MODELING AND CONTROL OF CYBER-PHYSICAL SYSTEMS

Lecture notes

Carlo Migliaccio

AA 2023/2024

Contents

Part I

Modeling of Cyber-Physical systems

Chapter 1

Introduction

1.1 Some definitions

Definition (Helen Gill, 2006) "Cyber-Physical systems are physical, biological, and engineered systems whose operations are integrated, **monitored, and/or controlled** by a **computational core**. Components are **networked** at every scale. Computing is deeply embedded into every physical component, possibly even into materials. The computational core is an embedded system, usually demands real-time response, and is most often distributed"

Roughly speaking a CPS is a **collection of devices** that:

1. compute
2. inter-communicate
3. interact with the physical word

Figure 1.1: Three Dimensions of CPSs

At this point, we can distinguish in this scenario two layers for CPS: (1) The **Cyber Layer** which is linked to the computation and communication issues, (2) the **Physical Layer** deals with the interaction of the devices with the physical word.

1.2 Some examples of CPS

Examples of Cyber-Physical systems could be:

- **Automotive vehicles**: in a car you can find hundreds of sensors, actually we have a computational core, are interconnected in some way (eg. CAN), moreover several feedback-control systems can be find;
- **Teams of mobile robots** that aim to get a target. Robots collaborate to achieve a goal that can be: the exchange of information for example;
- **Wireless sensor networks** in order to monitor and area (indoor localization without a GPS)

1.3 Enabling Technologies and related problems

We can wonder: "How can we deploy CPS?". The answer is by using:

1. **Embedded systems**: hardware and software integrated within mechanical and electrical systems
2. **Sensors and Actuators** for monitoring and control purposes
3. **Communication Networks** for example Wireless communication

Despite the **implementation** it's not too much expensis it raises several issue: at first the **Vulnerability** which is linked to the **Safety** of the overall system.

1.4 Mathematical models for CPS

We can modelize a Cyber-Physical System by using:

- **Basic Models**: continuos-discrete time LTI systems
- **Hybrid models**: they can describe the interaction between devices and the physical layer including both continuous and discrete events;
- **Systems under adversarial attacks**: a CPS could be attacked in order to manipulate the exchanged information;
- **Multi-agent systems**: networks of intercommunicating *agents* which may collaborate to reach a *common goal*: **Consensus**.

Hybrid systems

In order to **model the presence of physical events** that can change the dynamics, we can consider **hybrid systems**, also known as **switched linear systems**. We can describe them by using the following formalism:

$$x(k+1) = A_{q_k}x(k) + B_{q_k}u(k)$$
$$y(k) = C_{q_k}x(k) + D_{q_k}u(k)$$

for $k = 0, 1, ...$

- $x(k)$ is the continuous state
- $q_k \in \{1, 2, ..., Q\}$ is the discrete state or mode, if $q_k \neq q_{k+1}$ then at the time k, the dynamics changes, there is a **switch** for the system
- The parameters denoted with $A_{q_k}, B_{q_k}, C_{q_k}, D_{q_k}$ are associated witch the active submodel, the parameters are piecewise constant. For this reason the hybrid systems could be seen as a particular case of LTI systems.

Some examples of hybrid systems could be: a bouncing ball, a robot moving with obstacles (which are items of the physical layer) in a room etc.

Modeling the presence of attacks

An attack in the framework of CPS can be modeled as an additive term either on the actuator or the (distributed) sensors.

$$x(k+1) = Ax(k) + Bu(k) + b(k)$$
$$y(k) = Cx(k) + Du(k) + a(k)$$

Where we indicate with:

- $b(k)$ the attacks on the actuators
- $a(k)$ the attacks on the sensors

We focus only on the term $a(k)$ which can alter the dynamics of the system. We can't model the attacks as a *disturbance/noise* as we could face them by using some techniques of the classical control theory (eg. Loop shaping).
A "good attack" can't be modeled in a proper way like a disturbance, but fortunately as the sensors are distributed, the attacks can be done only on a subset of them. There are in the common case **sparse attacks.** Our reference model to develop the theory is then the following:

$$x(k+1) = Ax(k) + Bu(k)$$
$$y(k) = Cx(k) + Du(k) + a(k)$$

Chapter 2

Secure estimation of CPSs state under adversarial attacks

2.1 State estimation and Observabilty

Let us consider the LTI system (without input):

$$x(k+1) = Ax(k)$$
$$y(k) = Cx(k)$$

$y(k) \in \mathbb{R}^q$, $x(k) \in \mathbb{R}^n$, $A \in \mathbb{R}^{n,n}$, $C \in \mathbb{R}^{q,n}$

The **State estimation** is the procedure by which we can **recover** the state $x(k) \in \mathbb{R}^n$ of the system from measurements $y(k)$ for $k = 0, 1, 2, ...$ **Note that...** every element of the vector $y(k)$ is a measure from a sensor, and so we have $y_i(k) \in \mathbb{R}$.
We can say that a system is **Observable** if exists a finite time $T \in \{0, 1, ...\}$ such that $x(0)$ can be recovered from the measurements $y(k), \quad k = 0, 1, ..., T-1$.

But why only $x(0)$? Let us compute $y(0), ..., y(T-1)$ to verify this fact:

$$y(0) = Cx(0)$$
$$y(1) = Cx(1) = CAx(0)$$
$$y(2) = Cx(2) = CA^2x(0)$$
$$...$$
$$y(T-1) = ... = CA^{T-1}x(0)$$

At this point by using the vectorial notation, we have:

$$\begin{pmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(T-1) \end{pmatrix} = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{T-1} \end{pmatrix} x(0)$$

$$\mathcal{O}_T = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{T-1} \end{pmatrix}$$

$\in \mathbb{R}^{qT,n}$ where if $T = n$ we call $\mathcal{O}_n$ the **Observability matrix**.
When the equation

$$\begin{pmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(T-1) \end{pmatrix} = \mathcal{O}_T x(0)$$

has a **unique solution** the system is observable. At this point, we distinguish two different cases:

- $qt < N$ the system is *underdetermined*
- $qt \geq N$ and $rank(\mathcal{O}_T) = n$, then we can (pseudo)invert it. In particular if $qT = n$ then

$$x(0) = \mathcal{O}_T^{-1} y(k)$$

otherwise if $qT > n$ then

$$x(0) = (\mathcal{O}_T^T \mathcal{O}_T)^{-1} \mathcal{O}_T^T y(k)$$

Where we call **Moore-Penrose pseudoinverse** the matrix $\mathcal{O}_T^\dagger = (\mathcal{O}_T^T \mathcal{O}_T)^{-1} \mathcal{O}_T^T$

Theorem 1 *(Kalman, 1960) An LTI system is observable if and only if rank($\mathcal{O}_n$)=n*

In general there are two approaches for the state estimation:

- **static approach**: solving the equation like we have just seen.
- **dynamic approach**: by using a **Luemberger Observer** that allow us to recover the state after a bunch of steps.

2.2 Secure state estimation

In this section we make a try to expand the concept of **Observability and state estimation** when we have attacks on the sensors, and so the additive term $a(k)$ in the output equation.

$$\begin{aligned} x(k+1) &= Ax(k) \\ y(k) &= Cx(k) + a(k) \end{aligned}$$

Assumption The term $a(k) \in \mathbb{R}^q$ is **sparse** in the sense that no more than $h \ll q$ of its elements are non-zero. By using the l_0-norm, $\|a\|_0 \leq h$.

We refer to **Secure state estimation** when we want recover the state $x(0)$ from sensors' measurements $y(k), k = 0, 1, ...$ and **unknown attacks** $a(k)$. (As we said we are not able to model attacks in a proper way, we wouldn't have a different theory and several techniques to face the problems related to them).

Let us anylize the mentioned problem by spotting the differences that occurs in the case of attacks:

$$\begin{aligned} y(0) &= Cx(0) + a(0) \\ y(1) &= Cx(1) = CAx(0) + a(1) \\ y(2) &= Cx(2) = CA^2x(0) + a(2) \\ &... \\ y(T-1) &= ... = CA^{T-1}x(0) + a(T-1) \end{aligned}$$

Then

$$\begin{pmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(T-1) \end{pmatrix} = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{T-1} \end{pmatrix} x(0) + \begin{pmatrix} a(0) \\ a(1) \\ a(2) \\ \vdots \\ a(T-1) \end{pmatrix}$$

where we have qT equations for $n + qT$ unknowns! That would indicate potentially **infinitely many solutions**.
Despite this observation if we add the hypotesis that the vector containing the attack is **sparse**, in some situations we could have a unique solution to this problem.

Definition 1 *We say that h errors are **correctable** after T steps if its possible to recover any $x(0)$ given $y(0), ..., y(T-1)$ under the condition $\|a\|_0 \leq h$, for each $k = 0, ..., T-1$.*

This corresponds to ask that the problem

$$\begin{pmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(T-1) \end{pmatrix} = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{T-1} \end{pmatrix} x(0) + \begin{pmatrix} a(0) \\ a(1) \\ a(2) \\ \vdots \\ a(T-1) \end{pmatrix} \quad \text{s.t.} \quad \|a\|_0 \leq h, k = 0, ..., T-1$$

had a **unique solution**.
Now we have to face with two problems:

1. Under **which conditions** can I solve the proposed problem?
2. **How can I solve** it?

2.3 Well-Position of the problem (Static case)

Let us analyze the problem in a very particular case, that is when the system we want to ovbserve is **static**. That is the matrix $A \in \mathbb{R}^{n,n}$ is the identity matrix $\mathbb{I}_n$. This corresponds to state that $x(k+1) = x(k) = x$ (remember that we have no input $u(t)$).
In this case the observability matrix has a very simple form, in particular it is equal to the matrix $C \in \mathbb{R}^{q,n}$. We have that the problem becomes:

$$y = Cx + a \quad \text{s.t.} \quad \|a\|_0 \leq h$$

It is useful to give these formal definitions:

Definition 2 (h-sparsity) *A vector is said to be **h-sparse**, if $\|a\|_0 = h$*

Definition 3 (Support) *Given $a \in \mathbb{R}^q$ we denote with $Supp(a) = \{i : a_i \neq 0\}$, the set of i such that the i-th component of the vector is a non-zero number.*

How the attack could be done in order to not to be 'detectable'? Let's consider an $a = Cw, \quad w \in \mathbb{R}^n, w \neq 0$ and assume it is that $\|Cw\|_0 \leq h$ (for the assumption we made). We sobstitute in the equation and obtain:

$$y = Cx + Cw = C(x + w) \quad \text{s.t.} \quad \|Cw\|_0 \leq h$$

Actually, if such w exixts, the **attack is feasible** and so not detectable. This property depends strongly on the property of the matrix C.
Fortunately we have a **proposition** that provide us with an equivalence to determine the **resilience** of the system to h attacks.

Proposition 1 (Correctability) *h errors are correctable (or equivalently **the system is resilient against h attacks**) after T steps if and only if for all $z \in \mathbb{R}^n$, $\|\mathcal{O}_T z\|_0 > 2h$.*

$$\textbf{\textit{System correctable}} \iff \forall z \in \mathbb{R}^n, \quad \|\mathcal{O}_T z\|_0 > 2h$$

In the static case the Proposition 1 becomes:

$$\textbf{System correctable} \iff \forall z \in \mathbb{R}^n, \quad \|Cz\|_0 > 2h$$

Proof 1 *As this is a characterization property we have to proof the proposition in the two direction $\Leftarrow$ and $\Rightarrow$. We give this proof for the static case, but the procedure for the more general case is analogue.*

- ***[Proof of $\Leftarrow$]*** *Assuming that $\forall z \in \mathbb{R}^n$ we have $\|Cz\|_0 > 2h$ we want to demonstrate that h errors are correctable, that is the problem has a unique solution. As in many situation we want to demonstrate the uniqueness of something, we can go on **by contraddiction**. In particular assume that $y = Cx + a$ s.t. $\|a\|_0 \leq h$ has got two different solutions:*

$$\begin{pmatrix} x' \\ a' \end{pmatrix} \text{ and } \begin{pmatrix} x'' \\ a'' \end{pmatrix}$$

*then we have that $y = Cx' + a' \quad s.t. \quad \|a'\|_0 \leq h$ but also $y = Cx'' + a'' \quad s.t. \quad \|a''\|_0 \leq h$. $Cx' + a' = Cx'' + a'' \iff C(x' - x'') = a'' - a'$ But due to the fact that a' and a'' are h-sparse we can obtain by substracting them a vector which is **at most** 2h-sparse. We have finished because we found a contradiction , that is, $\exists z \in \mathbb{R}^n, \|Cz\|_0 \leq 2h$. In this case $w = a' - a''$*

- ***[Proof of $\Rightarrow$]*** *Assuming that h errors are correctable, we want to demonstrate that $\forall z \in \mathbb{R}^n$ we have $\|Cz\|_0 > 2h$. Similarly the former case, we can demonstrate the property by contradiction assuming that $\exists w \in \mathbb{R}^n \quad \|Cw\|_0 \leq 2h$. We can write such Cw like a sum of two (at most) h-sparse vectors g_1 and $g_2 \rightarrow Cw = g1 + g2$. But we can also say that $Cw = g_1 + g_2 \longleftrightarrow Cw - g_1 = g_2 = C0 + g_2$. In this way we are saying that we have **two distinct solutions**, this is in contradiction with our hypotesis.* `QED`

The proposition that has just been proved is very powerful, but how are we able to try for all z that the statement is valid? In the great majority of the situations is easier to provide a **counter-example**.
Now we are going to do some example and then to provide (at least) a necessary condition for well-position of the problem of correctability.

2.3.1 Some examples

Example 0: a "naive" example

Consider that we have $C = \mathbb{I}_n$, $q = n$, $h = 1$, $n = 3$. In this case the output equation is:

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= x_2 \\ y_3 &= x_3 \end{aligned}$$

As we can see the system is **perfectly observable** because each measurement of y gives an element of the state, but suppose that we know that there is an attack ($h = 1$), but we don't know where. If one of the measurement is corrupted, there is no way to recover the state of the system $\Rightarrow$ C is not correcting h errors $\Rightarrow$ the system is not resilient to h attacks. In an intuitive way we can say that we could add more sensors to improve the situation, it is 'the path' that follows the next example.

Example 1: add more sensors (increase q)

At this point we change the (sensing) matrix C by adding new measurements, so

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

What has changed here is that we have **two more** sensors. We have a duplication of C_1 and C_2 (we indicate with C_i the i-th row of C). If $y_1 = C_1x = y_4 = C_4x$ and $y_2 = C_2x = y_5 = C_5x$ than we can state that the attack is on the sensor 3. But **what if either** $y_1 \neq y_4$ **or** $y_2 \neq y_5$? We have no way to know which sensor has been attacked. If I switched off this couple of devices I can't observe anymore one of the component of the state vector.
This was only an **intuitive way** to understand this fact. How can we formalize it? We can say that the couple (C_1, C_2) has a **non trivial kernel**. That is the equation:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} z = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

has a non zero solution. In particular a solution could be $z = \begin{pmatrix} 0 \\ 0 \\ \alpha \end{pmatrix}$ such z produces

$$Cz = \begin{pmatrix} 0 \\ 0 \\ \alpha \\ 0 \\ 0 \end{pmatrix} \quad \alpha \in \mathbb{R}$$

which due to the fact which is 1-sparse, is **in contradiction** with the proposition.
Conclusion: **even this matrix does not correct h=1 error $\Rightarrow$ the system is not resilient**!

Example 2: more mixed measurements

Freezing the other parameters, let us change again the matrix C in the following way:

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \end{pmatrix}$$

It is quite clear that there is a triple of rows of C which are linearly dependent, specifically $C_5 = C_4 - 2C_2 \Rightarrow$ the triple has a non trivial kernel. By doing simple algebraic steps we desume that a solution could be $z = \begin{pmatrix} \alpha \\ 0 \\ -\alpha \end{pmatrix}$ which generates a 2-sparse Cz we have again a contradiction! Conclusion: **C is not resilient**.

Example 3: linearly independent measurements

Now we provide the following C matrix:

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & -1 \end{pmatrix}$$

In this case all the triples are linearly independent, this cause the relative kernel to be trivial $\rightarrow$ there is no way to produce a counter-example. Conclusion: this C matrix corrects h=1 error.

2.3.2 A necessary condition for well-position

Let $C \in \mathbb{R}^{q,n}$, $rank(C) = n$ and $q > n$, it is quite clear that any subset Ω composed by $n-1$ rows has got a non trivial kernel. This implicates that the sparsity is not larger than $q-(n-1)$, that is $2h < q-(n-1)$ or equivalently $h \leq q-n$. From which I can recover the following inequality:

$$q \geq 2h + n$$

It is immediate to understand that if I want to correct $h = 1$ error with $n = 3$, we need at least of $q = 5$ sensors.

This fact bring us to state that:

- Large q is not sufficient for **resilience**
- On the other hand there is a **minimum q** which I need to correct a certain number h of errors.

