INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Robust Model Predictive Control for Nonlinear Systems Based on Volterra Series

by

Shijiang Lu

A Thesis Submitted
in Partial Fulfillment of the Requirements for the Degree

Master of Science

Approved. Thesis Committee:

[Signatures]

Thomas A. Badgwell
Adjunct Professor of Chemical Engineering

Kyriacos Zygorakis
Professor of Chemical Engineering

Walter G. Chapman
Associate Professor of Chemical Engineering

Houston, Texas

April, 2001
Abstract

Robust Model Predictive Control for Nonlinear Systems based on Volterra series

by

Shijiang Lu

In this thesis we develop a Nonlinear Robust Model Predictive Control (NRMPC) algorithm for nonlinear plants modeled by second order Volterra series. Robust stability is achieved through the addition of cost function constraints that prevent the sequence of the optimal controller costs from increasing for the true plant. Model uncertainty is parameterized by ellipsoid bounds on the plant parameters. The same approach is used to reject constant output disturbances. If some certain assumptions concerning the disturbance and plant parameters are satisfied, the plant is guaranteed to reach an offset free steady state.
Acknowledgments

With my deepest appreciation, I acknowledge my advisor, Thomas Badgwell, for his support and guidance, for providing valuable perspective and insight and for allowing me considerable freedom in conducting research. He was very patient when I was not making progress and quick to point out what caused my problem. I learned from him both how to do good research and how to be a good person. I am also thankful to Kyriacos Zygourakis and Walter Chapman for serving on my thesis committee. I also greatly appreciate the help from members of the research group: Dean Kassmann, Sameer Rahal and Raymond Joe.

I would like to thank my mother for her love and support from the other side of this planet. I also want to thank my father although he never really helped me since I started this research. Although he is now in the heaven, I can still feel his love and I am sure he would always be happy for my achievements.

Finally, I would thank my wife Jinhua Bai. Without her love and help, this thesis may not even be finished.
## Contents

1 Introduction  .................................................. 1
  1.1 Model Predictive Control  ................................ 1
    1.1.1 A Brief History of MPC  ................................ 1
    1.1.2 MPC control Hierarchy  ................................ 4
  1.2 Typical MPC Scheme  ....................................... 6
    1.2.1 Model Forms  ........................................... 7
  1.3 Nominal MPC  .............................................. 8
  1.4 Robust MPC  ............................................... 9
  1.5 Nonlinear MPC  ............................................ 13
  1.6 Research Objectives  .................................... 15

2 Nonlinear models based on Volterra series  .................. 17
  2.1 Volterra Series  ........................................... 17
    2.1.1 A brief introduction to Volterra series  ................ 18
    2.1.2 Obtaining Volterra series models  ........................ 19
    2.1.3 Limitations of MPC based on Volterra series  .......... 20
  2.2 Model Uncertainty Parameterizations  ...................... 21
    2.2.1 Ellipsoidal Uncertainty  .............................. 21
    2.2.2 Box Uncertainty  ..................................... 23
3 A Robust MPC Algorithm Based on Volterra Series 
3.1 Plant Model and Control Problem 
3.2 Cost Function 
3.3 Robust SISO NQR 
3.4 A More General Case 
3.4.1 Plant Model and Control Problem 
3.4.2 Deviation Variable Formulation 
3.4.3 Controller Cost Functions 
3.4.4 Robustly Stabilizing Algorithm 
3.4.5 Robust Stability Theorem 
3.5 Robust Stability of the SISORNQR algorithm

4 Disturbance Rejection 
4.1 Plant Model and Control Problem 
4.2 Cost Functions 
4.3 Robust SISO NQR for Disturbance Rejection 
4.3.1 Robustness Constraints

5 Simulation Examples 
5.1 Effect of the size of $\Theta$ 
5.2 Effect of model mismatch 
5.3 Disturbance rejection 
5.4 SISORNQR with input constraints

6 Conclusions and Future Work
6.1 Summary of Contributions ........................................ 78
6.2 Suggestions for Future Work ........................................ 79

A Robustness Constraint: Analytical Solution 80

B Input Constraints 85

Bibliography 87
List of Figures

1.1 Model Predictive Control Hierarchy ........................................... 5
1.2 The idea behind MPC: future inputs are computed to drive predicted states or outputs to a setpoint ........................................... 8

2.1 Ellipsoid Uncertainty .............................................................. 22
2.2 Box Uncertainty ................................................................. 23

5.1 Effect of the size of $\Theta$ - $\bar{\gamma} = 0.1$ (--), $\bar{\beta} = 1$ (--) and $\bar{\beta} = 2$ (···) ........................................... 71
5.2 Effect of $\gamma - \gamma = 1$ (---), $\gamma = 1.2$ (_--) and $\gamma = 1.5$ (··) ........................................... 72
5.3 Disturbance rejection, when $\gamma = 3$, $\beta = 0.1$ and $d = 3$ ............. 73
5.4 Disturbance rejection, when $\gamma = 3$, $\beta = 2$ and $d = 3$ ................. 74
5.5 results of SISONRQR and ENLDMC algorithms .................................. 76
Chapter 1

Introduction

In this chapter we define Model Predictive Control (MPC), summarize the development of MPC technology, and outline the contents of this thesis.

1.1 Model Predictive Control

Model Predictive Control (MPC) refers to a family of controllers that use an explicit process model to compute a sequence of adjustments in order to optimize the future behavior of a plant. It can be used to solve complex control problems which are not solvable with traditional PID control concepts.

1.1.1 A Brief History of MPC

Traditional PID control has been widely used in industry for many years. However, traditional PID control concepts are inadequate for many advanced control applications. In fact, petrochemical plants as well as other manufacturing plants
often have the following characteristics that are difficult to address with PID control:

- Multiple inputs/outputs
- Difficult dynamics (large dead time, inverse response, zero gain, nonlinear, etc.)
- Non-square, time-varying structure
- Multiple process disturbances (measured and unmeasured)
- Highly constrained operation region
- Feed composition unknown and variable
- Tight product specifications

In some cases, a combination of simple control algorithms (PID) and logic blocks (high/low select) will address the characteristics listed above. However, such a solution will require considerable control engineering effort. Therefore, MPC was developed to solve these kinds of problems.

The history of MPC can be traced back to the work of Kalman in 1960 [28], but the major step forward was marked by a set of papers written in the late 1970s. In 1978, Richalet et al. [31] described successful applications of what they called "Model Predictive Heuristic Control". Their solution software was named IDCOM, an acronym for Identification and Command. In 1979, engineers from Shell presented "Dynamic Matrix Control" (DMC) with applications to a fluid catalytic cracker [11], [44]. Both algorithms use explicit dynamic plant models
to predict the future behavior of controlled variables. IDCOM and DMC represent the first generation of MPC technology: they focus mainly on decoupling multi-variable dynamics and treat constraints in a sub-optimal, heuristic manner.

In 1983, Cutler et al. [12] first described a Quadratic DMC (QDMC) algorithm in which constraints appear explicitly. A more comprehensive description of QDMC was given in 1986 by García and Morshed [21]. QDMC represents a second generation of MPC technology, which provided a systematic way to implement input and output constraints.

The second generation of MPC technology was successful in dealing with constraints, yet other practical problems remained unsolved. For example, disturbances entering the loop can drive the controller to an infeasible point, and loss of controlled variable measurements can change the control problem structure in real time. These and other issues were addressed in a third generation of MPC technology, first developed by Adersa and Setpoint, Inc. The version developed by Setpoint was called IDCOM-M (the M was to distinguish this from a single input/single output version called IDCOM-S) while the version by Adersa was called HIECON (Hierarchical Constraint Control). Other examples include PCT (Predictive Control Technology) by Profimatics and RMPC controller by Honeywell. The third generation of MPC technology distinguishes between several levels of constraints (hard, soft, ranked), provides some mechanism to recover from an infeasible solution, addresses the issues resulting from a control structure that changes in real time, and allows for a wider range of process dynamics and controller specifications.
In their review papers, Qin and Badgwell [45, 46] discuss in detail the evolution and industrial implementation of MPC technology.

1.1.2 MPC control Hierarchy

Figure (1.1) illustrates a representative model predictive control hierarchy. At the top level there is a plant-wide optimizer which calculates optimal steady-state settings for all local units. The local unit optimizer, which runs more frequently than the plant-wide optimizer, uses a more detailed unit model to compute optimal steady-state targets for the MPC algorithm. The role of the MPC algorithm is to move the plant from one constrained steady-state to another while minimizing constraint violations along the way. It can be further divided into two parts: steady-state target calculation and dynamic MPC calculation.

The steady-state calculation runs more frequently than the local unit optimizer and uses a less-detailed model (usually an approximation of the nonlinear model at each control execution, or the steady-state version of the dynamic model used for the dynamic optimization). Its role is to recalculate the targets for each input and output every time the MPC controller executes. This part is indispensable because the targets from the local optimizer may no longer be optimal due to disturbances entering the plant or other factors such as new input information from the operator, etc. This idea was first referenced in MPC by Cutler et al. [12] and is now a common part of industrial MPC technology.
Figure 1.1: Model Predictive Control Hierarchy
When given the steady-state targets, the dynamic MPC will calculate the best input adjustments to go from its current steady state to the new steady-state target. The dynamic MPC calculation has been studied extensively. Theoretical issues are discussed in review papers by García et al. '22', Morari and Lee '37', Muske and Rawlings '38', and Meadows and Rawlings '36'. The most comprehensive presentation is that of Mayne et al. '35'.

Some MPC implementations combine the steady-state and dynamic calculations into a single optimization. The advantage is that only one optimization problem needs to be solved. The disadvantage is that the steady-state and dynamic objectives may conflict.

1.2 Typical MPC Scheme

A typical MPC controller is defined as follows: Given the current state (or output) of the process, \( x_k \) (or \( y_k \)), find the future input sequence \( \pi_k = \{u_k, u_{k-1}, \cdots\} \), that will minimize an objective function (which usually penalizes the deviations of inputs and outputs from the desired set-point values and sometimes penalizes input/output changes as well). Once an optimal input sequence \( \pi_k \) is found, the first input is applied to the plant. The plant will then evolve and at next time step, the above calculation will be done again, and so on. Figure (1.2.1) is an illustration of the idea behind MPC.
1.2.1 Model Forms

The models that MPC controllers use can take many forms: state-space model, impulse/step response model, second-order Volterra model, etc. State-space models are generally derived from basic conservation laws (conservation of mass, momentum, and energy, for example) although any linear empirical model can also be expressed as a linear state-space model. A state-space model can represent all kinds of process dynamics: linear or nonlinear, stable or unstable. This is very promising since basically any plant can be represented.

Step/Impulse response models are determined from process response tests in which a specially designed test signal enters the plant and the output response is recorded. Model parameters of step/impulse response models are derived later on, based on the input and output data. Step/Impulse response models can only be used to represent linear and stable models because of their nature. They require less knowledge about the process, though, which makes them very popular in MPC applications.

Models based on Volterra series (which will be discussed in greater detail later in this chapter) can be considered an extension of step/impulse response models to nonlinear plants. As Froisy (17) pointed out, second-order Volterra models may bridge the gap between linear empirical models and nonlinear fundamental models. Second-order Volterra models can describe nonlinear behavior such as: asymmetric output changes to symmetric changes in the input; and, a sign change in a process gain. Boyd and Chua (8) considered the question of what class of system behavior may be approximated by such truncated Volterra models.
1.3 Nominal MPC

If an MPC controller uses a perfect model, it is called nominal MPC. Usually, nominal properties (such as nominal stability and nominal performance) are considered first by MPC control designers, since all MPC algorithms should work sufficiently well when the model is perfect. Meadows and Rawlings [36] discussed many aspects of nominal MPC. Mayne et al. [35] provide a more recent presentation of nominal MPC theory.
1.4 Robust MPC

In practice, the process model used by an MPC controller is never perfect. Modeling errors may be due to poor design or execution of test signals, invalid modeling assumptions, changes in plant operation, etc. For tank reactors or hydraulic systems, for example, complex reactions or nonlinear valve characteristics can cause strong nonlinearities and parameter uncertainty. The controller, which relies on the model to predict the future plant behavior, can cause the closed-loop system to go unstable for sufficiently large modeling errors.

Robust MPC algorithms are designed to tolerate modeling errors. The term "robust" here refers to the ability of the MPC algorithm to effectively deal with differences between the controller's internal model (also known as nominal model, which the controller uses to predict the system's behavior) and the actual plant or system. In most cases, one must trade performance for robustness, which means adding robustness will typically make optimizers run slower and/or control moves more conservative, etc. Similar to nominal MPC, there are two important concepts in robust MPC: robust performance and robust stability. By robust performance we mean the ability to achieve acceptable control performance in the presence of model uncertainty. Robust stability is a weaker requirement which ensures stability in the presence of modeling errors. Robust stability is the goal of most robust algorithms presented in the literature; however, in industry people usually demand robust performance.

Robust MPC algorithms can be divided into four categories.

- **Minimizing the worst-case controller cost**: These algorithms, called
*Min-Max* algorithms, minimize the worst case controller cost for a set of possible plants. The idea comes from the Minimax theorem used in same theory '9'. The idea was first brought forward by Campo and Morari '9' and was discussed in detail by Lee and Yu '33'. For time-invariant plants with infinite control horizon, a Min-Max cost function usually takes the following form:

$$\min_{\exists \kappa} \max_{\theta \in \Theta} \left\{ \sum_{k=0}^{\infty} (y_k^T Q y_k + r_k^T R r_k) \right\}$$  \hspace{1cm} (1.1)

where \( y_k \) and \( r_k \) are outputs and inputs, respectively; \( \tau_k \) is the future input sequence and \( \Theta \) is the set of all possible plant parameters \( \theta \). In the expression above, it is also assumed that \( Q \geq 0 \) and \( R > 0 \). For time-varying plants and/or finite horizon problems, the above cost function will change accordingly.

The Min-Max method is a "pessimistic" method in that it always try to control the worst parameter set. Hence, it always leads to conservative control moves. In other words, it doesn't take advantage of the sometimes known parameter uncertainties (we will discuss parameter uncertainties later in chapter 2). Nevertheless, this is the most heavily studied robust MPC approach in recent years.

- **Detuning the controller**: This method is motivated by the fact that one can always stabilize a stable plant by making the controller less aggressive. (When the controller is completely passive, i.e., there are no input changes, the system degrades to an open-loop one). Detuning is done by changing the tuning parameter(s) of the nominal MPC algorithm so as to suppress
input movements. Tuning parameter(s) must be chosen carefully so that the control moves are neither too conservative nor infeasible. Vuthandam et al. 58 provided an example which shows how to compute input weights in order to robustly stabilize a modified QDMC controller. Geneci and Nikolaou 25 use this approach on nonlinear processes modeled by Volterra series.

