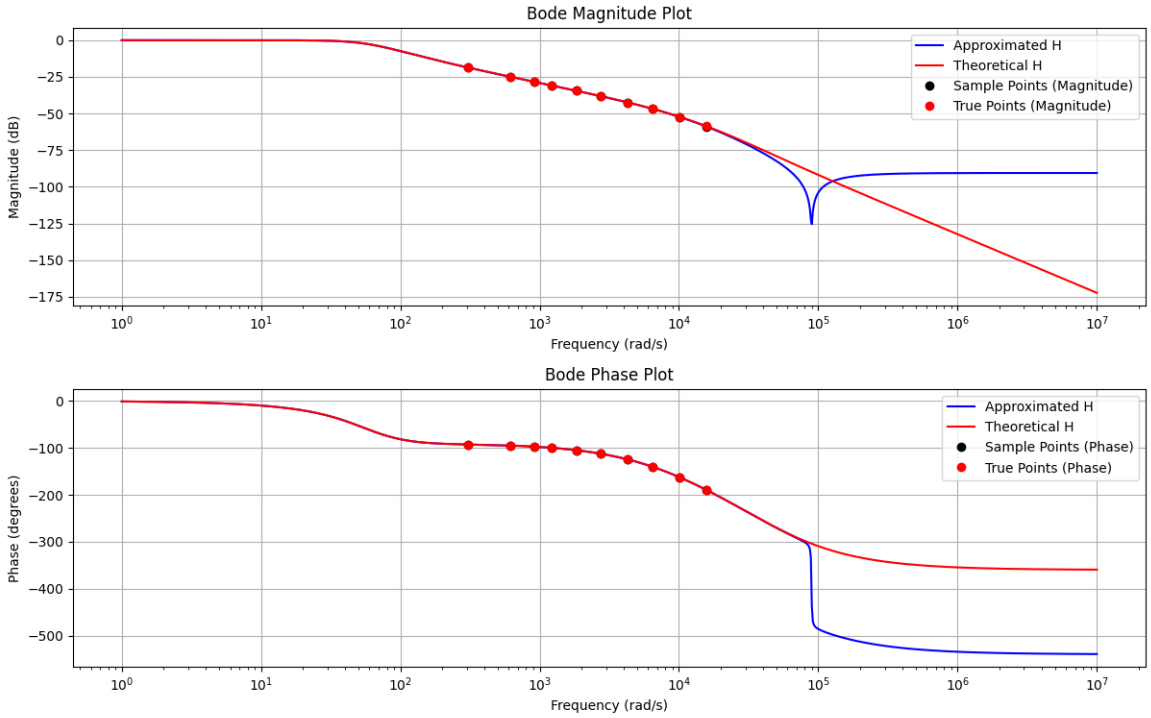


Figure 2.16: Bode plot for velocity loop model trained on 10 simulated sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.36e+02)
Iteration n.2	
Convergence Test (poles estimation)	passed (7.42e-02)
Final Model Error	1.82e-10
Average Relative Error	1.27e-04
Average Relative Error (random points)	2.78e-08

Table 2.11: Table of VF outputs for velocity loop model trained on 10 simulated sine waves.

The 10 frequencies inputted into the VF algorithm were those for Test 2 noted in Table A.1. Two iterations were needed for model convergence. From the plots, we saw that the VF model captures the theoretical TF for the frequency range we were interested in. The average relative error represents this accuracy as the error was 2.78e-08.

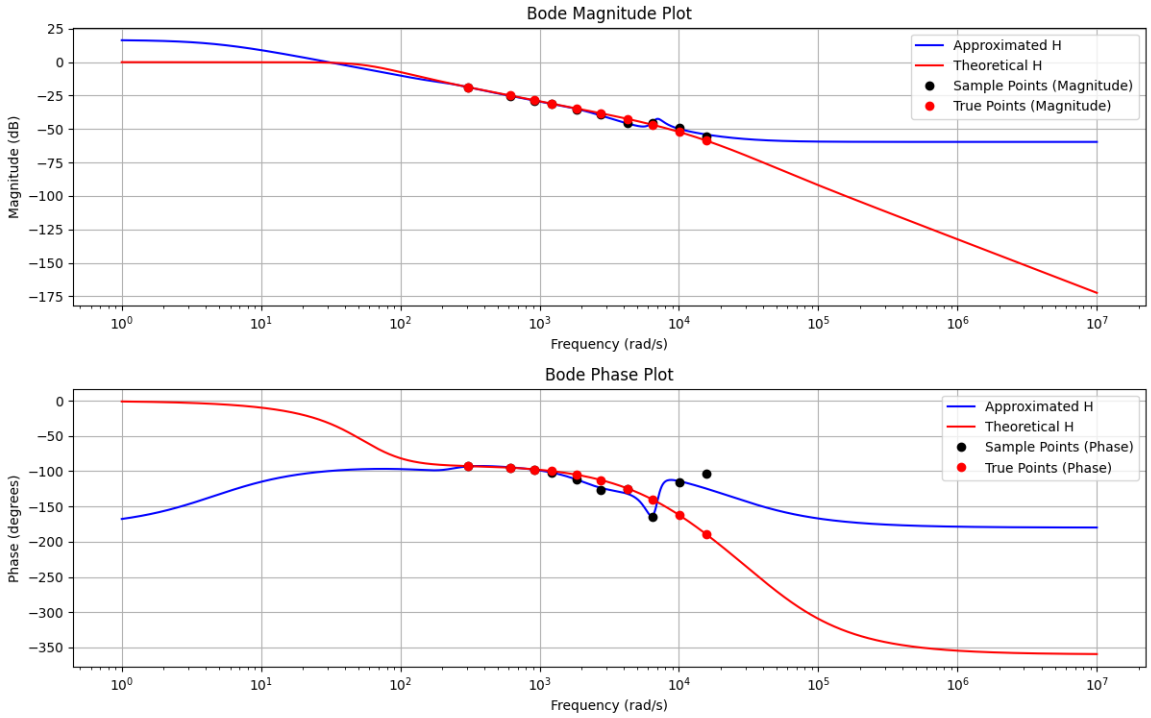


Figure 2.17: Bode plot for velocity loop model trained on 10 simulated sine waves with a noise level of 10 SNR.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.80e+01)
Iteration n.2	
Convergence Test (poles estimation)	passed (9.79e-01)
Final Model Error	5.12e-04
Average Relative Error	3.80e-01
Average Relative Error (random points)	5.50

Table 2.12: Table of VF outputs for velocity loop model trained on 10 simulated sine waves with a noise level of 10 SNR.

The 10 frequencies inputted into the VF algorithm were those for Test 2 noted in Table A.1. Two iterations were needed for model convergence. However, the error was quite large. We can see for large frequencies, the phase shift was clearly present. Peaks and dips showed up in both the phase and magnitude plots. The average relative error with random points was quite large as it was 5.50.

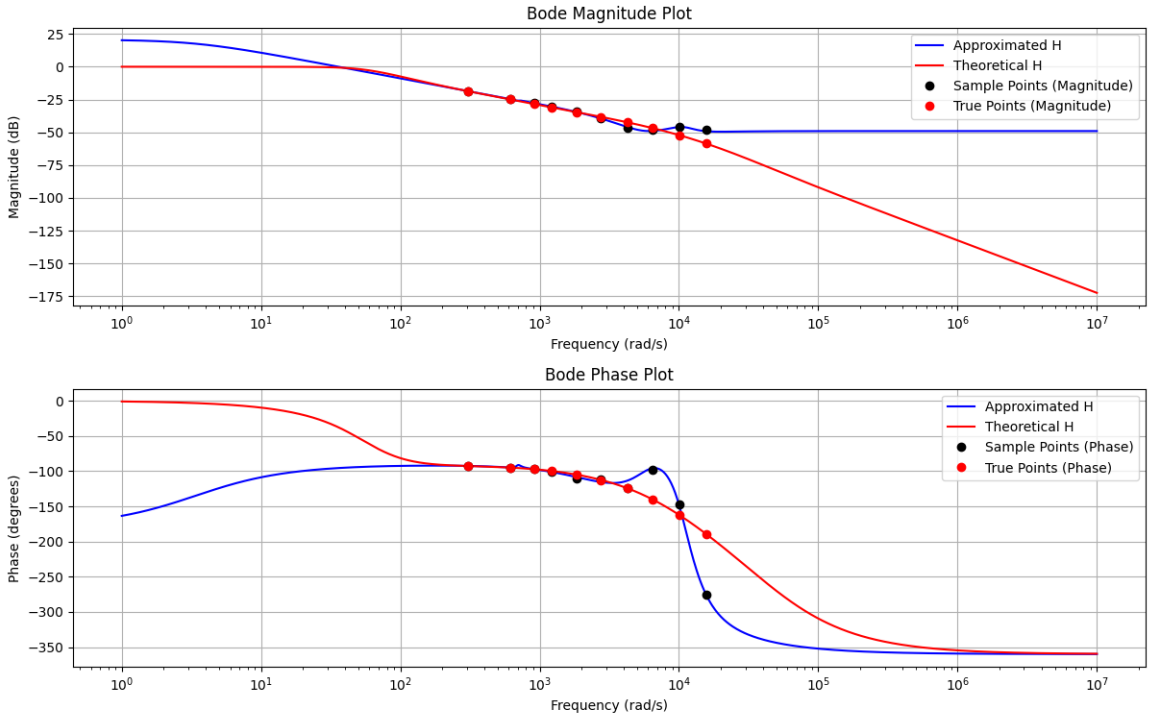


Figure 2.18: Bode plot for velocity loop model trained on 10 simulated sine waves with a noise level of 5 SNR.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.41e+01)
Iteration n.2	
Convergence Test (poles estimation)	passed (5.47e-01)
Final Model Error	6.72e-04
Average Relative Error	1.12
Average Relative Error (random points)	7.27

Table 2.13: Table of VF outputs for velocity loop model trained on 10 simulated sine waves with a noise level of 5 SNR.

The 10 frequencies inputted into the VF algorithm were those for Test 2 noted in Table A.1. Two iterations were needed for model convergence. However, the error is quite large. We can see for large frequencies, the phase shift is clearly present. Multiple peaks and dips show up in both the phase and magnitude plots. The average relative error with random points are quite large as it is 7.27.

2.4.4 Discussion on VF Performance with Simulated Data

As expected, for noise-free data, the VF algorithm was able to approximate the true transfer function with high accuracy. This held true for both the open and closed current loops. The effects of noisy data in VF was only noticeable once the SNR was set to 10 or below. The noise was added point-wise to only the outputs, y(t), of the simulated data before the FFT

transformation into the s-domain. fFor very large frequencies, we saw a bigger error between the sample points and the true points of the transfer functions. Due to these errors, for a larger range of frequencies (100 to 10000 Hz), VF does a worse job at approximating the true transfer function compared to smaller ranges (100 to 1000 Hz). In general, frequencies under 1200 Hz allow for accurate sample points. Anything over this threshold was another source of noisy data for VF. Another source of noise was derived from the non-steady state observations from the simulation. If the first period of the simulation was used for VF, the values are not in steady state, thus, causing noise to be present in our data. Using the second period of the simulation was tested to be enough time for the system to be in steady state. Another important point to mention was the order of the approximated VF transfer functions. For the true transfer functions of open, closed, velocity loops, the orders are (1, 2), (2, 3), and (5, 7) respectively. However, due to the structure of VF, the approximated transfer functions for open and closed loops are (2, 2), (3, 3), and (7,7) respectively. The extra term in the numerator is reflected by the ends of the VF line leveling out to a slope of 0 as the frequency goes to infinity. While this end behavior of the VF function causes VF not to be able to accurately capture the relationship between our inputs and outputs for large frequencies, this was not a huge problem as our real data will not have such large frequencies. The motor's sampling rate is fixed at 1/25000 seconds, thus, by the Nyquist–Shannon sampling theorem, the frequencies cannot exceed 12500 Hz.

2.5 Testing Vector Fitting on Real Data

In this section, we test the performance of VF algorithm on real data.

Real data was collected using the stepper motors generously provided by ADI. We looked at all three previously mentioned loops: current open loop, current closed loop, and velocity loop.

Fifty singular sines waves of different frequencies were fed into the system as inputs. The flash in the ADI motors is able to capture exactly 1024 data points at a fixed sampling rate of $4 \cdot 10^{-5}$. For a total runtime of 0.04092 seconds, only the data for the last period was used for the VF pipeline. Previously, it was found that random time delays were present in our real data.

As mentioned for the simulated data section, the FFT function in SciPy was used to transform the time domain signals into the frequency domain.

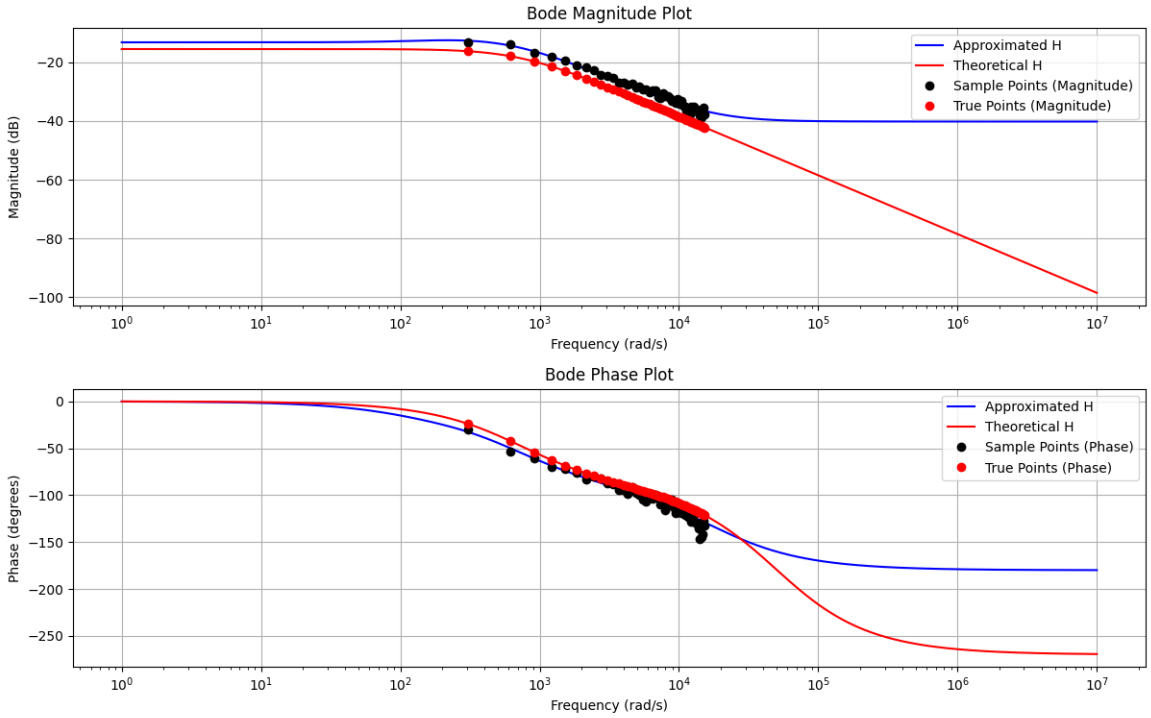


Figure 2.19: Bode plot for open loop model trained on 50 real sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.18e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (3.25e-01)
Final Model Error	0.0049670458387568155
Average Relative Error	0.41877192672092767
Average Relative Error (random points)	0.31014336226753353

Table 2.14: Table of VF outputs for open loop model trained on 50 real sine waves.

