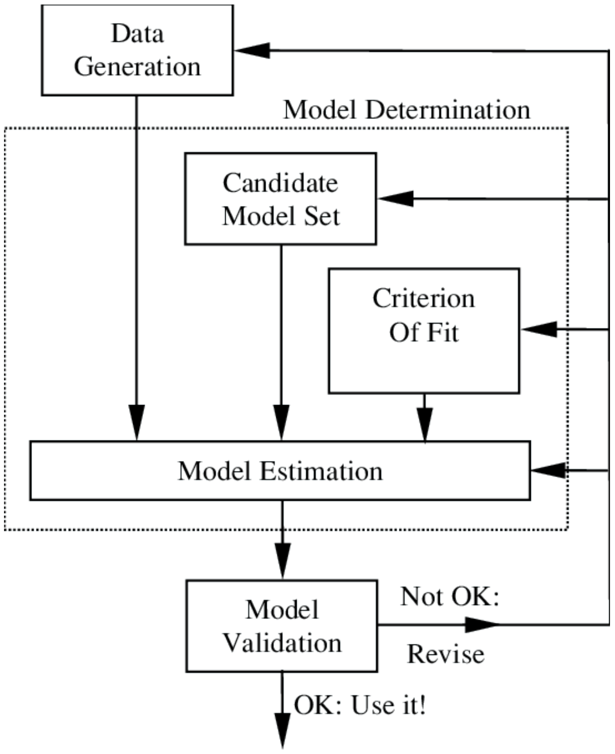
System Identification

Course by M. Baglietto

Notes by
Davide Lanza
with contributions by
Francesca Canale
Filippo Gandolfi

EMARO+ M2
2019-2020



Course outline

Contact
Marco Baglietto marco.baglietto@unige.it, 3rd Floor DIBRIS Building (E), receiving by appointment.

Earning outcomes
The goal of the course is to provide methodologies and tools for designing systems' models to be used for control, estimation, diagnosis, prediction, etc.
Different identification methods are considered, both in a "black box" context (where the structure of the system is unknown), as well as in a "grey box" (uncertainty on parameters) one.
Methods are provided for choosing the complexity of the models, for determining the values of their parameters, and to validate them.
Moreover, state estimation problems are addressed and their connections with control and identification are considered.

Contents
Different models for dynamic systems and their applications:

- Parametric and non-parametric models
- Identification techniques for linear models.
- Nonlinear models. Examples and identification methods.
- Validation procedures.
- Introduction to state estimation.
- State estimation in the presence of disturbances.
- Kalman filter and its extension to the nonlinear case.
- Techniques for parameter identification of linear systems in the presence of disturbances.

Exam
Oral exam, discussing course arguments, asking some demonstrations of algorithms and equations.

References (Not duly following)

- L. Ljung, *System Identification: Theory for the User*, Prentice Hall
- Sodestorm-Stoica, *System Identification*
- Y. Bar-Shalom, X. R. Li, T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*, John Wiley & Sons (→This book is useful for the full demonstrations and some applications like Kalman navigation)

Syllabus

List of arguments included this year.
(*) arguments only for 5 and 6 CFU.
(**) arguments only for 6 CFU.

- Motivations for modelling.
- Different kinds of models: graphical/non-parametric, black-box, grey-box (taylor-made).
- Examples of non-parametric models. Experimental identification of frequency response.
- Taylor-made identification: first order systems.
- Recap for discrete time system modelling of linear systems: difference equations, transfer functions. Hybrid notation.
- Black box modelling of linear discrete-time SISO systems
- Some particular cases: Output Error vs equation error (ARX, ARMAX)
- In/out general representation and corresponding prediction form.
- Identification based on minimum prediction error.
- Identification for ARX
- least squares one-shot solution

(**) recursive least squares method (hints on application to Adaptive control)

- Characterization of inputs: persistence of excitation.
- Mathematical programming (gradient-based and Hessian-based methods).
- Application of mathematical programming to identification. The ARMAX case (and OE case)
- Validation of models. How to choose complexity.
- Cross-validation: Training/Validation/Test sets.

(**) Whiteness tests

- Statistical corrections of the cost (e.g.; Final Prediction Error)

(*) Modelling non-linear systems: Nonlinear-ARX

(*) Use of parametrized approximating functions (Neural nets, RBFs, etc.)

(**) Hints on Approximating properties (Universal approximation, rate of approximation)

(*) Linear in the parameters N-ARX models and least squares identification.

- State Estimation, motivation and some examples.
- Reminders for state equations and possibility of “state augmentation” for joint estimation of state and parameters. Augmented-state estimation as an Identification method.
- Luenberger observer (and Nonlinear generalization)
- Kalman Filter and Extended K.F.

Contents

1 Introduction **1**
1.1 What is estimation . . . 1
1.2 What is (parameter) identification . . . 2

Identification **4**

2 The identification process **5**
2.1 Experimental measurement . . . 6
2.2 *Ad hoc* or Taylor-mode methods . . . 8

3 SISO LDTI description **11**
3.1 Hybrid notation . . . 11
3.2 Prediction form . . . 13
3.3 Prediction hypotheses . . . 16
3.4 Black Box models . . . 18
3.5 Output error model . . . 18
3.6 Equation error models (ARX, ARMAX, ARMA) . . . 18

4 Least-squares methods **21**
4.1 Identification of the best model . . . 21
4.2 Parameter identification: one-shot solution . . . 21
4.3 Parameter identification: dataset . . . 25
4.4 Parameter identification for black box models . . . 27

5 Mathematical programming methods **30**
5.1 Gradient descent . . . 30
5.2 Newton-Raphson algorithm . . . 30
5.3 Parameter identification . . . 32
5.4 Parameter identification for ARMAX . . . 34
5.5 Parameter Identification for Output Error . . . 37

6 Model Validation **38**
6.1 Complexity identification of n, m for ARX . . . 38
6.2 Other ways to chose complexity/validate the model . . . 41
6.3 Whiteness/Anderson test (validation between 2 models) . . . 42
6.4 Statistical corrections of J_N (FPE) . . . 45
6.5 Other methods . . . 47

7 Recursive Online Identification **48**
7.1 Recursive Least-squares with ARX . . . 49
7.2 Recursive LS Algorithm . . . 52
7.3 Algorithm initialization . . . 52
7.4 Algorithm matrix inversion . . . 54
7.5 One-step prediction form . . . 55

8 Identification for non-linear systems **56**
8.1 Function estimation . . . 57
8.2 Approximation properties . . . 58
8.3 Examples of approximated structures . . . 60
8.4 Parameters optimization . . . 62
8.5 N-ARX case . . . 66

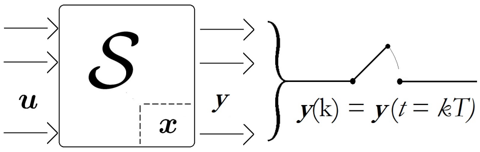
Estimation **67**

9 State (and Parameter) Estimation **68**
9.1 State equations . . . 68
9.2 Observers . . . 69
9.3 Linear Luenberger . . . 70
9.4 Nonlinear Luenberger . . . 71
9.5 Kalman filter . . . 71
9.6 Extended Kalman filter . . . 76

1 Introduction

1.1 What is estimation

Considering a system $\mathcal{S}$. The inputs will be denoted by a $\boldsymbol{u}$ vector ($\boldsymbol{u}(t)$ for continuous-time systems and $\boldsymbol{u}(k)$ for discrete-time systems), the state vector will be denoted bt $\boldsymbol{x}$ and the output vector by $\boldsymbol{y}$:



Let's consider, for example, $u(t) = F(t)$ the input force and $y(t) = p(t)$ the position. If we don't have immediate access to the velocity v, the computation of it it will be an **estimation problem**. We can obtain it deriving the position $p(t)$:

$$v(t) = \frac{d\,p(t)}{dt}$$

Probably we will work with discrete **sampled values**:

$$y(k) = y(t = kT)$$

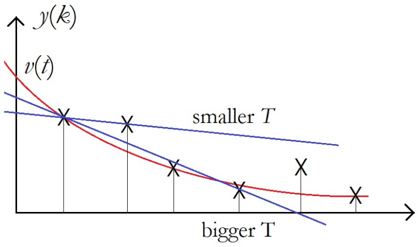
and we will have some additive noise:

$$y(k) = p(k) + \eta(k)$$

So, to estimate v we will have to derive $y(k)$:

$$\frac{y(k+1) - y(k)}{T}$$

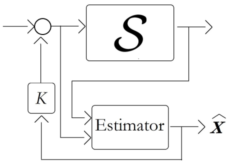
But as we see in the following figure, that's not so trivial. In fact, taking two samples near in time, the error contribution will be enhanced, but taking two samples far in time the result will be smoother w.r.t. the actual v:



Sometimes you want to pass from a transfer function $T(s)$ (time-continuous in this case) to the state equations:

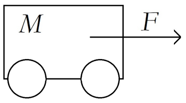
$$\begin{cases} \dot{\boldsymbol{x}} = F\boldsymbol{x}(t) + W\boldsymbol{u}(t) \\ \boldsymbol{y} = G\boldsymbol{x}(t) \end{cases}$$

In this case the state can represent **physical quantities**, but also could be an **abstract representation** without any direct physical meaning, end even with these estimated abstract representations we can stabilize a physical system closing the state-feedback loop correctly:



1.2 What is (parameter) identification

System identification instead is the art of making **models**. A very common model for a cart



is the following one:

$$F = M \cdot a = M \cdot \dot{v}(t)$$

But this is not the cart that we saw in the figure, it's a model (even though also the one in the figure is a model as well). We can even have other different models like:

$$\begin{cases} \dot{x_1} = x_2(t) \\ \dot{x_2} = \frac{1}{M}F(t) \end{cases}$$

where x_1 is the position and x_2 is the position of the cart. Or, again, we can write:

$$T(s) = \frac{1}{M} \cdot \frac{1}{s}$$

Is there a "best model"? It depends. For example, consider the problems of a model like this one:

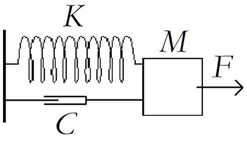
$$F = M \cdot \dot{v}(t)$$

In this model, the parameter M is the mass of the cart, and we can have no tool in our lab in order to measure it. In this case, if we have F and v we can compute it.

!!! We usually don't have all the measurement tools for all the parameters and variables. **!!!**

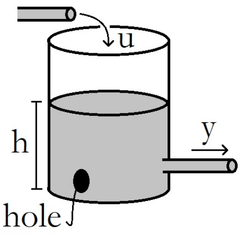
Most of the models are **parametric models**, for example:

$$F(t) - K\ y(t) - C\ \dot{y}(t) = M\ \ddot{y}(t)$$



Is this one a good model? Yes, but only until the spring does not break or that the it will be too hard too pull ("saturation"), as if I push it. So, in these cases, the **model changes**, because **non-linear effects arise**.

So, what a model can be used for? We need a model to **control** a system. Another reason could be to **simulate** a system, or to **predict** its behaviour. These are not the only goals though. For example, a model can be used for **fault detection** or to **estimate** state or parameters. Let's consider the following example:



Here, to identify the fault we have to compare the water input with the output and the water level. What if we don't know how much water is going out at a certain instant t? We have to use a model in order to identify anyway the entity of the fault.

We can distinguish between two main categories:

- **Physical models**
  scaled models like little cars,
  batteries of RC circuits in order to model the behaviour of energy batteries
  ...
- **Mathematical models**
  $T(s)$
  state equations
  ODEs
  ...

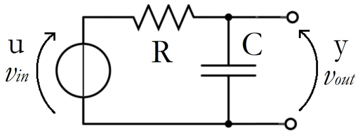
but we can have also other categories like:

- Graphical models
  drawings
  bode plots
  ...

As we already saw, we can divide as well between parametric and non parametric ones, having:

- **Parametric models**
  WB (white boxes)
  GB (gray boxes)
  BB (black boxes)
- **Non-parametric models**

Assume you have an RC system .



We can have as a mathematical model a parametric model like:

$$u = R\,i + y$$

but also the figure of the system that we put just before is a model as well (a graphical model). From the mathematical model we can also obtain an ODE:

$$u = RC\dot{y} + y \rightarrow [\tau = RC,\ i = C\dot{y}] \rightarrow \dot{y} + \frac{1}{\tau}y = \frac{1}{\tau}u$$

and from this we can obtain:

$$T(s) = \frac{1}{1 + s\tau}$$

In this case we knew how the physical system was, we knew its components, and from this knowledge we derived a math model (ODE, $T(s)$ etc...). This is what is called a **gray box** (**GB**). We can then explain the previous dichotomy for **parametric models**:

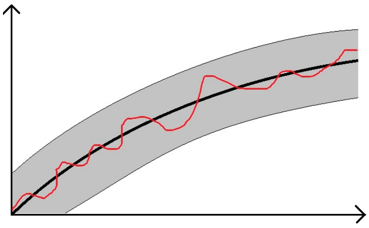
- **WB** (white boxes): know structure, know parameter
- **GB** (gray boxes): known structure, unknown parameter values
- **BB** (black boxes): unknown structure, unknown parameter values, but we can just measure the inputs and the outputs

Regarding **black boxes**, how we can act? We can, for example, assume that is a first order system, so that a $T(s)$ that will fit his behavior will be like:

$$T(s) = \frac{K}{1 + s\tau}$$

Is this a reasonable modeling? It depends, for example on the accuracy required (see the follwing figure). We can test different orders in order to see which one fits our measured behavior:[1]

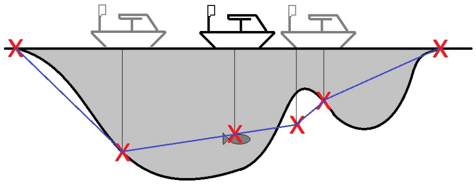
[1] In the following graph the actual behavior is in red, the model behavior in black and accuracy required in gray



1. $T(s) = \dfrac{b_0}{1 + a_1 s}$ (→ 2 parameters)
2. $T(s) = \dfrac{b_0 + sb_1}{1 + a_1 s + a_2 s^2}$ (→ 4 parameters)
3. $T(s) = \dfrac{b_0 + sb_1 + s^2 b_2}{1 + a_1 s + a_2 s^2 + a_3 s^3}$ (→ 6 parameters)

So, the question is, which order is required? **How many parameters** are required to a consistent modeling? In principle, the bigger si the number of the parameters, the better is the capability of approximation for the black box. But in practice? Sometimes using too many parameters could be dangerous or even wrong, obtaining a model that is worse than the one with less parameters.

