Chair of
Automatic Control

Sensitivity-based distributed optimal control with fixed-point iterations

Project thesis

of

Qinglin Yang

6. November 2024

Supervisor:	Maximilian Pierer von Esch, M.Sc.
Auditor:	Prof. Dr.-Ing. K. Graichen

Chair of Automatic Control
Faculty of Engineering
Friedrich-Alexander-Universität Erlangen-Nürnberg

Declaration

Abstract

This article describes an algorithm that is a combination of a Fixed-point iteration method and a Sensitivity-based algorithm. The algorithm aims to solve the local optimal control problem, which is one of the most critically important steps for distributed model predictive control. With the result of the algorithm, we can implement the whole distributed model predictive control. The combination is numerically efficient in the sense that the optimal conditions of the local optimal control problem are solved within a coupled forward-backward integration and the sensitivities can be calculated locally for each neighbor. It has one communication step per algorithm iteration. The whole process leads to a fully distributed algorithm. Convergence is shown for an upper bound on the horizon time. I use the Anderson acceleration to enlarge the original horizon time so that it has better performance. And when I implement the distributed model predictive control, the controllers have better predictive competence as well. In addition, the algorithm has an important trade-off between convergence speed and stability of the sensitivity-based algorithm. And except Anderson Acceleration the maximum horizon time length can typically be enlarged by damping the iterates as well.

Contents

1 Introduction

This chapter aims to provide a comprehensive overview to gain a good understanding of the present work and the associated field. Firstly the research background is stated, and an outline of the current state of the field is provided. Following that, the motivation and the specific tasks associated with the research are explained. Finally, the structure of the thesis is presented.

1.1 Research background

Nowadays to solve many complicated industrial controlling problems these processes are usually described with multi-input, multi-out variables and, dynamic models. These problems are usually decentralized and then single-input and single-output Proportional-Integral-Derivative(PID) loops namely decentralized controllers are applied to solve these subproblems. Initially, these controllers were designed without consideration of dynamic models among these decentralized controllers. Therefore the linear or nonlinear model predictive control based on optimization becomes crucial. And all kinds of constraints can be considered directly. [1]

Model Predictive Control(MPC) is a kind of advanced control strategy. It is widely used in industry control processes, autonomous driving, robotics, and ohter fields. The core thinking of MPC is predicting future states with dynamic models of systems basing current states. To solve large-scale distributed optimal control problems, the optimized method namely Distributed Model Predictive Control(DMPC) is extended from the original MPC. The distributed Model Predictive Control method splits up the original large system into some smaller subsystems and then solves these smaller problems in parallel by exchanging necessary information among these subsystems. The most important point of such algorithms is to force the solutions of local problems to converge to the solution of the central problem. The extended method DMPC does not only maintain the original features such as explicitly handling constraints or the applicability to nonlinear systems, but also the flexible, decentralized, and scalable structure of multi-agent systems.

1.2 State of the Art

The MPC concept was first introduced in the 1970s. In the 1980s it was applied widely and had deep development. In 1998 A decomposition-coordination algorithm was developed which was applied to optimal control of complex irrigation systems [2]. Completely centralized control of large, networked systems is impractical. In addition, completely decentralized controlling of such systems frequently results in unacceptable control performance. In 2002 a new DMPC scheme was described in this article [3]. In this article, the local subsystems exchange their predictions by communicating with other controllers about their local MPC problems. For the full local state feedback and one-step delayed prediction exchange case, stability is ensured for controllable systems. In 2005 in this paper [4] the author extended existing concepts in linear model predictive control to a unified, theoretical framework for distributed MPC with guaranteed nominal stability and performance properties. In 2007 this paper [5] stated a DMPC framework with guaranteed feasibility and nominal stability properties. In 2010 a cooperative distributed linear model predictive control strategy [6] was proposed for any finite number of subsystems satisfying a stability condition. With this strategy hard input constraints are satisfied. In 2013 this article [7] presented a stopping condition that guaranteed the feasibility of the optimization problem and a prespecified performance of the closed-loop system. In the same year, the multi-step gradient methods [8] for network-constrained optimization of strongly convex functions with Lipschitz-continuous gradients were presented. In addition, parallel implementation of hybrid MPC [9] was presented in this article. In this article, different methods for achieving parallelism at different levels of the algorithms are shown. In 2014 this paper [10] described the cooperative MPC of networked control systems with multiple wireless nodes. These nodes are chosen to calculate the control values to solve a cooperative MPC by communicating with their neighbors. In 2015 a paper presenting a DMPC strategy with application to AGC was published in the Chinese Control Conference. This paper addressed a new DMPC algorithm for load-frequency control(LFC) of a four-area interconnected power system. The overall system is decomposed into four subsystems, where each control area has an independent MPC controller.

1.3 Motivation

MPC computes the control input at the current moment by solving an online optimization problem. However, for complex and large-scale systems, centralized MPC needs to process a large number of decision variables and constraints in real-time, making the optimization problem too large and the computation time too long. So it is difficult to be applied in real-time scenarios. By dividing the system into several interrelated subsystems, each subsystem performs local optimization independently, allowing the global problem to be decomposed into multiple smaller optimization problems. This distributed approach significantly reduces the computational burden on each subsystem and allows for parallel computation. So it greatly improves computational efficiency and enables real-time control. In addition, to combine the optimal properties of centralized model predictive controllers and the modularity and flexibility of decentralized control systems, subsystems can communicate by state or input coupling items.

1.4 Objectives

This article presents an approach based on sensitivity to solve local optimal control problems. Except for state or input coupling items the coupling method of the cost function is presented as well. Firstly the algorithm is implemented with MATLAB. Then the algorithm is optimized with the Anderson Acceleration algorithm. Particularly using the fixed-point iteration scheme solves local optimal control problems. And particularly with the increment of the prediction horizon, the divergence will occur. That means an upper bound on the prediction horizon exists. So if the scheme is applied in the industrial process, there exists a trade-off between convergence and stability as well as the performance of the DMPC scheme. To deal with the trade-off problem better, the algorithm is optimized. By applying Anderson Acceleration the convergence occurs with less time. So the prediction horizon increases as well. At last six plots about trajectories of state variables x_i, adjoint state variables λ_i, and the input u_i about time can be obtained. These three plots of six plots are the result of Sensitivity-Based distributed optimal controllers. The other three plots are the result of the centralized controller. These two groups of plots are the same. When these trajectories of distributed optimal controllers subtract from the trajectories of centralized controllers, a convergence occurs.

1.5 Structure of the Thesis

This DMPC scheme article is presented as follows:

Chapter 2: In this chapter how to apply fixed-point method [11] to solve central problems is stated. The algorithm is emphasized and the mathematical derivation of the algorithm is presented in detail.

Chapter 3: Distributed Sensitivity-Based optimization algorithm is presented in detail based on the previous content. The Sensitivity-Based algorithm is shown by calculating the sensitivity among subsystems and states how to implement the whole algorithm with code.

Chapter 4: In this chapter how to apply Anderson acceleration algorithm [12] to make the convergence occurs earlier. In addition, I will increase the horizon time and the number of discrete time points over the time horizon and then compare two different results of the code without Anderson acceleration and with Anderson acceleration respectively.

Chapter 5: In this chapter I do some numerical evaluations with some different examples. The goal is to verify some properties of the algorithm. In addition, in this section, some graphs of different examples are shown. By comparing these graphs, some conclusions are presented.

2 Optimal central control problem

Firstly in the algorithm, the form of dynamic models of systems is nonlinear affine-input

$$\dot{\mathbf{x}} = \mathbf{f}_0(\mathbf{x}) + \sum_{i=1}^{m} \mathbf{g}_i(\mathbf{x})\mathbf{u}_i := \mathbf{f}(\mathbf{x},\mathbf{u}), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{2.1}$$

state $\mathbf{x} \in \mathbf{R}^n$ and input $\mathbf{u} = \left[u_1, \ldots, u_m\right]^\top \in \mathbf{R}^m$ It is assumed that the functions $\mathbf{f_0}, \mathbf{g_i} : \mathbb{R}^n \to \mathbb{R}^n$ are twice continuously differentiable in their arguments. The origin is an equilibrium of the system for u = 0, i.e. $f(0,0) = 0$. The input is subject to the (vector-valued) point-wise-in-time box constraints $\mathbf{u}(t) \in \left[\mathbf{u}^-, \mathbf{u}^+\right]$.

2.1 Optimal control problem

Optimal control problem(OCP) is one of the most important steps for designing a DMPC controller.

$$\min_{\bar{\mathbf{u}}} J(\mathbf{x}_k, \bar{\mathbf{u}}) := V(\bar{\mathbf{x}}(T)) + \int_0^T l(\bar{\mathbf{x}}(\tau), \bar{\mathbf{u}}(\tau))\, d\tau \tag{2.2a}$$

$$\text{s.t.} \quad \dot{\bar{\mathbf{x}}}(\tau) = f(\bar{\mathbf{x}}(\tau), \bar{\mathbf{u}}(\tau)), \quad \bar{\mathbf{x}}(0) = \mathbf{x}_k \tag{2.2b}$$

$$\bar{\mathbf{u}}(\tau) \in [\mathbf{u}^-, \mathbf{u}^+], \quad \tau \in [0, T] \tag{2.2c}$$

Sampling and Time Points is the first factor for solving the OCP problem. The model predictive controller can determine a stabilizing feedback control law for a nonlinear system in the form of equation (2.1) at discrete sampling time points $t_k = t_0 + k\Delta t$, where $k \in \mathbb{N}_{\not{k}}^{+}$, representing non-negative integers, and δt is the sampling time interval. $\mathbf{x}_k = \mathbf{x}(t_k)$: The state of the system at time t_k.The bar notation $\bar{x}(\tau)$ represents internal variables for time $\tau \in [0, T]$, where $T \geq \Delta t$ represents the prediction horizon.

The **Cost Function** $J(\mathbf{x}_k, \bar{\mathbf{u}})$ represents the objective to minimize in MPC. The cost function consists of two terms. Terminal cost $V(\mathbf{x}(\bar{T}))$ the cost at the end of the prediction horizon T. Integral cost $\int_0^T l(\bar{\mathbf{x}}(\tau), \bar{\mathbf{u}}(\tau))d\tau$: The accumulated cost over the time horizon $\tau \in [0, T]$, which accounts for both of state and control inputs of the system.

The integral cost function $l(x, u)$ is given in the particular form:

$$l(\mathbf{x}, \mathbf{u}) := l_0(\mathbf{x}) + \frac{1}{2}\sum_{i=1}^{m} \mathbf{R}_i \mathbf{u}_i^2 \tag{2.3}$$

$l_0(x)$: A state-dependent cost function, which is assumed to be twice continuously differentiable. $\frac{1}{2}\sum_{i=1}^{m} \mathbf{R}_i \mathbf{u}_i^2$: A quadratic control term, where $\mathbf{R}_i$ represents a positive weights, penalizing large control inputs $\mathbf{u}_i$.

The **dynamic function of the system** is described by the differential equation:

$$\dot{\mathbf{x}}(\tau) = f(\bar{\mathbf{x}}(\tau), \bar{\mathbf{u}}(\tau)),$$

which represents how the state $\mathbf{x}$ evolves over time based on the control input $\mathbf{u}$.

The initial condition is given by:

$$\bar{\mathbf{x}}(0) = \mathbf{x}_k,$$

meaning that the internal state at the beginning of the horizon is equal to the state at the sampling time t_k.

For **Constraints** the control input $\bar{u}(\tau)$ is constrained between an upper bound and a lower bound: $\bar{u}(\tau) \in [u^-, u^+]$, $\tau \in [0, T]$ This ensures that the control input is constrained within physically meaningful limits.

The function $V : \mathbb{R}^n \to \mathbb{R}^+$ represents the **Terminal Cost** $V(\mathbf{x})$, which is applied at the final time T. Both $V(\mathbf{x})$ and $l_0(\mathbf{x})$ are assumed to satisfy quadratic bounds:

$$m_l \|\mathbf{x}\|^2 \leq l_0(\mathbf{x}) \leq M_l \|\mathbf{x}\|^2, \tag{2.4}$$

$$m_v \|\mathbf{x}\|^2 \leq V(\mathbf{x}) \leq M_v \|\mathbf{x}\|^2, \tag{2.5}$$

for positive constants M_l, m_l, M_v, and m_v.

2.2 A fixed-point iteration scheme for the OCP problem

This section describes an optimization algorithm based on a fixed-point iteration scheme. The algorithm aims to obtain the numerical solution of an optimal control problem in the form of (2.1)-(2.2) within a real-time MPC framework. Particularly, this optimization algorithm not only takes advantage of the optimal control problem but also operates without any terminal constraints [13].

