Model Predictive Control: EPFL ME425

Jiaxuan Zhang

January 30, 2022

Forword

Jiaxuan Zhang
January 30, 2022

Contents

Chapter 1

Lecture 1: Introduction

1.1 Basic Concepts

MPC vs Classical Control Loop

Compared to classical control loop, in MPC, the controller is replaced by an optimizer program

Receding Horizon Control

In MPC, each timestep, we optimize a horizon with length N, but we only implement the first control action.

The Receding Horizon Control somehow introduces feedback.

Constraints in Control

For control and constraints, optimal solution often appears near constraints

1.2 MPC Formulation

Mathematical Composition

$$
\begin{aligned}
u^*(x) := \text{argmin} \quad & x_N^\top Q_f x_N + \sum_{i=0}^{N-1} x_i^\top Q x_i + u_i^T R u_i & & \\
\text{s.t.} \quad & x_0 = x & & \text{measurement} \\
& x_{i+1} = Ax_i + Bu_i & & \text{system model} \\
& Cx_i + Du_i \leq b & & \text{constraints} \\
& R \succ 0, Q \succ 0 & & \text{performance weights}
\end{aligned} \tag{1.1}
$$

Procedure

1. Measure/Estimate Current State
2. Calculate the optimal input sequence for the whole horizon N
3. Only implement the first action in the sequence

Main Challenges

- **Feasibility**: Sometimes the constraints cannot be fully satisfied at the same time
- **Implementation**: May not guarantee real-time control

- **Robustness**: The system may not be stable with disturbance and uncertainty
- **Stability**: Convergence may not be guaranteed

Chapter 2

Lecture 2: Unconstrained MPC: LQR

2.1 Model and Motivation of LQR

2.1.1 Model

Model 2.1.1 (cost of state)**.**

$$I(x,u) = x^T Q x + u^T R u$$

Model 2.1.2 (cost of trajectories)**.**

$$V(x_0,u) = \sum_{i=0}^{N} x_i^T Q x_i + u_i^T R u_i$$

2.1.2 Motivation

1. Can adjust the weight between input and state
2. Can be easily and quickly solve in Embedded Controller

2.2 Solution of Finite-Horizon LQR

2.2.1 Dynamic Programming Method (Bellman Recursion)

Basic Idea

This problem has the **optimal sub-structure** property: For the given optimal input sequence, no matter x_n, the input from n to N should make the sub-structure is also the optimal, otherwise, we can build a better solution from x_n which made the total value smaller than current optimal solution. Which also indicates:

1. no matter how we arrive at a state x_n, the optimal solution from x_n only depend on x_n, not related to how we arrive at x_n
2. Start from tail and iteration forward, we can find the best solution

Process

In the finite-horizon case, we explicitly knows that the final state is a state with quadratic cost $x^T H x$. Then we can use mathematical induction to prove $V^*(x)$ is quadratic. Assume for timestamp $i+1$, it holds, then

put it into timestamp i.

$$\begin{aligned} V_i\left(x_i\right) &= \min_{u_i} x_i^T Q x_i + u_i^\top R u_i + V_{i+1}\left(A x_i + B u_i\right) \\ &= \min_{u_i} \left(x_i^\top Q x_i + u_i^\top R u_i + \left(A x_i + B u_i\right)^\top H_{i+1} \left(A x_i + B u_i\right) \right) \end{aligned}$$

.

Then we can take the derivative to zero and get the $V_i^*(x)$.

1. start from the last step N and compute

$$\begin{aligned} V_N^\star\left(x_N\right) &:= \min_{u_N} x_N^T Q x_N + u_N^T R u_N \\ &= x_N^\top Q x_N \quad \text{(Quardratic)} \\ H_N &:= Q \end{aligned}$$

2. Iterate backwards for $i = N-1, \cdots, 0$, do DP iterations

$$V_i^\star\left(x_i\right) := \min_{u_i} x_i^\top Q x_i + u_i^\top R u_i + V_{i+1}^\star\left(A x_i + B u_i\right)$$

. A closed-form equation can be calculated:

$$\begin{aligned} u_i^\star\left(x_i\right) &= K_i x_i \quad K_i = -\left(R + B^\top H_{i+1} B\right)^{-1} B^\top H_{i+1} A \\ V_i^\star\left(x_i\right) &= x_i^T H_i x_i \quad H_i := Q + K_i^T R K_i + \left(A + B K_i\right)^\top H_{i+1}\left(A + B K_i\right) \end{aligned} \tag{2.1}$$

3. Finally, we will get $V^*(x_0) = V_0^*(x_0)$ and the optimal controller $u_0^*(x_0)$

Property

1. $V_i^*(x)$ is **quadratic** (can be proved by mathematical induction method)
2. $V_i^*(x)$ is **positive definite** (can be proved by mathematical induction method)
3. Optimizer $u_0^*(x)$ is **linear** of current state

2.2.2 Receding Horizon Method

Receding Horizon Method use optimization perspective to solve this problem. It first represent the problem to a model with single variable.

Model

$$V^\star\left(x_0\right) := \min_{\mathbf{u}} \quad \mathbf{x}^\top \mathcal{Q} \mathbf{x} + \mathbf{u}^\top \mathcal{R} \mathbf{u} \quad \text{s.t.} \quad \mathcal{A}\mathbf{x} + \mathcal{B}\mathbf{u} = \mathcal{C} x_0 \tag{2.2}$$

where

$$\mathbf{x} = \begin{bmatrix} x_1^T & \cdots & x_N^T \end{bmatrix}^T, \mathbf{u} = \begin{bmatrix} u_0^T & \cdots & u_{N-1}^T \end{bmatrix}^T$$

$$\mathcal{A} := \begin{bmatrix} -1 & 0 & \cdots & \cdots & \cdots & 0 \\ A & -1 & 0 & \cdots & \cdots & 0 \\ 0 & A & -1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & A & -1 \end{bmatrix} \quad \mathcal{B} := \begin{bmatrix} B & 0 & \cdots & 0 \\ 0 & B & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & B \end{bmatrix} \quad \mathcal{C} := \begin{bmatrix} -A \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.3}$$

$$\mathcal{Q} := \text{diag}(Q, \ldots, Q) \quad \mathcal{R} := \text{diag}(R, \ldots, R) \tag{2.4}$$

Solution

The solution of this perspective is:

$$\mathbf{u} = \mathcal{K}x_0 = \begin{bmatrix} K_0 \\ \vdots \\ K_{N-1} \end{bmatrix} x_0 \quad \mathcal{K} = -\left(\mathcal{R} + F^T \mathcal{Q} F\right)^{-1} F^T \mathcal{Q} G \tag{2.5}$$

2.2.3 Comparison

- Although the u with different equation, but they are the same in several simulation in MATLAB
- Because their u based on different reference, so the K(DB) may not equal to K(FRH)

Dynamic Programming

- leads to elegant closed-form solution for LQR
- provides a solution when $N \to \infty$
- Virtually no problems have simple, closed-form solutions (except LQR)

Optimization/Least Squares

- Can extend to nonlinear, constrained systems with complex cost-functions
- Finite horizon-only
- Computationally Intense

2.3 Stability of LQR Controllers

2.3.1 Problem of Finite Horizon Controller

Because using finite horizon, some early stage decision may be harmful to later stages optimization. One example of stability graph for system $x^+ = (A + BK_{R,N})\, x$ in Figure 2.3.1.

Another example is shown in Figure 2.3.1:

2.3.2 Extending to Infinite Horizon

Assumption

Assume system is controllable, i.e. have input sequence that generates a bounded cost. This is only possible when x,u finally goes to zero.

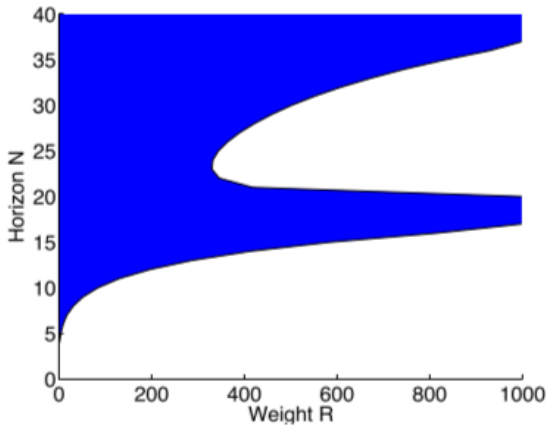
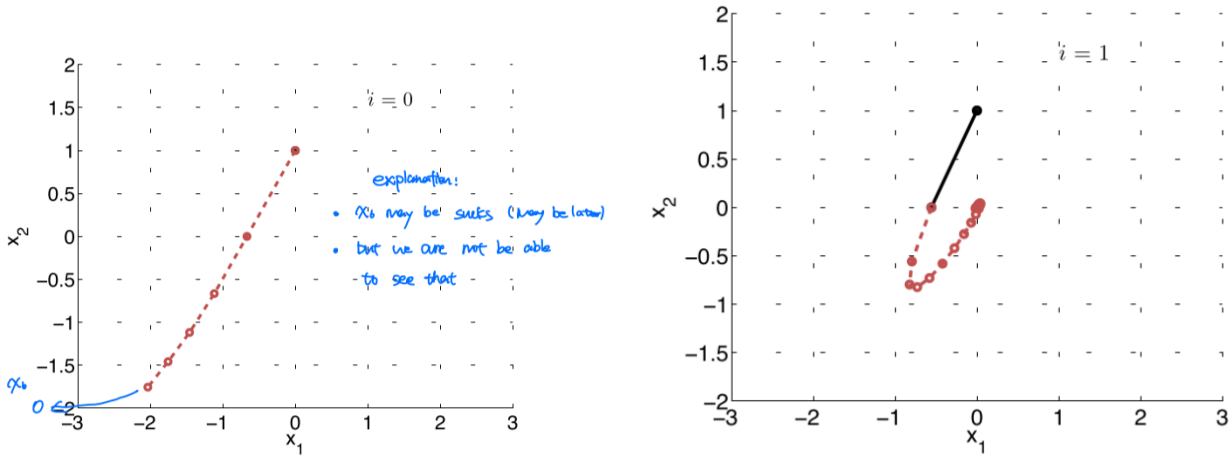


Figure 2.1: Problem of Finite Horizon Controller 1



(a) 2-1

(b) 2-2

Figure 2.2: Problem of Finite Horizon Controller 2

Solution

First, the optimal V should meet: $V^\star(x) := \min_u I(x,u) + V^\star(Ax+Bu)$. Second, there is a fact (can be proved by some complex mathematical theory): $V^*(x)$ is a quadratic function with form $V^*(x) = x^T P x$ with P positive definite. Then we can prove :

Theorem 2.3.1 (Lyapunov Function for LQR). *The optimal value function $V^\star(x) = x^\top P x$ is a **Lyapunov function** for the system $x^+ = (A+BK)x$ where $K = -\left(R+B^TPB\right)^{-1}B^TPA$ and P solves*

$$P = Q + A^TPA - A^TPB\left(R+B^TPB\right)^{-1}B^TPA$$

for some $Q \succeq 0, R \succ 0$

Conclusion

- Finite horizon LQR converges to static solution as $N \to \infty$
- Infinite-horizon LQR is nominally stabilizing

2.4 Summary

This chapter, we first learnt the LQR controller, we widely used it because it is solvable.

For a Finite-Horizon LQR controller, we can solve it by Dynamic Programming or Optimization Method. One big problem of Finite-Horizon LQR Controller is the finite-horizon may cause the system unstable.

Theoretically, an Infinite-Horizon LQR Controller can always stable, and a Finite-Horizon LQR Controller will converge to the Infinite-Horizon Controller as the horizon increase. And we can also get the closed-loop solution fo Infinite-horizon LQR Controller.

Chapter 3

Lecture 3: Basic Optimization

Chapter 4

Lecture 4: Introduction to Constrained Systems

4.1 Invariance

4.1.1 Conceptions

Definition 4.1.1 (Invariance). *Invariance:*
Region in which an ***autonomous*** *system will satisfy the constraints* ***for all time***

Definition 4.1.2 (Positive Invariant Set). *Positive Invariant Set:*
A set $\mathcal{O}$ is said to be a ***positive invariant set*** *for the autonomous system $x_{i+1} = f(x_i)$ if*

$$x_i \in \mathcal{O} \Rightarrow x_i \in \mathcal{O}, \forall i \in \{0, 1, \ldots\}$$

Notes: The invariant set provides a set of **initial states** from which the trajectory will **never violate** the system constraints.

Definition 4.1.3 (Maximal Positive Invariant Set). *Maximal Positive Invariant Set The set $\mathcal{O}_\infty \subset \mathbb{X}$ is the* ***maximal invariant set*** *with respect to $\mathbb{X}$ if $0 \in \mathcal{O}_\infty$. $\mathcal{O}_\infty$ is invariant and $\mathcal{O}_\infty$ contains all invariant sets that contain the origin.*

Notes: The maximal invariant set is the set of all states for which the system will remain feasible if it starts in $\mathcal{O}_\infty$.