Methods based on detuning the controller have two fundamental limitations. First, closed-loop performance may suffer unnecessarily when the model is accurate. Second, appropriate tuning parameter(s) must be computed from extensive closed loop simulation.

- **Using state constraints:** There are three major types of state constraints: terminal state constraints, finite horizon state contraction constraints and next-step state contraction. **Terminal state constraints** 55, 25 force the largest possible terminal state to lie within an invariant set, which is equivalent to appending

\[
\max |x_{k-Y}(\theta)| \in W_a \quad \forall \theta \in \Theta
\]  

(1.2)

to the nominal problem. **Finite horizon state contraction constraints** 61 require states to contract over a finite horizon for all possible plants, which is equivalent to appending

\[
|x_{k-Y}(\theta)| \leq \lambda |x_k(\theta)|, \quad \forall \theta \in \Theta
\]  

(1.3)

for \( \lambda \in (0, 1) \) to the nominal problem. **Next step state contraction** 31.
is equivalent to appending

$$\|r_{k-i-1}(\theta)\| \leq \lambda \|r_{k-i}(\theta)\|, \quad \forall \theta \in \Theta$$

(1.4)

for $\lambda \in (0, 1)$ to the nominal problem.

These methods may have feasibility problems or may be so conservative that they cause an unacceptable performance loss (if the finite horizon is short, for example).

- **Using cost function constraints:** These algorithms require the controller cost function to be non-increasing for all possible plants. The idea behind it is: *using the newly calculated input sequence should lead to a result that is not worse than using the feasible sequence calculated at last time step.*

This is accomplished by adding another constraint of the form

$$\Phi(x_k, \bar{\pi}_k, \theta) \leq \Phi(x_k, \hat{\pi}_k, \theta), \quad \forall \theta \in \Theta$$

(1.5)

to the problem. In the above inequality, $\Phi$ is the controller cost function, $\bar{\pi}_k$ is the feasible input sequence from last time step and $\pi_k$ is the input sequence that the controller tries to generate at time $k$. Because the cost function includes contributions from both inputs and states, this can be considered a natural generalization of state contraction constraints. Feasibility problems are avoided, however, since (1.5) is always feasible for $\pi_k = \hat{\pi}_k$, provided that the cost function is bounded.
This approach is similar to the Min-Max method in that it tries to satisfy all possible plant parameter sets. It's less conservative compared to Min-Max approach, however, since it tries to optimize based on nominal plant parameters $\theta_c$ instead of the worst plant parameter set.

The idea of using cost function constraints to get robustness was first developed by Badgwell in 1997 [4], [3] for both linear and nonlinear stable plants. Ralhan [49] used it on linear finite impulse response models and showed a modified algorithm that is able to reject constant disturbances. Ralhan later extended these results to include linear state-space systems, both stable and integrating, [50], [48], [47].

1.5 Nonlinear MPC

Linear MPC is widely accepted because of its simplicity. There are analytical solutions (for unconstrained systems), hence generally more rigorous stability and performance proofs are available. Also, the computational demands for linear system simulation (and implementation) are usually quite small when compared to a nonlinear simulation. However, all chemical process systems are nonlinear to some extent. Traditional approaches to dealing with nonlinearity include using local linearization and detuning linear controllers, etc. When nonlinearity is mild, a local linearization can be used in the neighborhood of a nominal operating point. Then a linear compensator design method can be applied on the linear
model. Or, one can simply detune linear controllers to compensate for nonlinearity. The above approaches have their limitations though: local linearization may work well in the close vicinity of the nominal operating point but its performance and stability become uncertain when the system is not operating close enough to the nominal point. As to detuning linear controllers, the nonlinearity is totally omitted. Although feedback control can decrease the effects of nonlinearity, both performance and stability may be problematic when the process is strongly nonlinear. In many cases, simple linear models are inadequate, therefore process control based directly upon nonlinear models becomes very important.

There are many types of nonlinear models. Some of them are difficult to use in an industrial application. This is partly because they are just too complex, making controlling the plant relatively hard. More importantly, these models usually require nonlinear differential equation process models (e.g., 23). However, in industrial practice, such first principles models are often not readily available because of a chronic lack of detailed and extensive process knowledge required for their development and the (usually prohibitive) cost to develop such models. Hence the question arises: can we find a relatively simple (yet powerful) approach which doesn’t have to rely on knowing the process very well?

Input-output models are good candidates since they are relatively simple and can be identified separately without knowing anything other than the corresponding inputs and outputs. There are a wide variety of nonlinear input-output models, for example, bilinear models(60, 5), Hammerstein models(15, 18, 62), Wiener models(41), Volterra models and other empirical nonlinear models(10, 40), etc. Among various nonlinear control approaches, the one based on the
Volterra series expansion is very promising. The reasons are as follows: Volterra series representation is an universal approximation to nonlinear dynamic systems [8]; it has a clear and useful structure in nonlinearity [14]; it can be identified by using input-output data without other knowledge of the process [6]; its control is also based on the input-output information (state variables are usually not accessible in complex chemical processes) [54] [14]. These features are very important to chemical process control practice. Therefore, process control schemes based on Volterra series have been studied by many researchers in the past decade [14], [25]. We will see later that the second order Volterra model is similar in form to the conventional MPC model and it gives rise to a controller that can be decomposed into a conventional linear model-based controller plus a sequence of analytical nonlinear “perturbation” (or correction terms). Hence performance can be improved for nonlinear systems with moderately increased effort.

1.6 Research Objectives

The goals of this research project are:

- Extend previous cost function constraint method to the case of nonlinear systems described by a Volterra series model

- Develop an efficient numerical solution method that exploits problem structure

- Demonstrate the potential of the new algorithm using simulation examples

This thesis is organized as follows. The first chapter introduces Volterra models for nonlinear systems. The next chapter presents a robust MPC algorithm for
Volterra series systems using cost function constraints. Subsequent chapters discuss disturbance rejection and constraint handling. The final chapters provide simulation examples and a summary of conclusions drawn from this research.
Chapter 2

Nonlinear models based on
Volterra series

In this chapter we provide an introduction to Volterra series models and describe two methods for parameterizing model uncertainty.

2.1 Volterra Series

The idea of using Volterra series to calculate analytically the control law required to eliminate successively higher-order effects in the closed-loop was originally introduced by Sain and Al-Baiyat [1, 2]. Volterra series have the following properties:

- Model parameters appear linearly, i.e., one can use $\xi = \mathcal{U}\theta$ (where $\xi$ is output vector, $\theta$ contains plant parameters and $\mathcal{U}$ is a matrix containing inputs and product of inputs) to calculate outputs.

- Modeling uncertainty can be quantified as confidence intervals for the model
parameters \[14\].

- For many chemical processes that cannot be approximated adequately by the linear model (2.6) (high-purity distillation columns and exothermic chemical reactors for example), second-order Volterra series model can be used as a nonlinear extension for (2.6) (since second-order Volterra model is a superset of the linear model (2.6)), which often leads to better control performance.

### 2.1.1 A brief introduction to Volterra series

For linear systems, consider the linear convolution integral

\[
y(t) = \int_0^\infty h_1(\sigma)u(t - \sigma)d\sigma
\]  

(2.1)

which is a functional that maps the entire past input to the value of the output at the present time. This is known as first order Volterra model, which is a very useful model for representing arbitrary system dynamic behavior. Here \(h_1(\sigma)\), also called the 'kernel' of the transformation, is the system's impulse response function. For nonlinear systems, this concept can be extended to add second and higher order Volterra series which contain cross-multiplications between past inputs, i.e., the functional shown in equation (2.1) can be generalized to a power series:

\[
y(t) = y_1(t) + y_2(t) + y_3(t) + \cdots.
\]  

(2.2)

where the first-order term \(y_1(t)\) is defined as in (2.1) and the second and higher order terms are defined as follows:
\[ y_2(t) = \int_0^\infty \int_0^\infty h_2(\sigma_1, \sigma_2) u(t - \sigma_1) u(t - \sigma_2) d\sigma_1 d\sigma_2 \]  
(2.3)

\[ y_i(t) = \int_0^\infty \cdots \int_0^\infty h_i(\sigma_1, \ldots, \sigma_i) u(t - \sigma_1) \cdots u(t - \sigma_i) d\sigma_1 \cdots d\sigma_i \]  
(2.4)

In chemical process control practice, we generally use discrete-time representations instead of continuous-time ones. The discrete-time representation takes the form of the power series below:

\[ y(k) = y_1(k) + y_2(k) + \cdots \]  
(2.5)

where the terms \( y_1(k) \), \( y_2(k) \) are given as follows (Higher-order terms have similar structures):

\[ y_1(k) = \sum_{i=1}^{\infty} h_1(i) u(k - i) \]  
(2.6)

\[ y_2(k) = \sum_{i=1}^{\infty} \sum_{l=1}^{\infty} h_2(i, l) u(k - i) u(k - l) \]  
(2.7)

### 2.1.2 Obtaining Volterra series models

Volterra series models can be obtained in the following ways:

- From a nonlinear first principles model, by obtaining the Carleman linearization (approximation) \(^\text{54}^\) \(^\text{52}^\) of the original system. The pertinent details of this procedure can be found in the two cited references.

- From a polynomial NARMAX model, discussed by Diaz and Desrochers \(^\text{13}^\).
• From an artificial neural network. It has been shown \[59\] that for every artificial neural network that employs the usual sigmoidal squashing function, there is an exact (not approximate) equivalent infinite Volterra series representation. By using the infinite series representation of these sigmoidal functions, it is possible, through straightforward (but quite tedious) algebraic manipulations, to compute precisely the equivalent Volterra kernels from the synaptic weights.

• From input/output data. A number of techniques have been proposed for computing the Volterra kernels strictly from input/output data gathered directly from the process. In each case the choice of input signals and the optimization criteria employed in computing parameter estimates are very critical elements of the identification procedure. New results and a detailed discussion of these and other related issues are given by Pearson et al. \[43\] \[42\]. Information about identifying Volterra series is also available in \[56\] \[54\] \[52\] \[29\] \[6\] \[42\].

2.1.3 Limitations of MPC based on Volterra series

Like all other control schemes, MPC based on Volterra series has its limitations. First, Volterra series representation is a temporal extension of the Taylor series expansion: it is therefore limited to approximating systems with fading memory \[8\] (roughly speaking, a fading memory system is one whose dependence on past inputs decreases "rapidly enough" with time. The concept of fading memory is important in that it provides a useful and broad partitioning of the enormous class of "nonlinear systems"). Second, if we have some (but not full) knowledge on how the plants behave, the knowledge usually cannot be put into our Volterra
model explicitly, making our model less precise than using the first principle models directly. A third limitation is that a large number of model parameters may be required in order to describe the nonlinearity. Recent work by Nikolaou, etc. [39] have shown how to reparameterize the Volterra series model using a small number of wavelets.

2.2 Model Uncertainty Parameterizations

The models that MPC controllers use can never be perfect in real life due to poor signal design, changes in plant operation, etc. Besides, MPC controllers usually use a single model to control a system. The model used may be accurate enough near some operating points but will not be as accurate in the system's whole operating region. Model uncertainty (which is usually described as ellipsoidal uncertainty or box uncertainty) therefore becomes an indispensable issue that robust MPC algorithms must address.

2.2.1 Ellipsoidal Uncertainty

As shown in figure (2.1), the plant parameter vector

$$
\bar{\theta} = \left[ \bar{\theta}_1 \cdots \bar{\theta}_{n_p} \right]^T \in \mathbb{R}^{n_p}
$$

(2.8)

where $\bar{\theta}_1, \bar{\theta}_2, \ldots, \bar{\theta}_{n_p}$ are plant parameters, is assumed to lie in the set $\Theta$:

$$
\bar{\theta} \in \Theta \triangleq \left\{ \theta : (\theta - \theta_c)^T W (\theta - \theta_c) \leq 1 \right\}
$$

(2.9)
Figure 2.1: Ellipsoid Uncertainty

The center of the ellipse is located at $\theta_c$ and the symmetric positive-definite matrix $W$ gives the size and orientation of the ellipsoid. In particular, the square roots of the reciprocals of the eigenvalues of $W$ are the lengths of the semi-axes of the ellipsoid, and the eigenvectors of $W$ define the directions of the semi-axes. If our assumption that all $\theta$'s lie in the ellipse fails, it means the model we are using is not accurate enough. In that case, the control designer needs to go through the identification procedure again to get a better set of plant parameters.

Ellipsoidal uncertainty arises naturally when noise and plant parameters are assumed to be Gaussian variables during the identification procedure, hence it has been widely used by researchers [27, 30, 32]. In this thesis, we will use the ellipsoidal uncertainty representation.
2.2.2 Box Uncertainty

Ellipsoidal uncertainty can be expressed in another form as

$$\Theta \triangleq \{ \theta : \| \mathbf{W}^\top (\theta - \theta_r) \|_2 \leq 1 \}$$  \hspace{1cm} (2.10)

where $\| \cdot \|_2$ denotes the Euclidean norm. If we use $\infty$-norm instead of the 2-norm above, we will get an expression for box uncertainty, which is illustrated in figure (2.2).

Box uncertainty has also been used by many researchers. \cite{9, 24}. 
2.3 Input/Output Constraints

All real world control algorithms must be able to handle input and output constraints as they bring the output to the origin:

\[ u_j \in \mathcal{U} \]
\[ y_j \in \mathcal{Y} \] (2.11)

The input constraint space \( \mathcal{U} \) and output constraint space \( \mathcal{Y} \) are assumed to be closed, with the point (0,0) contained in \( \mathcal{U} \times \mathcal{Y} \).

Input constraints considered in this thesis include both absolute (saturation) constraints:

\[ u_{\text{min}} \leq u_{j,k} \leq u_{\text{max}}, \quad j = 0, N - 1 \] (2.12)

and rate of change constraints:

\[ \Delta u_{\text{min}} \leq \Delta u_{j,k} \leq \Delta u_{\text{max}}, \quad j = 0, N \] (2.13)

where the change in the input is given by \( \Delta u_{j,k} = u_{j,k} - u_{j-1,k} \). As shown in Appendix B the constraints 2.12 and 2.13 can be written as a linear inequality in the input vector \( \pi_k \):

\[ D\pi_k \leq d \] (2.14)

In this thesis, we assume there is no output constraint. However, soft output constraints can be easily added to our algorithm without destroying the robustness behavior of our algorithm, as shown by Badgwell [4].
Chapter 3

A Robust MPC Algorithm Based on Volterra Series

In this chapter we describe a simplified robust MPC algorithm for stable nonlinear systems. Global robust stability is proven for the case in which the true plant lies within the uncertainty description.

