RBF NEURAL NETWORK BASED GENERALIZED PREDICTIVE CONTROL FOR NONLINEAR STOCHASTIC SYSTEMS

by

Qi Xin

Supervised by: Dr. Xiaoping Liu

Co-supervised by: Dr. Deli Li

A thesis submitted to the faculty of graduate studies

Lakehead University

in partial fulfillment of the requirements for the degree of

Masters of Science in Control Engineering

Department of Electrical Engineering

Lakehead University

March 2013

Lakehead

UNIVERSITY

OFFICE OF GRADUATE STUDIES

NAME OF STUDENT:	Qi Xiu
DEGREE AWARDED:	MScEng
ACADEMIC UNIT:	Faculty of Engineering
TITLE OF THESIS:	RBF Neural Network Based Generalized Predictive Control for Nonlinear Stochastic Systems

This thesis has been prepared

under my supervision

and the candidate has complied

with the Master's regulations.

Signature of Supervisor

Date

Acknowledgements

This thesis is a result of two year work. It would be impossible without the people who accompanied and supported me.

I would like to express my sincere thanks and deepest gratitude to my supervisor Dr. Xiaoping Liu for his guidance. His encouragement, suggestion and patience inspired me a lot during the past two years. I am glad to work under his supervision. I am grateful to my co-supervisor Dr. Deli Li for his help.

I am thankful to Dr. Tayebi and Dr. Uddin for providing a solid background for my studies. I would like to thank Dr. Kefu Liu, Dr. Krishnamoorthy Natarajan and Dr. Wilson Wang for their valuable suggestions.

Finally, I would like to thank Dr. Jian Wang and my parents for many reasons, which allow me to complete this work.

Contests

List of Tables

List of Figures

Abstract

Almost all practical systems are nonlinear, which are subject to disturbances and contain uncertainties. In most cases, disturbances and uncertainties can be modeled as stochastic processes, which make it necessary to develop controllers for nonlinear stochastic systems.

Due to the disturbances and uncertainties, it is difficult to get the exact model of the nonlinear stochastic systems. Neural network techniques are found to have advantages in system identification. Any unknown function can be approximated to any degree of accuracy by a multiple-layer neural network.

In addition, time delay occurs in many real systems. One of the most effective control methods to reduce the impact of delay on the closed-loop systems is predictive control, which is obtained by predicting the future control to minimize the errors.

A RBF Neural Network based Generalized Predictive Controller (NNGPC) is introduced to control nonlinear stochastic systems. The input-output relationship of a nonlinear stochastic system is approximated by an RBF neural network. A learning algorithm is developed to train the RBF neural network by updating the neural network parameters, such as centers, widths, and weights, either on-line or off-line. The parameters are updated using the modified gradient decent method to minimize a cost function, which is a quadratic function of errors between the real system output and the output from the neural network. Based on the model obtained from the neural network learning algorithm, a multistep-ahead generalized predictive control algorithm is designed to minimize a cost function, which is constructed using future control signals and errors between future references and future outputs estimated from the model. The optimization problem involved in the predictive control is solved using Nelder-Mead method and Quasi-Newton method. The comparison between these two methods is made using simulation results.

Chapter 1

Introduction

1.1 Generalized Predictive Control

Model Predictive Control (MPC) has established itself over the past several decades as an industrially important form of advanced control. Since the publication presented by Richalet et al. in 1976 [27] and the paper published in 1978 [28], Model Predictive Heuristic Control (MPHC) was the first description of MPC control applications. Later in 1980 and 1982 Cutler and Ramaker presented Dynamic Matrix Control (DMC) [11] [26], which was an unconstrained multivariable control algorithm. The MPC technology has gained widespread acceptance in academia and in industry. MPC displays improved performance because the process model allows current computations to consider future dynamic events. This provides benefit when controlling processes with large dead times or non-minimum phase behavior. MPC allows for the incorporation of hard and soft constraints directly in the objective function. In addition, the algorithm provides a convenient architecture for handling multivariable control due to the superposition of linear models within the controller.

Another form of MPC that has rapidly gained acceptance in the control community is Generalized Predictive Control (GPC) [10], which was introduced in 1987 by Clarke, Mohtadi, and Tuffs. GPC is a time-domain multi-input-multi-output (MIMO) predictive control algorithm. It employs a Controlled Auto-Regressive and Integrated Moving Average model (CARIMA) of the process which allows a rigorous mathematical treatment of the predictive control paradigm. The GPC performance objective is minimized via recursion on the Diophantine identity.

1.2 RBF Neural Network

A Neural Network (NN) is composed of interconnecting artificial neurons, which uses a mathematical or computational model for information processing [32]. One classical type of NN is the recurrent Hopfield net. Radial Basis Function Neural Network (RBF-NN) is a single hidden layer NN with radial basis functions as activation functions. RBF-NN was first proposed by Moody and Darken in 1988 [22]. The original RBF required that there be as many RBF centers as data points, which is rarely practical because the number of data points is usually very large. Broomhead and Lowe [4] removed the strict interpolation restriction. Their interpretation of the RBF-NN allows the use of only several connections that can affect output of network in some local range of input space. Thus, RBF-NN has faster rates of convergence than other feed forward NNs.

Fig. 1.1 shows the architecture of RBF-NN, which has an input layer, a hidden layer and an output layer. The neurons in the hidden layer contain Gaussian transfer functions whose outputs are inversely proportional to the distance from the center of the neuron. The input into an RBF-NN is nonlinear while the output is linear. Due to their nonlinear approximation properties, RBF-NNs are able to model complex functions.

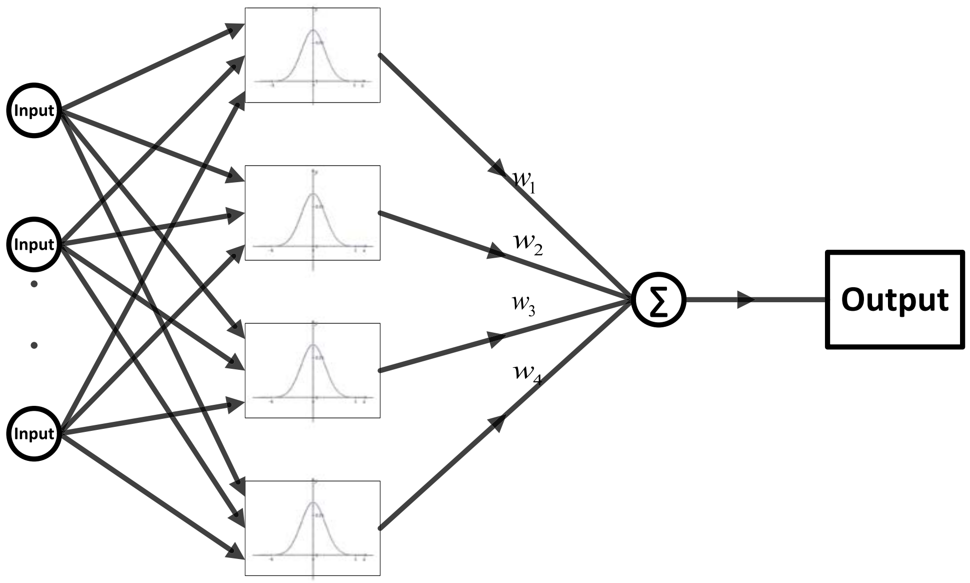


Figure 1.1: The RBF neural network architecture

RBF-NNs have three layers:

1. Input layer - There is one neuron in the input layer for each predictor variable. The input neurons standardize the range of the values by subtracting the median and dividing by the interquartile range. The input neurons then feed the values to each of the neurons in the hidden layer.

2. Hidden layer - This layer has a variable number of neurons (the optimal number is determined by the training process). Each neuron consists of a radial basis function centered on a point with dimension the same as the number of predictor variables. The width (radius) of the RBF function may be different for each dimension. The centers and widths are determined by the training process. When presented with the vector of input

values from the input layer, a hidden neuron computes the Euclidean distance of the test case from the neuron's center point and then applies the RBF kernel function to this distance using the width values. The resulting value is passed to the output layer.

3. Output layer - The value coming out of a neuron in the hidden layer is multiplied by a weight associated with the neuron and passed to the summation which adds up the weighted values and presents this sum as the output of the network.

In order to use a RBF-NN, it is necessary to specify the hidden unit activation function, the number of processing units, a criterion for modeling a given task, and a training algorithm for finding the parameters of the network. Finding the RBF weights is called network training. The proposed training algorithm is aimed at finding a least mean square estimator from the models defined. The following parameters are determined by the training process:

1. The number of neurons in the hidden layer.
2. The coordinates of the center of each hidden-layer RBF function.
3. The width (radius) of each RBF function in each dimension.
4. The weights applied to the RBF function outputs as they are passed to the output layer.

For a set of input-output pairs, called training data, the network parameters are optimized in order to fit the network outputs to the given inputs. The fit is evaluated by means of a cost function, usually assumed to be the mean square error. After training, the RBF-NN can be used with data whose underlying statistics is similar to that of the training set. On-line training algorithms adapt the network parameters to the changing data statistics. RBF-NNs have been successfully applied to a large diversity of applications including interpolation, chaotic time-series modeling, system identification, control engineering, electronic device parameter modeling, channel equalization, speech recognition, image restoration, shape-from-shading, 3-D object modeling, motion estimation and moving object segmentation, data fusion, etc. [12] [31].

1.3 Literature Review

Since NNs work successfully as model identifiers, they can be used to model complex nonlinear systems [23]. Thus, nonlinear predictive control can be applied for controlling real process [18].

Lazar and Pastravanu [19] used a recurrent NN with a single hidden layer as plant model and predictor for nonlinear systems. For the minimization of the cost function, the Matlab's Optimal Toolbox functions *fminunc* and *fmincon* were used to handle unconstrained and constrained optimization problems, respectively.

Lu and Tsai [21] developed NN predictive control for a class of MIMO nonlinear systems with unknown time-delay, in which the neural network was trained by the gradient descent method. The NN nonlinear predictive control law was introduced to minimize the cost function of the error between the reference signal and predicted signal.

In [14] a predictive control algorithm based on Tanaka-Sugeno fuzzy cerebellar model articulation controller (T-S-FCMAC) NN is presented. T-S-FCMAC was used to build the predictive model for the system, and the golden section method was adopted for searching the solution for a quadratic cost function.

In [35] Wang and Xu used a recurrent NN to approximate nonlinear time-delay system as parallel model, which is used to estimate future model predictive output by using input at current time, input before current time, model output at current time and before. Real time recurrent learning algorithm was used to adjust parameters of parallel NN model. Gradient decent algorithm was applied to minimize the quadratic cost function of the predictive controller.

Chidrawar and Patre [9] introduced a neural generalized predictive control with Newton-Raphson as minimization algorithm. Newton-Raphson method required Jacobian and Hessian matrix computation.

In [15] Hu used a Modified Elman Neural Network (MENN) based generalized predictive control to improve the predictive and control precision. MENN is a kind of dynamical recurrent NNs, which is a three-layer network with the addition of a set of "context units" connecting from the hidden layer. The context units always maintain a copy of the previous values of the hidden units at each time step, allowing the network to improve the predictive and control precision. The dynamical back propagation algorithm was used to adjust the MENN parameters. The cost function was minimized by using gradient descent optimization algorithm.

In [36], a hybrid controller blended the neural predictive and fuzzy logic controllers was designed. The neural predictive controller worked in parallel with fuzzy logic controller adjusting the output of the predictive controller. In this approach, the control output of the hybrid controller was the average output of neural predictive and fuzzy control.

However, the multi-layered structure and slow convergence via the use of the gradient descent algorithm with fixed learning rates made it difficult to guarantee a good performance for controlling the real process. On the other hand, as justified by the high convergence rate, Newton-Raphson method is computationally costly, making NGPC inefficient for complex nonlinear systems.

1.4 Thesis Overview

The research objectives are to study the design of GPC based on RBF-NNs for nonlinear stochastic systems. Several optimization methods are used to improve the performance of the GPC design. The efficiency and performance of the proposed GPC was tested by simulations on two different nonlinear stochastic systems. Particularly, the proposed design method is applied to the system of Heating, Ventilating, and Air Conditioning (HVAC). The thesis consists of six chapters. A general background on NNGPC is discussed in the first chapter. The second chapter focuses on solving the optimization problems. In this chapter, several different methods are discussed. Chapter 3 presents one modeling method for the nonlinear stochastic system by using RBF-NN. Design of RBF-NNGPC for multi-input multi-output (MIMO) nonlinear stochastic system is given in Chapter 4. Chapter 5 provides the simulation results to illustrate control performance of the designed controller. Different optimization algorithms are used to minimize the cost function. Chapter 6 concludes the thesis by comparing the simulation results based on different plants and different optimization algorithms and presents some proposals for future work.

Chapter 2

Optimization Problem

The optimization problem is the main concept for RBF-NN training and predictive controller. Many algorithms are used for solving optimization problems [25]. The gradient-based method is one of them.

We define a multivariable function$E(\theta)$, which has the input $\theta = [\theta_1, \theta_2, \cdots, \theta_n]^T$. An algorithm is applied to explore the input space efficiently, so that the optimal parameter θ^* can be found by minimizing $E(\theta)$. Let θ_k denote parameters in kth iteration, the parameters θ_{k+1} in the next iteration can be formulated as

$$\theta_{k+1} = \theta_k + \alpha p_k, \quad k = 1,2,3, \cdots \tag{2.1}$$

where p_k is the search direction, α is some positive step size regulating to what extent to proceed in that direction, k denotes the current iteration number. The θ_k is intended to converge to the optimal value θ^*.

First, we determine the direction p_k, then we calculate the step size α. The next step parameter θ_{k+1} should satisfy Equation (2.2).

$$E(\theta_{k+1}) = E(\theta_k + \alpha p_k) < E(\theta_k) \tag{2.2}$$

The optimization problems can be solved by using gradient-based methods, such as Newton method, Quasi-Newton method, gradient descent method, which require to compute gradient. They can be solved by using gradient-free method, such as Nelder-Mead method.

2.1 Gradient Descent Method

The gradient descent method, also called steepest descent method, is one of the oldest methods for finding the local minima in multidimensional input functions [25]. If the function $E(\theta)$ is defined and differentiable in a neighborhood of a point θ, the local minima can be found from α in a direction of the negative gradient of $E(\theta)$ at α. This method is most frequently used.

The gradient of a differentiable function $E: R^n \rightarrow R$ at θ is the vector of first derivatives of E, denoted as g

$$g(\theta) = \nabla E(\theta) \triangleq \left[\frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}, \cdots, \frac{\partial E(\theta)}{\partial \theta_n}\right]^T \tag{2.3}$$

Equation (2.1) can be changed into Equation (2.4) if we chose the search direction p_k as $-\mathrm{g}(\theta)$,

$$\theta_{k+1} = \theta_k - \alpha g(\theta_k) \tag{2.4}$$

The negative steepest descent direction $-g$, which is computed from θ_k, points to the locally steepest downhill direction. However, it may not point to the global minimum point θ^*, as shown in Fig. 2.1.

The basic idea of gradient descent method is to find a value of θ_{k+1} that satisfies the following:

$$g(\theta_{k+1}) = \frac{\partial E(\theta)}{\partial \theta}|_{\theta=\theta_{k+1}} = 0 \tag{2.5}$$

Equation (2.5) is a necessary condition for a given objective function $E(\theta)$ to reach its stationary point. In practice, however, it is difficult to solve Equation (2.5) analytically. For minimizing the objective function, the descent procedures are typically repeated until one of the following stopping criteria is satisfied:

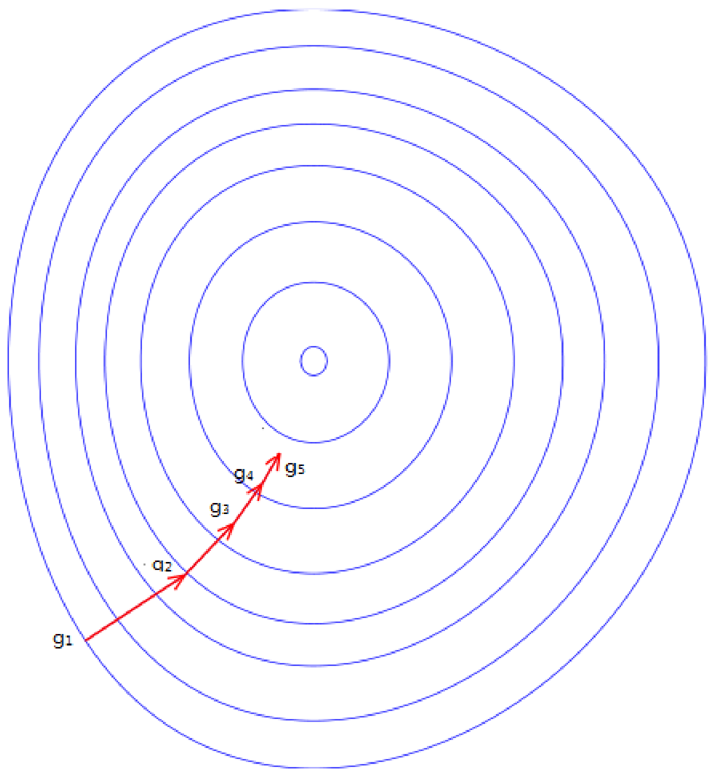


Figure 2.1: Gradient descent direction

1. The objective function value is sufficiently small;

2. The length of the gradient vector g is smaller than a specified value;

3. The specified computing time is exceeded.

In the gradient descent method, if we use a small fixed step size α, the magnitude of the step αg in Equation (2.4) automatically changes at each iteration due to different gradients of g. The convergence rate will be slow if g tends to be infinitesimally small. In this case, the minimization process is inefficient. On the other hand, if we chose a large step size, the gradient descent has a zigzag trajectory, and oscillatory behavior makes the search unstable when step size is too large. The effects of different fixed step sizes for gradient descent method are shown in Fig. 2.2.

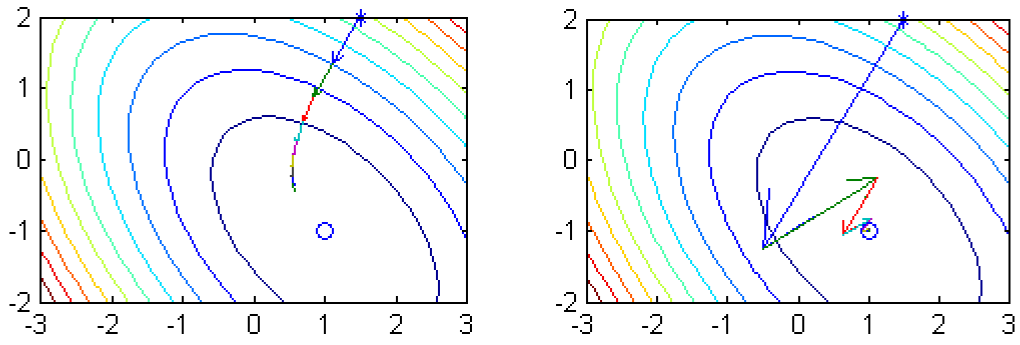


Figure 2.2: The effects of different fixed step size for gradient descent method

A different version of gradient descent method can be obtained by a small change of Equation (2.4),

$$\theta_{k+1} = \theta_k - \kappa \frac{g(\theta_k)}{||g(\theta_k)||} \tag{2.6}$$

where κ is called the actual step size, which indicates the Euclidean distance of the transition from θ_k to θ_{k+1}. This modified gradient descent method with a fixed κ always makes the same strides nomatter how steep the slope is, which is more efficient than the original method [16].

2.2 Quasi-Newton Method

The descent direction can also be determined by using second derivatives of function E if available, as in Newton method, which consumes more computation power. However, Quasi-Newton methods require only the gradient of the objective function, as in gradient descent method. By measuring the changes in gradients, the inverse of Hessian matrix is approximated. Since second derivatives are not required, Quasi-Newton methods are more efficient than Newton's method. Since introduced in 1970, Broyden-Fletcher-Goldfarb-Shanno(BFGS) method becomes one of the most popular methods of this class [16].

The approximated quadratic form of function E can be expressed by second order Taylor series expansion

$$E(\theta_{k+1}) \approx E(\theta_k) + g^T(\theta_{k+1} - \theta_k) + \frac{1}{2}(\theta_{k+1} - \theta_k)^T H_k(\theta_{k+1} - \theta_k) \tag{2.7}$$

H is the Hessian matrix, which is a square matrix of second derivatives of function E.

Differentiating Equation (2.7) with respect to $(\theta_{k+1} - \theta_k)$,

$$H_k(\theta_{k+1} - \theta_k) = g_{k+1} - g_k \tag{2.8}$$

This equation indicates that the Hessian matrix H can be approximated by $(\theta_{k+1} - \theta_k)$ and $g_{k+1} - g_k$,

$$(\theta_{k+1} - \theta_k) = H_k^{-1}(g_{k+1} - g_k) \tag{2.9}$$

Define $\Delta\theta_k = \theta_{k+1} - \theta_k$, and $\Delta g_k = g_{k+1} - g_k$. Then, Equation (2.9) can be rewritten as,

$$\Delta\theta_k = H_k^{-1}\Delta g_k \tag{2.10}$$

Let us now suppose that the search direction has the form,

$$p_k = H_k^{-1} g_k \tag{2.11}$$

Then, in the new iteration, Equation (2.1) becomes

$$\theta_{k+1} = \theta_k - \alpha_k p_k$$

where the initial H_0 is chosen as I, H_k is an approximation to the Hessian. Instead of computing H_k at every iteration, the inverse of H_k is updated at every iteration by a Quasi-Newton updating formula defined by Equation (2.12), which is called BFGS formula.

$$H_{k+1}^{-1} = H_k^{-1} + \frac{\left((\alpha_k p_k)^T \Delta g_k + \Delta g_k^T H_k^{-1} \Delta g_k\right)\left((\alpha_k p_k)(\alpha_k p_k)^T\right)}{((\alpha_k p_k)^T \Delta g_k)^2} - \frac{H_k^{-1}\Delta g_k (\alpha_k p_k)^T + (\alpha_k p_k) H_k^{-1} \Delta g_k^T H_k^{-1}}{(\alpha_k p_k)^T \Delta g_k} \tag{2.12}$$

When p_k is defined by Equation (2.10) and H_k is positive definite, we have

$$-g_k^T p_k = -g_k^T H_k^{-1} g_k < 0 \tag{2.13}$$

2.3 Line Search

For a general function E, the ideal choice would be the global minimizer of the univariate function ϕ, which is defined by

$$\phi(\alpha) = E(\theta_k + \alpha p) \tag{2.14}$$

Analytically solving $\phi'(\alpha) = 0$ will find the local minima, but in general, it is often impossible to identify this point. To find even a local minimizer to moderate precision generally requires too many evaluations of the objective function E and possibly the gradient g. That is, the univariate function $\phi(\alpha)$ should be minimized on the line determined by the current point θ_k and the direction p. This is accomplished by line search methods.

The line search algorithms try out a sequence of candidate values for α, stopping to accept one of these values when certain conditions are satisfied. The line search is done in two stages: A bracketing phase finds an interval containing desirable step lengths, and a bisection or interpolation phase computes a good step length within this interval.

In order to satisfy Equation (2.2), suitable step length α_k will be chosen by an inexact line search.

The process of determining α^* that minimizes $\phi(\alpha)$ is achieved by searching on the line for the minimum.

If $\phi(\alpha)$ and $\phi'(\alpha)$ are available, then the second derivative can be approximated by

$$\phi"(\alpha_k) = \frac{\phi\prime(\alpha_k) - \phi\prime(\alpha_{k-1})}{\alpha_k - \alpha_{k-1}} \tag{2.15}$$

The Newton method is used to determine α_{k+1}

$$\alpha_{k+1} = \alpha_k - \frac{\phi\prime(\alpha_k)}{\frac{\phi\prime(\alpha_k) - \phi\prime(\alpha_{k-1})}{\alpha_k - \alpha_{k-1}}} \tag{2.16}$$

A popular inexact line search condition stipulates that α_k should first of all give sufficient decrease in the objective function E, as measured by

$$E(\theta_k + \alpha p_k) \leq E(\theta_k) + c_1 \alpha g_k^T p_k \tag{2.17}$$

For some constant $c_1 \in (0,1)$, the reduction in E should be proportional to both the step length α and the directional derivative $g_k^T p_k$.