Definition 4.1.4 (Preset). *Preset: Given a set S and the dynamic system $x^+ = f(x)$, the pre-set of S is the set of states that evolve into the target set S in one time step:*

$$\text{pre}(S) := \{x \mid f(x) \in S\}$$

4.1.2 Conditions of Invariant Set

A set $\mathcal{O}$ is a positive invariant set **if and only if**

$$\mathcal{O} \subset \text{pre}(\mathcal{O})$$

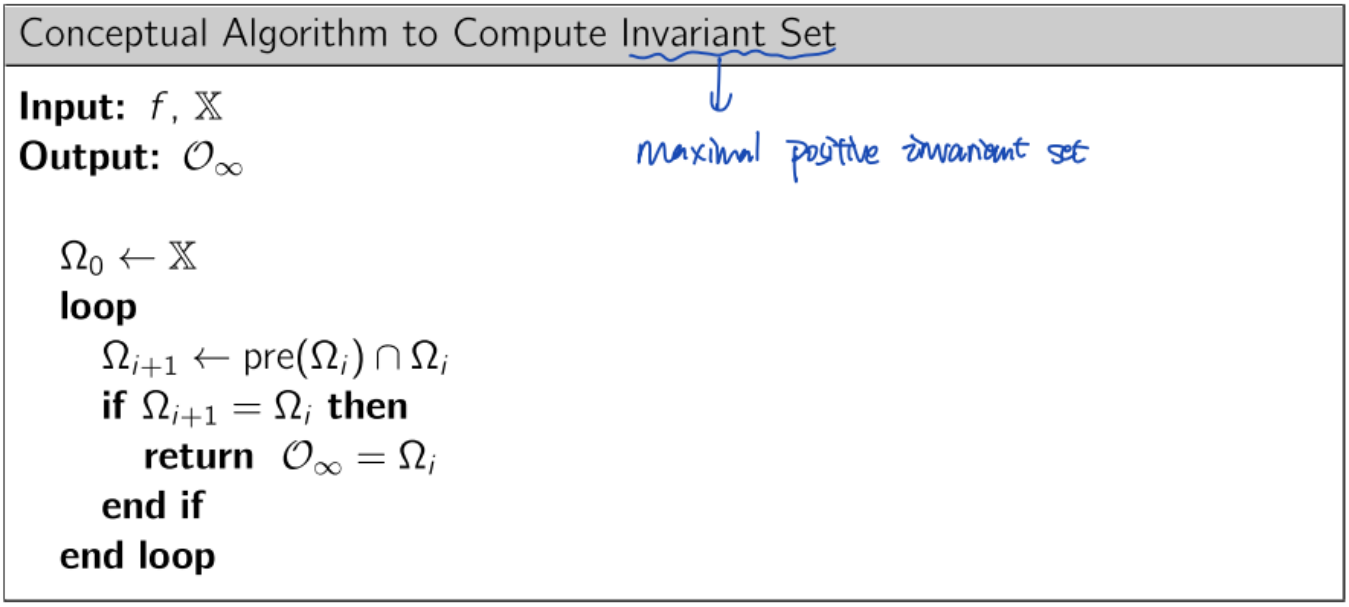


Figure 4.1: Computation of Maximal Invariant Set

4.1.3 Computation of Invariant Set

Notes: The algorithm generates the set sequence $\{\Omega_i\}$ satisfying $\Omega_{i+1} \subseteq \Omega_i$ for all $i \in \mathbb{N}$ and it terminates when $\Omega_{i+1} = \Omega_i$ so that Ω_i is the maximal positive invariant set $\mathcal{O}_\infty$ for $x^+ = f(x)$.

4.2 Control Invariance

4.2.1 Conceptions

Definition 4.2.1 (Controlled Invariance). *Controlled Invariance:*

*Region for which there **exists** a controller so that the system satisfies the constraints for all time*

Definition 4.2.2 (Control Invariant Set). *Control Invariant Set: A set $\mathcal{C} \subseteq \mathbb{X}$ is said to be a control invariant set if*

$$x_i \in \mathcal{C} \quad \Rightarrow \quad \exists u_i \in \mathbb{U} \quad \textit{such that } f(x_i, u_i) \in \mathcal{C} \quad \textit{for all } i \in \mathbb{N}^+$$

Definition 4.2.3 (Maximal Control Invariant Set). *Maximal Control Invariant Set:*

The set $\mathcal{C}_\infty$ is said to be the maximal control invariant set for the system $x^+ = f(x,u)$ subject to the constraints $(x,u) \in \mathbb{X} \times \mathbb{U}$ if it is control invariant and contains all control invariant sets contained in $\mathbb{X}$.

Definition 4.2.4 (Preset). *Preset:*

$$\text{pre}(S) := \{x \mid \exists u \in \mathbb{U} \ s.t. \ f(x,u) \in S\}$$

$\Omega_0 \leftarrow \mathbb{X}$
loop
  $\Omega_{i+1} \leftarrow \text{pre}(\Omega_i) \cap \Omega_i$
  if $\Omega_{i+1} = \Omega_i$ **then**
    return $\mathcal{C}_\infty = \Omega_i$
  end if
end loop

Figure 4.2: Computation of Maximal Controlled Invariant Set

4.2.2 Conditions of Controlled Invariant Set

A set $\mathcal{C}$ is a positive invariant set **if and only if**

$$\mathcal{C} \subset \text{pre}(\mathcal{C})$$

4.2.3 Computation of Controlled Invariant Set

4.2.4 Control Invariant Set and Control Law

4.3 Polytopes and Polytopic Computation

4.3.1 Conceptions

Definition 4.3.1 (Polytope and polyhedron). *[Polytope and polyhedron:*
*A **polyhedron** is the intersection of a finite number of halfspaces.*

$$P := \left\{x \mid a_i^T x \leq b_i, i = 1, \ldots, n\right\}$$

*A **polytope** is a bounded polyhedron.*

Definition 4.3.2 (Convex Hull). *Convex Hull For any subset S of $\mathbb{R}^d$, the convex hull conv (S) of S is the intersection of all convex sets containing S. Since the intersection of two convex sets is convex, it is the smallest convex set containing S.*

Theorem 4.3.3 (Minkowski-Weyl Theorem). *Minkowski-Weyl Theorem For $P \subseteq \mathbb{R}^d$, the following statements are equivalent:*

- *P is a polytope, i.e., P is bounded and there exist $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$ such that $P = \{x \mid Ax \leq b\}$*
- *P is finitely generated, i.e., there exist a finite set of vectors $\{v_i\}$ such that $P = \text{conv}\left(\{v_1, \ldots, v_s\}\right)$*

Definition 4.3.4 (Intersection). *The intersection $I \subseteq \mathbb{R}^n$ of sets $S \subseteq \mathbb{R}^n$ and $T \subseteq \mathbb{R}^n$ is*

$$I = S \cap T := \{x \mid x \in S \text{ and } x \in T\}$$

Notes: Intersection of polytopes in inequality form is easy:

$$\begin{array}{l} S := \{x \mid Cx \leq c\} \\ T := \{x \mid Dx \leq d\} \end{array} \quad S \cap T = \left\{ x \mid \begin{bmatrix} C \\ D \end{bmatrix} x \leq \begin{bmatrix} c \\ d \end{bmatrix} \right\}$$

Definition 4.3.5 (polytopic projection)**.** *polytopic projection Given a polytope* $P = \left\{(x, y) \in \mathbb{R}^n \times \mathbb{R}^d \mid Cx + Dy \leq b\right\}$ *find a matrix* E *and vector* e*, such that the polytope*

$$P_\pi = \{x \mid Ex \leq e\} = \{x \mid \exists y, (x, y) \in P\}$$

4.3.2 Polytopes in MPC

Input Saturation

$$\begin{array}{c} u_{lb} \leq u \leq u^{ub} \\ \Downarrow \\ \begin{bmatrix} 1 \\ -1 \end{bmatrix} u \leq \begin{bmatrix} u^{ub} \\ -u_{lb} \end{bmatrix} \end{array} \tag{4.1}$$

Magnitude Constraints

$$\begin{array}{c} \|Cx\|_\infty \leq \alpha \\ \Downarrow \\ \begin{bmatrix} C \\ -C \end{bmatrix} x \leq 1\alpha \end{array} \tag{4.2}$$

Rate Constraints

$$\begin{array}{c} \|x_i - x_{i+1}\|_\infty \leq \alpha \\ \Downarrow \\ \begin{bmatrix} 1 & -l \\ -1 & 1 \end{bmatrix} \begin{pmatrix} x_i \\ x_{i+1} \end{pmatrix} \leq \mathbf{1}\alpha \end{array} \tag{4.3}$$

Integral Constraints

$$\begin{array}{c} \|x\|_1 \leq \alpha \\ \Downarrow \\ x \in \operatorname{conv}(e_i \alpha) \end{array} \tag{4.4}$$

4.3.3 Computation of Pre-Set

Autonomous Systems

Method 4.3.6. *If* $S := \{x \mid Fx \leq f\}$*, then* $\operatorname{pre}(S) = \{x \mid FAx \leq f\}$

Controlled Systems

Consider the system $x^+ = Ax + Bu$ under the constraints $u \in \mathbb{U} := \{u \mid Gu \leq g\}$ and the set $S := \{x \mid Fx \leq f\}$.

$$\begin{aligned}\text{pre}(S) &= \{x \mid \exists u \in \mathbb{U}, Ax + Bu \in S\} \\ &= \{x \mid \exists u \in \mathbb{U}, FAx + FBu \leq f\} \\ &= \left\{x \mid \exists u, \begin{bmatrix} FA & FB \\ 0 & G \end{bmatrix} \begin{pmatrix} x \\ u \end{pmatrix} \leq \begin{bmatrix} f \\ g \end{bmatrix}\right\}\end{aligned}$$

Notes: this is actually a projection operation.

4.3.4 Equality Test

One important problem is how to check whether two set are the same? i.e. Is $P := \{x \mid Cx \leq c\}$ contained in $Q := \{x \mid Dx \leq d\}$?. The statement is true if $P \subset \{x | D_i x < d_i\}$ for each row D_i of D.

Define the support function of the set P :

$$\begin{aligned} h_P(D_i) := &\max_x D_i x \\ &\text{s.t. } Cx \leq c \end{aligned}$$

if $h_p(D_1) \leq d_1$, then it is true, if not , it is false.

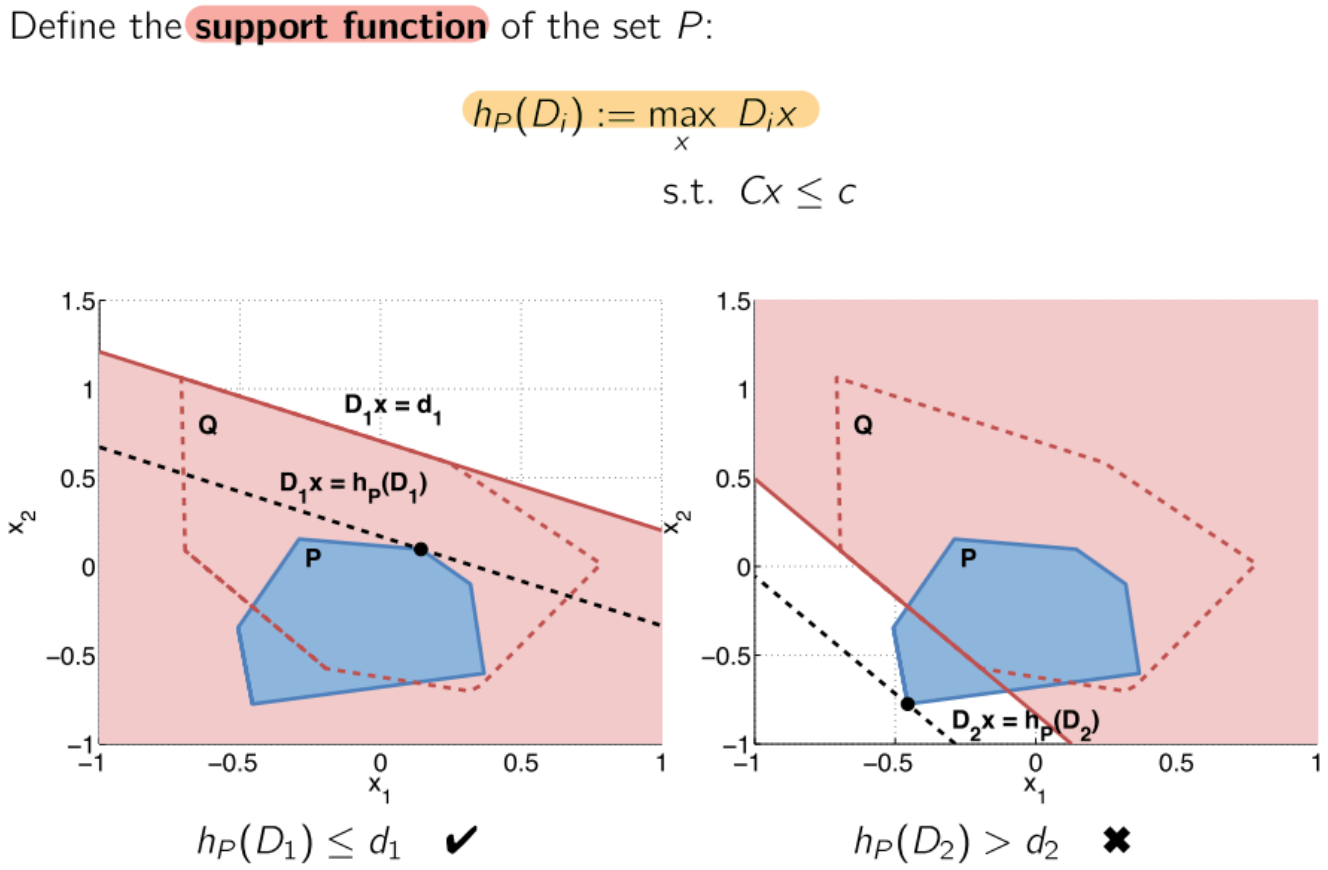


Figure 4.3: Equality Test

Notes: Do not try to translate to straight line representation. It can be directly understood by the definition of Q, if the false case happen, it means at least one of the constrain in Q is violated, because we use Q as $\leq$ format.

4.3.5 Convergence Discussion

Another problem is: Does the invariant set algorithm guarantees finite step termination?

In general, **no**! The boundary of the a maximal invariant set can be curvy, which needs infinite many half-space to define. In practice, to save memory and to ensure efficiency, the algorithm stop up to some **specific criteria or we use simpler convex set(i.e. box, ellipsoid)** to represent a smaller forward invariant set.

4.4 Ellipsoids and Invariance

4.5 Summary

This chapter introduce the content of invariant set and controlled invariant set and the method to compute them.