3.1 Plant Model and Control Problem

The plant to be controlled is nonlinear, stable, single input-single output (SISO) and is described to a sufficient accuracy by the following discrete time second order Volterra model which has finite plant parameters $h_i$ and $g_{dl}$:

$$y_k = \sum_{i=1}^{M} h_i u_{k-i} + \sum_{i=1}^{M} \sum_{l=1}^{M} g_{dl} u_{k-i} u_{k-l} \tag{3.1}$$

where $y_k \in \mathbb{R}$ and $u_k \in \mathbb{R}$ are the output and input of the plant at time $k$, respectively. The goal of the control system is to generate a sequence of inputs
$u_k$ that brings the output of the system from an initial nonzero value to the origin. The Volterra parameter of the true plant $\tilde{\theta}$:

$$\tilde{\theta} \triangleq \begin{bmatrix} \tilde{h}_1 & \tilde{h}_2 & \cdots & \tilde{h}_M & \tilde{g}_{11} & \tilde{g}_{12} & \cdots & \tilde{g}_{MM} \end{bmatrix}^T$$  \hspace{1cm} (3.2)

is not known exactly but is assumed to lie in a set $\Theta$:

$$\tilde{\theta} \in \Theta \triangleq \left\{ \theta : (\theta - \theta_c)^T W (\theta - \theta_c) \leq 1 \right\}$$  \hspace{1cm} (3.3)

where the matrix $W$ is symmetric and positive definite. In the algorithm presented here the control designer selects a nominal model $\tilde{\theta}$ from the set $\Theta$ as a basis for controlling the plant, i.e., the control designer uses $\tilde{\theta}$ to predict future plant behavior while the actual behavior is determined by $\tilde{\theta}$.

### 3.2 Cost Function

The controller here regulates the plant output by minimizing a quadratic cost function $\Phi : \mathbb{R}^N \times \mathbb{R}^M \times \Theta \to \mathbb{R}$ that penalizes deviations of the output and the input from the origin:

$$\Phi(\pi, x, \theta) \triangleq \sum_{j=0}^{\infty} \left( \lambda r_j^2 + z_j^2 \right)$$  \hspace{1cm} (3.4)

where

$$z_j = \sum_{i=1}^{M} h_i r_{j-i} + \sum_{i=1}^{M} \sum_{i=1}^{M} g_{i,i} r_{j-i} r_{j-i}$$  \hspace{1cm} (3.5)

$$r_j = 0, \quad \forall j \geq N$$  \hspace{1cm} (3.6)
\[
\pi = \begin{pmatrix} v_0 & v_1 & \cdots & v_{N-1} \end{pmatrix}^T \\
x = \begin{pmatrix} v_{-1} & v_{-2} & \cdots & v_{-M} \end{pmatrix}^T \\
z_j = 0 \quad \forall j \geq N + M \\
\lambda > 0
\] (3.7) (3.8) (3.9) (3.10)

In the cost function \( z_j \) are the predicted future outputs, \( v_j \) are past/current/future inputs, \( \pi \) is the collection of current and future inputs in vector form, and \( x \) contains the past inputs which affect the future outputs. The weighting factor \( \lambda > 0 \) is used to obtain a trade-off between the costs of input and output regulation. The input is set to the origin at time \( k + N \) and remains there for all future time. As a result the output settles down to the origin at time \( k + N + M \) and remains there. In other words, prediction horizon is \( N \) and control horizon is \( N + M \). The infinite horizon objective function in (3.4) can therefore be rewritten as

\[
\Phi(\pi, x, \theta) \triangleq \sum_{j=0}^{N-1} \lambda^j r_j^2 + \sum_{j=0}^{N-M} \theta_j^2
\] (3.11)

The nominal model cost function \( \Phi(\pi, x) \) to be minimized by the control algorithm is based on the nominal model \( \hat{\theta} \), and is defined as:

\[
\hat{\Phi}(\pi, x) \triangleq \Phi(\pi, x, \hat{\theta})
\] (3.12)

The true cost of an input sequence must be measured by its effect on the actual plant. For this purpose we define the true plant cost function

\[
\bar{\Phi}(\pi, x) \triangleq \Phi(\pi, x, \bar{\theta})
\] (3.13)
3.3 Robust SISO NQR

We now define a simplified robust nonlinear quadratic regulator in order to introduce the robustness constraint and to explore the stability properties of the algorithm in the absence of input/output constraints and disturbances. The robustness constraint is the same one presented earlier by Rakha [49].

**Definition:** SISORNQR The *Single Input Single Output Robust Nonlinear Quadratic Regulator* finds the input vector $\hat{\pi}_k$ that minimizes the nominal model cost

$$
\hat{\pi}_k \triangleq \arg \min \left( \Phi(\pi_k, x_k) \right)
$$

subject to a robustness constraint

$$
\Phi(\pi_k, x_k, \theta) \leq \Phi(\hat{\pi}_k, x_k, \theta) \forall \theta \in \Theta
$$

(3.15)

The first element of optimal input vector $\hat{\pi}_k$ is then injected into the plant:

$$
u_k = \hat{u}_{k-1}^* \\
(3.16)

The input $\hat{u}_k$, referred to as the *restriction of the input*, is a shifted version of the previous optimal input $\hat{\pi}_k$. If the optimal input at time $k-1$ is defined as:

$$
\hat{\pi}_{k-1} = \begin{bmatrix} \hat{u}_{k-1, k-1}^* & \hat{u}_{k-1, k-2}^* & \cdots & \hat{u}_{k-1, N-2, k-1}^* \end{bmatrix}^T
$$

(3.17)

then the restriction of the input at time $k$ is given by:

$$
\hat{\pi}_k = \begin{bmatrix} \hat{u}_{k-1, k-1}^* & \cdots & \hat{u}_{k-1, N-2, k-1}^* & 0 \end{bmatrix}^T
$$

(3.18)
At the first time step the restriction of the input is initialized to:

\[
\tilde{z}_0 = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}^T
\]  

(3.19)

The robustness constraint requires the cost function values for any plant in the uncertainty domain to either remain unchanged or to decrease at each time step, relative to the cost function values computed using the the restriction of the input. At time \(k\), the control problem changes in two ways relative to time \(k-1\):

- the state \(x_k\) is measured
- an additional input is available for optimization, \(u_{k-N-1}\)

By using cost function constraints, the controller will try to find a better input sequence with this additional information. If a better input sequence is found, then the first element in this new input vector will be injected into the plant. If a better input sequence cannot be found for whatever reason (either calculation time limit is exceeded or there is in fact no better input sequence), at least the controller has a feasible input: the first element in \(\tilde{z}_k\), which was calculated at last time step. An important property of the robustness constraint is that it is feasible at each time step for the choice \(\pi_k = \tilde{z}_k\). Another advantage is that this constraint introduces no additional tuning factors.

The robustness constraint (3.15) is infinite dimensional and may therefore be difficult to implement. So we reformulate (3.15) as:

\[
\max_{\theta \in \Theta} (\Delta \Phi(\pi_k, \tilde{z}_k, x_k, \theta)) \leq 0
\]

(3.20)
where
\[
\Delta \Phi(\pi_k, \tilde{\pi}_k, x_k, \theta) = \Phi(\pi_k, x_k, \theta) - \Phi(\tilde{\pi}_k, x_k, \theta)
\]  
(3.21)

Now the vector of future outputs at time k.

\[
\xi_k \overset{\Delta}{=} \begin{bmatrix} y_1 & y_2 & \cdots & y_{N-M} \end{bmatrix}^T
\]  
(3.22)

can be written as

\[
\xi_k = \mathcal{U}_k \theta
\]  
(3.23)

where

\[
\mathcal{U}_k(\pi_k, x_k) = \bigcup_{h,k} \bigcup_{g,k} \pi_k, x_k
\]  
(3.24)

\[
\mathcal{U}_{h,k}(\pi_k, x_k) = \begin{bmatrix}
\dot{u}_{k,k} & u_{k-1} & \cdots & u_{k-M-1} \\
\dot{u}_{k-1,k} & \ddots & \cdots & \ddots \\
\vdots & \ddots & \ddots & \ddots \\
\dot{u}_{k-M-2,k} & \dot{u}_{k-M-3,k} & \cdots & u_{k-1} \\
\dot{u}_{k-M-1,k} & \ddots & \cdots & \ddots \\
\vdots & \ddots & \ddots & \ddots \\
\dot{u}_{k-N-1,k} & \dot{u}_{k-N-2,k} & \cdots & u_{k-N-M} \\
0 & \ddots & \cdots & \ddots \\
0 & 0 & \ddots & \ddots \\
0 & 0 & 0 & \dot{u}_{k-N-1,k} \\
0 & 0 & 0 & 0
\end{bmatrix}
\]  
(3.25)
and

\[
U_{y,k}(\pi_k, x_k) = \begin{bmatrix}
\hat{u}_{k|k}^2 & \hat{u}_{k|k} \cdot u_{k-1} & \cdots & \hat{u}_{k|k} \cdot u_{k-M-1} & \cdots & \hat{u}_{k|k}^2 - M-1 \\
\hat{u}_{k-1|k}^2 & \hat{u}_{k-1|k} \cdot \hat{u}_{k|k} & \cdots & \hat{u}_{k-1|k} \cdot u_{k-M-2} & \cdots & \hat{u}_{k-1|k}^2 - M-2 \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\hat{u}_{k-M-2|k}^2 & \hat{u}_{k-M-2|k} \cdot \hat{u}_{k-M-1|k} & \cdots & \hat{u}_{k-M-2|k} \cdot u_{k-M-3} & \cdots & \hat{u}_{k-M-2|k}^2 - 1 \\
\hat{u}_{k-M-1|k}^2 & \hat{u}_{k-M-1|k} \cdot \hat{u}_{k-M-2|k} & \cdots & \hat{u}_{k-M-1|k} \cdot u_{k-M-4} & \cdots & \hat{u}_{k-M-1|k}^2 \\
0 & 0 & \cdots & 0 & \cdots & \hat{u}_{k-M-N-1|k}^2 \\
0 & 0 & \cdots & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & \cdots & 0
\end{bmatrix}
\]

Combining (3.11), (3.22) and (3.23) we get:

\[
\Phi(\pi_k, x_k, \theta) = \lambda \pi_k^T \pi_k + (y_k^2 + \theta^T U_k^T U_k \theta)
\]  

(3.26)

Similarly

\[
\Phi(\hat{\pi}_k, x_k, \theta) = \lambda \hat{\pi}_k^T \hat{\pi}_k + (y_k^2 + \theta^T \hat{U}_k^T \hat{U}_k \theta)
\]  

(3.27)

where

\[
\hat{U}_k = U_k(\hat{\pi}_k, x_k)
\]  

(3.28)

Inserting (3.26) and (3.27) into (3.21) we get

\[
\Delta \Phi(\pi_k, \hat{\pi}_k, x_k, \theta) = \lambda (\pi_k^T \pi_k - \hat{\pi}_k^T \hat{\pi}_k) + \theta^T (U_k^T U_k - \hat{U}_k^T \hat{U}_k) \theta
\]  

(3.29)

Since \(\pi_k\) and \(\hat{\pi}_k\) are independent of \(\theta\) in the above equation, the robustness
constraint (3.20) reduces to

\[
\lambda (\pi_k^T \pi_k - \hat{\pi}_k^T \hat{\pi}_k) + \hat{\theta} (U_k^T U_k - \hat{U}_k^T \hat{U}_k) \hat{\theta} \leq 0
\]  

(3.30)

where

\[
\hat{\theta} = \arg \max_{\theta \in \Theta} \left( \theta^T (U_k^T U_k - \hat{U}_k^T \hat{U}_k) \theta \right)
\]  

(3.31)

This optimization problem can be solved semi-analytically. The computation of \( \hat{\theta} \) is similar to the least squares problem subject to a quadratic constraint, investigated by Spjotvoll '57', Gander '79', Gander et al. '20', and the Min-Max problem considered by Lau et al. '32'. The derivation for finding \( \hat{\theta} \) is given in Appendix A.

Thus the SISORNQR algorithm is equivalent to solving the following nonlinear program:

\[
\hat{\pi}_k = \arg \min \left( \hat{\Phi}(\pi_k, r_k) \right)
\]  

(3.32)

subject to the robustness constraint described by (3.30) and (3.31).

While selecting an optimization technique to implement the SISORNQR algorithm we have to consider the fact that the robustness constraint is not necessarily differentiable and standard gradient based methods cannot be used to find the optimal solution. Some of the algorithms that can be used for such problems are given by Lau et al. '32' and Boyd and Barratt '77'.

We are interested in the robust stability of the SISORNQR algorithm described above. Before discussing this, however, let us consider a more general controller
where will use cost function constraints the same way as in the SISORNQR algorithm. We will prove that, for the more general case, the origin is an asymptotically stable steady-state for the system. Then we will show that the SISORNQR algorithm is a special case of the more general one. Therefore, the SISORNQR algorithm asymptotically stabilizes the closed-loop system.

3.4 A More General Case

3.4.1 Plant Model and Control Problem

Assume that we wish to control a discrete time, nonlinear, exponentially stable, time-invariant plant given by $\bar{p}: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$:

$$x_{k-1} = \bar{p}(x_k, u_k); \quad x_k \in \mathbb{R}^n; \quad u_k \in \mathbb{R}^m \quad (3.33)$$

The goal of the control system is to generate a sequence of inputs $\{u_k\}$ that bring the state of the system (3.33) from an initial value $x_0$ to a desired steady-state value $x_s$ when the plant $\bar{p}$ is not known exactly. It is assumed for simplicity that the process state of each time step $x_k$ is measured perfectly. As it brings the state to the origin, the controller must respect input constraints:

$$u_k \in \mathcal{U} \quad (3.34)$$

The input constraint space $\mathcal{U}$ is assumed to be closed. Enforcing state constraints of this form may lead to closed loop instability, even when the model is known perfectly. Soft state constraints may be enforced and will be discussed later.
Although the plant $\tilde{p}$ is not known perfectly, assume that it lies within a set $\Lambda$, which is the collection of all possible plants with the same dimensions:

$$\tilde{p} \in \Lambda$$  \hspace{1cm} (3.35)

Any $p \in \Lambda$ is assumed to satisfy the following conditions:

- $p$ is continuous in $x, u$

- there exists an input $u_s \in U$ such that $x_s = p(x_s, u_s)$

- $p$ is exponentially stable about the steady-state $(x_s, u_s)$. This means that for all $x_n \in \mathbb{R}^n$, there exists some $K > 0$ and $\lambda : 0 \leq \lambda < 1$ such that:

$$u_k = u_s \Rightarrow x_k - x_s \leq K \lambda^k (x_n - x_s)$$  \hspace{1cm} (3.36)

Assume that we select a nominal model $\tilde{p} \in \Lambda$ as our best guess for the true plant. We will show that it is possible to design an asymptotically stabilizing controller for the plant $\tilde{p}$ given $\tilde{p}$, $\Lambda$, $U$, $x_s$, and $u_s$.