The sufficient decrease condition is not enough by itself to ensure that the algorithm makes reasonable progress. To rule out unacceptably short steps, a second requirement, called the curvature condition, is introduced, which requires α satisfying

$$g(\theta_k + \alpha p_k)^T p_k \geq c_2 g_k^T p_k \tag{2.18}$$

For some constant $c_2 \in (0,1)$, the inequality (2.18) ensures that the slope has been reduced sufficiently. In practice, c_1 is usually chosen to be quite small while c_2 is much larger; for quasi-Newton methods, Nocedal [25] gives example values of $c_1 = 10^{-4}$ and $c_2 = 0.9$.

The sufficient decrease condition (2.17) and curvature condition (2.18) are known collectively as the Wolfe conditions [25].

However, a step length may satisfy the Wolfe conditions without being particularly close to a minimizer of ϕ. We can modify the curvature condition to force α to lie in at least a broad neighborhood of a local minimizer or stationary point of ϕ. The strong Wolfe conditions require α_k satisfying

$$|g(\theta_k + \alpha p_k)^T p_k| \leq |c_2 g_k^T p_k| \tag{2.19}$$

(2.17) and (2.19) together form the so-called strong Wolfe conditions [25], and force α to lie close to a critical point of ϕ.

In practice, it is nearly impossible to obtain the exact minimum point of the function ϕ by the aforementioned methods of line searches. A reasonable stopping criterion must be established to terminate the search procedures before they have converged.

The only difference between the Wolfe and strong Wolfe conditions is that the strong Wolfe conditions no longer allow the derivative $\phi'(\alpha_k)$ to be too large. Hence, we exclude points that are far from stationary points of ϕ.

It is not difficult to prove that there exists a step length α that satisfies the Wolfe conditions (2.17) and (2.18) for every function E which is smooth and bounded below [25]. Since $\phi(\alpha) = E(\theta_k + \alpha p_k)$ is bounded below for all $\alpha > 0$ and since $c_1 \in (0,1)$, the line $l(\alpha) = E(\theta_k) + c_1 \alpha g_k^T p_k$ must intersect the graph of $\phi(\cdot)$ at least once. Let $\alpha_s > 0$ be the smallest intersecting value of α, we have,

$$E(\theta_k + \alpha_s p_k) = E(\theta_k) + \alpha_s c_1 g_k^T p_k \tag{2.20}$$

The sufficient decrease condition (2.18) clearly holds for all step lengths less than α_s.

By the mean value theorem, there exists $\alpha_m \in (0, \alpha_s)$ such that

$$E(\theta_k + \alpha_s p_k) - E(\theta_k) = \alpha_s g((\theta_k + \alpha_m p_k)^T) p_k \tag{2.21}$$

By combining (2.19) and Equation (2.20), we obtain

$$g(\theta_k + \alpha_m p_k)^T p_k = c_1 g_k^T p_k > c_2 g_k^T p_k \tag{2.22}$$

since $c_1 < c_2$, and $g_k^T p_k < 0$, α_m satisfies the Wolfe conditions (2.17) and (2.18), and the inequalities hold strictly in both of the conditions. Hence function E is continuously differentiable since that we assume E is smooth, there is an interval around α_m for which the Wolfe conditions hold. Moreover, since the term in the left-hand side of (2.22) is negative due to $p_k < 0$, the strong Wolfe conditions (2.19) hold in the same interval.

2.4 Nelder-Mead Method

Since published in 1965, the Nelder–Mead method or downhill simplex method [24] or amoeba method is commonly used in nonlinear optimization problems. It is a type of unconstrained nonlinear optimization method. This method is derivative-free for multi-dimensional function optimization, which means this method does not need any functional derivative information to solve the local minima for a given function. Instead, this method relies exclusively on repeated evaluations of the objective function, and the subsequent search direction after each evaluation follows certain heuristic guidelines.

The Nelder-Mead method maintains at each step a non-degenerate simplex, a geometric figure in n dimensions of nonzero volume that is the convex hull of $n+1$ vertices [17].

Considering the same n-dimensional function $E(\theta)$ as gradient descent method, we use an initial simplex, which has $n+1$ points $\theta_1, \theta_2, \cdots, \theta_n, \theta_{n+1} \in R^n$, and the corresponding set of function values at $E_j = E(\theta_j)$ for $j = 1,2,\cdots,n,n+1$. Then the method repeatedly replaces the worst point with the new test point and so the technique progresses.

Ordering: re-arrange the order of $E_j$to satisfy $E_1 \le E_2 \le \cdots \le E_n \le E_{n+1}$.

Centroid: let θ_m be the centroid of all points except θ_{n+1}.

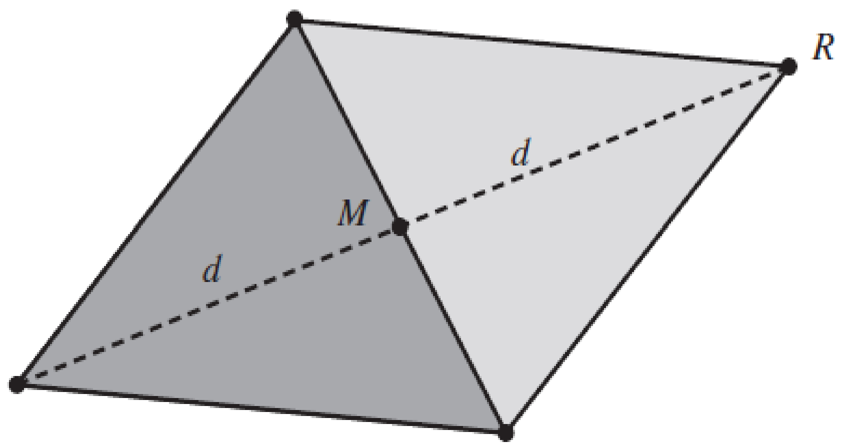


Figure 2.3: Reflection

Reflection: compute reflected point θ_r and its value E_r as

$$\theta_r = \theta_m + \alpha(\theta_m - \theta_{n+1})$$

$$E_r = E(\theta_r) \tag{2.23}$$

where α is the reflection. Thus θ_r is on the line joining θ_{n+1} and θ_m as shown in Fig. 2.3. If $E_1 \le E_r < E_n$, accept E_r and terminate the iteration, then obtain a new simplex by replacing the worst point θ_{n+1} with the reflected point θ_r.

Expansion: If the reflected point is the best point so far $E_r \le E_1$, define the expansion point θ_e and its value E_e as

$$\theta_e = \theta_m + \gamma(\theta_m - \theta_{n+1})$$

$$E_e = E(\theta_e) \tag{2.24}$$

where γ is the expansion coefficient which is greater than unity. If $E_e < E_r$, accept θ_e and terminate the iteration, then obtain a new simplex by replacing the worst point θ_{n+1} with the expanded point θ_e. Otherwise (if $E_e \ge E_r$), accept θ_r, terminate the iteration, and obtain a new simplex by replacing the worst point θ_{n+1} with the expanded point θ_r.

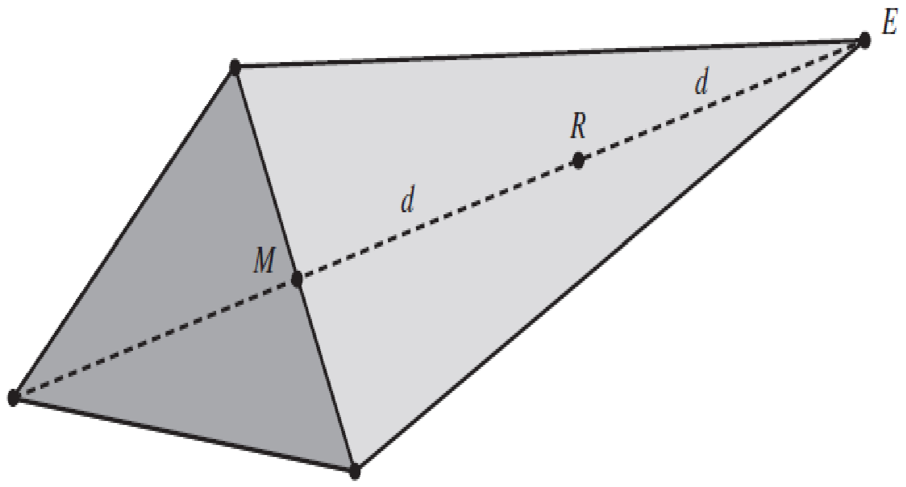


Figure 2.4: Expansion

Contraction: Define the contraction point θ_c and its value E_c as

$$\theta_c = \theta_m + \rho(\theta_m - \theta_{n+1})$$

$$E_c = E(\theta_c) \tag{2.25}$$

where ρ is the contraction coefficient. If the contracted point is better than the worst point $E_c < E_{n+1}$, accept θ_c, terminate the iteration, and obtain a new simplex by replacing the worst point θ_{n+1} with the expanded point θ_c.

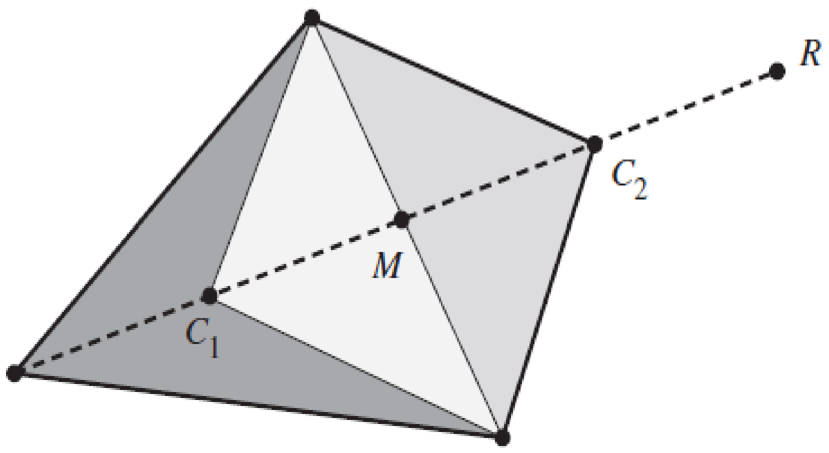


Figure 2.5: Contraction

Reduction: For all but the best point, replace the point with,

$$\theta_j = \theta_1 + \sigma(\theta_j - \theta_1) \quad \text{for } j = 2,3,\cdots,n+1 \tag{2.26}$$

where σ is shrink coefficient. Commonly, $\alpha = 1,\ \gamma = 2,\ \rho = -0.5$ and $\sigma = 0.5$ are used suggested by Nelder and Mead [24].

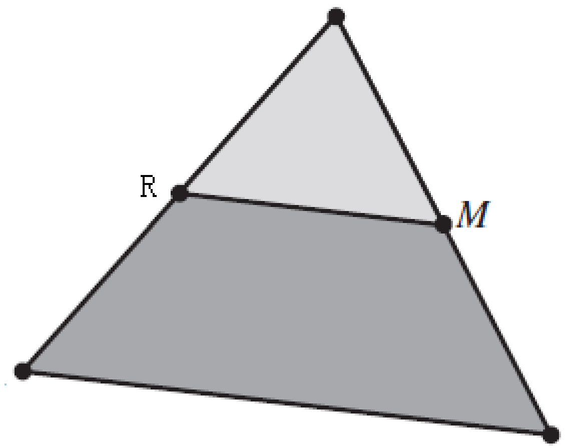


Figure 2.6: Reduction

The iteration is repeated until the stopping criterion is met. The stopping criteria are often composed of four different parts as follows:

Computation time: the process will stop when a specified iteration count is reached.

Optimization goal: the function value $E(\theta_k)$ is smaller than the desired goal value.

Minimal improvement: $|E(\theta_k) - E(\theta_{k-1})|$ is smaller than a given value.

Minimal relative improvement: $\frac{|E(\theta_k)-E(\theta_{k-1})|}{E(\theta_{k-1})}$ is smaller than a preset value.

Chapter 3

Modeling

3.1 NARMAX Model

In many cases the state variables of nonlinear stochastic systems (Fig. 3.1) are not measurable. This kind of systems is often characterized by an input-output equation. The Nonlinear Auto-Regressive Moving Average model with eXogenous input (NARMAX) [3] [20], is a general representation of a nonlinear system which is described by a set of nonlinear difference equations. The NARMAX model provides a unified solution for a finitely realizable nonlinear system [2] [8]. Consider a nonlinear stochastic system represented by NARMAX model,

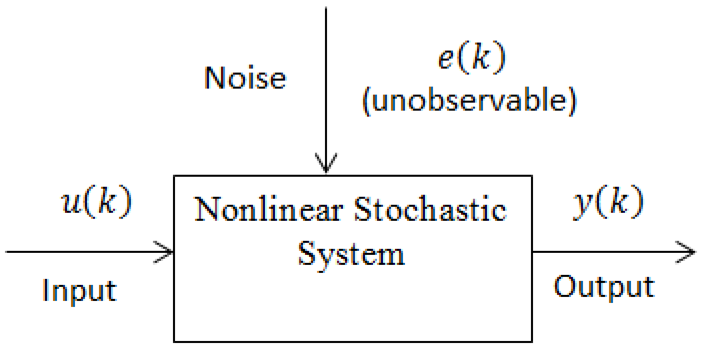


Figure 3.1: Block diagram of nonlinear stochastic system

$$y(k) = f(y(k-1), \cdots, y(k-n), u(k-1-d), \cdots, u(k-1-m-d),$$
$$e(k-1) \cdots, e(k-p)) + e(k) \tag{3.1}$$

where $y(k) \in R^{ny}$ $u(k) \in R^{nu}$ represent the output vector and the input vector of the Multi-Input Multi-Output (MIMO) nonlinear system at time k, $e(k) \in R^{ne}$ represent zero mean white noise, n m, and p are the orders of the output, input, and noise respectively; d is the time delay of the process; $f(\cdot)$ is a nonlinear function. The input-output relationship (3.1) is dependent upon the nonlinear function $f(\cdot)$, and $f(\cdot)$ is usually unknown. For simplification without loss of generality, a special case of NARMAX model is considered here which can be written as

$$y(k) = f(y(k-1), \cdots, y(k-n), u(k-1-d), \cdots, u(k-1-m-d)) + e(k) \tag{3.2}$$

3.2 Modeling by RBF NNs

The NARMAX represents stochastic system with finite order and can be readily implemented by NNs [5] [7]. RBF-NNs are artificial NNs that use radial basis functions as activation functions. Our aim is to use RBF-NNs to model nonlinear stochastic systems described by (3.2).

Define $p = n * ny + (m+1) * nu$, and

$$x(k) = \left[x_1, x_2, \cdots, x_p\right]^T = [y(k-1)^T, \cdots, y(k-n)^T,$$
$$u(k-1-d)^T, \cdots, u(k-1-d-m)^T]^T \tag{3.3}$$

$x(k) \in R^p$, $k = 1,2, \cdots, S$ is the *k*th input to the RBF-NN. *S* represents the number of samples. The jth output of the neural network is described by,

$$\hat{y}_j = \sum_{i=1}^{h} w_{ij}\varphi_i\big(x(k)\big) \quad j = 1,2,\cdots,L \tag{3.4}$$

where w_{ij} $(i = 1,2,\cdots,h)$ is the weights between hidden layer and output layer, h denotes the number of cells in hidden layer. $\varphi_i(\cdot)$ is the activation function between input layer and hidden layer chosen as

$$\varphi_i\big(x(k)\big) = exp\left(\sum_{a=1}^{a=p}\left[-\frac{(x_a-c_{ai})^2}{s_{ai}^2}\right]\right) \tag{3.5}$$

where $c_i \in R^p$ and $s_i \in R^p$ are the center and width of Gaussian activation functions for the neurons in the hidden layer.

Let $Z = \{c_i,\ s_i,\ w_{ij}\} = [z_1, z_2, \cdots, z_{nz}]$ be the parameter vector which is composed of all the widths, centers and weights of the RBF-NN where $nz = p * h + p * h + h * ny$. A three-phase training of RBF-NNs [29] is used by performing an adaptation of all parameters simultaneously. The neural network is then defined by the model,

$$\hat{y}(k, Z) = \hat{f}(x(k), Z) \tag{3.6}$$

The RBF-NN model (3.6) is a one-step-ahead predictive model for $y(k)$ and the prediction error is given by,

$$\varepsilon(k) = y(k) - \hat{y}(k, Z) \tag{3.7}$$

The parameter vector Z can be determined by minimizing the following objective function

$$J(Z) = \frac{1}{2}\sum_{k=1}^{S}\big(y(k) - \hat{y}(k, Z)\big)^T (y(k) - \hat{y}(k, Z)) \tag{3.8}$$

The optimization problem can be solved using modified gradient descent method. Let Z_K denotes the Kth iteration values of the parameters. Z_{K+1} in the next iteration can be formulated as

$$Z_{K+1} = Z_K - \kappa \frac{\nabla J(Z_K)}{||\nabla J(Z_K)||} \tag{3.9}$$

where κ is the real learning rate vector, K denotes iterative epochs, and $||\cdot||$ denotes the Euclidean norm. $\nabla J(\cdot)$ is the gradient search direction defined as

$$\nabla J(Z) = \left[\sum_{k=1}^{S}\big(y(k) - \hat{y}(k,Z)\big)^T \frac{\partial \hat{y}(k,Z)}{\partial Z}\right]^T \tag{3.10}$$

where

$$\left[\frac{\partial \hat{y}}{\partial Z}\right]^T = \left[\left[\frac{\partial \hat{y}}{\partial c}\right]^T, \left[\frac{\partial \hat{y}}{\partial s}\right]^T, \left[\frac{\partial \hat{y}}{\partial w}\right]^T\right]^T$$

$$[\frac{\partial \hat{y}}{\partial c}]^T = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial c_1} & \cdots & \frac{\partial \hat{y}_{ny}}{\partial c_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_1}{\partial c_p} & \cdots & \frac{\partial \hat{y}_{ny}}{\partial c_p} \end{bmatrix}_{ny\times p}$$

$$[\frac{\partial \hat{y}}{\partial s}]^T = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial s_1} & \cdots & \frac{\partial \hat{y}_{ny}}{\partial s_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_1}{\partial s_p} & \cdots & \frac{\partial \hat{y}_{ny}}{\partial s_p} \end{bmatrix}_{ny\times p}$$

$$[\frac{\partial \hat{y}}{\partial c}]^T = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial w_{11}} & \cdots & \frac{\partial \hat{y}_{ny}}{\partial w_{1ny}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_1}{\partial w_{p1}} & \cdots & \frac{\partial \hat{y}_{ny}}{\partial w_{pny}} \end{bmatrix}_{ny\times p}$$

$$\frac{\partial \hat{y}_j}{\partial c_{ai}} = -2w_{ij}\varphi_i \frac{x_a - c_{ai}}{s_{ai}^2}$$

$$\frac{\partial \hat{y}_j}{\partial s_{ai}} = -2w_{ij}\varphi_i \frac{(x_a - c_{ai})^2}{s_{ai}^3}$$

$$\frac{\partial \hat{y}_j}{\partial w_{ij}} = \varphi_i \tag{3.11}$$

The training procedure of the RBF-NN given by Equation (3.4) is presented below.

1) Select suitable input variables $x(k)$, p and the initial values for Z_0.

2) Compute $E = [\varepsilon(1), \varepsilon(2), \cdots, \varepsilon(S)]^T$ using Equation (3.7) for $k = 1,2, \cdots, S$.

3) Compute $\nabla J(Z)$ by using Equation (3.10) and Equation (3.11).

4) Update the parameter Z using Equation (3.9).

5) Repeat the step 2) to 4) until termination rule is satisfied. The iteration stops when $\left|\frac{\Delta J(Z)}{J(Z)}\right|$ is smaller than a preset value, or $J(Z)$ is smaller than the designed goal value, or a designed iteration count is reached.

3.3 Multistep-Ahead Predictive Model

From Equation (3.2), the NARMAX one-step-ahead predictive model approximated from RBF-NN is

$$y_f(k+1) = f(y(k), \cdots, y(k-n+1), u(k-d), \cdots, u(k-m-d)) \quad (3.12)$$

where $y_f(k+1)$ is the predictive output of the model. Extending Equation (3.12) one more step ahead, $y_f(k+2)$ can be obtained as

$$y_f(k+2) = f(y_f(k+1), y(k), \cdots, y(k-n+2), u(k-d+1), \cdots, u(k-m-d+1)) \quad (3.13)$$

Using a recursive technique, d-step-ahead predictor [33] can be derived

$$y_f(k+d) = f(y_f(k-1+d), \cdots, y_f(k+1), y(k), \cdots, y(k-n+d), u(k-1), \cdots, u(k-1-m)) \quad (3.14)$$

The d-step-ahead predictive model is given by Fig. 3.2. This predictor has a series connection architecture. In this architecture only one RBF-NN is used, but d iterations are required for long term prediction. In the first iteration, the RBF-NN is fed with the present process output $y(k)$ and several past process outputs $y(k-1),\cdots,y(k-n+1)$, together with the past inputs $u(k-d+1),\cdots,u(k-m-d+1)$. Then, the RBF-NN is iterated. In the iteration procedure, any error of the model is fed back into the RBF-NN. The prediction accuracy can be significantly degraded due to an accumulation of errors if the prediction horizon is large. To offset the error of the predictive model, a variable $y_c(k)$ is introduced to express the difference between actual output of system $y(k)$ and predictive output of RBF-NN predictor $y_f(k)$. Let

$$y_c(k) = y(k) - y_f(k) \tag{3.15}$$

The estimated output $y_e(k)$ can be described by

$$y_e(k+i) = y_f(k+i) + y_c(k) \quad i = 1,2,\cdots,d \tag{3.16}$$

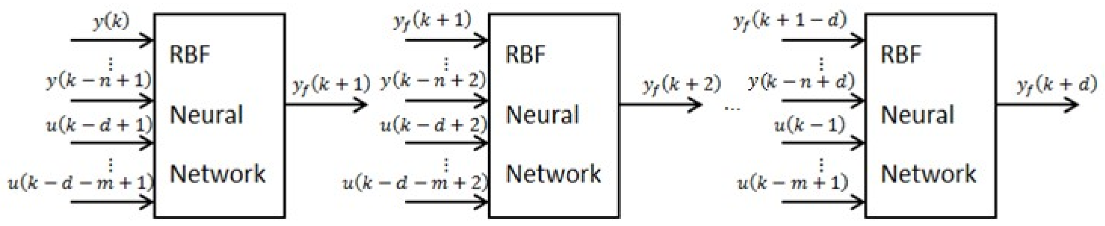


Figure 3.2: The architecture of d-step-ahead predictive model

Chapter 4

Design of Generalized Predictive Control

4.1 Cost Function

A large number of predictive control algorithms have been presented and widely used in industry. GPC has a precise model which is used to predict the system behavior. It shows good robustness of adaptive model-based predictive control [10]. The quality of the system model affects the accuracy of prediction. For a linear system, a number of techniques are available to make modeling easier; however it is more complicated for nonlinear systems. Currently there are two techniques used to model nonlinear systems. One is to linearize the system about a set of operating points. If the plant is highly non-linear the number of operating points can be very large. The second technique involves developing a nonlinear model which depends on making assumptions about the dynamics of the nonlinear system. The accuracy of the model will be reduced if these assumptions are incorrect.

NNs are powerful tools for identification and modeling of nonlinear systems. The model estimated using NN can be used to predict the system output. A nonlinear multi-step-ahead predictive model based on RBF-NN mentioned in Chapter 3 is employed to design GPC for a nonlinear stochastic system.