The 50 frequencies inputted into the VF algorithm were those for Test 3 noted in Table A.1. Two iterations were needed for model convergence. From the plots, we saw that the VF model and the sample points for magnitude were above the theoretical TF while it were below the theoretical TF for the phase. No peaks or dips were present in the VF model compared to what we saw with noisy simulated data.

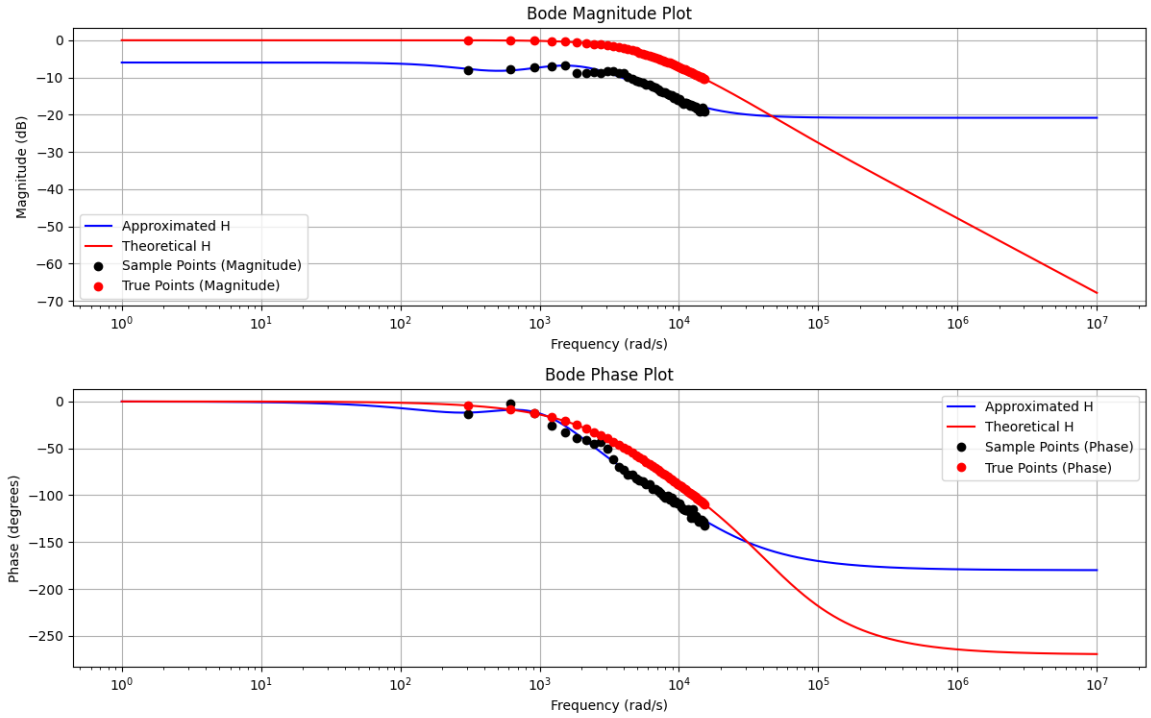


Figure 2.20: Bode plot for closed loop model trained on 50 real sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.28e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (3.42e-01)
Final Model Error	0.03107924082450081
Average Relative Error	0.41804130209788526
Average Relative Error (random points)	0.41804130209788526

Table 2.15: Table of VF outputs for closed loop model trained on 50 real sine waves.

The 50 frequencies inputted into the VF algorithm were those for Test 3 noted in Table A.1. Two iterations were needed for model convergence. From the plots, we saw that the VF model and the sample points for magnitude and phase were below the theoretical TF. A small dip in both the magnitude and phase plot was observed between 10^2 and 10^3 rad/s.

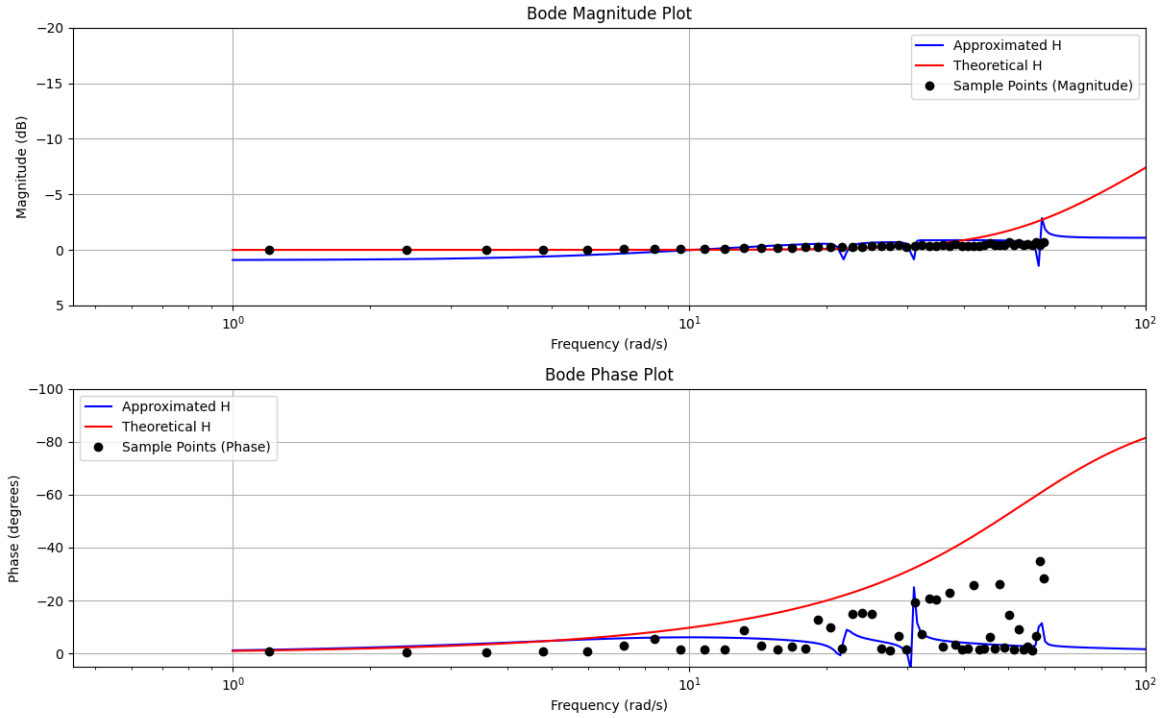


Figure 2.21: Bode plot for velocity loop model trained on 50 real sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (1.03e+03)
Iteration n.2	
Convergence Test (poles estimation)	failed (2.10e+00)
Iteration n.3	
Convergence Test (poles estimation)	passed (4.22e-01)
Final Model Error	0.13725214461446986
Average Relative Error	0.3416258761003864
Average Relative Error (random points)	0.2305205655291926

Table 2.16: Table of VF outputs for velocity loop model trained on 50 real sine waves.

The 50 frequencies inputted into the VF algorithm were those for Test 3 noted in Table A.1. Two iterations were needed for model convergence. From the plots, we saw that the VF model and the sample points for phase were below the theoretical TF. The magnitude for the sample points seemed relatively fine, however, the VF model has several peaks and dips as the frequency increases. The phase of the sample points were quite noisy, which resulted in visible peaks and dips in the plots.

2.5.1 Discussion on VF Performance with Real Data

Noise was presented in our real data. Interestingly, the noise was either all below or above the theoretical phase and magnitude for all three loops. This observation raised two concerns: either noise was non-random or the theoretical TF was incorrect. As the theoretical

TF was derived from physics based first-principles method, it could very much be true that the function computed from that method was simply incorrect. In that case, the average relative error, which was based off the difference between our VF model and theoretical H would not be a valid result.

Nonetheless, we could still observe from the outputs that the VF model was attempting to capture all the noise in the data. As the noisy samples were all that VF could base its model on, the VF model goes through most of the sample points.

The noise was the most evident in the velocity loop. When looking at Figure 2.21, the VF model had random peaks in both the magnitude and phase plots. These peaks represented VF's attempt at trying to fit the noisy data, which was not good in terms of trying to approximate the true transfer function. While it was not guaranteed that the theoretical TF was the correct model, it was safe to assume that the true TF will be a smooth curve as it would be a rational function. As this was not true for the VF model trained with real data, we concluded that VF did not work well with noisy inputs.

Chapter 3

Neural Networks

Since the VF algorithm does not perform well when the data is noisy, we want to design a different model which is robust to noise. We propose a neural network(NN) method to approximate the transfer function. In this Chapter, we first give an overview of neural networks in section 3.1. In section 3.2, we describe the three neural network architectures that we perform numerical experiments. The numerical results of these three architectures are then discussion in sections 3.3, 3.4, and. 3.5. To conclude, we discuss transfer learning and it's preliminary results in section 3.6.

3.1 Overview of Neural Network

Neural networks have been used in many fields, such as computer vision, large language model, and etc. Mathematically, a neural network consists of several layers, each performing a linear transformation followed by a non-linear activation function. This combination introduces non-linearity into the model, enabling it to capture complex patterns and relationships in data. Similar to rational approximation, neural networks can approximate any continuous function.

The operation of a single layer in a neural network can be expressed as:

$$x \to f(Wx + b) \tag{3.1}$$

where

- x is the input vector,
- W is the weight matrix,
- b is the bias vector,
- f is the non-linear activation function.

A neural network typically consists of an input layer, multiple hidden layers, and an output layer. Each hidden layer performs a linear transformation applies a linear transformation to its inputs and then passes the result through a non-linear activation function. This process can be recursively defined as:

$$h_i = f(W_i h_{i-1} + b_i) \quad \text{for} \quad i = 1, 2, \ldots, L \tag{3.2}$$

where $h_0 = x$ (the input), h_i represents the output of the i-th layer, W_i and b_i are the weight matrix and bias vector of the i-th layer, and L is the total number of layers. The

figure displays the structure of deep neural network, which is composed with several hidden layers that make the neural network 'deep'. Each layer transforms the input data through weighted sums and non-linear activations, allowing the network to learn and model complex functions.

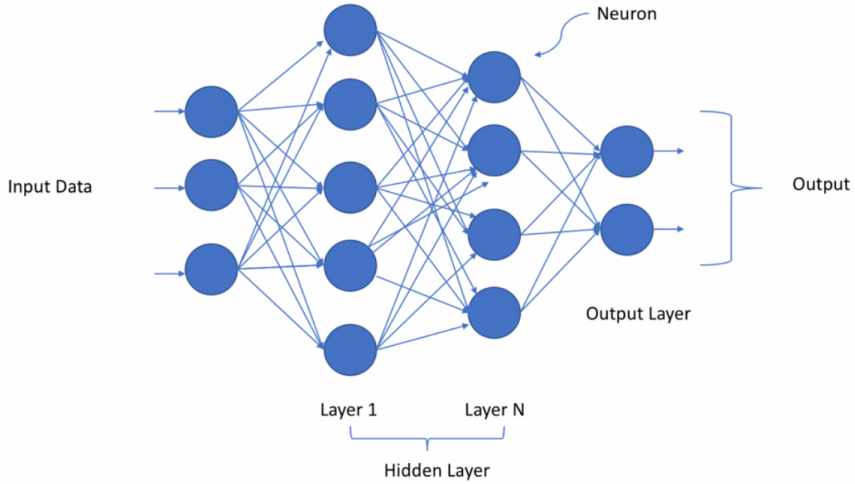


Figure 3.1: Structure of a deep neural network.

Usually, we select the activation function and the W matrix is unknown. To be able to have the optimal neural network, we need to have the optimal weights which can be done by solving an optimization problem to find the optimal weights. Activation functions introduce the non-linearity to the neural network which allows the neural network to model and understand the complex patterns in data. Different activation functions can affect the training process and the performance of the neural network. Two different activation functions were proposed in order to approximate the transfer function for our system: ReLU and rational. For the rational activation function, we examined two different architectures: singular node structure and multiple nodes structure based on the order of the target transfer function.

Rational activation functions use ratios of polynomials and are represented as:

$$f(x) = \frac{\sum_{i=1}^{n} a_i x^i}{\sum_{i=1}^{n} b_i x^i} \tag{3.3}$$

The coefficients $\{a_i\}_{i=1}^{n}$ and $\{b_i\}_{i=1}^{n}$ in the rational activation function are trainable parameters. Rational activation functions leverage mathematical properties similar to the rational function $H(s) = \frac{Y(s)}{X(s)}$, which can enhance learning efficiency. Using rational activation functions will make the entire neural network act as a rational function, which aligns with the principles of the VF algorithm. In theory, rational networks, with an error threshold ϵ, require fewer parameters compared to ReLU networks.The process of finding the optimal deep neural network is essentially equivalent to finding the optimal weights (W_i) and biases (b_i) that minimize the difference between the network's predictions and the actual target values. This is achieved by solving an optimization problem, typically using a method such as gradient descent.

