Distributed Autonomous Systems

Last update: 03 May 2025

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

Contents

1 Graphs

1.1 Definitions

Directed graph (digraph) 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

Undirected graph Digraph where $\forall i, j : (i, j) \in E \Rightarrow (j, i) \in E$. Undirected graph

Subgraph Given a graph (I, E), (I', E') is a subgraph of it if $I' \subseteq I$ and $E' \subset E$. Subgraph

Spanning subgraph Subgraph where $I' = I$.

In-neighbor A node $j \in I$ is an in-neighbor of $i \in I$ if $(j, i) \in E$. In-neighbor

Set of in-neighbors The set of in-neighbors of $i \in I$ is the set: Set of in-neighbors

$$\mathcal{N}_i^{\text{IN}} = \{j \in I \mid (j, i) \in E\}$$

In-degree Number of in-neighbors of a node $i \in I$: In-degree

$$\deg_i^{\text{IN}} = |\mathcal{N}_i^{\text{IN}}|$$

Out-neighbor A node $j \in I$ is an out-neighbor of $i \in I$ if $(i, j) \in E$. Out-neighbor

Set of out-neighbors The set of out-neighbors of $i \in I$ is the set: Set of in-neighbors

$$\mathcal{N}_i^{\text{OUT}} = \{j \in I \mid (i, j) \in E\}$$

Out-degree Number of out-neighbors of a node $i \in I$: Out-degree

$$\deg_i^{\text{OUT}} = |\mathcal{N}_i^{\text{OUT}}|$$

Balanced digraph A digraph is balanced if $\forall i \in I : \deg_i^{\text{IN}} = \deg_i^{\text{OUT}}$. Balanced digraph

Periodic graph Graph where there exists a period $k > 1$ that divides the length of any cycle. Periodic graph

| **Remark.** A graph with self-loops is aperiodic.

Strongly connected digraph Digraph where each node is reachable from any node. Strongly connected digraph

Connected undirected graph Undirected graph where each node is reachable from any node. Connected undirected graph

Weakly connected digraph 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 digraph

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 $\boldsymbol{A}$ such that $\boldsymbol{A}_{i,j} = a_{i,j}$: Weighted adjacency matrix

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

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

$$\boldsymbol{D}^{\text{IN}} = \begin{bmatrix} \deg_1^{\text{IN}} & 0 & \cdots & 0 \\ 0 & \deg_2^{\text{IN}} & & \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & \deg_N^{\text{IN}} \end{bmatrix} \qquad \boldsymbol{D}^{\text{OUT}} = \begin{bmatrix} \deg_1^{\text{OUT}} & 0 & \cdots & 0 \\ 0 & \deg_2^{\text{OUT}} & & \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & \deg_N^{\text{OUT}} \end{bmatrix}$$

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

Remark. It holds that:

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

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

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

1.3 Laplacian matrix

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

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

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

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

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

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

Remark. $\boldsymbol{L}^{\mathrm{IN}}$ 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:

Distributed algorithm

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

where x_i^k is the state of agent i at time k and $\texttt{stf}_i$ 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:

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

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:

Linear averaging distributed algorithm (in-neighbors)

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

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:

Linear time-invariant (LTI) autonomous system

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

In matrix form, it becomes:

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

where $\boldsymbol{A}$ is the adjacency matrix of G^{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 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:

Linear averaging distributed algorithm (out-neighbors)

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

In matrix form, it becomes:

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

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

2.1.1 Stochastic matrices

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

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

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

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

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

Lemma 2.1.1. An adjacency matrix $\boldsymbol{A}$ 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 $\boldsymbol{A}$, if G is strongly connected and aperiodic, and $\boldsymbol{A}$ is row stochastic, its eigenvalues are such that:

- $\lambda = 1$ is a simple eigenvalue (i.e., algebraic multiplicity of 1),
- All others μ 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 $\boldsymbol{A}$. Assume G strongly connected and aperiodic, and $\boldsymbol{A}$ 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:

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

where $\tilde{w}_i = \frac{w_i}{\sum_{i=j}^N w_j}$ are all normalized and sum to 1 (i.e., they produce a convex combination).
Moreover, if $\boldsymbol{A}$ is doubly stochastic, then it holds that the consensus is the average:

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

Sketch of proof. Let $\boldsymbol{T} = \begin{bmatrix}\mathbf{1} & \mathbf{v}^2 & \cdots & \mathbf{v}^N\end{bmatrix}$ be a change in coordinates that transforms an adjacency matrix into its Jordan form $\boldsymbol{J}$:

$$\boldsymbol{J} = \boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T}$$

As $\lambda = 1$ is a simple eigenvalue (Lemma 2.1.2), it holds that:

$$\boldsymbol{J} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \boldsymbol{J}_2 & \\ 0 & & & \end{bmatrix}$$

where the eigenvalues of $\boldsymbol{J}_2 \in \mathbb{R}^{(N-1)\times(N-1)}$ lie inside the open unit disk.
Let $\mathbf{x}^k = \boldsymbol{T}\bar{\mathbf{x}}^k$, then we have that:

$$\begin{aligned} &\mathbf{x}^{k+1} = \boldsymbol{A}\mathbf{x}^k \\ &\iff \boldsymbol{T}\bar{\mathbf{x}}^{k+1} = \boldsymbol{A}(\boldsymbol{T}\bar{\mathbf{x}}^k) \\ &\iff \bar{\mathbf{x}}^{k+1} = \boldsymbol{T}^{-1}\boldsymbol{A}(\boldsymbol{T}\bar{\mathbf{x}}^k) = \boldsymbol{J}\bar{\mathbf{x}}^k \end{aligned}$$

Therefore:

$$\lim_{k\to\infty} \bar{\mathbf{x}}^k = \bar{x}_1^0 \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\bar{x}_1^{k+1} = \bar{x}_1^k \quad \forall k \geq 0$$

$$\lim_{k\to\infty} \bar{x}_i^k = 0 \quad \forall i = 2, \ldots, N$$

□

Example (Metropolis-Hasting weights)**.** Given an undirected unweighted graph G with edges of degrees $d_1, \ldots, d_n$, Metropolis-Hasting weights are defined as:

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

The matrix $\boldsymbol{A}$ of Metropolis-Hasting 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 $\{G(k)\}_{k\geq 0}$.

Time-varying digraph

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

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

Jointly strongly connected digraph

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

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

Uniformly jointly strongly/B-strongly connected digraph

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:

Averaging distributed algorithm over time-varying digraph

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

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

Linear time-varying (LTV) discrete-time system

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

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 $\{\boldsymbol{A}(k)\}_{k\geq 0}$. Assume:

Discrete-time consensus over time-varying graphs

- 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:

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

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

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

2.3 Continuous-time averaging algorithm

2.3.1 Laplacian dynamics

Network of dynamic systems Network described by the ODEs:

Network of dynamic systems

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

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 a_{ij}):

Laplacian dynamics system

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

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 $\mathbf{x} = \begin{bmatrix} x_1 & \dots & x_N \end{bmatrix}^T$, the system can be written in matrix form as:

Linear time invariant (LTI) continuous-time system

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

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

Proof. The system is defined as:

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

By rearranging, we have that:

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

Which in matrix form is:

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

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

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

□

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:

$$\boldsymbol{L}\mathbf{1} = \boldsymbol{D}^{\text{OUT}}\mathbf{1} - \boldsymbol{A}\mathbf{1} = \begin{bmatrix} \text{deg}_1^{\text{OUT}} \\ \vdots \\ \text{deg}_i^{\text{OUT}} \end{bmatrix} - \begin{bmatrix} \text{deg}_1^{\text{OUT}} \\ \vdots \\ \text{deg}_i^{\text{OUT}} \end{bmatrix} = 0$$

Lemma 2.3.2. The Laplacian $\boldsymbol{L}$ 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 $\boldsymbol{L}$, the following are equivalent:

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

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

Continuous-time consensus

Theorem 2.3.2 (Continuous-time consensus)**.** Consider a continuous-time average system with a strongly connected weighted digraph G and Laplacian $\boldsymbol{L}$. Assume that the system follows the Laplacian dynamics $\dot{\mathbf{x}}(t) = -\boldsymbol{L}\mathbf{x}(t)$ for $t \geq 0$.
It holds that there exists a left eigenvector $\mathbf{w}$ of $\boldsymbol{L}$ with eigenvalue $\lambda = 0$ such that the consensus converges to:

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

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

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

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

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:

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

where $\boldsymbol{L}_f$ is the followers' Laplacian, $\boldsymbol{L}_l$ the leaders', and $\boldsymbol{L}_{fl}$ 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:

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

Example. Consider a path graph with four nodes:

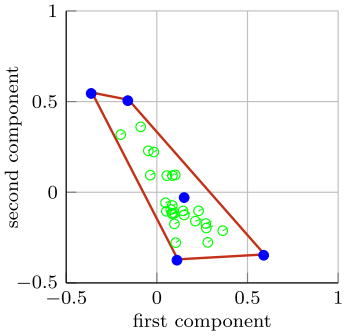
$$0 \leftrightarrow 1 \leftrightarrow 2 \leftrightarrow (3)$$

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

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

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:

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

In matrix form, it becomes:

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

where $\boldsymbol{L}_f$ can be seen as the state matrix and $\boldsymbol{L}_{fl}$ 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 $\boldsymbol{L}_f$ is positive definite.

Proof. We need to prove that:

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

As G is undirected, it holds that:

- The complete Laplacian $\boldsymbol{L}$ 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:

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

The following two arguments can be made:

1. By choosing $\bar{\mathbf{x}} = \begin{bmatrix} \mathbf{x}_f & 0 \end{bmatrix}^T$, it holds that:

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

2. The only case when $\mathbf{x}^T \boldsymbol{L}\mathbf{x} = 0$ 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 $\forall \mathbf{x}_f \neq 0 : \mathbf{x}_f^T \boldsymbol{L}_f \mathbf{x}_f \neq 0$.

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

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

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

Containment optimality

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

- 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:

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

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

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

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

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:

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

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 $\dot{\mathbf{x}}_f = -\boldsymbol{L}_f \mathbf{x}_f$ is globally asymptotically stable, by Lemma 3.1.3, it holds that $\dot{\mathbf{e}}(t) = -\boldsymbol{L}_f \mathbf{e}(t)$ 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 $\mathbf{x}_E = \begin{bmatrix} \mathbf{x}_{f,E} & \mathbf{x}_l \end{bmatrix}^T$ 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:

$$\left(\sum_{k=1}^{N} a_{ik}\right) x_{E,i} = \sum_{j=1}^{N} a_{ij} x_{E,j} \qquad \forall i \in \{1, \dots, N_f\}$$

$$x_{E,i} = \sum_{j=1}^{N} \frac{a_{ij}}{\sum_{k=1}^{N} a_{ik}} x_{E,j} \quad \forall i \in \{1, \dots, N_f\}$$

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

$$\dot{\mathbf{x}}_f(t) = -\boldsymbol{L}_f \mathbf{x}_f(t) - \boldsymbol{L}_{fl} \mathbf{x}_l(t) \qquad \mathbf{x}_f(0) = \mathbf{x}_f^{(0)}$$
$$\dot{\mathbf{x}}_l(t) = \mathbf{v}_0 \qquad \mathbf{x}_l(0) = \mathbf{x}_l^{(0)}$$

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:

$$0 = -\boldsymbol{L}_f \mathbf{x}_{f,E} - \boldsymbol{L}_{fl} \mathbf{x}_{l,E}$$
$$0 = \mathbf{v}_0$$

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

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

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

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

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

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

There are two cases:

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

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. □

3.3 Containment with non-static leaders and integral action

Containment with non-static leaders and integral action

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

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

where $\mathbf{u}_f(t)$ 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):

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

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

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

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

The control action can be defined as:

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

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

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

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

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

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

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

In matrix form, it can be defined as:

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

3.5 Containment with multivariate states

Containment with multivariate states

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

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

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

4 Optimization

4.1 Definitions

4.1.1 Unconstrained optimization

Unconstrained optimization Problem of form:

Unconstrained optimization

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

where $l : \mathbb{R}^d \to \mathbb{R}$ 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 $l : \mathbb{R}^d \to \mathbb{R}$ such that $l \in C^1$ in $B(\mathbf{z}^*, \varepsilon)$ (i.e., neighbors of $\mathbf{z}^*$ within a radius ε), it holds that:

First-order necessary condition of optimality

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

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

Second-order necessary condition of optimality

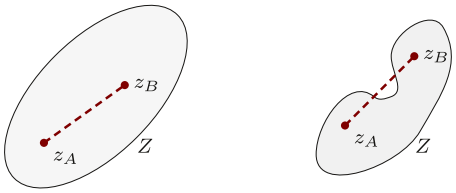
$$\mathbf{z}^* \text{ is local minimum } \Rightarrow \nabla^2 l(\mathbf{z}^*) \geq 0 \text{ (i.e., positive semidefinite)}$$

4.1.2 Convexity

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

Convex set

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

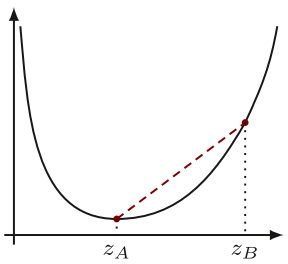


Convex set Nonconvex set

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

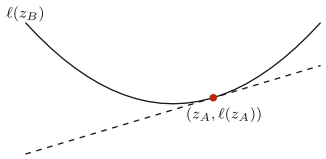
Convex function

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



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

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



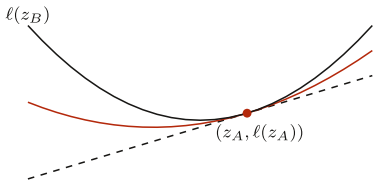
Strongly convex function Given a convex set $Z \subseteq \mathbb{R}^d$, a function $l : Z \to \mathbb{R}$ is strongly convex with parameter $\mu > 0$ if it holds that: Strongly convex function

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

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

Remark. Given a differentiable and μ-strongly convex function $l : Z \to \mathbb{R}$, it holds that any of its points lie above all the paraboloids with curvature determined by μ 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 ∇l is a monotone operator, which means that it satisfies: Convexity and gradient monotonicity

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

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

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

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

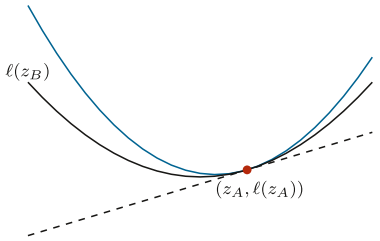
$$\forall \mathbf{z}_A, \mathbf{z}_B : \big(\nabla l(\mathbf{z}_A) - \nabla l(\mathbf{z}_B)\big)^T(\mathbf{z}_A - \mathbf{z}_B) \geq \mu\|\mathbf{z}_A - \mathbf{z}_B\|^2$$

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

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

Remark. Given a differentiable function l with L-Lipschitz continuous gradient ∇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.

Convexity and Lipschitz continuity of gradient

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

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

Strong convexity and Lipschitz continuity of gradient

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

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

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

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:

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

4.2.1 Gradient method

Gradient method

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

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

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 α^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 $\alpha^k \overset{k\to\infty}{\longrightarrow} 0$. To avoid decreasing the step too much, a typical choice is an α^k such that:

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

Line search Algorithmic methods such as the Armijo rule.

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

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

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

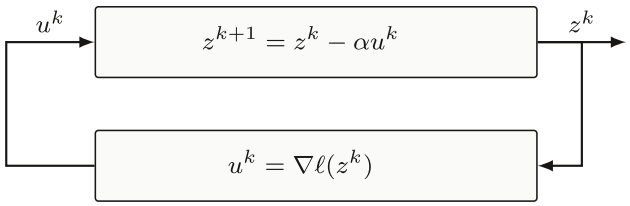
Possible choices for $\boldsymbol{D}^k$ are:

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

- ∇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., $\nabla l(\bar{z}) = 0$).
In addition, if l is μ-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 μ, 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 μ-strongly convex and the step size is constant, the sequence converges exponentially.