The block diagram of the designed GPC is shown in Fig. 4.1 [13]. The RBF-NNs work as a plant model and a predictor. To offset the error of the predictive model, a variable y_c is introduced to express the difference between the actual output of system y and predictive output of RBF-NN predictor y_f. By assuming y_c keeps the same for different time instants, the estimated output y_e can be obtained from the summation of y_f and y_c.

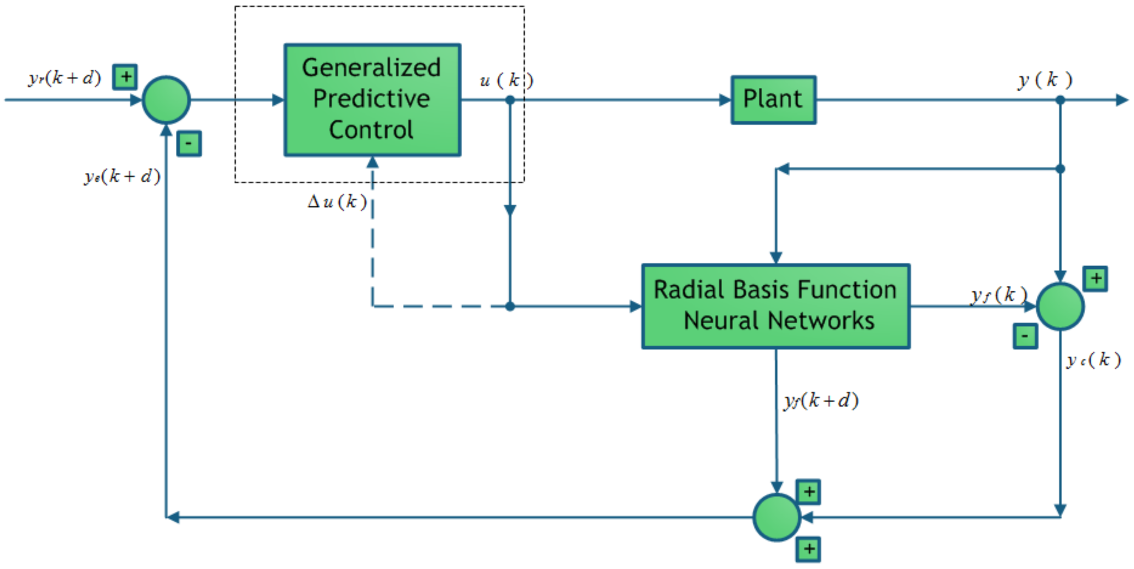


Figure 4.1: The block diagram of RBF-NN based GPC

The control objective of GPC is to use the NN predictor for predicting future outputs, and to minimize the following cost function in order to reduce prediction errors

$$J = \sum_{i=N_0}^{N_1}[y_r(k+i) - y_e(k+i)]^2 + \sum_{j=1}^{N_u}\lambda_j[\Delta u(k-1+j)]^2 \qquad (4.1)$$

where $y_r(k+i)$ $(i = N_0, N_0+1, \cdots, N_1)$ denotes the future reference trajectory,

$y_e(k+i)$ is the estimated output, N_0 and N_1 are the minimum and maximum prediction horizons, N_u is the control horizon, λ_j $(j = 1,2,\cdots,N_u)$ is a weighting coefficient penalizing changes in the control input.

This cost function minimizes not only the mean squared error between the reference and predicted outputs, but also the weighted squared rate of change of the control input with its constraints. When this cost function is minimized, a control input that meets the constraints is generated that allows the plant to track the reference trajectory within some tolerance.

There are four tuning parameters in the cost function, N_0, N_1, N_u and λ_j. The predictions of the plant will run from N_0 to N_1 future time steps. The bound on the control horizon is N_u. The only constraint on the values of N_0, and N_u is that these bounds must be less than or equal to N_1 as

$$1 \le N_u \le N_1,\ d+1 \le N_0 \le N_1 \tag{4.2}$$

The second summation contains a weighting factor λ_j, which is introduced to control the balance between the two summations. Obviously, in the GPC algorithm, the different selection of these four parameters can have important influence in the controlled performance of the system. N_0 should be larger than or equal to $d+1$, otherwise there will be some outputs that are independent of $u(k)$. Generally taking $N_0 = 1$ when d is unknown or varying. N_1 should include all responses that are explicitly dependent on current control. In order to reduce the online computation time, N_u is usually chosen such that $N_u \le N_1 - N_0 + 1$. λ_j acts as damper on the predicted $u(k)$, which is generally chosen as $\lambda_j = 0$ or a small value unless the change in control is too big.

4.2 Cost Function Minimization

Given N_u, future control inputs can be expressed as a vector

$$u = [u(k)^T, u(k+1)^T, \cdots, u(k+N_u-1)^T]^T \tag{4.3}$$

The control input $u(k) \in R^{nu}$ can be determined by minimizing J in Equation (4.1) with respect to u. This can be done by using the optimization methods mentioned in Chapter 2.

4.2.1 Nelder-Mead Algorithm

The Nelder-Mead algorithm does not need derivatives of the cost function in Equation (4.1); however, it is very sensitive to the initial points, and is not efficient when dimension of vector u becomes big. This algorithm only uses the basic operations of reflection, expansion, contraction and reduction to directly find the solution. Each step replaces at least one vertex, thus changing the shape and the location of the simplex.

First we generate and evaluate initial $N_u + 1$ points $\{u^1, u^2, \cdots, u^{N_u+1}\}$ in R^{N_u}. Set the initial point u^1 to be the initial input of system and form the other N_u points using

$$u^i = u^1 + \sigma q_i \tag{4.4}$$

where q_i is a unit vector, which guarantees that the vector set $\{u^2 - u^1, \cdots,\ u^{N_u+1} -$

$u^1\}$ is linearly independent, so the set of points $\{u^1, u^2, \cdots, u^{N_u+1}\}$ makes a convex hull in R^{N_u}.

Then the Nelder-Mead method will be applied for minimizing the cost function defined by Equation (4.1). The algorithm continues in refining the solution by replacing the worst solution with an improved one in every iteration until reaching the stopping criteria. The algorithm should be terminated if the value of cost function in Equation (4.1) is smaller than a defined threshold, or convergence is reached making no further improvement for further iterations.

4.2.2 Quasi-Newton Algorithm

The objective of the cost function minimization algorithm can also be accomplished by setting the Jacobian matrix of cost function to zero and solving for u in Equation (4.3). By using Quasi-Newton method, J is minimized iteratively to determine the best u. An iterative process yields intermediate values for J denoted J_K.

Quasi-Newton method is widely used in optimization problem. Applying Equation (2.10) to Equation (4.1), the update rule for u is given by Equation (4.5)

$$u_{K+1} = u_K - \alpha_K H_K^{-1} \frac{\partial J_K}{\partial u_K} \tag{4.5}$$

where

$$\frac{\partial J}{\partial u} = -\frac{\partial y_E}{\partial u} e + \lambda \Delta u \tag{4.6}$$

where y_E , e, and Δu are estimated output , error vector, and increment of control input, respectively, defined by

$$y_E = [y_e(k+N_0)^T, y_e(k+N_0+1)^T, \cdots, y_e(k+N_1)^T]^T \tag{4.7}$$

$$e = [[y_r(k+N_0) - y_e(k+N_0)]^T, \cdots [y_r(k+N_1) - y_e(k+N_1)]^T]^T \tag{4.8}$$

$$\Delta u = [[u(k) - u(k-1)]^T, \cdots [u(k+N_u-1) - u(k+N_u-2)]^T]^T \tag{4.9}$$

$$\lambda = diag[\lambda_1, \lambda_2, \cdots, \lambda_{N_u}] \tag{4.10}$$

$\frac{\partial y_E}{\partial u}$ is denoted as a $(N_1 - N_0 + 1) \times N_u$ matrix shown in Equation (4.11)

$$\frac{\partial y_E}{\partial u} = \begin{bmatrix} \frac{\partial y_e(k+N_0)}{\partial u(k)} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \frac{\partial y_e(k+N_u)}{\partial u(k)} & \cdots & \frac{\partial y_e(k+N_u)}{\partial u(k+N_u-1)} \\ \vdots & \cdots & \vdots \\ \frac{\partial y_e(k+N_1)}{\partial u(k)} & \cdots & \frac{\partial y_e(k+N_1)}{\partial u(k+N_u-1)} \end{bmatrix}_{(N_1-N_0+1)\times N_u} \tag{4.11}$$

The only non-zero element in the first row of the matrix is

$$\frac{\partial y_{ej}(k+N_0)}{\partial u(k)} = \sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(u(k)-c_{u(k)})}{s_{u(k)}^2}\right) \tag{4.12}$$

For the second row, there are two non-zero elements, and the chain rule will be applied since $y_e(k+N_0+1)$ is a function of $y_e(k+N_0)$, $u(k)$ and $u(k+1)$ in the neural network model,

$$\frac{\partial y_{ej}(k+N_0+1)}{\partial u(k)} = \frac{\partial y_{ej}(k+N_0+1)}{\partial y_{ej}(k+N_0)}\frac{\partial y_{ej}(k+N_0)}{\partial u(k)} + \sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(u(k)-c_{u(k)})}{s_{u(k)}^2}\right)$$

$$\frac{\partial y_{ej}(k+N_0+1)}{\partial y_{ej}(k+N_0)} = \sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(y_{ej}(k+N_0)-c_{y_{ej}(k+N_0)})}{s_{y_{ej}(k+N_0)}^2}\right)$$

$$\frac{\partial y_{ej}(k+N_0+1)}{\partial u(k+1)} = \sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(u(k+1)-c_{u(k+1)})}{s_{u(k+1)}^2}\right) \tag{4.13}$$

For the third row, 3 non-zero elements are given by

$$\frac{\partial y_{ej}(k+N_0+2)}{\partial u(k)} = \frac{\partial y_{ej}(k+N_0+2)}{\partial y_{ej}(k+N_0)}\frac{\partial y_{ej}(k+N_0)}{\partial u(k)} + \frac{\partial y_{ej}(k+N_0+2)}{\partial y_{ej}(k+N_0+1)}\frac{\partial y_{ej}(k+N_0+1)}{\partial u(k)}$$
$$+ \sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(u(k)-c_{u(k)})}{s_{u(k)}^2}\right)$$

$$\frac{\partial y_{ej}(k+N_0+2)}{\partial y_{ej}(k+N_0)} = \sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(y_{ej}(k+N_0+2)-c_{y_{ej}(k+N_0)})}{s^2_{y_{ej}(k+N_0)}}\right)$$

$$\frac{\partial y_{ej}(k+N_0+2)}{\partial y_{ej}(k+N_0+1)} = \sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(y_{ej}(k+N_0+2)-c_{y_{ej}(k+N_0+1)})}{s^2_{y_{ej}(k+N_0+1)}}\right)$$

$$\frac{\partial y_{ej}(k+N_0+2)}{\partial u(k+1)} = \frac{\partial y_{ej}(k+N_0+2)}{\partial y_{ej}(k+N_0+1)}\frac{\partial y_{ej}(k+N_0+1)}{\partial u(k+1)} + \sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(u(k)-c_{u(k+1)})}{s^2_{u(k+1)}}\right)$$

$$\frac{\partial y_{ej}(k+N_0+2)}{\partial u(k+2)} = +\sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(u(k)-c_{u(k+2)})}{s^2_{u(k+2)}}\right) \tag{4.14}$$

For the $(m+2)$th row $y_e(k+N_0+m+1)$ is a function of $y_e(k+N_0+m),\cdots,$ $y_e(k+N_0+m-n)$, $u(k+m+1),\cdots,u(k+1)$, different from previous partial differential equation, it is given by

$$\frac{\partial y_{ej}(k+N_0+m+1)}{\partial u(k)} = \frac{\partial y_{ej}(k+N_0+m+1)}{\partial y_{ej}(k+N_0+m)}\frac{\partial y_{ej}(k+N_0+m)}{\partial u(k)} + \cdots + \frac{\partial y_{ej}(k+N_0+m+1)}{\partial y_{ej}(k+N_0+m-n)}\frac{\partial y_{ej}(k+N_0+m-n)}{\partial u(k)}$$

$$\frac{\partial y_{ej}(k+N_0+m+1)}{\partial u(k+1)} = \frac{\partial y_{ej}(k+N_0+m+1)}{\partial y_{ej}(k+N_0+m)}\frac{\partial y_{ej}(k+N_0+m)}{\partial u(k+1)} + \cdots + \frac{\partial y_{ej}(k+N_0+m+1)}{\partial y_{ej}(k+N_0+m-n)}\frac{\partial y_{ej}(k+N_0+m-n)}{\partial u(k+1)}$$

$$\frac{\partial y_{ej}(k+N_0+m+1)}{\partial u(k+m+1)} = \sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(u(k)-c_{u(k+m+1)})}{s^2_{u(k+m+1)}}\right) \tag{4.15}$$

Extending q steps, the matrix will have q non-zero elements, and the pth element can be obtained by

$$\frac{\partial y_{ej}(k+N_0+q-1)}{\partial u(k+p-1)} = \frac{\partial y_{ej}(k+N_0+q-1)}{\partial y_{ej}(k+N_0+q-2)}\frac{\partial y_{ej}(k+N_0+q-2)}{\partial u(k+p-1)} + \cdots + \frac{\partial y_{ej}(k+N_0+q-1)}{\partial y_{ej}(k+N_0+q-1-n)}\frac{\partial y_{ej}(k+N_0+q-1-n)}{\partial u(k+p-1)} + \sum_{i=1}^{h} w_{ij}\varphi_i \left(\frac{2(u(k+p-1)-c_{u(k+p-1)})}{s^2_{u(k+p-1)}}\right) \tag{4.16}$$

where $N_0+q-1-n > p+1 \geq 0$, and the last summation term would be zero if $q-p > m+1$.

Hessian matrix H does not need to be computed. The Hessian is updated by analyzing successive gradient vectors instead. By choosing H_0 as an $N_u \times N_u$ identity matrix, for the BFGS update rule, recall Equation (2.10) here

$$H_{K+1}^{-1} = H_K^{-1} + \frac{\left((\alpha_K p_K)^T \Delta g_K + \Delta g_K^T H_K^{-1} \Delta g_K\right)\left((\alpha_K p_K)(\alpha_K p_K)^T\right)}{((\alpha_K p_K)^T \Delta g_K)^2} - \frac{H_K^{-1} \Delta g_K (\alpha_K p_K)^T + (\alpha_K p_K) H_K^{-1} \Delta g_K^T H_K^{-1}}{(\alpha_K p_K)^T \Delta g_K}$$

First we obtain the search direction p_K by solving

$$p_K = -H_0^{-1} \nabla J_K \tag{4.17}$$

then perform a line search to find an acceptable step size α_K in the direction found in the first step such that

$$J(u_K + \alpha_K p_K) = \min_{\alpha \geq 0}(J(u_K + \alpha_K p_K)) \tag{4.18}$$

In the inexact search scheme discussed in Section 2.3, it is selected to satisfy Wolfe conditions (2.16) and (2.18). Using the search result, the new control input vector is updated as

$$u_{K+1} = u_K + \alpha_K p_K \tag{4.19}$$

and the invers Hessian matrix can be updated from H_K^{-1} to H_{K+1}^{-1} by using the following

$$H_{K+1}^{-1} = H_K^{-1} + \frac{(s_K{}^T \Delta g_K + \Delta g_K^T H_K^{-1} \Delta g_K)(s_K s_K{}^T)}{(s_K{}^T \Delta g_K)^2} - \frac{H_K^{-1} \Delta g_K s_K{}^T + s_K H_K^{-1} \Delta g_K^T H_K^{-1}}{s_K{}^T \Delta g_K} \tag{4.20}$$

where

$$s_{K=} \alpha_K p_K$$

$$\Delta g_K = \nabla J_{K+1} - \nabla J_K \tag{4.21}$$

the stopping criterion is set as the cost function $J < \varepsilon$ or K exceeding allowed iteration times. Due to the better convergence property of the BFGS method the RBF-NN will require less hidden neurons to reach the same training error and result in better generalization ability.

Chapter 5

Simulation Results

In order to test the efficiency of the proposed control technique, two sets of simulations have been done to verify the neural network generalized predictive controller. Simula- tion studies were carried out in C++.

5.1 SISO Benchmark Simulation

The first part of simulation, a nonlinear SISO plant taken from literature [6] is considered:

$$y(k) = \frac{2.5y(k-1)y(k-2)}{1+y(k-1)^2+y(k-2)^2} + 0.3\cos\big(0.5y(k-1) - y(k-2)\big)$$
$$+1.2u(k-1) + e(k) \tag{5.1}$$

where y is the output of the plant, u is the plant input, and e is the disturbance.

The RBF neural network structure presented in Fig. 1.1 is used. Comparing Equation (3.3) with Equation (5.1), it is found that $n = 1$ $m = 0$ and $d = 0$. Totally 5 neurons in the hidden layer are chosen in simulations.

5.1.1 The NN Model Training for SISO Plant

A finite sequence of uniform pseudo random noise distributed in the interval

$[-1.5,1.5]$ is applied to Equation (5.1) as the input signal, which is shown in Fig. 5.1. The input signal are holding constant for 100 duration. The plant response to the input signal and its filtered version are shown in Fig. 5.2. The filtered response was obtained through a second-order Butterworth low pass filter with cutoff frequency of 3 Hz in order to remove most of the noise. The Signal-to-Noise Radios (SNR) of the response and the filtered response are 11.50 dB and 33.60 dB, respectively. The filtered response, together with the input signal in Fig. 5.1, is used to train the neural network. The modified gradient method defined in Equation (2.6) is chosen as the training algorithm. After several trial-and-error test, the step size of $\kappa = 0.005$ is selected. In the training procedure, the termination rules are chosen as,

1). The value of the objective function $J(Z) < 1$;

2). The learning iteration $K > 10000$;

3). The change of gradients $||g_k - g_{k-1}|| < 1 \times 10^{-8}$.

Fig. 5.3 shows the outputs of the plant and NN model after the identification procedure was terminated at iteration $K = 10000$. Fig. 5.4 shows the error between the outputs of the plant and NN model after training. Figs. from 5.5 to 5.7 show the parameters for the RBF-NN in the training process.

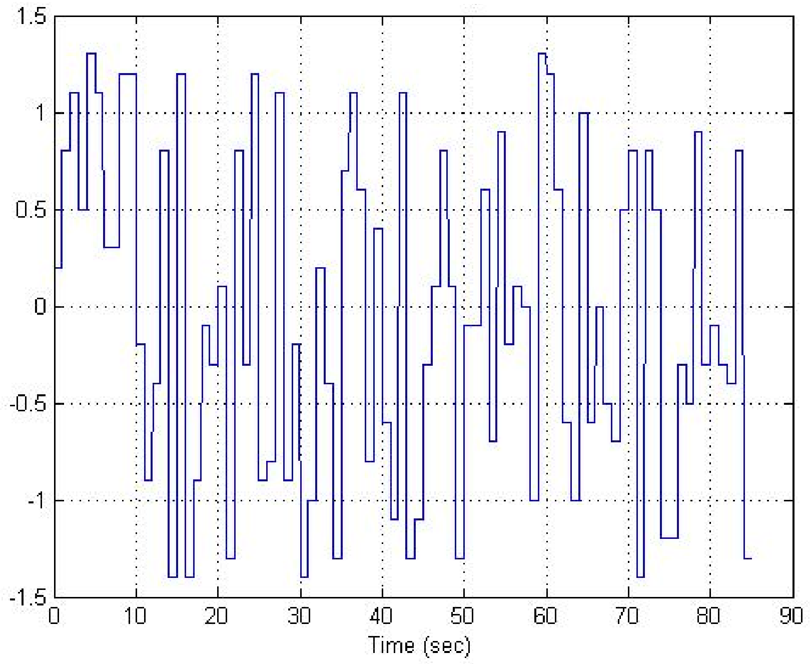


Figure 5.1: The random input signal applied to the plant

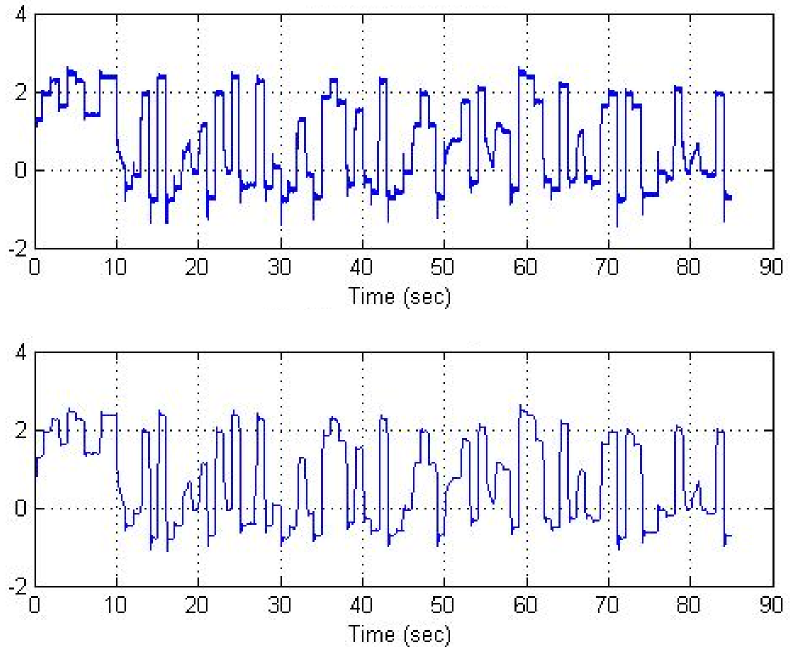


Figure 5.2: Plant response and filtered response (top) for random input signal (bottom)

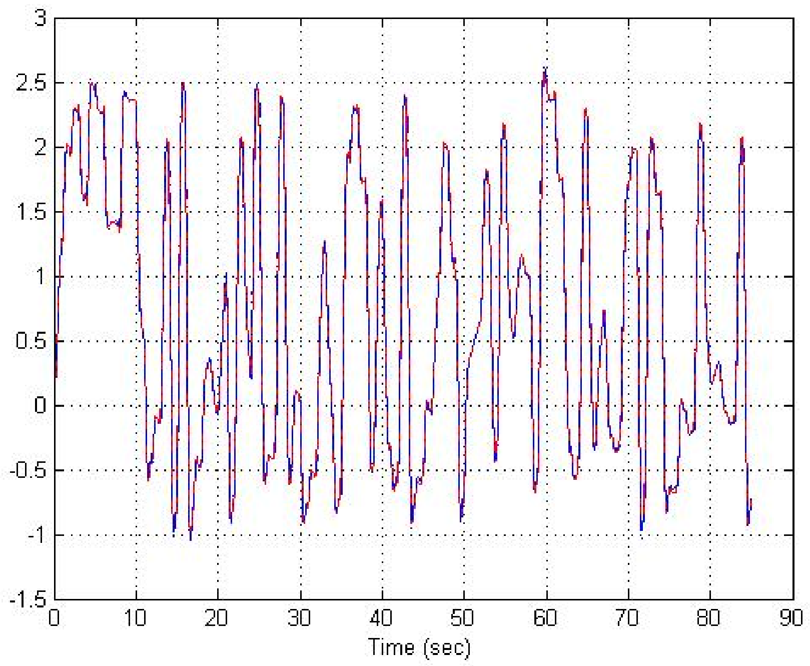


Figure 5.3: The training result after 10000 iterations (red line: filtered plant output, blue line: NN model output)

