4-2015

An Intelligent Nonlinear System Identification Method with an Application to Condition Monitoring

Clara Echavarria

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation
An Intelligent Nonlinear System Identification Method with an Application to Condition Monitoring

by

Clara Echavarria

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Mechanical Engineering

Supervised by

Associate Professor Dr. Jason Kolodziej
Department of Mechanical Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
April 2015
Acknowledgments

I would like to thank the very knowledgeable faculty that I was able to interact with and learn from over the course of my years at the Rochester Institute of Technology. They gave me the opportunity to learn much more than I thought I would during classes and office visits. Special thanks to my advisor, Dr. Kolodziej, for his time and guidance in completing this endeavor, and for all the insight he provided me with as a professor and thesis advisor.

I am also very grateful to the staff members in the Mechanical Engineering Department. Always friendly and willing to help.

Lastly, I would like thank my family for their patience and unwavering support.
Abstract

An Intelligent Nonlinear System Identification Method with an Application to Condition Monitoring

Clara Echavarria
Supervising Professor: Dr. Jason Kolodziej

Neural networks are black-box model structures that map inputs to outputs and do not require underlying mathematical models. They are frequently used in the field of system identification, the area that deals with the development of system models based on input-output data. In this work, a hybrid system identification method is implemented with neural networks (NN) and the Minimum Model Error estimator (MME) on different benchmark experimental setups, as well as simulations. The MME algorithm uses a cost function with a covariance constraint to determine smooth state estimates of a system given noisy measurement data and an assumed model. As a byproduct, it generates a vector of unmodeled nonlinear (or linear) system dynamics, which can then be modeled by a neural network. The purpose of neural networks in this research is two-fold: to demonstrate the advantages of combined MME/NN models over some common system identification methods and to investigate the feasibility of using the data stored in the network structure of those models to develop a classification scheme for condition monitoring. The approach to classification that is used in this research does not lead to successful implementation of such a scheme.
# Contents

Acknowledgments........................................................................................................... ii

Abstract............................................................................................................................ iii

List of Figures....................................................................................................................... vii

List of Tables....................................................................................................................... xii

Chapter 1: Introduction ...................................................................................................... 1
  1.1 Background ................................................................................................................. 1
  1.2 Motivation ................................................................................................................... 3

Chapter 2: Minimum Model Error Estimation ................................................................. 4
  2.1 General Problem Statement from Optimal Control ................................................. 4
  2.2 The Covariance Constraint ....................................................................................... 5
  2.3 Weight Matrix Selection ......................................................................................... 6
  2.4 Estimation Algorithm and Boundary Value Problem .......................................... 6
  2.5 Mass-Spring-Damper System with Cubic Spring ................................................... 8
  2.6 Algorithm Output ..................................................................................................... 12

Chapter 3: Artificial Neural Networks ............................................................................ 13
  3.1 Network Design ....................................................................................................... 15
  3.2 Dynamic Networks .................................................................................................. 17
    3.2.1 Input and Feedback Delays ............................................................................. 19

Chapter 4: MME/NN Combined Algorithm ...................................................................... 21
  4.1 Single pendulum Simulation .................................................................................... 23
    4.1.1 Physics Equations ............................................................................................ 24
    4.1.2 Clean Data ....................................................................................................... 26
    4.1.3 Noisy Data ...................................................................................................... 30
Chapter 5: Comparison to Other System Identification Methods .......................... 65

Chapter 6: Application of MME/NN Algorithm to Classification .......................... 81

Chapter 7: Summary and Conclusions  .................................................................. 91

4.2 Single Pendulum Experiment ........................................................................... 35
4.3 Double Pendulum Simulation ........................................................................... 38
  4.3.1 Physics Equations ....................................................................................... 39
  4.3.2 Ignorance Model ......................................................................................... 41
  4.3.3 Updated Model ............................................................................................ 46
4.4 Double Pendulum Experiment ........................................................................... 51
4.5 Two-Tank Experiment ....................................................................................... 55
  4.5.1 Physics Equations ....................................................................................... 56
  4.5.2 Sampling Rate: $f_s = 10$ Hz ................................................................... 57
  4.5.3 Sampling Rate: $f_s = 5$ Hz ..................................................................... 61
  4.5.4 Sampling Rate: $f_s = 1$ Hz ..................................................................... 62
  4.5.5 Sampling Rate: $f_s = 0.5$ Hz ................................................................... 63
4.6 Summary ......................................................................................................... 64

Chapter 5: Application of MME/NN Algorithm to Classification .......................... 81

Chapter 6: Summary and Conclusions  .................................................................. 91

7.1 Future Work ..................................................................................................... 92
  7.1.1 MME/NN Hybrid Algorithm ..................................................................... 92
  7.1.2 Condition Monitoring ................................................................................. 93
Bibliography.................................................................................................................................93

Appendix A: Simulated Systems Diagrams..................................................................................95

Appendix B: Algorithm to Train and Test Neural Networks......................................................101

Appendix C: Function to Train Networks to Map States to d(t)................................................107

Appendix D: Function to Implement Complete MME/NN Plant Models.................................109

Appendix E: Simulink Diagram of MME/NN Plant Model.......................................................111
List of Figures

Figure 2.1: Schematic representation of the MME algorithm ........................................ 4
Figure 2.2: State 1 - measurement data, truth data, and assumed model output ................. 9
Figure 2.3: State 2 – truth data and assumed model output ........................................... 9
Figure 2.4: State x_1 estimate, $\sigma_{2meas - est} \approx 0.09$, $\sigma_{2est - truth} \approx 0.03$ .......... 10
Figure 2.5: State 2 estimate, $\sigma_{2est - truth} \approx 0.70$ ............................................... 11
Figure 2.6: Estimated unmodeled dynamics, $d(t)$ ......................................................... 11
Figure 3.1: Basic neural network architecture - $h =$ hidden layer, $N =$ hidden node, $O =$ output node ........................................................................................................... 14
Figure 3.2: Conceptual representation of overfitting and underfitting ................................. 16
Figure 3.3: NARX Neural Network: $u_k =$ external input, $w_k =$ network output, $q =$ number of input delays, $n =$ number of output delays, $Z - 1 =$ delay operator, $k =$ sample number ........ 18
Figure 4.1: The output data from MME is passed to the neural network ............................. 22
Figure 4.2: Block diagram incorporating the neural network into the plant model after training 22
Figure 4.3: Single pendulum schematic ............................................................................ 23
Figure 4.4: Acceleration inputs - single pendulum simulation. Top: Training set input, Bottom: Test set input ................................................................................................................ 23
Figure 4.5: Performance of trained neural networks on clean data sets ............................ 26
Figure 4.6: State predictions for training data set – clean single pendulum data. Top: State $x_1$(angular displacement), Bottom: State $x_2$ (angular velocity) ................................................. 27
Figure 4.7: State predictions for test data set – clean single pendulum data. Top: State $x_1$(angular displacement), Bottom: State $x_2$ (angular velocity) ......................................................... 28
Figure 4.8: Truth and estimated model correction terms – clean single pendulum data. Top: training set $d(t)$, Bottom: test set $d(t)$ .................................................................................. 29
Figure 4.9: Performance of trained neural networks on noisy data sets ............................ 30
Figure 4.10: MME State estimates for **training** data set (with weight matrix $W = 1001.85$).
Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity) ......................... 31
Figure 4.11: MME/NN State estimates for **training** data set – noisy single pendulum data. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity) ................................. 32
Figure 4.12: MME/NN State estimates for **test** data set – noisy single pendulum data. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity) ......................... 33
Figure 4.13: Truth and estimated model correction terms – noisy single pendulum data. Top: Training set $d(t)$, Bottom: Test set $d(t)$ ................................................................. 34
Figure 4.14: Acceleration inputs - compound pendulum experiment. Top: Training set input, Bottom: Test set input ................................................................................................................................. 35
Figure 4.15: State predictions for **training** data set – compound pendulum experiment. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity) ......................... 36
Figure 4.16: State predictions for **test** data set – compound pendulum experiment. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity) ................................. 37
Figure 4.17: Compound double pendulum schematic ...................................................................... 38
Figure 4.18: Acceleration inputs - double pendulum simulation. Top: Training set input, Bottom: Test set input ................................................................................................................................. 38
Figure 4.19: Estimated model correction terms for **training** data – double pendulum with ignorance model. Top: Correction term for $\theta_1$, Bottom: Correction term for $\theta_2$ ......................... 42
Figure 4.20: Estimated model correction terms for **test** data – double pendulum with ignorance model. Top: Correction term for $\theta_1$, Bottom: Correction term for $\theta_2$ ......................... 43
Figure 4.21: State predictions for **training** data set – double pendulum with ignorance model. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link) ................................................................................................................................. 44
Figure 4.22: State predictions for **test** data set – double pendulum with ignorance model. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link) ................................................................................................................................. 45
Figure 4.23: Estimated model correction terms for **training** data – double pendulum with updated model. Top: Correction term for $\theta_1$, Bottom: Correction term for $\theta_2$ ......................... 47
Figure 4.24: Estimated model correction terms for test data – double pendulum with updated model. Top: Correction term for $\theta_1$, Bottom: Correction term for $\theta_2$...

Figure 4.25: State predictions for training data set – double pendulum with updated model. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link).

Figure 4.26: State predictions for test data set – double pendulum with updated model. Top to Bottom: State $x_1$, State $x_2$, State $x_3$, State $x_4$.

Figure 4.27: Acceleration inputs - double pendulum experiment. Top: Training set input, Bottom: Test set input.

Figure 4.28: Estimated model correction terms for training data – double pendulum experiment. Top: Correction term for $\theta_1$, Bottom: Correction term for $\theta_2$.

Figure 4.29: State predictions for training data set – double pendulum experiment. Top to Bottom: State $x_1$, State $x_2$, State $x_3$, State $x_4$.

Figure 4.30: State predictions for test data set – double pendulum experiment. Top to Bottom: State $x_1$, State $x_2$, State $x_3$, State $x_4$.

Figure 4.31: Two-tank experiment setup.

Figure 4.32: Measurement set and MME estimates – two-tank experiment with $f_s = 10$ Hz. Top to Bottom: Applied voltage, Water level in the upper tank (m), Water level in the lower tank (m).

Figure 4.33: Training/Testing Data distribution for neural network. From Top to Bottom, Left to Right: Pump voltage, Estimated water level in the upper tank ($x_1$), Estimated water level in the lower tank ($x_2$), Model correction for state $x_1$, Model correction for state $x_2$.

Figure 4.34: State predictions – two-tank experiment with $f_s = 10$ Hz. Top: Water level in the upper tank, Bottom: Water level in the lower tank.
Figure 4.35: State predictions – two-tank experiment with $f_s = 5$ Hz. Top: Water level in the upper tank, Bottom: Water level in the lower tank.

Figure 4.36: State predictions – two-tank experiment with $f_s = 1$ Hz. Top: Water level in the upper tank, Bottom: Water level in the lower tank.

Figure 4.37: State predictions – two-tank experiment with $f_s = 0.5$ Hz. Top: Water level in the upper tank, Bottom: Water level in the lower tank.

Figure 5.1: Model structure of for the output error method, $G = $ system transfer function

Figure 5.2: Correlation plots – clean single pendulum data. Left: Partial autocorrelation signal, Right: Cross-correlation signal.

Figure 5.3: State predictions for clean single pendulum data. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity)

Figure 5.4: Correlation plots – noisy single pendulum data. Left: Partial autocorrelation signal, Right: Cross-correlation signal.

Figure 5.5: State predictions for noisy single pendulum data. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity)

Figure 5.6: Correlation plots – single pendulum experiment. Left: Partial autocorrelation signal, Right: Cross-correlation signal.

Figure 5.7: State prediction for $x_1$ for single pendulum experiment.

Figure 5.8: Correlation plots – double pendulum simulation. From Left to Right, Top to Bottom: Partial autocorrelation of State $x_1$, Cross-correlation between State $x_1$ and the input $u(t)$, Partial autocorrelation of State $x_2$, Cross-correlation between State $x_2$ and the input $u(t)$.

Figure 5.9: State predictions for double pendulum simulation. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link).

Figure 5.10: Correlation plots – double pendulum experiment. From Left to Right, Top to Bottom: Partial autocorrelation of State $x_1$, Cross-correlation between State $x_1$ and the input $u(t)$, Partial autocorrelation of State $x_2$, Cross-correlation between State $x_2$ and the input $u(t)$.

Figure 5.11: State predictions for double pendulum experiment. Top: State $x_1$ (angular displacement of the top link), Bottom: State $x_2$ (angular displacement of the bottom link).
Figure 5.12: Correlation plots – two-tank experiment. From Left to Right, Top to Bottom: Partial autocorrelation signal of state $x_1$, Cross-correlation signal between state $x_1$ and the input $u(t)$, Partial autocorrelation signal of state $x_2$, Cross-correlation signal between state $x_2$ and the input $u(t)$. ................................................................. 78

Figure 5.13: State predictions for two-tank experiment. Top: Water level in the upper tank, Bottom: Water level in the lower tank ................................................................. 79

Figure 6.1: Neural network model of equation (6.3) for the unmodeled dynamics of the mass-spring-damper system ................................................................. 83

Figure 6.2: PCA reduces the number of features from 5 to 2. Class separation is possible. ....... 87

Figure 6.3: Output $d(t)$ terms for different damping conditions – double pendulum simulation data ................................................................. 88

Figure 6.4: First 2 components of the transformed data for the 1-node networks used to model the double pendulum system ................................................................. 89

Figure 7.1: Block diagram incorporating the neural network into the plant model after training 92
List of Tables

Table 5-1: Model performance – clean single pendulum data .................................................. 68
Table 5-2: Model performance – noisy single pendulum data .................................................. 70
Table 5-3: Model performance – single pendulum experiment ................................................. 72
Table 5-4: Model performance – double pendulum simulation ................................................. 74
Table 5-5: Model performance – double pendulum experiment ................................................. 77
Table 5-6: Model performance – two-tank experiment ............................................................ 79
Table 6-1: Input weight distributions for “good” networks (mse < 1) ........................................ 85
Table 6-2: Model performance with parameters from 3-node networks used to model the mass-
        spring-damper system ........................................................................................................ 86
Table 6-3: Model performance with parameters from 1-node networks used to model the mass-
        spring-damper system ........................................................................................................ 87
Table 6-4: Model performance with parameters from 6-node networks used to model the double
        pendulum system .............................................................................................................. 89
Table 6-5: Model performance with parameters from 1-node networks used to model the double
        pendulum system .............................................................................................................. 90
Chapter 1

Introduction

1.1 Background

System identification is the field that deals with the development of mathematical models of
dynamical systems based on measured input and output signals from the systems. The models
are inferred by relying heavily on statistical analysis of the available data. Over the years, many
algorithms and methodologies have been developed for this purpose, such as the well-known
Least Squares Regression algorithm. Due to the abundance of techniques that have been
developed for diverse applications, there is not a single method of performing model
identification for a given system. Instead, there is an assortment of methods available that can be
applied depending on the nature of a problem. Furthermore, there is continuous undergoing
research that aims to develop improved model estimation techniques, mostly concerning systems
with inherent nonlinear behavior. Such systems are of particular interest since most physical
systems are in fact nonlinear.

The field of system identification includes everything from the design of experiments
used to gather input/output signals, to the data analysis that leads to the acquisition of a
mathematical model of the system under consideration. The quality of the final model derived
depends highly on the quality of the collected data.

One approach to system identification involves the combination of two or more methods
(see for example [1] and [2]). This allows the user to take advantage the best traits of each
method and to obtain an overall improved algorithm that is potentially better or simpler to use
than the individual ones.