During training, the network adjusts its weights and biases by minimizing a loss function, which quantifies the error of the network's predictions.The loss function for our network aims to minimize the error between the actual and predicted values of $H(j\omega)$. For each frequency

input ω_k, the loss function compares the predicted $H(j\omega_k)$ with the actual values.The Mean Squared Error (MSE) was used as the loss function, expressed as:

$$\text{Loss} = \frac{1}{K}\sum_{k=1}^{K} |H(s_k) - \text{NN}(s_k)|^2 \tag{3.4}$$

Here,K is the number of training samples, $H(s_k)$ are the true function values, and $\text{NN}(s_k)$ are the predicted values by neural network for all $k \in [1:K]$.

3.2 Proposed Structure

We propose three different neural network architectures in this section. The first one is common ReLU Neural network. The second one uses rational activation function which is called Rational Neural Network Architecture 1 in our report. Using rational activation function again but with a different structure, Rational Neural Network Architecture 2 is considered as well.

3.2.1 ReLU Deep Neural Network

One of the most commonly used activation functions is ReLU, defined in Section 1.2. This research uses ReLU activation function for our deep neural network architecture and uses it as a benchmark for comparison with the rational neural network structures that we propose. ReLU is favored for its computational efficiency and speed. It helps networks learn complex patterns by introducing necessary non-linearity. Many theoretical studies have explored ReLU networks, particularly comparing deep versus shallow architectures [1]. The weights and biases in ReLU networks are the parameters trained to find the optimal transformations for the data.

The input layer of this neural network takes frequency values of the input signal after converted to frequency domain from time domain by Fourier Transform and outputs the calculated value of the transfer function at a particular value of s. This output is then updated using stochastic gradient descent by minimizing the loss function. The number of layers and nodes are structured after hyper parameter tuning, finding the most accurate combination of layers and nodes.

3.2.2 Rational Neural Networks Architecture 1

In this neural network architecture, the design includes three layers: an input layer, a hidden layer, and an output layer. The input layer receives the variable s, which represents the frequency or Laplace variable.

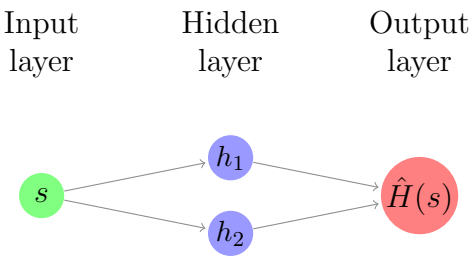


Figure 3.2: Rational NN architecture 1.

For a system of order 2, the hidden layer is structured with two nodes, each representing a rational function. The nodes are defined as:

$$h_1(s) = \frac{a_1 s + b_1}{c_1 s + d_1} \quad \text{and} \quad h_2(s) = \frac{a_2 s + b_2}{c_2 s + d_2}.$$

In this setup, the weights W and biases B for the nodes are fixed: the weights are set to 1, and the biases are set to 0. Consequently, only the coefficients a_i, b_i, c_i, and d_i are subject to training. These coefficients are optimized during the training process to best approximate the system's transfer function. The output of the network, denoted as $\hat{H}(s)$, is obtained by summing the contributions from each hidden node:

$$\hat{H}(s) = \sum_{i=1}^{n} h_i(s)$$

This summation provides an approximation of the desired transfer function. With the trained coefficients, it is straightforward to extract the poles and zeros from the resulting transfer function. The coefficients a_i, b_i, c_i, and d_i are initialized with values drawn from a normal distribution:

$$a_i, b_i, c_i, d_i \sim \mathcal{N}(0, 1)$$

This initialization ensures a diverse starting point for the optimization process, which helps in accurately modeling the transfer function.

3.2.3 Rational Neural Networks Architecture 2

Similar to the first neural network architecture, this architecture includes three layers: an input layer, one hidden layer, and an output layer. The input layer receives the variable s, which represents the frequency or Laplace variable.

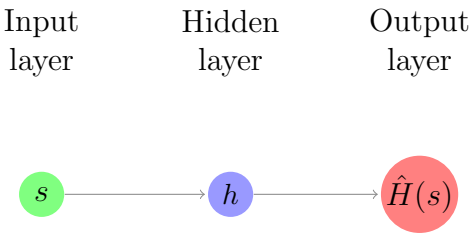


Figure 3.3: Rational NN architecture 2.

The hidden layer always consists of a singular node, defined as follows:

$$h(s) = \frac{\sum_{i=0}^{n} a_i s^i}{\sum_{i=0}^{n} b_i s^i}$$

The value n depends on the order of the system being modelled. Just like the first architecture, this architecture sets the weights W and biases B to be 1 and 0 respectively. As such, only the coefficients a_i and b_i are subject to training, and are optimized to best approximate the system's transfer function. The output of the network, $\hat{H}(s)$ is simply the value that the hidden node computes:

$$\hat{H}(s) = h(s) = \frac{\sum_{i=0}^{n} a_i s^i}{\sum_{i=0}^{n} b_i s^i}$$

The rational form of the neural network aligns well with the rational structure of transfer functions, suggesting potential efficacy in modeling such systems.

There are two methods of initialization investigated within this architecture of the Rational Neural Net. The first method is through random initialization, similar to the initialization mentioned in the first architecture. The initial values of a_i and b_i are drawn from a normal distribution:

$$a_i, b_i \sim \mathcal{N}(0, 1)$$

The second method is through VF initialization. This method involves first applying the VF algorithm to the data and computing the approximated coefficients a_i and b_i by using the returned poles and residues.

3.3 Numerical Results on ReLU Neural Networks

In this section, we will present the numerical comparison of Deep Neural Network structure with ReLU activation function against the VF algorithm for different cases using synthetic data.

3.3.1 Deep NN structure analysis

Initially, we investigated the significance of number of layers and nodes used in the neural network structure by using hyper parameter tuning. We grouped the number of layers as 2-4, 4-6 and 6-10 while grouping number of nodes as 10-30, 30-50 and 50-100. The models are tested using each of these combination of structures, see Figure 3.4.

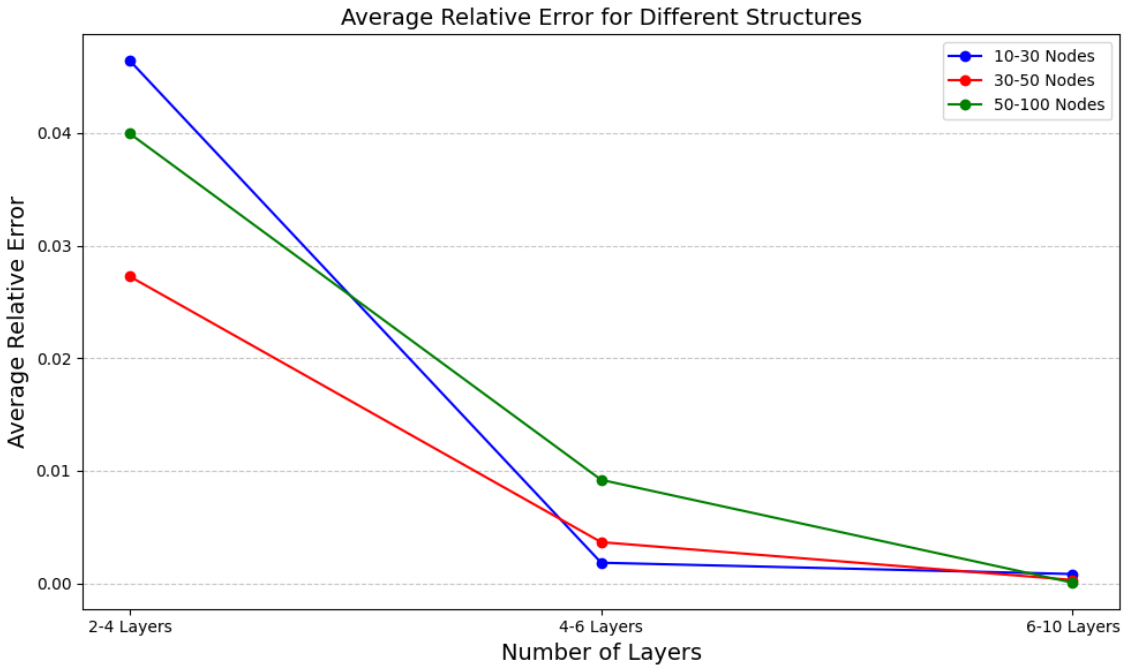
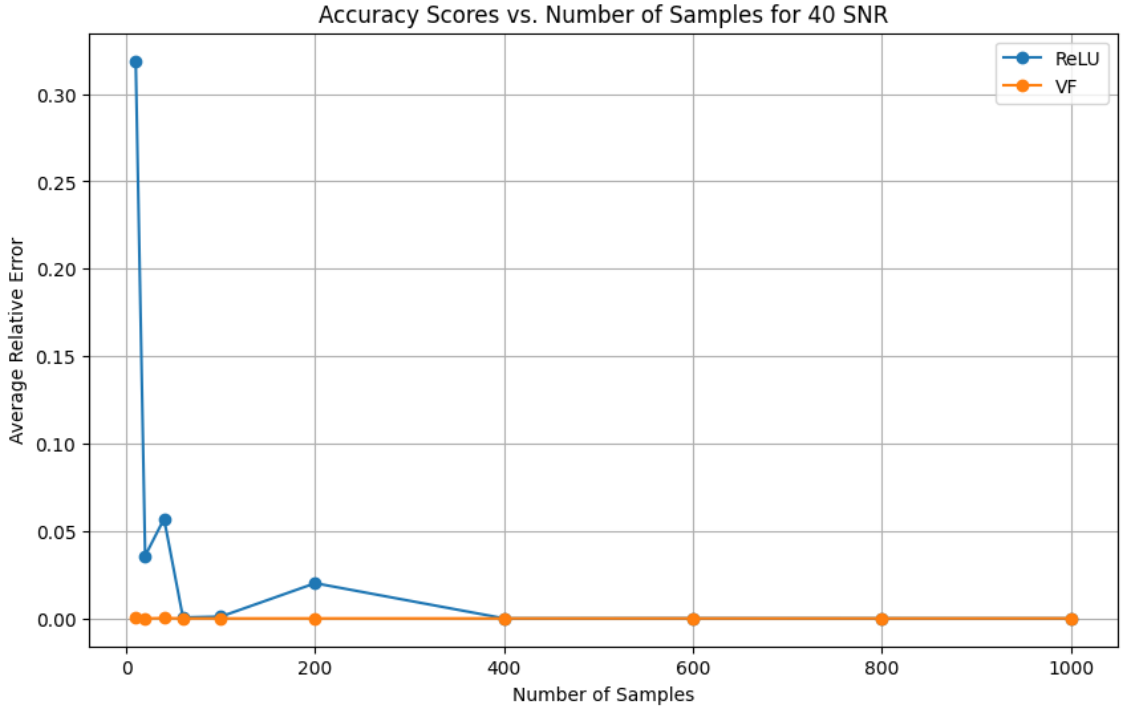


Figure 3.4: Performance of ReLU Deep Neural Network: average relative error versus the number of layers.

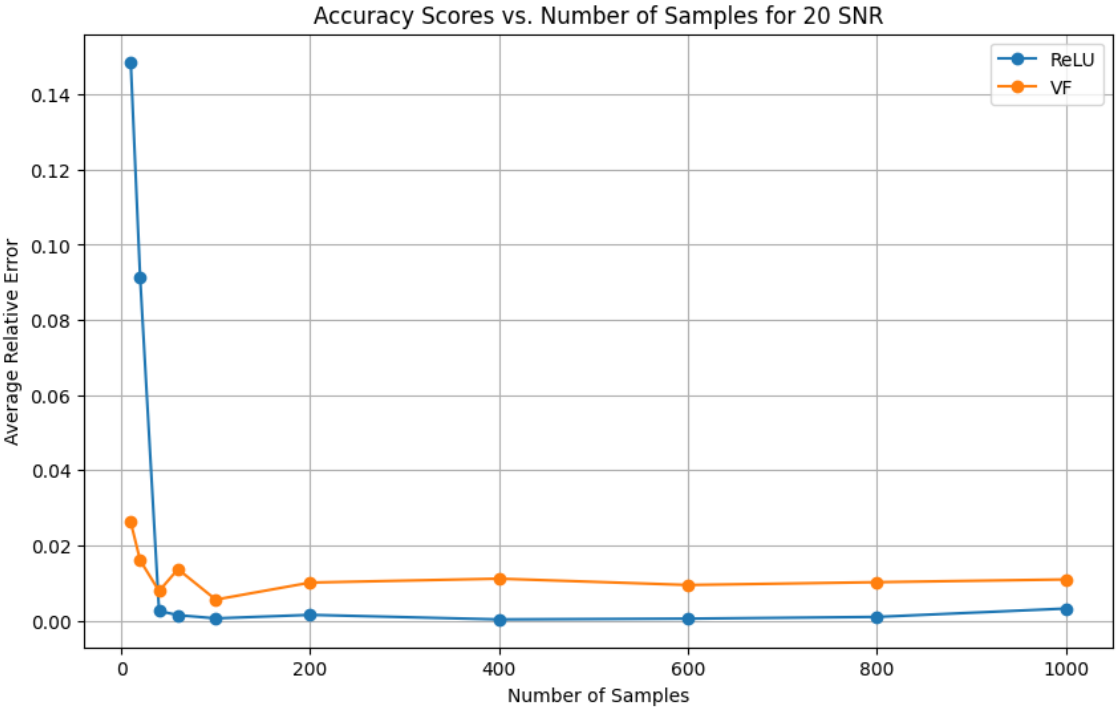
Figure 3.4 shows that the accuracy score of the model is highly sensitive to both number of layers and number of nodes. When the neural network is shallow, increasing the number of nodes do not necessarily improve the accuracy as it saturates very quickly. However, when the neural network is deeper, the number of notes at each layer has a significant positive correlation with the accuracy of the model. Overall, this shows that deep neural networks using ReLU activation function require large number of layers and nodes in order to perform well. Given that this synthetic data used contains no noise, unless the model is structured with many layers and nodes, VF provably performs better. Since the number of nodes and layers improve the amount of calculations performed during training, it increases the computational complexity of the model which is one of the main pitfalls of using a deep neural network structure for the task at hand.

