Numerical Methods for Model Predictive Control

Jing Yang

Kongens Lyngby
February 26, 2008

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

This thesis presents two numerical methods for the solutions of the unconstrained optimal control problem in model predictive control (MPC). The two methods are Control Vector Parameterization (CVP) and Dynamic Programming (DP). This thesis also presents a structured Interior-Point method for the solution of the constrained optimal control problem arising from CVP.

CVP formulates the unconstrained optimal control problem as a dense QP problem by eliminating the states. In DP, the unconstrained optimal control problem is formulated as an extended optimal control problem. The extended optimal control problem is solved by DP. The constrained optimal control problem is formulated into an inequality constrained QP. Based on Mehrotra's predictor-corrector method, the QP is solved by the Interior-Point method.

Each method discussed in this thesis is implemented in Matlab. The Matlab simulations verify the theoretical analysis of the computational time for the different methods. Based on the simulation results, we reach the following conclusion: The computational time for CVP is cubic in both the predictive horizon and the number of inputs. The computational time for DP is linear in the predictive horizon, cubic in both the number of inputs and states. The complexity is the same in terms of solving the constrained or unconstrained optimal control problem by CVP. Combining the effects of the predictive horizon, the number of inputs and the number of states, CVP is efficient for optimal control problems with relative short predictive horizons, while DP is efficient for optimal control problems with relative long predictive horizons.

The investigations of the different methods in this thesis may help others choose the efficient method to solve different optimal control problems. In addition, the

MPC toolbox developed in this thesis will be useful for forecasting and comparing the results between the CVP method and the DP method.

Acknowledgements

This work was not done alone and I am grateful to many people who have taught, inspired and encouraged me during my research. First and the foremost, I thank John Bagterp Jøgensen, my adviser, for providing me with an excellent research environment, for all his support and guidance. In addition, I would like to thank all the members of the Scientific Computing Group for their assistance and encouragement.

I also would like to thank Kai Feng and Guru Prasath for helpful discussion and for critical reading of the manuscript.

Finally, this work would not been possible without the support and encouragement from my family, my mother, my brother Quanli, and my best friend Yidi.

Contents

List of Figures

Chapter 1

Introduction

1.1 Model Predictive Control

Model predictive control (MPC) refers to a class of computer control algorithms that utilize a process model to predict the future response of a plant [14]. During the past twenty years, a great progress has been made in the industrial MPC field. Today, MPC has become the most widely implemented process control technology [12]. One of the main reasons for its application in the industry is that it can take account of physical and operational constraints, which are often associated with the industry cost. Another reason for its success is that the necessary computation can be carried out on-line with the speed of hardware increasing and optimizaiton algorithms improvement [8].

The basic idea of MPC is to compute an optimal control strategy such that outputs of the plant follow a given reference trajectory after a specified time. At sampling time k, the output of the plant, y_k, is measured. The reference trajectory r from time k to a future time, $k + N$, is known. The optimal sequence of inputs, $\{u^*\}_k^{k+N}$, and states, $\{x^*\}_k^{k+N}$, are calculated such that the output is as close as possible to the reference, and the behavior of the plant is subject to the physical and operation constraints. Afterwards the first element of the optimal input sequence is implemented in the plant. When the new output y_{k+1} is available, the prediction horizon is shifted one step forward, i.e. from $k + 1$

to $k+N+1$, and the calculations are repeated.

Figure (1.1) illustrates the flow chart of a representative MPC calculation at each control execution. The first step is to read the current values of inputs (manipulated variables, MVs) and outputs (controlled variables, CVs), i.e. y_k, from the plant. The outputs y_k then go into the second step, state estimation. This second step is to compensate for the model-plant mismatch and disturbance. An internal model is used to predict the behavior of the plant over the prediction horizon during the optimal computation. It is often the case that the internal model is not same as the plant, which will result in incorrect outputs. Therefore the internal model is adjusted to be accurate before it is used for calculations in the next step. The third step, dynamic optimization, is the critical step of the predictive control, and will be discussed heavily in this thesis. At this step, the estimated state, $\hat{x}$, together with the current input, u_{k-1}, and the reference trajectory, r, are used to compute a set of MVs and states. Since only the first element of MVs, u_0^*, is implemented in the plant, u_0^* goes to the last step. The first element of states returns to the second step for the next state estimation. At last step, the optimal input, u_0^* is sent to the plant.

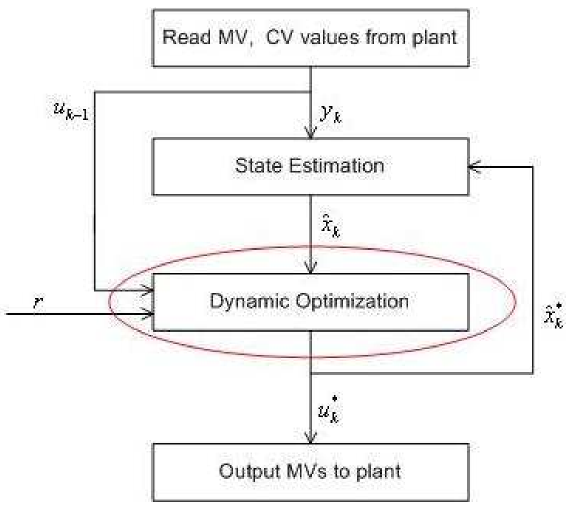

Figure 1.1: Flow chart of MPC calculation

1.2 Problem Formulation

As we mentioned before, a major advantage of MPC is its capability to solve the optimal control problem online. With the process industries developing and market competition increasing, however, the online computational cost has tended to limit MPC applications [15]. Consequently, more efficient solutions need to be developed. In recent years, many efforts have been made to simplify or/and speed up online computations.

In this thesis, we focus on numerical methods for the solution of the following optimal control problem

$$\min \ \phi = \frac{1}{2}\sum_{k=0}^{N} \|z_k - r_k\|_{Q_z}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2$$

subject to a linear state space model constraints:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k && k = 0, 1, ..., N-1 \\ z_k &= Cx_k && k = 0, 1, ..., N \end{aligned}$$

Two numerical solutions for solving this unconstrained optimal control problem are provided in this thesis. One method is the Control Vector Parameterization method (CVP) and the other is the Dynamic Programming based method (DP). The essence of both methods is to solve quadratic programs (QP). The difference between them lies in the numerical process. In CVP, the control variables over the predictive horizon are integrated as one vector. Thus the original optimal control problem is formulated as one QP with a dense Hessian matrix. All the computations of CVP are related to the dense Hessian matrix. Consequently the size of the dense Hessian matrix determines the computational time for solving the optimal control problem. DP is based on the idea of the principle of optimality. The optimal control problem is simplified into a sequence of subproblems. Each subproblem is a QP and corresponds to a stage in the predictive horizon. The QPs are solved stage-by-stage starting from the last stage. The computational time of DP is determined by the number of stages and the size of the Hessian matrix in each QP.

We also solve the above optimal control problem with input and input rate constraints

$$\begin{aligned} u_{min} \le u_k &\le u_{max} && k = 0, 1, ..., N-1 \\ \Delta u_{min} \le \Delta u_k &\le \Delta u_{max} && k = 0, 1, ..., N-1 \end{aligned}$$

The problem is transformed into an inequality constrained QP by CVP. The Interior-Point method, which is based on Mehrotra's predictor-corrector method, is employed to solve the inequality constrained QP. The optimal solution is obtained by a sequence of Newton steps with corrected search directions and step lengths. The computational time depends on the number of Newton steps and the computations in each step.

To simplify the problem, we make a few assumptions listed below. These assumptions are not valid in industrial practice, but for the development and comparison of numerical methods, they are both reasonable and useful. The assumptions are

- The internal model is an ideal model, meaning that the model is the same as the plant.
- The environment is noise free. There is no input and output disturbances and measurement noise.

  Since the internal model and the plant are matched, and no disturbances and measurement noise exist, state estimation (the second step in Figure 1.1) can be omitted from MPC computations when simulating.
- The system is time-invariant, meaning that, the system matrices A, B, C and the weight matrices Q, S are constant with respect to time.

1.3 Thesis Objective and Structure

We investigate two different methods for solving the unconstrained optimal control problem. The first method is CVP, and the second method is DP. CVP uses the model equation to eliminate states and establish a QP with a dense Hessian matrix. DP is based on the principle of optimality to solve the QP stage by stage. We also investigate the Interior-Point method for solving the constrained optimal control problem. The methods are implemented in MATLAB. Simulations are used to verify correctnesses of the implementations, and also to study effects of various factors on the computational time.

The thesis is organized as follows:

Chapter 2 presents the Control Vector Parameterization method (CVP). The unconstrained linear-quadratic (LQ) output regulation problem is formulated as a QP by removing the unknown states of the model. The solution of the QP is derived. The computational complexity of CVP is discussed at the end of the chapter.

Chapter 3 presents the Dynamic Programming based method (DP). Based on the dynamic programming algorithm, Riccati recursion procedures for both the standard and the extended LQ optimal control problem are stated. The unconstrained LQ output regulation problem is formulated as an extended LQ optimal control problem. The computational complexity of DP is estimated at the end of the chapter.

Chapter 4 presents the Interior-Point method for the constrained optimal control problem. The constrained LQ output regulation problem is formulated as an inequality constrained QP. The principle behind the Interior-Point method is illustrated by solving a simple structural inequality constrained QP. Finally the algorithm for the constrained LQ output regulation problem is developed.

Chapter 5 presents the MATLAB implementations of the methods in this thesis. The Matlab toolbox includes implementations of CVP and DP for solving the unconstrained LQ output regulation problem. It also includes the implementations of the Interior-Point method for solving the constrained LQ output regulation problem.

Chapter 6 presents the simulation results. The implementations of CVP and DP are tested on different systems. The factors that effect computational time are investigated. The implementation of the Interior-Point method is tested and its computational time for solving the constrained LQ output regulation problem is studied as well.

Chapter 7 summarizes the main conclusions of this thesis and proposes certain future directions of the project.

Chapter 2

Control Vector Parameterization

This chapter presents the Control Vector Parameterization method (CVP) for the solution of the optimal control problem, in particularly we solve the unconstrained linear quadratic (LQ) output regulation problem. CVP corresponds to state elimination such that the remaining decision variables are the manipulated variables (MVs).

2.1 Unconstrained LQ Output Regulation Problem

The formulation of the unconstrained LQ output regulation problem may be expressed by the following QP:

$$\min \phi = \frac{1}{2}\sum_{k=0}^{N} \|z_k - r_k\|_{Q_z}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 \tag{2.1}$$

subject to the following equality constraints:

$$x_{k+1} = Ax_k + Bu_k \qquad k = 0, 1, ..., N-1 \tag{2.2}$$

$$z_k = C_z x_k \qquad k = 0, 1, ..., N \tag{2.3}$$

in which $x_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^m$, $z_k \in \mathbb{R}^p$ and $\Delta u_k = u_k - u_{k-1}$.

The cost function (2.1) penalizes the deviations of the system output, z_k, from the reference, r_k. It also penalizes the changes of the input, Δu_k. The equality constraints function (2.2) is a linear discrete state space model. x_k is the state at the sampling time k, i.e. $x_k = x(k \cdot Ts)$. u_k is the manipulated variable (MV). (2.3) is the system output function where z_k is the controlled variable (CV).

Here the weight matrices Q and S are assumed to be symmetric positive semidefinite such that the quadratic program (2.1) is convex and its unique global minimizer exists.

2.2 Control Vector Parameterization

The straightforward way to solve the problem (2.1)-(2.3) is to remove all unknown states, and represent the states, x_k, and output, z_k, in terms of the initial state, x_0, and the past inputs, $\{u_i\}_{i=0}^{k-1}$ [6]. Therefore, by induction, (2.2) can be rewritten in:

$$
\begin{aligned}
x_1 &= Ax_0 + Bu_0 \\
x_2 &= Ax_1 + Bu_1 = A(Ax_0 + Bu_0) + Bu_1 \\
&= A^2x_0 + ABu_0 + Bu_1 \\
x_3 &= Ax_2 + Bu_2 = A(A^2x_0 + ABu_0 + Bu_1) + Bu_2 \\
&= A^3x_0 + A^2Bu_0 + ABu_1 + Bu_2 \\
&\vdots \\
x_k &= A^kx_0 + A^{k-1}Bu_0 + A^{k-2}Bu_1 + \ldots + ABu_{k-2} + Bu_{k-1} \\
&= A^kx_0 + \sum_{j=0}^{k-1} A^{k-1-j}Bu_j
\end{aligned} \tag{2.4}
$$

Substitute (2.4) into (2.3), then

$$\begin{aligned} z_k &= C_z x_k \\ &= C_z(A^k x_0 + \sum_{j=0}^{k-1} A^{k-1-j} B u_j) \\ &= C_z A^k x_0 + \sum_{j=0}^{k-1} C_z A^{k-1-j} B u_j \\ &= C_z A^k x_0 + \sum_{j=0}^{k-1} H_{k-j} u_j \end{aligned} \tag{2.5}$$

where $H_i = \begin{cases} 0 & i < 1 \\ C_z A^{i-1} B & i \geq 1 \end{cases}$

Having eliminated unknown states, we express the variables in stacked vectors.

The objective function (2.1) can be divided into two parts, ϕ_z and $\phi_{\Delta u}$

$$\phi_z = \frac{1}{2} \sum_{k=0}^{N} \|z_k - r_k\|_{Q_z}^2 \tag{2.6}$$

$$\phi_{\Delta u} = \frac{1}{2} \sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 \tag{2.7}$$

Since the first term of (2.6), $\frac{1}{2}\|z_0 - r_0\|_{Q_z}^2$ is constant and can not be affected by $\{u_k\}_{k=0}^{N-1}$, (2.6) is considered as:

$$\phi_z = \frac{1}{2} \sum_{k=1}^{N} \|z_k - r_k\|_{Q_z}^2 \tag{2.8}$$

To express (2.8) in stacked vectors, the stacked vectors Z, R and U are introduced as:

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_N \end{bmatrix}, \qquad R = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_N \end{bmatrix}, \qquad U = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix}$$

Then

$$\phi_z = \frac{1}{2}\|Z - R\|_{\mathcal{Q}}^2 \tag{2.9}$$

in which $\mathcal{Q} = \begin{bmatrix} Q_z & & & & \\ & Q_z & & & \\ & & Q_z & & \\ & & & \ddots & \\ & & & & Q_z \end{bmatrix}$.

Also express (2.5) in stacked vector form:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_N \end{bmatrix} = \begin{bmatrix} C_zA \\ C_zA^2 \\ C_zA^3 \\ \vdots \\ C_zA^N \end{bmatrix} x_0 + \begin{bmatrix} H_1 & 0 & 0 & \cdots & 0 \\ H_2 & H_1 & 0 & \cdots & 0 \\ H_3 & H_2 & H_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ H_N & H_{N-1} & H_{N-2} & \cdots & H_1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix}$$

and denote

$$\Phi = \begin{bmatrix} C_zA \\ C_zA^2 \\ C_zA^3 \\ \vdots \\ C_zA^N \end{bmatrix}, \quad \Gamma = \begin{bmatrix} H_1 & 0 & 0 & \cdots & 0 \\ H_2 & H_1 & 0 & \cdots & 0 \\ H_3 & H_2 & H_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ H_N & H_{N-1} & H_{N-2} & \cdots & H_1 \end{bmatrix},$$

Then

$$Z = \Phi x_0 + \Gamma U \tag{2.10}$$

Substitute (2.10) into (2.9), such that:

$$\phi_z = \frac{1}{2}\|\Gamma U - b\|_{\mathcal{Q}}^2 \qquad b = R - \Phi x_0 \tag{2.11}$$

(2.11) may be expressed as a quadratic function

$$\begin{aligned} \phi_z &= \frac{1}{2}\|\Gamma U - b\|_{\mathcal{Q}}^2 \\ &= \frac{1}{2}(\Gamma U - b)^{'}\mathcal{Q}(\Gamma U - b) \\ &= \frac{1}{2}U^{'}\Gamma^{'}\mathcal{Q}\Gamma U - (\Gamma^{'}\mathcal{Q}b)^{'}U + \frac{1}{2}b^{'}\mathcal{Q}b \end{aligned} \tag{2.12}$$

$\frac{1}{2}b^{'}\mathcal{Q}b$ can be discarded from the minimization because it has no influences on the solution.

The function $\phi_{\Delta u}$ can also be expressed as a quadratic function

$$
\begin{aligned}
\phi_{\Delta u} &= \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 \\
&= \frac{1}{2}\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}' \underbrace{\begin{bmatrix} 2S & -S & & & \\ -S & 2S & -S & & \\ & & \ddots & & \\ & & -S & 2S & -S \\ & & & -S & S \end{bmatrix}}_{H_S} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \\
&\quad + \left(- \underbrace{\begin{bmatrix} S \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{M_{u-1}} u_{-1} \right)' \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} + \frac{1}{2} u_{-1} S u_{-1} \\
&= \frac{1}{2} U' H_S U + (M_{u-1} u_{-1})' U + \frac{1}{2} u_{-1} S u_{-1} \qquad (2.13)
\end{aligned}
$$

$\frac{1}{2}u_{-1}Su_{-1}$ can be discarded from the minimization problem as it is a constant, independent of $\{u_k\}_{k=0}^{N-1}$.

Combining (2.12) with (2.13), the QP formulation of the problem (2.1)-(2.3) is:

$$
\begin{aligned}
\min\ \phi &= \phi_z + \phi_{\Delta u} \\
&= \frac{1}{2} U' \Gamma' \mathcal{Q} \Gamma U - (\Gamma' \mathcal{Q} b)' U + \frac{1}{2} b' \mathcal{Q} b \\
&\quad + \frac{1}{2} U' H_S U + (M_{u-1} u_{-1})' U + \frac{1}{2} u_{-1} S u_{-1} \\
&= \frac{1}{2} U' H U + g' U + \rho \qquad (2.14)
\end{aligned}
$$

in which the Hessian matrix is

$$H = \Gamma' \mathcal{Q} \Gamma + H_S \qquad (2.15)$$

and the gradient is

$$
\begin{aligned}
g &= -\Gamma' \mathcal{Q} b + M_{u_{-1}} u_{-1} \qquad (2.16) \\
&= -\Gamma' \mathcal{Q} (R - \Phi x_0) + M_{u_{-1}} u_{-1} \\
&= \Gamma' \mathcal{Q}\ \Phi x_0 - \Gamma' \mathcal{Q} R + M_{u_{-1}} u_{-1} \\
&= M_{x_0} x_0 + M_R R + M_{u_{-1}} u_{-1} \qquad (M_{x_0} = \Gamma' \mathcal{Q}\ \Phi,\ M_R = -\Gamma' \mathcal{Q})
\end{aligned}
$$

which is a linear function of $x_0,\ R$ and u_{-1}. And

$$\rho \quad = \quad \frac{1}{2}b^{'}Qb + \frac{1}{2}u_{-1}Su_{-1} \tag{2.17}$$

As we mentioned before, $\frac{1}{2}b^{'}Qb$ and $\frac{1}{2}u_{-1}Su_{-1}$ have no influences on the optimal solution, so we solve the unconstrained QP

$$\min_{U}\ \psi = \frac{1}{2}U'HU + g'U \tag{2.18}$$

The matrix Q and S are assumed to be positive definite, thus $\Gamma'Q\Gamma$ and H_S in (2.15) are positive definite. The Hessian matrix H is positive definite, and (2.18) has unique global minimizer. The necessary and sufficient condition for U^* being a global minimizer of (2.18) is

$$\nabla\psi = HU^* + g = 0 \tag{2.19}$$

The unique global minimizer is obtained by the solution of (2.19) :

$$\begin{aligned} U^* &= -H^{-1}g \\ &= -H^{-1}(M_{x_0}x_0 + M_R R + M_{u_{-1}}u_{-1}) \\ &= L_{x_0}x_0 + L_R R + L_{u_{-1}}u_{-1} \end{aligned} \tag{2.20}$$

in which

$$L_{x_0} = -H^{-1}M_{x_0} \tag{2.21}$$

$$L_R = -H^{-1}M_R \tag{2.22}$$

$$L_{u_{-1}} = -H^{-1}M_{u_{-1}} \tag{2.23}$$

Here the Hessian matrix H is a dense matrix. To make the computation easier, the Hessian matrix is decomposed into an upper triangular matrix and a lower triangular matrix by the Cholesky factorization. That is

$$H = LL' \tag{2.24}$$

Substitute (2.24) into (2.21)-(2.23),

$$L_{x_0} = -L'^{-1}(L^{-1}M_{x_0}) \tag{2.25}$$

$$L_R = -L'^{-1}(L^{-1}M_R) \tag{2.26}$$

$$L_{u_{-1}} = -L'^{-1}(L^{-1}M_{u_{-1}}) \tag{2.27}$$

Since the only the first element of U^* is implemented in the plant, we define the first block row of $L_{x_0},\ L_R$ and $L_{u_{-1}}$ as

$$K_{x_0} = (L_{x_0})_{1:m,1:n} \tag{2.28}$$

$$K_R = (L_R)_{1:m,1:p} \tag{2.29}$$

$$K_{u_{-1}} = (L_{u_{-1}})_{1:m,1:m} \tag{2.30}$$

Thus, the first element of U^* is given by the linear control law

$$u_0^* = K_{x_0}x_0 + K_R R + K_{u_{-1}}u_{-1} \tag{2.31}$$

2.3 Computational Complexity Analysis

In CVP, most of the computational time is spending on the Cholesky factorization of the Hessian matrix, H. From (2.14), the size of the Hessian matrix H is $mN \times mN$, N is the predictive horizon and m is the number of inputs. The Cholesky factorization for a $n \times n$ matrix costs about $n^3/3$ operations [11]. Therefore the operations to factorize the Hessian matrix are $(mN)^3/3$. The computational complexity of CVP is $\mathcal{O}(m^3N^3)$. The notation $\mathcal{O}$ describes how the input data, e.g. m and N, affect the usage of the algorithm, e.g computational time. Hence, the computational time of CVP is cubic in both the predictive horizon and the number of inputs.

Since the Hessian matrix is fixed for the unconstrained output regulation problem, the factorization of the Hessian matrix can be carried out off-line. From (2.25)-(2.30), K_{x_0}, K_R and $K_{u_{-1}}$ can also be calculated off-line. Thus the on-line computations only involve (2.31). (2.31) is simply matrix-vector computations. Therefore, the online computational time may be very short for solving unconstrained output regulation problem by CVP.

What we are concerned about, however, is the constrained output regulation problem. (2.19) is involved in the on-line computations for solving the constrained output regulation problem. The factorization of the Hessian matrix, H, is the major computation for the solution of (2.19). Therefore the factorization of the Hessian matrix dominates the on-line computational time for solving the constrained output regulation problem.

2.4 Summary

In this chapter, the unconstrained LQ output regulation problem is formulated as an unconstrained QP problem by CVP and the solution for the unconstrained QP problem is derived.

Problem: Unconstrained LQ Output Regulation

$$\begin{aligned}
\min \quad & \phi = \frac{1}{2}\sum_{k=0}^{N} \|z_k - r_k\|_{Q_z}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 && (2.32) \\
st. \quad & x_{k+1} = Ax_k + Bu_k \qquad k = 0, 1, ..., N-1 && (2.33) \\
& z_k = C_z x_k \qquad k = 0, 1, ..., N && (2.34)
\end{aligned}$$

Solution by Control Vector Parameterization:

Assume that weight matrices Q and S of (2.32) are symmetric positive semidefinite. Define:

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_N \end{bmatrix} \qquad R = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_N \end{bmatrix} \qquad U = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix} \tag{2.35}$$

$$\Phi = \begin{bmatrix} C_z A \\ C_z A^2 \\ C_z A^3 \\ \vdots \\ C_z A^N \end{bmatrix} \tag{2.36}$$

$$\mathcal{Q} = \begin{bmatrix} Q_z & & & & \\ & Q_z & & & \\ & & Q_z & & \\ & & & \ddots & \\ & & & & Q_z \end{bmatrix} \tag{2.37}$$

$$H_i = C_z A^{i-1} B \qquad i \geq 1 \tag{2.38}$$

$$\Gamma = \begin{bmatrix} H_1 & 0 & 0 & \cdots & 0 \\ H_2 & H_1 & 0 & \cdots & 0 \\ H_3 & H_2 & H_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ H_N & H_{N-1} & H_{N-2} & \cdots & H_1 \end{bmatrix} \tag{2.39}$$

$$H_s = \begin{bmatrix} 2S & -S & & & \\ -S & 2S & -S & & \\ & & \ddots & & \\ & & -S & 2S & -S \\ & & & -S & S \end{bmatrix} \tag{2.40}$$

$$M_{u_{-1}} = -\begin{bmatrix} S' & 0 & 0 & \dots & 0 \end{bmatrix}' \tag{2.41}$$

We also define

$$H = \Gamma' \mathcal{Q} \Gamma + H_S \tag{2.42}$$

$$M_{x_0} = \Gamma' \mathcal{Q} \Phi \tag{2.43}$$

$$M_R = -\Gamma' \mathcal{Q} \tag{2.44}$$

$$L_{x_0} = -H^{-1} M_{x_0} \tag{2.45}$$

$$L_R = -H^{-1} M_R \tag{2.46}$$

$$L_{u_{-1}} = -H^{-1} M_{u_{-1}} \tag{2.47}$$

The problem (2.32)-(2.34) is formulated as the unconstrained QP problem

$$\min_U \ \psi = \frac{1}{2} U' H U + g' U \tag{2.48}$$

in which

$$g = M_{x_0} x_0 + M_R R + M_{u_{-1}} u_{-1}$$

The necessary and sufficient condition for U^* being a global minimizer of (2.48) is

$$\nabla \psi = H U^* + g = 0 \tag{2.49}$$

Then the unique global minimizer U^* of (2.32)-(2.34) is:

$$U^* = L_{x_0} x_0 + L_R R + L_{u_{-1}} u_{-1} \tag{2.50}$$

The first element of U^* is

$$u_0^* = K_{x_0} x_0 + K_R R + K_{u_{-1}} u_{-1} \tag{2.51}$$

where

$$
\begin{aligned}
K_{x_0} &= (L_{x_0})_{1:m,1:n} \\
K_R &= (L_R)_{1:m,1:p} \\
K_{u_{-1}} &= (L_{u_{-1}})_{1:m,1:m}
\end{aligned}
$$

The computational complexity of CVP is $\mathcal{O}(m^3N^3)$. The computational time for CVP is cubic in both the predictive horizon and the number of the inputs.

Chapter 3

Dynamic Programming

This chapter presents the Dynamic Programming based method (DP) for the solution of the standard and extended LQ optimal control problems. We transform the unconstrained LQ output regulation problem into the extended LQ optimal control problem, so that the unconstrained LQ output regulation problem can be solved by DP.

DP solves the optimal control problem based on the principle of optimality. The idea of this principle is to simplify the optimization problem into subproblems at each stage and solve the subproblems from the last one.

3.1 Dynamic Programming

In this section, we describe the dynamic programming algorithm and the principle of optimality. This is the theoretical foundation for solving the standard and extended LQ optimal control problem. The completed dynamic programming theory may refer to [1].

3.1.1 Basic Optimal Control Problem

Consider that the optimal control problem may be expressed as the following mathematical program:

$$\min_{\{x_{k+1},u_k\}_{k=0}^{N-1}} \quad \phi = \sum_{k=0}^{N-1} g_k(x_k,u_k) + g_N(x_N) \tag{3.1}$$

$$s.t. \quad x_{k+1} = f_k(x_k,u_k) \qquad k = 0,1,...,N-1 \tag{3.2}$$

$$u_k \in \mathcal{U}_k(x_k) \qquad k = 0,1,...,N-1 \tag{3.3}$$

in which $x_k \in \mathbb{R}^n$ is the state, $u_k \in \mathbb{R}^m$ is the input, the system equation $f_k : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$, a nonempty subset $\mathcal{U}_k(x_k) \subset \mathbb{R}^m$.

The optimal solution is $\left\{x^*_{k+1}, u^*_k\right\}_{k=0}^{N-1} = \left\{x^*_{k+1}(x_0), u^*_k(x_0)\right\}_{k=0}^{N-1}$ and $\phi^* = \phi^*(x_0)$.

3.1.2 Optimal Policy and Principle of Optimality

Optimal Policy

There exists an optimal policy $\pi^* = \left\{u^*_0(x_0), u^*_1(x_1), ..., u^*_{N-1}(x_{N-1})\right\} = \{u^*_k(x_k)\}_{k=0}^{N-1}$, for the optimal control problem (3.1)-(3.3), if

$$\phi(\{x^*_k\}_{k=0}^{N}, \{u^*_k(x^*_k)\}_{k=0}^{N-1}) \leq \phi(\{x_k\}_{k=0}^{N}, \{u_k\}_{k=0}^{N-1})$$

Principle of Optimality

Let $\pi^* = \left\{u^*_0, u^*_1, ..., u^*_{N-1}\right\}$ be an optimal policy for (3.1). For the subproblem

$$\min_{\{x_{k+1},u_k\}_{k=i}^{N-1}} \sum_{k=i}^{N-1} g_k(x_k,u_k) + g_N(x_N)$$

$$s.t. \quad x_{k+1} = f_k(x_k,u_k) \qquad k = i, i+1, ..., N-1$$

$$u_k \in U_k(x_k) \qquad k = i, i+1, ..., N-1$$

the optimal policy is the truncated policy $\left\{u^*_i, u^*_{i+1}, ..., u^*_{N-1}\right\}$.

The principle of optimality implies that the optimal policy can be constructed from the last stage. For the subproblem involving the last stage, g_N, the optimal policy is $\left\{u_{N-1}^*\right\}$. When the subproblem is extended to the last two stages, $g_{N-1} + g_N$, the optimal policy will be extended to $\left\{u_{N-2}^*,\ u_{N-1}^*\right\}$. In the same way, the optimal policy can be constructed with the subproblem being extended stage by stage, until the entire problem are involved.

3.1.3 The Dynamic Programming Algorithm

The dynamic programming algorithm is based on the idea of the principle of optimality we discussed above.

Dynamic Programming Algorithm

For every initial state x_0, the optimal cost $\phi^*(x_0)$ to (3.1) is

$$\phi^*(x_0) = V_0(x_0) \tag{3.4}$$

in which the value function $V_0(x_0)$ can be computed by the recursion

$$\begin{aligned} V_N(x_N) &= g_N(x_N) && (3.5)\\ V_k(x_k) &= \min_{u_k \in \mathcal{U}_k(x_k)} g_k(x_k, u_k) + V_{k+1}(f_k(x_k, u_k)) \\ & \qquad\qquad k = N-1, N-2, ..., 1, 0 && (3.6) \end{aligned}$$

Furthermore, if $u_k^* = u_k^*(x_k)$ minimizes the right hand side of (3.6) for each x_k and k, then the policy $\pi^* = \left\{u_0^*, ..., u_{N-1}^*\right\}$ is optimal.

Figure 3.1 illustrates the process of the dynamic programming algorithm. The optimal solution of the tail subproblem $V_N(x_N)$ can be obtained immediately by solving (3.5). After that the tail subproblem $V_{N-1}(x_{N-1})$ is solved by using the solution of $V_N(x_N)$. The solution of $V_{N-1}(x_{N-1})$ is used to solve $V_{N-2}(x_{N-2})$. This process is repeated until the original problem $V_0(x_0)$ is solved.

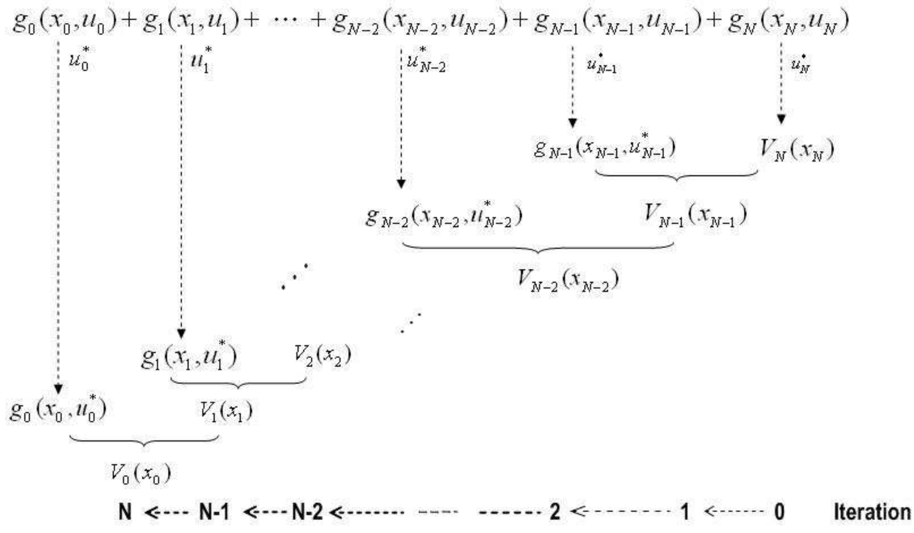

Figure 3.1: The process of the dynamic programming algorithm

3.2 The Standard and Extended LQ Optimal Control Problem

This section presents the standard and extended LQ optimal control problems, and their solutions of DP. The algorithm and principle are described in [6].

The standard LQ optimal control problem is identical with the LQ output regulation problem. The extended LQ optimal control problem extends the LQ optimal control problem by linear terms and zero order terms in its objective function, and an affine term in its dynamic equation. The extended terms are important to solve both the nonlinear optimal control problem and the constrained LQ optimal control problem.

3.2.1 The Standard LQ Optimal Control Problem and its Solution

The standard LQ optimal control problem consists of the solution for the quadratic cost function

$$\min_{\{x_{k+1},u_k\}_{k=0}^{N-1}} \quad \phi = \sum_{k=0}^{N-1} l_k(x_k, u_k) + l_N(x_N) \tag{3.7}$$

$$s.t. \quad x_{k+1} = A_k x_k + B_k u_k \quad k = 0, 1, ..., N-1 \tag{3.8}$$

with the stage costs given by

$$l_k(x_k, u_k) = \frac{1}{2}x_k' Q_k x_k + x_k' M_k u_k + \frac{1}{2}u_k' R_k u_k \quad k = 0, 1..., N-1 \tag{3.9}$$

$$l_N(x_N) = \frac{1}{2}x_N' P_N x_N \tag{3.10}$$

x_0 in (3.7) is known. The stage costs (3.9), can also be expressed as

$$\begin{aligned} l_k(x_k, u_k) &= \frac{1}{2}x_k' Q_k x_k + x_k' M_k u_k + \frac{1}{2}u_k' R_k u_k \\ &= \frac{1}{2}\begin{pmatrix} x_k \\ u_k \end{pmatrix}' \begin{pmatrix} Q_k & M_k \\ M_k' & R_k \end{pmatrix} \begin{pmatrix} x_k \\ u_k \end{pmatrix} \quad k = 0, 1, ..., N-1 \end{aligned} \tag{3.11}$$

Solution of the Standard LQ Optimal Control Problem :
Assume that the matrices

$$\begin{pmatrix} Q_k & M_k \\ M_k' & R_k \end{pmatrix} \quad k = 0, 1, ...N-1 \tag{3.12}$$

and P_N are symmetric positive semi-definite. Assume the matrices R_k, $k = 0, 1, ..., N-1$ are positive definite. Then the unique global minimizer, $\left\{x_{k+1}^*, u_k^*\right\}_{k=0}^{N-1}$, of (3.7)-(3.8) may be obtained by first computing

$$R_{e,k} = R_k + B_k' P_{k+1} B_k \tag{3.13}$$

$$K_k = -R_{e,k}^{-1}(M_k + A_k' P_{k+1} B_k)' \tag{3.14}$$

$$P_k = Q_k + A_k' P_{k+1} A_k - K_k' R_{e,k} K_k \tag{3.15}$$

for $k = N-1, N-2, ..., 1, 0$ and subsequent computation of

$$u_k^* = K_k x_k^* \tag{3.16}$$

$$x_{k+1}^* = A_k x_k^* + B_k u_k^* \tag{3.17}$$

for $k = 0, 1, ..., N-1$ with $x_0^* = x_0$. The corresponding optimal value can be computed by

$$\phi^* = \frac{1}{2}x_0' P_0 x_0 \tag{3.18}$$

3.2.2 The Extended LQ Optimal Control Problem and its Solution

The extended LQ optimal control problem consists of the solution for the quadratic cost function

$$\min_{\{x_{k+1},u_k\}_{k=0}^{N-1}} \quad \phi = \sum_{k=0}^{N-1} l_k(x_k,u_k) + l_N(x_N) \tag{3.19}$$

$$s.t. \quad x_{k+1} = A_k x_k + B_k u_k + b_k \qquad k = 0,1,...,N-1 \tag{3.20}$$

with the stage costs given by

$$l_k(x_k,u_k) = \frac{1}{2}x_k' Q_k x_k + x_k' M_k u_k + \frac{1}{2}u_k' R_k u_k + q_k' x_k + r_k' u_k + f_k \qquad k = 0,1...,N-1 \tag{3.21}$$

$$l_N(x_N) = \frac{1}{2}x_N' P_N x_N + p_N' x_N + \gamma_N \tag{3.22}$$

x_0 in (3.19) is known.

In contrast to the standard LQ optimal control problem (3.7)-(3.10), the extended LQ optimal control problem has (a) the affine terms b_k in its dynamic equation (3.20), (b) the linear terms $q_k' x_k$, $r_k' u_k$, $p_N' x_N$ and (c) the zero order terms f_k, γ_N in the stage cost functions (3.21)-(3.22).