The Minimum Model Error estimator (MME) was first presented in 1988 by Mook and
Junkins [3]. It has since been improved upon and implemented, in some cases, with a few
modifications [4]-[8]. MME assumes a linear model of a dynamic system \( \dot{x} = f(x(t), u(t), t) \) and outputs smooth state estimates based on measured state data and an assumed covariance.

Apart from smooth state estimates, \( \hat{x}(t) \), the MME algorithm also generates a time vector of unmodeled dynamics, \( d(t) \), that serves as a correction to the assumed model (further details regarding the theoretical foundation of MME can be found in Chapter 2. In 1992, Stry attempted to use a function library to develop explicit input-output relationships between the system states and the correction term [5]. The idea was to add functions to an ongoing expression for \( d(t) \) in order to continuously improve the functional relationship between the states and the model correction term, until the results could be improved no further. In 2001, Kolodziej expanded the idea of a function library by implementing the ability to both add and subtract functions to the expression for \( d(t) \) via step wise regression [6]. This method proved to be an enhancement to Stry’s work, since it provided the ability to use statistical techniques to evaluate the goodness of the contribution of each function to the ongoing representation of \( d(t) \).

The last documented study on how to model the correction term was done by Kolodziej in 2011 [7]. The research performed at that time replaced function libraries with Artificial Neural Networks (ANN) that used the estimated system states as inputs and the correction vector as output. This method took advantage of the black-box nature of neural networks and eliminated the need to obtain specific input-output relationships between the states and the correction terms to complete a system plant model.

The benefits of the combined methodology were demonstrated through the implementation of a MME/NN model on a simulated Van der Pol oscillator. The results were compared to the results of a stand-alone neural network, and the hybrid algorithm performed significantly better than the neural network. With the purpose of keeping the number of calculations between the two methods similar, the architecture of the network used in conjunction with MME was the same as the one used for the stand-alone network model. It was a static, feedforward neural network with five neurons, or nodes, in a single hidden layer. However, given that the data came from a dynamic system, a static network architecture was at a disadvantage with respect to the hybrid algorithm from the start and was therefore met with suboptimal results.
1.2 Motivation

Although the work by Kolodziej was effective in showing how to implement the combined algorithm, it lacked a robust methodology for the design of the neural networks (both the one in the hybrid algorithm and the stand-alone network used for performance comparisons.) Furthermore, the research for a hybrid MME/NN model relied solely on simulation data. The present work aims to implement the method on real systems. More specifically, mathematical models will be derived from first principles and will then be improved upon by means of a MME/NN methodology. Additionally, the performance of this method will be compared against properly designed stand-alone neural networks and against parametric state-space models.

The second aim of this research is to investigate the use of neural network parameters as predictor features in condition monitoring systems. To that end, the weights of neural networks from MME/NN models will be used as inputs to different classifiers. Neural networks are also commonly found in the pattern recognition literature, but as classifiers [8], not as feature-generating structures. If feasible, such a method could lead to an integrated modeling and monitoring scheme.
Chapter 2

Minimum Model Error Estimation

The Minimum Model Error algorithm is an optimal control approach to dynamic system identification. It works under the assumption that, given a set of state-observable data (not necessarily full state data) and a mathematical model for a system, the measurement-minus-estimate error covariance matrix must match the measurement-minus-truth error covariance matrix. This constraint arises from the idea that measurement devices contain some degree of uncertainty due to noise and, as a result of this uncertainty, the states can only be estimated to approximately the same variance as that of the measurement device to avoid tracking noise.

![Diagram](image)

Figure 2.1: Schematic representation of the MME algorithm

2.1 General Problem Statement from Optimal Control

Given a dynamic system defined by

\[ \dot{x} = f[x(t), u(t), t] \]  

(2.1)

Where

- \( x = n \times 1 \) state vector,
- \( f = n \times 1 \) vector of model equations,
- \( u = r \times 1 \) vector of input terms,
And given a set of discrete time, state-observable measurements

\[
\tilde{y}_k = g_k(x(t_k), t_k) + v_k
\]

\[k = 1, ..., m\] (2.2)

Where

\[
\tilde{y}_k = m \times 1 \text{ measurement set at time } t_k,
\]

\[g_k = m \times 1 \text{ measurement model at time } t_k,
\]

\[v_k = m \times 1 \text{ Gaussian distributed noise with zero mean and known covariance, } R_k,
\]

Determine the optimal state trajectory estimates, \(\hat{x}(t)\), for the given time interval \(t_0 \leq t \leq t_f\).

### 2.2 The Covariance Constraint

The covariance constraint requires the covariance of the residuals between measured and estimated states to be approximately equal. It can be mathematically expressed as

\[
E\{[\tilde{y}_k - g_k(\hat{x}(t_k), t_k)][\tilde{y}_k - g_k(\hat{x}(t_k), t_k)]^T\} \approx R_k
\]

(2.3)

Ultimately, it means that the model estimate of a system has to match the measured data with approximately the same error covariance that the measured data matches the truth data. Given a poorly modeled system, this constraint can only be met if there is a correction to the model. In the case of MME, the correction is achieved by adding an unmodeled correction term, \(d(t)\), to an originally assumed linear model of the system:

\[
\dot{x} = f[(x(t), u(t), t) + d(t)]
\]

(2.4)

The assumed model may be analytically derived or found through other linear system identification techniques. The unmodeled correction term can be interpreted as an error or disturbance vector. That is, the amount by which the assumed model estimates miss the underlying truth states. There are several reasons why the assumed model might not fit the measured data. Noise is one of them, but it could also be that the system contains nonlinearities for which linear approximations do not suffice. Going back to the optimal control problem, the cost function used by the MME algorithm is the following:

\[
J = \sum_{k=1}^{M} [\tilde{y}_k - g_k(\hat{x}(t_k), t_k)]^T R_k^{-1} [\tilde{y}_k - g_k(\hat{x}(t_k), t_k)] + \int_{t_0}^{t_f} d(t)^T W d(t) \, dt
\]

(2.5)
Where $W$ is a positive definite, $n \times n$ weight matrix for the model correction term, $d(t)$. The value of $W$ is determined such that the covariance constraint is met.

Notice that the first term in the cost function is the weighted sum of square residuals at each measurement time, which comes from the covariance constraint. The weighting parameter for the residuals is the inverse of the error covariance from the measurement device, $R_k^{-1}$. Consequently, data with little noise penalizes deviation of the estimates from the measurements more heavily than data with significant noise. The second term of the cost function integrates the model correction term over the time interval of the measured data. The value of $W$, which is adjusted by the user, penalizes the correction term such that the estimated states vary between those obtained from the assumed model and those obtained from measurement data. A value of $W$ equal to zero places no penalty on the model correction term such that the estimates match the measured data exactly. On the other hand, a value of $W$ equal to infinity yields no correction, such that the estimates match the assumed model exactly. Successful minimization of the cost function generates optimal state estimates and model correction terms.

### 2.3 Weight Matrix Selection

Selection of the weight matrix can be done in different ways. One is to vary the value of $W$ over a large range and examine the resulting plot of measurement-minus-estimate covariance vs. $W$ to find a minimum. A minimum value may be found in such plot since there are two effects at play in the minimization of $J$: a small weight matrix results in a large $d(t)$, which decreases the cost of the first term of $J$ but at the same time increases the cost of the second term, and vice versa. A less systematic, yet simple approach is to choose the value of $W$ on a trial-and-error basis until a reasonable combination of estimates and error terms is obtained.

### 2.4 Estimation Algorithm and Boundary Value Problem

The necessary conditions to solve the control problem of section 2.1 are called Pontryagin's necessary conditions, since they are derived from Pontryagin's minimum principle [4]. The procedure to obtain those conditions is similar to that of a typical regulator problem.
In the context of the MME algorithm, the application of Pontryagin’s minimum principle to minimize $J$ with respect to $d(t)$ results in the following Two Point Boundary Value Problem (TPBVP):

$$\dot{x} = f(\dot{x}(t), u(t), t) + d(t) \quad (2.6)$$

$$\dot{\lambda} = -\left[\frac{\partial f}{\partial \dot{x}}\right]^T \lambda \quad (2.7)$$

$$d = -\frac{1}{2W} \left[\frac{\partial f}{\partial d}\right]^T \lambda \quad (2.8)$$

With boundary conditions

$$\dot{x}(t_0) = \text{specified} \quad (2.9)$$

$$\lambda(t_0^-) = \lambda(t_0^+) = 0$$

Where $\lambda$ represents the co-states of a classic control problem (or Lagrange multipliers in other contexts).

Recall that the cost function for MME (equation (2.5)) has both a continuous and a discrete term. The discrete term requires the modification of Pontryagin’s necessary conditions to account for the internal penalties at the measurement times (see [4] for the development of how to deal with a cost functional with both discrete and continuous penalty terms.)

Modifying Pontryagin’s necessary conditions leads to the addition of internal co-state boundary conditions given by

$$\lambda(t_k^+) = \lambda(t_k^-) + 2H_k^T R_k^{-1} \left[\ddot{y}_k - g_k(\dot{x}(t_k), t_k)\right] \quad (2.10)$$

Where

$$H_k \equiv \frac{\partial g_k}{\partial \dot{x}} \bigg|_{\dot{x}(t_k)} \quad (2.11)$$

Many techniques exist for the solution of TPBVP, some of which involve shooting methods, finite difference methods and variational methods [9], [10], [11]. In a nutshell, shooting methods convert boundary value problems to initial value problems by guessing the differential equations’ initial conditions that are not given and “shooting” (integrating forward). If the result at the final time does not match the actual boundary value, the initial guess is modified until reasonable agreement is achieved. Multiple shooting takes this a step further by adding several
shooting intervals within the time interval of the problem. For the present work, multiple shooting is used.

2.5 Mass-Spring-Damper System with Cubic Spring

This section shows the application of the MME algorithm on data from a simulated mass-spring-damper system with cubic spring response. An external input, \( u(t) \), acts as a forcing function to the system. Gaussian distributed noise with zero mean and variance of 0.09 is added to the position data, which is the only measured data provided to the MME estimator. The assumed system model is linear with a damping coefficient of zero. Thus, the unmodeled dynamics term, \( d(t) \), has to correct the system for the spring nonlinearity, the damping, and the additive noise. The truth model equation is

\[
mx'' + cx' + k(x + x^3) = u(t)
\]  
(2.12)

In state-space form, with \( x_1 \) representing the displacement and \( x_2 \) representing the velocity, the model equations are:

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
0 & 1 \\
-\frac{k}{m} & -\frac{c}{m}
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
0 \\
1
\end{bmatrix} u + \begin{bmatrix}
0 \\
-\frac{k}{m} x_1^3
\end{bmatrix}
\]  
(2.13)

With parameters

\[
m = 1 \quad c = 0.1 \quad k = 0.5
\]

\[
u = 10 \times \sin(t)
\]

\[
t = 0, 0.05, \ldots, 10
\]

\[
x_1(0) = 5 \quad x_2(0) = 0
\]

The linear assumed model provided to MME, which does not account for the spring nonlinearity or the damping term, is

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
0 & 1 \\
-\frac{k}{m} & 0
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
0 \\
1
\end{bmatrix} u
\]  
(2.14)

Figure 2.2 and Figure 2.3 show the truth and assumed data for both states, respectively. Figure 2.2 also shows the noisy data, which is the only state data provided to MME.
Figure 2.2: State 1 - measurement data, truth data, and assumed model output

Figure 2.3: State 2 – truth data and assumed model output
MME allows the user to select which states need correction. In this case, it is known for a fact that state $x_2$ is the derivative of state $x_1$ so this need not be corrected. Thus, only one error vector is needed, the one for state $x_2$. Referring to equations (2.6) and (2.14), this can be expressed as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u + \begin{bmatrix} 0 \\ d(t) \end{bmatrix} \tag{2.15}$$

Furthermore, comparison of equations (2.13) and (2.15) indicates that the truth model correction term is

$$d(t) = -\frac{k}{m} x_1^3 - \frac{c}{m} x_2$$ \tag{2.16}

Using a trial-and-error approach, the weight matrix selected is $W = \begin{bmatrix} 1e^{-10} & 0 \\ 0 & 0.115 \end{bmatrix}$. This weight matrix results in a measurement-minus-estimate variance of approximately 0.09 for the measured state, as required by the covariance constraint. The results after implementing MME are shown in the following three figures. Notice that the algorithm output is not only the smooth estimate of the measured state, $x_1$, but also the estimate of the other system state, $x_2$, for which measured data was not provided.

Figure 2.4: State $x_1$ estimate, $\sigma^2(\text{meas} - \text{est}) \approx 0.09$, $\sigma^2(\text{est} - \text{truth}) \approx 0.03
Figure 2.5: State 2 estimate, $\sigma^2 (est - truth) \approx 0.70$

Figure 2.6: Estimated unmodeled dynamics, $d(t)$
2.6 Algorithm Output

The MME estimator routine ends once smooth state estimates and a correction vector are obtained. Having the unmodeled system dynamics is a powerful tool to improve upon the assumed model and develop a plant model for further use. Ideally, it would be possible to correlate the unmodeled dynamics to linear or nonlinear functions of the states (note that any noise contained within the $d(t)$ term would have no influence here since it is assumed to be Gaussian distributed and therefore would not correlate well with the states). In the case of a truth linear model, an update to the assumed state space model could be obtained.

It is also possible to use a black-box approach to model the $d(t)$ and add it to the system equations, rather than modify them (although one could also update the state equations using linear correlations, which would leave any nonlinear modeling to the black-box device.) In the following chapters, artificial neural networks are shown to be appropriate black-box devices for this application.
Chapter 3

Artificial Neural Networks

Artificial Neural Networks are structures that mimic the workings of biological neural networks. They are capable of machine learning and pattern recognition. As defined in [12],

A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

Knowledge is acquired by the network from its environment through a learning process.
Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

Simply put, neural networks map inputs to outputs by means of interconnected neurons, or nodes, eliminating the need for explicit mathematical models between the two. In order to accomplish this, the network is first trained to learn and modify the connection weights between the neurons such that the outputs match known outputs within a certain error tolerance. A successfully trained network can further be used to extrapolate system outputs based on different inputs than those that were shown during the training phase.

Neural network structures are organized in the form of layers, each of which contains a certain number of neurons. The layers between the input layer and output layer are known as hidden layers. Figure 3.1 illustrates a simple network configuration.
Neural networks are strictly data-driven models and, as such, there is very little the user can do to incorporate a priori knowledge into them. Consequently, network performance (how well the predicted outputs match the target outputs) depends greatly on the quality of the training data. Some of the common applications of these learning machines include modeling systems with unknown parameters, prediction of time series outcomes, and pattern recognition. Since there are several types of network topologies and learning functions, the nature of the problem determines the nature of the network used.

The feedforward network architecture is the one that is most commonly encountered in the literature, due to its relative simplicity and wide range of applications. As the name suggests, the flow of information in the network is strictly forward, as opposed to other types of networks that include feedback loops. Given that the outputs are not fed back to the network, this type of structure can be referred to as a static network. Its counterpart, the dynamic network, has feedback loops that connect layer outputs to intermediate layer nodes as well as back to the input nodes.
The most widely used network training routine is the backpropagation algorithm. It performs a backwards propagation of the output layer’s error in order to update the network’s connection weights. More specifically, once a network is initialized with random weights, training patterns are passed on to it. Updating the connection weights requires the use of an optimization routine with a loss function (or cost function). The loss function to be minimized is the mean of the squared errors between the target output data and the network output data (equation (3.1)). Thus, the goal of the backpropagation algorithm is to adjust the network weights such that the mean of the squared residuals is minimized over all training cases.

$$\varepsilon(w) = \frac{1}{2N} \sum_{i=1}^{N} [d(i) - F(x(i); w)]^2$$  

(3.1)

In (3.1), $i$ is the sample number, $N$ is the total number of training samples, $d$ is the target output data, $x$ is the input data, $w$ represents the internal weights of the network, and $F$ represents the nonlinear function that the inputs get mapped through to obtain the network output. A single training sample, including input and output, looks like this: $\{x(i), d(i)\}$.