Definition 2.1 (Hamiltonian function). *The* ***Hamiltonian function*** *for the optimal control problem (2.2) is defined by*

$$H(\mathbf{x}, \mathbf{u}, \lambda, t) = l(\mathbf{x}, \mathbf{u}, t) + \lambda^\top f(\mathbf{x}, \mathbf{u}, t) \tag{2.6}$$

with the ***Lagrange multipliers*** $\lambda(t) \in \mathbb{R}^n$.

$\lambda(t) \in \mathbb{R}^n$ $\mathbf{x}^*(t), and \lambda(t)$ satisfy the canonical equations:

Theorem 2.1 (Necessary first-order optimality conditions). ***Suppose*** *that the functions l and f of the optimal control problem (2.2) are continuously differentiable in (x, u) and continuous in t. Suppose that $u^* \in C_m^0[t_0, t_f]$ is a (local) optimal solution of (2.2) with associated state trajectory $x^* \in C_n^1[t_0, t_f]$. Then, there exist time functions $\lambda^* = [\lambda_1^*, \dots, \lambda_n^*]^\top$ such that the following conditions are satisfied:*

(a) (u^*, x^*, λ^*) satisfy

$$\dot{x}^* = H_\lambda(x^*, u^*, \lambda^*, t) = f(x^*, u^*, t) \tag{2.7a}$$
$$\dot{\lambda}^* = -H_x(x^*, u^*, \lambda^*, t) = -l_x(x^*, u^*, t) - f_x(x^*, u^*, t)^\top \lambda^* \tag{2.7b}$$
$$\tag{2.2}$$

for $t \in [t_0, t_f]$ with the boundary conditions

$$x^*(t_0) = x_0 \tag{2.8a}$$
$$x_i^*(t_f) = x_{f,i}, \quad i \in \mathcal{I}_f \tag{2.8b}$$

and the transversality condition

$$\lambda_i^*(t_f) = V_{x_i}(x(t_f), t_f), \quad i \notin \mathcal{I}_f \tag{2.9}$$

(b) If the end time t_f is unspecified and t_f^* is (locally) optimal, then (2.7) is satisfied for $t \in [t_0, t_f^*]$ and (2.7), (2.8) for $t_f = t_f^*$. In addition, the Hamiltonian satisfies the transversality condition

$$H(x^*(t_f^*), u^*(t_f^*), \lambda^*(t_f^*), t_f^*) = -V_t(x^*(t_f^*), t_f^*) \tag{2.10}$$

The function H_x and V_x denote the partial derivatives of H and V concerning x.

Pontryagin's Maximum Principle (PMP) [14] The algorithm is based on Pontryagin's Maximum Principle (PMP) to ensure the solution of the optimal control problem(2.1) - (2.3).

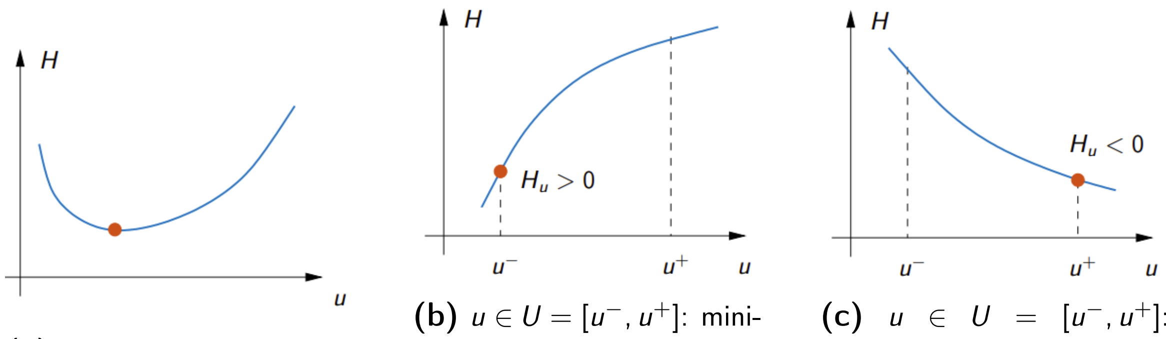
Terminal Cost $V(\mathbf{x})$: The function $V : \mathbb{R}^n \to \mathbb{R}^+$ represents the terminal cost, which is applied at the final time T. Both $V(\mathbf{x})$ and $l_0(\mathbf{x})$ are assumed to satisfy quadratic bounds:

$$m_l\|\mathbf{x}\|^2 \le l_0(\mathbf{x}) \le M_l\|\mathbf{x}\|^2,$$

$$m_v\|\mathbf{x}\|^2 \le V(\mathbf{x}) \le M_v\|\mathbf{x}\|^2,$$

for positive constants M_l, m_l, M_v, and m_v

This function describes that an adjoint variable trajectory $\bar{\lambda}_k^*(\tau)$ and a state variable trajectory $\bar{\mathbf{x}}_k^*(\tau)$ $\tau \in [0, T]$ satisfy the canonical function. With these trajectories, the function (2.3) can be minimized with the optimal trajectory of input $\bar{\mathbf{u}}_k^*, \tau \in [0, T]$



(a) u unconstrained: $H_u = 0$, $H_{uu} \ge 0$

(b) $u \in U = [u^-, u^+]$: minimum at left boundary ($H_u > 0$)

(c) $u \in U = [u^-, u^+]$: minimum at right boundary ($H_u < 0$)

Figure 2.1: Minimum of the Hamiltonian in the unconstrained and constrained case.

The application of **Pontryagin's Maximum Principle**:

1. Set-up of the Hamiltonian with

$$H(\mathbf{x}, \mathbf{u}, \lambda) = l(\mathbf{x}, \mathbf{u}) + \lambda^\top f(\mathbf{x}, \mathbf{u}). \tag{2.11}$$

2. Computation of $\mathbf{u}$ from (5.46c) such that $\mathbf{u}$ can be expressed as a function of $(\mathbf{x}, \boldsymbol{\lambda}, t)$

$$\mathbf{u} = \psi(\mathbf{x}, \boldsymbol{\lambda}, t). \tag{2.12}$$

3. Inserting the function (2.12) into the canonical equation (2.7) leads to the boundary value problem

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \psi(\mathbf{x}, \lambda)) \tag{2.13a}$$

$$\dot{\lambda} = -H_x(\mathbf{x}, \psi(\mathbf{x}, \lambda), \lambda) \tag{2.13b}$$

4. The numerical solution of the boundary value problem consists of $x^*(t), \lambda^*(t)$. The optimal control trajectory $\mathbf{u}^*(\mathbf{t})$ is obtained afterward from evaluating (2.12).

The minimization of $H(x, \lambda, u)$ can be split into the sub-problems with the affine-input system (1) and the integral cost(3)

$$\min_{u \in [u_i^-, u_i^+]} \frac{1}{2} r_i u_i^2 + \lambda^\top g_i(x) u_i \tag{2.14}$$

for which the minimizing control function is given by

$$u_i = h_i(x, \lambda) := \begin{cases} u_i^0 & \text{if } u_i^0 \in (u_i^-, u_i^+), \\ u_i^- & \text{if } u_i^0 \leq u_i^-, \\ u_i^+ & \text{if } u_i^0 \geq u_i^+ \end{cases} \tag{2.15}$$

with the unconstrained minimizer $u_i^0 = -\frac{1}{r_i} \lambda^\top g_i(x)^2$ and inserting it into the canonical BVP equations, two differential equations are obtained

$$\dot{x}_{k+1} = F(x_{k+1}, \lambda), x(0) = x_k \tag{2.16}$$

$$\dot{\lambda}_{k+1} = G(x_{k+1}, \lambda_{k+1}), \tag{2.17a}$$

$$\lambda_{k+1}(T) = V_x(x_{k+1}(T)) \tag{2.17b}$$

The equations (2.16) and (2.17) are a kind of boundary value problem. They have n initial conditions for n state variables $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and n terminal conditions for the adjoint states λ. In the following section by solving the differential equations (11) and (12), this feature can be exploited.

2.3 A fixed-point iteration scheme and implementation

Assumption: Assuming that $x(t)$ is a continuous trajectory in X for $t \in [0, T]$, equation (2.13a) and (2.13b) has a bounded solution. Specifically, there exists a compact set X_λ such that $\lambda(t) \in X_\lambda$ for all $t \in [0, T]$.

$x(0) = x_k$, a known variable, is an initial condition. And choosing the initial adjoint state condition $\bar{\lambda}_k^{(0)} \in X_\lambda$, $\tau \in [0, T]$. And ensure the iteration times j. Using the command ode4 to solve the equation (11) in the forward integration method with initial conditions within the horizon [0,T]. The numerical solution can be obtained in this kind of forward integration method. Then the numerical solution of the last sampling time is inserted into $\lambda(T) = V_x(x(T))$. $\lambda(T) = V_x(x(T))$ is a terminal condition. Then using the backward integration method can obtain the numerical solution of equation (2.13). At last two trajectories of state x and adjoint state λ are obtained.

(1) **Initialization:**

- choose $\bar{\lambda}_k^{(0)}(\tau) \in X_\lambda$, $\tau \in [0, T]$

(2) **Fixed-point iteration** $(1 \leq j \leq r)$

- compute $\bar{x}_k^{(j)}(\tau)$, $\tau \in [0, T]$ by forward integration of

$$\dot{\bar{x}}_k^{(j)}(\tau) = F\left(\bar{x}_k^{(j)}(\tau), \bar{\lambda}_k^{(j-1)}(\tau)\right) \tag{2.18}$$

  with initial condition $\bar{x}_k^{(j)}(0) = x_k$

- compute $\bar{\lambda}_k^{(j)}(\tau)$, $\tau \in [0, T]$ by backward integration of

$$\dot{\bar{\lambda}}_k^{(j)}(\tau) = G\left(\bar{x}_k^{(j)}(\tau), \bar{\lambda}_k^{(j)}(\tau)\right) \tag{2.19}$$

  with terminal condition $\bar{\lambda}_k^{(j)}(T) = V_x\left(\bar{x}_k^{(j)}(T)\right)$

- Stop if a suitable convergence criterion is fulfilled or if $j = r$. Otherwise set $j \leftarrow j+1$ and return to (2).

The fixed-point iteration consists of a forward integration and a backward integration per iteration. In practice, the horizon time T within [0,T] should be discrete. The number of discrete time points affects both the numerical accuracy as well as the computation time. In order to implement the algorithm within a real-time model predictive control framework, a fixed number of fixed-point iterations is considered in each model predictive control step. This strategy naturally leads to a sub-optimal optimal control problem solution. Therefore the investigation of stability requires additionally considering the evolution of the optimization error as a measure for the sub-optimal in each model predictive control step. [15]

2.4 Example

The optimal control problem with the state $\mathbf{x} = [x_1, x_2]\top$ is formulated as:

$$\begin{aligned}
\min_{u(\cdot)} \quad & \int_{t_0=0}^{t_f} \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} + \frac{1}{2}\mathbf{u}^\top \mathbf{R}\mathbf{u}\, dt \\
\text{s.t.} \quad & \dot{x}_1 = x_1 + x_2 + x_1 u, \\
& \dot{x}_2 = x_2 + x_2 u, \\
& x(0) = x_0, \\
& |u(t)| \leq 1 \quad \forall t \in [0, t_f].
\end{aligned}$$

$\lambda = [\lambda_1, \lambda_2]^\top$ With the Hamiltonian function:

$$H(x, u, \lambda) = \frac{1}{2}x^\top Qx + \frac{1}{2}u^\top Ru + \lambda_1(x_1 + x_2 + x_1 u) + \lambda_2(x_2 + x_2 u) \tag{2.15}$$

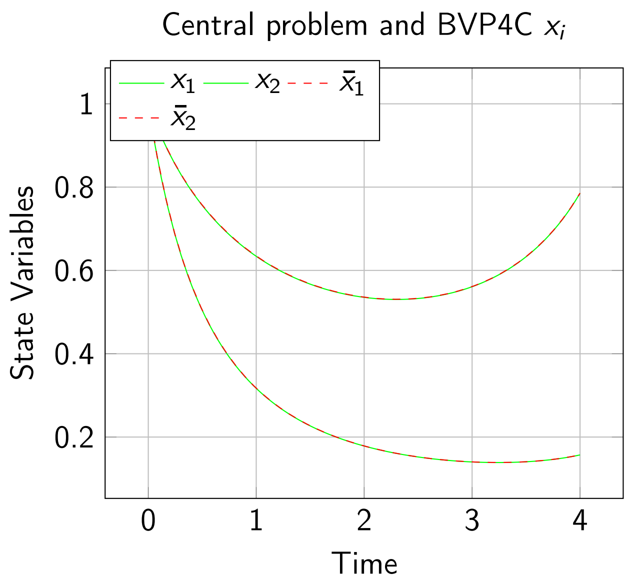
Q and R are weight matrices, Q=diag(1,1), R=1 inserting $u_i^0 = -\frac{1}{\mathbf{r_i}}\lambda^T g(x)$..

