

# **Distributed Autonomous Systems**

Last update: 03 June 2025

Academic Year 2024 – 2025  
Alma Mater Studiorum · University of Bologna

# Contents

<b>1 Graphs</b>	<b>1</b>
1.1 Definitions . . . . .	1
1.2 Weighted digraphs . . . . .	2
1.3 Laplacian matrix . . . . .	2
<b>2 Averaging systems</b>	<b>4</b>
2.1 Discrete-time averaging algorithm . . . . .	4
2.1.1 Stochastic matrices . . . . .	5
2.1.2 Consensus . . . . .	5
2.2 Discrete-time averaging algorithm over time-varying graphs . . . . .	6
2.2.1 Time-varying digraphs . . . . .	6
2.2.2 Consensus . . . . .	6
2.3 Continuous-time averaging algorithm . . . . .	7
2.3.1 Laplacian dynamics . . . . .	7
2.3.2 Consensus . . . . .	8
2.A Appendix: Discrete time averaging system consensus proof . . . . .	8
<b>3 Containment</b>	<b>12</b>
3.1 Containment with static leaders . . . . .	12
3.2 Containment with non-static leaders . . . . .	15
3.3 Containment with non-static leaders and integral action . . . . .	16
3.4 Containment with discrete-time . . . . .	17
3.5 Containment with multivariate states . . . . .	17
<b>4 Optimization</b>	<b>18</b>
4.1 Definitions . . . . .	18
4.1.1 Unconstrained optimization . . . . .	18
4.1.2 Convexity . . . . .	18
4.2 Iterative descent methods . . . . .	20
4.2.1 Gradient method . . . . .	20
4.2.2 Accelerated gradient methods . . . . .	24
4.3 Cost-coupled optimization . . . . .	25
4.3.1 Learning paradigms . . . . .	25
4.4 Federated learning . . . . .	25
4.4.1 Batch gradient method . . . . .	25
4.4.2 Incremental gradient method . . . . .	25
4.4.3 Stochastic gradient descent . . . . .	26
4.4.4 Adaptive momentum . . . . .	27
4.5 Distributed cost-coupled/consensus optimization . . . . .	27
4.5.1 Distributed gradient algorithm . . . . .	27
4.5.2 Gradient tracking algorithm . . . . .	29
4.A Appendix: Gradient tracking optimality and stability proof . . . . .	30

<b>5 Formation control</b>	<b>32</b>
5.1 Intuition with mass-spring systems . . . . .	32
5.2 Formation control based on potential functions . . . . .	34
<b>6 Cooperative robotics</b>	<b>36</b>
6.1 Aggregative optimization . . . . .	36
6.1.1 Centralized gradient method . . . . .	37
6.1.2 Aggregative tracking distributed optimization algorithm . . . . .	37
6.1.3 Online aggregative optimization . . . . .	38
<b>7 Safety controllers</b>	<b>39</b>
7.1 Safety filter via control barrier certificate . . . . .	40
7.1.1 Single-robot obstacle avoidance with single integrator models . . . . .	41
7.1.2 Multi-robot collision avoidance with single integrator models . . . . .	41
7.1.3 Multi-robot collision avoidance with unicycle control . . . . .	42
<b>8 Feedback optimization</b>	<b>44</b>
8.1 Centralized feedback optimization . . . . .	45
8.2 Distributed feedback optimization . . . . .	46
<b>9 Neural networks</b>	<b>48</b>
9.1 Training problem definition . . . . .	48
9.2 Backpropagation . . . . .	49
9.2.1 Preliminaries . . . . .	49
9.2.2 Adjoint method for neural networks . . . . .	50
9.2.3 Federated machine learning . . . . .	51
9.2.4 Distributed machine learning . . . . .	51

# 1 Graphs

## 1.1 Definitions

<b>Directed graph (digraph)</b>	Pair $G = (I, E)$ where $I = \{1, \dots, N\}$ is the set of nodes and $E \subseteq I \times I$ is the set of edges.	Directed graph
<b>Undirected graph</b>	Digraph where $\forall i, j : (i, j) \in E \Rightarrow (j, i) \in E$ .	Undirected graph
<b>Subgraph</b>	Given a graph $(I, E)$ , $(I', E')$ is a subgraph of it if $I' \subseteq I$ and $E' \subset E$ .	Subgraph
<b>Spanning subgraph</b>	Subgraph where $I' = I$ .	
<b>In-neighbor</b>	A node $j \in I$ is an in-neighbor of $i \in I$ if $(j, i) \in E$ .	In-neighbor
<b>Set of in-neighbors</b>	The set of in-neighbors of $i \in I$ is the set:	Set of in-neighbors
	$\mathcal{N}_i^{\text{IN}} = \{j \in I \mid (j, i) \in E\}$	
<b>In-degree</b>	Number of in-neighbors of a node $i \in I$ :	In-degree
	$\deg_i^{\text{IN}} =  \mathcal{N}_i^{\text{IN}} $	
<b>Out-neighbor</b>	A node $j \in I$ is an out-neighbor of $i \in I$ if $(i, j) \in E$ .	Out-neighbor
<b>Set of out-neighbors</b>	The set of out-neighbors of $i \in I$ is the set:	Set of in-neighbors
	$\mathcal{N}_i^{\text{OUT}} = \{j \in I \mid (i, j) \in E\}$	
<b>Out-degree</b>	Number of out-neighbors of a node $i \in I$ :	Out-degree
	$\deg_i^{\text{OUT}} =  \mathcal{N}_i^{\text{OUT}} $	
<b>Balanced digraph</b>	A digraph is balanced if $\forall i \in I : \deg_i^{\text{IN}} = \deg_i^{\text{OUT}}$ .	Balanced digraph
<b>Periodic graph</b>	Graph where there exists a period $k > 1$ that divides the length of any cycle.	Periodic graph
<b>Remark.</b>	A graph with self-loops is aperiodic.	
<b>Strongly connected digraph</b>	Digraph where each node is reachable from any node.	Strongly connected digraph
<b>Connected undirected graph</b>	Undirected graph where each node is reachable from any node.	Connected undirected graph
<b>Weakly connected digraph</b>	Digraph where its undirected version is connected.	Weakly connected digraph

## 1.2 Weighted digraphs

**Weighted digraph** Triplet  $G = (I, E, \{a_{i,j}\}_{(i,j) \in E})$  where  $(I, E)$  is a digraph and  $a_{i,j} > 0$  is a weight for the edge  $(i, j)$ .

**Weighted in-degree** Sum of the weights of the inward edges:

Weighted in-degree

$$\deg_i^{\text{IN}} = \sum_{j=1}^N a_{j,i}$$

**Weighted out-degree** Sum of the weights of the outward edges:

Weighted out-degree

$$\deg_i^{\text{OUT}} = \sum_{j=1}^N a_{i,j}$$

**Weighted adjacency matrix** Non-negative matrix  $\mathbf{A}$  such that  $\mathbf{A}_{i,j} = a_{i,j}$ :

Weighted adjacency matrix

$$\begin{cases} \mathbf{A}_{i,j} > 0 & \text{if } (i, j) \in E \\ \mathbf{A}_{i,j} = 0 & \text{otherwise} \end{cases}$$

**In/out-degree matrix** Matrix where the diagonal contains the in/out-degrees:

In/out-degree matrix

$$\mathbf{D}^{\text{IN}} = \begin{bmatrix} \deg_1^{\text{IN}} & 0 & \dots & 0 \\ 0 & \deg_2^{\text{IN}} & & \\ \vdots & & \ddots & \\ 0 & \dots & 0 & \deg_N^{\text{IN}} \end{bmatrix} \quad \mathbf{D}^{\text{OUT}} = \begin{bmatrix} \deg_1^{\text{OUT}} & 0 & \dots & 0 \\ 0 & \deg_2^{\text{OUT}} & & \\ \vdots & & \ddots & \\ 0 & \dots & 0 & \deg_N^{\text{OUT}} \end{bmatrix}$$

**Remark.** Given a digraph with adjacency matrix  $\mathbf{A}$ , its reverse digraph has adjacency matrix  $\mathbf{A}^T$ .

**Remark.** It holds that:

$$\mathbf{D}^{\text{IN}} = \text{diag}(\mathbf{A}^T \mathbf{1}) \quad \mathbf{D}^{\text{OUT}} = \text{diag}(\mathbf{A} \mathbf{1})$$

where  $\mathbf{1}$  is a vector of ones.

**Remark.** A digraph is balanced iff  $\mathbf{A}^T \mathbf{1} = \mathbf{A} \mathbf{1}$ .

## 1.3 Laplacian matrix

**(Out-degree) Laplacian matrix** Matrix  $\mathbf{L}$  defined as:

Laplacian matrix

$$\mathbf{L} = \mathbf{D}^{\text{OUT}} - \mathbf{A}$$

**Remark.** The vector  $\mathbf{1}$  is always an eigenvector of  $\mathbf{L}$  with eigenvalue 0:

$$\mathbf{L} \mathbf{1} = (\mathbf{D}^{\text{OUT}} - \mathbf{A}) \mathbf{1} = \mathbf{D}^{\text{OUT}} \mathbf{1} - \mathbf{D}^{\text{OUT}} \mathbf{1} = 0$$

**In-degree Laplacian matrix** Matrix  $\mathbf{L}^{\text{IN}}$  defined as:

In-degree Laplacian matrix

$$\mathbf{L}^{\text{IN}} = \mathbf{D}^{\text{IN}} - \mathbf{A}^T$$

| **Remark.**  $L^{\text{IN}}$  is the out-degree Laplacian of the reverse graph.

## 2 Averaging systems

**Distributed algorithm** Given a network of  $N$  agents that communicate according to a (fixed) digraph  $G$  (each agent receives messages from its in-neighbors), a distributed algorithm computes:

$$x_i^{k+1} = \text{stf}_i(x_i^k, \{x_j^k\}_{j \in \mathcal{N}_i^{\text{IN}}}) \quad \forall i \in \{1, \dots, N\}$$

where  $x_i^k$  is the state of agent  $i$  at time  $k$  and  $\text{stf}_i$  is a local state transition function that depends on the current input states.

| **Remark.** Out-neighbors can also be used.

| **Remark.** If all nodes have a self-loop, the notation can be compacted as:

$$x_i^{k+1} = \text{stf}_i(\{x_j\}_{j \in \mathcal{N}_i^{\text{IN}}}) \quad \text{or} \quad x_i^{k+1} = \text{stf}_i(\{x_j\}_{j \in \mathcal{N}_i^{\text{OUT}}})$$

Distributed algorithm

### 2.1 Discrete-time averaging algorithm

**Linear averaging distributed algorithm (in-neighbors)** Given the communication digraph with self-loops  $G^{\text{comm}} = (I, E)$  (i.e.,  $(j, i) \in E$  indicates that  $j$  sends messages to  $i$ ), a linear averaging distributed algorithm is defined as:

$$x_i^{k+1} = \sum_{j \in \mathcal{N}_i^{\text{IN}}} a_{ij} x_j^k \quad i \in \{1, \dots, N\}$$

where  $a_{ij} > 0$  is the weight of the edge  $(j, i) \in E$ .

**Linear time-invariant (LTI) autonomous system** By defining  $a_{ij} = 0$  for  $(j, i) \notin E$ , the formulation becomes:

$$x_i^{k+1} = \sum_{j=1}^N a_{ij} x_j^k \quad i \in \{1, \dots, N\}$$

Linear averaging distributed algorithm (in-neighbors)

In matrix form, it becomes:

$$\mathbf{x}^{k+1} = \mathbf{A}^T \mathbf{x}^k$$

where  $\mathbf{A}$  is the adjacency matrix of  $G^{\text{comm}}$ .

| **Remark.** This model is inconsistent with respect to graph theory as weights are inverted (i.e.,  $a_{ij}$  refers to the edge  $(j, i)$ ).

Linear time-invariant (LTI) autonomous system

**Linear averaging distributed algorithm (out-neighbors)** Given a fixed sensing digraph with self-loops  $G^{\text{sens}} = (I, E)$  (i.e.,  $(i, j) \in E$  indicates that  $j$  sends messages to  $i$ ), the algorithm is defined as:

$$x_i^{k+1} = \sum_{j \in \mathcal{N}_i^{\text{OUT}}} a_{ij} x_j^k = \sum_{j=1}^N a_{ij} x_j^k$$

Linear averaging distributed algorithm (out-neighbors)

In matrix form, it becomes:

$$\mathbf{x}^{k+1} = \mathbf{A}\mathbf{x}^k$$

where  $\mathbf{A}$  is the weighted adjacency matrix of  $G^{\text{sens}}$ .

### 2.1.1 Stochastic matrices

**Row stochastic** Given a square matrix  $\mathbf{A}$ , it is row stochastic if its rows sum to 1:

Row stochastic

$$\mathbf{A}\mathbf{1} = \mathbf{1}$$

**Column stochastic** Given a square matrix  $\mathbf{A}$ , it is column stochastic if its columns sum to 1:

Column stochastic

$$\mathbf{A}^T\mathbf{1} = \mathbf{1}$$

**Doubly stochastic** Given a square matrix  $\mathbf{A}$ , it is doubly stochastic if it is both row and column stochastic.

Doubly stochastic

**Lemma 2.1.1.** An adjacency matrix  $\mathbf{A}$  is doubly stochastic if it is row stochastic and the graph  $G$  associated to it is weight balanced and has positive weights.

**Lemma 2.1.2.** Given a digraph  $G$  with adjacency matrix  $\mathbf{A}$ , if  $G$  is strongly connected and aperiodic, and  $\mathbf{A}$  is row stochastic, its eigenvalues are such that:

- $\lambda = 1$  is a simple eigenvalue (i.e., algebraic multiplicity of 1),
- All others  $\mu$  are  $|\mu| < 1$ .

**Remark.** For the lemma to hold, it is necessary and sufficient that  $G$  contains a globally reachable node and the subgraph of globally reachable nodes is aperiodic.

### 2.1.2 Consensus

**Theorem 2.1.1** (Discrete-time consensus). Consider a discrete-time averaging system with digraph  $G$  and weighted adjacency matrix  $\mathbf{A}$ . Assume  $G$  strongly connected and aperiodic, and  $\mathbf{A}$  row stochastic.

Discrete-time consensus

It holds that there exists a left eigenvector  $\mathbf{w} \in \mathbb{R}^N$ ,  $\mathbf{w} > 0$  such that the consensus converges to:

$$\lim_{k \rightarrow \infty} \mathbf{x}^k = \mathbf{1} \frac{\mathbf{w}^T \mathbf{x}^0}{\mathbf{w}^T \mathbf{1}} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \frac{\sum_{i=1}^N w_i x_i^0}{\sum_{j=1}^N w_j} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \sum_{i=1}^N \frac{w_i}{\sum_{j=1}^N w_j} x_i^0$$

where  $\tilde{w}_i = \frac{w_i}{\sum_{j=1}^N w_j}$  are all normalized and sum to 1 (i.e., they produce a convex combination).

Moreover, if  $\mathbf{A}$  is doubly stochastic, then it holds that the consensus is the average as  $\mathbf{w} = \mathbf{1}$ :

$$\lim_{k \rightarrow \infty} \mathbf{x}^k = \mathbf{1} \frac{1}{N} \sum_{i=1}^N x_i^0$$

**Example** (Metropolis-Hastings weights). Given an undirected unweighted graph  $G$  with

edges of degrees  $d_1, \dots, d_n$ , Metropolis-Hastings weights are defined as:

$$a_{ij} = \begin{cases} \frac{1}{1 + \max\{d_i, d_j\}} & \text{if } (i, j) \in E \text{ and } i \neq j \\ 1 - \sum_{h \in \mathcal{N}_i \setminus \{j\}} a_{ih} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

The matrix  $\mathbf{A}$  of Metropolis-Hastings weights is symmetric and doubly stochastic.

## 2.2 Discrete-time averaging algorithm over time-varying graphs

### 2.2.1 Time-varying digraphs

**Time-varying digraph** Graph  $G = (I, E(k))$  that changes at each iteration  $k$ . It can be described by a sequence  $\{G(k)\}_{k \geq 0}$ .

**Jointly strongly connected digraph** Time-varying digraph that is asymptotically strongly connected:

$$\forall k \geq 0 : \bigcup_{\tau=k}^{+\infty} G(\tau) \text{ is strongly connected}$$

**Uniformly jointly strongly/B-strongly connected digraph** Time-varying digraph that is strongly connected in  $B$  steps:

$$\forall k \geq 0, \exists B \in \mathbb{N} : \bigcup_{\tau=k}^{k+B} G(\tau) \text{ is strongly connected}$$

**Remark.** (Uniformly) jointly strongly connected digraph can be disconnected at some time steps  $k$ .

**Averaging distributed algorithm** Given a time-varying digraph  $\{G(k)\}_{k \geq 0}$  (always with self-loops), in- and out-neighbors distributed algorithms can be formulated as:

$$x_i^{k+1} = \sum_{j \in \mathcal{N}_i^{\text{IN}}(k)} a_{ij}(k) x_j^k \quad x_i^{k+1} = \sum_{j \in \mathcal{N}_i^{\text{OUT}}(k)} a_{ij}(k) x_j^k$$

**Linear time-varying (LTV) discrete-time system** In matrix form, it can be formulated as:

$$\mathbf{x}^{k+1} = \mathbf{A}(k) \mathbf{x}^k$$

### 2.2.2 Consensus

**Theorem 2.2.1** (Discrete-time consensus over time-varying graphs). Consider a time-varying discrete-time average system with digraphs  $\{G(k)\}_{k \geq 0}$  (all with self-loops) and weighted adjacency matrices  $\{\mathbf{A}(k)\}_{k \geq 0}$ . Assume:

- Each non-zero edge weight  $a_{ij}(k)$ , self-loops included, are larger than a constant  $\varepsilon > 0$ ,
- There exists  $B \in \mathbb{N}$  such that  $\{G(k)\}_{k \geq 0}$  is  $B$ -strongly connected.

It holds that there exists a vector  $\mathbf{w} \in \mathbb{R}^N$ ,  $\mathbf{w} > 0$  such that the consensus converges to:

$$\lim_{k \rightarrow \infty} \mathbf{x}^k = \mathbf{1} \frac{\mathbf{w}^T \mathbf{x}^0}{\mathbf{w}^T \mathbf{1}}$$

Time-varying  
digraph

Jointly strongly  
connected digraph

Uniformly jointly  
strongly/B-strongly  
connected digraph

Averaging  
distributed  
algorithm over  
time-varying digraph

Linear time-varying  
(LTV) discrete-time  
system

Discrete-time  
consensus over  
time-varying graphs

Moreover, if each  $\mathbf{A}(k)$  is doubly stochastic, it holds that the consensus is the average:

$$\lim_{k \rightarrow \infty} \mathbf{x}^k = \mathbf{1} \frac{1}{N} \sum_{i=1}^N x_i^0$$

## 2.3 Continuous-time averaging algorithm

### 2.3.1 Laplacian dynamics

**Network of dynamic systems** Network described by the ODEs:

$$\dot{x}_i(t) = u_i(t) \quad \forall i \in \{1, \dots, N\}$$

with states  $x_i \in \mathbb{R}$ , inputs  $u_i \in \mathbb{R}$ , and communication following a digraph  $G$ .

**Laplacian dynamics system** Consider a network of dynamic systems where  $u_i$  is defined as a proportional controller (i.e., only communicating  $(i, j)$  have a non-zero weight):

$$\begin{aligned} u_i(t) &= - \sum_{j \in \mathcal{N}_i^{\text{OUT}}} a_{ij} (x_i(t) - x_j(t)) \\ &= - \sum_{j=1}^N a_{ij} (x_i(t) - x_j(t)) \end{aligned}$$

Network of dynamic systems

Laplacian dynamics system

**Remark.** With this formulation, consensus can be seen as the problem of minimizing the error defined as the difference between the states of two nodes.

**Remark.** A definition with in-neighbors also exists.

**Theorem 2.3.1** (Linear time invariant (LTI) continuous-time system). With  $\mathbf{x} = [x_1 \ \dots \ x_N]^T$ , the system can be written in matrix form as:

$$\dot{\mathbf{x}}(t) = -\mathbf{L}\mathbf{x}(t)$$

Linear time invariant (LTI) continuous-time system

where  $\mathbf{L}$  is the Laplacian associated with the communication digraph  $G$ .

*Proof.* The system is defined as:

$$\dot{x}_i(t) = - \sum_{j=1}^N a_{ij} (x_i(t) - x_j(t))$$

By rearranging, we have that:

$$\begin{aligned} \dot{x}_i(t) &= - \left( \sum_{j=1}^N a_{ij} \right) x_i(t) + \sum_{j=1}^N a_{ij} x_j(t) \\ &= -\deg_i^{\text{OUT}} x_i(t) + (\mathbf{A}\mathbf{x}(t))_i \end{aligned}$$

Which in matrix form is:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= -\mathbf{D}^{\text{OUT}}\mathbf{x}(t) + \mathbf{A}\mathbf{x}(t) \\ &= -(\mathbf{D}^{\text{OUT}} - \mathbf{A})\mathbf{x}(t) \end{aligned}$$

By definition,  $\mathbf{L} = \mathbf{D}^{\text{OUT}} - \mathbf{A}$ . Therefore, we have that:

$$\dot{\mathbf{x}}(t) = -\mathbf{L}\mathbf{x}(t)$$

□

**Remark.** By Theorem 2.3.1, row/column stochasticity is not required for consensus. Instead, the requirement is for the matrix to be the Laplacian.