### 3.4.2 Deviation Variable Formulation

It is convenient to recast the control problem in terms of deviations from the desired steady-state $x_s$ and input $u_s$, so that the control problem becomes one of driving a new system to the origin. Define deviation variables $\dot{x}_k$ and $\dot{u}_k$ as follows:

$$\dot{x}_k = x_k - x_s \quad \dot{u}_k = u_k - u_s$$  \hspace{1cm} (3.37)
For each \( p_i \in \Lambda \), define plant \( f_i \) as follows:

\[
f_i(\dot{x}_k, \dot{u}_k, x_s, u_s) \overset{\Delta}{=} p_i(\dot{x}_k + x_s, \dot{u}_k + u_s) - p_i(x_s, u_s); \tag{3.38}
\]

This plant evolves according to:

\[
\dot{x}_{k-1} = f_i(\dot{x}_k, \dot{u}_k, x_s, u_s) \tag{3.39}
\]

It has a fixed point at the origin:

\[
0 = f_i(0, 0, x_s, u_s) \tag{3.40}
\]

Using (3.38) we can map the elements of the set \( \Lambda \) to a new set \( \Omega \), which is defined as the collection of all possible \( f \)'s. The nominal model \( \tilde{f} \) corresponding to \( \tilde{p} \) and the true plant \( \check{f} \) corresponding to \( \check{p} \) are then both elements of \( \Omega \):

\[
\tilde{f} \in \Omega; \quad \check{f} \in \Omega \tag{3.41}
\]

The input constraint (3.34) can also be transformed to the closed space \( \check{U} \):

\[
\dot{u}_k \in \check{U} \tag{3.42}
\]

If \( p_i \) is linearized about the point \( (x_s, u_s) \), we get:

\[
f_i(\dot{x}_k, \dot{u}_k, x_s, u_s) = A(x_s, u_s)\dot{x}_k + B(x_s, u_s)\dot{u}_k \tag{3.43}
\]

\[
A(x_s, u_s) = \frac{\partial p_i}{\partial x} \big|_{x_s, u_s}, \quad B(x_s, u_s) = \frac{\partial p_i}{\partial u} \big|_{x_s, u_s} \tag{3.44}
\]
For linear systems $A$ and $B$ becomes constants and lose their dependence on the steady-state $(x_s, u_s)$.

### 3.4.3 Controller Cost Functions

MPC controllers regulate the plant state by minimizing a cost function that penalizes deviations of the state and input away from the origin. Here we use a cost function similar to that described by Meadows et al. (1995), in which a stage cost $L$ is summed over an infinite horizon. The cost function $\Phi \colon \mathbb{R}^n \times \mathbb{R}^m \times \Omega \to \mathbb{R}$ is given by:

$$
\Phi(x, \pi, f) \triangleq \sum_{j=0}^{N-1} L(z_j, r_j) + \sum_{j=N}^{\infty} L(z_j, 0)
$$

where the predicted state sequence $\{z_j\}$ satisfies:

$$
z_{j-1} = f(z_j, r_j, x_s, u_s) \quad z_0 = x
$$

and the sequence of future inputs $\{r_j\}_{N-1}^{\infty}$ is represented by the vector $\pi$:

$$
\pi \triangleq \begin{bmatrix} r_0^T & \cdots & r_{N-1}^T \end{bmatrix}^T
$$

The stage cost $L$ is assumed to have the following properties:

1. $L(0,0) = 0$

2. $L(0,0) > 0 \quad \forall (x, u) \neq (0,0)$
3. \( L(x, u) \to 0 \Rightarrow (x, u) \to (0, 0) \)

4. \( a(x^\sigma \leq L(x, u) \) and \( L(x, 0) \leq b(x^\sigma \) for some \( a, b > 0 \) and \( \sigma \geq 1 \)

5. \( L(x, u) \) is continuous in \( x, u \)

6. \( L(x, u) \to \infty \) as \( u \to \infty \)

With these properties the cost function \( \Phi \) converges for all \( f \in \Omega \) and \( x \in \mathbb{R}^n \).

To derive an upper bound, consider first the infinite sum in (3.45):

\[
\Phi_\infty = \sum_{j=1}^{\infty} L(z_j, 0) = \sum_{k=0}^{\infty} L(\dot{x}_k, 0): \quad \dot{x}_o = z_N
\]  

(3.48)

From property 4:

\[
\Phi_\infty \leq \sum_{k=0}^{\infty} b(\dot{x}_k)^\sigma
\]

(3.49)

From the exponential stability requirement 3.36:

\[
\Phi_\infty \leq \sum_{k=0}^{\infty} b(K \dot{x}_o \lambda^k)^\sigma
\]

(3.50)

\[
= bK^\sigma (\dot{x}_o)^\sigma \sum_{k=0}^{\infty} \lambda^{k\sigma}
\]

(3.51)

\[
\leq bK^\sigma (\dot{x}_o)^\sigma \sum_{k=0}^{\infty} \lambda^k
\]

(3.52)

\[
= bK^\sigma (\dot{x}_o)^\sigma \frac{\lambda}{1 - \lambda}
\]

(3.53)

\[
= bK^\sigma (z_N)^\sigma \frac{\lambda}{1 - \lambda}
\]

(3.54)

The cost function \( \Phi \) can then be bounded from above by:

\[
\Phi(x, x, f) \leq \sum_{j=0}^{N-1} L(z_j, c_j) + bK^\sigma (z_N)^\sigma \frac{\lambda}{1 - \lambda}
\]

(3.55)
As an example consider the cost function defined by:

\[ L(x, u) = x^T Q x + u^T R u; \quad Q > 0, \quad R > 0 \quad (3.56) \]

This satisfies conditions 1-6 with \( \sigma = 2, \ a = \lambda_{\text{min}}(Q), \ b = \lambda_{\text{max}}(Q), \) and \( x = (x^T x)^{1/2}. \)

Now define the *nominal model cost function* \( \hat{\Phi} \), which is the function that will be minimized by the controller:

\[ \hat{\Phi}(x, \pi) = \Phi(x, \pi, \dot{f}) \quad (3.57) \]

Also define the *plant cost function*, \( \Phi \), which measures performance of the algorithm on the true plant:

\[ \Phi(x, \pi) = \Phi(x, \pi, \dot{f}) \quad (3.58) \]

### 3.4.4 Robustly Stabilizing Algorithm

We can now define a robustly stabilizing algorithm. The RNMPC algorithm finds the input vector \( \pi* \) such that:

\[ \pi* = \arg \min(\Phi(\dot{x}_k, \pi)) \quad (3.59) \]

such that

\[ \pi \in \Pi \triangleq \{ \pi \mid v_j \in \Omega: \ J = 0, \ldots, N - 1 \} \quad (3.60) \]

\[ \Phi(\dot{x}_k, \pi, f) \leq \Phi(\dot{x}_k, \pi*, f) \quad \forall f \in \Omega \quad (3.61) \]
The first element of $\hat{\pi}_k$ is then injected into the plant:

$$u_k = \dot{v}_o + u_s$$  \hspace{1cm} (3.62)

The input $\hat{\pi}_k$, called the restriction of the input, is a shifted version of the previous optimal input $\hat{\pi}_{k-1}$. If the optimal input at time $k-1$ is given by:

$$\hat{\pi}_{k-1} = \begin{bmatrix} \dot{v}_{o}^{T} & \cdots & \dot{v}_{N-1}^{T} \end{bmatrix}$$  \hspace{1cm} (3.63)

then the restriction at time $k$ is given by:

$$\hat{\pi}_k = \begin{bmatrix} \dot{v}_{1}^{T} & \cdots & \dot{v}_{N-1}^{T} & 0^{T} \end{bmatrix}^{T}$$  \hspace{1cm} (3.64)

At the initial time step we define $\hat{\pi}_0$ as:

$$\hat{\pi}_0 = \begin{bmatrix} 0^{T} & \cdots & 0^{T} \end{bmatrix}^{T}$$  \hspace{1cm} (3.65)

The restriction $\hat{\pi}$ has the special property that it is a feasible solution of the non-linear problem (NLP) defined by equations (3.59)-(3.61). The following lemma shows that feasibility implies the existence of an optimal solution.

**Lemma:** Given a bounded initial state $x_k$, the RNMPC algorithm has an optimal solution $\hat{\pi}_k$.

**Proof:**

The Weierstrass Maximum Theorem states that a NLP has an optimal solution
if the objective is continuous and the constraint set is compact. Continuity of \( \hat{\Phi}(\hat{x}_k, \pi) \) follows from continuity of \( \hat{f} \) and \( L \). To find a compact constraint set, consider the level set \( \Gamma \) defined by:

\[
\Gamma \triangleq \{ \pi \mid \Phi(\hat{x}_k, \pi, f) \leq \Phi(\hat{x}_k, \hat{\pi}_k, f); \quad \forall f \in \Omega \} \tag{3.66}
\]

Property 6 of the stage cost \( L \) implies that:

\[
\Phi(\hat{x}_k, \pi, f) \to \infty \quad \text{as} \quad \pi \to \infty \tag{3.67}
\]

This means that \( \pi \) must be bounded if \( \pi \in \Gamma \). The set \( \Gamma \) is therefore bounded. Now consider the set \( C \) consisting of all inputs \( \pi \) belonging to \( \Gamma \) and satisfying input constraints (3.60):

\[
C \triangleq \{ \pi \mid \pi \in \Gamma \text{ and } \pi \in \Pi \} \tag{3.68}
\]

The input space \( \Pi \) is closed, therefore the set \( C \) is both closed and bounded and therefore compact. It is also nonempty since \( \hat{x}_k \in C \). The conditions of the Weierstrass Maximum Theorem are satisfied and the RNMPC algorithm therefore has an optimal solution \( \hat{\pi}_k \). QED

And so the RNMPC algorithm has both feasible and optimal solutions at each time step.
3.4.5 Robust Stability Theorem

We are ready to show that the RNMPC algorithm robustly stabilizes the plant $\tilde{f}$, even though it minimizes the nominal model cost $\tilde{\Phi}$.

\textit{Theorem:} when the input $\hat{u}_m(\hat{x}_k)$ is computed using the RNMPC algorithm, the origin is an asymptotically stable steady-state for the system:

$$\dot{\hat{x}}_{k-1} = \tilde{f}(\hat{x}_k, \hat{u}(\hat{x}_k)); \quad \dot{\hat{x}}_0 = x_0 - x_s$$  \hspace{1cm} (3.69)

with a region of attraction consisting of all $\hat{x}_0 \in \mathbb{R}^n$.

\textbf{Proof:}

First it is shown that the input and true plant state converge to the origin, and then it is shown that the origin is a stable steady-state for the closed loop system. The combination of convergence and stability gives asymptotic stability.

\textbf{Convergence:}

At each time step k we know that the restriction of the input $\hat{\tau}_k$ is a feasible solution of our algorithm. Let us define the plant cost using this input as the \textit{feasible plant cost} $\hat{\Phi}_k$:

$$\hat{\Phi}_k \triangleq \Phi(\hat{x}_k, \hat{\tau}_k)$$  \hspace{1cm} (3.70)

Once the optimal solution $\hat{\tau}_k$ is found at time step k, we can evaluate the cost
for the true plant by computing the optimal plant cost $\Phi$:

$$\dot{\Phi}_k \triangleq \Phi(\dot{x}_k, \dot{z}_k)$$  \hspace{1cm} (3.71)

Assume that we have found the optimal solution of the initial step $k=0$, given by $\hat{\pi}_o$. Let us denote the optimal open loop input sequence of time $k=0$ as follows:

$$\hat{\pi}_o \triangleq \begin{bmatrix} \dot{r}_o^T & \ldots & \dot{r}_{N-1}^T \end{bmatrix}$$  \hspace{1cm} (3.72)

At time step $k=0$ the optimal plant cost is given by:

$$\dot{\Phi}_o = \Phi(\dot{x}_o, \dot{z}_o) = \sum_{j=0}^{N-1} L(z_j, \dot{r}_j) + \sum_{j=N}^{\infty} L(z_j, 0)$$  \hspace{1cm} (3.73)

$$z_j = \tilde{f}(z_{j-1}, \dot{r}_{j-1}, x_s, u_s); \quad z_o = \dot{x}_o$$  \hspace{1cm} (3.74)

Assume that the first optimal input $\dot{r}_o$ is injected into the plant. The state at the next time step is then given by:

$$\dot{x}_1 = \tilde{f}(\dot{x}_o, \dot{r}_o, x_s, u_s)$$  \hspace{1cm} (3.75)

The feasible plant cost at time $k=1$ can be written as:

$$\dot{\Phi}_1 = \Phi(\dot{x}_1, \dot{z}_1) = \sum_{j=1}^{N} L(z_j, \dot{r}_j) + \sum_{j=N+1}^{\infty} L(z_j, 0)$$  \hspace{1cm} (3.76)

$$z_j = \tilde{f}(z_{j-1}, \dot{r}_{j-1}, x_s, u_s); \quad z_1 = \dot{x}_1$$  \hspace{1cm} (3.77)

Because they are computed using the true plant, the state sequences $\{z_j\}$ in (3.73) and (3.76) are identical. Subtract (3.73) from (3.76) to get:
\[ \Phi_1' - \Phi_o' = -L(\dot{x}_o, \dot{u}_o) \]  

(3.78)

Now assume that the optimal solution is found at time \( k = 1 \). We know that the true plant lies in the family \( \Omega \), so constraint (3.61) must be satisfied for the true plant at time \( k = 1 \). This means that the optimal plant cost cannot exceed the feasible plant cost:

\[ \Phi_1' \leq \Phi_1 \]  

(3.79)

Combine (3.78), (3.79) to get:

\[ \Phi_1' - \Phi_o' \leq -L(\dot{x}_o, \dot{u}_o) \]  

(3.80)