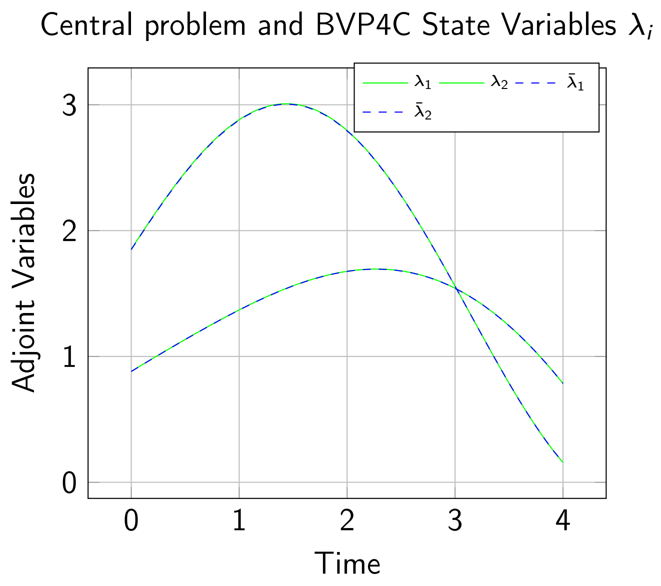
$$\dot{x} = \frac{\partial H}{\partial \lambda} = f(x) \tag{2.16}$$

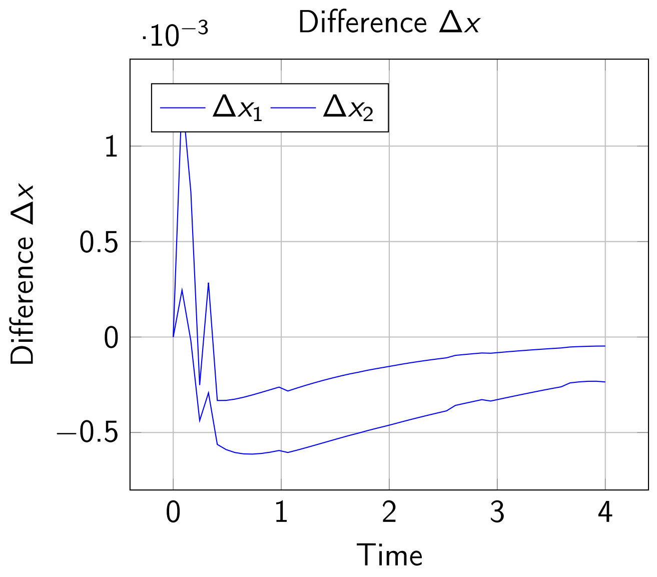
$$\dot{\lambda} = \frac{\partial H}{\partial x} = \frac{\partial l(x, u)}{\partial x} + [\frac{\partial f(x, u)}{\partial x}]^\top \lambda \tag{2.17}$$

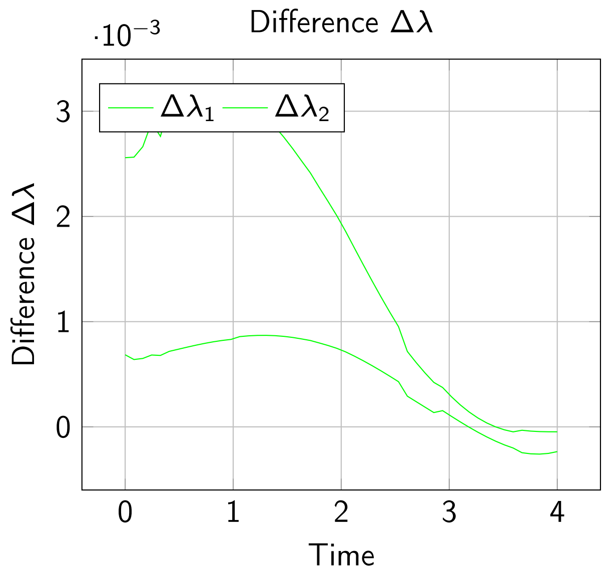
Before implementing the algorithm, the initial conditions of states $\mathbf{x}$ and adjoint states λ should be ensured. In addition, the number of iterations j is initialized. Setting $\mathbf{x_0} = [1, 1]^\top$, $\lambda = [1, 1 \ldots, 1]$. The dimension of λ depends on the number of sampling times. Then using the command ode4 can solve the differential function (2.16) in a forward integration way. The solution is trajectories of state variables of the horizon T. From the trajectories inserting the state of the last sampling time into $\lambda(T) = V_x(x(T))$ can obtain the last adjoint state and then using the 'ode4' function to solve the differential equation (2.17) as well but implement backward integration. The trajectories of state variables and adjoint state variables are obtained. In the next iteration, the initial condition x_0 does not change and it is used as the initial condition in the next iteration. However, the trajectory of the adjoint states is used as the new initial condition in the next iteration and then repeat the previous steps until the last iteration. The trajectories of the last iteration are the final results.

In addition, after I obtain the optimal trajectories, I want to verify the correctness of our result. So I can use the command "bvp4c" of Matlab to verify the result. Because this is a boundary problem, the command "bvp4c" can solve it and the results are trajectories that are same as the results of fixed-point iteration. The graphs of the results of the Fixed-point iteration method and "bvp4c" are almost same and shown:


Central problem and BVP4C x_i
x_1 x_2 $\bar{x}_1$ $\bar{x}_2$
State Variables
Time


Central problem and BVP4C State Variables λ_i
λ_1 λ_2 $\bar{\lambda}_1$ $\bar{\lambda}_2$
Adjoint Variables
Time


Difference Δx
$\cdot 10^{-3}$
Δx_1 Δx_2
Difference Δx
Time


Difference $\Delta\lambda$
$\cdot 10^{-3}$
$\Delta\lambda_1$ $\Delta\lambda_2$
Difference $\Delta\lambda$
Time

3 Distributed sensitivity-based optimization algorithm

In this chapter, the DMPC algorithm based on Sensitivity exchange is presented specifically.

3.1 Local OCP with central dynamics

Firstly, the general framework for describing multi-agent systems using a graph-theoretic approach is presented in the following part. Some conceptions are explained here.

Graph representation:
The multi-agent system is described by a graph $G = (\mathcal{V}, \mathcal{E})$, where: $\mathcal{V}$ is the set of vertices (or nodes), representing the agents. The set of vertices is labeled as $\mathcal{V} = \{1, \dots, N\}$, where N is the total number of agents. $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges (or connections), representing the interaction or relationship among the agents.

Adjacency Matrix:
The edges of the graph define the *adjacency matrix* A, which is a key representation in graph theory for capturing which agents are connected to others in the system.

Agents and Subsystems:
The vertices $i \in \mathcal{V}$ correspond to *agents*. And each of them represents a dynamic subsystem. These agents are the building blocks of the system, and their interactions define the system's behavior.

Neighborhood:
The neighborhood $\mathcal{N}_i$ of agent i is defined as the set of agents directly connected to agent i. Mathematically, this is written as:

$$\mathcal{N}_i = \{j \in \mathcal{V} : (j, i) \in \mathcal{E}, j \neq i\}$$

This means that agent i's neighborhood includes all other agents j that share an edge with i and are not i itself.

Agent Dynamics:
The agent dynamics are described by *nonlinear neighbor- and input-affine systems*. This means the behavior of each agent is affected by its neighbors (agents directly connected to

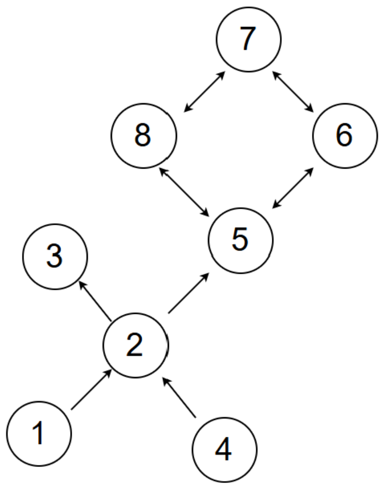


Figure 3.1: coupling agents model

it) and by the inputs applied to the system. These dynamics are crucial for defining how each agent responds to its local environment and inputs.

In this article, the agent dynamics are described in the following form, nonlinear neighbor and input-affine systems

$$\dot{\mathbf{x}}_i = \mathbf{f}_{ii}(\mathbf{x}_i) + \mathbf{G}_i(\mathbf{x}_i)\mathbf{u}_i + \sum_{j\in\mathcal{N}_i} \mathbf{f}_{ij}(\mathbf{x}_i, \mathbf{x}_j), \quad \mathbf{x}_i(0) = \mathbf{x}_{i,0} \tag{3.1}$$

$$=: \mathbf{f}_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}), \quad \mathbf{x}_i(0) = \mathbf{x}_{i,0}$$

$\mathbf{x}_i \in \mathbb{R}^{n_i}$ refers to the state and $\mathbf{u}_i \in \mathbb{R}^{m_i}$ refers to the input of agent $i \in \mathcal{V}$ as well as state $\mathbf{x}_j \in \mathbb{R}^{n_j}$ of neighbor $j \in \mathcal{N}_i$. It is assumed that the function $\mathbf{f}_{ii} : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$, $G_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i \times m_i}$ as well as $f_{ij} : \mathbb{R}^{n_i} \times \mathbb{R}^{n_j} \to \mathbb{R}^{n_i}$ are continuously differentiable in their arguments. The input is subject to the (vector-valued) point-wise-in-time box constraints $u_i \in [u_i^-, u_i^+]$.

The local optimal control problem is described in this form:

$$\min_{\mathbf{u}_\mathcal{V}} \quad \sum_{i\in\mathcal{V}} J_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}) \tag{3.2a}$$

$$\text{s.t.} \quad \dot{\mathbf{x}}_i = f_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}), \quad \mathbf{x}_i(0) = \mathbf{x}_{i,0}, \quad i \in \mathcal{V} \tag{3.2b}$$

$$\mathbf{u}_i \in [\mathbf{u}_i^-, \mathbf{u}_i^+], \quad i \in \mathcal{V}. \tag{3.2c}$$

The local cost function of (3.1)

$$J_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}) := V_i(\mathbf{x}_i(T)) + \int_0^T l_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i})\, dt \tag{3.3}$$

with state $\mathbf{x}^\top(t) = \left[\mathbf{x}_1^\top, \dots, \mathbf{x}_{|\mathcal{V}|}^\top\right] \in \mathbb{R}^n$, input $\mathbf{u}^\top(t) = \left[\mathbf{u}_1^\top, \dots, \mathbf{u}_{|\mathcal{V}|}^\top\right] \in \mathbb{R}^m$ and initial state $\mathbf{x}_0 \in \mathbb{R}^n$.Furthermore, $V_i(x_i(T))$ refers to the terminal cost, and $l_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i})$ denotes the integral cost. T represents the horizon time, while $\mathbf{x}_{\mathcal{N}_i}$ denotes the states of neighboring agents.