### 2.3.2 Consensus

**Lemma 2.3.1.** It holds that:

$$\mathbf{L}\mathbf{1} = \mathbf{D}^{\text{OUT}}\mathbf{1} - \mathbf{A}\mathbf{1} = \begin{bmatrix} \deg_1^{\text{OUT}} \\ \vdots \\ \deg_i^{\text{OUT}} \end{bmatrix} - \begin{bmatrix} \deg_1^{\text{OUT}} \\ \vdots \\ \deg_i^{\text{OUT}} \end{bmatrix} = \mathbf{0}$$

**Lemma 2.3.2.** The Laplacian  $\mathbf{L}$  of a weighted digraph has an eigenvalue  $\lambda = 0$  and all the others have strictly positive real part.

**Lemma 2.3.3.** Given a weighted digraph  $G$  with Laplacian  $\mathbf{L}$ , the following are equivalent:

- $G$  is weight balanced.
- $\mathbf{1}$  is a left eigenvector of  $\mathbf{L}$ :  $\mathbf{1}^T \mathbf{L} = 0$  with eigenvalue 0.

**Lemma 2.3.4.** If a weighted digraph  $G$  is strongly connected, then  $\lambda = 0$  is a simple eigenvalue of  $\mathbf{L}$ .

**Theorem 2.3.2** (Continuous-time consensus). Consider a continuous-time average system with a strongly connected weighted digraph  $G$  and Laplacian  $\mathbf{L}$ . Assume that the system follows the Laplacian dynamics  $\dot{\mathbf{x}}(t) = -\mathbf{L}\mathbf{x}(t)$  for  $t \geq 0$ .

It holds that there exists a left eigenvector  $\mathbf{w}$  of  $\mathbf{L}$  with eigenvalue  $\lambda = 0$  such that the consensus converges to:

$$\lim_{t \rightarrow \infty} \mathbf{x}(t) = \mathbf{1} \left( \frac{\mathbf{w}^T \mathbf{x}(0)}{\mathbf{w}^T \mathbf{1}} \right)$$

Moreover, if  $G$  is weight balanced, then it holds that the consensus is the average:

$$\lim_{t \rightarrow \infty} \mathbf{x}(t) = \mathbf{1} \frac{\sum_{i=1}^N x_i(0)}{N}$$

**Remark.** The result also holds for unweighted digraphs as  $\mathbf{1}$  is both a left and right eigenvector of  $\mathbf{L}$ .

Continuous-time consensus

## 2.A Appendix: Discrete time averaging system consensus proof

**Positive matrix characterization** Given  $\mathbf{A} \in \mathbb{R}^{N \times N}$ , it can be:

**Non-negative**  $\mathbf{A} \geq 0$ .

Non-negative matrix

**Irreducible**  $\sum_{h=0}^{N-1} \mathbf{A}^h > 0$ .

Irreducible matrix

**Primitive**  $\exists h \in \{1, \dots, N\} : \mathbf{A}^h > 0$ .

Primitive matrix

**Positive**  $\mathbf{A} > 0$ .

Positive matrix

**Theorem 2.A.1.** Given a weighted digraph  $G$  with  $N \geq 2$  nodes and adjacency matrix  $\mathbf{A}$ , it holds that:

- $\mathbf{A}$  is irreducible  $\iff G$  is strongly connected.
- $\mathbf{A}$  is primitive  $\iff G$  is strongly connected and aperiodic.

**Theorem 2.A.2** (Gershgorin). For any square matrix  $\mathbf{A} \in \mathbb{C}^{N \times N}$ , it holds that the spectrum of  $\mathbf{A}$  (i.e., set of eigenvalues) is contained in the Gershgorin disks:

$$\text{spec}(\mathbf{A}) \subset \bigcup_{i=1}^N \left\{ s \in \mathbb{C} \mid |s - a_{ii}| \leq \sum_{j=1, j \neq i}^N |a_{ij}| \right\}$$

In other words, it is the union of the disks with center  $a_{ii}$  and radius  $\sum_{j=1, j \neq i}^N |a_{ij}|$ .

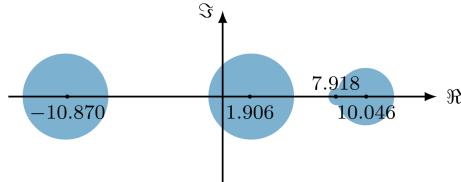
| **Remark.** This theorem provides an approximate location of the eigenvalues.

**Example.** Consider the matrix:

$$\begin{bmatrix} 10 & 1 & 0 & 1 \\ 0.2 & 8 & 0.2 & 0.2 \\ 1 & 1 & 2 & 1 \\ -1 & -1 & -1 & -11 \end{bmatrix}$$

Its eigenvalues are  $\{-10.870, 1.906, 7.918, 10.046\}$ .

The Gershgorin disks are:



**Theorem 2.A.3** (Perron-Frobenius). Let  $\mathbf{A} \in \mathbb{R}^{N \times N}$  with  $N \geq 2$  be a non-negative matrix. It holds that:

Perron-Frobenius theorem

- There exists a real eigenvalue  $\lambda \geq 0$  that is dominant for all the other eigenvalues  $\mu \in \text{spec}(\mathbf{A}) \setminus \{\lambda\}$  (i.e.,  $\lambda \geq |\mu|$ ),
- The right eigenvector  $\mathbf{v} \in \mathbb{R}^N$  and left eigenvector  $\mathbf{w} \in \mathbb{R}^N$  associated to  $\lambda$  can be chosen to be non-negative.

If  $\mathbf{A} \in \mathbb{R}^{N \times N}$  is irreducible, then:

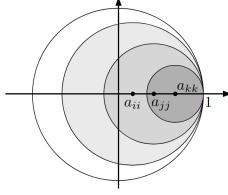
- The eigenvalue  $\lambda$  is strictly positive ( $\lambda > 0$ ) and simple.
- The right and left eigenvectors  $\mathbf{v}$  and  $\mathbf{w}$  associated to  $\lambda$  are unique and positive.

If  $\mathbf{A} \in \mathbb{R}^{N \times N}$  is primitive, then:

- The eigenvalue  $\lambda$  is strictly dominant for all  $\mu \in \text{spec}(\mathbf{A}) \setminus \{\lambda\}$  (i.e.,  $\lambda > |\mu|$ ).

**Lemma 2.A.1.** Given a row stochastic matrix  $\mathbf{A}$ , it holds that:

- $\lambda = 1$  is an eigenvalue,
- By Gershgorin Theorem,  $\text{spec}(\mathbf{A})$  is a subset of the unit disk (i.e., all Gershgorin disks lie inside the unit disk).



**Corollary 2.A.2.** The eigenvalue  $\lambda = 1 \geq |\mu|$  is dominant.

**Lemma 2.A.3.** Given a row stochastic and primitive matrix  $\mathbf{A}$ , by Lemma 2.A.1 and Perron-Frobenius Theorem it holds that  $\lambda = 1$  is simple and strictly dominant.

**Corollary 2.A.4.** The consensus averaging system is marginally stable (i.e., converges but not necessarily to the origin) as the largest distinct eigenvalue is  $\lambda = 1$ .

**Theorem 2.A.4** (Discrete-time consensus). Consider a discrete-time averaging system with digraph  $G$  and weighted adjacency matrix  $\mathbf{A}$ . Assume  $G$  strongly connected and aperiodic, and  $\mathbf{A}$  row stochastic.

It holds that there exists a left eigenvector  $\mathbf{w} \in \mathbb{R}^N$ ,  $\mathbf{w} > 0$  such that the consensus converges to:

$$\lim_{k \rightarrow \infty} \mathbf{x}^k = \mathbf{1} \frac{\mathbf{w}^T \mathbf{x}^0}{\mathbf{w}^T \mathbf{1}} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \frac{\sum_{i=1}^N w_i x_i^0}{\sum_{j=1}^N w_j} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \sum_{i=1}^N \frac{w_i}{\sum_{j=1}^N w_j} x_i^0$$

where  $\tilde{w}_i = \frac{w_i}{\sum_{j=1}^N w_j}$  are all normalized and sum to 1 (i.e., they produce a convex combination).

Moreover, if  $\mathbf{A}$  is doubly stochastic, then it holds that the consensus is the average as  $\mathbf{w} = 1$ :

$$\lim_{k \rightarrow \infty} \mathbf{x}^k = \mathbf{1} \frac{1}{N} \sum_{i=1}^N x_i^0$$

*Proof (Jordan-form approach).* As is  $G$  strongly connected and aperiodic, and  $\mathbf{A}$  is row stochastic, it holds that:

- By Theorem 2.A.1,  $\mathbf{A}$  is primitive.
- By Perron-Frobenius Theorem and Lemma 2.A.1, the eigenvalue  $\lambda = 1$  is strictly dominant and it is associated to the right eigenvector  $\mathbf{1}$  (row stochasticity) and left eigenvector  $\mathbf{w}$ .

Consider the non-singular matrix  $\mathbf{T} \in \mathbb{R}^{N \times N}$  defined as:

$$\mathbf{T} = \begin{bmatrix} | & | & & | \\ \mathbf{1} & \mathbf{v}^2 & \dots & \mathbf{v}^N \\ | & | & & | \end{bmatrix} = [\mathbf{1} \quad \mathbf{W}_R] \quad \mathbf{T}^{-1} = \begin{bmatrix} - & (\mathbf{w})^T & - \\ - & (\mathbf{w}^2)^T & - \\ - & \vdots & - \\ - & (\mathbf{w}^N)^T & - \end{bmatrix} = [\mathbf{w}^T \quad \mathbf{W}_L]$$

A change in coordinates defined as:

$$\mathbf{x} \mapsto \tilde{\mathbf{x}} = \mathbf{T}^{-1} \mathbf{x}$$

allows to obtain the Jordan form  $\mathbf{T}^{-1} \mathbf{A} \mathbf{T}$ :

$$\mathbf{T}^{-1} \mathbf{A} \mathbf{T} = \begin{bmatrix} 1 & 0 & \dots \\ 0 & & \\ \vdots & & \mathbf{J}_2 \end{bmatrix}$$

Discrete-time  
consensus

with  $\mathbf{J}_2 \in \mathbb{R}^{(N-1) \times (N-1)}$  Schur (i.e.,  $\text{spec}(\mathbf{J}_2)$  inside the open unit disk). The dynamics  $\mathbf{x}^{k+1} = \mathbf{A}\mathbf{x}^k$  in the new coordinate system is:

$$\begin{aligned}\tilde{\mathbf{x}}^{k+1} &= \mathbf{T}^{-1}\mathbf{x}^{k+1} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}\tilde{\mathbf{x}}^k \\ &= \begin{bmatrix} 1 & 0 & \dots \\ 0 & & \\ \vdots & & \mathbf{J}_2 \end{bmatrix} \tilde{\mathbf{x}}^k = \begin{bmatrix} 1 & 0 & \dots \\ 0 & & \\ \vdots & & \mathbf{J}_2 \end{bmatrix}^{k+1} \tilde{\mathbf{x}}^0\end{aligned}$$

Let's denote:

$$\tilde{\mathbf{x}}^k = \mathbf{T}^{-1}\mathbf{x}^k = \begin{bmatrix} \mathbf{w}^T \mathbf{x}^k \\ \mathbf{W}_L \mathbf{x}^k \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{x}}_m^k \\ \tilde{\mathbf{x}}_\perp^k \end{bmatrix}$$

We have that:

$$\begin{aligned}\lim_{k \rightarrow \infty} \tilde{\mathbf{x}}^k &= \lim_{k \rightarrow \infty} \begin{bmatrix} 1 & 0 & \dots \\ 0 & & \\ \vdots & & \mathbf{J}_2 \end{bmatrix}^k \tilde{\mathbf{x}}^0 \\ &= \lim_{k \rightarrow \infty} \begin{bmatrix} 1 & 0 & \dots \\ 0 & & \\ \vdots & & (\mathbf{J}_2)^k \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_m^0 \\ \tilde{\mathbf{x}}_\perp^0 \end{bmatrix} \\ &= \begin{bmatrix} 1 \cdot \tilde{\mathbf{x}}_m^0 \\ \lim_{k \rightarrow \infty} (\mathbf{J}_2)^k \tilde{\mathbf{x}}_\perp^0 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{w}^T \mathbf{x}^0 \\ 0 \end{bmatrix}\end{aligned}$$

Note that  $\lim_{k \rightarrow \infty} \mathbf{J}_2^k = 0$  as it is stable (i.e., all eigenvalues are in the open unit disk  $|\mu| < 1$ ).

In the original coordinate system, the limit is:

$$\begin{aligned}\lim_{k \rightarrow \infty} \mathbf{x}^k &= \lim_{k \rightarrow \infty} \mathbf{T}\tilde{\mathbf{x}}^k \\ &= \mathbf{T} \lim_{k \rightarrow \infty} \tilde{\mathbf{x}}^k \\ &= [\mathbf{1} \quad \mathbf{W}_R] \begin{bmatrix} \mathbf{w}^T \mathbf{x}^0 \\ 0 \end{bmatrix} = \mathbf{1}(\mathbf{w}^T \mathbf{x}^0)\end{aligned}$$

**Remark.** It is assumed that  $\|\mathbf{w}\| = 1$  (i.e., no normalization term).

□

# 3 Containment

**Leader-follower network** Consider  $N$  agents partitioned into  $N_f$  followers and  $N - N_f$  leaders.

Leader-follower network

The state vector can be partitioned as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_f \\ \mathbf{x}_l \end{bmatrix}$$

where  $\mathbf{x}_f \in \mathbb{R}^{N_f}$  are the followers' states and  $\mathbf{x}_l \in \mathbb{R}^{N-N_f}$  the leaders'.

The Laplacian can also be partitioned as:

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_f & \mathbf{L}_{fl} \\ \mathbf{L}_{fl}^T & \mathbf{L}_l \end{bmatrix}$$

where  $\mathbf{L}_f$  is the followers' Laplacian,  $\mathbf{L}_l$  the leaders', and  $\mathbf{L}_{fl}$  is the part in common. Assume that leaders and followers run the same Laplacian-based distributed control law (i.e., a normal averaging system), the system can be formulated as:

$$\begin{bmatrix} \dot{\mathbf{x}}_f(t) \\ \dot{\mathbf{x}}_l(t) \end{bmatrix} = - \begin{bmatrix} \mathbf{L}_f & \mathbf{L}_{fl} \\ \mathbf{L}_{fl}^T & \mathbf{L}_l \end{bmatrix} \begin{bmatrix} \mathbf{x}_f(t) \\ \mathbf{x}_l(t) \end{bmatrix}$$

**Example.** Consider a path graph with four nodes:

$$0 \leftrightarrow 1 \leftrightarrow 2 \leftrightarrow (3)$$

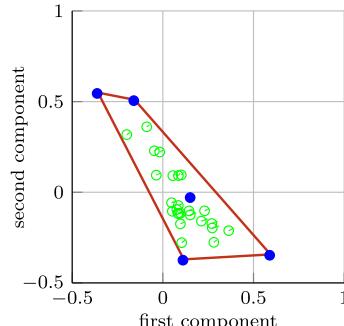
The nodes 0, 1, 2 are followers and 3 is a leader. The system is:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = - \left[ \begin{array}{ccc|c} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{array} \right] \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix}$$

## 3.1 Containment with static leaders

**Containment with static leaders** Task where leaders are stationary and the goal is to drive followers within the convex hull enclosing the leaders. Followers can communicate with agents of any type while leaders do not communicate.

Containment with static leaders



**Containment control law** Given  $N_f$  followers and  $N - N_f$  leaders, the control law to solve the containment task have:

Containment control law

- Followers running Laplacian dynamics.
- Leaders being stationary.

The system is:

$$\begin{aligned}\dot{x}_i(t) &= - \sum_{j \in \mathcal{N}_i} a_{ij} (x_i(t) - x_j(t)) \quad \forall i \in \{1, \dots, N_f\} \\ \dot{x}_i(t) &= 0 \quad \forall i \in \{N_f + 1, \dots, N\}\end{aligned}$$

In matrix form, it becomes:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= -\mathbf{L}\mathbf{x}(t) \\ \begin{bmatrix} \dot{\mathbf{x}}_f(t) \\ \dot{\mathbf{x}}_l(t) \end{bmatrix} &= - \begin{bmatrix} \mathbf{L}_f & \mathbf{L}_{fl} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_f(t) \\ \mathbf{x}_l(t) \end{bmatrix} \\ \dot{\mathbf{x}}_f(t) &= -\mathbf{L}_f \mathbf{x}_f(t) - \mathbf{L}_{fl} \mathbf{x}_l\end{aligned}$$

where  $\mathbf{L}_f$  can be seen as the state matrix and  $\mathbf{L}_{fl}$  as the input matrix. The input  $\mathbf{x}_l = \mathbf{x}_l(0) = \mathbf{x}_l(t)$  is constant.

**Lemma 3.1.1.** If the interaction graph  $G$  between leaders and followers is undirected and connected, then the followers' Laplacian  $\mathbf{L}_f$  is positive definite.

*Proof.* We need to prove that:

$$\mathbf{x}_f^T \mathbf{L}_f \mathbf{x}_f > 0 \quad \forall \mathbf{x}_f \neq 0$$

As  $G$  is undirected, it holds that:

- The complete Laplacian  $\mathbf{L}$  is symmetric and thus have real-valued eigenvalues.
- By Lemma 2.3.2, all its non-zero eigenvalues are positive.
- By Lemma 2.3.4, as  $G$  is connected, the eigenvalue  $\lambda = 0$  is simple.

Therefore:

- $\mathbf{x}^T \mathbf{L} \mathbf{x} \geq 0$  as all eigenvalues are non-negative.
- $\mathbf{x}^T \mathbf{L} \mathbf{x} = 0 \iff \mathbf{x} = \alpha \mathbf{1}$  for  $\alpha \in \mathbb{R}$ , as  $\lambda = 0$  is simple.

The following two arguments can be made:

1. By choosing  $\bar{\mathbf{x}} = [\mathbf{x}_f \ 0]^T$ , it holds that:

$$\begin{aligned}\bar{\mathbf{x}}^T \mathbf{L} \bar{\mathbf{x}} &\geq 0 \quad \forall \bar{\mathbf{x}} \\ [\mathbf{x}_f \ 0] \begin{bmatrix} \mathbf{L}_f & \mathbf{L}_{fl} \\ \mathbf{L}_{fl}^T & \mathbf{L}_l \end{bmatrix} \begin{bmatrix} \mathbf{x}_f \\ 0 \end{bmatrix} &\geq 0 \quad \forall \mathbf{x}_f \\ \mathbf{x}_f^T \mathbf{L}_f \mathbf{x}_f &\geq 0 \quad \forall \mathbf{x}_f\end{aligned}$$

2. The only case when  $\mathbf{x}^T \mathbf{L} \mathbf{x} = 0$  for  $\mathbf{x} \neq 0$  is with  $\mathbf{x} = \alpha \mathbf{1}$  for  $\alpha \neq 0$ . As  $\forall \mathbf{x}_f : \bar{\mathbf{x}} \neq \alpha \mathbf{1}$ , it holds that  $\forall \mathbf{x}_f \neq 0 : \mathbf{x}_f^T \mathbf{L}_f \mathbf{x}_f \neq 0$ .

Therefore,  $\mathbf{L}_f$  is positive definite as  $\forall \mathbf{x}_f \neq 0 : \mathbf{x}_f^T \mathbf{L}_f \mathbf{x}_f > 0$ .  $\square$

**Lemma 3.1.2.** It holds that  $\dot{\mathbf{x}}_f = -\mathbf{L}_f \mathbf{x}_f$  is globally exponentially stable (i.e., converges to 0 exponentially).

*Proof.* As  $\mathbf{L}_f$  is symmetric and positive definite by Lemma 3.1.1, its eigenvalues are real and positive. Therefore,  $-\mathbf{L}_f$  have real and negative eigenvalues, which is the condition of a globally exponentially stable behavior.  $\square$

**Theorem 3.1.1** (Containment optimality). Given a leader-follower network such that:

Containment optimality

- Followers run Laplacian dynamics,
- Leaders are stationary,
- The interaction graph  $G$  is fixed, undirected, and connected.

It holds that all followers asymptotically converge to a state (not necessarily the same) within the convex hull containing the leaders.

*Proof.* The proof is done in two parts:

**Unique globally asymptotically stable equilibrium** We want to prove that the followers' state  $\mathbf{x}_f(t)$  converges to some value  $\mathbf{x}_{f,E}$  for any initial state. The equilibrium can be found by solving:

$$0 = -\mathbf{L}_f \mathbf{x}_{f,E} - \mathbf{L}_{fl} \mathbf{x}_l$$

where  $\dot{\mathbf{x}}_f = 0$  (i.e., reached convergence) and  $\mathbf{x}_{f,E}$  is the equilibrium state.