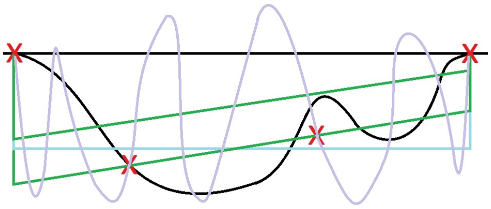
Let's take, for example, a very deep lake like the one in figure (measurements are in red and rough interpolation in blue):



To measure how deep is it, you'll need some tools, for example a stone with a rope. You can measure the rope length in different places. As seen in the figure, you will have a certain degree of uncertainty and measurement error. If you take these measurements, interpolating with straight rows, you'll have an approximation like the one shown in figure, but you can smooth the result using polynomials:

$$P_n(x) = \sum_{i=0}^{n} a_i s^i$$

Let's consider only two measurements:



You can use $P_0(x)$ (cyan) assuming as a_0 the LSE of the measurement. Or, using $P_1(x)$ (green) you can obtain something like shown in figure. Let's say, you can use a $P_{20}(x)$ (violet) that will exactly pass through the points. In this case, this is even worse that the P_1 approximation. A $P_{i<20}$ will be better, maybe something less precise but more generalizable. There is a **tradeoff** between the number of data and the number of parameters (a good rule of thumb is to **never use more parameters than data** to avoid lack of generalization, as shown with $P_{20}(x)$).

2 The identification process

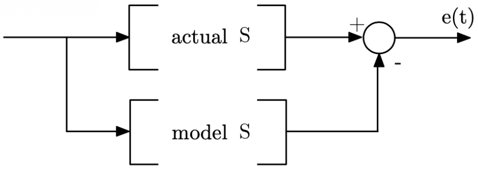
The relevant aspects in building a model are:

1. What you is the **use of the model** (control, estimation, prediction...)
2. Understand what the input and the output (**IN/OUT** $u(t)$ and $y(t)$) will be

   → possibly, the inputs that you want (measurable quantities and directly accessible)
3. **Dataset accessibility** (regarding the chosen IN/OUT): datasets have to be available *a priori* or they can be obtained with experiments

   → normally people don't let you access directly the model, but they just give you datasets. This can happen for different reason, for example in order to not stop a working plant in order to perform identification, or to avoid you test it with treating inputs that can damage the plant
4. Decide the **performance indexes** with which you want to evaluate the quality of the system (you have to decide what "good" does it means w.r.t. the model of system)

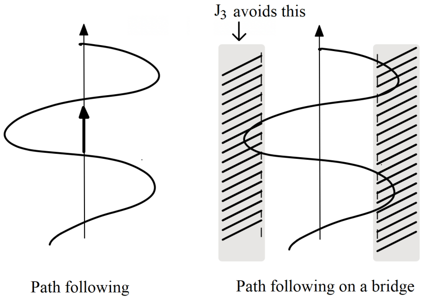
   → For example, in simulation, we want the model output as close as possible to the actual output. Consider a situation like this one

   A "good" model will be a model that minimizes $e(t)$ in a reasonably "short" amount of time. In order to evaluate this we can use, for examples, the **∞-norm**, or the **2-norm** (LSE) if you want to "penalize big errors more"), or the **maximum value of the norm** if you want the biggest error it will be the smallest possible):

   $$J_1 = \sum_{k=1}^{N} ||\boldsymbol{e}(k)|| \qquad J_2 = \sum_{k=1}^{N} ||\boldsymbol{e}(k)||^2 \qquad J_3 = \max_{k=1\ldots N} ||\boldsymbol{e}(k)||$$

   You can chose the criteria w.r.t. to the single case:

   J_3 is the only one which assures to not fall from the bridge
5. Decide which **class** of model is the suitable one. If we call $\mathcal{M}$ the class of the model $\mathcal{S}$, it will depend on the parameters of $\mathcal{S}$, so $\mathcal{M}(\boldsymbol{\theta})$
6. After we decided the class of the model, we have to **design experiments** for testing $\mathcal{S}$
7. Then you have to make the experiments already designed (if the plant is available, w.r.t. step 3) and **record data** from the experiments

8. Before reasoning about the actual value of the parameters, we have to think about the **complexity** of the model, for example the degree of the polynomials that we use. So we will analyse some so-called **meta-parameters**

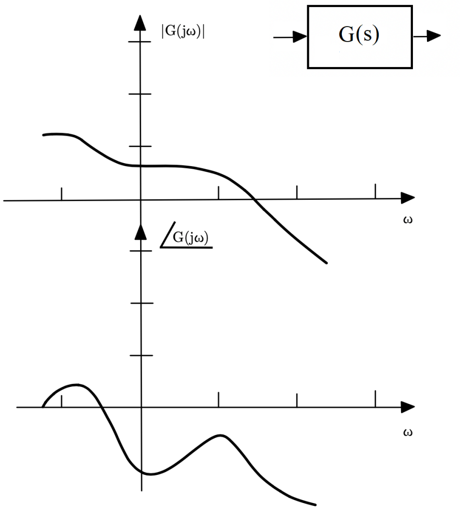
   → for example *ARX* complexity (*ARX* stands for *AutoRegressive eXogenous*, a particular case of *ARMAX* models)
9. After reasoning about the complexity of the model you have to use dataset (directly or given, w.r.t. step 3) to **minimize the cost** w.r.t. the performance indexes decided, in order to perform the **parameter identification**
10. Before actually using your model you can want to validate your model. You can ask yourself "Is this a 'good' model?". There can be some steps to reconsider, like "Is it the 'best' model that I can obtain?", or "Is it what i wanted at the beginning?". So, you have to **validate** your model trying to understand if the model is **satisfactory**

    → if the model is satisfactory, stop the identification process. If the model is not, go back and change some steps, e.g. you can design better experiments, change the class of the model, or determine a different complexity of the model etc...

Sometimes steps 5 and 6 are switched: for example, some experiments are well suited to identify a specific class of model. Sometimes steps 8, 9 and 10 are seen as a unique step.

2.1 Experimental measurement

The Bode plot a classic example of graphical model. Bode plot It is not just a parametric model, even though we are used to think about it as associated to a some transfer function $G(s)$:



Here, we don't have the polynomials of the transfer function, but we know that there are some that constitute the transfer function. Normally the plot is given in a logarithmic or dB scale. In University of Genoa normally it is tradition indicate simply the values $|G(j\omega)|$ in a logarithmic scale $10\log_{10}|G(j\omega)|$ instead of using the dB scale $20\log_{10}|G(j\omega)|$). Regarding the phase of $G(s)$, the x-axis is a linear representation, while ω is still in a logarithmic scale (as seen in the previous figure).

We normally **assume** that the system is **linear** (it obeys to the superposition law regarding inputs and initial conditions) and it is **asymptotically stable**. Why are we supposing that is asymptotically stable? Because the output of the system is composed by two parts, the forced output (null initial condition, $\boldsymbol{x}_0 = 0$) and the free output (null input, $u = 0$):

$$y(t) = y_{u=0}(t) + y_f(t)$$

So, assuming that is asymptotically stable means that:

$$y_{u=0}(t) \underset{t\to\infty}{\rightarrow} 0 \quad \forall \boldsymbol{x}_0$$

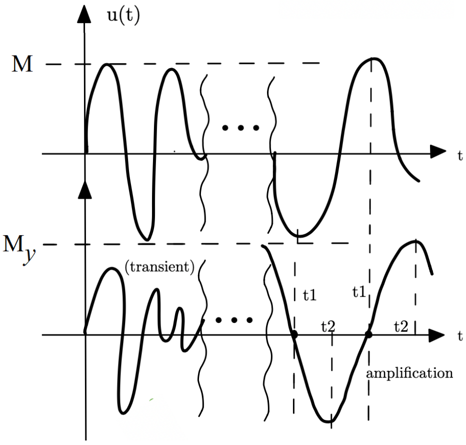
So, if $u(t) = M\sin(\omega t)$, we will have a forced transient response and a forced permanent response:

$$y_f(t) = \underbrace{y_{f_t}}_{\to 0} + y_{f_p}$$

So, asymptotically, every element will go to 0 but the forced permanent response, that will define the **regime** of the system:

$$y_{f_t}(t) \underset{t\to\infty}{\to} 0 \qquad y_f(t) \underset{t\to\infty}{\to} y_{f_p}(t)$$