The integral cost $l_i(\mathbf{x_i}, \mathbf{u_i}, \mathbf{x}_{\mathcal{N}_i})$ is given in this kind of form:

$$l_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}) := l_{ii}(\mathbf{x}_i) + \frac{1}{2}\bar{\mathbf{u}}_i^\top \mathbf{R}_i \bar{\mathbf{u}}_i + \sum_{j \in \mathcal{N}_i} l_{ij}(\mathbf{x}_i, \mathbf{x}_j) \tag{3.4}$$

$l_{ii}(\mathbf{x}_i)$ represents the integral cost of the current agent. $\frac{1}{2}\bar{\mathbf{u}}_i^\top R_i \bar{\mathbf{u}}_i$ refers to time or energy optimality. The term 'energy optimality' for the quadratic control term $\frac{1}{2}\bar{\mathbf{u}}_i^\top R_i \bar{\mathbf{u}}_i$ stems from interpreting the control as an electrical signal. In addition, the quadratic term for the control $\bar{\mathbf{u}}_i := \mathbf{u}_i - \mathbf{u}_{i,ref}$. some reference $\mathbf{u}_{i,\text{ref}} \in \mathbb{R}^{m_i}$ and the diagonal positive-definite weighting matrix $\mathbf{R}_i \in \mathbb{R}^{m_i \times m_i}$. Moreover, the integral cost functions $l_{ii} : \mathbb{R}^{n_i} \to \mathbb{R}$, $l_{ij} : \mathbb{R}^{n_i} \times \mathbb{R}^{n_j} \to \mathbb{R}$ and terminal cost function $V_i : \mathbb{R}^{n_i} \to \mathbb{R}$ are assumed to be **continuously differentiable** in their arguments. The optimal control problems (3.2) is solved repeatedly at each sampling time within a finite time horizon T. The result of the optimal control problem $u^*(t)$ is a trajectory, which is applied to the system for one sampling step. In the next sampling time, a new measurement is obtained. The horizon is shifted and (3.2) are solved again. [16]

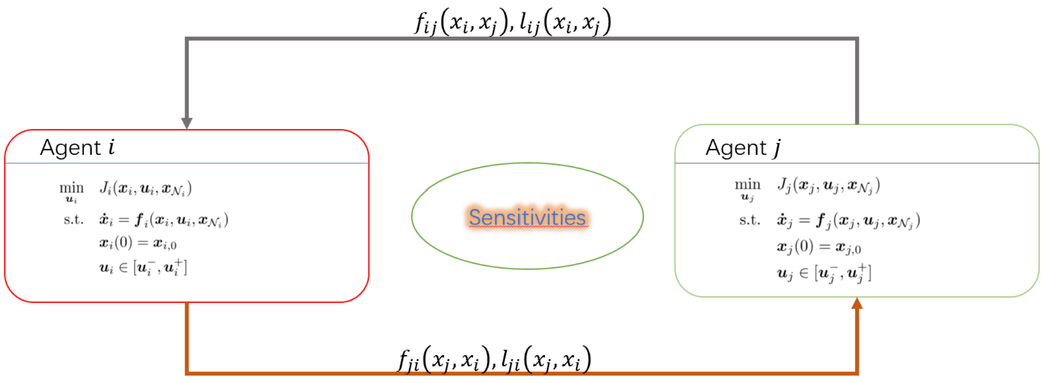

$f_{ij}(x_i, x_j), l_{ij}(x_i, x_j)$
Agent i
$\min_{u_i} \; J_i(x_i, u_i, x_{\mathcal{N}_i})$
s.t. $\dot{x}_i = f_i(x_i, u_i, x_{\mathcal{N}_i})$
$x_i(0) = x_{i,0}$
$u_i \in [u_i^-, u_i^+]$
Sensitivities
Agent j
$\min_{u_j} \; J_j(x_j, u_j, x_{\mathcal{N}_j})$
s.t. $\dot{x}_j = f_j(x_j, u_j, x_{\mathcal{N}_j})$
$x_j(0) = x_{j,0}$
$u_j \in [u_j^-, u_j^+]$
$f_{ji}(x_j, x_i), l_{ji}(x_j, x_i)$

Figure 3.2: Sensitivity exchange.

3.2 Sensitivity calculation

The goal of distributed model predictive control is to solve (3.1) in a distributed fashion. Sensitivities account for the information about the expected change in the cost of a specific subsystem w.r.t the agent's own state and control trajectories. The local problems to be solved on the agent level with sensitivities are the following optimal control problems in iteration k. The local problem of agent $i \in \mathcal{V}$ reads

$$\begin{aligned}
\min_{\mathbf{u}_i} \quad & \bar{\mathbf{J}}_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}^{k-1}) := \mathbf{J}_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}^{k-1}) \\
& + \sum_{j \in \mathcal{N}_i} \delta \bar{\mathbf{J}}_j(\mathbf{x}_j^{k-1}, \mathbf{u}_j^{k-1}, \mathbf{x}_{\mathcal{N}_j}^{k-1})(\delta \mathbf{x}_i) && (3.5a) \\
\text{s.t.} \quad & \dot{\mathbf{x}}_i = \mathbf{f}_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}^{k-1}), \quad \mathbf{x}_i(0) = \mathbf{x}_{i,0} && (3.5b) \\
& \mathbf{u}_i \in [\mathbf{u}_i^-, \mathbf{u}_i^+] && (3.5c)
\end{aligned}$$

with the cost sensitivity of neighbors represented by the corresponding Gâteaux derivative $\delta \bar{J}_j$ in direction of $\delta \mathbf{x}_i := \mathbf{x}_i - \mathbf{x}_i^{k-1}$. After finishing the calculation of all agents about trajectories of states and adjoint states, the result should be communicated by each agent. These communicated trajectories are the initial conditions of the next iteration $k+1$. In addition, these trajectories at iteration k are used to decouple the dynamics (3.5a) and cost terms (3.5b). The first-order optimal conditions for each agent i consist of the canonical boundary value problem (3.6)-(3.7):

$$\dot{\mathbf{x}}_i = \mathbf{f}_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}^{k-1}), \quad \mathbf{x}_i(0) = \mathbf{x}_{i,0} \tag{3.6}$$

$$\dot{\lambda}_i = -\partial_{\mathbf{x}_i} \mathbf{H}_i(\mathbf{x}_i, \mathbf{u}_i, \lambda_i), \quad \lambda_i(T) = \lambda_{i,T} \tag{3.7}$$

with the adjoint state $\lambda_i \in \mathbb{R}^{n_i}$ and the terminal value $\lambda_{i,T} := \partial_{\mathbf{x}_i} V_i(\mathbf{x}_i(T))$ and the minimization of the local Hamiltonian

$$\begin{aligned} H_i(\mathbf{x}_i, \mathbf{u}_i, \lambda_i) &:= l_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}^{k-1}) + \lambda_i^\top f_i(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{\mathcal{N}_i}^{k-1}) \\ &\quad + \sum_{j \in \mathcal{N}_i} \left(\partial_{x_i} l_{ji}^{k-1} + \left(\partial_{\mathbf{x}_i} \mathbf{f}_{ji}^{k-1} \right)^\top \lambda_j^{k-1} \right)^\top \delta \mathbf{x}_i \\ &\quad \left(\partial_{x_i} l_{ji}^{k-1} + \left(\partial_{\mathbf{x}_i} \mathbf{f}_{ji}^{k-1} \right)^\top \lambda_j^{k-1} \right) =: g_{ji}^{k-1}(t) \end{aligned} \tag{3.8}$$

The calculation of sensitivity begins with **First variation / Gâteaux derivative** [14]

Definition 3.1 (Gâteaux directional derivative).

$$\delta J_j(\mathbf{x}_j, \mathbf{u}_j, \mathbf{x}_{\mathcal{N}_j})(\delta \mathbf{x}_i) = \left. \frac{\mathrm{d} J_j(\mathbf{x}_j, \mathbf{u}_j, \mathbf{x}_{\mathcal{N}_j} + \epsilon \delta \mathbf{x}_i)}{\mathrm{d}\epsilon} \right|_{\epsilon=0} \tag{3.9}$$

If $\delta \mathcal{J}(\mathbf{x}; \mathbf{y})$ exists for all $\mathbf{y} \in \mathcal{X}$, then $\mathcal{J}$ is Gâteaux differentiable at point $\mathbf{x} \in \mathcal{X}$.

In addtion, the existence of the Gâteaux derivative $\delta \mathcal{J}(\mathbf{x}; \mathbf{y})$ requires that the functionals $\mathcal{J}(\mathbf{x})$ and $\mathcal{J}(\mathbf{x} + \epsilon \mathbf{y})$ are defined for all sufficiently small ϵ and that the derivative of $\mathcal{J}(\mathbf{x} + \epsilon \mathbf{y})$ w.r.t. ϵ must exist. The following example illustrates the necessity behind this.

Example 3.1 For the functional $\mathcal{J}(x) = \int_{t_0}^{t_f} x(t)^2 + x(t)\, dt$ with $x \in C^1[t_0, t_f]$, the Gâteaux derivative in direction $y \in C^1[t_0, t_f]$ is

$$\begin{aligned} \delta \mathcal{J}(x; y) &= \lim_{\epsilon \to 0} \frac{\mathcal{J}(x + \epsilon y) - \mathcal{J}(x)}{\epsilon} \\ &= \lim_{\epsilon \to 0} \left[\frac{1}{\epsilon} \int_{t_0}^{t_f} \left((x(t) + \epsilon y(t))^2 + x(t) + \epsilon y(t) - x(t)^2 - x(t) \right) dt \right] \\ &= \lim_{\epsilon \to 0} \int_{t_0}^{t_f} \left[2x(t)y(t) + y(t) + \epsilon y(t)^2 \right] dt \\ &= \int_{t_0}^{t_f} \left[2x(t)y(t) + y(t) \right] dt. \end{aligned}$$

Definition as the Gâteaux directional derivative

$$\delta J_j(\mathbf{x}_j, \mathbf{u}_j, \mathbf{x}_{\mathcal{N}_j})(\delta \mathbf{x}_i) = \left. \frac{\mathrm{d} J_j(\mathbf{x}_j, \mathbf{u}_j, \mathbf{x}_{\mathcal{N}_j} + \epsilon \delta \mathbf{x}_i)}{\mathrm{d}\epsilon} \right|_{\epsilon=0} \tag{3.10}$$

with direction vector $\delta \bar{\mathbf{x}}_i = [0^\top \dots 0^\top \delta \mathbf{x}_i^\top 0^\top \dots 0^\top]^\top$

$$\delta J_j(x_j, u_j, x_{\mathcal{N}_j})(\delta x_i) = \int_0^T \left(\partial_{x_i} l_{ji}(x_j, x_i) + (\partial_{x_i} f_{ji}(x_j, x_i))^\top \lambda_j\right)^\top \delta x_i \, dt \tag{3.11a}$$

$$= \int_0^T g_{ji}^\top \delta x_i \, dt \tag{3.11b}$$

Algorithm 1 Distributed sensitivity-based solution of OCP (2)

```
   Initialize & send $u_i^0(t)$, $x_i^0(t) = x_{i,0}$, $\lambda_i^0(t) = \partial_{x_i} V_i(x_{i,0})$ to neighborhood $\mathcal{N}_i$, set
   $k \leftarrow 1$, and choose $k_{\max}$
2: while $k \leq k_{\max}$ do
       Compute $g_{ji}^{k-1}$ locally for all neighbors $j \in \mathcal{N}_i$
4:     Compute $(u_i^k, x_i^k, \lambda_i^k)$ locally by solving (3.6) and (3.7)
       Send trajectories $(u_i^k, x_i^k, \lambda_i^k)$ to all neighbors $j \in \mathcal{N}_i$
6:     Set $k \leftarrow k + 1$
   end while
```

1. **Initialization:**

   - For each node i, sending the initial control input $u_i^0(t)$, the initial state $x_i^0(t) = x_{i,0}$, and the initial value of the adjoint variable $\lambda_i^0(t) = \partial_{x_i} V_i(x_{i,0})$ to the neighboring nodes in set $\mathcal{N}_i$.
   - Set the iteration index $k \leftarrow 1$ and choose the maximum number of iterations $k_{\max}$.
   - determine the maximum iteration times k_{max}

2. **Start the loop (While loop):** Iterate until the maximum number of iterations $k_{\max}$ is reached.

   - **Step 2: Compute the gradient from neighbors:**
     - Each node i must compute the gradient g_{ji}^{k-1} for all neighboring nodes $j \in \mathcal{N}_i$. This step is based on the information received from the neighboring nodes and ensures that each node has sufficient information about its neighbors.
   - **Step 3: Compute the local control input, states, and adjoint variables:**
     - Based on steps (3.6) and (3.7), each node computes the control input u_i^k, the state x_i^k, and the adjoint variable λ_i^k with fixed-point iteration scheme mentioned in last chapter. The form of these computations is dependent on the dynamics and cost functions of the system.
   - **Step 4: Send information:**

  - Each node sends the current control input u_i^k, the state x_i^k, and the adjoint variable λ_i^k to all its neighbors $j \in \mathcal{N}_i$, enabling the neighboring nodes to update their information.

- **Step 5: Increase the iteration index:**
  - Increment the iteration index $k \leftarrow k+1$ and continue to the next iteration.

3. **Termination:**
   - When the maximum number of iterations $k_{\max}$ is reached, the algorithm terminates.

In each iteration k of algorithm 2, the optimal control problem must be solved with the result of the last iteration and the communicated information among neighbors. In addition, the minimization problem is strictly convex and is separable in the single elements of u_i. The control u_i is computed as $h_i(x_i, \lambda_i)$, which projects the unconstrained minimize $\hat{u}_i$ based on certain bounds:

- $u_i = u_i^-$ if $\hat{u}_i \leq u_i^-$ (lower bound).
- $u_i = u_i^+$ if $\hat{u}_i \geq u_i^+$ (upper bound).
- $u_i = \hat{u}_i$ if $\hat{u}_i \in (u_i^-, u_i^+)$ (within bounds).