The stage costs (3.21) can be expressed as

$$\begin{aligned} l_k(x_k,u_k) &= \frac{1}{2}x_k' Q_k x_k + x_k' M_k u_k + \frac{1}{2}u_k' Q_k u_k + q_k' + r_k' u_k + f_k \\ &= \frac{1}{2}\begin{pmatrix} x_k \\ u_k \end{pmatrix}' \begin{pmatrix} Q_k & M_k \\ M_k' & R_k \end{pmatrix} \begin{pmatrix} x_k \\ u_k \end{pmatrix} + \\ &\qquad \begin{pmatrix} q_k \\ r_k \end{pmatrix}' \begin{pmatrix} x_k \\ u_k \end{pmatrix} + f_k \end{aligned} \tag{3.23}$$

Solution of the Extended LQ Optimal Control Problem
Assume that the matrices

$$\begin{pmatrix} Q_k & M_k \\ M_k' & R_k \end{pmatrix} \qquad k = 0,1,...N-1 \tag{3.24}$$

and P_N are symmetric positive semi-definite. R_k is positive definite.

Define the sequence of matrices $\{R_{e,k}, K_k, P_k\}_{k=0}^{N-1}$ as

$$R_{e,k} = R_k + B_k^{'} P_{k+1} B_k \tag{3.25}$$
$$K_k = -R_{e,k}^{-1}(M_k + A_k^{'} P_{k+1} B_k)^{'} \tag{3.26}$$
$$P_k = Q_k + A_k^{'} P_{k+1} A_k - K_k^{'} R_{e,k} K_k \tag{3.27}$$

Define the vectors $\{c_k, d_k, a_k, p_k\}_{k=0}^{N-1}$ as

$$c_k = P_{k+1} b_k + p_{k+1} \tag{3.28}$$
$$d_k = r_k + B_k^{'} c_k \tag{3.29}$$
$$a_k = -R_{e,k}^{-1} d_k \tag{3.30}$$
$$p_k = q_k + A_k^{'} c_k + K_k^{'} d_k \tag{3.31}$$

Define the sequence of scalars $\{\gamma_k\}_{k=0}^{N-1}$ as

$$\gamma_k = \gamma_{k+1} + f_k + p_{k+1}' b_k + \frac{1}{2} b_k' P_{k+1} b_k + \frac{1}{2} d_k' a_k \tag{3.32}$$

Let x_0^* to equal x_0. Then the unique global minimizer of (3.19)-(3.20) will be obtained by the iteration

$$u_k^* = K_k x_k^* + a_k \tag{3.33}$$
$$x_{k+1}^* = A_k x_k^* + B_k u_k^* + b_k \tag{3.34}$$

The corresponding optimal value can be computed by

$$\phi^* = \frac{1}{2} x_0' P_0 x_0 + P_0^{'} x_0 + \gamma_0 \tag{3.35}$$

[6] provides the complete proofs for the solutions of both the standard and the extended LQ optimal control problem.

3.2.3 Algorithm for Solution of the Extended LQ Optimal Control Problem

To make the computations easier for solving the extended LQ optimal control problem, the matrices $R_{e,k}$ of (3.25) are factorized into two matrices by the Cholesky factorization: the lower triangular matrices and the upper triangular matrices. The operations on triangle matrices are much easier than that on the original matrices $R_{e,k}$. Hence, we obtain the following corollary.

Corollary
Assume the matrices

$$\begin{pmatrix} Q_k & M_k \\ M_k' & R_k \end{pmatrix} \qquad k = 0, 1, ...N-1 \tag{3.36}$$

and P_N are symmetric positive semi-definite. R_k is positive definite. Let $\{R_{e,k}, K_k, P_k\}_{k=0}^{N-1}$ and $\{c_k, d_k, a_k, p_k\}_{k=0}^{N-1}$ be defined as (3.25) to (3.31). Then $R_{e,k}$ is positive definite and has the Cholesky factorization

$$R_{e,k} = L_k L_k' \tag{3.37}$$

in which L_k is a non-singular lower triangular matrix.

Moreover, define

$$Y_k = (M_k + A_k' P_{k+1} B_k)' \tag{3.38}$$

and

$$Z_k = L_k^{-1} Y_k \tag{3.39}$$

$$z_k = L_k^{-1} d_k \tag{3.40}$$

Then

$$P_k = Q_k + A_k' P_{k+1} A_k - Z_k' Z_k \tag{3.41}$$

$$p_k = q_k + A_k' c_k - Z_k' z_k \tag{3.42}$$

and $u_k = K_k x_k + a_k$ may be computed by

$$u_k = -(L_k')^{-1}(Z_k x_k + z_k) \tag{3.43}$$

Algorithm 1

Algorithm 1 provides the major steps in factorizing and solving the extended LQ optimal problem (3.19)-(3.20).

Algorithm 1: Solution of the extended LQ optimal control problem.

Require: $N, (P_N, p_N, \gamma_N), \{Q_k, M_k, R_k, q_k, f_k, r_k, A_k, B_k, b_k\}_{k=0}^{N-1}$ and x_0.

Assign $P \leftarrow P_N, p \leftarrow p_N$ and $\gamma \leftarrow \gamma_N$.

for $k = N-1 : -1 : 0$ **do**

Compute the temporary matrices and vectors

$$R_e = R_k + B_k^{'} P B_k$$
$$S = A_k^{'} P$$
$$Y = (M_k + S B_k)^{'}$$
$$s = P b_k$$
$$c = s + p$$
$$d = r_k + B_k^{'} c$$

Cholesky factorize R_e

$$R_e = L_k L_k^{'}$$

Compute Z_k and z_k by solving

$$L_k Z_k = Y$$
$$L_k z_k = d$$

Update P, γ, and p by

$$P \leftarrow Q_k + S A_k - Z_k^{'} Z_k$$
$$\gamma \leftarrow \gamma + f_k + p^{'} b_k + \tfrac{1}{2} s^{'} b_k - \tfrac{1}{2} z_k^{'} z_k$$
$$p \leftarrow q_k + A_k^{'} c - Z_k^{'} z_k$$

end for

Compute the optimal value by

$$\phi = \tfrac{1}{2} x_0^{'} P x_0 + p^{'} x_0 + \gamma$$

for $k = 0 : 1 : N-1$ **do**

Compute

$$y = Z_k x_k + z_k$$

and solve the linear system of equations

$$L_k^{'} u_k = -y$$

for u_k.

Compute

$$x_{k+1} = A_k x_k + B_k u_k + b_k.$$

end for

Return $\{x_{k+1}, u_k\}_{k=0}^{N-1}$ and ϕ.

In some practical situations, the matrices $\{Q_k, M_k, R_k, A_k, B_k\}_{k=0}^{N-1}, P_N$ are fixed, while the vectors $(x_0, \{q_k, f_k, r_k, b_k\}_{k=0}^{N-1}, \{p_N, \gamma_N\})$ are altered. Algorithm 1 can be separated into a factorization part and a solution part. The factorization part, which is stated in Algorithm 2, is to compute $\{P_k, L_k, Z_k\}_{k=0}^{N-1}$ for the fixed matrices. The solution part, which is stated in Algorithm 3, is to solve the extended LQ optimal control problem based on the given $\{P_k, L_k, Z_k\}_{k=0}^{N-1}$ and $(x_0, \{q_k, f_k, r_k, b_k\}_{k=0}^{N-1}, \{p_N, \gamma_N\})$.

The unconstrained LQ output regulation problem (2.1)-(2.3) is an instance of the extended LQ optimal control problem with unaltered $\{Q_k, M_k, R_k, A_k, B_k\}_{k=0}^{N-1}$.

Algorithm 2: Factorization for the extended LQ optimal control problem.

Require: N, P_N, and $\{Q_k, M_k, R_k, A_k, B_k\}_{k=0}^{N-1}$.

for $k = N-1 : -1 : 0$ **do**

Compute the temporary matrices

$$
\begin{aligned}
&R_e = R_k + B_k' P_{k+1} B_k \\
&S = A_k' P_{k+1} \\
&Y = (M_k + S B_k)'
\end{aligned}
$$

Cholesky factorize R_e

$$R_e = L_k L_k'$$

Compute Z_k by solving

$$L_k Z_k = Y$$

Compute

$$P_k = Q_k + S A_k - Z_k' Z_k$$

end for

Return $\{P_k, L_k, Z_k\}_{k=0}^{N-1}$.

Algorithm 3: Solve a factorized extended LQ optimal control problem.

Require: $N, (P_N, p_N, \gamma_N), \{Q_k, M_k, R_k, q_k, f_k, r_k, A_k, B_k, b_k\}_{k=0}^{N-1}$, x_0 and $\{P_k, L_k, Z_k\}_{k=0}^{N-1}$.

Assign $p \leftarrow p_N$ and $\gamma \leftarrow \gamma_N$.

for $k = N-1 : -1 : 0$ **do**

Compute the temporary vectors

$$\begin{aligned} s &= P_{k+1} b_k \\ c &= s + p \\ d &= r_k + B_k^{'} c \end{aligned}$$

Solve the lower triangular system of equations

$$L_k z_k = d$$

for z_k.

Update γ and p by

$$\begin{aligned} \gamma &\leftarrow \gamma + f_k + p^{'} b_k + \tfrac{1}{2} s^{'} b_k - \tfrac{1}{2} z_k^{'} z_k \\ p &\leftarrow q_k + A_k^{'} c - Z_k^{'} z_k \end{aligned}$$

end for

Compute the optimal value by

$$\phi = \tfrac{1}{2} x_0^{'} P x_0 + p^{'} x_0 + \gamma$$

for $k = 0 : 1 : N-1$ **do**

Compute

$$y = Z_k x_k + z_k$$

and solve the upper triangular system of equations

$$L_k^{'} u_k = -y$$

for u_k.

Compute

$$x_{k+1} = A_k x_k + B_k u_k + b_k$$

end for

Return $\{x_{k+1}, u_k\}_{k=0}^{N-1}$ and ϕ.

3.3 Unconstrained LQ Output Regulation Problem

In this section, the unconstrained LQ output regulation problem is transformed into the extended LQ optimal control problem, so that it can be solved by DP.

The formulation of the unconstrained LQ output regulation problem is

$$\min \quad \phi = \frac{1}{2}\sum_{k=0}^{N} \|z_k - r_k\|_{Q_z}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 \tag{3.44}$$

$$s.t. \quad x_{k+1} = Ax_k + Bu_k \qquad k = 0, 1, ..., N-1 \tag{3.45}$$

$$z_k = C_z x_k \qquad k = 0, 1, ..., N \tag{3.46}$$

The objective function of (3.44) can be expressed by:

$$\begin{aligned} \phi &= \frac{1}{2}\sum_{k=0}^{N} \|z_k - r_k\|_{Q_z}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 \\ &= \frac{1}{2}\sum_{k=0}^{N-1} \|z_k - r_k\|_{Q_z}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 + \frac{1}{2}\|z_N - r_N\|_{Q_z}^2 \\ &= \frac{1}{2}\sum_{k=0}^{N-1} (\|z_k - r_k\|_{Q_z}^2 + \|\Delta u_k\|_S^2) + \frac{1}{2}\|z_N - r_N\|_{Q_z}^2 \end{aligned} \tag{3.47}$$

In contrast to the extended LQ optimal control problem, the stage costs will be,

$$l_k(x_k, u_k) = \frac{1}{2}\left(\|z_k - r_k\|_{Q_z}^2 + \|\Delta u_k\|_S^2\right) \qquad k = 0, 1, ..., N-1 \tag{3.48}$$

$$l_N(x_N) = \frac{1}{2}\|z_N - r_N\|_{Q_z}^2 \tag{3.49}$$

Since $\Delta u_k = u_k - u_{k-1}$, (3.48) is related to both u_k and u_{k-1}. We reconstructe the state vector as

$$\bar{x}_k = \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} \tag{3.50}$$

Then the dynamic equation (3.45) becomes:

$$\begin{aligned} \bar{x}_{k+1} &= \begin{bmatrix} x_{k+1} \\ u_k \end{bmatrix} = \begin{bmatrix} Ax_k + Bu_k \\ u_k \end{bmatrix} \\ &= \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} + \begin{bmatrix} B \\ I \end{bmatrix} u_k \\ &= \bar{A}\bar{x}_k + \bar{B}u_k + \bar{b} \end{aligned} \tag{3.51}$$

where $\bar{A} = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}$, $\bar{B} = \begin{bmatrix} B \\ I \end{bmatrix}$, $\bar{b} = \mathbf{0}$.

Combined with (3.46), the stage cost (3.48) becomes,

$$\begin{aligned}
l_k(x_k, u_k) &= \frac{1}{2}\left(\|z_k - r_k\|_{Q_z}^2 + \|\Delta u_k\|_S^2\right) \qquad (3.52)\\
&= \frac{1}{2}[(Cx_k - r_k)^{'} Q_z (Cx_k - r_k) + (u_k - u_{k-1})^{'} S (u_k - u_{k-1})]\\
&= \frac{1}{2}x_k' C^{'} Q_z C x_k - r_k' Q_z C x_k + \frac{1}{2} r_k' Q_z r_k + \frac{1}{2} u_k' S u_k - u_{k-1}' S u_k\\
&\quad + \frac{1}{2} u_{k-1}' S u_{k-1}\\
&= \frac{1}{2}\begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}^{'} \begin{bmatrix} C^{'} Q_z C & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} + \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}^{'} \begin{bmatrix} 0 \\ -S \end{bmatrix} u_k\\
&\quad + \frac{1}{2} u_k' S u_k + \begin{bmatrix} -C^{'} Q_z r_k \\ 0 \end{bmatrix}^{'} \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} + \frac{1}{2} r_k' Q_z r_k\\
&= \frac{1}{2}\bar{x}_k' \bar{Q} \bar{x}_k + \bar{x}_k \bar{M} u_k + \frac{1}{2} u_k' \bar{R} u_k + \bar{q}_k' \bar{x}_k + \bar{r}_k' u_k + \bar{f}_k
\end{aligned}$$

where
$\bar{Q} = \begin{bmatrix} C^{'} Q_z C & 0 \\ 0 & S \end{bmatrix}$, $\bar{M} = \begin{bmatrix} 0 \\ -S \end{bmatrix}$, $\bar{R} = S$, $\bar{q}_k = \begin{bmatrix} -C^{'} Q_z r_k \\ 0 \end{bmatrix}$, $\bar{r}_k = 0$
$\bar{f}_k = \frac{1}{2} r_k' Q_z r_k$.

And the final stage cost (3.49) is,

$$\begin{aligned}
l_N(x_N) &= \frac{1}{2}\|z_N - r_N\|_{Q_z}^2 \qquad (3.53)\\
&= \frac{1}{2}(Cx_N - r_N)^{'} Q_z (Cx_N - r_N)\\
&= \frac{1}{2}\left(x_N' C' Q_z C x_N - 2 r_N' Q_z C x_N + r_N' Q_z r_N\right)\\
&= \frac{1}{2}\begin{bmatrix} x_N \\ u_{N-1} \end{bmatrix}^{'} \begin{bmatrix} C^{'} Q_z C & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_N \\ u_{N-1} \end{bmatrix} + \begin{bmatrix} C^{'} Q_z r_N \\ 0 \end{bmatrix}^{'} \begin{bmatrix} x_N \\ u_{N-1} \end{bmatrix}\\
&\quad + \frac{1}{2} r_N' Q_z r_N\\
&= \frac{1}{2}\bar{x}_N' \bar{P}_N \bar{x}_N + \bar{p}_N' \bar{x}_N + \bar{\gamma}_N
\end{aligned}$$

where
$\bar{P}_N = \begin{bmatrix} C' Q_z C & 0 \\ 0 & 0 \end{bmatrix}$, $\bar{p}_N = \begin{bmatrix} -C' Q_z r_N \\ 0 \end{bmatrix}$, $\bar{\gamma}_N = \frac{1}{2} r_N' Q_z r_N$.

Thus far, the unconstrained LQ output regulation problem (3.44)-(3.46) is formulated as the extended LQ optimal control problem

$$\min_{\{\bar{x}_{k+1},u_k\}_{k=0}^{N-1}} \quad \phi = \sum_{k=0}^{N-1} l_k(\bar{x}_k, u_k) + l_N(\bar{x}_N) \tag{3.54}$$

$$s.t. \quad \bar{x}_{k+1} = \bar{A}\bar{x}_k + \bar{B}u_k + \bar{b} \qquad k = 0, 1, ..., N-1 \tag{3.55}$$

with the stage costs given by

$$l_k(\bar{x}_k, u_k) = \frac{1}{2}\bar{x}_k'\bar{Q}\bar{x}_k + \bar{x}_k\bar{M}u_k + \frac{1}{2}u_k'\bar{R}u_k + \bar{q}_k'\bar{x}_k + \bar{r}_k'u_k + \bar{f}_k \tag{3.56}$$

$$l_N(\bar{x}_N) = \frac{1}{2}\bar{x}_N'\bar{P}_N\bar{x}_N + \bar{p}_N'\bar{x}_N + \bar{\gamma}_N \tag{3.57}$$

where

$$\bar{x}_k = \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix},$$

$$\bar{A} = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}, \qquad \bar{B} = \begin{bmatrix} B, \\ I \end{bmatrix}, \qquad \bar{b} = \mathbf{0}$$

$$\bar{Q} = \begin{bmatrix} C^{'}Q_zC & 0 \\ 0 & S \end{bmatrix}, \quad \bar{M} = \begin{bmatrix} 0 \\ -S \end{bmatrix}, \qquad \bar{R} = S,$$

$$\bar{q}_k = \begin{bmatrix} -C^{'}Q_zr_k \\ 0 \end{bmatrix}, \qquad \bar{r}_k = 0, \qquad \bar{f}_k = \tfrac{1}{2}r_k'Q_zr_k,$$

$$\bar{P}_N = \begin{bmatrix} C'Q_zC & 0 \\ 0 & 0 \end{bmatrix}, \quad \bar{p}_N = \begin{bmatrix} -C'Q_zr_N \\ 0 \end{bmatrix}, \quad \bar{\gamma}_N = \tfrac{1}{2}r_N'Q_zr_N$$

This extended LQ optimal control problem can be solved by DP introduced in previous section.

Note that the matrices $\bar{A}, \bar{B}, \bar{Q}, \bar{M}, \bar{R}, \bar{P}_N$ depend on the matrices A, B, C, Q, S. Since the matrices A, B, C, Q, S are fixed over time in a time-invariant system, $\bar{A}, \bar{B}, \bar{Q}, \bar{M}, \bar{R}, \bar{P}_N$ are fixed as well. Thus the matrices $\bar{A}, \bar{B}, \bar{Q}, \bar{M}, \bar{R}, \bar{P}_N$ can be computed offline. Because $\bar{q}, \bar{f}, \bar{p}_N, \bar{\gamma}_N$ depend on the reference r_k, which may change in terms of k, $\bar{q}, \bar{f}, \bar{p}_N, \bar{\gamma}_N$ change in terms of k as well. Therefore, $\bar{q}, \bar{f}, \bar{p}_N, \bar{\gamma}_N$ have to be computed online.

3.4 Computational Complexity Analysis

In DP, the major computational step is to compute $\bar{A}'_k P_{k+1} \bar{A}_k$ of the Riccuti equation

$$P_k = \bar{Q}_k + \bar{A}'_k P_{k+1} \bar{A}_k - K'_k R_{e,k} K_k \tag{3.58}$$

in which both $\bar{A}_k$ and P_k are $(n+m) \times (n+m)$ matrices, m is the number of inputs and n is the number of states

For a matrix multiplication operation $C = AB$, where the size of the matrices A, B and C is $a \times b, b \times c$ and $a \times c$, each element of the matrix C involves b muliplication operations. Thus, the whole matrix C involves $b \cdot (a \cdot c)$ muliplication operations.

Therefore the computational cost of $\bar{A}'_k P_{k+1} \bar{A}_k$ is

$$\begin{aligned} & 2(n+m)^3 \\ = \;& \mathcal{O}(n^3 + m^3) \end{aligned} \tag{3.59}$$

As k in P_k is from 1 to N, where N is the predictive horizon, the computational complexity of DP is $\mathcal{O}(N \cdot (n^3 + m^3))$. In other words, the computational time for DP is linear in the predictive horizon and cubic in both the number of states and the number of inputs.

3.5 Summary

In the light of the principle of optimality, the optimal control problem can be solved by DP. DP starts with solving the tail subproblem involving the last stage. The second step is to solve the tail subproblem involving the last two stages. DP continues in this way until the solution of the original problem is obtained.

The standard and extended LQ optimal control problem are formulated and their solutions are derived. Algorithm 1 provides the major steps in factorizing and solving the extended LQ optimal control problem. In some situations, part of matrices are fixed. To avoid unecessary computations performing on the fixed matrices, Algorithm 2 is separated from Algorithm 1 to factorize the fixed matrices. Algorithm 3 finishes the rest steps in Algorithm 1.

Finally, the unconstrained LQ output regulation problem is transformed into the extended LQ optimal control problem, which can be solved by DP.

The unconstrained LQ output regulation problem is:

$$\min \quad \phi = \frac{1}{2}\sum_{k=0}^{N} \|z_k - r_k\|_{Q_z}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 \tag{3.60}$$

$$s.t. \quad x_{k+1} = Ax_k + Bu_k \qquad k = 0, 1, ..., N-1 \tag{3.61}$$

$$z_k = C_z x_k \qquad k = 0, 1, ..., N \tag{3.62}$$

Define

$$\bar{x}_k = \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix} \tag{3.63}$$

Formulate the problem (3.60)-(3.62) as the extended LQ optimal control problem, we have

$$\min_{\{\bar{x}_{k+1}, u_k\}_{k=0}^{N-1}} \quad \phi = \sum_{k=0}^{N-1} l_k(\bar{x}_k, u_k) + l_N(\bar{x}_N) \tag{3.64}$$

$$s.t. \quad \bar{x}_{k+1} = \bar{A}\bar{x}_k + \bar{B}u_k + \bar{b}_k \qquad k = 0, 1, ..., N-1 \tag{3.65}$$

with the stage costs given by

$$l_k(\bar{x}_k, u_k) = \frac{1}{2}\bar{x}_k'\bar{Q}\bar{x}_k + \bar{x}_k'\bar{M}u_k + \frac{1}{2}u_k'\bar{R}u_k + \bar{q}_k'\bar{x}_k + \bar{r}_k'u_k + \bar{f}_k$$
$$k = 0, 1..., N-1 \tag{3.66}$$

$$l_N(\bar{x}_N) = \frac{1}{2}\bar{x}_N'\bar{P}_N\bar{x}_N + \bar{p}_N'\bar{x}_N + \bar{\gamma}_N \tag{3.67}$$

in which

$$\bar{A} = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \tag{3.68}$$

$$\bar{B} = \begin{bmatrix} B \\ I \end{bmatrix} \tag{3.69}$$

$$\bar{Q} = \begin{bmatrix} C' Q_z C & 0 \\ 0 & S \end{bmatrix} \tag{3.70}$$

$$\bar{M} = \begin{bmatrix} 0 \\ -S \end{bmatrix} \tag{3.71}$$

$$\bar{R} = S \tag{3.72}$$

$$\bar{P}_N = \begin{bmatrix} C' Q_z C & 0 \\ 0 & 0 \end{bmatrix} \tag{3.73}$$

$$\bar{b} = \mathbf{0} \tag{3.74}$$

$$\bar{q}_k = \begin{bmatrix} -C' Q_z r_k \\ 0 \end{bmatrix} \tag{3.75}$$

$$\bar{r}_k = \mathbf{0} \tag{3.76}$$

$$\bar{f}_k = \frac{1}{2} r_k' Q_z r_k \tag{3.77}$$

$$\bar{p}_N = \begin{bmatrix} -C' Q_z r_N \\ 0 \end{bmatrix} \tag{3.78}$$

$$\bar{\gamma}_N = \frac{1}{2} r_N' Q_z r_N \tag{3.79}$$

This extended LQ optimal control problem may be solved by DP stated as below.

Define the sequence of matrices $\{R_{e,k}, K_k, P_k\}_{k=0}^{N-1}$ as

$$R_{e,k} = \bar{R}_k + \bar{B}_k' P_{k+1} \bar{B}_k \tag{3.80}$$

$$K_k = -R_{e,k}^{-1}(\bar{M}_k + \bar{A}_k' P_{k+1} \bar{B}_k') \tag{3.81}$$

$$P_k = \bar{Q}_k + \bar{A}_k' P_{k+1} \bar{A}_k - K_k' R_{e,k} K_k \tag{3.82}$$

Define the vectors $\{c_k, d_k, a_k, p_k\}_{k=0}^{N-1}$ as

$$c_k = P_{k+1} \bar{b}_k + p_{k+1} \tag{3.83}$$

$$d_k = \bar{r}_k + \bar{B}_k' c_k \tag{3.84}$$

$$a_k = -R_{e,k}^{-1} d_k \tag{3.85}$$

$$p_k = \bar{q}_k + \bar{A}_k' c_k + K_k' d_k \tag{3.86}$$

Define the sequence of scalars $\{\gamma_k\}_{k=0}^{N-1}$ as

$$\gamma_k = \gamma_{k+1} + \bar{f}_k + p_{k+1}' \bar{b}_k + \frac{1}{2} \bar{b}_k' P_{k+1} \bar{b}_k + \frac{1}{2} d_k' a_k \tag{3.87}$$

Let $\bar{x}_0^*$ equal $\bar{x}_0$. Then the unique global minimizer of (3.64)-(3.67) will be obtained by the iteration

$$u_k^* = K_k \bar{x}_k^* + a_k \tag{3.88}$$
$$\bar{x}_{k+1}^* = A_k \bar{x}_k^* + B_k u_k^* + \bar{b}_k \tag{3.89}$$

The corresponding optimal value can be computed by

$$\phi^* = \frac{1}{2}\bar{x}_0' P_0 \bar{x}_0 + P_0^{'} \bar{x}_0 + \gamma_0 \tag{3.90}$$

In practice, this procedure can be implemented by Algorithm 1 or the combination of Algorithm 2 and Algorithm 3.

The computational complexity for DP is $\mathcal{O}(N \cdot (n^3 + m^3))$. The computational time of DP is linear in the predictive horizon and cubic in both the number of states and the number of inputs.

CHAPTER 4

Interior-Point Method

In previous two chapters, we discussed how to solve unconstrained LQ output regulation problem by CVP and DP. Essentially, the unconstrained LQ output regulation problem is transformed into unconstrained QP problems by these two methods. In this chapter, we solve the LQ output regulation problem with input and input-rate constraints. The constrained LQ output regulation problem is transformed into an inequality constrained QP problem using CVP. The Interior-Point algorithm is introduced to solve the inequality constrained QP problem.

4.1 Constrained LQ Output Regulation Problem

In this section, the constrained LQ output regulation problem is formulated into an inequality constrained QP problem by CVP.

The formulation of the LQ output regulation problem with input and input-rate constraints may be expressed by the following QP:

$$\min \phi = \frac{1}{2}\sum_{k=0}^{N} \|z_k - r_k\|_{Q_z}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 \tag{4.1}$$

subject to the following equality constraints:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \qquad k = 0, 1, ..., N-1 && (4.2)\\ z_k &= C_z x_k \qquad\quad\;\; k = 0, 1, ..., N && (4.3) \end{aligned}$$

and inequality constraints:

$$\begin{aligned} u_{min} \leq \;\; & u_k \;\; \leq u_{max} \qquad k = 0, 1, ..., N-1 && (4.4)\\ \Delta u_{min} \leq \;\; & \Delta u_k \;\; \leq \Delta u_{max} \qquad k = 0, 1, ..., N-1 && (4.5) \end{aligned}$$

in which $x_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^m$, $z_k \in \mathbb{R}^p$ and $\Delta u_k = u_k - u_{k-1}$. The constraints (4.2)-(4.3) stand for a discrete state space model. (4.4) is the input constraints, and (4.5) is the input-rate constraint.

Based on the conclusion in Chapter 2, (4.1)-(4.3) are transformed into an unconstrained QP problem:

$$\min \;\; \psi = \frac{1}{2}U'HU + g'U \tag{4.6}$$

in which H is the Hessian matrix.

Likewise, (4.4) may be expressed in stacked vectors

$$\underbrace{\begin{bmatrix} u_{min} \\ u_{min} \\ \vdots \\ u_{min} \end{bmatrix}}_{U_{min}} \leq \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \leq \underbrace{\begin{bmatrix} u_{max} \\ u_{max} \\ \vdots \\ u_{max} \end{bmatrix}}_{U_{max}} \tag{4.7}$$

which can be written in

$$U_{min} \leq U \leq U_{max} \tag{4.8}$$

And (4.5) may be expressed in stacked vectors

$$\begin{bmatrix} \Delta u_{min} \\ \Delta u_{min} \\ \vdots \\ \Delta u_{min} \end{bmatrix} \leq \begin{bmatrix} u_0 - u_{-1} \\ u_1 - u_0 \\ \vdots \\ u_{N-1} - u_{N-2} \end{bmatrix} \leq \begin{bmatrix} \Delta u_{max} \\ \Delta u_{max} \\ \vdots \\ \Delta u_{max} \end{bmatrix} \Longleftrightarrow$$
$$\begin{bmatrix} \Delta u_{min} + u_{-1} \\ \Delta u_{min} \\ \vdots \\ \Delta u_{min} \end{bmatrix} \leq \begin{bmatrix} I & & & \\ -I & I & & \\ & & \ddots & \\ & & -I & I \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \leq \begin{bmatrix} \Delta u_{max} + u_{-1} \\ \Delta u_{max} \\ \vdots \\ \Delta u_{max} \end{bmatrix} \tag{4.9}$$

The first row of (4.9) can be expressed as

$$\Delta u_{min} + u_{-1} \leq u_0 \leq \Delta u_{max} + u_{-1} \tag{4.10}$$

Consequently, the remaining part of (4.9) is

$$\underbrace{\begin{bmatrix} \Delta u_{min} \\ \vdots \\ \Delta u_{min} \end{bmatrix}}_{\Delta U_{min}} \leq \underbrace{\begin{bmatrix} -I & I & & & \\ & -I & I & & \\ & & \ddots & & \\ & & & -I & I \end{bmatrix}}_{\Lambda} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \leq \underbrace{\begin{bmatrix} \Delta u_{max} \\ \vdots \\ \Delta u_{max} \end{bmatrix}}_{\Delta U_{max}} \tag{4.11}$$

which is written in:

$$\Delta U_{min} \leq \Lambda U \leq \Delta U_{max} \tag{4.12}$$

Combining the first row of (4.7) and (4.10), we have

$$\max(u_{min}, \Delta u_{min} + u_{-1}) \leq u_0 \leq \min(u_{max}, \Delta u_{max} + u_{-1}) \tag{4.13}$$

Therefore, the constrained LQ output regulation problem (4.1)-(4.5) may be expressed as an inequality constrained QP problem,

$$\min_{U} \quad \psi = \frac{1}{2} U' H U + g' U \tag{4.14}$$
$$s.t. \quad U_{min} \leq U \leq U_{max} \tag{4.15}$$
$$\Delta U_{min} \leq \Lambda U \leq \Delta U_{max} \tag{4.16}$$

4.2 Interior-Point Method

In this section, we explain the principles applied in the Interior-Point algorithm, which are the foundation to solve the inequality constrained QP problem (4.14)-(4.16). Many techniques described here are originally established in [11]. One can read [11] for further information.

4.2.1 Optimality Condition

Consider a simple convex QP problem

$$\min \quad \frac{1}{2}x'Hx + g'x \tag{4.17}$$

$$s.t. \quad A'x \geq b \tag{4.18}$$

where $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, H is a positive definite matrix, A is a $n \times m$ matrix with full rank.

The Lagrangian is utilized to solve this QP problem. We introduce the Lagrange multiplier λ, then the Lagrangian of (4.17)-(4.18) is

$$\mathcal{L}(x, \lambda) = \frac{1}{2}x'Hx + g'x - \lambda'(A'x - b) \tag{4.19}$$

The constrained QP problem (4.17)-(4.18) is reduced to an unconstrained QP problem (4.19). The optimal solution of (4.19) is determined by the first order KKT conditions

$$\nabla_x \mathcal{L}(x, \lambda) = Hx + g - A\lambda = 0 \tag{4.20}$$

$$A'x - b \geq 0 \ \perp \ \lambda \geq 0 \tag{4.21}$$

in which $\perp$ denotes complementarity. Since the problem (4.17)-(4.18) is convex, the first order KKT conditions are both necessary and sufficient.

Define the slack variables s as

$$s = A'x - b \geq 0 \tag{4.22}$$

Then (4.20)-(4.21) may be expressed as

$$Hx + g - A\lambda = 0 \tag{4.23}$$

$$s - A'x + b = 0 \tag{4.24}$$

$$S\Lambda e = 0 \tag{4.25}$$

$$(s, \lambda) \geq 0 \tag{4.26}$$

where S = diag(s$_1, s_2, ..., s_m$), Λ= diag($\lambda_1, \lambda_2, ..., \lambda_m$) and e = $(1, 1, ..., 1)'$.

4.2.2 Newton Step

The nonlinear system of equations (4.23)-(4.25) may be solved numerically by Newton's method.

Given a current iterate (x, λ, s) satisfying $(s, \lambda) \geq 0$, the search direction $(\Delta x, \Delta\lambda, \Delta s)$ may be obtained by solving the following system functions

$$\begin{bmatrix} H & -A & 0 \\ -A' & 0 & I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = - \begin{bmatrix} r_L \\ r_s \\ r_{s\lambda} \end{bmatrix} \tag{4.27}$$

where

$$r_L = Hx + g - A\lambda \tag{4.28}$$

$$r_s = s - A'x + b \tag{4.29}$$

$$r_{s\lambda} = S\Lambda e \tag{4.30}$$

Usually, a full step along the direction $(\Delta x, \Delta\lambda, \Delta s)$ would violate the bound $(\lambda, s) \geq 0$ [11], so we introduce a line search parameter $\alpha \in (0, 1]$ such that the maximum step length α_{max} satisfies

$$s + \alpha_{max}\Delta s \geq 0 \tag{4.31}$$

$$\lambda + \alpha_{max}\Delta\lambda \geq 0 \tag{4.32}$$

The new iterate in the Newton iteration is

$$\begin{bmatrix} x \\ \lambda \\ s \end{bmatrix} \leftarrow \begin{bmatrix} x \\ \lambda \\ s \end{bmatrix} + \alpha_{max} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} \tag{4.33}$$

4.2.3 Predictor-Corrector Interior-Point Method

To avoid being restricted to small step lengths, (4.30) is modified such that the search directions are biased toward the interior of the nonnegative orthant defined by $(\lambda, s) \geq 0$. It is possible to take a longer step along the biased direction than along the pure Newton direction before violating the positivity condition [11]. (4.30) is modified as

$$r_{s\lambda} = S\Lambda e - \sigma\mu e = 0 \tag{4.34}$$

$$\mu = \frac{s'\lambda}{m} = \frac{\sum_{i=1}^{m} s_i\lambda_i}{m} \tag{4.35}$$

where μ is the duality gap, and its ideal value is 0. $\sigma \in [0,1]$ is the centering parameter. When $\sigma = 1$, (4.34) defines a centering direction. When $\sigma = 0$, (4.34) gives the standard Newton step. The Newton step is called as an affine step.

A practical implementation of Interior-Point algorithm is Mehrotra's predictor-corrector method. The essence of this method is using the corrector steps to compensate for the errors made by the Newton (affine) step. Consider the affine direction $(\Delta x, \Delta\lambda, \Delta s)$ defined by

$$\begin{bmatrix} H & -A & 0 \\ -A' & 0 & I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x^{aff} \\ \Delta\lambda^{aff} \\ \Delta s^{aff} \end{bmatrix} = - \begin{bmatrix} r_L \\ r_s \\ S\Lambda e \end{bmatrix} \tag{4.36}$$

Taking a full step in this direction, we obtain

$$\begin{aligned} &(s + \Delta s^{aff})(\lambda + \Delta\lambda^{aff}) \\ &= s\lambda + s\Delta\lambda^{aff} + \lambda\Delta s^{aff} + \Delta s^{aff}\Delta\lambda^{aff} = \Delta s^{aff}\Delta\lambda^{aff} \end{aligned}$$

The updated value of $s\lambda$ is $\Delta s^{aff}\Delta\lambda^{aff}$ rather than the ideal value 0. To correct this deviation, we solve the following system to obtain a step $(\Delta x^{cor}, \Delta\lambda^{cor}, \Delta s^{cor})$

$$\begin{bmatrix} H & -A & 0 \\ -A' & 0 & I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x^{cor} \\ \Delta\lambda^{cor} \\ \Delta s^{cor} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\Delta S^{aff}\Delta\Lambda^{aff} e \end{bmatrix} \tag{4.37}$$

Usually the corrected step $(\Delta x^{aff}, \Delta\lambda^{aff}, \Delta s^{aff}) + (\Delta x^{cor}, \Delta\lambda^{cor}, \Delta s^{cor})$ reduces the duality gap more than the affine step alone does [11].