2.4 Reformulation of $y = Cx + a \quad$ s.t. $\quad \|a\|_0 \leq h$

The second question to answer is:

How can we solve the problem $y = Cx + a \quad$ **s.t.** $\quad \|a\|_0 \leq h$**?**

The problem can be reformulated as follows:

$$\min_{x\in\mathbb{R}^n, a\in\mathbb{R}^q} \|a\|_0 \quad \text{s.t.} \quad y = Cx + a$$

It can be proved that **if the system is resilient to h attacks the solution to this problem corrects h errors**. Unfortunately, even this form of the problem has its drawbacks:

- As the problem is a combinatorial one, is **not feasible** (NP-Hard)
- The **objective function** which contains an ℓ_0 norm is **not convex, non continuous and non differentiable in 0**

The solution is going in the direction of **convex relaxation**:

1. In the objective function, we choose to relax $\|a\|_0$ with its **best CONVEX approximation** $\|a\|_1$; it is CONVEX, CONTINUOUS but NON DIFFERENTIABLE IN 0 (we can face this problem);
2. In the real world, we must take into account the noise that could appear in the equation $y = Cx + a$ and so it becomes $y \approx Cx + a$, this leads to the **Least Squares (LS) problem**

$$\min \frac{1}{2}\|y - Cx - a\|_2^2 = \min \frac{1}{2}\left\| y - \begin{pmatrix} C & I \end{pmatrix} \begin{pmatrix} x \\ a \end{pmatrix} \right\|$$

Combining this two approximations, due to the fact we want to minimize both the terms, the **resulting problem to solve** is:

$$\min_{x\in\mathbb{R}^n, a\in\mathbb{R}^q} \frac{1}{2}\left\| y - \begin{pmatrix} C & I \end{pmatrix} \begin{pmatrix} x \\ a \end{pmatrix} \right\| + \lambda\|a\|_1, \quad \lambda > 0$$

2.4.1 Why ℓ_1-regularization promotes sparsity?

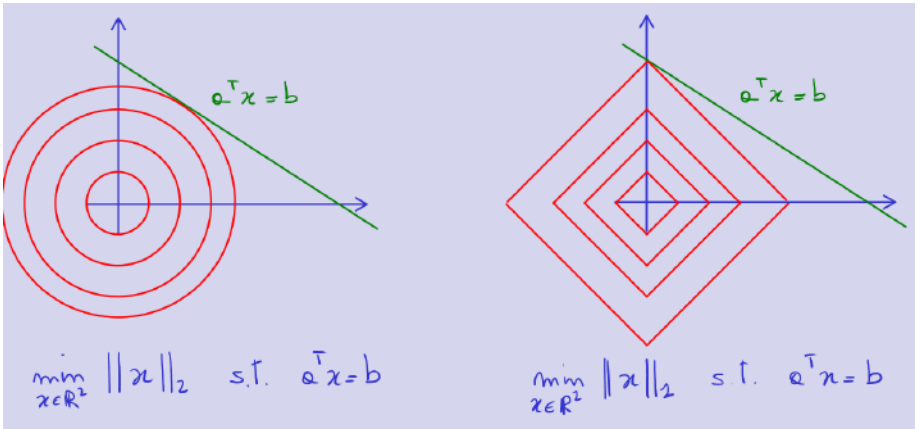


Figure 2.1: ℓ_p-norms and sparsification

In the approximation we have just made, we have considered the ℓ_1-norm. But why it is the best we could do? Let us consider to solve the problem

$$\min_{x\in\mathbb{R}^2}\|x\|_2 \quad \text{s.t.} \quad a^T x = b$$

The term $\|x\|_2 = k, k \in \mathbb{R}$ is a circle in the plane. We start from $k = 0$, and we increase it until we reach the line to satisfy the (linear) constraint. The solution we obtain is certainly not sparse because it is of the form $x^* = (x_1^*, x_2^*)$ both non-zero real number.
On the other hand if we consider the problem:

$$\min_{x\in\mathbb{R}^2}\|x\|_1 \quad \text{s.t.} \quad a^T x = b$$

we note that the solution we can find (by using the same method as before) is sparse. We have seen in an intuitive way that ℓ_2-norm takes the components of the solution close to zero in a "democratic way", in other words without 'reset' any component. At the opposite the ℓ_1-norm push to zero **only some components** making a selection among them $\longleftrightarrow$ this promotes **"sparsification"**.

[Note that we have just showed a trivial way to understand why we have chose one regularization instead of another. The topic would require a more accurate and rigorous explanation, which we don't care.]

2.4.2 Some observations

The problem has a connection with LASSO

The problem we found to solve after applying **relaxation** is very similar to the problem of **LASSO** (Least Absolute Shrinking and Selection Operator) in fact in its original formulation we have:

$$\min_{x\in\mathbb{R}^n}\frac{1}{2}\|Ax - y\|_2^2 + \lambda\|x\|_1$$

but this give an entire sparse solution. In our problem only a piece of the solution is sparse, whose linked to attacks a, so we can see our problem as a "**Partial LASSO**" because we have **no regularization** on the term x of the solution $\begin{pmatrix} x \\ a \end{pmatrix}$.

The problem has a connection with "Compressed Sensing"

The problem

$$\min_{x\in\mathbb{R}^n, a\in\mathbb{R}^q}\frac{1}{2}\left\| y - G\begin{pmatrix} x \\ a \end{pmatrix}\right\| + \lambda\|a\|_1, \quad \lambda > 0, \quad G = \begin{pmatrix} C & I \end{pmatrix}$$

as the G matrix has more columns than rows, is a **fat matrix** and without regularization the problem has infinitely many solutions. We can consider the y a **linear compressed measurement** vector. The problem of **recover a sparse vector from compressed linear measurement** has a strong connection with the COMPRESSED SENSING problem. Also here we have a **"Partial Compressed Sensing"** as only a part of the solution is sparse.

After a "long journey" we have understood that for the **secure state estimation of CPS under adversarial attacks** we have to solve a **Partial LASSO** problem. For this reason we seek a way to solve it $\Rightarrow$ Iterative Algorithms

2.5 IST: an algorithm for LASSO

Premise *In this course we will not see any black box technique, we will go through the direction of* ***iterative algorithms*** *to solve the LASSO problem.*
In the framework of ***Convex Optimization*** *one of the most used algorithms (also in Data Science/Machine Learning...) is the* ***Gradient Descent Algorithm****. But in the LASSO we have a ℓ_1-regularization, which is not differentiable in 0. We can't even avoid to consider it because we want to reach the (global)* ***minima****. We wonder if using an approximation could be the solution (eg.* ***subgradients****), however again this not produce a sparse solution.*

Once the proper promises have been done, we can introduce this algorithm for solving the **(sparse) optimization problem** of LASSO. The algorithm is called **Iterative Shrinkage/Thresholding (IST)** and it is a variant of the *Descent Gradient Algorithm*. After the definition and a general description, we are goingo to list the steps of the algorithm itself. Initially we consider the original LASSO problem

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}\|Ax-y\|_2^2 + \lambda\|x\|_1$$

in which we call $F(x) = \|Ax-y\|_2^2$ the Least-Squares functional. Moreover we know that $\nabla F(x) = A^T(Ax-y)$.

Definition (Shrinkage/Thresholding operator) The **Shrinkage/Thresholding operator** $\mathbb{S}_\alpha$ is a component wise operator $\mathbb{S}_\alpha : \mathbb{R}^p \to \mathbb{R}^p, \alpha > 0$. For any $x_i \in \mathbb{R}$:

$$\mathbb{S}_\alpha(x_i) = \begin{cases} x_i - \alpha & \text{if } x_i > \alpha \\ x_i + \alpha & \text{if } x_i < \text{-}\alpha \\ 0 & \text{if } |x_i| \leq \alpha \end{cases}$$

IST Algorithm

1. Initialization: $x_0 \in \mathbb{R}^p$, e.g. $x_0 = 0$
2. For $k = 0, ..., T_{max}$

$$x(k+1) = \mathbb{S}_{\lambda\tau}\left[x(k) - \tau\nabla F(x(k))\right]$$

The parameter τ has to be "small enough" so that the algorithm works as desidered. It has been demonstrated that the algorithm converge to the minimum of the LASSO functional $\frac{1}{2}\|Ax-y\|_2^2 + \lambda\|x\|_1$. The **iterative nature** and simplicity of this method, makes it adaptable also for: **dynamic** and **distributed** systems.

2.5.1 Derivation of IST Algorithm

Let us do a quick RECAP of the latest notions we introduced...
Generally speaking, when we have some measurements $y = A\tilde{x} + \eta$ where $\tilde{x}$ is the **true vector** to recover, we are able to find an **estimate** of it by solving the problem

$$x^* = \arg\min_{x\in\mathbb{R}^n} \frac{1}{2}\|y-Ax\|_2^2$$

also known as the **Least Squares (LS)** problem, where the obtained x^* is the estimate of the true vector. How can we solve this problem?
The **Gradient Descent** method is one of the most used in this field to retrieve a solution for LS problem. This algorithm step by step proceeds in the direction of the gradient, moving from the previous point of a **step size** called τ (small enough). Then we have

$$x(k+1) = x(k) - \tau \nabla F(x), \quad F(x) = \|y - Ax\|_2^2 \quad \text{(LS functional)} \tag{2.1}$$

for $k = 0, 1, 2, ...$
We can observe that the expression (2.1) is a Discrete Time Linear Time Invariant System (DT LTI), therefore we can study its properties by using the known results from the Theory.

Differently from the case which has just mentioned, in the LASSO problem we have the ℓ_1-norm too, because we are interested in finding a **sparse solution** (in our framework of **CPS under adversarial attacks** this is an important requirement). It is good to remind that the LASSO is a **convex, non-differentiable** problem.
Then, how can we solve it? Its solution can be retrieved by using any convex optimization solver (`cvx,lasso, ...`). On the other hand, the *Gradient Descent* algorithm is not effective! (Gradient of a non differentiable functional?? We can't!). Our **approach** is using the **ISTA** Algorithm of which it is given a **possible interpretation**.

Let the functional $F(x) = \frac{1}{2}\|y - Ax\|_2^2 + \lambda\|x\|_1$ be the LASSO functional. We define now a **surrogate functional** which will guide our *derivation of ISTA* as

$$\mathcal{R}(x, b) = F(x) + \frac{1}{2\tau}\|x - b\|_2^2 - \frac{1}{2}\|Ax - Ab\|_2^2, \quad \tau > 0 \tag{2.2}$$

where x is my 'standard' variable, while b is an auxiliary variable. By using a little bit of linear algebra we note that:

$$\frac{1}{2\tau}\|x - b\|_2^2 - \frac{1}{2}\|Ax - Ab\|_2^2 = \frac{1}{2\tau}\|x - b\|_2^2 - \frac{1}{2}\|A\|_2^2\|x - b\|_2^2 = \frac{1}{2}\left(\frac{1}{\tau} - \|A\|_2^2\right)\|x - b\|_2^2$$

and this quantity is non-negative if and only if $\frac{1}{\tau} > \|A\|_2^2 \Rightarrow \tau < \|A\|_2^{-2}$. This provides a guide to choose the τ and ensures that the **surrogate part** of the functional (2.2) is never negative, moreover it can immediately be noted that it is null if and only if $x = b \Rightarrow F(x) = R(x, x)$. That is the global minimum of $F(x)$ is the same of the global minimum of $R(x, x) \Rightarrow F(x^*) = R(x^*, x^*)$. At this point: our aim is to develop a **descent algorithm** for $F(x)$ which we will see that it is eased by $R(x, b)$.

It is not trivial to find a closed form for the solution of minimizing the surrogate functional considering both variables x, b together. Therefore, we move in the direction of an **alternating minimization** (alternating $\Rightarrow$ in turn). This is both **feasible** and **leads to the minimum** of $F(x)$. More clearly:

$$\begin{cases} \min_{b \in \mathbb{R}^n} \mathcal{R}(x, b), & \text{keeping } x \text{ as a constant} \\ \min_{x \in \mathbb{R}^n} \mathcal{R}(x, b), & \text{keeping } b \text{ as a constant} \end{cases}$$

First step: minimizing with respect to b

It is more trivial because we have seen recently that the *surrogate part* depends on b, and it is minimum if $x = b$, then

$$x = \arg\min_{b \in \mathbb{R}^n} \mathcal{R}(x, b)$$

Second step: minimizing with respect to x

The problem here is find $\arg\min_{x\in\mathbb{R}^n} \mathcal{R}(x,b)$. In this case we have to expand the functional by expressing the squared norm explicitly, to reach a conclusion.

$$\begin{aligned}
\arg\min_{x\in\mathbb{R}^n} \quad & \mathcal{R}(x,b) = \arg\min_{x\in\mathbb{R}^n} \quad \frac{1}{2}\|Ax\|_2^2 + \frac{1}{2}\|y\|_2^2 - yA^Tx + \lambda\|x\|_1 + && (2.3)\\
& \frac{1}{2\tau}\|x\|_2^2 + \frac{1}{2\tau}\|b\|_2^2 - \frac{1}{\tau}b^Tx + && (2.4)\\
& -\frac{1}{2}\|Ax\|_2^2 - \frac{1}{2}\|Ax\|_2^2 - b^TA^TAx = && (2.5)\\
\arg\min_{x\in\mathbb{R}^n} \quad & \lambda\|x\|_1 + \frac{1}{2\tau}\|x\|_2^2 - x^T\left(\frac{1}{\tau}b + A^Ty - A^TAb\right) = && (2.6)\\
& \tau\lambda\|x\|_1 + \frac{1}{2}\|x\|_2^2 - x^T[b + \tau A^T(y - Ab)] + \frac{1}{2}\|b + \tau A^T(y - Ab)\|_2^2 = && (2.7)\\
\arg\min_{x\in\mathbb{R}^n} \quad & \tau\lambda\|x\|_1 + \frac{1}{2}\|x - q\|_2^2 = \arg\min_{x\in\mathbb{R}^n} \quad \sum_{i=1}^{n}\left(\tau\lambda|x_i| + \frac{1}{2}(x_i - q_i)^2\right) && (2.8)
\end{aligned}$$

In (2.7), by multiplying all terms in (2.6) by τ and by adding a constant term $q = b + \tau A^T(y - Ab)$ nothing change. Moreover the final step (2.8) is separable, in the sense that If I want minimize the sum, I can proceed **component wise**. Therefore

$$\arg\min_{x\in\mathbb{R}^n} \quad \sum_{i=1}^{n}\left(\tau\lambda|x_i| + \frac{1}{2}(x_i - q_i)^2\right) \Longleftrightarrow \arg\min_{x_i\in\mathbb{R}} \quad \tau\lambda|x_i| + \frac{1}{2}(x_i - q_i)^2 = (...) = \tag{2.9}$$

$$= \begin{cases} q_i - \tau\lambda & \text{if} \quad q_i > \tau\lambda \\ q_i + \tau\lambda & \text{if} \quad q_i < -\tau\lambda \\ 0 & \text{if} \quad q_i \in [-\tau\lambda, \tau\lambda] \end{cases} \tag{2.10}$$

but this is the operator we formerly introduced as $\mathbb{S}_{\tau\lambda}(q)$ which pushes the component of the vector q close to zero. We can reach a **conclusion**.
Given $x_0 = 0$ for any $k = 0, 1, ...$

1. $b(k+1) = \arg\min_{b\in\mathbb{R}^n} \mathcal{R}(x(k), b) = x(k)$
2. $x(k{+}1) = \arg\min_{x\in\mathbb{R}^n} \mathcal{R}(x, b(k{+}1)) = \arg\min_{x\in\mathbb{R}^n} \mathcal{R}(x, x(k)) = \mathbb{S}_{\tau\lambda}[x(k) + \tau A^T(y - Ax(k))]$

Then,

$$x(k+1) = \mathbb{S}_{\tau\lambda}[x(k) + \tau A^T(y - Ax(k))] \tag{2.11}$$

which is our **Iterative Shrinkage/Thresholding algorithm (ISTA)**. Then, at the end of this discussion we have seen that the algorithm on which we focus comes from an **alternating minimization** of the surrogate functional $\mathcal{R}(x, b)$.

Theorem (1) ISTA is a **descent algorithm** for LASSO, then $F(x(k+1)) \leq F(x(k))$

Proof The proof of the theorem above is very simple:

$$\begin{aligned}
F(x(k)) = & \mathcal{R}(x(k), x(k)) \geq && \text{(minimizing over x)}\\
& \mathcal{R}(x(k+1), x(k)) \geq && \text{(minimizing over b)}\\
& \mathcal{R}(x(k+1), x(k+1)) = F(x(k+1), x(k+1))
\end{aligned}$$

Theorem (2) ISTA converges to the minimum of LASSO.

2.5.2 Curiosity

The equation (2.11) combines a linear part (the argument of the operator $\mathbb{S}$) and a **non linear part** that is the operator itself, being defined in a piece-wise fashion. The resulting system is a **Discrete Time Non Linear Time Invariant system** which is not trivial to treat.
This reminds a little the structure of a **neural network** where we have a **linear part** and a **non linear part** constituted by the **activating function**. For this reason the **evolution of ISTA** can be seen as a **Neural Network** application.

2.5.3 Application of ISTA on CPS framework

We have seen recently that the problem of Cyber-Physical system under attacks can be formulated as follows:

$$\min_{x\in\mathbb{R}^n, a\in\mathbb{R}^q} \frac{1}{2}\left\| y - G\begin{pmatrix} x \\ a \end{pmatrix} \right\| + \lambda\|a\|_1 \tag{2.12}$$

that is a **partial LASSO** because only a part of the found solution is sparse, whose related to the (sparse) attacks. What about ISTA for Partial LASSO? I have no limitations about the choice of the coefficient λ, in particular I can get $\lambda = 0$ for the elements of the solution which are not interested by sparsity as it is the state of the system. Then,

$$\lambda\|a\|_1 = \lambda\left\| \begin{pmatrix} x \\ a \end{pmatrix} \right\|_1 = 0|x_1| + ... + 0|x_n| + \lambda|a_1| + \lambda|a_q| \tag{2.13}$$

This changes nothing, but it is essential that we made this variation in a way that the derivated descent algorithm could fit the "Partial LASSO" problem used for CPS purposes.

Chapter 3

Localization by RSS fingerprinting

3.1 Introduction

Localization is the problem related to the **estimation** of the position of a **target**. Other problems are those related to the **detection** (whether a target is present or not) and **tracking** that is we track the position of a **moving target**.
Nowadays the focus is in particular on **Indoor Localization** whose mathematical modeling is quite challenging due to the presence of *multipath* and *reflecting surfaces*. For these reasons there is not an available **unified approach**.
GPS is not usable for indoor localization, the so called **WSN (Wirelless Sensor Networks)** are used for this purpose. A WSN is essentially a network of devices equipped with sensors. We can find two main approaches:

1. **Triangulation and Trilateration**: in this case we assume that the target is a **transmitting device** that broadcasts a signal. The former method deploy the **direction of arrival** of such signal, the latter use instead the **distance** from the source's signal (*id est* the target). For this approach, in the case that the measurement are exact, 3 non-aligned sensors are sufficient, alternatively the higher number of sensors, the higher precision reached.

2. **Fingerprinting methods** refers to techniques that match the **fingerprint** of some characteristics of a signal which is **location-dependent**. For our purposes we consider the **Received Signal Strength** or **RSS**.

3.2 RSS-fingerprinting: general description

In general for all **Fingerprinting method**, we identify **two phases**:

1. **Training phase** in which we collect the fingerprints of a scene;

2. **Runtime phase** in which we match the online measurement with the closest a-priori location fingerprints.

As we said, we focus our attention on *RSS-fingerprinting*.

3.2.1 (Phase 0) Initialization

Given the room where the localization has to be done, we deploy the WSN in some way (For example: grid deployment, or random (uniform) deployment, split the room into p cells. **Localize the target** $\longrightarrow$ detect in which cell the target is placed.