### 3.1 Network Design

Depending on a system’s complexity and desired modeling accuracy, the design and implementation of a neural network can lead to expensive algorithms in term of training time, execution time, and processing power. Sizing a neural network is often a task empirical in nature, as there are no well-developed theoretical methods that can achieve this for every type of network. Although there are some heuristics available in the literature [13], the appropriate size and architecture are problem specific. In order to avoid any theoretical complications that could arise from using some of those heuristics, a practical approach will be employed to size the neural networks throughout this work.

It is also important to mention that even after a network architecture has been chosen, the values of the connection weights and the biases after training depend on the random initial values they were given. Thus, if $N$ neural networks with the same architecture but different initial parameters are shown the same training data, the weights and biases could converge to very different values and the performance of the networks could be completely different. The sizing procedure used in this work, both for static and dynamic networks, is as follows:
• If more than one set of measurements are available for a given system, use one of them for training (including validation) and another one for testing.

• For a single measurements set, use 80% of the data for training (including validation), and the remaining 20% for testing.

• Train networks with 1-10 nodes in the hidden layer. If satisfactory performance is not achieved, increase the number of nodes. The most suitable network architecture among the ones tested is selected based on the analysis of the error variance, according to the guidelines that follow.

• For each hidden layer size, ten networks are trained, and the value of the mean-squared-error with respect to both the training and the test data computed in each case.

• The network with the best balance between training and test set mean-squared-error, as well as size, is selected as the most appropriate network for a particular application.

In the context of statistics and machine learning, underfitting and overfitting are phenomena associated with suboptimal model complexity. Overly simplistic models tend to produce inaccurate predictions and therefore underfit the data, whereas overly complex models tend to predict noise and therefore overfit the data, making them poorly suited for generalization.

The conceptual relationship between those two phenomena is illustrated in Figure 3.2. The region of low model complexity corresponds to high training and test error. The region of high model complexity corresponds to high test error but low training error. Optimal models are mostly found in the region of medium complexity, where there is a fair trade-off between underfitting and overfitting data.

![Figure 3.2: Conceptual representation of overfitting and underfitting](image.png)
This idea is carried on for the examples in the later sections. The model complexity is assessed by the number of free parameters in the network architecture, which is directly related to the number of hidden neurons. The metric for model error is the mean squared error of the network outputs with respect to measured data.

### 3.2 Dynamic Networks

Generally speaking, there are three different types of neural network architectures: single-layer feedforward, multilayer feedforward, and recurrent [14]. The first two types involve strictly input-output mapping of data with no output feedback. The last type, on the other hand, differs from the first two in that it contains at least one feedback loop; that is, at least one output neuron is fed back as an input to another neuron. Among the neural network types, dynamic nonlinear systems are best modeled by recurrent networks, due to the interacting nature of present and past values of the inputs and outputs via feedback loops. As a matter of fact, a recurrent network can approximate any nonlinear dynamic system as long as it is provided with sufficient hidden neurons [15].

Moreover, there is a distinction between dynamic networks that can take only past input values to predict present output values (NAR networks) and those that can also take present values of external input signals (NARX networks). The acronyms NAR and NARX are commonly used in system identification to refer to dynamic systems that do and do not have external inputs applied. NAR stands for Nonlinear Auto Regressive, while NARX stands for Nonlinear Auto Regressive with Exogenous input. The NARX network topology is illustrated in Figure 3.3 for a single feedback loop, where \( u(k) \) represents the external model input and \( \hat{w}(k) \) represents the model output. Note that NARX networks have two tapped delayed signals, one for the input and another one for the output.
In NARX models, future values of the output depend only on past and present values of the input and the output. The part of the network identified as multilayer perceptron represents a feedforward network that can have more than a single hidden layer. It has certain inputs that are mapped to outputs by undergoing a series of operations as they pass through the network layers. Again, the difference between the regular feedforward structure and the structure presented here is in the feedback loop and in the tapped delayed signals for both the input and the output.

In functional form, the output of the model in Figure 3.3 is

\[
\hat{w}(k) = F(\hat{w}(k - 1), \ldots, \hat{w}(k - n); u(k - 1), \ldots, u(k - q))
\]  

(3.2)

Where \( F \) is a nonlinear function of its arguments [14].

In order to initialize this network, it is necessary to have delayed values of the input and output, which act as initial conditions in the tapped delayed signals. These initial conditions are obtained by shifting the input and output data as many steps as the number of delays in each case.
3.2.1 Input and Feedback Delays

Typically, the input and output delays of a NARX model is determined by studying the cross correlation of the input and output signals and the autocorrelation of the output signal. An example of this can be found in [16]. Signal correlation quantifies the similarity between samples as a function of the time lag between them. The samples can be either from two different signals (cross-correlation) or from the same signal (autocorrelation).

The purpose behind using signal correlation in a NARX model is to find the delays (or lags) in the signals that are most statistically significant with respect to the output signal. Using the results from correlation plots leads to a high probability of obtaining model outputs that correctly estimate the target outputs. Additionally, evaluating the correlation sequence of the signals helps ensure that enough delays, but not too many, are used as part of the model. If an insufficient number of delays is used, it is difficult for the model parameters (the connection weights and biases of the neural network) to converge to values that result in a suitable fit for the data. On the other hand, too many delays leads to an unnecessarily high number of parameters, which can result in suboptimal generalization performance, not to mention the expense of the extra processing requirements.

For two signals $x(k)$ and $y(k)$, the autocorrelation and cross-correlation signals are defined as follows:

\[
\phi_{xx}(h) = \sum_{k=0}^{N-h-1} x(k)x(k + h) \quad (3.3)
\]

\[
\phi_{xy}(h) = \begin{cases} 
\sum_{k=0}^{N-h-1} x(k + h)y(k), & 0 \leq h \leq N - 1 \\
\phi_{yx}(h), & - (N - 1) \leq h \leq 0
\end{cases} \quad (3.4)
\]

The index $h$ is the lag or step by which a signal is shifted; its value can be either a positive or a negative integer. $N$ is the length of the signals $x(k)$ and $y(k)$, and the subscripts in the formulas indicate the signals being correlated. The formulas in (3.3) and (3.4) give the correlation sequences in the scales of the signals. For a more intuitive use, these quantities are typically normalized to be strictly in the range $[-1, 1]$. Once normalized, the obtained signals are known as correlation coefficient sequences.
Normalized Autocorrelation sequence:  \[
\rho_{xx}(h) = \frac{\phi_{xx}(h)}{\phi_{xx}(0)} \tag{3.5}
\]

Normalized Cross-correlation sequence:  \[
\rho_{xy}(h) = \frac{\phi_{xy}(h)}{\sqrt{\phi_{xx}(0)\phi_{yy}(0)}} \tag{3.6}
\]

Notice that at \(h = 0\), the autocorrelation coefficient is \(\rho_{xx}(0) = 1\). This happens because in the absence of a shift, a signal is perfectly aligned with itself. A correlation coefficient of zero is caused if the signals have the exact same shape, but are opposite in sign. As a last remark, notice that if \(\phi_{xx} = \phi_{yy}\), equation (3.6) becomes the same as (3.5), which means that autocorrelation is simply the cross-correlation of a signal with itself.

Correlation between variables can arise due to a mutual linear dependence on other variables [17]. In order to account for this, the partial autocorrelation finds the autocorrelation between \(x(k)\) and \(x(k + h)\) after removing the linear dependence on the intermediate observations \(x(k + 1), ... , x(k + h - 1)\). The partial autocorrelation is essentially the correlation between the residuals of the linear regressions of \(x(k)\) and \(x(k + h)\) with each one of their intermediate observations. This function was first introduced by Box and Jenkins, for further details please refer to [17].

For the examples of this work, only the cross-correlation coefficients (XCC) and the partial autocorrelation coefficients (PAC) of the signals were analyzed.
Chapter 4

MME/NN Combined Algorithm

Chapter 2 ended with a discussion regarding how to use the output from MME to improve the state equations of a poorly modeled dynamic system. The method under consideration in this work is a combination of the MME estimation algorithm and a static, feedforward neural network with backpropagation, which was first attempted by Kolodziej [7].

As previously discussed, MME uses a cost function based on a covariance constraint to develop a better estimate of a system’s states than an initially assumed model while generating the correction term used to improve the model. Though it is possible to correlate mathematical functions of state to the obtained $d(t)$ ([5], [6]), this can be a time-consuming task, given that a search algorithm must be performed to find the functions that best fit the unmodeled dynamics of the system. A neural network can be a simple, yet powerful alternative to a mathematically derived function, and it has the advantage of being able to capture underlying nonlinear functional relationships between inputs and outputs, as well as linear ones. Typically, if there is available but possibly incomplete information about the dynamics of a system, a neural network can be implemented alongside other system identification techniques ([18], [19]).

Neural networks are known to be universal approximators of nonlinear functions [20]. Furthermore, they tend to be robust in the presence of noise. These two traits make them ideal as black-box structures that can be combined with the Minimum Model Error estimator to form a hybrid modeling methodology. The purpose of neural networks in such a scenario is to relate presumably unmodeled system dynamics to the system states in some sort of fashion. This configuration allows for the trained neural network to be eventually incorporated as part of a plant model.
The first phase of the hybrid MME/NN algorithm consists on processing measured system data with MME. In order to do this, a linear model form must be assumed. At this point, smooth state estimates and a correction vector are generated. Having all the output data from MME, the next step is to train a static, feedforward neural network with backpropagation, to map the system states to $d(t)$. After the neural network has been trained with the output data from MME, it can output the correction terms that need to be added to the assumed linear model.

Figure 4.1 below is an extended version of Figure 2.1 that shows the information flow between MME and the neural network during training of the latter:

Besides the implementation of MME, most of the processing effort in this algorithm goes into the design of the neural network and its incorporation in the system model. The time and cost of processing for a neural network are not nearly as much as what they can potentially be with function libraries, nor is the network as theoretically complex.

Designing the static neural network for use in the combined algorithm follows the same ideas that were explained in Chapter 3. The selected network is ideally the smallest-size network that meets a prescribed performance criterion. The remainder of this chapter is devoted to exposing several examples of the application of MME/NN models on both simulated and experimental data.
4.1 Single pendulum Simulation

This example goes through the process of modeling a point-mass pendulum with a displacement at the pivot point, as seen in Figure 4.3. The system is simulated, so the truth system model and response are known. Instead of the applied force, the acceleration of the pivot point is selected as the system input. In order to assess the performance of the combined MME/NN model under different noise conditions, the process is performed twice: once with the truth, clean data and once with noise added to the truth data. The inputs to the system are shown in Figure 4.4.

Figure 4.3: Single pendulum schematic

Figure 4.4: Acceleration inputs - single pendulum simulation. Top: Training set input, Bottom: Test set input.
4.1.1 Physics Equations

In order to derive the physics equations of this system, the Lagrangian method is used. The general derivation of a system’s Lagrangian takes the form

\[ L = K - U \]  \hspace{1cm} (4.1)

\[ \frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_k} \right) - \frac{\partial L}{\partial q_k} + \frac{\partial R}{\partial q_k} = F \]  \hspace{1cm} (4.2)

Where

- \( L \) = Lagrangian,
- \( K \) = kinetic energy,
- \( U \) = potential energy,
- \( q \) = generalized coordinate,
- \( k \) = generalized coordinate index,
- \( R \) = non-conservative forces,
- \( F \) = external forces applied along coordinate \( q \).

For the single pendulum with point mass, equations (4.1) and (4.2) become

\[ \frac{d}{dt} \left( \frac{\partial L}{\partial \theta} \right) - \frac{\partial L}{\partial \theta} + \frac{\partial R}{\partial \theta} = 0 \]  \hspace{1cm} (4.3)

\[ \frac{d}{dt} \left( \frac{\partial L}{\partial x} \right) - \frac{\partial L}{\partial x} = F_a \]  \hspace{1cm} (4.4)

Where \( \theta \) and \( x \) represent the angular displacement from the vertical and the horizontal displacement of the pivot point, respectively, and \( F_a \) is the applied force along the \( x \) coordinate.

With

- \( m \) = pendulum mass,
- \( L_p \) = pendulum length,
- \( I \) = moment of inertia,
- \( d \) = damping coefficient at pivot,
- \( g \) = gravitational constant,

The energy-related terms become

\[ K = \frac{1}{2} m \dot{x}^2 + \frac{1}{2} \left( \frac{mL_p^2}{4} + I \right) \dot{\theta}^2 + \frac{1}{2} m L_p \dot{x} \dot{\theta} \cos \theta \]  \hspace{1cm} (4.5)
\[ U = -\frac{mgL_p}{2} \cos \theta \]  \hfill (4.6)

\[ R = \frac{1}{2} d \dot{\theta}^2 \]  \hfill (4.7)

Substituting equations (4.5)-(4.7) into equations (4.3) and (4.4) results in the system equations:

\[ mx + mL_p \ddot{\theta} \cos \theta - mL_p \dot{\theta}^2 \sin \theta = F_a \]  \hfill (4.8)

\[ L_p \ddot{\theta} + g \sin \theta + d \dot{\theta} = -\dot{x} \cos \theta \]  \hfill (4.9)

Notice that equation (4.9) is a stand-alone equation with input \( \dot{x} \). That is, knowledge of \( F_a \) is not required in order to solve (4.9), as long as \( \dot{x} \) is known. Solving the latter for \( \ddot{\theta} \) yields

\[ \ddot{\theta} = -\frac{g}{L_p} \sin \theta - \frac{d}{L_p} \dot{\theta} - \frac{\cos \theta}{L_p} \dot{x} \]  \hfill (4.10)

Equation (4.10) was used to generate the data for this example. It can also be expressed in state space form as follows, with \( x_1 = \theta, x_2 = \dot{\theta} \) and \( u = \dot{x} \):

\[ \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{L_p} & -\frac{d}{L_p} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{L_p} \end{bmatrix} u \]  \hfill (4.11)

Given that MME can only assume linear system models, the best linear model it can be given based on the physics model for this example is

\[ \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{L_p} & -\frac{d}{L_p} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{L_p} \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} d(t) \]  \hfill (4.12)

The first row of the state space model is \( \dot{x}_1 = x_2 \). This is true based on how the states were defined, and does not need to be altered. Hence, only the state equation for \( x_2 \) needs to be corrected. Adding the unmodeled dynamics term to the state space model, the corrected model is obtained:

\[ \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{L_p} & -\frac{d}{L_p} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{L_p} \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} d(t) \]  \hfill (4.13)

The added term, \( d(t) \), is estimated during the MME operation.

Sections 4.1.2 and 4.1.3 show the results of implementing the MME/NN algorithm on simulated data for the single pendulum for clean data and noisy data. The only state data
provided to MME is the angular displacement, $\theta$. The noisy data was obtained by adding Gaussian noise with a variance of 0.1 to the original clean data. The assumed model in both cases is (4.12), the best linear model estimate obtained from the derived nonlinear physics model.