The next step is to calculate the centering step. From (4.31)(4.32), the maximum steplengths along the affine direction (4.36) may be calculated by

$$\alpha^{\lambda}_{max} = \min\left(1, \min_{\Delta\lambda<0} -\frac{\lambda}{\Delta\lambda^{aff}}\right) \tag{4.38}$$

$$\alpha^{s}_{max} = \min\left(1, \min_{\Delta s<0} -\frac{s}{\Delta s^{aff}}\right) \tag{4.39}$$

Then $\alpha_{max} = \min(\alpha^{\lambda}_{max}, \alpha^{s}_{max})$ is used to calculate the affine duality gap

$$\mu_{aff} = (\lambda + \alpha_{max}\Delta\lambda)'(s + \alpha_{max}\Delta s)/m \tag{4.40}$$

The centering parameter σ is set

$$\sigma = \left(\frac{\mu_{aff}}{\mu}\right)^3 \tag{4.41}$$

Finally, the centering step $(\Delta x^{cen}, \Delta\lambda^{cen}, \Delta s^{cen})$ is calculated by

$$\begin{bmatrix} H & -A & 0 \\ -A' & 0 & I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x^{cen} \\ \Delta\lambda^{cen} \\ \Delta s^{cen} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sigma\mu e \end{bmatrix} \tag{4.42}$$

Therefore, Mehrotra's direction is

$$\begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} \Delta x^{aff} \\ \Delta\lambda^{aff} \\ \Delta s^{aff} \end{bmatrix} + \begin{bmatrix} \Delta x^{cor} \\ \Delta\lambda^{cor} \\ \Delta s^{cor} \end{bmatrix} + \begin{bmatrix} \Delta x^{cen} \\ \Delta\lambda^{cen} \\ \Delta s^{cen} \end{bmatrix} \tag{4.43}$$

In summary, the search direction is calculated by solving

$$\begin{bmatrix} H & -A & 0 \\ -A' & 0 & I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} -r_L \\ -r_s \\ -r_{s\lambda} - \Delta S^{aff}\Delta\Lambda^{aff} e + \sigma\mu e \end{bmatrix} \tag{4.44}$$

The following step is to update iterate (x, λ, s). Similar to (4.38)(4.39), the maximum allowable step length is calculated by

$$\alpha_{max}^{\lambda} = \min\left(1, \min_{\Delta\lambda<0} -\frac{\lambda}{\Delta\lambda}\right) \tag{4.45}$$

$$\alpha_{max}^{s} = \min\left(1, \min_{\Delta s<0} -\frac{s}{\Delta s}\right) \tag{4.46}$$

Then $\alpha_{max} = \min(\alpha_{max}^{\lambda}, \alpha_{max}^{s})$ is used to update (x, λ, s)

$$\begin{bmatrix} x \\ \lambda \\ s \end{bmatrix} = \begin{bmatrix} x + \eta\alpha_{max}\Delta x \\ \lambda + \eta\alpha_{max}\Delta\lambda \\ s + \eta\alpha_{max}\Delta s \end{bmatrix} \tag{4.47}$$

in which $\eta \in]0, 1[$.

4.2.4 Algorithm

This section presents the efficient solution of the convex QP problem (4.17)-(4.18).

Because of the presence of the Hessian matrix, H, the solution of the system (4.27) is the major computational operation in the Interior-Point method. In order to solve the system efficiently, we exploit the structure of the system.

Note that the second block row of (4.27) yields

$$\Delta s = -r_s + A'\Delta x \tag{4.48}$$

Since $S > 0$ and it is a diagonal matrix with positive entries, it is inverted easily. The third block row of (4.27) along with (4.48) yields

$$\begin{aligned} \Delta\lambda &= -S^{-1}(r_{s\lambda} + \Lambda\Delta s) \\ &= S^{-1}(-r_{s\lambda} + \Lambda r_s) - S^{-1}\Lambda A'\Delta x \end{aligned} \tag{4.49}$$

Finally, the first block row of (4.27) along with (4.49) yields

$$\begin{aligned} -r_L &= H\Delta x - A\Delta\lambda \\ &= (H + AS^{-1}\Lambda A')\Delta x - AS^{-1}(-r_{s\lambda} + \Lambda r_s) \\ &= \bar{H}\Delta x + \bar{r} \end{aligned} \tag{4.50}$$

in which

$$\bar{H} = H + A(S^{-1}\Lambda)A' \tag{4.51}$$

$$\bar{r} = A[S^{-1}(r_{s\lambda} - \Lambda r_s)] \tag{4.52}$$

Consequently, Δx may be obtained from

$$\bar{H}\Delta x = -r_L - \bar{r} \tag{4.53}$$

Subsequently Δs may be obtained from (4.48), and $\Delta\lambda$ may be obtained from (4.49).

Algorithm 1 specifies the steps solving (4.17)-(4.18).

Algorithm 1: Interior-Point algorithm for (4.17).

Require: $(H \in S^n_{++}, g \in \mathbb{R}^n, A \in \mathbb{R}^{n\times m}, b \in \mathbb{R}^m)$
  Residuals and Duality Gap:
  $r_L = Hx + g - A\lambda, r_s = s - A'x + b, r_{s\lambda} = S\Lambda e$
  Duality gap: $\mu = \frac{s'\lambda}{m}$
  while NotConverged **do**
    Compute $\bar{H} = H + A(S^{-1}\Lambda)A'$
    Cholesky factorization: $\bar{H} = \bar{L}\bar{L}'$
    Affine Predictor Step:
    Compute $\bar{r} = A(S^{-1}(r_{s\lambda} - \Lambda r_s)), -\bar{g} = -(r_L + \bar{r})$
    Solve: $\bar{L}\bar{L}'\Delta x = -\bar{g}$
    $\Delta s = -r_s + A'\Delta x$
    $\Delta\lambda = -S^{-1}(r_{s\lambda} + \Lambda\Delta s)$
    Determine the maximum affine step length
      $\lambda + \alpha_{max}\Delta\lambda \geq 0, \;\; s + \alpha_{max}\Delta s \geq 0$
    Select affine step length: $\alpha \in (0, \alpha_{max}]$
    Compute affine duality gap: $\mu_a = \frac{(\lambda+\alpha\Delta\lambda)'(s+\alpha\Delta s)}{m}$
    Centering parameter: $\sigma = \left(\frac{\mu_a}{\mu}\right)^3$
    Center Corrector Step:
    Modified complementarity:
      $r_{s\lambda} \leftarrow r_{s\lambda} + \Delta S\Delta\Lambda e - \sigma\mu e$
    Compute $\bar{r} = A(S^{-1}(r_{s\lambda} - \Lambda r_s)), -\bar{g} = -(r_L + \bar{r})$
    Solve: $\bar{L}\bar{L}'\Delta x = -\bar{g}$
    $\Delta s = -r_s + A'\Delta x$
    $\Delta\lambda = -S^{-1}(r_{s\lambda} + \Lambda\Delta s)$
    Determine the maximum center step length
      $\lambda + \alpha_{max}\Delta\lambda \geq 0 \;\; s + \alpha_{max}\Delta s \geq 0$
    Select step length: $\alpha \in (0, \alpha_{max}]$
    Step:$x \leftarrow x + \alpha\Delta x, \lambda \leftarrow \lambda + \alpha\Delta\lambda, s \leftarrow s + \alpha\Delta s$
    Residuals and Duality Gap:
    $r_L = Hx + g - A\lambda, r_s = s - A'x + b, r_{s\lambda} = S\Lambda e$
    Duality gap: $\mu = \frac{s'\lambda}{m}$
  end while
  Return: (x, λ)

4.3 Interior-Point Algorithm for MPC

In this section we specialize the Interior-Point algorithm to a convex quadratic program with the structure

$$\min_{x\in\mathbb{R}^n} \quad \frac{1}{2}x'Hx + g'x \tag{4.54}$$

$$s.t. \quad x_l \leq x \leq x_u \tag{4.55}$$

$$b_l \leq A'x \leq b_u \tag{4.56}$$

This optimization problem has the same structure as (4.14)-(4.16). It may be written in the form of (4.17)-(4.18)

$$\min_{x\in\mathbb{R}^n} \quad \frac{1}{2}x'Hx + g'x \tag{4.57}$$

$$s.t. \quad x \geq x_l \tag{4.58}$$

$$-x \geq -x_u \tag{4.59}$$

$$A'x \geq b_l \tag{4.60}$$

$$-A'x \geq -b_u \tag{4.61}$$

Apparently, (4.57)-(4.61) has the same structure as (4.17)-(4.18), except that the former has three more inequality constraints than the latter. Therefore, the problem (4.54)-(4.56) may be solved in the same way as the problem (4.17)-(4.18). In the case of (4.54)-(4.56), 3 more Lagrangian multipliers and 3 more slack variables are introduced.

4.3.1 Optimality Conditions

Define the Lagrangian multipliers $\lambda, \mu, \delta, \kappa$. The Lagrangian associated to (4.57)-(4.61) is

$$\begin{aligned} \mathcal{L} \;=\; & \frac{1}{2}x'Hx + g'x - \lambda'(x - x_l) - \mu'(x_\mu - x) \\ & -\delta'(A'x - b_l) - \kappa'(b_u - A'x) \end{aligned} \tag{4.62}$$

and its corresponding stationary point is determined by

$$\nabla_x \mathcal{L} = Hx + g - \lambda + \mu - A(\delta - \kappa) = 0 \tag{4.63}$$

Consequently, the KKT conditions for (4.57)-(4.61) are

$$
\begin{aligned}
&Hx + g - \lambda + \mu - A(\delta - \kappa) = 0 && (4.64)\\
&x - x_l \geq 0 \quad \perp \quad \lambda \geq 0 && (4.65)\\
&x_u - x \geq 0 \quad \perp \quad \mu \geq 0 && (4.66)\\
&A'x - b_l \geq 0 \quad \perp \quad \delta \geq 0 && (4.67)\\
&b_u - A'x \geq 0 \quad \perp \quad \kappa \geq 0 && (4.68)
\end{aligned}
$$

Define the slack variables l, u, s, t as

$$
\begin{aligned}
&l = x - x_l \geq 0 && (4.69)\\
&u = x_u - x \geq 0 && (4.70)\\
&s = A'x - b_l \geq 0 && (4.71)\\
&t = b_\mu - A'x \geq 0 && (4.72)
\end{aligned}
$$

Then we have following KKT conditions

$$
\begin{aligned}
&r_L = Hx + g - \lambda + \mu - A(\delta - \kappa) = 0 && (4.73)\\
&r_l = l - x + x_l = 0 && (4.74)\\
&r_\mu = \mu + x - x_u = 0 && (4.75)\\
&r_s = s - A'x + b_l = 0 && (4.76)\\
&r_t = t + A'x - b_u = 0 && (4.77)\\
&r_{l\lambda} = L\Lambda e = 0 && (4.78)\\
&r_{u\mu} = UMe = 0 && (4.79)\\
&r_{s\delta} = SDe = 0 && (4.80)\\
&r_{t\kappa} = TKe = 0 && (4.81)\\
&(l, u, s, t, \lambda, \mu, \delta, \kappa) \geq 0 && (4.82)
\end{aligned}
$$

with $L, \Lambda, U, M, S, D, T$ and K being positive diagonal matrix representations of $l, \lambda, u, \mu, s, \delta, t$, and κ, respectively.

4.3.2 Newton Step

The Newton direction of (4.73)-(4.81) is computed by solution of the following system

$$H\Delta x - \Delta\lambda + \Delta\mu - A(\Delta\delta - \Delta\kappa) = -r_L \tag{4.83}$$
$$\Delta l - \Delta x = -r_l \tag{4.84}$$
$$\Delta\mu + \Delta x = -r_\mu \tag{4.85}$$
$$\Delta s - A'\Delta x = -r_s \tag{4.86}$$
$$\Delta t + A'\Delta x = -r_t \tag{4.87}$$
$$\Lambda\Delta l + L\Delta\lambda = -r_{l\lambda} \tag{4.88}$$
$$M\Delta u + U\Delta\mu = -r_{u\mu} \tag{4.89}$$
$$D\Delta s + S\Delta\delta = -r_{s\delta} \tag{4.90}$$
$$K\Delta t + T\Delta\kappa = -r_{t\kappa} \tag{4.91}$$

This system may be solved by substituting

$$\Delta l = -r_l + \Delta x \tag{4.92}$$
$$\Delta u = -r_u - \Delta x \tag{4.93}$$
$$\Delta s = -r_s + A'\Delta x \tag{4.94}$$
$$\Delta t = -r_t - A'\Delta x \tag{4.95}$$

in

$$\Delta\lambda = -L^{-1}(r_{l\lambda} + \Lambda\Delta l) \tag{4.96}$$
$$\Delta\mu = -U^{-1}(r_{u\mu} + M\Delta u) \tag{4.97}$$
$$\Delta\delta = -S^{-1}(r_{s\delta} + D\Delta s) \tag{4.98}$$
$$\Delta\kappa = -T^{-1}(r_{t\kappa} + K\Delta t) \tag{4.99}$$

to obtain

$$\Delta\lambda = L^{-1}(-r_{l\lambda} + \Lambda r_l) - L^{-1}\Lambda\Delta x \tag{4.100}$$
$$\Delta\mu = U^{-1}(-r_{u\mu} + M r_u) + U^{-1}M\Delta x \tag{4.101}$$
$$\Delta\delta = S^{-1}(-r_{s\delta} + D r_s) - S^{-1}DA'\Delta x \tag{4.102}$$
$$\Delta\kappa = T^{-1}(-r_{t\kappa} + K r_t) + T^{-1}KA'\Delta x \tag{4.103}$$

Substitute (4.100)-(4.103) into (4.83), yields

$$\begin{aligned} -r_L &= H\Delta x - \Delta\lambda + \Delta\mu - A(\Delta\delta - \Delta\kappa) \\ &= \bar{H}\Delta x + \bar{r} \end{aligned} \tag{4.104}$$

with

$$\bar{H} = H + L^{-1}\Lambda + U^{-1}M + A(S^{-1}D + T^{-1}K)A' \tag{4.105}$$

and

$$\begin{aligned}\bar{r} \quad = \quad & -L^{-1}(-r_{l\lambda} + \Lambda r_l) + U^{-1}(-r_{u\mu} + Mr_u) \\ & -A[S^{-1}(-r_{s\delta} + Dr_s) - T^{-1}(-r_{t\kappa} + Kr_t)]\end{aligned} \tag{4.106}$$

Then Δx may be solved using Cholesky factorization of $\bar{H}$. The remaining part may be obtained by subsequent substitution in (4.92)-(4.99).

4.3.3 Interior-Point Algorithm

The duality gap in the Interior-Point algorithm for (4.54)-(4.56) is

$$gap = \frac{l'\lambda + u'\mu + s'\delta + t'\kappa}{2(n+m)} \tag{4.107}$$

in which n and m are the number of variables and constrains, respectively.

The modified residuals used in the center corrector step of Mehrotra's primal-dual interior point algorithm are

$$r_{l\lambda} = L\Lambda e + \Delta L \Delta\Lambda e - \sigma \cdot gap \cdot e \tag{4.108}$$

$$r_{u\mu} = UMe + \Delta U \Delta M e - \sigma \cdot gap \cdot e \tag{4.109}$$

$$r_{s\delta} = SDe + \Delta S \Delta D e - \sigma \cdot gap \cdot e \tag{4.110}$$

$$r_{t\kappa} = TKe + \Delta T \Delta K e - \sigma \cdot gap \cdot e \tag{4.111}$$

in which $\Delta L, \Delta\Lambda, \Delta U, \Delta M, \Delta S, \Delta D, \Delta T$ and ΔK are diagonal matrices of the search direction obtained in the affine step.

The steps in the Interior-Point method for solution of (4.54)-(4.56) are listed in Algorithm 2.

Algorithm 2: Interior-Point algorithm for (4.54)-(4.56).

Require: $(H, g, x_l, x_u, b_l, A, b_u)$
  Compute residuals (4.73)-(4.81) and duality gap (4.107)
  while NotConverged **do**
    Compute $\bar{H}$ by (4.105) and Cholesky factorization: $\bar{H} = \bar{L}\bar{L}'$
    Affine Predictor Step:
    Compute $\bar{r}$ by(4.106) and $-\bar{g} = -r_L - \bar{r}$
    Solve: $\bar{L}\bar{L}'\Delta x^a = -\bar{g}$
    Compute $\Delta l^a, \Delta u^a, \Delta s^a, \Delta t^a$ by (4.92)-(4.95)
    Compute $\Delta\lambda^a, \Delta\mu^a, \Delta\delta^a, \Delta\kappa^a$ by (4.96)-(4.99)
    Compute affine step length, α^a
    Compute affine variables and affine duality gap
    Centering parameter:$\sigma = (gap^a/gap)^3$
    Center Corrector Step
    Compute modified residuals (4.108)-(4.111)
    Compute $\bar{r}$ by(4.106) and $-\bar{g} = -r_L - \bar{r}$
    Solve: $\bar{L}\bar{L}'\Delta x^a = -\bar{g}$
    Compute $\Delta l^a, \Delta u^a, \Delta s^a, \Delta t^a$ by (4.92)-(4.95)
    Compute $\Delta\lambda^a, \Delta\mu^a, \Delta\delta^a, \Delta\kappa^a$ by (4.96)-(4.99)
    Compute step length, α
    Compute new variables: $x \leftarrow x + \alpha\Delta x, ...$
    Compute residuals (4.73)-(4.81) and duality gap (4.107)
  end while
  Return: $(x, \lambda, \mu, \delta, \kappa)$

Since the matrix Λ defined in (4.16) corresponds to the matrix A' in (4.56), the special structure of Λ can be utilized to simplify the computations in Algorithm 2. Consider the case with $N = 4$ such that

$$\Lambda = \begin{bmatrix} -I & I & & \\ & -I & I & \\ & & -I & I \end{bmatrix}, \; x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \; s = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}, \; D = \begin{bmatrix} D_1 & & \\ & D_2 & \\ & & D_3 \end{bmatrix}$$

Then the operations involving Λ are

$$\Lambda x = \begin{bmatrix} -x_1 + x_2 \\ -x_2 + x_3 \\ -x_3 + x_4 \end{bmatrix}, \qquad \Lambda' s = \begin{bmatrix} 0 - s_1 \\ s_1 - s_2 \\ s_2 - s_3 \\ s_3 - 0 \end{bmatrix}$$

and $$\Lambda' D \Lambda = \begin{bmatrix} 0 + D_1 & -D_1 & & \\ -D_1 & D_1 + D_2 & -D_2 & \\ & -D_2 & D_2 + D_3 & -D_3 \\ & & -D_3 & D_3 + 0 \end{bmatrix}$$

4.4 Computational Complexity Analysis

In Chapter 2, we analyze the computational complexity of solving the unconstrained LQ output regulation problem by CVP. For the constrained LQ output regulation problem, the computational time is determined by computations in each iteration. The major computation in each iteration is on the Cholescky factorization of the Hessian matrix $\bar{H}$. As we show in Chapter 2, the complexity of the Cholescky factorization of the Hessian matrix is $\mathcal{O}(m^3N^3)$. Therefore the total computational time for solving the constrained LQ output regulation problem is proportional to

$$\text{Number of iteration } \times m^3N^3$$

Since the number of iteration depends weakly on the predictive horizon [15], the computational complexity is $\mathcal{O}(m^3N^3)$. The solution time depends on the predictive horizon and the number of inputs cubically.

4.5 Summary

In this chapter, we apply Interior-Point method to solve the constrained LQ output regulation problem. The constrained LQ output regulation problem is formulated as an inequality constrained QP problem by CVP. The Interior-Point method specialized for the inequality constrained QP problem is developed.

Problem 1: Convex Inequality QP

$$\min \quad \frac{1}{2}x'Hx + g'x \tag{4.112}$$

$$s.t. \quad A'x \geq b \tag{4.113}$$

Primal-Dual Interior-Point Solution:

The problem (4.112)-(4.113) may be solved by a sequence of Newton steps with modified search directions and step lengths. In each Newton step, firstly the affine step is calculated by

$$\begin{bmatrix} H & -A & 0 \\ -A' & 0 & I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x^{aff} \\ \Delta\lambda^{aff} \\ \Delta s^{aff} \end{bmatrix} = - \begin{bmatrix} r_L \\ r_s \\ r_{s\lambda} \end{bmatrix} \tag{4.114}$$

where

$$r_L = Hx + g - A\lambda \tag{4.115}$$

$$r_s = s - A'x + b \tag{4.116}$$

$$r_{s\lambda} = S\Lambda e \tag{4.117}$$

Then the maximum steplengths along the affine direction is calculated by

$$\alpha_{max}^{\lambda} = \min\left(1, \min_{\Delta\lambda<0} -\frac{\lambda}{\Delta\lambda^{aff}}\right) \tag{4.118}$$

$$\alpha_{max}^{s} = \min\left(1, \min_{\Delta s<0} -\frac{s}{\Delta s^{aff}}\right) \tag{4.119}$$

$$\alpha_{max} = \min(\alpha_{max}^{\lambda}, \alpha_{max}^{s}) \tag{4.120}$$

The affine duality gap is

$$\mu_{aff} = (\lambda + \alpha_{max}\Delta\lambda^{aff})'(s + \alpha_{max}\Delta s^{aff})/m \tag{4.121}$$

The centering parameter σ is chosen by,

$$\sigma = \left(\frac{\mu_{aff}}{\mu}\right)^3 \tag{4.122}$$

where $\mu = \frac{\lambda' s}{m}$.

Next, the centering corrector step is calculated by

$$\begin{bmatrix} H & -A & 0 \\ -A' & 0 & I \\ 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = - \begin{bmatrix} r_L \\ r_s \\ S\Lambda e + \Delta S^{aff} \Delta\Lambda^{aff} e - \sigma\mu e \end{bmatrix} \tag{4.123}$$

The last step is to update (x, λ, s). Calculate the maximum allowable step length by

$$\alpha_{max}^{\lambda} = \min\left(1, \min_{\Delta\lambda<0} -\frac{\lambda}{\Delta\lambda}\right) \tag{4.124}$$

$$\alpha_{max}^{s} = \min\left(1, \min_{\Delta s<0} -\frac{s}{\Delta s}\right) \tag{4.125}$$

Then $\alpha_{max} = \min(\alpha_{max}^{\lambda}, \alpha_{max}^{s})$ is used to update (x, λ, s)

$$\begin{bmatrix} x \\ \lambda \\ s \end{bmatrix} = \begin{bmatrix} x + \eta\alpha_{max}\Delta x \\ \lambda + \eta\alpha_{max}\Delta\lambda \\ s + \eta\alpha_{max}\Delta s \end{bmatrix} \tag{4.126}$$

where $\eta \in]0, 1[$.

In practice, this procedure can be fulfilled by Algorithm 1 efficiently.

Problem 2: LQ Output Regulation with Input and Input Rate Constraints

$$
\begin{aligned}
\min \quad & \phi = \frac{1}{2}\sum_{k=0}^{N} \|z_k - r_k\|_{Q_z}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 && && (4.127)\\
s.t. \quad & x_{k+1} = Ax_k + Bu_k && k = 0, 1, ..., N-1 && (4.128)\\
& z_k = C_z x_k && k = 0, 1, ..., N && (4.129)\\
& u_{min} \le u_k \le u_{max} && k = 0, 1, ..., N-1 && (4.130)\\
& \Delta u_{min} \le \Delta u_k \le \Delta u_{max} && k = 0, 1, ..., N-1 && (4.131)
\end{aligned}
$$

The Interior-Point Solution:

The first step is to express the constrained output regulation problem (4.127)-(4.131) as a convex inequality constrained QP problem

$$
\begin{aligned}
\min_{U} \quad & \psi = \frac{1}{2}U'HU + g'U && (4.132)\\
s.t. \quad & U_{min} \le U \le U_{max} && (4.133)\\
& \Delta U_{min} \le \Lambda U \le \Delta U_{max} && (4.134)
\end{aligned}
$$

in which the Hessian matrix is

$$
H = \Gamma' \mathcal{Q} \Gamma + H_S \tag{4.135}
$$

and the gradient is

$$
g = \Gamma' \mathcal{Q}\, \Phi x_0 - \Gamma' \mathcal{Q} R + M_{u_{-1}} u_{-1} \tag{4.136}
$$

where

$$
R = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_N \end{bmatrix} \qquad U = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix} \qquad \Phi = \begin{bmatrix} C_z A \\ C_z A^2 \\ C_z A^3 \\ \vdots \\ C_z A^N \end{bmatrix} \tag{4.137}
$$

$$
\Delta U_{min} = \begin{bmatrix} \Delta u_{min} \\ \vdots \\ \Delta u_{min} \end{bmatrix} \qquad \Delta U_{max} = \begin{bmatrix} \Delta u_{max} \\ \vdots \\ \Delta u_{max} \end{bmatrix} \tag{4.138}
$$

$$
U_{min} = \begin{bmatrix} \max(u_{min},\ \Delta u_{min} + u_{-1}) \\ u_{min} \\ \vdots \\ u_{min} \end{bmatrix} \tag{4.139}
$$

$$U_{max} = \begin{bmatrix} \min(u_{max},\ \Delta u_{max} + u_{-1}) \\ u_{max} \\ \vdots \\ u_{max} \end{bmatrix} \tag{4.140}$$

$$\Lambda = \begin{bmatrix} -I & I & & & \\ & -I & I & & \\ & & \ddots & & \\ & & & -I & I \end{bmatrix} \tag{4.141}$$

$$\mathcal{Q} = \begin{bmatrix} Q_z & & & & \\ & Q_z & & & \\ & & Q_z & & \\ & & & \ddots & \\ & & & & Q_z \end{bmatrix} \tag{4.142}$$

$$H_i = C_z A^{i-1} B \qquad i \geq 1 \tag{4.143}$$

$$\Gamma = \begin{bmatrix} H_1 & 0 & 0 & \cdots & 0 \\ H_2 & H_1 & 0 & \cdots & 0 \\ H_3 & H_2 & H_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ H_N & H_{N-1} & H_{N-2} & \cdots & H_1 \end{bmatrix} \tag{4.144}$$

$$H_s = \begin{bmatrix} 2S & -S & & & \\ -S & 2S & -S & & \\ & & \ddots & & \\ & & -S & 2S & -S \\ & & & -S & S \end{bmatrix} \tag{4.145}$$

$$M_{u_{-1}} = -\begin{bmatrix} S' & 0 & 0 & \dots & 0 \end{bmatrix}' \tag{4.146}$$

The second step is to solve (4.132)-(4.134) by the Interior-Point method. Algorithm 2 provides an efficient implementation of the Interior-Point method for solving the problem (4.132)-(4.134).

The computational time for solving the constrained optimal control problem arising from CVP is $\mathcal{O}(m^3N^3)$.

Chapter 5

Implementation

In this chapter, we describe Matlab implementations of the methods presented in Chapter 2, 3 and 4. The Matlab toolbox includes implementations of the CVP method and the DP method for solving the unconstrained LQ output regulation problem. The Interior-Point method based on CVP is implemented for solving the constrained LQ output regulation problem.

The unconstrained LQ output regulation problem is

$$\min \quad \phi = \frac{1}{2}\sum_{k=0}^{N} \|z_k - r_k\|_{Q_z}^2 + \frac{1}{2}\sum_{k=0}^{N-1} \|\Delta u_k\|_S^2 \tag{5.1}$$

$$s.t. \quad x_{k+1} = Ax_k + Bu_k \qquad k = 0, 1, ..., N-1 \tag{5.2}$$

$$z_k = C_z x_k \qquad k = 0, 1, ..., N \tag{5.3}$$

The constrained LQ output regulation problem is (5.1)-(5.3) with input and input-rate constraints

$$u_{min} \leq u_k \leq u_{max} \qquad k = 0, 1, ..., N-1 \tag{5.4}$$

$$\Delta u_{min} \leq \Delta u_k \leq \Delta u_{max} \qquad k = 0, 1, ..., N-1 \tag{5.5}$$

Table 5.1 lists all Matlab functions in the MPC toolbox. A brief description is provided for each function.

Matlab Function	Description
simMPC	Solve the unconstrained LQ output regulation problem
SEsolver	Solve the unconstrained LQ output regulation problem with CVP
MPCDesignSE	Design matrices of CVP
MPCComputeSE	Compute optimal MVs by CVP
MPCPredict	Compute predictions of states and outputs
DPslover	Solve the unconstrained LQ output regulation problem with DP
MPCDesignDP	Design matrices of DP
MPCComputeDP	Compute optimal MVs by DP
DPFactSolve	Algorithm 1. page 25
factorize	Algorithm 2. page 26
solveELQ	Algorithm 3. page 27
DesignDPU	Design unaltered matrices for DP
DesignDPA	Design altered data for DP
InteriorPoint	Solve the inequality constrained QP by the Interior-Point method
MPCInteriorPoint	Solve the constrained optimal control problem by the Interior-Point method

Table 5.1: Functions in the implemented Matlab MPC Toolblox

The method implementations consist of three essential phases:

1. Design: Given the specifications, the necessary matrices required for online computation are computed.
2. Compute: The optimal MVs are computed online.
3. Predict: The future MVs and CVs are computed

In the following, we describe the implementations of CVP and DP for solving the unconstrained LQ output regulation problem (5.1)-(5.3) individually.

5.1 Implementation of Control Vector Parameterization

In the first phase of implementing an MPC based on CVP, the function `MPCDesignSE` is used to design the necessary matrices for online computations. The function `MPCDesignSE` calculates the matrices $\{H, M_{x_0}, M_{u-1}, M_R, \phi, \Gamma\}$ using the model matrices A, B, C, the weight matrices Q, S and the predictive horizon N (see (2.36)-(2.44)). The Matlab source code of `MPCDesignSE` is provided in Appendix C.1.3.

After the necessary matrices are designed, the optimal MV is calculated online. The function `MPCComputeSE` computes the optimal inputs, $\{u_k^*\}_{k=0}^{N-1}$. (See (2.45)-(2.50)) The Matlab source code of `MPCComputeSE` is provided in Appendix C.1.4.

In the last phase of implementing an MPC based on CVP, the future MVs and CVs is computed by the function `MPCPredict`. The function `MPCPredict` uses the optimal inputs, $\{u_k^*\}_{k=0}^{N-1}$, to predict future states $\{x_{k+1}\}_{k=0}^{N-1}$ and CVs $\{z_{k+1}\}_{k=0}^{N-1}$ (see (2.2)-(2.3)). The Matlab source code of `MPCPredict` is provided in Appendix C.1.5.

The following is an example, in which we present the three phases for solving the unconstrained LQ output regulation problem (5.1)-(5.3) by CVP:

```
%-------------- Design --------------------------------
% define system
nSys = 2;                    % the number of states
% generate a discrete random state space model
sys = drss(nSys);
% retrieve the matrices for model
[A,B,C,D] = ssdata(sys);
sys.d = 0;                         % no disturbance
% weight matrix
Q = 1;
S = 0.0001;
% predictive horizon
N = 50;

% design necessary matrices
[H,Mx0,Mum1,MR] = MPCDesignSE(A,B,C,Q,S,N);

%-------------- Compute -------------------------------
```

```
% start point
x0 = zeros(nSys,1);
u_1 = 0;
% steady state level
xs = zeros(nSys,1);
us = 0;
zs = C*xs;
usN = repmat(us,N,1);
zsN = repmat(zs,N,1);
% form deviation  variable
dev_x = x0-xs;
dev_u_1 = u_1-us;
% reference
R = 20*ones(10,1);
ref = [R(:,2:end) repmat(R(:,end),1,N)];  ref = ref(:);

% compute the optimal input
[u,dev_u] = MPCComputeSE(H,Mx0,MR,Mum1,dev_x,ref,zsN,dev_u_1,usN);

%-------------- Predict -------------------------------
% predict states and outputs
[Xp,Zp] = MPCPredict(x0,u,N,A,B,C);
```

Furthermore, the function `SEsolver` is available in the MPC toolbox to simulate the behavior of the close loop optimal control system. The following pseudocode provides major part of the algorithm.

```
function [u,y] = SEsolver(A,B,C,Q,S,N,R,x0,u_1,xs,us)
```

The required input arguments:

```
% A: n by n matrix
% B: n by m matrix
% C: p by n matrix
% Q: weight matrix, symmetric p by p matrix
% S: weight matrix, symmetric m by m matrix
% N: prediction horison, scalor
% R: reference trajectory, p by t matrix
% x0: start state, n by 1 matrix
% u_1: start input, m by 1 matrix
% xs: steady-state value of states, n by 1 matrix
% us: steady-state value of inputs, m by 1 matrix
```

The first step is to initialize the parameters:

```
% identify the number of dimensions
n = size(A);        % dimension of state x
m = size(B,2);      % dimension of input u
p = size(C,1);      % dimension of output z,y
% form deviation variables
dev_u_1 = u_1-us;
dev_x = x0-xs;
dev_z = C*dev_x;
% steady-state level of model responds
zs = C*xs;
% reconstruct needed variables
R = [R(:,2:end) repmat(R(:,end),1,N)];  % reference
usN = repmat(us,N,1);
zsN = repmat(zs,N,1);
% initialize state
x = x0;
```

Then the necessary matrices for online computations are computed

```
[H,Mx0,Mum1,MR] = MPCDesignSE(A,B,C,Q,S,N);
```

Then, it is time to simulate the behavior of the closed loop control system:

```
% time sequence
t = 0:length(R)-1;
for k = 1:length(t)
    %------------- Compute On-line  -------------------
    % get N steps ahead of reference and transform it into
    % vertical vectors
    ref = R(:,k:k+N-1); ref = ref(:);
    % compute optimal inputs sequence up and
    % their deviations dev_u_1
    [up,dev_u_1] = MPCComputeSE(H,Mx0,MR,Mum1,...
                                 dev_x,ref,zsN,dev_u_1,usN);
    % store the first block row of the optimal inputs and
    % deviations
    u(:,k) = up(1:m);
    dev_u_1 = dev_u_1(1:m);
```

```
    %------------ Predict ---------------------------
    % predict states and outputs
    [Xp,Zp] = MPCPredict(x0,up,N,A,B,C);

    %------------ Update ----------------------------
    % update the current state
    x = Xp(1:n);
    dev_x = x - xs;
    % store the first block row of the controlled output
    y(:,k) = Zp(1:p);
end
```

In each loop, the reference sequence, `ref`, is updated primarily at the phase `Compute`. Sequentially the optimal inputs `up` and their deviations `dev_u_1` are computed on line. At the phase `Predict`, the optimal inputs `up` are used to predict the future states sequence `Xp` and the outputs sequence `Zp`. Finally, the current state `x` and its deviation `dev_x` are updated, and will be used for the next round of computations. The first block row of the output sequence `Zp` is stored in `y`.

The sequence of the optimal input `u` and the output `y` are returned at the end of the function.

5.2 Implementation of Dynamic Programming

As we discussed in Chapter 3, the LQ output regulation problem (5.1)-(5.3) needs to be transformed into the extended LQ optimal control problem before being solved by DP. The extended LQ optimal control problem is

$$\min_{\{\bar{x}_{k+1},u_k\}_{k=0}^{N-1}} \quad \phi = \sum_{k=0}^{N-1} l_k(\bar{x}_k, u_k) + l_N(\bar{x}_N) \tag{5.6}$$

$$s.t. \quad \bar{x}_{k+1} = \bar{A}\bar{x}_k + \bar{B}u_k + \bar{b} \qquad k = 0, 1, ..., N-1 \tag{5.7}$$

with the stage costs given by

$$l_k(\bar{x}_k, u_k) = \frac{1}{2}\bar{x}_k'\bar{Q}\bar{x}_k + \bar{x}_k\bar{M}u_k + \frac{1}{2}u_k'\bar{R}u_k + \bar{q}_k'\bar{x}_k + \bar{r}_k'u_k + \bar{f}_k \tag{5.8}$$

$$l_N(\bar{x}_N) = \frac{1}{2}\bar{x}_N'\bar{P}_N\bar{x}_N + \bar{p}_N'\bar{x}_N + \bar{\gamma}_N \tag{5.9}$$

The matrices $\bar{A}, \bar{B}, \bar{Q}, \bar{M}, \bar{R}, \bar{P}_N$ in the problem (5.6)-(5.9) are constant over time. Thus the matrices $\bar{A}, \bar{B}, \bar{Q}, \bar{M}, \bar{R}, \bar{P}_N$ can be computed offline. Furthermore, the factorization of the matrices $\bar{A}, \bar{B}, \bar{Q}, \bar{M}, \bar{R}, \bar{P}_N$, i.e. the computations of L_k, Z_k, P_k, can be carried out offline. The matrices (vectors) $\bar{b}, \bar{q}, \bar{r}, \bar{f}, \bar{p}_N, \bar{\gamma}_N$ in the problem (5.6)-(5.9) may change over time. Therefore the matrices $\bar{b}, \bar{q}, \bar{r}, \bar{f}, \bar{p}_N, \bar{\gamma}_N$ are computed online.

The first phase of the implementations of DP is to design the necessary matrices. The function `MPCDesignDP` is built to design the necessary matrices for online computations. Two subfunctions are involved in this phase:

- `DesignDPU`: computes the matrices $\bar{A}_k, \bar{B}_k, \bar{Q}_k, \bar{M}_k, \bar{R}_k$ $\bar{P}_N$ in the extended LQ optimal control problem (5.6)-(5.9) from the matrices A, B, C, Q, S of the LQ output regulation problem (5.1)-(5.3) (see (3.68)-(3.73)).

  ```
  [Abar,Bbar,Qbar,Mbar,Rbar,Pbar] = DesignDPU(A,B,C,Q,S);
  ```

  The Matlab source code of `DesignDPU` is provided in Appendix C.1.12.