3.3.2 Data requirements at different noise levels

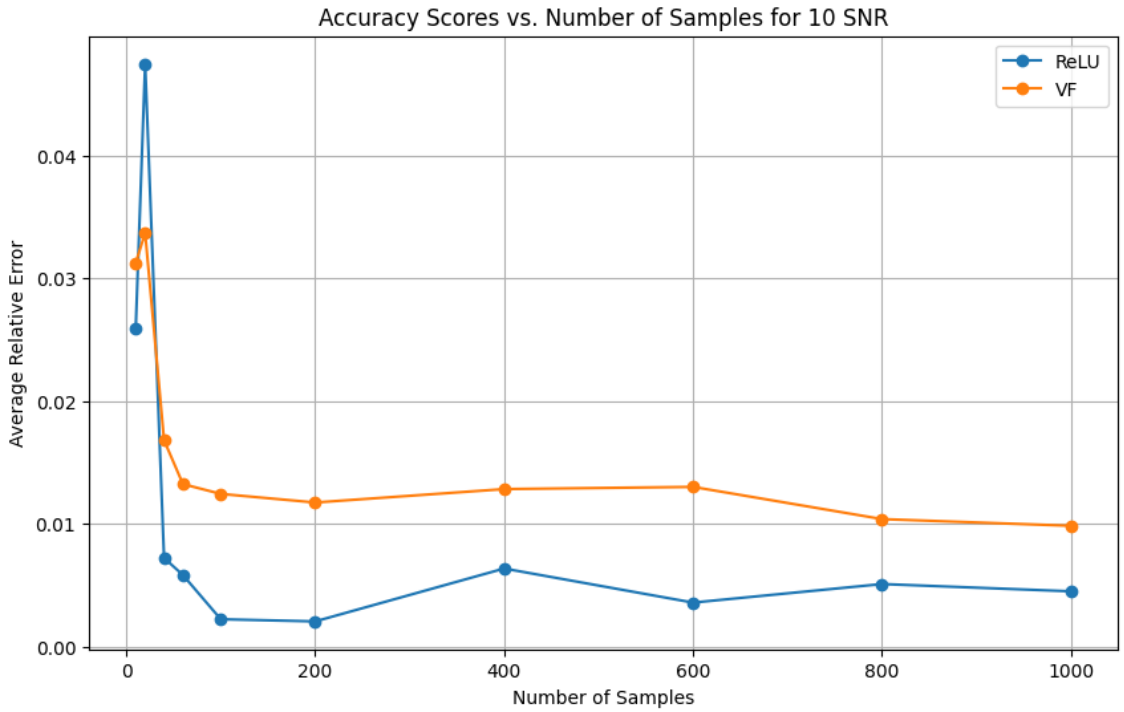
In this subsection, we want to explore the amount of data required to train ReLU neural network and VF algorithm. This an important consideration given that collecting data from hardware is usually time-consuming and costly. Therefore, we investigated the number of data samples required at different noise levels to achieve high accuracy modelling.



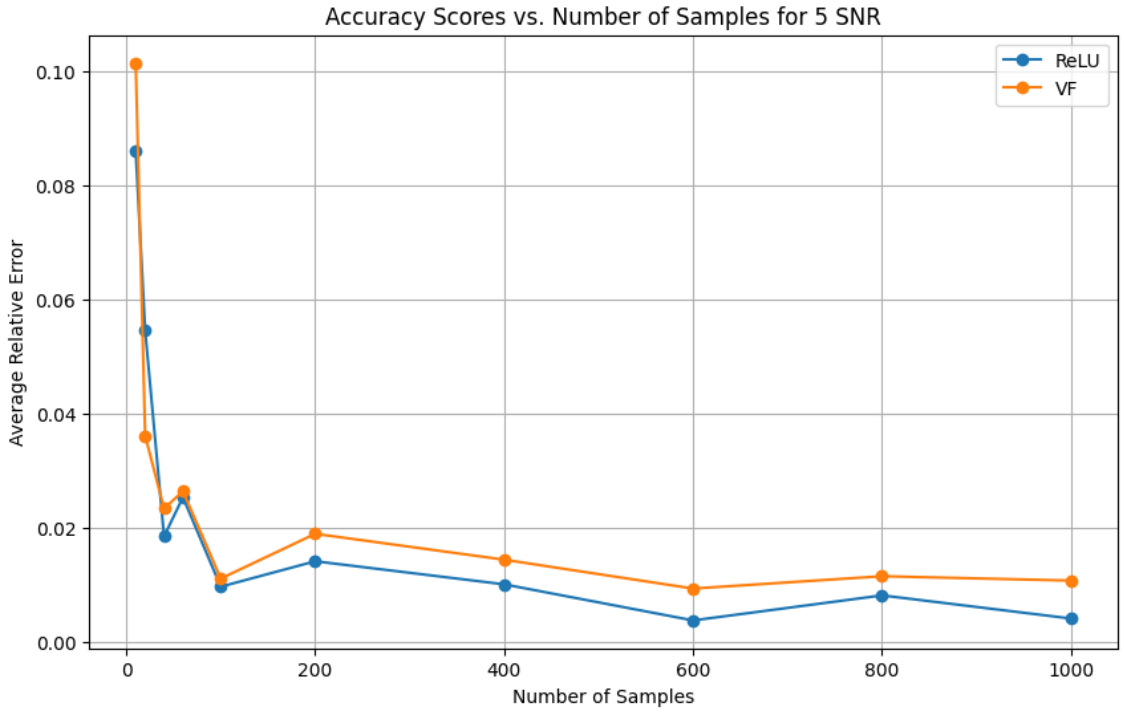
(a) 40 SNR



(b) 20 SNR



(c) 10 SNR



(d) 5 SNR

Figure 3.5: Average relative error versus number of samples under different noisy levels (10 SNR and 5 SNR).

Figure 3.5 shows that with low level of noise, VF performs better with fewer number of samples and deep neural network structure using ReLU activation function achieves similar

results only if sufficient amount of data is provided which is 400 samples in this case. Moreover, we observe that the increased noise level causes VF to converge at a higher noise level compared to deep neural network. Nevertheless, for data samples fewer than 50, VF performs significantly better than ReLU deep neural network. In addition, VF performs worse than deep neural networks given that there is sufficient amount of data available for training. In particular, we can observe that deep neural network outperforms VF after 50 or more samples. Lastly, when the the training data is significantly noisy, deep neural network using ReLU outperforms VF for any given number of samples. Overall, these graphs clearly indicates that deep neural network outperforms VF when the data available is noisy. Nevertheless, deep neural networks require many number of samples to achieve reasonable accuracy levels.

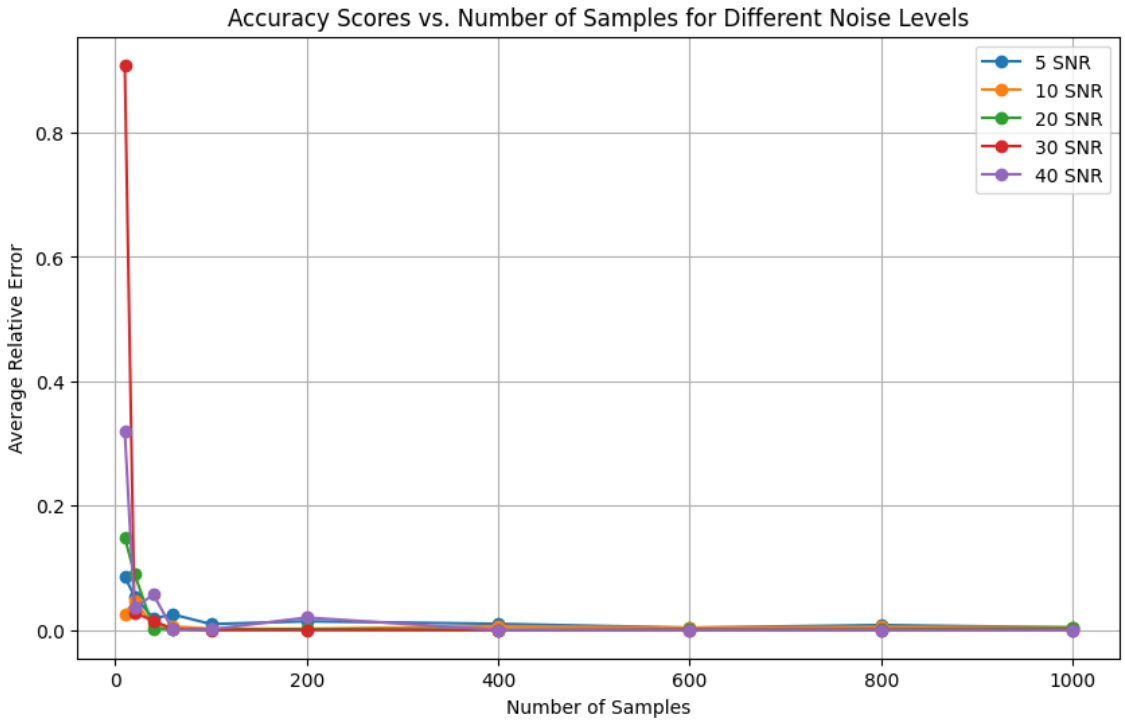


Figure 3.6: Performance of ReLU neural network: average relative error versus number of samples under different noise levels.

Figure 3.6 shows that deep neural network structure using ReLU activation function performs very well regardless of the noise level given that there is sufficient amount of data available to train the neural network. The accuracy scores saturate after 100 samples for all noise levels. This is an important insight into the use of neural networks for transfer function modelling, especially when the system is noisy. While physics based modelling methods and VF can be prone to noisy data, this deep neural network structure using ReLU is immune to noise, which makes it a potential candidate for a system identification tool. The main problems with this structure is that it is sensitive to the amount of data samples and the number of layers and nodes used, making it a computationally expensive method.

3.4 Numerical results on Rational Architecture 1

This section presents a comparative analysis of Rational Architecture 1 and the VF model. The objective is to evaluate the performance of these two models across various datasets—synthetic, real, and simulated—under different noise conditions. This thorough assessment aims to identify the strengths and weaknesses of each model, providing a deeper understanding of their behavior in diverse scenarios.

3.4.1 Training on Synthetic Data

The first phase of training focuses on synthetic data, generated from a second-order system. This controlled environment allows us to examine how each model responds to different levels of noise and varying sizes of training samples. After training both models on the synthetic data under various conditions, they are tested on 1,000 samples.

Training Samples	SNR	Rational Model Error	VF Model Error
10	Noise-free	0.08279	0.000
50	Noise-free	0.01154	0.000
100	Noise-free	0.000246	0.000
10	10 dB	0.30738	0.40425
50	10 dB	0.13575	0.53133
100	10 dB	0.03900	0.70630
10	5 dB	0.53178	1.07197
50	5 dB	0.16444	0.93528
100	5 dB	0.02870	1.21592

Table 3.1: Average test relative error comparison between Rational and VF models for 1,000 test samples.

The results from the noise-free conditions demonstrate the VF model's exceptional accuracy, with errors consistently at or near zero across all training sample sizes. This indicates the VF model's strong ability to precisely capture the underlying system dynamics when the data is undisturbed by noise. In contrast, the Rational model, while still performing well, shows higher errors in noise-free data. However, as the number of training samples increases to 100, the Rational model's error reduces significantly to 0.000246, demonstrating its ability to improve with more data.

When noise is introduced into the data, particularly at a 10 dB SNR, the performance dynamics shift. The Rational model's error starts at 0.30738 with 10 samples and decreases to 0.03900 with 100 samples. On the other hand, the VF model, which excels in noise-free conditions, begins to struggle as the noise level increases. Its error climbs from 0.40425 with 10 samples to 0.70630 with 100 samples, suggesting a growing difficulty in handling noisy data as the training set expands.

The scenario becomes even more challenging at a 5 dB noise level, where the Rational model further showcases its robustness. In Figure 3.7, we observe that the rational neural network outperforms VF model since the Rational model is closer to the theoretical transfer function.

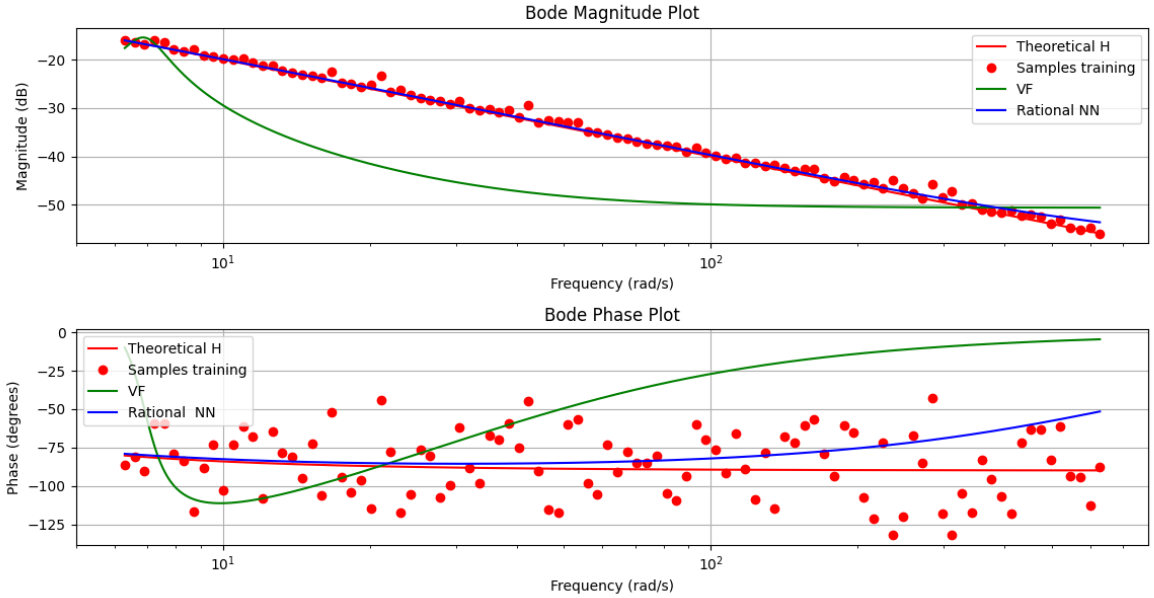


Figure 3.7: Performance of rational neural network and VF algorithm trained on 100 training samples and the noisy level is 5 SNR.