So, probably, at the beginning you will not be able to record really clear data, but after a reasonable time you will start to have some reasonable data, like a delayed amplification of the input:



After some time, you will be able to recognize a shifted amplified sinusoidal with the same frequency of the input one:

$$y_{f_p}(t) = \underbrace{M|G(j\omega)|}_{M_y}\sin[\omega t + \underbrace{\angle G(j\omega)}_{\text{phase shift}}]$$

So, we will be able to record the different amplitude M_y and perform the following operation (see previous figure as reference):

$$|G_{(}j\omega)| = \frac{M_y}{M}$$

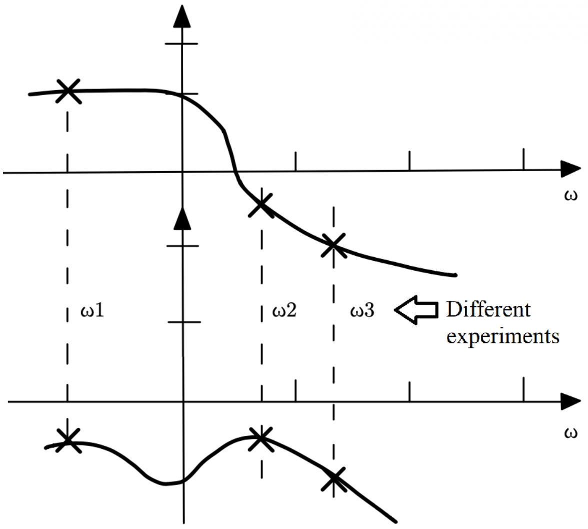
And, regarding the phase:

$$y_{f_p}(t) = M|G(j\omega)|\sin\left[\omega\left(t + \underbrace{\frac{\angle G(j\omega)}{\omega}}_{\text{time delay}}\right)\right]$$

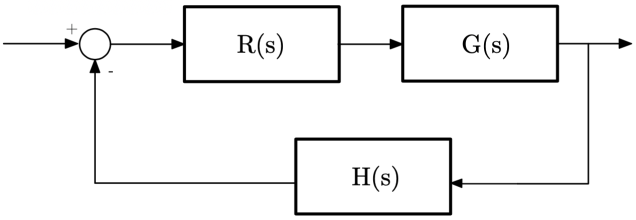
So (again, it could be useful to use as a reference the previous figure):

$$\angle G(j\omega) = (t_1 - t_2)\omega$$

But how much time we will need to wait in order to have the system at regime? The output behavior will tell us: if it has a sinusoidal behavior, it is in regime. So, we will choose a frequency ω, we will make an experiment and we will register the values of $|G(j\omega)|$ and $\angle G(j\omega)$ for that frequency. Then, we will chose a different ω. If we report these samples on a plot, we can interpolate them in order to guess the corresponding Bode plot, at least a reasonable one, as the one in the following figure:



Note: After identifying the system, normally you want to apply to it a control, normally with a regulator and a feedback ($R(s)$ and $H(s)$ notation) as can be seen in the following figure:



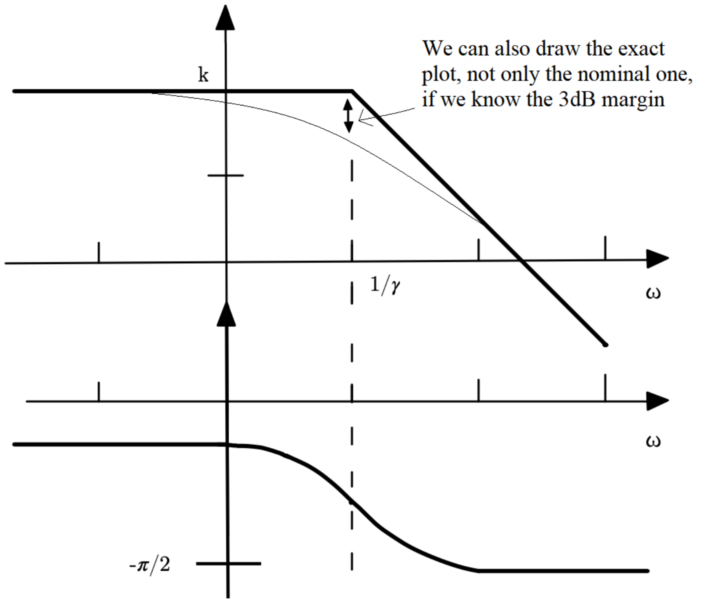
If the plant is instead **unstable** you cannot apply a direct control, you have to stabilize it before. If you want to **identify an unstable system** you can start trying to control it but trying not to make it "explode".

2.2 *Ad hoc* or Taylor-mode methods

First order systems Consider a system such that

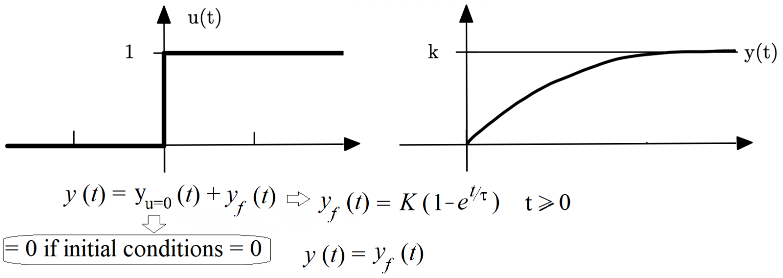
$$G(s) = \frac{K}{1 + s\tau}$$

This is normally the situation in which we are when we have some physical knowledge, like for a RC circuit (→ grey box model) with $\tau = RC$. If you have to identify a system like this, from the obtained $G(s)$ you cannot have the knowledge of R and C, but only of their product τ. If you plot the Bode plot of this system you will have something like the one shown in the following figure:


k
We can also draw the exact plot, not only the nominal one, if we know the 3dB margin
$1/\gamma$
ω
ω
$-\pi/2$

As we can see, we can draw some exact values, even though is mostly a qualitative-approximated plot. That is possible because we can identify the parameters K and τ with several methods:

- We can know the exact value of K because it is the regime response for $u = \sin(\omega t)$ for a reasonably low frequency ω_{low} s.t. $|G(j\omega_{\text{low}})| = K$ → **frequency response** analysis.
- Since we know that $\angle G(1/\tau) = -45°$, we can try increasing values of ω until $\angle G(\omega) = -45°$. At that point we will know that $\tau = 1/\omega$ → **frequency response** analysis (be careful, it can explode when you increase ω if it's not asymptotically stable).
- We can also compute K from the **step response**. We will be in a situation like the one in the following figure:


u(t)
1
k
y(t)
$y(t) = y_{u=0}(t) + y_f(t)$ ⇨ $y_f(t) = K(1-e^{t/\tau})$ $t \geqslant 0$
= 0 if initial conditions = 0
$y(t) = y_f(t)$

But how we can reach the $\boldsymbol{x}_0 = 0$ requirement (null initial conditions)? For a RC circuit, we will have for example to discharge the circuit, the capacitors, and how we will make it? With null input, but this is true only if it is an **asymptotically stable** system! The way to obtain initial condition for such a system is then just let the input null for some time t_{wait}, and then put as input the step:

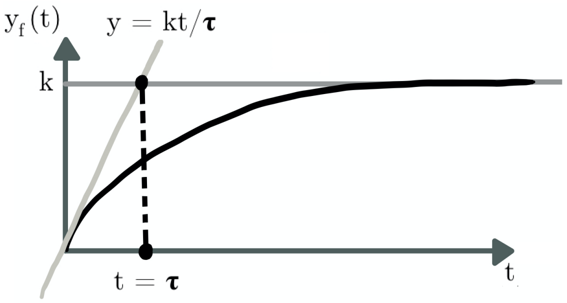
$$u(t) = \mathbf{1}(t - t_{\text{wait}})$$

We will indicate $t - t_{\text{wait}}$ as the 0 point, thanks to the time invariance of the response of the system.

