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

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

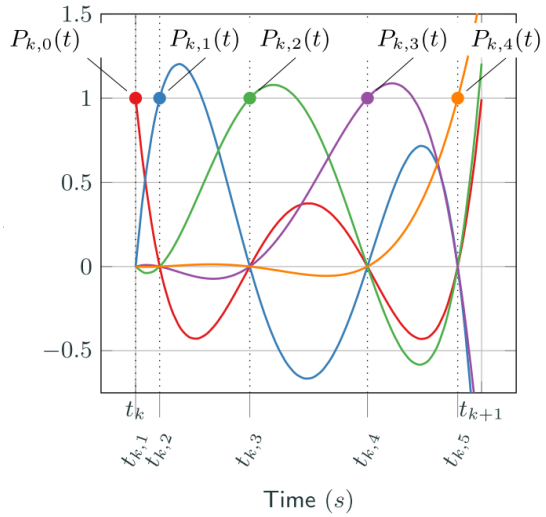

Figure 9.1: Polymial Interpolation

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.

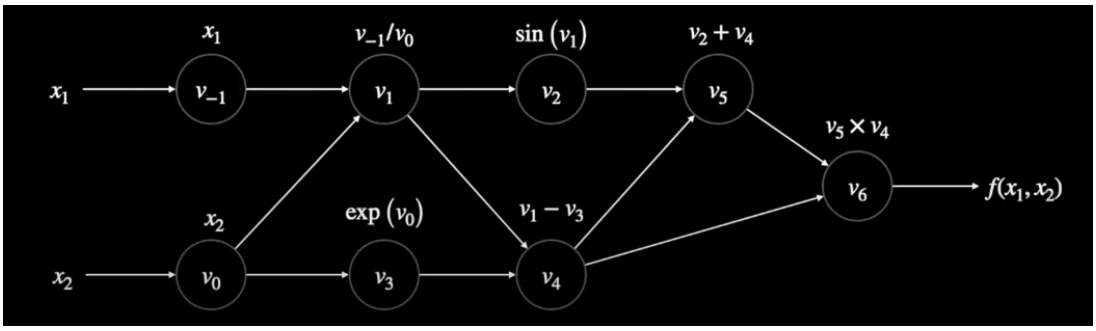

Figure 9.2: AD Example Computation Graph

9.4.1 Forward Algorithmic Differentiation