By Lemma 3.1.1,  $\mathbf{L}_f$  is positive definite and thus invertible, therefore, we have that:

$$\mathbf{x}_{f,E} = -\mathbf{L}_f^{-1} \mathbf{L}_{fl} \mathbf{x}_l$$

Let  $\mathbf{e}(t) = \mathbf{x}_f(t) - \mathbf{x}_{f,E}$  (intuitively, the distance to equilibrium). As the rate of change of  $\mathbf{e}(t)$  depends only on  $\mathbf{x}_f(t)$  (i.e.,  $\dot{\mathbf{e}}(t) = \dot{\mathbf{x}}_f(t)$ ), we have that:

$$\begin{aligned} \dot{\mathbf{e}}(t) &= \dot{\mathbf{x}}_f(t) \\ &= -\mathbf{L}_f \mathbf{x}_f(t) - \mathbf{L}_{fl} \mathbf{x}_l \\ &= -\mathbf{L}_f(\mathbf{e}(t) + \mathbf{x}_{f,E}) - \mathbf{L}_{fl} \mathbf{x}_l \\ &= -\mathbf{L}_f \mathbf{e}(t) + \cancel{\mathbf{L}_f \mathbf{L}_f^{-1} \mathbf{L}_{fl} \mathbf{x}_l} - \cancel{\mathbf{L}_{fl} \mathbf{x}_l} \end{aligned}$$

**Lemma 3.1.3.** Any equilibrium or trajectory based on an LTI system enjoys the same stability property of that system.

As Lemma 3.1.2 states that  $\dot{\mathbf{x}}_f = -\mathbf{L}_f \mathbf{x}_f$  is globally asymptotically stable, by Lemma 3.1.3, it holds that  $\dot{\mathbf{e}}(t) = -\mathbf{L}_f \mathbf{e}(t)$  is also a globally asymptotically stable system and  $\mathbf{x}_{f,E}$  is the unique globally stable equilibrium of the followers' dynamics.

**Equilibrium within convex hull** We want to prove that each element of  $\mathbf{x}_{f,E}$  falls within the convex hull of the leaders.

For simplicity, let us denote the states vector as  $\mathbf{x}_E = [\mathbf{x}_{f,E} \ \mathbf{x}_l]^T$  and its  $i$ -th component as  $x_{E,i}$ .

The dynamics at convergence of the  $i$ -th follower is:

$$0 = - \sum_{j=1}^N a_{ij} (x_{E,i} - x_{E,j}) \quad \forall i \in \{1, \dots, N_f\}$$

Therefore, we have that:

$$\begin{aligned} \left( \sum_{k=1}^N a_{ik} \right) x_{E,i} &= \sum_{j=1}^N a_{ij} x_{E,j} \quad \forall i \in \{1, \dots, N_f\} \\ x_{E,i} &= \sum_{j=1}^N \frac{a_{ij}}{\sum_{k=1}^N a_{ik}} x_{E,j} \quad \forall i \in \{1, \dots, N_f\} \end{aligned}$$

As  $\frac{a_{ij}}{\sum_{k=1}^N a_{ik}}$  define a convex combination (i.e., sum of all of them is 1), each follower's equilibrium  $x_{E,i}$  belongs to the convex hull of all the other agents (both leaders and followers). As leaders are stationary, they are not affected by this constraint and it can be concluded that followers' equilibria fall within the convex hull of the leaders.

□

**Remark** (Leader-follower containment weakness). The final part of the proof of Theorem 3.1.1 also shows that if there is an adversarial follower that does not change its state, all others will converge towards it.

## 3.2 Containment with non-static leaders

**Containment with non-static leaders** Containment problem where leaders' dynamics is a non-zero constant (i.e., they also move):

$$\begin{aligned} \dot{\mathbf{x}}_f(t) &= -\mathbf{L}_f \mathbf{x}_f(t) - \mathbf{L}_{fl} \mathbf{x}_l(t) & \mathbf{x}_f(0) &= \mathbf{x}_f^{(0)} \\ \dot{\mathbf{x}}_l(t) &= \mathbf{v}_0 & \mathbf{x}_l(0) &= \mathbf{x}_l^{(0)} \end{aligned}$$

where  $\mathbf{v}_0$  is the leaders' velocity.

**Theorem 3.2.1** (Containment with non-static leaders non-equilibrium). Naive containment with non-static leaders does not have an equilibrium.

*Proof.* Ideally, the equilibria for followers' and leader's dynamics are:

$$\begin{aligned} 0 &= -\mathbf{L}_f \mathbf{x}_{f,E} - \mathbf{L}_{fl} \mathbf{x}_{l,E} \\ 0 &= \mathbf{v}_0 \end{aligned}$$

Let's define the containment error (can also be seen as the error to reach the followers' equilibrium) as:

$$\mathbf{e}(t) = \mathbf{L}_f \mathbf{x}_f(t) + \mathbf{L}_{fl} \mathbf{x}_l(t)$$

Its dynamics depends on the ones of the followers' and leaders':

$$\begin{aligned} \dot{\mathbf{e}}(t) &= \mathbf{L}_f \dot{\mathbf{x}}_f(t) + \mathbf{L}_{fl} \dot{\mathbf{x}}_l(t) \\ &= \mathbf{L}_f (-\mathbf{L}_f \mathbf{x}_f(t) - \mathbf{L}_{fl} \mathbf{x}_l(t)) + \mathbf{L}_{fl} \mathbf{v}_0 \\ &= -\mathbf{L}_f \mathbf{e}(t) + \mathbf{L}_{fl} \mathbf{v}_0 \end{aligned}$$

By inspecting the value of the containment error  $\mathbf{e}(t)$  when it reaches equilibrium, we have that:

$$\begin{aligned} 0 &= \dot{\mathbf{e}}(t) \\ \iff 0 &= -\mathbf{L}_f \mathbf{e}(t) + \mathbf{L}_{fl} \mathbf{v}_0 \\ \iff \mathbf{e}(t) &= \mathbf{L}_f^{-1} \mathbf{L}_{fl} \mathbf{v}_0 \end{aligned}$$

Containment with non-static leaders

There are two cases:

$$\mathbf{e}(t) = \begin{cases} 0 & \text{if } \mathbf{v}_0 = 0 \text{ (i.e., same case of Theorem 3.1.1)} \\ \mathbf{L}_f^{-1} \mathbf{L}_{fl} \mathbf{v}_0 & \text{if } \mathbf{v}_0 \neq 0 \end{cases}$$

Therefore, when leaders are non-static, the containment error converges to a non-zero constant. Thus, followers' equilibrium is never reached (i.e., they keep moving) and the containment problem cannot be solved.  $\square$

### 3.3 Containment with non-static leaders and integral action

**Containment with non-static leaders and integral action** Leader-follower dynamics defined as:

$$\begin{aligned} \dot{\mathbf{x}}_f(t) &= -\mathbf{L}_f \mathbf{x}_f(t) - \mathbf{L}_{fl} \mathbf{x}_l(t) + \mathbf{u}_f(t) & \mathbf{x}_f(0) &= \mathbf{x}_f^{(0)} \\ \dot{\mathbf{x}}_l(t) &= \mathbf{v}_0 & \mathbf{x}_l(0) &= \mathbf{x}_l^{(0)} \end{aligned}$$

Containment with non-static leaders and integral action

where  $\mathbf{u}_f(t)$  is a distributed control action (can be seen as a correction) that processes the containment error  $\mathbf{e}(t)$ . It is composed of a proportional controller (i.e., value proportional to the error) and an integral controller (i.e., value proportional to the integral of the error):

$$\mathbf{u}_f(t) = \mathbf{K}_P \mathbf{e}(t) + \mathbf{K}_I \int_0^t \mathbf{e}(\tau) d\tau$$

where  $\mathbf{K}_P$  and  $\mathbf{K}_I$  are coefficients for the proportional and integral controller, respectively.

By defining a proxy  $\xi$  for the integral of the error (i.e., sort of accumulator) as follows:

$$\begin{aligned} \dot{\xi}(t) &= \mathbf{e}(t) \\ &= \mathbf{L}_f \mathbf{x}(t) + \mathbf{L}_{fl} \mathbf{x}_l(t) & \xi(0) &= \xi^{(0)} \end{aligned}$$

The control action can be defined as:

$$\mathbf{u}_f(t) = \mathbf{K}_P \mathbf{e}(t) + \mathbf{K}_I \xi(t)$$

In the simplest case,  $\mathbf{u}_f(t)$  is a pure integral control where  $\mathbf{K}_I = -\kappa_I \mathbf{I}$ ,  $\kappa_I > 0$  is a sparse matrix (e.g., diagonal) and  $\mathbf{K}_P = 0$ . The overall system can be defined in matrix form as:

$$\begin{bmatrix} \dot{\mathbf{x}}_f(t) \\ \dot{\mathbf{x}}_l(t) \\ \dot{\xi}(t) \end{bmatrix} = \begin{bmatrix} -\mathbf{L}_f & -\mathbf{L}_{fl} & \mathbf{K}_I \\ 0 & 0 & 0 \\ \mathbf{L}_f & \mathbf{L}_{fl} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_f(t) \\ \mathbf{x}_l(t) \\ \xi(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{I} \\ 0 \end{bmatrix} \mathbf{v}_0$$

**Remark.** The value of this formulation of the control action for an agent  $i$  is:

$$u_{F_i}(t) = \kappa_I \xi_i(t)$$

It can be seen that it is computable as a distributed system as  $\kappa_I$  is constant and  $\xi_i(t)$  is based on the Laplacian (i.e., it is sufficient to look up the neighbors' states).

**Theorem 3.3.1** (Containment with non-static leaders and integral action optimality).  
With the integral action, containment with non-static leaders converges to a valid solution.

### 3.4 Containment with discrete-time

**Containment with discrete-time** Containment can be discretized using the forward-Euler discretization. Its dynamics is defined as:

$$\begin{aligned}\dot{x}_i(t) &= - \sum_{j \in \mathcal{N}_i} a_{ij}(x_i(t) - x_j(t)) \quad \forall i \in \{1, \dots, N_f\} \\ \dot{x}_i(t) &= 0 \quad \forall i \in \{N_f + 1, \dots, N\}\end{aligned}$$

And the followers' states are sampled with a time-step  $\varepsilon > 0$  while the leaders' is constant:

$$\begin{aligned}x_i^{k+1} &= x_i(t)|_{t=(k+1)\varepsilon} \\ &= x_i^k + \varepsilon \dot{x}_i(t)|_{t=k\varepsilon} \\ &= \left(1 - \varepsilon \sum_{j \in \mathcal{N}_i} a_{ij}\right) x_i^k + \varepsilon \sum_{j \in \mathcal{N}_i} a_{ij} x_j^k \quad \forall i \in \{1, \dots, N_f\} \\ x_i^{k+1} &= x_i^k \quad \forall i \in \{N_f + 1, \dots, N\}\end{aligned}$$

In matrix form, it can be defined as:

$$\begin{bmatrix} \mathbf{x}_f^{k+1} \\ \mathbf{x}_l^{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{I} - \varepsilon \mathbf{L}_f & -\varepsilon \mathbf{L}_{fl} \\ 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x}_f^k \\ \mathbf{x}_l^k \end{bmatrix}$$

### 3.5 Containment with multivariate states

**Containment with multivariate states** With multivariate states, it can be shown that the dynamics is described as:

$$\dot{\mathbf{x}}(t) = -\mathbf{L} \otimes \mathbf{I}_d \mathbf{x}(t)$$

where  $\otimes$  is the Kronecker product (i.e., apply the same matrix across each dimension).

Containment with discrete-time

Containment with multivariate states

# 4 Optimization

## 4.1 Definitions

### 4.1.1 Unconstrained optimization

**Unconstrained optimization** Problem of form:

$$\min_{\mathbf{z} \in \mathbb{R}^d} l(\mathbf{z})$$

where  $l : \mathbb{R}^d \rightarrow \mathbb{R}$  is the cost function and  $\mathbf{z}$  the decision variables.

**Theorem 4.1.1** (First-order necessary condition of optimality). Given a point  $\mathbf{z}^*$  and a cost function  $l : \mathbb{R}^d \rightarrow \mathbb{R}$  such that  $l \in C^1$  in  $B(\mathbf{z}^*, \varepsilon)$  (i.e., neighbors of  $\mathbf{z}^*$  within a radius  $\varepsilon$ ), it holds that:

$$\mathbf{z}^* \text{ is local minimum } \Rightarrow \nabla l(\mathbf{z}^*) = 0$$

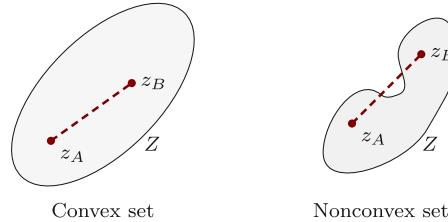
**Theorem 4.1.2** (Second-order necessary condition of optimality). Given a point  $\mathbf{z}^*$  and a cost function  $l : \mathbb{R}^d \rightarrow \mathbb{R}$  such that  $l \in C^2$  in  $B(\mathbf{z}^*, \varepsilon)$ , it holds that:

$$\mathbf{z}^* \text{ is local minimum } \Rightarrow \nabla^2 l(\mathbf{z}^*) \geq 0 \text{ (i.e., positive semidefinite)}$$

### 4.1.2 Convexity

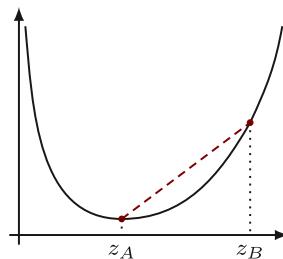
**Convex set** A set  $Z \subseteq \mathbb{R}^d$  is convex if it holds that:

$$\forall \mathbf{z}_A, \mathbf{z}_B \in Z : \left( \forall \alpha \in [0, 1] : (\alpha \mathbf{z}_A + (1 - \alpha) \mathbf{z}_B) \in Z \right)$$



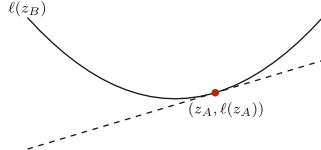
**Convex function** Given a convex set  $Z \subseteq \mathbb{R}^d$ , a function  $l : Z \rightarrow \mathbb{R}$  is convex if it holds that:

$$\forall \mathbf{z}_A, \mathbf{z}_B \in Z : \left( \forall \alpha \in [0, 1] : l(\alpha \mathbf{z}_A + (1 - \alpha) \mathbf{z}_B) \leq \alpha l(\mathbf{z}_A) + (1 - \alpha) l(\mathbf{z}_B) \right)$$



**Remark.** Given a differentiable and convex function  $l : Z \rightarrow \mathbb{R}$ , it holds that any of its points lie above all its tangents:

$$\forall \mathbf{z}_A, \mathbf{z}_B \in Z : l(\mathbf{z}_B) \geq l(\mathbf{z}_A) + \nabla l(\mathbf{z}_A)^T (\mathbf{z}_B - \mathbf{z}_A)$$



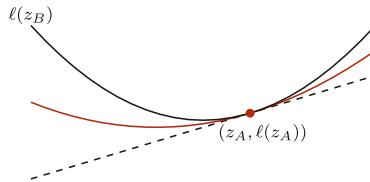
**Strongly convex function** Given a convex set  $Z \subseteq \mathbb{R}^d$ , a function  $l : Z \rightarrow \mathbb{R}$  is strongly convex with parameter  $\mu > 0$  if it holds that:

$$\forall \mathbf{z}_A, \mathbf{z}_B \in Z, \mathbf{z}_A \neq \mathbf{z}_B : \left( \forall \alpha \in (0, 1) : l(\alpha \mathbf{z}_A + (1 - \alpha) \mathbf{z}_B) < \alpha l(\mathbf{z}_A) + (1 - \alpha) l(\mathbf{z}_B) - \frac{1}{2} \mu \alpha (1 - \alpha) \|\mathbf{z}_A - \mathbf{z}_B\|^2 \right)$$

Intuitively, it is strictly convex and grows as fast as a quadratic function.

**Remark.** Given a differentiable and  $\mu$ -strongly convex function  $l : Z \rightarrow \mathbb{R}$ , it holds that any of its points lie above all the paraboloids with curvature determined by  $\mu$  and tangent to a point of the function:

$$\forall \mathbf{z}_A, \mathbf{z}_B \in Z : l(\mathbf{z}_B) \geq l(\mathbf{z}_A) + \nabla l(\mathbf{z}_A)^T (\mathbf{z}_B - \mathbf{z}_A) + \frac{\mu}{2} \|\mathbf{z}_B - \mathbf{z}_A\|^2$$



A geometric interpretation is that strong convexity imposes a quadratic lower-bound to the function.

**Lemma 4.1.1** (Convexity and gradient monotonicity). Given a differentiable and convex function  $l$ , its gradient  $\nabla l$  is a monotone operator, which means that it satisfies:

$$\forall \mathbf{z}_A, \mathbf{z}_B : (\nabla l(\mathbf{z}_A) - \nabla l(\mathbf{z}_B))^T (\mathbf{z}_A - \mathbf{z}_B) \geq 0$$

**Lemma 4.1.2** (Strict convexity and gradient monotonicity). Given a differentiable and strictly convex function  $l$ , its gradient  $\nabla l$  is a strictly monotone operator, which means that it satisfies:

$$\forall \mathbf{z}_A, \mathbf{z}_B : (\nabla l(\mathbf{z}_A) - \nabla l(\mathbf{z}_B))^T (\mathbf{z}_A - \mathbf{z}_B) > 0$$

**Lemma 4.1.3** (Strong convexity and gradient monotonicity). Given a differentiable and  $\mu$ -strongly convex function  $l$ , its gradient  $\nabla l$  is a strongly monotone operator, which means that it satisfies:

$$\forall \mathbf{z}_A, \mathbf{z}_B : (\nabla l(\mathbf{z}_A) - \nabla l(\mathbf{z}_B))^T (\mathbf{z}_A - \mathbf{z}_B) \geq \mu \|\mathbf{z}_A - \mathbf{z}_B\|^2$$

**Lipschitz continuity** Given a function  $l$ , it is Lipschitz continuous with parameter  $L > 0$  if:

$$\forall \mathbf{z}_A, \mathbf{z}_B : \|l(\mathbf{z}_A) - l(\mathbf{z}_B)\| \leq L \|\mathbf{z}_A - \mathbf{z}_B\|$$

Strongly convex function

Convexity and gradient monotonicity

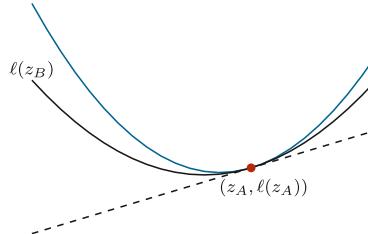
Strict convexity and gradient monotonicity

Strong convexity and gradient monotonicity

Lipschitz continuity

**Remark.** Given a differentiable function  $l$  with  $L$ -Lipschitz continuous gradient  $\nabla l$ , it holds that any of its points lie below all the paraboloids with curvature determined by  $L$  and tangent to a point of the function:

$$\forall \mathbf{z}_A, \mathbf{z}_B \in Z : l(\mathbf{z}_B) \leq l(\mathbf{z}_A) + \nabla l(\mathbf{z}_A)^T (\mathbf{z}_B - \mathbf{z}_A) + \frac{L}{2} \|\mathbf{z}_B - \mathbf{z}_A\|^2$$



A geometric interpretation is that Lipschitz continuity of the gradient imposes a quadratic upper-bound to the function.

**Lemma 4.1.4** (Convexity and Lipschitz continuity of gradient). Given a differentiable convex function  $l$  with  $L$ -Lipschitz continuous gradient  $\nabla l$ , its gradient is a co-coercive operator, which means that it satisfies:

$$\forall \mathbf{z}_A, \mathbf{z}_B : (\nabla l(\mathbf{z}_A) - \nabla l(\mathbf{z}_B))^T (\mathbf{z}_A - \mathbf{z}_B) \geq \frac{1}{L} \|\nabla l(\mathbf{z}_A) - \nabla l(\mathbf{z}_B)\|^2$$

**Lemma 4.1.5** (Strong convexity and Lipschitz continuity of gradient). Given a differentiable  $\mu$ -strongly convex function  $l$  with  $L$ -Lipschitz continuous gradient  $\nabla l$ , its gradient is a strongly co-coercive operator, which means that it satisfies:

$$\forall \mathbf{z}_A, \mathbf{z}_B : (\nabla l(\mathbf{z}_A) - \nabla l(\mathbf{z}_B))^T (\mathbf{z}_A - \mathbf{z}_B) \geq \underbrace{\frac{\mu L}{\mu + L}}_{\gamma_1} \|\mathbf{z}_A - \mathbf{z}_B\|^2 + \underbrace{\frac{1}{\mu + L}}_{\gamma_2} \|\nabla l(\mathbf{z}_A) - \nabla l(\mathbf{z}_B)\|^2$$

Convexity and Lipschitz continuity of gradient

Strong convexity and Lipschitz continuity of gradient

## 4.2 Iterative descent methods

**Theorem 4.2.1.** Given a convex function  $l$ , it holds that a local minimum of  $l$  is also global.

Moreover, in the unconstrained optimization case, the first-order necessary condition of optimality is sufficient for a global minimum.