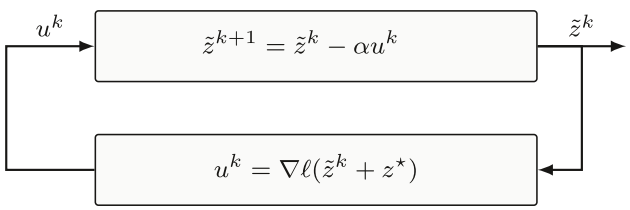
   Remark. As l is convex, its equilibrium is also the global minimum $\mathbf{z}^*$.

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

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

   The system in the new coordinates becomes:

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

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

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

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

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

It holds that:

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

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:

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

□

Gradient method for a quadratic function

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

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

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

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

For a sufficiently small α, the matrix $(\boldsymbol{I} - \alpha\boldsymbol{Q})$ 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:

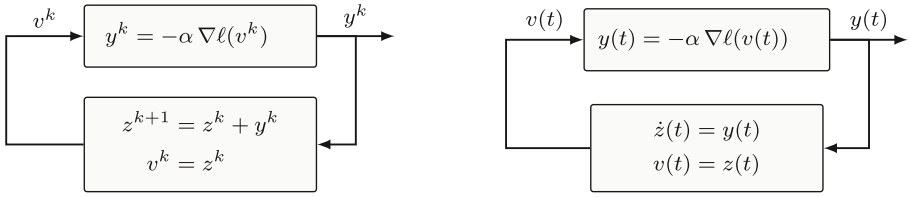
$$\begin{aligned} \mathbf{z}^k &= (\boldsymbol{I} - \alpha\boldsymbol{Q})^k\mathbf{z}^0 - \alpha\sum_{\tau=0}^{k-1}(\boldsymbol{I} - \alpha\boldsymbol{Q})^\tau\mathbf{r} \\ &\overset{k\to\infty}{\longrightarrow} -\alpha\left(\sum_{\tau=0}^{\infty}(\boldsymbol{I} - \alpha\boldsymbol{Q})^\tau\right)\mathbf{r}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

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:

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 η^0 and η^{-1}, the algorithm is defined as:

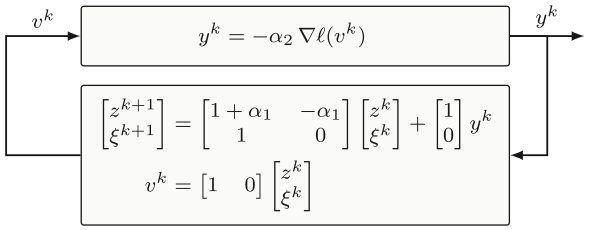
Heavy-ball method

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

with $\alpha_1, \alpha_2 > 0$.

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

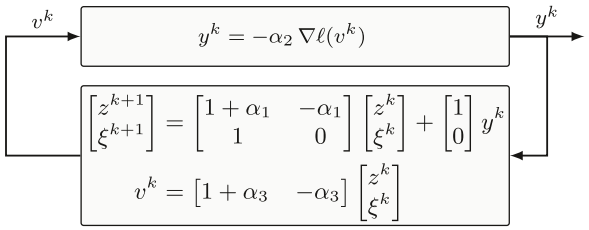
Generalized heavy-ball method Given ζ^0 and ζ^{-1}, the algorithm is defined as:

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 $l_i : \mathbb{R}^d \to \mathbb{R}$, each local and private to an agent: Cost-coupled optimization