### 4.1.2 Clean Data

The neural network for the MME/NN model is trained with the states and the unmodeled dynamics estimated with MME. Additionally, $\dot{x}_2$ is not linearly related to $u(t)$, so the correction term also needs to account for this. The inputs and outputs of the neural network are then:

- **Inputs:** $x_{1,est}, x_{2,est}, u(t)$
- **Output:** $d(t)$

After training and testing networks with 1-10 hidden nodes, the mean squared error in each case is computed. A look at the figure below shows that the 2-node network yields the best results.

![MSE for Trained Networks](image)

Figure 4.5: Performance of trained neural networks on clean data sets
Figure 4.6 presents the results of both MME alone and MME/NN for the training data set, and they are all excellent. As expected for noiseless data, the MME estimates match the truth data perfectly ($R^2 = 1$). Additionally, the chosen network captures the underlying dynamics that the assumed linear model does not. As a result, the MME/NN output also matches the truth data. The output from the assumed linear model, included on the same figure, is poor, showing that MME had a significant amount of correction to perform.

**Training Data**

![Graph of Training Data](image)

Figure 4.6: State predictions for training data set – clean single pendulum data. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity)
Nonetheless, matching clean training data is a relatively easy task for a MME/NN model, since the neural network in it is trained precisely to be able to do this. The challenge lies in being able to use the same model on a data set not shown to the network during the training phase. Even so, the plant model that includes the 2-node network does an exceptional job at matching the truth test data, as shown in Figure 4.7. The correlation coefficient in this case is also approximately $R^2 \approx 1$ for both states.

![Test Data](image)

Figure 4.7: State predictions for test data set – clean single pendulum data. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity)

In this case, because both the assumed and truth models are known, it is possible to solve for the truth unmodeled dynamics term.

$$
\dot{x}_{true} = \dot{x}_{assumed} + d(t) \Rightarrow d(t) = \dot{x}_{true} - \dot{x}_{assumed}
$$

(4.14)
From (4.11) and (4.12):

\[ \dot{x}_{\text{truth}} = -\frac{g}{L_p} \sin x_1 - \frac{d}{L_p} x_2 - \frac{\cos \theta}{L_p} u \] (4.15)

\[ \dot{x}_{\text{assumed}} = -\frac{g}{L_p} x_1 - \frac{d_a}{L_p} x_2 - \frac{1}{L_p} u \] (4.16)

Therefore, the truth model correction that MME calculates and the neural network generates is

\[ d(t) = \frac{g}{L_p} (x_1 - \sin x_1) + \frac{1}{L_p} (1 - \cos \theta) u \] (4.17)

Figure 4.8 shows how well MME and MME/NN estimate the truth model correction term, \( d(t) \). Both algorithms have a near perfect correlation coefficient and very low variances.

Figure 4.8: Truth and estimated model correction terms – clean single pendulum data. Top: training set \( d(t) \), Bottom: test set \( d(t) \)
4.1.3 Noisy Data

Figure 4.9: Performance of trained neural networks on noisy data sets

Figure 4.9 reveals that there are a few networks that can model the noisy data for the single pendulum system. Although a 7-node network gives very good results, the change in performance over the 2 or 3-node networks is not that great. The 3-node network is chosen because of its balanced trade-off between training and test set errors.

The MME results for the training set are shown below. Notice that the estimate of state $x_1$ has a variance of approximately 0.01 with respect to the measured data. This matches the covariance constraint from MME, as the Gaussian noise that was added to the clean data had a variance of 0.01. Furthermore, the estimate-minus-truth data variance is approximately 0.001, even lower than the former value, demonstrating that the state estimates from MME are closer to the truth system data than to the noisy measured data. State $x_2$, which is not provided to MME, is also successfully estimated and shown to match the truth values closely.
Training Data

\[
\begin{array}{ll}
\text{MME Est. - Truth: } & R_1^2 = 0.999 \quad | \quad \sigma_1^2 = 0.00179 \\
\text{MME Est. - Noisy: } & R_1^2 = 0.988 \quad | \quad \sigma_1^2 = 0.00978 \\
\end{array}
\]

Figure 4.10: MME State estimates for training data set (with weight matrix \( W = \begin{bmatrix} 1 & 0 \\ 0 & 1.85 \end{bmatrix} \)).

Top: State \( x_1 \) (angular displacement), Bottom: State \( x_2 \) (angular velocity).

Additionally, the MME/NN results are very similar to those obtained with MME alone. The variance of the MME/NN output with respect to the noisy data is approximately 0.01 for both the training and test data, and the variance with respect to the truth data is also lower. Once again, the selected neural network successfully captures the underlying dynamics of the correction term. In this case, however, it is able to do so with noise present in the data provided to MME. These results can be seen in Figure 4.11 and Figure 4.12.
Figure 4.11: MME/NN State estimates for training data set – noisy single pendulum data.
Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity)
Figure 4.12: MME/NN State estimates for test data set – noisy single pendulum data. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity)
Figure 4.13: Truth and estimated model correction terms – noisy single pendulum data. Top: Training set $d(t)$, Bottom: Test set $d(t)$

Though not as perfect as they were for the clean data, the correction terms for the noisy data are still very good. Moreover, adding them to the assumed portion of the MME/NN model leads to estimated states that meet the covariance constraint and have correlation coefficients close to $R^2 \approx 1$ with respect to the truth states.
4.2 Single Pendulum Experiment

Data was collected from a single compound pendulum with an acceleration at the pivot point, which is achieved by moving the cart where the pendulum is pivoted. The setup is similar to the one used for the simulation (see Figure 4.3), with the difference that the pendulum link is a solid bar rather than a mass hanging from the pivot. The acceleration data is obtained from measurements, which were collected for a total of 18 seconds at a rate of 20 Hz.

![Pivot Acceleration Input](image)

Figure 4.14: Acceleration inputs - compound pendulum experiment. Top: Training set input, Bottom: Test set input.

The form of the physics model for the compound pendulum is similar to that of the single pendulum, the only difference being the coefficients of the variables. Thus, the best linear model that can be assumed for this example is in the form of (4.18). Notice that it has the same structure as (4.12).

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2 
\end{bmatrix} = \begin{bmatrix}
0 & 1 \\
-a_1 & -a_2
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
0 \\
-b
\end{bmatrix} u 
\begin{equation}
(4.18)
\end{equation}
\]
The coefficients $a_1, a_2$ and $b$ can be approximated by performing a linear regression of the $\dot{x}_2$ term estimated from an ignorance MME model using the state estimates and the input as regressor variables. This is explained in more detail in section 4.3.3. The linear regression results in the coefficients shown in (4.19). The assumed model by itself performs very poorly (21% fit with respect the measured angular displacement data).

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -10.6 & -0.0430 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1.26 \end{bmatrix} u$$

(4.19)

A 6-node network is selected to map $x_1, x_2$ and $u$ to $d(t)$. Figure 4.15 and Figure 4.16 contain the results for the training and test data sets.

Figure 4.15: State predictions for training data set – compound pendulum experiment. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity)
Figure 4.16: State predictions for test data set – compound pendulum experiment. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity)

Given the initial conditions and the inputs in both cases (0.898 rad, -2.38 rad/s for the training data and 0 rad, 0 rad/s for the test data), the state predictions of the combined MME/NN algorithm are quite good. The correlation coefficients are above $R^2 = 0.9$ in both cases and the variances with respect to the measured data are approximately $\sigma^2_{measured-est} \approx 0.05$ and $\sigma^2_{measured-est} \approx 0.05$, respectively.
4.3 Double Pendulum Simulation

A double pendulum is a benchmark experiment for control of nonlinear systems. This system is highly nonlinear and chaotic. To add another level of complexity, this particular experiment includes a displacement at the pivot point, as in Figure 4.17. The two links of the pendulum are assumed to be solid rods with centers of mass located at their geometrical center. After mathematically deriving the physical equations, the system is simulated with the same inputs for the training and test sets as those used for the single pendulum simulation.

![Figure 4.17: Compound double pendulum schematic](image)

Pivot Acceleration Input

Training Data: $u(t) = 10\sin(0.2\pi t)$

Test Data: $u(t) = -10\sin(0.4\pi t)$

![Figure 4.18: Acceleration inputs - double pendulum simulation. Top: Training set input, Bottom: Test set input.](image)
The linear model assumed by MME may contain as much information as the user desires to include. Some of the options are:

- A linear model estimated from physics principles, as the one in the single pendulum simulation
- An ignorance model, which relies on MME to capture all the dynamics in the system while smoothing the data
- A linear model using regressed constants

This example covers the last two cases. Sections 0 and 4.3.3 discuss the results of modeling a double pendulum when MME is given an ignorance model and when it is given a regressed linear model.

### 4.3.1 Physics Equations

Once again, the Lagrangian method is used to derive the equations of motion. The same basic equations apply in this case, with the kinetic and potential energy terms having two components, one for each link of the pendulum.

\[
\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_k} \right) - \frac{\partial L}{\partial q_k} + \frac{\partial R}{\partial q_k} = F
\]

\[
L = K_1 + K_2 - U_1 - U_2
\]

Where

\[
K_1 = \frac{1}{2} m_1 \dot{x}^2 + \frac{1}{2} \left( \frac{m_1 L_1^2}{4} + I_1 \right) \dot{\theta}_1^2 + \frac{1}{2} m_1 \dot{x} \dot{\theta}_1 \cos \theta_1
\]

\[
K_2 = \frac{1}{2} m_2 \dot{x}^2 + \frac{1}{2} m_2 L_2 \dot{\theta}_2^2 + \frac{1}{8} m_2 L_2^2 \dot{\theta}_2^2 + m_2 L_1 \dot{x} \dot{\theta}_1 \cos \theta_1 + \frac{1}{2} m_2 L_2 \dot{\theta}_2 \cos \theta_2 + \frac{1}{2} m_2 L_1 L_2 \dot{\theta}_1 \dot{\theta}_2 \cos (\theta_1 - \theta_2) + \frac{1}{2} L_2 \dot{\theta}_2^2
\]

\[
U_1 = -\frac{m_1 g L_1}{2} \cos \theta_1
\]

\[
U_2 = -m_2 g \left( L_1 \cos \theta_1 + \frac{L_2}{2} \cos \theta_2 \right)
\]
Substituting equations (4.22)-(4.25) into (4.20) and (4.21), the equations of motion of the double pendulum are obtained. After some mathematical manipulation, they are decoupled to produce a separate equation for each link of the pendulum.

\[
\ddot{\theta}_1 = \frac{1}{c_3^2 \cos^2(\theta_1 - \theta_2) - 4c_1c_2} (Q_1 + Q_2 + Q_3\ddot{x}) \tag{4.26}
\]

\[
\ddot{\theta}_2 = \frac{1}{c_3^2 \cos^2(\theta_1 - \theta_2) - 4c_1c_2} (V_1 + V_2 + V_3\ddot{x}) \tag{4.27}
\]

The constant terms in (4.26) and (4.27) are defined as follows:

\[
Q_1 = 2c_2c_4\sin\theta_1 + c_3^2 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) \cos(\theta_1 - \theta_2) + 2c_2c_3\dot{\theta}_2^2 \sin(\theta_1 - \theta_2) - c_3c_5 \cos(\theta_1 - \theta_2) + \sin\theta_2 2c_2D_1
\]

\[
Q_2 = -c_3 \cos(\theta_1 - \theta_2)D_2 + 2b_1c_2\cos\theta_1 - b_2c_2 \cos(\theta_1 - \theta_2) \cos\theta_2 + \sin\theta_2 2c_2D_1 - c_3 \cos(\theta_1 - \theta_2)D_2
\]

\[
Q_3 = 2b_1c_2 \cos\theta_1 - b_2c_3 \cos(\theta_1 - \theta_2) \cos\theta_2
\]

\[
V_1 = 2c_1c_5 \sin\theta_2 + c_3^2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) \cos(\theta_1 - \theta_2) + 2c_1c_3\dot{\theta}_1^2 \sin(\theta_1 - \theta_2) - c_3c_4 \cos(\theta_1 - \theta_2) \sin\theta_1
\]

\[
V_2 = 2c_1D_2 - c_4 \cos(\theta_1 - \theta_2)D_1
\]

\[
V_3 = 2b_2c_1 \cos\theta_2 - b_1c_3 \cos(\theta_1 - \theta_2) \cos\theta_1
\]

Where

\[
b_1 = -\left(\frac{1}{2}m_1 + m_2\right)L_1
\]

\[
b_2 = -\frac{1}{2}m_2L_2
\]

\[
c_1 = \frac{m_1L_1^2}{8} + \frac{L_1}{2} + m_2L_1^2
\]

\[
c_2 = \frac{m_2L_2^2}{8} + \frac{L_2}{2}
\]

\[
c_3 = \frac{m_2L_1L_2}{2}
\]

\[
c_4 = \frac{1}{2}(m_1 + m_2)gL_1
\]

\[
c_5 = \frac{1}{2}m_2gL_2
\]

\[
D_1 = d_1\dot{\theta}_1 - d_2(\dot{\theta}_2 - \dot{\theta}_1)
\]

\[
D_2 = d_2(\dot{\theta}_2 - \dot{\theta}_1)
\]
4.3.2 Ignorance Model

For this 4-state system, the ignorance model is

\[
\begin{bmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2 \\
\ddot{\theta}_1 \\
\ddot{\theta}_2
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2 \\
\ddot{\theta}_1 \\
\ddot{\theta}_2
\end{bmatrix} +
\begin{bmatrix}
0 \\
0 \\
0 \\
0
\end{bmatrix}
\ddot{x}
\tag{4.28}
\]

Notice that the only non-zero terms in the model are the ones; they are necessary as this is known to be absolutely true of the system and does not need to be corrected. The only terms that need correction are \(\ddot{\theta}_1\) and \(\ddot{\theta}_2\), so the system model, including the correction terms, is

\[
\begin{bmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2 \\
\ddot{\theta}_1 \\
\ddot{\theta}_2
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2 \\
\ddot{\theta}_1 \\
\ddot{\theta}_2
\end{bmatrix} +
\begin{bmatrix}
0 \\
0 \\
0 \\
0
\end{bmatrix}
\ddot{x} +
\begin{bmatrix}
0 & 0 \\
0 & 0 \\
1 & 0 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
d_1(t) \\
d_2(t)
\end{bmatrix}
\tag{4.29}
\]

Where \(d_1(t)\) and \(d_2(t)\) are functions of the states and the input. From (4.29), it can be concluded that they are equivalent to \(\ddot{\theta}_1\) and \(\ddot{\theta}_2\), respectively and, therefore, (4.26) applies to \(d_1(t)\) and (4.27) applies to \(d_2(t)\). The model correction terms for both sets are displayed on Figure 4.19 and Figure 4.20. Compared to the results that have been obtained for \(d(t)\) until now, these seem rather poor, especially the ones for \(d_1(t)\). Both the MME and MME/NN outputs have very large amounts of variance, and the correlation coefficients are low.
Figure 4.19: Estimated model correction terms for training data – double pendulum with ignorance model. Top: Correction term for $\dot{\theta}_1$, Bottom: Correction term for $\dot{\theta}_2$. 
Based on these plots, it is natural to guess that the state predictions from both algorithms are just as bad as the $d(t)$ predictions; however, as Figure 4.21 and Figure 4.22 show, that is not really the case. On the contrary, the correlation coefficients for all states, including training and test sets, are above $R^2 = 0.95$, and the variances of the prediction-minus-truth data are also much lower for the state outputs than for the actual $d(t)$ terms. These might not seem like great results, but they are decent considering the number of nonlinearities present in the truth system model.
Figure 4.21: State predictions for **training** data set – double pendulum with ignorance model. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link).
Figure 4.22: State predictions for test data set – double pendulum with ignorance model. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link).
The MME estimates overlay the truth data, demonstrating the ability of the MME estimator to produce accurate state estimates even in the absence of any kind of system model. Likewise, the results of the MME/NN model are excellent. Given an ignorance model of a very nonlinear system, the model was able to predict the outputs to within a variance of $10^{-7}$ for all states in the training set. For the test data case, the results were also very good. With a different initial condition and different input from the training case, the output predictions still had correlation coefficients above 0.95 for all four states.