**Theorem 4.2.2.** Given a convex function  $l$ , it holds that  $\mathbf{z}^*$  is a global minimum if and only if  $\nabla f(\mathbf{z}^*) = 0$ .

**Iterative descent** Given a function  $l$  and an initial guess  $\mathbf{z}^0$ , an iterative descent algorithm iteratively moves to new points  $\mathbf{z}^k$  such that:

$$\forall k \in \mathbb{N} : l(\mathbf{z}^{k+1}) < l(\mathbf{z}^k)$$

Iterative descent

### 4.2.1 Gradient method

**Gradient method** Algorithm that given the function  $l$  to minimize and the initial guess  $\mathbf{z}^0$ , computes the update as:

$$\mathbf{z}^{k+1} = \mathbf{z}^k - \alpha^k \nabla l(\mathbf{z}^k)$$

Gradient method

where  $\alpha^k > 0$  is the step size and  $-\nabla l(\mathbf{z}^k)$  is the step direction.

**Theorem 4.2.3.** For a sufficiently small  $\alpha^k > 0$ , the gradient method is an iterative descent algorithm:

$$l(\mathbf{z}^{k+1}) < l(\mathbf{z}^k)$$

*Proof.* Consider the first-order Taylor approximation of  $l(\mathbf{z}^{k+1})$  about  $\mathbf{z}^k$ :

$$\begin{aligned} l(\mathbf{z}^{k+1}) &= l(\mathbf{z}^k) + \nabla l(\mathbf{z}^k)^T (\mathbf{z}^{k+1} - \mathbf{z}^k) + o(\|\mathbf{z}^{k+1} - \mathbf{z}^k\|) \\ &= l(\mathbf{z}^k) - \alpha^k \|\nabla l(\mathbf{z}^k)\|^2 + o(\alpha^k) \end{aligned}$$

Therefore,  $l(\mathbf{z}^{k+1}) < l(\mathbf{z}^k)$  for some  $\alpha^k$ . □

**Remark (Step size choice).** Possible choices for the step size are:

Step size choice

**Constant**  $\forall k \in \mathbb{N} : \alpha^k = \alpha > 0$ .

**Diminishing**  $\alpha^k \xrightarrow{k \rightarrow \infty} 0$ . To avoid decreasing the step too much, a typical choice is an  $\alpha^k$  such that:

$$\sum_{k=0}^{\infty} \alpha^k = \infty \quad \sum_{k=0}^{\infty} (\alpha^k)^2 < \infty$$

**Line search** Algorithmic methods such as the Armijo rule.

**Generalized gradient method** Gradient method where the update rule is generalized as:

Generalized gradient method

$$\mathbf{z}^{k+1} = \mathbf{z}^k - \alpha^k \mathbf{D}^k \nabla l(\mathbf{z}^k)$$

where  $\mathbf{D}^k \in \mathbb{R}^{d \times d}$  is uniformly positive definite (i.e.,  $\delta_1 \mathbf{I} \leq \mathbf{D}^k \leq \delta_2 \mathbf{I}$  for some  $\delta_2 \geq \delta_1 > 0$ ).

Possible choices for  $\mathbf{D}^k$  are:

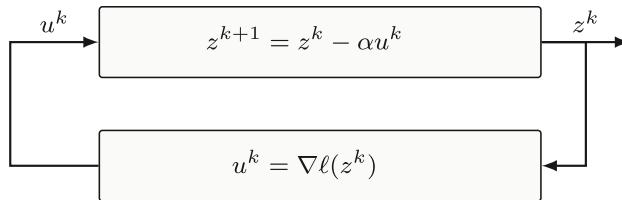
- Steepest descent:  $\mathbf{D}^k = \mathbf{I}$ .
- Newton's method:  $\mathbf{D}^k = (\nabla^2 l(\mathbf{z}^k))^{-1}$ .
- Quasi-Newton method:  $\mathbf{D}^k = (H(\mathbf{z}^k))^{-1}$ , where  $H(\mathbf{z}^k) \approx \nabla^2 l(\mathbf{z}^k)$ .

**Gradient method as discrete-time integrator with feedback** The gradient method can be interpreted as a discrete-time integrator with a feedback loop. This means that it is composed of:

Gradient method as discrete-time integrator with feedback

**Integrator** A linear system that defines the update:  $\mathbf{z}^{k+1} = \mathbf{z}^k - \alpha \mathbf{u}^k$ .

**Plant** A non-linear (bounded) function whose output is re-injected into the integrator. In this case, it is the gradient:  $\mathbf{u}^k = \nabla l(\mathbf{z}^k)$ .



**Theorem 4.2.4 (Gradient method convergence).** Consider a function  $l$  such that:

Gradient method convergence

- $\nabla l$  is  $L$ -Lipschitz continuous,

- The step size is constant or diminishing.

Let  $\{\mathbf{z}^k\}_{k \in \mathbb{N}}$  be the (bounded) sequence generated by the gradient method. It holds that every limit point  $\bar{\mathbf{z}}$  of the sequence  $\{\mathbf{z}^k\}_{k \in \mathbb{N}}$  is a stationary point (i.e.,  $\nabla l(\bar{\mathbf{z}}) = 0$ ).

In addition, if  $l$  is  $\mu$ -strongly convex and the step size is constant, then the convergence rate of the sequence  $\{\mathbf{z}^k\}_{k \in \mathbb{N}}$  is exponential (also said geometric or linear):

$$\|\mathbf{z}^k - \mathbf{z}^*\| \leq M\rho^k$$

where  $\rho \in (0, 1)$  and  $M > 0$  depends on  $\mu$ ,  $L$ , and  $\|\mathbf{z}^0 - \mathbf{z}^*\|$ .

*Proof.* We need to prove the two parts of the theorem:

1. We want to prove that any limit point of the sequence generated by the gradient method is a stationary point.

In other words, by considering the gradient method as an integrator with feedback, we want to analyze the equilibrium of the system. Assume that the system converges to some equilibrium  $\mathbf{z}_E$ . To be an equilibrium, it must be that the feedback loop stopped updating the system (i.e.,  $\mathbf{u}^k = 0$  for  $k$  after some threshold) so that:

$$\mathbf{z}_E = \mathbf{z}_E - \alpha \nabla l(\mathbf{z}_E)$$

Therefore, an equilibrium point is necessarily a stationary point of  $l$  as it must be that  $\nabla l(\mathbf{z}_E) = 0$ .

2. We want to prove that if  $l$  is  $\mu$ -strongly convex and the step size is constant, the sequence converges exponentially.

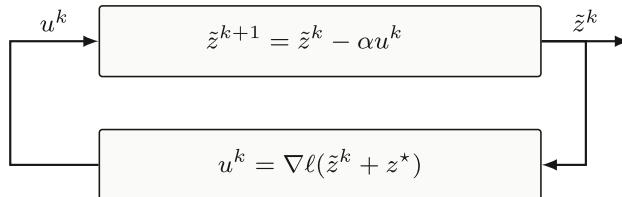
| **Remark.** As  $l$  is convex, its equilibrium is also the global minimum  $\mathbf{z}^*$ .

Consider the following change in coordinates (i.e., a translation):

$$\begin{aligned} \mathbf{z}^k &\mapsto \tilde{\mathbf{z}}^k \\ \text{with } \tilde{\mathbf{z}}^k &= \mathbf{z}^k - \mathbf{z}_E = \mathbf{z}^k - \mathbf{z}^* \end{aligned}$$

The system in the new coordinates becomes:

$$\begin{aligned} \tilde{\mathbf{z}}^{k+1} &= \tilde{\mathbf{z}}^k - \alpha \mathbf{u}^k \\ \mathbf{u}^k &= \nabla l(\mathbf{z}^k) \\ &= \nabla l(\tilde{\mathbf{z}}^k + \mathbf{z}^*) \\ &= \nabla l(\tilde{\mathbf{z}}^k + \mathbf{z}^*) - \nabla l(\mathbf{z}^*) \quad \nabla l(\mathbf{z}^*) = 0, \text{ but useful for Lemma 4.1.5} \end{aligned}$$



| **Remark.** As  $l$  is strongly convex and its gradient Lipschitz continuous, by Lemma 4.1.5 it holds that:

$$-(\mathbf{u}^k)^T \tilde{\mathbf{z}}^k \leq -\gamma_1 \|\tilde{\mathbf{z}}^k\|^2 - \gamma_2 \|\mathbf{u}^k\|^2$$

Consider a Lyapunov function  $V : \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$  defined as:

$$V(\tilde{\mathbf{z}}) = \|\tilde{\mathbf{z}}\|^2$$

It holds that:

$$\begin{aligned} V(\tilde{\mathbf{z}}^{k+1}) - V(\tilde{\mathbf{z}}^k) &= \|\tilde{\mathbf{z}}^{k+1}\|^2 - \|\tilde{\mathbf{z}}^k\|^2 \\ &= \|\tilde{\mathbf{z}}^k - \alpha \mathbf{u}^k\|^2 - \|\tilde{\mathbf{z}}^k\|^2 \\ &= \|\tilde{\mathbf{z}}^k\|^2 - 2\alpha (\mathbf{u}^k)^T \tilde{\mathbf{z}}^k + \alpha^2 \|\mathbf{u}^k\|^2 - \|\tilde{\mathbf{z}}^k\|^2 \\ &\leq -2\alpha \gamma_1 \|\tilde{\mathbf{z}}^k\|^2 + \alpha(\alpha - 2\gamma_2) \|\mathbf{u}^k\|^2 \end{aligned} \quad \text{Lemma 4.1.5}$$

By choosing  $\alpha \leq 2\gamma_2$ , we have that:

$$\begin{aligned} V(\tilde{\mathbf{z}}^{k+1}) - V(\tilde{\mathbf{z}}^k) &\leq -2\alpha \gamma_1 \|\tilde{\mathbf{z}}^k\|^2 \\ \iff \|\tilde{\mathbf{z}}^{k+1}\|^2 - \|\tilde{\mathbf{z}}^k\|^2 &\leq -2\alpha \gamma_1 \|\tilde{\mathbf{z}}^k\|^2 \\ \iff \|\tilde{\mathbf{z}}^{k+1}\|^2 &\leq (1 - 2\alpha \gamma_1) \|\tilde{\mathbf{z}}^k\|^2 \end{aligned}$$

Finally, as the gradient method is an iterative descent algorithm, it holds that:

$$\begin{aligned} \|\tilde{\mathbf{z}}^{k+1}\|^2 &\leq (1 - 2\alpha \gamma_1) \|\tilde{\mathbf{z}}^k\|^2 \\ &\leq \dots \\ &\leq (1 - 2\alpha \gamma_1)^k \|\tilde{\mathbf{z}}^0\|^2 \end{aligned}$$

Therefore, the sequence  $\{\tilde{\mathbf{z}}^k\}_{k \in \mathbb{R}}$  goes exponentially fast to zero and we have shown that:

$$\begin{aligned} \|\mathbf{z}^{k+1} - \mathbf{z}^*\|^2 &\leq (1 - 2\alpha \gamma_1)^k \|\mathbf{z}^0 - \mathbf{z}^*\|^2 \\ &= \rho^k M \end{aligned}$$

□

**Remark** (Gradient method for a quadratic function). Given the problem of minimizing a quadratic function:

$$\min_{\mathbf{z}} \frac{1}{2} \mathbf{z}^T \mathbf{Q} \mathbf{z} + \mathbf{r}^T \mathbf{z} \quad \nabla l = \mathbf{Q} \mathbf{z} + \mathbf{r}$$

Gradient method for a quadratic function

The gradient method can be reduced to an affine linear system:

$$\begin{aligned} \mathbf{z}^{k+1} &= \mathbf{z}^k - \alpha(\mathbf{Q} \mathbf{z}^k + \mathbf{r}) \\ &= (\mathbf{I} - \alpha \mathbf{Q}) \mathbf{z}^k - \alpha \mathbf{r} \end{aligned}$$

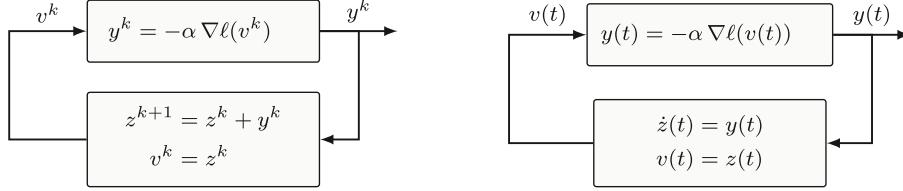
For a sufficiently small  $\alpha$ , the matrix  $(\mathbf{I} - \alpha \mathbf{Q})$  is Schur (i.e.,  $\forall \boldsymbol{\rho}, |\boldsymbol{\rho}| < 1 : \sum_{i=0}^{\infty} \boldsymbol{\rho}^i = (1 - \boldsymbol{\rho})^{-1}$ ). Therefore, the solution can be computed in closed form as:

$$\begin{aligned} \mathbf{z}^k &= (\mathbf{I} - \alpha \mathbf{Q})^k \mathbf{z}^0 - \alpha \sum_{\tau=0}^{k-1} (\mathbf{I} - \alpha \mathbf{Q})^\tau \mathbf{r} \\ &\xrightarrow{k \rightarrow \infty} -\alpha \left( \sum_{\tau=0}^{\infty} (\mathbf{I} - \alpha \mathbf{Q})^\tau \right) \mathbf{r} = -\mathbf{Q}^{-1} \mathbf{r} \end{aligned}$$

**Remark** (Gradient flow). By inverting the integrator and plant of the discrete-time integrator of the gradient method, and considering the continuous-time case, the result is the gradient flow:

$$\dot{\mathbf{z}}(t) = -\nabla l(\mathbf{z}(t))$$

which has a solution if the vector field is Lipschitz continuous.



#### 4.2.2 Accelerated gradient methods

**Heavy-ball method** Given  $\eta^0$  and  $\eta^{-1}$ , the algorithm is defined as:

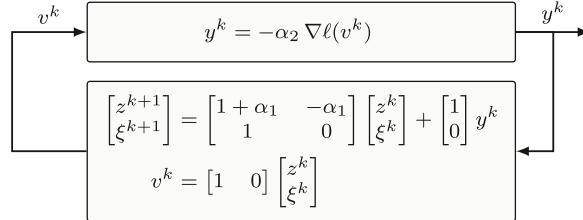
Heavy-ball method

$$\eta^{k+1} = \eta^k + \alpha_1(\eta^k - \eta^{k-1}) - \alpha_2 \nabla l(\eta^k)$$

with  $\alpha_1, \alpha_2 > 0$ .

**Remark.** With  $\alpha_1 = 0$ , the algorithm is reduced to the gradient method with step size  $\alpha_2$ .

**Remark.** The algorithm admits a state-space representation as a discrete-time integrator with a feedback loop:



Note that the matrix  $\begin{bmatrix} 1 + \alpha_1 & -\alpha_1 \\ 1 & 0 \end{bmatrix}$  is row stochastic.

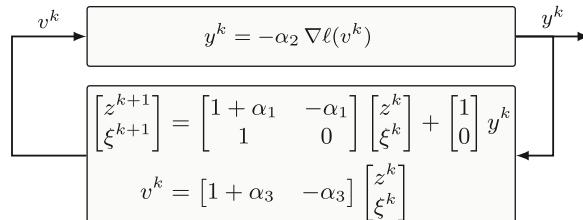
**Generalized heavy-ball method** Given  $\zeta^0$  and  $\zeta^{-1}$ , the algorithm is defined as:

Generalized heavy-ball method

$$\zeta^{k+1} = \zeta^k + \alpha_1(\zeta^k - \zeta^{k-1}) - \alpha_2 \nabla l(\zeta^k + \alpha_3(\zeta^k - \zeta^{k-1}))$$

with  $\alpha_1, \alpha_2, \alpha_3 > 0$ .

**Remark.** The algorithm admits a state-space representation as a discrete-time integrator with a feedback loop:



## 4.3 Cost-coupled optimization

**Cost-coupled optimization** Problem of minimizing  $N$  cost functions  $l_i : \mathbb{R}^d \rightarrow \mathbb{R}$ , each local and private to an agent:

$$\min_{\mathbf{z} \in \mathbb{R}^d} \sum_{i=1}^N l_i(\mathbf{z})$$

Cost-coupled optimization

### 4.3.1 Learning paradigms

**Federated learning** Problem where  $N$  agents with their local and private data  $\mathcal{D}^i$  want to learn a common set of parameters  $\mathbf{z}^*$  based on the same loss function (evaluated on different data points):

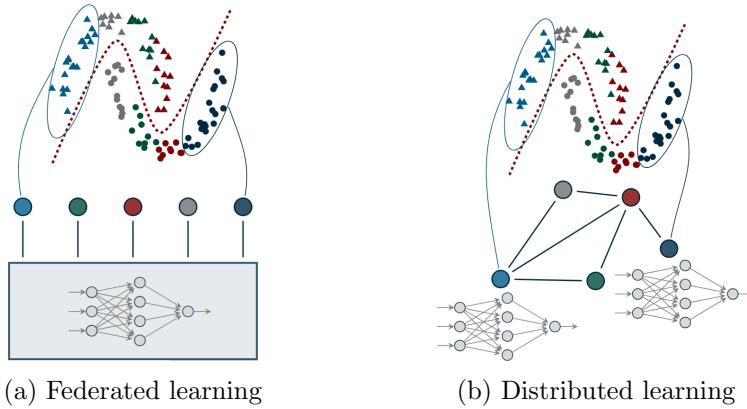
$$\min_{\mathbf{z}} \sum_{i=1}^N l(\mathbf{z}; \mathcal{D}^i)$$

Federated learning

A centralized parameter server (master) is responsible for aggregating the estimates of the agents (e.g., pick some nodes and average them).

**Distributed learning** Federated learning where there is no centralized entity and agents communicate with their neighbors only.

Distributed learning



## 4.4 Federated learning

### 4.4.1 Batch gradient method

**Batch gradient method** Compute the direction for the gradient method by considering all the losses:

$$\mathbf{z}^{k+1} = \mathbf{z}^k - \alpha \sum_{i=1}^N \nabla l_i(\mathbf{z}^k)$$

Batch gradient method

| **Remark.** Computation in this way can be expensive.

### 4.4.2 Incremental gradient method

**Incremental gradient method** At each iteration  $k$ , compute the direction by considering the loss of a single agent  $i^k$ :

$$\mathbf{z}^{k+1} = \mathbf{z}^k - \alpha \nabla l_{i^k}(\mathbf{z}^k)$$

Incremental gradient method

**Remark.** Two possible rules to select the agent at each iteration are:

**Cyclic**  $i^k = 1, 2, \dots, N, 1, 2, \dots, N, \dots$ , or cyclic in any order (essentially cyclic).

**Randomized** Draw  $i^k$  from a uniform distribution.

**Remark.** A single gradient is not necessarily a descent direction.

**Theorem 4.4.1.** If the step size is diminishing, the incremental gradient method converges.

#### 4.4.3 Stochastic gradient descent

**Stochastic gradient descent (SGD)** Instance of incremental gradient method where the selection rule follows an unknown distribution.

The problem can be formulated as:

$$\min_{\mathbf{z} \in \mathbb{R}^d} \mathbb{E}_{\mathcal{W}}[l(\mathbf{z}, \mathcal{W})]$$

where  $\mathcal{W}$  is a random variable with possibly an unknown distribution.

It is assumed that, given any realization  $\bar{w}$  of  $\mathcal{W}$  (e.g., the index of an agent or a single data point), it is possible to obtain the gradient  $\nabla l(\bar{\mathbf{z}}, \bar{w})$  at any query point  $\bar{\mathbf{z}}$ . The optimization step at each iteration is then:

$$\mathbf{z}^{k+1} = \mathbf{z}^k - \alpha \nabla l(\mathbf{z}^k, w^k)$$

**Remark.** Monte Carlo approximation can be used to represent the expected value with a finite sequence of realizations:

$$\mathbb{E}_{\mathcal{W}}[l(\mathbf{z}, \mathcal{W})] \approx \frac{1}{K} \sum_{k=1}^K l(\mathbf{z}, w^k)$$

**Theorem 4.4.2** (SGD convergence with constant step size). Given a function  $l$  such that:

- $l$  is  $\mu$ -strongly convex with  $L$ -Lipschitz continuous gradient (i.e., bounded),
- $\nabla l(\mathbf{z}, \mathcal{W})$  is an unbiased estimate of  $\nabla_{\mathbf{z}} \mathbb{E}_{\mathcal{W}}[l(\mathbf{z}, \mathcal{W})]$ ,
- $\|\nabla l(\mathbf{z}, \mathcal{W})\| \leq M_{\nabla}$  almost surely (i.e., asymptotically with probability 1) for some  $M_{\nabla} > 0$ .