After that, probably our output $y(t) = y_f(t)$ it will have some noise, so we will have to wait until it will be **reasonably** constant. At regime, the variations in the value of $y_f(t)$ will give us the uncertainty on the measurement of K.

But what if there is measurement noise? We can take different values K_i and then doing the average in order to reduce the effect of noise.

- From the **step response** of the asymptotically stable system we can also identify graphically the value of τ, without a frequency response analysis:



Let's take the output:

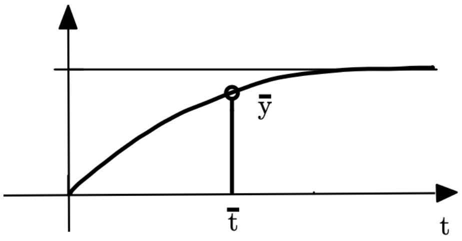
$$y_f(t) = K(1 - e^{-t/\tau}) \qquad t \geq 0$$

and derive it:

$$\dot{y_f}(t) = \frac{K}{\tau} e^{-t/\tau} \underset{t\to 0^+}{\rightarrow} \frac{K}{\tau}$$

So, the slope of the tangent of $y(0^+) = y_f(0^+)$ will give us $\frac{K}{\tau}$, from which we can obtain τ. In this case is normally better to use a ruler instead of using numerical values, because of the problems linked to the numerical derivation process.

- There is also a third method to determine τ from **step response**. We can define a value of the output such that:



$$\bar{y} = K\left(1 - e^{-\bar{t}/\tau}\right) \quad \rightarrow \quad \bar{y} - K = -Ke^{-\bar{t}/\tau} \quad \rightarrow \quad e^{-\bar{t}/\tau} = \frac{K - \bar{y}}{K}$$

So:

$$\tau = -\frac{\bar{t}}{\ln\left(\dfrac{K-\bar{y}}{K}\right)}$$

But how we chose $\bar{t}$? If it is too big, we will be at regime, so there will be a big problem, because $\bar{y} = K$ and the formula just seen for τ will not be defined. To identify correctly the time constant τ, $\bar{t}$ has to be chosen in the transient time-domain, not in the regime time-domain!

Also, another strong assumption of this method is the absence of measurement noise. But with noise, it may happen to consider some values that is too big or too small w.r.t. what we need. To solve this, we can take different values $\bar{t_i}$ and then probably after doing an average between the τ_i obtained (while staying out of the regime time-domain for each t_i !)

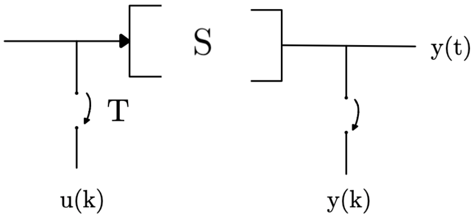
***n*-order systems** We can solve the same problem for second order systems, dealing with the analytical forms of the step response (it is more difficult for $n > 2$). It could be easier to use frequency response analysis, but it will be difficult anyway. **For *n*-order systems it is better to user not the Taylor-made approach, but the least-squares approach** (see Section 4).

3 SISO LDTI description

In this section we will present the main Single Input Single Output (SISO) Linear Discrete-Time and Time Invariant (LDTI) systems descriptions. But before we start, we have to make a parenthesis regarding the notation we will use.

3.1 Hybrid notation

Let's consider a SISO system as the one shown in this figure, in which we have continuous and discrete values for the input and the output $u(t), u(k), y(t), y(k)$: [2]



Regarding the continuous domain, we have

$$Y_f(s) = G_c(s)U(s)$$

while for the discrete domain, we have

$$Y_f(z) = G(z)U(z)$$

Let's **consider** the system as **asymptotically stable** (it is sufficient but not necessary for our purposes, it is a stronger hypothesis, but makes things easier).

Why are we going to work in discrete time domain? Because we will deal with measurements, and the measurements will be **sampled values**. Then we will have to pass from $G(k)$ to $G_c(t)$ (e.g. bilinear transformation).

We will adopt here an **hybrid**, formally wrong, **notation**:

$$Y(z) = G(z)\,U(z) \qquad \text{Notation: } \; y(k) = \underline{\underline{G(z)}}\,u(k)$$

This notation implicitly implies (so, it is formally wrong because of these assumptions):

- $y(k) = y_f(k)$
- $G(z) = \dfrac{B(z)}{A(z)} = \begin{cases} \text{Notation \#1 e.g.} & \dfrac{3z^2+1}{z^3-(0.5)z^2} \\ \\ \text{Notation \#2, e.g.} & \dfrac{3z^{-1}+z^{-3}}{1-(0.5)z^{-1}} \end{cases}$

  So, we will prefer the Notation #2:

  $$B(z) = \sum_{i=0}^{m} b_i z^{-i} \qquad A(z) = 1 + \sum_{i=1}^{n} a_i z^{-i}$$

  Is there any condition for which this is related to a real (causal) system? No, a system like this is **always a causal system**! If you think about some condition like $n > m$, that's because you where thinking about Notation #1. For having a proper transfer function you don't have to check the maximum value of n, but that the maximum degree of z which appears in the polynomial $A(z)$ is not lower than the one that appears in $B(z)$. But in Notation #2 there is 1 as bigger degree, it does not rely on the value of n! This is possible because we have assumed $a_0 = 1$ **<u>!!!</u>**. Then, b_0 can be equal to 0, but the polynomial in the denominator cannot have lower degree that the numerator, that's why we prefer Notation #2: it implies a causal system no matter what (→ always proper transfer function, physically realizable).

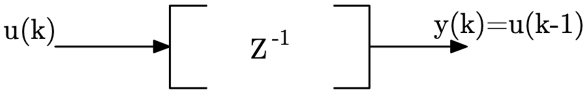
[2] **Recall:** $\mathcal{Z}$-transform for a discrete signal $v(k) \rightarrow V(z) = \sum_{k=0}^{\infty} v(k)z^{-k}$

In the differential equation form we will have another formally wrong element of our **hybrid notation**. Let's consider a unit delay:

$$y(k) = u(k-1) \rightarrow Y(z) = z^{-1}U(z) \qquad \text{Notation:} \quad y(k) = \underline{\underline{z^{-1}}}u(k)$$

So, we will have the following behaviour:

k	0	1	2	..
u(k)	u(0)	u(1)	u(2)	...
y(k)	0	u(0)	u(1)	...



So, as seen in the figure above, we can think the delay as a z^{-1} transfer function block .