### 4.3.3 Updated Model

Having found the model correction estimates with an ignorance MME model, the results can be used to update the assumed model via a linear regression. The regressed variables are the $d(t)$ terms, while the regressor variables are $\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2$, and $\ddot{x}$. The constants $B_1, B_2$ and $a_1$ through $a_8$ are estimated during the regression procedure.

$$
d_1(t) = \ddot{\theta}_1 = f_1(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, \ddot{x}) \approx a_1 \theta_1 + a_2 \theta_2 + a_3 \dot{\theta}_1 + a_4 \dot{\theta}_2 + B_1 \ddot{x}
$$

$$
d_2(t) = \ddot{\theta}_2 = f_2(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, \ddot{x}) \approx a_5 \theta_1 + a_6 \theta_2 + a_7 \dot{\theta}_1 + a_8 \dot{\theta}_2 + B_2 \ddot{x}
$$

(4.30)

A least-squares regression results in the following equations for the model correction terms:

$$
d_1(t) \approx -19.1 \theta_1 + 3.96 \theta_2 - 1.07 \dot{\theta}_1 - 0.408 \dot{\theta}_2 - 2.08 \ddot{x}
$$

$$
d_2(t) \approx 7.47 \theta_1 - 18.9 \theta_2 + 1.15 \dot{\theta}_1 + 0.267 \dot{\theta}_2 - 1.4305 \ddot{x}
$$

(4.31)

Using the information in (4.31) to update the ignorance model, the best linear model that can be assumed for MME is

$$
\begin{bmatrix}
\ddot{\theta}_1 \\
\ddot{\theta}_2
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 1 & 0 \\
-19.1 & 3.96 & -1.07 & -0.408 \\
7.47 & -18.9 & 1.15 & 0.267
\end{bmatrix}
\begin{bmatrix}
\theta_1 \\
\theta_2 \\
\dot{\theta}_1 \\
\dot{\theta}_2
\end{bmatrix}
+ 
\begin{bmatrix}
0 \\
0 \\
-2.08 \\
-1.4305
\end{bmatrix} \ddot{x}
$$

(4.32)

Naturally, performing MME with this updated assumed model gives rise to different model correction terms than the ones that were obtained with the ignorance model. In comparison, the updated model gives much better results regarding how well the MME and MME/NN methods are able to predict the $d(t)$ terms. Figure 4.19 and Figure 4.20 showed a percent fit as low as $R^2 \approx 0.508$ and a variances as high as $\sigma^2 \approx 67.9$. In this case, the percent fits are all above $R^2 = 0.9$ and the highest variance is $\sigma^2 \approx 15$. 

46
Figure 4.23: Estimated model correction terms for training data – double pendulum with updated model. Top: Correction term for $\dot{\theta}_1$. Bottom: Correction term for $\dot{\theta}_2$. 

MME Est. - Truth: $R^2_1 = 0.995$ | $\sigma^2_1 = 0.611$

MME/NN - Truth: $R^2_1 = 0.995$ | $\sigma^2_1 = 0.594$

MME Est. - Truth: $R^2_2 = 0.996$ | $\sigma^2_2 = 0.623$

MME/NN - Truth: $R^2_2 = 0.955$ | $\sigma^2_2 = 15$
Figure 4.24: Estimated model correction terms for test data – double pendulum with updated model. Top: Correction term for $\ddot{\theta}_1$, Bottom: Correction term for $\ddot{\theta}_2$.

The poorly estimated correction terms obtained with the ignorance model were not a setback for the MME/NN model to generate reasonable state predictions; however, better model corrections are desirable. Having the estimated $d(t)$ resemble the truth $d(t)$ more closely results in better state predictions, as shown in Figure 4.25 and Figure 4.26. In both cases, the correlation coefficients were all above $R^2 = 0.98$ and the prediction-minus-estimate variances were all below $\sigma^2 = 1$. 
Figure 4.25: State predictions for training data set – double pendulum with updated model. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link).
Figure 4.26: State predictions for test data set – double pendulum with updated model. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link).
All the systems considered in this chapter are given assumed models that are either derived from physics principles or determined using linear regressions on the model correction terms. This is met with better results than using an ignorance model, as this example shows.

### 4.4 Double Pendulum Experiment

An experimental setup resembling the simulation setup in Figure 4.17 was used in order to collect data from a double pendulum system with a displacement at the pivot point. The data was collected for 21 seconds at a rate of 20 Hz.

![Pivot Acceleration Input](image)

**Figure 4.27**: Acceleration inputs - double pendulum experiment. Top: Training set input, Bottom: Test set input.
A linear regression on data from MME with an assumed ignorance model, as the one performed in section 4.3.3, yields the assumed linear model in equation (4.33).

\[
\begin{bmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2 \\
\ddot{\theta}_1 \\
\ddot{\theta}_2
\end{bmatrix} = 
\begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
-28.1 & 18.3 & -0.181 & -0.300 \\
39.0 & -69.5 & 2.06 & 1.52
\end{bmatrix}
\begin{bmatrix}
\theta_1 \\
\theta_2 \\
\ddot{\theta}_1 \\
\ddot{\theta}_2
\end{bmatrix} \quad + \quad \begin{bmatrix}
0 \\
0 \\
-2.99 \\
3.54
\end{bmatrix} \ddot{x}
\]  

(4.33)

The \(d(t)\) terms predicted by the neural network in the MME/NN model seem smoother than the MME estimates. This is to be expected, since MME not only estimates the system states, but also corrects any noise present in the measured data, inducing the presence of some noise into the model correction term, \(d(t)\). Conversely, the MME/NN algorithm relies on a neural network that is trained to fit this correction term. Unless the network is over trained, it tends to produce a smooth fit of the output data.

Figure 4.28: Estimated model correction terms for training data – double pendulum experiment. Top: Correction term for \(\dot{\theta}_1\), Bottom: Correction term for \(\ddot{\theta}_2\).
Figure 4.29: State predictions for training data set – double pendulum experiment. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link).
Figure 4.30: State predictions for test data set – double pendulum experiment. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link).
4.5 Two-Tank Experiment

For this experiment, a two-tank setup similar to the one in Figure 4.31 is used. A small hole drilled at the bottom of each tank allows the contents of the tanks to flow out. A pump provides an input flow of water to the upper tank, which then exits through the hole and into the lower tank. Likewise, the water in the lower tank flows out through the hole at the bottom. The aim of a model for this experiment would be to predict the level of the water in the tanks based on the voltage applied to the pump. The effect of sampling the model prediction is analyzed by varying the sampling rates of the measured data. Sampling rates of 10 Hz, 5 Hz, 1 Hz and 0.5 Hz are considered.

\[ Q_{in}(t) = c_1 * u(t) \]

Figure 4.31: Two-tank experiment setup
4.5.1 Physics Equations

\[ m = \text{mass of fluid in the tanks}, \]
\[ \rho = \text{fluid density}, \]
\[ V = \text{fluid volume}, \]
\[ A = \text{cross-sectional area of the tanks}, \]
\[ H = \text{height of fluid in the tanks}, \]
\[ Q_{in} = \text{inflow rate}, \]
\[ k = \text{flow coefficient}. \]

A mass balance equation for each tank can be expressed as

\[ \dot{m} = m_{in} - m_{out} \quad (4.34) \]

Where

\[ \dot{m} = \frac{d}{dt}(\rho V) = \rho \dot{V} + \dot{\rho} V = \rho \dot{V} \quad (4.35) \]
\[ \dot{V} = \frac{d}{dt}(A \cdot H) = A \dot{H} + \dot{A} H = A \dot{H} \quad (4.36) \]

Combining equations (4.34)-(4.36):

\[ \dot{m}_{in} - \dot{m}_{out} = \rho A \dot{H} \quad (4.37) \]
\[ \dot{H} = \frac{1}{\rho A}(\dot{m}_{in} - \dot{m}_{out}) \quad (4.38) \]

Moreover, the mass flow rate through an orifice can be approximated by

\[ \dot{m}_{out} \approx k\sqrt{H} \quad (4.39) \]

And

\[ \dot{m}_{in} = \rho (A \dot{H}) = \rho Q_{in} \quad (4.40) \]

Substituting (4.39) and (4.40) into (4.38) gives

\[ \dot{H} = \frac{1}{\rho A}(\rho Q_{in} - k\sqrt{H}) \quad (4.41) \]

Let

\[ \alpha = \frac{k}{\rho A} \quad (4.42) \]

Equation (4.41) can then be written as

\[ \dot{H} = -\alpha \sqrt{H} + \frac{Q_{in}}{A} \quad (4.43) \]
Which applies to both tanks. For the upper tank, \( Q_{in} \) is supplied by the pump; for the lower tank, \( Q_{in} \) is the flow out of the upper tank. Applying (4.43) to both tanks, the resulting system equations are

\[
\begin{align*}
\dot{H}_1 &= -\alpha_1 \sqrt{H_1} + \frac{Q_{in}}{A} \\
\dot{H}_2 &= -\alpha_2 \sqrt{H_2} + \alpha_1 \sqrt{H_1}
\end{align*}
\]  

(4.44)

Where the subscripts 1 and 2 refer to the upper and lower tanks, respectively. The flow provided to the upper tank, \( Q_{in} \), is related to the applied voltage by

\[
Q_{in} = C_1 * u
\]

(4.45)

With \( u \) representing the voltage applied to the pump and \( C_1 \) is a constant of proportionality. Given the form of the nonlinear equations in (4.44), the best linear model that can be assumed for this system is

\[
\begin{bmatrix} 
\dot{H}_1 \\
\dot{H}_2
\end{bmatrix} = 
\begin{bmatrix}
-a_1 & 0 \\
a_2 & -a_3
\end{bmatrix} 
\begin{bmatrix} H_1 \\
H_2
\end{bmatrix} + 
\begin{bmatrix} C_1/A \\
0
\end{bmatrix} u
\]

(4.46)

The unknown coefficients can be determined via a linear regression, as previously done. The regressors for \( \dot{H}_1 \) are \( H_1 \) and \( u \); whereas the regressors for \( \dot{H}_2 \) are \( H_1 \) and \( H_2 \). The regression procedure is repeated for each of the sampling rates considered.

### 4.5.2 Sampling Rate: \( f_s = 10 \) Hz

The data collected for this example runs for a time span of 357 seconds, with a data point collected every 0.1 seconds. The training data for the plant neural network in the MME/NN model is taken from two different portions of the entire data set: from 38.5 s to 168.1 s and from 205.9 s to 270 s. The rest of the points in the data set that are not used to train the neural network are used to test it. The assumed model, after performing a linear regression on the training data, is

\[
\begin{bmatrix} 
\dot{H}_1 \\
\dot{H}_2
\end{bmatrix} = 
\begin{bmatrix}
-0.0304 & 0 \\
0.0404 & -0.0459
\end{bmatrix} 
\begin{bmatrix} H_1 \\
H_2
\end{bmatrix} + 
\begin{bmatrix} 0.00191 \\
0
\end{bmatrix} u
\]

(4.47)

The resulting state estimates from MME on the entire data set are shown in Figure 4.32 along with the assumed model estimates from (4.47).
The assumed linear model does not capture most of the dynamics of the system. The output states from MME are smooth and they do capture the system’s dynamics. The pump voltage, water level estimates and the model corrections from MME compose the neural network training set. As mentioned previously, these training data are taken from two different portions of the original data; they are shown in light blue in the figure below. Half the points in each of the two portions is picked at random, so not all the points in the ranges $38.5 \leq t \leq 168.1$ and $205.9 \leq t \leq 270$ are included in the training set.

Figure 4.32: Measurement set and MME estimates – two-tank experiment with $f_s = 10$ Hz. Top to Bottom: Applied voltage, Water level in the upper tank (m), Water level in the lower tank (m).
Figure 4.33: Training/Testing Data distribution for neural network. From Top to Bottom, Left to Right: Pump voltage, Estimated water level in the upper tank ($x_1$), Estimated water level in the lower tank ($x_2$), Model correction for state $x_1$, Model correction for state $x_2$. 
The results of the MME/NN combined algorithm for the two-tank data sampled at 10 Hz are shown below. After training the network and including it in the plant model of the system, the input voltage is applied and the output response of the two tanks is predicted.

![Graph showing state predictions for two-tank experiment with $f_s = 10$ Hz](image)

Figure 4.34: State predictions – two-tank experiment with $f_s = 10$ Hz. Top: Water level in the upper tank, Bottom: Water level in the lower tank.

The predicted response is almost as good as the MME estimate. Notice that the input for this example resembles a signal with step changes, demonstrating the ability of the MME/NN model to react to abrupt variations in the system input.
4.5.3 Sampling Rate: \( f_s = 5 \) Hz

The assumed model using data sampled at 5 Hz is very similar to the one obtained for a 10 Hz sampling rate, which indicates that the change in sampling rate was not met with a decrease in the ability of the data to capture the dynamics of the system.

\[
\begin{bmatrix}
\dot{H}_1 \\
\dot{H}_2
\end{bmatrix} = \begin{bmatrix}
-0.0308 & 0 \\
0.0399 & -0.0452
\end{bmatrix} \begin{bmatrix}
H_1 \\
H_2
\end{bmatrix} + \begin{bmatrix}
0.00193 \\
0
\end{bmatrix} u
\]

(4.48)

This result is not only true of the assumed model, but also of the MME and MME/NN results. The states predicted based on the applied input are essentially the same as in the previous case.

Figure 4.35: State predictions – two-tank experiment with \( f_s = 5 \) Hz. Top: Water level in the upper tank, Bottom: Water level in the lower tank.
4.5.4 Sampling Rate: $f_s = 1$ Hz

Reducing the sampling period to a sample per second gives rise to the following assumed linear model for the data:

$$
\begin{bmatrix}
\dot{H}_1 \\
\dot{H}_2
\end{bmatrix}
= 
\begin{bmatrix}
-0.0333 & 0 \\
0.0392 & -0.0445
\end{bmatrix}
\begin{bmatrix}
H_1 \\
H_2
\end{bmatrix}
+ 
\begin{bmatrix}
0.00208 \\
0
\end{bmatrix} u
$$

(4.49)

The model predictions are still excellent even at $1/10$th of the original sampling rate, as evidenced by Figure 4.36.

---

Figure 4.36: State predictions – two-tank experiment with $f_s = 1$ Hz. Top: Water level in the upper tank, Bottom: Water level in the lower tank.
4.5.5 Sampling Rate: $f_s = 0.5$ Hz

At 0.5 Hz, the sampling rate is $1/20$th of what it was originally. Yet with such a decrease, the coefficients of the linear regression are not all that different from the original coefficients in (4.47).

$$
\begin{bmatrix}
\dot{H}_1 \\
\dot{H}_2
\end{bmatrix} =
\begin{bmatrix}
-0.0356 & 0 \\
0.0382 & -0.0433
\end{bmatrix}
\begin{bmatrix}
H_1 \\
H_2
\end{bmatrix} +
\begin{bmatrix}
0.00222 \\
0
\end{bmatrix} u 
$$

(4.50)

Figure 4.37: State predictions – two-tank experiment with $f_s = 0.5$ Hz. Top: Water level in the upper tank, Bottom: Water level in the lower tank.
The results from MME are as good as they have been for all the sampling rates considered, but there is a slight difference in performance in the model prediction of the MME/NN model. Since the MME estimates were good, it is safe to assume that the source of the error is the neural network in the plant model. This phenomenon is most likely a consequence of various events at play. For one, the sampling rate reduction for the same time span of data means there are less data points available to train the network, so there is a chance it could be undertrained. Also, any network prediction errors propagate through the integration, which, given the larger integration time step, could produce inaccurate results. Nevertheless, it is important to notice that the decrease in performance, small as it may be, was only noticeable at 0.5 Hz, or a sample every two seconds, which is a very slow sampling rate for any modern sensing and measuring devices.