This formula ensures that the control is within the specified bounds. The unconstrained control $\hat{u}_i$ is calculated using the equation:

$$\hat{\mathbf{u}}_i = \mathbf{u}_{i,\text{ref}} - \mathbf{R}_i^{-1}\mathbf{G}_i(x_i)^\top \lambda_i \tag{3.12}$$

Here, $\mathbf{u}_{i,\text{ref}}$ is a reference value, $\mathbf{R}_i^{-1}$ represents the inverse of a matrix $\mathbf{R}_i$, and $\mathbf{G}_i(x_i)^\top \lambda_i$ is a term based on the system's dynamics and adjoint variables. The unconstrained minimize $\hat{u}_i$ is found by solving the stationary condition:

$$\partial H_i(x_i, u_i, \lambda_i) = 0$$

This condition ensures that the Hamiltonian function H_i is minimized concerning u_i. With the unconstrained minimizer $\hat{u}_i$ and inserting (3.12) into canonical equations (3.6) and (3.7) yields:

$$\dot{x}_i = F_i(x_i, \lambda_i, x_{\mathcal{N}_i}^{k-1}), \quad x_i(0) = x_{i,0} \tag{3.13a}$$

$$\dot{\lambda}_i = H_i(x_i, \lambda_i, x_i^{k-1}, x_{\mathcal{N}_i}^{k-1}, \lambda_i^{k-1}), \quad \lambda_i(T) = \lambda_{i,T} \tag{3.13b}$$

with the new local functions

$$F_i(x_i, \lambda_i, x_{\mathcal{N}_i}^{k-1}) := f_i(x_i, h_i(x_i, \lambda_i), x_{\mathcal{N}_i}^{k-1}) \tag{3.14}$$

and

$$H_i(x_i, \lambda_i, x_i^{k-1}, x_{\mathcal{N}_i}^{k-1}, \lambda_i^{k-1}) := -\partial_{x_i} H_i(x_i, h_i(x_i, \lambda_i), \lambda_i) \tag{3.15}$$

by slight abuse of notations. Note that the dependencies on external trajectories of neighbors in the previous iteration are explicitly captured in the functions $\mathbf{F}_i$ and $\mathbf{H}_i$.

Algorithm 2 **Solution of OCP (3.2) by local fixed-point iteration**

```
Initialize $\boldsymbol{\lambda}_i^{0|k} = \boldsymbol{\lambda}_i^{k-1}$, set $j \leftarrow 1$ and choose $j_{\max} =: r$
1: while $j \leq r$ do
2:     Compute $x_i^{j|k}(t)$ via forward integration of
           $$\dot{\mathbf{x}}_i^{j|k} = \mathbf{F}_i(\mathbf{x}_i^{j|k}, \boldsymbol{\lambda}_i^{j-1|k}, \mathbf{x}_{\mathcal{N}_i}^{k-1})$$
       with initial condition $x_i^{j|k}(0) = x_{i,0}$
3:     Compute $\boldsymbol{\lambda}_i^{j|k}(t)$ via backward integration of
           $$\dot{\boldsymbol{\lambda}}_i^{j|k} = \mathbf{H}_i(x_i^{j|k}, \boldsymbol{\lambda}_i^{j|k}, \mathbf{x}_{\mathcal{N}_i}^{k-1}, \boldsymbol{\lambda}_{\mathcal{N}_i}^{k-1})$$
       with terminal condition $\boldsymbol{\lambda}_i^{j|k}(T) = \partial_{\mathbf{x}_i} V_i(\mathbf{x}_i^{j|k}(T))$
4:     Set $j \leftarrow j + 1$
5: end while
6: Compute $u_i^k = \mathbf{h}_i(\mathbf{x}_i^{r|k}, \boldsymbol{\lambda}_i^{r|k})$, set $\mathbf{x}_i^k = \mathbf{x}_i^{r|k}$, $\boldsymbol{\lambda}_i^k = \boldsymbol{\lambda}_i^{r|k}$ and return to Step 4 of
   Algorithm 1
```

From the algorithm 2, the process described applies r iterations to solve the optimal conditions of the local Optimal Control Problem (OCP) (3.2a)-(3.2c). After that, the calculated trajectories are sent to the respectively all neighbors in step 4 of Algorithm 1, as follows:

- The state $\mathbf{x}_i^{r|k}$ and adjoint-state $\boldsymbol{\lambda}_i^{r|k}$ become the current state and adjoint-state for the next optimization iteration: $\mathbf{x}_i^k := \mathbf{x}_i^{r|k}$ and $\boldsymbol{\lambda}_i^k := \boldsymbol{\lambda}_i^{r|k}$.

For the next optimization step $k + 1$, Algorithm 2 is re-initialized with:

$$\boldsymbol{\lambda}_i^{0|k+1} = \boldsymbol{\lambda}_i^k$$

This means that the adjoint state for the next iteration is set to the result from the previous one, continuing the iterative process. The equation ensures that the optimization progresses smoothly across iterations.

Analysis of Convergence Properties

This section analyzes the convergence properties of the fixed-point iteration scheme used in Algorithm 2, which is crucial for ensuring the overall convergence of Algorithm 1.

Key points of the analysis:

- The **convergence** of Algorithm 2 is studied as a prerequisite for understanding how Algorithm 1 converges. This is because the iterations of Algorithm 2 feed into the larger scheme used by Algorithm 1.

- The length of the **horizon** T plays a significant role in ensuring the convergence of the fixed-point iterations. In this case, there is a critical upper bound $\overline{T}$, above which convergence is not guaranteed.

- The **theorem** presented states that when the horizon length T is below this upper bound $\overline{T}$, Algorithm 2 is **contracting**. This means that the iterations bring the state and co-state variables progressively closer to the solution.

- There is a **contraction factor** $p \in [0, 1)$ that ensures that with each iteration, the difference between consecutive solutions (for both the state x_i and co-state λ_i) is reduced by a factor of p. This reduction is represented in the norms of the differences of state and adjoint-state values:

$$||\Delta x_i^{j|k}||_\infty \leq p||\Delta x_i^{j-1|k}||_\infty, \quad j = 2, 3, \dots \tag{3.16a}$$

$$||\Delta \lambda_i^{j|k}||_\infty \leq p||\Delta \lambda_i^{j-1|k}||_\infty, \quad j = 2, 3, \dots \tag{3.16b}$$

  Here, the term $\Delta x_i^{j|k}$ represents the difference between the state variables in two consecutive iterations $\Delta x_i^{j|k} := x_i^{j|k} - x_i^{j-1|k}$, while $\Delta \lambda_i^{j|k}$ is the difference between the co-state variables $\Delta \lambda_i^{j|k} := \lambda_i^{j|k} - \lambda_i^{j-1|k}$. The result indicates that these differences diminish over successive iterations, leading to convergence.

Comments:

- **Assumption**:

  The theorem assumes continuous differentiability of the system dynamics and cost functions, which is a standard assumption for ensuring smooth and predictable behavior in optimization and control problems.

- **Maximum prediction horizon**:

  The maximum prediction horizon $\bar{T}$ depends on the system characteristics and costs. This means that the larger the prediction horizon, the more difficult it is to ensure convergence, as system complexity and cost structures can affect stability.

- **Constructive approach**:

  The constructive approach mentioned is often conservative for design purposes. In practice, this means that while the theoretical bounds may ensure convergence, they might not be optimal for all scenarios, and more aggressive design choices might work without violating stability.

- **Sufficient prediction horizon:**

  A sufficiently small prediction horizon T can always be found to ensure convergence. This indicates that even if the system is complex, by reducing the prediction horizon, the algorithm can still guarantee convergence.

This analysis ensures that under the right conditions (specifically when $T \leq \overline{T}$), the fixed-point iteration scheme in Algorithm 2 is stable and will converge toward the solution.

3.3 Algorithmic extension

3.3.1 Anderson Acceleration

In mathematics, Anderson acceleration, also called Anderson mixing, is a method for the acceleration of the convergence rate of fixed-point iterations. It was introduced by Donald G.Anderson [17]. This algorithm can be used to find the solution to fixed point equations $f(x) = x$ often arising in the field of computational science. I use the algorithm. The goal is to make the convergence of outer iteration occur earlier. And in the original algorithm if I increase the horizon time T, at a certain threshold the system divergence and the model predictive controller can not work anymore. However after the Anderson acceleration algorithm is implemented in the original algorithm, the threshold rises. That means the optimized model predictive controller has a larger horizon time than before.

Now given a function $f : \mathbb{R}^n \to \mathbb{R}^n$, consider the problem of finding a *fixed point* of f, which is a solution to the equation $f(x) = x$. A classical approach to the problem is to employ a fixed-point iteration scheme; that is, given an initial guess x_0 for the solution, to compute the sequence $x_{i+1} = f(x_i)$ until some convergence criterion is met. However, the convergence of such a scheme is not guaranteed in general [18]; moreover, the rate of convergence is usually linear, which can become too slow if the evaluation of the function f is computationally expensive. Anderson acceleration is a method to accelerate the convergence of the fixed-point sequence.

Now define the residual $g(x) = f(x) - x$, and denote $f_k = f(x_k)$ and $g_k = g(x_k)$ (where g_k corresponds to the sequence of iterates from the previous paragraph). Given an initial guess x_0 and an integer parameter $m \geq 1$, the method can be formulated as follows:

$$
\begin{gathered}
x_1 = f(x_0) \\
\forall k = 1, 2, \ldots \\
m_k = \min\{m, k\} \\
G_k = [g_{k-m_k} \ \ldots \ g_k] \\
\alpha_k = \arg\min_{\alpha \in A_k} \|G_k \alpha\|_2, \quad \text{where} \quad A_k = \left\{ \alpha = (\alpha_0, \ldots, \alpha_{m_k}) \in \mathbb{R}^{m_k+1} : \sum_{i=0}^{m_k} \alpha_i = 1 \right\} \\
x_{k+1} = \sum_{i=0}^{m_k} (\alpha_k)_i f_{k-m_k+i}
\end{gathered}
\tag{3.17}
$$

where the matrix-vector multiplication $G_k \alpha = \sum_{i=0}^{m_k} (\alpha)_i g_{k-m_k+i}$, and $(\alpha)_i$ is the i-th element of α. Conventional stopping criteria are used to end the implementation of iterations in the algorithm. For example, implementation of iteration stops when $\|x_{k+1} - x_k\|$ falls under a prescribed tolerance ϵ, or when the residual $g(x_k)$ falls under a prescribed tolerance ϵ.

With respect to the standard fixed-point iteration, the method has been found to converge faster and be more robust, and in some cases avoid the divergence of the fixed-point sequence. I will present the property in the next chapter Numerical analysis with an explicit example specifically. When the algorithm optimized with Anderson acceleration is implemented, the result converges with less iteration amount.

Choice of m: The parameter m determines how much information from previous iterations is used to compute the new iteration x_{k+1} [19]. On the one hand, if m is chosen to be too small, too little information is used and convergence may be undesirably slow. On the other hand, if m is too large, information from old iterations may be retained for too many subsequent iterations, so that again convergence may be slow. Moreover, the choice of m affects the size of the optimization problem. A too-large value of m may worsen the conditioning of the least squares problem and the cost of its solution. In general, the particular problem to be solved determines the best choice of the m parameter.

Choice of m_k: With respect to the algorithm described above, the choice of m_k at each iteration can be modified. One possibility is to choose $m_k = k$ for each iteration k (sometimes referred to as Anderson acceleration without truncation). This way, every new iteration x_{k+1} is computed using all the previously computed iterations. A more sophisticated technique is based on choosing m_k so as to maintain a small enough conditioning for the least-squares problem. [20]

By standard fixed-point theory, there exists a unique $x^* \in X$ such that $G(x^*) = x^*$. We will make specific choices for G and X, we discuss the acceleration algorithm in this form to emphasize its more general applicability.

3.3.2 Implementation of Anderson acceleration

Firstly the original algorithm is implemented and all trajectories of state variables $\mathbf{x}_i$ and adjoint variables λ_i are obtained. These trajectories are a function of t and are assigned to vectors. The initial conditions state variable x_0 and the adjoint state $\boldsymbol{\lambda_0}$ have been obtained from these vectors. In addition, the tolerance parameter needs to be initialized as well. Then using these trajectories in the first iteration , $\bar{\mathbf{x}}_\mathbf{1} = G(\mathbf{x}_0)$ is obtained as well and then assigning the $\bar{\mathbf{x}}_\mathbf{1}$ to $\mathbf{x}_1$. After that initialize parameters k = 1 and m. How to choose m has been explained in the above part. Then assign the trajectories of the second iteration to $\bar{\mathbf{x}}_\mathbf{2}$. After obtaining $\bar{\mathbf{x}}_\mathbf{2}$, the formula about a minimization problem for $\aleph_j^{k+1}$ can be obtained as following :

In Matlab, the command "fmincon" can solve such a problem.
Finds the minimum of a problem specified by

$$\min_x f(x) \quad \text{such that} \quad \begin{cases} c(x) \leq 0 \\ c_{eq}(x) = 0 \\ A \cdot x \leq b \\ A_{eq} \cdot x = b_{eq} \\ l_b \leq x \leq u_b \end{cases} \tag{3.18}$$

where b and b_{eq} are vectors, A and A_{eq} are matrices, $c(x)$ and $c_{eq}(x)$ are functions that return vectors, and $f(x)$ is a function that returns a scalar. $f(x)$, $c(x)$, and $c_{eq}(x)$ can be nonlinear functions.