Chapter 5

Lecture 5: Basic MPC

5.1 Conception

Definition 5.1.1 (Feasible Set). *Feasible Set*

*The feasible set $\mathcal{X}_N$ is defined as the set of **initial states** x for which the MPC problem with horizen N is feasible, i.e. $\mathcal{X}_N := \{x \mid \exists [u_0, \ldots, u_{N-1}]$ such that $Cu_i + Dx_i \leq b, i = 1, \ldots, N\}$)*

Definition 5.1.2 (Recursive Feasibility). *Recursive Feasibility*

The MPC problem is called recursively feasible, if for all feasible initial states, feasibility is guaranteed at every state along the closed-loop trajectory

Definition 5.1.3 (Lyapunov Stability). *Lyapunov Stability*

The equilibrium point at the origin of system $x_{k+1} = Ax_k + B\kappa(x_k) = f_\kappa(x_k)$ is said to be (Lyapunov) stable in $\mathcal{X}$ if for every $\epsilon > 0$, there exists a $\delta(\epsilon) >$ such that, for every $x(0) \in (X)$

$$\|x(0)\| \leq \delta(\epsilon) \Rightarrow \|x(k)\| < \epsilon \forall k \in \mathbb{N}$$

5.2 Design a Stable MPC

5.2.1 Problem of Standard MPC Model

Model 5.2.1 (MPC Model Without Terminal Cost). *MPC Model Without Terminal Cost*

$$\begin{aligned} &\min_{x,u} \sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i \\ &s.t.\ x_{i+1} = Ax_i + Bu_i \\ &b \geq Cx_i + Du_i \end{aligned} \tag{5.1}$$

Because we only use Finite-Horizon MPC without terminal cost, there may be two big problems here:

- we cannot guarantee feasibility: the MPC problem may not have a solution now or later
- we cannot guarantee stability: the trajectory may not converge to the origin

5.2.2 Main Idea

Introduce **terminal cost** and **constraints** to explicitly ensure stability and feasibility.

Model 5.2.2 (MPC with Terminal Cost). *MPC with Terminal Cost*

$$\begin{aligned} J^*(x) = \min_{x,u} \sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i + x_N^T P x_N \ \textit{Terminal cost} \\ s.t. \quad x_{i+1} = Ax_i + Bu_i \\ Cx_i + Du_i \leq b \\ x_N \in \mathcal{X}_f \ \textit{Terminal constraint} \\ x_0 = x \end{aligned} \tag{5.2}$$

5.2.3 Zero Terminal State Case

Requirements

Terminal constraint $X_N = 0$

5.2.4 Property

By doing:

- Assume feasibility of x and let $\left[u_0^*, u_1^*, \ldots, u_{N-1}^*\right]$ be the optimal control sequence computed at x
- At $x^+$the control sequence $\left[u_1^*, u_2^*, \ldots, u_{N-1}^*, 0\right]$ is feasible (apply 0 control input $\Rightarrow x_{N+} = 0$)

Then the controller meet **recursive feasibility** and $J^*(x)$ is a Lyapunov function.

$$\begin{aligned} J^*(x_0) &= \sum_{i=0}^{N-1} I(x_i^*, u_i^*) \\ J^*(x_1) \leq J(x_1) &= \sum_{i=1}^{N} I(x_i^*, u_i*) \\ &= \sum_{i=0}^{N-1} I(x_i^*, u_i^*) - I(x_0, u_0^*) + I(x_N, u_N) \\ &= J^*(x_0) - \underbrace{I(x, u_0^*)}_{\text{Subtract cost}} + \underbrace{I(0,0)}_{\text{at stage 0}} \text{ staying at 0} \end{aligned} \tag{5.3}$$

5.2.5 More General Terminal State Case

The terminal constraint $X_N = 0$ reduce the feasible set. We can somehow loose the constraints. We can prove that if the following assumptions hold:

1. the stage cost is a positive definite function, i.e. it is strictly positive and only zero at the origin
2. The terminal set is invariant under the local control law $\kappa_f(x)$:

$$x^+ = Ax + B\kappa_f(x) \in \mathcal{X}_f \quad \text{for all } x \in \mathcal{X}_f$$

All state and input constraints are satisfied in $\mathcal{X}_f$: terminal state

$$\mathcal{X}_f \subseteq \mathbb{X}, \kappa_f(x) \in \mathbb{U} \text{ for all } x \in \mathcal{X}_f$$

, i.e. we can find a "virtual terminal controller"

3. Terminal cost is a continuous Lyapunov function in the terminal set $\mathcal{X}_f$:

$$V_f\left(x^+\right) - V_f(x) \leq -l\left(x, \kappa_f(x)\right) \text{ for all } x \in \mathcal{X}_f$$

Then we have the following theorem:

Theorem 5.2.3 (MPC with established terminal set)**.** *The closed-loop system under the MPC control law $u_0^*(x)$ is stable and the system $x^+ = Ax + Bu_0^*(x)$ is invariant in the feasible set $\mathbb{X}_N$.*

Notes: Although better than zero terminal constraints, this method still reduces the region of attraction.

5.2.6 Extension to Nonlinear MPC

Model 5.2.4 (Nonlinear MPC Problem)**.** *Nonlinear MPC Problem*

$$\begin{aligned} J^*(x) = \min_{x,u} \sum_{i=0}^{N-1} I\left(x_i, u_i\right) \quad & +V_f\left(x_N\right) \\ s.t.\ \ x_{i+1} \quad & = f\left(x_i, u_i\right) \\ g\left(x_i, u_i\right) \quad & \leq 0 \\ x_N \quad & \in \mathcal{X}_f \\ x_0 \quad & = x \end{aligned} \tag{5.4}$$

Notes: Previous assumptions on the terminal set and cost did not rely on linearity, so the result can be directly extended to nonlinear systems. However, computing the sets $\mathcal{X}_{\{}$ and V_f may be very difficult in a nonlinear system.

5.3 Summary

Finite-Horizon MPC cannot guarantee feasibility and stability because the finite-horizon. We can solve this problem by introducing terminal set and terminal cost. However, then a big problem is how to guarantee stability and feasibility with terminal set and terminal cost.

We have shown that for zero terminal set, it will be stable and feasible. And we can prove that more generally, if we can build a final virtual controller and final cost meet 3 assumptions, we will get a feasible and stable MPC controller.

The next problem is, how can we find the virtual controller and build a final cost meet the assumptions.

Chapter 6

Lecture 6: Practical MPC

6.1 How to Choose Terminal Set

From last chapter, we knows that in order to prevent finite-horizon MPC from limited horizon, we need a terminal set.

6.1.1 Infinite LQR Controller Solution

How to choose terminal set can be difficult in general, but one case is constructive: that is, suppose a **final infinite LQR controller**

$$\begin{gathered} f(x,u) = Ax + Bu \quad \mathbb{X} \text{ and } \mathbb{U} \text{ polytopes} \quad l(x,u) = x^T Q x + u^T R u \\ \kappa_f(x) = Kx \quad K = -\left(R + B^T P B\right)^{-1} B^T P A \\ P = Q + A^T P A - A^T P B \left(R + B^T P B\right)^{-1} B^T P A \\ V_f(x) := x^T P x = \sum_{i=0}^{\infty} x_i^T Q x_i + x_i^T K^T R K x_i \end{gathered} \tag{6.1}$$

By using a LQR controller, we can then define terminal invariant set: choose the terminal set $\mathcal{X}_f$ to be the maximum invariant set for the closed-loop system $x^+ = (A + BK)x$ subject to $\mathcal{X}_f \subset \mathbb{X}, K\mathcal{X}_f \subset \mathbb{U}$.

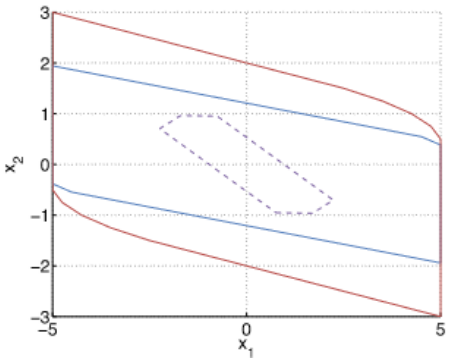
6.1.2 An Interesting Phenomenon

If the stage cost function outside terminal set has the same format as the cost function in terminal set, during the receding horizon process of an MPC, once in one optimization, we found the final step state is not at the edge of the terminal set, then the following optimization will keep on the same trajectory.

6.2 Practical Problem 1: Enlarging the Feasible Set

A significant problem take by terminal set is the terminal set reduces the feasible set. One example is shown in Figure 6.1.

Tow methods can be used to solve this problem: **Remove the terminal set** or **Soft Constrained MPC**.



- Blue line: Feasible set with terminal constraint
- Red line: Feasible set without terminal constraint
- Dashed line: Terminal set

Figure 6.1: Terminal Set reduces the Feasible Set

6.2.1 MPC without Terminal Set

We can remove terminal constraints while maintaining stability if

- initial state lie in sufficiently small subset of feasible set
- N is sufficiently large

such that terminal state satisfies terminal constraint without enforcing it in the optimization. Solution of the finite horizon MPC problem corresponds to the infinite horizon solution.

6.2.2 Soft Constrained MPC

Motivation

1. State Constraints may lead to infeasibility
2. Controller must provide some input in every circumstances
3. Input constraints are always mandatory and state constraints can always be temporarily violated

Objectives

1. Minimize the duration of violation
2. Minimize the size of violation

Model: Self-Constrained Model

We can relax constraints by introducing so-called **slack variables** $\epsilon_i \in \mathcal{R}^p$. And then penalize the amount of constraint violation in the cost by means of penalty $p(\epsilon)$.

$$\begin{aligned} & \min_u \textstyle\sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i + \rho(\epsilon_i) + x_N^T P x_N + \rho(\epsilon_N) \\ \text{s.t.} \quad & x_{i+1} = A x_i + B u_i \\ & H_x x_i \leq k_x + \epsilon_i, \\ & H_u u_i \leq k_u, \\ & \epsilon_i \geq 0 \end{aligned} \tag{6.2}$$

Another question is how to choose penalty? There are two main trends:

- Quadratic penalty: $\rho(\epsilon_i) = \epsilon_i^T S \epsilon_i$
- Quadratic and linear norm penalty $\rho(\epsilon_i) = \epsilon_i^T S \epsilon_i + s \|\epsilon_i\|_{1/\infty}$

And some experience are as follows:

- For Quadratic Penalty: Increase in S leads to reduced size of violation but longer duration
- If weight s is large enough, constraints are satisfied if possible
- Increasing s results in increasing peak violations and decreasing duration

Notes: Standard methods for soft constraints MPC do not provide a stability guarantee for open-loop unstable systems

Model: Separation of Objectives

We can also separate the objectives into two steps.

First, we try to minimize the violation over the horizon.

$$\begin{aligned} \epsilon^{\min} = \text{argmin}_{\text{u},\epsilon} & \epsilon_i^T S \epsilon_i + s^T \epsilon_i \\ \text{s.t. } & x_{i+1} = Ax_i + Bu_i \\ & H_x x_i \le K_x + \epsilon_i \\ & H_u u_i \le K_u \\ & \epsilon_i \ge 0 \end{aligned} \tag{6.3}$$

Then after get the ϵ_i^{min}, we optimize the controller performance:

$$\begin{aligned} \min_u & \sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i + x_N^T P x_N \\ \text{s.t. } & x_{i+1} = Ax_i + Bu_i \\ & H_x x_i \le k_x + \epsilon_i^{\min} \\ & H_u u_i \le k_u \end{aligned} \tag{6.4}$$

6.3 Tracking Problem

In tracking problem, we try to track given reference r such that $y_k \to r$ as $k \to \infty$

We can also use tracking problem method in tracking piecewise constant reference, but the change of reference can render the optimization problem infeasible.

6.3.1 Model

Target Condition

$$\begin{aligned} x_s &= Ax_s + Bu_s \\ Cx_s &= r \end{aligned} \Rightarrow \underbrace{\begin{bmatrix} 1-A & -B \\ C & 0 \end{bmatrix}}_{(n_x+n_y)\times(n_x+n_u)} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ r \end{bmatrix} \tag{6.5}$$

When implementing, we can solve this problem by solving an optimization problem.

- If target problem is feasible:

$$\begin{aligned} &\min u_s^T R_s u_s \\ &\text{s.t.} \begin{bmatrix} I-A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ r \end{bmatrix} \\ &H_x x_s \le k_x \\ &H_u u_s \le k_u \end{aligned} \tag{6.6}$$

- If target problem in infeasible:

$$\begin{aligned} \min \quad & (Cx_s - r)^T Q_s (Cx_s - r) \\ \text{s.t.} \quad & x_s = Ax_s + Bu_s \\ & H_x x_s \le k_x \\ & H_u u_s \le k_u \end{aligned} \tag{6.7}$$

$$\begin{aligned} \Delta x = x - x_s \\ \Delta u = u - u_s \end{aligned} \Rightarrow \begin{aligned} \Delta x_{k+1} &= x_{k+1} - x_s \\ &= Ax_k + Bu_k - (Ax_s + Bu_s) \\ &= A\Delta x_k + B\Delta u_k \end{aligned} \tag{6.8}$$

Delta Formation

After get the suitable target, we can generate a MPC model problem in Delta Formation

$$\begin{aligned} \min \quad & \textstyle\sum_{i=0}^{N-1} \Delta x_i^T Q \Delta x_i + \Delta u_i^T R \Delta u_i + V_f(\Delta x_N) \\ \text{s.t.} \quad & \Delta x_0 = \Delta x \\ & \Delta x_{i+1} = A\Delta x_i + B\Delta u_i \\ & H_x \Delta x_i \le k_x - H_x x_s \\ & H_u \Delta u_i \le k_u - H_u u_s \\ & \Delta x_N \in \mathcal{X}_f \end{aligned} \tag{6.9}$$

The control framework is shown as 6.2. The control framework then is divided into two steps:

- Find optimal sequence of Δu^*
- Input applied to the system $u_0^* = \Delta u_0^* + u_s$

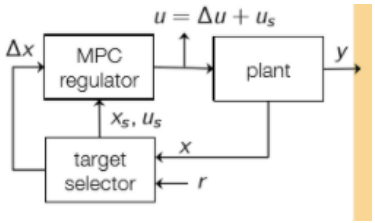


Figure 6.2: Tracking Problem

6.3.2 Convergence Analysis

Assume target is feasible with $x_s \in \mathbb{X}, u_s \in \mathbb{U}$ and choose terminal weight $V_f(x)$ and constraint $\mathcal{X}_f$ as in the regulation case satisfying:

- $\mathcal{X}_f \subseteq \mathbb{X}, Kx \in \mathbb{U}$ for all $x \in \mathcal{X}_f$
- $V_f(x^+) - V_f(x) \leq -I(x, Kx)$ for all $x \in \mathcal{X}_f$

If in addition the target reference x_s, u_s is such that

- $x_s \oplus \mathcal{X}_f \subseteq \mathbb{X}, K\Delta x + u_s \in \mathbb{U}$ for all $\Delta x \in \mathcal{X}_f$

then the closed-loop system converges to the target reference, i.e. $x_k \to x_s$ and therefore $y_k = Cx_k \to r$ for $k \to \infty$

6.3.3 Terminal Set Problem

Based on previous convergence analysis, because of shifted terminal set, if we want to guarantee convergence, the set of feasible targets is really small. Which is shown in Figure 6.3.