- `factorize`: computes the factorization of the matrices $\bar{A}, \bar{B}, \bar{Q}, \bar{M}, \bar{R}, \bar{P}_N$ in the extended LQ optimal control problem (5.6)-(5.9) (see section 3.2.3 Algorithm 2).

  ```
  [P,L,Z] = factorize(Qbar,Mbar,Rbar,Abar',Bbar',Pbar,N);
  ```

  The Matlab source code of `factorize` is provided in Appendix C.1.10.

After the necessary matrices are designed, the optimal MV is calculated online. The function `MPCComputeDP` computes the optimal inputs, $\{u_k^*\}_{k=0}^{N-1}$. There are two subfunctions involved in this phase:

- `DesignDPA`: computes the matrices (vectors) $(\bar{b}_k, \bar{q}_k, \bar{r}_k, \bar{f}_k, \bar{p}_N, \bar{\gamma}_N)$ in the extended LQ optimal control problem (5.6)-(5.9) (see (3.74)-(3.79)).

  ```
  [bar,qbar,rbar,fbar,pbar,gammabar] = DesignDPA(A,C,Q,S,r,N);
  ```

  The Matlab source code of `DesignDPA` is provided in Appendix C.1.13.

- `solveELQ`: computes the optimal inputs $\{u_k^*\}_{k=0}^{N-1}$ and states $\left\{x_{k+1}^*\right\}_{k=0}^{N-1}$ (see section 3.2.3 Algorithm 3).

  ```
  [x,u] = solveELQ (Qbar,Mbar,Rbar,qbar,rbar,fbar,Abar',...
                      Bbar',bar,X0,Pbar,pbar,gammabar,N,P,L,Z);
  ```

  Matlab source code of `solveELQ` is provided in Appendix C.1.11.

In the last phase of the implementations of DP, the function `MPCPredict` is used to predict future states, $\{x_{k+1}\}_{k=0}^{N-1}$ and CVs, $\{z_{k+1}\}_{k=0}^{N-1}$.

The following is an example, in which we present the three phases for solving the unconstrained LQ output regulation problem (5.1)-(5.3) using DP:

```
%-------------- Design --------------------------------
% define system
nSys = 2;                   % the number of states
% generate a discrete random state space model
sys = drss(nSys);
% retrieve the matrices for model
[A,B,C,D] = ssdata(sys);
sys.d = 0;                      % no disturbance
% weight matrix
Q = 1;
S = 0.0001;
% predictive horizon
N = 50;

% design necessary matrices
[Abar,Bbar,Qbar,Mbar,Rbar,Pbar,P,L,Z] = MPCDesignDP(A,B,C,Q,S);

%-------------- Compute -------------------------------
% start point
x0 = zeros(nSys,1);
u_1 = 0;
% steady state level
xs = zeros(nSys,1);
us = 0;
usN = repmat(us,N,1);
zs = C*xs;
% form variable deviation
dev_x = x0-xs;
dev_u_1 = u_1-us;
% expand state for DP
X = [dev_x; dev_u_1];
% reference
R = 20*ones(10,1);
ref = [R repmat(R(:,end),1,N)];

% compute the optimal input
[u] = MPCComputeDPI(A,C,Q,S,X,usN,Abar,Bbar,...
```

```
                        Qbar,Mbar,Rbar,Pbar,P,L,Z,ref,N);

%-------------- Predict -------------------------------
% predict states and outputs
[Xp,Zp] = MPCPredict(x0,u,N,A,B,C);
```

The function `QPsolver` is available in the MPC toolbox to simulate the behavior of the close loop optimal control system. The following pseudocode provides major part of the algorithm.

```
function [u,y] = DPsolver(A,B,C,Q,S,N,R,x0,u_1,xs,us)
```

The required input arguments are:

```
% A: n by n matrix
% B: n by m matrix
% C: p by n matrix
% Q: weight matrix, symmetric p by p matrix
% S: weight matrix, symmetric m by m matrix
% N: prediction horison, scalor
% R: reference trajectory, p by t matrix
% x0: start state, n by 1 matrix
% u_1: start input, m by 1 matrix
% xs: steady-state value of states, n by 1 matrix
% us: steady-state value of inputs, m by 1 matrix
```

The first step is to initialize the parameters:

```
% identify the number of dimensions
n = size(A);        % dimension of state x
m = size(B,2);      % dimension of input u
p = size(C,1);      % dimension of output z,y
% form deviation variables
dev_u_1 = u_1-us;
dev_x = x0-xs;
dev_z = C*dev_x;
% steady-state level of model responds
zs = C*xs;
% reconstruct needed variables
R = [R repmat(R(:,end),1,N)];
```

```
usN = repmat(us,N,1);
zsN = repmat(zs,N,1);
% reconstruct start state
X = [dev_x;delt_u_1];
```

Then the necessary matrices for online computations are computed

```
[Abar,Bbar,Qbar,Mbar,Rbar,Pbar,P,L,Z] = MPCDesignDP(A,B,C,Q,S);
```

Finally, it is time to simulate the behavior of the closed loop control system:

```
% time sequence
t = 0:length(R)-1;
for k = 1:length(t)
    %------------- Compute On-line  --------------------
    % get N steps ahead of reference and transform it into
    % vertical vectors
    ref = R(:,k:k+N-1);
    % compute optimal inputs sequence up and
    % their deviations dev_u_1
    [up,dev_u_1] = MPCComputeDP(A,C,Q,S,X,usN,Abar,Bbar,...
                          Qbar,Mbar,Rbar,Pbar,P,L,Z,ref,N);

    % store the first block row of the optimal inputs and
    % deviations
    u(:,k) = up(1:m);
    dev_u_1 = dev_u_1(1:m);

    %------------- Predict ----------------------------
    % predict states and outputs
    [Xp,Zp] = MPCPredict(x0,up,N,A,B,C);

    %------------- Update -----------------------------
    % update the current state and the augmented state
    x0 = Xp(1:n);
    dev_x = x0- xs;
    X = [dev_x; dev_u_1];
    % store the first block row of the controlled output
    y(:,k) = Zp(1:p);
end
```

After the reference sequence, `ref`, is updated at the phase `Compute`, the optimal inputs `up` and their deviations `dev_u_1` are calculated. The sequence of the future states `Xp` and the outputs `Zp` are predicted at the phase `Predict`. Finally, the current state `x` and the augmented state `X` are updated and used for the next round of computations. The first block row of the output sequence `Zp` is stored in `y`.

At the end of the function, the sequence of the optimal input `u` and the output `y` are returned.

The Matlab MPC toolbox provides the function `simMPC` to solve the problem (5.1)-(5.3) by either CVP or DP.

The pseudocode of `simMPC` is:

```
function [u,y] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,method)

check input arguments

if method == 1
   [u,y] = SEsolver(A,B,C,Q,S,N,R,x0,u_1,xs,us);
end
if method == 2
   [u,y] = DPsolver(A,B,C,Q,S,N,R,x0,u_1,xs,us);
end
```

The input arguments are examined before solving the problem. Then the function `simMPC` solves the optimal control problem (5.1)-(5.3) by `SEsolver` if `method` is `1`, or by `DPsolver` if `method` is `2`. The Matlab source code of `simMPC` is provided in Appendix C.1.1.

Figure 5.1 illustrates the data flow of the process for solving the output regulation problem (5.1)-(5.3) by CVP and DP. The test systems in Figure 5.1 can be both stable and unstable systems. In this thesis, we consider the stable systems: either single-input single-output (SISO) or multiple-input multiple-output (MIMO) systems in discrete time. The system matrices A, B, C, with other specified parameters, are passed to `SEsolver/DPsolver` by `simMPC`. The closed loop simulation is performed in `SEsolver/DPsolver`. At the end of the process, the sequence of the optimal input u^* and the predicted control output z^* are returned from `simMPC`.

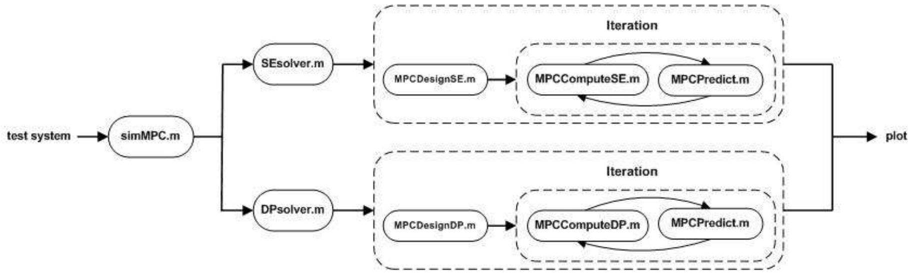

Figure 5.1: Data flow of the process for solving the LQ output regulation problems

5.2.1 Example 1:

This example demonstrates how to solve an unconstrained LQ output regulation problem using the Matlab functions developed in the MPC toolbox.

First, specify a discrete-time plant with one input, one controlled output (SISO), and two states:

```
nSys = 2;
sys = drss(nSys);
% retrieve the system matrices
[A,B,C,D] = ssdata(sys);
% set matrix D to zero since no disturbance exists
sys.d = 0;
```

Convert the system to the zero-pole-gain format:

```
zpk = zpk(sys);
```

That is:

$$G(z) = \frac{-0.47915(z+0.656)}{(z+0.08706)(z+0.963)} \tag{5.10}$$

The step response of the plant is:

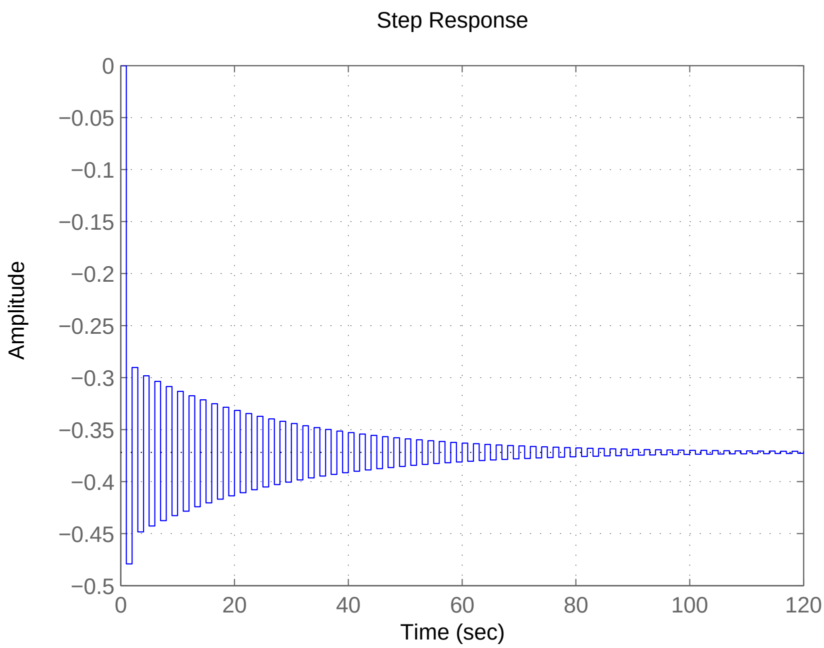

Figure 5.2: Example 1: Step response of the plant

Specify the weight matrices Q, S:

```
Q = eye(1);
S = 0.0001*eye(1);
```

Specify the prediction horizon N, the reference R

```
N = 50;
R = 1*[30*ones(1,50)];
```

Specify the initial state, x_0, and initial input, u_{-1}

```
x0 = zeros(nSys,1);
u_1 = 0;
```

Specify the steady-state values of states, xs, and input, us:

```
xs = zeros(nSys,1);
us = 0;
```

The solution of the unconstrained optimal control problem is

```
% by CVP
[u1,y1] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,1);

% by DP
[u2,y2] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,2);
```

Finally, present the results:

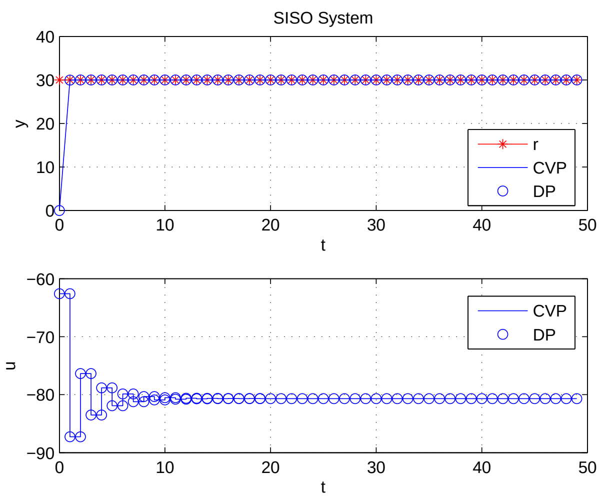

Figure 5.3: Example 1: Solve a LQ output regulation problem by CVP and DP

The upper graph in Figure 5.3 is output profile and the lower one is the input profile. The red line represents reference and the blue line represents the simulation result from DP. The circle represents the simulation result from CVP. It is obvious that both methods give same results. This serves as a verification of correct implementation of the CVP and the DP method.

The complete Matlab code of Example 1 is provided in Appendix C.2.1.

5.3 Implementation of Interior-Point Method

In this section, we describe the implementation of the interior point method. The implementation for solving a convex inequality constrained QP problem, followed by the implementation for solving the constrained LQ output regulation problem (5.1)-(5.5), is presented.

5.3.1 Convex Inequality Constrained QP Problem

The function `InteriorPoint` implements the Interior-Point method for solving a convex inequality constrained QP problem

$$\min \quad \frac{1}{2}x'Hx + g'x \tag{5.11}$$

$$s.t. \quad A'x \geq b \tag{5.12}$$

The following pseudocode describes the major part of the implementation.

```
function [x,lam,info,QPinfo] = InteriorPoint(H,g,A,b)

Initialize start point x, Lagrangian multiplier lam
and the slack variable s

% Calculat residuals rL, rs, rslam and Duality Gap mu
rL = H*x + g - A*lam;
rs = s - A'*x + b ;
rslam = s.*lam;
mu = sum(rslam)/m;

while ~Converged && ite < Max_ite

  Hbar = H + A*diag(lam./s)*A';
  Lbar = chol(Hbar,'lower');

  % Affine Predictor Step
  rbar = A*((rslam - lam.*rs)./s);
  gbar = rL + rbar;
  delt_x_aff = -Lbar'\(Lbar\gbar);
  delt_s_aff = -rs + A'*delt_x_aff;
  delt_lam_aff = -(rslam + lam.*delt_s_aff)./s;
```

```
% Determine the maximum affine step length
alpha_aff = 1;
idx = find(delt_lam_aff < 0);
if ~isempty(idx)
   alpha_aff = min( alpha_aff, min( -lam(idx)./delt_lam_aff(idx) ));
end

idx = find(delt_s_aff < 0);
if ~isempty(idx)
   alpha_aff = min( alpha_aff, min( -s(idx)./delt_s_aff(idx) ));
end

% Compute affine duality gap mua
mua = (lam + alpha_aff*delt_lam_aff)' * (s +  alpha_aff*delt_s_aff)/m;
sigma = (mua / mu)^3; % Centering parameter

% Center Corrector Step
% Modify complementary
rslam = rslam + delt_s_aff.*delt_lam_aff - sigma*mu;

% Center Corrector Step
rbar = A*((rslam - lam.*rs)./s);
gbar = rL + rbar;
delt_x = -Lbar'\(Lbar\gbar);
delt_s = -rs + A'*delt_x;
delt_lam = -(rslam + lam.*delt_s)./s;

% Determine the maximum affine step length
alphamax = 1;
idx = find(delt_lam < 0);
if ~isempty(idx)
   alphamax = min( alphamax, min( -lam(idx)./delt_lam(idx) ));
end

idx = find(delt_s < 0);
if ~isempty(idx)
   alphamax = min( alphamax, min( -s(idx)./delt_s(idx) ));
end
alpha = 0.995*alphamax;

% update x, lam and s
x = x + alpha*delt_x;
lam = lam + alpha*delt_lam;
s = s + alpha*delt_s;
```

```
    % Calculate residuals rL, rs, rslam and Duality Gap mu
    rL = H*x + g - A*lam;
    rs = s - A'*x + b ;
    rslam = s.*lam ;
    mu = sum(rslam)/m;

end
```

The Matlab source code of `InteriorPoint` is provided in Appendix C.1.14.

Example 2:

This example demonstrates how to solve a convex inequality constrained QP problem using the Matlab functions developed in the MPC toolbox.

Suppose a convex QP problem

$$\begin{aligned} \min \quad & \frac{1}{2}x'Hx + g'x \\ s.t. \quad & A'x \geq b \end{aligned}$$

where

```
  H = [1 0; 0 1];
  g = [2; 1];
  A = [-1 0.5 0.5;1 1 -1];
  b = [-1 -2 -1]';
```

Plot the problem

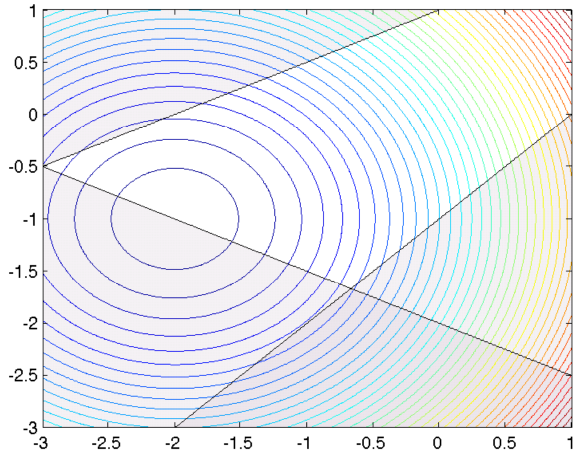

Figure 5.4: Example 2: Convex QP problem

The white part in Figure 5.4 is the feasible area. The gray shadow part is the infeasible area.

We solve the problem by

```
[x,lam,info,QPinfo] = InteriorPoint(H,g,A,b);
```

Finally, plot the solution

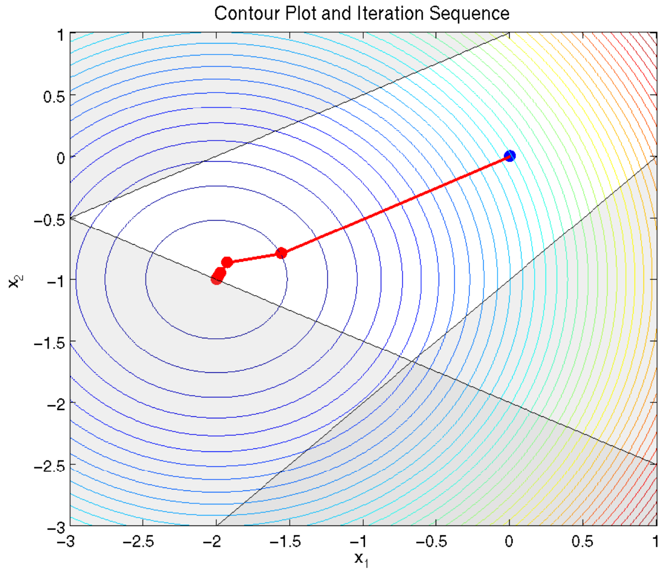

Figure 5.5: Example 2: The optimal solution (contour plot)

Figure 5.5 shows the iteration sequence. The iteration starts from the blue point. The optimal solution is obtained along the red lines.

Plot sequence of x_1 and x_2:

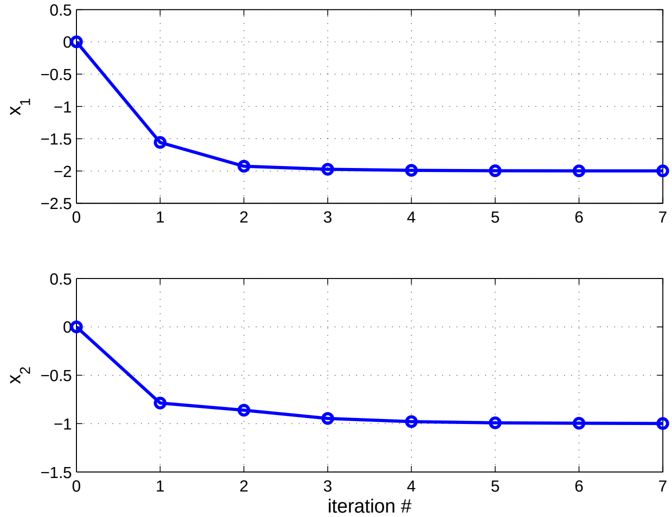

Figure 5.6: Example 2: The optimal solution (iteration sequence)

Figure 5.6 shows that the optimal solution is obtained at iteration 7, where x_1 converges to -2 and x_2 converges to -1.

5.3.2 Constrained Output Regulation Problem

The constrained output regulation problem (5.1)-(5.5) is formulated into an inequality constrained QP problem as

$$\min_{U} \quad \psi = \frac{1}{2}U'HU + g'U \tag{5.13}$$

$$s.t. \quad U_{min} \leq U \leq U_{max} \tag{5.14}$$

$$\Delta U_{min} \leq \Lambda U \leq \Delta U_{max} \tag{5.15}$$

The function `MPCInteriorPoint` implements the Interior-Point method for solving the inequality constrained QP problem (5.13)-(5.15) (see section 4.3 Algorithm 2). The Matlab source code of `MPCInteriorPoint` is provided in Appendix C.1.15.

Example 3:

This example demonstrates how to use the Interior-Point method solve a LQ output regulation problem with input and input-rate constraints.

First, specify a discrete-time plant with one input, one controlled output (SISO) and two states:

```
nSys = 2;
sys = drss(nSys);
% retrieve the system matrices
[A,B,C,D] = ssdata(sys);
% set matrix D to zero since no disturbance exists
sys.d = 0;
```

Specify the weight matrices Q, S:

```
Q = eye(1);
S = 0.0001*eye(1);
```

Specify the prediction horizon N, the reference R

```
N = 50;
R = [0*ones(1,5) 20*ones(1,25) 0*ones(1,15)];
R1 = [R(:,2:end) repmat(R(:,end),1,N)];
```

Specify the initial state, x_0, and initial input, u_{-1}

```
x0 = zeros(nSys,1);
u_1 = 0;
```

Specify the steady-state values of states, xs, input, us and output zs:

```
xs = zeros(nSys,1);
us = 0;
zs = C*xs;
```

The deviation variables are:

```
dev_u_1 = u_1-us;
dev_x = x0-xs;
dev_z = C*dev_x;
```

Specify the input and input-rate constraints:

```
% input constraints
umin = 0;
umax = 25;
Umin = repmat(umin,N,1);
Umax = repmat(umax,N,1);
% input rate constraints
dumin = -10;
dumax = 10;
DUmin = repmat(dumin,N-1,1);
DUmax = repmat(dumax,N-1,1);
```

After the variables are specified, the necessary matrices are computed:

```
[H,Mx0,Mum1,MR,Lambda] = MPCDesignSE(A,B,C,Q,S,N);
```

The behavior of the closed loop control system is simulated using:

```
t = 0:length(R)-1; % time sequence
for k = 1:length(t)
    %------------ Compute On-line  -------------------
    % get N steps ahead of reference and transform it into
    % vertical vectors
    ref = R1(:,k:k+N-1); ref = ref(:);

    % calculate the gradient
    g = Mx0*dev_x + MR*(ref-zsN) + Mum1*dev_u_1;

    % update input restriction
    Umin(1) = max(umin, dumin+dev_u_1);
    Umax(1) = min(umax, dumax+dev_u_1);

    % compute optimal inputs deviations dev_u_1
    [dev_u_1] = MPCInteriorPoint(H,g,Lambda',...
                                Umin,Umax,DUmin,DUmax);
```

```
    % convert to physical inputs and store the first element of
    % the optimal inputs and deviations
    up = dev_u_1+usN;  % physical input
    u(k) = up(1);
    dev_u_1 = dev_u_1(1);

    %------------ Predict ----------------------------
    % predict states and outputs
    [Xp,Zp] = MPCPredict(x0,up,N,A,B,C);

    %------------ Update -----------------------------
    % update the current state
    x0 = Xp(1);
    dev_x = x0 - xs;

    % store the first block row of the controlled output
    y(k) = Zp(1);
end
```

For each sampling step, we do the following: The reference sequence, `ref`, is primarily updated. Then the gradient `g` is calculated. The input restrictions `Umin` and `Umax` are updated. The optimal input deviations `dev_u_1` are calculated by the function `MPCInteriorPoint`. The future states sequence `Xp` and the outputs sequence `Zp` are predicted at the phase `Predict`. Finally, the current state `x0` and its deviation `dev_x` are updated and used for the next round of computations. The first element of the output sequence `Zp` is stored in `y`.

After the simulation, the sequence of the optimal inputs and controlled outputs are plotted in graphs.

The upper graph in Figure 5.7 is the profile of the outputs, in which the red stars are the reference and the blue line is the controlled outputs. There are steady state errors between controlled outputs and the reference due to active input constraints. The upper limit of the input is 25, and the reference is not reachable within this limit. The lower graph in Figure 5.7 is the profile of the optimal inputs. The changes of the optimal inputs are restricted to between -10 and 10.

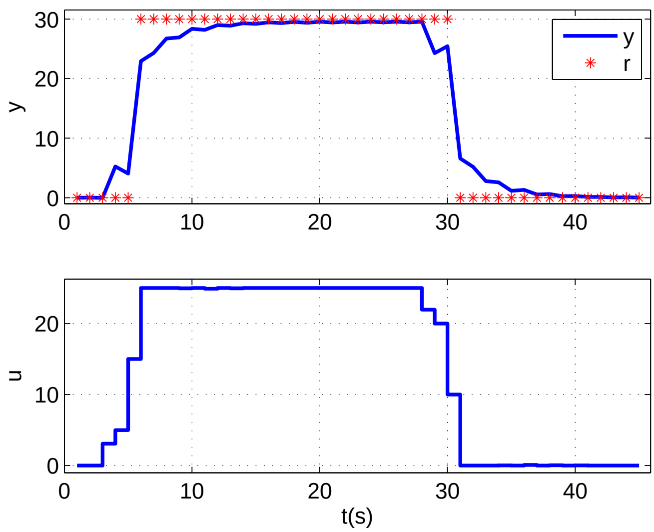

Figure 5.7: Example 3: The solutions

Chapter 6

Simulation

In this chapter, we perform simulations by using the Matlab functions built in Chapter 5. The goals of the simulations are

- To verify correctness of the implementations. We test the performances of the methods and compare the results obtained by different methods.
- To verify the theoretical analysis stated in the previous chapters, such as: (1) the computational time that different methods spend on solving unconstrained or constrained LQ outpt regulation problems; (2) the factors that affect the computational time.

6.1 Performance Test

The purpose of this section is to verify the implementations of CVP and DP. The performances of the two methods are examined using a single-input single-output (SISO) system and a 2×2 multiple-input multiple-output (MIMO) system as test systems.

The stable discrete-time models may be generated randomly by the Matlab function `drss`. We assume the internal model of the MPC controller is an ideal model. Therefore the internal model is the same as the plant. Thus, the state-space model generated by Matlab represents both the internal model and the plant.

To describe the model conveniently, the state-space model may be represented by an input-output transfer function

$$G(z) = \frac{Y(z)}{U(z)} \tag{6.1}$$

where $Y(z)$ is the z-transform of the output $y(k)$, $U(z)$ is the z-transform of the input $u(k)$ and $G(z)$ is the transfer function of the model.

Test 1: We start with testing the performance of two methods on a 2-state SISO system. A 2-state stable discrete-time model is generated by Matlab. The transfer function of the model is:

$$G(z) = \frac{-0.1364(z - 0.5416)}{(z + 0.9607)(z + 0.6762)} \tag{6.2}$$

There is one zero and two poles. The system is stable because the zero and poles are inside the unit circle in the z-plane.

Figure 6.1 shows the step response of the system.

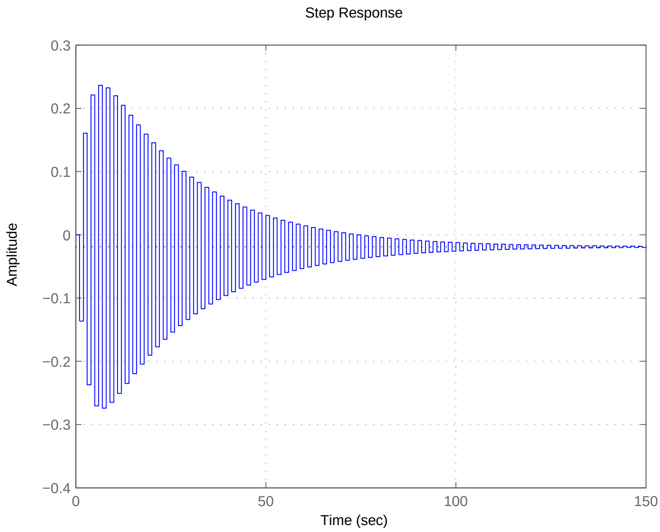

Figure 6.1: Performance test 1: Step response of 2-state SISO system

The given reference is 30. The predictive horizon is set to be 50. The initial states and inputs are set to be zeros. All the related parameters are listed as below:

system	SISO
number of states	2
weight Matrix	$Q = 1$
weight Matrix	$S = 0.0001$
reference	$R = [30\ 30\ ...\ 30]$
predictive horizon	$N = 50$
start state	$x_0 = [0\ 0]'$
steady-state value of state	$x_s = [0\ 0]'$
start input	$u_{-1} = 0$
steady-state value of input	$u_s = 0$

To solve the unconstrained output regulation problem, the parameters above are passed to the function `SEsolver` and the function `DPsolver` through the function `simMPC`. The function `SEsolver` solves the problem by CVP, and the function `DPsolver` solves the problem by DP. The Matlab code is provided in Appendix C.3.1. The optimal results from the two methods are plotted in

Figure 6.2.

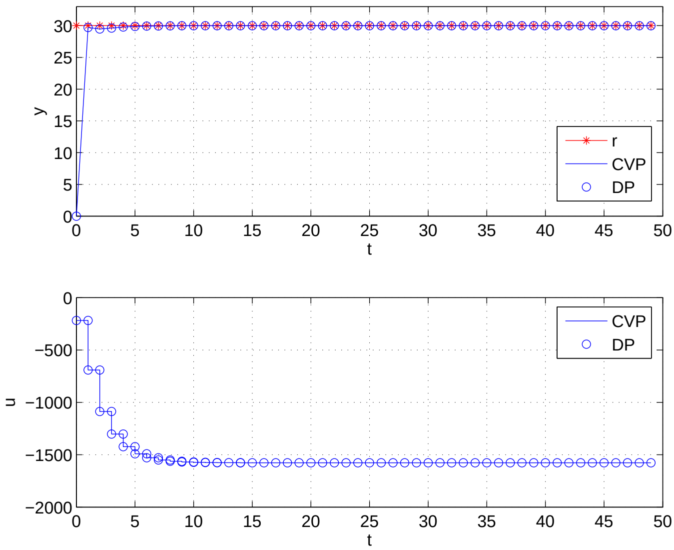

Figure 6.2: Performance test 1: The optimal solutions

The optimal inputs are shown in the lower graph and the corresponding system outputs are shown in the upper graph. It is clear that the results from the two methods are identical. This indicates that two methods are correctly implemented.

Test 2: The performance of two methods is examined on a 4-state stable 2×2 MIMO discrete-time model. The transfer functions of a 2×2 MIMO model can be expressed as

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \tag{6.3}$$

where Y_1 and Y_2 are the outputs, U_1 and U_2 are the inputs. $\begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix}$ is the transfer function matrix. Each entry of the transfer function matrix is the transfer function relationship between a particular input and a particular output, e.g. G_{11} relates U_1 to Y_1, G_{21} relates U_1 to Y_2.

A 4-state MIMO dicrete-time state space model is generated by Matlab. The transfer functions between each individual input, and each individual output are

$$\begin{aligned}
G_{11}\left(\frac{Y_1}{U_1}\right) &= \frac{0.51845(z-0.4358)(z^2+0.4155z+0.193)}{(z+0.5947)(z-0.3443)(z^2+0.1336z+0.7161)} \\
G_{21}\left(\frac{Y_2}{U_1}\right) &= \frac{1.1978(z-0.3424)(z^2+1.337z+0.7162)}{(z+0.5947)(z-0.3443)(z^2+0.1336z+0.7161)} \\
G_{12}\left(\frac{Y_1}{U_2}\right) &= \frac{0.081384(z+0.0342)(z^2-0.6739z+2.73)}{(z+0.5947)(z-0.3443)(z^2+0.1336z+0.7161)} \\
G_{22}\left(\frac{Y_2}{U_2}\right) &= \frac{0.1501(z-4.674)(z-0.3113)(z+0.1669)}{(z+0.5947)(z-0.3443)(z^2+0.1336z+0.7161)}
\end{aligned} \tag{6.4}$$

Figure 6.3 shows the step response of the system.

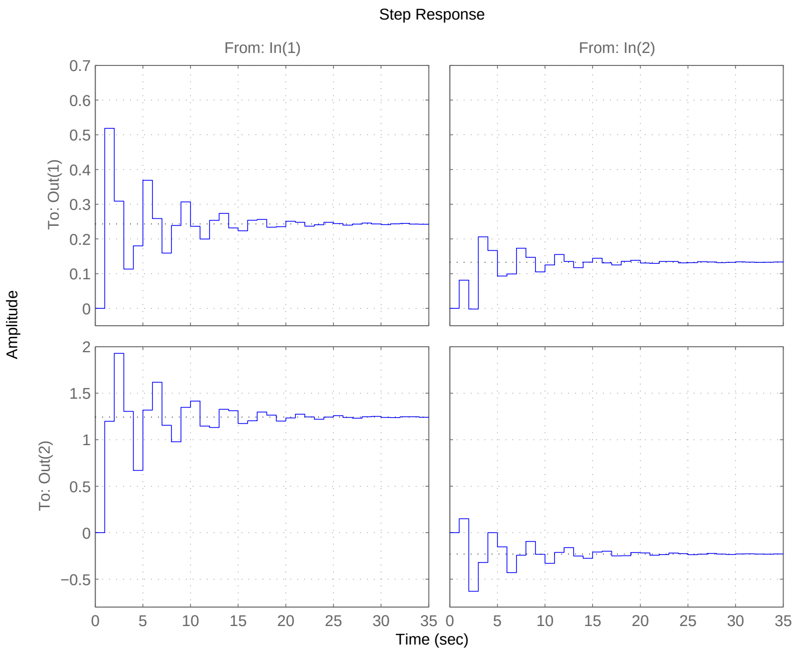

Figure 6.3: Performance test 2: Step response of 4-state 2x2 MIMO system

In this test the reference varies with respect to time. All parameters are listed as below:

system	2×2 MIMO
number of states	4
weight Matrix	$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
weight Matrix	$S = 0.0001 * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
reference	$\mathbf{R = [30 ... 30 \quad 40 ... \ 40 \quad 20 \ ... \ 20]}$
predictive horizon	$N = 50$
start state	$x_0 = [0\ 0\ 0\ 0]'$
steady-state value of state	$x_s = [0\ 0\ 0\ 0]'$
start input	$u_{-1} = [0\ 0]'$
steady-state value of input	$u_s = [0\ 0]'$

We solve the unconstrained output regulation problem by CVP and DP. The Matlab code is provided in Appendix C.3.2. The optimal solution is plotted in Figure 6.4. The optimal inputs u_1 and u_2 are shown in the two lower graphs. The corresponding system outputs, y_1 and y_2, are in the two upper graphs.

Figure 6.4 shows that the results from the two methods are identical.

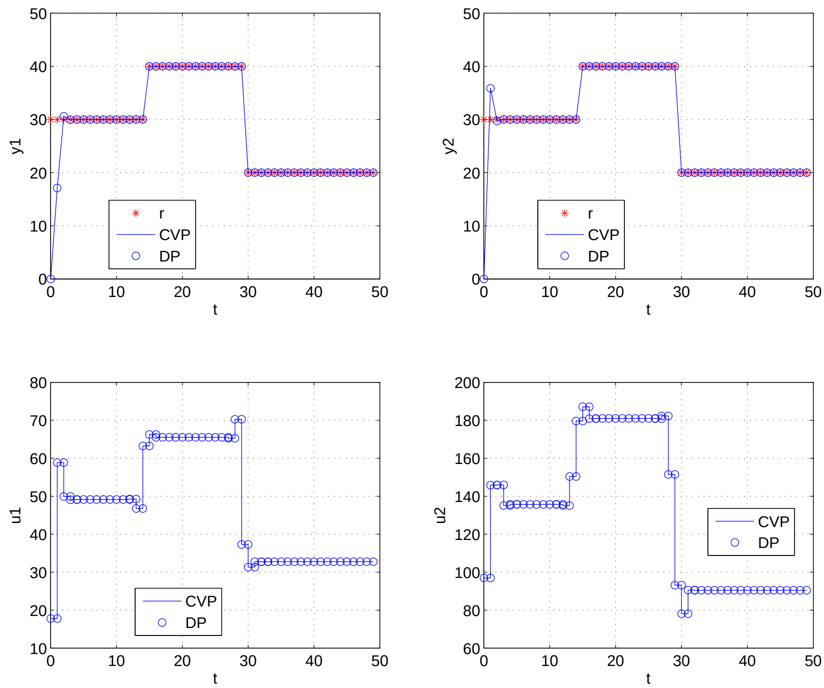

Figure 6.4: Performance test 2: The optimal solutions

The simulation results shown here demonstrate that both CVP and DP yield the same solution in terms of a same optimal control problem. Importantly, these data suggest that the implementations we established are correct.