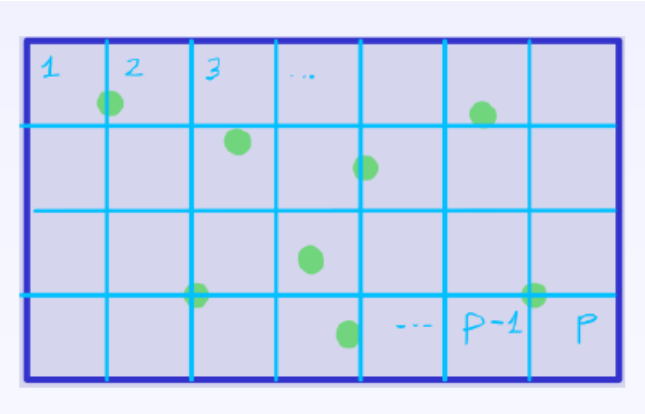


Figure 3.1: (Phase 0) - Initialization

3.2.2 (Phase 1) Training phase

We put the target in each cell. According to the fact that the target itself broadcasts a signal, **each sensor** measures and stores the RSS, and create a **signature map** or **dictionary**. More specifically, the dictionary can be represented by a matrix D, in which each entry $D_{i,j}$ denotes the RSS-measurement the sensor i takes when the target is in the j-th cell.
Each sensor builds its own dictionary, and the WSN builds an **overall dictionary** $D \in \mathbb{R}^{q,p}$. This phase takes some time and requires that the nodes of the network saved the information, it makes the runtime phase more accurate.

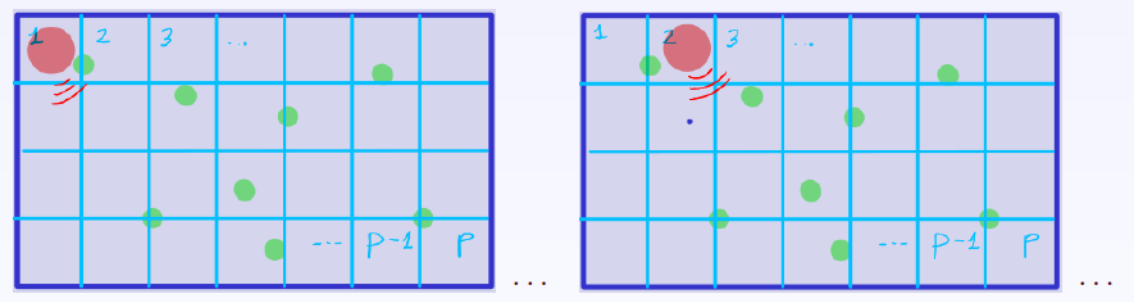


Figure 3.2: (Phase 1) - Training phase

3.2.3 (Phase 2): Runtime phase

Is the phase in which the localization is performed. Each sensor takes a measurement y_i, the j-th column of the dictionary indicates the j-th cell of the room in which the target is located! If there is one target, then one of the columns of the dictionary would correspond to the measurement (in theory), then the target is located in the j-th cell where j is such that $y = D_j$, with D_j the $j - th$ column of the dictionary but usually, we can have *multiple target* in the same room moreover it's very difficult that $y = D_j$ due to the presence of noise on sensors.

Then, in this scenario we have that the vector of measurements y is a sum of a subset of the columns of the matrix D, to which we must add an additional term related to the noise. We have:

$$y = Dx + \text{noise} \tag{3.1}$$

Where $x \in \{0, 1\}^p$ and $x_i = 1$ when in the i-th cell there is a target.
A further observation which can be done is that, in the real problems of *indoor localization*

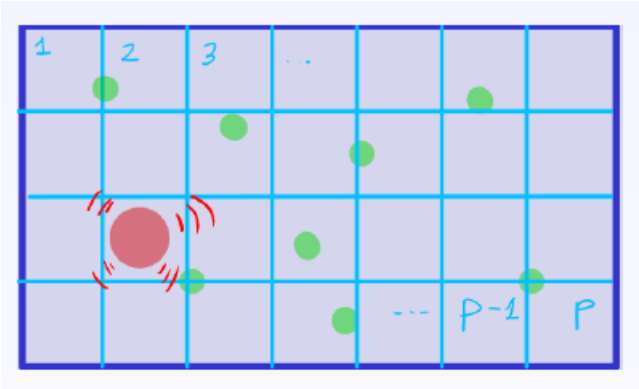


Figure 3.3: (Phase 2) - Runtime phase

the number of the targets one would like to localize is **much smaller** than the number of the cells, this leads to the sparsity of the solution $x \in \{0,1\}^p$. Summarizing: (i) We want to find a solution to the problem (3.1), (ii) We know that such solution is **sparse**. Then,

$$x^* = \arg \min_{x \in \{0,1\}^p} \frac{1}{2}\|y - Dx\|_2^2 + \lambda \|x\|_1 \tag{3.2}$$

This type of formulation leads to a **mixed integer** combinatorial problem, that again is NP-hard, so non-tractable. We can relax the constraint $x \in \{0,1\}^p$ into $x \in \mathbb{R}^p$, obtaining:

$$x^* = \arg \min_{x \in \mathbb{R}^p} \frac{1}{2}\|y - Dx\|_2^2 + \lambda \|x\|_1 \tag{3.3}$$

This is the problem of LASSO in the original formulation. It is important to remember that: the solution of the problem (3.3) is not the original x due to: (i) the presence of the noise, (ii) a bias introduced by the ℓ_1-regularization term which in part promotes sparsity on the other hand inserts an error. One can use of course the **IST Algorithm** to find a solution.

3.3 Localization under sparse sensor attacks

Assume that, in a realistic way, the training phase is *attack-free*, therefore the matrix D is attack free. Let us focus the attention again in the runtime phase.
What if the sensors were under adversarial attacks? One can apply the theory we developed in the former chapter! Then, we assume that the sensors under attacks are much smaller than the total number q of them. In this context the equation (3.1) becomes:

$$y = Dx + a + noise \tag{3.4}$$

Then, the formulation of problem changes as follows:

$$x^* = \arg \min_{x \in \mathbb{R}^p} \frac{1}{2}\|y - Dx - a\|_2^2 + \lambda_1 \|x\|_1 + \lambda_2 \|a\|_1 \tag{3.5}$$

where we are using different weights λ_1, λ_2 to give more or less importance to the term related to the solution x and to the attack a. According to this novel aspect, the risen problem is a **weighted LASSO**. **IST algorithm** continues to work, as we saw in the case of presence of *non-sparse* part of the solution in the case of SSE under adversarial attacks.

3.4 Other approaches to Localization

The approach we have just seen is not the only one. Next paragraphs are in order to give some alternatives which differ from the first we have presented in computational complexity, time of convergence and so on.

3.4.1 k-Nearest Neighbour (K-NN)

Assume to know that there is only **one target**, given the vector $y \in \mathbb{R}^q$, we could find the $j-th$ column of the dictionary D that is the **nearest** with respect to y. The localization problem in this specific scenario becomes:

$$\hat{j} = \arg \min_{j=1,\dots,p} \|D_j - y\|_2^2 \tag{3.6}$$

where D_j is the j-th column of D. Note that in the problem there is not the factor $\frac{1}{2}$ but from the moment we have a minimization problem nothing changes.
RSS is additive so we can localize more than one target in the *runtime phase*. Suppose that the targets are $k = 2$, the vector y can be seen as a sum of the columns, so the problem (3.6) gets transformed in:

$$(\hat{j_1}, \hat{j_2}) = \arg \min_{(j_1,j_2)=1,\dots,p} \|D_{j_1} + D_{j_2} - y\|_2^2 \tag{3.7}$$

In general, we have to check a number of configurations that is equal to $\binom{p}{k} \rightarrow$ **NP-Hard**. Note that: this approach can be used if we have small p, k otherwise other techniques ought to be used in order to promote efficiency.

3.4.2 Linear Regression (noise-free case)

If we have multiple targets, in absence of noise the localization problem could be formulated as a **binary linear regression**. We have to solve in x the equation $y = Dx$ and add the constraints about the x domain and sparsity. Then, it is obtained:

$$\begin{aligned} & Dx = y \\ & \text{s.t. } x \in \{0,1\}^p, \ \sum_{j=1}^{p} x_j = k \end{aligned} \tag{3.8}$$

Even in this case it has been risen a mixed-integer combinatorial problem $\rightarrow$ NP-Hard. The choice should be taken in an accurate way according to the dimension of the problem.

3.5 Some comments on the setting

These algorithms assume that there is a **Fusion Center** where data from the sensors are collected. In this way: the Fusion Center stores the whole dictionary D and the runtime measurements, so that it can run the localization algorithm which is nothing but one of the exposed methods.

Chapter 4

Dynamic Secure State Estimation of CPSs under adversarial attacks

It has been explained in the previous chapter that for a system

$$\begin{aligned} x(k+1) &= Ax(k) \\ y(k) &= Cx(k) \end{aligned} \tag{4.1}$$

if one collects n measurements $y(i), i = 1, ..., n$, we can recover the state $x(k)$ at each k, if we are able to find $x(0)$ and then invert the equation

$$y = \mathcal{O}_n x(0)$$

the Theorem by Kalman, states that this is possible, that is the system is **observable**, if and only if $\text{rank}(\mathcal{O}_n) = n$. This is the static-batch approach to the **state estimation problem**, on the other hand - as an alternative technique - if the system is observable we can estimate $x(k)$(then $x(0)$) dynamically by constructing a device called the **Observer**, in the deterministic case it is called **Luemberger Observer**.

4.1 Review of Luemberger Observer

A copy of the system (4.1) is made with the only difference of adding a correction term. Starting from this point we use $\hat{x}(k)$ to indicate the **estimate of the state at the time** k (discretized time), and $\hat{y}(k)$ is the output computed by using the estimate. The Luemberger Observer has the following equations:

$$\begin{aligned} \hat{x}(k+1) &= A\hat{x}(k) - L[\hat{y}(k) - y(k)] \\ \hat{y}(k) &= C\hat{x}(k) \end{aligned} \tag{4.2}$$

The quantity $e(k) = \hat{x}(k) - x(k)$ is the error of the estimate at time k, the aim is to design an online algorithm which could make $e(k) \to 0$.
In order to understand the role of the matrix $L \in \mathbb{R}^{n,q}$, called the **observer gain matrix**, we can write:

$$\begin{aligned} e(k+1) = \hat{x}(k) - x(k) &= A\hat{x}(k) - L[\hat{y}(k) - y(k)] - Ax(k) = && (4.3) \\ &= A\hat{x}(k) - LC\hat{x}(k) - LCx(k) - Ax(k) = && (4.4) \\ &= A[\hat{x}(k) - x(k)] - LC[\hat{x}(k) - x(k)] = && (4.5) \\ &= (A - LC)[\hat{x}(k) - x(k)] = && (4.6) \\ &= (A - LC)\ e(k) && (4.7) \end{aligned}$$

The resulting equation $e(k+1) = (A - LC)\, e(k)$ describes an LTI discrete time dynamical system in which the state matrix is represented by $A - LC$. From the theory of dynamical systems, we know that the system is asymptotically (internally) stable if after a certain time k, it is verified that $e(k) \to 0$, the result we are seeking, it is verified when the *eigenvalues* of $A - LC$ are in the unitary circle.

Regarding the matrix A it is not very interesting to track $x(k)$ if it is asymptotically stable because in this situation $\lim_{k\to\infty} x(k) = 0$. So A is required to be stable but not asymptotically (some authors refers to this type of stability as **marginal stability**).

Theorem If the system is observable, then there exists L such that $A-LC$ is asymptotically stable. (We can drive the error to zero in order to track the state of the system).

4.2 State estimation by Least-squares approach

At a certain time k, given the current measurement $y(k) = Cx(k)$, we might estimate $x(k)$ by solving the following problem:

$$\hat{x}(k) = \arg\min_{x\in\mathbb{R}^n} \frac{1}{2}\|y(k) - Cx\|_2^2 \tag{4.8}$$

if $q > n$ and C is full rank. We call $\mathcal{F}(x) = \frac{1}{2}\|y(k) - Cx\|_2^2$ the Least-Squares functional. There is a problem: the (pseudo)inversion of the matrix C, could be non-trivial for a medium-large dimensional problem. Even the classical Gradient Descent Algorithm would be too slow!
A solution is: at each k, we run a **single step** of gradient descent resulting in an **Online gradient descent**.

Online Gradient Descent (OGD)
Given the measurement $y(k) = Cx(k)$ and $\hat{x}(k)$ computed before time k

$$\hat{x}^+(k) = \hat{x}(k) - \tau\nabla\mathcal{F}(\hat{x}(k)) = \hat{x}(k) - \tau C^T[C\hat{x}(k) - y(k)] \tag{4.9}$$
$$= \hat{x}(k) - \tau C^T\,[\hat{y}(k) - y(k)] \leftarrow \text{estimate of } x(k) \tag{4.10}$$
$$\hat{x}(k+1) = A\hat{x}^+(k) \leftarrow \text{prediction of } x(k+1) \tag{4.11}$$
$$\hat{y}(k) = C\hat{x}(k) \tag{4.12}$$

By merging estimate and prediction we obtain:

$$\hat{x}(k+1) = A\hat{x}(k) - \tau AC^T[\hat{y}(k) - y(k)] \tag{4.13}$$

It can be noted that there is a certain similarity of the system (4.13) and the (4.2), in particular the OGD is a Lumberger Observer with $L_g = \tau AC^T$.
Since we have fixed, in a certain way, the matrix L_g, one would wonder when

$$A - L_gC \tag{4.14}$$

is asymptotically stable. We can rewrite it as $A(I - \tau C^TC)$. If we choose $\tau < \frac{1}{\|C\|_2^2}$ then we obtain that $\|I - \tau C^TC\| \leq 1$. Finally, two cases are to be considered:

- If $\|I - \tau C^TC\| = 1$, and A is *marginally stable*, then $A - L_gC$ is marginally stable;
- If $\|I - \tau C^TC\| < 1$, and A is *marginally stable*, then $A - L_gC$ is **asymptotically stable**.

Until this moment, we have presented these results for an LTI DT dynamical system in which there are not attacks. What about **Dynamic Secure State Estimation**?

4.3 Dynamic SSE with constant attack

Recalling that a CPS under adversarial attacks on the sensors can be described by the system:

$$\begin{aligned} x(k+1) &= Ax(k) \\ y(k) &= Cx(k) + a(k) \end{aligned} \tag{4.15}$$

We have seen in the Second Chapter that in this case the problem of observability results in:

$$\begin{pmatrix} y(0) \\ \vdots \\ y(T-1) \end{pmatrix} = \mathcal{O}_T x(0) + \begin{pmatrix} a(0) \\ \vdots \\ a(T-1) \end{pmatrix} \tag{4.16}$$

We have solved this problem for the static case in which we have seen the IST Algorithm, but in the case in which A is not the identity matrix, the problem is not trivial to solve!
If an assumption is done on the 'shape' of the attacks the problem could be well posed, in particular we should assume that the attacks are constant and equal to a vector $a \in \mathbb{R}^q$, at this point the problem (4.16) results in:

$$\begin{pmatrix} y(0) \\ \vdots \\ y(T-1) \end{pmatrix} = \mathcal{O}_n x(0) + \begin{pmatrix} a \\ \vdots \\ a \end{pmatrix} = \begin{pmatrix} C & I \\ CA & I \\ \vdots & \vdots \\ CA^{T-1} & I \end{pmatrix} \begin{pmatrix} x(0) \\ a \end{pmatrix} \tag{4.17}$$

where the matrix

$$\mathcal{O}'_T = \begin{pmatrix} C & I \\ CA & I \\ \vdots & \vdots \\ CA^{T-1} & I \end{pmatrix} \tag{4.18}$$

is an **augmented observability matrix**. Is this CPS observable?

In order to clarify this aspect, let us consider a couple of measurements for $k = 0, 1$:

$$y(0) = Cx(0) + a \tag{4.19}$$
$$y(1) = Cx(1) + a = CAx(0) + a \tag{4.20}$$

We might manipulate algebraically these equations in order to eliminate the attack a which is assumed to be constant. For example, one can subtract the (4.20) from the (4.19), and it will be obtained

$$y(1) - y(0) = CAx(0) + a - Cx(0) - a = \tag{4.21}$$
$$[CA - C]x(0) = \tag{4.22}$$
$$C[A - I]x(0) \tag{4.23}$$

Moreover, let us suppose that $q = n$ so that the matrix $C[A-I] \in \mathbb{R}^{n,n}$. If such matrix would be invertible, we could recover $x(0)$ without problem by inverting the equation (4.23). One might think that I could go further in the computation of $y(k)$ and by using such manipulations, I can eliminate the attack and recover without problems the state.
BUT in many situations the square matrix $A - I$ is not invertible, we have seen that is reasonable that in the Spec(A) (set of the eigenvalues of A) there is an eigenvalue $\lambda_i = 1$ for some i. This imply that the matrix $A - I$ has a **null eigenvalue**, that is the same to confirm

that the matrix is **not full rank** and for this reason **non invertible**. One wonder if we might have a generalization of this concept. It is possible by analysing the **kernel of** $\mathcal{O}'_T$. To this aim, again, we will exploit some algebraic tricks. This time we subtract couple of rows of the matrix (4.18) from the bottom to the top, obtaining:

$$\begin{pmatrix} C & I \\ C(A-I) & I \\ \vdots & \vdots \\ CA^{T-3}(A-I) & 0 \\ CA^{T-2}(A-I) & 0 \end{pmatrix} \begin{pmatrix} x \\ a \end{pmatrix} \tag{4.24}$$

This is nothing but the system (4.16) rewritten in a different form. Let us neglect at the moment the first row of the rewritten matrix. It is recognizable the matrix $\mathcal{O}_{T-1}$, however due to the fact of being multiplied by $A - I$ the linear system

$$\mathcal{O}_{T-1}(A-I)x = 0$$

is underdetermined and has got infinitely many solutions. Despite $\mathcal{O}_{T-1}$ is full rank (it is a minimum requirement because if the system without attack is not observable, I cannot recover the state with attacks!) we do not have a trivial kernel because of $A - I$ which is not full rank. Moreover if we add the first equation we obtain the total system

$$\begin{cases} (A-I)x = 0 \\ Cx + a = 0 \end{cases}$$

We are ready to give the following proposition:

Proposition If the matrix A has an eigenvalue $= 1$, the dynamic CPS with constant attack is not observable.