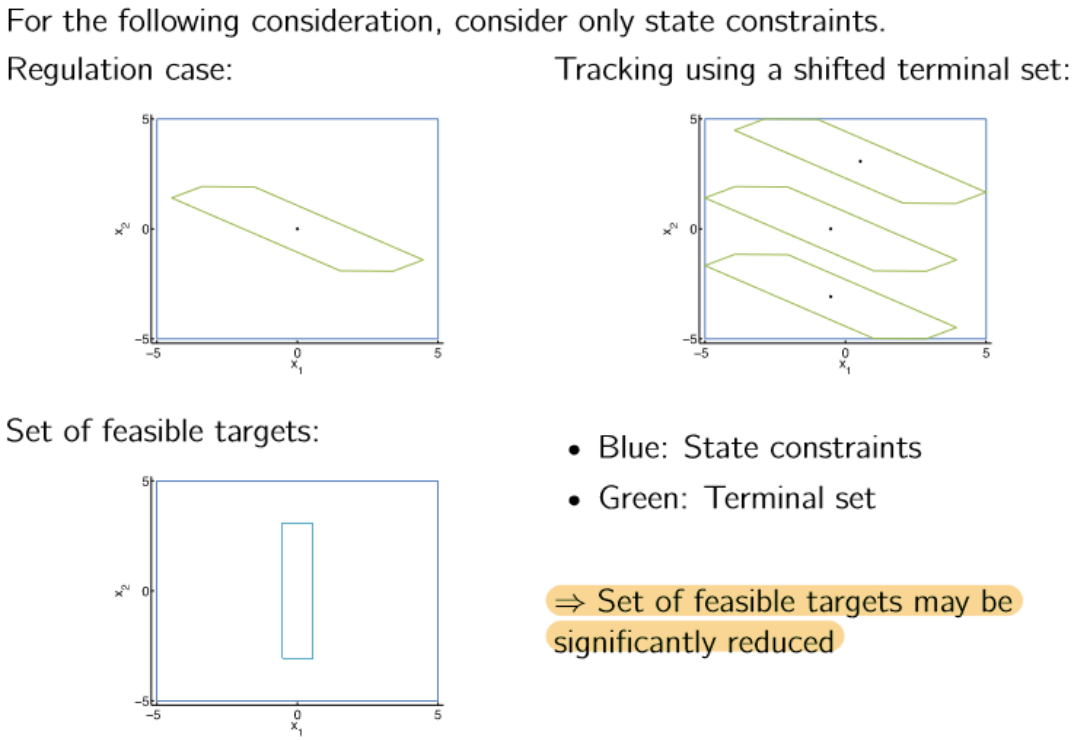


Figure 6.3: Track Terminal Set Problem

A possible solution is enlarging set of feasible targets by scaling terminal set for regulation. The idea is squeeze the terminal set when the setting point get close to the boundary of the state constraints. This method is illustrated in Figure 6.4.

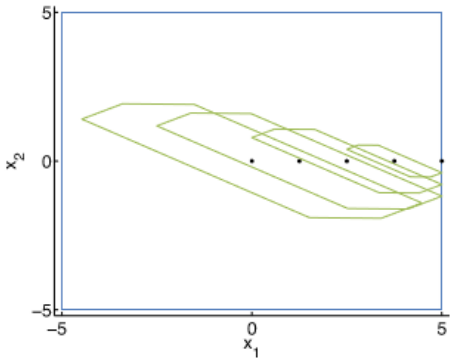
6.4 Offset Free Control (with Constant Disturbance)

If a system is in the presence of a disturbance, then we want it converges to set-point with zero offset.

6.4.1 Basic Idea and Augmented Model

In order to finish that, our basic idea is:

- Model the disturbance



- Scale terminal set by scaling factor α, i.e. $\mathcal{X}_f^{\text{scaled}} = \alpha\mathcal{X}_f$
- Invariance is maintained: If $\mathcal{X}_f$ is invariant, then also $\alpha\mathcal{X}_f$
- Choose scaling factor α such that state and input constraints are still satisfied

$\rightarrow$ Scaling is dependent on target

Figure 6.4: Scaling Terminal Set

- Use the output measurements and model to estimate the state and the disturbance
- Find control inputs that use the disturbance estimate to remove offset

We can then build an augmented model for the system

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k + B_d d_k \\ d_{k+1} &= d_k \\ y_k &= Cx_k + C_d d_k \end{aligned} \tag{6.10}$$

6.4.2 Linear State Estimation

For estimating state and disturbances, we can use a linear estimator to estimate. The model is as following:

$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{d}_{k+1} \end{bmatrix} = \begin{bmatrix} A & B_d \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_k \\ \hat{d}_k \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k + \begin{bmatrix} L_x \\ L_d \end{bmatrix} \left(C\hat{x}_k + C_d\hat{d}_k - y_k \right) \tag{6.11}$$

Then the error dynamic will become:

$$\begin{aligned} \begin{bmatrix} x_{k+1} - \hat{x}_{k+1} \\ d_{k+1} - \hat{d}_{k+1} \end{bmatrix} &= \begin{bmatrix} A & B_d \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ d_k \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k \\ &\quad - \begin{bmatrix} A & B_d \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_k \\ \hat{d}_k \end{bmatrix} - \begin{bmatrix} B \\ 0 \end{bmatrix} u_k - \begin{bmatrix} L_x \\ L_d \end{bmatrix} \left(C\hat{x}_k + C_d\hat{d}_k - Cx_k - C_d d_k \right) \\ &= \left(\begin{bmatrix} A & B_d \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} L_x \\ L_d \end{bmatrix} \begin{bmatrix} C & C_d \end{bmatrix} \right) \begin{bmatrix} x_k - \hat{x}_k \\ d_k - \hat{d}_k \end{bmatrix} \end{aligned} \tag{6.12}$$

Which means we can let the estimator stable by choosing appropriate $L = \begin{bmatrix} L_x \\ L_d \end{bmatrix}$, i.e. choosing L let error dynamics become stable and converges to zero. This can be done by a lot of methods, for example, pole configuration.

6.4.3 Deal With Noise

If instead of constant disturbance, we have some non-constant noise:

$$\begin{aligned}\begin{bmatrix} x_{k+1} \\ d_{k+1} \end{bmatrix} &= \begin{bmatrix} A & B_d \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ d_k \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k + w \\ y_k &= Cx_k + C_d d_k + v \end{aligned} \tag{6.13}$$

where w, v are white noise with covariance matrices Q_e R_e respectively.

There are mainly two ways to be used

- use Kalman filter to estimate both the state and the intergrating disturbance
- optimal estimator gain L that minimizes the variance of the estimation error.

6.5 OffSet Free Tracking

6.5.1 Model

The basic model of off-set free tracking problem is

$$\begin{aligned} x_s &= Ax_s + Bu_s + B_d d_s \\ y_s &= Cx_s + C_d d_s = r \end{aligned} \tag{6.14}$$

So, now we should set our target condition according to the disturbance.

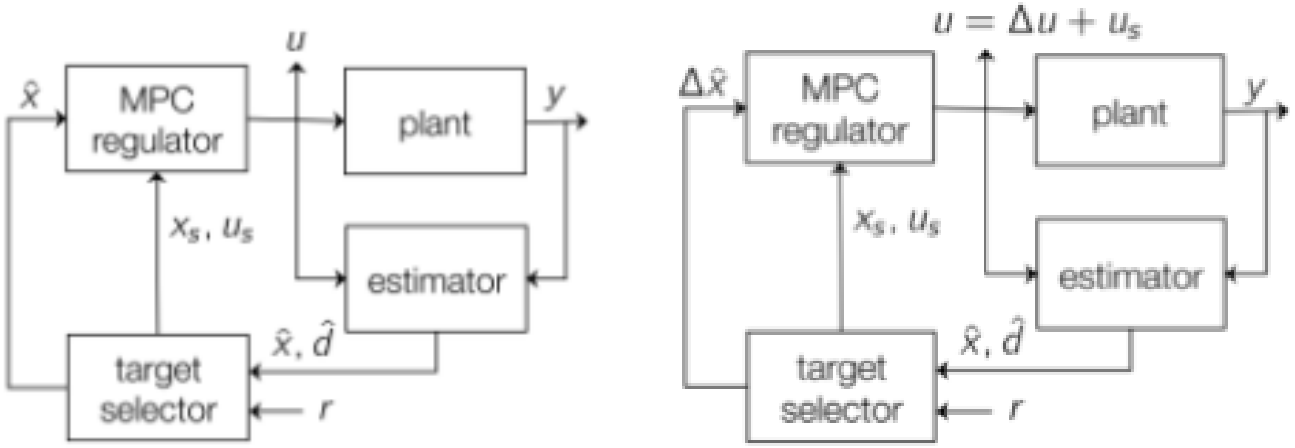
$$\begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} B_d \hat{d} \\ r - C_d \hat{d} \end{bmatrix} \tag{6.15}$$

Normal Formation

$$\begin{aligned} \min \quad & \sum_{i=0}^{N-1} (x_i - x_s)^T Q (x_i - x_s) + (u_i - u_s)^T R (u_i - u_s) + V_f (x_N - x_s) \\ s.t. \quad & x_0 = \hat{x} \\ & d_i = \hat{d} \\ & x_{i+1} = Ax_i + Bu_i + d_i \\ & H_x x_i \leq k_x \\ & H_u u_i \leq k_u \\ & x_N - x_s \in \mathcal{X}_f \end{aligned} \tag{6.16}$$

The whole process is: at each sampling time

1. Estimate state and disturbance $\hat{x}$, $\hat{d}$
2. Obtain (x_s, u_s) from steady-state target problem using disturbance estimate
3. Solve MPC problem for tracking using disturbance estimate $\hat{d}$



(a) off set free tracking normal formation

(b) off set free tracking delta formation

Figure 6.5: off set free tracking

Delta Formation

$$\begin{aligned} \min \quad & \textstyle\sum_{i=0}^{N-1} \Delta x_i^T Q \Delta x_i + \Delta u_i^T R \Delta u_i + V_f\left(\Delta x_N\right) \\ \text{s.t.} \quad & \Delta x_0 = \Delta \hat{x} \\ & \Delta x_{i+1} = A\Delta x_i + B\Delta u_i \\ & H_x \Delta x_i \leq k_x - H_x x_s \\ & H_u \Delta u_i \leq k_u - H_u u_s \\ & \Delta x_N \in \mathcal{X}_f \end{aligned} \tag{6.17}$$

The whole process is: at each sampling time

1. Estimate state and disturbance $\hat{x}$, $\hat{d}$
2. Obtain (x_s, u_s) from steady-state target problem using disturbance estimate
3. initial state $\Delta\hat{x} = \hat{x} - x_s$
4. Solve MPC problem for tracking in Delta Formation.

The structure of these two methods are shown in Figure 6.5.

6.5.2 Convergence Analysis

Assumption

- Consider the case $n_d = n_y$, i.e. number of disturbance states equal to number of measured outputs.
- Assume target steady-state problem is feasible and constraints are not active at steady-state.

Convergence

If closed-loop system converges to $\hat{x}_s, \hat{d}_s, y_s$, i.e. $\hat{x}_k \to \hat{x}_s, \hat{d}_k \to \hat{d}_s, y_k \to y_s$ as $k \to \infty$, then

$$y_k = Cx_k \to r \text{ for } k \to \infty$$

Summary

In this chapter, we first introduce a common method to determine the terminal controller and terminal set, that is use a "virtual LQR controller". Then we proposed that the terminal set may shrink the feasible set and we proposed two possible solution to solve this problem: 1. MPC without constraints and 2. MPC with soft constrained.

Then we introduce the tracking problem, in tracking problem, we first compute a steady state based on reference. Then we introduce how to due with constant disturbance in control, that is add a stable observer to estimate both the state and the disturbance. We call this problem the offset-free control problem. Finally we combine offset-free control and tracking problem and introduce how to solve offset-free tracking problem.

Chapter 7

Lecture 7: Robust MPC

7.1 Introduction to Robust Constrained Control

The real world: full of noise, parameters inaccurate, system structure not fully known.

Uncertain Constrained System Model

$$x^+ = f(x, u, w; \theta) \quad (x, u) \in \mathbb{X}, \mathbb{U} \quad w \in \mathbb{W} \quad \theta \in \Theta \tag{7.1}$$

7.1.1 Different Uncertainty Model

Measurement / Input Bias Model

$$g(x, u, w; \theta) = f(x, u) + \theta$$

θ unknown, but constant

Linear Parameter Varying System

$$g(x, u, w; \theta) = \sum_{k=0}^{l} \theta_k A_k x + \sum_{k=0}^{\imath} \theta_k B_k u, \quad 1^\top \theta = 1, \theta \geq 0$$

A_k, B_k known, θ_k unknown, but constant

Polytopic Uncertainty

$$g(x, u, w; \theta) = \sum_{k=0}^{l} w_k A_k x + \sum_{k=0} w_k B_k u, \quad \mathbf{1}^\top w = 1, w \geq 0$$

A_k, B_k known, w_k unknown and changing at each sample time

Additive Stochastic Noise

$$g(x, u, w; \theta) = Ax + Bu + w$$

Distribution of w known

In this Chapter, we will mainly focus on **Additive Stochastic Noise** Model.