Let's consider now a FIR filter transfer function:

$$G(z) = \frac{0.5 + 2z^{-2} - 3z^{-3}}{1}$$

This is a proper transfer function, thanks to our notation, in fact:

$$G(z) = \frac{0.5 + 2z^{-2} - 3z^{-3}}{1} = \frac{0.5z^3 + 2z - 3}{z^3}$$

We have three poles in 0, it is the most stable 3-poles system we can meet! Now, considering z^{-1} as the delay block, we can easily get the ODE from our Notation #2 transfer function:

$$y(k) = G(z)u(k) = 0.5u(k) + 2[z^{-2}]u(k) - 3[z^{-3}]u(k) = 0.5u(k) + 2u(k-2) - 3u(k-3)$$

Let's take a system whose $G(z)$ is not as good as the previous one:

$$G(z) = \frac{2 - z^{-1} + z^{-2}}{1 - z^{-1}}$$

The idea is to do the same trick of the previous case, in order to write y(k) as a product of z^{-n} blocks. So, we will divide the numerator for the denominator:

$$\begin{array}{r|l} 2 - z^{-1} + z^{-2} & \underline{1 - z^{-1}} \\ \underline{2 - 2z^{-1} \qquad} & 2 + z^{-1} \\ z^{-1} + z^{-2} & \\ \underline{z^{-1} - z^{-2}} & \\ 2z^{-2} & \end{array}$$

This division never ends, but we can go as we want:

$$\begin{array}{r|l} 2 - z^{-1} + z^{-2} \qquad\quad & \underline{1 - z^{-1} \qquad\qquad\qquad} \\ \underline{2 - 2z^{-1} \qquad\qquad\qquad\qquad} & 2 + z^{-1} + 2z^{-2} + ... \\ z^{-1} + z^{-2} \qquad\quad & \\ \underline{z^{-1} - z^{-2}} \qquad\quad & \\ 2z^{-2} \qquad\qquad\quad & \\ \underline{2z^{-2} - 2z^{-3}} & \\ 2z^{-3} & \\ ... & \end{array}$$

The division ends only if simplifications are possible (F.I.R. case), but we can rewrite $G(z)$ as:

$$G(z) = \frac{2 - z^{-1} + z^{-2}}{1 - z^{-1}} = 2 + z^{-1} + 2z^{-2} + ...$$

and we will have:

$$y(k) = G(z)u(k) = (2 + z^{-1} + 2z^{-2} + ...)u(k) =$$
$$= 2u(k) + [z^{-1}]u(k) + 2[z^{-2}]u(k) + ... =$$
$$= 2u(k) + u(k-1) + 2u(k-2) + ...$$

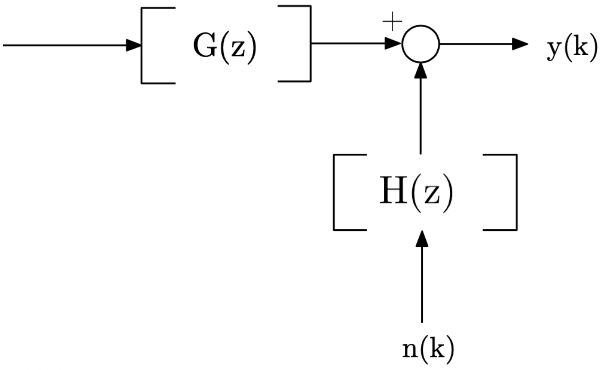
We can think then in general $G(z)$ **as the never ending division** of $B(z)$ by $A(z)$:

$$y(k) = G(k)u(k) = (g_0 + g_1 z^{-1} + g_2 z^{-2} + g_3 z^{-3} + ...)u(k) =$$
$$= g_0 u(k) + g_1 u(k-1) + g_2 u(k-2) + g_3 u(k-3) + ...$$

This is what is called the **convolution** of the input with the **input response** of the system.

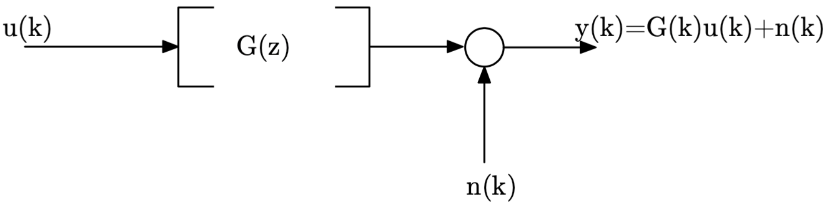
3.2 Prediction form

Consider the following open-loop scheme:



$$y(k) = G(z) \underset{(control)}{u(k)} + H(z) \underset{(noise)}{n(k)}$$

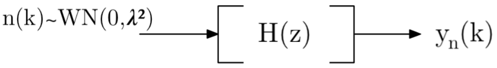
Here, $H(z)n(k)$ includes the unmodelled dynamics but also the effect of disturbances. For example, if $H(z) = 1$ we will have an output noise (like a measurement noise):



The signal $n(k)$ is assumed to have zero mean, unknown variance and to be white:

$$n(k) \sim W_N(0, \lambda^2)$$

Be careful, $N(0, \lambda^2)$ is a Gaussian probability distribution. $W_N(0, \lambda^2)$ is instead **white noise**, so a sequence of **uncorrelated** values. But, having $n(k) \sim W_N(0, \lambda^2)$, we will have:



$$H(z)n(k) = y_n(k) = h_0 n(k) + h_1 n(k-1) + h_2 n(k-2) + h_3 n(k-3) + ...$$

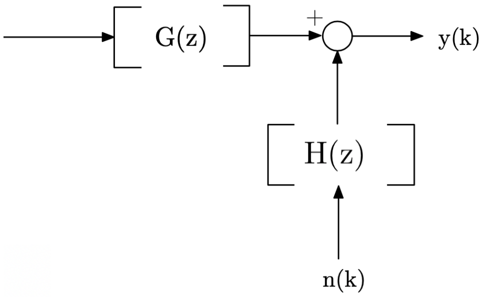
and:

$$y_n(k-1) = h_0 n(k-1) + h_1 n(k-2) + h_2 n(k-3) + ...$$

We easily notice that $y_n(k-1)$ and $y_n(k)$ are **correlated**: even if the noise is white, the output is correlated! So, $H(z)n(k)$ won't be a white sequence, but something that we can call a "**coloured sequence**".

The expected value of $n(k)$ is 0. During simulations, we cannot know $H(z)$, but we can assume $H(z) = 1$ and use white noise in $n(k)$ and register in average what happens to $y(k)$.

So, the overall scheme for us will be the one previously seen:



$$G(z) = \frac{B(z)}{A(z)} \qquad H(z) = \frac{D(z)}{C(z)}$$

This will lead to:

$$y(k) = \frac{B(z)}{A(z)}u(k) + \frac{D(z)}{C(z)}n(k)$$

$G(z)$ and $H(z)$ are causal transfer functions, and:

$$A(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + ... + a_n z^{-n}$$

where n is the minimum degree at which z appears. Similarly: [3]

$$B(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + ... + b_m z^{-m}$$

$A(z)$ is **monic**, as well as $C(z)$:

$$C(z) = 1 + c_1 z^{-1} + c_2 z^{-2} + c_3 z^{-3} + ... + c_l z^{-l}$$

Even $D(z)$ is monic as well:

$$D(z) = 1 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3} + ... + d_q z^{-q}$$

[3] In some textbooks we can find the following notation:

$$G(z) = z^r \frac{\tilde{B}(z)}{\tilde{A}(z)}$$

We can obtain the same notation taking out from B(z) the first zero coefficient terms (if $b_{0,1,...,i} = 0$).