With a constant step size  $\alpha \leq \frac{1}{2}\mu$ , it holds that at any time step  $k$ :

$$\|\mathbf{z}^k - \mathbf{z}^*\| \leq \underbrace{(1 - 2\mu\alpha)^k \left( \|\mathbf{z}^0 - \mathbf{z}^*\| - \frac{\alpha M_{\nabla}^2}{2\mu} \right)}_{\text{Error term}} + \underbrace{\frac{\alpha M_{\nabla}^2}{2\mu}}_{\text{Residual term}}$$

where the error diminishes over time and the residual term is constant.

**Theorem 4.4.3** (SGD convergence with diminishing step size). With a diminishing step size, both the error and the residual converge to 0.

**Mini-batch SGD** SGD where the update at each time step  $k$  is based on a set  $\mathcal{I}^k \subset \{1, \dots, N\}$  of realizations of  $\mathcal{W}$ :

$$\mathbf{z}^{k+1} = \mathbf{z}^k - \alpha \sum_{i \in \mathcal{I}^k} \nabla l(\mathbf{z}^k, w^i)$$

Stochastic gradient descent (SGD)

SGD convergence with constant step size

SGD convergence with diminishing step size  
Mini-batch SGD

#### 4.4.4 Adaptive momentum

**Adaptive momentum (ADAM)** Method based on the first and second momentum of the gradient:

$$\begin{aligned}\mathbf{m}^{k+1} &= \beta_1 \mathbf{m}^k + (1 - \beta_1) \nabla l(\mathbf{z}^k, w^k) \\ \mathbf{v}^{k+1} &= \beta_2 \mathbf{v}^k + (1 - \beta_2) (\nabla l(\mathbf{z}^k, w^k))^2\end{aligned}$$

where  $\beta_1, \beta_2 \in (0, 1)$  are hyperparameters.

The descent direction is defined as:

$$\begin{aligned}\hat{\mathbf{m}} &= \frac{1}{1 - \beta_1^{k+1}} \mathbf{m}^{k+1} \quad \hat{\mathbf{v}} = \frac{1}{1 - \beta_2^{k+1}} \mathbf{v}^{k+1} \\ \mathbf{d}^k &= -\frac{\hat{\mathbf{m}}}{\sqrt{\hat{\mathbf{v}}} + \varepsilon}\end{aligned}$$

The update is performed as:

$$\mathbf{z}^{k+1} = \mathbf{z}^k + \alpha \mathbf{d}^k$$

Adaptive momentum (ADAM)

### 4.5 Distributed cost-coupled/consensus optimization

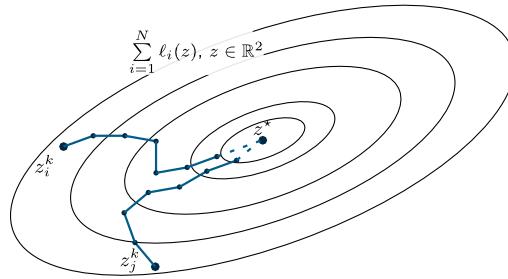
**Distributed cost-coupled optimization** Optimization problem with  $N$  agents that communicate according to a graph  $G$  aiming at learning a common set of parameters  $\mathbf{z}$  such that:

$$\min_{\mathbf{z} \in Z} \sum_{i=1}^N l_i(\mathbf{z})$$

Distributed cost-coupled optimization

where:

- Each agent  $i$  knows its loss  $l_i$  (based on its available data) and the parameter space  $Z$ ,
- At each time step  $k$ , each agent  $i$  estimates a set of parameters  $\mathbf{z}_i^k$ .



**Remark.** Using as direction the sum of the gradients of all agents is not possible as not everyone can communicate with everyone.

#### 4.5.1 Distributed gradient algorithm

**Distributed gradient algorithm** Method that estimates a (more precise) set of parameters as a weighted sum those of its neighbors' (self-loop included):

$$\mathbf{v}_i^{k+1} = \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{z}_j^k$$

Distributed gradient algorithm

Then, the update step is performed using  $\mathbf{v}_i^{k+1}$  and the agent's own local loss  $l_i$ :

$$\begin{aligned}\mathbf{z}_i^{k+1} &= \mathbf{v}_i^{k+1} - \alpha^k \nabla l_i(\mathbf{v}_i^{k+1}) \\ &= \left( \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{z}_j^k \right) - \alpha^k \nabla l_i \left( \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{z}_j^k \right)\end{aligned}$$

**Theorem 4.5.1** (Distributed gradient algorithm convergence). Assume that:

- The matrix  $\mathbf{A}$  associated to the undirected and connected communication graph  $G$  is doubly stochastic and such that  $a_{ij} > 0$ ,
- The step size is diminishing,
- Each  $l_i$  is convex, has gradients bounded by a scalar  $C_i > 0$ , and there exists at least one optimal solution.

Distributed gradient algorithm convergence

Then, the sequence of local solutions  $\{\mathbf{z}_i^k\}_{k \in \mathbb{N}}$  of each agent  $i$  produced using the distributed gradient algorithm converges to a common optimal solution  $\mathbf{z}^*$ :

$$\lim_{k \rightarrow \infty} \|\mathbf{z}_i^k - \mathbf{z}^*\| = 0$$

**Distributed projected subgradient algorithm** Distributed gradient algorithm extended to the case where  $l_i$  are non-smooth convex functions and  $\mathbf{z}$  is constrained to a closed convex set  $Z \subseteq \mathbb{R}^d$ . The distributed step is the following:

$$\begin{aligned}\mathbf{v}_i^{k+1} &= \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{z}_j^k \\ \mathbf{z}_i^{k+1} &= P_Z(\mathbf{v}_i^{k+1} - \alpha^k \tilde{\nabla} l_i(\mathbf{v}_i^{k+1}))\end{aligned}$$

Distributed projected subgradient algorithm

where  $P_Z(\cdot)$  is the Euclidean projection onto  $Z$  and  $\tilde{\nabla} l_i$  is a subgradient of  $l_i$ .

**Theorem 4.5.2** (Distributed projected subgradient algorithm convergence). Assume that:

- The adjacency matrix  $\mathbf{A}$  associated to  $G$  is doubly stochastic and  $a_{ij} > 0$ ,
- The step size is diminishing,
- Each  $l_i$  is convex, has subgradients bounded by a scalar  $C_i > 0$ , and there exists at least one optimal solution.

Distributed projected subgradient algorithm convergence

Then, each agent converges to an optimal solution  $\mathbf{z}^*$ .

**Theorem 4.5.3.** The distributed gradient algorithm does not converge with a constant step size.

*Proof idea.* We want to check whether the optimum  $\mathbf{z}^*$  with a constant step size  $\alpha$  is an equilibrium:

$$\begin{aligned}\mathbf{z}^* &= \sum_{j=1}^N a_{ij} \mathbf{z}^* - \alpha \nabla l_i \left( \sum_{j=1}^N a_{ij} \mathbf{z}^* \right) \\ &= \mathbf{z}^* - \alpha \nabla l_i(\mathbf{z}^*)\end{aligned}$$

$\mathbf{A}$  doubly stochastic and  $\mathbf{z}^*$  constant

In general,  $\nabla l_i(\mathbf{z}^*) \neq 0$  ( $\mathbf{z}^*$  is the optimum for the whole problem, but  $l_i$  depends on the subset of data available to the agent). Therefore,  $\mathbf{z}^*$  is not an equilibrium.  $\square$

#### 4.5.2 Gradient tracking algorithm

**Dynamic average consensus** Consensus algorithm where each agent measures a signal  $r_i^k$  and wants to estimate the average signal of all agents:

$$\bar{r}^k = \frac{1}{N} \sum_{i=1}^N r_i^k$$

The average signal estimated by an agent is represented by a state  $s_i^k$  and we want that  $\lim_{k \rightarrow \infty} \|s_i^k - \bar{r}^k\| = 0$ . This can be achieved using a perturbed consensus algorithm:

$$s_i^{k+1} = \underbrace{\sum_{j \in \mathcal{N}_i} a_{ij} s_j^k}_{\text{Consensus}} + \underbrace{(r_i^{k+1} - r_i^k)}_{\text{Innovation}}$$

where:

- The consensus term converges to the states average.
- The local innovation allows converging to the common signal.

**Theorem 4.5.4** (Dynamic average consensus convergence). If the first-order differences are bounded (i.e.,  $\|r_i^{k+1} - r_i^k\| \leq C_1$ ), then the tracking error is bounded by some  $C_2 > 0$ :

$$\lim_{k \rightarrow \infty} \|s_i^k - \bar{r}^k\| \leq C_2$$

Moreover, the error is zeroed if the signal becomes constant after some time  $k$  (i.e.,  $\|r_i^{k+1} - r_i^k\| \rightarrow 0$ ).

**Gradient tracking algorithm** Method that chooses the local descent direction attempting to asymptotically track the true gradient:

$$d_i^k \xrightarrow[k \rightarrow \infty]{} -\frac{1}{N} \sum_{h=1}^N \nabla l_h(\mathbf{z}_h^k)$$

By using dynamic average consensus, we consider as signal the local gradient:

$$\mathbf{r}_i^k = \nabla l_i(\mathbf{z}_i^k)$$

Then, the estimate of the average signal (i.e., gradient) is given by:

$$\mathbf{s}_i^{k+1} = \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{s}_j^k + (\nabla l_i(\mathbf{z}_i^{k+1}) - \nabla l_i(\mathbf{z}_i^k)) \quad \mathbf{s}_i^0 = \nabla l_i(\mathbf{z}_i^0)$$

The update step is then performed as:

$$\mathbf{z}_i^{k+1} = \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{z}_j^k - \alpha \mathbf{s}_i^k$$

**Theorem 4.5.5** (Gradient tracking algorithm optimality). If:

- $\mathbf{A}$  is the adjacency matrix of an undirected and connected communication

Dynamic average consensus

Gradient tracking algorithm

Gradient tracking algorithm optimality

graph  $G$  such that it is doubly stochastic and  $a_{ij} > 0$ .

- Each cost function  $l_i$  is  $\mu$ -strongly convex and its gradient  $L$ -Lipschitz continuous.

Then, there exists  $\alpha^* > 0$  such that, for any choice of the step size  $\alpha \in (0, \alpha^*)$ , the sequence of local solutions  $\{\mathbf{z}_i^k\}_{k \in \mathbb{N}}$  of each agent generated by the gradient tracking algorithm asymptotically converges to a consensual optimal solution  $\mathbf{z}^*$ :

$$\lim_{k \rightarrow \infty} \|\mathbf{z}_i^k - \mathbf{z}^*\| = 0$$

Moreover, the convergence rate is linear and stability is exponential:

$$\exists \rho \in (0, 1) : \|\mathbf{z}_i^k - \mathbf{z}^*\| \leq \rho \|\mathbf{z}_i^{k+1} - \mathbf{z}^*\| \wedge \rho \|\mathbf{z}_i^{k+1} - \mathbf{z}^*\| \leq \rho^k \|\mathbf{z}_i^0 - \mathbf{z}^*\|$$

**Remark.** It can be shown that gradient tracking also works with non-convex optimization and, under the correct assumptions, converges to a stationary point.

## 4.A Appendix: Gradient tracking optimality and stability proof

**Theorem 4.A.1** (Gradient tracking algorithm optimality). If:

Gradient tracking  
algorithm optimality

- $\mathbf{A}$  is the adjacency matrix of an undirected and connected communication graph  $G$  such that it is doubly stochastic and  $a_{ij} > 0$ .
- Each cost function  $l_i$  is  $\mu$ -strongly convex and its gradient  $L$ -Lipschitz continuous.

Then, there exists  $\alpha^* > 0$  such that, for any choice of the step size  $\alpha \in (0, \alpha^*)$ , the sequence of local solutions  $\{\mathbf{z}_i^k\}_{k \in \mathbb{N}}$  of each agent generated by the gradient tracking algorithm asymptotically converges to a consensual optimal solution  $\mathbf{z}^*$ :

$$\lim_{k \rightarrow \infty} \|\mathbf{z}_i^k - \mathbf{z}^*\| = 0$$

Moreover, the convergence rate is linear and stability is exponential:

$$\exists \rho \in (0, 1) : \|\mathbf{z}_i^k - \mathbf{z}^*\| \leq \rho \|\mathbf{z}_i^{k+1} - \mathbf{z}^*\| \wedge \rho \|\mathbf{z}_i^{k+1} - \mathbf{z}^*\| \leq \rho^k \|\mathbf{z}_i^0 - \mathbf{z}^*\|$$

*Proof.* Consider the gradient tracking algorithm written in matrix form:

$$\begin{aligned} \mathbf{z}^{k+1} &= \mathbf{A}\mathbf{z}^k - \alpha \mathbf{s}^k \\ \mathbf{s}^{k+1} &= \mathbf{A}\mathbf{s}^k + (\nabla l(\mathbf{z}^{k+1}) - \nabla l(\mathbf{z}^k)) \end{aligned}$$

where  $\nabla l(\mathbf{z}^k) = [l_1(\mathbf{z}_1^k) \dots l_N(\mathbf{z}_N^k)]$ .

**Equilibrium** We want to find the equilibrium points  $(\mathbf{z}_{\text{eq}}, \mathbf{s}_{\text{eq}})$  that satisfies:

$$\begin{aligned} \mathbf{s}_{\text{eq}} &= \mathbf{A}\mathbf{s}_{\text{eq}} + \nabla l(\mathbf{z}_{\text{eq}}) - \nabla l(\mathbf{z}_{\text{eq}}) \iff (\mathbf{I} - \mathbf{A})\mathbf{s}_{\text{eq}} = 0 \\ \mathbf{z}_{\text{eq}} &= \mathbf{A}\mathbf{z}_{\text{eq}} - \alpha \mathbf{s}_{\text{eq}} \iff (\mathbf{I} - \mathbf{A})\mathbf{z}_{\text{eq}} = -\alpha \mathbf{s}_{\text{eq}} \end{aligned}$$

It must be that:

- $\mathbf{s}_{\text{eq}} \in \ker(\mathbf{I} - \mathbf{A}) = \{\mathbf{1}\beta_1 \mid \beta_1 \in \mathbb{R}\}$  (as  $\mathbf{A}$  is doubly stochastic).
- $(\mathbf{I} - \mathbf{A})\mathbf{z}_{\text{eq}} = -\alpha \mathbf{1}\beta_1$ . As  $\mathbf{1}(-\alpha\beta_1) \in \ker(\mathbf{I} - \mathbf{A})$ , it must be that  $\beta_1 = 0$  (as the image cannot be mapped into the kernel).

Therefore, we end up with:

$$\begin{aligned}\mathbf{s}_{\text{eq}} &= \mathbf{1}\beta_1 = 0 \\ \mathbf{z}_{\text{eq}} &= \mathbf{A}\mathbf{z}_{\text{eq}} - \alpha 0 = \mathbf{1}\beta_2 \quad \text{i.e., eigenvector of } \mathbf{A}\end{aligned}$$

In addition, by pre-multiplying the equation of  $\mathbf{s}$  by  $\mathbf{1}^T$ , we obtain:

$$\begin{aligned}\mathbf{1}^T \mathbf{s}^{k+1} &= \mathbf{1}^T \mathbf{A}\mathbf{s}^k + \mathbf{1}^T \nabla l(\mathbf{z}^{k+1}) - \mathbf{1}^T \nabla l(\mathbf{z}^k) \\ &= \mathbf{1}^T \mathbf{s}^k + \mathbf{1}^T \nabla l(\mathbf{z}^{k+1}) - \mathbf{1}^T \nabla l(\mathbf{z}^k)\end{aligned}$$

Which shows the following invariance condition:

$$\begin{aligned}\mathbf{1}^T \mathbf{s}^{k+1} - \mathbf{1}^T \nabla l(\mathbf{z}^{k+1}) &= \mathbf{1}^T \mathbf{s}^k - \mathbf{1}^T \nabla l(\mathbf{z}^k) \\ &= \mathbf{1}^T \mathbf{s}_{\text{eq}} - \mathbf{1}^T \nabla l(\mathbf{z}_{\text{eq}}) \\ &= \mathbf{1}^T \mathbf{s}^0 - \mathbf{1}^T \nabla l(\mathbf{z}^0)\end{aligned}$$

Thus, we have that:

$$\begin{aligned}\mathbf{1}^T \mathbf{s}_{\text{eq}} - \mathbf{1}^T \nabla l(\mathbf{z}_{\text{eq}}) &= \mathbf{1}^T \mathbf{s}^0 - \mathbf{1}^T \nabla l(\mathbf{z}^0) \\ \iff 0 - \mathbf{1}^T \nabla l(\mathbf{1}\beta_2) &= 0\end{aligned}$$

Then, it must be that  $\mathbf{z}_{\text{eq}} = \mathbf{1}\beta_2$  is an optimum with  $\beta_2 = z^*$ .

**Stability** Apply the following change in coordinates:

$$\begin{bmatrix} \mathbf{z}^k \\ \xi^k \end{bmatrix} \mapsto \begin{bmatrix} \tilde{\mathbf{z}}^k \\ \tilde{\xi}^k \end{bmatrix} = \begin{bmatrix} \mathbf{z}^k - \mathbf{1}z^* \\ \xi^k - \alpha \nabla l(\mathbf{1}z^*) \end{bmatrix}$$

So that the equilibrium of the system is shifted to 0.

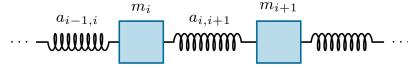
Then, exploit strong convexity to re-formulate the overall system in such a way that the Lyapunov theorem can be applied to prove exponential stability.

□

# 5 Formation control

## 5.1 Intuition with mass-spring systems

**Mass-spring system** System of  $N$  masses where each mass  $i$  has a position  $x_i \in \mathbb{R}$  and is connected through a sprint to mass  $i - 1$  and  $i + 1$ . Each spring has an elastic constant  $a_{j,i} = a_{i,j} > 0$ .



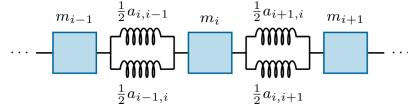
The elastic force  $F_{e,i}(\mathbf{x})$  at mass  $i$  is given by:

$$F_{e,i}(\mathbf{x}) = -a_{i,i-1}(x_i - x_{i-1}) - a_{i,i+1}(x_i - x_{i+1})$$

Equivalently, it is possible to express the elastic force as the negative gradient of the elastic potential energy:

$$F_{e,i}(\mathbf{x}) = -\frac{\partial}{\partial x_i} \left( \frac{1}{2} a_{i,i-1} \|x_i - x_{i-1}\|^2 + \frac{1}{2} a_{i,i+1} \|x_i - x_{i+1}\|^2 \right)$$

**Mass-spring system with two springs** Assume that the springs of a mass-spring system can be split with halved elastic constants.



Mass-spring system with two springs

Accordingly, the elastic force can be defined as:

$$\begin{aligned} F_{e,i}(\mathbf{x}) &= -\frac{1}{2} a_{i,i-1}(x_i - x_{i-1}) - \frac{1}{2} a_{i-1,i}(x_i - x_{i-1}) - \frac{1}{2} a_{i,i+1}(x_i - x_{i+1}) - \frac{1}{2} a_{i+1,i}(x_i - x_{i+1}) \\ &= -\frac{\partial}{\partial x_i} \left( \frac{1}{2} \frac{a_{i,i-1}}{2} \|x_i - x_{i-1}\|^2 + \frac{1}{2} \frac{a_{i-1,i}}{2} \|x_i - x_{i-1}\|^2 + \frac{1}{2} \frac{a_{i,i+1}}{2} \|x_i - x_{i+1}\|^2 + \frac{1}{2} \frac{a_{i+1,i}}{2} \|x_i - x_{i+1}\|^2 \right) \\ &= -\frac{\partial}{\partial x_i} \left( \frac{1}{2} \frac{a_{i,i-1}}{2} \|x_i - x_{i-1}\|^2 + \frac{1}{2} \frac{a_{i-1,i}}{2} \|x_{i-1} - x_i\|^2 + \frac{1}{2} \frac{a_{i,i+1}}{2} \|x_i - x_{i+1}\|^2 + \frac{1}{2} \frac{a_{i+1,i}}{2} \|x_{i+1} - x_i\|^2 \right) \end{aligned}$$

The total potential energy (i.e., sum of the function in the derivative over all masses) can be compactly defined as:

$$\begin{aligned} V(\mathbf{x}) &= \sum_{i=1}^N \sum_{j \in \mathcal{N}_i} \frac{1}{2} \frac{a_{i,j}}{2} \|x_i - x_j\|^2 & V_{i,j}(x_i, x_j) &= \frac{1}{2} \frac{a_{i,j}}{2} \|x_i - x_j\|^2 \\ &= \sum_{i=1}^N \sum_{j \in \mathcal{N}_i} V_{i,j}(x_i, x_j) \end{aligned}$$

where  $\mathcal{N}_i = \{i - 1, i + 1\}$ .