The same argument can be repeated at subsequent time steps to show that:

\[ \Phi_{k-1}' - \Phi_k' \leq -L(\dot{x}_k, \dot{u}_k) \quad k = 0, \infty \]  

(3.81)

Properties 1 and 2 of the stage cost \( L \) guarantee it to be non-negative, so (3.81) shows that the sequence of optimal plant costs \( \{\Phi_k\} \) is non-increasing. Property 4 of the stage cost \( L \) implies that the plant cost is bounded below by zero, therefore as \( k \to \infty \) the left hand side of (3.81) approaches zero. By property 3 of \( L \), as the left hand side of (3.81) approaches zero, the input and state deviations must converge to the origin:

\[ \dot{x}_k \to 0 \quad \text{and} \quad \dot{u}_k \to 0 \quad \text{as} \quad k \to \infty \]  

(3.82)

**Stability:**
Using the cost function bound (3.55) and the definition of the initial feasible input (3.65), the initial feasible plant cost can be bounded above by:

\[ \hat{\Phi}_o \leq b K^\sigma ( \dot{x}_o )^\sigma \frac{\lambda}{1 - \lambda} \]  

(3.83)

The constraint (3.61) ensures that the optimal plant cost does not exceed the feasible plant cost:

\[ \dot{\hat{\Phi}}_o \leq \dot{\Phi}_o \]  

(3.84)

From (3.81) the sequence of optimal plant costs is non-increasing, this implies:

\[ \dot{\Phi}_k \leq \dot{\Phi}_o \quad k = 1, \infty \]  

(3.85)

From property 4 of \( L \) we can derive a lower bound for the optimal plant cost:

\[ a( \dot{x}_k )^\sigma \leq L(\dot{x}_k, \dot{\tau}_o) \leq \dot{\Phi}_k \]  

(3.86)

Combining (3.83) through (3.86) leads to:

\[ a( \dot{x}_k )^\sigma \leq b K^\sigma ( \dot{x}_o )^\sigma \frac{\lambda}{1 - \lambda} \]  

(3.87)

\[ \dot{x}_k \leq \alpha \dot{x}_o \quad \text{with} \quad \alpha = K \left[ \frac{b \lambda}{a 1 - \lambda} \right]^{1/\sigma} \]  

(3.88)

If we restrict the initial state to:

\[ \dot{x}_o < r : \quad r = \frac{\rho}{\alpha} \]  

(3.89)
Then from (3.87) it follows that:

\[ \dot{x}_k < \rho: \quad k = 1, \infty \quad (3.90) \]

And so the closed loop system is stable. The combination of convergence and stability implies that the origin is an asymptotically stable steady-state for the RNMPC algorithm. **QED**

### 3.5 Robust Stability of the SISORNQR algorithm

Now back to our SISORNQR algorithm. Let us consider that initially at time \( k = 0 \), the output of the plant is at \( y_0 \neq 0 \) and \( x_0 \) is \( \neq 0 \). The following theorem shows that the SISORNQR algorithm asymptotically stabilizes the closed loop system even though it minimizes the nominal model cost function at each time step.

**Theorem 2.1: Robust Stability of the SISORNQR algorithm.** When the input is computed using the SISORNQR algorithm, the origin is an asymptotically stable equilibrium point of the plant for all \( y_0 \in \mathbb{R} \).

**Proof:** As stated above, we will need to show that our SISORNQR algorithm is a special case of the more general one discussed in last section. We will prove this by proving that, first, our plant representation is a special one of the general case and, second, our cost function is a special case too.
1. Plant. Apparently our plant representation is continuous in $x$ (in our SISORNQR algorithm, $x$ is output) and $u$ (in our SISORNQR algorithm, $u$ is input). Also, steady-state output is feasible, since there exist $u_s = 0 \in U$ such that $x_s = 0$. Our plant is also exponentially stable about the steady-state $(0, 0)$ since $u_k = 0 \Rightarrow x_k - x_s = 0$.

2. Cost function. It is straightforward that our cost function satisfies properties 1, 2, 3, 5, and 6 of the general case. The only thing needed to discuss is property 4. To study the cost of SISORNQR algorithm, let us look at (3.56). Again, variable $x$ in (3.56) is future output in SISORNQR algorithm and if we choose $Q = \text{diag}(1, 1, \cdots, 1)$ and $R = \text{diag}(\lambda, \lambda, \cdots, \lambda)$, then property 4 is satisfied with $\sigma = 2$, $a = 1$, $b = 1$, and $x = (x^T x)^{1/2}$.

The above discussion showed that SISORNQR algorithm is just a special case of the more general result. So when the input is computed using the SISORNQR algorithm, the origin is an asymptotically stable equilibrium point of the plant for all $y_0 \in \mathbb{R}$. QED
Chapter 4

Disturbance Rejection

In the previous chapter closed-loop stability was discussed for constrained stable plants in the absence of any disturbances. When there are disturbances, however, we can no longer guarantee feasibility of the robustness constraint or the reduction of the infinite horizon objective function to a finite one. Even if the constraints remain feasible, there may be a steady-state offset. In this thesis we propose a two-stage algorithm in order to reject constant output disturbances and obtain an offset-free control. In the first stage an optimization is done to find the steady-state input target so that the steady-state offset is minimized, while in the second stage this steady state input is used to calculate future inputs. This two-stage optimization is similar to the idea of doing a steady-state target optimization suggested by Meadows and Rawlings [36] and was used by Ralhan [49] to reject constant output disturbances for linear plants.
4.1 Plant Model and Control Problem

The plant model is now updated at every time step by adding a disturbance estimate to the model equation. Now we have:

\[
y_{k-j} = d_k + \sum_{i=1}^{M} h_{i} u_{k-j-i} + \sum_{i=1}^{M} \sum_{l=1}^{M} g_{i,l} u_{k-j-i} u_{k-j-l}, \quad \forall \ j \geq 1 \tag{4.1}
\]

\[
d_k = y_k - \sum_{i=1}^{M} h_{i} u_{k-i} - \sum_{i=1}^{M} \sum_{l=1}^{M} g_{i,l} u_{k-i} u_{k-l} \tag{4.2}
\]

The assumption here is that model error is due to a step disturbance entering at the output that will remain constant for all future time. This is the same disturbance model as used in most industrial MPC applications (Qiu and Badgwell '45). The goal of the controller is to bring the output of the system from an initial nonzero value to the origin in the presence of a constant output disturbance when the uncertainty description is given by (2.9).

4.2 Cost Functions

The robust controller considered here uses separate cost functions for the primary and secondary optimizations. The cost function for the primary optimization \( \Psi^d \) depends on the steady-state error and the secondary stage cost function \( \Phi^d \) depends on the deviation of the output and input from their steady-state targets over the whole horizon.

**Stage I Cost Function:**

\[
\Psi^d(x, y, \omega, \theta) = z_{N-M}^2 \tag{4.3}
\]
Stage II Cost Function:

\[ \Phi^d(x, y, \pi, \omega, \theta) \overset{\triangle}{=} \sum_{j=0}^{N-1} (v_j - \omega)^2 + \lambda \sum_{j=0}^{N-M} z_j^2 \]
\[ = (\pi - \pi^p)^T (\pi - \pi^p) + \lambda \left[ y^2 + \xi^T \xi \right] \] (4.4) (4.5)

where

\[ z_j = d(\theta) + \sum_{i=1}^{M} h_i v_{j-i} + \sum_{i=1}^{M} \sum_{l=1}^{M} g_{i l} v_{j-1} v_{j-1-l} \] (4.6)
\[ d(\theta) = y - \sum_{i=1}^{M} h_i v_{i-1} - \sum_{i=1}^{M} \sum_{l=1}^{M} g_{i l} v_{i-1} v_{l-1} \] (4.7)
\[ v_j = \omega \quad \forall \ j \geq N \] (4.8)
\[ \pi = \begin{bmatrix} \omega_1 & \cdots & \omega_{N-1} \end{bmatrix}^T \] (4.9)
\[ \pi^p = \begin{bmatrix} \omega_1 & \cdots & \omega \end{bmatrix}^T \] (4.10)
\[ x = \begin{bmatrix} v_{N-1} & \cdots & v_0 \end{bmatrix} \] (4.11)
\[ \xi = \begin{bmatrix} z_1 & \cdots & z_{N-M} \end{bmatrix}^T \] (4.12)
\[ \lambda > 0 \] (4.13)

In this case the input is set to a constant value $\omega$ at time $N$ and remains there for all future times. As a result the output settles down to a constant value, $z_{N-M}$, at time $(N + M)$ and remains there, where

\[ z_{N-M} = d(\theta) + \sum_{i=1}^{M} h_i \omega + \sum_{i=1}^{M} \sum_{l=1}^{M} g_{i l} \omega^2 \] (4.14)
4.3 Robust SISO NQR for Disturbance Rejection

Now we define a simplified robust linear quadratic regulator in order to introduce the robustness constraints and to explore the stability properties of the algorithm in the absence of input and output constraints.

**Definition: SISODRNQR** The two-stage SISO Disturbance rejection Robust Nonlinear Quadratic Regulator is now defined as:

**Stage I: Steady State Error Minimization**

\[
\hat{x}_k \triangleq \text{arg min} \left( \Psi^d(x_k, y_k, \omega, \theta) \right)
\]  

subject to the constraint:

\[
\Psi^d(x_k, y_k, \omega, \theta) \leq \Psi^d(x_k, y_k, \hat{x}_k, \theta) \quad \forall \theta \in \Theta
\]

where

\[
\hat{x}_k = \hat{x}_{k-1}; \hat{x}_0 = 0
\]

**Stage II: Dynamic Error Minimization**

\[
\hat{\tau}_k \triangleq \text{arg min} \left( \Phi^d(x_k, y_k, \tau_k, \hat{x}_k, \hat{\theta}) \right)
\]
subject to the constraint:

\[ \Phi^d(x_k, y_k, \hat{\pi}_k, \hat{\omega}_k, \theta) \leq \Phi^d(x_k, y_k, \hat{\pi}_k, \hat{\omega}_k, \theta) \quad \forall \theta \in \Theta \]  

(4.19)

The first element of optimal input vector \( \hat{\pi}_k \) is then injected into the plant.

The restriction of the input, \( \hat{\pi}_k \), is defined as:

\[ \hat{\pi}_k \triangleq S \hat{\pi}_{k-1} + T \hat{\omega}_k; \quad \hat{\pi}_0 = 0 \cdots 0^T \]  

(4.20)

where

\[ S = \begin{bmatrix} 0 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & 1 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix} \]

(4.21)

\[ T = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix}^T \]

(4.22)

### 4.3.1 Robustness Constraints

The robustness constraints (4.16) and (4.19) are infinite dimensional and may be difficult to implement. So we reformulate the robustness constraints as follows:

**Stage I Robustness Constraint:** As done in the Chapter 2, the robustness constraint can be reformulated as:

\[ \max_{\theta \in \Theta} \left( \Delta \Phi^d(x_k, y_k, \hat{\omega}_k, \hat{\omega}_k, \theta) \right) \leq 0 \]  

(4.23)
where

$$\Delta \Psi^d(x_k, y_k, \omega_k, \dot{\omega}_k, \theta) \triangleq \Psi^d(x_k, y_k, \omega_k, \theta) - \Psi^d(x_k, y_k, \dot{\omega}_k, \theta)$$  \hspace{1cm} (4.24)$$

The Stage I cost function can be written as:

$$\Psi^d(x_k, y_k, \omega_k, \theta) = \left[ d_k(\theta) + (I_d^T \theta_h) \omega_k + (I_d^T G I_d) \omega_k^2 \right]^2$$  \hspace{1cm} (4.25)$$

Similarly

$$\Psi^d(x_k, y_k, \dot{\omega}_k, \theta) = \left[ d_k(\theta) + (I_d^T \theta_h) \dot{\omega}_k + (I_d^T G I_d) \dot{\omega}_k^2 \right]^2$$  \hspace{1cm} (4.26)$$

where

$$I_d = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$$  \hspace{1cm} (4.27)$$

$$\theta_h = \begin{bmatrix} h_1 & h_2 & \cdots & h_M \end{bmatrix}^T$$  \hspace{1cm} (4.28)$$

and

$$G = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1M} \\ g_{21} & g_{22} & \cdots & g_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ g_{M1} & g_{M2} & \cdots & g_{MM} \end{bmatrix}$$  \hspace{1cm} (4.29)$$

Notice that

$$(I_d^T \theta_h) \omega_k = (I_d^T \omega_k) \theta_h = \begin{bmatrix} (I_d^T \omega_k) & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \theta_h \\ \theta_g \end{bmatrix} = D_1 \theta$$  \hspace{1cm} (4.30)$$
and

\[(I_d^T G I_d)^2 \omega_k = I_d^T \theta_g \omega_k^2\]

\[= \begin{bmatrix} I_d^T \omega_k^2 \end{bmatrix} \theta_g\]

\[= \begin{bmatrix} 0 \cdots 0 \ (I_d^T \omega_k^2) \end{bmatrix} \begin{bmatrix} \theta_h \\ \theta_g \end{bmatrix}\]

\[= D_2 \theta\]

where

\[I_d = \begin{bmatrix} I_1^T & I_2^T & \cdots & I_N^T \end{bmatrix}^T\]

\[\theta_g = \begin{bmatrix} g_1 \cdots g_M & g_{21} \cdots g_{NM} \end{bmatrix}^T\]

\[D_1 = \begin{bmatrix} (I_d^T \omega_k) & 0 \cdots 0 \end{bmatrix}\]

\[D_2 = \begin{bmatrix} 0 \cdots 0 \ (I_d^T \omega_k^2) \end{bmatrix}\]

\[\theta = \begin{bmatrix} \theta_h \\ \theta_g \end{bmatrix}\]

So equation (4.25) can be rewritten as:

\[\Psi^d(x_k^l, y_k^l, \omega_k, \theta) = \dot{d}_k(\theta) + (D_1 + D_2)\theta^2 \quad (4.31)\]

Similarly, we have

\[\Psi^d(x_k^l, y_k^l, \omega_k, \theta) = \left[ d_k(\theta) + (\dot{D}_1 + \dot{D}_2)\theta \right]^2 \quad (4.32)\]
where
\[
\hat{D}_1 = \begin{bmatrix}
I_d^T \hat{\omega}_k & 0 & \cdots & 0
\end{bmatrix}
\]
\[
\hat{D}_2 = \begin{bmatrix}
0 & \cdots & 0 & \begin{bmatrix} I_D^T \hat{\omega}_k \end{bmatrix}
\end{bmatrix}
\]