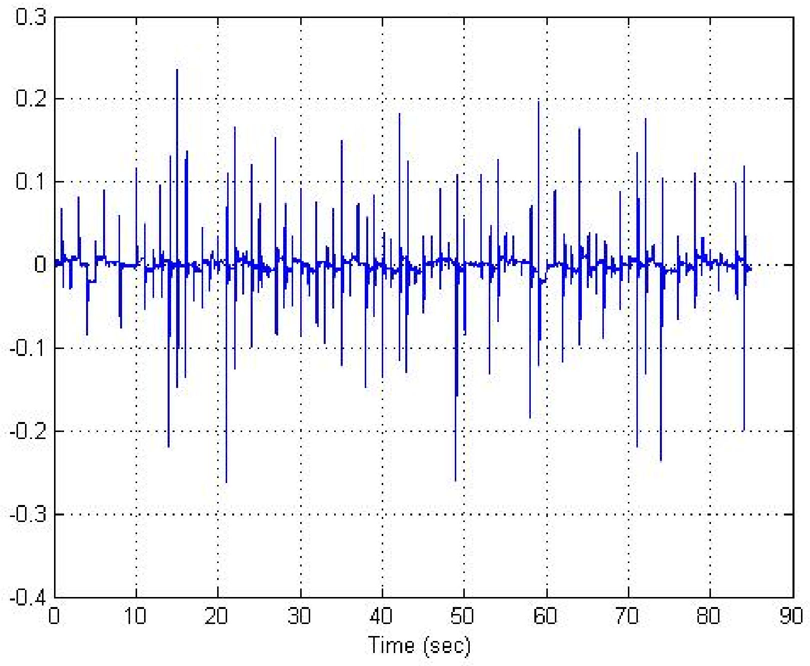


Figure 5.4: The error between the outputs of the plant and model

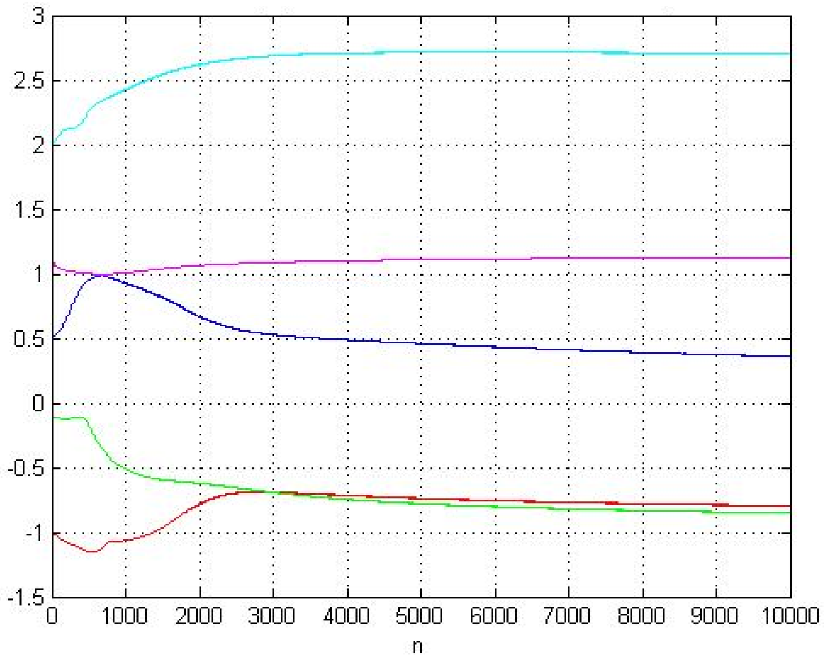


Figure 5.5: The centers of the first neuron in the input layer

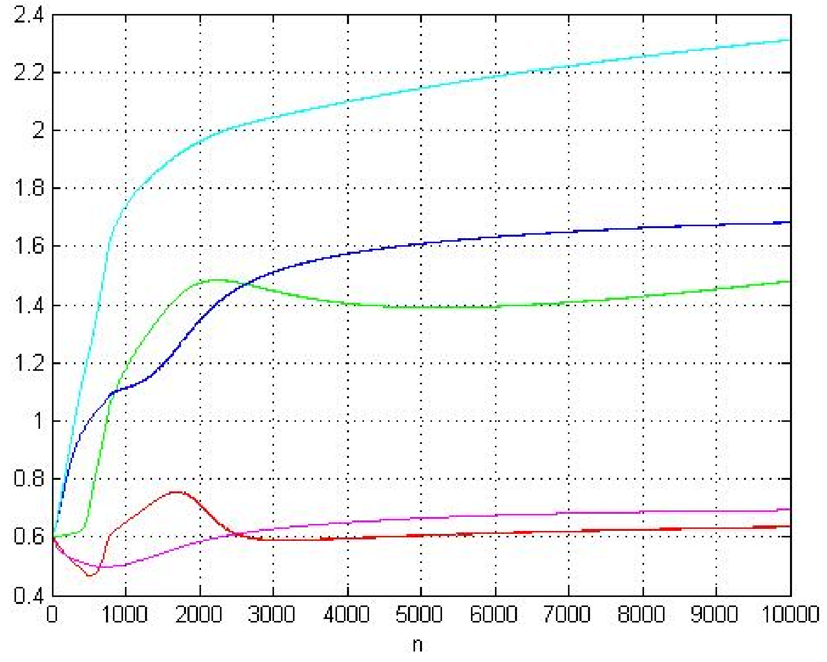


Figure 5.6: The widths of the first neuron in the input layer

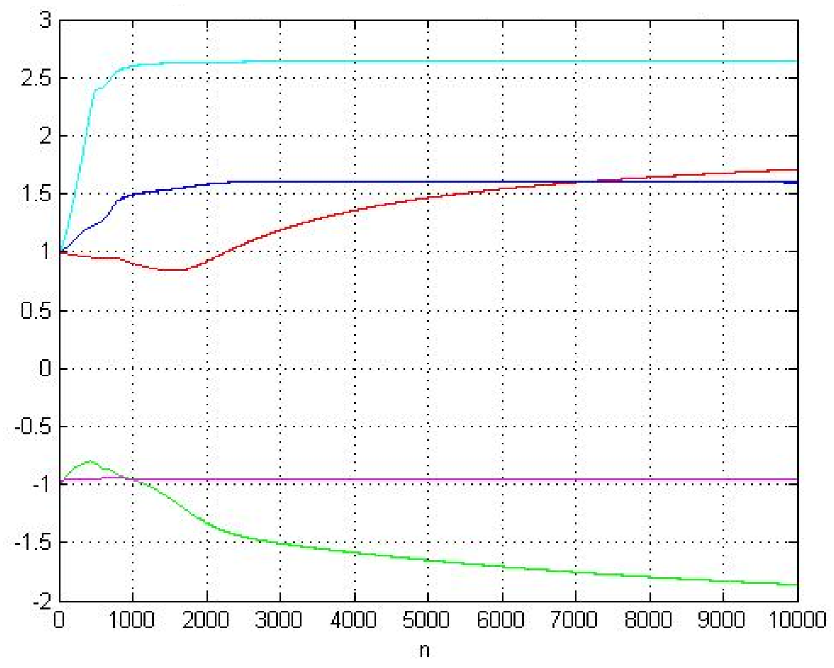


Figure 5.7: The weights of the output layer

5.1.2 The NN Model Validation for SISO Plant

The model is validated using a time validation test for a sinusoidal input signal for this plant, namely

$$u(k) = A * \sin(2\pi f T k) + D \tag{5.2}$$

with the amplitude $A = 1$. The sampling time T is set to $0.01s$. The frequency $f = 1/85$ Hz. The bias is $D = 0.25$. The initial condition of the plant is chosen $(y(k-1), y(k-2)) = (0,0)$. The plant response is filtered by a second-order Butterworth low pass filter with cutoff frequency of 3 Hz. The SNRs of the plant response and filtered response are 31.50 dB and 33.41 dB, respectively. The plant response, the NN model response, and the validation error are shown in Figs. from 5.8 to 5.10, respectively. For this input signal, the Relative Root Mean Squared Error (rRMSE) over 85 seconds has a value of 0.0047, Absolute Mean Error (AME) over 85 seconds has a value of 0.0051, which means that the RBF-NN model is a very good approximation of the nonlinear system.

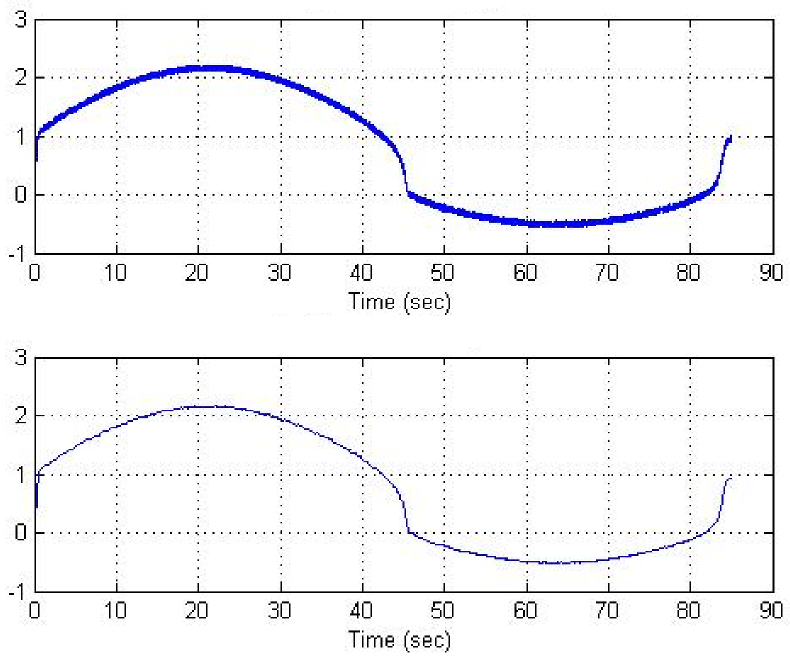


Figure 5.8: Plant response (top) and filtered response (bottom) to sinusoidal input signal

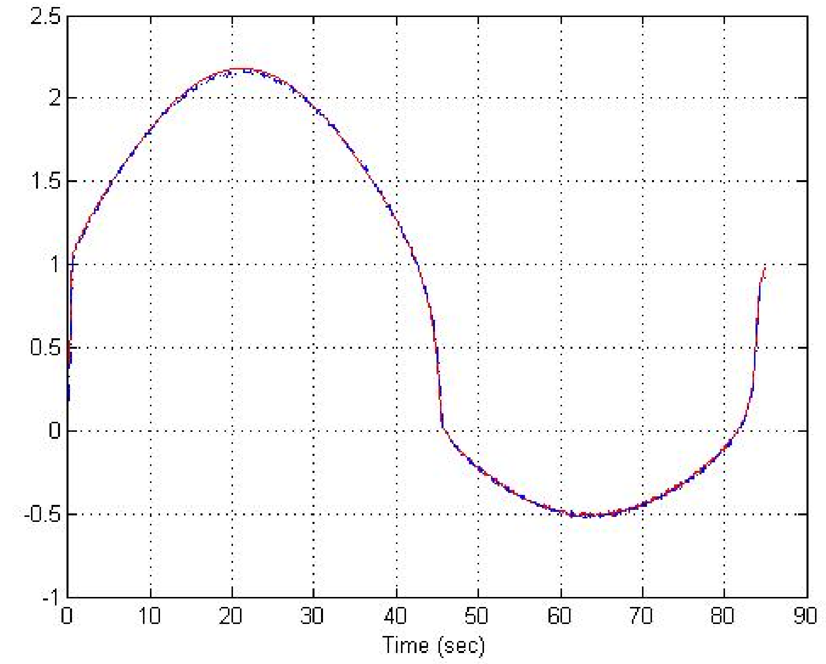


Figure 5.9: Outputs of the plant and the NN model (red line: filtered plant output, blue line: NN model output)

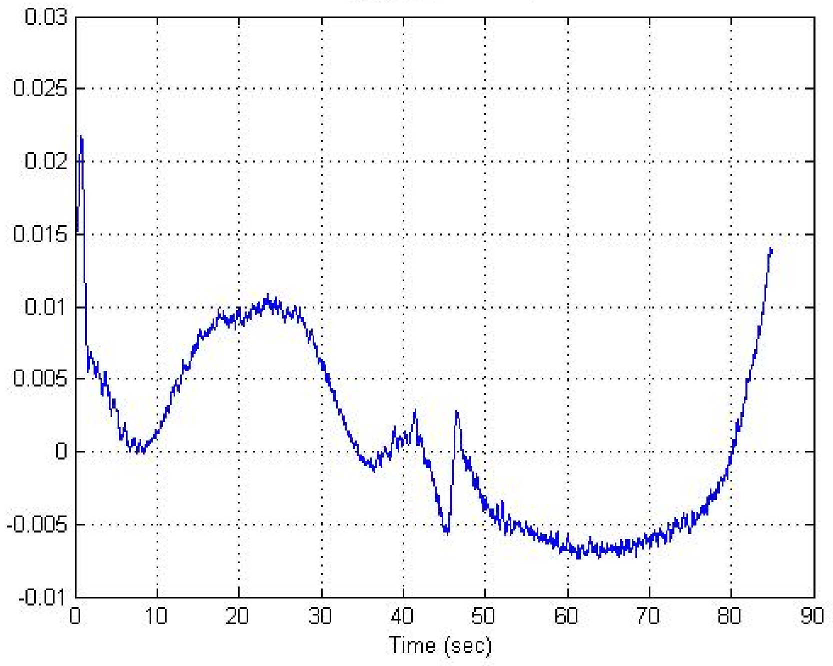


Figure 5.10: The error between the outputs of the plant and NN model

5.1.3 The Predictive Control for SISO Plant

With respect to the notations introduced in Chapter 4, the parameters of the GPC are chosen as follows: minimum cost horizon $N_0 = 1$, maximum cost horizon $N_1 = 4$, and control horizon $N_u = 4$. The initial condition of the plant is $(y(k-1), y(k-2)) = (0,0)$. In order to check the performance of the neural generalized predictive control algorithm, a sinusoidal signal is used as reference. The goal is to control plant defined in Equation (5.1) to track the reference defined in Equation (5.2). Both Quasi-Newton method and Nelder-Mead method are applied for solving the cost function minimization problem separately. The termination rules of Quasi-Newton method are chosen as,

1). The value of the objective cost function $J < 1 \times 10^{-6}$;

2). The maximum iteration $K > 500$;

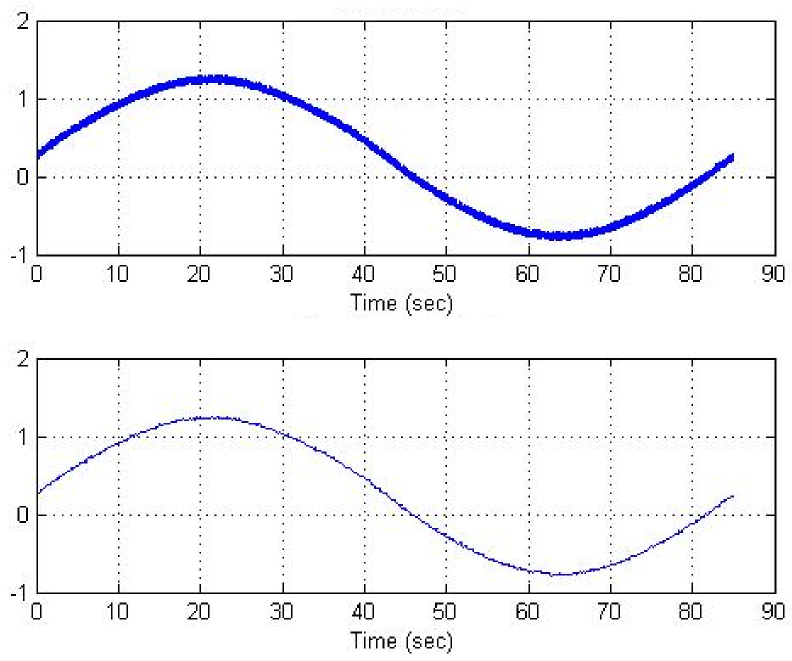
3). The change of gradients $||g_k - g_{k-1}|| < 1 \times 10^{-6}$.

The termination rules of Nelder-Mead method are chosen as,

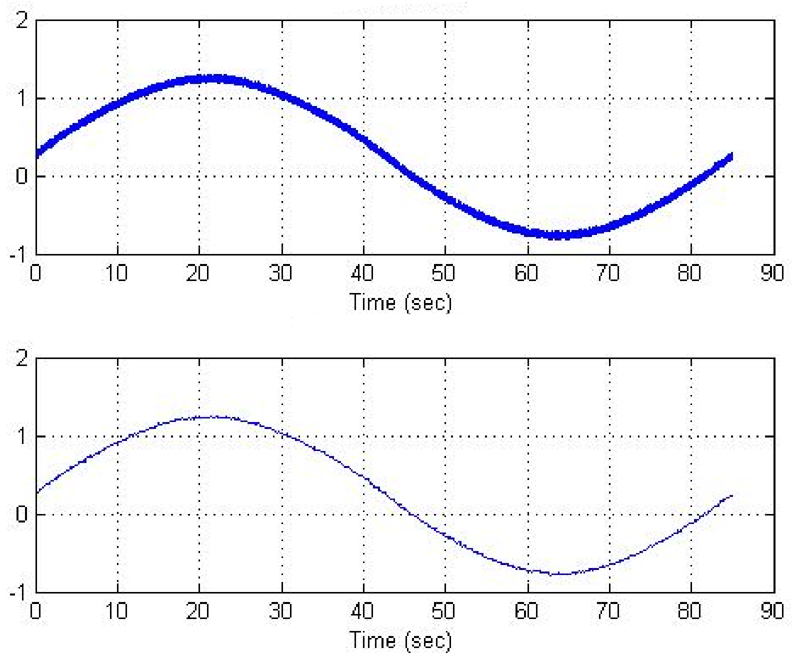
1). The value of the objective cost function $J < 1 \times 10^{-6}$;

2). The maximum iteration $K > 500$;

If the feedback of the plant has a uniform pseudo random noise with an amplitude in the interval $[-0.05, 0.05]$. The feedback needs to be filtered with a Butterworth filter. Fig. 5.11 shows the result when the cutoff frequency is 3 Hz. The control signal, the reference and feedback, the tracking error, and the computation time of both Quasi-Newton method and Nelder-Mead method are shown in Figs. from 5.12 to 5.15, respecttively.

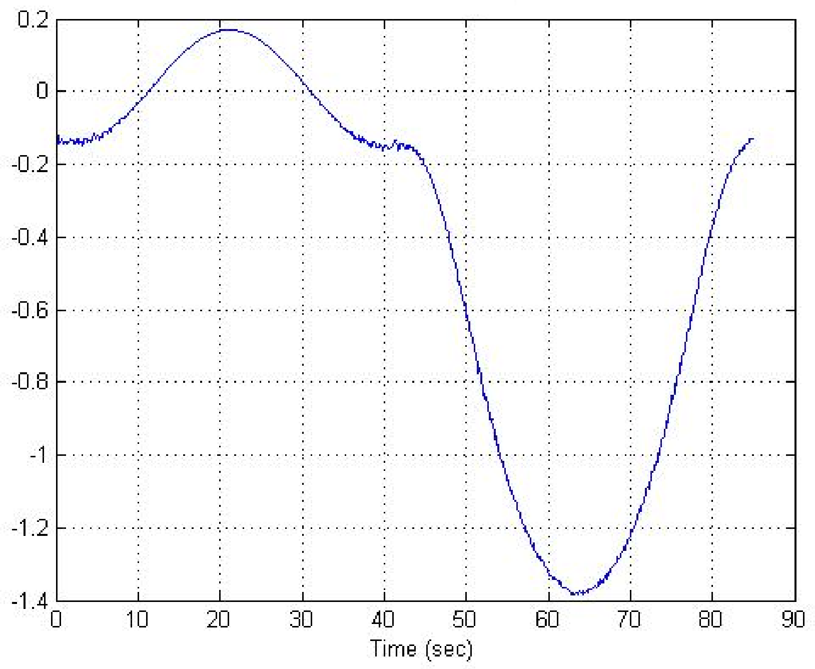


(a)

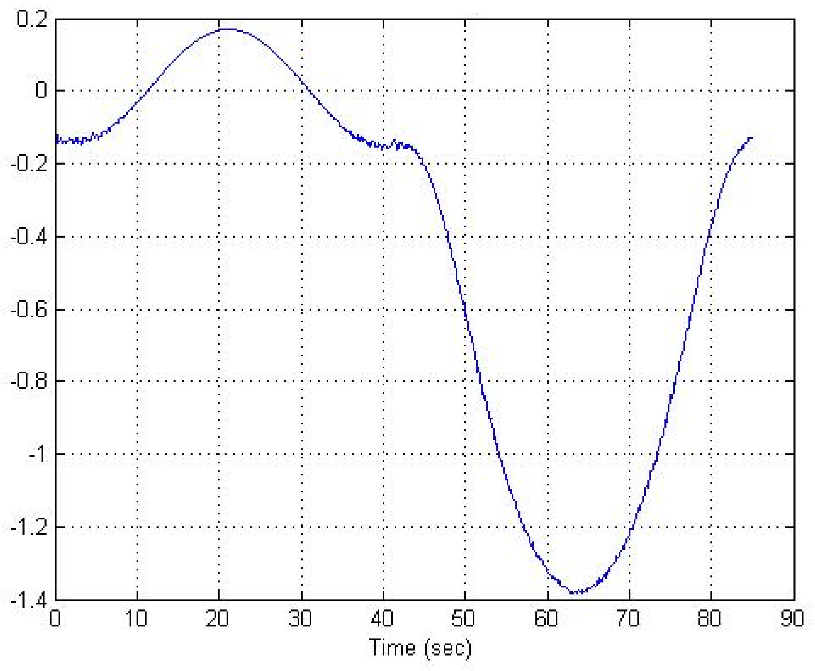


(b)

Figure 5.11: The feedback (top) and filtered feedback (bottom): (a) Quasi-Newton method; (b) Nelder-Mead method

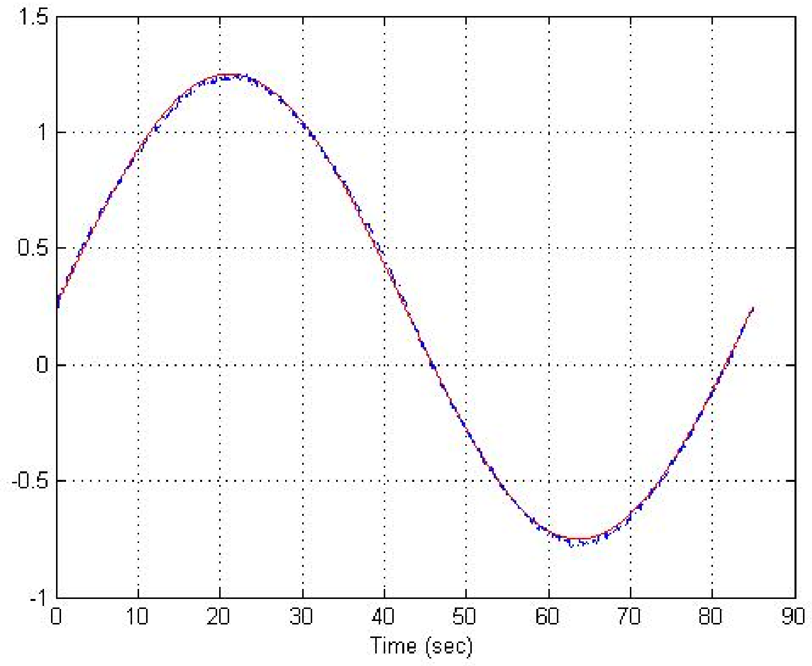


(a)