Why $D(z)$ is monic? $B(z)$ was not demonic because there is no sense in putting b_0 to one, unless we know the exact value of $u(k)$. Regarding $D(z)$ instead, consider this following case with $\tilde{D}(z)$ (non-monic version):

$$\frac{\tilde{D}(z)}{C(z)}n(k) = \frac{\tilde{d}_0 + \tilde{d}_1 z^{-1} + ... + \tilde{d}_q z^{-q}}{1 + c_1 z^{-1} + ... + c_l z^{-l}}\tilde{n}(k)$$

So, if $\tilde{d}_0 \neq 0$, we can write $1 + d_1$ having $d_1 = \frac{\tilde{d}_1}{\tilde{d}_0}$ and so on, so:

$$\frac{D(z)}{C(z)}n(k) = \frac{1 + \frac{\tilde{d}_1}{\tilde{d}_0} z^{-1} + ... + \frac{\tilde{d}_q}{\tilde{d}_0} z^{-q}}{1 + c_1 z^{-1} + ... + c_l z^{-l}}\tilde{d}_0\tilde{n}(k) = \frac{1 + d_1 z^{-1} + ... + d_q z^{-q}}{1 + c_1 z^{-1} + ... + c_l z^{-l}}\tilde{d}_0\tilde{n}(k)$$

But what if $\tilde{d}_0 = 0$? Let's assume $\tilde{d}_0 = 0$ but $\tilde{d}_1 \neq 0$. We will divide $\tilde{D}(z)$ by $\tilde{d}_1 z^{-1}$:

$$\frac{D(z)}{C(z)}n(k) = \frac{1 + d_1 z^{-1} + ... + d_q z^{-q}}{1 + c_1 z^{-1} + ... + c_l z^{-l}}\tilde{d}_1 z^{-1}\tilde{n}(k)$$

But $\tilde{d}_1 z^{-1}\tilde{n}(k)$ is a noise (delayed and changed in power) that we will call $n(k)$. We will never use the values of the white noise, so we have no interest in using the true values (that's the difference we where stressing before between $B(z)$ and $D(z)$: for $B(z)$ we are interested in the real values of $u(k)$, for $D(z)$ we're not interested in the real values of $n(k)$).

If we're not interested in the values of $n(k)$, we could nevertheless be interested in estimating its variance $\tilde{\lambda}^2 \rightarrow \lambda^2$, but it won't be used in system identification: the only requirement we need is the **zero mean** requirement: time shifting and change of power will not change the expected value (*remind*: the requirements where zero mean, **unknown variance** and **whiteness**).

Now, it is useful to write $y(k)$ in the so-called **prediction form**. What does "predicting" means? Let's assume that the system $\mathcal{S}$ is perfectly modelled by:

$$y(k) = \frac{B(z)}{A(z)}u(k) + \frac{D(z)}{C(z)}n(k)$$

Assume also that all the measurements $y(\bar{k} < k)$ are available and also $u(\bar{k} \leq k)$ (hence $u(k)$ is available, $y(k)$ isn't):

k	...	$k-2$	$k-1$	k
$u(k)$	...	$u(k-2)$	$u(k-1)$	$u(k)$
$y(k)$	...	$y(k-1)$	$y(k-1)$	**?**

Here lies the "predictive" nature of the process: form the past history of output and the history and the current value of the input, we want to predict the current output $y(k)$. So, we want to rewrite $y(k)$ as a function of $u(\bar{k}), \forall\bar{k} \leq k$ and $y(\bar{k}), \forall\bar{k} < k$. But there is a problem in here, the unknown quantities:

$$y(k) = \underset{(we\ can\ simulate)}{\frac{B(z)}{A(z)}u(k)} + \underset{(unknown)}{\frac{D(z)}{C(z)}n(k)}$$

So, we have to ask ourselves: are $n(k-1), n(k-2)...$ correleated with $y(k-1), y(k-2)...$? Yes, we saw it before. But $n(k)$ is **not correlated** to $y(k)$! Then, what is the prediction for $n(k)$? It is clearly:

$$n(k) \sim W_N(0, \lambda^2) \qquad \text{(Mean value = 0 !!!)}$$

We can then rewrite:

$$H(z)^{-1}y(k) = H(z)^{-1}G(z)u(k) + n(k)$$

and then, to isolate y:

$$y(k) - y(k) + H(z)^{-1}y(k) = H(z)^{-1}G(z)u(k) + n(k)$$

we have:

$$y(k) = [1 - H(z)^{-1}]y(k) + H(z)^{-1}G(z)u(k) + n(k)$$

Using the matrices notation:

$$y(k) = \left[1 - \frac{C(z)}{D(z)}\right] y(k) + \frac{C(z)B(z)}{D(z)A(z)} u(k) + n(k)$$

This is the system in the **prediction form**. But it seems we have two transfer functions $\left[1 - \frac{C(z)}{D(z)}\right]$ and $\frac{C(z)B(z)}{D(z)A(z)}$. But this is quite surprising, because it seems that $y(k)$ is function of $y(k)$ itself! That is not really true. Let's consider the **predictive part of the output considering the input**:

$$y_u^p(k) = \frac{C(z)B(z)}{D(z)A(z)} u(k)$$

We can compute y_u^p only if the block $\frac{C(z)B(z)}{D(z)A(z)}$ is a causal transfer function. We know that the part $\frac{B(z)}{A(z)}$ is causal, and then both $C(z)$ and $D(z)$ are monic, so the block is causal!

Considering also the **predictive part of the output considering the output**:

$$y_y^p(k) = \left[1 - \frac{C(z)}{D(z)}\right] y(k) = \left[1 - \frac{1 + c_1 z^{-1} + ... + c_l z^{-l}}{1 + d_1 z^{-1} + ... + d_q z^{-q}}\right] y(k)$$

We can, again, divide the two polynomials:

$$\begin{array}{r|l} C(z) & D(z) \\ \cline{2-2} (...) & 1 + \alpha_1 z^{-1} + \alpha_2 z^{-2} ... \end{array}$$

$$y_y^p(k) = \left[1 - \frac{C(z)}{D(z)}\right] y(k) = \left[1 - 1 + \alpha_1 z^{-1} + \alpha_2 z^{-2} ...\right] y(k) = \left[\alpha_1 z^{-1} + \alpha_2 z^{-2} ...\right] y(k)$$

So then, $y_y^p(k)$ is not function of $y(k)$ but:

$$y_y^p(k) = \alpha_1 y(k-1) + \alpha_2 y(k-2) ...$$

We can also rewrite:

$$\left[1 - \frac{C(z)}{D(z)}\right] = \left[z \frac{D(z) - C(z)}{D(z)}\right] z^{-1}$$

So:

$$y_y^p(k) = \left[z \left(\frac{D(z) - C(z)}{D(z)}\right)\right] y(k-1) = \left[\frac{(d_1 - c_1) z^{-1} + (d_2 - c_2) z^{-2} + ...}{1 + d_1 z^{-1}}\right] y(k-1)$$

This is a proper transfer function! So it is ok to write the predictive form like this:

$$\boxed{y(k) = \left[z \left(\frac{D(z) - C(z)}{D(z)}\right)\right] y(k-1) + \left[\frac{C(z)B(z)}{D(z)A(z)}\right] u(k) + n(k)}$$

3.3 Prediction hypotheses

What is the prediction $\hat{y}(k)$ of $y(k)$ in this case? The best prediction you can make, even in condition of perfect knowledge of the system parameters, is to put $n(k) = 0$ (it is not possible to predict the effect of the disturbances!):

$$\hat{y}(k) = \left[z \left(\frac{D(z) - C(z)}{D(z)}\right)\right] y(k-1) + \left[\frac{C(z)B(z)}{D(z)A(z)}\right] u(k)$$

Consider the $A(z)$ polynomial: [4]

$$A(z) = 1 + a_1 z^{-1} + ... + a_n z^{-n}$$

[4] At the exam, don't make confusion between this A and the A matrix of the state space representation.