4.6 Summary

Through the various examples examined in this chapter, the performance of the combined MME/NN algorithm was evaluated under different experimental conditions: noise vs. no noise, ignorance model vs. updated model, and high vs. low sampling rate. Overall, the hybrid models implemented were successful at producing accurate state predictions for all the data sets available. These results speak of the fidelity of a plant model derived using the MME/NN method.

All-in-all, the algorithm being considered can handle varying degrees of noise and sampling frequencies, and though assuming a complete ignorance model produces adequate results, performing a model update via linear regression of the unmodeled dynamics leads to improved prediction accuracy.
Chapter 5

Comparison to Other Identification Methods

From a system identification standpoint, it is desirable to compare various model types and structures before deciding on a model for a system. In this portion of the work, the performance of the algorithm under consideration is compared to the performance of two other identification methods: Nonlinear Auto Regressive networks with Exogenous input, covered in Chapter 3, and linear, parametric state space models, which will be covered briefly. The performance criteria used are the correlation coefficient and the error variance. The first one offers insight into how well a model captures the dynamics of a system, while the latter serves to determine overfitting and underfitting.

The training and validation techniques for the NARX networks are the same that were explained in Chapter 3, with the partial autocorrelation and cross-correlation functions used to determine the number of delays for the feedback loop of the network. Recall that the NARX model can only predict the states for which measurements are available, unlike the MME/NN and linear state space models. Except for the two-tank example, which consists of a single time history data set, only the training sets that were used in the previous chapter are presented here for comparison.
5.1 Output Error Estimation

The output error method is an iterative algorithm that estimates the free parameters of a given linear model structure while minimizing the error of the model output with respect to given data.

Unlike other parametric methods, the output error approach does not describe the properties of the disturbance signal. Essentially, it assumes that any disturbances in the truth output signal are due to Gaussian distributed, random error and therefore need not be included in the system model.

\[ y(t) = G(q, \theta)u(t) + e(t) \]  

(5.1)

For continuous time state space models, (5.1) can be rewritten as

\[ \dot{x}(t) = A(\theta)x(t) + B(\theta)u(t) \]
\[ y(t) = Cx(t) + v(t) \]

(5.2)

The optimization requires the free parameters in \( \theta \) to converge to values that make the state space in (5.2) stable.

For example, for a mass-spring-damper system with unknown stiffness and damping constants, the general state space model would be

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2 
\end{bmatrix} =
\begin{bmatrix}
0 & 1 \\
-\frac{\theta_1}{m} & -\frac{\theta_2}{m}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 
\end{bmatrix} +
\begin{bmatrix}
0 \\
1 
\end{bmatrix} u
\]

(5.3)

With \( \theta = [\theta_1 \, \theta_2]^T \).
5.2 Single Pendulum Simulation – Clean Data

- Linear Model Approximation: the state space model is initialized with the parameters from the best linear physics model derived on the last chapter:

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
0 & \frac{1}{L_p} \\
-\frac{g}{L_p} & -\frac{d}{L_p}
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
0 \\
-\frac{1}{L_p}
\end{bmatrix} u
\]  

(5.4)

The output error method updates the free parameters, which in this case are the values defined by \( \frac{g}{L_p}, \frac{d}{L_p}, \) and \( \frac{1}{L_p} \);

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
0 & 1 \\
-32.1 & -0.00
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
0 \\
-2.69
\end{bmatrix} u
\]  

(5.5)

- NARX Network Model: based on the correlation coefficient plots, the appropriate number of input and output delays for the NARX network is six. A one-node network is sufficient to model the data with the specified number of delays.

Figure 5.2: Correlation plots – clean single pendulum data. Left: Partial autocorrelation signal, Right: Cross-correlation signal.
• Comparison of results:

<table>
<thead>
<tr>
<th>Model</th>
<th>$x_1$</th>
<th>$x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% Fit</td>
<td>Var.</td>
</tr>
<tr>
<td>MME/NN</td>
<td>100</td>
<td>7.69e-6</td>
</tr>
<tr>
<td>NARX</td>
<td>99.9</td>
<td>2.45e-4</td>
</tr>
<tr>
<td>Linear</td>
<td>96.6</td>
<td>2.66e-2</td>
</tr>
</tbody>
</table>

Table 5-1: Model performance – clean single pendulum data

Figure 5.3: State predictions for clean single pendulum data. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity)

Even though the performance measures indicate that all three models can predict state $x_1$ to a very good degree, the hybrid model is slightly better and, in addition to that, it is capable of producing an accurate estimate of state $x_2$. 
5.3 Single Pendulum Simulation – Noisy Data

- Linear Model Approximation: initialized in the same way as with the clean data. The resulting model is similar to 5.1.

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
0 & 1 \\
-32.3 & -0.0313
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
0 \\
-2.73
\end{bmatrix} u
\]  

(5.6)

- NARX Network Model: there is significant autocorrelation for up to three lags, but the results with that many feedback delays are not adequate. Testing a few alternatives results in six being the least number of input and feedback delays required to accurately model the data. Once again, a one-node network is enough to get the job done.

![Figure 5.4: Correlation plots – noisy single pendulum data. Left: Partial autocorrelation signal, Right: Cross-correlation signal.](image-url)
Comparison of results: the table below shows not only the performance measured with respect to the given, noisy data, but also the with respect to the truth data

<table>
<thead>
<tr>
<th>Model</th>
<th>$x_1$ (wrt. truth)</th>
<th>$x_1$ (wrt. noisy)</th>
<th>$x_2$</th>
<th>$x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% Fit</td>
<td>Var.</td>
<td>% Fit</td>
<td>Var.</td>
</tr>
<tr>
<td>MME/NN</td>
<td>99.9</td>
<td>2.00e-3</td>
<td>98.7</td>
<td>1.07e-2</td>
</tr>
<tr>
<td>NARX</td>
<td>99.9</td>
<td>2.50e-3</td>
<td>98.8</td>
<td>9.50e-3</td>
</tr>
<tr>
<td>Linear</td>
<td>97.5</td>
<td>2.03e-2</td>
<td>96.0</td>
<td>3.21e-2</td>
</tr>
</tbody>
</table>

Table 5-2: Model performance – noisy single pendulum data

In the presence of noise, the results from the NARX network are almost identical to the results from MME/NN. The linear approximation is satisfactory for state $x_1$, but not that great for state $x_2$. The hybrid algorithm has a slightly lower variance with respect to the truth data compared to the dynamic network. This is a desirable trend, as the end goal of any system identification algorithm is to model the truth dynamics of a system and not the disturbances or noise present in the measurement signals.

Figure 5.5: State predictions for noisy single pendulum data. Top: State $x_1$ (angular displacement), Bottom: State $x_2$ (angular velocity)
5.4 Single Pendulum Experiment

- Linear Model Approximation:

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} =
\begin{bmatrix}
0 & 1 \\
-9.39 & -0.194
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} +
\begin{bmatrix}
0 \\
-2.79
\end{bmatrix} u
\]  \hspace{1cm} (5.7)

- NARX Network Model: the cross-correlation and partial autocorrelation plots suggest that up to seven delays are necessary to properly model the data. Using this number for both the input and feedback delays produces an accurate estimate for the angular displacement data.

Figure 5.6: Correlation plots – single pendulum experiment. Left: Partial autocorrelation signal, Right: Cross-correlation signal.
• Comparison of results:

<table>
<thead>
<tr>
<th>Model</th>
<th>% Fit</th>
<th>Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MME/NN</td>
<td>98.1</td>
<td>6.39e-2</td>
</tr>
<tr>
<td>NARX</td>
<td>99.9</td>
<td>2.32e-4</td>
</tr>
<tr>
<td>Linear</td>
<td>98.0</td>
<td>1.15e-1</td>
</tr>
</tbody>
</table>

Table 5-3: Model performance – single pendulum experiment.

Figure 5.7: State prediction for $x_1$ for single pendulum experiment.

Since the angular displacement is not measured and there is no truth data for this system, only the predictions for state $x_1$ can be compared. The combined MME/NN method and the linear model display similar performances. In both cases, the model overestimates the amplitude of the response signal, hence the correlation coefficients are great, but the variances are higher than the variance obtained with the NARX network. Nevertheless, based on the variance, MME/NN provided a better model estimate than the linear model.
5.5 Double Pendulum Simulation

- Linear Model Approximation: given only the model structure (i.e. $\dot{x}_1 = x_3$, $\dot{x}_2 = x_4$), the free parameters in the rows of $\dot{x}_3$ and $\dot{x}_4$ were updated.

$$
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2 \\
\dot{x}_3 \\
\dot{x}_4
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
-5.67 & -8.02 & 0.926 & -0.240 \\
6.10 & -24.2 & 2.74 & -1.36
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix} +
\begin{bmatrix}
0 \\
0 \\
-1.15 \\
-1.54
\end{bmatrix} u
$$

- NARX Network Model: the partial autocorrelation plots suggest that 14 is the least number of delays required to model the data; however, that leads to very poor results. Increasing the number of input and feedback delays to 25 gives much better network prediction accuracy.

Figure 5.8: Correlation plots – double pendulum simulation. From Left to Right, Top to Bottom: Partial autocorrelation of State $x_1$, Cross-correlation between State $x_1$ and the input $u(t)$, Partial autocorrelation of State $x_2$, Cross-correlation between State $x_2$ and the input $u(t)$. 

73
• Comparison of results:

<table>
<thead>
<tr>
<th>Model</th>
<th>$x_1$ % Fit</th>
<th>$x_1$ Var.</th>
<th>$x_2$ % Fit</th>
<th>$x_2$ Var.</th>
<th>$x_3$ % Fit</th>
<th>$x_3$ Var.</th>
<th>$x_4$ % Fit</th>
<th>$x_4$ Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MME/NN</td>
<td>99.9</td>
<td>4.13e-4</td>
<td>99.9</td>
<td>7.22e-4</td>
<td>99.7</td>
<td>1.50e-2</td>
<td>99.7</td>
<td>2.28e-2</td>
</tr>
<tr>
<td>NARX</td>
<td>99.9</td>
<td>6.243-4</td>
<td>99.8</td>
<td>1.9e-3</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Linear</td>
<td>37.5</td>
<td>1.28</td>
<td>18.4</td>
<td>2.59</td>
<td>-21.9</td>
<td>21.2</td>
<td>-22.9</td>
<td>50.7</td>
</tr>
</tbody>
</table>

Table 5-4: Model performance – double pendulum simulation

The linear fit is extremely poor, even leading to negative correlation coefficients, which means that the prediction is out of phase with respect to the data. On the contrary, the other two methods offer near perfect fits for the angular displacement data. Once again, MME/NN is slightly better than the NARX network for states $x_1$ and $x_2$, and it also outputs estimates states $x_3$ and $x_4$. 
Figure 5.9: State predictions for double pendulum simulation. Top to Bottom: State $x_1$ (angular displacement of the top link), State $x_2$ (angular displacement of the bottom link), State $x_3$ (angular velocity of the top link), State $x_4$ (angular velocity of the bottom link).
5.6 Double Pendulum Experiment

- Linear Model Approximation: initialized using the assumed linear model for the MME operation.

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2 \\
\dot{x}_3 \\
\dot{x}_4
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
-25.3 & 13.3 & 0.208 & 0.205 \\
32.3 & -61.1 & -0.611 & -1.33
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix} +
\begin{bmatrix}
0 \\
0 \\
-2.42 \\
2.90
\end{bmatrix} u
\] (5.9)

- NARX Network Model: there are statistically significant correlations up to 20 delays, which is the maximum number of lags shown. Using 20 gives underperforming results, so the delays are increased to 25, which gives suitable estimates.

Figure 5.10: Correlation plots – double pendulum experiment. From Left to Right, Top to Bottom: Partial autocorrelation of State \( x_1 \), Cross-correlation between State \( x_1 \) and the input \( u(t) \), Partial autocorrelation of State \( x_2 \), Cross-correlation between State \( x_2 \) and the input \( u(t) \).
• Comparison of results:

<table>
<thead>
<tr>
<th>Model</th>
<th>$x_1$</th>
<th>$x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MME/NN</td>
<td>97.3</td>
<td>98.6</td>
</tr>
<tr>
<td>NARX</td>
<td>99.5</td>
<td>99.6</td>
</tr>
<tr>
<td>Linear</td>
<td>98.2</td>
<td>97.5</td>
</tr>
</tbody>
</table>

Table 5-5: Model performance – double pendulum experiment

In this case, the NARX network shows an improved performance over the MME/NN model. The error variance is an order of magnitude lower and the correlation coefficient is approximately 2% higher. The performance of the linear model is very similar to that of the hybrid model; it is a little better for state $x_1$ than it is for state $x_2$. 

Figure 5.11: State predictions for double pendulum experiment. Top: State $x_1$ (angular displacement of the top link), Bottom: State $x_2$ (angular displacement of the bottom link).
5.7 Two-Tank Experiment - $f_s = 5$ Hz

- Linear Model Approximation: the model structure derived from physics equations is kept ($\dot{x}_1 = f_1(x_1, u), \dot{x}_2 = f_1(x_1, x_2)$).

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -0.109 & 0 \\ 2.62e4 & -2.99e4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0.0066 \\ 0 \end{bmatrix} u$$  \hspace{1cm} (5.10)

- NARX Network Model: different delay arrangements would be possible based on the correlation plots, but a 4-delay setup generates the best results for the dynamic network model.

Figure 5.12: Correlation plots – two-tank experiment. From Left to Right, Top to Bottom: Partial autocorrelation signal of state $x_1$, Cross-correlation signal between state $x_1$ and the input $u(t)$, Partial autocorrelation signal of state $x_2$, Cross-correlation signal between state $x_2$ and the input $u(t)$.  

78
• Comparison of results:

<table>
<thead>
<tr>
<th>Model</th>
<th>(x_1) % Fit</th>
<th>(x_1) Var.</th>
<th>(x_2) % Fit</th>
<th>(x_2) Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MME/NN</td>
<td>99.4</td>
<td>6.72e-5</td>
<td>98.7</td>
<td>3.62e-5</td>
</tr>
<tr>
<td>NARX</td>
<td>97.1</td>
<td>4.08e-4</td>
<td>87.1</td>
<td>3.93e-4</td>
</tr>
<tr>
<td>Linear</td>
<td>81.2</td>
<td>3.10e-3</td>
<td>31.6</td>
<td>1.2e-3</td>
</tr>
</tbody>
</table>

Table 5-6: Model performance – two-tank experiment

Figure 5.13: State predictions for two-tank experiment. Top: Water level in the upper tank, Bottom: Water level in the lower tank.