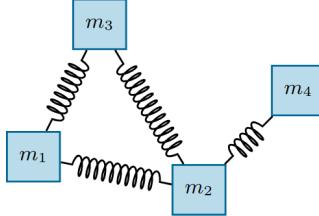
Then, the potential energy at mass  $i$  can be written as:

$$V_i(\mathbf{x}) = \sum_{j \in \mathcal{N}_i} (V_{i,j}(x_i, x_j) + V_{j,i}(x_j, x_i))$$

Finally, the elastic force at mass  $i$  can be reformulated as:

$$\begin{aligned} F_{e,i}(\mathbf{x}) &= -\frac{\partial}{\partial x_i} \left( V_{i,i-1}(x_i, x_{i-1}) + V_{i-1,i}(x_{i-1}, x_i) + V_{i,i+1}(x_i, x_{i+1}) + V_{i+1,i}(x_{i+1}, x_i) \right) \\ &= -\frac{\partial}{\partial x_i} \left( \sum_{j \in \mathcal{N}_i} (V_{i,j}(x_i, x_j) + V_{j,i}(x_j, x_i)) \right) \\ &= -\frac{\partial}{\partial x_i} V_i(\mathbf{x}) \\ &= -\frac{\partial}{\partial x_i} V(\mathbf{x}) \end{aligned}$$

**Remark.** The system can be generalized to a graph of interconnected masses.



By adding a constant damping coefficient (i.e., dispersion of velocity)  $c = 1$ , the overall system dynamics can be defined as:

$$\begin{aligned} \dot{x}_i &= v_i \\ m_i \dot{v}_i &= -v_i - c \frac{\partial}{\partial x_i} V(\mathbf{x}) = -v_i - \frac{\partial}{\partial x_i} V(\mathbf{x}) \end{aligned}$$

where  $m_i$  is the mass of the  $i$ -th mass.

By assuming small masses  $m_i$ , the following approximation can be made:

$$\begin{aligned} m_i \dot{v}_i &= -v_i - \frac{\partial}{\partial x_i} V(\mathbf{x}) \Rightarrow v_i \approx -\frac{\partial}{\partial x_i} V(\mathbf{x}) \\ \dot{x}_i &= -\frac{\partial}{\partial x_i} V(\mathbf{x}) = F_{e,i}(\mathbf{x}) \end{aligned}$$

By more explicitly expanding the dynamics of the  $i$ -th mass, we have that:

$$\begin{aligned} \dot{x}_i &= -\sum_{j \in \mathcal{N}_i} \frac{\partial}{\partial x_i} (V_{i,j}(x_i, x_j) + V_{j,i}(x_j, x_i)) \\ &= -\sum_{j \in \mathcal{N}_i} \frac{\partial}{\partial x_i} \left( \frac{1}{2} \frac{a_{i,j}}{2} \|x_i - x_j\|^2 + \frac{1}{2} \frac{a_{j,i}}{2} \|x_j - x_i\|^2 \right) \\ &= -\sum_{j \in \mathcal{N}_i} \left( \frac{1}{2} a_{i,j} (x_i - x_j) - \frac{1}{2} a_{j,i} (x_j - x_i) \right) && a_{i,j} = a_{j,i} \text{ (G undirected)} \\ &= -\sum_{j \in \mathcal{N}_i} a_{i,j} (x_i - x_j) && \text{i.e., Laplacian dynamics} \end{aligned}$$

Therefore, the overall system follows a Laplacian dynamics and can be equivalently formulated as the gradient flow of  $V$ :

$$\begin{aligned}\dot{\mathbf{x}} &= -\mathbf{L}\mathbf{x} = -\nabla V(\mathbf{x}) \\ \begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_N \end{bmatrix} &= -\begin{bmatrix} \frac{\partial}{\partial x_1} V(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_N} V(\mathbf{x}) \end{bmatrix}\end{aligned}$$

And consensus is reached at a stationary point of  $V(\mathbf{x})$ .

## 5.2 Formation control based on potential functions

**Remark.** The gradient flow based on the global potential function/energy can be computed in a distributed way (i.e., it depends only on neighboring states):

$$\dot{x}_i(t) = - \sum_{j \in \mathcal{N}_i} \frac{\partial}{\partial x_i} (V_{i,j}(x_i, x_j) + V_{j,i}(x_j, x_i))$$

**Formation control** Consider  $N$  agents with states  $\mathbf{x}_i(t) \in \mathbb{R}^d$  and communicating according to a fixed undirected graph  $G$ . The goal is to position each agent respecting the desired distances  $d_{ij} = d_{ji}$  between them:

$$\forall (i, j) \in E : \|\mathbf{x}_i^{\text{form}} - \mathbf{x}_j^{\text{form}}\| = d_{ij}$$

Formation control

To solve the problem, the potential function can be defined as:

$$\begin{aligned}V^{\text{form}}(\mathbf{x}) &= \sum_{i=1}^N \sum_{j \in \mathcal{N}_i} V_{ij}^{\text{form}}(\mathbf{x}_i, \mathbf{x}_j) \\ V_{ij}^{\text{form}}(\mathbf{x}_i, \mathbf{x}_j) &= \frac{1}{8} (\|\mathbf{x}_i - \mathbf{x}_j\|^2 - d_{ij}^2)^2\end{aligned}$$

where  $\frac{1}{8}$  is used to cancel out the coefficients when deriving.

The gradient flow dynamics is then:

$$\begin{aligned}\dot{\mathbf{x}}_i &= - \sum_{j \in \mathcal{N}_i} \frac{\partial}{\partial \mathbf{x}_i} (V_{ij}^{\text{form}}(\mathbf{x}_i, \mathbf{x}_j) + V_{ji}^{\text{form}}(\mathbf{x}_j, \mathbf{x}_i)) \\ &= - \sum_{j \in \mathcal{N}_i} (\|\mathbf{x}_i - \mathbf{x}_j\|^2 - d_{ij}^2) (\mathbf{x}_i - \mathbf{x}_j)\end{aligned}$$

**Remark.** Apart from the desired formation, another equilibrium of this dynamics is  $x_1 = x_2 = \dots = x_N$  (i.e., a collision).

**Collision avoidance potential function/Barrier function** Function  $V_{ij}^{\text{ca}}(\mathbf{x}_i, \mathbf{x}_j)$  such that:

$$\lim_{\|\mathbf{x}_i - \mathbf{x}_j\| \rightarrow 0} V_{ij}^{\text{ca}}(\mathbf{x}_i, \mathbf{x}_j) = +\infty$$

Collision avoidance potential function/Barrier function

**Remark.** A possible barrier function is:

$$V_{ij}^{\text{ca}}(\mathbf{x}_i, \mathbf{x}_j) = -\log(\|\mathbf{x}_i - \mathbf{x}_j\|^2 - d^2)$$

| where  $d$  is the safety distance.

**Formation control with obstacle avoidance** Formation control where agents avoid collisions and obstacles. The dynamics is:

$$\dot{\mathbf{x}}_i = -\frac{\partial}{\partial \mathbf{x}_i} \left( V^{\text{form}}(\mathbf{x}) + V^{\text{ca}}(\mathbf{x}) + V^{\text{obs}}(\mathbf{x}) \right)$$

Formation control  
with obstacle  
avoidance

where:

- $V^{\text{ca}}(\mathbf{x}) = \sum_{i=1}^N \sum_{j \in \mathcal{N}_i} V_{ij}^{\text{ca}}(\mathbf{x}_i, \mathbf{x}_j)$  and  $V_{ij}^{\text{ca}}(\mathbf{x}_i, \mathbf{x}_j)$  is the barrier function to avoid collisions between agents.
- $V^{\text{obs}}(\mathbf{x}) = \sum_{i=1}^N V_i^{\text{obs}}(\mathbf{x}_i)$  and  $V_i^{\text{obs}}(\mathbf{x}_i)$  is the barrier function to avoid collisions between an agent and an obstacle.

# 6 Cooperative robotics

**Cooperative robotics** Problem where  $N$  agents want to optimize their positions  $\mathbf{z}_i \in \mathbb{R}^2$  to perform multi-robot surveillance in an environment with:

- A static target to protect  $\mathbf{r}_0 \in \mathbb{R}^2$ .
- Static intruders/opponents  $\mathbf{r}_i \in \mathbb{R}^2$ , each assigned to the respective agent  $i$ .

The average position of the agents define the barycenter:

$$\sigma(\mathbf{z}) = \frac{1}{N} \sum_{i=1}^N \mathbf{z}_i$$

The local cost function of agent  $i$  is:

$$l_i(\mathbf{z}_i, \sigma(\mathbf{z})) = \gamma_i \underbrace{\|\mathbf{z}_i - \mathbf{r}_i\|^2}_{\text{close to opponent}} + \underbrace{\|\sigma(\mathbf{z}) - \mathbf{r}_0\|^2}_{\text{barycenter close to target}}$$

Note that the opponent component only depends on local variables while the target component needs global information.

| **Remark.** The barycenter  $\sigma(\mathbf{z}) : \mathbb{R}^{2N} \rightarrow \mathbb{R}^2$  can be seen as an aggregation function.

| **Remark.** A scenario that this formulation fails to handle is when the agents are placed symmetrically and moves symmetrically as the barycenter remains the same even if the agents move farther away.

## 6.1 Aggregative optimization

**Aggregative optimization** Problem defined as:

Aggregative optimization

$$\min_{\mathbf{z}_1, \dots, \mathbf{z}_N} \sum_{i=1}^N l_i(\mathbf{z}_i, \sigma(\mathbf{z}))$$

where:

- $\mathbf{z} = (\mathbf{z}_1, \dots, \mathbf{z}_N)$  with  $\mathbf{z}_i \in \mathbb{R}^{n_i}$ ,
- $l_i : \mathbb{R}^{n_i} \times \mathbb{R}^d \rightarrow \mathbb{R}$  is the loss function of the agent  $i$ ,
- $\sigma(\mathbf{z})$  is an aggregation function generically defined as  $\sigma(\mathbf{z}) = \frac{1}{N} \sum_{i=1}^N \phi_i(\mathbf{z}_i)$ , for some  $\phi_i : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^d$

**Distributed aggregative optimization** Distributed case of aggregative optimization where each agent has only access to the loss  $l_i$ , the operator of the aggregation function  $\phi_i$ , and the position  $\mathbf{z}_i$  of itself and its neighbors.

Distributed aggregative optimization

| **Remark.** The goal of the task is not to reach consensus among agents.

### 6.1.1 Centralized gradient method

**Centralized gradient method (scalar)** Consider  $N$  agents with  $z_i \in \mathbb{R}$  and  $\sigma : \mathbb{R}^N \rightarrow \mathbb{R}$ .

The update step, assuming global access to the parameters, can be performed as:

$$z_i^{k+1} = z_i^k - \alpha \frac{\partial}{\partial z_i} \left( \sum_{j=1}^N l_j(z_j, \sigma(z_1, \dots, z_N)) \right) \Big|_{z_j=z_j^k}$$

By expanding the derivative, we have that:

$$\begin{aligned} & \frac{\partial}{\partial z_i} \left( \sum_{j=1}^N l_j(z_j, \sigma(z_1, \dots, z_N)) \right) \Big|_{z_j=z_j^k} \\ &= \frac{\partial}{\partial z_i} l_i(z_i, \sigma) \Big|_{\substack{z_i=z_i^k, \\ \sigma=\sigma(\mathbf{z}^k)}} + \sum_{j=1}^N \left( \left( \frac{\partial}{\partial \sigma} l_j(z_j, \sigma) \right) \Big|_{\substack{z_j=z_j^k, \\ \sigma=\sigma(\mathbf{z}^k)}} \cdot \frac{\partial}{\partial z_i} \sigma(z_1, \dots, z_N) \Big|_{z_j=z_j^k} \right) \end{aligned}$$

**Centralized gradient method (vector)** Generalized to the vector case, the update step becomes:

$$\mathbf{z}_i^{k+1} = \mathbf{z}_i^k - \alpha \left[ \nabla \left( \sum_{j=1}^N l_j(\mathbf{z}_j, \sigma(\mathbf{z}_1, \dots, \mathbf{z}_N)) \right) \Big|_{\mathbf{z}_j=\mathbf{z}_j^k} \right]_{(i)}$$

And the gradient can be expanded as:

$$\begin{aligned} & \left[ \nabla \left( \sum_{j=1}^N l_j(\mathbf{z}_j, \sigma(\mathbf{z}_1, \dots, \mathbf{z}_N)) \right) \Big|_{\mathbf{z}_j=\mathbf{z}_j^k} \right]_{(i)} \\ &= \nabla_{[\mathbf{z}_i]} l_i(\mathbf{z}_i, \sigma) \Big|_{\substack{\mathbf{z}_i=\mathbf{z}_i^k, \\ \sigma=\sigma(\mathbf{z}^k)}} + \sum_{j=1}^N \nabla_{[\sigma]} l_j(\mathbf{z}_j, \sigma) \Big|_{\substack{\mathbf{z}_j=\mathbf{z}_j^k, \\ \sigma=\sigma(\mathbf{z}^k)}} \cdot \frac{1}{N} \nabla \phi_i(\mathbf{z}_i) \Big|_{\mathbf{z}_i=\mathbf{z}_i^k} \end{aligned}$$

where  $\nabla_{[\mathbf{z}_i]} l_i(\mathbf{z}_i, \sigma)$  is the gradient w.r.t. the first argument and  $\nabla_{[\sigma]} l_i(\mathbf{z}_i, \sigma)$  is w.r.t. the second one.

### 6.1.2 Aggregative tracking distributed optimization algorithm

**Aggregative tracking distributed optimization algorithm** Algorithm where each agent  $i$  has:

- An estimate  $\mathbf{z}_i^k$  of its optimal position  $\mathbf{z}_i^*$ ,
- An estimate  $\mathbf{s}_i^k$  of the aggregation function  $\sigma(\mathbf{z}^k) = \frac{1}{N} \sum_{j=1}^N \phi_j(\mathbf{z}_j^k)$ ,
- An estimate  $\mathbf{v}_i^k$  of the gradient with respect to the second argument of the loss  $\frac{1}{N} \sum_{j=1}^N \nabla_{[\sigma(\mathbf{z}^k)]} l_j(\mathbf{z}_j^k, \sigma(\mathbf{z}^k))$ .

Centralized gradient method (scalar)

Centralized gradient method (vector)

Aggregative tracking distributed optimization algorithm

The step is based on the centralized gradient method using the local estimates:

$$\begin{aligned}\mathbf{z}_i^{k+1} &= \mathbf{z}_i^k - \alpha \left( \nabla_{[\mathbf{z}_i]} l_i(\mathbf{z}_i^k, \mathbf{s}_i^k) + \mathbf{v}_i^k \nabla \phi_i(\mathbf{z}_i^k) \right) & \mathbf{z}_i^0 \in \mathbb{R}^{n_i} \\ \mathbf{s}_i^{k+1} &= \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{s}_j^k + \left( \phi_i(\mathbf{z}_i^{k+1}) - \phi_i(\mathbf{z}_i^k) \right) & \mathbf{s}_i^0 = \phi_i(\mathbf{z}_i^0) \\ \mathbf{v}_i^{k+1} &= \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{v}_j^k + \left( \nabla_{[\mathbf{s}_i^{k+1}]} l_i(\mathbf{z}_i^{k+1}, \mathbf{s}_i^{k+1}) - \nabla_{[\mathbf{s}_i^k]} l_i(\mathbf{z}_i^k, \mathbf{s}_i^k) \right) & \mathbf{v}_i^0 = \nabla_{[\mathbf{s}_i^0]} l_i(\mathbf{z}_i^0, \mathbf{s}_i^0)\end{aligned}$$

where the estimates  $\mathbf{s}_i^k$  and  $\mathbf{v}_i^k$  are obtained through dynamic average consensus (Section 4.5.2).

**Remark.**  $\mathbf{v}_i^k$  is a double approximation as it uses  $\mathbf{s}_i^{k+1}$  and  $\mathbf{s}_i^k$  instead of the real  $\sigma$ .

**Theorem 6.1.1** (Aggregative tracking distributed optimization algorithm convergence). If:

- The communication digraph  $G$  is strongly connected and aperiodic, and  $\mathbf{A}$  is doubly stochastic,
- $\sum_{i=1}^N l_i(\cdot, \sigma(\cdot))$  strongly convex with  $\phi_i(\cdot)$  differentiable and Lipschitz continuous.
- $\nabla_{[\mathbf{z}]} l_i(\cdot, \cdot)$ ,  $\nabla_{[\sigma]} l_i(\cdot, \cdot)$ , and  $\nabla \phi_i(\cdot) \nabla_{[\sigma]} l_i(\cdot, \cdot)$  are Lipschitz continuous.

Then, there exists an  $\alpha^*$  such that, for any step size  $\alpha \in (0, \alpha^*)$ , the sequences of local estimates  $\{\mathbf{z}_i^k, \dots, \mathbf{z}_N^k\}_{k \in \mathbb{N}}$  generated using the aggregative tracking distributed optimization algorithm converge to the optimal solution at a linear rate:

$$\lim_{k \rightarrow \infty} \|\mathbf{z}_i^k - \mathbf{z}_i^*\| = 0$$

### 6.1.3 Online aggregative optimization

**Online aggregative optimization** Time-varying case of aggregative optimization where intruders also move. The problem can be defined as:

$$\min_{\mathbf{z}=(\mathbf{z}_1, \dots, \mathbf{z}_N)} \sum_{i=1}^N l_i^k(\mathbf{z}_i, \sigma^k(\mathbf{z})) \quad \text{subject to } \mathbf{z}_i \in Z_i^k$$

where  $Z_i^k$  is a closed convex set.

**Remark.** As intruders are dynamic, the optimum of each agent  $\mathbf{z}_i^{k,*}$  changes over time.

**Projected aggregative tracking** Algorithm for online aggregative optimization defined as:

$$\begin{aligned}\tilde{\mathbf{z}}_i^k &= P_{Z_i^k} \left[ \mathbf{z}_i^k - \alpha \left( \nabla_{[\mathbf{z}_i^k]} l_i^k(\mathbf{z}_i^k, \mathbf{s}_i^k) + \mathbf{v}_i^k \nabla \phi_i^k(\mathbf{z}_i^k) \right) \right] \\ \mathbf{z}_i^{k+1} &= \mathbf{z}_i^k + \delta (\tilde{\mathbf{z}}_i^k - \mathbf{z}_i^k) & \mathbf{z}_i^0 \in \mathbb{R}^{n_i} \\ \mathbf{s}_i^{k+1} &= \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{s}_j^k + \left( \phi_i^{k+1}(\mathbf{z}_i^{k+1}) - \phi_i^k(\mathbf{z}_i^k) \right) & \mathbf{s}_i^0 = \phi_i(\mathbf{z}_i^0) \\ \mathbf{v}_i^{k+1} &= \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{v}_j^k + \left( \nabla_{[\mathbf{s}_i^{k+1}]} l_i^{k+1}(\mathbf{z}_i^{k+1}, \mathbf{s}_i^{k+1}) - \nabla_{[\mathbf{s}_i^k]} l_i^k(\mathbf{z}_i^k, \mathbf{s}_i^k) \right) & \mathbf{v}_i^0 = \nabla_{[\mathbf{s}_i^0]} l_i(\mathbf{z}_i^0, \mathbf{s}_i^0)\end{aligned}$$

where  $P_{Z_i^k}$  is the Euclidean projection and  $\delta \in (0, 1)$  is a hyperparameter.

Online aggregative optimization

Projected aggregative tracking

# 7 Safety controllers

**Control-affine non-linear dynamical system** System whose dynamics follows:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) + g(\mathbf{x}(t))\mathbf{u}(t) \quad \mathbf{x}(0) = \mathbf{x}_0$$

Control-affine  
non-linear dynamical  
system

with  $\mathbf{x}(t) \in \mathbb{R}^n$ ,  $\mathbf{u}(t) \in U \subseteq \mathbb{R}^m$ ,  $f(\mathbf{x}(t)) \in \mathbb{R}^n$ , and  $g(\mathbf{x}(t)) \in \mathbb{R}^{n \times m}$ .

$f(\mathbf{x}(t))$  can be seen as the drift of the system and  $\mathbf{u}(t)$  a coefficient that controls how much  $g(\mathbf{x}(t))$  is injected into  $f(\mathbf{x}(t))$ .

The overall system can be interpreted as composed of:

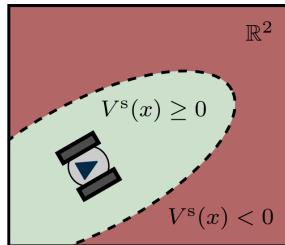
- A high-level controller that produces the direction  $\mathbf{u}^{\text{ref}}(\mathbf{x})$  towards the target position.
- A safety layer that modifies  $\mathbf{u}^{\text{ref}}(\mathbf{x})$  into  $\mathbf{u}(t) = \kappa(\mathbf{x})$  to account for obstacles.