The fact that an eigenvalue of A might be equal to one, is quite common from the moment we do not desire the situation in which the state tends to zero when $k \to \infty$.
Then, the proposition states that in general a CPS under attacks **is not observable** even if the attack is **constant**. However, we have not exploited yet the information about the **sparsity of the attacks** which allows us to develop a so-called **SPARSE OBSERVER**.
Before giving the final result is useful to give a little of notation:

$$z(k) = \begin{pmatrix} x(k) \\ a(k) \end{pmatrix} \quad \hat{z}(k) = \begin{pmatrix} \hat{x}(k) \\ \hat{a}(k) \end{pmatrix} \quad \hat{z}^+(k) = \begin{pmatrix} \hat{x}^+(k) \\ \hat{a}^+(k) \end{pmatrix} \quad G = \begin{pmatrix} C & I \end{pmatrix}$$

We want to solve the problem of recover the state of the dynamic CPS under constant attacks that is to solve:

$$\min_{x \in \mathbb{R}^n, a \in \mathbb{R}^q} \frac{1}{2} \|y(k) - Gz(k)\|_2^2 + \lambda \|a\|_1$$

It can be demonstrated that after a sufficient number T of steps the solution is given by the following algorithm:

SPARSE OBSERVER

Given $y(k) = Gz(k)$ and $\hat{z}(k)$,

$$\begin{aligned}
\hat{z}^+(k) &= \hat{z}(k) - \tau G^T[G\hat{z}(k) - y(k)] && \leftarrow \text{estimate of } z(k) \\
\hat{x}(k+1) &= A\hat{x}^+(k) && \leftarrow \text{prediction of} x(k+1) \\
\hat{a}^(k+1) &= \mathbb{S}_{\tau\lambda}[\hat{a}^+(k)] && \leftarrow \text{"sparsify" the attacks} \\
\hat{y}(k) &= G\hat{z}(k)
\end{aligned}$$

What are the differences from the previous version? The matrix A is not the identity matrix, the state of the CPS **changes** $\Rightarrow$ in general $x(k) \neq x(k+1)$.

Chapter 5

Distributed Consensus based algorithms for CPSs

We have seen since the first chapter of these notes that **Cyber-physical systems** are sets of interconnected devices which take measurements from the physical world through the use of sensors. Such framework is **intrinsically distributed** even in both measurement and processing stages.

5.1 Toward the Fusion Center removal

In the first part of this course we have seen that there was a **fusion center** which collected and processedS all the data (it can be for example a computer connected via Wi-Fi), another way to perform processing in the CPSs framework is the **distributed way**, then by using **distributed** (or **decentralized) algorithms**. The figure below shows the differences between a centralized and a distributed setting and respectively:

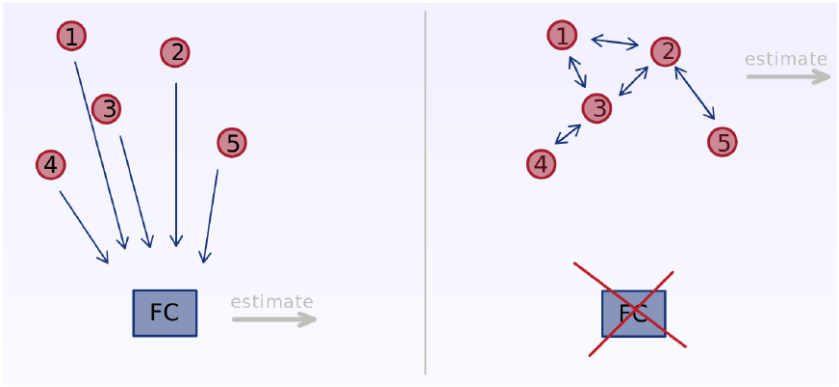


Figure 5.1: Centralized and Distributed processing

- In the first case all the data are conveyed to the fusion center to which is assigned the work of collecting and processing it; by using centralized algorithms, some estimates can be given;
- In the second case of a **distributed** approach, the agents (or nodes) cooperate by **exchanging information** in order to perform the elaboration.

In this last chapter of the 'Modeling' part, we are going toward the **removal of the Fusion Center**, in order to understand the methods can be used for *decentralized processing*.

5.1.1 Motivations

The motivation that guide us to the decentralization, is quite practical: a large number of **cheap** interconnected devices which cooperate is better than few **expensive** devices. The adjectives **cheap/expensive** have to be declined in term of: computational capability, accuracy, power consumption and so on. Decentralized systems are: (i) **more robust** to failures, (ii) cheaper to maintain. However, some **distributed algorithms** are needed. In the figure below are presented further information about advantages and disadvantages between the two approaches.

Centralized processing	Distributed processing
+ usually, the FC is a powerful computer	- devices are scarcely powerful (e.g., microprocessors)
+ global information is available for processing	- devices are locally interconnected (i.e., connected with neighbors), then only local information is available
+ centralized algorithms are "standard"	- distributed algorithms are less "standard" and usually sub-optimal with respect to the centralized algorithms
- a failure in the FC stops the overall processing	+ a failure of a single device would not prevent the overall processing
- long-range communication to transmit data to FC	+ short range communication
- no privacy	+ privacy: each device can keep some information private

Figure 5.2: Pros and cons

It is quite common that in this field one talk about **multi-agent systems**. They are in a nutshell collections of devices that communicate to **achieve a common goal** (we will see that this concept is strongly connected to the **consensus**). An **agent/node** is a device that communicate with other agents and takes autonomous decisions.

5.2 Distributed Least Squares (I)

As we understood is the problem of estimate the state $\tilde{x} \in \mathbb{R}^n$ given linear measurements of the type $y = C\tilde{x} + \eta$, where η is a generic noise of measurement. We can use a Least-Square approach:

$$\min_{x \in \mathbb{R}^n} \|y - Cx\|_2^2 \tag{5.1}$$

The minimizer x^* gives us the estimate of $\tilde{x}$. Since we have removed the Fusion Center(FC) we cannot run any algorithm (eg. Gradient Descent) to compute the estimate. In the scenario in which the agents are **sensors**, for example each sensor has its own C_i (the $i-th$ row of the matrix C) and its own (distributed) measurement. It is remarkable that such quantities are not shared due to issues related to **privacy** and **storage capacity**! However another 'smaller' information can be exploited: the local estimate $x^{(i)} \in \mathbb{R}^n$ of the state. On the other hand if each sensor node i had solved the LS problem, the resulting estimated would have been very inaccurate. **Solution** $\Leftarrow$ we need **cooperation** among sensor nodes.

Math tool: Graphs

At this point, our next aim is to find a **model** to analyze the cooperation among agents. The best way is using a very powerful tool from the discrete Maths: Graphs. Essential definitions and features are given in this paragraph in order to continue the discussion.

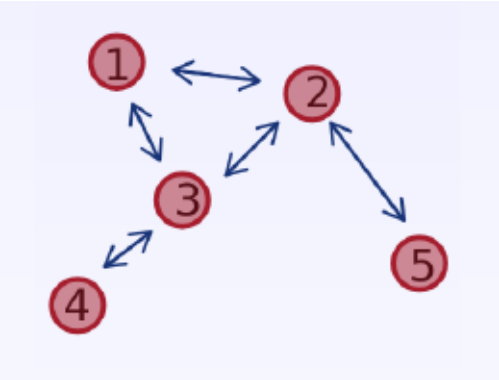


Figure 5.3: Example of a graph

A **directed graph** $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ is given by a couple of sets:

- A set of **nodes or vertices** $\mathcal{N} = \{1, 2, ..., q\}$
- A set of **links or edges** $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$.

If $(i, j) \in \mathcal{E}$ we say that (i, j) are neighbours. We define the **neighbourhood** of a nodes as the set of the nodes from which it receives information, or equivalently, the vertices from which derive the **incoming edges**. Just to give an example, the neighbourhood of 3 is the set $\mathcal{N}_3 = \{1, 2, 4\}$.
Our aims bring us to make a consideration if there is an edge between two nodes, this means that they communicate.
Some other feature are relevant for our purposes:

- A graph is said to be **complete** if $\mathcal{E} = \mathcal{N} \times \mathcal{N}$;
- The graphs we will analyze are assumed to be **strongly connected**, that is there is always a path between each pair of nodes;

5.2.1 Introduction to Distributed Gradient Descent

For the aim of decentralizing the agents share their **local estimate**, $x^{(i)}(k)$, which denotes the estimate of $\tilde{x}$ at time k, let us assume for simplicity and without loss of generality, that such estimate is a **scalar**.
Once a generic sensor receives the estimate coming from its neighbourhood $\mathcal{N}_i$, it computes a **local mean**

$$\bar{x}^{(i)}(k) = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} x^{(j)}(k) \tag{5.2}$$

where $|\mathcal{N}_i|$ is the cardinality of the neighbourhood. Despite it is true that a local solution of the LS problem (by using the Gradient Descent) would give an inaccurate estimate, the partial functional $F_i = y_i - C_i x$ can be used to run a step of the gradient descent to update the estimate by using the information of the calculated local mean. More precisely each node executes

$$x^{(i)}(k+1) = \bar{x}^{(i)}(k) - \tau \nabla(x^i(k)) \tag{5.3}$$

This equation raises the problem known as the **Distributed Gradient Descent** which mixes: (i) **local** communication by performing the local mean, (ii) individual processing for the gradient step. By iterating this computation an estimate of the global state can be achieved. Note that: we assume that in the neighbourhood of a vertex there is the vertex itself, as if there was a self-loop. One can wonder: Why local mean and not another metric? This observation gives the motivation to entry more in details talking about **Consensus algorithm**.

5.3 The Consensus algorithm

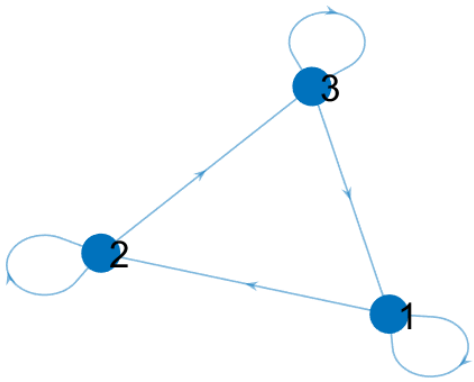
The central point is: **what should it happen if I iterate the local mean?** Under some conditions for each node $i = 1, ..., q$, for a certain time $k \to \infty$

$$\begin{aligned} x^{(1)}(k) &\to x^* \\ x^{(2)}(k) &\to x^* \\ &\vdots \\ \vdots x^{(q)}(k) &\to x^* \end{aligned}$$

Then, the nodes estimate the same global quantity x^*, they reach a **consensus**. In some cases we are interested in **average consensus**, that is

$$x^* = \frac{1}{q} \sum_{i=1}^{q} x^{(i)}(0) \tag{5.4}$$

Let us consider an example for a graph of only three agents which is showed in the figure below (it has been used the `digraph` command of MATLAB):



At a certain time k we have the following local means:

$$\begin{aligned} x^{(1)}(k+1) = \bar{x}^{(1)} &= \frac{1}{2}(x^{(1)}(k) + x^{(3)}(k)) \\ x^{(2)}(k+1) = \bar{x}^{(2)} &= \frac{1}{2}(x^{(1)}(k) + x^{(2)}(k)) \\ x^{(3)}(k+1) = \bar{x}^{(2)} &= \frac{1}{2}(x^{(3)}(k) + x^{(2)}(k)) \end{aligned} \tag{5.5}$$

We have for each node a **local weighted mean** (only the neighbours), each weight can be represented by a scalar Q_{ij} and at this point the system above becomes

$$\begin{aligned} x^{(1)}(k+1) &= Q_{11}x^{(1)}(k) + Q_{13}x^{(3)}(k) \\ x^{(2)}(k+1) &= Q_{11}x^{(1)}(k) + Q_{12}x^{(2)}(k) \\ x^{(3)}(k+1) &= Q_{12}x^{(2)}(k) + Q_{13}x^{(3)}(k) \end{aligned} \tag{5.6}$$

By collecting the Q_{ij} in a matrix

$$Q = \begin{pmatrix} Q_{11} & Q_{12} & Q_{13} \\ Q_{21} & Q_{22} & Q_{23} \\ Q_{31} & Q_{32} & Q_{33} \end{pmatrix} \in \mathbb{R}^{3,3}$$

we can represent the system (5.5) in a matrix form as

$$x(k+1) = Qx(k) \tag{5.7}$$

which summarize in a succinct way the Consensus algorithm where

$$x(k) = \begin{pmatrix} x^{(1)}(k) \\ x^{(2)}(k) \\ \vdots \\ x^{(q)}(k) \end{pmatrix}$$

An important observation can be done on the matrix Q, each element $Q_{ij} \leq 0$, if $Q_{ij} = 0$ then there is no link between i and j. Moreover for each row the sum of the element is equal to one, that is $\sum_{j=1}^{q} x_{ij} = 1$ (convex combination), such a matrix is called **stochastic matrix** (or row stochastic).
We can observe that the Consensus problem is an LTI DT dynamical system and its formulation is completely coincident with the formulation of **Page Rank algorithm** and **Markov Chains**. Now we are going to investigate the fact that the properties of the matrix Q are very important.

5.3.1 Properties of stochastic matrices

The following properties are relevant for our purposes:

- Let Spec(Q)=$\{\lambda_1, ..., \lambda_q\}$ be the set of the eigenvalues of the stochastic matrix Q, it holds that $\lambda_i \leq 1,\ i = 1, ..., q$, so the *spectral radius* of Q is $\rho(Q) = \max_{i=1,...,q}\{|\lambda_i|\} = 1$
- Let $\mathbf{1} = (1, 1, ..., 1)^T$, $Q\mathbf{1} = 1$ this is equivalent to state that $\mathbf{1} \in \mathbb{R}^q$ is an eigenvector of Q and its associated eigenvalues is $\lambda = 1$, this results in a non-asymptotical stability of the dynamical system associated to the consensus problem;
- The first eigenvalue of Q is denoted with λ_{PF} and it is called **Perron-Frobenius** or **leading eigenvalue**.

Starting from $k = 0$ Let us have a look to the evolution of the system, what is its behaviour for $k \to \infty$?

$$\begin{aligned} &x(0) \\ &x(1) = Qx(0) \\ &x(2) = Qx(1) = Q^2x(0) \\ &\dots \\ &x(k+1) = Q^{k+1}x(0) \end{aligned}$$

We can note that it depends only on the power of the matrix Q. Some examples can be useful to try explaining this behaviour.

Example 1

$$Q = \begin{pmatrix} 0.1 & 0.2 & 0.7 \\ 1 & 0 & 0 \\ 0.3 & 0.3 & 0.4 \end{pmatrix} \Longrightarrow \lim_{k\to\infty} Q^k = \begin{pmatrix} 0.3681 & 0.2025 & 0.4294 \\ 0.3681 & 0.2025 & 0.4294 \\ 0.3681 & 0.2025 & 0.4294 \end{pmatrix}$$

The limit has been computed by using numerical techniques, now we know what is the behaviour for $k \to \infty$, because

$$\lim_{k\to\infty} x(k) = \lim_{k\to\infty} Q^k x(0) = [0.3681x^{(1)}(0) + 0.2025x^{(2)}(0) + 0.4294x^{(3)}(0)]\mathbf{1}$$

In this way we reach the consensus, in the sense that the value of the local mean converge to a single value.

Example 2

$$Q = \begin{pmatrix} 0.1 & 0.2 & 0.7 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \Longrightarrow \lim_{k\to\infty} Q^k = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Even in this case we analyze the behaviour for increasing k and we can note that:

$$\lim_{k\to\infty} x(k) = \lim_{k\to\infty} Q^k x(0) = x^{(3)}(0)\mathbf{1}$$

Since the matrix Q has got three identic rows, it achieves the consensus.

Example 3

$$Q = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \Longrightarrow \lim_{k\to\infty} Q^k = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In this case nothing change, as k increases, the global decision is not achieved. In particular:

$$\lim_{k\to\infty} x(k) = \begin{pmatrix} x^{(1)}(0) \\ x^{(2)}(0) \\ x^{(3)}(0) \end{pmatrix}$$

Example 4

$$Q = \begin{pmatrix} 0.1 & 0.2 & 0.7 \\ 0.2 & 0.8 & 0 \\ 0.7 & 0 & 0.3 \end{pmatrix} \Longrightarrow \lim_{k\to\infty} Q^k x(0) = \frac{1}{3}\left[\sum_{i=1}^{3} x^{(i)}(0)\right]\mathbf{1}$$

This is the case in which we have the **average consensus**, this is due to the fact the matrix Q is **doubly stochastic** because is both stochastic and symmetric, this reveal a property about the structure of the graph: it is undirected.

After these examples have been given, we are ready to give the following definitions:

Definition (CONSENSUS)
Given the dynamical system $x(k+1) = Qx(k)$, with $Q \in \mathbb{R}^{q,q}$, $x(k) \in \mathbb{R}^n$, we say that Q achieve the consensus if there exists an $\alpha \in \mathbb{R}$ such that

$$\lim_{k\to\infty} x(k) = \alpha \mathbf{1}$$

Instead we reach the **average consensus** if

$$\alpha = \frac{1}{q}\sum_{i=1}^{q} x^{(i)}(0)$$

The stochastic matrix Q is so important that we can have sufficient conditions for consensus only by analyzing its eigenvalues. In particular the following theorem it has been demonstrated:

Theorem (Perron-Frobenius)
Let $\lambda_1 = \lambda_{PF} = 1$ and $\lambda_1 > |\lambda_2| \leq ... \leq |\lambda_q|$, then Q achieves **consensus**.

Proof (for diagonalizable Q) *Let us assume for simplicity that Q is diagonalizable in a way that is simpler doing the computation Q^k. If Q is diagonalizable, it is similar to a diagonal matrix that is*

$$Q = V\Lambda V^{-1} = V \begin{pmatrix} 1 & 0 & ... & 0 \\ 0 & \lambda_2 & ... & 0 \\ & \vdots & \vdots & \ddots \\ 0 & 0 & ... & \lambda_q \end{pmatrix} V^{-1}$$

$$Q^k = V\Lambda^k V^{-}1 = V \begin{pmatrix} 1 & 0 & ... & 0 \\ 0 & \lambda_2^q & ... & 0 \\ & \vdots & \vdots & \ddots \\ 0 & 0 & ... & \lambda_q^q \end{pmatrix} V^{-1} \to V \begin{pmatrix} 1 & 0 & ... & 0 \\ 0 & 0 & ... & 0 \\ & \vdots & \vdots & \ddots \\ 0 & 0 & ... & 0 \end{pmatrix} V^{-1} =$$

$$= \begin{pmatrix} V_{11} & 0 & ... & 0 \\ V_{21} & 0 & ... & 0 \\ & \vdots & \vdots & \ddots \\ V_{n1} & 0 & ... & 0 \end{pmatrix} V^{-1} = \frac{1}{\sqrt{q}} \begin{pmatrix} 1 & 0 & ... & 0 \\ 1 & 0 & ... & 0 \\ & \vdots & \vdots & \ddots \\ 1 & 0 & ... & 0 \end{pmatrix} V^{-1} = \begin{pmatrix} \beta_1 & \beta_2 & ... & \beta_q \\ \beta_1 & \beta_2 & ... & \beta_q \\ \vdots & & \vdots & \\ \beta_1 & \beta_2 & ... & \beta_q \end{pmatrix}$$

At this point given Q^k, we can compute the limit

$$\lim_{k\to\infty} Q^k x(0) = \sum_{j=1}^{q} \beta_j x^{(j)}(0)\mathbf{1} \Longrightarrow \textbf{\textit{we reach consensus}}$$

In a similar way we can give a sufficient condition for achieving the **average consensus**:

Theorem
Let Q be a doubly stochastic matrix. Let $\lambda_1 = \lambda_{PF} = 1$ and $\lambda_1 > |\lambda_2| \leq ... \leq |\lambda_q|$. Then Q achieves **average consensus**.