3.4.2 Training on Simulated Data

The transfer function for an open-loop circuit is used to simulate data for training and testing the Rational and VF models. We analyze the performance of the models with varying numbers of training samples and different levels of noise. The training samples are either 10 or 50 samples simulated after applying the transformation on the time signals for both input and output, while the test samples are 1,000 samples.

Training Samples	SNR	Rational Model Error	VF Model Error
10	Noise-free	1.17590	0.00020
50	Noise-free	0.71070	0.00011
10	10 dB	1.88890	0.10700
50	10 dB	1.10925	0.70930
10	5 dB	1.92200	1.87630
50	5 dB	1.08463	3.81020

Table 3.2: Average test relative error comparison between Rational and VF models for 1,000 test samples.

Similar to the synthetic training sample results, Table 3.2 shows that VF has better accuracy, with errors near zero for noise-free data. However, as the number of training samples increases to 50, the Rational model's error reduces significantly to 0.71070, demonstrating its capability to improve with more data.

When noise is introduced into the data, particularly at a 10 dB signal-to-noise ratio, the performance dynamics shift. The VF model struggles as the noise level increases. Its error climbs from 1.87630 with 10 samples to 3.81020 with 50 samples, suggesting a growing

difficulty in handling noisy data as the training set expands for 5 dB. However, the Rational model performs better as the error decreases from 1.92200 with 10 samples to 1.08463 with 50 samples under the same noise level. Figure 3.8 illustrates the performance of rational neural network and VF algorithm.VF model is fitting to the noisy training samples and the rational neural network is closer to the theoretical transfer function.

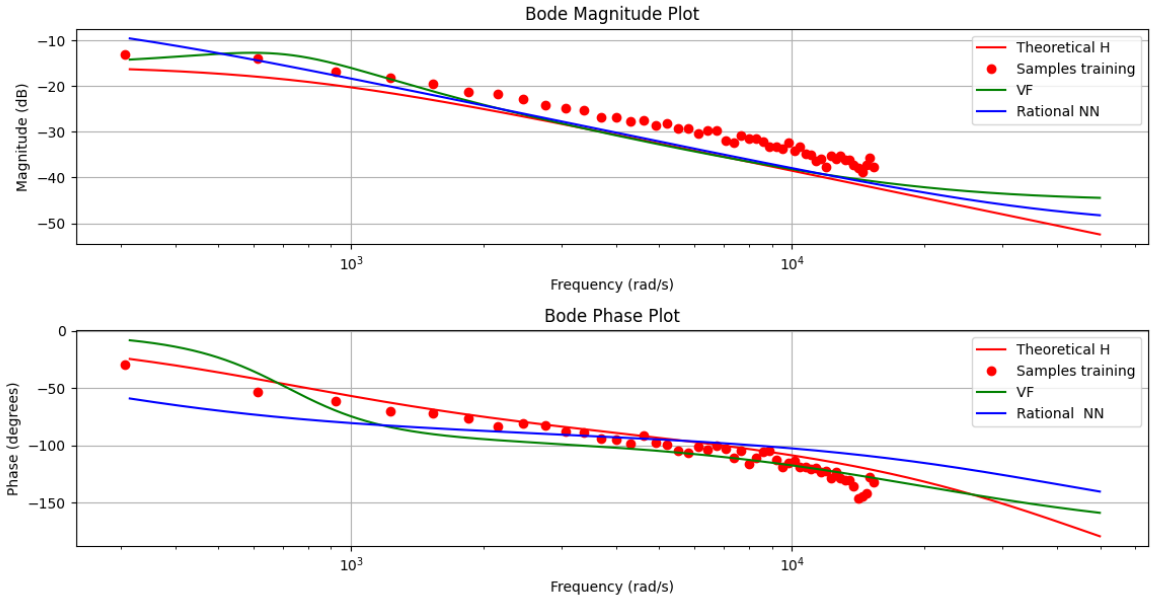


Figure 3.8: Performance of rational neural network and VF algorithm trained on 50 training samples and noisy level 5 SNR.

3.4.3 Training on Real Data

In this phase, the models are trained using real data generated from motors, with open-loop and closed-loop transfer functions. This practical scenario allows us to evaluate the models' performance under more realistic conditions compared to synthetic or simulated data.

Loop Type (50 samples)	Rational Model Error	VF Model Error
Open Loop	0.328912	0.3889
Closed Loop	0.292844	0.4025

Table 3.3: Average test relative error comparison between Rational and VF models for 1,000 test samples.

Table 3.3 summarizes the average test relative errors of both models when applied to real data from motors with open-loop and closed-loop configurations. The errors are calculated based on 1,000 test samples. For the open-loop system, the Rational model shows a slightly lower average error of 0.328912 compared to the VF model's error of 0.3889. Similarly, for the closed-loop system, the Rational model achieves a lower average error of 0.292844, while the VF model's error is 0.4025. These results suggest that the Rational model performs

better in capturing the dynamics of real systems, providing a more accurate approximation of the transfer functions.

Figure 3.9 displays the performance of both models for open-loop systems with 50 real samples. The plot illustrates that the Rational model aligns more closely with the theoretical transfer function compared to the VF model. Additionally, Figure 3.10 displays the performance of both models for closed-loop systems with 50 real samples. The plot illustrates that the Rational model aligns more closely with the theoretical transfer function compared to the VF model.

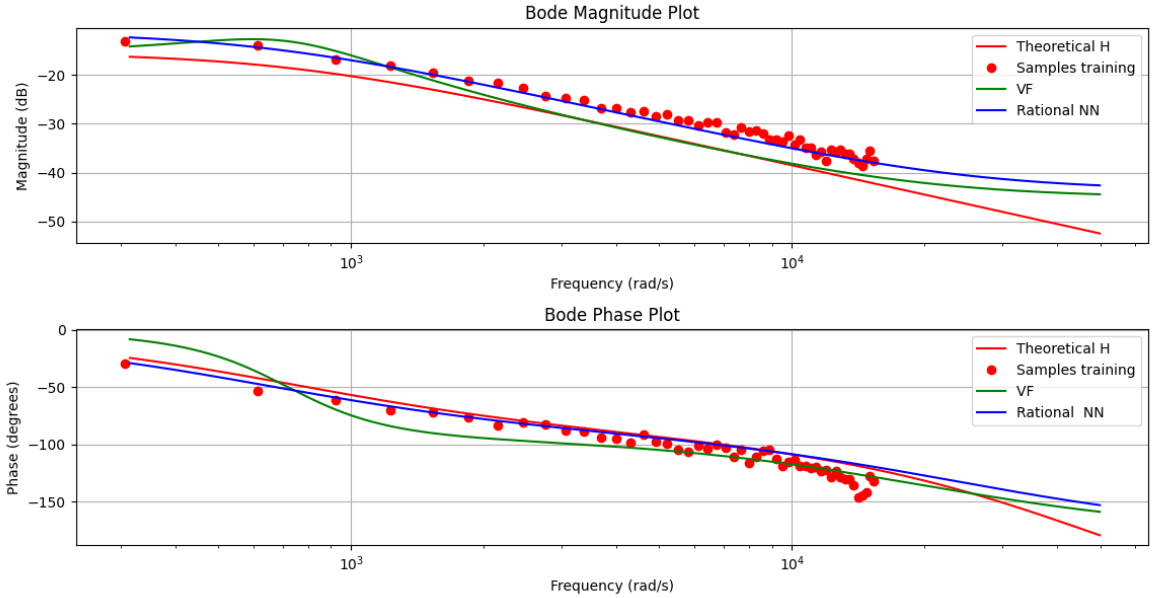


Figure 3.9: Performance with 50 real samples for open loop.

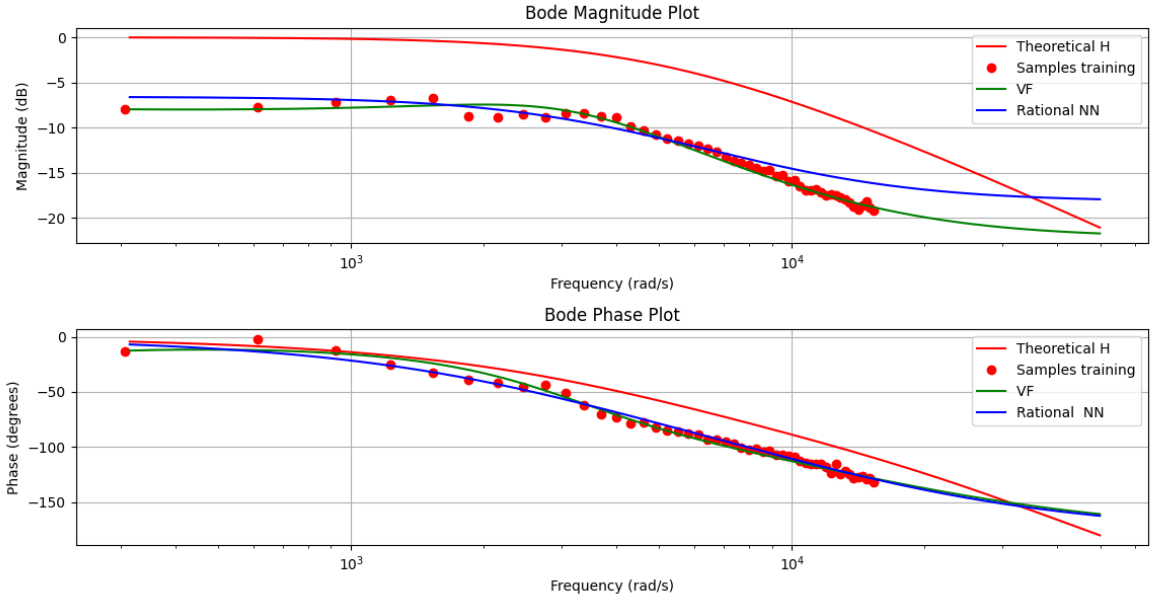


Figure 3.10: Performance with 50 real samples for closed loop.

3.5 Numerical results on Rational Architecture 2

This section presents a comparative analysis of Rational Architecture 2 and the VF model. We evaluate the performance of these two models across various datasets—synthetic, real, and simulated—under different noise conditions. This thorough assessment aims to identify the strengths and weaknesses of each model, providing a deeper understanding of their behavior in diverse scenarios.

3.5.1 Training and Testing on Synthetic Data

We first trained and tested the second architecture of the Rational Neural Net on synthetic data, generated from a third-order system. We train the VF model, Rational Neural Net model with Random Initialization, and Rational Neural Net model with VF initialization on noisy data with 20 dB SNR, 15 dB SNR, 10 dB SNR, and 5 dB SNR, and with different numbers of training samples: 20, 50, 100, 200, and 500 frequency and responses. The results are shown in Figure 3.11.

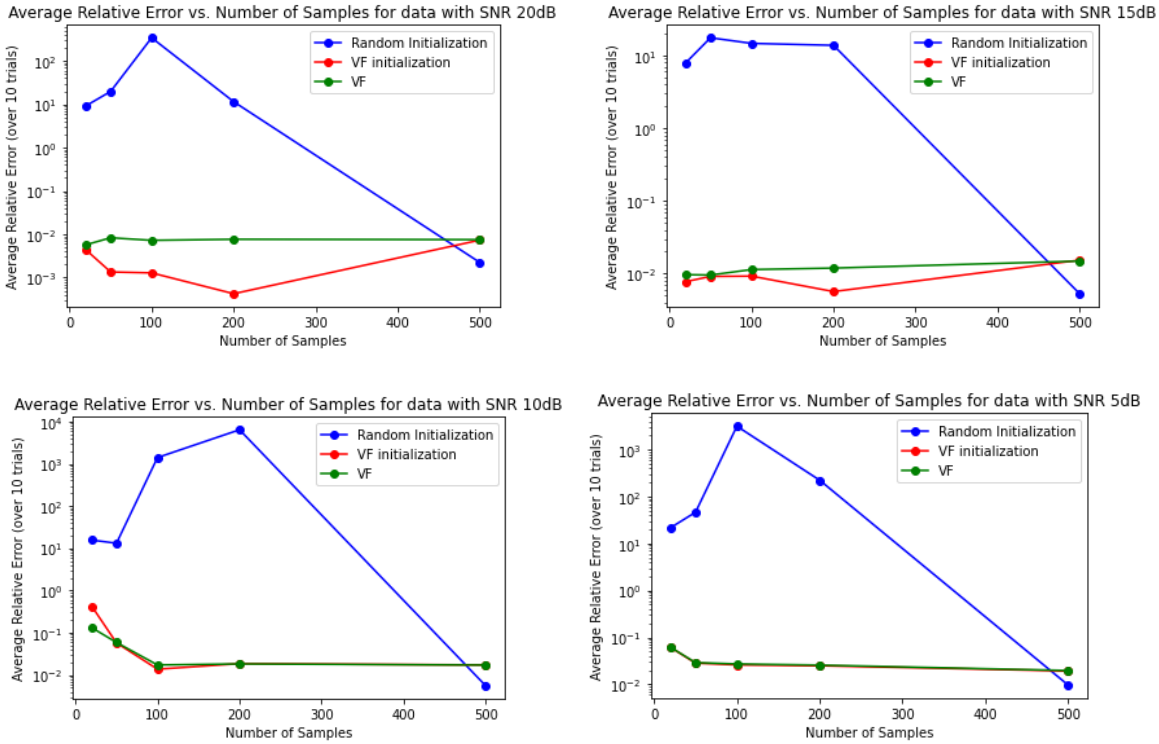


Figure 3.11: Average test relative error comparison between Rational Architecture 2 with random and VF initialization and VF for data with different noise levels.