7.2 Impact of Bounded Additive Noise

7.2.1 Model of Bounded Additive Noise

Model 7.2.1 (Model of Bounded Additive Noise)**.** *Model of Bounded Additive Noise:*

$$g(x, u, w; \theta) = Ax + Bu + w, \quad w \in \mathbb{W}$$

A, B known, w unknown and changing with each sample

1. Dynamics are linear, but impacted by **random, bounded noise at each time step**
2. It is always a **conservative model**
3. The noise is **persistent**, i.e. it does not converge to zero in the limit

So the basic motivation is to design a control lay that will satisfy constraints and **stabilize the system for all possible disturbances**.

7.2.2 Uncertain State Evolution

Define $\phi_i(x_0, \mathbf{u}, \mathbf{w})$ as the state that the system will be in at time i if the state at time zero is x_0, we apply the input $\mathbf{u} := \{u_0 \ldots., u_{N-1}\}$ and we observe the disturbance $\mathbf{w} := \{w_0, \ldots, w_{N-1}\}$.

Then we can compare the nominal system and uncertain system:

For Nominal System: $x^+ = Ax + Bu$, so we will have:

$$\begin{aligned} x_1 &= Ax_0 + Bu_0 \\ x_2 &= A^2x_0 + ABu_0 + Bu_1 \\ &\vdots \\ x_i &= A^ix_0 + \sum_{k=0}^{i-1} A^kBu_{i-k} \end{aligned} \tag{7.2}$$

For Uncertain System: $x^+ = Ax + Bu + w \quad w \in \mathcal{W}$, we will have:

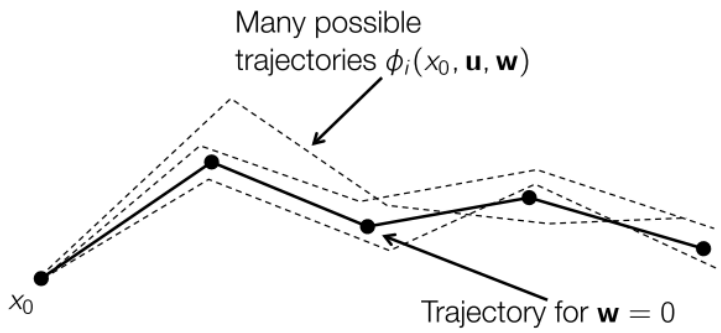
$$\begin{aligned} \phi_1 &= Ax_0 + Bu_0 + w_0 \\ \phi_2 &= A^2x_0 + ABu_0 + Bu_1 + Aw_0 + w_1 \\ &\vdots \\ \phi_i &= A^ix_0 + \sum_{k=0}^{i-1} A^kBu_{i-k} + \sum_{k=0}^{i-1} A^kw_{i-k} \\ \phi_i &= x_i + \sum_{k=0}^{i-1} A^kw_{i-k} \end{aligned} \tag{7.3}$$

Notes: Uncertain evolution is the nominal system + offset caused by the disturbance

The process is illustrated in Figure 7.1.

7.2.3 Cost Function Definitions

Another problem is how to define cost function in uncertain system. The cost is now a function of the disturbance, and therefore each possible trajectory has a different cost:



Define $\phi_i(x_0, \mathbf{u}, \mathbf{w})$ as the state that the system will be in at time i if the state at time zero is x_0, we apply the input $\mathbf{u} := \{u_0, \ldots, u_{N-1}\}$ and we observe the disturbance $\mathbf{w} := \{w_0, \ldots, w_{N-1}\}$.

Figure 7.1: Uncertain State Evolution

$$J(x_0, \mathbf{u}, \mathbf{w}) := \sum_{i=0}^{N-1} I(\phi_i(x_0, \mathbf{u}, \mathbf{w}), u_i) + V_f(\phi_N(x_0, \mathbf{u}, \mathbf{w})) \tag{7.4}$$

Minimize the Expected Value

$$V_N(x_0, \mathbf{u}) := \mathbf{E}[J(x_0, \mathbf{u}, \mathbf{w})] \tag{7.5}$$

Minimize the Variance

$$V_N(x_0, \mathbf{u}) := \mathrm{Var}(J(x_0, \mathbf{u}, \mathbf{w})) \tag{7.6}$$

Take the Worst-Case

$$V_N(x_0, \mathbf{u}) := \max_{\mathbf{w} \in \boldsymbol{W}^{N-1}} J(x_0, \mathbf{u}, \mathbf{w}) \tag{7.7}$$

Take the Nominal-Case

$$V_N(x_0, \mathbf{u}) := J(x_0, \mathbf{u}, 0) \tag{7.8}$$

7.2.4 Robust Invariant Set

Robust Positive Invariant Set and Robust Pre Set

Definition 7.2.2 (Robust Positive Invariant Set). *Robust Positive Invariant Set*

A set $\mathcal{O}^W$ is said to be a robust positive invariant set for the autonomous system $x_{i+1} = f(x_i, w)$ if

$$x \in \mathcal{O}^W \Rightarrow f(x, w) \in \mathcal{O}^W, \text{ for all } w \in \mathbb{W}$$

Definition 7.2.3 (Robust Pre Set). *Robust Pre Set:*

*Given a set Ω and the dynamic system $x^+ = f(x, w)$, the pre-set of Ω is the set of states that evolve into the target set Ω in one time step **for all** values of the disturbance $w \in \mathbb{W}$:*

$\mathrm{pre}^W(\Omega) := \{x \mid f(x, w) \in \Omega \text{ for all } w \in \mathbb{W}\}$

These two definitions are illustrated in Figure 7.2.

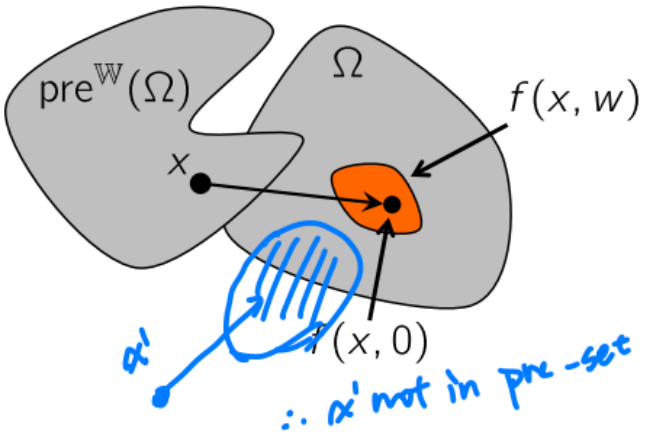


Figure 7.2: Robust Positive Invariant Set and Robust Pre Set

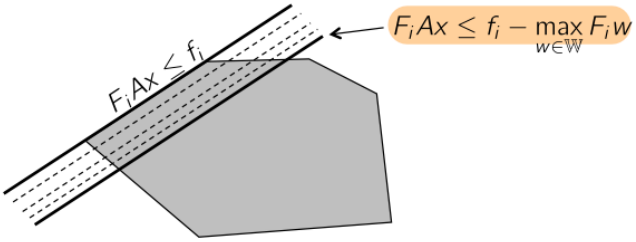
Geometric Condition for Robust Invariance

Theorem 7.2.4 (Geometric Condition for Robust Invariance)**.** *Geometric Condition for Robust Invariance: A set $\mathcal{O}$ is a robust positive invariant set if and only if*

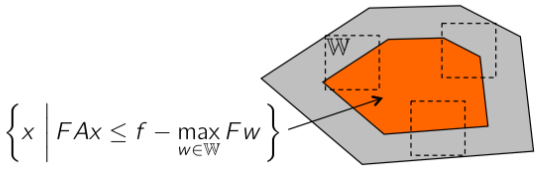
$$\mathcal{O} \subseteq \text{pre}^{\text{W}}(\mathcal{O})$$

7.2.5 Computation of Robust Pre Set

The basic idea of compute the Robust Pre Set is shown in Figure 7.3.



(a) Computation of Pre Set 1



(b) Computation of Pre Set 2

Figure 7.3: Computation of Pre Set

The robust pre set then can be computed by:

$$\text{pre}^{\text{W}}(\Omega) = \left\{ x \mid FAx \leq f - \max_{w \in W} F_W \right\} = \{ x \mid FAx \leq f - h_W(F) \} = A(\Omega \ominus \mathbb{W}) \tag{7.9}$$

One thing to be noticed is that the support function actually does not depend on state. So actually we can pre-compute if offline.

Computation of Robust Invariant Set

The computation process of the robust invariant set is shown in Figure 7.4.

Conceptual Algorithm to Compute Robust Invariant Set

Input: f, $\mathbb{X}$, $\mathbb{W}$
Output: $\mathcal{O}_\infty^{\mathbb{W}}$

$\Omega_0 \leftarrow \mathbb{X}$
loop
  $\Omega_{i+1} \leftarrow \text{pre}^{\mathbb{W}}(\Omega_i) \cap \Omega_i$
  if $\Omega_{i+1} = \Omega_i$ **then**
    return $\mathcal{O}_\infty = \Omega_i$
  end if
end loop

Figure 7.4: Computation of Robust Invariant Set

Figure 7.5 shows the relations between constraint set, maximum robust invariant set and maximum nominal invariant set.

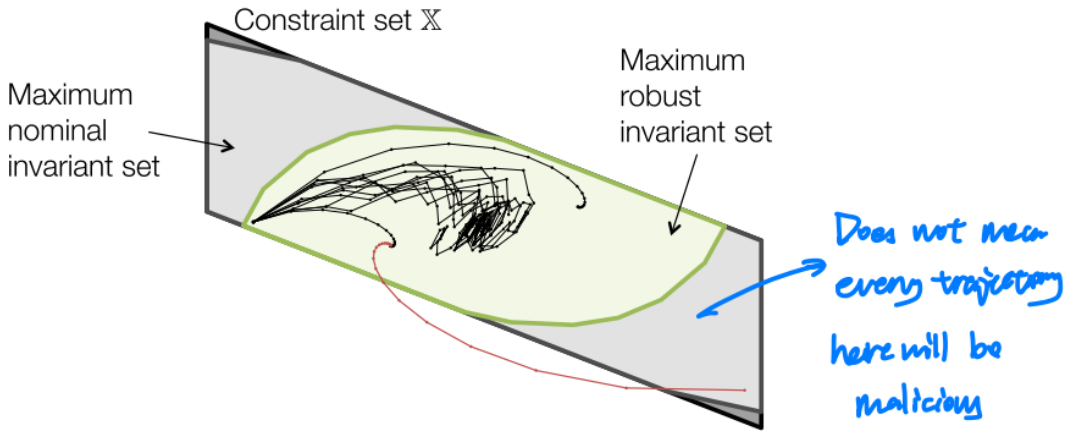


Figure 7.5: Computation of Robust Invariant Set 2

One important thing to be noticed is that because in the robust invariant set, we considered "for all possible disturbance", it is quite conservative. So, in Figure 7.5. A state in the gray area does not indicate that every trajectory here will be malicious.

7.2.6 From Robust Invariant Set to Dealing with Uncertainty

Motivation

Idea: Compute a set of tighter constraints such that if the nominal system meets these constraints, then the uncertain system will too. We then do MPC on the nominal system.

This motivation is illustrated in Figure 7.6.

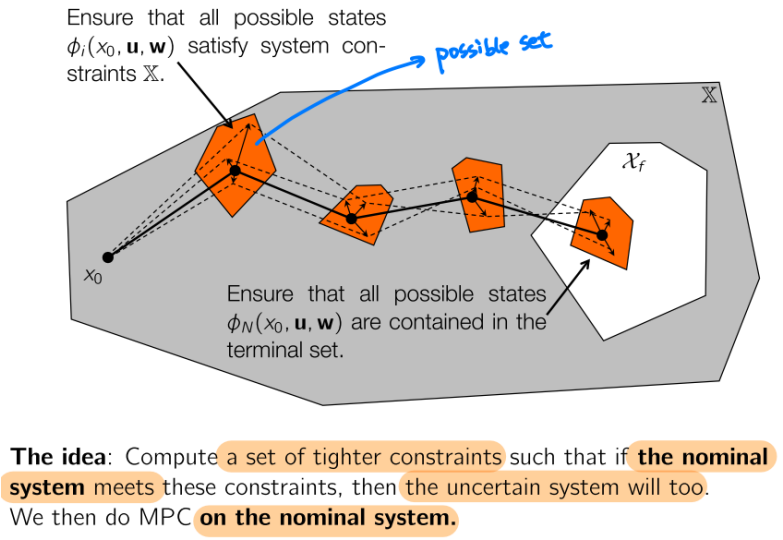


Figure 7.6: CDealing with Uncertainty

Robust Constraint Satisfication

So the first problem is what should we do on the constraint satisfication. Remember the uncertain evaluation:

$$\phi_i(x_0, \mathbf{u}, \mathbf{w}) = \left\{ x_i + \sum_{k=0}^{i-1} A^k w_k \mid \mathbf{w} \in \mathbb{W}^i \right\} \subseteq \mathbb{X}$$

Assume that $\mathbb{X} = \{x \mid Fx \leq f\}$, then this is equivalent to

$$Fx_i + F \sum_{k=0}^{i-1} A^k w_k \leq f \forall \mathbf{w} \in \mathbb{W}^i$$

We've seen this before while computing the robust pre-set:

$$Fx_i \leq f - \max_{w \in Wi} F \sum_{k=0}^{i-1} A^k w_k = f - h_{W,i}\left(F \sum_{k=0}^{i-1} A^k \right)$$

Definition 7.2.5 (Support Function)**.** *Support Function*

let $A \subset \mathbb{R}^n$ be a non-empty convex set. The support function of A $h_A : \mathbb{R}^n \to (-\infty, \infty]$ is defined as: $h_A(u) := \sup\{\langle x, u \rangle : x \in A\}, u \in \mathbb{R}^n$.

We always use $h(A, .) := h_A$ to represent the support function of A

After tightening the constraints on the nominal system, we can guarantee that the system will meet constraints even with noise.

Termianl State Constraints

Another problems is hot to deal with terminal state constraints:

$$\phi_N(x_0, \mathbf{u}, \mathbf{w}) \subseteq \mathcal{X}_f \tag{7.10}$$

Which can be done in a similar way.