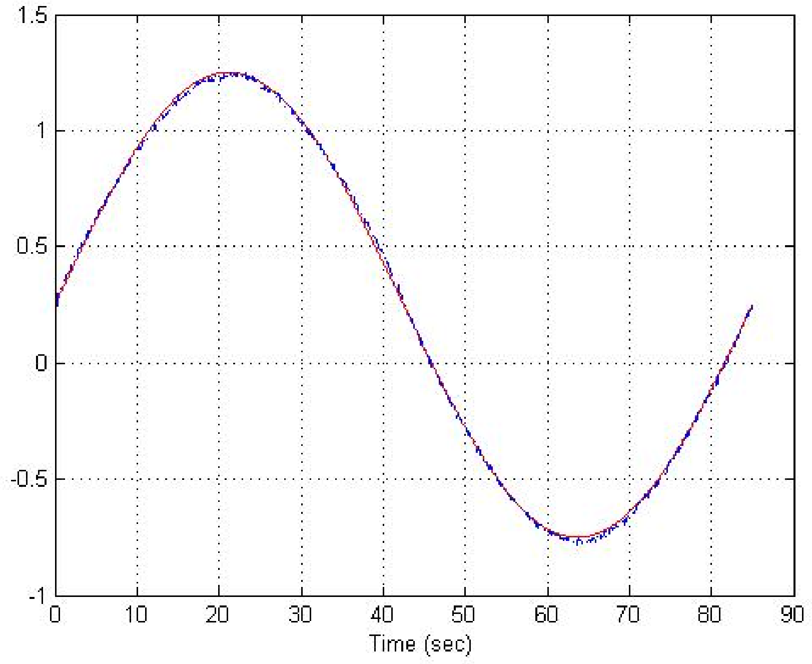


(b)

Figure 5.12: The control signal: (a) Quasi-Newton method; (b) Nelder-Mead method

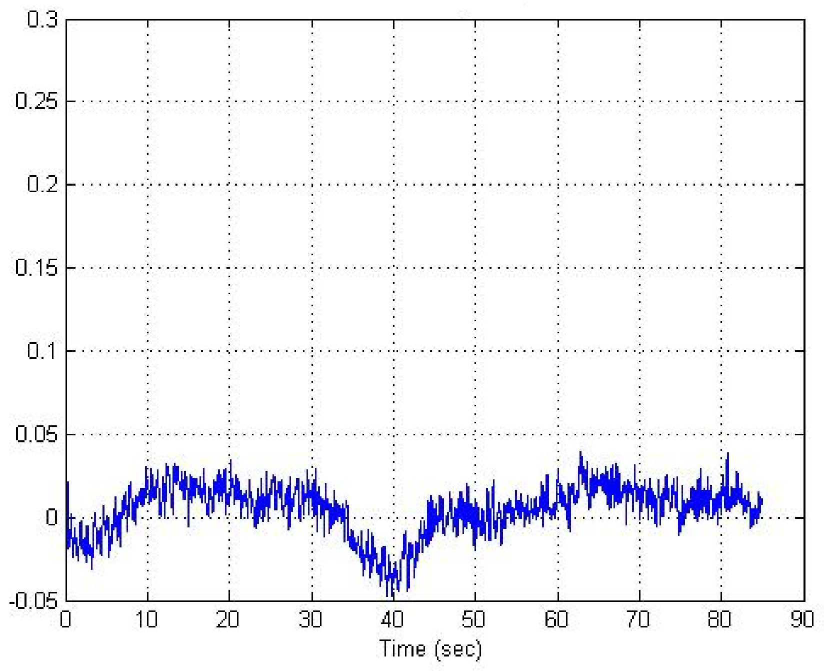


(a)

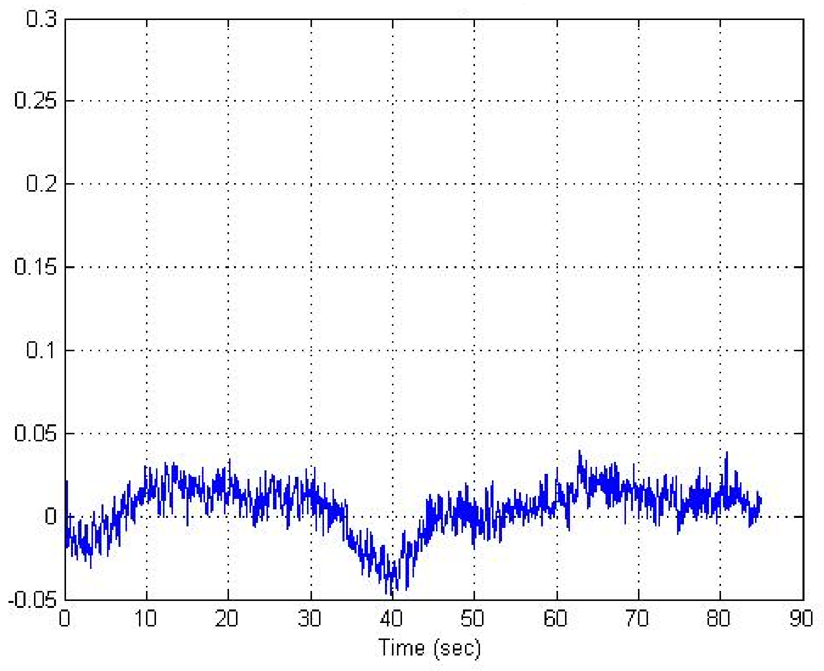


(b)

Figure 5.13: The tracking performance (red line: reference trajectory, blue line: filtered feedback): (a) Quasi-Newton method; (b) Nelder-Mead method

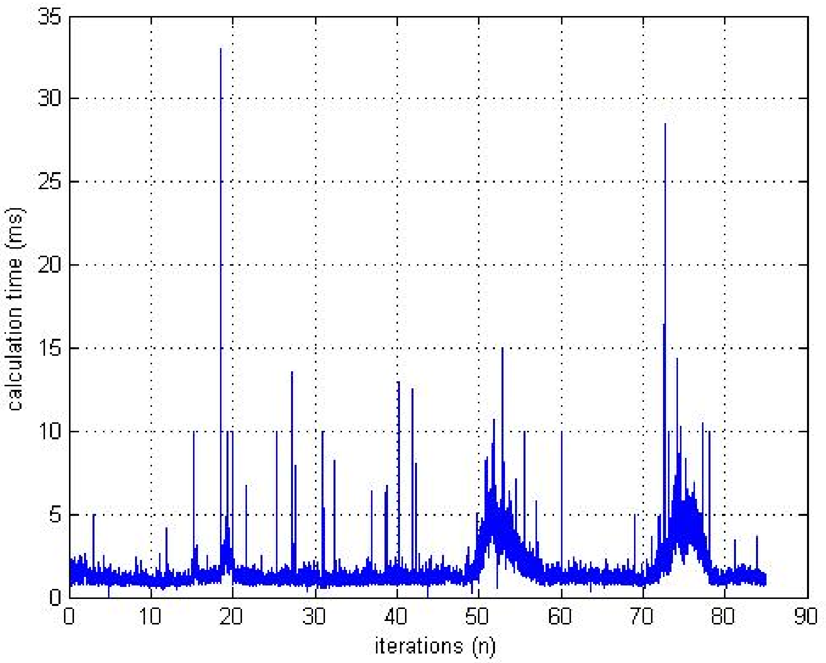


(a)

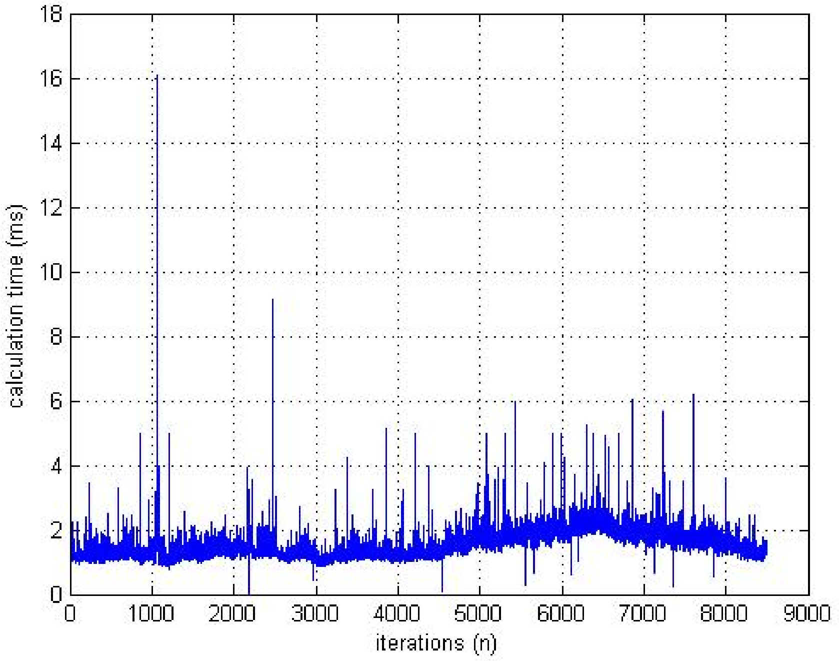


(b)

Figure 5.14: The tracking errors: (a) Quasi-Newton method; (b) Nelder-Mead method



(a)



(b)

Figure 5.15: The calculation time: (a) Quasi-Newton method; (b) Nelder-Mead method

The numerical comparison between the two algorithms are given in Table 5.1. It is seen from Table 5.1 that the SNRs of the feedback (SNRB) and the filtered feedback (SNRF) are 26.63 dB and 35.47 dB, which implies that most of the noise were filtered out and the filter worked well. Both optimization methods produce acceptable tracking performances. The average computing time (AvgT) and Max conputing time (MaxT) are shown as well.

More results are shown in Tables from 5.2 to 5.4 by using different cutoff frequencies under the same simulation conditions. If the cutoff frequency is too low, the filtered feedback will have a time-delay because of the lag of Butterworth filter. Meanwhile, if the cutoff frequency is too high, most noise components are not removed. The performance of the controller is poor if the cutoff requency is not well chosen.

Table 5.1: Control results for SISO benchmark plant with cutoff frequency 3 Hz

	rRMSE tracking	MAE tracking	AvgT (ms)	MaxT (ms)	SNRB (dB)	SNRF (dB)
QN	0.02253	0.01317	1.507	33.01	28.63	35.47
NM	0.02252	0.01316	1.548	16.10	28.63	35.47

Table 5.2: Control results for SISO plant with cutoff frequency 0.5 Hz

	rRMSE tracking	MAE tracking	AvgT (ms)	MaxT (ms)	SNRB (dB)	SNRF (dB)
QN	0.04530	0.02657	1.608	17.81	28.63	27.86
NM	0.04526	0.02656	1.529	16.53	28.63	27.85

Table 5.3: Control results for SISO plant with cutoff frequency 8 Hz

	rRMSE tracking	MAE tracking	AvgT (ms)	MaxT (ms)	SNRB (dB)	SNRF (dB)
QN	0.02372	0.01448	1.487	19.12	28.63	34.66
NM	0.02489	0.01473	1.542	14.76	28.63	34.51

Table 5.4: Control results for SISO plant with cutoff frequency 15 Hz

	rRMSE tracking	MAE tracking	AvgT (ms)	MaxT (ms)	SNRB (dB)	SNRF (dB)
QN	0.02739	0.01675	1.558	153.9	28.63	32.73
NM	0.02817	0.01687	1.633	27.72	28.63	32.65

Another set of simulations have been done by increasing the amplitude of uniform pseudo random noise with a second order Butterworth filter of 3 Hz. The numerical results are shown in Tables from 5.5 to 5.7. It is obvious that the performances of the controller are poorer if the noise amplitude is larger. These results also show that the control using Quasi-Newton method has a better tracking performance than the one using Nelder- Mead method. However, the quasi-Newton method takes more computing time than the Nelder-Mead method.

Table 5.5: Control results for plant noise amplitude between [-0.15, 0.15]

	rRMSE tracking	MAE tracking	AvgT (ms)	MaxT (ms)	SNRB (dB)	SNRF (dB)
QN	0.03988	0.02554	3.552	242.2	19.07	28.52
NM	0.04357	0.02658	1.557	13.14	19.07	27.97

Table 5.6: Control results for plant with noise amplitude between [-0.25, 0.25]

	rRMSE tracking	MAE tracking	AvgT (ms)	MaxT (ms)	SNRB (dB)	SNRF (dB)
QN	0.05906	0.03811	5.455	297.0	14.61	24.76
NM	0.06928	0.04250	4.565	34.92	14.61	24.52

Table 5.7: Control results for plant with noise amplitude between [-0.40, 0.40]

	rRMSE tracking	MAE tracking	AvgT (ms)	MaxT (ms)	SNRB (dB)	SNRF (dB)
QN	0.09051	0.05878	5.554	278.8	10.50	20.93
NM	0.1097	0.06739	4.828	41.76	10.50	19.27

5.2 HVAC System Simulation

In this part, the simulation is done by applying the designed controller to the non-linear MIMO HVAC system. The model of the HVAC system is shown in Fig. 5.16 [1]. It is assumed that the air in the room has a uniform temperature distribution. According to the energy conservation principle, the model proposed in [1] and [30] which includes temperature and humidity ratio is as follows:

$$\begin{aligned}
\dot{T}_3(t) &= \frac{60f_r}{V_s}\left(T_2(t)-T_3(t)\right)-\frac{60h_{fg}f_r}{c_pV_s}\left(W_s-W_3(t)\right)+ \\
&\qquad \frac{1}{\mu\rho_a c_p V_s}\left(Q_o-h_{fg}M_o\right)+e_1(t) \\
\dot{W}_3(t) &= \frac{60f_r}{V_s}\left(W_s-W_3(t)\right)+\frac{M_o}{\rho_a V_s}+e_2(t) \\
\dot{T}_2(t) &= \frac{60f_r}{V_{he}}\left(T_2(t)-T_3(t)\right)+\frac{60\mu f_r}{V_{he}}\left(T_0-T_3(t)\right)-\frac{60h_w f_r}{c_pV_{he}}\times \\
&\qquad \left(\mu W_0-(1-\mu)W_3(t)-W_s\right)-6000\frac{\text{gpm}}{\rho_a c_p V_{he}}+e_3(t)
\end{aligned} \tag{5.3}$$

The desired goal of this system is to control the temperature T_3 and humidity ratio W_3 in thermal space to the desired values. The control inputs of the system are the flow rate gpm of cold water from chiller to the heat exchanger and the air flow rate f_r using the variable speed fan. $e(t) = [e_1(t), e_2(t), e_3(t)]^T$ denotes the system noises. $Q_o$and W_0 are disturbances to the system. Table 5.8 gives the notations and parameter values in the system model [30] [34].

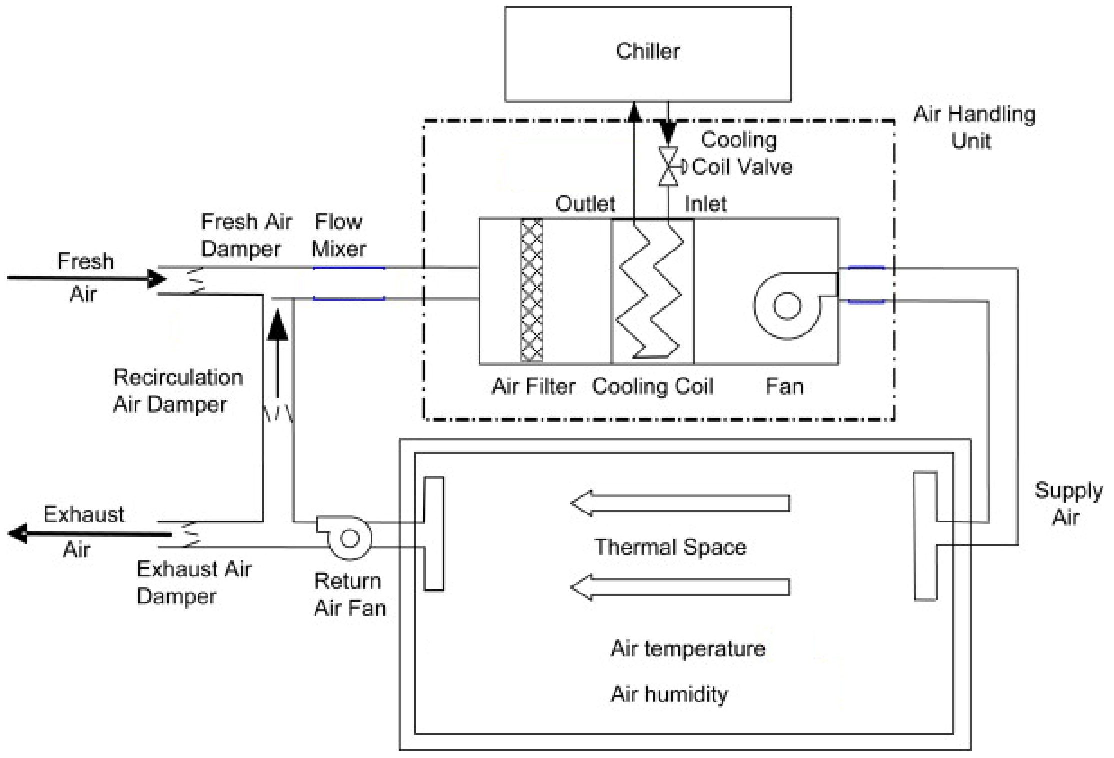


Figure 5.16: Model of the HVAC system

Table 5.8: HVAC system variables and parameters

Parameters	Parameter meanings and values
c_p	Specific heat of air 0.24 $(\text{btu/lb} \cdot {}^{\circ}\text{F})$
ρ_a	Air mass density 0.074 (lb/ft^3)
V_{he}	Volume of heat exchanger 60.75 (ft^3)
V_s	Volume of thermal space 58464 (ft^3)
W_0	Humidity ratio of outdoor air 0.018 (lb/lb)
W_s	Humidity ratio of supply air 0.0070 (lb/lb)
W_3	Humidity ratio of thermal space (lb/lb)
T_0	Temperature of outdoor air 85 $({}^{\circ}\text{F})$
T_2	Temperature of supply air $({}^{\circ}\text{F})$
T_3	Temperature of thermal space $({}^{\circ}\text{F})$
M_o	Moisture load 166.06 (lb/hr)
Q_o	Sensible heat load 289897.52 (btu/hr)
h_w	Enthalpy of liquid water 340 (btu/lb)
h_{fg}	Enthalpy of water vapor 1078.25 (btu/lb)
f_r	Volumetric flow rate of air $(\text{cfm} = \text{ft}^3/\text{min})$
gpm	Flow rate of chilled water (gal/min)
μ	Ventilation rate of air in system (25%)

Redefine the variables and parameters as follows:

$$u_1 = f_r, u_2 = \text{gpm}, x_1 = T_3, x_2 = W_3, x_3 = T_2$$

$$\alpha_1 = \frac{60}{V_s}, \alpha_2 = \frac{60 f_r}{c_p V_s}, \alpha_3 = \frac{1}{\mu \rho_a c_p V_s}, \alpha_4 = \frac{1}{\rho_a V_s}$$

$$\beta_1 = \frac{60}{V_{he}}, \beta_2 = \frac{1}{\rho_a c_p V_{he}}, \beta_3 = \frac{60 h_w}{c_p V_{he}}$$

with $u(t) = [u_1(t),\ u_2(t)]^T$ being the control signal applied to the plant, $y(t) = [x_1(t),\ x_2(t)]^T$ being the output of the plant. The 2-input, 2-output system can be described as

$$\begin{aligned} \dot{x}_1(t) &= [\alpha_1(x_3(t) - x_1(t)) - \alpha_2(W_s - x_2(t))]u_1(t) + \alpha_3\left(Q_o - h_{fg} M_o\right) + e_1(t) \\ \dot{x}_2(t) &= \alpha_1(W_s - x_2(t))u_1(t) + \alpha_4 M_o + e_2(t) \\ \dot{x}_3(t) &= [\beta_1(x_1(t) - x_3(t)) + \beta_1 \mu(T_0 - x_1(t))]u_1(t) - u_1(t) \times \\ &\quad [\beta_3(\mu W_0 - (1-\mu)x_2(t) - W_s)] - 6000\beta_2 u_2(t) + e_3(t) \end{aligned} \tag{5.4}$$

The objective is to apply the designed controller to control the HVAC system so that the output $y(t)$ tracks the desired temperature and humidity ratio. The simulation conditions are given in Table 5.9.

Table 5.9: Simulation conditions of HVAC plant

Parameters	Values
Neurons of network	p=12, h=5, L=2
Plant delay	n=3, m=2, d=0
Parameters of controller	$N_0 = 1$, $N_1 = 4$, $N_u = 4$
Sampling time	0.01 hr

5.2.1 The NN Model Training for HVAC System

The RBF-NN is used to estimate the HVAC system given by Equation (5.4). Since the order of magnitude of values in HVAC system are not the same, scaling factors are used to normalize the system inputs and outputs within the interval $[-1.0,1.0]$. Fig. 5.17 shows the normalized input signal applied to the neural network for offline training. The corresponding normalized filtered response, which is shown in Fig. 5.18, will be used to train the neural network. The SNR of the HVAC system responses are 25.53 dB and 23.43 dB, and the SNR of filtered responses are 36.16 dB and 34.07 dB when the cutoff frequency is 0.0022 Hz.