Our results with synthetic data indicate that regardless of the noise level, the rational neural net initialized with random coefficients performs better than VF when it has at least 500 training samples. However, when trained with less than 500 training samples, VF consistently performs better, even with noisy data.

In contrast, the rational neural net initialized with the VF outputs performs better than VF with just 50 training samples with lower levels of noise. However, with a larger number of samples or with higher levels of noise, initializing the rational neural net with the VF outputs has the same performance as the VF model.

The bode plots in Figures 3.12 and 3.13 further solidify these results.

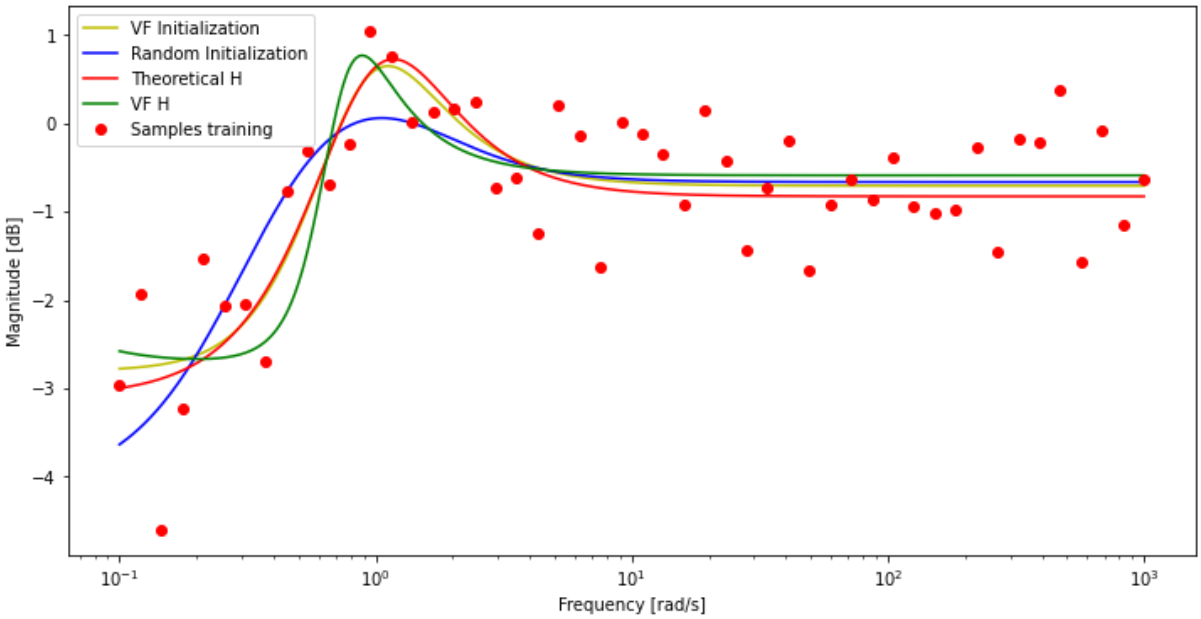


Figure 3.12: Magnitude bode plot of third order transfer function with the VF model and rational neural net trained on 50 data samples with noise level 20 SNR.

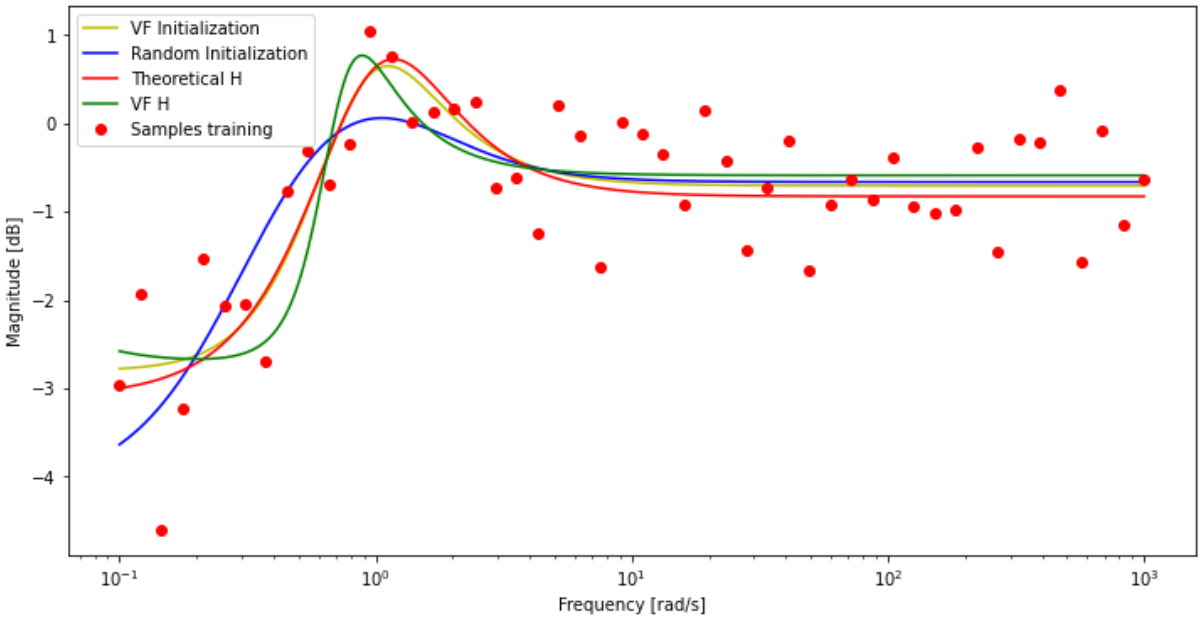


Figure 3.13: Magnitude bode plot of third order transfer function with the VF model and rational neural net trained on 500 data samples with noise level 5 SNR.

3.5.2 Training and Testing on Simulated Data

The transfer function for the open-loop circuit is used to simulate data for training and testing the rational neural net and the VF model. We analyze the performance of the models with different levels of noise, and we train the neural net initialized with random coefficients, the neural net initialized with VF outputs, and the VF algorithm on 50 samples.

Noise Level	Random Initialization	VF Initialization	VF Model
20 dB	0.1645	0.0417	0.2341
15 dB	0.1752	0.0522	0.3961
10 dB	0.1924	0.0465	0.3761
5 dB	0.1546	0.1768	0.4136

Table 3.4: Average test relative error comparison between Rational (random initialization and VF initialization) and VF models with 50 training samples on closed loop, with varying levels of noise.

Table 3.4 depicts the average test relative errors of the rational neural net initialized with random coefficients, the rational neural net initialized with VF coefficients, and the VF algorithm. All models are trained on 50 training frequencies which are logarithmically spaced apart, and the test errors are computed on 1000 test frequencies. The rational model initialized with VF show a smaller average error than VF with lower noise levels. With higher noise levels results, initializing with the VF outputs still performs better than VF, but still worse than with lower levels of noise. The performance of the rational neural net initialized with random coefficients is consistent throughout all levels of noise, but is worse than initializing with the VF output with lower levels of noise.

3.5.3 Training and Testing on Real Data

In addition to synthetic and simulated data, the rational neural net architecture 2 and the VF model were tested on the open loop and closed loop transfer function using real data.

Loop Type	Random Initialization	VF Initialization	VF Model
Open Loop	0.3342	0.3892	0.3889
Closed Loop	0.3019	0.3995	0.4025

Table 3.5: Average test relative error comparison between Rational (random initialization and VF initialization) and VF models for 1,000 test samples.

The results in this table indicate that the rational neural net with random initialization has a slightly better performance than the VF algorithm, and the rational neural net with VF initialization performs the same as the VF model. The bode plots for the closed loop transfer function in Figures 3.14 and 3.15 further solidify these results.

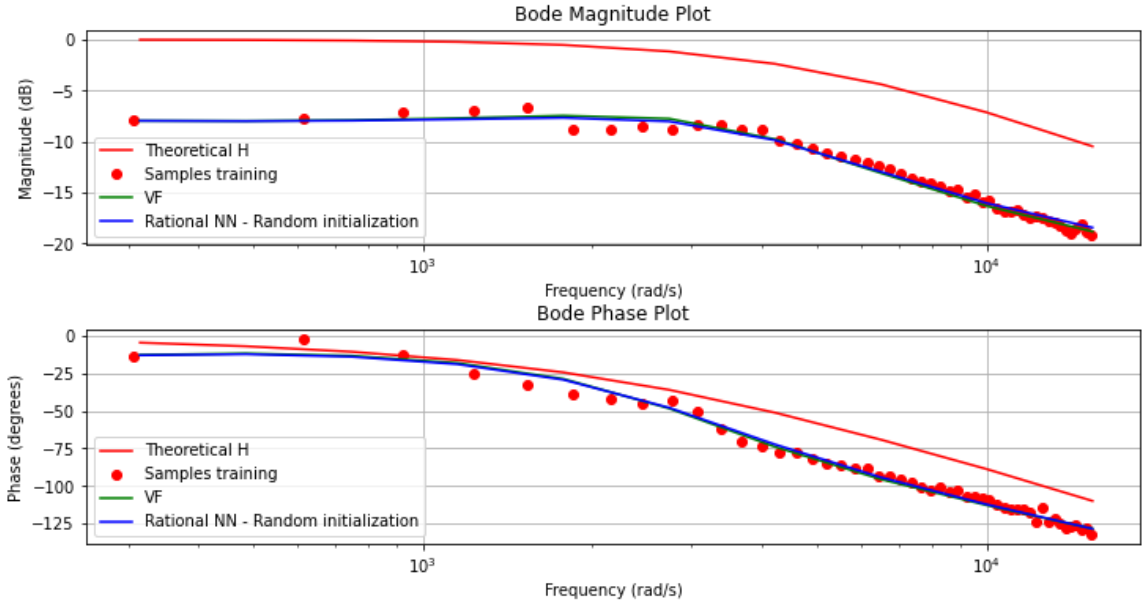


Figure 3.14: Performance of Rational Neural Net initialized with VF, trained with 50 real samples for closed loop.

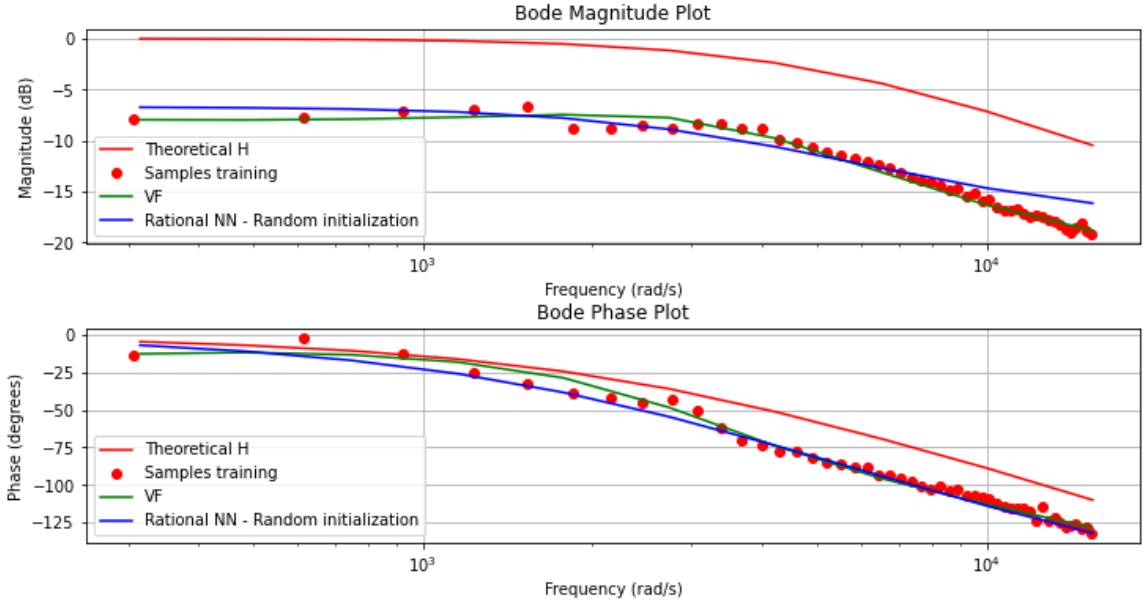


Figure 3.15: Performance of Rational Neural Net initialized with random coefficients, trained with 50 real samples for closed loop.

3.6 Transfer Learning

One of the main challenges in our project is the limited availability of real data for training and testing our neural networks. Generating real data is a time-consuming process, which constrains the amount of data we can utilize. To mitigate the issue of sparse data, we employ a technique known as Transfer Learning.

Transfer Learning involves pre-training a model on a large dataset and then fine-tuning

it on a different, but related, dataset. The key idea is that while the datasets may differ, the learning objective remains similar. This method leverages the knowledge gained from the initial training to enhance the model's performance on the new dataset, especially when data is scarce [3].