The hybrid model has the best performance out of the three models tested. As the plot shows, it is the only model that can predict the system response to the given input to within low error variances for both states. The linear model performs very poorly, and the neural network estimate is adequate for the state \(x_1\), but it underperforms for state \(x_2\).
5.8 Summary

Comparing the results of the previous chapter to results obtained with two common identification methods demonstrates the modeling power of the combined MME/NN algorithm. The linear model approximations are met with poor performances in most of the examples examined. This is to be expected given the fact that all of the systems are nonlinear.

The NARX network has an overall satisfactory performance, mostly comparable to that of the hybrid model; however, it only generates predictions of those states that are provided as measured states. In contrast to that, the algorithm presented in this work is able to generate estimates for all states as long as they were accounted for in the linear portion of the model. Another weakness of the NARX network is that it requires multiple data points in order to be initialized, which makes it unsuitable as a plant model. This might be acceptable for very simple systems that only need one of two feedback delays, but it becomes a problem for those systems that require several delays, like the double pendulum cases, which require 25 delays each. The delays can be provided for training data sets, but they would need to be estimated for testing sets, and that would add more complexity to the NARX network model.

These results confirm the suitability of the MME/NN method as an estimation and modeling tool for nonlinear systems, which typically pose a modeling challenge.
Chapter 6

Application of MME/NN Algorithm to Classification

Fault detection and classification is a data-driven field of ever increasing importance in many applications, ranging from industrial equipment monitoring to system failure detection and prevention. The basic purpose may vary from case to case (prevention vs. identification), but the aim is essentially the same: to detect abnormal running conditions that may already exist in the system or may occur within a certain time frame.

Some classification algorithms rely on the analysis of system models to assess the condition of a component or system; like the coefficients in a linear model, for example. Given that neural networks are model blocks with internal parameters, it may be possible to use those parameters for a fault detection scheme. That is, assessing system condition based solely on the weight variation of the neural network from an MME/NN model estimate.

In practice, the method would involve collecting data for the expected operating conditions of a system or subsystem and store it as historical data that can regularly be compared against current data. Should any changes in the network parameters occur, a faulty condition could potentially be prevented or identified.

In this work, four common statistical pattern recognition methods are tested: linear classifiers, quadratic classifiers, k-nearest neighbor and naïve Bayes classifiers. The first two, as their names imply, attempt to separate data classes by means of planar and quadratic surfaces, respectively. For the k-nearest neighbor method, the classes of the $k$ closest labeled points to a test sample (in terms of Euclidean distance) are counted, and the class with the most representatives is the class the test point is assigned to. Naïve Bayes classifiers minimize the probability of error by assigning a test sample to the most probable class assuming the features are independent from one another.
The first step in a classification routine is feature extraction, or selection of the explanatory variables used as inputs to the classifiers. This can be achieved, and is very commonly done, by means of Principal Components Analysis (PCA), a method that allows for dimensionality reduction and variable decorrelation. The aforementioned classifiers are trained with two different sets of features: in the original data space (i.e. original network parameters) and in a transformed space obtained via PCA. A brief description of PCA is presented next.

### 6.1 Principal Components Analysis

The method of principal components can take any number of features, or variables, and apply a transformation such that the output variables are spread along the directions of highest variance. Those directions are called principal components, and they are obtained in such a way that orthogonality is guaranteed.

In mathematical terms, PCA seeks to determine the linear combinations of the available features that yield the greatest variances along axes, or components, which successively decrease on importance. This allows the user to work with reduced-dimensionality data that contains most of the variability present in the original data set. Let $X_{p	imes n}$ be a matrix containing the data gathered from an experiment. It has $n$ observations and $p$ measured quantities, or features. PCA uses a transformation matrix $A_{p	imes p}$ to obtain the matrix $\xi_{n	imes p}$ of transformed data features.

The steps involved in this process are the following:

I. Subtract the sample mean from matrix $X$; let the resulting matrix be called $X_{new}$

II. Determine $\Sigma$, the covariance matrix of $X_{new}$

III. Perform the eigen decomposition of $\Sigma$ to determine its eigen values and eigen vectors

IV. Form the transformation matrix $A$, whose columns are the eigenvectors of $\Sigma$

V. For dimensionality reduction, select the first $k$ principal components that can account for a fraction $d$ of the total variance of the data. For this, use the eigenvalues from III and the formula $d = \frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{p}\lambda_i}$

VI. Map the original data to the transformed space with $\xi_k = A_k^T X_{new}$

For a more thorough explanation of this procedure, or for different derivations, see [8].
6.2 Mass-Spring-Damper Case Study

Consider again the simulated nonlinear mass-spring-damper example presented in Chapter 2, which had the truth system model

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
0 & 1 \\
-k/m & -c/m
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
0 \\
1/m
\end{bmatrix} u + \begin{bmatrix}
0 \\
-k/m x_1^3
\end{bmatrix}
\] (6.1)

And the assumed linear model

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2
\end{bmatrix} = \begin{bmatrix}
0 & 1 \\
-k/m & 0
\end{bmatrix} \begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} + \begin{bmatrix}
0 \\
1/m
\end{bmatrix} u
\] (6.2)

Based on (6.1) and (6.2), the truth error vector, or model correction term, can be calculated as

\[
d(t) = -\frac{k}{m} x_1^3 - \frac{c}{m} x_2
\] (6.3)

The neural network in the MME/NN model is tasked with modeling (6.3), therefore, its inputs are \(x_1\) and \(x_2\), and its only output is \(d(t)\).

Figure 6.1: Neural network model of equation (6.3) for the unmodeled dynamics of the mass-spring-damper system
Parameters $i\omega_{1,1}, ..., i\omega_{2,3}$, $l_\omega_1$, $l_\omega_2$, $l_\omega_3$, and $b_1, ..., b_4$ are estimated during the network training phase. Noiseless data of the first 10 seconds of the system response is used for this case study. In order to create a “faulty” condition, three damping coefficients are used: 0.1, 0.15 and 0.2. For each damping condition, 100 networks with 3 hidden nodes are trained and the ones with the best mean-square-error (mse) performance selected for further examination (the cut-off value for mse is 1.0). The hidden layer of nodes $N_1$, $N_2$, and $N_3$ creates a weighted combination of the inputs, $x_1$ and $x_2$, before passing it on to the output layer. Therefore, any weight changes specifically associated with one state ($x_2$ in this case) can only be observed on the input layer weights, $i\omega_{1,1}, ..., i\omega_{2,3}$.

Table 6-1 shows a comparison of the distributions for each input weight of the network. They are divided both by node number and the state they are associated with. The x-axis shows the weight value, while the y-axis shows how many out of the trained networks had that particular weight value. The first thing to notice is that the distributions are not Gaussian; instead, they appear to have several peaks. The values of the peaks do change from one distribution to another, hinting to a change in network parameter values, but a quantitative assessment is required to back up the qualitative observation.

Notice that there is nothing in the distributions that suggests a change in damping coefficient has occurred. For this to be the case, the plots in the last column, which correspond to the weights associated with the velocity state, $x_2$, would need to show a greater change than those associated with the position state, $x_1$. 
Table 6-1: Input weight distributions for “good” networks (mse < 1)
Next, the trained network parameters are used as features for PCA. There are 13 features in total: 6 input weights, 3 hidden layer biases, 3 layer weights, and the output layer bias. In order to account at least 90% of the data variability, 8 components are needed. Ideally, the directions of maximum variance would also be the dimensions along which the data can be separated into the three available classes. Unfortunately, it is not possible to visualize this in a 2D or 3D plot since 8 dimensions are needed. The results of the different classifiers are summarized in Table 6-2.

<table>
<thead>
<tr>
<th>Model</th>
<th>Error % (no PCA)</th>
<th>Error % (after PCA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Discriminant</td>
<td>37.0</td>
<td>48.1</td>
</tr>
<tr>
<td>Quadratic Discriminant</td>
<td>44.4</td>
<td>63.0</td>
</tr>
<tr>
<td>k-NN</td>
<td>22.2</td>
<td>22.2</td>
</tr>
<tr>
<td>Bayes Classifier</td>
<td>14.8</td>
<td>37.0</td>
</tr>
</tbody>
</table>

Table 6-2: Model performance with parameters from 3-node networks used to model the mass-spring-damper system

The classification performance in all cases is very poor. None of the classifiers attain errors below 10%. Also, PCA has the opposite effect of what was intended, as it does not help with data separability. Since the data sets used are noiseless data from a fairly simple, second-order system, the results would most likely not be any better for a more realistic system with some noise, more nonlinearities, or simply higher order.

The parameters used above are from 3-node networks that can map the states to the error vector for the given data sets with high accuracy. Alternatively, for classification purposes, the parameters from networks with fewer nodes could be used for analysis. That implies a compromise in modeling performance for the possibility of better classification performance. To this end, 1-node networks were trained with the same data sets that were used to train the 3-node networks. A 1-node network has 5 parameters, and applying PCA reveals that only the first two components are required to account for 98% of data variability. The PCA results can actually be plotted this time:
Figure 6.2: PCA reduces the number of features from 5 to 2. Class separation is possible.

Even though there were 240 networks in the training set (80 for each damping class), only 6 data points are visible on Figure 6.2. This happens because the network parameters converge to the same two values for each class. That is, of the 80 networks that represent any given class, there are only two unique combinations of network parameters. The classification results are shown in Table 6-3.

<table>
<thead>
<tr>
<th>Model</th>
<th>Error % (no PCA)</th>
<th>Error % (after PCA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Discriminant</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Quadratic Discriminant</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>k-NN</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Bayes Classifier</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 6-3: Model performance with parameters from 1-node networks used to model the mass-spring-damper system

With or without PCA, the classification performance is perfect. Reducing the number of nodes certainly would not benefit modeling performance, but it allows the network parameters to converge to values that can be easily separated by several classifiers.
6.3 Double Pendulum Case Study

Classification is possible for the mass-spring-damper data when the number of nodes in the MME/NN neural network is reduced to one. To see if that method also works for a more complex system, similar classification procedures are used on the data for the double pendulum simulation from section 4.3, which required a 6-node network to accurately model the data. In this case, the damping coefficient at the location where the pendulum links are joined is changed and five different conditions are considered: 0.0024, 0.0027, 0.0030, 0.0033, and 0.0036. The actual $d(t)$ terms that need to be modeled by the trained networks are shown in Figure 6.3. With the four system states as network inputs and two correction terms as network outputs, the complete network architecture consists of 44 weight parameters: 24 input weights, 6 hidden layer biases, 12 layer weights, and 2 output weights.

![Figure 6.3: Output $d(t)$ terms for different damping conditions – double pendulum simulation data.](image-url)
The results of the classifiers trained with the parameters of the 6-node networks are shown in Table 6-4. Once again, the parameters from the neural network appropriately sized for a MME/NN model lead to very poor classification performance.

<table>
<thead>
<tr>
<th>Model</th>
<th>Error % (no PCA)</th>
<th>Error % (after PCA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Discriminant</td>
<td>83.0</td>
<td>83.0</td>
</tr>
<tr>
<td>Quadratic Discriminant</td>
<td>80.0</td>
<td>80.0</td>
</tr>
<tr>
<td>k-NN</td>
<td>87.0</td>
<td>77.0</td>
</tr>
<tr>
<td>Bayes Classifier</td>
<td>76.0</td>
<td>71.0</td>
</tr>
</tbody>
</table>

Table 6-4: Model performance with parameters from 6-node networks used to model the double pendulum system

Next, the network size is reduced to one node. From the 9 starting parameters (4 input weights, 1 hidden layer bias, 2 layer weights, and 2 output weights), 4 features are obtained with PCA that describe up to 95% of the data variability. However, only 2 features are needed to account for 67% of the data variability, which could give the user some insight regarding the data distribution and possible separability.

![Figure 6.4: First 2 components of the transformed data for the 1-node networks used to model the double pendulum system](image)

Figure 6.4: First 2 components of the transformed data for the 1-node networks used to model the double pendulum system
An examination of the first two components of the transformed data reveals no meaningful patterns in the data regarding class separation. These results are further backed up by the results on Table 6-5.

<table>
<thead>
<tr>
<th>Model</th>
<th>Error % (no PCA)</th>
<th>Error % (after PCA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Discriminant</td>
<td>59.0</td>
<td>62.0</td>
</tr>
<tr>
<td>Quadratic Discriminant</td>
<td>53.0</td>
<td>56.0</td>
</tr>
<tr>
<td>k-NN</td>
<td>45.0</td>
<td>27.0</td>
</tr>
<tr>
<td>Bayes Classifier</td>
<td>66.0</td>
<td>52.0</td>
</tr>
</tbody>
</table>

Table 6-5: Model performance with parameters from 1-node networks used to model the double pendulum system

The results for the 1-node case are significantly better than the results for 6-node case, but they are still far from satisfactory. The smallest error is obtained with the k-nearest neighbor classifier after applying PCA, but even then the classification error is 27%, which is quite high.

6.4 Summary

The case studies investigated in this chapter show that satisfactory identification of faulty conditions by analyzing neural network weight variation is only possible for very low order systems. Given a fourth order nonlinear system, classification was not possible, with or without a data coordinate transformation, even with a single node in the hidden layer. This leads to the conclusion that network parameters from MME/NN models are poor predictors for condition monitoring of realistic systems, which are typically complex.

Close examination of the mass-spring-damper data reveals something interesting. Though it was not shown, of the several networks trained for $c = 0.1$, some performed satisfactorily when tested on data for $c = 0.2$, and vice versa. Hence, some networks converged to similar weight values for the two data sets and therefore separation was not possible from the start.

Class separability was possible in the first case because the trained networks converged to one of two local minima, limiting the variability of in-class samples. In the second case, the trained networks converged to several different configurations that did not exhibit any kind of underlying trends for each of the classes, even after transforming the features via PCA.
Chapter 7

Summary and Conclusions

This work had two main objectives. The first objective was to successfully implement a system identification methodology combining the MME estimator and feedforward neural networks on simulated and experimental data. The second objective was to investigate the feasibility of using the network parameters from MME/NN models to conduct condition monitoring of systems.

The first objective was fully met, as several nonlinear simulated and real systems were successfully modeled by means of the MME/NN hybrid methodology. Not only that, but compared to the performance of two common system identification techniques (NARX models and output error models), the combined algorithm was shown to be superior regarding ease of implementation and prediction accuracy.

Output error models failed to perform well in most cases because their linear nature did not allow them to predict the response of systems that were highly nonlinear. NARX models displayed performances that were on par with the hybrid models’; however, in practical use, they are more limited due to the fact that they need a potentially high number of initial conditions (delays) to be initialized. Also, the fact that they rely explicitly on past and present data to predict future data makes them sensitive to changes in sampling rate. On the other hand, sampling proved not to be a problem for the hybrid algorithm, which showed adequate prediction accuracies even down to a sampling rate of 0.5 Hz for the two tank experiment.

The second portion of the research showed no real promise for network parameters as predictors in fault detection systems. Even if the classifier results were great for the second order mass-spring-damper system with 1-node networks, these results did not extend to the double pendulum system. As a result of their data-dependent nature, neural network design is both application specific and empirical. This gives rise to very particular situations when using neural
networks in systems modeling. For example, two different network architectures could predict
the behavior of a system with no significant performance changes from one to the other. Also, as
exposed in the previous chapter, a single network could accurately model a system operating
under different conditions (i.e. with changes in damping coefficients.)

Neural networks are flexible modeling structures that do not require any pre conceived
relations between inputs and outputs. This is a major strength that makes them useful in the
MME/NN method because it replaces complex alternatives like function libraries, but it also
becomes their weakness for fault detection applications. The seemingly random values to which
the internal weights converge a lot of times make neural networks unsuitable for statistical
pattern recognition, which relies on tangible underlying trends in data to perform well.