$x = fmincon(fun, x0, A, b, Aeq, beq)$ minimizes the function subject to the linear equalities $Aeq * x = beq$ and inequality $A * x \leq b$. If no inequalities exist, set $A = []$ and $b = []$. When we use the command "fmincon", we have to initialize the condition x_0. The result of this minimization problem is vector about $\aleph = [\aleph_1, \aleph_2, \ldots, \aleph_{k-m_k}]$. With the result $\aleph$, x_{k+1} is obtained by implementing step c. Assigning $k = k + 1$ and repeating the above steps until the last time iteration or $x_{k+1} - x_k \leq tolerance$. After that the above steps need to be repeated until the the difference between the actual result and the last result $u_k - u_{k-1} \leq tolerance$ For the adjoint variables, the process of implementing fixed-point iteration can be accelerated with these steps as well.

Algorithm 3 **Implementation steps of Anderson acceleration**

```
Initialize Choose $\boldsymbol{x}_0 \in \boldsymbol{X}$
1: Find $\bar{\boldsymbol{x}} \in \boldsymbol{X}$ such that $\bar{\boldsymbol{x}} = \boldsymbol{G}(\boldsymbol{u}_0)$. Set $\boldsymbol{x}_1 = \bar{\boldsymbol{u}}_1$
2: For $k+1 = 1, 2, 3, \ldots$, set $m_k = \min\{k, m\}$.
    (a) Find $\tilde{u}_{k+1} = G(u_k)$.
    (b) Solve the minimization problem for $\{\alpha_j^{k+1}\}_{j=k-m_k}^{k}$:
        $$\min_{\sum_{j=k-m_k}^{k} \alpha_j^{k+1}=1} \left\| \sum_{j=k-m_k}^{k} \alpha_j^{k+1}(\tilde{u}_{j+1} - u_j) \right\|_* .$$
    (c) Set:
        $$u_{k+1} = \sum_{j=k-m_k}^{k} \alpha_j^{k+1} \tilde{u}_{j+1}.$$
```

4 Numerical evaluation

4.1 Examples

In this chapter, I use four examples. The first example aims to solve a more complicated coupling case. For this example, the dynamic model is more complicated with more subsystems, coupling relations, states, and adjoint states. The goal of the second example is to solve the cost function coupling case. The third example verifies the single coupling direction. The last example has a more complicated coupling function. for these examples, four evaluations are done. The results of central problems subtract from the results of the sensitivity-based distributed model predictive control method respectively. At last, the convergences are presented in these figures. It means the differences disappear.

Example 1 This is the first example. The coupling way is shown in figure 4.1.

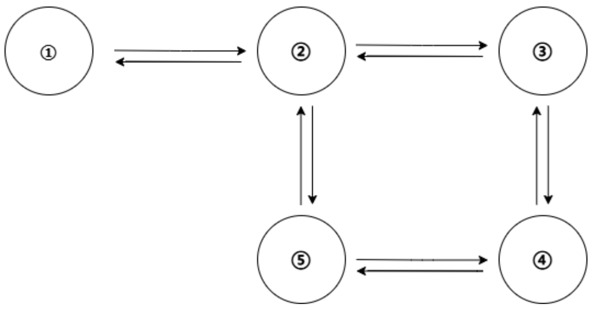
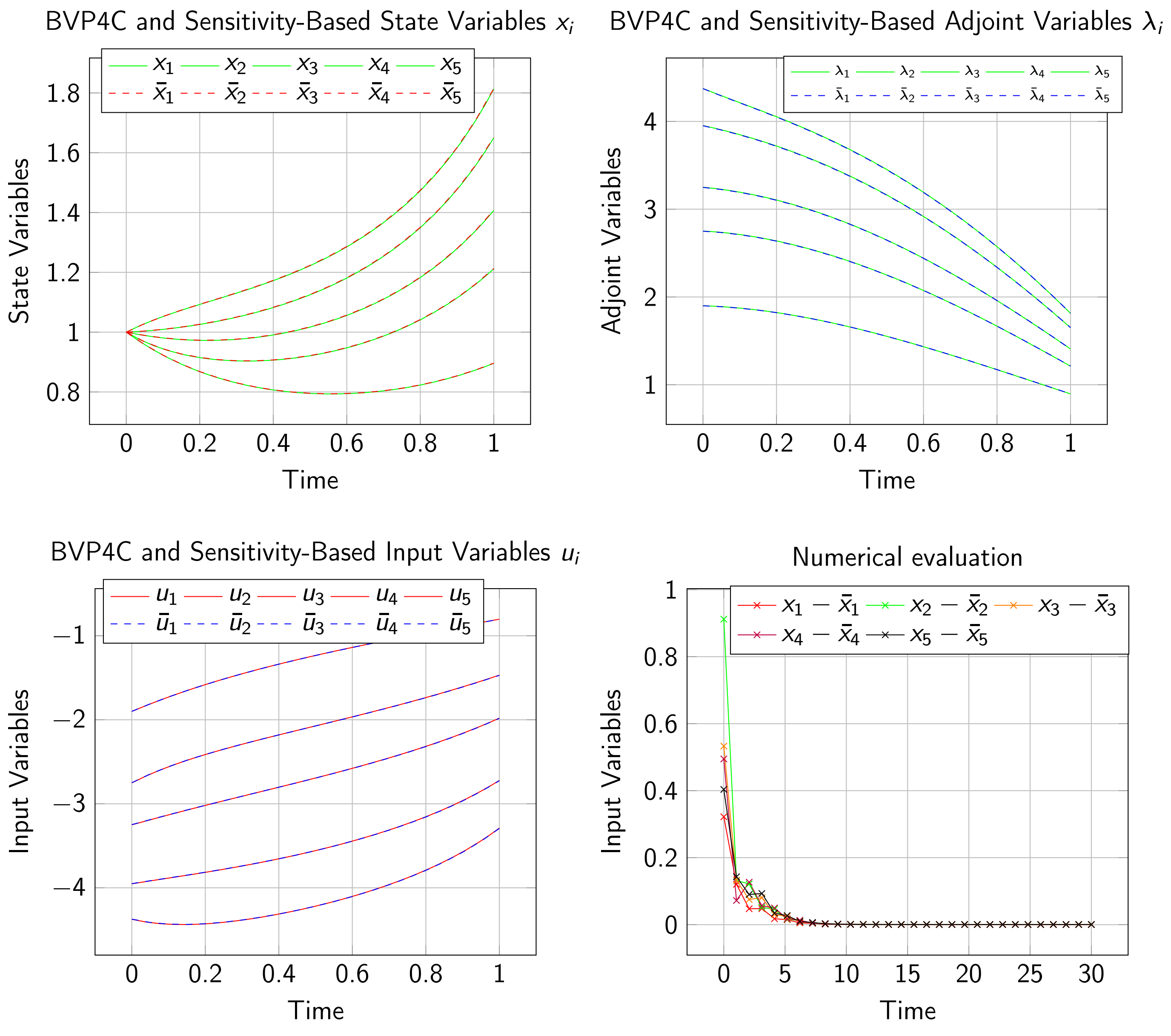


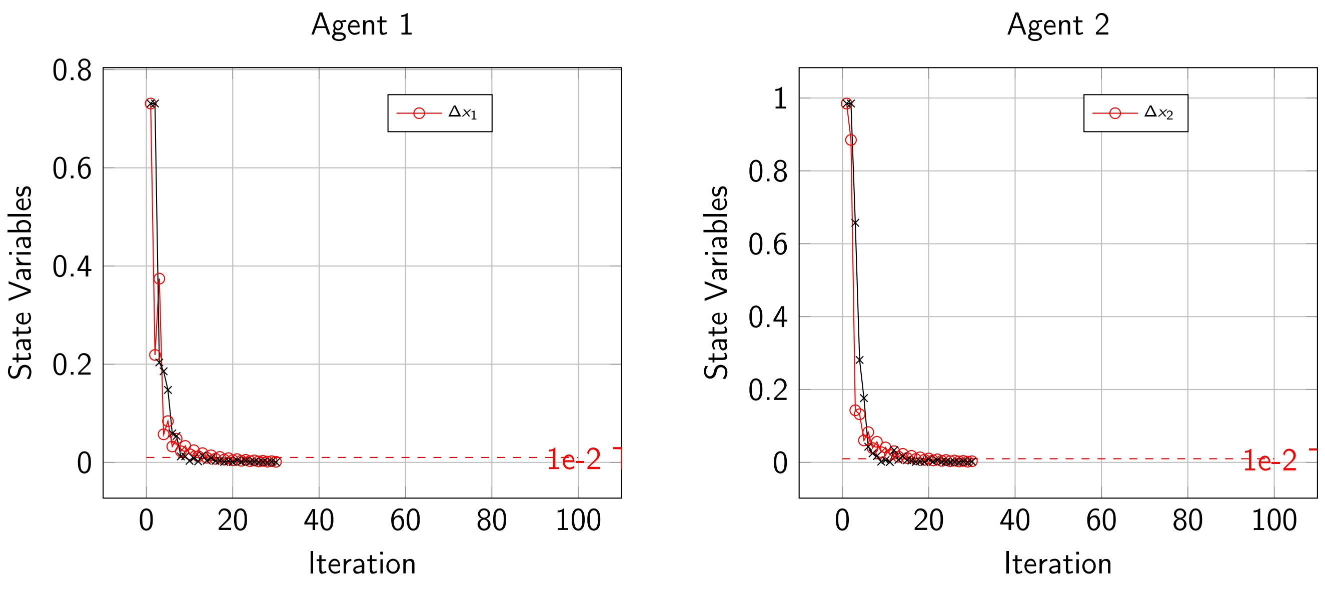
Figure 4.1: example 1

$$\dot{x} = \begin{bmatrix} x_1 + x_1 u_1 + x_2 - x_1 \\ 2x_2 + u_2 x_2 + x_1 - x_2 + x_3 - x_2 + x_5 - x_2 \\ 3x_3 + u_3 x_3 + x_2 - x_3 + x_4 - x_3 \\ 4x_4 + u_4 x_4 + x_3 - x_4 + x_5 - x_4 \\ 5x_5 + u_5 x_5 + x_2 - x_5 + x_4 - x_5 \end{bmatrix} \tag{4.1}$$

The equation (4.1) is a mathematical dynamic model of a central system. It is divided into five agents. Each agent has a mathematical dynamic model in the form of an equation (3.1). For agent 1, $x_2 - x_1$ is the coupling term with agent 2. Agent 2 has 3 coupling agents. $x_1 - x_2$ is the coupling term with agent 1, $x_3 - x_2$ is the coupling term with agent 3, and $x_5 - x_2$ is the coupling term with agent 5. For agent 3 $x_3 - x_4$ and $x_5 - x_4$ are the coupling terms with agent 2 and agent 4 respectively. For agent 4 and agent 5 their coupling terms include $x_3 - x_4$, $x_5 - x_4$ $x_2 - x_5$, and $x_4 - x_5$. The trajectories of state variables, adjoint variables, and inputs based on the sensitivity method and 'BVP4C' in solid line and dashed line are shown in the same plots respectively:



Now I optimize the original algorithm with Anderson Acceleration. I set the parameter m = 3. In the above part, I have mentioned how to choose the parameter m. In addition, I set the tolerance on the residual $\epsilon = 1e-2$. Before I optimize the algorithm, the convergences occur after the 14th, 17th, 18th, 19th, and 20th iterations respectively for five agents. But after the algorithm is optimized, the convergences occur after the 10th, 9th, 10th, 9th, and 10th respectively. For each agent, the convergence occurs earlier. That means the convergence rates are higher than before. And it saves a lot of computational time for the system. At the same time, it reduces the computational burden for the system. For a system with very complicated dynamic equations, the optimization has significant sense.