In our context, we first train the model using synthetically generated data, consisting of various frequencies and their corresponding H values across a range of transfer functions in a second-order system. After training, we test the model on a different set of transfer functions to evaluate its performance. All transfer functions in our study adhere to the following form for a second-order system:

$$H(s) = \frac{\gamma(s - z_1)}{(s - p_1)(s - p_2)} = \frac{a_1 s + a_0}{b_2 s^2 + b_1 s + b_0}$$

The parameters of the transfer functions are defined as follows:

- **Gain**: γ (*real, positive/negative constant*)
- **Zero**: z_{1r} (*real, positive/negative constant*)
- **Poles**: (p_{1r}, p_{2r}) (*real, negative constants*)
  OR
- (p_{1c}, p_{1c}^*) (*complex conjugate pairs with a negative real part, e.g.,* $(1 + 3j)$ *and* $(1 - 3j)$)

The transfer functions used to train the model are sampled from the following parameter ranges:

- **Gain range**: $[0.1, 500]$
- **Poles range**: $[-10, -3000]$
- **Zero range**: $[-500, 500]$

The neural network is trained using an input-output pair (X_i, Y_i), where X_i represents the input features and Y_i represents the output labels for each transfer function i. The input matrix X_i is composed of ℓ pairs of frequency and corresponding H values, and the output Y_i is a tuple containing the gain, zero, and poles:

$$X_i = \begin{pmatrix} \left(\omega_{i_1}, H^{(i)}(j\omega_1)\right) \\ \left(\omega_{i_2}, H^{(i)}(j\omega_2)\right) \\ \vdots \\ \left(\omega_{i_\ell}, H^{(i)}(j\omega_\ell)\right) \end{pmatrix}, \quad \text{and} \quad Y_i = (\gamma_i, z_{1i}, p_{1i}, p_{2i})\,.$$

We trained a ReLU-based neural network architecture with 5 hidden layers consisting of 3, 64, 32, 16, and 7 neurons, respectively. The input layer has 3 nodes to account for the frequency, real part, and imaginary part of $H(j\omega)$, and the output layer has 7 nodes to represent the gain, real zero, complex zero (with real and imaginary parts), and real poles or a pair of complex conjugate poles. The training was conducted on datasets comprising 10, 50, and 100 transfer functions, with 10 ω values sampled from a logarithmic scale ranging from 0 to 3. During the training process, the neural network was fully trained on each $H_i(j\omega_l)$, and the training parameters were optimized based on the corresponding output Y_i.

For 10 training transfer functions, the average test relative error between a synthetic transfer test function from a second-order system, where $H_{\text{test}} \notin X_{\text{train}}$, and the approximated transfer function by the model was 0.971080. For 50 training transfer functions, the average test relative error was 0.60499. For 100 training transfer functions, the average test relative error was 0.10245. Figure 3.16 shows the approximated transfer function by the model compared to the theoretical H used for testing. The approximated H by transfer learning closely matches the theoretical H, and VF demonstrates the model's effectiveness in accurately approximating the test transfer functions given noise-free data.

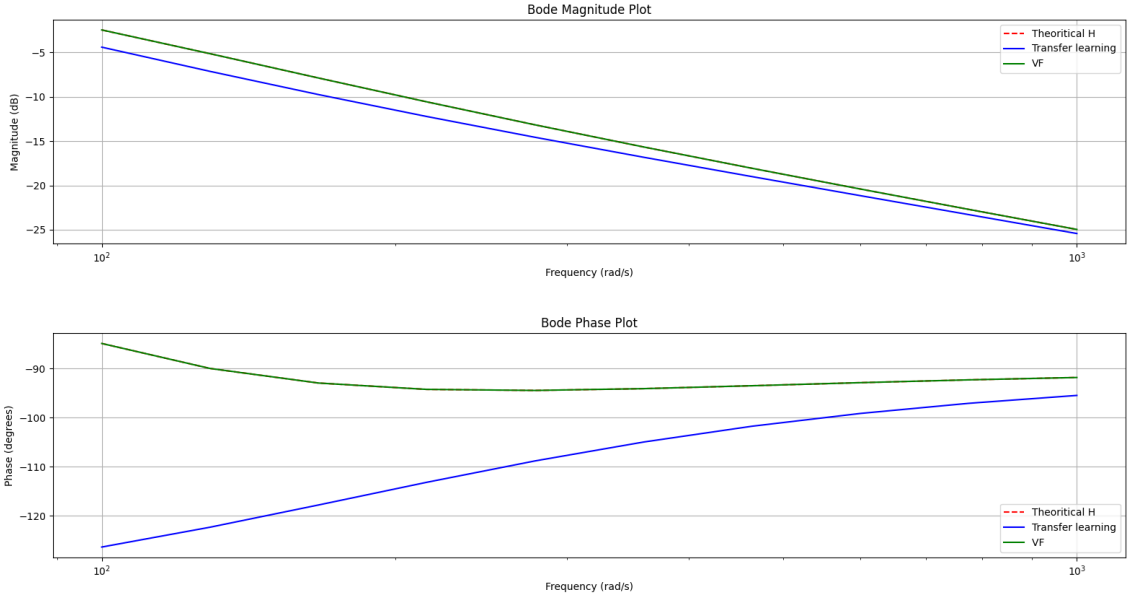


Figure 3.16: Performance of the Transfer Learning model trained with 100 transfer functions.

Chapter 4

Conclusion

This report aimed to research various data-driven transfer function modelling techniques as an alternative to physics based methods. In particular, this research involved a comparative analysis of VF and neural network based models for transfer function approximation. We began by rigorously analyzing the performance of VF across different data conditions, including synthetic, simulated, and real-world data, with varying levels of noise. A novel metric was developed to assess the accuracy of the model's performance. Our findings indicated that although VF works perfectly with accurate data, the algorithm encounters problems when noise is introduced. We then explored neural networks as a way to improve upon the performance of VF. Our findings indicate the potential of deep learning and rational neural networks to approximate transfer functions, as these models demonstrated superior performance compared to VF, especially in noisy environments.

Some issues that we encountered throughout this process involved collecting real data to test our models on. Further, when using real data, we did not have a reliable ground truth model to analyze the performance of the neural nets, thus making drawing conclusions regarding performance difficult. Finally, we encountered difficulties in model consistency during the development of the rational neural networks.

By conducting extensive numerical experiments under various data conditions, we have demonstrated the effectiveness of rational neural networks. Our findings suggest that rational neural networks offer significant advantages, particularly in noisy environments, where they consistently outperform traditional methods. This research contributes to the advancement of system identification techniques and has the potential to impact a wide range of applications reliant on accurate transfer function modeling.

A deeper investigation into transfer learning techniques could unlock additional performance gains, especially in data-scarce environments. Furthermore, expanding the analysis to encompass a wider range of real-world datasets would enhance the generalizability of the proposed models. Finally, a thorough examination of computational efficiency is warranted to identify potential optimizations and bottlenecks.

Appendix A

Plots and Tables

A.1 Test Frequencies

Test 1	Test 2	Test 3
97.65625	48.828125	48.828125
146.484375	97.65625	97.65625
292.96875	146.484375	146.484375
488.28125	195.3125	195.3125
781.25	292.96875	244.140625
1269.53125	439.453125	292.96875
2148.4375	683.59375	341.796875
3564.453125	1025.390625	390.625
5957.03125	1611.328125	...
10009.765625	2490.234375	2441.40625

Table A.1: Table of frequencies for experiments.

Note that Test 3 is 50 frequencies that are equally spaced apart by a factor of $\frac{25000}{512}$.

A.2 Testing Vector Fitting on Simulated Data

A.2.1 Current Open Loop

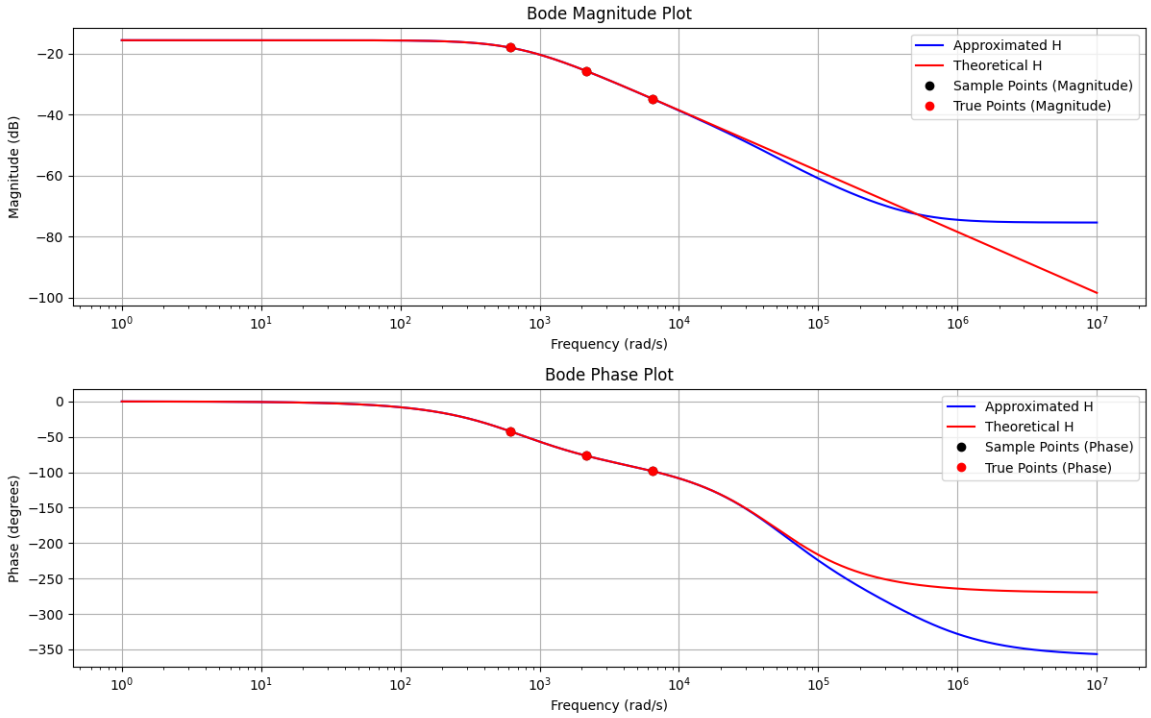


Figure A.1: Bode plot for open loop model trained on 3 sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (5.23e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (1.00e-04)
Final Model Error	1.17e-07
Average Relative Error	1.02e-05
Average Relative Error (random points)	7.18e-05

Table A.2: Table of VF outputs for open loop model trained on 3 sine waves.

The 3 frequencies inputted into the VF algorithm were 97.65625, 341.796875, 1025.390625 Hz. As seen from the table of output, three frequencies were sufficient for VF to approximate the theoretical TF in 2 iterations with a model convergence error of 1.17e-07.

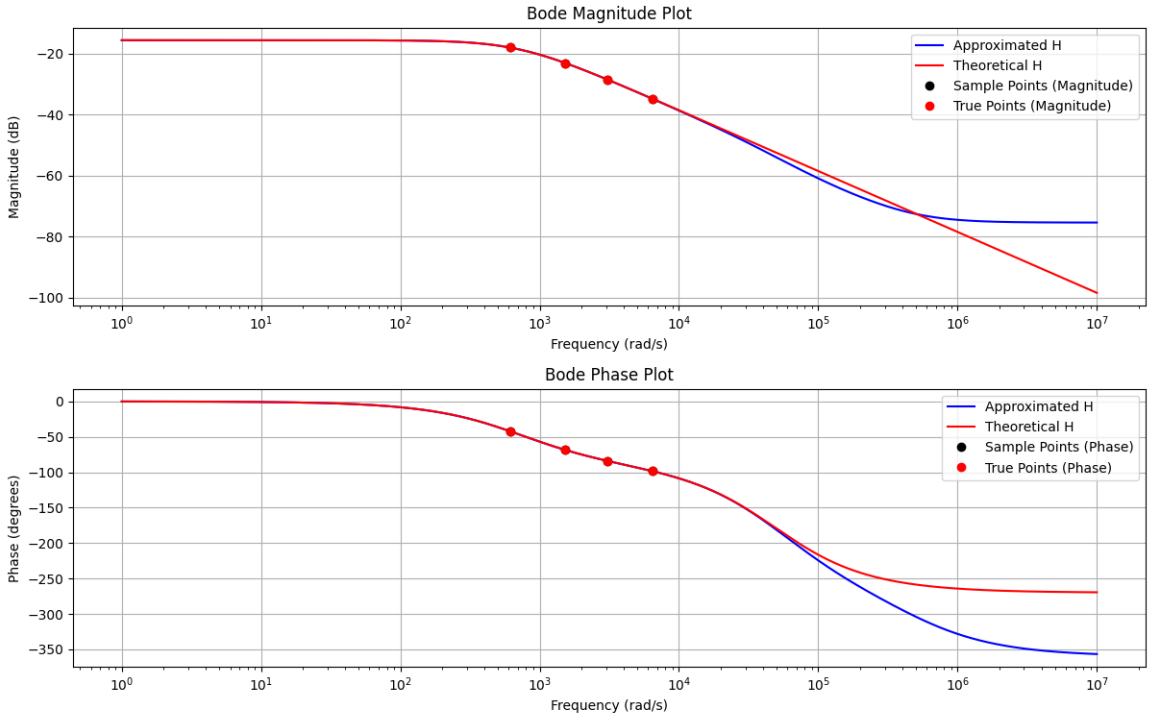


Figure A.2: Bode plot for open loop model trained on 4 sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (5.44e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (1.69e-04)
Final Model Error	1.44e-07
Average Relative Error	7.96e-06
Average Relative Error (random points)	1.01e-06

Table A.3: Table of VF outputs for open loop model trained on 4 sine waves.

The 4 frequencies inputted into the VF algorithm were 97.65625, 244.140625, 488.28125, 1025.390625 Hz. Two iterations were needed for model convergence. From the table, the model error was 1.44e-07, which was larger than the error in Table A.2 for 3 sine waves.