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

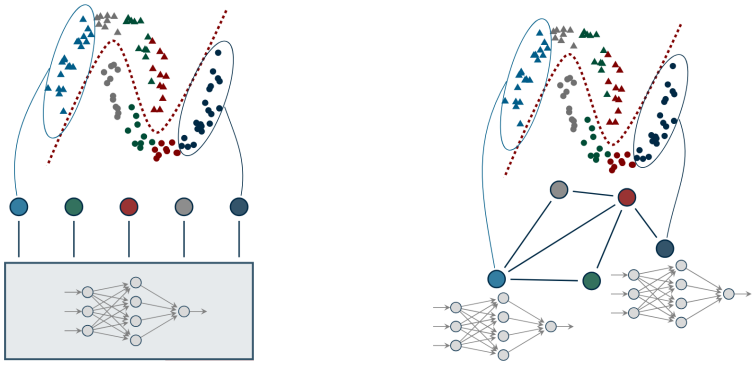
4.3.1 Learning paradigms

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): Federated learning

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

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



(a) Federated learning (b) 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: Batch gradient method

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

| **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: Incremental gradient method

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

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.

Stochastic gradient descent (SGD)

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:

SGD convergence with constant step size

- l is μ-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.

SGD convergence with diminishing step size

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 $\mathcal{W}$:

Mini-batch SGD

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

4.4.4 Adaptive momentum

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

$$\mathbf{m}^{k+1} = \beta_1 \mathbf{m}^k + (1-\beta_1)\nabla l(\mathbf{z}^k, w^k)$$
$$\mathbf{v}^{k+1} = \beta_2 \mathbf{v}^k + (1-\beta_2)\left(\nabla l(\mathbf{z}^k, w^k)\right)^2$$

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

The descent direction is defined as:

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

The update is performed as:

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

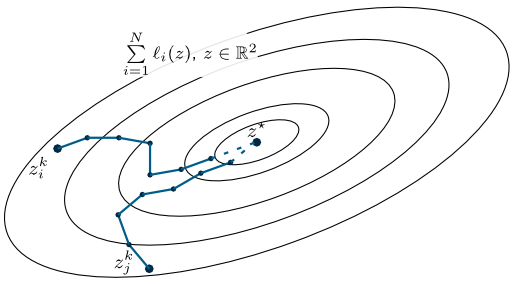
4.5 Distributed cost-coupled/consensus optimization

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: Distributed cost-coupled optimization

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

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): Distributed gradient algorithm

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

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

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

Distributed gradient algorithm convergence

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

- The matrix $\boldsymbol{A}$ 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.

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 $\mathbf{z}^*$:

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

Distributed projected subgradient algorithm

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:

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

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

Distributed projected subgradient algorithm convergence

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

- The adjacency matrix $\boldsymbol{A}$ 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.

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 α is an equilibrium:

$$\begin{aligned}\mathbf{z}^* &= \sum_{j=1}^N a_{ij}\mathbf{z}^* - \alpha \nabla l_i \left(\sum_{j=1}^N a_{ij}\mathbf{z}^*\right) \\ &= \mathbf{z}^* - \alpha \nabla l_i(\mathbf{z}^*) \qquad \boldsymbol{A} \text{ doubly stochastic and } \mathbf{z}^* \text{ constant}\end{aligned}$$

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. □

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:

Dynamic average consensus

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

The average signal estimated by an agent is represented by a state s_i^k and we want that $\lim_{k\to\infty} \|s_i^k - \bar{r}^k\| = 0$. 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 $C_2 > 0$:

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

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

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

Gradient tracking algorithm

$$d_i^k \underset{k\to\infty}{\longrightarrow} -\frac{1}{N}\sum_{h=1}^{N} \nabla l_h(\mathbf{z}_h^k)$$

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:

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

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:

Gradient tracking algorithm optimality

- $\boldsymbol{A}$ 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 μ-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 $\mathbf{z}^*$:

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

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

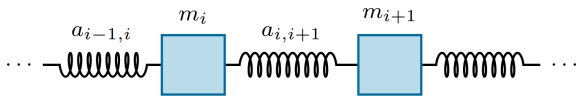
$$\exists \rho \in (0,1) : \|\mathbf{z}_i^k - \mathbf{z}^*\| \leq \rho \|\mathbf{z}_i^{k+1} - \mathbf{z}^*\| \ \wedge \ \rho \|\mathbf{z}_i^{k+1} - \mathbf{z}^*\| \leq \rho^k \|\mathbf{z}_i^0 - \mathbf{z}^*\|$$