6.2 Computational Time Study

We investigate the computational time for CVP and DP in this section. As we stated in Chapter 5, the computations of the optimal solution involve two steps: Design of the necessary matrices and the optimal input computation. Therefore, the combined computational time for the design of the matrices and the optimal input computation is measured for each method. The optimal input is computed online. Since what we are most interested in is the online computations, the computational time for the optimal input computation is measured as well. The measurement is performed by using the Matlab function `cputime`.

As discussed in Chapter 2, the computational complexity of CVP is $\mathcal{O}((mN)^3)$, meaning that the computational time for CVP is cubic in both the predictive horizon and the number of inputs. The computational complexity of DP, which is discussed in Chapter 3, is $\mathcal{O}(N(n^3 + m^3))$. The computational time for DP is linear in the predictive horizon, and cubic in both the number of states and the number of inputs. Three parameters, which are the predictive horizon, N, the number of states, n, and the number of inputs, m, influence the computational time to solve the optimal control problem. Here, the effects of the three parameters on the computational time are investigated.

6.2.1 CPU Time vs. N

In this section, a 2-state SISO system is employed to investigate the effect of the predictive horizon N on the computational time for the two methods. The parameters are set as below

system	SISO
number of states	$n = 2$
number of inputs	$m = 1$
weight matrix	$Q = 1$
weight matrix	$S = 0.0001$
reference	$R = 1$
predictive horizon	**N = 10:10:500**
start state	$x_0 = [0\ 0]'$
steady-state value of state	$x_s = [0\ 0]'$
start input	$u_{-1} = 0$
steady-state value of input	$u_s = 0$

The predictive horizon N is set to be 10 to 500, and the reference R is set to be a scalar since the length of reference does not influence the result. We measure the CPU time for solving the unconstrained LQ output regulation problem with different N. The Matlab code is provided in Appendix C.3.3.

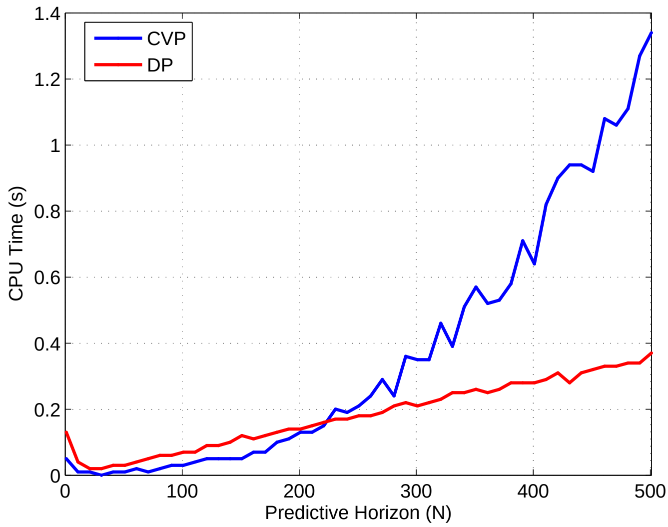

Figure 6.5: CPU time vs N (n=2, m=1)

Figure 6.5 plots the combined computational time, and Figure 6.6 plots the online computational time. Both figures show that the CPU time for DP is linear in the predictive horizon, while the CPU time for CVP is cubic in the predictive horizon. This result is identical with the complexity analysis, in which the computational complexity of DP is $\mathcal{O}(N)$, and the computational complexity of CVP is $\mathcal{O}(N^3)$.

As shown in Figure 6.6, the two curves of the online CPU time intersect at N equals 340. When N is less than 340, the online CPU time for CVP is smaller. When N is greater than 340, the online CPU time for DP is smaller. That is, CVP is efficient for solving the optimal control problem with N less than 340, and DP is efficient for solving the optimal control problem with N greater than 340.

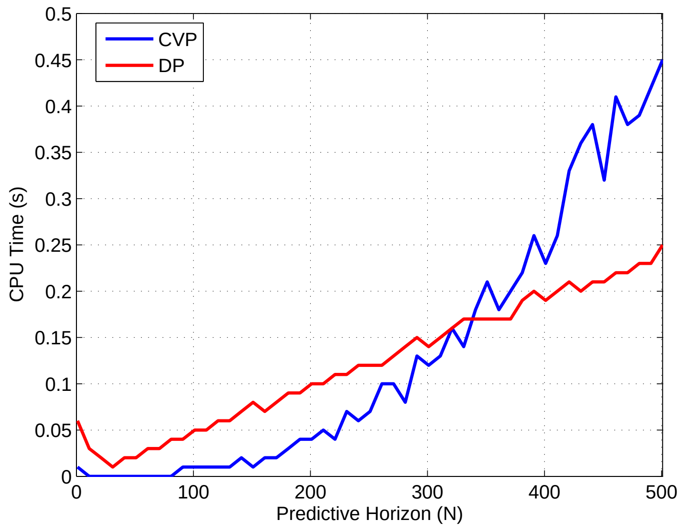

Figure 6.6: Online CPU time vs N (n=2, m=1)

6.2.2 CPU Time vs. n

The effect of the number of states on the computational time is investigated in this section. The SISO system is used to test the computational time. The number of states changes from 10 to 300. The parameters are set as below

system	SISO
number of states	**n = 10:10:300**
number of inputs	$m = 1$
weight matrix	$Q = 1$
weight matrix	$S = 0.0001$
reference	$R = 1$
predictive horizon	$N = 100$
start state	$x_0 = [0 \ldots 0]'$
steady-state value of state	$x_s = [0 \ldots 0]'$
start input	$u_{-1} = 0$
steady-state value of input	$u_s = 0$

We measure the computational time for solving the unconstrained LQ output regulation problem with different number of states. The Matlab code is provided in Appendix C.3.4.

The curves in Figure 6.7 (combined computational time) and Figure 6.8 (online computational time) indicate that the computational time for DP is cubic in the number of states. It also indicates that the computational time for CVP is independent of the number of states. This result is identical with the previous complexity analysis result, in which the computational complexity of DP is $\mathcal{O}(n^3)$, and the number of states has no effect on the complexity of CVP.

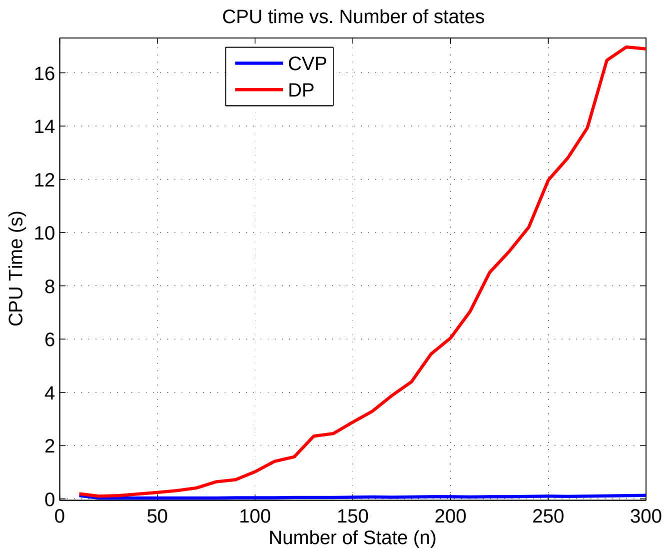

Figure 6.7: CPU time vs. n (N=100, m=1)

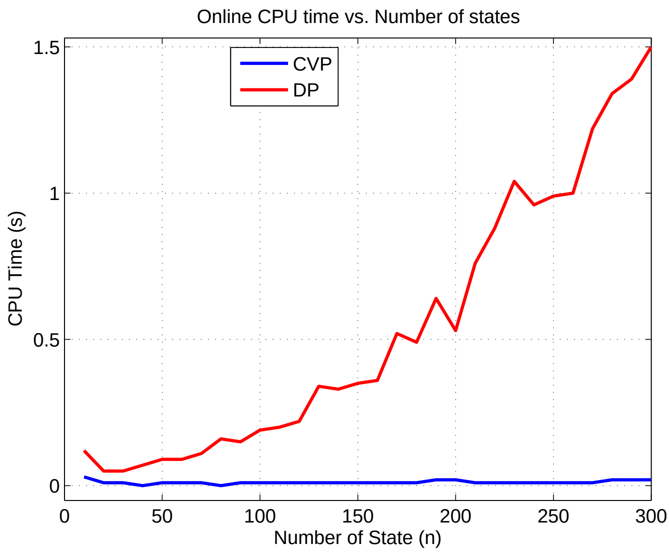

Figure 6.8: Online CPU time vs. n (N=100, m=1)

6.2.3 CPU Time vs. m

The effect of the number of inputs on the computational time is investigated in this section. The MIMO system is used to test the computational time. The number of inputs varies from 5 to 100. The parameters are set as below

system	MIMO
number of states	$n = 2$
number of inputs	**m = 5:5:100**
weight matrix	$Q = 1$
weight matrix	$S = 0.0001 * \begin{bmatrix} 1 & \cdots \\ \cdots & 1 \end{bmatrix}$
reference	$R = 1$
predictive horizon	$N = 50$
start state	$x_0 = [0\ 0]'$
steady-state value of state	$x_s = [0\ 0]'$
start input	$u_{-1} = [0\ ...\ 0]'$
steady-state value of input	$u_s = [0\ ...\ 0]'$

We measure the computational time required to solve the unconstrained LQ output regulation problem with different number of inputs. The Matlab code is provided in Appendix C.3.5.

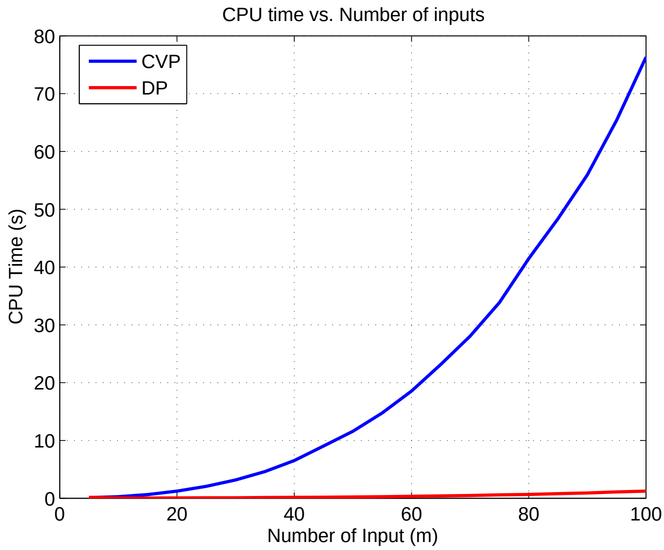

Figure 6.9: CPU time vs. m (N=50, n=2)

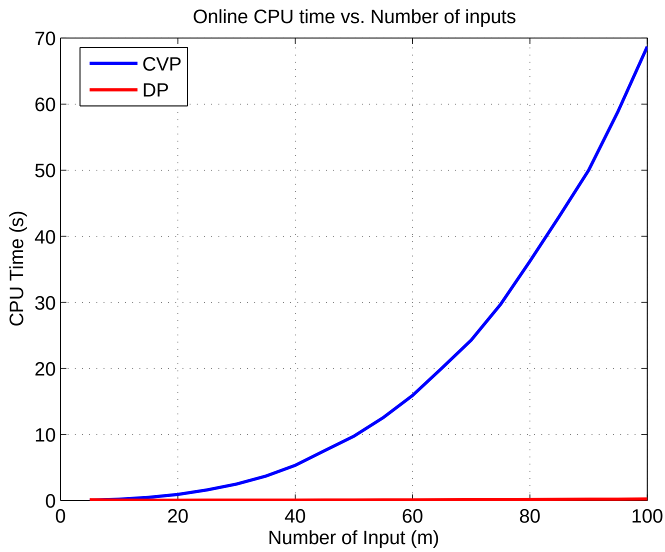

Figure 6.10: Online CPU time vs. m (N=50, n=2)

Figure 6.9 and 6.10 indicate that the computational time for CVP is cubic in the number of inputs. The computational time for DP is almost constant with respect to the number of inputs. This result seems different from our previous analysis, in which the computational time for DP is cubic in the number of inputs. However, as shown in Figure 6.11, with a more appropriate y-axis, the computational time is cubic in the number of inputs as expected by the complexity analysis.

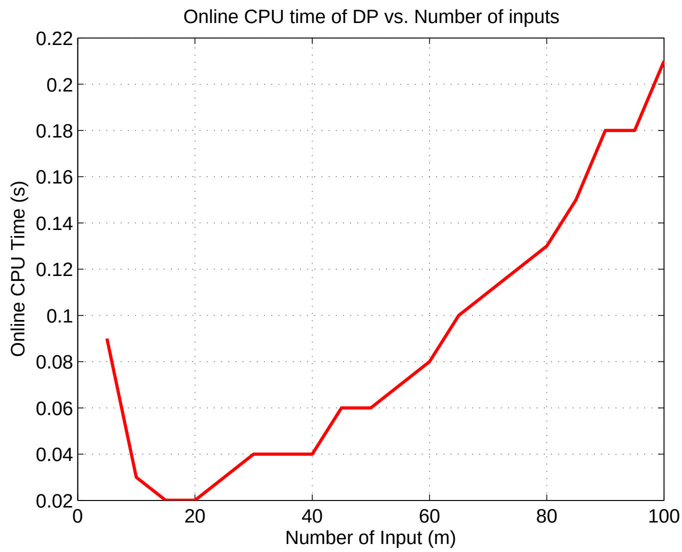

Figure 6.11: Online CPU time for DP vs m (N=50, n=2)

6.2.4 Combined effect

As discussed in the previous sections, the computational time for CVP depends on both the number of states and the predictive horizon. For DP, the computational time depends on the number of states, the number of inputs and the predictive horizon. In the following, we examine the total effects of the three factors on the computational time.

Define

$$\text{Ratio} = \frac{T_{CVP}}{T_{DP}} \tag{6.5}$$

in which T_{CVP} is the online computational time for CVP, T_{DP} is the online computational time for DP.

The online computational time for the two methods is same when Ratio is equal to 1. CVP is more efficient when Ratio is smaller than 1, and DP is more efficient when Ratio is greater than 1.

Figure 6.12 and 6.13 show the simulation results of Ratio versus the predictive horizon as the number of inputs varies from 1 to 5. Its Matlab code is provided in Appendix C.3.6.

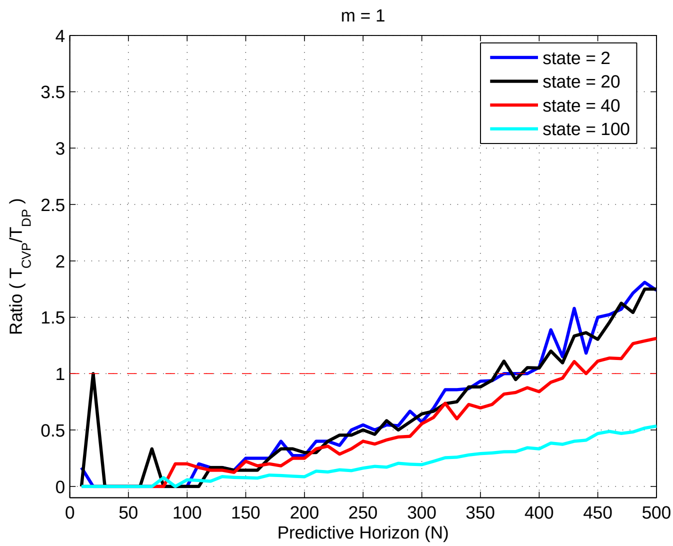

Figure 6.12: Combined effect (m=1)

The curves with different colors represent different number of states, n = 2, 20, 40, 100. A red dashed line is draw at the position where Ratio equals 1. There is an intersection point between each colored curve and the red dashed line. We define the predictive horizon at the intersection point as N_0. N_0 divides the predictive horizon into two parts: the short predictive horizon and the long predictive horizon. The predictive horizon with $N < N_0$ is the short predictive horizon. The predictive horizon with $N > N_0$ is the long predictive horizon. In the short predictive horizon, the curves are below the red dashed line. CVP is more efficient than DP. In the long predictive horizon, the curves are above the red dashed line. DP is more efficient than CVP.

As shown in Figure 6.12, N_0 increases with the number of states. For example, N_0 is around 370 for state being 2, and N_0 is around 420 for state being 40.

Comparing 6.13 with 6.12, N_0 decreases when the number of inputs increases from 1 to 5. It means that the short predictive horizon shrinks and the long predictive horizon expands as the number of inputs increases.

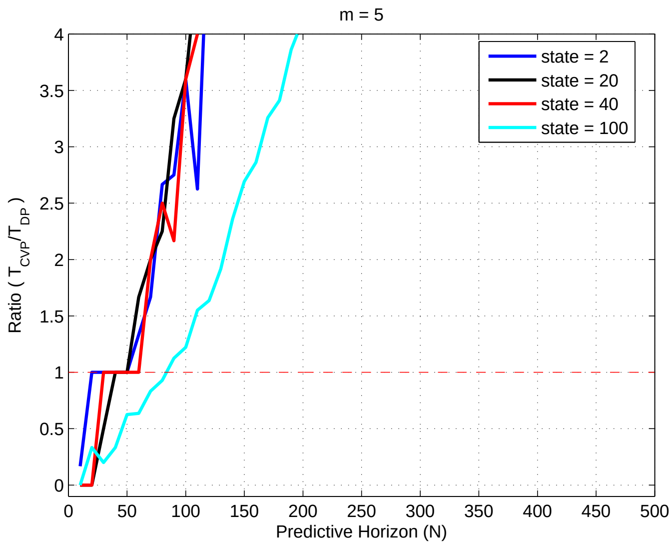

Figure 6.13: Combined effect (m=5)

In conclusion, the relative predictive horizon determines which one of the two methods is more efficient than the other. For instance, as shown in Figure 6.12, $N = 500$ is a long predictive horizon for the system with 2 states. While it is a short predictive horizon for the system with 100 states. Consider another instance, $N = 200$ is a short predictive horizon for the systems with input $= 1$, as shown in Figure 6.12, while it is a long predictive horizon for the systems with input$= 5$, as shown in Figure 6.13. Therefore, the three factors, which are the predictive horizon, the number of states and the number of inputs, have to be taken into account simultaneously when choosing one of the two methods for a practical application.

6.3 Interior-Point Algorithm for MPC

In this section, we test the implementation of the Interior-Point method for the LQ output regulation problem with input and input rate constraints. We also investigate the computational time for solving the constrained LQ output regulation problem.

6.3.1 Performance test

The purpose of this subsection is to verify the implementation of the Interior-Point method for the constrained LQ output regulation problem. We test the implementation on a 2-state SISO stable discrete-time system. The transfer function of the system is

$$G(z) = \frac{1.6944(z - 0.5579)}{(z + 0.9607)(z - 0.6762)} \tag{6.6}$$

The step response of the system is shown in Figure 6.14.

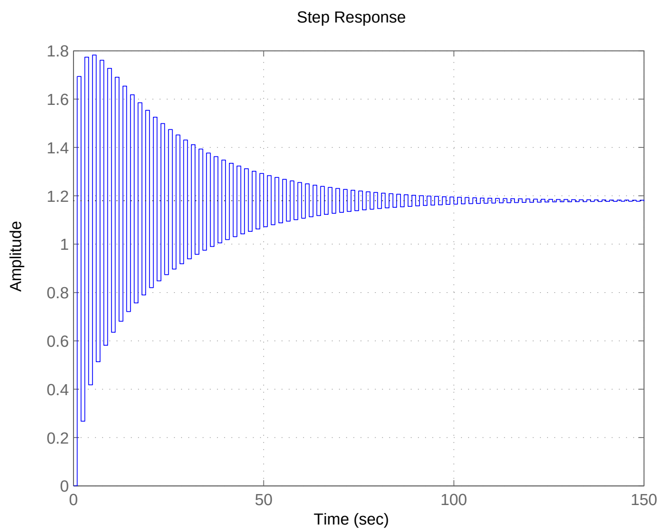

Figure 6.14: Performance test on the Interior-Point method: Step response of 2-state SISO system

Suppose the lower limit and the upper limit of inputs are 0 and 100, respectively. The input rate is from -10 to 10. All the related parameters are listed as below:

system	SISO
number of states	2
number of inputs	$m = 1$
weight Matrix	$Q = 1$
weight Matrix	$S = 0.0001$
reference	$R = [0...0\ 40...40\ 0...0]$
predictive horizon	$N = 50$
input	[0 100]
input rate	$[-10\ 10]$

we use `MPCInteriorPoint` to solve the constrained optimal control problem. The Matlab code is provided in Appendix C.3.8 . Figure 6.15 shows the results. The upper graph is the profile of the system outputs and the lower graph is the profile of the optimal inputs.

As shown in lower graph of Figure 6.15, the inputs vary within the range 0 to 21. The input rate is between -10 and 10. Both input and input rate satisfy the constraints.

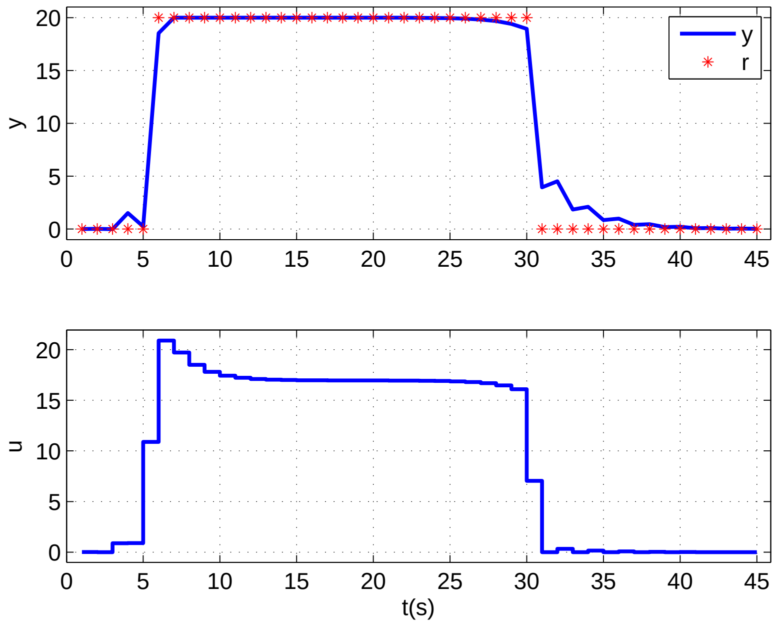

Figure 6.15: Performance test on the Interior-Point method: Input constraint unactive

Next, we set the upper limit of inputs as 15. Figure 6.16 shows the result of the outputs sequence and inputs sequence. As shown in the upper graph, a steady state error between the output and the reference shows up because of the restricted input.

The results of simulations above suggest that the implementation of the Interior-Point method for the LQ output regulation problem with input and input rate constraints is correct.

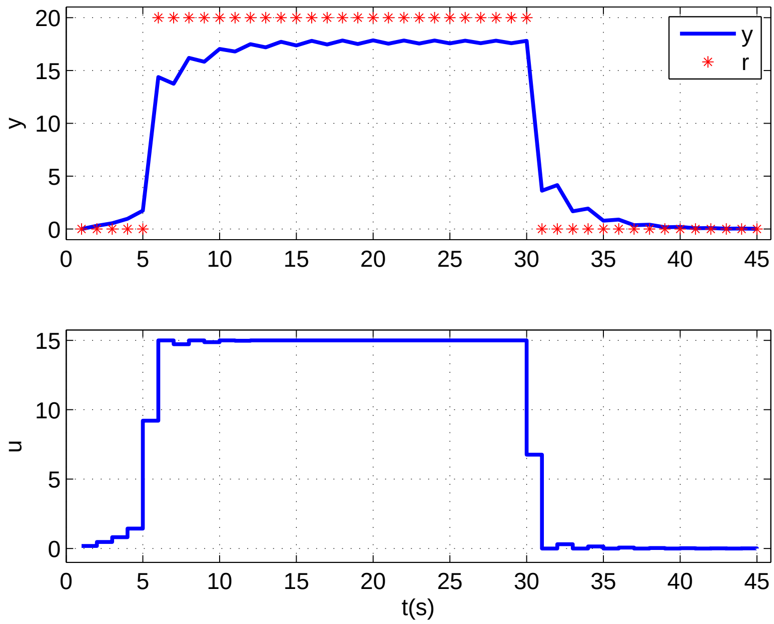

Figure 6.16: Performance test of the Interior-Point method: Input constraint active

6.3.2 Computational Time Study

In Chapter 4, we analyze the computational complexity of the Interior-Point method for solving the constrained optimal control problem arising by CVP. The computational complexity is $\mathcal{O}(m^3N^3)$. The computational time is proportional to

$$\text{Number of iteration } \times m^3N^3$$

In this section, we investigate the effect of the predictive horizon, N, and the number of inputs, m, on the computational time for solving the constrained optimal control problem.

CPU Time vs. N

We consider the system given in 6.3.1 again. In order to investigate the effect of the predictive horizon on the computational time, the predictive horizon is varied from 10 to 300.

As shown in Figure 6.17, it is clear that the computational time is cubic in the predictive horizon.

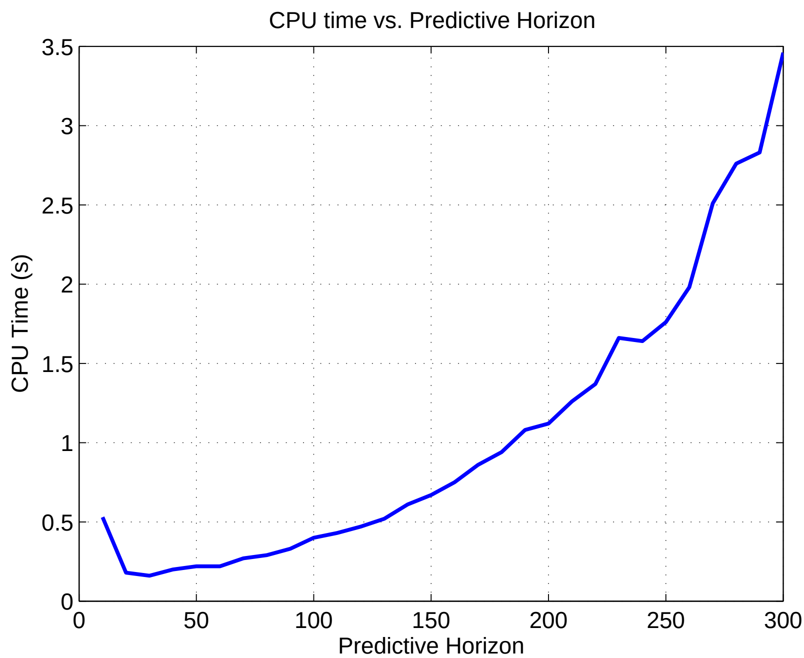

Figure 6.17: CPU time vs. N (n=2, m=1)

CPU Time vs. m

The other factor that may affect the computational time is the number of inputs. So we vary the number of inputs from 1 to 10 in the system 6.3.1. Figure 6.18 shows the result.

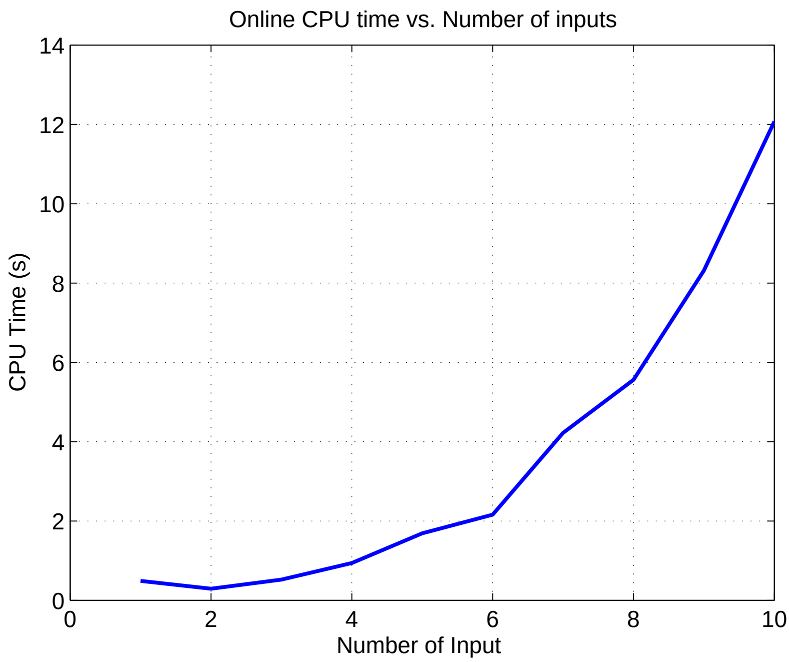

Figure 6.18: CPU time vs. m (N=50, n=2)

Apparently, the computational time is cubic in the number of inputs.

In summary, the computational time for the constrained optimal control problem arising from CVP is cubic in both the predictive horizon and the number of inputs. The simulation results agrees with the previous theoretical analysis.

Chapter 7

Conclusion

The solution of the unconstrained output regulation problem by CVP and DP is investigated in this thesis. We also study the Interior-Point method for the constrained optimal control problem arising by CVP.

CVP formulates the unconstrained LQ output regulation problem as a dense QP problem by eliminating the states. DP formulates the unconstrained LQ output regulation problem as an extended LQ optimal control problem. The extended LQ optimal control problem is solved by DP based on the principle of optimality.

The predictive horizon, together with the number of inputs and the number of states, are the three main factors that influence the computational efficiencies of CVP and DP. The computational time for CVP is cubic in both the predictive horizon and the number of inputs. The computational time for DP is linear in the predictive horizon, cubic in both the number of inputs and the number of states. The efficiency of the methods depends on the combined effect of the three factors. DP is more efficient for the optimal control problem with a relative long predictive horizon, while CVP is more efficient for the optimal control problem with a relative short predictive horizon.

In order to solve an LQ output regulation problem with input and input rate constraints, we use CVP to construct an inequality constrained QP problem.

Based on Mehrotra's predictor-corrector method, the Interior-Point method is developed to solve the inequality constrained QP problem. The computational time required to solve the optimal control problem arising by CVP is cubic in both the predictive horizon and the number of inputs.

In this thesis, we develop the MPC toolbox in Matlab. The MPC toolbox provides the functions to implement the methods discussed in this thesis. It also provides the functions for closed loop MPC simulations.

In practice, the theoretical investigations we made on the CVP and DP method in this thesis, may help in choosing the efficient method to solve the different optimal control problems. The MPC toolbox can be used to forecast and compare the results of different methods by simulations.

Applying the Interior-Point algorithm to solve the optimal control problem with output constraints could be the future direction of this thesis project. [8] and [6] provide the technology for the output constrained optimal control problem. In this thesis, we develop the Interior-Point algorithm for solving the constrained optimal control problem arising by CVP. It would be interesting to use DP to solve the constrained optimal control problem. [15] provides a discussion on related issues.

The MPC toolbox can be extended in several ways. Most importantly, it can be extended with a state estimation which is based on output feedback. Secondly, it can be extended to include stochastic process and measurement noise. Thirdly, it can be extended to the optimal control problem with soft output constraints.

Appendix A

Some Background Knowledge

A.1 Convexity

Definition 1: Convecity of a set
The set $D \subseteq R^n$ is convex if the following holds for arbitrary $x, y \in D$,

$$\theta x + (1-\theta) y \in D, \text{ for all } \theta \in [0,1].$$

Definition 2: Convecity of a function
Let $D \subseteq R^n$ be convex. The function f is convex on D if the following holds for arbitrary $x, y \in D$,

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y) \text{ for all } \theta \in [0,1].$$

f is strictly convex on D if

$$f(\theta x + (1-\theta)y) < \theta f(x) + (1-\theta) f(y) \text{ for all } \theta \in [0,1].$$

Theorem 1
If $D \subseteq R^n$ is convex and f is twice differentiable on D, then

1. f is convex on $D \Leftrightarrow \bigtriangledown^2 f(x)$ is positive semidefinite for all $x \in D$.
2. f is strictly convex on $D \Leftrightarrow \bigtriangledown^2 f(x)$ is positive definite for all $x \in D$.

Theorem 2. First sufficient condition
If D is bounded and convex and if f is convex on D, then

f has a unique global minimizer in D.

Theorem 3
If f is twice differentiable on $x \in D$, then

1. f is convex at $x \Leftrightarrow \bigtriangledown^2 f(x)$ is positive semidefinite.
2. f is strictly convex at $x \in D$ if $\bigtriangledown^2 f(x)$ is positive definite.

[7] provides further information.

A.2 Newton Method

Newton Method, also called the Newton-Raphson method, is a technique to find the approximated roots of the equation $f(x) = 0$, when the analytic solution is difficult or impossible to be obtained.

The procedure of Newton method is demonstrated in Figure A.1.

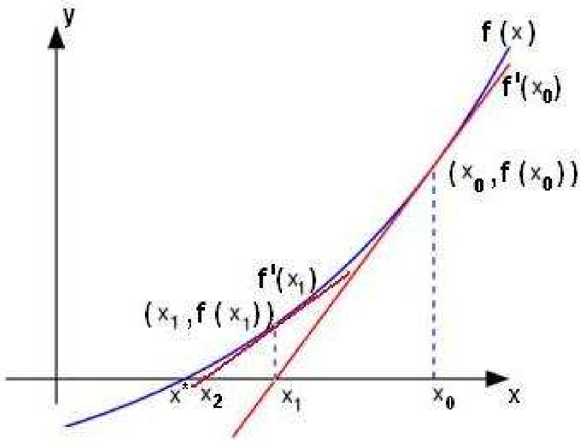

Figure A.1: Newton Method

Suppose we are looking for the root x^* of equation $y = f(x)$ (the blue curve in the graph) , which is the intersection between $y = f(x)$ and $y = 0$. we start with an initial approximition x_0. The tangent line of f at point x_0 is $f'(x_0)$ (the red straight line in the graph) . The tangent line $f'(x_0)$ intersects with x-axis at x_1. Thus the slope of $f'(x_0)$ is expressed as

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1} \tag{A.1}$$

Assume $f'(x_0) \neq 0$, then

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{A.2}$$

x_1 is closer to x^* than x_0. Define Δx as $\Delta x = x_1 - x_0$, (A.2) may be written in

$$\Delta x = -\frac{f(x_0)}{f'(x_0)} \tag{A.3}$$

and

$$f'(x_0)\ \Delta x = -f(x_0) \tag{A.4}$$

In the same way, the tangent line of f at point x_1 is found out to be $f'(x_1)$, and the intersect point between $f'(x_1)$ and x-axis may be calculated to be x_2. x_2 is closer than x_1 to the solution x^*. The improvement of the approximation is $\Delta x = x_2 - x_1$. Repeat this procedure untill Δx converges to zero.

A.3 Lagrangian Function and Karush-Kuhn-Tucker Conditions

Consider the problem

$$
\begin{array}{lll}
 & & z = \min f(x) \qquad \text{(A.5)} \\
(P) & s.t. & g(x) \geq 0 \qquad \text{(A.6)} \\
 & & x \in R^n \qquad \text{(A.7)}
\end{array}
$$

Introduce any value $\lambda \geq 0$ and we define the Lagrangian function as

$$L(x, \lambda) = f(x) - \lambda g(x) \qquad \text{(A.8)}$$

and the problem

$$
\begin{array}{llcl}
 & z(\lambda) & = & \min L(x, \lambda) \qquad \text{(A.9)} \\
(P(\lambda)) & s.t. & & x \in R^n \qquad \text{(A.10)}
\end{array}
$$

Problem $P(\lambda)$ is a relaxation of problem P. Thus $z(\lambda) \leq z$ and the optimal solution of $z(\lambda)$ is the lower bound on the optimal solution of problem P. To solve the best (biggest) lower bound over the all possible value for λ, we solve

$$
\begin{array}{ll}
 & V_d = \max z(\lambda) \qquad \text{(A.11)} \\
(LD) & \lambda \geq 0 \qquad \text{(A.12)}
\end{array}
$$

From (A.9), if $g(x) = 0$, there is no restriction on the sign of λ, problem LD becomes

$$V_d = \max z(\lambda) \qquad \text{(A.13)}$$

Solving the problem $P(\lambda)$ may find out the optimal solution of the problem P.

Proposition If $\lambda \geq 0$,
(1) $x(\lambda)$ *is an optimal solution of* $z(\lambda)$, *and*
(2) $g(x) \geq 0$, *and*
(3) $g(x) = 0$ *whenever* $\lambda > 0$
then $x(\lambda)$ is optimal in P.

Proof.
(1) $\Rightarrow V_d \geq z(\lambda) = f(x) - \lambda g(x)$
(3) $\Rightarrow f(x) - \lambda g(x) = f(x)$
(3) $\Rightarrow x(\lambda)$ is feasible in P
Therefore, $V_d \geq f(x) - \lambda g(x) = f(x) \geq z$. Since $V_d \leq z$, equality holds and x_λ is optimal in P.

Furthermore, $x(\lambda)$, the optimal solution of $z(\lambda)$ can be obtained by

$$\nabla_x z(\lambda) = 0 \tag{A.14}$$

Consequently, we have the following theorem, which is called first order necessary optimality conditions, also called Karush-Kuhn-Tucker (KKT) conditions.