7.1 Future Work

7.1.1 MME/NN Hybrid Algorithm

Even though the performance of the MME/NN models in this work was very good,
 improvement might be possible by optimizing the cost function of the constituent neural
 networks in such a way that it incorporates the state predictions. The cost function that has been
employed thus far focuses on minimizing the prediction error of the model correction term, \( d(t) \).
Instead, the focus could be shifted to what is actually most important for the MME/NN models in
practice: the state estimates. Recall from Chapter 4 that the completed plant model with the
hybrid methodology looks like the schematic in Figure 4.2, included below for convenience.

![Block diagram incorporating the neural network into the plant model after training](image)

Figure 7.1: Block diagram incorporating the neural network into the plant model after training
Assuming that the parameters in the linear portion of the model are fixed, the state estimates are functionally dependent on the weight parameters of the neural network. In light of this, a training algorithm could be developed that optimizes the network parameters based on a cost function similar to the one in (7.1), where

$$\varepsilon = \text{mean square error of state estimates}$$

$$i = \text{sample number}$$

$$N = \text{total number of training samples}$$

$$\hat{x}_{mme} = \text{state estimates from MME}$$

$$F = \text{nonlinear function describing the input - output network mapping}$$

$$w = \text{network weights and biases}$$

$$\varepsilon(w) = \frac{1}{2N} \sum_{i=1}^{N} [(\hat{x}_{mme}(i) - F(w))^2]$$  \hspace{1cm} (7.1)

This would be an interesting problem to look into since there is an integration step between the network output and plant model output.

### 7.1.2 Condition Monitoring

Classification was ineffective for the double pendulum case using samples in both the original feature space and a transformed feature space obtained with PCA. However, future work in this area could focus in finding nonlinear transformations that might actually lead to class separability.
Bibliography


Appendix A: Simulated Systems Diagrams

Figure A.1: Simulink model of single pendulum system with displacement input
Figure A.2: Simulation diagram of $\ddot{\theta}$ in single pendulum system

Figure A.3: Subsystem with energy normalization terms for solver analysis in single pendulum system
Figure A.4: Simulink model of double-pendulum system with displacement input.

This model simulates a nonlinear double pendulum with decoupled equations. The model calculates the normalized pendulum energy, $E$. 
Figure A.5: Subsystem with equation for $\ddot{\theta}_1$ in double pendulum system
Figure A.6: Subsystem with equation for $\ddot{\theta}_2$ in double pendulum system
Figure A.7: Subsystem with energy normalization terms for solver analysis of double pendulum system
Appendix B: Algorithm to Train and Test Neural Networks

clear all; clc; close all; bdclose all

% load all data for training and testing

data1 = load('p3noise_p1damping_train_data_mme.mat');
data2 = load('p3noise_p1damping_test_data_mme.mat');

% Training data sets
T = data1;
% Test set
testset = data2;

% prepare data for processing

num_Train_Sets = length(T);
% Determine number of correction terms
num_dt = length(T(1).d_1(1,1:end));
% Determine number of system states
numStates = length(T(1).EST_1(1,1:end/2));

% If downsampling is desired (ds = 1 - no downsampling)
ds = 1;

% Prepare training data
xm = []; u = []; state_est = []; train_targets = [];
for n = 1:num_Train_Sets
    ind2 = length(T(n).d_1);
    xm = [xm; T(n).xm(1:ds:ind2,:)];
    u = [u; T(n).u(1:ds:ind2)];
    state_est = [state_est; T(n).EST_1(1:ds:ind2,1:numStates)];
    train_targets = [train_targets; T(n).d_1(1:ds:ind2,:)];
end

% algorithm parameters

nn_input_states = 1:numStates;
% Specify which states were measured
measured_states = [1 2];
% Specify which states need correction
corrected_states = [3 4];
% Maximum number of nodes in the hidden layer
numNodes = 10;
% Specify 'IC' or 'u' for free system response or forced response, respectively
inputType = 'u';
% Number of networks trained for each particular size
numIt = 10;

% NN training

tStart = tic;

% Select generic simulation file and define training set depending on the type of input
switch inputType
    case 'IC'
        load_system('mme_nn_x0Driven_sim')
        train_inputs = state_est(:,nn_input_states);
    case 'u'
        load_system('mme_nn_InputDriven_sim')
        train_inputs = [state_est(:,nn_input_states) u];
end

% Initialize data matrices
nets = cell(1,numel(numNodes));
dt_mse_train = cell(1,numel(numNodes));
dt_R_train = cell(1,numel(numNodes));
dt_train_results = cell(1,numel(numNodes));

% Train networks for each particular hidden layer size
for n = 1:numel(numNodes)
    hiddenNodes = numNodes(n);
    % Use neural network training function (appendix C) with provided data
    % [nets{n}, dt_mse_train{n}, dt_R_train{n}, dt_train_results{n}] = ...
    % BestNet_dt(train_inputs,train_targets,hiddenNodes,comb_case,numIt);
end

tEnd = toc(tStart);

% NN train performance in mme/nn model
x_train_temp = cell(1,num_Train_Sets);
dt_train_temp = cell(1,num_Train_Sets);
x_train_mse = zeros(numIt,numel(numNodes));
x_train_R2 = zeros(numIt,numel(numNodes));
x_train_results = cell(numIt,numel(numNodes));
dt_train_results = cell(numIt,numel(numNodes));

% Calculate output states and d(t)
for it = 1:numIt
    for n = 1:numel(numNodes)
        x_concat_datasets = [];
        for k = 1:num_Train_Sets
            trainset = T(k);
            % Selected training set
            x0_train = trainset.EST_1(1,1:numStates)';
            % Signal length
            ind2 = length(trainset.d_1);

            % Display progress in the command window
            disp(['Evaluating ' num2str(it) '-iteration, ' num2str(numNodes(n)) '-node networks'])
            disp('
')
            disp(['Training set ' num2str(k) '/' num2str(num_Train_Sets)])
            disp('
')

            % Implements complete MME/NN plant models (appendix D)
            [x_train_temp{k},dt_train_temp{k}] = runsim(nets{n}{it},x0_train,...
                nn_input_states,corrected_states,numStates,num_dt,...
                trainset.u(1:ind2),trainset.tm(1:ind2),trainset.xm(1:ind2,:),measured_states);

            % Concatenate results of different training sets
            x_concat_datasets = [x_concat_datasets;
                x_train_temp{k}(:,measured_states)];
        end
        % Evaluate mean square error
        x_train_mse(it,n) = goodnessOfFit(x_concat_datasets,xm,'mse');
        % Evaluate percent fit
        x_train_R2(it,n) = corr_coeff(x_concat_datasets,xm);
    end
    % Store state predictions
    x_train_results{it,n} = x_train_temp;
    % Store d(t) predictions
    dt_train_results{it,n} = dt_train_temp;
% Implement model on test data

indx = 1:length(testset.xm(:,1));
% Pick initial condition
x0_test = testset.EST_1(1,1:numStates)';

% Initialize data matrices
x_test_mse = zeros(numIt,numel(numNodes));
x_test_R2 = zeros(numIt,numel(numNodes));
x_test = cell(1,numel(numNodes));
dt_test = cell(1,numel(numNodes));

% Calculate output states and d(t)
for n = 1:numel(numNodes)
    for it = 1:numIt
        % Display progress in the command window
        disp(['Evaluating ' num2str(numNodes(n)) ' node networks on test set'])
        disp(' ')
        disp(['Network ' num2str(it) '/' num2str(numIt)])
        disp(' ')

        % Use function that implements complete MME/NN plant models (appendix D)
        [x_test{n}( :, :, it), dt_test{n}( :, :, it), x_test_mse(it, n), x_test_R2(it, n)] = ...
            runsim(nets{n}{it}, x0_test, nn_input_states, corrected_states, numStates, num_dt, ...
            testset.u(indX), testset.tm(indX), testset.xm(indX,:), measured_states);
    end
end

% Plot performance results

all_MSE = cat(3, x_train_mse, x_test_mse);
[~,I] = min(mean(all_MSE,3));
all_R2 = cat(3, x_train_R2, x_test_R2);
[~,V] = max(mean(all_R2,3));

for k = 1:numel(numNodes)
    Train_MSE(k) = x_train_mse(I(k),k);
    Test_MSE(k) = x_test_mse(I(k),k);
    Train_R2(k) = x_train_R2(I(k),k);
    Test_R2(k) = x_test_R2(I(k),k);
end

figure
plot(numNodes,[Train_MSE' Test_MSE'], '*--');
legend('Train','Test')
xlabel('No. Hidden Nodes'); ylabel('MSE')
title('MSE for Trained Networks')

figure
plot(numNodes,[Train_R2' Test_R2'], '*--');
legend('Train','Test')
xlabel('No. Hidden Nodes'); ylabel('R^2')
title('R^2 for Trained Networks')

% Plot final results

prompt = 'Best node = '
clear input

best_node_ind = input(prompt);
best_it = I(best_node_ind);

numPlots = numel(measured_states);
if numPlots == 1 || 2
    numCol = 1;
    numRows = numPlots;
else
    numCol = 2;
    numRows = ceil(numPlots/2);
end

% training data plots
for k = 1:num_Train_Sets
    plotSet = T(k);
    ind2 = length(plotSet.d_1);
    figure
    for plotNum = 1:numPlots
        subplot(numRows,numCol,plotNum)
        R = corr_coeff(plotSet.xm(1:ds:ind2,plotNum),... x_train_results{best_it,best_node_ind}{k}{:,measured_states(plotNum)});

        plot(plotSet.tm(1:ds:ind2),[plotSet.xm(1:ds:ind2,plotNum) x_train_results{best_it,best_node_ind}{k}{:,measured_states(plotNum)}])

        xlabel('Time - (sec)'); grid on
ylabel(['x_',num2str(measured_states(plotNum))])
legend('measured','model');
title([ 'Train Set, R = ',num2str(R)])
end
end

% test data plot
figure
for plotNum = 1:numPlots
  subplot(numRows,numCol,plotNum)

    R = corr_coeff(testset.xm(indX,plotNum),...
                   x_test{best_node_ind}(:,measured_states(plotNum),best_it));

    plot(testset.tm(indX),[testset.xm(indX,plotNum)
                   x_test{best_node_ind}(:,measured_states(plotNum),best_it)]);

    xlabel('Time - (sec)'); grid on
    ylabel(['x_',num2str(measured_states(plotNum))])
    legend('measured','model');
    title([ 'Test Set, R = ',num2str(R)])
end
Appendix C: Function to Train Networks to Map States to d(t)

```matlab
function [nets, dt_mse, dt_R2, dt_out] = BestNet_dt(Inputs,Targets,hiddenNodes,numIt)
% This function is used to train neural networks of a prescribed size (hiddenNodes) to fit the data provided (Inputs, Targets). numIt is the number of networks that are trained for any particular network size. Used 10 for all the examples evaluated.
% The outputs are: all the networks resulting from training (equal to numIt), the mean square error of the output data with respect to the target data, the percent fit between those 2 sets of data, and the output signals d(t) for each one of the networks.

% Initialize arrays
nets{numIt,1} = [];
dt_R2 = zeros(numIt,1);
dt_mse = zeros(numIt,1);
dt_out = zeros([size(Targets) numIt]);

% Seed random number generator for repeatability of results
rng(0)

% Initialize feedforward neural network
net = fitnet(hiddenNodes,'trainlm');
    % Hide window with training progress
    net.trainParam.showwindow = false;

    net.trainParam.epochs = 500;
    % Training goal for mean square error = 0
    net.trainParam.goal = 0;
    % Minimum gradien change required to proceed with training
    net.trainParam.min_grad = 1e-10;
    % Maximum number of epochs allowed for the validation error % to NOT decrease
    net.trainParam.max_fail = 6;
    % Data division
    net.divideFcn = 'divideint';
    % Network performance criteria
    net.performFcn = 'mse';
```
% Ratio of data used explicitely for training
net.divideParam.trainRatio = 80/100;
% Ratio of data used explicitely for validation
% during training phase
net.divideParam.valRatio   = 20/100;
% No data is used for testing within this function.
% Testing is performed separately with extra
% measurement set
net.divideParam.testRatio  = 0/100;

for it = 1:numIt
    % Display training progress in command window
    disp(['Training ' num2str(it) '/ ' num2str(numIt) ...
        ' networks for ' num2str(hiddenNodes) ' hidden nodes'])
    disp(' ')
    % Reinitialize network weights randomly
    net = init(net);
    % Train and store net
    nets{it}= train(net,Inputs',Targets');
    % d(t) prediction
    outputs = nets{it}(Inputs');
    % Evaluate mean square error
    dt_mse(it) = perform(nets{it},Targets',outputs);
    % Store vectors of predicted d(t)
    dt_out(:,:,it) = outputs';
    % Evaluate percent fit
    dt_R2(it) = corr_coeff(Targets,dt_out(:,:,it));
end
end
Appendix D: Function to Implement Complete MME/NN Plant Models

function [x_test,dt_test,x_test_mse,x_test_R2] = ...
    runsim(net,x0,nn_input_states,corrected_states,numStates,num_dt,input,...
    simTime,x_test_measured,measured_states)

% % This function is used to implement the complete MME/NN plant model
% It is used after the network training phase has taken place.
% The inputs are: the time span of the simulation, the trained network,
% the initial conditions of the system, the total number of states,
% information about which states need correction, number of correction terms,
% external input signal, and for cases where comparison is possible,
% the measured output signals as well as information about which states were measured.

% The outputs are: the output state prediction signals, the d(t) generated,
% the mean square error between output and target signals, and the percent
% fit between these signals.

% Load assumed system model (linear portion of MME/NN model)
assumedmodel

% Resize the B matrix of the assumed model to include the
% d(t) terms added as external inputs to it.
B_dt = zeros(numStates,num_dt);
for k = 1:num_dt
    B_dt(corrected_states(k),k) = 1;
end
if any(B)
    B_aug = [B B_dt];
else
    B_aug = B_dt;
end

% Determine the number of of states generated by the neural network
numOutputs = numel(nn_input_states);

% Define the matrix C of the combined MME/NN model. This matrix
% specifies which neural network output states are required.
C_nnIn = zeros(numOutputs,numStates);
for k = 1:numOutputs
    C_nnIn(nn_input_states(k),k) = 1;
end
% Size the matrix of the combined MME/NN model to appropriately match the sizes of the previously defined matrices.
D_aug = zeros(numOutputs,size(B_aug,2));

% Load simulation file for the completed model
sysName = get_param(gcs,'Name');
% Specify the solver
set_param(sysName,'Solver','ode23tb')
% Generate simulink block for neural object stored in "net"
gensim(net,'Name','trained_net');

% Replace generic network block in simulation file with
% network block for trained network "net"
replace_block(sysName,'Name','Net Fitting d(t)','trained_net/Function Fitting Neural Network','noprompt')
% Save simulink diagram of complete MME/NN with appropriately trained neural network
save_system(sysName)

% Close all simulink files
bdclose all

% Use current function workspace
options = simset('SrcWorkspace','current');

% Simulate system based on desired time span, initial conditions, % and external inputs
[~,x_test,dt_test] = sim(sysName,simTime,options);

% Store matrix of output states that can be compared to measured states
x_test_measured_output = x_test(:,measured_states);
% Evaluate mean square error
x_test_mse = goodnessOfFit(x_test_measured_output,x_test_measured,'mse');
% Evaluate percent fit
x_test_R2 = corr_coeff(x_test_measured_output,x_test_measured);

end
Appendix E: Simulink Diagram of MME/NN Plant Model

Figure E.1: Generic MME/NN Plant model used in conjunction with the code in Appendix C