**Safety control** Given a (sufficiently regular) function  $V^s : X \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ , it is possible to define a safe state set as:

Safety control

$$X^s = \{\mathbf{x} \in X \subseteq \mathbb{R}^n \mid V^s(\mathbf{x}) \geq 0\}$$

The goal is to design a feedback control law  $\kappa^s : X \rightarrow \mathbb{R}^m$  for a control-affine non-linear dynamical system such that the set  $X^s$  is forward invariant (i.e., any trajectory starting in  $X^s$  remains in  $X^s$ ).



**Remark.** The time derivative of  $V^s(\mathbf{x}(t))$  along the system trajectories is given by:

$$\begin{aligned} \frac{d}{dt}V^s(\mathbf{x}(t)) &= \nabla V^s(\mathbf{x}(t))^T \frac{d}{dt}\mathbf{x}(t) \\ &= \nabla V^s(\mathbf{x}(t))^T (f(\mathbf{x}(t)) + g(\mathbf{x}(t))\mathbf{u}(t)) \\ &= \nabla V^s(\mathbf{x}(t))^T f(\mathbf{x}(t)) + \sum_{h=1}^m (\nabla V^s(\mathbf{x}(t))^T g_h(\mathbf{x}(t))\mathbf{u}_h(t)) \\ &= L_f V^s(\mathbf{x}(t)) + L_g V^s(\mathbf{x}(t))\mathbf{u}(t) \end{aligned}$$

where  $L_h V^s(\mathbf{x}(t)) = \nabla V^s(\mathbf{x}(t))^T h(\mathbf{x}(t))$  is the lie derivative.

**Control barrier function (CBF)** A function  $V^s$  is a control barrier function if there exists a continuous strictly increasing function  $\gamma : \mathbb{R} \rightarrow \mathbb{R}$  with  $\gamma(0) = 0$  such that the

Control barrier  
function (CBF)

following inequality (control barrier certificate) holds:

$$\sup_{\mathbf{u} \in U} \{L_f V^s(\mathbf{x}) + L_g V^s(\mathbf{x})\mathbf{u} + \gamma(V^s(\mathbf{x}))\} \geq 0 \quad \forall \mathbf{x} \in X$$

$\gamma$  can be interpreted as a degree of movement freedom since, as long as it holds that  $V^s(\mathbf{x}(t)) > 0$ , it is allowed that  $\frac{d}{dt}V^s(\mathbf{x}(t)) < 0$  (i.e., the agent can move closer to the border between safe and unsafe region).

**Remark.** In principle, the negative part of  $\gamma$  is not necessary (the agent should start in a safe area). However, as it is strictly increasing, it allows to move out the unsafe region if the agent ever ends up there.

**Example.** A simple choice for  $\gamma$  is a linear function  $\gamma(r) = \gamma r$  with  $\gamma > 0$ .

**Set of admissible safe controllers** The set of inputs that satisfy the control barrier certificate for a given state  $\mathbf{x}$  is:

$$U^s(\mathbf{x}) = \{\mathbf{u} \in U \mid L_f V^s(\mathbf{x}) + L_g V^s(\mathbf{x})\mathbf{u} + \gamma(V^s(\mathbf{x})) \geq 0\}$$

Set of admissible safe controllers

## 7.1 Safety filter via control barrier certificate

**Safety filter via control barrier certificate** Given a possibly unsafe reference input (from the high-level controller)  $\mathbf{u}^{\text{ref}}(\mathbf{x}) \in \mathbb{R}^m$ , the safety controller (i.e., rectifying controller) based on the control barrier certificate is designed to be minimally invasive (i.e., alter the reference as little as possible).

Safety filter via control barrier certificate

The policy  $\mathbf{u} = \kappa^s(\mathbf{x})$  can be defined as:

$$\begin{aligned} \kappa^s(\mathbf{x}) &= \arg \min_{\mathbf{u} \in U} \|\mathbf{u} - \mathbf{u}^{\text{ref}}(\mathbf{x})\|^2 \\ \text{subject to } &-L_f V^s(\mathbf{x}) - L_g V^s(\mathbf{x})\mathbf{u} - \gamma(V^s(\mathbf{x})) \leq 0 \end{aligned}$$

**Remark.** In the general case, this problem should be solved at each  $t \geq 0$ .

**Single integrator model** Control-affine non-linear dynamical system where  $f(\mathbf{x}(t)) = 0$  and  $g(\mathbf{x}(t)) = \mathbf{I}$ . The dynamics is:

$$\begin{aligned} \dot{\mathbf{x}} &= 0 + \mathbf{I}\mathbf{u} \\ &= \mathbf{u} \end{aligned}$$

Single integrator model

with  $\mathbf{x} \in \mathbb{R}^d$  and  $\mathbf{u} \in \mathbb{R}^d$ .

**Remark.** In the case of single integrators, we have that:

- $L_f V^s(\mathbf{x}) = \nabla V^s(\mathbf{x})^T 0 = 0$ ,
- $L_g V^s(\mathbf{x}) = \nabla V^s(\mathbf{x})^T \mathbf{I} = \nabla V^s(\mathbf{x})^T$ .

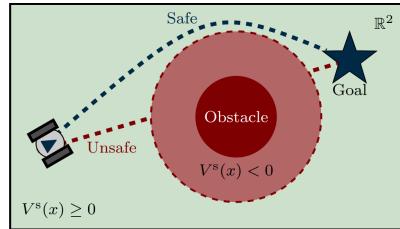
Therefore:

$$\begin{aligned} \frac{d}{dt} V^s(\mathbf{x}(t)) &= L_f V^s(\mathbf{x}(t)) + L_g V^s(\mathbf{x}(t))\mathbf{u}(t) \\ &= \nabla V^s(\mathbf{x}(t))^T \mathbf{u}(t) \end{aligned}$$

### 7.1.1 Single-robot obstacle avoidance with single integrator models

**Single-robot obstacle avoidance** Task where the goal is to keep an agent to a safety distance  $\Delta > 0$  from an obstacle.

Single-robot obstacle avoidance



A control barrier function to solve the task (i.e., rectify the trajectory of the high level controller) can be:

$$V^s(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_{\text{obs}}\|^2 - \Delta^2 \quad \nabla V^s(\mathbf{x}) = 2(\mathbf{x} - \mathbf{x}_{\text{obs}})$$

The CBF-based safety policy  $\kappa^s(\mathbf{x})$  can be obtained by solving:

$$\begin{aligned} & \arg \min_{\mathbf{u} \in U} \|\mathbf{u} - \mathbf{u}^{\text{ref}}(\mathbf{x})\|^2 \\ \text{subject to } & -2(\mathbf{x} - \mathbf{x}_{\text{obs}})^T \mathbf{u} - \gamma(\|\mathbf{x} - \mathbf{x}_{\text{obs}}\|^2 - \Delta^2) \leq 0 \end{aligned}$$

As there are two constants in the constraint  $a = -2(\mathbf{x} - \mathbf{x}_{\text{obs}})^T$  and  $b = \gamma(\|\mathbf{x} - \mathbf{x}_{\text{obs}}\|^2 - \Delta^2)$ , the problem can be reformulated as:

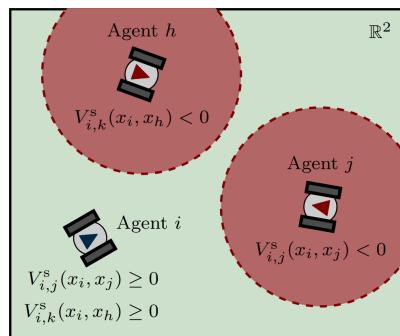
$$\arg \min_{\mathbf{u} \in U} \|\mathbf{u} - \mathbf{u}^{\text{ref}}(\mathbf{x})\|^2 \quad \text{subject to } a^T \mathbf{u} + b \leq 0$$

**Remark.** If  $U$  is a polytope (or unconstrained:  $U = \mathbb{R}^d$ ), the problem becomes a quadratic program.

### 7.1.2 Multi-robot collision avoidance with single integrator models

**Multi-robot collision avoidance** Task with  $N$  single integrator agents that want to keep a safety distance  $\Delta > 0$  among them.

Multi-robot collision avoidance



The local control barrier function to solve the task can be defined as:

$$V^s_{i,j}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^2 - \Delta^2 \quad \begin{aligned} \nabla_{[\mathbf{x}_i]} V^s_{i,j}(\mathbf{x}_i, \mathbf{x}_j) &= 2(\mathbf{x}_i - \mathbf{x}_j) \\ \nabla_{[\mathbf{x}_j]} V^s_{i,j}(\mathbf{x}_i, \mathbf{x}_j) &= 2(\mathbf{x}_j - \mathbf{x}_i) \end{aligned}$$

The safe region  $X_i$  for agent  $i$  can be defined as:

$$X_i = \{\mathbf{x} \in \mathbb{R}^d \mid \forall j \in \mathcal{N}_i : V_{i,j}^s(\mathbf{x}) \geq 0\}$$

The set of admissible controllers is:

$$\begin{aligned} U^s(\mathbf{x}) &= \left\{ \mathbf{u} \in \mathbb{R}^{dN} \mid -\nabla_{[\mathbf{x}_i]} V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j)^T \mathbf{u}_i - \nabla_{[\mathbf{x}_i]} V_{ji}^s(\mathbf{x}_j, \mathbf{x}_i)^T \mathbf{u}_j - \gamma(V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j)) \leq 0 \right. \\ &\quad \left. \forall j \in \mathcal{N}_i, \forall i \in \{1, \dots, N\} \right\} \\ &= \left\{ \mathbf{u} \in \mathbb{R}^{dN} \mid -2(\mathbf{x}_i, \mathbf{x}_j)^T \mathbf{u}_i - 2(\mathbf{x}_j - \mathbf{x}_i)^T \mathbf{u}_j - \gamma(V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j)) \leq 0 \right. \\ &\quad \left. \forall j \in \mathcal{N}_i, \forall i \in \{1, \dots, N\} \right\} \end{aligned}$$

**Centralized safety controller** The CBF-based policy can be obtained by solving:

Centralized safety controller

$$\begin{aligned} &\arg \min_{\mathbf{u} \in \mathbb{R}^N} \sum_{i=1}^N \|\mathbf{u}_i - \mathbf{u}_i^{\text{ref}}\|^2 \\ \text{subject to } &-2(\mathbf{x}_i, \mathbf{x}_j)^T \mathbf{u}_i - 2(\mathbf{x}_j - \mathbf{x}_i)^T \mathbf{u}_j - \gamma(V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j)) \leq 0 \\ &\|\mathbf{u}_i\| \leq \mathbf{u}_i^{\max} \\ &\forall j \in \mathcal{N}_i, \forall i \in \{1, \dots, N\} \end{aligned}$$

where  $\mathbf{u}_i^{\text{ref}}$  is the reference input of the high level controller and  $\mathbf{u}_i^{\max}$  is the bound.

| **Remark.** The policy should be computed continuously for each  $\mathbf{x}_i(t)$ .

**Decentralized safety controller** The CBF-based policy can be obtained by solving a more constrained problem compared to the centralized formulation:

Decentralized safety controller

$$\begin{aligned} &\arg \min_{\mathbf{u}_i \in \mathbb{R}^d} \|\mathbf{u}_i - \mathbf{u}_i^{\text{ref}}\|^2 \\ \text{subject to } &-\nabla_{[\mathbf{x}_i]} V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j)^T \mathbf{u}_i - \frac{1}{2}\gamma(V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j)) \leq 0 \\ &\|\mathbf{u}_i\| \leq \mathbf{u}_i^{\max} \\ &\forall j \in \mathcal{N}_i \end{aligned}$$

| **Remark.** If  $\forall i \in \{1, \dots, N\} : \nabla_{[\mathbf{x}_i]} V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j)^T \mathbf{u}_i \geq \frac{1}{2}\gamma(V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j))$ , then it holds that:

$$\begin{aligned} \nabla_{[\mathbf{x}_i]} V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j)^T \mathbf{u}_i + \nabla_{[\mathbf{x}_i]} V_{ji}^s(\mathbf{x}_j, \mathbf{x}_i)^T \mathbf{u}_j &\geq -\frac{1}{2}\gamma(V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j)) - \frac{1}{2}\gamma(V_{ji}^s(\mathbf{x}_j, \mathbf{x}_i)) \\ &\geq -\gamma(V_{ij}^s(\mathbf{x}_i, \mathbf{x}_j)) \end{aligned}$$

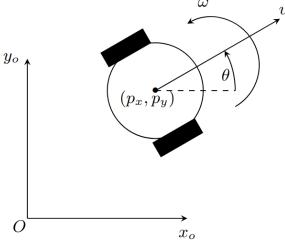
### 7.1.3 Multi-robot collision avoidance with unicycle control

**Unicycle model with non-holonomic constraints** Model that captures the constraints given by wheels. Its dynamics is:

$$\begin{aligned} \dot{\mathbf{p}}_x &= v \cos(\theta) \\ \dot{\mathbf{p}}_y &= v \sin(\theta) \\ \dot{\theta} &= \omega \end{aligned}$$

where:

- $(\mathbf{p}_x, \mathbf{p}_y)$  is the position of the center of mass,
- $\dot{\theta}$  is the orientation,
- $v$  is the linear velocity,
- $\omega$  is the angular velocity.



| **Remark.** It is assumed that the robot does not drift sideways ( $v_{\perp} = 0$ ).

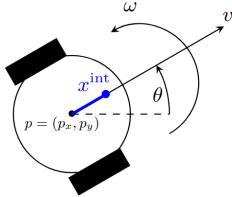
**Single integrator to unicycle control mapping** Consider a point  $\mathbf{x}^{\text{int}}$  longitudinal to  $v$  that is not the barycenter:

$$\mathbf{x}^{\text{int}} = \begin{bmatrix} \mathbf{p}_x \\ \mathbf{p}_y \end{bmatrix} + \rho \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}$$

where  $\rho > 0$  is the distance to the barycenter.

By differentiating w.r.t. time, the dynamics is:

$$\dot{\mathbf{x}}^{\text{int}} = \begin{bmatrix} \dot{\mathbf{p}}_x \\ \dot{\mathbf{p}}_y \end{bmatrix} + \rho \dot{\theta} \begin{bmatrix} -\sin(\theta) \\ \cos(\theta) \end{bmatrix}$$



By using the unicycle model dynamics, it becomes:

$$\begin{aligned} \dot{\mathbf{x}}^{\text{int}} &= \begin{bmatrix} \cos(\theta) & -\rho \sin(\theta) \\ \sin(\theta) & \rho \cos(\theta) \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \\ \dot{\theta} &= \omega \end{aligned}$$

By formulating  $v$  and  $\omega$  as a state-feedback control with input  $\mathbf{u}^{\text{int}} \in \mathbb{R}^2$  as:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\frac{1}{\rho} \sin(\theta) & \frac{1}{\rho} \cos(\theta) \end{bmatrix} \mathbf{u}^{\text{int}}$$

The result is a single-integrator  $\dot{\mathbf{x}}^{\text{int}} = g(\mathbf{x})\mathbf{u}^{\text{int}}$ .

A choice of  $\mathbf{u}$  can be:

$$\mathbf{u}^{\text{int}} = k(\mathbf{x}^{\text{int}} - \mathbf{x}^{\text{dest}}) + \dot{\mathbf{x}}^{\text{dest}}$$

Single integrator to unicycle control mapping

# 8 Feedback optimization

**Feedback optimization** Consider a continuous-time non-linear dynamical system:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \quad \mathbf{x}(0) = \mathbf{x}_0$$

Feedback optimization

Assume that there exists a steady-state map  $h : \mathbb{R}^m \rightarrow \mathbb{R}^n$  (i.e., sort of oracle) that given the input  $\bar{\mathbf{u}}$  returns the equilibrium state  $\bar{\mathbf{x}}$  associated to it:

$$f(h(\bar{\mathbf{u}}), \bar{\mathbf{u}}) = 0$$

Moreover, it is assumed that, for any  $\bar{\mathbf{u}} \in \mathbb{R}^m$ ,  $\bar{\mathbf{x}} = h(\bar{\mathbf{u}})$  is a globally exponentially stable equilibrium for the system  $f(\mathbf{x}(t), \bar{\mathbf{u}})$ .

The goal of feedback optimization is to design a dynamic feedback law  $\mathbf{u}(t) = \kappa(\mathbf{x}(t), t)$  such that  $\mathbf{u}(t)$  and  $\mathbf{x}(t)$  converge to the solution of the problem:

$$\begin{aligned} & \min_{\mathbf{z} \in \mathbb{R}^n, \mathbf{u} \in \mathbb{R}^m} l(\mathbf{z}) \\ & \text{subject to } \mathbf{z} = h(\mathbf{u}) \end{aligned}$$

where  $h$  is not explicitly available and the control law can only access the current state  $\mathbf{x}(t)$  ( $\nabla h$  can however be used).

**Reduced feedback optimization problem** By injecting the constraint of the feedback optimization problem into the cost function, the problem becomes unconstrained:

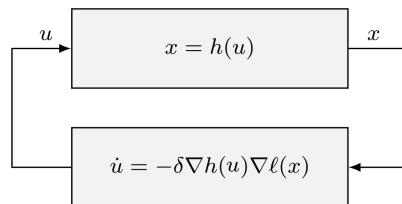
$$\min_{\mathbf{u} \in \mathbb{R}^m} l(h(\mathbf{u}))$$

Reduced feedback optimization problem

and can be solved using the gradient flow:

$$\dot{\mathbf{u}}(t) = -\delta \nabla h(\mathbf{u}(t)) \nabla l(h(\mathbf{u}(t)))$$

where  $\delta$  is a hyperparameter (note that it is not the step size as it is not needed in continuous time).

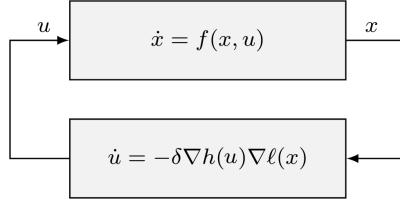


| **Remark.** By assumption,  $h$  is not available.

## 8.1 Centralized feedback optimization

**Feedback optimization based on time-scale separation** Gradient flow of the reduced feedback optimization problem where the steady-state  $h(\mathbf{u}(t))$  is approximated with the current state  $\mathbf{x}(t)$ .

Feedback optimization based on time-scale separation



Intuitively, with  $\delta > 0$  sufficiently small, we are creating a time-scale separation (i.e., dynamics working on different time-scales) between plant and optimization dynamics:

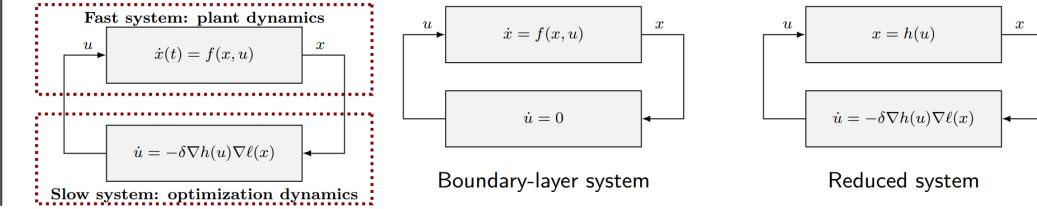
- The plant  $\mathbf{x}(t)$  is a fast dynamics that tracks  $h(\mathbf{u}(t))$  (i.e., the equilibrium for the current  $\mathbf{u}$ ) so that  $\nabla l(\mathbf{x}(t)) \approx \nabla l(h(\mathbf{u}(t)))$  for the optimization dynamics.
- The optimization dynamics is a slow dynamics that changes the input, thus refining the equilibrium that minimizes the loss.

**Remark** (Time-scale separation theory). Method to analyze a dynamical system by identifying a slow and fast dynamics, and studying them separately. Auxiliary systems are introduced:

**Boundary-layer system** with  $\mathbf{u}$  constant to model the fast system. It is exponentially stable by assumption and can be associated with the Lyapunov function  $U(\mathbf{x})$ .

**Reduced system** with the gradient flow without the state  $\mathbf{x}$  to model the slow system. It can be shown that it is asymptotically stable with the Lyapunov function  $W(\mathbf{u})$ .

The stability of the whole dynamics is studied by using  $U + W$  as Lyapunov function and a sufficiently small  $\delta$ .



**Theorem 8.1.1** (Feedback optimization based on time-scale separation convergence). Consider a closed-loop system and assume that:

- $\bar{\mathbf{x}} = h(\bar{\mathbf{u}})$  is a globally exponentially stable equilibrium,
- $l(h(\cdot))$  is radially unbounded and possibly non-convex,
- $\nabla l(h(\cdot)), \nabla l, h$ , and  $f$  are Lipschitz continuous.

It holds that, for a sufficiently small  $\delta > 0$ , the trajectory  $(\mathbf{x}(t), \mathbf{u}(t))$  of the system satisfies:

$$\lim_{t \rightarrow \infty} \text{dist} \left( \begin{bmatrix} \mathbf{x}(t) - h(\mathbf{u}(t)) \\ \mathbf{u}(t) \end{bmatrix}, \begin{bmatrix} 0 \\ \mathbf{u}^* \in \mathcal{U}^* \end{bmatrix} \right) = 0$$

where  $\text{dist}$  is the set distance and  $\mathcal{U}^*$  is the set of stationary points of  $l(h(\cdot))$ .