7.3 Robust Open-Loop MPC

To be concluded, we can propose the **Robust Open-Loop MPC**.

Model 7.3.1 (Robust Open-Loop MPC)**.** *Robust Open-Loop MPC*

$$\begin{aligned} \min_u \sum_{i=0}^{N-1} \quad & l(x_i, u_i) + V_f(x_N) \\ s.t.\ x_{i+1} \quad & = Ax_i + Bu_i \\ x_i \quad & \in \mathbb{X} \ominus \mathcal{A}_i \mathbb{W}^i \\ u_i \quad & \in \mathbb{U} \\ x_N \quad & \in \mathcal{X}_f \ominus \mathcal{A}_N \mathbb{W}^N \end{aligned}$$

where $\mathcal{A}_i := \begin{bmatrix} A^0 & A^1 & \dots & A^i \end{bmatrix}$ *and* $\tilde{\mathcal{X}}_f$ *is a robust invariant set for the system* $x^+ = (A+BK)x$ *for some stabilizing* K*.*

That is we do nominal MPC, but with tighter constraints.

One important thing to be noticed that, in Robust Open-Loop MPC, we need the A to be a stable A. If not, the noisy part (the support function) will become larger and larger.

Property

If $\mathbf{u}^*(x)$ is the optimizer of the robust open-loop MPC problem, then the system $Ax + Bu_0^*(x) + w \in \mathbb{X}$ for all $w \in \mathbb{W}$.

However, robust open-loop MPC has a very small region of attraction. This is a reason that we almost never use it in reality.

7.4 Closed-Loop Prediction

7.4.1 MPC: A Game Perspective

7.4.2 Closed-Loop MPC

From the game perspective, when in MPC problem, what we want to do is to optimize over **a sequence of functions** $\{u_0, \mu_1(\cdot) \dots\dots, \mu_{N-1}(\cdot)\}$. where $\mu_i(x_i) : \mathbb{R}^n \to \mathbb{R}^m$ is called a **control policy**. and maps the state at time i to an input at time i.

Notes:

1. This is the same as making μ a **function** of the disturbances to time i, since the state is a function of the disturbances up to that point
2. The first input u_0 is a function of the current state, which is known. Therefore it is not a function, but a single value.

It is hard to search for a "optimal solution" in the whole space. So firstly, we need to assume the form of the function. There are several popular forms:

- **Pre-Stabilization**: $\mu_i(x) = Kx + v_i \to$
  - Fixed K, such that $A + BK$ is stable
  - Simple, often conservative
  - v_i is optimized online
- **Linear Feedback**: $\mu_i(x) = K_i x + v_i$

  - Optimize over K_i and v_i
  - Non-convex. Extremely difficult to solve...
- **Disturbance Feedback**:$\mu_i(x) = \sum_{j=0}^{i-1} M_{ij} w_j + v_i$
  - Optimize over M_{ij} and v_i
  - Equivalent to linear feedback, but convex!
  - Can be very effective, but computationally intense.
- **Tube-MPC**: $\mu_i(x) = v_i + K(\underbrace{x - \bar{x}_i}_{})$
  - Fixed K, such that $A + BK$ is stable
  - Optimize over $\bar{x}_i$ and v_i
  - Simple, and can be effective

7.5 Tube-MPC

Model amd Motivation

$$x^+ = Ax + Bu + w \quad (x, u) \in X \times \mathrm{U} \quad w \in W$$

The idea is to seperate the available control authority into two parts:

- A portion that steers the noise-free system to the origin $z^+ = Az + Bv$
- A portion that compensates for deviations from this system $e^+ = (A + BK)e + w$

7.5.1 System Decomposition

First, we define a **nominal noise-free subsystem**:

$$z_{i+1} = Az_i + Bv_i$$

The Kz_i part of the Tube-MPC controller is used to help the system stabilize this subsystem.

Then we can define a **error subsystem** $e_i = x_i - z_i$

$$\begin{aligned} e_{i+1} &= x_{i+1} - z_{i+1} \\ &= Ax_i + Bu_i + w_i - Az_i - Bv_i \\ &= Ax_i + BK\left(x_i - z_j\right) + Bv_i + w_i - Az_i - Bv_i \\ &= (A + BK)\left(x_i - z_i\right) + w_i \\ &= (A + BK)e_i + w_i \end{aligned} \tag{7.11}$$

Dynamics $A + BK$ are stable, and the set W is bounded, so there is some set $\mathcal{E}$ that e will stay inside for all time.

We know that the real trajectory stays 'nearby' the nominal one: $x_i \in z_i \oplus \mathcal{E}$ because we plan to apply the controller $u_i = K\left(x_j - z_j\right) + v_i$ in the future (we won't actually do this, but it's a valid sub-optimal plan)

Uncertain State Evolution

Consider the system $x^+ = Ax + w$ and assume that x_0=0

As sum goes to infinity, we arrive at the **minimum robust invariant set**

$$F_\infty = \bigoplus_{k=0}^{\infty} A^k W, \quad F_0 := \{0\}$$

If there exists an n such that $F_n = F_{n+1}$, then $F_n = F_\infty$

The minimal invariant set can be calculate following the progress in Figure 7.7.

Minimal Invariant Set

Input: A
Output: F_∞
  $\Omega_0 \leftarrow \{0\}$
  loop
    $\Omega_{i+1} \leftarrow \Omega_i \oplus A^i \mathbb{W}$
    if $\Omega_{i+1} = \Omega_i$ **then**
      return $F_\infty = \Omega_i$
    end if
  end loop

Figure 7.7: Computation of Minimum Invariant Set

Notes: A finite n does not always exist, but a 'large' n is a good approximation.

Method 7.5.1 (Computing Minkowski Sums). *Computing Minkowski Sums:*
Given $P := \{x \mid Tx \leq t\}$ and $Q := \{x \mid Rx \leq r\}$, the Minkowski sum is:

$$\begin{aligned} P \oplus Q &:= \{x + y \mid x \in P, y \in Q\} \\ &= \{z \mid \exists x, yz = x + y, Tx \leq t, Ry \leq r\} \\ &= \{z \mid \exists y \quad Tz - Ty \leq t, Ry \leq r\} \\ &= \left\{ z | \exists y | \begin{bmatrix} T & -T \\ 0 & R \end{bmatrix} \begin{pmatrix} z \\ y \end{pmatrix} \leq \begin{pmatrix} t \\ r \end{pmatrix} \right\} \end{aligned}$$

Notes: This is a **projection** of a polyhedron from (z, y) onto z

Constraint Tightening

Definition 7.5.2 (Pontryagin Difference). *Pontryagin Difference*
Let A and B be subsets of $\mathbb{R}^n$. The Pontryagin Difference is

$$A \in B := \{x \mid x + e \in A \forall e \in B\}$$

Then the state constraints and the input constraints can be calculated by:

$$\begin{aligned} z_i \oplus \mathcal{E} \subseteq \mathbb{X} \quad &\Leftarrow \quad z_i \in \mathbb{X} \ominus \mathcal{E} \\ u_i \in K\mathcal{E} \oplus v_i \subset \mathbb{U} \quad &\Leftarrow \quad v_i \in \mathbb{U} \ominus K\mathcal{E} \end{aligned} \tag{7.12}$$

And the terminal constraint can be calculated by:

$$\text{pre}\,(\mathcal{X}_f) \subseteq \mathcal{X}_f \text{ and } x_f \subseteq \mathbb{X} \in \mathcal{E} \text{ and } K\mathcal{X}_f \subseteq \mathbb{U} \in K\mathcal{E} \tag{7.13}$$

7.5.2 Tube-MPC Problem Formulation

$$\text{Feasible set: } \mathcal{Z}(x_0) := \left\{ \begin{array}{c} \left\{ \begin{array}{cc} z_{i+1} = Az_i + Bv_i & i \in [0, N-1] \\ z_i \in X \oplus \mathcal{E} & i \in [0, N-1] \\ v_i \in U \oplus K\mathcal{E} & i \in [0, N-1] \\ z_N \in \mathcal{X}_f & x_0 \in z_0 \oplus \mathcal{E} \end{array} \right\} \\ \text{Cost function: } \quad V(\mathbf{z}, \mathbf{v}) := \sum_{i=0}^{N-1} I(z_i, v_i) + V_f(z_N) \end{array} \right\} \tag{7.14}$$

$$\text{Optimization problem: } (\mathbf{v}^*(x_0), \mathbf{z}^*(x_0)) = \text{argmin}_{v,z} \{V(\mathbf{z}, \mathbf{v}) \mid (\mathbf{z}, \mathbf{v}) \in \mathcal{Z}(x_0)\}$$

$$\text{Control law; } \mu_{\text{tube}}(x) := K\left(x - z_0^+(x)\right) + v_0^*(x)$$

Stability Analysis

First we assume:

1. The stage cost is a positive definite function.
2. The terminal set is invariant for the nominal system under the local control law $K_f(z) : z^+ = Az + B\kappa_f(z) \in X_f \quad$ for all $z \in X_f$ All tightened state and input constraints are satisfied in $\mathcal{X}_f : \mathcal{X}_f \subseteq X \ominus \mathcal{E}, \kappa_f(z) \in \text{U} \in \mathcal{E}$ for all $z \in \mathcal{X}_f$
3. Terminal cost is a continuous Lyapunov function in the terminal set $\mathcal{X}_f : V_f(Az + B\kappa_f(z)) - V_f(z) \leq -l(z, \kappa_f(z))$ for all $z \in \mathcal{X}_f$

The we have th following theorem

Theorem 7.5.3 (Robust Stability of Tube-MPC)**.** *Robust Stability of Tube-MPC: The state x of the system $x^+ = Ax + B\mu_{tube}(x) + w$ converges in the limit to the set $\mathcal{E}$.*

That is the state x does not converge to zero.

The Overal Procedure of Tube-MPC

We can do the following thing **offline**:

1. Choose a stabilizing controller K so that $\|A + BK\| < 1$
2. Compute the minimal robust invariant set $\mathcal{E} = F_\infty$ for the system $x^+ = (A + BK)x + w, w \in W^1$
3. Compute the tightened constraints $\tilde{X} := X \ominus \mathcal{E}$, UU $: = U \ominus \mathcal{E}$
4. Choose terminal weight function V_f and constraint $\mathcal{X}_f$ satisfying assumptions

And the following thing can be done **online**:

1. Measure / estimate state x
2. Solve the problem $(\mathbf{v}^*(x), \mathbf{z}^*(x)) = \text{argmin}_{v,z}\{v(\mathbf{z}, v) \mid (\mathbf{z}, \mathbf{v}) \in \mathcal{Z}(x)\}$
3. Set the input to $u = K(x - z_0^*(x)) + v_0^*(x)$

7.6 Nominal MPC with noise

In this section, we will discuss what will happen if we just ignore the noise and design controller without considering the noise.

One important fact is: For some states, the system will sometimes work fine.

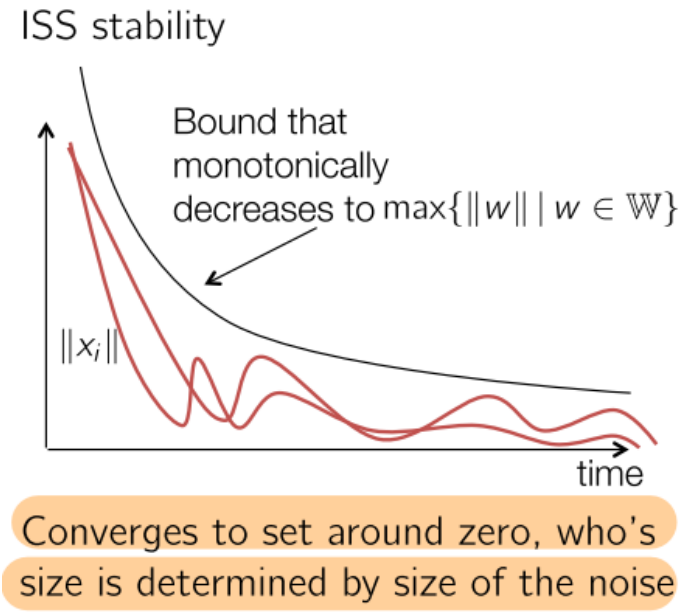


Figure 7.8: ISS Stability

7.6.1 Stability

Lyapunov Stability Analysis

Assume the optimal cost V^* is continuous

$$\begin{aligned} &\mid V^*\left(Ax+Bu^*(x)+w\right)-V^*\left(Ax+Bu^*(x)\right)\mid \\ &\qquad\qquad \leq \gamma\left\|Ax+Bu^*(x)+w-\left(Ax+Bu^\star(x)\right)\right\|=\gamma\|w\| \end{aligned} \tag{7.15}$$

Then the Lyapunov decrease con be calculated:

$$\begin{aligned} &V^*\left(Ax+Bu^*(x)+w\right)-V^*(x) \\ &=V^\star\left(Ax+Bu^\star(x)+w\right)-V^\star(x)-V^\star\left(Ax+Bu^\star(x)\right)+V^\star\left(Ax+Bu^\star\right) \\ &\leq V^*\left(Ax+Bu^\star(x)\right)-V^*(x)+\gamma\|w\| \\ &\leq \underbrace{\frac{-l\left(x,u^\star(x)\right)}{\text{Decrease}}}+\underbrace{\gamma\|w\|}\ \text{Increase} \end{aligned} \tag{7.16}$$

That is:

- Amount of decrease grows with $||x||$
- Amount of increase is upper bounded by $\max\{\|w\| \mid w \in \mathbb{W}\}$

Therefore, the system will move towards the origin until there is a balance between the size of x and the size of w

Input-To-State Stability (ISS Stability)

Roughly speaking, a control system is **ISS** if it is globally asymptotically stable in the absence of external inputs and if its trajectories are bounded by a function of the size of the input for all sufficiently large times.

Based on the Lyapunov analysis, we knows that the system ignoring noise is ISS stable.

The Figure 7.8 illustrates the ISS stability.

7.6.2 Design of Nominal MPC

So in reality, we can just ignore the noise and hope it will work.

The analysis of Nominal MPC also tells as, for robust MPC, most of times we need to know the largest noise W, but even if we take do not know that, and we take W with **a smaller boundary assumption**, it will at least ISS.