Proof

In general for a symmetric matrix

$$Q = V\Lambda V^T = V \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ & \vdots & \vdots & \ddots \\ 0 & 0 & \dots & \lambda_q \end{pmatrix} V^{-1}$$

$$Q^k = V\Lambda^k V^{-}1 = V \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & \lambda_2^q & \dots & 0 \\ & \vdots & \vdots & \ddots \\ 0 & 0 & \dots & \lambda_q^q \end{pmatrix} V^T \to V \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ & \vdots & \vdots & \ddots \\ 0 & 0 & \dots & 0 \end{pmatrix} V^T =$$

$$= \begin{pmatrix} V_{11} & 0 & \dots & 0 \\ V_{21} & 0 & \dots & 0 \\ & \vdots & \vdots & \ddots \\ V_{n1} & 0 & \dots & 0 \end{pmatrix} V^T = \frac{1}{\sqrt{q}} \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ & \vdots & \vdots & \ddots \\ 1 & 0 & \dots & 0 \end{pmatrix} V^T = \frac{1}{q} \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & & \vdots & \\ 1 & 1 & \dots & 1 \end{pmatrix}$$

At this point given Q^k, we can compute the limit

$$\lim_{k\to\infty} Q^k x(0) = \frac{1}{q}\sum_{j=1}^{q} x^{(j)}(0)\mathbf{1} \implies \textbf{\textit{we reach average consensus}}$$

5.3.2 On the connectivity

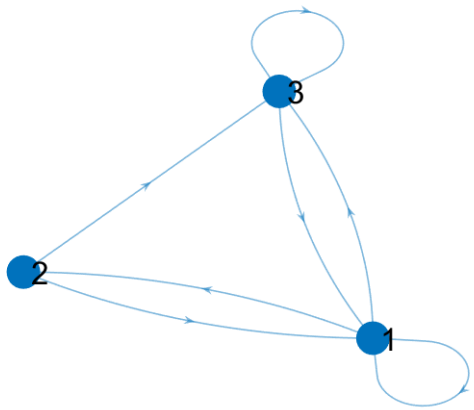
What is the relationship between consensus and the graph topology? In order to investigate such aspects, let us take some of the matrices Q of the previous examples.

Strongly connected graphs

Let us consider the following Q matrix:

$$Q = \begin{pmatrix} 0.1 & 0.2 & 0.7 \\ 1 & 0 & 0 \\ 0.3 & 0.3 & 0.4 \end{pmatrix}$$

The associated graph is the following: We can note that it is **strongly connected**. The



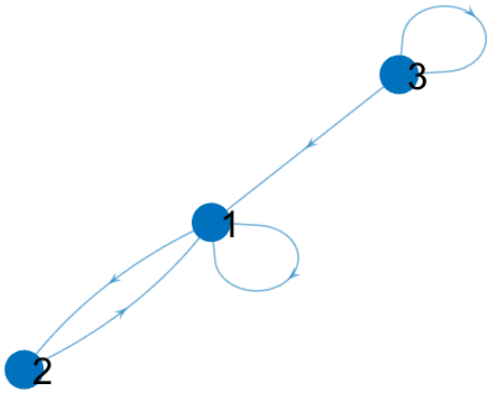
following result is given: **each stochastic matrix related to a strongly connected graph reaches consensus** (for the properties which are required from the Perron-Frobenius theorem).

Leader-follower

From the **Example 2** we had the following matrix:

$$Q = \begin{pmatrix} 0.1 & 0.2 & 0.7 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

From the limit we can note that the node (3) does not change its own state and the consensus is a global mean which depends only on its state, for this reason can be recognized as a **leader agent** that propagates its own information without receiving nothing. The other agents are instead the **follower nodes**. The associated graph shows the exposed property:



The importance of Q_{ij} magnitude

Experimental results are showing that the specific magnitude of Q_{ij} plays not a fundamental role for the fact that a system may reach the consensus. However, in some situations might be important taking into account even the weight of a certain edge. Let us suppose that on a link there is an attack in the sense that one of the sensors spread the value of the state corrupted by an attack a, in those situations, the reduction in magnitude of Q_{ij} can be crucial for the secure estimation of the state.

5.3.3 Convergence rate of the Consensus algorithm

In order to conclude the explanation obout *Consensus*, we give another important result that help us to estimate the convergence of the Consensus Algorithm.

Theorem (convergence rate of Consensus)

The convergence rate of the *consensus algorithm* is determined by the **essential spectral radius** $\texttt{esr}(Q)$ which is the second **largest eigenvalue in magnitude**.

5.4 Uses of the Consensus algorithm

5.4.1 Distributed Least-Squares (II) and Distributed Gradient Descent

The attempt which this paragraph try to meet is that of understanding and formalizing the problem of the **distributed least squares** - just mentioned at the beginning - using also the notions introduced talking about Consensus algorithm.

At first, in order to generalize the problem, let us consider the following *convex optimization problem*

$$\min_{x \in \mathbb{R}^n} F(x) \tag{5.8}$$

where $F : \mathbb{R}^n \to \mathbb{R}$ is a **convex function**. If $F(x)$ is a the sum of several convex functions, it is called a **composite functional**

$$F(x) = \sum_{i=1}^{q} F_i(x) \tag{5.9}$$

In our case, our functional $F(x)$ is the Least-Square one, which can be separated into different convex functionals, in the following way

$$F(x) = \|Cx - y\|_2^2 = \sum_{i=1}^{q} (C_i x - y_i)^2 \tag{5.10}$$

$$F_i(x) := (C_i x - y_i)^2 \tag{5.11}$$

5.4.2 Problem $\mathcal{P}$: distributed minimization of composite functionals

We want to solve the problem

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{q} F_i(x)$$

under the following assumption:

1. each agent knows its own part of the functional F_i
2. each agent i collaborate and share its own local estimate $x^{(i)} \in \mathbb{R}^n$, while no information on F_i is shared.
3. The local estimate is updated:
   (a) Using the functional F_i, by computing its **gradient**
   (b) By collecting the information from neighbours.

If $F(x)$ is convex and differentiable, one can apply the Gradient Descent algorithm in order to reach the **global minimum**, under certain condition. The further step is to decentralize the GD as follows:

DISTRIBUTED GRADIENT DESCENT (DGD)

At each time k, the agent i holds a local estimate $x^i(k) \in \mathbb{R}^n$
For $k = 1, ..., T_{max}$
For each agent $i = 1, ..., q$, the following steps are performed:

1. **Individual step**: each agent computes its own gradient $u^{(i)}(k) = \nabla F_i(x^{(i)}(k))$
2. **Consensus step** a **weighted local mean** of the neighbours' state is computed
$$\bar{x}^{(i)}(k) = \sum_{j=1}^{q} Q_{i,j} x^{(j)}(k)$$
3. **"Merge" step**: $x^{(i)}(k+1) = \bar{x}^{(i)}(k) - \tau u^{(i)}(k)$

The just presented definition was a theoretical one, but **in practice** between the first and the second step, there is a 2-phase **Communication step** in which each agent: (i) broadcasts its own local estimate to the neighbourhood $\mathcal{N}_i$; (ii) receives from the other agents the local estimate. Once this phase is concluded the **Consensus step** can be done. In real world applications, some communication delays ought to be taken into account, but for our purposes we are allowed to neglect them.

On the convergence of DGD

The analysis of the convergence of **Distributed Gradient Descent** is quite challenging. In some papers the study on the convergence is done focusing the attention on the **stopped model**, that is, after a certain time T_s the agents stop the computation of the gradient $\longrightarrow$ for $k > T_s$, the DGD becomes a consensus algorithm.
If $x^* = \arg\min_{x \in \mathbb{R}^n} \sum_{i=1}^{q} F_i(x)$ is the optimal solution and z^* is the solution obtaindd by DGD, then for certain constant c we can have the **convergence** in the sense that

$$\left| \sum_{i=1}^{q} F_i(x^*) - \sum_{i=1}^{q} F_i(z^*) \right| < c$$

5.4.3 Distributed ISTA (DISTA)

What if we had the LASSO functional instead of the Least-Squares one?

We are talking about the well-known least-squares functional to which an ℓ_1-regularization is added. This results in the fact that the Distributed gradient descent is unusable: the ℓ_1-norm is not differentiable!

We can adapt the ISTA algorithm in a distributed context by doing some trivial modifications, the operator $\mathbb{S}_{\lambda\tau}$ (shrinkage and thresholding) has to be used. Recalling that we want to minimize the functional

$$F(x) = \|y - Ax\|_2^2 + \lambda \|x\|_1$$

now is provided the algorithm to recover in a distributed way a sparse solution.

Distributed Iterative Soft Thresholding algorithm (DISTA)

1. **Initialization**: $x^{(i)}(0) \in \mathbb{R}^n$ (eg. $x^{(i)}(0) = 0$)
2. For $k = 1, ..., T_{max}$
3. For each agent $i = 1, ..., q$

$$x^{(i)}(k+1) = \mathbb{S}_{\lambda\tau}\left[\sum_{i=1}^{q} Q_{i,j}x^{(j)}(k) + \tau A_i^T(y_i - A_i x^{(i)}(k))\right]$$

DISTA and Secure State Estimation of CPSs

By doing the proper adjustments to the LASSO functional, we can use the DISTA algorithm for **RSS-fingerprinting localization** and in general for the problem of **Distributed Secure State Estimation of CPSs under attacks**. [It is important to note that that we are now in the **static setting** and so we assume that no observers are employed, the integration between decentralization and dynamic approach passing through the use of a *sparse observer* is different and challenging!]

Part II

Control of Cyber-Physical systems

Chapter 6

Introduction to Control of CPSs

6.1 CPSs as Multi-agent systems

In this second part we will consider **Cyber-physical systems** as **multiagent systems** which in general can be modeled by using a directed graph (**digraph**). The figure below is an example:

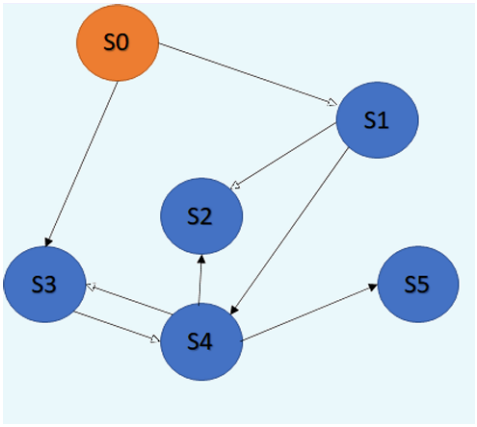


Figure 6.1: A digraph describing a multi-agent CPS

Each agent, denoted with S_i, can be thought as an LTI dynamical system, furthermore the digraph is used in order to represent the **communication** among the agents which compose the *overall system*.
In this particular **framework of multi-agent CPSs** two control problems can be recognized:

- **Cooperative Tracking problem** in which there is one **leader node**, denoted with S_0 and the remaing part of the nodes are N other **identical agents** which are the so-called **follower nodes**; in this type of control setting the agents *cooperate* in order to follow a reference *dictated by the leader node*. Examples of this approach can be: platooning, formation control and so on. We will focus our attention mainly on this one...
- **Cooperative synchronization problem**, the cooperation is also here in order to reach a Consensus, but in this case there is an **absence of a leader node**.

Starting from this point we have to do an **assumption** in order to develop properly the theory:

> **Assumption** The N agents of the multi-agent system describing the CPS are **identical**, including the leader.

More later through the discussion on these topics, we will exploit a trick in order to generalize as much as possible the description of this framework.

6.2 Review of LTI systems structural properties

Before to start talking about control algorithms and protocols can be used in the framework of multi-agent systems, it is better to retrieve and/or introduce some notions which are particularly important:

1. At first a mathematical structure for the agents will be assumed (LTI systems) and some specific **structural properties** will be reminded;

2. Then some notions on **How we can derive the mathematical model of the agents** will be given in the general framework of **black-box Error-In-Variables (EIV) System Identification**

At the moment we can assume that the structure of the dynamical system associated to each agent S_i is the Linear Time Invariant (LTI) one. Then, we can say that in the framework of **Cooperative Tracking Problem** holds that:

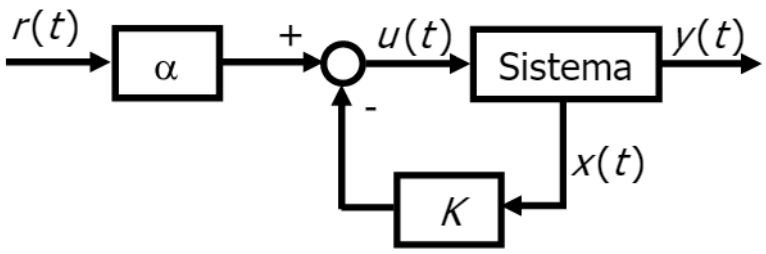
1. The **leader node** S_0 is the following, at the moment we will consider that on the leader node no input is applied
$$S_0 : \begin{cases} \dot{x}_0 = Ax_0 \\ y_0 = Cx_0 \end{cases} \tag{6.1}$$

2. The **follower nodes** $S_i,\ i = 1, ..., N$ are modeled as
$$S_i = \begin{cases} \dot{x}_i = Ax_i + Bu_i \\ y_i = Cx_i \end{cases} \tag{6.2}$$

At this point it is important to assume that **the triple** (A, B, C) is **stabilizable** and **detectable**. Let us remind these important **structural properties** these dynamical systems.

6.2.1 Controllabilty of an LTI system

Roughly speaking we can say that a system is **completely (or fully) controllable** if one can **impose the behaviour** of **ALL the state variables** 'only' by acting on the inputs.
Practically speaking if the system is *fully controllable* then we are able to design a **state feedback** controller K which is able to assign to the matrix $A - BK$ an **arbitrary** selected set of eigenvalues. We are applying to the system the law control input
$$u = -Kx + \alpha r \tag{6.3}$$



Result (Controllability)
An LTI system is **fully controllable** if and only if the **controllability matrix** defined as following

$$M_r = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-b}B \end{bmatrix}, \quad b = rank(B) \tag{6.4}$$

is such that

$$\text{rank}(M_r) = n$$

We can impose to the resulting closed-loop system arbitrary eigenvalues.

What happens if the system is not fully controllable? This is equivalent to state that rank(M_r)=$\rho < n$ and so we can only impose ρ out of n eigenvalues by design a proper state feeedback matrix K, while $n - \rho$ eigenvalues cannot be modified. Two major situations can occur:

(A) The $n - \rho$ eigenvalues have *real part* which is strictly negative $\iff$ the system is **stabilizable** .

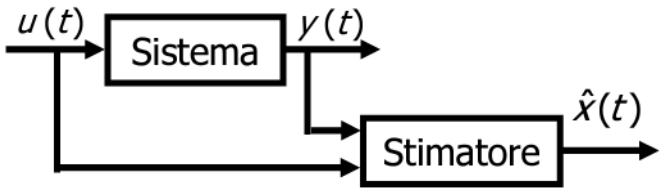
(B) The remaining $n - \rho$ eigenvalues have real part which is greater or equal than zero $\iff$ the system is **not stabilizable**.

We have just understood that when the fully controllability property is not respected the LTI system $\mathcal{S}$ can be divided into two parts:

$$\mathcal{S} = \begin{cases} \text{controllable part} & \rightarrow \rho \text{ eigenvalues} \\ \text{non-controllable part} & \rightarrow n - \rho \text{ eigenvalues} \end{cases}$$

6.2.2 Observability of an LTI system

Roughly speaking we can say that an LTI system is **completely (or fully) observable** if and only if the behaviour of *all the state variables* can be **reconstructed** by only measuring the outputs. This is equivalent to state that **each variable has an effect on the output**.
Practically speaking, if the system if *fully observable*, then we can always design a device called (in general) **Luemberger Observer** which is able to provide an **estimate** asymptotically convergent of all the state variables by only **exploiting the information** provided by the inputs and the outputs.



Then, the **observer** is useful for designing a feedback controller when we can only measure y and not the entire **state vector**. In order to be more specific, on such aspect, if the system is **fully observable**, then we can always find the **observer gain matrix** L, to arbitrarily assign the eigenvalues of the matrix $A - LC$, which describe the dynamics of the **estimation error**.

Result(Observability)
An LTI system is **fully observable** if and only if the **observability matrix** defined as following

$$M_o = \begin{bmatrix} C \\ CA \\ \vdots \\ C^{n-c}A \end{bmatrix}, \; c = rank(C) \tag{6.5}$$

is such that

$$\text{rank}(M_O) = n$$

We can impose to the matrix $A - LC$ a set of eigenvalues which affect the dynamic of the **estimation error**.

What is happening if the system is not fully observable? For sure we can state that rank(M_O)=$\gamma < n$, so we can impose only γ eigenvalues to the matrix $A - LC$.
The remaining $n - \gamma$ eigenvalues can be such that:

(A) Their real part is **strictly negative**, in this case we say that the system is **detectable**. Otherwise

(B) If at least one eigenvalue, among the $n - \gamma$ which are not observable, is null or positive, the system is **not detectable**.

Similarly than the controllability, also in this case we can decompose the system into two other parts:

$$\mathcal{S} = \begin{cases} \text{observable part} & \rightarrow \rho \text{ eigenvalues} \\ \text{non-observable part} & \rightarrow n - \rho \text{ eigenvalues} \end{cases}$$

It is important to remember that:

> Any control technique can work only with the parts **controllable** and **observable** of the system $\mathcal{S}$. The remaining *non-observable* and *non-controllable* part have to be **asymptotically stable**, otherwise whatever is the control method used, those part cannot be modified.

Remark (Transfer function description) Note that we are focusing on the **state-space description** because the tractation we will do is essentially based on this type of model, BUT without loss of generalities, we might use also the description by mean of the **transfer function**. Because:

1. If the system $\mathcal{S}$ is fully observable and controllable, the **poles** of the transfer function $H(s)$ or $H(z)$ are the eigenvalues of the system;
2. If the system is neither fully observable nor controllable, the **poles of the transfer function** are a subset of the eigenvalues of the system, and in particular whose related to the controllable and observable part.

We can obtain from the state-space representation the transfer function directly by applying the **Laplace transform**, the inverse step can be done by using the properties of the **realizability theory**, this frees us to use other types of techniques frequency-based such as *loop-shaping*, LQR, $\mathcal{H}_\infty$ synthesis and so on.