Theorem
For optimal problem P, if x^* is a local constrained minimizer, then there exist Lagrangian multiplier λ such that
(1) $\nabla_x L(x^*, \lambda^*) = 0$
(2) $g(x^*) \geq 0$
(3) $\lambda^* g(x^*) = 0$

A.4 Cholesky Factorization

Cholesky factorization is one of the most important factorizations which can solve linear system equations efficiently. A symmetric positive definite matrix A is decomposed into a lower triangular matrix L and the transpose of the lower triangular matrix L', that is

$$A = LL' \tag{A.15}$$

which can also be expressed in

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \dots & l_{n1} \\ 0 & l_{22} & \dots & l_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0\dots & l_{nn} \end{bmatrix} \tag{A.16}$$

The above matrix multiplication can be separated in,

the first column

$$\begin{array}{lll} a_{11} = l_{11}^2 & \rightarrow & l_{11} = \sqrt{a_{11}} \\ a_{21} = l_{21} l_{11} & \rightarrow & l_{21} = a_{21}/l_{11} \\ \vdots & & \\ a_{n1} = l_{n1} l_{11} & \rightarrow & l_{n1} = a_{n1}/l_{11} \end{array}$$

the second column

$$\begin{array}{lcl} a_{22} = l_{21}^2 + l_{22}^2 & \rightarrow & l_{22} = \sqrt{a_{22} - l_{21}^2} \\ a_{32} = l_{31}l_{21} + l_{32}l_{22} & \rightarrow & l_{32} = (a_{32} - l_{31}l_{21})/l_{22} \\ \quad\vdots & & \\ a_{n2} = l_{n1}l_{21} + l_{n2}l_{22} & \rightarrow & l_{n2} = (a_{n2} - l_{n1}l_{21})/l_{22} \end{array}$$

and so forth, the n'th column is

$$a_{nn} = l_{n1}^2 + l_{n2}^2 + ... + l_{nn}^2 \rightarrow l_{nn} = \sqrt{a_{nn} - l_{n1}^2 - ... - l_{n(n-1)}^2}$$

in general, for $i = 1, ..., n$ and $j = i + 1, ..., n$

$$\begin{aligned} l_{ii} &= \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \\ l_{ji} &= (a_{ji} - \sum_{k=1}^{i-1} l_{jk}l_{ik})/l_{ii} \end{aligned}$$

The algorithm can be specified as follows

Algorithm: Cholesky Factorization

Given $A \in R^{n\times n}$ symmetric positive definite;
for $i = 1, 2, ..., n$
    $L_{ii} \leftarrow \sqrt{A_{ii}}$;
    for $j = i + 1, 2, ..., n$
        $L_{ji} \leftarrow A_{ji}/L_{ii}$;
        for $k = i + 1, 2, ..., j$
            $A_{jk} \leftarrow A_{jk} - L_{ji}/L_{ki}$;
        end(for)
    end(for)
end(for)

APPENDIX B

Extra Graphs

B.1 Combined Effect of N, n and m

Figure B.1 and B.2 show that the simulation results of the combined CPU time versus the predictive horizon with different the number states and the number of inputs.

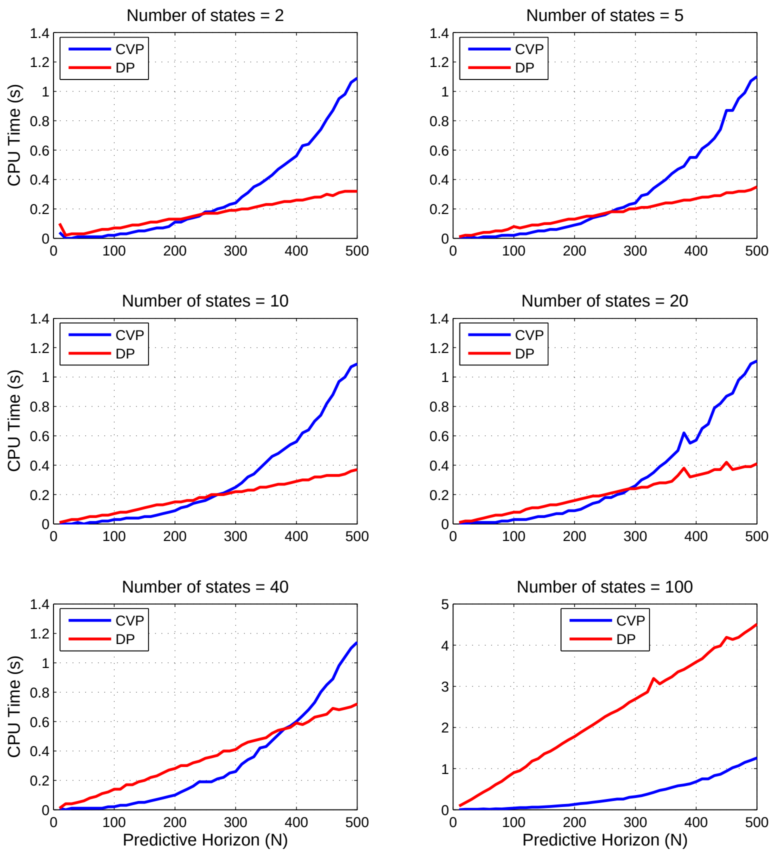

Figure B.1: Combined effect (m=1)

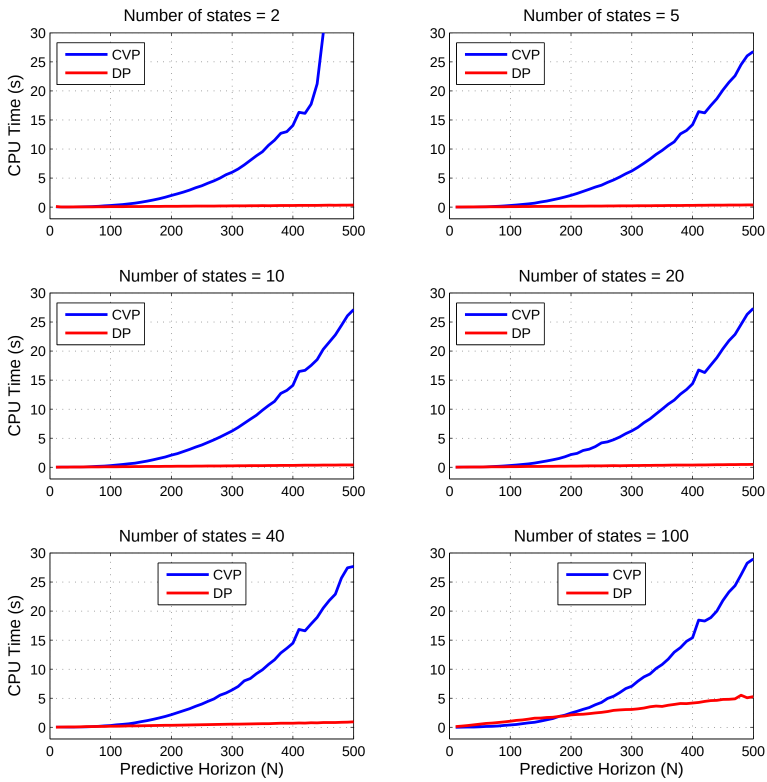

Figure B.2: Combined effect (m=5)

B.2 Algorithms for Solving the Extended LQ Optimal Control Problem

In section 3.2, we state two processes to solve the extended LQ optimal problem. They are Algorithm 1 and the combination of Algorithm 2 and Algorithm 3. Figure B.3 presents the computational time of these two different processes.

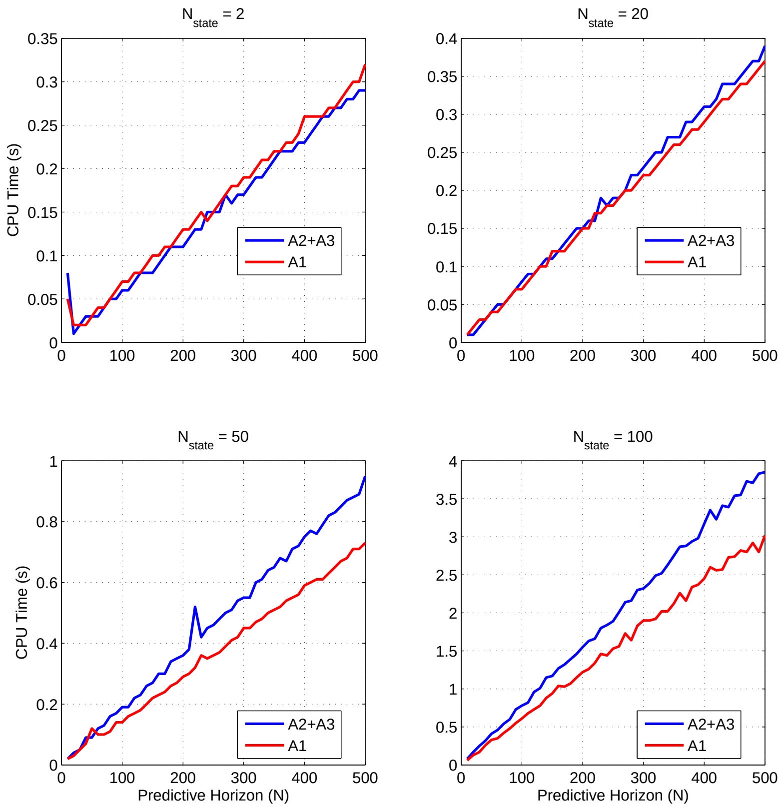

Figure B.3: CPU time of Algorithm 1 and sequential Algorithm 2 and 3

Figure B.4 and B.5 plot the computational time for processes Algorithm 2 $\rightarrow$ 3 $\rightarrow$ 3 and Algorithm 1 $\rightarrow$ 1.

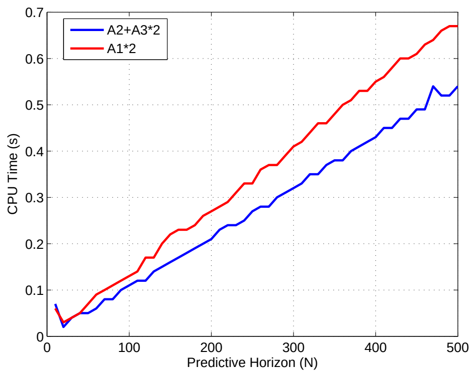

Figure B.4: Two ocmputtation processes (state=2)

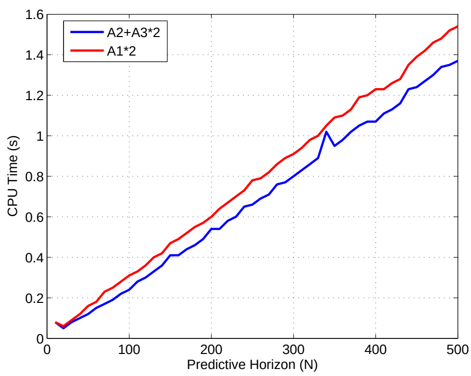

Figure B.5: Two ocmputtation processes (state=50)

B.3 CPU time for Interior Point Method vs n

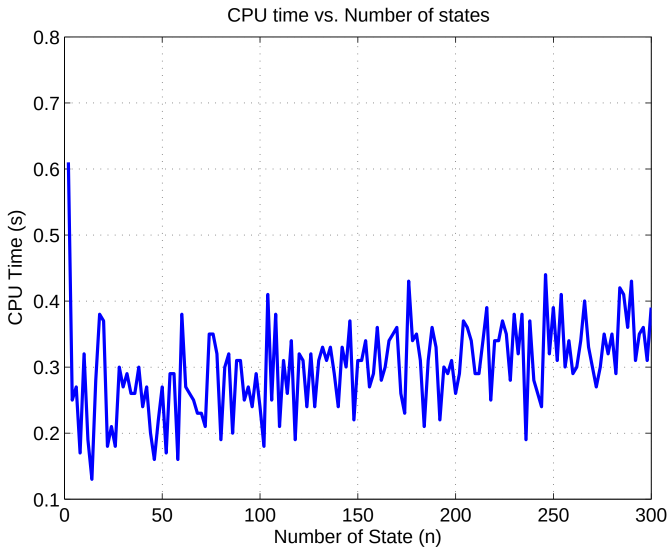

Figure B.6: CPU time vs. n (N=100, m=1)

Appendix C

MATLAB-code

C.1 Implementation Function

C.1.1 simMPC.m

```
function [u,y] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,method)
% SIMMPC solve unconstraints LQ output regulation problem by two methods,
%        vector parameterization based method or dynamic programming based
%        method.
%
%    The unconstraints LQ output regulation problem is
%                      N                        N-1
%    min  phi=0.5*sum||z(k)- r(k)||^2 + 0.5*sum||delt(k)||^2
%                    k=0                        k=0
%       s.t.    x(k+1) = Ax(k) + Bu(k)               0 <= k <= N-1
%               z(k)  = Cx(k)                        0 <= k <= N
%    where the dimension of the states x is n, the dimension of the input u
%    is m and the dimension of the output is p.
%
% [u,y] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,method)
%
% Input parameters:
%       A: n by n matrix
%       B: n by m matrix
%       C: q by n matrix
%       Q: weight matrix, symmetric p by p matrix
%       S: weight matrix, symmetric m by m matrix
```

```
%       N: prediction horison, scalor
%       R: reference trajectory, p by t matrix
%       x0: start state, n by 1 matrix
%       u_1: start input, m by 1 matrix
%       xs: steady-state value of states, n by 1 matrix
%       us: steady-state value of inputs, m by 1 matrix
%       method:
%           1: vector parameterization based method
%           2: dynamic programming based method
%
% Output parameters:
%       u: the optimal input solution
%       y: system output
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Aug.2007


% check the number of input variable
if nargin < 12,method = 1;end  % default method = 1
if nargin < 11
    error(' Input arguments are too less.')
end

% check invalid method
if method~= 1 & method~=2
    error(' Method MUST to be 1 or 2.')
end

% check dimension of matrices
[rA,cA] = size(A);
[rB,cB] = size(B);
[rC,cC] = size(C);
[rQ,cQ] = size(Q);
[rS,cS] = size(S);
[rR,cR] = size(R);
[rx0,cx0] = size(x0);
[ru_1,cu_1] = size(u_1);
[rxs,cxs] = size(xs);
[rus,cus] = size(us);

if rA ~= cA
    error(' Matrix A must be square matrix.')
end
if rQ ~= cQ
    error(' Matrix Q must be square matrix.')
end
if rS ~= cS
    error(' Matrix S must be square matrix.')
end
 if   rA ~= rB  |...
      rA ~= rx0 |...
      rA ~= rxs |...
      cB ~= cS  |...
      cB ~= ru_1  |...
      cB ~= rus  |...
      rC ~= rQ  |...
      rC ~= rR
      error(' Dimensions of matrices are mismatched.')
 end

% solve MPC problem by vector paramerization based method
```

```
if method == 1
    [u,y] = SEsolver(A,B,C,Q,S,N,R,x0,u_1,xs,us);
end

% solve MPC problem by dynamic programming based method
if method == 2
    [u,y] = DPsolver(A,B,C,Q,S,N,R,x0,u_1,xs,us);
end
```

C.1.2 SEsolver.m

```
function [u,y] = SEsolver(A,B,C,Q,S,N,R,x0,u_1,xs,us)
% SESOLVER solves the unconstraints LQ output regulation problem by
%            vector parameterization based method.
%
%    The optimal control problem is
%                      N                            N-1
%    min  phi=0.5*sum||z(k)- r(k)||^2 + 0.5*sum||delt(k)||^2
%                     k=0                           k=0
%      s.t.    x(k+1) = Ax(k) + Bu(k)                  0 <= k <= N-1
%              z(k) = Cx(k)                            0 <= k <= N
%    where the dimension of the states x is n, the dimension of the
    input u
%    is m and the dimension of the output is p.
%
% [u,y] = SEsolver(A,B,C,E,Q,S,N,R,x0,u_1,xs,us)
%
% Input parameters:
%       A: n by n matrix
%       B: n by m matrix
%       C: p by n matrix
%       Q: weight matrix, symmetric p by p matrix
%       S: weight matrix, symmetric m by m matrix
%       N: prediction horison, scalor
%       R: reference trajectory, p by t matrix
%       x0: start state, n by 1 matrix
%       u_1: start input, m by 1 matrix
%       xs: steady-state value of states, n by 1 matrix
%       us: steady-state value of inputs, m by 1 matrix
%
% Output parameters:
%       u: the optimal input solution
%       y: system output
%
% By           : Jing Yang, s032574
% Subject      : Numerical Methods for Model Predictive Control
%                Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor   : John Bagterp Jørgensen
% Date         : Aug.2007

%======================== Initialize
    ========================================

% identify the number of dimensions
n = size(A);        % dimension of state x
m = size(B,2);      % dimension of input u
p = size(C,1);      % dimension of output z,y

% form deviation variables
delt_u_1 = u_1-us;
x = x0-xs;
z = C*x;
```

```

% steady-state level of model responds
zs = C*xs;

% time sequence
t = 0:length(R)-1;

% reconstruct needed variables
R = [R(:,2:end) repmat(R(:,end),1,N)];  % reference
us = repmat(us,N,1);
zs = repmat(zs,N,1);

%======================== MPC design ========================================
[H,Mx0,Mum1,MR] = MPCDesignSE(A,B,C,Q,S,N);

%======================== Simulation ========================================
% prelocated u and y
u = zeros(m,length(t));    % input
y = zeros(p,length(t));    % plant output

for k = 1: length(t)
      %-------------- Measurement ---------------------------
      y(:,k) = z;

      %-------------- On-line computation ----------------
      % get N steps ahead of reference and transform it into
      % vertical vectors
      ref = R(:,k:k+N-1); ref = ref(:);

      % compute optimal input and its deviation
      [up,delt_u_1] = MPCComputeSE(H,Mx0,MR,Mum1,...
                        x,ref,zs,delt_u_1,us);

      % store the first optimal input and its deviation
      u(:,k) = up(1:m);
      delt_u_1 = delt_u_1(1:m);

      %-------------- Prediction ----------------------------
      % predict states and outputs
      [xp,zp] = MPCPredict(x,up,N,A,B,C);

      %-------------- Update --------------------------------
      % update the current state
      x = xp(1:n);

      % form output deviation and store the first one
      z = zp - zs;
      z = z(1:p);
end
```

C.1.3 MPCDesignSE.m

```
function [H,Mx0,Mum1,MR,Lambda] = MPCDesignSE(A,B,Cz,Qz,S,N)
% MPCDESIGNSE designs matrices for solving unconstraints LQ output regulation
%            problem by vector parameterization based method.
%
%   The optimal control problem is
%                    N                      N-1
```

```
%    min   phi=0.5*sum||z(k)- r(k)||^2 + 0.5*sum||delt(k)||^2
%                 k=0                          k=0
%       s.t.    x(k+1) = Ax(k) + Bu(k)                    0 <= k <= N-1
%               z(k) = Cz x(k)                            0 <= k <= N
%    where the dimension of the states x is n, the dimension of the input u
%    is m, and the dimension of the output is p.
%
% [H,Mx0,Mum1,MR] = MPCDesignSE(A,B,Cz,Qz,S,N)
%
% Input parameters:
%       A: n by n matrix
%       B: n by m matrix
%       Cz: p by n matrix
%       Qz: weight matrix, symmetric p by p matrix
%       S: weight matrix, symmetric m by m matrix
%       N: prediction horison, scalor
%
% Output parameters:
%       H = Gamma'*Q*Gamma + Hs
%       Mx0 = Gamma'*Q*phi
%       MR = -Gamma'*Q
%       Mum1 = -[S 0 0 ... 0]'
% in which
%       H(k) = CA^(k-1)B,        k=1,..,N
%
%       Gamma = | H(1)                  |       Q = | Qz               |
%               |  H(2) H(1)            |           |     Qz           |
%               |   ...                 |           |          ...     |
%               |  H(N) H(N-1)  ...H(1)|;           |               Qz|;
%
%       phi = [ CzA CzA^2 ... CzA^N ]';
%
%       Hs = | 2S  -S          |
%            |  -S  2S -S      |
%            |        ...      |
%            |        -S 2S -S|
%            |           -S  S|;
%
% By           : J.B.J?rgensen
% Modify       : Jing Yang, s032574
% Subject      : Numerical Methods for Model Predictive Control
%                Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor : John Bagterp Jørgensen
% Date         : Aug.2007

nx = size(A,2);     % number of states x
nu = size(B,2);     % number of input u
nz = size(Cz,1);    % number of output z

% Form Gamma, Gammad, Phi
Gamma = zeros(N*nz,N*nu);
%Gammad = zeros(N*nz,N*nd);
Phi = zeros(N*nz,nx);

T = Cz;
kz = 0;
for k=1:N
    Gamma(kz+1:kz+nz,1:nu) = T*B;
%     Gammad(kz+1:kz+nz,1:nd) = T*E;
    T = T*A;
    Phi(kz+1:kz+nz,1:nx) = T;
    kz = kz+nz;
end

```

```
for k=2:N
     Gamma((k-1)*nz+1:end,(k-1)*nu+1:k*nu ) = Gamma(1:(N+1-k)*nz,1:nu
          );
end

% Form QZ
QZ = zeros(N*nz,N*nz);
kz = 0;
for k=1:N
     QZ(kz+1:kz+nz,kz+1:kz+nz) = Qz;
     kz = kz+nz;
end

% Form HS
HS = zeros(N*nu,N*nu);

if N == 1
    HS = S;
else
     k=0;
     HS(1:nu,1:nu) = 2*S;
     HS(1+nu:nu+nu,1:nu) = -S;

     for k=1:N-2
          ku = k*nu;
          HS(ku-nu+1:ku,ku+1:ku+nu) = -S;
          HS(ku+1:ku+nu,ku+1:ku+nu) = 2*S;
          HS(ku+nu+1:ku+2*nu,ku+1:ku+nu) = -S;
     end

     k=N-1;
          ku = k*nu;
          HS(ku-nu+1:ku,ku+1:ku+nu) = -S;
          HS(ku+1:ku+nu,ku+1:ku+nu) = S;
end


% Form Mum1
Mum1 = [-S; zeros((N-1)*nu,nu)];

% Form H,Mx0,MR
T = Gamma'*QZ;
H = T*Gamma + HS; H = (H+H')/2;
Mx0 = T*Phi;
MR = -T;

% Form Lambda
Lambda = zeros((N-1)*nu,N*nu);
T = [-eye(nu,nu) eye(nu,nu)];

for k=1:N-1
    Lambda((k-1)*nu+1:k*nu,(k-1)*nu+1:(k+1)*nu) = T;
end
```

C.1.4 MPCComputeSE.m

```
function [U,delt_u] = MPCComputeSE(H,Mx0,MR,Mum1,x,r,zs,u_1,us)
% MPCCOMPUTESE solves the unconstraints QP problem:
%          min phi = 0.5*U'*H*U + g'*U
% where U = [u0 u1 ... uN-1]';
%        H is Hessian matrix, and
```

```
%        g  =  Mx0*x0+MR*(r-zs)+Mum1*u_1
% the optimal solution of the QP problem is:
%        U*  = -H^(-1)*g
%            = -H^(-1)*(Mx0*x0+MR*(r-zs)+Mum1*u_1)
%
% [u,delt_u] = MPCComputeSE(H,Mx0,MR,Mum1,MD,x,r,zs,u_1,us,d)
%
% Input parameters:
%       H: Hessian matrix, positive semidefinite
%       Mx0,MR,Mum1: necessary matrices for solving the QP problem
%       x: state vector
%       r: set-point
%       zs: steady-state value of output
%       u_1: start input
%       us: steady-state value of inputs
%
% Output parameters:
%       U: the optimal solution
%       delt_u: deviation of the optimal solution
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Aug.2007

% factorize Hessian matrix
[R,p1]=chol(H);
if(p1>0)
    error('H not pos.def')
end

Lx0=-(R\(R'\Mx0));
Lr=-(R\(R'\MR));
Lu=-(R\(R'\Mum1));

% optimal input deviation
delt_u = Lx0*x+Lr*(r-zs)+Lu*u_1;

% physical optimal input
U = delt_u+us;
```

C.1.5 MPCPredict.m

```
function [xp,zp] = MPCPredict(x0,u,N,A,B,C)
% MPCPREDICT predicts the future system states xp and output zp:
%
%          xp(k+1) = A*x0 + B*u
%          zp(k+1) = C*xp(k+1)       k = 0,1,...N-1
%
%    where the dimension of the states x0 is n, the dimension of the
     input u
%    is m and the dimension of the output zp is p.
%
% [xp,zp] = MPCPredict(x0,u,N,A,B,C)
%
% Input parameters:
%       A: n by n matrix
%       B: n by m matrix
%       C: p by n matrix
%       x0: start state, n by 1 matrix
```

```
%       u: start input, m by 1 matrix
%       N: prediction horison, scalor
%
% Output parameters:
%       xp: future states, nN by 1 matrix
%       zp: future outputs, pN by 1 matrix
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Aug.2007

x = x0;
n = size(A,2);
m = size(B,2);  % the number of input
p = size(C,1);  % the number of output

xp = zeros(n*N,1);
zp = zeros(p*N,1);

% the sequence of future predicted states and predicted outputs
for k = 0:N-1
    xp(k*n+1:(k+1)*n)= A*x + B*u(k*m+1:(k+1)*m);
    zp(k*p+1:(k+1)*p) = C*xp(k*n+1:(k+1)*n);
end
```

C.1.6 DPsolver.m

```
function [u,y] = DPsolver(A,B,C,Q,S,N,R,x0,u_1,xs,us)
% DPSOLVER solves the unconstraints LQ output regulation problem by
%          dynamic programming based method.
%
%    The unconstraints LQ output regulation problem:
%                   N                          N-1
%    min  phi=0.5*sum||z(k)- r(k)||^2 + 0.5*sum||delt(k)||^2
%                  k=0                         k=0
%      s.t.   x(k+1) = A*x(k) + B*u(k)            0 <= k <= N-1
%             z(k) = C*x(k)                       0 <= k <= N
%    where the dimension of the states x is n, the dimension of the
   input u
%    is m and the dimension of the output z and set-point r is p.
%
% [u,y] = DPsolver(A,B,C,Q,S,N,R,x0,u_1,xs,us)
%
% Input parameters:
%       A: n by n matrix
%       B: n by m matrix
%       C: p by n matrix
%       Q: weight matrix, symmetric p by p matrix
%       S: weight matrix, symmetric m by m matrix
%       N: prediction horison, scalor
%       R: reference trajectory, p by t matrix
%       x0: start state, n by 1 matrix
%       u_1: start input, m by 1 matrix
%       xs: steady-state value of states, n by 1 matrix
%       us: steady-state value of inputs, m by 1 matrix
%
% Output parameters:
%       u: the optimal input solution
%       y: system output
```

```
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Aug.2007

%========================== Initialize ==========================
% identify the number of dimensions
n = size(A);          % dimension of state x
m = size(B,2);        % dimension of input u
p = size(C,1);        % dimension of output z,y

% form deviation variables
delt_u_1 = u_1-us;
x = x0-xs;
z = C*x;

% steady-state level of model responds
zs = C*xs;

% time sequence
t = 0:length(R)-1;

% reconstruct needed variables
R = [R repmat(R(:,end),1,N)];    % reference
us = repmat(us,N,1);
zs = repmat(zs,N,1);

% reconstruct start state
X = [x;delt_u_1];

%========================== MPC design ==========================
[Abar,Bbar,Qbar,Mbar,Rbar,Pbar,P,L,Z] = MPCDesignDPI(A,B,C,Q,S,N);

%========================== Simulation ==========================
% prelocated u and y
u = zeros(length(u_1),length(t));  % input
y = zeros(size(C,1),length(t));    % plant output

for k = 1:length(t)
      %---------- Measurement ------------------------
      y(:,k)= z;

      %---------- On-line computation ----------------
      % get N steps ahead of reference
      ref = R(:,k:k+N);

      % compute optimal input and its deviation
      t0 = cputime;
      [up,delt_u] = MPCComputeDPI(A,C,Q,S,X,us,Abar,Bbar,...
                                 Qbar,Mbar,Rbar,Pbar,...
                                 P,L,Z,ref,N);
      t1 = cputime - t0;
      % store the first optimal input
      u(:,k) = up(1:m);

      %---------- Prediction -------------------------
      % predict states and outputs
      [xp,zp] = MPCPredict(x,up,N,A,B,C);

      %---------- Update -----------------------------
```

```matlab
        % update the current state and reconstruct
        x = xp(1:n)-xs;
        X = [x;delt_u(1:m)];

        % form output deviation and store the first one
        z = zp - zs;
        z = z(1:p);
end
```

C.1.7 MPCDesignDP.m

```matlab
function [Abar,Bbar,Qbar,Mbar,Rbar,Pbar,P,L,Z]=...
                                        MPCDesignDP(A,B,C,Q,S,N)
% MPCDESIGNDP computes unaltered augmented matrices Abar,Bbar,Qbar,Mbar,
%    Rbar,Pbar and compute the factorizations of the unaltered augmented
%    matrice, P,L and Z.
%
%    The unconstraints LQ output regulation problem:
%                   N                         N-1
%    min   phi=0.5*sum||z(k)- r(k)||^2 + 0.5*sum||delt(k)||^2
%                  k=0                        k=0
%       s.t.    x(k+1) = A*x(k) + B*u(k)             0 <= k <= N-1
%               z(k) = C*x(k)                        0 <= k <= N
%    where the dimension of the states x is n, the dimension of the input u
%    is m, and the dimension of the output z and set-point r is p.
%
%    To solve this optimal control problem by dynamic programming based
%    method, the problem has to be formulated as the extended LQ optimal
%    control problem.
%
%    The extended LQ optimal control problem:
%                   N-1
%    min phi = sum l(k) (xbar(k),u(k)) + l(N) (xbar(N))
%                   k=0
%       s.t. xbar(k+1) = Abar*x(k) + Bbar*u(k) + bbar(k)       0 <= k <= N-1
%    with stage costs given by
%    l(k) (xbar(k),u(k)) = 0.5*xbar'(k)*Qbar*xbar(k) + xbar'(k)*Mbar*u(k)
%                          + 0.5*u'(k)*Rbar*u(k) + qbar'(k)*xbar(k)
%                          + rbar(k)*u(k) + fbar(k)             k = 0,1,...,N-1
%    l(N) (xbar(N)) = 0.5*xbar'(N)*Pbar*xbar(N) + pbar'(N)*xbar(N) + gammabar(N)
%    where xbar(k) = [x(k) u(k-1)]'
%       Abar = | A   0 |    Bbar = | B |    Qbar = | C'*Q*C  0 |   Mbar = | 0 |
%              | 0   0 |,          | 0 |,          |    0    S |,         |-S |
%
%       Rbar = S, b(k) = 0, rbar(k) = 0, fbar(k) = 0.5*r(k)'*Q*r(k),
%
%       qbar(k) = |-C'*Q*r(k)|   Pbar(N) =| C'*Q*C  0 |   pbar(N) = | -C'*Q*r(N)|
%                 |     0    |,           |   0     0 |,            |     0     |,
```

```
%
%        gammaba(N)  =0.5*r(N)'*Q*r(N)
%
%     [Abar,Bbar,Qbar,Mbar,Rbar,Pbar,P,L,Z]= MPCDesignDP(A,B,C,Q,S,N)
%
% Input parameters:
%       A: n by n matrix
%       B: n by m matrix
%       C: p by n matrix
%       Q: weight matrix, symmetric p by p matrix
%       S: weight matrix, symmetric m by m matrix
%       N: prediction horison, scalor
%
% Output parameters:
%       Abar,Bbar,Qbar,Mbar,Rbar,Pbar: unaltered augmented matrices
%       P,L,Z: factorized matrices
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Aug.2007

% compute augmented matrices
[Abar,Bbar,Qbar,Mbar,Rbar,Pbar] = DesignDPU(A,B,C,Q,S);
% factorizes the unaltered matrices
[P,L,Z] = factorize(Qbar,Mbar,Rbar,Abar',Bbar',Pbar,N);
```

C.1.8 MPCComputeDP.m

```
function [U,delt_u] = MPCComputeDP(A,C,Q,S,xbar,us,Abar,Bbar,Qbar
    ,...
                                    Mbar,Rbar,Pbar,P,L,Z,R,N)
% MPCCOMPUTEDP computes altered augmented matrices bbar,qbar,rbar,
    fbar,
%     pbar,gammabar and solves the unconstraints LQ output regulation
%     problem.
%
%     The unconstraints LQ output regulation problem:
%                        N                         N-1
%     min   phi=0.5*sum||z(k)- r(k)||^2 + 0.5*sum||delt(k)||^2
%                       k=0                        k=0
%        s.t.   x(k+1) = A*x(k) + B*u(k)              0 <= k <= N-1
%               z(k) = C*x(k)                         0 <= k <= N
%     where the dimension of the states x is n, the dimension of the
%    input u
%     is m, and the dimension of the output z and set-point r is p.
%
%     To solve this optimal control problem by dynamic programming
%    based
%     method, the problem has to be formulated as the extended LQ
%    optimal
%     control problem.
%
%     The extended LQ optimal control problem:
%                  N-1
%     min phi = sum  l(k) (xbar(k),u(k)) + l(N) (xbar(N))
%                  k=0
%        s.t. xbar(k+1) = Abar*x(k) + Bbar*u(k) + bbar(k)       0 <= k
%    <= N-1
%     with stage costs given by
```

```
%    l(k) (xbar(k),u(k)) = 0.5*xbar'(k)*Qbar*xbar(k) + xbar'(k)*Mbar
    *u(k)
%                       + 0.5*u'(k)*Rbar*u(k) + qbar'(k)*xbar(k)
%                       + rbar(k)*u(k) + fbar(k)          k =
    0,1,...,N-1
%    l(N) (xbar(N)) = 0.5*xbar'(N)*Pbar*xbar(N) + pbar'(N)*xbar(N) +
     gammabar(N)
%    where xbar(k) = [x(k) u(k-1)]'
%       Abar = | A  0 |   Bbar = | B |   Qbar = | C'*Q*C 0 |   Mbar =
     | 0 |
%              | 0  0 |,         | 0 |,         |    0   S |,
           |-S |
%
%       Rbar = S, b(k) = 0, rbar(k) = 0, f(k) = 0.5*r(k)'*Q*r(k),
%
%       qbar(k) = |-C'*Q*r(k)|   Pbar(N) =| C'*Q*C 0 |   pbar(N) = | -
    C'*Q*r(N)|
%                 |    0     |,           |    0   0 |,             |
       0     |,
%
%       gamma(N)  =0.5*r(N)'*Q*r(N)
%
% [U,delt_u] = MPCComputeDP(A,C,Q,S,xbar,us,Abar,Bbar,Qbar,...
%                           Mbar,Rbar,Pbar,P,L,Z,R,N)
%
% Input parameters:
%       A: n by n matrix
%       C: p by n matrix
%       Q: weight matrix, symmetric p by p matrix
%       S: weight matrix, symmetric m by m matrix
%       xbar: augmented state vector
%       us: steady-state value of inputs, m by 1 matrix
%       Abar,Bbar,Qbar,Mbar,Rbar,Pbar: augmented matrices
%       R: reference trajectory, p by t matrix
%       P,L,Z: factorized matrices
%
% Output parameters:
%       U: the optimal input solution
%       delt_u: deviation of the optimal input solution
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Aug.2007

% compute altered augmented matrices
[bbar,qbar,rbar,fbar,pbar,gamma] = DesignDPA(A,C,Q,S,R,N);

% compute optimal input U and state
[Xnew,delt_u] = solveELQ (Qbar,Mbar,Rbar,qbar,rbar,fbar,Abar',...
                          Bbar',bbar,xbar,Pbar,pbar,gamma,N,P,L,Z);

% form physical variable
delt_u = delt_u(:);
U = delt_u + us;
```