7.7 Summary

Chapter 8

Lecture 8: Explicit MPC

8.1 Motivation

Traditional MPC cannot guarantee real-time. And a classical MPC problem has following properties:

- Optimization problem is function parameterized by state
- Control law piecewise affine for linear systems/constraints
- Pre-compute control law as function of state x

So we can build a **pre-compute control law**.

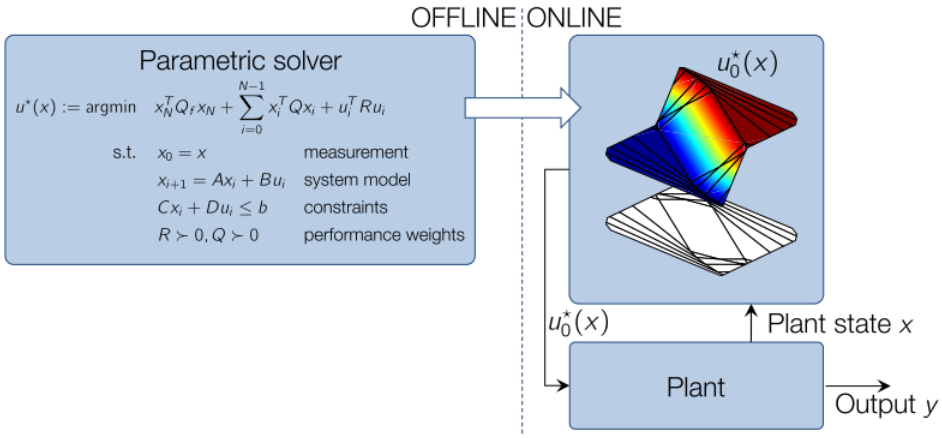


Figure 8.1: Motivation of Explicit MPC

8.2 MPC = Parametric Quadratic Programming

8.2.1 MPC to Parametric Quadratic Programming

A classical MPC can be translated from

$$\begin{aligned} J^\star(x) = \min\, & x_N^T Q_f x_N + \sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i \\ \text{s.t. } & x_0 = x \\ & x_{i+1} = Ax_i + Bu_i \\ & Cx_i + Du_i \leq b \end{aligned} \tag{8.1}$$

to:

$$\begin{aligned} J^\star(x) = \min_u \frac{1}{2} u^T Q u + (Fx + f)^T u \\ \text{s.t. } Gu \leq Ex + e \end{aligned} \tag{8.2}$$

8.2.2 KKT Optimality Condition

For optimization problem:

$$\begin{aligned} \min_x f(x) \\ \text{s.t. } g(x) \leqslant 0 \\ h(x) = 0 \end{aligned} \tag{8.3}$$

The optimum always meet:

Proposition 8.2.1 (Karush-Kuhn-Tucker conditions)**.**

$$\begin{aligned} \nabla f(x) + \nabla g(x)\mu + \nabla h(x)\lambda = 0 \\ \mu^T g(x) = 0 \\ \mu \geqslant 0 \\ h(x) = 0 \\ g(x) \leqslant 0 \end{aligned} \tag{8.4}$$

KKT for ConvexQPs

For conex QPs:

$$\begin{aligned} \min f(z) := \frac{1}{2} z^\top Q z \\ \text{s.t.} \quad Az \leq b \end{aligned} \tag{8.5}$$

We have

$$\begin{aligned} &Qz + A^\top \lambda = 0, \lambda \geq 0 && \text{Gradient is in the normal cone} \\ &Az \leq b && \text{Optimal point must be feasible} \\ &\lambda^\top (Az - b) = 0 && \text{Normal cone contains only active constraints} \end{aligned} \tag{8.6}$$

8.3 Parametric Linear Complementarity Problems

8.3.1 Parametric Linear Complementarity

Definition 8.3.1 (Parametric Linear Complementarity)**.** *Given matrices M, q and Q, find functions $w(x), z(x)$ such that*

$$\begin{aligned} w - Mz = q + Qx \\ w^\top z = 0 \\ w, z \geq 0 \end{aligned}$$

we can always convert a pQP problem to pLCP problem by the following way:

$$\begin{aligned} J^\star(x) := \min_u \frac{1}{2} u^T Q u + (Fx + f)^T u \\ \text{s.t. } Gu \geq Ex + e \\ u \geq 0 \end{aligned} \tag{8.7}$$

First, derives the KKT Conditions

$$\begin{aligned} &Qu + Fx + f - G^T\lambda - \nu = 0 && \text{Stationarity} \\ &-s + Gu = Ex + e, u \geq 0 \quad s \geqslant 0 && \text{Primary feasibility} \\ &\lambda, \nu \geq 0 && \text{Dual feasibility} \\ &\nu^T u = 0, \lambda^T s = 0 && \text{Complementarity} \end{aligned} \tag{8.8}$$

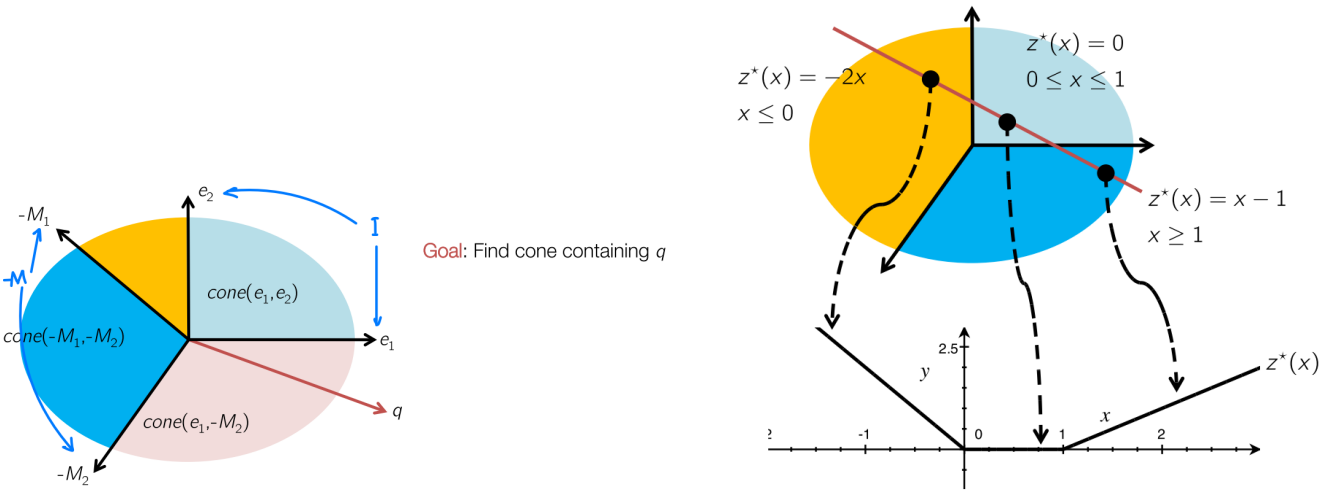
Then transform to Matrix Format:

$$\begin{aligned} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} \nu \\ s \end{pmatrix} - \begin{bmatrix} Q & -G^T \\ G & 0 \end{bmatrix} \begin{pmatrix} u \\ \lambda \end{pmatrix} = \begin{bmatrix} F \\ -E \end{bmatrix} x + \begin{bmatrix} f \\ -e \end{bmatrix} \\ \nu, s, u, \lambda \geq 0 \\ \nu^T u = s^T \lambda = 0 \end{aligned} \tag{8.9}$$

8.3.2 The Geometry Perspective

we have

1. $w^\top z = 0, w, z \geq 0$: either w_i or z_i is zero for all i
2. $Iw - Mz = q$: q is in the cone of non-zero variables



(a) (b)

Figure 8.2: pLCP Geometry

8.3.3 The Algebra Perspective

Definition 8.3.2 (complementary basis)**.**

8.3.4 Efficient Solution Methods

We can define a graph G, in which vertices are non-empty CRs and edges are adjacent CRs, then we can find all critical regions by standard graph enumeration

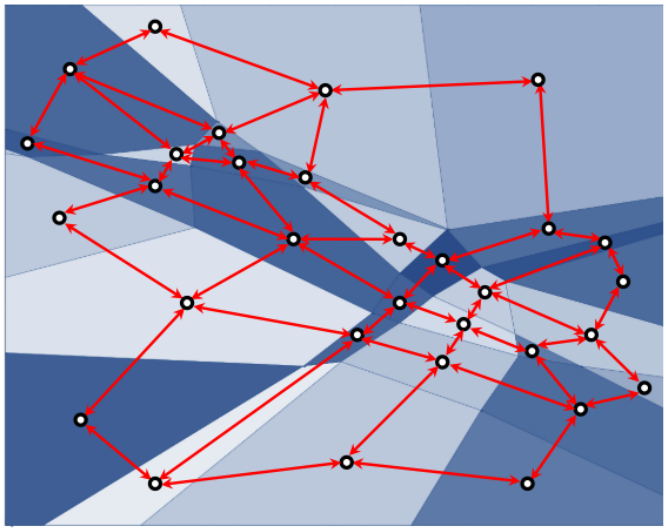


Figure 8.3: Efficient Enumeration By Graph

Basic Idea

Definition 8.3.3 (Sufficient Matrix)**.** *Sufficient Matrix*
A matrix $M \in {}^{n\times n}$ is called column sufficient if it satisfies the implication

$$[z_i(Mz)_i \leq 0 \text{ for all } i] \implies [z_i(Mz)_i = 0 \text{ for all } i]\,.$$

The matrix M is called row sufficient if its transpose is column sufficient. If M is both column and row sufficient, then it is called sufficient.

Proposition 8.3.4. *Positive semi-definite matrices are sufficient. (applies to non-symmetric PSD matrices too)*

Proposition 8.3.5 (LCPs with Sufficient Matrices)**.** *LCPs with Sufficient Matrices*

- *If M is a sufficient matrix, then the relative interiors of any two distinct complementary cones are disjoint (Cones cannot overlap)*
- *If M is a sufficient matrix, then the union of all complementary cones $K(M)$ is a convex polyhedral cone $K(M) = cone([I - M])$ (Neighbour graph is connected)*

Computing Adjacent Regions

Method 8.3.6.

8.4 Point Location Problem

Sequential Search

Method 8.4.1 (Sequential Search). *Sequential Search*
Test each region one by one whether the given point meets its constraints

8.4.1 Bisection Search

Method 8.4.2 (Bisection Search). *Bisection Search*

1. *Find hyperplane that separates regions into two equal sized sets*
2. *Repeat for left and right sets*

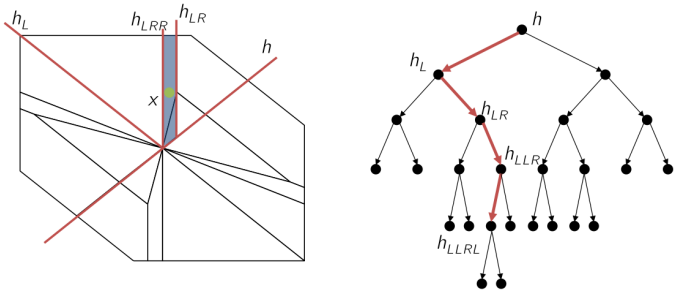


Figure 8.4: Bisection Search

8.5 Toolbox

Chapter 9

Lecture 9: Non-Linear MPC

9.1 Introduction

9.1.1 Model

$$\begin{aligned} u^{\star}(x_0) = \operatorname{argmin} & \textstyle\sum_{i=0}^{N-1} l(x_i, u_i) + V_f(x_N) \\ \text{s.t.} \quad & x_{i+1} = f(x_i, u_i) && \forall i = 0, \dots, N-1 \\ & g(x_i, u_i) \leq 0 && \forall i = 0, \dots, N-1 \\ & h(x_N) \leq 0 \end{aligned} \tag{9.1}$$

where f, g and h are continuous.

Compared to Linear MPC

- **Feasibility:** same assumptions on terminal constraint
- **Stability:** same assumptions on stage cost and terminal cost

9.1.2 Hardness and Challenge

Challenge

Discretization Methods and **How to compute gradients**

Hardness

- invariant set is hard to compute
- drop to local minimum

9.2 Nonlinear Programming

9.2.1 Newton's Method for Convex Optimization

$$\Delta z_{nt} = -\nabla^2 f(z)^{-1} \nabla f(z) \tag{9.2}$$

Proposition 9.2.1.

- *$z + \Delta z_{nt}$ minimizes second order approximation*
- *if z is close to optimum, $\|\nabla f(z)\|_2$ converges to zero quadratically*

9.2.2 Gauss-Newton Method for NLPs: Sequential Quadatic Programming

General Cases

$$\begin{aligned} &\min F(z) \\ &\text{s.t.} G(z) \leq 0 \\ &H(z) = 0 \end{aligned} \tag{9.3}$$

compute quadratic approximation

$$\begin{aligned} &\min_{\boldsymbol{\Delta z}} \nabla F\left(z^k\right)^T \Delta z + \frac{1}{2}\Delta z^T A^k \Delta z \\ &\text{s.t. } G\left(z^k\right) + \nabla G\left(z^k\right)^T \Delta z \leq 0 \\ &H\left(z^k\right) + \nabla H\left(z^k\right)^T \Delta z = 0 \end{aligned} \tag{9.4}$$

where A^k is the Hessian of the Lagrangian Function

Method 9.2.2. *After computing the quadratic approximation:*

1. *using the appoximation format to compute a direction*
2. *for the direction, do step size search*

Quadratic Case

$$\begin{aligned} &\min F(z) = ||R(z)||^2 \\ &\text{s.t.} G(z) \leq 0 \\ &H(z) = 0 \end{aligned} \tag{9.5}$$

compute quadratic approximation

$$\begin{aligned} \min \quad & \left\| R\left(z^k\right) + \nabla R\left(z^k\right)^T \Delta z \right\|^2 \\ \text{s.t.} \quad & G\left(z^k\right) + \nabla G\left(z^k\right)^T \Delta z \leq 0 \\ & H\left(z^k\right) + \nabla H\left(z^k\right)^T \Delta z = 0 \end{aligned} \tag{9.6}$$

9.2.3 Other Methods

such as Interior-Point, and Operator Splitting Methods, etc.