6.3 The communication network through a digraph

We have mentioned that the communication among the agents is modeled by mean of a directed graph. Let us give some more details which are useful when some control algorithms and synchronization protocols are formalized.
The communication network among the agents is represented by the digraph

$$\mathcal{G} = \{\mathcal{V}, \mathcal{E}\},\ \mathcal{V} = \{v_1, v_2, ..., v_N\},\ \mathcal{E} \subset \mathcal{V} \times \mathcal{V} \tag{6.6}$$

However, when we treat the **synchronization problem** it is useful include in the vertices set $\mathcal{V}$ also the vertex associated to the *leader node* obtaining an **augmented graph**

$$\bar{\mathcal{G}} = \{\bar{\mathcal{V}}, \bar{\mathcal{E}}\},\ \bar{\mathcal{V}} = \{v_0, v_1, ..., v_N\},\ \bar{\mathcal{E}} = \bar{\mathcal{V}} \times \bar{\mathcal{V}} \tag{6.7}$$

Chapter 7

Experimental modeling of the agents: System identification

The focus of this chapter is on the problem of **deriving a mathematical model** of N identical LTI agents S_i, $i = 1, ..., N$. The agents can be described:

- In the **Discrete Time Domain** ($t \in \mathbb{R}^+$):

state-space description	$\begin{cases} \dot{x}_i(t) = Ax_i(t) + Bu_i(t) \\ y_i(t) = Cx_i(t) \end{cases}$
transfer function description	$H(s) = C(sI - A)^{-1}B$

- In the **Continuous Time Domain** ($k \in \mathbb{N}$):

state-space description	$\begin{cases} \dot{x}_i(k) = Ax_i(k) + Bu_i(k) \\ y_i(k) = Cx_i(k) \end{cases}$
transfer function description	$H(z) = C(zI - A)^{-1}B$

7.1 What kind of model?

Different approaches are available which are based on **physical insights** and/or **input-output collected experimental data**. We can split this approaches mainly into three groups:

1. **First-principle modeling**: here the structure for the system is derived only by applying the physics, moreover the equation structure is known and taken from the physic theory and, sometimes **the value of the parameter are known**, other times dedicated experiments are conducted in order to retrieve them with a certain approximation; such models are also knows as **white-box models**.

2. **System identification**: is an approach that we find on the opposite part. In fact, here the model is derived by using **both** input-output **collected data** and some **a-priori information** which are fundamental. The parameters come up as an **output** of the SysID procedure but do not have a physical meaning, such models bring to **black-box models**.

3. **Mixed approach**: here the structure of the equation is taken from the Physics (even if partially), the physical parameter to be estimated are meaningful and come up, again, as output of the procedure. The techniques which are based on these considerations are **gray-box models**.

7.2 System Identification: black-box general setting

In general, the first approach which use **first-principle modeling** is not so used because the equation of the Physics are always *approximation of the reality*, sometimes peculiar aspects - not trivial to mathematically formulate - can be wrongly neglected leading to bad models.
The **gray-box approach** is rarely used because similarly than the first case impose strong and not mild constraints on the structure of the equations and on the parameter which being related to the physical meaning cannot be arbitrarily chosen. (We will go more in details on such aspect later).
The most general approach is represented by the **System Identification approach** in which:

1. Input-Output data are exploited, these are affected by *measurement noise*;
2. Some a-priori assumption are used;
3. The **most natural** description, since input and output data are employed, is the **transfer function**. Finally, using the *realization theory* we can obtain the state state description.

$$\text{State-Space} \overset{\mathcal{L}}{\Longrightarrow} \text{Transfer function}$$
$$\text{Transfer function} \overset{\text{Realization}}{\Longrightarrow} \text{State-Space}$$

This approach is very used because of a **great flexibility**, for instance we need only some weak assumption based on physical insights. Now, since the input-output data are **samples** of some signals we can start to discuss the System Identification(SysID) procedure for *discrete-time systems*.
Without loss of generality we can say that the system to be identified (which may be in general nonlinear) can be described by using the *regression form*

$$\begin{aligned} y(k) =& f(y(k-1), y(k-2), ..., y(k-n), \\ & u(k), u(k-1), ..., u(k-m), \theta), \ m \leq n \end{aligned} \tag{7.1}$$

Where n is the order of the system, while m is frequently assumed to be equal to n. In some situation I know the value of n, in other situation One can choose a value for n which is **sufficiently large**, then the data will guide us on the choice of the **system order**. $u(k)$ and $y(k)$ are the error-free signals.
A quite general setting for the SysID procedure can be surely the **Error-in-Variable (EIV)** one in which the measurement noise affects both the input and output **collected data**.

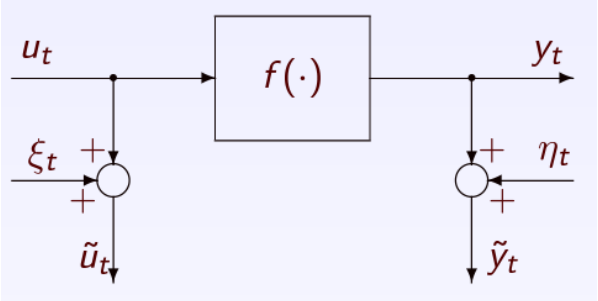


Figure 7.1: EIV System Identification setup

How we outline before:

- ☐ The function $f(\cdot)$ is that to be identified;
- ☐ The terms $\tilde{u}_t$ and $\tilde{y}_t$ are the input-output collected data, which are both the real quantities (u_t and y_t) to which is added
- ☐ The **noise term** indicated with ξ_t and η_t.

At this point, in order to be more specific, let us clarify some aspects about the important **a-priori assumptions** which are basically of three types:

1. On which part of the system the noise enters?
2. Assumption on the class $\mathcal{F}$ of the function $f(\cdot)$ to identify;
3. Is the noise of a certain type? (hypotesis on statistichal distribution, boundness, energy...)

7.2.1 Example (noise free): 2nd order LTI system

We have said that the most natural way to express the structure of $f(\cdot)$ is by using a **transfer function**. But, **How can we obtain it from the regression form?** We can show it using an example.
Let us consider that on the system to be identified we have the following a-priori assumptions:

1. There is no noise, so $\tilde{u}_t = u_t$ and $\tilde{y}_t = y_t$;
2. The system is *linear*, in this way we have choose the class $\mathcal{F}$ for the function f
3. The order of the system is $n = 2$ and $m = n$.

Using the assumption (3), the equation (7.1), in this case becomes

$$y(k) = f(y(k-1) + y(k-2) + u(k) + u(k-1) + u(k-2)) \tag{7.2}$$

Moreover, from the moment we know that the function is linear, we can express it in the following way:

$$y(k) = -\theta_1 y(k-1) - \theta_2 y(k-2) + \theta_3 u(k) + \theta_4 u(k-1) + \theta_5 u(k-2) \tag{7.3}$$

Using the **backward shift operator** according to which $s(k-r) = q^{-r}s(k)$, we rewrite it as

$$\begin{aligned}
&y(k) = -\theta_1 q^{-1} y(k) - \theta_2 q^{-2} y(k) + \theta_3 u(k) + \theta_4 q^{-1} u(k) + \theta_5 q^{-2} u(k) \iff \\
&y(k) + \theta_1 q^{-1} y(k) + \theta_2 q^{-2} y(k) = \theta_3 u(k) + \theta_4 q^{-1} u(k) + \theta_5 q^{-2} u(k) \iff \\
&y(k)[1 + \theta_1 q^{-1} + \theta_2 q^{-2}] = u(k)[\theta_3 + \theta_4 q^{-1} + \theta_5 q^{-2}] \iff \\
&y(k) = \frac{\theta_3 + \theta_4 q^{-1} + \theta_5 q^{-2}}{1 + \theta_1 q^{-1} + \theta_2 q^{-2}} u(k)
\end{aligned}$$

There is a (not trivial) formal proof that the backward shift operator is equivalent to the delay z^{-1} in the $\mathcal{Z}$-transform domain. It is immediate now, to find the transfer function

$$G(z) = \frac{\theta_3 z^2 + \theta_4 z + \theta_5}{z^2 + \theta_1 z + \theta_2} \tag{7.4}$$

Then, we have understood that passing through the *regression form* we can obtain a **transfer function** using the a-priori assumptions and the I/O collected data.

7.2.2 Estimation of the parameters

Assuming, for the moment, we can experimentally collect I/O data without adding any noise, how can we obtain the parameters $\theta_1, ..., \theta_5$? The equation (7.3) have 5 unknowns, we may find a unique solution by adding other equations.
Before start, we have to collect a sufficient number of samples in order to compute $y(k)$, here we need the measurement $y(1), y(2), u(1), u(2)$ in order to compute $y(3)$.[In general if n is the order of the system we start computing $y(k)$ from the instant $k = n + 1$]. It is obtained:

$$\begin{aligned} y(3) &= -\theta_1 y(2) - \theta_2 y(1) + \theta_3 u(3) + \theta_4 u(2) + \theta_5 u(1) \\ y(4) &= -\theta_1 y(3) - \theta_2 y(2) + \theta_3 u(4) + \theta_4 u(3) + \theta_5 u(2) \\ &\vdots \\ y(7) &= -\theta_1 y(6) - \theta_2 y(5) + \theta_3 u(7) + \theta_4 u(6) + \theta_5 u(5) \end{aligned}$$

that it can be rewritten in matrix form as

$$\underbrace{\begin{bmatrix} y(3) \\ y(4) \\ \vdots \\ y(H) \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} y(2) & y(1) & u(3) & u(2) & u(1) \\ y(3) & y(2) & u(4) & u(3) & u(2) \\ & \vdots & \vdots & \vdots & \vdots \\ y(H-1) & y(H-2) & u(H) & u(H-1) & u(H-2) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_H \end{bmatrix}}_{\theta} \tag{7.5}$$

[In our case$H = 7$]. If the matrix A is invertible we can find the solution θ only by inverting the system

$$\theta = A^{-1} y \tag{7.6}$$

In order to guarantee that A is invertible one might think that we can apply a *random* input sequence, so that the columns of the matrix would be linearly independent, but it is not realistic because we want to excite the system with a particular class of signals, moreover **it is not necessary** doing such assumption: it is known what is for example the **step response** of a second-order LTI systems. There is a transient in which the output of the system oscillates a lot:

> it is practically impossible to obtain a matrix whose columns are linearly dependent, even if the system is excited with a very simple signal of the type $u(k) = c \in \mathbb{R}$ the response has oscillations which ensure us, at a certain extent, to take independent measurements. Clearly, **the more exciting the input, the more information can be retrieved by analysing the output**.

7.2.3 Black-box models vs Gray-box models

We have analysed a bit the main features of the SysID approach, so that some more specific comparison can be done with the gray-box approach.
The a-priori assumption on the *linearity* of the system allowed us to pick a function in which **the parameters appear linearly in the model**, on the other hand their meaning is not relevant.
If we used more the physical insights, we are constraining the structure of the equations and so of the transfer function. For example, from the Physics one can find that

$$G(s) = \frac{\sqrt{\gamma_1} s^2 + \gamma_2 s + \gamma_3/\gamma_4}{s^2 + \frac{1}{\gamma_3} s + \gamma_5^2}$$

Here we find that:

- The **system** associated with $G(s)$ is **linear**;
- The **parameters** appear in the equation **non-linearly**, and here they are meaningful because are the parameters derived from the Physics.

In general, it is not trivial to find the solution in the case that **the system is not linear-in-parameters**, so it clearly appears that a SysID approach, without loss of generality, could bring to a simpler solution. What we lose is the meaning of the estimated parameters.

7.2.4 The role of the a-priori information

Introducing the System Identification approach, we have remarked that is crucial that *collected data* and *a-priori assumption* are used in order to estimate the parameters. **What if we do not use the physical insights to retrieve the structure of the system?** Let us give an example for a **static** and **non-linear** system.

$$y(k) = \theta_1 u(k) + \theta_2 u(k)^2 + \theta_3 u(k)^3 + \theta_4 u(k)^4 + \theta_5 u(k)^5$$

Even if the equation linked to the system is **non-linear**, the parameters appear **linearly** in the equation so that we can apply the same approach of the first example, we have five parameters and we need 5 equation, furthermore, since the system is static we need only 5 samples for u and 5 samples for y:

$$\begin{aligned} y(1) &= \theta_1 u(1) + \cdots + \theta_5 u(1)^5 \\ y(2) &= \theta_2 u(2) + \cdots + \theta_5 u(2)^5 \\ \vdots \quad &= \qquad \vdots \\ y(5) &= \theta_2 u(5) + \cdots + \theta_5 u(5)^5 \end{aligned}$$

That in matrix form

$$\begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(5) \end{bmatrix} = \begin{bmatrix} u(1) & u(1)^2 & \dots & u(1)^5 \\ u(2) & u(2)^2 & \dots & u(2)^5 \\ \vdots & \vdots & \vdots & \vdots \\ u(5) & u(5)^2 & \dots & u(5)^5 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_5 \end{bmatrix}$$

If we invert the system by applying the equation (7.6) we will obtain the parameter θ, BUT **what if we use the resulting $y(k)$ to predict the behaviour of the system?** We neglected the relevant fact the system is of the second order! At this point:

1. If we use the collected data to do the prediction, it will be correct for sure **(we built the model relying only on them)**
2. If we apply a certain sequence $u(k)$ to the system $\mathcal{S}$ and we take the corrensponding $y(k)$ samples, we will note that the prediction done according to the identified model is **completely wrong!**.

Conclusion:

> In the System Identification procedure if we rely only on the input-output collected data, we will **overfit** the data, this is the reason why **the a-priori assumptions act a fundamental role in the identification procedure**.

7.2.5 From the ideal situation to the real experiment

Only in order to simplify the explanation, we started with the assumption that the input-output collected data were **noise-free**. In the real-world experiments we cannot do this assumption! Always we have that the collected data is:

$$\tilde{u}_t = u_t + \xi_t \tag{7.7}$$
$$\tilde{y}_t = y_t + \eta_t \tag{7.8}$$

Making us guided from an example, this situation will be more clear. Let us consider a very simple example: a **resistor** through which a current $i(t)$ is passing. The input $u(t)$ is the current $i(t)$ itself, while the output is clearly the voltage $v(t)$ on the resistor. The simple equation which describe this system is

$$y(k) = \theta u(k) \tag{7.9}$$

Following the procedure we did before and considering that only the output $\tilde{y}(k)$ is affected by the noise, we have a **single parameter**; then a single equation, and a **single sample** to be taken of the input and the output, might be sufficient in order to obtain an **estimate** $\hat{\theta}$ of the parameter, which in this case we know that is a **resistance**. If we had used the real $u(k)$ and $y(k)$, the θ parameter would have been

$$\theta = \frac{y(k)}{u(k)} \tag{7.10}$$

but, if we apply our naive approach using the collected data which are corrupted by the **measurement noise** we obtain

$$\hat{\theta} = \frac{\tilde{y}(k)}{u(k)} = \frac{y(k)}{u(k)} + \frac{\eta k}{u(k)} = \theta + \frac{\eta(k)}{u(k)} \neq \theta \tag{7.11}$$

Conclusion:

> If we use a single sample, we will obtain for sure a wrong estimate of the single parameter $\theta \rightarrow$ a **wrong model**. We need more samples and we will look for a θ which is providing the **best-fitting** of the data.

In order to exit from this **wrong situation**, we have to collect for sure more *input-output* pairs in a number $H \gg 2n+1$, where $2n+1$ given a system of order n is the number of parameters to estimate $\rightarrow$ the output of our system identification procedure. However at this point we cannot apply anymore the equation (7.6) because the matrix A becomes a tall matrix, which cannot be inverted! We use the Moore-Penrose pseudo inverse defined as

$$A^{\dagger} = (A^T A)^{-1} A^T$$

and we can solve the problem by simply substituting the inverse with the pseudoinverse:

$$\hat{\theta} = A^{\dagger} y = (A^T A)^{-1} A^T y \tag{7.12}$$

This is the solution of the Least-Squares problem (LS) to the equation (7.5)

$$\hat{\theta} = \arg\min_{\theta} \|A\theta - y\|_2^2 \tag{7.13}$$

In this way $\hat{\theta}$, in the context of System Identification, is the LS estimate of the *parameter vector*. This formulation of the problem has interesting properties if some hypothesis are satisfied:

1. The error apper in the equation as an additive term $e(k)$ called the **equation error**;
2. The $e(k) k = 1, ..., H$ represents a white gaussian noise, that is the samples are *indipendent and identically distributed* (iid).

If such assumption are fullfilled, then

$$\lim_{H \to \infty} \mathbb{E}\left[\hat{\theta}\right] = \theta \tag{7.14}$$

Where θ is the **true parameter vector**. The Least Square approach, no matter what are the dimension of the matrix is **very fast**, moreover there is also an *online recursive way* to compute them.
At this point we are assuming that:

$$y(k) = -\theta_1 y(k-1) - \theta_2 y(k-2) - ... - \theta_n y(k-n) + \theta_{n+1} u(k) + ... + \theta_{n+m+1} u(k-m) + e(k) \iff$$
$$y(k) = -\theta_1 q^{-1} y(k) - \theta_2 q^{-2} y(k) - ... - \theta_n q^{-n} y(k) + \theta_{n+1} u(k) + ... + \theta_{n+m+1} q^{-(n+m+1)} u(k) + e(k) \iff$$
$$y(k)\left(1 + \theta_1 q^{-1} + ... + \theta_n q^{-n}\right) = u(k)\left(\theta_{n+1} q^{n+1} + ... + \theta_{n+m+1} q^{n+m+1}\right) + e(k) \iff$$
$$y(k) D_d(q^{-1}) = u(k)\ N_d(q^{-1}) + e(k) \iff y(k) = \frac{N_d(q^{-1})}{D_d(q^{-1})} u(k) + \frac{1}{D_d(q^{-1})} e(k) \quad \blacksquare$$

This last step we did tells us that the output of the system **to be identified** depends:

1. On the transfer function to be identified $\frac{N_d(q^{-1})}{D_d(q^{-1})}$ (this is ok);
2. On the error filtered by the transfer function $\frac{1}{D_d(q^{-1})}$ which depends directly on something to be identified yet!

Conclusion The Least-Squares estimator used to solve the SysID problem has some advantages: fast convergence and simplicity, however in order to be used requires **strong assumption** on the error, moreover even if this can be obtained, having an acceptable estimate requires the collection of a big quantity of data.

We need an approach where:

1. A **small amount of data** is required;
2. **mild assumption** on the error has to be done.

These needs leads to the **Set-Membership System Identification** theory.