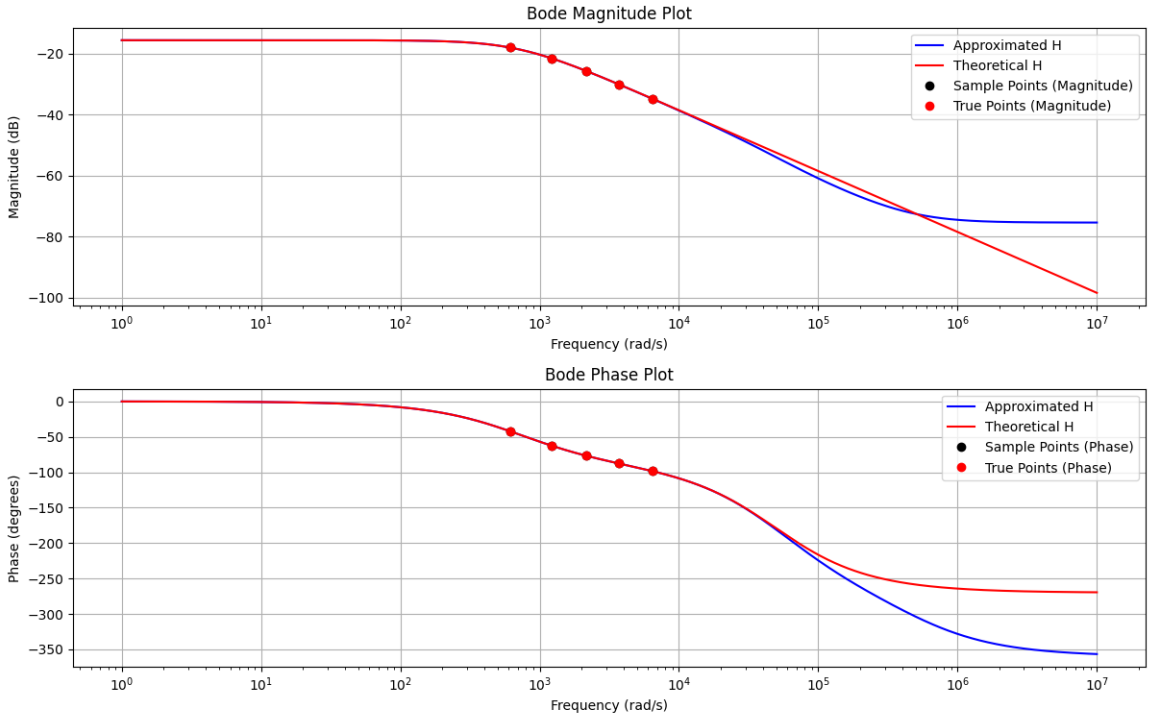


Figure A.3: Bode plot for open loop model trained on 5 sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (5.51e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (1.79e-04)
Final Model Error	1.48e-07
Average Relative Error	6.77e-06
Average Relative Error (random points)	1.01e-06

Table A.4: Table of VF outputs for open loop model trained on 5 sine waves.

The 5 frequencies inputted into the VF algorithm were 97.65625, 195.3125, 341.796875, 585.9375, 1025.390625 Hz. Two iterations were needed for model convergence. From the table, the model error was 1.48e-07, which was larger than the error in Table A.2 for 3 sine waves and Table A.3 for 4 sine waves.

A.2.2 Current Closed Loop

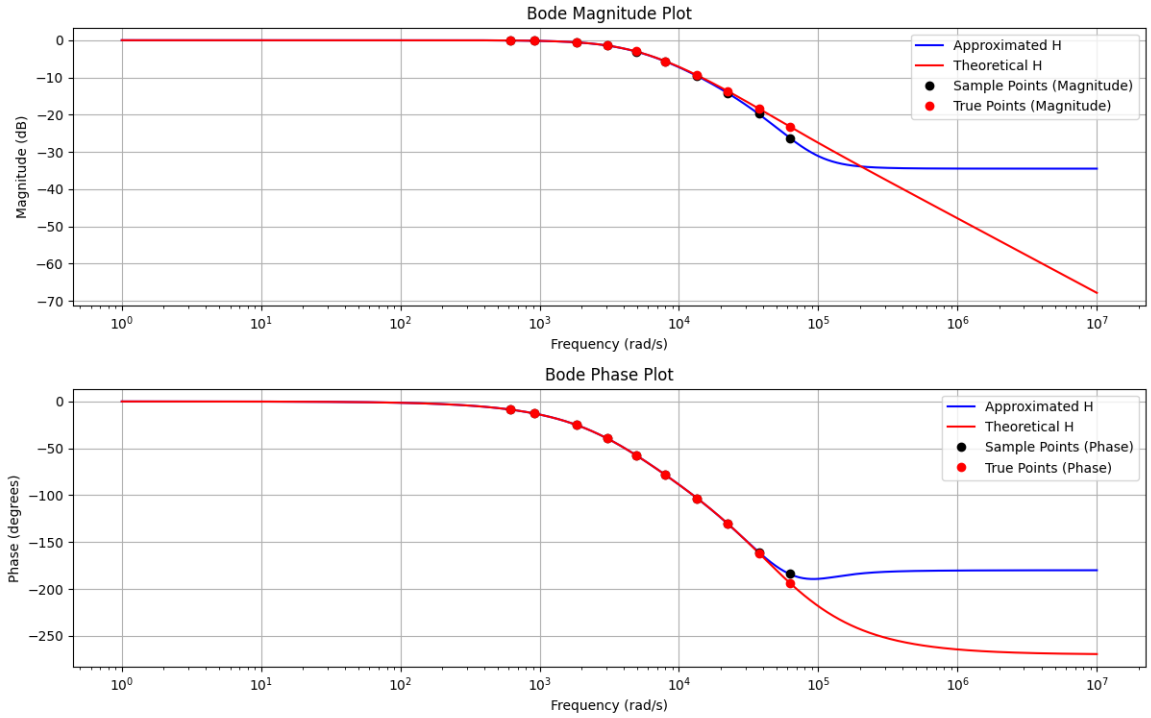


Figure A.4: Bode plot for closed loop model trained on 10 sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (2.39e+00)
Iteration n.2	
Convergence Test (poles estimation)	passed (2.08e-03)
Final Model Error	4.74e-04
Average Relative Error	1.37e-02
Average Relative Error (random points)	4.09e-03

Table A.5: Table of VF outputs for closed loop model trained on 10 sine waves.

The 10 frequencies inputted into the VF algorithm were 97.65625, 146.484375, 292.96875, 488.28125, 781.25, 1269.53125, 2148.4375, 3564.453125, 5957.03125, 10009.765625 Hz. Two iterations were needed for model convergence. From the plots, we observed that for large frequencies, there was more error between our sample points and true points.

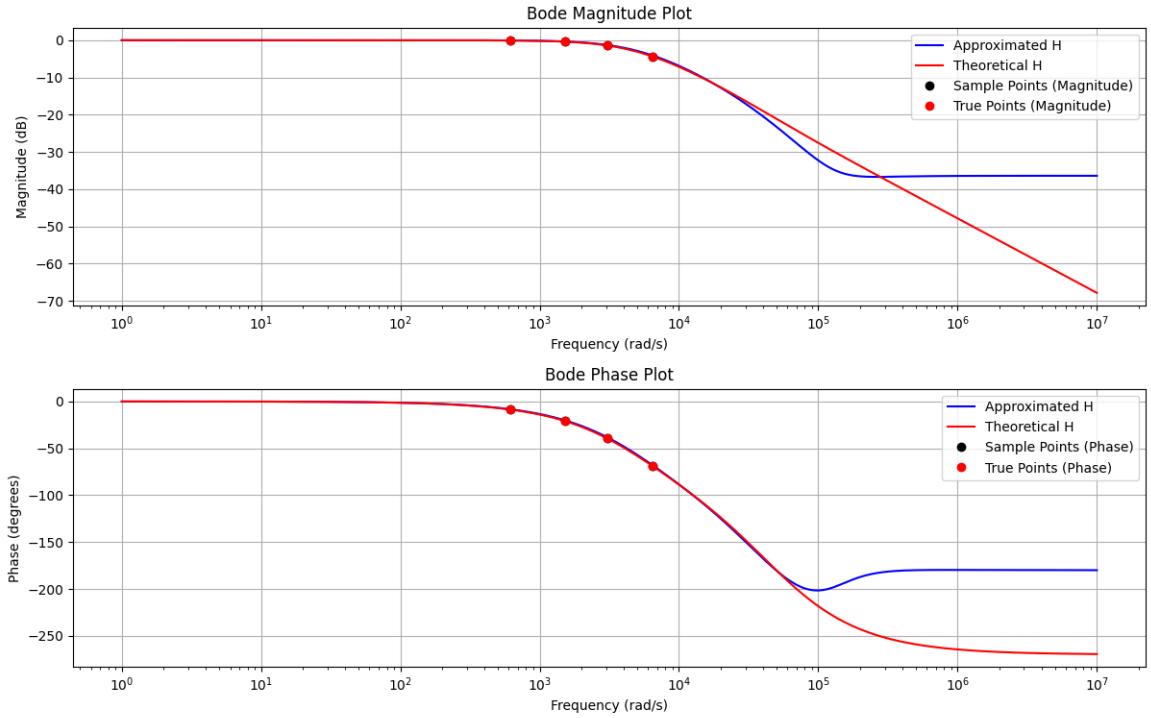


Figure A.5: Bode plot for closed loop model trained on 4 sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (6.83e+02)
Iteration n.2	
Convergence Test (poles estimation)	passed (3.84e-05)
Final Model Error	1.53e-10
Average Relative Error	7.89e-06
Average Relative Error (random points)	2.29e-06

Table A.6: Table of VF outputs for closed loop model trained on 4 sine waves.

The 4 frequencies inputted into the VF algorithm were 97.65625, 244.140625, 488.28125, 1025.390625 Hz. Two iterations were needed for model convergence. As the frequencies were of a smaller range than the ones in A.4, we observed that the results of VF were more accurate as the sample points were closer to the true points.

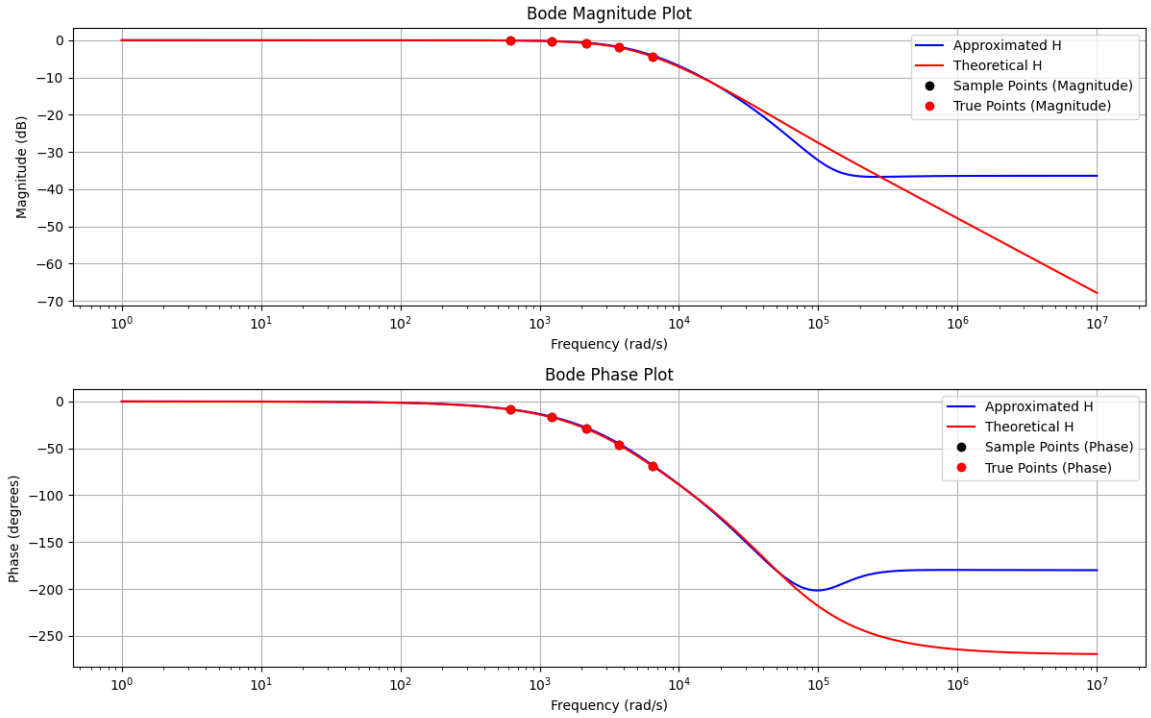


Figure A.6: Bode plot for closed loop model trained on 5 sine waves.

Iteration n.1	
Convergence Test (poles estimation)	failed (6.51e+02)
Iteration n.2	
Convergence Test (poles estimation)	passed (1.43e-04)
Final Model Error	5.313e-10
Average Relative Error	6.71e-06
Average Relative Error (random points)	2.29e-06

Table A.7: Table of VF outputs for closed loop model trained on 5 sine waves.

The 5 frequencies inputted into the VF algorithm were 97.65625, 195.3125, 341.796875, 585.9375, 1025.390625 Hz. Two iterations were needed for model convergence. From the plots, we observed that VF was doing a really good job at approximating our theoretical TF with high accuracy. The model error was 5.313e-10, which was one of the smallest we've seen so far.

Selected Bibliography Including Cited Works

[1] N. Boullé, Y. Nakatsukasa, and A. Townsend, *Rational neural networks*. https://arxiv.org/abs/2004.01902, (2020).

[2] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of computation, 19 (1965), pp. 297–301.

[3] A. Hosna, E. Merry, J. Gyalmo, et al., *Transfer learning: a friendly introduction*, Journal of Big Data, 9 (2022).

[4] V. Kotsovsky, A. Batyuk, and M. Yurchenko, *New approaches in the learning of complex-valued neural networks*, in 2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP), IEEE, (2020), pp. 50–54.

[5] J. Lee, *Understanding motors*. https://www.youtube.com/playlist?list=PLaBr_WzeIAixidGwqfcrQlwKZX4RZ2E7D, (2020). Accessed 2024-07-01.

[6] S. Lefteriu and A. C. Antoulas, *On the convergence of the vector-fitting algorithm*, IEEE transactions on microwave theory and techniques, 61 (2013), pp. 1435–1443.

[7] E. Por, M. van Kooten, and V. Sarkovic, *Nyquist–Shannon sampling theorem*, Leiden University, 1 (2019), pp. 1–2.

[8] J. Santarcangelo, *IBM: Pytorch basics for machine learning*. https://www.edx.org/learn/pytorch/ibm-pytorch-basics-for-machine-learning. Accessed 2024-07-01.

[9] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, *Deep complex networks*, arXiv preprint arXiv:1705.09792, (2017).

[10] P. Triverio, *Vector fitting*, Handbook on Model Order Reduction; Benner, P., Grivet-Talocia, S., Quarteroni, A., Rozza, G., Schilders, WHA, Silveira, LM, Eds, (2019), pp. 275–310.

[11] T. A. Tutunji, *Approximating transfer functions using neural network weights*, in 2009 4th International IEEE/EMBS Conference on Neural Engineering, IEEE, (2009), pp. 641–644.