C.1.9 DPFactSolve.m

```
function [x,u,phi] = DPFactSolve (Q,M,R,q,r,f,A,B,b,x0,PN,pN,gammaN,
    N)
```

```
% DPFACTSOLVE solves the extended LQ optimal control problem. It
%              implements Algorithm 1.
%
%    The extended linear-quadratic optimal control problem is
%                                  N-1
%           min          phi = sum l(k)(x(k),u(k)) + l(N)(x(N))
%    {x(k+1),u(k)}                 k=0
%      s.t.      x(k+1) = A(k)x(k) + B(k)u(k) +b(k)     0 <= k <= N-1
%    with the stage cost
%    l(k) = 0.5*x(k)'Q(k)x(k) + x(k)'M(k)u(k) + 0.5*u(k)'R(k)u(k) +
%                              q(k)'x(k) + r(k)'u(k) + f(k)    0 <= k <= N-1
%    l(N) = 0.5*x(N)'P(N)x(N) + pN'x(N) +gamma
%
%    where the dimension of the states x is n, the dimension of the input u
%    is m.
%
% [x,u,phi] = DPFactSolve (Q,M,R,q,r,f,A,B,b,x0,PN,pN,gammaN,N)
%
% Input parameters:
%       Q: weight matrix, symmetric n by n matrix
%       M: n by m matrix
%       R: weight matrix, symmetric m by m matrix
%       q: n by N matrix
%       r: m by N matrix
%       f: N by 1 matrix
%       A: n by n matrix
%       B: m by n matrix
%       b: n by N matrix
%       x0: start state. n by 1 matrix
%       PN: n by n matrix
%       pN: n by 1 matrix
%       gammaN: scalor
%       N: prediction horison, scalor
%
% Output parameters:
%         u: optimal inputs
%         x: future states
%         phi: the optimal objective value
%
% By           : Jing Yang, s032574
% Subject      : Numerical Methods for Model Predictive Control
%                Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor   : John Bagterp Jørgensen
% Date         : Aug.2007

n = size(Q,1); % number of states x
m = size(R,1); % number of input u

% prelocate
P = 0.5*(PN+PN');
p = pN;            % assign pN to p
gamma = gammaN;    % assign gammaN to gamma

for k=N:-1:1
    Re = R+B*P*B';
    S = A*P;
    Y = (M+S*B')';

    s = P*b(:,k);
    c = s+p;
    d = r(:,k)+B*c;

    L((k-1)*m+1:k*m,:) = chol(Re,'lower');
```

```

    Z((k-1)*m+1:k*m,:) = L((k-1)*m+1:k*m,:)\Y;
    z((k-1)*m+1:k*m,:) = L((k-1)*m+1:k*m,:)\d;

    Ptmp = Q+S*A'-Z((k-1)*m+1:k*m,:)'*Z((k-1)*m+1:k*m,:);
    P = 0.5*(Ptmp+Ptmp');
    gamma = gamma+f(k)+p'*b(:,k)+0.5*s'*b(:,k)-0.5*z((k-1)*m+1:k*m,:)'*z((k-1)*m+1:k*m,:);
    p = q(:,k)+A*c-Z((k-1)*m+1:k*m,:)'*z((k-1)*m+1:k*m,:);
end

% compute the optimal value
phi = 0.5*x0'*P*x0+p'*x0+gamma;

% prelocate
x = zeros(n,N+1); x(:,1) = x0;
u = zeros(m,N);

% compute optimal u and x
for k = 1:N
    y = Z((k-1)*m+1:k*m,:)*x(:,k)+z((k-1)*m+1:k*m,:);

    u(:,k) = -L((k-1)*m+1:k*m,:)'\y;
    x(:,k+1) = A'*x(:,k)+B'*u(:,k)+b(:,k);
end

% truncate the first state
x = x(:,2:end);
```

C.1.10 factorize.m

```
function [P,L,Z] = factorize(Q,M,R,A,B,PN,N)
% FACTORIZE factorizes the matrices of the extended LQ optimal control
%           problem. It implements algorithm 2.
%    The extended linear-quadratic optimal control problem is
%                              N-1
%          min          phi = sum  l(k)(x(k),u(k)) + l(N)(x(N))
%    {x(k+1),u(k)}             k=0
%      s.t.     x(k+1) = A(k)x(k) + B(k)u(k) +b(k)      0 <= k <= N-1
%    with the stage cost
%    l(k) = 0.5*x(k)'Q(k)x(k) + x(k)'M(k)u(k) + 0.5*u(k)'R(k)u(k) +
%                              q(k)'x(k) + r(k)'u(k) + f(k)    0 <= k <= N-1
%    l(N) = 0.5*x(N)'P(N)x(N) + pN'x(N) +gamma
%
%    where the dimension of the states x is n, the dimension of the input u
%    is m.
%
% [P,L,Z] = factorize(Q,M,R,A,B,PN,N)
%
% Input parameters:
%       Q: weight matrix, symmetric n by n matrix
%       M: n by m matrix
%       R: weight matrix, symmetric m by m matrix
%       A: n by n matrix
%       B: m by n matrix
%       PN: n by n matrix
%       N: prediction horison, scalor
%
```

```
% Output parameters:
%       P,L,Z: factorized matrices
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Aug.2007

n = size(Q,1); % number of states x
m = size(R,1); % number of input u

% prelocate
P = zeros((N+1)*n,n);
L = zeros(N*m,m);
Z = zeros(N*m,n);

% set the last block row of P to be PN
P(N*n+1:end,:) = 0.5*(PN+PN');

% iteration
for k=N:-1:1
     Re = R+B*P(k*n+1:(k+1)*n,:)*B';
     S = A*P(k*n+1:(k+1)*n,:);
     Y = (M+S*B')';

     Ltemp = chol(Re);
     L((k-1)*m+1:k*m,:) = Ltemp';
     Z((k-1)*m+1:k*m,:) = L((k-1)*m+1:k*m,:)\Y;

     Ptmp = Q+S*A'-Z((k-1)*m+1:k*m,:)'*Z((k-1)*m+1:k*m,:);
     P((k-1)*n+1:k*n,:) = 0.5*(Ptmp+Ptmp');
end

% truncate the last block rows
P = P(1:N*n,:);
```

C.1.11 solveELQ.m

```
function [x,u,phi] = solveELQ(Q,M,R,q,r,f,A,B,b,x0,PN,pN,gammaN,N,P,L,Z)
% SOLVEELQ solve the factorized extended LQ optimal control problem. It
%           implements algorithm 3.
%
%    The extended linear-quadratic optimal control problem is
%                             N-1
%         min         phi = sum l(k)(x(k),u(k)) + l(N)(x(N))
%    {x(k+1),u(k)}            k=0
%       s.t.      x(k+1) = A(k)x(k) + B(k)u(k) +b(k)      0 <= k <= N-1
%    with the stage cost
%    l(k) = 0.5*x(k)'Q(k)x(k) + x(k)'M(k)u(k) + 0.5*u(k)'R(k)u(k) +
%                             q(k)'x(k) + r(k)'u(k) + f(k)    0 <= k <= N-1
%    l(N) = 0.5*x(N)'P(N)x(N) + pN'x(N) +gamma
%
%    where the dimension of the states x is n, the dimension of the input u
%    is m.
%
% [x,u,phi] = solveELQ (Q,M,R,q,r,f,A,B,b,x0,PN,pN,gammaN,N,P,L,Z)
```

```
%
% Input parameters:
%        Q: weight matrix, symmetric n by n matrix
%        M: n by m matrix
%        R: weight matrix, symmetric m by m matrix
%        A: n by n matrix
%        B: m by n matrix
%        PN: n by n matrix
%        q: n by N matrix
%        r: m by N matrix
%        f: N by 1 matrix
%        b: n by N matrix
%        pN: n by 1 matrix
%        gammaN: scalor
%        x0: start state. n by 1 matrix
%        P: factorized n*N by n matrix
%        L: factorized m*N by m matrix
%        Z: factorized m*N by n matrix
%
% Output parameters:
%        x: the future states
%        u: the optimal inputs
%        phi: the optimal value
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor : John Bagterp Jørgensen
% Date        : Aug.2007

n = size(Q,1);  % number of states x
m = size(R,1);  % number of input u

p = pN;              % assign pN to p
gamma = gammaN;      % assign gammaN to gamma
P = [P;PN];          % append PN to P

% iteration
for k = N:-1:1
    % compute the temporary vectors
    s = P(k*n+1:(k+1)*n,:)*b(:,k);
    c = s+p;
    d = r(:,k)+B*c;

    % compute z by solving lower triangular system equation
    z((k-1)*m+1:k*m,:) = L((k-1)*m+1:k*m,:)\d;

    % update gamma and p
    gamma = gamma+f(k)+p'*b(:,k)+0.5*s'*b(:,k)-0.5*z((k-1)*m+1:k*m,:)'*z((k-1)*m+1:k*m,:);
    p = q(:,k)+A*c-Z((k-1)*m+1:k*m,:)'*z((k-1)*m+1:k*m,:);
end

% compute the optimal value
phi = 0.5*x0'*P(1:n,:)*x0+p'*x0+gamma;

% prelocate
x = zeros(n,N+1); x(:,1) = x0;
u = zeros(m,N);

% compute optimal u and x
for k = 1:N
    y = Z((k-1)*m+1:k*m,:)*x(:,k)+z((k-1)*m+1:k*m,:);

    u(:,k) = -L((k-1)*m+1:k*m,:)'\y;
```

```
    x(:,k+1) = A'*x(:,k)+B'*u(:,k)+b(:,k);
end

% truncate the first state
x = x(:,2:end);
```

C.1.12 DesignDPU.m

```
function [Abar,Bbar,Qbar,Mbar,Rbar,Pbar] = ...
                                    DesignDPU(A,B,C,Q,S)
% DESIGNDPU computes unaltered augmented matrices Abar,Bbar,Qbar, Mbar,Rbar,
%            Pbar for solving the unconstrained LQ optimal control problem
%            by dynamic programming based method.
%
%    The unconstraints LQ output regulation problem:
%                   N                         N-1
%    min   phi=0.5*sum||z(k)- r(k)||^2 + 0.5*sum||delt(k)||^2
%                  k=0                        k=0
%       s.t.    x(k+1) = A*x(k) + B*u(k)             0 <= k <= N-1
%               z(k) = C*x(k)                        0 <= k <= N
%    where the dimension of the states x is n, the dimension of the input u
%    is m, and the dimension of the output z and set-point r is p.
%
%    To solve this optimal control problem by dynamic programming based
%    method, the problem has to be formulated as the extended LQ optimal
%    control problem.
%
%    The extended LQ optimal control problem:
%                N-1
%    min phi = sum l(k) (xbar(k),u(k)) + l(N) (xbar(N))
%                k=0
%       s.t. xbar(k+1) = Abar*x(k) + Bbar*u(k) + bbar(k)           0 <= k <= N-1
%    with stage costs given by
%    l(k) (xbar(k),u(k)) = 0.5*xbar'(k)*Qbar*xbar(k) + xbar'(k)*Mbar*u(k)
%                            + 0.5*u'(k)*Rbar*u(k) + qbar'(k)*xbar(k)
%                            + rbar(k)*u(k) + fbar(k)              k = 0,1,...,N-1
%    l(N) (xbar(N)) = 0.5*xbar'(N)*Pbar*xbar(N) + pbar'(N)*xbar(N) + gammabar(N)
%    where xbar(k) = [x(k) u(k-1)]'
%       Abar = | A   0 |    Bbar = | B |    Qbar = | C'*Q*C 0 |   Mbar = | 0 |
%              | 0   0 |,          | 0 |,          |    0   S |,         |-S |
%
%       Rbar = S, b(k) = 0, rbar(k) = 0, f(k) = 0.5*r(k)'*Q*r(k),
%
%       Pbar(N) =| C'*Q*C 0 |   pbar(N) = | -C'*Q*r(N)|
%                |    0   0 |,            |     0     |,
%
%       gamma(N) =0.5*r(N)'*Q*r(N)
%
% [Abar,Bbar,Qbar,Mbar,Rbar,Pbar] = DesignDPU(A,B,C,Q,S)
%
```

```
% Input parameters:
%       A: n by n matrix
%       B: n by m matrix
%       C: p by n matrix
%       Q: weight matrix, symmetric p by p matrix
%       S: weight matrix, symmetric m by m matrix
%
% Output parameters:
%       Abar,Bbar,Qbar,Mbar,Rbar,Pbar: unaltered augmented matrices
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Aug.2007

nx = size(A,2);   % number of states x
nu = size(B,2);   % number of input u
nz = size(C,1);   % number of output z

% Form Abar, Bbar, Ebar
Abar = [ A              zeros(nx,nu);
         zeros(nu,nx)   zeros(nu)    ];

Bbar = [ B;
         eye(nu)];

% Form Qbar, Mbar, Rbar, Sbar, Pbar
Qbar = [ C'*Q*C        zeros(nx,nu);
         zeros(nu,nx)  S            ];

Mbar = [ zeros(nx,nu);
         -S            ];

Rbar = S;

Pbar = [ C'*Q*C        zeros(nx,nu);
         zeros(nu,nx)  zeros(nu)    ];
```

C.1.13 DesignDPA.m

```
function [bbar,qbar,rbar,fbar,pbar,gammabar] = DesignDPA(A,C,Q,S,R,N
    )

% DESIGNDPA computes altered augmented matrices b,qbar,rbar,f,pbar,
    gamma
%           for solving the unconstraints LQ optimal control problem
    by
%           dynamic programming based method.
%
%     The unconstraints LQ output regulation problem:
%                    N                         N-1
%     min   phi=0.5*sum||z(k)- r(k)||^2 + 0.5*sum||delt(k)||^2
%                   k=0                        k=0
%       s.t.    x(k+1) = A*x(k) + B*u(k)              0 <= k <= N-1
%               z(k) = C*x(k)                         0 <= k <= N
%     where the dimension of the states x is n, the dimension of the
    input u
%     is m, and the dimension of the output z and set-point r is p.
%
```

```
%    To solve this optimal control problem by dynamic programming based
%    method, the problem has to be formulated as the extended LQ optimal
%    control problem.
%
%    The extended LQ optimal control problem:
%                N-1
%    min phi = sum l(k) (xbar(k),u(k)) + l(N) (xbar(N))
%                k=0
%       s.t. xbar(k+1) = Abar*x(k) + Bbar*u(k) + bbar(k)          0 <= k <= N-1
%    with stage costs given by
%    l(k) (xbar(k),u(k)) = 0.5*xbar'(k)*Qbar*xbar(k) + xbar'(k)*Mbar*u(k)
%                            + 0.5*u'(k)*Rbar*u(k) + qbar'(k)*xbar(k)
%                            + rbar(k)*u(k) + fbar(k)             k = 0,1,...,N-1
%    l(N) (xbar(N)) = 0.5*xbar'(N)*Pbar*xbar(N) + pbar'(N)*xbar(N) + gammabar(N)
%    where xbar(k) = [x(k) u(k-1)]'
%        Abar = | A  0 |    Bbar = | B |    Qbar = | C'*Q*C 0 |   Mbar = | 0 |
%               | 0  0 |,          | 0 |,          |    0   S |,         |-S |
%
%        Rbar = S, b(k) = 0, rbar(k) = 0, f(k) = 0.5*r(k)'*Q*r(k),
%
%        Pbar(N) =| C'*Q*C 0 |   pbar(N) = | -C'*Q*r(N)|
%                 |    0   0 |,            |     0     |,
%
%        gammabar(N) =0.5*r(N)'*Q*r(N)
%
% [bbar,qbar,rbar,fbar,pbar,gammabar] = DesignDPA(A,C,Q,S,ref,N)
%
% Input parameters:
%       A: n by n matrix
%       C: p by n matrix
%       Q: weight matrix, symmetric p by p matrix
%       S: weight matrix, symmetric m by m matrix
%       R: set-point, p by t matrix
%       N: prediction horison, scalor
%
% Output parameters:
%       b,qbar,rbar,f,pbar,gamma: altered augmented matrices
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Aug.2007

nx = size(A,2);    % number of states x
nu = size(S,2);    % number of input u
nr = size(R,2);    % number of reference r

refN = R(:,end);
ref = R(:,1:end-1);

% form b
bbar = zeros((nx+nu),N);

% form qbar, f, rbar,
qbar = [ -C'*Q*ref;
           zeros(nu,nr-1)];
```

```
fbar = 0.5*diag(ref'*Q*ref);

rbar = zeros(nu,N);

% Form pbar, gamma
pbar = [ -C'*Q*refN;
         zeros(nu,1) ];

gammabar = 0.5*diag(refN'*Q*refN);
```

C.1.14 InteriorPoint.m

```
function [x,lam,info,QPinfo] = InteriorPoint(H,g,A,b)
% INTERIORPOINT Solve convex inequality constrained QP problems by
%     interior-point algorithm
%
%     The convex inequality constrained QP problem
%                  min   0.5* x'Hx + g'x
%                   x
%                  s.t.  A'x >= b
%     where H is the Hessian matrix, g is the gradient. A is a  n by m matrix,
%     n is the dimension of x, m is the number of the constraints.
%
% Syntax: [x,lam,info,QPinfo] = InteriorPoint(H,g,A,b,x0,s0,lam0)
%
% Output parameters:
%       x: the optimal solution
%       lam: the optimal Lagrangian multiplier
%       info:  1  converged
%              0  not converged
%       QPinfo: a structure array, including sequence of x,lam,s, iter and
%              alpha
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Nov.2007

[n,m] = size(A);

% Initialize start point, Lagrangian multiplier and the slack variable
x = zeros(n,1);
lam = ones(m,1);
s = ones(m,1);

% Residuals and Duality Gap
rL = H*x + g - A*lam;
rs = s - A'*x + b ;
rslam = s.*lam;
mu = sum(rslam)/m;

%e = ones(m,1);
stop = 1.0e-16;
i = 0;
count = 100;
Converged =    ( max(abs(rL)) < stop ) && ...
```

```
                ( max(abs(rs)) < stop ) && ...
                ( abs(mu) < stop );
xArray = x';
lamArray = lam';
sArray = s';

while ~Converged && i<count
    i = i +1;

    Hbar = H + A*diag(lam./s)*A';
    Rbar = chol(Hbar);  % Rbar is upper triangular
    Lbar = Rbar';

    % Affine Predictor Step
    rbar = A*((rslam - lam.*rs)./s);
    gbar = rL + rbar;
    delt_x_aff = -Lbar'\(Lbar\gbar);
    delt_s_aff = -rs + A'*delt_x_aff;
    delt_lam_aff = -(rslam + lam.*delt_s_aff)./s;

    % Determin the maximum affine step length
    alpha_aff = 1;
    idx = find(delt_lam_aff < 0);
    if ~isempty(idx)
       alpha_aff = min( alpha_aff, min( -lam(idx)./delt_lam_aff(idx) ));
    end

    idx = find(delt_s_aff < 0);
    if ~isempty(idx)
       alpha_aff = min( alpha_aff, min( -s(idx)./delt_s_aff(idx) ));
    end

    % Compute affine duality gap
    mua = (lam + alpha_aff*delt_lam_aff)' * (s +  alpha_aff*delt_s_aff)/m;
    sigma = (mua / mu)^3; % Centering parameter

    % Center Corrector Step
    % Modify complementary
    rslam = rslam + delt_s_aff.*delt_lam_aff - sigma*mu;%*e;

    % Center Corrector Step
    rbar = A*((rslam - lam.*rs)./s);
    gbar = rL + rbar;
    delt_x = -Lbar'\(Lbar\gbar);
    delt_s = -rs + A'*delt_x;
    delt_lam = -(rslam + lam.*delt_s)./s;

    % Determin the maximum affine step length
    alphamax = 1;
    idx = find(delt_lam < 0);
    if ~isempty(idx)
       alphamax = min( alphamax, min( -lam(idx)./delt_lam(idx) ));
    end

    idx = find(delt_s < 0);
    if ~isempty(idx)
       alphamax = min( alphamax, min( -s(idx)./delt_s(idx) ));
    end
    alpha = 0.995*alphamax;

    % update x, lam and s, Lamda, S
    x = x + alpha*delt_x;
    lam = lam + alpha*delt_lam;
```

```
    s = s + alpha*delt_s;

    % Residuals and Duality Gap
    rL = H*x + g - A*lam;
    rs = s - A'*x + b ;
    rslam = s.*lam ;
    mu = sum(rslam)/m;

    Converged = ( max(abs(rL)) < stop ) && ...
                ( max(abs(rs)) < stop ) && ...
                ( abs(mu) < stop );
    if nargout == 4
        xArray = [xArray; x'];
        lamArray = [lamArray; lam'];
        sArray = [sArray; s'];

        alphaArray(i) = alpha;
    end
end

% Report solution and information
if Converged
    info = 1;
else
    info = 0;
end

if nargout == 4
    QPinfo = struct(...
                'x',xArray, ...
                'lam',lamArray, ...
                's',sArray, ...
                'iter',i, ...
                'alpha',alphaArray);
end
```

C.1.15 MPCInteriorPoint.m

```
function [x,info,QPinfo] = MPCInteriorPoint(H,g,A,xl,xu,bl,bu)
% MPCINTERIORPOINT Solve inequality constrained MPC problems by
%    interior-point algorithm
%
%    The inequality constrained MPC problem
%              min  0.5*x'Hx + g'x
%               x
%              s.t. xl <= x <= xu
%                   bl <= A'x <= bu
%    where H is the Hessian matrix, g is the gradient. A is a n by
     m matrix
%    n is the dimension of x, m is the number of the input rate
     constraints.
%    xl is the lower limit of inputs. xu is the upper limit of
     inputs. bl is
%    the lower limit of input rates. bu is the upper limit of input
     rates.
%
% Syntax: [x,info,QPinfo] = MPCInteriorPoint(H,g,A,xl,xu,bl,bu)
%
% Output parameters:
%      x: the optimal solution
%      info: 1 converged
```

```matlab
%              0   not converged
%          QPinfo: a structure array, including sequence of x, the sequence of
%               the Lagrangian multiplier l,u,s and t, the sequence of slack
%               variables lam,mu,delt and k, the sequence of alpha and the
%               number of iteration.
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Nov.2007

[n,m] = size(A); % n is number of inputs, m is the number of constraints

%-------------------- Initialize start point, --------------------
%---------- Lagrangian multiplier and the slack variable -------
x = zeros(n,1);
l = ones(n,1);
u = ones(n,1);
s = ones(m,1);
t = ones(m,1);
lam = ones(n,1);
mu = ones(n,1);
delt = ones(m,1);
k = ones(m,1);

%---------------------- Compute Residuals ------------------------------
%rL = H*x + g - lam + mu - A*(delt - k);
rL = H*x + g - lam + mu - SimpleStruct(A,(delt - k),2);

rl = l - x + xl;
ru = u + x - xu;
%rs = s - A'*x +bl;
rs = s - SimpleStruct(A,x,1) +bl;

%rt = t + A'*x -bu;
rt = t + SimpleStruct(A,x,1) -bu;

r_l_lam = l.*lam;
r_u_mu = u.*mu;
r_s_delt = s.*delt;
r_t_k = t.*k;

%--------------------- Compute Duality Gap ----------------------------
Gap = (l'*lam + u'*mu + s'*delt + t'*k)/2/(m+n);

%--------------------- Iteration ------------------------------------------------
stop = 1e-10;
ite = 0;
iteMax = 100;

Converged = (max(abs(rL)) < stop ) && ...
            (max(abs(rl)) < stop ) && ...
            (max(abs(ru)) < stop ) && ...
            (max(abs(rs)) < stop ) && ...
            ( abs(Gap) < stop );

xArray = x';
```

```
lArray = l';
uArray = u';
sArray = s';
tArray = t';
lamArray = lam';
muArray = mu';
deltArray = delt';
kArray = k';

while ~Converged && ite < iteMax
    ite = ite + 1;

    %Hbar = H + diag(lam./l) + diag(mu./u) + A*diag((delt./s + k./t))*A';
    Hbar = H + diag(lam./l) + diag(mu./u) + SimpleStruct(A,(delt./s + k./t),3);
    Lbar = chol(Hbar,'lower');  % Rbar is upper triangular
    %Lbar = Rbar';

    %------------- Affine Predictor Step ------------------------------
%     rbar = -(-r_l_lam + lam.*rl)./l...
%            +(-r_u_mu + mu.*ru)./u...
%            -A*((-r_s_delt + delt.*rs)./s - (-r_t_k + k.*rt)./t);
    rbar = -(-r_l_lam + lam.*rl)./l...
           +(-r_u_mu + mu.*ru)./u...
           -SimpleStruct(A,((-r_s_delt + delt.*rs)./s - (-r_t_k + k.*rt)./t),2);

    gbar = rL + rbar;

    delt_x_aff = -Lbar'\(Lbar\gbar);

    delt_l_aff = -rl + delt_x_aff;
    delt_u_aff = -ru - delt_x_aff;
    %delt_s_aff = -rs + A'*delt_x_aff;
    delt_s_aff = -rs + SimpleStruct(A,delt_x_aff,1);

    %delt_t_aff = -rt - A'*delt_x_aff;
    delt_t_aff = -rt - SimpleStruct(A,delt_x_aff,1);

    delt_lam_aff = -(r_l_lam + lam.*delt_l_aff)./l;
    delt_mu_aff = -(r_u_mu + mu.*delt_u_aff)./u;
    delt_delt_aff = -(r_s_delt + delt.*delt_s_aff)./s;
    delt_k_aff = -(r_t_k + k.*delt_t_aff)./t;

    %---------------- Determin the maximum affine step length ------------
    alpha_aff = 1;
    idx = find(delt_l_aff < 0);
    if( ~isempty(idx) )
        alpha_aff = min(alpha_aff,min(-l(idx,1)./delt_l_aff(idx,1)));
    end
    idx = find(delt_u_aff < 0);
    if( ~isempty(idx) )
        alpha_aff = min(alpha_aff,min(-u(idx,1)./delt_u_aff(idx,1)));
    end
    idx = find(delt_s_aff < 0);
    if( ~isempty(idx) )
        alpha_aff = min(alpha_aff,min(-s(idx,1)./delt_s_aff(idx,1)));
    end
    idx = find(delt_t_aff < 0);
```

```
    if( ~isempty(idx) )
        alpha_aff = min(alpha_aff,min(-t(idx,1)./delt_t_aff(idx,1)));
    end
    idx = find(delt_lam_aff < 0);
    if( ~isempty(idx) )
        alpha_aff = min(alpha_aff,min(-lam(idx,1)./delt_lam_aff(idx,1)));
    end
    idx = find(delt_mu_aff < 0);
    if( ~isempty(idx) )
        alpha_aff = min(alpha_aff,min(-mu(idx,1)./delt_mu_aff(idx,1)));
    end
    idx = find(delt_delt_aff < 0);
    if( ~isempty(idx) )
        alpha_aff = min(alpha_aff,min(-delt(idx,1)./delt_delt_aff(idx,1)));
    end
    idx = find(delt_k_aff < 0);
    if( ~isempty(idx) )
        alpha_aff = min(alpha_aff,min(-k(idx,1)./delt_k_aff(idx,1)));
    end

    %------------- Compute affine duality gap --------------------------------

    Gap_aff = (  (l+alpha_aff*delt_l_aff)'*(lam+alpha_aff*delt_lam_aff)...
                +(u+alpha_aff*delt_u_aff)'*(mu +alpha_aff*delt_mu_aff)...
                +(s+alpha_aff*delt_s_aff)'*(delt+alpha_aff*delt_delt_aff)...
                +(t+alpha_aff*delt_t_aff)'*(k+alpha_aff*delt_k_aff)...
              )/2 /(m+n);
    sigma = (Gap_aff/Gap)^3; % Centering parameter

    %------------- Center Corrector Step ------------------------------------
    % Modify complementary

    r_l_lam = l.*lam + delt_l_aff.*delt_lam_aff - sigma*Gap;%*e;
    r_u_mu = u.*mu + delt_u_aff.*delt_mu_aff - sigma*Gap;%*e;
    r_s_delt = s.*delt + delt_s_aff.*delt_delt_aff - sigma*Gap;%*e1;
    r_t_k = t.*k + delt_t_aff.*delt_k_aff - sigma*Gap;%*e1;

%     rbar = -(-r_l_lam + lam.*rl)./l...
%             +(-r_u_mu + mu.*ru)./u...
%             -A*((-r_s_delt + delt.*rs)./s - (-r_t_k + k.*rt)./t);
    rbar = -(-r_l_lam + lam.*rl)./l...
           +(-r_u_mu + mu.*ru)./u...
           -SimpleStruct(A,((-r_s_delt + delt.*rs)./s - (-r_t_k + k.*rt)./t),2);

    gbar = rL + rbar;

    delt_x = -Lbar'\(Lbar\gbar);

    delt_l = -rl + delt_x;
    delt_u = -ru - delt_x;
    %delt_s = -rs + A'*delt_x;
    delt_s = -rs + SimpleStruct(A,delt_x,1);

```

```
    %delt_t = -rt - A'*delt_x;
    delt_t = -rt - SimpleStruct(A,delt_x,1);

    delt_lam = -(r_l_lam + lam.*delt_l)./l;
    delt_mu = -(r_u_mu + mu.*delt_u)./u;
    delt_delt = -(r_s_delt + delt.*delt_s)./s;
    delt_k = -(r_t_k + k.*delt_t)./t;

    %------------------- Determin the maximum step length
        -------------------------
    alphaMax = 1;
    idx = find(delt_l < 0);
    if( ~isempty(idx) )
        alphaMax = min(alphaMax,min(-l(idx,1)./delt_l(idx,1)));
    end
    idx = find(delt_u < 0);
    if( ~isempty(idx) )
        alphaMax = min(alphaMax,min(-u(idx,1)./delt_u(idx,1)));
    end
    idx = find(delt_s < 0);
    if( ~isempty(idx) )
        alphaMax = min(alphaMax,min(-s(idx,1)./delt_s(idx,1)));
    end
    idx = find(delt_t < 0);
    if( ~isempty(idx) )
        alphaMax = min(alphaMax,min(-t(idx,1)./delt_t(idx,1)));
    end
    idx = find(delt_lam < 0);
    if( ~isempty(idx) )
        alphaMax = min(alphaMax,min(-lam(idx,1)./delt_lam(idx,1)));
    end
    idx = find(delt_mu < 0);
    if( ~isempty(idx) )
        alphaMax = min(alphaMax,min(-mu(idx,1)./delt_mu(idx,1)));
    end
    idx = find(delt_delt < 0);
    if( ~isempty(idx) )
        alphaMax = min(alphaMax,min(-delt(idx,1)./delt_delt(idx,1)))
            ;
    end
    idx = find(delt_k < 0);
    if( ~isempty(idx) )
        alphaMax = min(alphaMax,min(-k(idx,1)./delt_k(idx,1)));
    end

    alpha = 0.995*alphaMax;

    %----------------------------- Update x,l,u,s,t,lam,mu,delt and k
        -----------
    x = x + alpha*delt_x;
    l = l + alpha*delt_l;
    u = u + alpha*delt_u;
    s = s + alpha*delt_s;
    t = t + alpha*delt_t;
    lam = lam + alpha*delt_lam;
    mu = mu + alpha*delt_mu;
    delt = delt + alpha*delt_delt;
    k = k + alpha*delt_k;

    %---------------------- Residuals and Duality Gap
        ---------------------------
    %rL = H*x + g - lam + mu - A*(delt - k)
    rL = H*x + g - lam + mu - SimpleStruct(A,(delt - k),2);
    rl = l - x + xl;
    ru = u + x - xu;
```

```
    %rs = s - A'*x +bl;
    rs = s - SimpleStruct(A,x,1) +bl;

    %rt = t + A'*x -bu;
    rt = t + SimpleStruct(A,x,1) -bu;

    r_l_lam = l.*lam;
    r_u_mu = u.*mu;
    r_s_delt = s.*delt;
    r_t_k = t.*k;

    %---------------------- Compute Duality Gap ------------------------------------
    Gap = (l'*lam + u'*mu + s'*delt + t'*k)/2/(m+n);

    Converged = (max(abs(rL)) < stop ) && ...
              (max(abs(rl)) < stop ) && ...
              (max(abs(ru)) < stop ) && ...
              (max(abs(rs)) < stop ) && ...
              ( abs(Gap) < stop );
    if nargout == 3
        xArray = [xArray; x'];
        lArray = [lArray; l'];
        uArray = [uArray; u'];
        sArray = [sArray; s'];
        tArray = [tArray; t'];
        lamArray = [lamArray; lam'];
        muArray = [muArray; mu'];
        deltArray = [deltArray; delt'];
        kArray = [kArray; k'];

        alphaArray(ite) = alpha;
    end

end

% Report solution and information
if Converged
    info = 1;
else
    info = 0;
end

if nargout == 3
    QPinfo = struct(...
                'x',xArray, ...
                'l',lArray, ...
                'u',uArray, ...
                's',sArray, ...
                't',tArray, ...
                'lam',lamArray, ...
                'mu',muArray, ...
                'delt',deltArray, ...
                'k',kArray, ...
                'iter',ite, ...
                'alpha',alphaArray);
end

function LDL = SimpleStruct(A,D,flag)
[rA,cA] = size(A);
[rD,cD] = size(D);
m = rA - cA;
if m == 1
    if flag == 1 % Lambda * x, A' * x
        LDL = -D(1:end-1) + D(2:end);
```

```
    end
    if flag == 2 % Lambda' * x, A * x
        LDL = [0; D] - [D; 0];
    end
    if flag == 3 % Lambda' * x * Lambda
        LDL = -diag(D,-1)-diag(D,1)+...
              diag([D; 0])+diag([0; D]);
    end
else
    DM = reshape(D,m,rD/m)';
    if flag == 1 % Lambda * x, A' * x
        LDL = -DM(1:end-1,:) + DM(2:end,:);
        LDL = reshape(LDL',cA,1);
    end
    if flag == 2 % Lambda' * x, A * x
        LDL = [zeros(1,m); DM] - [DM; zeros(1,m)];
        LDL = reshape(LDL',rA,1);
    end
    if flag == 3 % Lambda' * x * Lambda
        mainD = [zeros(1,m); DM] + [DM; zeros(1,m)];
        mainD = reshape(mainD',rA,1);
        LDL = -diag(D,-m)-diag(D,m)+diag(mainD);
    end

end
```

C.2 Example

C.2.1 Example 1

```
% EXAMPLE 1: Solve an unconstrained optimal control problem by
    control vector
%             paramterization or dynamic programming.
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Nov.2007

clear all
close all

%======== Define System ========================
nSys = 2;              % number of states
% load performance1
sys = drss(nSys);
[A,B,C,D] = ssdata(sys);
D = 0;

% weight matrix
Q = eye(1);
S = 0.0001*eye(1);

% reference
R = 1*[30*ones(1,50)];

```

```
%========== Initialization ==============================
x0 = zeros(nSys,1);
u_1 = 0;
xs = zeros(nSys,1);
us = 0;

%========== MPC Control Simulation ===============
N = 50; % predictive horizon

%[u,y] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,1);
[u,y] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,2);

%========== MPC Plot ==============================
figure
subplot(211);plot(0:length(R)-1,R(1,:),'r*-',0:length(R)-1,y(1,:)),
             legend('Location','Best','r','DP')
             ylabel('y')
             xlabel('t')
             title('SISO System')
             grid;

subplot(212);stairs(0:length(R)-1,u(1,:)),
             legend('Location','Best','DP')
             ylabel('u');
             xlabel('t');
             grid;
```

C.2.2 Example 2

```
% Example 2: Solve a simple convex inequality constrained QP problem
      by
%            interior-point method.
%
% By         : Jing Yang, s032574
% Subject    : Numerical Methods for Model Predictive Control
%              Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor : John Bagterp Jørgensen
% Date       : Nov.2007

clear all
close all

G = [1 0; 0 1];
g = [2; 1];

A = [-1 0.5 0.5;1 1 -1];
b = [-1 -2 -1]';

% Solve the QP
[x,lam,info,QPinfo] = InteriorPoint(G,g,A,b);

% Plot the problem
x1 = -3:0.01:1;
x2 = -3:0.01:1;

[X1,X2]=meshgrid(x1,x2);
F = 0.5*X1.^2 + 0.5*X2.^2 + 2*X1 + X2;
v = linspace(-12,5,100);

yc1 = x1 - 1;
yc2 = -0.5*x1 - 2;
```

```
yc3 = 0.5*x1 + 1;

figure
contour(X1,X2,F,v)
hold on;
xaxis = xlim;
yaxis = ylim;

fill([x1 x1(end) x1(1)],[yc1 yaxis(1) yaxis(1)],[0.7 0.7 0.7],'facealpha',0.2);
fill([x1 x1(end) x1(1)],[yc2 yaxis(1) yaxis(1)],[0.7 0.7 0.7],'facealpha',0.2);
fill([x1 x1(end) x1(1)],[yc3 yaxis(2) yaxis(2)],[0.7 0.7 0.7],'facealpha',0.2);

% Plot the solution
plot(QPinfo.x(:,1),QPinfo.x(:,2),'r-','linewidth',2)
plot(QPinfo.x(1,1),QPinfo.x(1,2),'.','markersize',25)
plot(QPinfo.x(2:end,1),QPinfo.x(2:end,2),'r.','markersize',25)
hold off
xlabel('x_1')
ylabel('x_2')
title('Contour Plot and Iteration Sequence','fontsize',12)

figure
subplot(211);
plot(0:size(QPinfo.x,1)-1,QPinfo.x(:,1),'-o','linewidth',2),grid
ylabel('x_1','fontsize',12)
ylim([-2.5 0.5])

subplot(212);
plot(0:size(QPinfo.x,1)-1,QPinfo.x(:,2),'-o','linewidth',2),grid
ylabel('x_2','fontsize',12)
xlabel('iteration #','fontsize',12)
ylim([-1.5 0.5])
```

C.3 Test Function

C.3.1 perform1.m

```
% PERFORM1: Test the performance of the control vector parameterization
%           method and the dynamic programming method on a SISO system.
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Nov.2007

clear all
close all

%========== Define System ============================

nSys = 2;        % state
```

```
load performance1
%sys = drss(nSys);
[A,B,C,D] = ssdata(sys);
D = 0;

% weight matrix
Q = eye(1);
S = 0.0001*eye(1);

% reference
R = 1*[30*ones(1,50)];

% predictive horizon
N = 50;

% start point
x0 = zeros(nSys,1);
u_1 = 0;

% steady state level
xs = zeros(nSys,1);
us = 0;

%========== MPC Control Simulation ==============
% control vector parameterization
[u1,y1] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,1);

% dynamic programming
[u2,y2] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,2);

%========== MPC Plot ============================
figure
subplot(211);plot(0:length(R)-1,R(1,:),'r*-',0:length(R)-1,y1(1,:)),
              hold on; plot(0:length(R)-1,y2(1,:),'o'),
              legend('Location','Best','r','VP','DP')
              ylabel('y')
              xlabel('t')
              title('SISO System')
              grid;

subplot(212);stairs(0:length(R)-1,u1(1,:)),
              hold on; stairs(0:length(R)-1,u2(1,:),'o'),
              legend('Location','Best','VP','DP')
              ylabel('u');
              xlabel('t');
              grid;
```