Example 2 This is the second example. The coupling way is presented in Figure 4.4. In this example, the coupling way is the cost function. It does not have any state variables $\sum_{j\in\mathcal{N}_i} \mathbf{f}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ coupling terms, but cost coupling terms $\sum_{j\in\mathcal{N}_i} \mathbf{l}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$. For agent 1 the cost coupling term with agent 2 is $l_{12} = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2$. For agent 2 the cost coupling terms with agent 1 and 3 are $l_{21} = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2$ $l_{23} = \frac{1}{2}x_2^2 + \frac{1}{2}x_3^2$ respectively. For agent 3 the cost coupling term with agent 2 is $l_{32} = \frac{1}{2}x_2^2 + \frac{1}{2}x_3^2$.

$$\dot{x} = \begin{bmatrix} x_1 + x_1 u_1 \\ 2x_2 + u_2 x_2 \\ 3x_3 + u_3 x_3 \end{bmatrix} \tag{4.2}$$

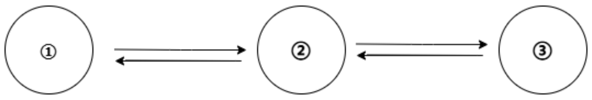
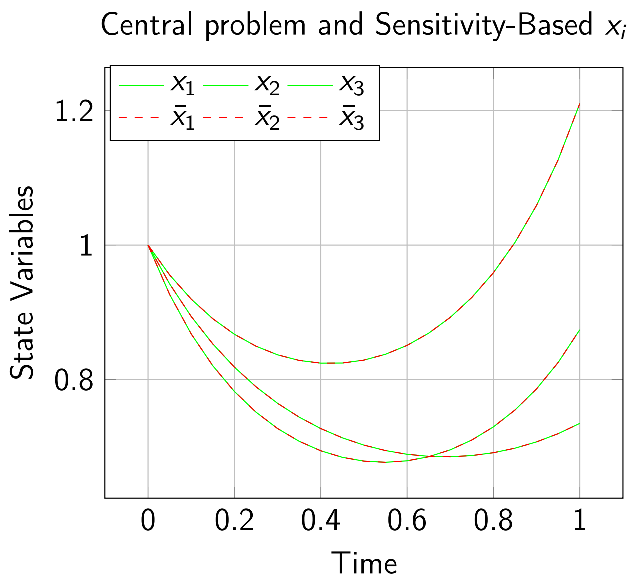
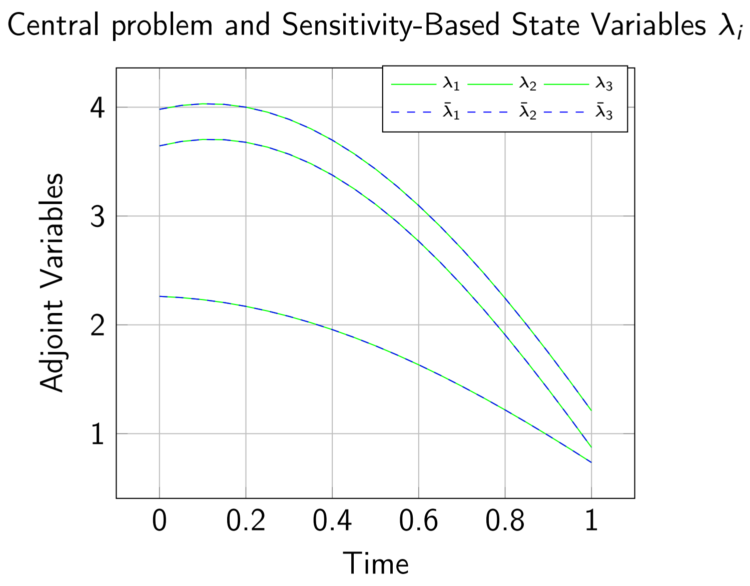
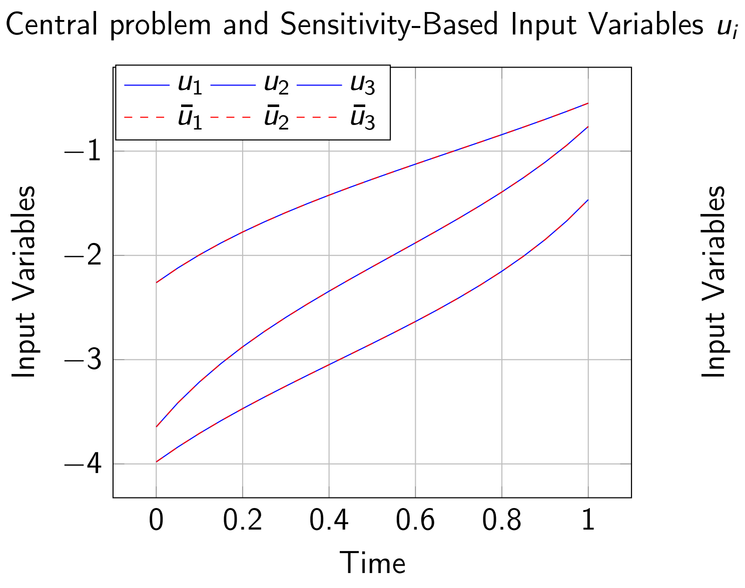


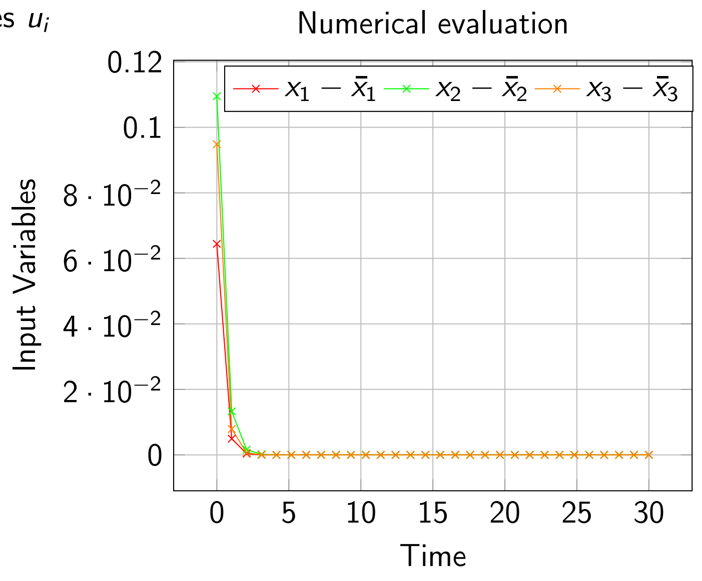
Figure 4.4: example 2

Firstly I use two ways mentioned in chapters 2 and 3 to solve this problem. The following plots, show the results of two different kinds of methods mentioned in chapter 2 and chapter 3. The results of the Fixed-point iteration method are drawn in dashed line. The results of the Sensitivity-based method are drawn in solid lines. In the first plot, the dashed line is the trajectory of the Fixed-point iteration method and the solid line is the trajectory of the Sensitivity-based method. The plots at the first row and second column and at the second row and first column are the trajectories of Adjoint variables and input variables respectively. The results are almost the same respectively. The results are shown:


Central problem and Sensitivity-Based x_i
x_1 x_2 x_3
$\bar{x}_1$ $\bar{x}_2$ $\bar{x}_3$
State Variables
1.2
1
0.8
0
0.2
0.4
0.6
0.8
1
Time


Central problem and Sensitivity-Based State Variables λ_i
λ_1 λ_2 λ_3
$\bar{\lambda}_1$ $\bar{\lambda}_2$ $\bar{\lambda}_3$
Adjoint Variables
4
3
2
1
0
0.2
0.4
0.6
0.8
1
Time


Central problem and Sensitivity-Based Input Variables u_i
u_1 u_2 u_3
$\bar{u}_1$ $\bar{u}_2$ $\bar{u}_3$
Input Variables
-1
-2
-3
-4
0
0.2
0.4
0.6
0.8
1
Time


Numerical evaluation
$x_1 - \bar{x}_1$ $x_2 - \bar{x}_2$ $x_3 - \bar{x}_3$
Input Variables
0.12
0.1
$8 \cdot 10^{-2}$
$6 \cdot 10^{-2}$
$4 \cdot 10^{-2}$
$2 \cdot 10^{-2}$
0
0
5
10
15
20
25
30
Time

Example 3 In this example, the coupling way is a single direction. Agent 2 sends its information to Agent 3, but Agent 3 does not send its information to Agent 2. In equation (4.3), $x_2 - x_1$ and $x_1 - x_2$ are the coupling terms between agent 1 and agent2. $x_3 - x_2$ is the coupling term of agent 2 with agent 3. Agent 3 does not have any coupling terms. When the algorithm is implemented, it is assumed that agent 3 has a coupling term with agent 2 but the coupling term is 0. So the code can run in the same way as before.

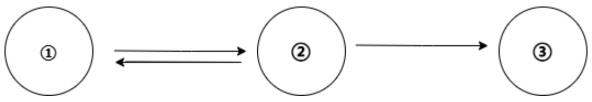
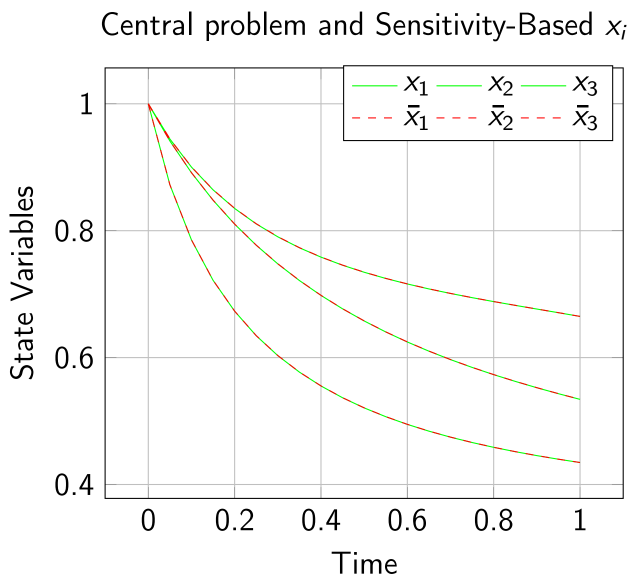
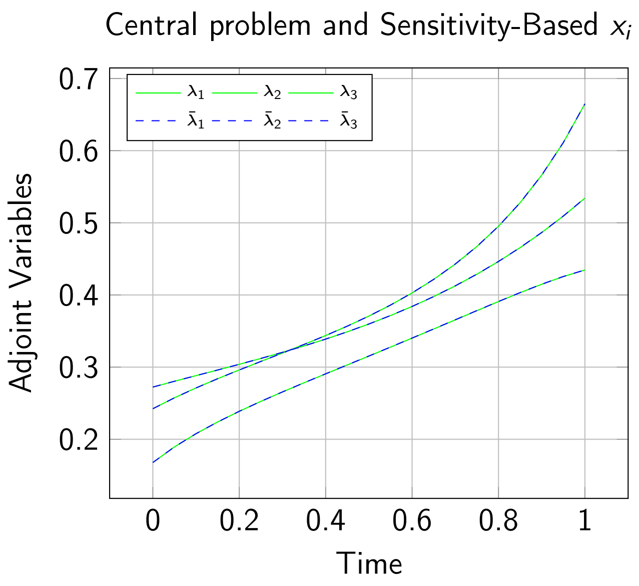


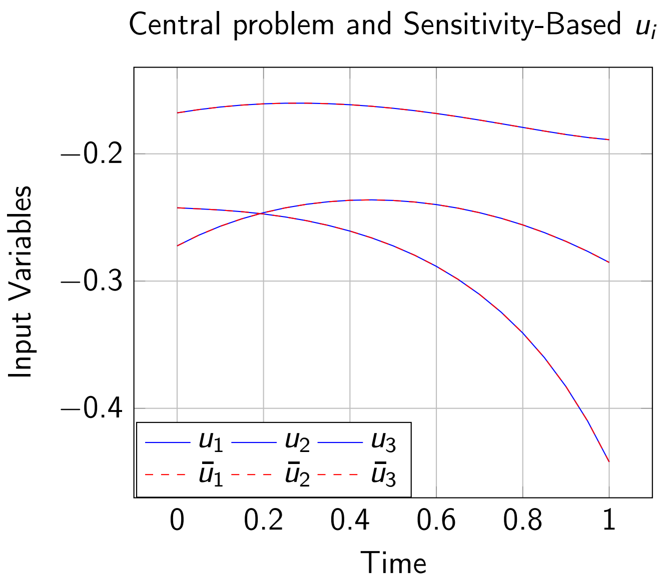
Figure 4.6: example 3

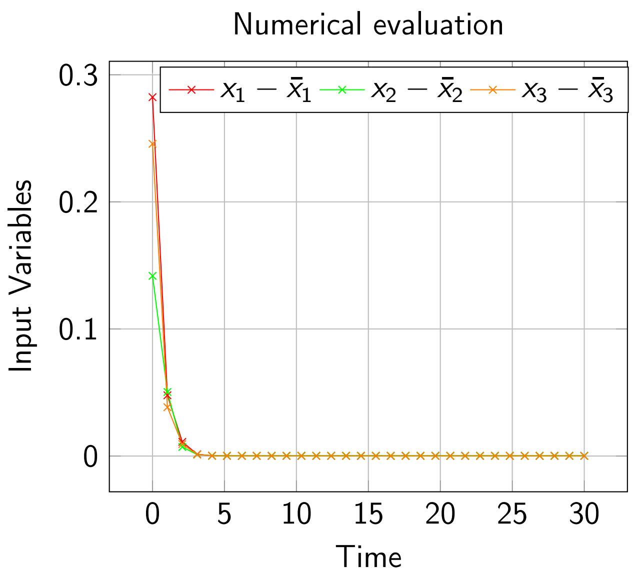
$$\dot{x} = \begin{bmatrix} x_1 + x_1 u_1 + x_2 - x_1 \\ 2x_2 + u_2 x_2 + x_1 - x_2 + x_3 - x_2 \\ 3x_3 + u_3 x_3 \end{bmatrix} \tag{4.3}$$