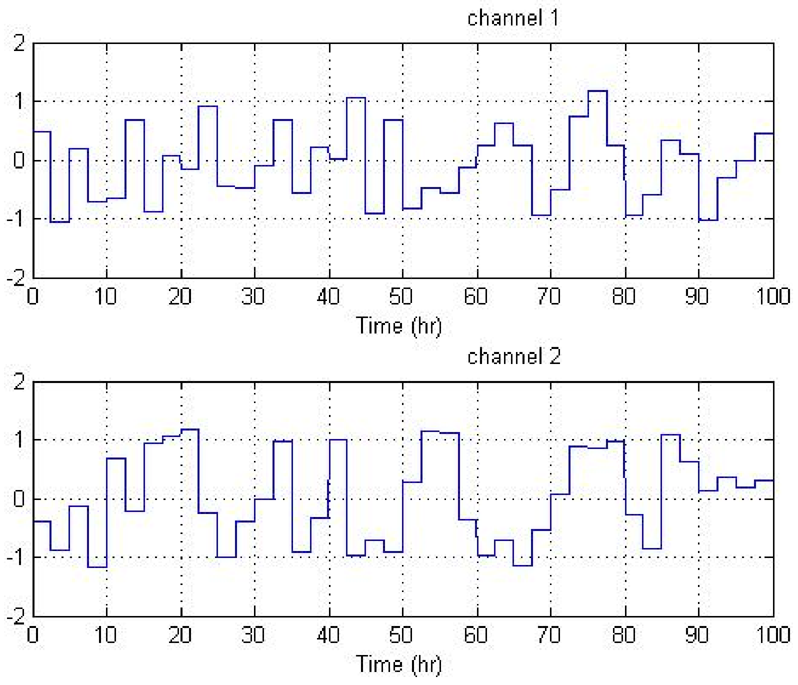
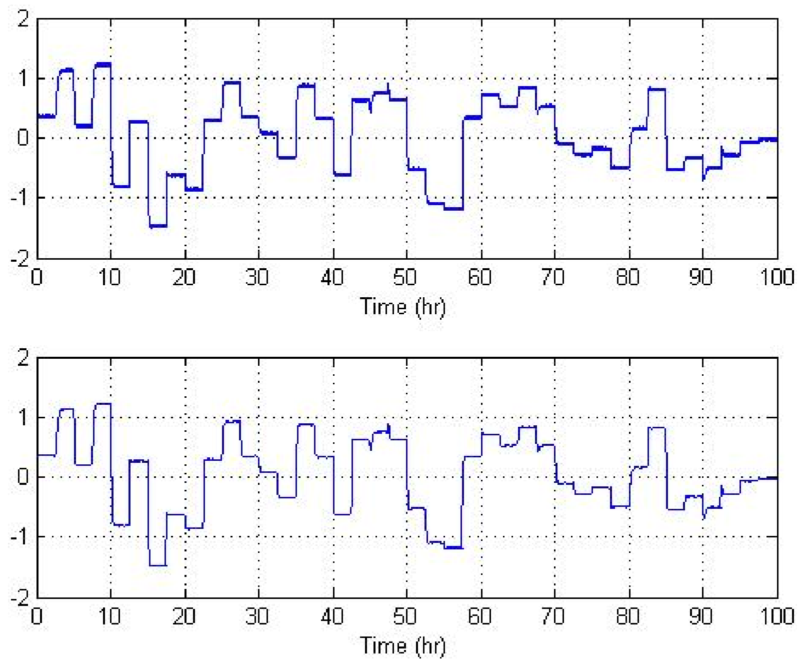
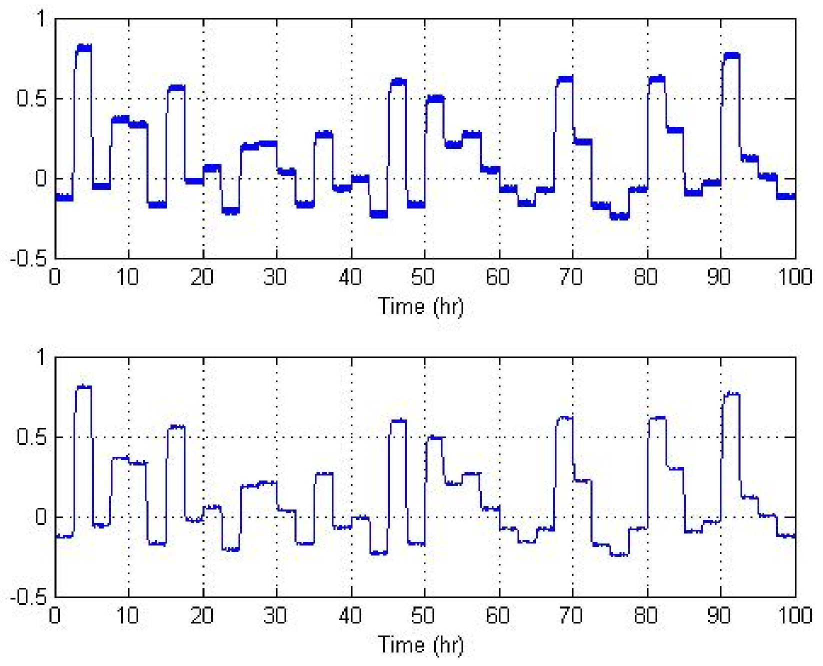


Figure 5.17: The input signals for neural network training: (a) Channel 1; (b) Channel 2



(a)



(b)

Figure 5.18: The HVAC system responses (top) and filtered responses (bottom) for training: (a) Channel 1; (b) Channel 2

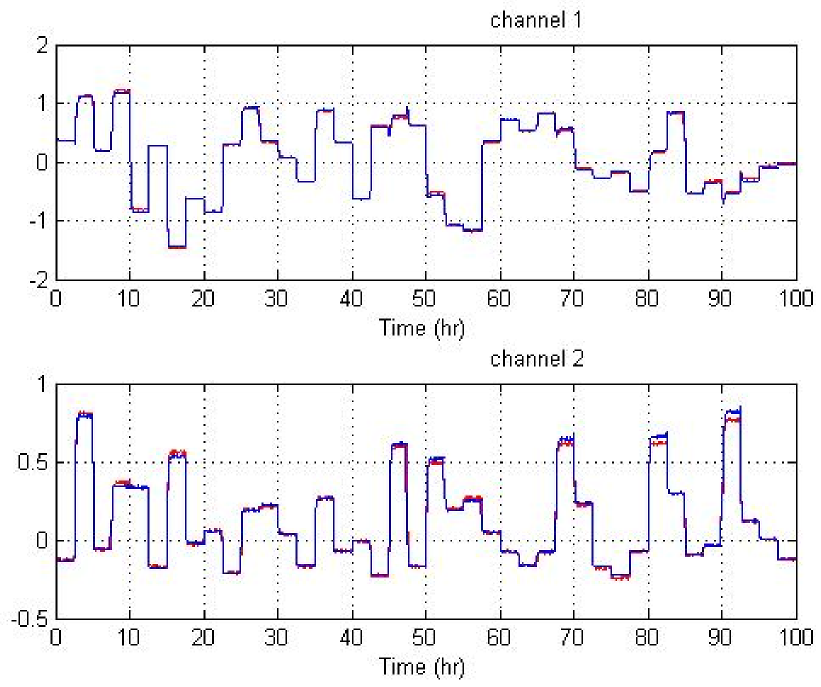
The modified gradient method defined in Equation (2.6) is adapted to minimize the objective function defined in Equation (3.8). The actual step size $\kappa = 0.005$ is used. The termination rules are chosen as,

1). The value of the objective function $J(Z) < 1$;

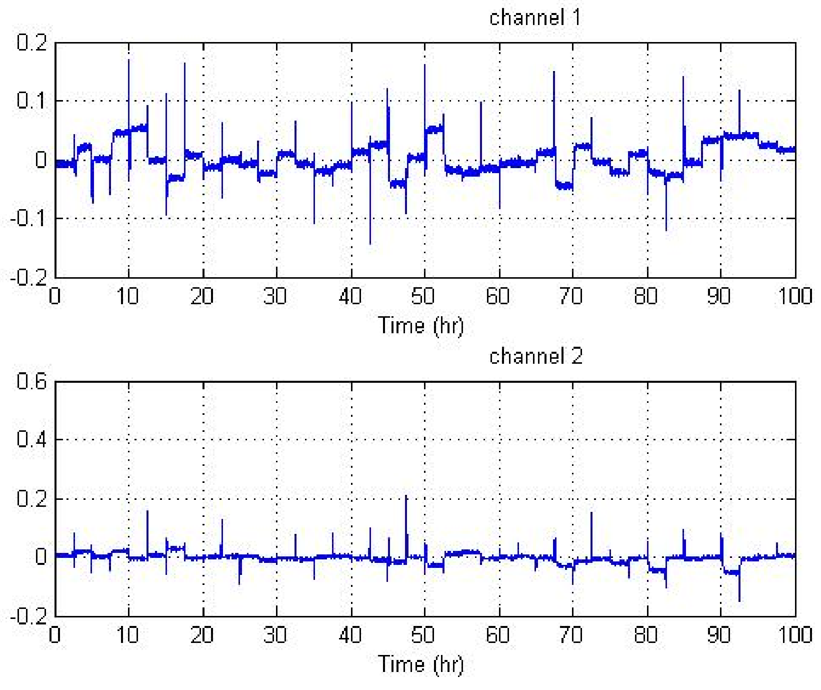
2). The learning iteration $K > 10000$;

3). The changing of gradients $||g_k - g_{k-1}|| < 1 \times 10^{-8}$.

The training results after 10000 iterations are provided. Fig. 5.19 (a) displays the outputs of the NN model and HVAC system. Fig. 5.19 (b) shows the error between the outputs of the HVAC and the NN model. The responses of the neural network parameters are plotted in Figs. from 5.20 to 5.22. It can be seen that the parameters of the neural network change fast during the first 1000 iterations, which means that the values of cost function is large and the norms of the gradients are large. After the first 1000 iterations, the parameters slowly converge to the vicinity of some values, which can be considered as the near-optimal parameters.



(a)



(b)

Figure 5.19: The training results for HVAC system: (a) The responses (red line: filtered system response, blue line: NN model response); (b) The training error

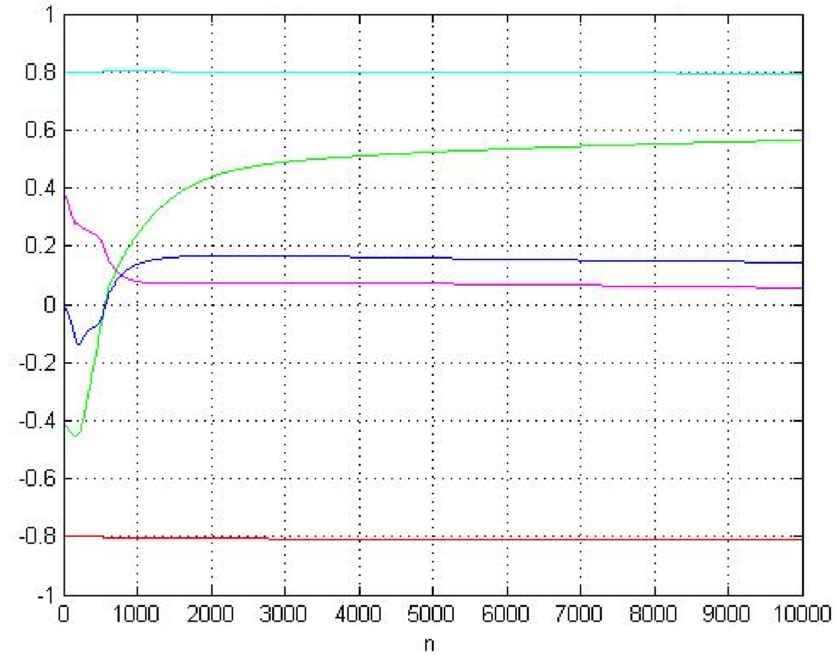


Figure 5.20: The centers of the first neuron in the input layer for HVAC training

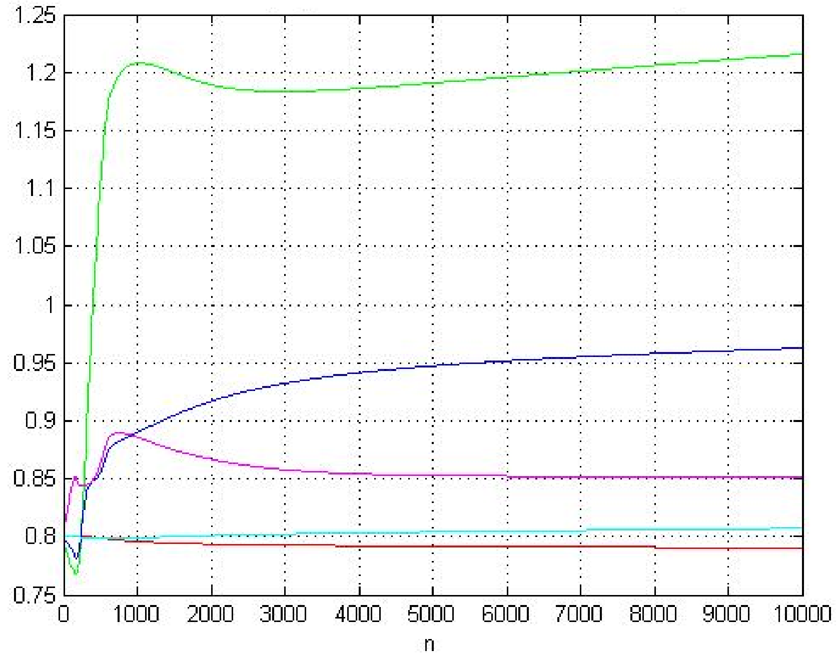


Figure 5.21: The widths of the first neuron in the input layer for HVAC training

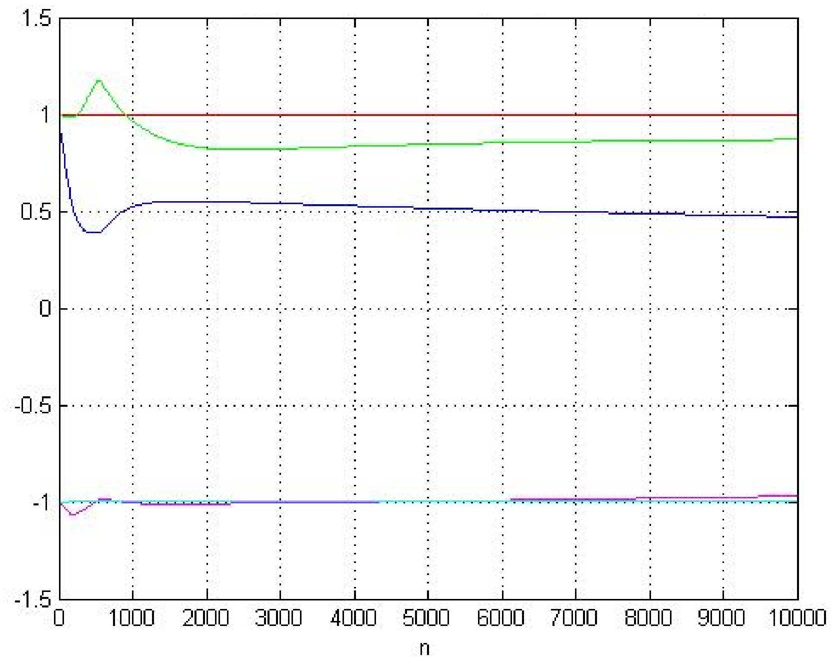
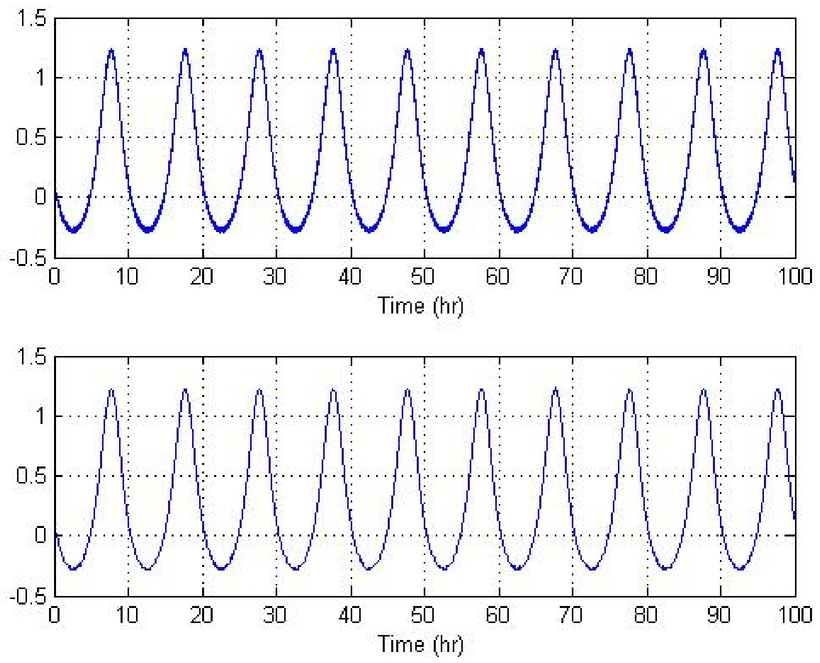


Figure 5.22: The weights of the first neuron in output layer for HVAC training

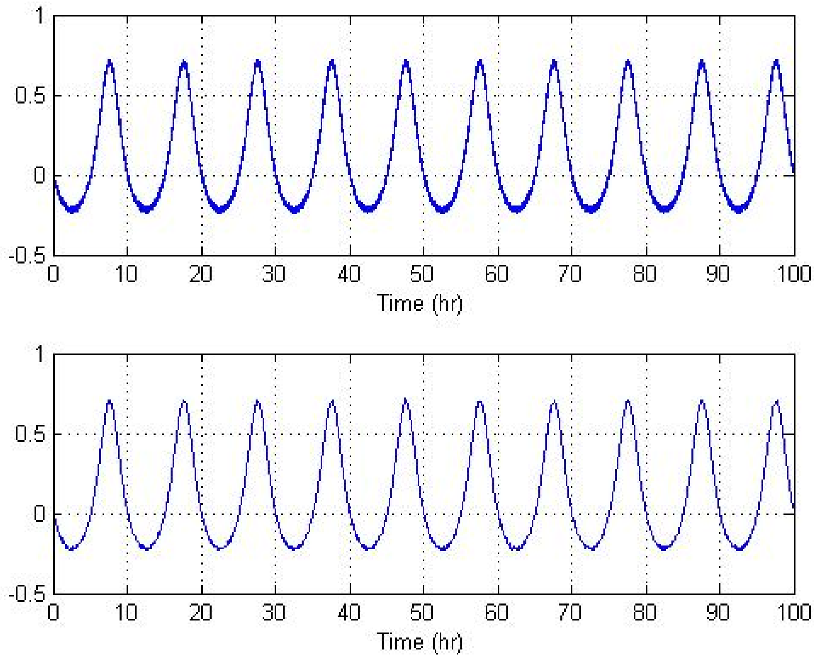
5.2.2 The NN Model Validation for HVAC System

The NN model is validated using a sinusoidal input signal defined in Equation (5.2) for this plant, with the amplitude $A = 1$, the sampling time $T = 0.01$ hr, the frequency $f = 1/36000$ Hz, and the bias *D*=0. The HVAC system response is filtered by a second-order Butterworth low pass filter with cutoff frequency 0.0022 Hz. The SNR of filtered system response for two output channels are 31.92 dB and 30.89 dB.

The HVAC system response, the NN model response and the validation error are shown in Fig. 5.23 and Fig. 5.24, respectively. The numerical results are given in Table 5.10 which shows that the RBF neural network model is a very good approximation of the HVAC system.

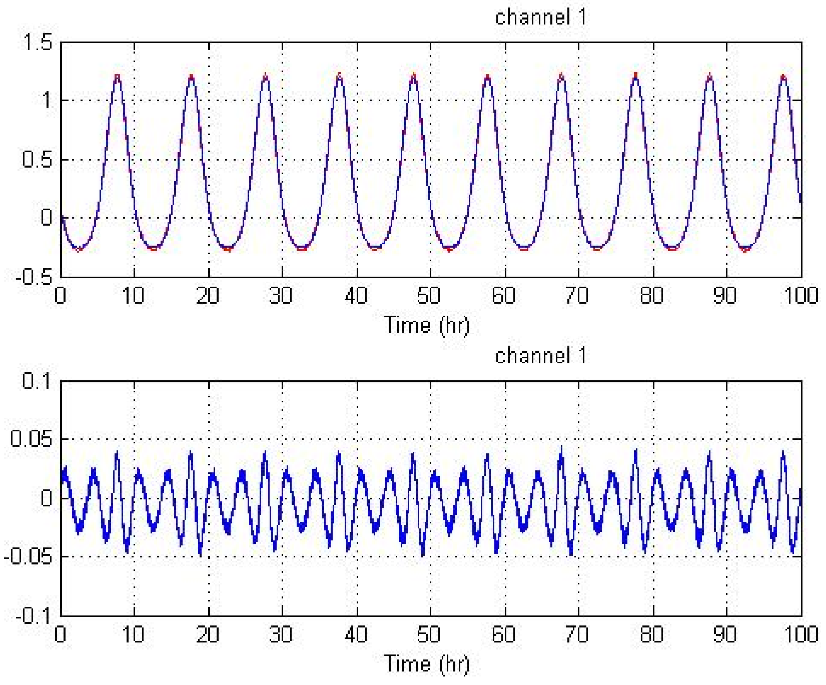


(a)

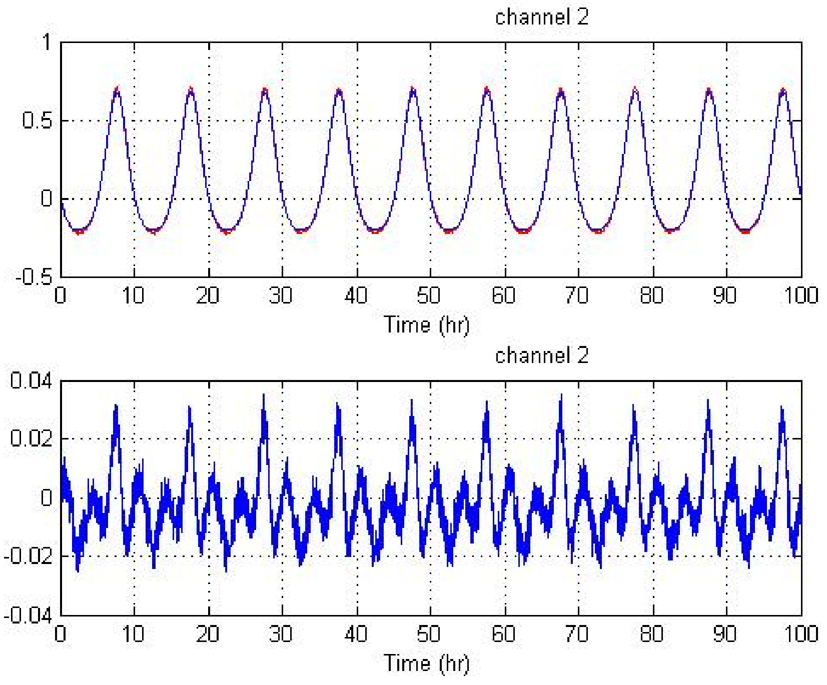


(b)

Figure 5.23: The HVAC system response (top) and filtered response (bottom) for validation: (a) Channel 1; (b) Channel 2



(a)



(b)

Figure 5.24: The NN model response (top) and error (bottom) for validation (red line: filtered system response, blue line: NN model response): (a) Channel 1; (b) Channel 2

Table 5.10: The results of neural network model for validation signal

	rRMSE	MAE	Filtered feedback SNR (dB)	Feedback SNR (dB)
Channel 1	0.01896	0.01196	31.92	29.16
Channel 2	0.02756	0.01383	30.89	26.92

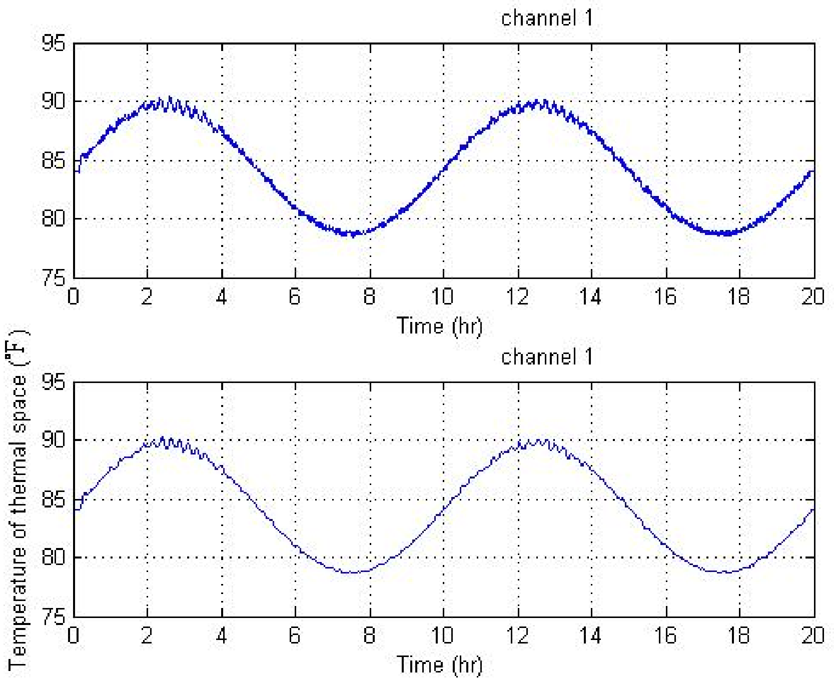
5.2.3 The Predictive Control for HVAC System

The objective is to track the desired temperature and humidity ratio in thermal space. Suppose that the designed thermal loads are constant, and all the parameter values are given as in Table 5.8. The initial values of the state variables are set to $T_3 = 84,\ T_2 = 55,\ W_3 = 0.0092$.

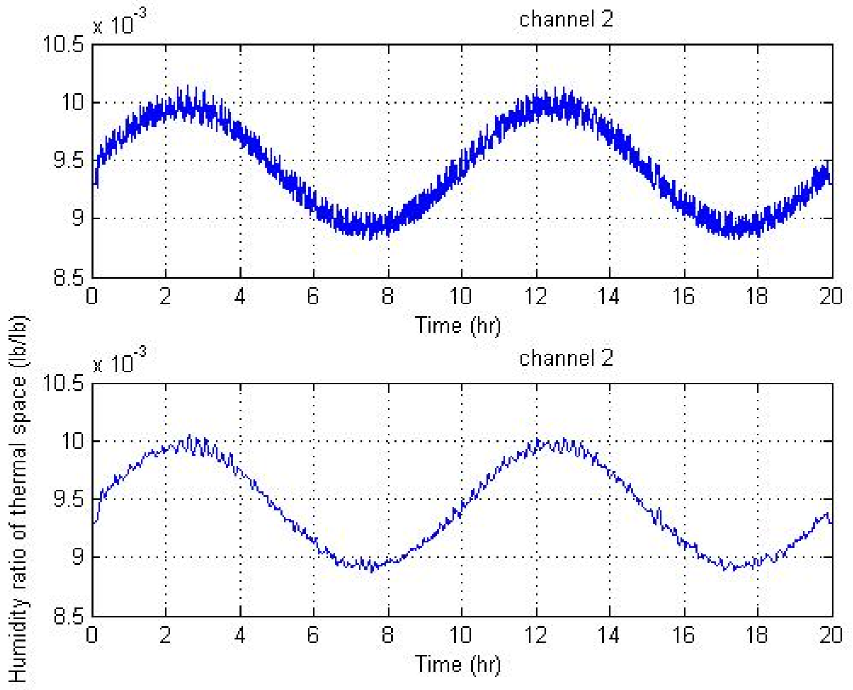
Quasi-Newton method and Nelder-Mead method are applied for solving the cost function minimization problem separately. The termination rules are chosen as same as the rules for SISO benchmark plant.