Now consider \( d_k(\theta) \):
\[
d_k = y_k - \sum_{i=1}^M h_i v_{k-i} - \sum_{i=1}^M \sum_{l=1}^M g_{i,l} v_{k-i} v_{k-l}
\]
\[
= y_k - x_k^T \theta_h - x_k^T G x_k
\]
\[
= y_k - \begin{bmatrix} x_k^T & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \theta_h \\
\theta_g \end{bmatrix} - x_k^T G x_k
\]
\[
= y_k - D_3 \theta - x_k^T G x_k
\] (4.36)

where
\[
D_3 = \begin{bmatrix} x_k^T & 0 & \cdots & 0 \end{bmatrix}
\] (4.37)

\( x_k^T G x_k \) can be rewritten as:
\[
x_k^T G x_k = D_4 \theta_g = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \theta_h \\
\theta_g \end{bmatrix} = D_5 \theta
\] (4.38)

where
\[
D_4 = \begin{bmatrix}
\tau_{k-1}^2 & \tau_{k-2} & \cdots & \tau_{k-M} & \tau_{k-2} & \cdots & \tau_{k-M}^2
\end{bmatrix}
\]
\[
D_5 = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix} D_4
\]
So (4.31) now becomes:

\[
\Psi^d(x_k, y_k, \omega_k, \theta) = \dot{y}_k - D_3 \theta - D_3 \theta + (D_1 + D_2) \theta^2 \\
= \dot{y}_k + D \theta^2 \\
\]

(4.39) \hspace{1cm} (4.40) \hspace{1cm} (4.41)

where

\[
D = D_1 + D_2 - D_3 - D_3 \\
\]

(4.42)

Similarly,

\[
\Psi^d(x_k, y_k, \omega_k, \theta) = \dot{y}_k + \dot{D} \theta^2 \\
\]

(4.43)

where

\[
\dot{D} = \dot{D}_1 + \dot{D}_2 - D_3 - D_3 \\
\]

(4.44)

Substituting for \( \Psi^d(x_k, y_k, \omega_k, \theta) \) and \( \Psi^d(x_k, y_k, \omega_k, \theta) \) in (4.24), we get:

\[
\Delta \Psi^d(x_k, y_k, \omega_k, \theta) = \Delta \Psi^d_1(x_k, \omega_k, \omega_k, \theta) + \Delta \Psi^d_2(x_k, y_k, \omega_k, \omega_k, \theta) \\
\]

(4.45)

where

\[
\Delta \Psi^d_1(x_k, \omega_k, \omega_k, \theta) = \theta^T \left[ (D^T D - \dot{D}^T \dot{D}) \right] \theta \\
\Delta \Psi^d_2(x_k, y_k, \omega_k, \omega_k, \theta) = 2 y_k (D - \dot{D}) \theta \\
\]

(4.46) \hspace{1cm} (4.47)

In this case it is not possible to find analytically the \( \theta \in \Theta \) for which the maximum
in the robustness constraint (4.23) is obtained. However

\[
\max_{\theta \in \Theta} \left( \Delta \Psi^d(x_k, y_k, \dot{\omega}_k, \dot{\omega}_k, \theta) \right) \leq \max_{\theta \in \Theta} \left( \Delta \Psi^d_1(x_k, \dot{\omega}_k, \dot{\omega}_k, \theta) \right) + \max_{\theta \in \Theta} \left( \Delta \Psi^d_1(x_k, y_k, \dot{\omega}_k, \dot{\omega}_k, \theta) \right)
\]

and it is possible to find the maximum of \( \Delta \Psi^d_1 \) and \( \Delta \Psi^d_1 \) semi-analytically, so we can find a conservative solution by reformulating the robustness constraint (4.23) as:

\[
\max_{\theta \in \Theta} \left( \Delta \Psi^d_1(x_k, \dot{\omega}_k, \dot{\omega}_k, \theta) \right) + \max_{\theta \in \Theta} \left( \Delta \Psi^d_1(x_k, y_k, \dot{\omega}_k, \dot{\omega}_k, \theta) \right) \leq 0
\]

\[
\Delta \Psi^d_1(x_k, \dot{\omega}_k, \dot{\omega}_k, \theta_1) + \Delta \Psi^d_1(x_k, y_k, \dot{\omega}_k, \dot{\omega}_k, \theta_2) \leq 0
\]

where

\[
\dot{\theta}_1 = \arg \max_{\theta \in \Theta} \Delta \Psi^d_1(x_k, \dot{\omega}_k, \dot{\omega}_k, \theta) \quad (4.48)
\]

\[
\dot{\theta}_2 = \arg \max_{\theta \in \Theta} \Delta \Psi^d_1(x_k, y_k, \dot{\omega}_k, \dot{\omega}_k, \theta) \quad (4.49)
\]

The computation of \( \dot{\theta}_1 \) is similar to the solution of equation (3.31). The computation of \( \dot{\theta}_2 \) is just the solution of a minimization problem with a linear objective function subject to a quadratic constraint and can found easily.

**Stage II Robustness Constraint:** As done above, the robustness constraint (4.19) can be reformulated as:

\[
\max_{\theta \in \Theta} \left( \Delta \Phi^d(x_k, y_k, \pi_k, \tilde{n}_k, \dot{\omega}_k, \theta) \right) \leq 0 \quad (4.50)
\]
where

$$\Delta \Phi^d(x_k, y_k, \hat{r}_k, \hat{\omega}_k, \theta) = \Phi^d(x_k, y_k, \hat{r}_k, \hat{\omega}_k, \theta) - \Phi^d(x_k, y_k, \bar{r}_k, \bar{\omega}_k, \theta) \quad (4.51)$$

Now the vector of future inputs, $\xi_k$, can also be written as

$$\xi_k = \dot{d}_k(\theta) d_k(\theta) \cdots d_k(\theta) \dot{ \theta}^T + \mathcal{P}^d_k \theta$$

$$= \bar{y}_k + (\mathcal{P}_k^d - \mathcal{Q}_k^d) \theta$$

$$= \bar{y}_k + \mathcal{U}_k^d \theta \quad (4.52)$$

where

$$\bar{y}_k = \begin{bmatrix} y_k \\ y_k \\ \vdots \\ y_k_{(N-M)^t} \end{bmatrix}$$

$$\mathcal{U}_k^d = \mathcal{P}_k^d - \mathcal{Q}_k^d$$

$$\mathcal{P}_k^d(\hat{r}_k, x_k, \hat{\omega}_k) = \begin{bmatrix} \mathcal{P}_{\hat{n},k}^d : \mathcal{P}_{\hat{g},k}^d \end{bmatrix}_{(N-M)^t \times (N-M)^t}$$

$$\mathcal{Q}_k^d(x_k) = \begin{bmatrix} \mathcal{Q}_{\hat{n},k}^d : \mathcal{Q}_{\hat{g},k}^d \end{bmatrix}_{(N-M)^t \times (N-M)^t}$$
\[
Q_{h,k}^d = \begin{bmatrix}
  v_{k-1} & v_{k-2} & \cdots & v_{k-M} \\
v_{k-1} & v_{k-2} & \cdots & v_{k-M} \\
  \vdots  & \vdots  & \ddots & \vdots  \\
v_{k-1} & v_{k-2} & \cdots & v_{k-M}
\end{bmatrix}_{(N-M) \times M}
\]

\[
Q_{g,k}^d = \begin{bmatrix}
  r_{k-1}^2 & r_{k-1} \ast r_{k-2} & \cdots & r_{k-1} \ast r_{k-M} & r_{k-2} \ast r_{k-1} & \cdots & r_{k-M}^2 \\
r_{k-1}^2 & r_{k-1} \ast r_{k-2} & \cdots & r_{k-1} \ast r_{k-M} & r_{k-2} \ast r_{k-1} & \cdots & r_{k-M}^2 \\
  \vdots  & \vdots  & \ddots & \vdots  & \vdots  & \ddots & \vdots  \\
r_{k-1}^2 & r_{k-1} \ast r_{k-2} & \cdots & r_{k-1} \ast r_{k-M} & r_{k-2} \ast r_{k-1} & \cdots & r_{k-M}^2
\end{bmatrix}
\]

Substituting for \(\xi_k\) in (4.5), we get

\[
\Phi^d(x_k, y_k, \bar{\pi}_k, \omega_k, \theta) = \lambda \theta^T \hat{U}_k^d T U_k^d \theta + 2\lambda \bar{y}_k^T \hat{U}_k^d \theta + \lambda y_k^2
+ (\bar{\pi}_k - \bar{\pi}_k^{sp})^T (\bar{\pi}_k - \bar{\pi}_k^{sp}) + \lambda \bar{y}_k^T \bar{y}_k
\quad (4.53)
\]

Similarly

\[
\Phi^d(x_k, y_k, \bar{\pi}_k, \omega_k, \theta) = \lambda \theta^T \hat{U}_k^d T U_k^d \theta + 2\lambda \bar{y}_k^T \hat{U}_k^d \theta + \lambda y_k^2
+ (\bar{\pi}_k - \bar{\pi}_k^{sp})^T (\bar{\pi}_k - \bar{\pi}_k^{sp}) + \lambda \bar{y}_k^T \bar{y}_k
\quad (4.54)
\]

where

\[
\hat{U}_k^d = U_k^d(\bar{\pi}_k, x_k, \omega_k)
\quad (4.55)
\]

Substituting for \(\Phi^d(x_k, y_k, \pi_k, \omega_k, \theta)\) and \(\Phi^d(x_k, y_k, \bar{\pi}_k, \omega_k, \theta)\) in equation (4.51) we
get:

\[
\Delta \Phi^d(x_k, y_k, \pi_k, \tilde{\pi}_k, \omega_k, \theta) = \Delta \Phi^d_1(\pi_k, \tilde{\pi}_k, \omega_k) + \Delta \Phi^d_\Pi(x_k, \pi_k, \tilde{\pi}_k, \omega_k, \theta) + \Delta \Phi^d_{\Pi\Pi}(x_k, y_k, \pi_k, \tilde{\pi}_k, \omega_k, \theta) \tag{4.56}
\]

where

\[
\Delta \Phi^d_1 = (\pi_k - \pi_k^*P)^T(\pi_k - \pi_k^*P) - (\tilde{\pi}_k - \pi_k^*P)^T(\tilde{\pi}_k - \pi_k^*P) \tag{4.57}
\]
\[
\Delta \Phi^d_\Pi = \lambda \theta^T \left[ U_k^\tau U_k^\tau - \tilde{U}_k^\tau \tilde{U}_k^\tau \right] \theta \tag{4.58}
\]
\[
\Delta \Phi^d_{\Pi\Pi} = 2\lambda \tilde{y}_k^T \left[ U_k^\tau - \tilde{U}_k^\tau \right] \theta \tag{4.59}
\]

Again it is not analytically possible to find the \( \theta \in \Theta \) for which the maximum in equation (4.50) is attained. However

\[
\max_{\theta \in \Theta} \left( \Delta \Phi^d \right) \leq \Delta \Phi^d_1 + \max_{\theta \in \Theta} \left( \Delta \Phi^d_\Pi \right) + \max_{\theta \in \Theta} \left( \Delta \Phi^d_{\Pi\Pi} \right) \tag{4.60}
\]

and it is possible to find the maximum of \( \Delta \Phi^d_1 \) and \( \Delta \Phi^d_{\Pi\Pi} \) semi-analytically, so we can replace the robustness constraint by the conservative approximation:

\[
\Delta \Phi^d_1 + \max_{\theta \in \Theta} \left( \Delta \Phi^d_\Pi(x_k, \pi_k, \tilde{\pi}_k, \omega_k, \theta) \right) + \max_{\theta \in \Theta} \left( \Delta \Phi^d_{\Pi\Pi}(x_k, y_k, \pi_k, \tilde{\pi}_k, \omega_k, \theta) \right) \leq 0 \tag{4.61}
\]
\[
\Delta \Phi^d_1 + \Delta \Phi^d_\Pi(x_k, \pi_k, \tilde{\pi}_k, \omega_k, \theta_3) + \Delta \Phi^d_{\Pi\Pi}(x_k, y_k, \pi_k, \tilde{\pi}_k, \omega_k, \theta_3) \leq 0 \tag{4.62}
\]

where

\[
\theta_3 = \arg \max_{\theta \in \Theta} \left( \Delta \Phi^d_1(x_k, \pi_k, \tilde{\pi}_k, \omega_k, \theta) \right) \tag{4.63}
\]
\begin{equation}
\dot{\theta}_4 = \arg \max_{\theta \in \Theta} \left( \Delta \Phi_{II}^d(x_k, y_k, \hat{\pi}_k, \hat{\pi}_k, \omega_k, \theta) \right) \tag{4.64}
\end{equation}

Thus a conservative solution for the SISODRNQR algorithm can be found by solving:

**Stage I: Steady-State Error Minimization**

\begin{equation}
\dot{\omega}_k \overset{\Delta}{=} \arg \min \left( \Psi^d(x_k, y_k, \omega_k, \dot{\theta}) \right) \tag{4.65}
\end{equation}

subject to the constraint:

\begin{equation}
\Delta \Psi_1^d(x_k, y_k, \omega_k, \dot{\omega}_k, \dot{\theta}_1) + \Delta \Psi_{II}^d(x_k, y_k, \omega_k, \dot{\omega}_k, \dot{\theta}_2) \leq 0 \tag{4.66}
\end{equation}

**Stage II: Dynamic Error Minimization**

\begin{equation}
\dot{\pi}_k \overset{\Delta}{=} \arg \min \left( \Phi^d(x_k, y_k, \pi_k, \dot{\pi}_k, \dot{\theta}) \right) \tag{4.67}
\end{equation}

subject to the constraint:

\begin{equation}
\Delta \Phi_1^d + \Delta \Phi_{II}^d(x_k, \pi_k, \dot{\pi}_k, \dot{\theta}_3) + \Delta \Phi_{III}^d(x_k, y_k, \pi_k, \dot{\pi}_k, \dot{\theta}_4) \leq 0 \tag{4.68}
\end{equation}

where \( \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3 \) and \( \dot{\theta}_4 \) are given by the equations (4.48), 4.49, 4.63 and (4.64) respectively.

The first element of optimal input vector \( \hat{\pi}_k \) is then injected into the plant. The steady-state properties of this two-stage algorithm can now be discussed.

**Lemma 4.1:** The steady-state input target computed by the stage I optimizer
asymptotically converges to a constant value for any constant output disturbance \( d \in \mathbb{R} \).