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

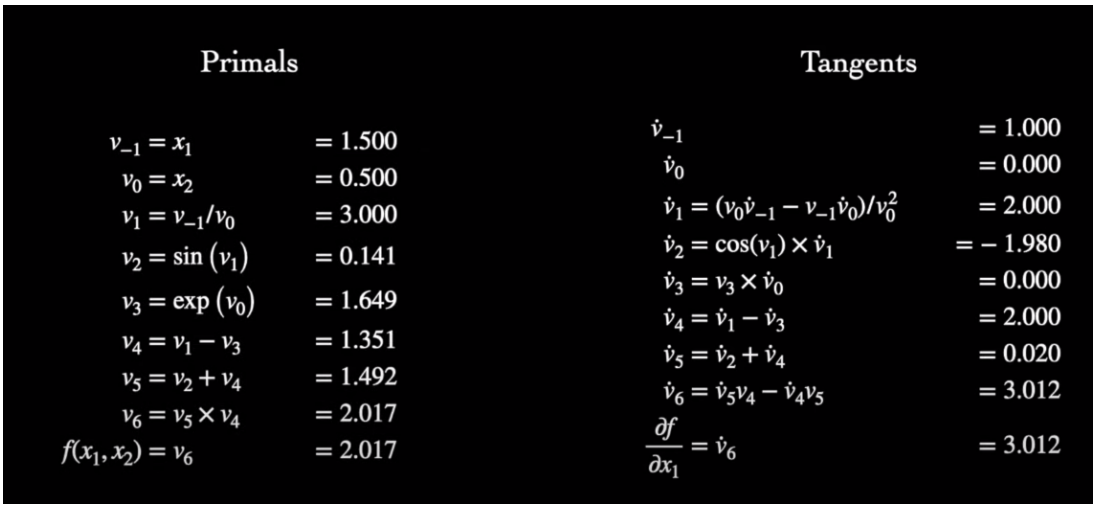

Figure 9.3: AD Example Forward Mode

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

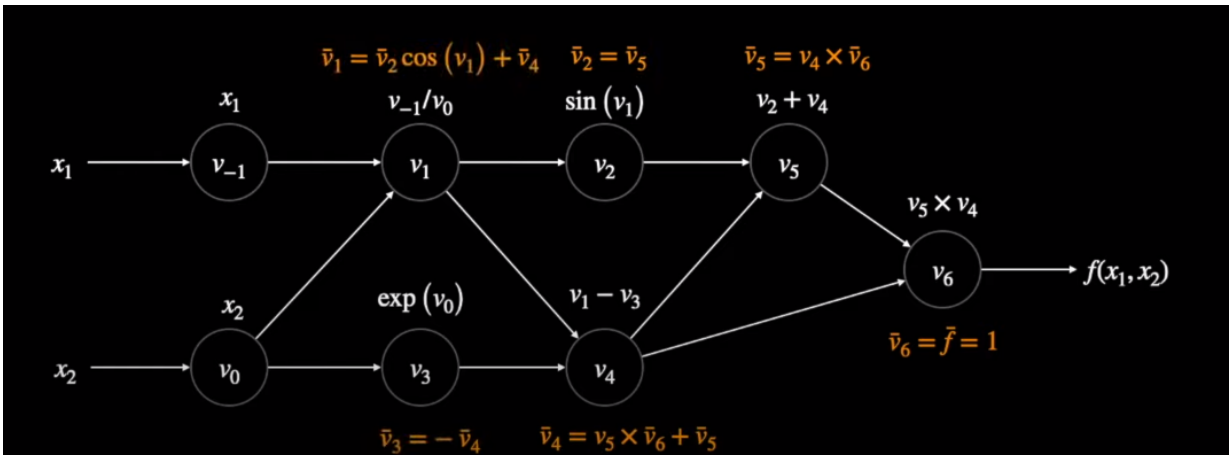

Figure 9.5: AD Example Backward Mode 1

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.

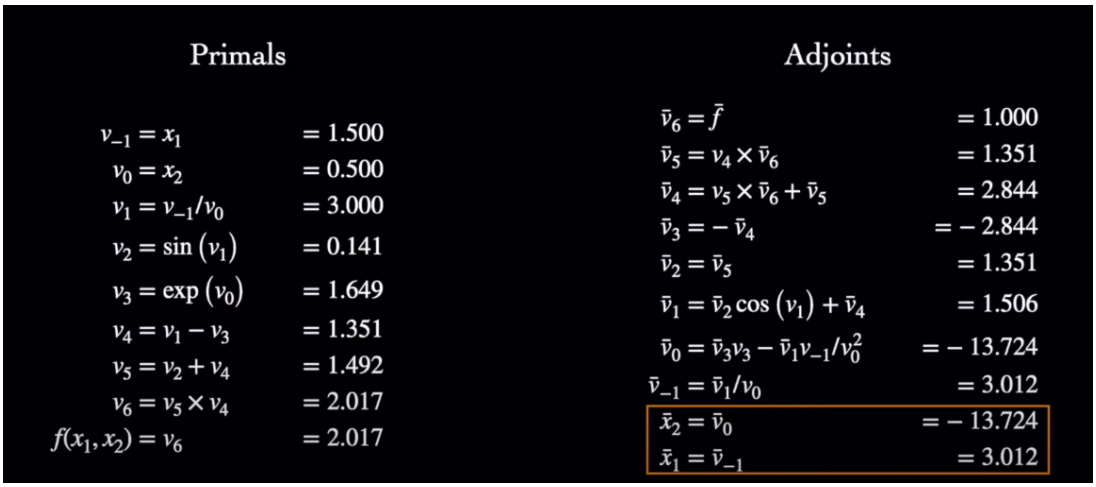

Figure 9.6: AD Example Backward Mode 2