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

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 $a_{j,i} = a_{i,j} > 0$. Mass-spring system



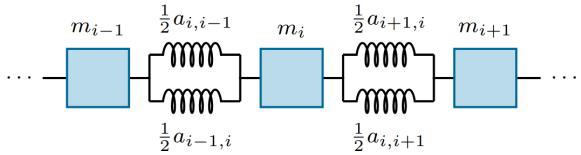
The elastic force $F_{e,i}(x)$ at mass i is given by:

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

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

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

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:

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

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

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

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

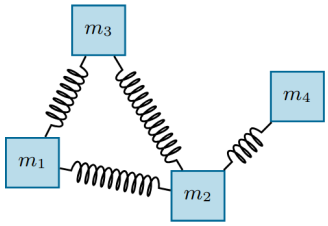
Then, the potential energy at mass i can be written as:

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

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

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

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:

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

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

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

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

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

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

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

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

And consensus is reached at a stationary point of $V(x)$.

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):

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

Formation control

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:

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

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

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

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

The gradient flow dynamics is then:

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

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

Collision avoidance potential function/Barrier function

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

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

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:

Formation control with obstacle avoidance

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

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: Cooperative robotics

- 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 $\sigma(\mathbf{z}) : \mathbb{R}^{2N} \to \mathbb{R}^2$ 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 $\mathbf{z}_i \in \mathbb{R}^{n_i}$,
- $l_i : \mathbb{R}^{n_i} \to \mathbb{R}^d$ 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 $\phi_i : \mathbb{R}^{n_i} \to \mathbb{R}^d$

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 ϕ_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 $\sigma : \mathbb{R}^N \to \mathbb{R}$. The update step, assuming global access to the parameters, can be performed as: Centralized gradient method (scalar)

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

By expanding the derivative, we have that:

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

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

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

And the gradient can be expanded as:

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

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: Aggregative tracking distributed optimization algorithm

- 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 $\sum_{j=1}^{N} \nabla_{[\sigma(\mathbf{z}^k)]} l_j(\mathbf{z}_j^k, \sigma(\mathbf{z}^k))$.

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 σ.

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

- The communication digraph G is strongly connected and aperiodic, and $\boldsymbol{A}$ 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 α^* 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:

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

6.1.3 Online aggregative optimization

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

Projected aggregative tracking Algorithm for online aggregative optimization defined as:

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

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

7 Multi-robot safety controllers

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

Control-affine non-linear dynamical system

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

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:

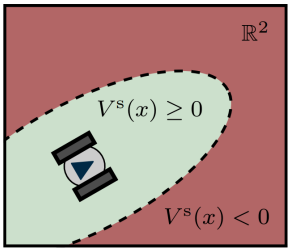
- A high-level controller that produces the direction $\mathbf{u}^{\text{ref}}(\mathbf{x})$ towards the target position.
- A safety layer that modifies $\mathbf{u}^{\text{ref}}(\mathbf{x})$ into $\mathbf{u}(t) = \kappa(\mathbf{x})$ to account for obstacles.

Safety control Given a (sufficiently regular) function $V^s : X \subseteq \mathbb{R}^n \to \mathbb{R}$, 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 $\kappa^s : X \to \mathbb{R}^m$ 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:

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

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 $\gamma : \mathbb{R} \to \mathbb{R}$ with $\gamma(0) = 0$ such that the

Control barrier function (CBF)

following inequality (control barrier certificate) holds:

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

γ 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 γ 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 γ 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:

Set of admissible safe controllers

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

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:

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

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 $g(\mathbf{x}(t)) = \boldsymbol{I}$. The dynamics is:

Single integrator model

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

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

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

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

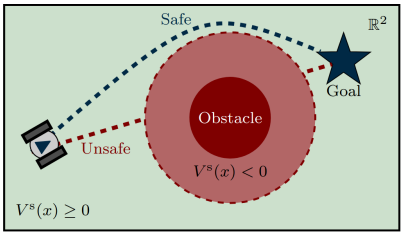
Therefore:

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

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 can be:

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

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

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

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:

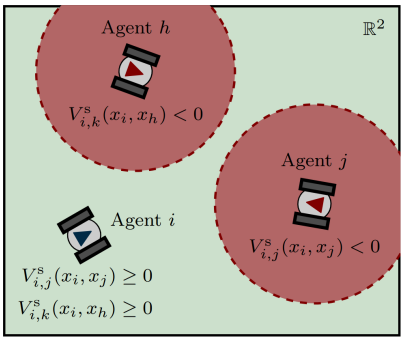
$$\arg\min_{\mathbf{u} \in U} \|\mathbf{u} - \mathbf{u}^{\text{ref}}(\mathbf{x})\|^2 \quad \text{subject to } a^T \mathbf{u} + b \leq 0$$

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:

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

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

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

8 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 $\mathbf{u}$. Supervised learning

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: Neuron model

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

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

> **Remark.** The bias can be easily added by considering as weights $\begin{bmatrix} u_b & \mathbf{u} \end{bmatrix}^T$ and as input $\begin{bmatrix} 1 & \mathbf{x} \end{bmatrix}^T$.

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: Multi-layer perceptron

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

In matrix form, it becomes:

$$\begin{bmatrix} x_{1,t+1} \\ \vdots \\ x_{d,t+1} \end{bmatrix} = \begin{bmatrix} \sigma(\mathbf{x}_t^T \mathbf{u}_{1,t}) \\ \vdots \\ \sigma(\mathbf{x}_t^T \mathbf{u}_{d,t}) \end{bmatrix}$$

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

8.1 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: Single sample training

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

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 ϕ 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:

Multiple samples training

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

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

The optimization problem becomes:

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

And its solution with the gradient method is:

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

8.2 Backpropagation

8.2.1 Preliminaries

Finite-horizon optimal control problem Optimization problem defined as:

Finite-horizon optimal control problem

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

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,
- $f_t : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ for $t = 0, \dots, T-1$ are the dynamics,
- $l_t : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ for $t = 0, \dots, T-1$ are the stage costs,
- $l_T : \mathbb{R}^n \to \mathbb{R}$ 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 $k = 0, 1, \ldots$:
   a) Perform backward simulation of the co-state λ for $t = T-1, \ldots, 0$:
   $$\lambda_t = \nabla_{[\mathbf{x}_t^k]} l_t(\mathbf{x}_t^k, \mathbf{u}_t^k) + \nabla_{[\mathbf{x}_t^k]} f_t(\mathbf{x}_t^k, \mathbf{u}_t^k)\lambda_{t+1} \qquad \lambda_T = \nabla l_T(\mathbf{x}_T^k)$$
   $$\Delta\mathbf{u}_t^k = \nabla_{[\mathbf{u}_t^k]} l_t(\mathbf{x}_t^k, \mathbf{u}_t^k) + \nabla_{[\mathbf{u}_t^k]} f_t(\mathbf{x}_t^k, \mathbf{u}_t^k)\lambda_{t+1}$$
   > **Remark.** Intuitively, λ_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 $t = 0, \ldots, T-1$:
   $$\mathbf{u}_t^{k+1} = \mathbf{u}_t^k - \alpha^k \Delta\mathbf{u}_t^k$$
   c) Apply forward simulation of the dynamics for $t = 0, \ldots, T-1$:
   $$\mathbf{x}_{t+1}^{k+1} = f_t(\mathbf{x}_t^{k+1}, \mathbf{u}_t^{k+1}) \qquad \mathbf{x}_0^{k+1} = \mathbf{x}_{\text{init}}$$

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

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

The backward simulation of the co-state becomes:

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

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

8.2.2 Adjoint method for neural networks

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

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

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

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

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

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

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: Backpropagation (multiple samples)

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

8.2.3 Federated machine learning

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

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

Communication is only between the parameter server and the agents.

Federated backpropagation Algorithm that works as follows: Federated backpropagation

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:

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

8.2.4 Distributed machine learning

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

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

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:

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