A normalized uniform pseudo random noise with amplitude in the interval $[-0.02,0.02]$ for the first output and $[-0.08,0.08]$ for the second output is added to the feedback of the plant. The feedback needs to be filtered by a Butterworth filter before applied to the controller. Fig. 5.25 and Fig. 5.26 show the results when the cutoff frequency is 0.0022 Hz for Quasi-Newton method and Nelder-Mead method. The control signal, the tracking performance, the tracking error, and the computation time of both Quasi-Newton method and Nelder-Mead method are shown in Figs. from 5.27 to

5.30, respectively.

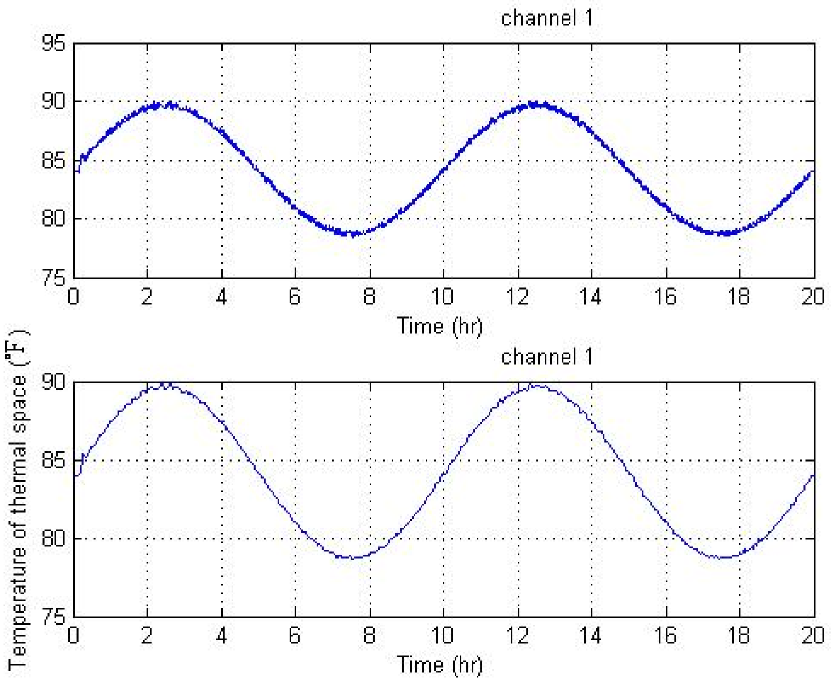


(a)

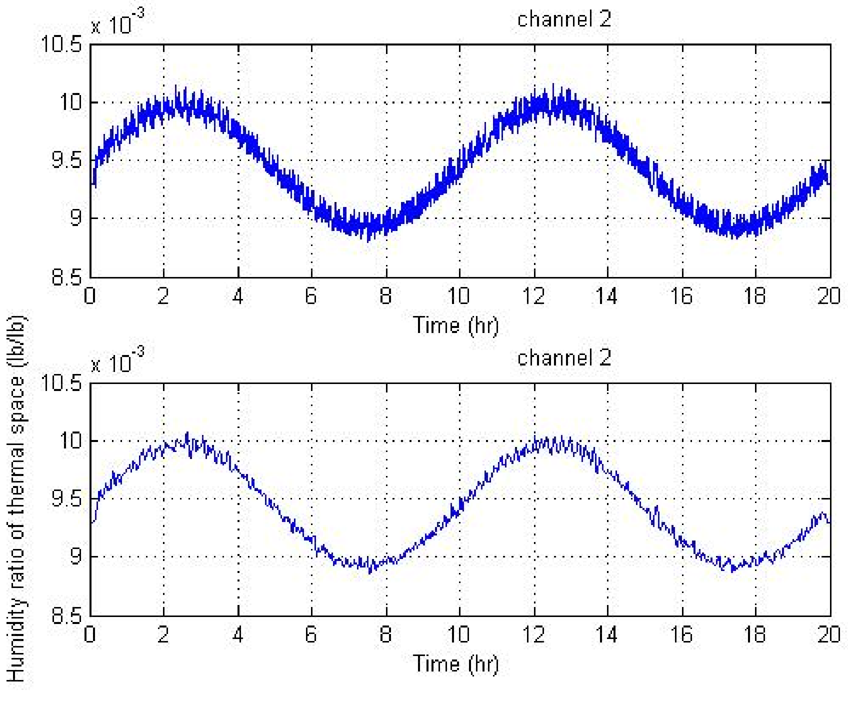


(b)

Figure 5.25: The HVAC system feedback (top) and the filtered feedback (bottom) of Quasi-Newton method: (a) The temperature of thermal space; (b) The humidity ratio

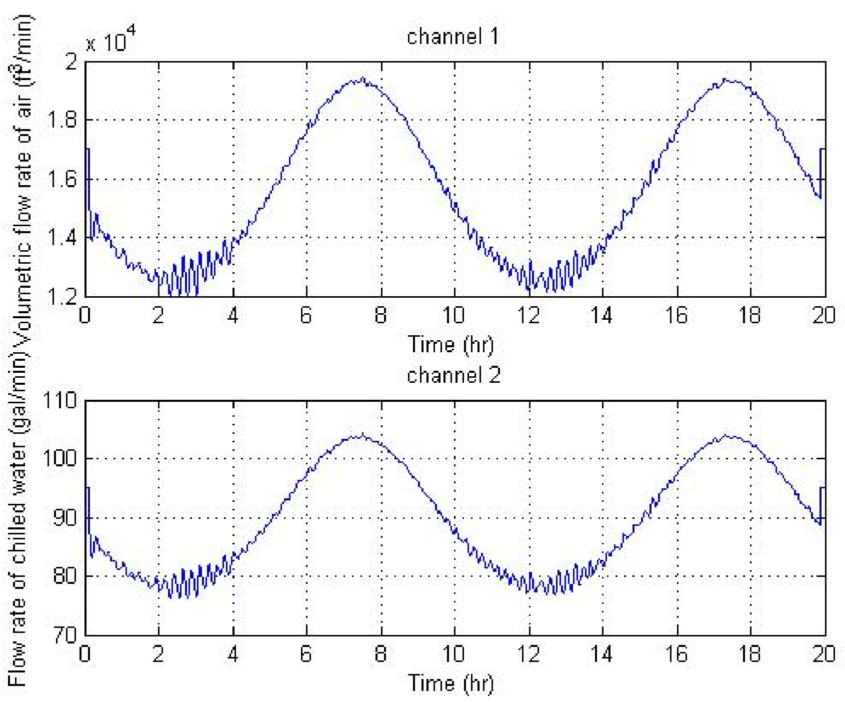


(a)

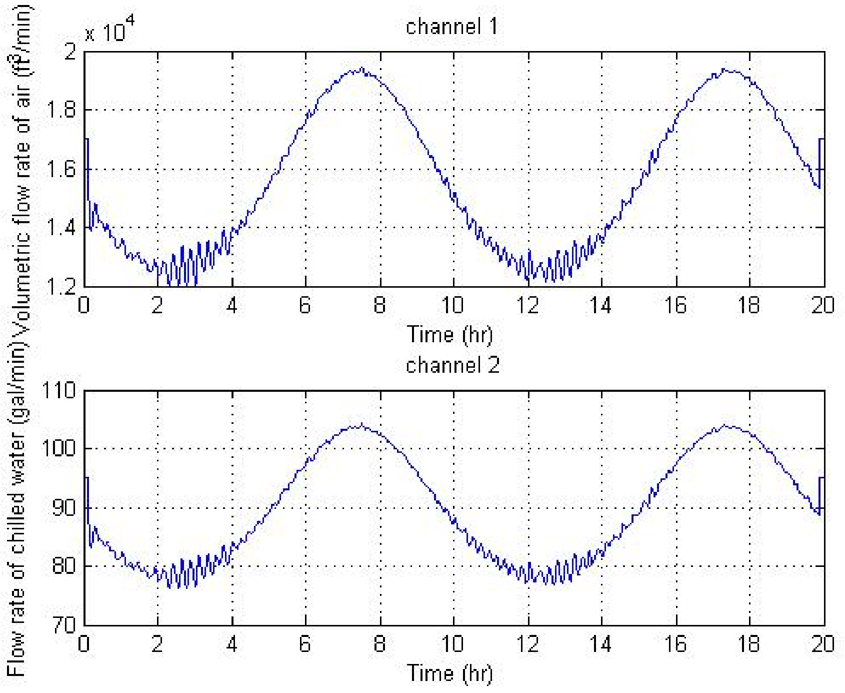


(b)

Figure 5.26: The HVAC system feedback (top) and the filtered feedback (bottom) of Nelder-Mead method: (a) The temperature of thermal space; (b) The humidity ratio

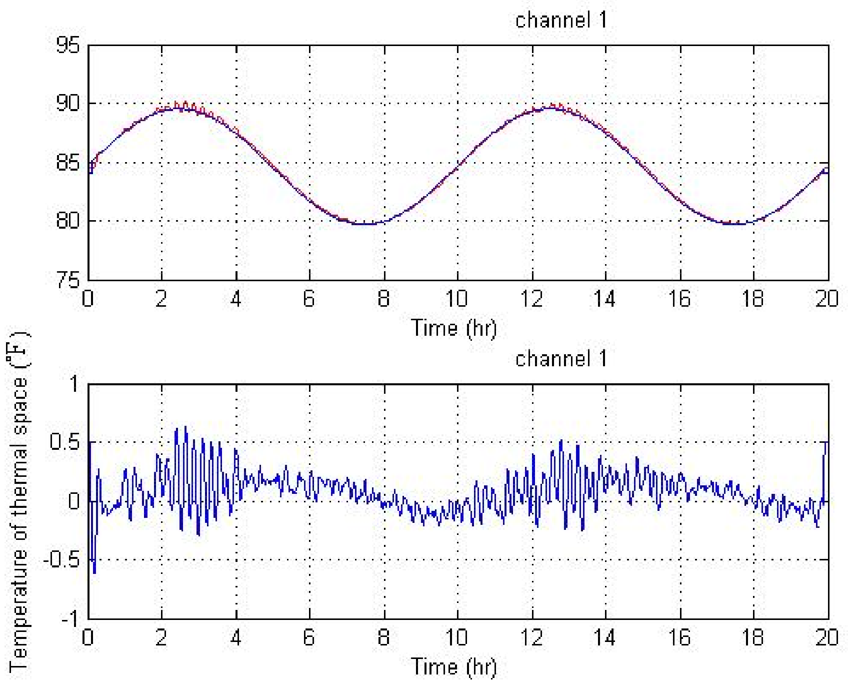


(a)

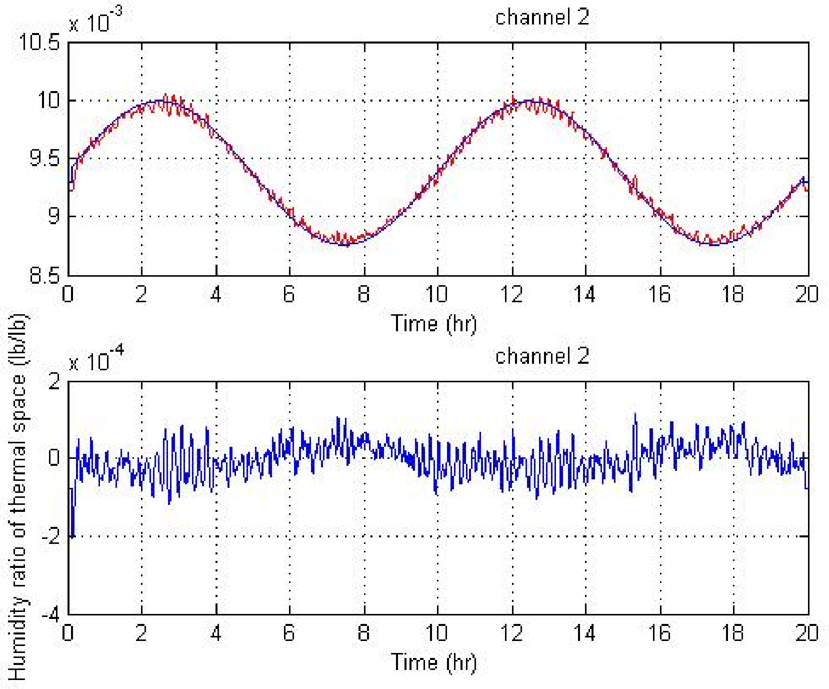


(b)

Figure 5.27: The control signal of HVAC system: (a) using Quasi-Newton method; (b) Nelder-Mead method

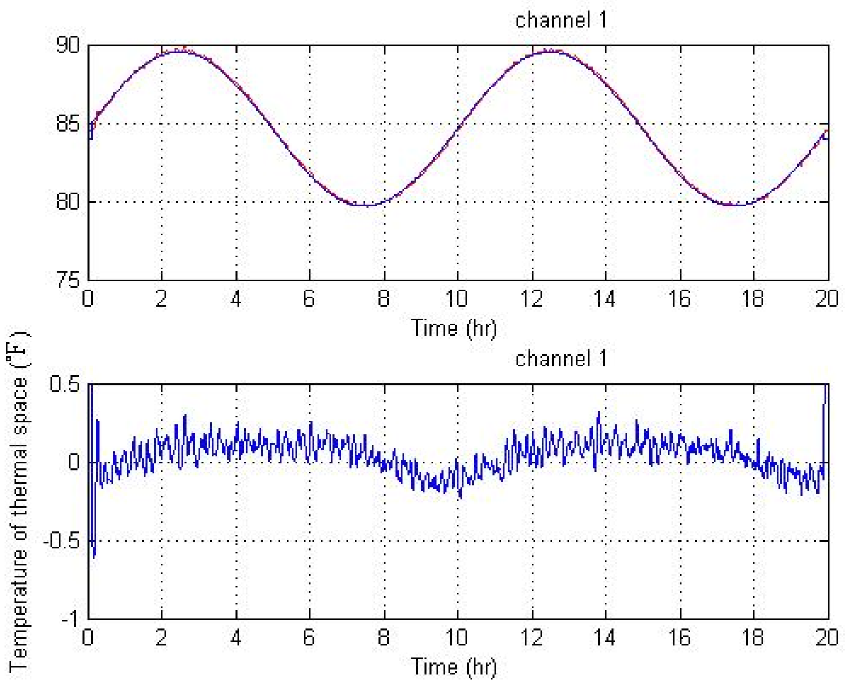


(a)

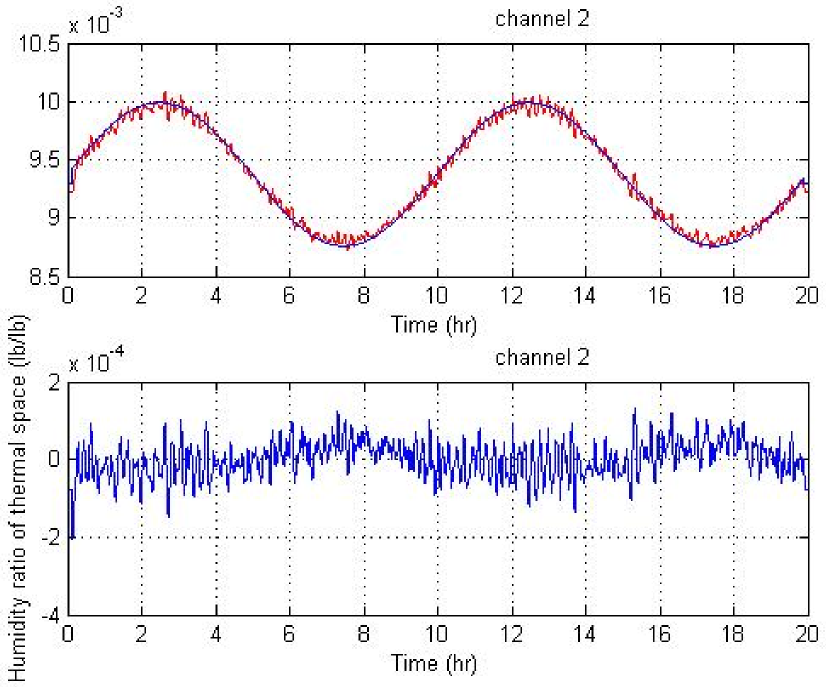


(b)

Figure 5.28: The tracking performance (top) and tracking error (bottom) of HVAC system for Quasi-Newton method (red line: filtered feedback, blue line: reference trajectory): (a) The temperature of thermal space; (b) The humidity ratio

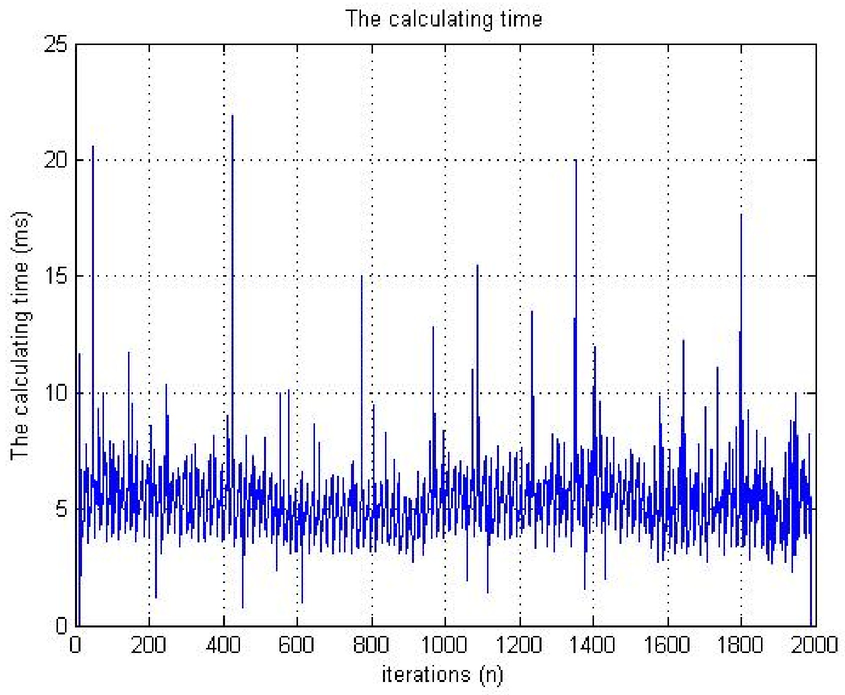


(a)

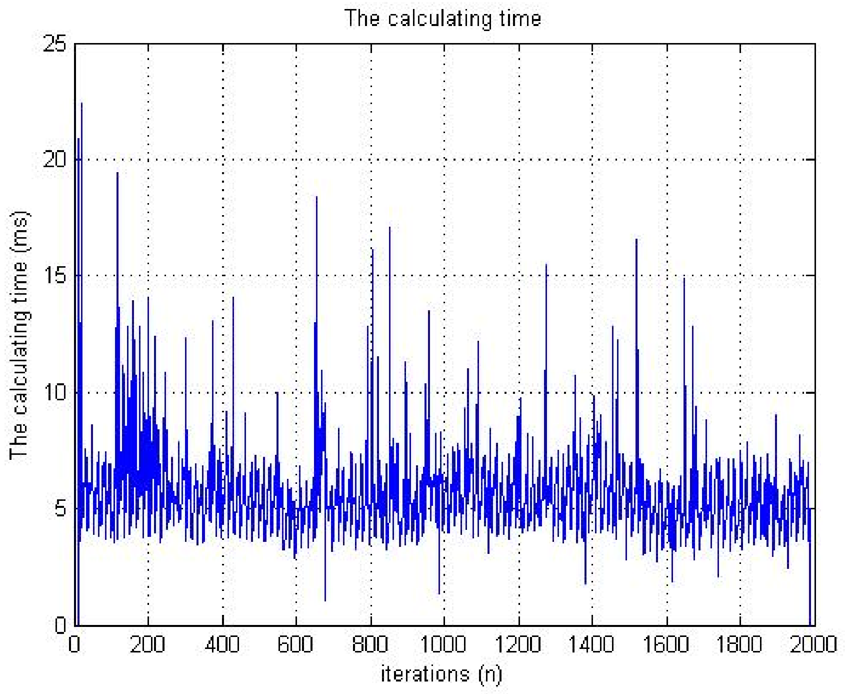


(b)

Figure 5.29: The tracking performance (top) and tracking error (bottom) of HVAC system for Nelder-Mead method (red line: filtered feedback, blue line: reference trajectory): (a) The temperature of thermal space; (b) The humidity ratio



(a)



(b)

Figure 5.30: The calculating time of predictive control for HVAC system: (a) Quasi-Newton Method; (b) Nelder-Mead Method

The comparsion between the two algorithms are given in Table 5.11. Like previous work on SISO benchmark plant, two different methods were applied to solve the cost function minimization problem. The results show that both methods can solve the problem in a short time periord. The system outputs are able to track the desired reference trajectories with very small tracking error.

Table 5.11: The control results for HVAC system with normalized noise amplitude between [-0.02, 0.02] for the first output and [-0.08, 0.08] for the second output

	CH	rRMSE tracking	MAE tracking	SNRB (dB)	SNRF (dB)	AvgT (ms)	MaxT (ms)
QN	1	0.001702	0.1029 °F	29.26	30.54	5.978	81.96
	2	0.004348	0.000033 lb/lb	15.53	20.72		
NM	1	0.001689	0.1023 °F	29.31	30.54	5.552	69.79
	2	0.004864	0.000036 lb/lb	15.61	20.24		

Both Quasi-Newton method and Nelder-Mead method have almost the same control results for the HVAC system with small noise. Further simulations are done by changing the amplitude of random noise. The other conditions for simulations stay the same. The numerical results are given in Tables from 5.12 to 5.15. The HVAC system cannot be stabilized if the amplitude of normalized noise is larger than 0.5. By using Quasi-

Newton method with normalized noise amplitude between [-0.45, 0.45], the feedback of the system, tracking performance, and control signal are shown in Figs. from 5.31 to 5.33.