Chapter 8

Set-membership System Identification

The main motivation which leads us to look for another approach to be used is that we want to:

1. Use a small amount of data
2. Have *mild* assumptions on the noise.

8.1 Main ingredients

In order to formulate the **Set-Membership System Identification problem** we need some important ingredient:

- We start from a discrete time *parametrized regression form*

$$y(k) = f(y(k-1), ..., y(k-n), u(k), u(k-1), u(k-m), \theta_1, ..., \theta_{n+m+1}), \quad m \leq n$$

- A-priori assumptions on the **system** $\mathcal{S}$:
  - m, n are known;
  - The function $f \in \mathcal{F}$ (for the moment we choose the LTI class of dynamical systems).
- A-priori assumptions on the **noise**:
  - Noise structure (**OE, EIV**);
  - The input and output noise is part of some **bounded sets** $\mathcal{B}$ and in particular:

$$\eta(k) \in \mathcal{B}_\eta, \quad \xi(k) \in \mathcal{B}_\xi$$

Tipically the sets $\mathcal{B}$ are defined by **polynomial constraints**, it is common that

$$\mathcal{B}_\eta = \{\eta : \ |\eta(k)| \leq \Delta_\eta\}, \quad \mathcal{B}_\xi = \{\xi : \ |\xi(k)| \leq \Delta_\xi\}$$

The **key element** of the Set-Membership identification method is the research of the **Feasible parameter set (FPS)** that we denote with $\mathcal{D}_\theta$. In a nutshell, it is the set of the parameter θ such that all the conditions exposed are fullfilled.

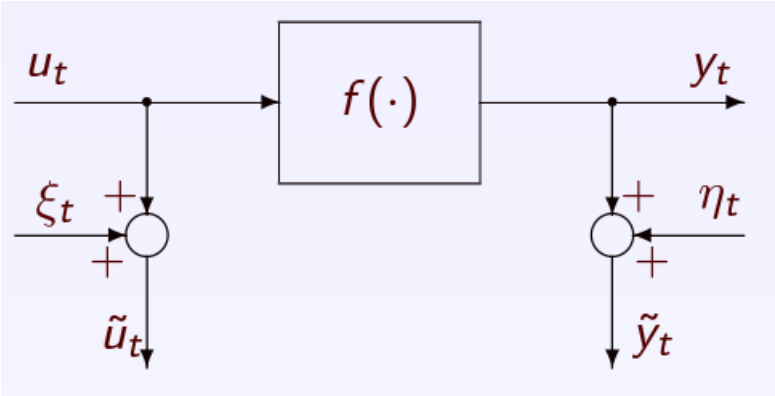


Figure 8.1: EIV set-up

8.2 Noise structures

8.2.1 Error-In-Variable set-up (EIV)

Error-In-Variables (EIV) problem refers to the most general case where the measurement noise is added both in the input and the output.
Due to the conditions we mentioned, we have that:

$$\xi = [\xi(1) \quad ... \quad \xi(H)] \in \mathcal{B}_\xi, \quad \eta = [\eta(1) \quad ... \quad \eta(H)] \in \mathcal{B}_\eta$$

8.2.2 Output-Error set-up(OE)

Output-Error(OE) problem arises when the measurement noise η enters only on the output of the system, while the system is assumed to be **exactly known**. [This is not strange to assume because sometimes one build the sequence of input to stimulate the system].
Due to the condition we mentioned, we have that:

$$\eta = [\eta(1) \quad ... \quad \eta(H)] \in \mathcal{B}_\eta$$

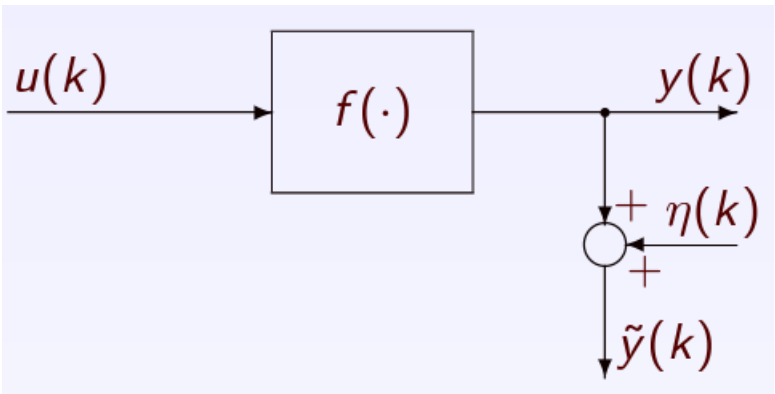


Figure 8.2: OE set-up

8.3 Feasible Parameter Set (FPS)

In the framework of **Set-Membership(SM) Identification**, all the parameters values consistent with the (i) *a-priori information on the model*, (ii) *a-priori information on the noise*, (iii) collected input/output data are considered as:

Feasible solution of the system identification problem

The set of all the parameters which satisfies these conditions is called the **Feasible Parameter Set**. For a **general EIV problem** it can be implicitly defined as:

Feasible Parameter Set (FPS)

$$\begin{aligned}\mathcal{D}_\theta =&\{\theta \in \mathbb{R}^p :\ y(k) = f(y(k-1),...,y(k-n),u(k),u(k-1),...,u(k-m),\theta),\\ &k = n+1,...,H,\\ &y(k) = \tilde{y}(k) - \eta(k), \quad u(k) = \tilde{u}(k) - \xi(k),\ k = 1,...,H\\ &|\xi(k)| \le \Delta_\xi,\ |\eta(k)| \le \Delta_\eta,\ k = 1,...,H\}\end{aligned} \tag{8.1}$$

In the case that we want to identify a LTI discrete time system the set (8.1) is:

$$\begin{aligned}\mathcal{D}_\theta =&\{\theta \in \mathbb{R}^p :\ (\tilde{y}(k) - \eta(k)) + \sum_{i=1}^{n} \theta_i(\tilde{y}(k-1) - \eta(k-1)) =\\ &= \sum_{j=0}^{m} \theta_j(u(k-j) - \xi(k-j)),\ k = n+1,...,H\\ &|\xi(k)| \le \Delta_\xi,\ |\eta(k)| \le \Delta_\eta,\ k = 1,...,H\}\end{aligned} \tag{8.2}$$

The *Feasible Parameter Set* enjoys the following properties:

1. The *true value* of the parameter vector θ is guaranteed to belong to $\mathcal{D}_\theta$;
2. $\mathcal{D}_\theta$ implicitly quantify the uncertainty affecting the found mathematical model.

Once we have found the FPS, we can use it to find **for each parameter** θ_k the *Parameter Uncertainty Interval (PUI)* which is formally defined as

$$PUI_k = [\underline{\theta}_k;\ \overline{\theta}_k], \tag{8.3}$$

$$\underline{\theta}_k = \min_{\theta \in \mathcal{D}_\theta} \theta_k, \quad \overline{\theta} = \max_{\theta \in \mathcal{D}_\theta} \theta_k \tag{8.4}$$

The computation of PUI_k requires to compute the **global optimal solution** of the optimization problems in (8.6).

8.3.1 Example #1

Let us suppose that the model to identify is a static system: a resistor.

Ingredients

1. A-PRIORI INFORMATION ON THE SYSTEM: $y(k) = \theta u(k)$
2. A-PRIORI INFORMATION ON THE NOISE here we assume a **OE** noise structure, where $|\eta(k)| \le \Delta\eta$
3. A-POSTERIORI INFO (I/O COLLECTED DATA), I have the pairs $[u(k), \tilde{y}(k)], k = 1,...,H$

Feasible Parameter Set Computation

$$
\begin{aligned}
\mathcal{D}_\theta =& \{\theta \in \mathbb{R} : y(k) = \theta u(k) \forall k = 1, ..., H \\
& y(k) = \tilde{y}(k) - \eta(k), \quad \forall k = 1, ..., H \\
& |\eta(k)| \leq \Delta_\eta\} \Longleftrightarrow \\
\mathcal{D}_\theta =& \{\theta \in \mathbb{R} : \tilde{y}(k) - \eta(k) = \theta u(k), \quad |\eta(k)| \leq \Delta_\eta, \ k = 1, ..., H\} \\
\mathcal{D}_\theta =& \{\theta \in \mathbb{R} : \eta(k) = \tilde{y}(k) - \theta u(k), \quad |\eta(k)| \leq \Delta_\eta, \ k = 1, ..., H\} \\
\mathcal{D}_\theta =& \{\theta \in \mathbb{R} : \ -\Delta_\eta \leq \tilde{y}(k) - \theta u(k) \leq \Delta_\eta, \quad k = 1, ..., H\} \\
\mathcal{D}_\theta =& \left\{\theta \in \mathbb{R} : \ \theta \geq \frac{\tilde{y}(k) - \Delta_\eta}{u(k)}, \ \theta \leq \frac{\tilde{y}(k) + \Delta_\eta}{u(k)}, \quad \forall k = 1, ..., H\right\}
\end{aligned}
$$

In this case the FPS is the intersection between H intervals, and the PUI for the only parameter θ is coincident with this found interval. We succeded in eliminating the dependence on ξ, η, but this was only a particular case.
If only the system to identify had been only a little bit more complicated, the trick we used to eliminate η from the set, would not have work.

8.4 Extended Feasible Parameter Set (EFPS)

This fact highlight the necessity to enlarge the FPS in such a way, it is able to contain also the variable θ, ξ which depends on the input and output error, defining the **Extended Feasible Parameter Set (EFPS)**.

Extended feasible parameter set (EFPS)

$$
\begin{aligned}
\mathcal{D}_{\theta,\xi,\eta} =& \big\{\theta \in \mathbb{R}^p, \ \xi \in \mathbb{R}^H, \ \eta \in \mathbb{R}^H : \ (\tilde{y}(k) - \eta(k)) + \sum_{i=1}^{n} \theta_i(\tilde{y}(k-1) - \eta(k-1)) = \\
& = \sum_{j=0}^{m} \theta_j(u(k-j) - \xi(k-j)), \ k = n+1, ..., H \\
& |\xi(k)| \leq \Delta_\xi, \ |\eta(k)| \leq \Delta_\eta, \ k = 1, ..., H\big\}
\end{aligned} \tag{8.5}
$$

This is the set of the parameters θ and noise samples ξ, η which are consistent with the *a-priori assumptions*. The FPS $\mathcal{D}_\theta$ we defined in the previous paragraphs is only the projection in the parameter space $\mathbb{R}^p$ of the EFPS $\mathcal{D}_{\theta,\xi,\eta}$. Moreover the PUIs have to be computed on the extended space that in general is a **non-convex set** defined by **polynomial constraints**, in the following way:

$$
\underline{\theta}_k = \min_{\theta,\xi,\eta \in \mathcal{D}_{\theta,\xi,\eta}} \theta_k, \quad \overline{\theta} = \max_{\theta,\xi,\eta \in \mathcal{D}_{\theta,\xi,\eta}} \theta_k \tag{8.6}
$$

One can wonder how to use the PUI for each parameter once you found it. Usually for each PUI_k, the best choice is the **central estimate** θ_c defined as

$$
\theta_{c,k} = \frac{\underline{\theta}_k + \overline{\theta}_k}{2}
$$

By doing an example, there will be the possibility to better clarify some aspects even on the EFPS, that on the **central-estimate**, this gives us also the possibility to introduce some other theoretical aspects.

8.4.1 Example: First order system

Let us consider the problem of identifying a **first order system** of the form:

$$y(k) = -\theta_1 y(k-1) + \theta_2 u(k)$$

The EFPS for such system is as following:

$$\begin{aligned}\mathcal{D}_{\theta,\xi,\eta} = \big\{\theta \in \mathbb{R}^2, \xi \in \mathbb{R}^H, \eta \in \mathbb{R}^H : \tilde{y}(k) - \eta(k) = -\theta_1 \tilde{y}(k-1) + \theta_1 \eta(k-1) + \\ \theta_2 \tilde{u}(k) - \theta_2 \xi(k) \forall k = 2, ..., H \\ |\eta(k)| \leq \Delta_\eta,\ |\xi(k)| \leq \Delta_\xi \big\}\end{aligned}$$

We can note that the equations are exactly the same, with the difference that the FPS $\mathcal{D}_\theta$ is a **projection** of the EFPS onto the **parameter space**. This set is characterized by:

- nonlinear **equality constraints**, in particular they are bilinear and in general non-convex!
- **linear inequality constraints**

The projection of the EFPS on $\mathcal{D}_\theta$ is a non-convex set which can have some strange shape. The figure below constitutes an example of the FPS for the proposed Set-Membership identification problem. The extremities of each PUI is depicted. In this way we have obtain the minimum (2D) hyper-rectangle which contains the Feasible Parameter Set.

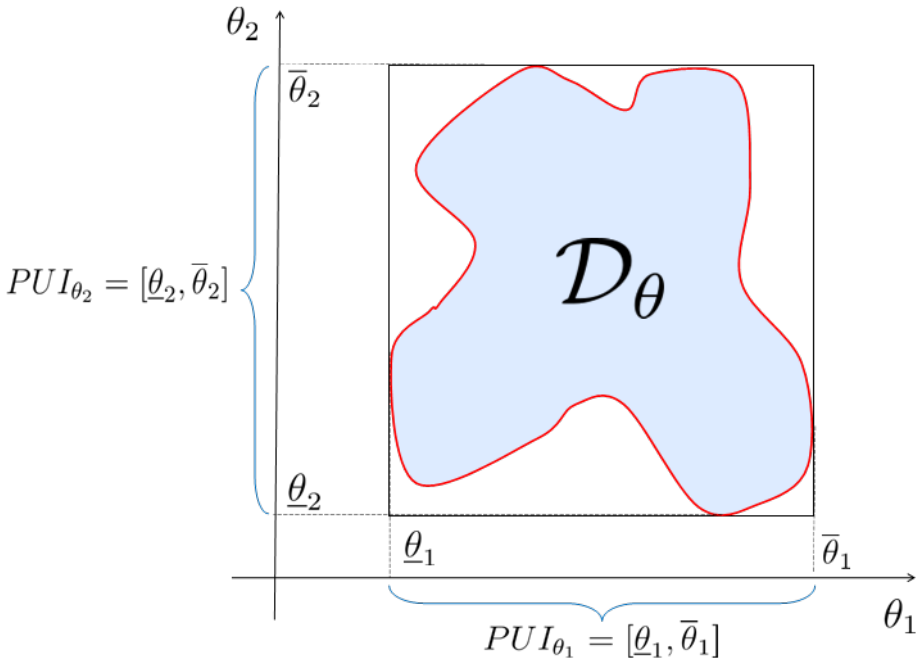


Figure 8.3: FPS for a first order SysID problem

After having completed the computation of the Parameter uncertainty intervals for all θ_k, we can derive using the *backward-shift operator* and the $\mathcal{Z}$-transform the **transfer function** of the agent

$$G(z) = \frac{\theta_2 z}{z + \theta_1}$$
$$\text{where} \quad \theta_1 \in [\ \underline{\theta}_1,\ \overline{\theta}_1\ ],\ \theta_2 \in [\ \underline{\theta}_2,\ \overline{\theta}_2\ ]$$

It is remarkable that for each point inside the set in blue we have a **different model**, but one can wonder: **How can we use this** PUI**?**

1. If we are going to apply either robust control or robust simulation (for example using $\mathcal{H}_\infty, \mu$-synthesis and so on), this modelis already in the **correct form**!

2. In other situations we would like to find a **single model** inside $\mathcal{D}_\theta$ to be used as the "best" (or **nominal model**). That is, we need to find a rule for selecting a single value of (θ_1, θ_2) one single point inside the blue region. In this case it is a common choice to pick *for each PUI* the **central estimate**

$$\theta_k^c = \frac{\underline{\theta}_k + \overline{\theta}_k}{2}$$

this represents the center of the hyper-rectangle inside which the FPS lies. More rigorously, we define the **central estimate** as the solution of the following optimization problem:

$$\theta_c = \min_{\theta \in \mathbb{R}^2} \max_{\theta' \in \mathcal{D}_\theta} \|\theta - \theta'\|_\infty \tag{8.7}$$

which is also known as the **Chebyshev center of $\mathcal{D}_\theta$ in ℓ_∞ norm**

[Note that... in some particular cases the FPS might be so strange that is not connected, in this course the problems which arises these cases will not be discussed.]
This method is guaranteed to be a good choice inside the hyper-rectangle, however such center **is not guaranteed to be inside the FPS**. Consider for example the following case:

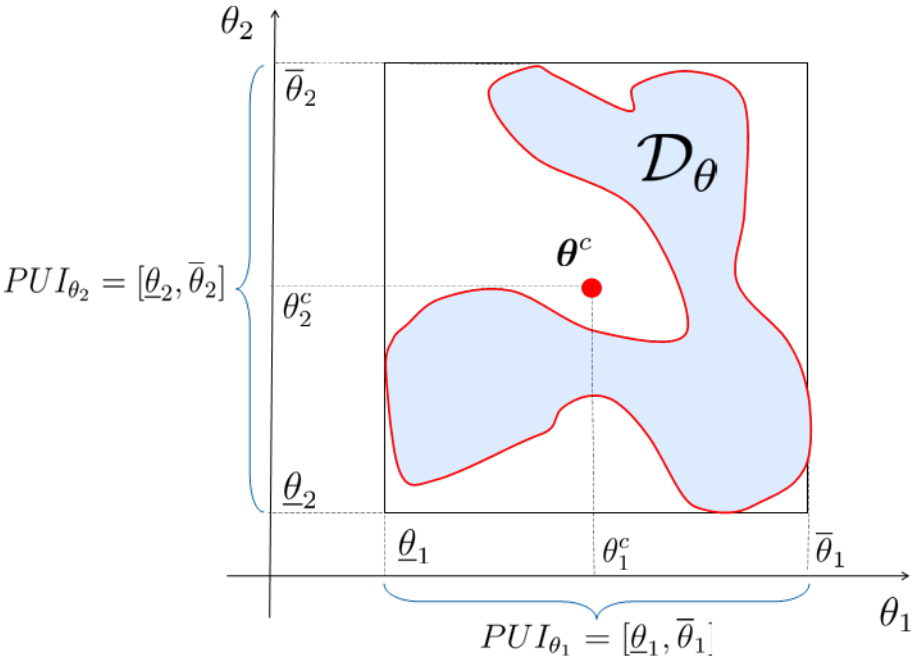


Figure 8.4: FPS with θ^c outside the set

[For this reason, it can be done another type of choice which is based on the use of the **conditional central estimate**, this and other related aspects are outside the purposes of this course].