**Proof:** At the initial time step, \( k = 0 \), we don't have any feedback, so we assume that there is no disturbance i.e.

\[
d_k(\theta) = 0 \quad \forall \theta \in \Theta \tag{4.69}
\]

\[
\Rightarrow \dot{\omega}_0 = 0 \tag{4.70}
\]

At time \( k = 1 \), the output of the plant is measured and the disturbance estimate is made for each \( \theta \in \Theta \). The feasible cost function value for the plant is calculated as:

\[
\dot{\psi}_1^d = \Psi^d(\tilde{\omega}_1, \tilde{\theta}) \tag{4.71}
\]

Combining equations (4.69), (4.25), (4.17) and (4.70).

\[
\dot{\psi}_1^d = (d)^2 \tag{4.72}
\]

Let us assume that the optimal solution at time \( k = 1 \) is obtained and is given by \( \dot{\omega}_1 \). The optimal plant cost function value is now given as:

\[
\dot{\psi}_1^d = \Psi^d(\dot{\omega}_1, \tilde{\theta}) \tag{4.73}
\]

\[
= \left[ d + (\tilde{\theta}^T_I d) \dot{\omega}_1 + (I_d^T G d)(\dot{\omega}_1)^2 \right]^2 \tag{4.74}
\]

Now at time \( k = 2 \), the output value of the plant is measured and the feasible
cost function value for the plant is given by:

\[
\dot{\Psi}_2^d = \left[ d + (\bar{\theta}_n^T I_d) \dot{\omega}_2 + (I_d^T G I_d) (\dot{\omega}_2)^2 \right]^2
\]

(4.75)

\[
= \dot{\Psi}_1^d
\]

(4.76)

Since the robustness constraint is satisfied for all the plants at \( k = 2 \):

\[
\dot{\Psi}_2^d \leq \dot{\Psi}_2^d
\]

(4.77)

Combining (4.76) and (4.77):

\[
\dot{\Psi}_2^d \leq \dot{\Psi}_1^d
\]

(4.78)

The same argument can be repeated at subsequent time steps to show that:

\[
\dot{\Psi}_{k-1}^d \leq \dot{\Psi}_k^d
\]

(4.79)

The above equation shows that the sequence of optimal plant cost values \( \dot{\Psi}_k^d \) is non-increasing. The plant cost is bounded below by zero and thus has a non-negative limit. So

\[
\dot{\Psi}_{k-1}^d - \dot{\Psi}_k^d \rightarrow 0 \text{ as } k \rightarrow \infty
\]

(4.80)

If no other feasible solution can be found in future steps, apparently the steady-state input target will converge. If other feasible solutions are available, however, the implementation of our algorithm can be coded such that for a certain value of the cost function, it will return a certain solution rather than bouncing between
different solutions. In either case, we have

\[ \dot{\omega}_{k-1} \rightarrow \dot{\omega}_k \]  \hspace{1cm} (4.81)

Thus as \( k \rightarrow \infty \), the steady-state input target computed by the stage I converges i.e. \( \dot{\omega}_k \rightarrow \dot{\omega}_\infty \). QED

**Theorem 4.1: Robust Steady-State behavior of the SISODRNQR algorithm**

If the following conditions are all satisfied, the plant is guaranteed to reach a steady state and there is no steady-state offset, i.e., \( y_k \rightarrow 0 \).

- \( K \) has the same sign \( \forall \theta \in \Theta \)
- \( 2K'u < K \quad \forall \theta \in \Theta \) and \( \forall u \in U \)
- \( d \in D \)

where \( K, K' \) and \( D \) are defined as follows:

\[ K'(\theta) = \theta_k^T I_d \]
\[ K''(\theta) = I_d^T G I_d \]

\[ D = \{ d \ K'u^2 + Ku + d = 0 \text{ has solution } u \in U \ \forall \theta \in \Theta \} \]  \hspace{1cm} (4.82)

**Proof:**
Lemma 4.1 proved that the steady-state input target is guaranteed to converge to a constant value. Besides, the output disturbance is constant, therefore the plant is guaranteed to reach a steady-state. The following part of the proof will show that there is no steady-state offset.

Assume that the plant reaches a steady-state for all \( k \geq k^* \) i.e.

\[
\begin{align*}
v_k & \rightarrow \ddot{\omega}_\infty \quad \forall \ k \geq k^* \\
y_k & \rightarrow \dddot{y}_\infty \quad \forall \ k \geq k^*
\end{align*}
\]

Since the stage I optimizer has reached a steady state there does not exist any \( \ddot{\omega}_k \) such that

\[
\Psi_k^d(\ddot{\omega}_k, \theta) < \Psi_k^d(\ddot{\omega}_k, \theta) \quad \forall \ k \geq k^* \quad \text{and} \quad \forall \ \theta \in \Theta
\]

(4.83)

where

\[
\ddot{\omega}_k = \dot{\omega}_\infty
\]

(4.84)

Assume that \( \dddot{y}_\infty \neq 0 \) i.e. there is a steady-state offset. Now for some \( k > k^* \):

\[
d_k(\theta) = \dddot{y}_\infty - \theta_k^T \dddot{y}_{k-1}^* - (r_{k-1}^{\dddot{y}})_T G \dddot{y}_{k-1}^*
\]

(4.85)

where

\[
r_{k-1}^{\dddot{y}} = \begin{bmatrix} \dddot{\omega}_\infty & \dddot{\omega}_\infty & \cdots & \dddot{\omega}_\infty \end{bmatrix}^T
\]

(4.86)
So the feasible cost function for each model is:

\[
\Psi_k^d(\hat{\omega}_k, \theta) = \left[ d_k(\theta) + K^s_k \hat{\omega}_\infty + K'(\hat{\omega}_\infty)^2 \right]^2
= \left[ y^s_\infty - \theta^T v^s_k - (v^s_k)^T G v^s_k + K^s_k \hat{\omega}_\infty + K'(\hat{\omega}_\infty)^2 \right]^2
= (y^s_\infty)^2
\]  \hspace{1cm} (4.87)

Since \( 2K'u < K \) \( \forall \theta \in \Theta \) and \( \forall u \in U \), we have

\[
K'(\omega_k)^2 - K'(\hat{\omega}_\infty)^2 = K'(\hat{\omega}_\infty + \omega_k)(\omega_k - \hat{\omega}_\infty) \\
\in \left( -K(\omega_k - \hat{\omega}_\infty), K(\omega_k - \hat{\omega}_\infty) \right)
\]  \hspace{1cm} (4.88)

Define \( K_{\text{max}} \) to be \( K \) that has the maximum absolute value for any model in the uncertainty domain. Now let us choose:

\[
\omega_k = \hat{\omega}_\infty - \frac{y^s_\infty}{K_{\text{max}}}
\]  \hspace{1cm} (4.89)

The cost function with this input is:

\[
\Psi_k^d(\omega, \theta) = \left[ y^s_\infty - K^s_k \hat{\omega}_\infty - K'(\hat{\omega}_\infty)^2 + K\omega_k + K'(\omega_k)^2 \right]^2 \\
\in \left( \left[ y^s_\infty (1 - 2K/K_{\text{max}}) \right]^2, (y^s_\infty)^2 \right) \\
< \Psi_k^d(\hat{\omega}_k, \theta) \ \forall \theta \in \Theta \text{ and some } k > k^*
\]  \hspace{1cm} (4.90)

However this is a contradiction of the equation (4.83). Hence \( y^s_\infty \rightarrow 0 \) i.e. there is no offset at steady state. QED
Discussion about the three conditions of this theorem:

- $K'$ has the same sign $\forall \theta \in \Theta$.

  This is useful in (4.90) to guarantee that $(1 - 2K'/K_{\text{max}})$ is less than 1. This is also a limitation of our cost function constraint algorithms, because if the gains of some plants in the uncertainty domain are of different sign, our algorithm will not be able to find a better input that will lead to smaller cost functions for all possible plants.

- $2K'n < K \ \forall \theta \in \Theta$ and $\forall u \in U$.

  This is useful when deriving (4.88) and (4.90). Since second-order terms are used as "correction terms", their magnitudes are generally much less than the first order terms, therefore our requirement of $2K'n < K$ is reasonable.

- $d \in D$

  This guarantees the feasibility of $\omega_{\infty}$ which will make our controller offset free. Without having $d \in D$, the $\omega_k$ chosen in equation (4.89) may be infeasible. It's apparent that when the constant disturbance is large enough, no controller will be able to generate an offset-free control while still respecting input constraints.

In addition to input constraints, the plant may subject to output constraints as well. Output constraint are typically handled as hard output constraints or soft output constraints, more desirably soft output constraints. In our simulation examples, we assume no output constraints. Our algorithm can be very easily
generalized to use soft output constraints while still maintaining the robustness behavior. This can be done by using the procedures discussed by Ralhan [49]. The same proof then applies for our algorithm with few modifications.
Chapter 5

Simulation Examples

In this chapter we will apply the robust MPC algorithms developed in the previous chapters to simulate SISO plants.

Computer used in the following simulations has a PentiumII 400MHz CPU and 256MB memory. Typical CPU time required is several minutes for all 10 - 15 time steps. MATLAB built-in function “CONSTR” is used (CONSTR Finds the constrained minimum of a function of several variables. In later versions of MATLAB, this function becomes “FMINCON”).

In these examples, we assume:

\[ h_c = \begin{bmatrix} 20 \\ 50 \end{bmatrix} \]
\[ g_c = \begin{bmatrix} 1 & 0.5 \\ 0.2 & 0.5 \end{bmatrix} \]

Therefore \( \theta_c = \begin{bmatrix} 20 & 50 & 1 & 0.5 & 0.2 & 0.5 \end{bmatrix} \). Other parameters are assumed to be:
nominal model $\dot{\theta} = \theta_c$, $N = 2$, $M = 2$, $y_o = 36$, $u_o = 0.5$, input set point $u^{sp} = 0$, and output setpoint $y^{sp} = 0$.

**Note:** The above parameters are the same for the following several examples unless otherwise indicated.

### 5.1 Effect of the size of $\Theta$

Assume we have a perfect model ($\theta_p = \theta_c$ where $\theta_p$ is the true plant parameters) and study the effect of the size of uncertainty domain $\Theta$. Define

$$
\beta = \Delta \theta / \theta_c
$$

(5.1)

to be the parameter that measures the size of $\Theta$. The diagonal elements of the corresponding $W$ (as in equation 2.9) are then given by (the off-diagonal elements of $W$ are all zeroes):

$$
W(i, i) = 1 / (\Delta \theta(i))^2
$$

(5.2)

Figure (5.1) shows the calculated inputs and outputs for $\beta = 0.1$, $\beta = 1$, and $\beta = 2$. The integral squared error for these cases is 528.81 ($\beta = 0.1$), 546.77 ($\beta = 0.1$) and 576.70 ($\beta = 2$), respectively. From the figure we can see that the algorithm converges for all 3 cases when we have a perfect model, but as the uncertainty domain becomes larger and larger, the performance becomes worse and worse. This is because, as the size of $\Theta$ increases, the set feasible inputs that our controller can choose from becomes smaller and smaller, therefore performance suffers.
Figure 5.1: Effect of the size of $\Theta - \beta = 0.1 (---), \beta = 1 (-) and \beta = 2 (\cdots)$

5.2 Effect of model mismatch

Same as last example, except that $\Delta \theta = 0.5 \times \theta_c$ (i.e., $\Delta$ is fixed to be 0.5). Define

$$\gamma = \theta_p / \theta_c$$

(5.3)

to be the parameter that measures the model mismatch. Figure (5.2) shows the effect of model mismatch for $\gamma = 1, 1.2$ and 1.5, respectively. The integral squared error for these three cases are 530.4 ($\gamma = 1$), 763.7 ($\gamma = 1.2$) and 1193.4 ($\gamma = 1.5$), respectively. We can see that for fixed $\beta$, the calculated inputs are the same for different values of $\gamma$. The reason for this is, at each time step, the plant output is not really fed back to the controller and the controller works kind of “blindly”. However, because of the introduction of cost function constraint, it is guaranteed that the inputs calculated by the controller will lead the output to its set point as long as the real plant parameters are inside the uncertainty domain $\Theta$. 
5.3 Disturbance rejection

Now let us consider disturbance rejection. We use the same set of plant parameters as used in the previous examples, except that $\beta = 0.1$, $\gamma = 3$. In addition, a constant output disturbance, $d = 3$, is assumed to enter the plant at time $k = 0$.

Figure (5.3) shows the inputs/outputs of the plant. We can see that, for strong model mismatch and real plant parameters out of bound ($\beta = 0.1$ and $\gamma = 3$), the controller was not able to bring the plant to the desired set point with the presence of a large constant disturbance.

In the next example, let's change $\beta$ to $\beta = 2$ (now the real plant is at the boundary of $\Theta$). All other parameters are the same as last example (model mismatch is
still significant and the constant disturbance is still large). Figure (5.4) shows the result. This time, the controller can successfully bring the plant to the desired set point even with the presence of a large constant disturbance.

The last two examples in combination also shows that the enlargement of the uncertainty domain can increase the robustness of the controller (with the cost of possible performance loss, of course).

5.4 SISORNQR with input constraints

Here we consider a CSTR example published by Genceli and Nikolaou '25', in which the exothermal reaction $A \rightarrow B$ takes place.

$$\frac{dC_A(t)}{dt} = \frac{F}{V}(C_{Ai} - C_A(t)) - C_A(t)ke^{-\frac{E}{kT(t)}}$$  \hspace{1cm} (5.4)
\[
\frac{dT(t)}{dt} = \frac{F(T_i - T(t))}{V} - \frac{\Delta H_R}{\rho c_p V} \cdot k \cdot e^{(-E/RT(t))} C_{A}(t) - \frac{Q_s}{\rho V} (1 + u(t))
\]  
where \( u(t) \) is the manipulated input and is dimensionless. \( T(t) \) is our controlled output. The numerical values of other parameters are given in Table 1. For these values, the CSTR has three steady states and the one we chose is stable (the one with highest steady-state temperature).