Table 5.12: The control results for HVAC system with normalized noise amplitude between [-0.05, 0.05]

	CH	rRMSE tracking	MAE tracking	SNRB (dB)	SNRF (dB)	AvgT (ms)	MaxT (ms)
QN	1	0.001881	0.1255 °F	21.33	26.41	3.394	31.72
	2	0.003504	0.000026 lb/lb	19.60	24.67		
NM	1	0.002092	0.144 °F	21. 10	25.73	3.733	41.18
	2	0.003259	0.000025 lb/lb	20.91	24.86		

Table 5.13: The control results for HVAC system with normalized noise amplitude between [-0.15, 0.15]

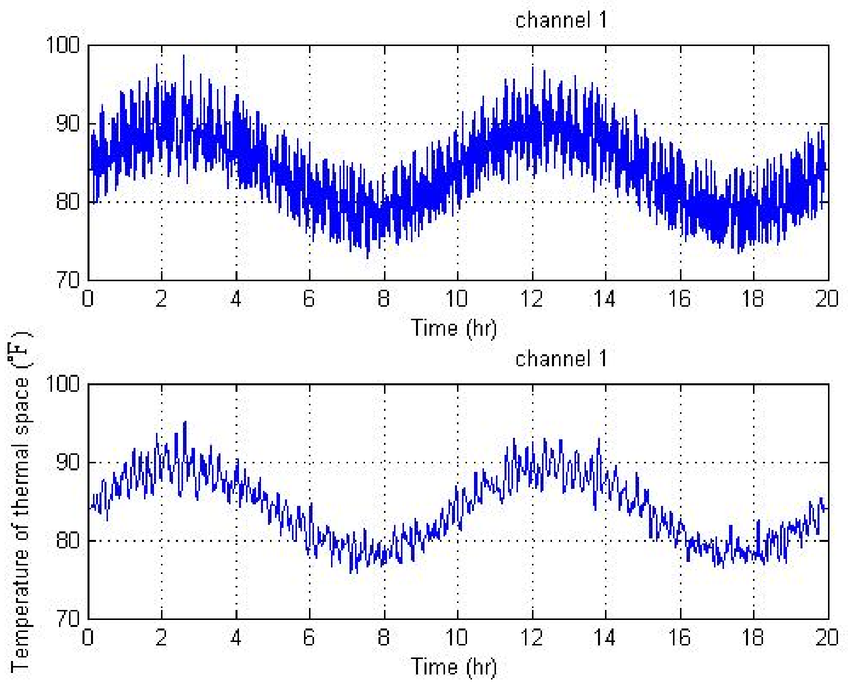
	CH	rRMSE tracking	MAE tracking	SNRB (dB)	SNRF (dB)	AvgT (ms)	MaxT (ms)
QN	1	0.005425	0.3654 °F	11.89	16.96	3.479	505.1
	2	0.007734	0.000057 lb/lb	10.25	15.30		
NM	1	0.005963	0.4162 °F	11.88	16.42	3.872	523.6
	2	0.008805	0.000067 lb/lb	11.70	15.53		

Table 5.14: The control results for HVAC system with normalized noise amplitude between [-0.25, 0.25]

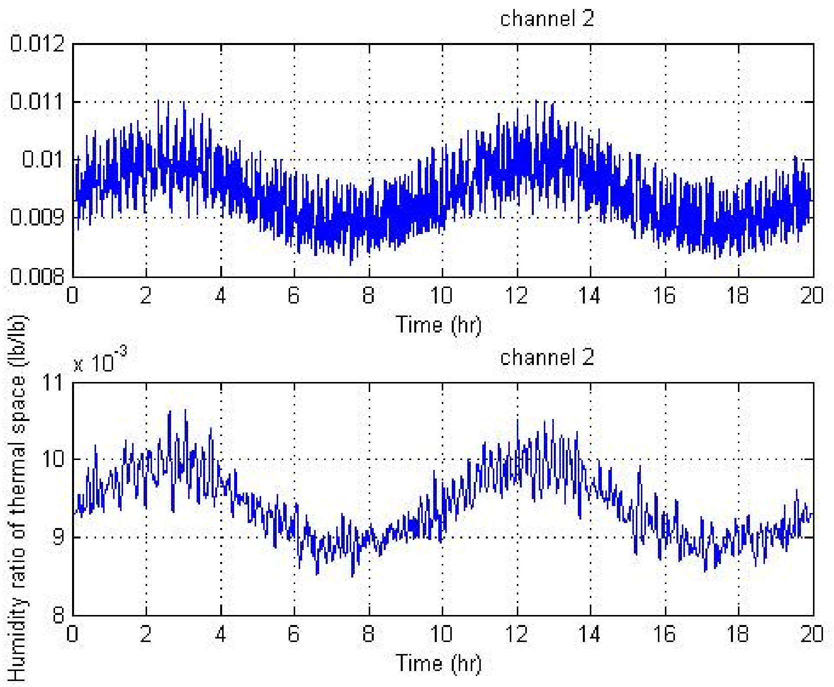
	CH	rRMSE tracking	MAE tracking	SNRN (dB)	SNRF (dB)	AvgT (ms)	MaxT (ms)
QN	1	0.009087	0.6124 °F	7.596	12.64	5.401	695.0
	2	0.01269	0.000094 lb/lb	6.062	11.07		
NM	1	0.009959	0.6961 °F	7.556	12.30	3.872	523.6
	2	0.01461	0.000111 lb/lb	7.531	11.37		

Table 5.15: The control results for HVAC system with normalized noise amplitude between [-0.45, 0.45]

	CH	rRMSE tracking	MAE tracking	SNRN (dB)	SNRF (dB)	AvgT (ms)	MaxT (ms)
QN	1	0.01653	1.115 °F	2.943	7.894	29.25	696.2
	2	0.02404	0.000181 lb/lb	1.661	6.548		
NM	1	0.01801	1.264 °F	3.467	7.986	33.59	694.4
	2	0.02650	0.000203 lb/lb	3.357	7.118		

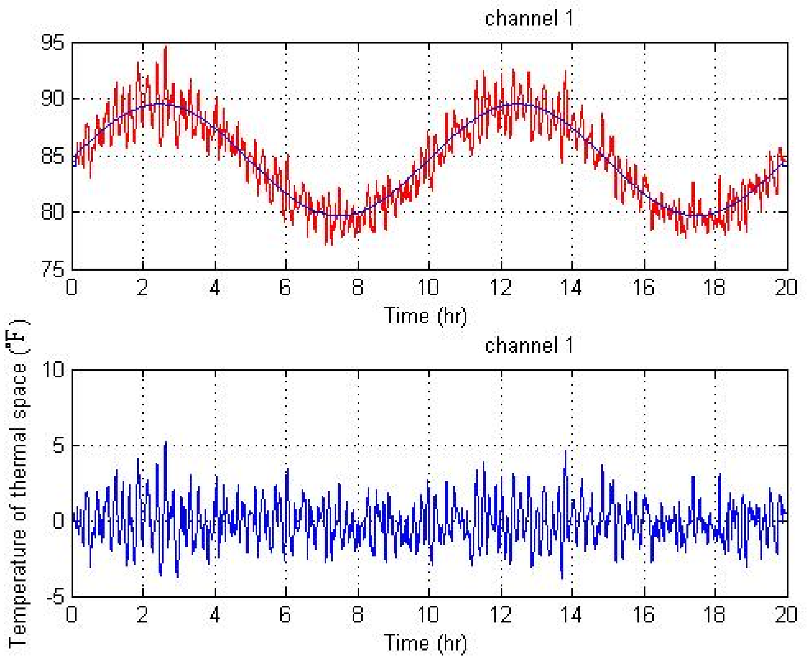


(a)

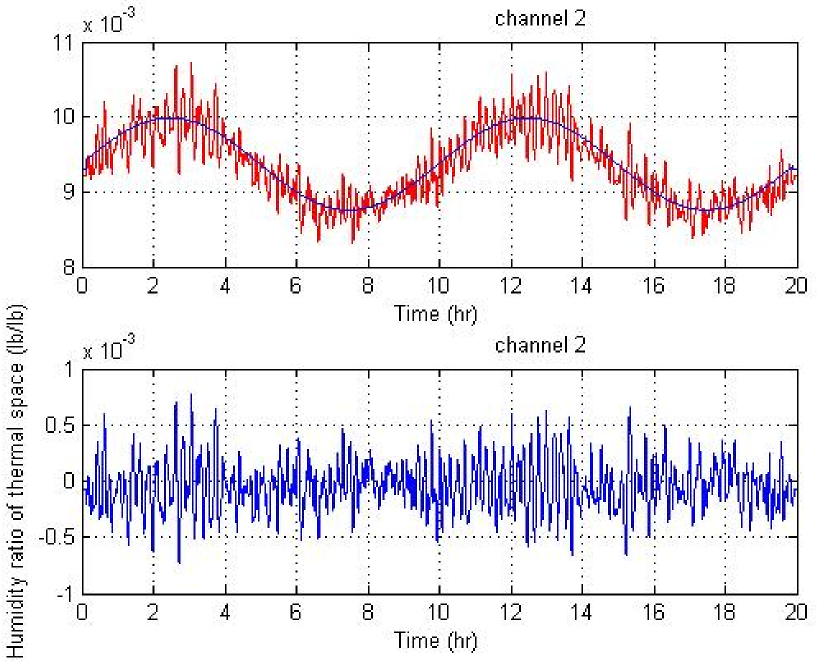


(b)

Figure 5.31: The HVAC system feedback (top) and the filtered feedback (bottom) of Quasi-Newton method with normalized noise amplitude between [-0.45, 0.45]: (a) The temperature of thermal space; (b) The humidity ratio



(a)



(b)

Figure 5.32: The tracking performance (top) and tracking error (bottom) of HVAC system for Quasi-Newton method with noise amplitude between [-0.45, 0.45] (red line: filtered feedback, blue line: reference trajectory): (a) The temperature of thermal space; (b) The humidity ratio

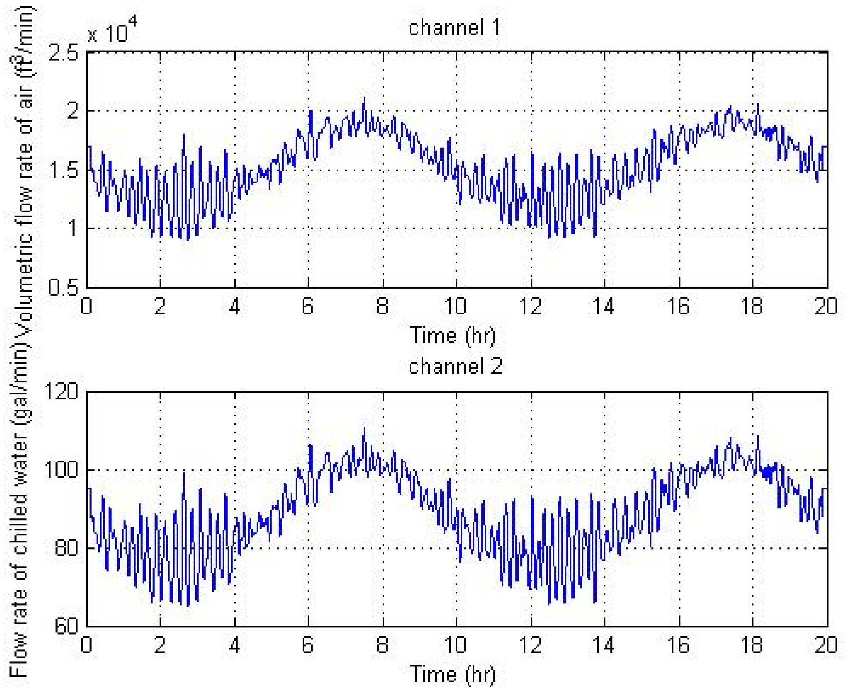
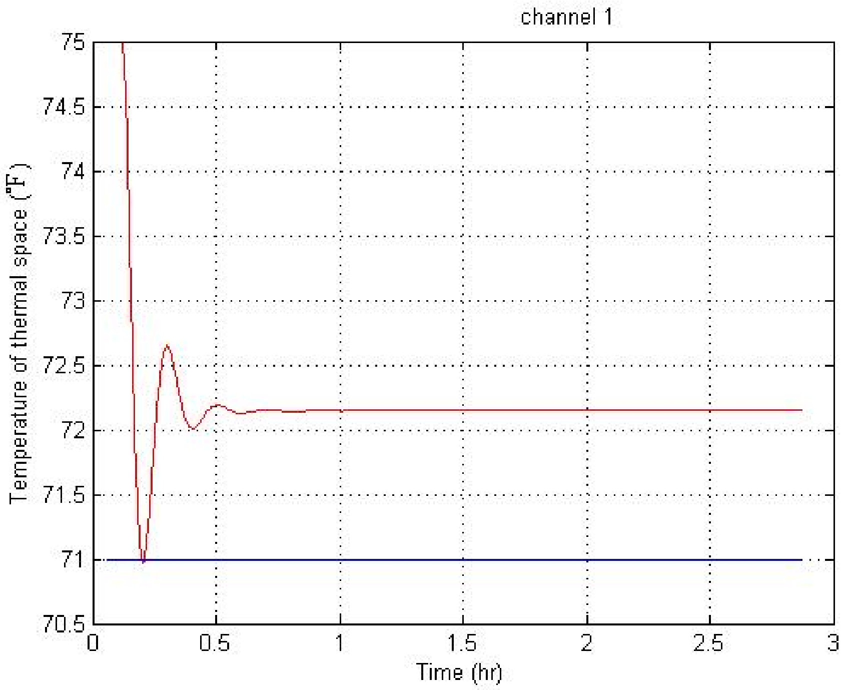


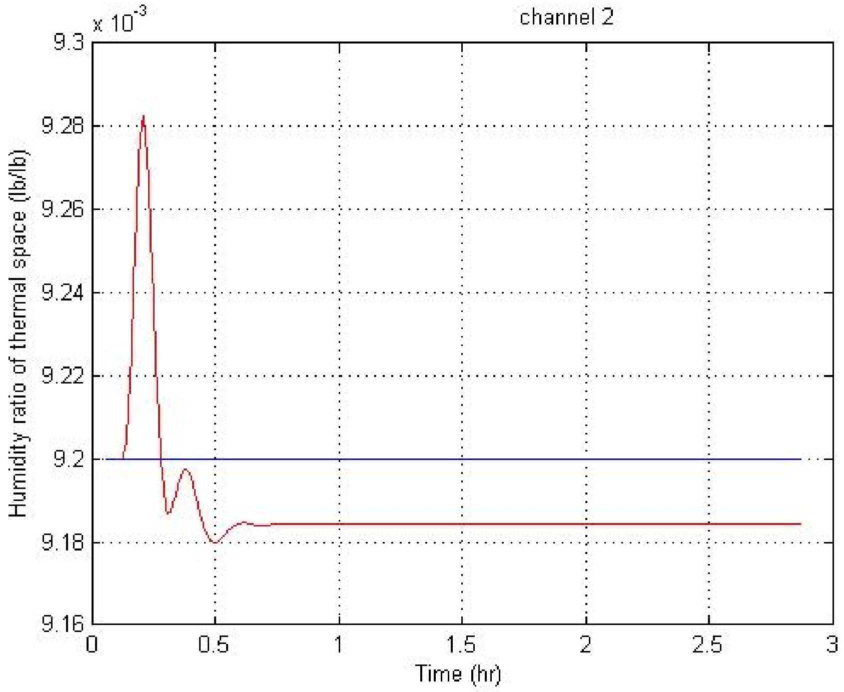
Figure 5.33: The close-loop HVAC system control signal for Quasi-Newton method with normalized noise amplitude between [-0.45, 0.45]

From Tables 5.11 to 5.15, it can be seen that the performance of the controller is influenced by the amplitude of noise. The control effect becomes worse if the noise is larger. The control using Quasi-Newton method usually has a better tracking performance than the one using Nelder-Mead method.

If the HVAC system is acted upon by non-design thermal loads, the performance of the GPC is compared with the controller in [1]. The value of the loads for this simulation are $Q_o = 350000$ btu/hr and $M_o = 196$ lb/lb. The initial values of the state variables are set to $T_3 = 75$, $T_2 = 55$, $W_3 = 0.0092$



(a)



(b)

Figure 5.34: The performance of the GPC with non-design thermal loads (red line: feedback, blue line: reference trajectory): (a) The temperature of thermal space; (b) The humidity ratio

The time responses shown Fig. 5.34 indicate that the GPC minimizes the effect of the disturbances. The offsets present at the outputs are about 1.15°F for the room temperature and -1.6×10^{-5} lb/lb for room humidity. These offsets are about three times smaller than the controller without disturbance rejection [1] and similar to the disturbance rejection controller [1].

Chapter 6

Conclusions and Future Work

6.1 Conclusions

A GPC based on RBF-NN is designed to control multi-input multi-output nonlinear stochastic systems. The fundamental issues are cost function minimization and system modeling, identification. Three different methods of solving the optimization problems are used. The modified gradient descent method is the simplest method, which is a normalized original gradient descent method. This modified gradient descent method converges faster than the original one. It can be used for solving a simple optimization problem. However, in a complex problem, it requires long calculation time before the problem is solved, which is inefficient.

The Quasi-Newton method is based on Newton's method, but the Hessian matrix does not need to be calculated. The approximation of the inverse Hessian matrix is updated by certain updating rules, which usually is the BFGS method. This Quasi-Newton method has a better convergence rate, and is used for multi-dimensional optimization problems.

Another method introduced is Nelder-Mead method. Unlike gradient descent method and Quasi-Newton method, Nelder-Mead method is a gradient free method. The biggest advantage of this method is that it can be easily implemented. Instead of finding the functional derivatives, the optimization problems are solved by repeating some simplex searches. This method is commonly used for minimizing problems with non-differentiable functions. However, this method is inefficient for high-dimensional problems.

An RBF-NN has been presented by using gradient descent for learning algorithm, and a generalized predictive controller is presented by using Quasi-Newton method and Nelder-Mead method for cost function minimization algorithms.

The simulations for tracking control of a SISO benchmark plant and a MIMO HVAC system have been conducted. Different amplitude noise is applied to test the stability of the closed-loop systems.

The results have shown that the designed controller based on both algorithms perform similarly when the noise is small. However, when the larger noise is added to the system, the controller based on Nelder-Mead method has a worse tracking performance than the other one.

6.2 Future work

The simulation results are satisfactory based on given functional nonlinear stochastic systems. However, the designed generalized predictive controller based on RBF-NN has not been implemented on experimental system yet, the following work should be done in the future: (1) Test the controller on a one-DC-motor system for tracking control. (2) Implement the controller on a parallel robot system. (3) Apply other predictive control methods to further improve the control performances.

References

[1] B. Arguello-Serrano and M. Vélez-Reyes, "Nonlinear control of a heating, ventilating, and air conditioning system with thermal load estimation," *Control Systems Technology, IEEE Transactions on,* vol. 7, pp. 56-63, 1999.

[2] L. Bai and D. Coca, "Nonlinear predictive control based on NARMAX models," in *Optimization of Electrical and Electronic Equipment, 2008. 11th International Conference on*, pp. 3-10.

[3] S. Billings and I. Leontaritis, "Parameter estimation techniques for nonlinear systems," *6th IFAC Symp. Ident. Syst. Param. Est.*, Washington, D.C. pp. 427-432, 1982.

[4] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," *Complex Syst.*, vol. 2, pp. 321-355, 1988.

[5] W. Chan, C. Chan, K. Cheung, and Y. Wang, "Modeling of nonlinear stochastic dynamical systems using neurofuzzy networks," in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, 1999, pp. 2643-2648.

[6] F. C. Chen and H. K. Khalil, "Adaptive control of a class of nonlinear discrete-time systems using neural networks," *Automatic Control, IEEE Transactions on,* vol. 40, pp. 791-801, 1995.

[7] S. Chen, S. Billings, and P. Grant, "Non-linear system identification using neural networks," *International Journal of Control,* vol. 51, pp. 1191-1214, 1990.

[8] S. Chen and S. A. Billings, "Representations of non-linear systems: the NARMAX model," *International Journal of Control,* vol. 49, pp. 1013-1032, 1989.

[9] S. Chidrawar and B. Patre, "Generalized Predictive Control and Neural Generalized Predictive Control," *Leonardo Journal of Sciences,* vol. 7, pp. 133-152, 2008.

[10] D. W. Clarke, C. Mohtadi, and P. Tuffs, "Generalized predictive control—Part I. The basic algorithm," *Automatica,* vol. 23, pp. 137-148, 1987.

[11] C. R. Cutler and B. Ramaker, "Dynamic matrix control-a computer control algorithm," in *Proceedings of the joint automatic control conference*, 1980.

[12] V. Elanayar and Y. C. Shin, "Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems," *Neural Networks, IEEE Transactions on,* vol. 5, pp. 594-603, 1994.

[13] W. Guo and M. Han, "Generalized predictive controller based on RBF neural network for a class of nonlinear system," in *American Control Conference, 2006*, pp. 1569-1574.

[14] L. Haichuan, D. Wenzhan, and L. Meizhen, "A novel ACI motor vector method based on TS-FCMAC neural network predictive control algorithm," in *Information and Automation, 2008. International Conference on*, pp. 232-237.

[15] S. F. Hu, S. J. Zhu, M. J. Zhong, and Q. W. He, "Neural network modeling and generalized predictive control for Giant Magnetostrictive actuators," in *Control and Decision Conference, 2009. CCDC'09. Chinese*, 2009, pp. 2981-2985.

[16] J-S. R. Jang, C, Sun, *Neuro-Fuzzy and Soft Computing*, Prentice Hall, 1997.

[17] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence Properties of the Nelder--Mead Simplex Method in Low Dimensions," *SIAM Journal on Optimization,* vol. 9, pp. 112-147, 1998.

[18] M. ŁAwryńCzuk, "A family of model predictive control algorithms with artificial neural networks," *International Journal of Applied Mathematics and Computer Science,* vol. 17, pp. 217-232, 2007.

[19] M. Lazar and O. Pastravanu, "A neural predictive controller for non-linear systems," *Mathematics and Computers in Simulation,* vol. 60, pp. 315-324, 2002.

[20] I. Leontaritis and S. A. Billings, "Input-output parametric models for non-linear systems part I: deterministic non-linear systems," *International journal of control,* vol. 41, pp. 303-328, 1985.

[21] C. H. Lu and C. C. Tsai, "MIMO neural-network predictive controller design," in *Industrial Electronics Society, 2004. IECON 2004. 30th Annual Conference of IEEE*, 2004, pp. 1733-1738.

[22] J. Moody and C. Darken, *Learning with localized receptive fields*: Yale Univ., Department of Computer Science, 1988.

[23] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *Neural Networks, IEEE Transactions on,* vol. 1, pp. 4-27, 1990.

[24] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal,* vol. 7, pp. 308-313, 1965.

[25] J. Nocedal and S. J. Wright, *Numerical optimization*: Springer verlag, 1999.

[26] D. M. Prett, B. L. Ramaker, and C. R. Cutler, *Dynamic matrix control method*, U.S. Patent, 1982.

[27] J. Richalet, A. Rault, J. Testud, and J. Papon, "Algorithmic control of industrial processes," in *Proceedings of the 4th IFAC symposium on identification and system parameter estimation*, 1976, pp. 1119-1167.

[28] J. Richalet, A. Rault, J. Testud, and J. Papon, "Model predictive heuristic control: Applications to industrial processes," *Automatica,* vol. 14, pp. 413-428, 1978.

[29] F. Schwenker, H. A. Kestler, and G. Palm, "Three learning phases for radial-basis-function networks," *Neural Networks,* vol. 14, pp. 439-458, 2001.

[30] E. Semsar, M. J. Yazdanpanah, and C. Lucas, "Nonlinear control and disturbance decoupling of an HVAC system via feedback linearization and back-stepping," in *Control Applications, 2003. CCA 2003. Proceedings of 2003 IEEE Conference on*, 2003, pp. 646-650.

[31] J. C. Spall and J. A. Cristion, "Direct adaptive control of nonlinear systems using neural networks and stochastic approximation," in *Decision and Control, 1992.,*

Proceedings of the 31st IEEE Conference on, 1992, pp. 878-883.

[32] J. C. Spall and J. A. Cristion, "Model-free control of nonlinear stochastic systems in discrete time," in *Neural Networks, 1996., IEEE International Conference on*, 1996, pp. 1859-1864.

[33] Y. Tan and A. Van Cauwenberghe, "Neural-network-based *d*-step-ahead predictors for nonlinear systems with time delay," *Engineering applications of artificial intelligence,* vol. 12, pp. 21-35, 1999.

[34] B. T. Thumati, M. A. Feinstein, J. W. Fonda, A. Turnbull, F. J. Weaver, M. E. Calkins*, et al.*, "An online model-based fault diagnosis scheme for HVAC systems," in *Control Applications (CCA), 2011 IEEE International Conference on*, 2011, pp. 70-75.

[35] D. Wang and S. Xu, "Parallel model predictive control of nonlinear time-delay systems based on recurrent neural network," in *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*, 2008, pp. 677-680.

[36] F. Zada, S. K. Guirguis, and W. M. Sead, "Hybrid Neural Predictive-Fuzzy Controller for Motorized Robot Arm," in *Computer Science and Society (ISCCS), 2011 International Symposium on*, 2011, pp. 157-160.