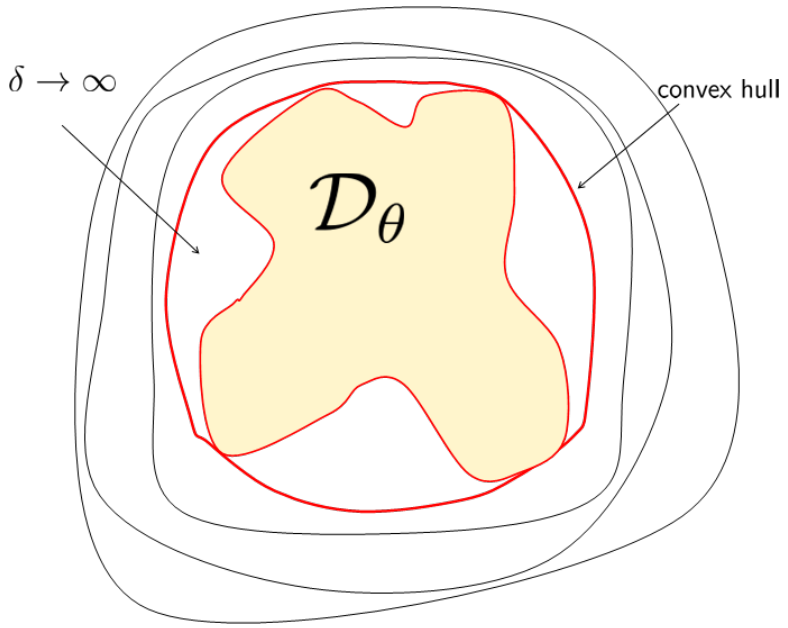
8.5 Convex relaxation for PUIs computation

The fact that the feasible parameter set, in the most general caseof an EIV setting, is characterized by **bilinear constraints** can be used to note that **bilinearity** can be seen also as a special case of a **polynomial constraint**. We know that the FPS, as it is a projection

of a non-convex set, itself it is not convex so we should solve a **non-convex optimization problem** which shows possibly a large number of local minima/maxima (optimal solution). Standard algorithm for nonlinear optimization are not able to compute the **global optimal solution**, that is the same to say that they potentially trap in local minima.
This is not a good news for us, since the intervals resulting from such local optimal solution cannot be qualified as PUIs, because it is not guaranteed that we can find θ_{true} inside them.

For the specific class of polynomial optimization problem (POP), powerful results are available to compute the solution which are based on the Moment Theory (Lassere). This tool can turn the original non-convex problem into a sequence of Semidefinite Programs (SDP) which instead are **convex**, their size depends on a parameter δ which is called the relaxation order.

The method is based on finding a convex approximation $\mathcal{D}_\theta^\delta$ for the FPS which is able to contain it. The higher the relaxation order δ, the better the approximation.
It can be proved that when $\delta \to \infty$ a **convex hull** is obtained which is qualified as the smallest convex set which contains our (non-convex) FPS.



The relaxation order has to have a minimum value which is:

$$\delta_{min} = \left\lceil \max\left(\frac{\deg f_k(x)}{2}\right)\right\rceil \tag{8.8}$$

where $f_k(x)$ are the constraints of the optimization problem. For any relaxation order δ which satisfies the constraint above, which is $\delta \geq \delta_{min}$ it is guaranteed that:

$$\underline{\theta}_k^\delta \leq \underline{\theta}_k, \quad \overline{\theta}_k^\delta \geq \overline{\theta}_k \quad \forall k = 1, ..., n+m+1$$

Moreover it holds that:

$$\lim_{\delta\to\infty} \underline{\theta}_k^\delta = \underline{\theta}_k, \quad \lim_{\delta\to\infty} \overline{\theta}_k^\delta = \overline{\theta}_k \tag{8.9}$$

A **big disadvantage** of such relaxation method is the **computational complexity** that in particular grows exponentially in the relaxation order δ. Therefore, the problem is tractable only in the case that δ is small.

The convex relaxation technique can be applied by using the freely available MATLAB tool `SparsePOP`. Given a POP, this tool automatically computes the convex SDP relaxation for a given relaxation order δ. Finally, `SparsePOP` calls another software which is called `SeDuMi` in order to *solve the SDP problems* obtained by convex relaxation.

8.5.1 Solution of a generic POP using `SparsePOP`

The tool `SparsePOP` is able to solve any optimization problem like:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f_0(x) \\ & \text{s.t.} \\ & f_k(x) \geq 0 \quad (k = 1, ..., l) \\ & f_k(x) = 0 \quad (k = l+1, ..., m) \\ & \mathtt{lb}_i \leq x_i \leq \mathtt{ub}_i \end{aligned}$$

where $f_0, ..., f_k$ are *multivariate polynomial* function of the optimization variable $x \in \mathbb{R}^n$. Let us introduce some examples in order to show how `SparsePOP` formulates the optimization problems, particular data structures has to be introduced for this aim.

Example #1 (general polynomial optimization problem)

It is given the following optimization problem:

$$\begin{aligned} \min_{x\in\mathbb{R}^3} \quad & (-2x_1 + 3x_2 - 2x_3) \\ & \text{s.t.} \\ & 6x_1^2 + 3x_2^2 - 2x^2x_3 + 3x_3^2 - 17x_1 + 8x_2 - 14x_3 \geq -19 \\ & x_1 + 2x_3 + x_3 \leq 5 \\ & 5x_2 + 3x_3 \leq 7 \\ & 0 \leq x_1 \leq 2, \quad 0 \leq x_2 \leq 2 \end{aligned}$$

SparsePOP formulation of the problem:

In the comments of the code we give some extra information, for further information see the SparsePOP manual that is available online.
See: https://sourceforge.net/projects/sparsepop/files/UserGuide.pdf/download

1) Objective function (data structures)

```
objPoly.typeCone=1;          %always 1 here
objPoly.dimVar=3;            %no opt. variables (including the constraints)
objPoly.degree=1;            %degree of f_0
objPoly.noTerms=3;           %number of monomials in f_0
objPoly.supports=support;    %(matrix) see description below
objPoly.coef=coef;           %(matrix) see description below
```

`support` and `coef` are data structures used to describe the objective function. In particular:

- `support` is a matrix with number of rows equal to `noTerms`, number of columns equal to `dimvar`, each entry of such matrix is a *real number* which is the degree of the optimization

variable involved in the **considered term**. In our case since we have $f_0(x) = -2x_1 + 3x_2 - 2x_3$ we have that `support` is equal to

$$\texttt{support} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

it is only a case that it is an identity matrix.

- `coef` is a *column vector* with coefficient of the different terms involved in $f_0(x)$. In our case we have that:

$$\texttt{coef} = \begin{bmatrix} -2 \\ 3 \\ 2 \end{bmatrix}$$

2) Constraints (data structures)

The data structures related to the constraints are very similar, with the only difference that we need to put inequality constraints in the form $f_k(x) \geq 0$. The following data structures have to be filled **for each constraint** of the optimization problem. Finally note that, in the code, the notation `ineqPolySys1` denotes the part of the data structure related to the first constraint. Then, let us bring the first constraint of the optimization problem in a SparsePOP-compatible form:

$$f_1(x) = 19 - 17x_1 + 8x_2 - 14x_3 + 6x_1^2 + 3x_2^2 - 2x_2x_3 + 3x_3^2 \geq 0$$

```
ineqPolySys{1}.typeCone=1;          % 1 if >=, -1 if =
ineqPolySys{1}.dimVar=3;            % the same as before
ineqPolySys{1}.degree=2;            % inequality degree (deg max of the monomials)
ineqPolySys{1}.noTerms=8;           % no terms in the inequality
ineqPolySys{1}.support=support;     %as before
ineqPolySys{1}.coef=coef;           %the same as before
```

Following the fashion of the previous step, let us define `support` and `coef`:

$$\texttt{support} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \qquad \texttt{coef} = \begin{bmatrix} 19 \\ -17 \\ 8 \\ -14 \\ 6 \\ 3 \\ -2 \\ 3 \end{bmatrix}$$

3) Lower and upper bounds

The lower and upper bounds are indicated by employing two vectors called `lbd` and `ubd`, where respectively `lbd`=[$\texttt{lb}_i$,...,$\texttt{lb}_{\texttt{dimVar}}$], `ubd`=[$\texttt{ub}_i$,...,$\texttt{ub}_{\texttt{dimVar}}$]. In our example:

$$\texttt{ubd} = \begin{bmatrix} 2 \\ 1 \\ 1e10 \end{bmatrix}, \quad \texttt{lbd} = \begin{bmatrix} 0 \\ 0 \\ -1e10 \end{bmatrix}$$

Where there are no upper and/or lower bound on a variable, mathematically speaking you should write $-\infty \leq x_i \leq +\infty$. Obviously in MATLAB we cannot use infinite values, for this reason we express this concept by introducing a very large number in the vector at the positions in which such bounds are required.

Example #2 (PUIs computation)

Let us consider an example in which the optimization problems we want to solve comes from a *Set-Membership system identification problem.* In particular our aim is to obtain a mathematical model for the system characterized by the following properties.

First ingredient: A-priori information on the *system*:

- Linear Time Invariant system;
- First order system;

For sure we can assume that such a system it can be described by the transfer function:

$$G(z) = \frac{\theta_2 z + \theta_3}{z + \theta_1}$$

By doing simple calculation we obtain:

$$y(z) = \frac{\theta_2 z + \theta_3}{z + \theta_1} = \frac{\theta_2 + \theta_3 z^{-1}}{1 + \theta_1 z^{-1}} u(z) \rightarrow y(k) = \frac{\theta_2 + \theta_3 q^{-1}}{1 + \theta_1 q^{-1}} u(k) \rightarrow$$
$$y(k)(1 + \theta_1 q^{-1}) = (\theta_2 + \theta_3 q^{-1}) u(k) \rightarrow y(k) + \theta_1 y(k-1) = \theta_2 u(k) + \theta_3 u(k-1)$$

and finally:

$$y(k) + \theta_1 y(k-1) - \theta_2 u(k) - \theta_3 u(k-1) = 0 \tag{8.10}$$

Second ingredient: A-priori information on the *noise*:

- The input $u(k)$ is perfectly known;
- The output $y(k)$ is corrupted by noise obtaining $\tilde{y}(k) = y(k) + \eta(k)$
- The noise η is bounded for each k

Third ingredient: A-posteriori information: Input/Output data experimentally collected. After having collected the I/O data we can exploit them to rewrite the equation (8.10):

$$\tilde{y}(k) - \eta(k) + \theta_1 \tilde{y}(k-1) - \theta_1 \eta(k-1) - \theta_2 u(k) - \theta_3 u(k-1) = 0, \quad k = 2, ..., H$$

where H is the number of collected I/O pairs.

By using a-priori and a-posteriori information we can formulate the *EFPS* as:

$$\begin{aligned} \mathcal{D}_{\theta,\eta} = \{\theta \in \mathbb{R}^3, \eta \in \mathbb{R}^H : \; & \tilde{y}(k) - \eta(k) + \theta_1 \tilde{y}(k-1) - \theta_1 \eta(k-1) + \\ & - \theta_2 u(k) - \theta_3 u(k-1) = 0, \quad k = 2, ..., H \\ & - \Delta_\eta \leq \eta(k) \leq \Delta_\eta, \quad k = 1, ..., H\} \end{aligned}$$

Now for each $\theta_i,\ i = 1, ..., 3$ we have to compute the Parameter uncertainty Intervals, then for each parameter we have to solve the following (non-convex) optimization problems:
1) Computation of $\underline{\theta}_i$

$$\begin{aligned} & \min_{\theta,\eta \in \mathcal{D}_{\theta,\eta}} \theta_i \qquad i = 1, ...3 \\ & \text{subject to:} \\ & \tilde{y}(k) - \eta(k) + \theta_1 \tilde{y}(k-1) + \\ & - \theta_1 \eta(k-1) - \theta_2 u(k) - \theta_3 u(k-1) = 0, \quad k = 2, ..., H \\ & - \Delta_\eta \leq \eta(k) \leq \Delta_\eta, \quad k = 1, ..., H \end{aligned} \tag{OP1}$$

2) Computation of $\overline{\theta}_i$[1]

$$\begin{aligned}
&\min_{\theta,\eta\in\mathcal{D}_{\theta,\eta}} -\theta_i \qquad i=1,...3\\
&\text{subject to:}\\
&\tilde{y}(k)-\eta(k)+\theta_1\tilde{y}(k-1)+\\
&-\theta_1\eta(k-1)-\theta_2u(k)-\theta_3u(k-1)=0, \quad k=2,...,H\\
&-\Delta_\eta\leq\eta(k)\leq\Delta_\eta, \quad k=1,...,H
\end{aligned} \tag{OP2}$$

PUI in SparsePOP

The objective of this paragraph is giving an accurate description of the data structures to be employed in order to solve our optimization problem using the SparsePOP tool. For simplicity we assume that $H=4$ I/O pairs are collected. Since the problems to be solved are very similar we will focus our attention on (OP1) without loss of generality. Moreover the focus will be on the computation of PUI for the parameter θ_1.

Step I: Objective function (`objPoly`)

$$f_0=\theta_i \qquad \text{(objective function)}$$

```
objPoly.typeCone=1;          % always 1 here
objPoly.dimVar=7;            % # opt. variables (including the constraints)
objPoly.degree=1;            % degree of f_0
objPoly.noTerms=1;           % number of monomials in f_0
```

The data structure `objPoly.supports` in this case is a row-vector because f_0 has only one term, while the columns are `objPoly.dimVar`.

$$\texttt{supports}=\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

The term `objPoly.coef` is a scalar due to the fact that there is one monomial in f_0, then `objPoly.coef=1`[2]. The structure `objPoly` is completed as follows:

```
objPoly.supports=supports;
objPoly.coef=1;
```

Step II: equality constraints (`ineqPolySys`)

Let's discuss how can we represent in SparsePOP for the PUI of θ_1 the constraints:

$$\tilde{y}(k)-\eta(k)+\theta_1\tilde{y}(k-1)-\theta_1\eta(k-1)-\theta_2u(k)-\theta_3u(k-1)=0, \quad \text{for } k=2,...,H$$

This means that, since $H=4$ the constraints are:

$$\begin{aligned}
k=2 \quad &\tilde{y}(2)-\eta(2)+\theta_1\tilde{y}(1)-\theta_1\eta(1)-\theta_2u(2)-\theta_3u(1)=0\\
k=3 \quad &\tilde{y}(3)-\eta(3)+\theta_1\tilde{y}(2)-\theta_1\eta(2)-\theta_2u(3)-\theta_3u(2)=0\\
k=4 \quad &\tilde{y}(4)-\eta(4)+\theta_1\tilde{y}(3)-\theta_1\eta(3)-\theta_2u(4)-\theta_3u(3)=0
\end{aligned}$$

[1] Remember that

$$\max_{\theta,\eta\in\mathcal{D}_{\theta,\eta}}\theta_i=\min_{\theta,\eta\in\mathcal{D}_{\theta,\eta}}-\theta_i$$

[2] Note that in the case of the problem (OP2) the only thing which changes for each problem is only the term `coef` that will be `objPoly.coef=-1`

The data structures for these constraints are required to be packed together in the structure `ineqPolySys`, for indicate the subset of data structure for the first constraint we written, for example we use `ineqPolySys{1}`. In the following we are going to discuss their composition only for one constraint, since the procedure is almost the same. Let us analyse the constraint

$$\tilde{y}(2) - \eta(2) + \theta_1\tilde{y}(1) - \theta_1\eta(1) - \theta_2 u(2) - \theta_3 u(1) = 0$$

```
ineqPolySys{1}.typeCone=-1;
ineqPolySys{1}.dimVar=7;
ineqPolySys{1}.degree=2;
ineqPolySys{1}.noTerms=6;
```

The composition rule for `supports` even in the case of equality constraints is the same; we highlighted in blue the optimization variables across the columns and the monomial composing the constraint across the rows.

$$\texttt{supports} = \begin{bmatrix} & \theta_1 & \theta_2 & \theta_3 & \eta(1) & \eta(2) & \eta(3) & \eta(4) \\ \tilde{y}(2) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\eta(2) & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ +\theta_1\tilde{y}(1) & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\theta_1\eta(1) & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -\theta_2 u(2) & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -\theta_3 u(1) & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

As usual in `coef` we put the coefficients of the monomials of the constraints [3]

$$\texttt{coef} = \begin{bmatrix} \tilde{y}(2) \\ -1 \\ \tilde{y}(1) \\ -1 \\ -u(2) \\ -u(1) \end{bmatrix}$$

The procedure must be repeated for `ineqPolySys{2}` (constraint with k=3) and `ineqPolySys{3}` (constraint with k=4). [4]

Step III: Lower and upper bounds The θ_i are unconstrained, then we put $\pm$`1e10` in the corresponding position in the vectors `lbd` and `ubd`. We have instead information on the boundedness of the noise sample $\eta(k)$. Let's assume that $|\Delta_\eta| = 5$. Then:

$$\texttt{lbd} = \begin{bmatrix} -\texttt{1e10} \\ -\texttt{1e10} \\ -\texttt{1e10} \\ -5 \\ -5 \\ -5 \end{bmatrix}, \quad \texttt{ubd} = \begin{bmatrix} \texttt{1e10} \\ \texttt{1e10} \\ \texttt{1e10} \\ 5 \\ 5 \\ 5 \end{bmatrix}$$

Step IV: other parameters
Before calling the `sparsePOP()` command we have to define other two parameters (in the structure called `param`):

[3] All the terms in the monomials different than θ, η are only numbers!

[4] It is quite clear that due to the structure of the problem to be solved, in MATLAB, can be useful using some for cycles to make the notation compact. More specifically there is an external cycle on θ with index i in (OP1) and (OP2) and another cycle on the constraints with index k

1. `param.relaxOrder`: it is the order of relaxation δ, this must be quite 'low' due to the fact that computational complexity grows exponentially with respect to θ; usually `param.relaxOrder=1`
2. `param.POPSolver`: it is for the **solution refinement** after the optimization problem solution; for our aim `param.POPSolver='active-set'`

Final Step: using the `sparsePOP()` function and retrieve the solution
We are ready to solve our optimization problem! How it is showed in the SparsePOP manual, there many ways to call the POP solver, we use the following:

```
[param,SDPobjValue,POP,elapsedTime,SDPsolverInfo,SDPinfo]  = ...
sparsePOP(objPoly,ineqPolySys,lbd,ubd,param);
```

In order to retrieve the solution of the optimization problem:

- `POP.objValueL` contains the **optimal solution** of the optimization problem [5];
- `POP.xVectL` contains the **optimizer** (minimizer in our case);

At the end of this procedure as we said:

- The problem is in the correct form in the case that we want to control the system by using a robust control techniques, first among the others $\mathcal{H}_\infty$ [6];
- If we need to pick a single model, we can select the **central estimate** for each parameter such that

$$\theta_i^c = \frac{\underline{\theta}_i + \overline{\theta}_i}{2}$$

[5] `theta_min(i)=POP.objValueL` if you use a vector to store the $\underline{\theta}_i$. **Important:** when you are computing $\overline{\theta}_i$ you must write `theta_max(i)=-POP.objValueL`
[6] In particular we can use the $\mathcal{H}_\infty$ technique with **structured uncertainty**

Chapter 9

State variable feedback control (SVFB)

Chapter 10

Formation contorl