9.3 Discretization

9.3.1 Direct Integration

Use a integration algorithm to compute $x(k+1) = x(k) + \int_{t=T_s k}^{T_s(k+1)} f(x,u)dt$

Eular Approximation

$$x^+ = x + hf(x,u) \tag{9.7}$$

Runge-Kutta (Order Two)

$$\begin{aligned} x^+ &= x + hf(x) + \frac{h^2}{2} J_f(x) f(x) + \mathcal{O}\left(h^3\right) \\ &= x + \frac{h}{2} f(x) + \frac{h}{2} \left(f(x) + h J_f(x) f(x)\right) + \mathcal{O}\left(h^3\right) \\ &\approx x + \frac{h}{2} f(x) + \frac{h}{2} f(x + hf(x)) \end{aligned} \tag{9.8}$$

So, we will have

$$\begin{aligned} x^+ &\approx x + h(\frac{1}{2}k_1 + \frac{1}{2}k_2) \\ k_1 &= f(x), \quad k_2 = f(x + hk_1) \end{aligned} \tag{9.9}$$

Runge-Kutta (Order Four)

$$\begin{aligned} x_{k+1} &= x_k + h\left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right) \\ k_1 &= f\left(t_k, x_k\right) \\ k_2 &= f\left(t_k + \frac{h}{2}, x_k + \frac{h}{2} k_1\right) \\ k_3 &= f\left(t_k + \frac{h}{2}, x_k + \frac{h}{2} k_2\right) \\ k_4 &= f\left(t_k + h, x_k + h k_3\right) \end{aligned} \tag{9.10}$$

9.3.2 Collocation

The Collocation method has two steps:

1. first, interpolation based on base functions

$$x(t) = \sum_{i=0}^{q} w_i \beta_i(t) \tag{9.11}$$

2. Enforce that the dynamic equations are met at the collocation points

$$\dot{x}\left(t_k\right) = f\left(x_k, u_k\right) = \sum_{i=0}^{q} w_i \dot{\beta}_i\left(t_k\right) \tag{9.12}$$

Polynomial Interpolation

Time grid

$$\{t_{k,0}, \ldots, t_{k,K}\} \in [t_k, t_{k+1}] \tag{9.13}$$

Lagrange Polynomials

$$P_{k,i}(t) = \prod_{j=0, j\neq i}^{K} \frac{t - t_{k,j}}{t_{k,i} - t_{k,j}} \in \mathbb{R} \tag{9.14}$$

We have the property

$$P_{k,i}\left(t_{k,l}\right) = \begin{cases} 1 & \text{if } l = i \\ 0 & \text{if } l \neq i \end{cases} \tag{9.15}$$

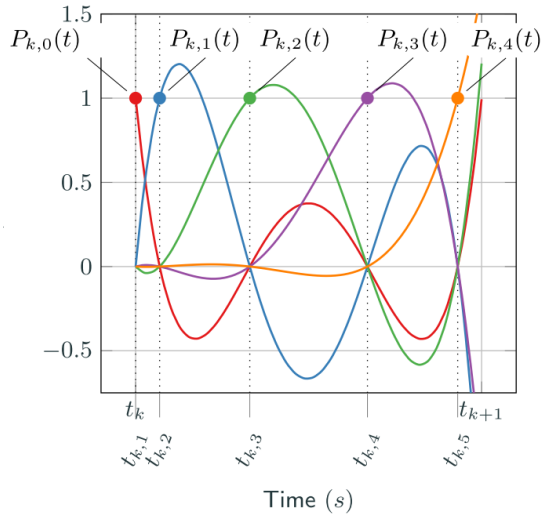


Figure 9.1: Polymial Interpolation

Collation

For the given interpolation, we have:

Proposition 9.3.1.

$$x\left(\theta_k, t\right) := \sum_{j=0}^{K} \theta_{k,j} P_{k,j}(t) \quad x\left(\theta_k, t_{k,j}\right) = \theta_{k,j}$$

Then we can define two collation constraints:

$$\begin{aligned} &x\left(\theta_k, t_k\right) = x_k \quad \text{Initial condition} \\ &\frac{\partial}{\partial t} x\left(\theta_k, t_{k,j}\right) = f\left(x\left(\theta_k, t_{k,j}\right), u_k\right) \quad \text{Dynamics} \end{aligned} \tag{9.16}$$

Then we will have

$$\begin{aligned} \theta_{k,0} &= x_k \\ D\theta_k &= f\left(\theta_k, u_k\right) \end{aligned} \tag{9.17}$$

where the elements of the D are the constants $\dot{P}_{k,j}\left(t_{k,j}\right)$ and $f\left(\theta_k, u_k\right) = \begin{bmatrix} f\left(\theta_{k,0}, u_k\right) & \cdots & f\left(\theta_{k,K}, u_k\right) \end{bmatrix}^T$

9.4 Gradients: Using Algorithmic Differentiation

[Youtube] What is Automatic Differentiation

There are mainly 4 methods of differentiation:

- Manual: error prone, data entry impractical
- Numerical: error prone, truncation error in finite differences
- Symbolic: too dens, based on trees, too impractical
- Automatic: exact to roundoff, can even apply to each side of discontinuous

In this section, we will use the following example to introduce algorithmic differentiation

$$f\left(x_1, x_2\right) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - \mathrm{e}^{x_2}\right] \times \left[\frac{x_1}{x_2} - \mathrm{e}^{x_2}\right] \tag{9.18}$$

The computation graph is shown in Figure 9.2.

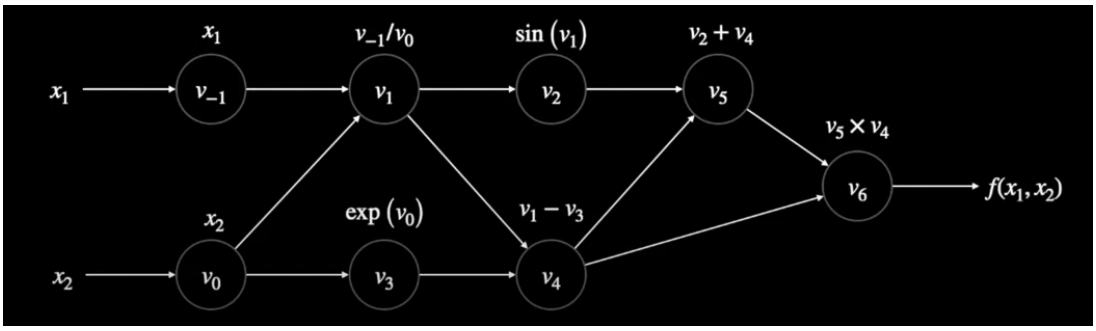


Figure 9.2: AD Example Computation Graph

9.4.1 Forward Algorithmic Differentiation

The forward method of the example is shown in Figure 9.3.

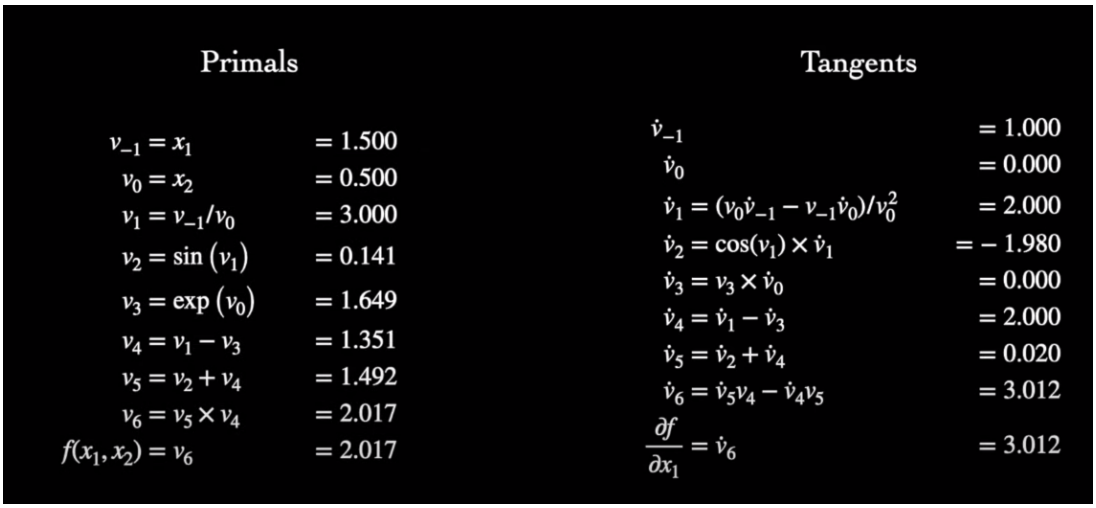


Figure 9.3: AD Example Forward Mode

From the graph, it is easily to show for each x, we need a series of computation. The number of computation highly depend on the input dimension. So for a function: $f : \mathbb{R}^n \to \mathbb{R}^m$. Forward mode is ideal when $n << m$.

One method to implement forward function is by operator overloading, that is when calculating value, we simultaneously compute the derivation. An example is shown in Figure 9.4.

9.4.2 Backward Algorithmic Differentiation

The method of Backward is use the follow rule:

$$\text{"Adjoint"} \quad \bar{v}_i = \frac{\partial f}{\partial v_i} = \sum_{j:\text{ child of } i} \bar{v}_j \frac{\partial v_j}{\partial v_i} \tag{9.19}$$

The backward method of the example is shown in Figure 9.5 and Figure 9.6.

It can be seen that what we want is $\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right] = [\bar{x}_1, \bar{x}_2]$. The computation number highly depend on the output of the function. So for a function: $f : \mathbb{R}^n \to \mathbb{R}^m$. Backward mode is ideal when $m << n$.

```python
class Variable:

    def __init__(self, val, dot=0):
        self.val = val
        self.dot = dot

    def __add__(self, other):
        res = Variable(self.val + other.val)
        res.dot = self.dot + other.dot
        return res
```

Figure 9.4: AD Example Forward Mode Implementation

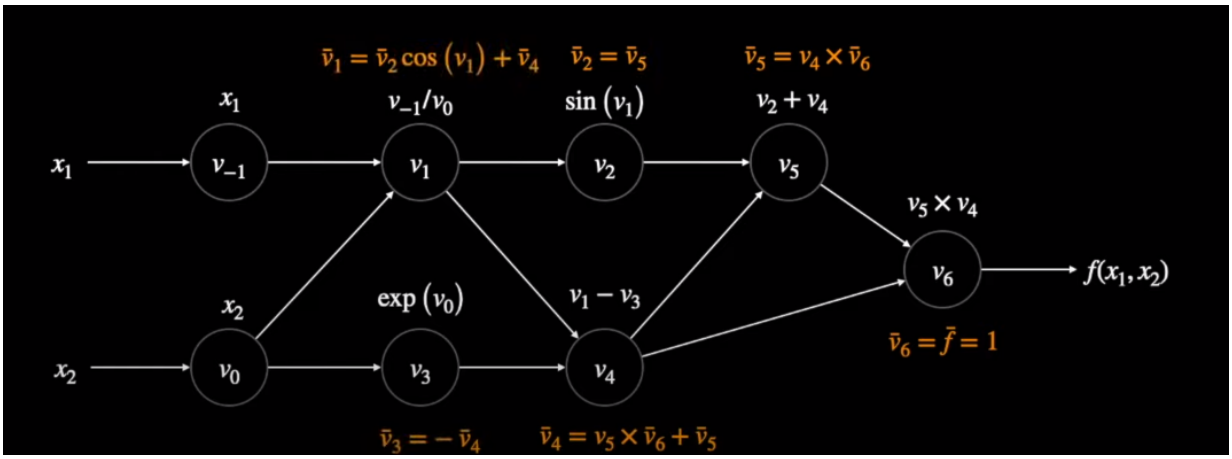


Figure 9.5: AD Example Backward Mode 1

9.5 Summary

In this chapter, we introduced the Nonlinear MPC. There are three main problems for nonlinear MPC: nonlinear programming, discretization, and derivation calculation.

So we first introduce how to do nonlinear programming.

Then we introduce how to discretization: one way is to use approximation integration, like Eular Approximation, Runge-Kutta Methods; another way is collocation, that is we first define basis function and use boundary condition to specify the parameters.

Finally, we introduce how to use algorithmic differentiation to do derivation calculation.

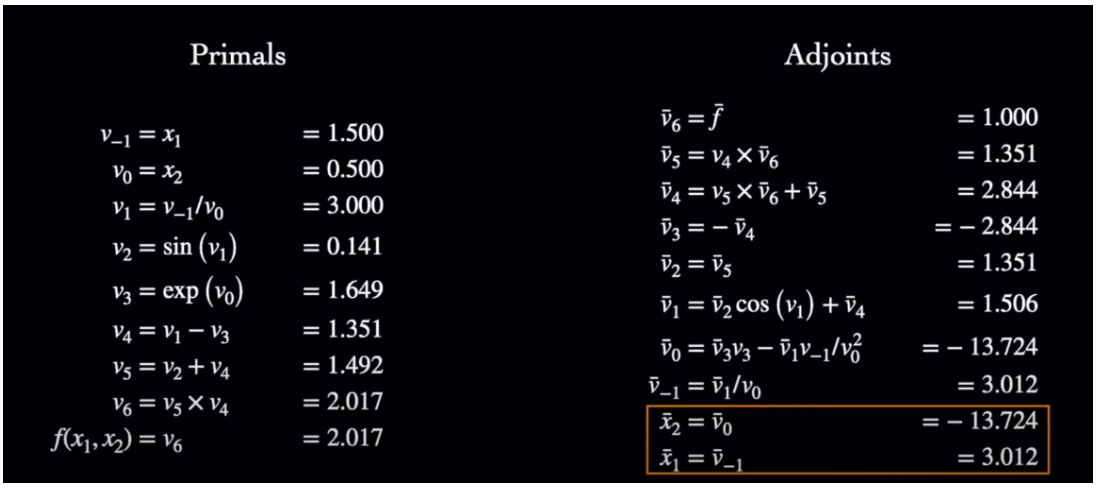


Figure 9.6: AD Example Backward Mode 2