C.3.2 perform2.m

```
% PERFORM1: Test the performance of the control vector parameterization
%           method and the dynamic programming method on a 2X2 MIMO system.
%
% By         : Jing Yang, s032574
% Subject    : Numerical Methods for Model Predictive Control
%              Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor : John Bagterp Jørgensen
% Date       : Nov.2007

clear all
```

```
close all

%========== Define System =========================

nSys = 4;          % state
nInput = 2;        % input
nOutput = 2;       % output
%sys = drss(nSys,nOutput,nInput);
load performance2
[A,B,C,D] = ssdata(sys);
D = 0;

% weight matrix
Q = eye(nOutput);
S = 0.0001*eye(nInput);

% reference
R = 1*[30*ones(nOutput,15) 40*ones(nOutput,15) 20*ones(nOutput,20)];

% predictive horizon
N = 50;

% start point
x0 = zeros(nSys,1);
u_1 = zeros(nInput,1);

% steady state level
xs = zeros(nSys,1);
us = zeros(nInput,1);

%========== MPC Simulation ========================
% control vector parameterization
[u1,y1] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,1);

% dynamic programming
[u2,y2] = simMPC(A,B,C,Q,S,N,R,x0,u_1,xs,us,2);

%========== MPC Plot ==============================
figure
subplot(221);plot(0:length(R)-1,R(1,:),'r*',0:length(R)-1,y1(1,:)),
             hold on; plot(0:length(R)-1,y2(1,:),'o'),
             legend('Location','Best','r','VP','DP')
             ylabel('y1')
             xlabel('t')
             grid;

subplot(222);plot(0:length(R)-1,R(2,:),'r*',0:length(R)-1,y1(2,:)),
             hold on; plot(0:length(R)-1,y2(2,:),'o'),
             legend('Location','Best','r','VP','DP')
             ylabel('y2')
             xlabel('t')
             grid;

subplot(223);stairs(0:length(R)-1,u1(1,:)),hold on;
             stairs(0:length(R)-1,u2(1,:),'o'),
             legend('Location','Best','VP','DP')
             ylabel('u1')
             xlabel('t')
             grid;

subplot(224);stairs(0:length(R)-1,u1(2,:)),hold on;
             stairs(0:length(R)-1,u2(2,:),'o'),
             legend('Location','Best','VP','DP')
             ylabel('u2')
             xlabel('t')
```

```
            grid;
```

C.3.3 runningTimeN.m

```
% RUNNINGTIMEN: Test the effect of the predictive horizon on the running
%               time (CPU time) of solving problems by the control
%               vector parameterization(CVP) method and by the dynamic
%               programming(DP) method.
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Nov.2007

clear all
close all

%==================== Define System ==========================
nSys = 2;       % number of states
%load performance1
sys = drss(nSys);
[A,B,C,D] = ssdata(sys);
D = 0;

% weight matrix
Q = 1;
S = 0.0001;

% reference
R = 1;

% start point
x0 = zeros(nSys,1);
u_1 = 0;

% steady state level
xs = zeros(nSys,1);
us = 0;
zs = C*xs;

% form variable deviation
delt_u_1 = u_1-us;
x = x0-xs;

% expand state for dynamic programming
X = [x;delt_u_1];

%==================== Measure CPU time ==========================
i = 0;
for N = 10:10:500
  i = i + 1;

  % reference for CVP
  ref = [R(:,2:end) repmat(R(:,end),1,N)];   ref = ref(:);
  % reference for DP
  ref1 = [R repmat(R(:,end),1,N)];

  usN = repmat(us,N,1);
```

```
  zsN = repmat(zs,N,1);

  %------------------ CPU time of CVP ----------------------
  t0 = cputime;                  % CVP start
  [H,Mx0,Mum1,MR] = MPCDesignSE(A,B,C,Q,S,N);
  t0a = cputime;                 % CVP online computation start
  [up] = MPCComputeSE(H,Mx0,MR,Mum1,...
                          x,ref,zsN,delt_u_1,usN);
  tSE(i) = cputime-t0;           % CPU time of CVP
  tSEonline(i) = cputime-t0a;    % CPU time of online computation

  %------------------ CPU time of DP ----------------------
  x0 = zeros(nSys,1);
  t1 = cputime;                  % DP start
  [Abar,Bbar,Qbar,Mbar,Rbar,Pbar,P,L,Z] = MPCDesignDP(A,B,C,Q,S,N);
  t1a = cputime;                 % DP online computation start
  [up,delt_u] = MPCComputeDP(A,C,Q,S,X,usN,Abar,Bbar,...
                              Qbar,Mbar,Rbar,Pbar,...
                              P,L,Z,ref1,N);
  tDP(i) = cputime-t1;           % CPU time of DP
  tDPonline(i) = cputime-t1a;    % CPU time of online computation
end

%================== Plot result =========================
figure
plot(10:10:N,tSE,'-','LineWidth',2),hold on
plot(10:10:N,tDP,'r-','LineWidth',2),grid on
xlabel('Predictive Horizon (N)','FontSize',12)
xlim([0 501])
ylabel('CPU Time (s)','FontSize',12)
legend('CVP','DP','Location','Best')
set(gca,'FontSize',12)
hold off

figure
plot(10:10:N,tSEonline,'LineWidth',2),,hold on
plot(10:10:N,tDPonline,'r','LineWidth',2),grid on
xlabel('Predictive Horizon (N)','FontSize',12)
xlim([0 501])
ylabel('CPU Time (s)','FontSize',12)
legend('CVP','DP','Location','Best')
set(gca,'FontSize',12)
hold off
```

C.3.4 runningTimeState.m

```
% RUNNINGTIMESTATE: Test the effect of the number of states on the
    running
%                time (CPU time) of solving problems by the control
%                vector parameterization(CVP) method and by the
    dynamic
%                programming(DP) method.
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp J?rgensen
% Date        : Nov.2007

clear all
close all
```

```

% predictive horizon
N = 100;

% weight matrix
Q = 1;
S = 0.0001;

% reference
R = 1;
% reference for CVP
ref = [R(:,2:end) repmat(R(:,end),1,N)];   ref = ref(:);
% reference for DP
ref1 = [R repmat(R(:,end),1,N)];

u_1 = 0;
us = 0;
delt_u_1 = u_1-us;
usN = repmat(us,N,1);

%==================== Measure CPU time ==========================
i = 0;
for nSys = 10:10:300
     i = i+1;
     sys = drss(nSys);     % number of state
     [A,B,C,D] = ssdata(sys);
     D = 0;

     E = zeros(nSys,1);
     x0 = zeros(nSys,1);
     xs = zeros(nSys,1);
     x = x0-xs;
     zs = C*xs;
     zsN = repmat(zs,N,1);

     %------------------- CPU time of CVP -------------------
     t0 = cputime;                        % CVP start
     [H,Mx0,Mum1,MR] = MPCDesignSE(A,B,C,Q,S,N);
     t0a = cputime;                       % CVP online computation start
     [up] = MPCComputeSE(H,Mx0,MR,Mum1,...
                           x,ref,zsN,delt_u_1,usN);
     tSE(i) = cputime - t0;               % CPU time of CVP
     tSEonline(i) = cputime-t0a;          % CPU time of online computation

     %------------------- CPU time of DP --------------------
     X = [x;delt_u_1];    % expand state for dynamic programming
     t1 = cputime;        % DP start
     [Abar,Bbar,Qbar,Mbar,Rbar,Pbar,P,L,Z] = MPCDesignDP(A,B,C,Q,S,N);
     t1a = cputime;               % DP online computation start
     [up,delt_u] = MPCComputeDP(A,C,Q,S,X,usN,Abar,Bbar,...
                                 Qbar,Mbar,Rbar,Pbar,...
                                 P,L,Z,ref1,N);
     tDP(i) = cputime - t1;       % CPU time of DP
     tDPonline(i) = cputime-t1a;  % CPU time of online computation
end

%==================== Plot result ========================
figure
plot(10:10:nSys,tSE,'LineWidth',2), hold on
plot(10:10:nSys,tDP,'r','LineWidth',2), grid on
xlabel('Number of State (n)','FontSize',12)
ylabel('CPU Time (s)','FontSize',12)
ylim([-0.05 max(tDP)*1.02])
title('CPU time vs. Number of states','FontSize',12)
```

```
legend('CVP','DP','Location','Best')
set(gca,'FontSize',12)

figure
plot(10:10:nSys,tSEonline,'LineWidth',2),hold on
plot(10:10:nSys,tDPonline,'r','LineWidth',2),grid on
xlabel('Number of State (n)','FontSize',12)
ylabel('CPU Time (s)','FontSize',12)
ylim([-0.05 max(tDPonline)*1.02])
title('Online CPU time vs. Number of states','FontSize',12)
legend('CVP','DP','Location','Best')
set(gca,'FontSize',12)
hold off
```

C.3.5 runningTimeInput.m

```
% RUNNINGTIMEINPUT: Test the effect of the number of inputs on the
    running
%                   time (CPU time) of solving problems by the control
%                   vector parameterization(CVP) method and by the
    dynamic
%                   programming(DP) method.
%
% By           : Jing Yang, s032574
% Subject      : Numerical Methods for Model Predictive Control
%                Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor   : John Bagterp J?rgensen
% Date         : Nov.2007

clear all
close all

% predictive horizon
N = 50;

% weight matrix
Q = 1;

% reference
R = 1;
% reference for CVP
ref = [R(:,2:end) repmat(R(:,end),1,N)];   ref = ref(:);
% reference for DP
ref1 = [R repmat(R(:,end),1,N)];

nSys = 2;          % number of states

x0 = zeros(nSys,1);
xs = zeros(nSys,1);
x = x0-xs;

%==================== Measure CPU time ==========================
i = 0;
for Input = 5:5:100
    i = i+1;
    sys = drss(nSys,1,Input);
    [A,B,C,D] = ssdata(sys);
    D = 0;

    S = 0.0001*eye(Input);

```

```
    u_1 = zeros(Input,1);
    us = zeros(Input,1);
    usN = repmat(us,N,1);
    delt_u_1 = u_1-us;

    zs = C*xs;
    zsN = repmat(zs,N,1);

    %------------------------ CPU time of CVP ------------------------
    t0 = cputime;                           % CVP start
    [H,Mx0,Mum1,MR] = MPCDesignSE(A,B,C,Q,S,N);
    t0a = cputime;                          % CVP online computation start
    [up] = MPCComputeSE(H,Mx0,MR,Mum1,...
                        x,ref,zsN,delt_u_1,usN);
    tSE(i) = cputime - t0;                  % CPU time of CVP
    tSEonline(i) = cputime-t0a; % CPU time of online computation

    %------------------------ CPU time of DP ------------------------
    X = [x;delt_u_1];       % expand state for dynamic programming
    t1 = cputime;           % DP start
    [Abar,Bbar,Qbar,Mbar,Rbar,Pbar,P,L,Z] = MPCDesignDP(A,B,C,Q,S,N);
    t1a = cputime;              % DP online computation start
    [up1,delt_u] = MPCComputeDP(A,C,Q,S,X,usN,Abar,Bbar,...
                                Qbar,Mbar,Rbar,Pbar,...
                                P,L,Z,ref1,N);
    tDP(i) = cputime -t1;     % CPU time of DP
    tDPonline(i) = cputime-t1a; % CPU time of online computation
end

%========================== Plot result ==========================
figure
plot(5:5:Input,tSE,'LineWidth',2), hold on
plot(5:5:Input,tDP,'r','LineWidth',2),grid,hold off
xlabel('Number of Input (m)','FontSize',12)
xlim([0 100])
ylabel('CPU Time (s)','FontSize',12)
title('CPU time vs. Number of inputs','FontSize',12)
legend('CVP','DP','Location','Best')
set(gca,'FontSize',12)

figure
plot(5:5:Input,tSEonline,'LineWidth',2),hold on
plot(5:5:Input,tDPonline,'r','LineWidth',2),grid on
xlabel('Number of Input (m)','FontSize',12)
xlim([0 100])
ylabel('CPU Time (s)','FontSize',12)
title('Online CPU time vs. Number of inputs','FontSize',12)
legend('CVP','DP','Location','Best')
set(gca,'FontSize',12)
hold off

figure
plot(5:5:Input,tDPonline,'r','LineWidth',2), grid
xlabel('Number of Input (m)','FontSize',12)
ylabel('Online CPU Time (s)','FontSize',12)
title('Online CPU time of DP vs. Number of inputs','FontSize',12)
set(gca,'FontSize',12)
```

C.3.6 runningTime.m

```
% RUNNINGTIMEINPUT: Test the combined effect of the predictive horizon,
%                   the number of states and the number of inputs on the
%                   runningtime (CPU time) of solving problems by the control
%                   vector parameterization(CVP) method and by the dynamic
%                   programming(DP) method.
%
% By         : Jing Yang, s032574
% Subject    : Numerical Methods for Model Predictive Control
%              Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor : John Bagterp Jørgensen
% Date       : Nov.2007

clear all
close all

N_state = [2 5 10 20 40 100];  % number of state
length_N_state = length(N_state);

Hori = 500;
nInput = 5;
%nInput = 1;

for l = 1: length_N_state
      nSys = N_state(l);
      sys = drss(nSys,1,nInput);
      [A,B,C,D] = ssdata(sys);
      D = 0;

      % weight matrix
      Q = 1;
      S = 0.0001*eye(nInput);

      % reference
      R = 1;

      % start point
      x0 = zeros(nSys,1);
      u_1 = zeros(nInput,1);
      % expand state for dynamic programming
      X = [x0;u_1];

      % steady state level
      xs = zeros(nSys,1);
      us = zeros(nInput,1);
      zs = C*xs;

      % form variable deviation
      x = x0-xs;
      delt_u_1 = u_1-us;

%==================== Measure CPU time ============================
      i = 0;
      for N = 10:10:Hori
          i = i+1;
          usN = repmat(us,N,1);
          zsN = repmat(zs,N,1);

          %------------------ CPU time of DP -----------------------
          ref = [R repmat(R(:,end),1,N)];
```

```
            t0 = cputime;               % DP start
            [Abar,Bbar,Qbar,Mbar,Rbar,Pbar,P,L,Z] = MPCDesignDP(A,B,C,Q,S,N);
            [up1,delt_u1] = MPCComputeDP(A,C,Q,S,X,usN,Abar,Bbar,...
                                         Qbar,Mbar,Rbar,Pbar,...
                                         P,L,Z,ref,N);
            tDP(1,i) = cputime-t0;   % CPU time of DP

            %------------------------------ CPU time of CVP --------------------
            x0 = zeros(nSys,1);
            ref1 = [R(:,2:end) repmat(R(:,end),1,N)]; ref1 = ref1(:);
            t1 = cputime;              % CVP start
            [H,Mx0,Mum1,MR] = MPCDesignSE(A,B,C,Q,S,N);
            [up2,delt_u2] = MPCComputeSE(H,Mx0,MR,Mum1,...
                                  x,ref1,zsN,delt_u_1,usN);
            tSE(1,i) = cputime-t1; % CPU time of CVP
        end
end

%====================== Plot result ===============================
% figure(1)
% subplot(321)
% plot(10:10:N,tSE(1,:),'LineWidth',2),hold on
% plot(10:10:N,tDP(1,:),'r','LineWidth',2),grid on
% ylabel('CPU Time (s)','FontSize',12)
% ylim([-.02 1.02*max(tSE(1,:))])
% title('Number of states = 2','FontSize',12)
% legend('CVP','DP','Location','Best')
% hold off
%
% subplot(322)
% plot(10:10:N,tSE(2,:),'LineWidth',2),hold on
% plot(10:10:N,tDP(2,:),'r','LineWidth',2),grid on
% ylim([-.02 1.02*max(tSE(2,:))])
% title('Number of states = 5','FontSize',12)
% legend('CVP','DP','Location','Best')
% hold off
%
% subplot(323)
% plot(10:10:N,tSE(3,:),'LineWidth',2),hold on
% plot(10:10:N,tDP(3,:),'r','LineWidth',2),grid on
% ylabel('CPU Time (s)','FontSize',12)
% ylim([-.02 1.02*max(tSE(3,:))])
% title('Number of states = 10','FontSize',12)
% legend('CVP','DP','Location','Best')
% hold off
%
% subplot(324)
% plot(10:10:N,tSE(4,:),'LineWidth',2),hold on
% plot(10:10:N,tDP(4,:),'r','LineWidth',2),grid on
% ylim([-.02 1.02*max(tSE(4,:))])
% title('Number of states = 20','FontSize',12)
% legend('CVP','DP','Location','Best')
% hold off
%
% subplot(325)
% plot(10:10:N,tSE(5,:),'LineWidth',2),hold on
% plot(10:10:N,tDP(5,:),'r','LineWidth',2),grid on
% xlabel('Predictive Horizon (N)','FontSize',12)
% ylabel('CPU Time (s)','FontSize',12)
% ylim([-.02 1.02*max(tSE(5,:))])
% title('Number of states = 40','FontSize',12)
% legend('CVP','DP','Location','Best')
% hold off
%
```

```
% subplot(326)
% plot(10:10:N,tSE(6,:),'LineWidth',2),hold on
% plot(10:10:N,tDP(6,:),'r','LineWidth',2),grid on
% xlabel('Predictive Horizon (N)','FontSize',12)
% ylim([-.02 1.02*max(tDP(6,:))])
% title('Number of states = 100','FontSize',12)
% legend('CVP','DP','Location','Best')
% hold off

figure(2)
rate = tSE./tDP;
plot(10:10:N,rate(1,:),'LineWidth',2),hold on
plot(10:10:N,rate(4,:),'k','LineWidth',2),
plot(10:10:N,rate(5,:),'r','LineWidth',2),
plot(10:10:N,rate(6,:),'c','LineWidth',2)
plot(0:N,ones(1,N+1),'r--'),grid, hold off
xlim([0 500])
ylim([-0.1 4])
title('m = 5','FontSize',11)
xlabel('Predictive Horizon (N)','FontSize',11)
ylabel('Rate of CPUTime (CVP/DP)','FontSize',11)
legend('state = 2','state = 20','state = 40','state = 100','Location','Best')
set(gca,'FontSize',11)
```

C.3.7 CompareAlg.m

```
% COMPAREALG: Compare two procedureS for solving the extend LQ optimal
%         problem. Algorithm 1 provides whole steps to solve problem.
%         The sequence of Algorithm 2 and Algorithm 3 factorize the matrices
%         first and then solve the factorized problem.
%
% By           : Jing Yang, s032574
% Subject      : Numerical Methods for Model Predictive Control
%                Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor   : John Bagterp Jørgensen
% Date         : Nov.2007

clear all
close all

states = [2 20 50 100];  % number of states
for j = 1: length(states)
    nSys = states(j);
    sys = drss(nSys);
    [A,B,C,D] = ssdata(sys);
    D = 0;

    % weight matrix
    Q = 1;
    S = 0.0001;

    % reference
    R = 1;

    % start point
    x0 = zeros(nSys,1);
    u_1 = 0;

```

```
    % steady state level
    xs = zeros(nSys,1);
    us = 0;
    zs = C*xs;

    % form variable deviation
    delt_u_1 = u_1-us;
    x = x0-xs;

    % expand state
    X = [x;delt_u_1];

%======================= Measure CPU time ==============================
    i = 0;
    for N = 10:10:500
        i = i+1;
        ref1 = [R repmat(R(:,end),1,N)];    % reference

        % transform optimal control problem into the extended LQ
        % optimal problem
        [Abar,Bbar,Qbar,Mbar,Rbar,Pbar] = ...
                        DesignDPU(A,B,C,Q,S);
        [b,qbar,rbar,f,pbar,gamma] = DesignDPA(A,C,Q,S,ref1,N);

        n = size(Qbar,1); % number of states x
        m = size(Rbar,1); % number of input u

        % prelocated
        P = zeros((N+1)*n,n);
        L = zeros(N*m,m);
        Z = zeros(N*m,n);
        x = zeros(n,N);
        u = zeros(m,N);
        Xnew = zeros(n,N);
        delt_u = zeros(m,N);

        %------------------ CPU time of Algorithm 1 ----------------------
        t2 = cputime;                  % Algorithm 1 start
        [x,u] = DPFactSolve (Qbar,Mbar,Rbar,qbar,rbar,f,Abar',Bbar',b,...
                            X,Pbar,pbar,gamma,N);
%           [x,u] = DPFactSolve (Qbar,Mbar,Rbar,qbar,rbar,f,Abar',Bbar',b,...
%                               X,Pbar,pbar,gamma,N);
        tDP1(j,i) = cputime - t2; %CPU time of Algorithm 1

        %------------------ CPU time of Algorithm 2+3 ----------------------
        t1 = cputime;                  % Algorithm 2 start
        % Algorithm 2
        [P,L,Z] = factorize(Qbar,Mbar,Rbar,Abar',Bbar',Pbar,N);
        % Algorithm 3
        [Xnew,delt_u] = solveELQ (Qbar,Mbar,Rbar,qbar,rbar,f,Abar',Bbar',b,...
                            X,Pbar,pbar,gamma,N,P,L,Z);
%           Algorithm 3
%           [Xnew,delt_u] = solveELQ (Qbar,Mbar,Rbar,qbar,rbar,f,Abar',Bbar',b,...
%                               X,Pbar,pbar,gamma,N,P,L,Z);
        tDP(j,i) = cputime - t1;  %CPU time of Algorithm 2+3
    end
end

%======================= Plot result ==============================
```

```
subplot(221)
plot(10:10:N,tDP(1,:),'LineWidth',2),hold on
plot(10:10:N,tDP1(1,:),'r','LineWidth',2),grid on
ylabel('CPU Time (s)','FontSize',12)
title('n = 2')
legend('A2+A3','A1','Location','Best')
set(gca,'FontSize',12)
hold off

subplot(222)
plot(10:10:N,tDP(2,:),'LineWidth',2),hold on
plot(10:10:N,tDP1(2,:),'r','LineWidth',2),grid on
title('n = 20')
legend('A2+A3','A1','Location','Best')
set(gca,'FontSize',12)
hold off

subplot(223)
plot(10:10:N,tDP(3,:),'LineWidth',2),hold on
plot(10:10:N,tDP1(3,:),'r','LineWidth',2),grid on
xlabel('Predictive Horizon (N)','FontSize',12)
ylabel('CPU Time (s)','FontSize',12)
title('n = 50')
legend('A2+A3','A1','Location','Best')
set(gca,'FontSize',12)
hold off

subplot(224)
plot(10:10:N,tDP(4,:),'LineWidth',2),hold on
plot(10:10:N,tDP1(4,:),'r','LineWidth',2),grid on
xlabel('Predictive Horizon (N)','FontSize',12)
title('n = 100')
legend('A2+A3','A1','Location','Best')
set(gca,'FontSize',12)
hold off
```

C.3.8 testMPCInteriorPoint.m

```
% TESTMPCInteriorPoint: Test the implementation of interior-point method
%         for MPC with input and input rate constraints.
%
% By           : Jing Yang, s032574
% Subject      : Numerical Methods for Model Predictive Control
%                Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor   : John Bagterp Jørgensen
% Date         : Nov.2007

clear all
close all
%------------------------ Generate Model ------------------------
nSys = 2;  % the number of states
n = 1;     % the number of inputs
m = 1;     % the number of outputs

load interiorP
%sys = drss(nSys)
[A,B,C,D] = ssdata(sys);
sys.d = 0;
syszpk = zpk(sys);

```

```
step(sys),grid % step response
%------------------------- weight matrices -------------------------
Q = 1;
S = 0.0001;

N = 3; % predictive horizon
%------------------------- initial ----------------------------------
x0 = zeros(nSys,1);
u_1 = 0;
%------------------------- stable value -----------------------------
xs = zeros(nSys,1);
us = 0;
zs = C*xs;
usN = repmat(us,N,1);
zsN = repmat(zs,N,1);
%------------------------- deviation --------------------------------
d_u_1 = u_1-us;
x = x0-xs;
z = C*x;
%------------------------ input constraints ------------------------
umin = 0;
umax = 100;
Umin = repmat(umin,N,1);
Umax = repmat(umax,N,1);
%------------------------- input rate constraints ------------------
dumin = -10;
dumax = 10;
DUmin = repmat(dumin,N-1,1);
DUmax = repmat(dumax,N-1,1);

%------------------------- reference --------------------------------
R = [0*ones(1,5) 20*ones(1,25) 0*ones(1,15)];
R1 = [R(:,2:end) repmat(R(:,end),1,N)];

%------------------------ MPC design --------------------------------
[H,Mx0,Mum1,MR,Lambda] = MPCDesignSE(A,B,C,Q,S,N);

%========================== Simulation ==============================
t = 0:length(R)-1; % time sequence

% prelocated u and y
u = zeros(1,length(t));     % input
y = zeros(1,length(t));     % plant output

count = 0;
for k = 1: length(t)
    count = count +1;
    y(:,k) = z;

    ref = R1(:,k:k+N-1); ref = ref(:);
    %--------------- Computation ---------------------------------
    g = Mx0*x + MR*(ref-zsN) + Mum1*d_u_1;

    Umin(1:n,:) = max(umin, dumin+d_u_1);
    Umax(1:n,:) = min(umax, dumax+d_u_1);
    [delt_u_1,info,QPinfo] = MPCInteriorPoint(H,g,Lambda',Umin,Umax,
        DUmin,DUmax);

    up = delt_u_1+us;  % physical input
    u(:,k) = up(1:m);
    d_u_1 = up(1:m);

    %--------------- Prediction ----------------------------------
    % predict states and outputs
    [xp,zp] = MPCPredict(x,up,N,A,B,C);
```

```
    %------------ Update ---------------------------------
    % update the current state
    x = xp(1:nSys);

    % form output deviation and store the first one
    z = zp - zs;
    z = z(1:1);
end

%================= Plot result ==========================
figure
subplot(211)
plot(1:k,y,'LineWidth',2),hold on
plot(1:k,R,'r*'),grid
xlim([0 max(k)*(1+.02)])
ylabel('y','FontSize',12)
ylim([-1 max(R)*(1+.05)])
legend('y','r')
set(gca,'FontSize',12), hold off

subplot(212)
stairs(1:k,u,'LineWidth',2),grid
xlabel('t(s)','FontSize',12)
xlim([0 max(k)*(1+.02)])
ylabel('u','FontSize',12)
ylim([-1 max(u)*(1+.05)])
set(gca,'FontSize',12)
```

C.3.9 testMPCInteriorPointTime-N.m

```
% TESTMPCINTERIORPOINTTIM_N: Test the effect of the predictive
    horizon on
%         the computational time of solving the constrained optimal
%         control problem arsing by control vector parameterization
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Nov.2007

clear all
close all
%----------------- Generate Model -------------------------
nSys = 2;      % the number of states
n = 1;         % the number of inputs
m = 1;         % the number of outputs
load interiorMPC1
%sys = drss(nSys)
[A,B,C,D] = ssdata(sys);
sys.d = 0;

%----------------- weight matrices ------------------------
Q = 1;
S = 0.0001;
%----------------- reference ------------------------------
R = [20*ones(1,3) 20*ones(1,7)];
R = repmat(R,1,60);
%----------------- initial --------------------------------
x0 = zeros(nSys,1);
```

```
u_1 = 0;
%------------------------ stable value ------------------------------
xs = zeros(nSys,1);
us = 0;
zs = C*xs;
%------------------------ deviation ---------------------------------
d_u_1 = u_1-us;
x = x0-xs;
z = C*x;

%========================== Measure CPU time ===========================
count = 0;
for N = 10:10:300
    count = count + 1;
    usN = repmat(us,N,1);
    zsN = repmat(zs,N,1);
    %------------------------ input constraints ----------------------
    umin = 0;
    umax = 100;
    Umin = repmat(umin,N,1);
    Umax = repmat(umax,N,1);
    %------------------------ input rate constraints ------------------
    dumin = -10;
    dumax = 10;
    DUmin = repmat(dumin,N-1,1);
    DUmax = repmat(dumax,N-1,1);
    %------------------------ reference --------------------------------
    ref = R(1:N)';
    %------------------------ CPU time ---------------------------------
    tCPU = cputime;
    % MPC design
    [H,Mx0,Mum1,MR,Lambda] = MPCDesignSE(A,B,C,Q,S,N);
    % Computation
    g = Mx0*x + MR*(ref-zsN) + Mum1*d_u_1;

    Umin(1:n,:) = max(umin, dumin+d_u_1);
    Umax(1:n,:) = min(umax, dumax+d_u_1);
    [delt_u_1,info,QPinfo] = MPCInteriorPoint(H,g,Lambda',Umin,Umax,
        DUmin,DUmax);
    T(count) = cputime-tCPU;
end

%====================== Plot result ====================================
plot(10:10:N,T,'LineWidth',2),grid
xlabel('Predictive Horizon','FontSize',12)
xlim([0 N])
ylabel('CPU Time (s)','FontSize',12)
title('CPU time vs. Predictive Horizon','FontSize',12)
set(gca,'FontSize',12)
```

C.3.10 testMPCInteriorPointTime-Input.m

```
% TESTMPCINTERIORPOINTTIM_INPUT: Test the effect of the number of
%          input on the computational time of solving the constrained
%          optimal control problem arsing by control vector
%          parameterization
%
% By           : Jing Yang, s032574
% Subject      : Numerical Methods for Model Predictive Control
%                Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor   : John Bagterp Jørgensen
```

```
% Date          : Nov.2007

clear all
close all
%------------------------ Generate Model ------------------------------
nSys = 2;       % the number of states
m = 1;          % the number of outputs
%------------------------ predictive horizon --------------------------
N = 100;
%------------------------ reference -----------------------------------
R = [20*ones(1,3) 20*ones(1,7)];
R = repmat(R,1,60);
ref = R(1:N)';
%======================== Measure CPU time ============================
count = 0;
for n = 5:5:55
    count = count +1;
    sys = drss(nSys,1,n);
    [A,B,C,D] = ssdata(sys);
    sys.d = 0;
    %------------------------ weight matrices -------------------------
    Q = 1;
    S = 0.0001*eye(n);
    %------------------------ initial ---------------------------------
    x0 = zeros(nSys,1);
    u_1 = zeros(n,1);
    %------------------------ stable value ----------------------------
    xs = zeros(nSys,1);
    us = zeros(n,1);
    zs = C*xs;
    %------------------------ deviation -------------------------------
    d_u_1 = u_1-us;
    x = x0-xs;
    z = C*x;
    %------------------------ -----------------------------------------
    usN = repmat(us,N,1);
    zsN = repmat(zs,N,1);
    %----------------------- input constraints ------------------------
    umin = 0*ones(n,1);
    umax = 100*ones(n,1);
    Umin = repmat(umin,N,1);
    Umax = repmat(umax,N,1);
    %------------------------ input rate constraints ------------------
    dumin = -10*ones(n,1);
    dumax = 10*ones(n,1);
    DUmin = repmat(dumin,N-1,1);
    DUmax = repmat(dumax,N-1,1);
    %----------------------- CPU time --------------------------------
    tCPU = cputime;
    % MPC design
    [H,Mx0,Mum1,MR,Lambda] = MPCDesignSE(A,B,C,Q,S,N);
    % Computation
    g = Mx0*x + MR*(ref-zsN) + Mum1*d_u_1;

    Umin(1:n,:) = max(umin, dumin+d_u_1);
    Umax(1:n,:) = min(umax, dumax+d_u_1);
    delt_u_1 = MPCInteriorPoint(H,g,Lambda',Umin,Umax,DUmin,DUmax);
    T(count) = cputime-tCPU;
end

%===================== Plot result ====================================
plot( 5:5:55,T,'LineWidth',2),grid
xlabel('Number of Inpute ','FontSize',12)
xlim([0 n])
ylabel('CPU Time (s)','FontSize',12)
```

```
title('CPU time vs. Number of inputs','FontSize',12)
set(gca,'FontSize',12)
```

C.3.11 testMPCInteriorPointTime-State.m

```
% TESTMPCINTERIORPOINTTIM_STATE: Test the effect of the number of
    states
%         on the computational time of solving the constrained
    optimal
%         control problem arsing by control vector parameterization
%
% By          : Jing Yang, s032574
% Subject     : Numerical Methods for Model Predictive Control
%               Master Thesis, IMM, DTU, DK-2800 Lyngby.
% Supervisor  : John Bagterp Jørgensen
% Date        : Nov.2007

clear all
close all
%---------------------- weight matrices ----------------------
Q = 1;
S = 0.0001;
%---------------------- predictive horizon ----------------------
N = 50;
%---------------------- reference ----------------------
R = [20*ones(1,3) 20*ones(1,7)];
R = repmat(R,1,60);
ref = R(1:N)';
%---------------------- input constraints ----------------------
umin = 0;
umax = 100;
Umin = repmat(umin,N,1);
Umax = repmat(umax,N,1);
%---------------------- input rate constraints ----------------------
dumin = -10;
dumax = 10;
DUmin = repmat(dumin,N-1,1);
DUmax = repmat(dumax,N-1,1);
%====================== Measure CPU time ======================
count = 0;
for nSys = 2:2:20
    count = count +1;
    %---------------------- Generate Model ----------------------
    n = 1;          % the number of inputs
    m = 1;          % the number of outputs
    sys = drss(nSys);
    [A,B,C,D] = ssdata(sys);
    sys.d = 0;
    %---------------------- initial ----------------------
    x0 = zeros(nSys,1);
    u_1 = 0;
    %---------------------- stable value ----------------------
    xs = zeros(nSys,1);
    us = 0;
    zs = C*xs;
    usN = repmat(us,N,1);
    zsN = repmat(zs,N,1);
    %---------------------- deviation ----------------------
    d_u_1 = u_1-us;
    x = x0-xs;
    z = C*x;
```

```
    %------------------------- CPU time ------------------------------------
    tCPU = cputime;
    % MPC design
    [H,Mx0,Mum1,MR,Lambda] = MPCDesignSE(A,B,C,Q,S,N);
    % Computation
    g = Mx0*x + MR*(ref-zsN) + Mum1*d_u_1;

    Umin(1:n,:) = max(umin, dumin+d_u_1);
    Umax(1:n,:) = min(umax, dumax+d_u_1);
    delt_u_1 = MPCInteriorPoint(H,g,Lambda',Umin,Umax,DUmin,DUmax);
    T(count) = cputime-tCPU;
end

%========================= Plot result ===================================
plot(2:2:nSys,T,'LineWidth',2),grid
xlabel('Number of State (n)','FontSize',12)
xlim([0 nSys])
ylabel('CPU Time (s)','FontSize',12)
title('CPU time vs. Number of states','FontSize',12)
set(gca,'FontSize',12)
```

Bibliography

[1] Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control,* second edition, volume 1. Athena Scientific, Belmont, Massachusettes. ISBN 1-886529-09-4.

[2] Brosilow, C. and Joseph, B. (2002). *Techniques of Model-Based Control.* Prentice Hall. ISBN-13: 978-0-13-028078-7

[3] Faires, J.D. Burden, R. *Numerical Methods,* seconde edition Brooks/Cole Publishing Company

[4] Froberg, C.-E. *Introduction to Numerical Analysis,* seconde edition Addision-Wesley Publishing Company, Inc

[5] Grewal, M.S. Andrews A.P. *Kalman Filtering: Theory and Practice Using MATLAB* John Wiley Sons, Inc.

[6] Jørgensen, J. B. (2005). *Moving Horizon Estimation and Control.* Ph.D. thesis, Department of Chemical Engineering, Technical University of Denmark.

[7] Jørgensen, J.B.; Madsen, K.; Nielsen, H.B. and Rojas, M. (2006). *Introduction to Optimization and Data Fitting.* Informatics and Mathematical Modeling, Technical University of Denmark.

[8] Maciejowski, J.M. (2002). *Predictive Control with Constraints.* Prentice Hall, Harlow, England. ISBN 0-201-39823-0.

[9] Mehne, H.H. Borzabadi, A.H. A numerical method for solving optimal control problem using state parameterization *Springer Science + Business Media B.V.2006*

[10] Nikolaou, M. *Model Predictive Controllers: A Critical Synthesis of Theory and Industrial Needs.* Chemical Engineering Dept, University of Houston, Houston.
http://www.chee.uh.edu/faculty/nikolaou/MPCtheoryRevised.pdf

[11] Nocedal, J. and Wright S.J. *Numerical Optimization,* seconde edition Springer. ISBN-13: 978-0387-30303-1

[12] Odelson, B.J.; Rajamani, M.R. and Rawlings, J.B. (2005). *A new autocovariance least-squares method for estimating noise covariances.*
http://jbrwww.che.wisc.edu/tech-reports/twmcc-2003-04.pdf

[13] Pannocchia, G. Rawlings, J.B. Disturbance Models for Offset-Free Model-Predictive Control. *AIChE Journal* February 2003 Vol 49, No.2

[14] Qin, S.J. and Badgwell, T.A. (2002). A Survey of Industrial Model Predictive Control Technology. *Control Engineering Practice,* 11, 733-764

[15] Rao, C.V.; Wright, S.J. and Rawlings, J.B. (1998). Application of Interior-Point Methods to Model Predictive Control. *Journal of Optimization Theory and Applications,* 99, 723-757

[16] Wolsey, L.A. (1998) *Interger Programming,* Jone Wiley and Sons, Inc. ISBN 0-471-28366-5