The results of the method based on the sensitivity method and Fixed-point iteration method are shown:


Central problem and Sensitivity-Based x_i
State Variables
Time
x_1 x_2 x_3
$\bar{x}_1$ $\bar{x}_2$ $\bar{x}_3$
0.4
0.6
0.8
1
0
0.2
0.4
0.6
0.8
1


Central problem and Sensitivity-Based x_i
Adjoint Variables
Time
λ_1 λ_2 λ_3
$\bar{\lambda}_1$ $\bar{\lambda}_2$ $\bar{\lambda}_3$
0.2
0.3
0.4
0.5
0.6
0.7
0
0.2
0.4
0.6
0.8
1


Central problem and Sensitivity-Based u_i
Input Variables
Time
u_1 u_2 u_3
$\bar{u}_1$ $\bar{u}_2$ $\bar{u}_3$
−0.4
−0.3
−0.2
0
0.2
0.4
0.6
0.8
1


Numerical evaluation
Input Variables
Time
$x_1 - \bar{x}_1$ $x_2 - \bar{x}_2$ $x_3 - \bar{x}_3$
0
0.1
0.2
0.3
0
5
10
15
20
25
30

The same results are shown. That means the method based on sensitivity distributed model predictive controllers has the same performance as solving the central problem method with only the Fixed-point iteration method.

Example 4 For this example, the dynamic function has more complicated coupling terms.

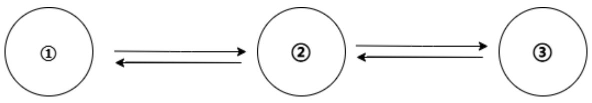
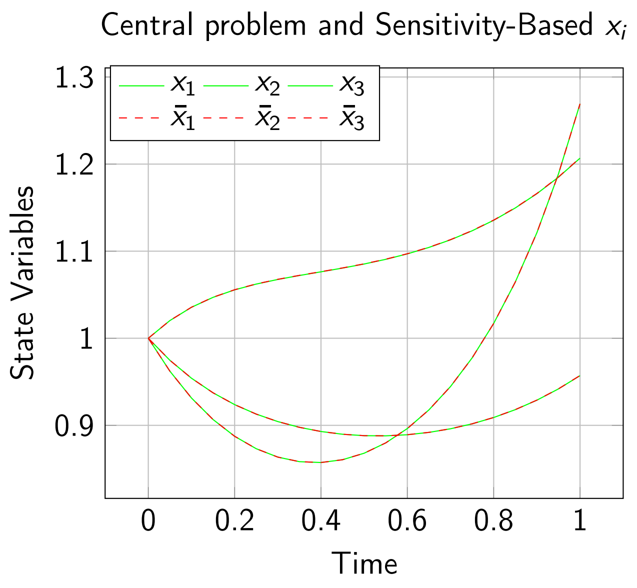


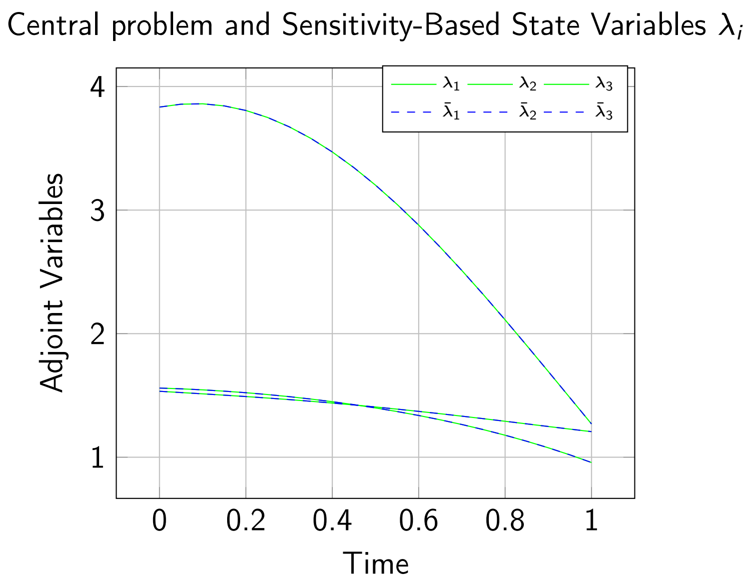
Figure 4.8: example 4

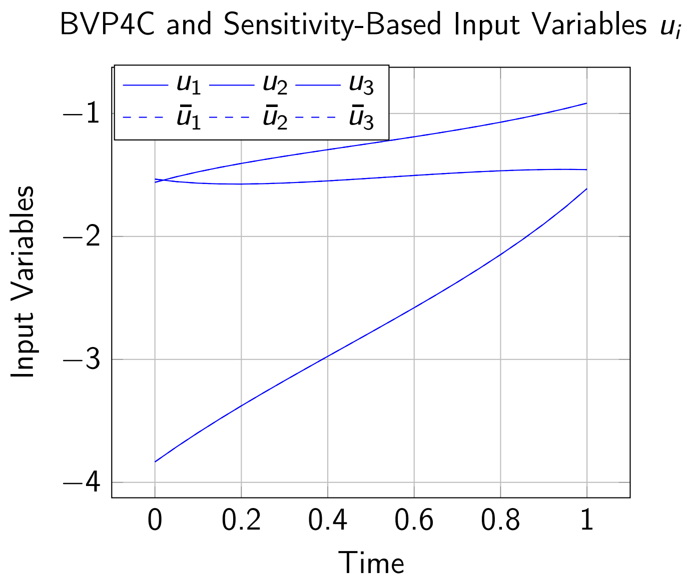
We will solve this problem with the central problem method first and then use the distributed model prediction control method. At last, we compare the results of the two methods.

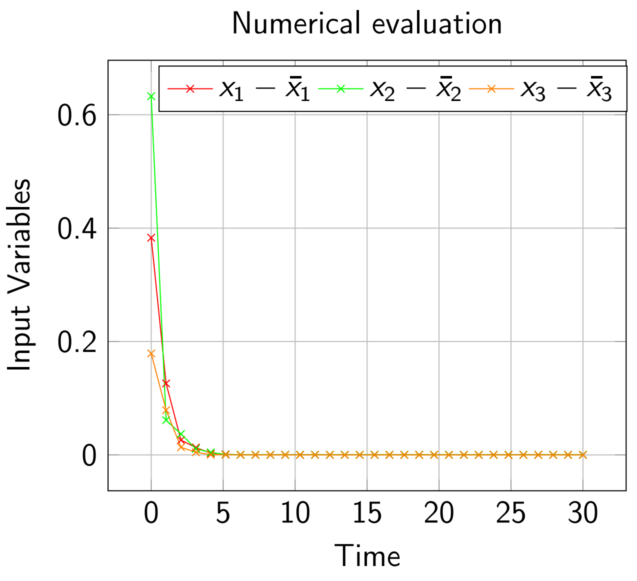
$$\dot{x} = \begin{bmatrix} -x_1^3 + u_1x_1 + \sin(x_2 - x_1) \\ -2x_2^3 + u_2x_2 + \cos(x_1 - x_2) + \sin(x_3 - x_2) \\ -3x_3^3 + u_3x_3 + \sin(x_2 - x_3) \end{bmatrix} \tag{4.4}$$

The results of the method based on sensitivity and the central problem solved by the Fixed-point iteration method are shown:


Central problem and Sensitivity-Based x_i
x_1 x_2 x_3
$\bar{x}_1$ $\bar{x}_2$ $\bar{x}_3$
State Variables
Time


Central problem and Sensitivity-Based State Variables λ_i
λ_1 λ_2 λ_3
$\bar{\lambda}_1$ $\bar{\lambda}_2$ $\bar{\lambda}_3$
Adjoint Variables
Time


BVP4C and Sensitivity-Based Input Variables u_i
u_1 u_2 u_3
$\bar{u}_1$ $\bar{u}_2$ $\bar{u}_3$
Input Variables
Time


Numerical evaluation
$x_1 - \bar{x}_1$ $x_2 - \bar{x}_2$ $x_3 - \bar{x}_3$
Input Variables
Time

Bibliography

[1] H. Huber and K. Graichen, "A sensitivity-based distributed model predictive control algorithm for nonlinear continuous-time systems," in *2021 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2021, pp. 195–201.

[2] H. Scheu and W. Marquardt, "Sensitivity-based coordination in distributed model predictive control," *Journal of Process Control*, vol. 21, no. 5, pp. 715–728, 2011.

[3] P. D. Christofides, R. Scattolini, D. M. De La Pena, and J. Liu, "Distributed model predictive control: A tutorial review and future research directions," *Computers & Chemical Engineering*, vol. 51, pp. 21–41, 2013.

[4] P. D. Christofides, R. Scattolini, D. M. de la Pena, and J. Liu, "Distributed model predictive control: a tutorial review," *Proc. Chem. Process Control*, vol. 8, p. 22pp.

[5] K. Graichen, "A fixed-point iteration scheme for real-time model predictive control," *Automatica*, vol. 48, no. 7, pp. 1300–1305, 2012.

[6] H. Fawal, D. Georges, and G. Bornard, "Optimal control of complex irrigation systems via decomposition-coordination and the use of augmented lagrangian," in *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, vol. 4. IEEE, 1998, pp. 3874–3879.

[7] A. N. Venkat, J. B. Rawlings, and S. J. Wright, "Distributed model predictive control of large-scale systems," in *Assessment and Future Directions of Nonlinear Model Predictive Control*. Springer, 2007, pp. 591–605.

[8] P. Giselsson and A. Rantzer, "On feasibility, stability and performance in distributed model predictive control," *IEEE Transactions on Automatic Control*, vol. 59, no. 4, pp. 1031–1036, 2013.

[9] D. Jia and B. H. Krogh, "Distributed model predictive control," in *Proceedings of the 2001 american control conference.(Cat. No. 01CH37148)*, vol. 4. IEEE, 2001, pp. 2767–2772.

[10] E. Ghadimi, I. Shames, and M. Johansson, "Multi-step gradient methods for networked optimization," *IEEE Transactions on Signal Processing*, vol. 61, no. 21, pp. 5417–5429, 2013.

[11] B. T. Stewart, A. N. Venkat, J. B. Rawlings, S. J. Wright, and G. Pannocchia, "Cooperative distributed model predictive control," *Systems & Control Letters*, vol. 59, no. 8, pp. 460–469, 2010.

[12] A. N. Venkat, J. B. Rawlings, and S. J. Wright, "Stability and optimality of distributed model predictive control," in *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 6680–6685.

[13] I. Jurado, D. E. Quevedo, K. H. Johansson, and A. Ahlén, "Cooperative dynamic mpc for networked control systems," *Distributed Model Predictive Control Made Easy*, pp. 357–373, 2014.

[14] D. Axehill and A. Hansson, "Parallel implementation of hybrid mpc," in *Distributed Model Predictive Control Made Easy*. Springer, 2013, pp. 375–392.

[15] Y. Yan, Y. Zhang, and X. Liu, "Distributed mpc strategy with application to agc in the presence of variable speed wind turbine," in *2015 34th Chinese Control Conference (CCC)*. IEEE, 2015, pp. 4151–4155.

[16] S. Pollock, L. G. Rebholz, and M. Xiao, "Anderson-accelerated convergence of picard iterations for incompressible navier–stokes equations," *SIAM Journal on Numerical Analysis*, vol. 57, no. 2, pp. 615–637, 2019.

[17] ——, "Anderson-accelerated convergence of picard iterations for incompressible navier–stokes equations," *SIAM Journal on Numerical Analysis*, vol. 57, no. 2, pp. 615–637, 2019.

[18] D. G. Anderson, "Iterative procedures for nonlinear integral equations," *Journal of the ACM (JACM)*, vol. 12, no. 4, pp. 547–560, 1965.

[19] H. F. Walker and P. Ni, "Anderson acceleration for fixed-point iterations," *SIAM Journal on Numerical Analysis*, vol. 49, no. 4, pp. 1715–1735, 2011.

[20] V. Eyert, "A comparative study on methods for convergence acceleration of iterative vector sequences," *Journal of Computational Physics*, vol. 124, no. 2, pp. 271–285, 1996.