**Table 1. CSTR Parameters**

<table>
<thead>
<tr>
<th>( F ) (m³/h)</th>
<th>( V ) (m³)</th>
<th>( C_{A_i} ) (mol/m³)</th>
<th>( k ) (h⁻¹)</th>
<th>( E/RT ) (K⁻¹)</th>
<th>( T_i ) (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.133</td>
<td>1.36</td>
<td>8008</td>
<td>7.0e7</td>
<td>8375</td>
<td>373.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \Delta H_R ) (J/mol)</th>
<th>( c_p ) (J/kg/K)</th>
<th>( \rho ) (kg/m³)</th>
<th>( Q_s ) (J/h)</th>
<th>( T_s ) (K)</th>
<th>( C_{A_s} ) (mol/m³)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-69.775</td>
<td>3140</td>
<td>800.8</td>
<td>1.055e8</td>
<td>547.6</td>
<td>393.2</td>
</tr>
</tbody>
</table>

In order to get a dimensionless output, we choose \( y = 100 \cdot T(t) - T_s \cdot T_s \) as the output to be used in our Volterra model. The input-output relationship
(relationship between \( u \) and \( y \)) can be identified from random step and pulse tests (in the least-squares sense) and can be described using second order Volterra series as follows:

\[
h = \begin{bmatrix}
-1 & 5.3 & 2.4 & 1.04 & 0.45 & 0.2 & 0.09 & 0.06 & 0.04 & 0.02 & 0.01 \\
-0.96 & 0 & 0 & 0 & 0 \\
-0.4 & -0.35 & 0 & 0 & 0 \\
-0.08 & -0.1 & -0.15 & 0 & 0 \\
0 & 0 & -0.05 & -0.1 & 0 \\
0 & 0 & 0 & -0.01 & -0.06
\end{bmatrix}^T
\]

\[
g = \begin{bmatrix}
0.1 & 0.1 & 0.05 & 0.04 & 0.03 & 0.02 & 0.01 & 0 & 0 \\
0.05 & 0 & 0 & 0 & 0 \\
0.02 & 0.04 & 0 & 0 & 0 \\
0.01 & 0.01 & 0.03 & 0 & 0 \\
0 & 0 & 0.01 & 0.02 & 0 \\
0 & 0 & 0 & 0.0 & 0.01
\end{bmatrix}
\]

\[
\Delta h = \begin{bmatrix}
0.1 & 0.1 & 0.05 & 0.04 & 0.03 & 0.02 & 0.01 & 0 & 0 \\
0.05 & 0 & 0 & 0 & 0 \\
0.02 & 0.04 & 0 & 0 & 0 \\
0.01 & 0.01 & 0.03 & 0 & 0 \\
0 & 0 & 0.01 & 0.02 & 0 \\
0 & 0 & 0 & 0.0 & 0.01
\end{bmatrix}
\]

\[
\Delta g = \begin{bmatrix}
0.1 & 0.1 & 0.05 & 0.04 & 0.03 & 0.02 & 0.01 & 0 & 0 \\
0.05 & 0 & 0 & 0 & 0 \\
0.02 & 0.04 & 0 & 0 & 0 \\
0.01 & 0.01 & 0.03 & 0 & 0 \\
0 & 0 & 0.01 & 0.02 & 0 \\
0 & 0 & 0 & 0.0 & 0.01
\end{bmatrix}
\]

where \( \Delta h \) and \( \Delta g \) are model uncertainty terms. The constraints are \(-0.35 \leq u(t) \leq 0.35 \) and \( du/dt \leq 0.14 \text{ } h^{-1} \). The typical settling time for this CSTR is 10 hours and for simplicity, we choose control horizon \( N = 10 \).

The 3 assumptions of Theorem 4.1 are all satisfied for this case. This is because, \( K' \in [-2.46, -2.06] \) and \( K \in [-10.06, -9.16] \) for the given plant parameters. Apparently the first two assumptions of Theorem 4.1 are satisfied. It can also be easily proved that the third assumption is satisfied by choosing \( u = \frac{-K - \sqrt{K'^2 - 4K'd}}{2K'} \).
Figure 5.5: results of SISONRQR and ENLDMC algorithms

for all $\theta \in \Theta$.

A more detailed description of this CSTR can be found in the paper written by Genceli and Nikolaou [25]. In the same paper, they presented a control scheme using nonlinear dynamic matrix control with end-condition (ENLDMC) and tested it with this CSTR.

Figure (5.5) shows a comparison of the SISONRQR algorithm with their ENLDMC algorithm for the same CSTR. Both algorithms converge and showed very similar behavior. Our SISONRQR algorithm worked well under input constraints (both input absolute constraints and rate-of-change constraints). The input plot in Figure (5.5) showed that at time $t = 0$ and time $t = 1$, both algorithms respect rate-of-change input constraint, while at time $t = 2$, both algorithms respect input absolute constraint.
**Note:** Our SISONRQR algorithm used 2-norm for cost functions. ENLDMC algorithm described in Genceli and Nikolaou's paper '25' used 1-norm instead. Which norm to choose doesn't seem to affect the stability and robustness properties of either algorithm, however it will make our comparison above qualitative instead of quantitative.
Chapter 6

Conclusions and Future Work

6.1 Summary of Contributions

The idea of using cost function constraints in MPC was first presented by Badgwell and then was used to control FIR models by Ralhan. In this thesis, the same idea was generalized to control nonlinear plants modeled by second order Volterra series. Model uncertainty is parameterized by ellipsoid bounds on the plant parameters. Another major issue studied in this thesis is rejection of constant output disturbance. It was shown in this thesis that if some reasonable conditions are satisfied, our controller is guaranteed to reach an offset-free steady state when the plant is subject to a constant output disturbance.

In chapter 3 we proved the robust stability of our SISONRQR algorithm with the absence of disturbances and input/output constraints. The origin is shown to be an asymptotically stable steady state for the system.
Meadows and Rawlings [36] have shown that when there is constant output disturbance, offset free control can be achieved if the plant reaches a steady state and if some other conditions are satisfied. However, there is no guarantee in that claim since the plant may not even reach a steady state. In chapter 4, we proved that our controller is guaranteed to reach such an offset-free steady state provided that the disturbance is within a certain region and the plant parameters satisfy some conditions.

6.2 Suggestions for Future Work

Recommendations for future work include:

- In this thesis, we only discussed plant represented by Volterra model. The same approach probably can be generalized to control plants described by other nonlinear models as well, as long as $\xi = U^T \theta$. It would also be interesting to study whether this cost function constraint idea works for other linear/nonlinear models where $\xi = U^T \theta$ does not hold.

- We showed in chapter 4 that if some conditions are satisfied, our controller is guaranteed to reach an offset-free steady state. We can study these conditions and see if they can be relaxed while maintaining the same guarantee.
Appendix A

Robustness Constraint:
Analytical Solution

Note: Appendices A & B are borrowed from Ralhan’s Master thesis [49] with some minor modifications.

In this appendix we will discuss the analytical solution of equation (3.31):

\[
\hat{\theta} = \arg \max_{\theta \in \Theta} \left( \theta^T (\tilde{U}_k^T \tilde{U}_k - \bar{U}_k^T \bar{U}_k) \theta \right)
\]  (A.1)

where

\[
\Theta \triangleq \left\{ \theta \mid (\theta - \theta_c)^T W (\theta - \theta_c) \leq 1 \right\}
\]  (A.2)

Since \( W \) is symmetric and positive definite, it can be diagonalized by a unitary
matrix,

$$W = V \Sigma V^T$$  \hspace{1cm} (A.3)  

where $\Sigma$ has eigenvalues of $W$ as its diagonal elements and the columns of $V$ are the eigenvectors of $W$. We can now transform $\Theta$ in (A.2) as follows:

$$\Theta_x \hat{=} \left\{ x \mid x^T x \leq 1 \right\}$$  \hspace{1cm} (A.4)  

where

$$x = \Sigma^{1/2} V^T (\theta - \theta_c)$$  \hspace{1cm} (A.5)  

Substituting

$$\theta = \theta_c + V \Sigma^{-1/2} x$$  \hspace{1cm} (A.6)  

in eqn (A.1), we get

$$x^* = \arg \max_{x^T x \leq 1} \Psi(x)$$  \hspace{1cm} (A.7)  

where

$$\Psi(x) = (x - b)^T A (x - b)$$  \hspace{1cm} (A.8)  

$$A = (\Sigma^{-1/2})^T V (U_k^T U_k - \hat{U}_k^T \hat{U}_k) V^T \Sigma^{-1/2}$$  \hspace{1cm} (A.9)  

$$b = -\Sigma^{1/2} V^T \theta_c$$  \hspace{1cm} (A.10)  

Let \( \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \) be the eigenvalues of \( A \), and \( w_1, w_2, \cdots w_n \) be a corresponding orthonormal set of eigenvectors with

\[
A w_i = \lambda_i w_i, \quad i = 1, \cdots, n \tag{A.11}
\]

Let \( b = \sum_{i=1}^{n} b_i w_i \) and \( x = \sum_{i=1}^{n} x_i w_i \). Then from (A.8) we get

\[
\Psi(x) = \sum_{i=1}^{n} \lambda_i (x_i - b_i)^2 \tag{A.12}
\]

The problem (A.7) can be further simplified by using the following results given by Spjotvoll \cite{57}.

**Theorem**: The point \( x^* \) at which \( \Psi \) attains its maximum value depends on \( \lambda_n = \max \lambda_i \) and there are three distinct cases:

1. If \( \lambda_n > 0 \), then the maximum is obtained at some point on the boundary \( x^T x = 1 \).

2. If \( \lambda_n \leq 0 \) and \( \sum_{i=1}^{n} b_i^2 \leq 1 \), then from (A.12) the maximum value of \( \Psi \) is 0 and it is attained if \( x_i^* = b_i, \forall i \in \{ i \mid \lambda_i < 0 \} \).

3. If \( \lambda_n \leq 0 \) and \( \sum_{i=1}^{n} b_i^2 > 1 \), then the maximum is attained for some value of \( x \) s.t. \( x^T x = 1 \).

For the cases 1, 3 of the above theorem, the problem in (A.7) reduces to

\[
x^* = \arg \max_{x^T x = 1} \Psi(x) \tag{A.13}
\]
The Lagrangian for this problem can be written as,

\[
L = (x - b)^T A (x - b) + \mu (1 - x^T x) \tag{A.14}
\]

The necessary conditions for a stationary point are:

\[
\frac{\partial L}{\partial x} = 2 Ax - 2 Ab - 2 \mu x = 0 \tag{A.15}
\]

\[
\frac{\partial L}{\partial \mu} = 1 - x^T x = 0 \tag{A.16}
\]

or

\[
Ax = \mu x + Ab \tag{A.17}
\]

\[
x^T x = 1 \tag{A.18}
\]

A proof very similar to the one given by Gander et al. \cite{20} can be used to show the following result:

**Lemma:** If \((x_1, \mu_1)\) and \((x_2, \mu_2)\) satisfy (A.15) and (A.16) and \(\mu_1 > \mu_2\), then

\[
\Psi(x_1) > \Psi(x_2) \tag{A.19}
\]

Thus we need to find \(\mu\) to solve the maximization problem (A.13). In Gander et al. \cite{20} and Lau et al. \cite{32} it is shown that (A.15) and (A.16) can be transformed to a quadratic eigenvalue problem,

\[
(A - \mu I)^2 z = (Ab)^T Ab z \tag{A.20}
\]
This quadratic eigenvalue problem can be further reduced to an ordinary eigenvalue problem by finding the eigenvalues of

\[ M = \begin{bmatrix} A & -I \\ -A b (A b)^T & A \end{bmatrix} \]  \hspace{1cm} (A.21)

The solution of the maximization problem (A.13) can now be summarized in the following theorem given by Gander et al. \'20\'.

**Theorem:** Let \( \mu^* \) be the largest eigenvalue of \( M \). Then there are two possible cases for the maximizer of (A.13):

1. If \( \mu^* \) is not an eigenvalue of \( A \), then \( x^* = (A - \mu^* I)^{-1} A b^T \).

2. If \( \mu^* \) is an eigenvalue of \( A \), then let \( \nu = (A - \mu^* I)^\dagger A b^T \), where \( \dagger \) denotes the pseudoinverse, and
   - if \( x = \nu \) satisfies (A.15) and (A.16), then \( x^* = \nu \)
   - If \( x = \nu \) satisfies (A.15) and \( \nu^T \nu < 1 \), then \( x^* = \nu + \eta \) is one of the many solutions, where \( \eta \) is the eigenvector corresponding to the eigenvalue \( \mu^* \) of \( A \) with \( \eta^T \eta = 1 - \nu^T \nu \).

Interesting discussion on similar results and their applications can be found in the articles by Marquardt \'34\', Forsythe and Golub \'66\', Ratishauer \'53\', Golub \'26\'.

Appendix B

Input Constraints

The absolute input constraints (2.12) for the SISO case can be written as:

\[
\begin{bmatrix}
I \\
-I
\end{bmatrix}
\bar{u}_k \leq \begin{bmatrix}
\bar{u}_{\text{max}} \\
-\bar{u}_{\text{cur}}
\end{bmatrix}
\]  \hspace{1cm} (B.1)

where

\[
\bar{u}_{\text{max}} = \begin{bmatrix}
u_{\text{max}} \\
\vdots \\
u_{\text{max}}
\end{bmatrix}, \quad \bar{u}_{\text{cur}} = \begin{bmatrix}u_{\text{cur}} \\
\vdots \\
u_{\text{cur}}\end{bmatrix}
\]

Similarly we can write the rate of change constraints (2.13) as:

\[
\begin{bmatrix}
\Delta I \\
-\Delta I
\end{bmatrix}
\bar{\nu}_k \leq \begin{bmatrix}
\Delta \bar{u}_{\text{max}} \\
-\Delta \bar{u}_{\text{cur}}
\end{bmatrix}
\]  \hspace{1cm} (B.2)
where
\[
\Delta \bar{u}_{\text{max}} = \begin{bmatrix}
\Delta u_{\text{max}} + u_{K-1} \\
\Delta u_{\text{max}} \\
\vdots \\
\Delta u_{\text{max}} 
\end{bmatrix}, \quad \Delta \bar{u}_{\text{dir}} = \begin{bmatrix}
\Delta u_{\text{dir}} + u_{K-1} \\
\Delta u_{\text{dir}} \\
\vdots \\
\Delta u_{\text{dir}} 
\end{bmatrix}
\]
and
\[
\Delta I = \begin{bmatrix}
I & 0 & \cdots & 0 \\
-I & I & \cdots & 0 \\
\vdots & \ddots & \ddots & \vdots \\
0 & \cdots & -I & I \\
0 & \cdots & 0 & -I 
\end{bmatrix}
\]
Combining (B.1) and (B.2) we get:
\[
D \bar{\pi} \leq d \quad (B.3)
\]
where
\[
D = \begin{bmatrix}
I \\
-I \\
\Delta I \\
-\Delta I
\end{bmatrix}, \quad d = \begin{bmatrix}
\bar{u}_{\text{max}} \\
-\bar{u}_{\text{dir}} \\
\Delta \bar{u}_{\text{max}} \\
-\Delta \bar{u}_{\text{dir}}
\end{bmatrix}.
Bibliography