## 8.2 Distributed feedback optimization

**Distributed feedback optimization** Network of  $N$  agents each with dynamics:

$$\dot{\mathbf{x}}_i(t) = f_i(\mathbf{x}_i(t), \mathbf{u}_i(t))$$

Distributed feedback optimization

The goal is to design a control law  $\mathbf{u}_i(t) = \kappa_i(\{\mathbf{x}_j(t)\}_{j \in \mathcal{N}_i}, t)$  such that  $\mathbf{u}_1(t), \dots, \mathbf{u}_N(t)$  and  $\mathbf{x}_1(t), \dots, \mathbf{x}_N(t)$  converge to the solution of the problem:

$$\begin{aligned} & \min_{\substack{(\mathbf{z}_1, \dots, \mathbf{z}_N) \\ (\mathbf{u}_1, \dots, \mathbf{u}_N)}} \sum_{i=1}^N l_i(\mathbf{z}_i, \sigma(\mathbf{z})) \\ & \text{subject to } \mathbf{z}_i = h_i(\mathbf{u}_i) \end{aligned}$$

where  $\sigma(\mathbf{z}) = \frac{1}{N} \sum_{i=1}^N \phi_i(\mathbf{z}_i)$  is an aggregation function.

**Reduced distributed feedback optimization** By injecting the constraint into the cost function, the problem becomes:

$$\min_{(\mathbf{u}_1, \dots, \mathbf{u}_N)} \sum_{i=1}^N l_i(h_i(\mathbf{u}_i), \sigma(h(\mathbf{u})))$$

Reduced distributed feedback optimization

and can be solved using the parallel gradient flow:

$$\dot{\mathbf{u}}_i = -\delta_1 \nabla h_i(\mathbf{u}_i) \left( \nabla_{[h_i(\mathbf{u}_i)]} l_i(h_i(\mathbf{u}_i), \sigma(h(\mathbf{u}))) + \left( \sum_{j=1}^N \nabla_{[\sigma(h(\mathbf{u}))]} l_j(h_j(\mathbf{u}_j), \sigma(h(\mathbf{u}))) \right) \frac{1}{N} \nabla \phi_i(h_i(\mathbf{u}_i)) \right)$$

where  $\delta_1 > 0$  is a hyperparameter.

**Remark.** This formulation uses  $h$  which is not available and requires global information.

**Aggregative tracking feedback** Use  $\mathbf{x}_i$  to approximate  $h_i(\mathbf{u}_i)$  and dynamic average consensus for the aggregation function. The dynamics is:

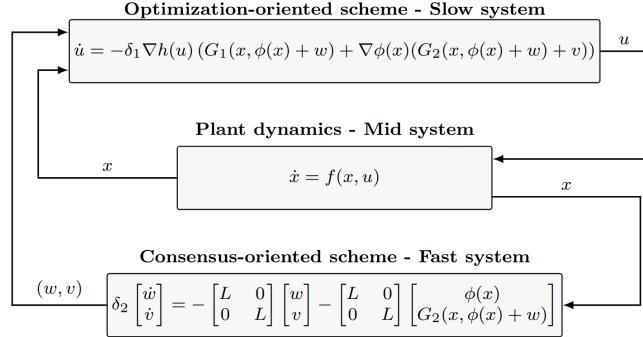
$$\begin{aligned} \dot{\mathbf{u}}_i &= -\delta_1 \nabla h_i(\mathbf{u}_i) \left( \nabla_{[\mathbf{x}_i]} l_i(\mathbf{x}_i, \phi_i(\mathbf{x}_i) + \mathbf{w}_i) + (\nabla_{[\phi_i(\mathbf{x}_i) + \mathbf{w}_i]} l_i(\mathbf{x}_i, \phi_i(\mathbf{x}_i) + \mathbf{w}_i) + \mathbf{v}_i) \nabla \phi_i(\mathbf{x}_i) \right) \\ \delta_2 \dot{\mathbf{w}}_i &= - \sum_{j \in \mathcal{N}_i} a_{ij} (\mathbf{w}_i - \mathbf{w}_j) - \sum_{j \in \mathcal{N}_i} a_{ij} (\phi_i(\mathbf{x}_i) - \phi_i(\mathbf{x}_j)) \\ \delta_2 \dot{\mathbf{v}}_i &= - \sum_{j \in \mathcal{N}_i} a_{ij} (\mathbf{v}_i - \mathbf{v}_j) - \sum_{j \in \mathcal{N}_i} a_{ij} (\nabla_{[\phi_i(\mathbf{x}_i) + \mathbf{w}_i]} l_i(\mathbf{x}_i, \phi_i(\mathbf{x}_i) + \mathbf{w}_i) - \nabla_{[\phi_j(\mathbf{x}_j) + \mathbf{w}_j]} l_j(\mathbf{x}_j, \phi_j(\mathbf{x}_j) + \mathbf{w}_j)) \end{aligned}$$

Aggregative tracking feedback

where  $\delta_1 > 0$  is used to slow down  $\dot{\mathbf{u}}_i$  and  $\delta_2 > 0$  to speed up  $\dot{\mathbf{w}}_i$  and  $\dot{\mathbf{v}}_i$ .

**Remark.** The formulation for dynamic average consensus is for the continuous case with the Laplacian.

**Remark.** The overall system works on three time-scales.



**Theorem 8.2.1** (Aggregative tracking feedback convergence). If:

- The aggregative tracking feedback algorithm is initialized so that  $\sum_{i=1}^N \text{col}(\mathbf{w}_i(0), \mathbf{v}_i(0)) = 0$ ,
- $G$  is strongly connected and weight balanced,
- $\bar{\mathbf{x}}_i = h_i(\bar{\mathbf{u}}_i)$  is globally exponentially stable,
- $\sum_{i=1}^N l_i(h_i(\cdot), \sigma(h(\cdot)))$  is radially unbounded and possibly non-convex,
- $\nabla_{[\cdot]} l_i$ ,  $\phi_i$ , and  $h_i$  are Lipschitz continuous.

Then, for small  $\delta_1 > 0$  and  $\delta_2 > 0$ , the trajectory of the system satisfies:

$$\lim_{t \rightarrow \infty} \text{dist} \left( \begin{bmatrix} \mathbf{x}(t) - h(\mathbf{u}(t)) \\ \mathbf{u}(t) \end{bmatrix}, \begin{bmatrix} 0 \\ \mathbf{u}^* \in \mathcal{U}^* \end{bmatrix} \right) = 0$$

where  $\mathcal{U}^*$  is the set of stationary points of  $\sum_{i=1}^N l_i(h_i(\cdot), \sigma(h(\cdot)))$ .

# 9 Neural networks

**Supervised learning** Given  $M$  data-label samples  $\{(\mathcal{D}^1, p^1), \dots, (\mathcal{D}^M, p^M)\}$ , the goal is to approximate the mapping through a non-linear function  $\phi(\cdot; \mathbf{u})$  parametrized on  $\mathbf{u}$ .

**Neuron model** Computational unit composed of a set of weights  $\mathbf{u} \in \mathbb{R}^d$  ( $\mathbb{R}^{d+1}$  if with bias) that, given an input  $\mathbf{x} \in \mathbb{R}^d$ , computes:

$$x^+ = \sigma(\mathbf{x}^T \mathbf{u} + u_b)$$

where  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  is an activation function.

**Remark.** The bias can be easily added by considering as weights  $[u_b \quad \mathbf{u}]^T$  and as input  $[1 \quad \mathbf{x}]^T$ .

**Multi-layer perceptron** Network with  $T$  layers each (for simplicity) with  $d$  neurons where the  $h$ -th unit at layer  $t$  has weights  $\mathbf{u}_{h,t} \in \mathbb{R}^d$ . The update at each neuron is defined as:

$$x_{h,t+1} = \sigma(\mathbf{x}_t^T \mathbf{u}_{h,t}) \quad x_{h,0} = \mathcal{D}_h^i$$

In matrix form, it becomes:

$$\begin{aligned} \begin{bmatrix} x_{1,t+1} \\ \vdots \\ x_{d,t+1} \end{bmatrix} &= \begin{bmatrix} \sigma(\mathbf{x}_t^T \mathbf{u}_{1,t}) \\ \vdots \\ \sigma(\mathbf{x}_t^T \mathbf{u}_{d,t}) \end{bmatrix} \\ \mathbf{x}_{t+1} &= f(\mathbf{x}_t, \mathbf{u}_t) \quad \mathbf{u}_t = \begin{bmatrix} \mathbf{u}_{1,t} \\ \vdots \\ \mathbf{u}_{d,t} \end{bmatrix} \in \mathbb{R}^{d^2} \end{aligned}$$

Supervised learning

Neuron model

Multi-layer perceptron

## 9.1 Training problem definition

**Single sample training** Task of finding  $\mathbf{u} = (\mathbf{u}_0, \dots, \mathbf{u}_{T-1})$  such that at the last layer  $t = T$  the prediction is as accurate as possible:

$$\|\mathbf{x}_T - p\| < \varepsilon$$

Single sample training

By using forward simulation of the dynamics  $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$ , we can obtain the output of the last layer as:

$$\mathbf{x}_T = \phi(\mathbf{x}_0; \mathbf{u}) = \phi(\mathcal{D}; \mathbf{u})$$

where  $\phi$  is called shooting map and it passes the data sample through the layers (from a deep learning point-of-view, it represents the composition of function).

The best weights  $\mathbf{u}^*$  can be obtained by solving:

$$\min_{\mathbf{u}} l(\mathbf{x}_T; p) = \min_{\mathbf{u}} l(\phi(\mathcal{D}; \mathbf{u}); p)$$

where  $l$  is the loss.

**Remark.** In optimal control, the learning problem is a reduced/condensed problem and the algorithm to solve it is a direct single shooting.

By defining:

$$J(\mathbf{u}) = l(\phi(\mathcal{D}; \mathbf{u}); p)$$

The reduced optimization problem is:

$$\min_{\mathbf{u}} J(\mathbf{u})$$

And can be solved using the gradient method:

$$\mathbf{u}^{k+1} = \mathbf{u}^k - \alpha^k \nabla J(\mathbf{u}^k)$$

**Multiple samples training** With multiple samples, the shooting function is applied at each data point:

$$\mathbf{x}_T^m = \phi(\mathbf{x}_0^m; \mathbf{u})$$

Multiple samples training

**Remark.**  $\mathbf{u}$  is independent of  $m$  (it is called ensemble control).

The optimization problem becomes:

$$\min_{\mathbf{u}} \sum_{m=1}^M J_m(\mathbf{u}) \quad J_m(\mathbf{u}) = l(\phi(\mathbf{x}_0^m; \mathbf{u}); p^m)$$

And its solution with the gradient method is:

$$\mathbf{u}^{k+1} = \mathbf{u}^k - \alpha^k \sum_{m=1}^M \nabla J_m(\mathbf{u}^k)$$

## 9.2 Backpropagation

### 9.2.1 Preliminaries

**Finite-horizon optimal control problem** Optimization problem defined as:

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{u}} & \sum_{t=0}^{T-1} l_t(\mathbf{x}_t, \mathbf{u}_t) + l_T(\mathbf{x}_T) \quad \mathbf{x}_0 = \mathbf{x}_{\text{init}} \\ \text{subject to } & \mathbf{x}_{t+1} = f_t(\mathbf{x}_t, \mathbf{u}_t) \end{aligned}$$

Finite-horizon optimal control problem

where:

- $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_T)$  are the state trajectories,
- $\mathbf{u} = (\mathbf{u}_1, \dots, \mathbf{u}_{T-1})$  are the input trajectories,
- $f_t : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$  for  $t = 0, \dots, T-1$  are the dynamics,
- $l_t : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$  for  $t = 0, \dots, T-1$  are the stage costs,
- $l_T : \mathbb{R}^n \rightarrow \mathbb{R}$  is the terminal cost.

**Adjoint method (general case)** Algorithm to compute the gradient of the cost function of a finite-horizon optimal control problem.

Adjoint method (general case)

Given the initial trajectory  $(\mathbf{x}^0, \mathbf{u}^0)$ , the method works as follows:

1. Repeat for the number of iterations  $k = 0, 1, \dots$ :

a) Perform backward simulation of the co-state  $\lambda$  for  $t = T - 1, \dots, 0$ :

$$\begin{aligned}\lambda_t &= \nabla_{[\mathbf{x}_t^k]} l_t(\mathbf{x}_t^k, \mathbf{u}_t^k) + \nabla_{[\mathbf{x}_t^k]} f_t(\mathbf{x}_t^k, \mathbf{u}_t^k) \lambda_{t+1} & \lambda_T &= \nabla l_T(\mathbf{x}_T^k) \\ \Delta \mathbf{u}_t^k &= \nabla_{[\mathbf{u}_t^k]} l_t(\mathbf{x}_t^k, \mathbf{u}_t^k) + \nabla_{[\mathbf{u}_t^k]} f_t(\mathbf{x}_t^k, \mathbf{u}_t^k) \lambda_{t+1}\end{aligned}$$

**Remark.** Intuitively,  $\lambda_t$  is the derivative of the cost function w.r.t. the first argument and  $\Delta \mathbf{u}_t^k$  is w.r.t. the second.

b) Apply descent step on the control input for  $t = 0, \dots, T - 1$ :

$$\mathbf{u}_t^{k+1} = \mathbf{u}_t^k - \alpha^k \Delta \mathbf{u}_t^k$$

c) Apply forward simulation of the dynamics for  $t = 0, \dots, T - 1$ :

$$\mathbf{x}_{t+1}^{k+1} = f_t(\mathbf{x}_t^{k+1}, \mathbf{u}_t^{k+1}) \quad \mathbf{x}_0^{k+1} = \mathbf{x}_{\text{init}}$$

**Adjoint method (simplified)** Without stage cost and with a time-invariant dynamics, the problem becomes:

$$\begin{aligned}\min_{\mathbf{x}, \mathbf{u}} l_T(\mathbf{x}_T) & \quad \mathbf{x}_0 = \mathbf{x}_{\text{init}} \\ \text{subject to } \mathbf{x}_{t+1} &= f(\mathbf{x}_t, \mathbf{u}_t)\end{aligned}$$

Adjoint method (simplified)

The backward simulation of the co-state becomes:

$$\begin{aligned}\lambda_t &= \nabla_{[\mathbf{x}_t^k]} f(\mathbf{x}_t^k, \mathbf{u}_t^k) \lambda_{t+1} & \lambda_T &= \nabla l_T(\mathbf{x}_T^k) \\ \Delta \mathbf{u}_t^k &= \nabla_{[\mathbf{u}_t^k]} f(\mathbf{x}_t^k, \mathbf{u}_t^k) \lambda_{t+1}\end{aligned}$$

**Remark.** The co-states  $\lambda_t$  represent the partial derivatives necessary to apply the chain rule and  $\Delta \mathbf{u}_t = \frac{\partial J(\mathbf{u})}{\partial \mathbf{u}_t}$ .

### 9.2.2 Adjoint method for neural networks

**Backpropagation (one-sample)** The simplified adjoint method is equivalent to the back-propagation algorithm for neural networks with:

$$f(\mathbf{x}_t, \mathbf{u}_t) = \begin{bmatrix} f_1(\mathbf{x}_t^k, \mathbf{u}_t^k) \\ \vdots \\ f_d(\mathbf{x}_t^k, \mathbf{u}_t^k) \end{bmatrix} = \begin{bmatrix} \sigma(\mathbf{x}_t^T \mathbf{u}_{1,t}) \\ \vdots \\ \sigma(\mathbf{x}_t^T \mathbf{u}_{d,t}) \end{bmatrix} \quad t = 0, 1, \dots, T - 1$$

Backpropagation (one-sample)

The gradient w.r.t. the first argument is:

$$\begin{aligned}\nabla_{[\mathbf{x}_t^k]} f(\mathbf{x}_t^k, \mathbf{u}_t^k) &= \left[ \nabla_{[\mathbf{x}_t^k]} f_1(\mathbf{x}_t^k, \mathbf{u}_t^k) \quad \dots \quad \nabla_{[\mathbf{x}_t^k]} f_d(\mathbf{x}_t^k, \mathbf{u}_t^k) \right] \\ &= \left[ \mathbf{u}_{1,t}^k \sigma'((\mathbf{x}_t^k)^T \mathbf{u}_{1,t}^k) \quad \dots \quad \mathbf{u}_{d,t}^k \sigma'((\mathbf{x}_t^k)^T \mathbf{u}_{d,t}^k) \right] \in \mathbb{R}^{d \times d}\end{aligned}$$

The gradient w.r.t. the second argument is:

$$\begin{aligned}\nabla_{[\mathbf{u}_t^k]} f(\mathbf{x}_t^k, \mathbf{u}_t^k) &= \left[ \nabla_{[\mathbf{u}_t^k]} f_1(\mathbf{x}_t^k, \mathbf{u}_t^k) \quad \dots \quad \nabla_{[\mathbf{u}_t^k]} f_d(\mathbf{x}_t^k, \mathbf{u}_t^k) \right] \\ &= \begin{bmatrix} \mathbf{x}_t^k \sigma'((\mathbf{x}_t^k)^T \mathbf{u}_{1,t}^k) & \dots & 0_d \\ 0_d & \ddots & 0_d \\ \vdots & & \vdots \\ 0_d & \dots & \mathbf{x}_t^k \sigma'((\mathbf{x}_t^k)^T \mathbf{u}_{d,t}^k) \end{bmatrix} \in \mathbb{R}^{d^2 \times d}\end{aligned}$$

**Remark.** When computing  $\nabla_{[\mathbf{u}_t^k]} f(\mathbf{x}_t^k, \mathbf{u}_t^k) \lambda_{t+1}$ , a summation is sufficient instead of performing the complete matrix multiplication:

$$\begin{bmatrix} \mathbf{x}_t^k \sigma'((\mathbf{x}_t^k)^T \mathbf{u}_{1,t}^k) & \dots & 0_d \\ 0_d & \ddots & 0_d \\ \vdots & & \vdots \\ 0_d & \dots & \mathbf{x}_t^k \sigma'((\mathbf{x}_t^k)^T \mathbf{u}_{d,t}^k) \end{bmatrix} \begin{bmatrix} \lambda_{1,t+1} \\ \vdots \\ \lambda_{d,t+1} \end{bmatrix}$$

**Backpropagation (multiple samples)** With  $M$  data points,  $\Delta \mathbf{u}_t^{m,k}$  is computed individually for each example and the update step is performed as:

$$\mathbf{u}_t^{k+1} = \mathbf{u}_t^k - \alpha^k \sum_{m=1}^M \Delta \mathbf{u}_t^{m,k}$$

Backpropagation  
(multiple samples)

### 9.2.3 Federated machine learning

**Federated machine learning** Given a parameter server and  $N$  agents each with  $M_i$  data points, the problem is defined as:

$$\min_{\mathbf{u}} \sum_{i=1}^N \sum_{m=1}^{M_i} l(\phi(\mathcal{D}^m; \mathbf{u}); p^m) = \min_{\mathbf{u}} \sum_{i=1}^N J_i(\mathbf{u})$$

Federated machine learning

Communication is only between the parameter server and the agents.

**Federated backpropagation** Algorithm that works as follows:

1. Repeat for the number of iterations  $k = 0, 1, \dots$ :
  - a) The parameter server sends the current weights  $\mathbf{u}_k$  to the agents.
  - b) Each agent computes the step direction  $\mathbf{d}_i^k = -\nabla J_i(\mathbf{u}^k)$  and sends it to the parameter server.
  - c) The parameter server performs the update step:

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \alpha^k \sum_{i=1}^N \mathbf{d}_i^k$$

Federated backpropagation

### 9.2.4 Distributed machine learning

**Distributed machine learning** Given  $N$  agents each with  $M_i$  data points, the problem is defined as:

$$\min_{\mathbf{u}} \sum_{i=1}^N \sum_{m=1}^{M_i} l(\phi(\mathcal{D}^m; \mathbf{u}); p^m) = \min_{\mathbf{u}} \sum_{i=1}^N J_i(\mathbf{u})$$

Distributed machine learning

Communication is only between neighboring agents.

**Distributed backpropagation** Algorithm that works as follows:

Distributed backpropagation

1. Repeat for the number of iterations  $k = 0, 1, \dots$ :
  - a) Each agent sends its local weights  $\mathbf{u}_i^k$  to its neighbors.
  - b) Each agent computes the local step direction  $\mathbf{d}_i^k = -\nabla J_i \left( \sum_{j \in \mathcal{N}_i} a_{ij} \mathbf{u}_j^k \right)$ .

c) Each agent performs the local update step:

$$\mathbf{u}_i^{k+1} = \sum_{j \in \mathcal{N}_i} \left( a_{ij} \mathbf{u}_j^k + \alpha^k \mathbf{d}_i^k \right)$$

<end of course>