We can write it as a function of all the parameters $\boldsymbol{\theta} = [a_1, ..., a_n, b_0, ..., b_m, c_1, ..., c_l, d_1, ..., d_q]^T$:

$$A(z) = A(z|\boldsymbol{\theta})$$

And so on for the others polynomials. So, we can rewrite:

$$(Hp.1) \qquad y(k) = \frac{B(z|\boldsymbol{\theta})}{A(z|\boldsymbol{\theta})}u(k) + \frac{D(z|\boldsymbol{\theta})}{C(z|\boldsymbol{\theta})}n(k)$$

This, again, is not true, is an **hypothesis**! Maybe the system is not linear, maybe we don't know the value of parameters: our hypothesis is that the system obeys to this equation (in our mind the system ideally is this one). Similarly, we don't have the true values of the parameters[5], but ideal values. So, this is our second hypothesis:

$$(Hp.2) \qquad \boldsymbol{\theta} = \{\text{true values of the parameters}\}$$

So, that's our **ideal case**:

1. $\mathcal{S}$ is perfectly described by $y(k) = \frac{B(z|\boldsymbol{\theta})}{A(z|\boldsymbol{\theta})}u(k) + \frac{D(z|\boldsymbol{\theta})}{C(z|\boldsymbol{\theta})}n(k)$
2. $\boldsymbol{\theta}$ is known

So, our prediction (**ideal estimate**) is:

$$\boxed{\hat{y}(k|\boldsymbol{\theta}) = \left[z\left(\frac{D(z|\boldsymbol{\theta}) - C(z|\boldsymbol{\theta})}{D(z|\boldsymbol{\theta})}\right)\right]y(k-1) + \left[\frac{C(z|\boldsymbol{\theta})B(z|\boldsymbol{\theta})}{D(z|\boldsymbol{\theta})A(z|\boldsymbol{\theta})}\right]u(k)}$$

So, our two hypothesis will lead to a **prediction error** that is:

$$y(k) - \hat{y}(k|\boldsymbol{\theta}) = n(k)$$

That is the error of our best ideal prediction! There is a good news anyway: our error will have a zero-mean value ($W_N(0, \lambda^2)$).
But what if our hypothesis will not hold?

$$(\underline{not}\ \ Hp.2) \qquad \boldsymbol{\theta} \xrightarrow{\sim} \hat{\boldsymbol{\theta}} \equiv (\text{estimated parameters})$$

So, if $\boldsymbol{\theta}$ is not known (but we have still some hypothesis on n, m, l, q) we will have :

$$A(z|\hat{\boldsymbol{\theta}}) = 1 + \hat{a_1}z^{-1} + ... + \hat{a_n}z^{-n}$$

In this case then, our Hp.1 still holds, but $\boldsymbol{\theta}$ is unknown. So, our prediction (**real estimate**) is:

$$\boxed{\hat{y}(k|\hat{\boldsymbol{\theta}}) = \left[z\left(\frac{D(z|\hat{\boldsymbol{\theta}}) - C(z|\hat{\boldsymbol{\theta}})}{D(z|\hat{\boldsymbol{\theta}})}\right)\right]y(k-1) + \left[\frac{C(z|\hat{\boldsymbol{\theta}})B(z|\hat{\boldsymbol{\theta}})}{D(z|\hat{\boldsymbol{\theta}})A(z|\hat{\boldsymbol{\theta}})}\right]u(k)}$$

Note: apart from this theoretic introduction, we will use

$$A,\ A(z),\ A(z|\boldsymbol{\theta}), \qquad y,\ y(k)$$

indifferently for situations in which **Hp.1 and Hp.2 hold**, while

$$A(z|\hat{\boldsymbol{\theta}}),\ \hat{A},\ \hat{A}(z),\ \hat{A}(z|\hat{\boldsymbol{\theta}}) \qquad \hat{y},\ \hat{y}(k),\ \hat{y}(k|\hat{\boldsymbol{\theta}})$$

will be used indifferently for situations in which **only Hp.1 holds**.

[5] n, m, l, q are the **meta-parmeters**

3.4 Black Box models

We have (if Hp.1 holds) this **input-output representation**:

$$y(k) = \frac{B(z|\boldsymbol{\theta})}{A(z|\boldsymbol{\theta})}u(k) + \frac{D(z|\boldsymbol{\theta})}{C(z|\boldsymbol{\theta})}n(k)$$

that in **predictive form** is:

$$y(k) = \left[z\left(\frac{D(z|\boldsymbol{\theta}) - C(z|\boldsymbol{\theta})}{D(z|\boldsymbol{\theta})}\right)\right]y(k-1) + \left[\frac{C(z|\boldsymbol{\theta})B(z|\boldsymbol{\theta})}{D(z|\boldsymbol{\theta})A(z|\boldsymbol{\theta})}\right]u(k) + n(k)$$

These models are usually **black-box models** for LTI Discrete-Time systems. We have a lot of situations that can be represented by this form, and we can make some further assumptions.

Before doing it, we have to note that the predictive form not only gives us clearly the current y value in function of the previous ones, but also isolates the noise $n(k)$. In fact, a wrong assumption is that, since $n(k)$ is a white noise with expectation value 0, we **don't consider the noise**. But this **is wrong**! Because, as we already seen, we can rewrite

$$\frac{D(z|\boldsymbol{\theta})}{C(z|\boldsymbol{\theta})}n(k)$$

as a function of the history of y. So, it is $n(k)$ not correlated, but the **previous noise values** $n(\bar{k} \leq k)$ are **correlated**! So, the predictive form isolates the only value that is not correlated $n(k)$.

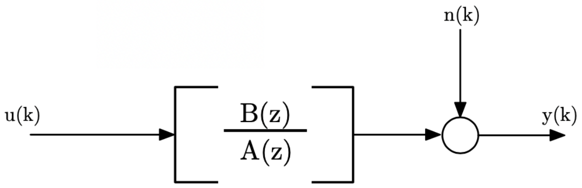
So, "what we ask" to the predictive form? Two requirements:

- ***Re*1** explicit dependence of $y(k)$ from $y(\bar{k} \leq k)$
- ***Re*2** uncorrelated actual noise value $n(k)$ isolation

3.5 Output error model

If we are in a situation like the following one, we are in the most obvious special case, where

$$C(z) = D(z) = 1$$



This is the system in which the error is added in the output → output error model:

$$\boxed{y(k) = \frac{B(z|\boldsymbol{\theta})}{A(z|\boldsymbol{\theta})}u(k) + n(k)}$$

This model is both in input-output representation and in predictive form (it respects $Re1$ and $Re2$ requirements).

3.6 Equation error models (ARX, ARMAX, ARMA)

Output error models are not the only way to represent a system. For continuous time systems we have ODEs, for discrete-time systems we have difference equations like this one:

$$A(z)y(z) = B(z)u(k)$$

If the polynomials are not exact, we can write the error like:

$$A(z)y(k) - B(z)u(k) = \epsilon(k)$$

But we are not interested in this because is not physically characterised! It does not have zero means or any particular property. For us it will be good only if we assume that:

$$\epsilon(k) = \begin{cases} n(k) \sim W_N(0, \lambda^2) \quad \rightarrow ARX \\ \quad \text{or} \\ D(z)n(k) = n(k) + d_1 n(k-1) + ... + d_q n(k-q) \quad \rightarrow ARMAX \end{cases}$$

where $D(z)n(k)$ is the weighted noise of the last q values of the white noise. Considering the first case, we have the so-called **equation error model** (ARX), a model for which $C(z) = A(z)$ and $D(z) = 1$:

$$A(z)y(k) = B(z)u(k) + n(k)$$

$$\boxed{y(k) = \frac{B(z)}{A(z)}u(k) + \frac{1}{A(z)}n(k)} \quad \text{(ARX I/O)}$$

This is the input-output representation. How to put it in predictive form in order to respect $Re1$ and $Re2$? Let's expand the polynomials:

$$A(z)y(k) = B(z)u(k) + n(k)$$

$$\left[1 + a_1 z^{-1} + a_2 z^{-2} ... + a_n z^{-n}\right] y(z) = \left[b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_m z^{-m}\right] u(k) + n(k)$$

$$y(k) + a_1 y(k-1) ... + a_n y(k-n) = b_0 u(k) + b_1 u(k-1) + ... + b_m u(k-m) + n(k)$$

And isolating $y(k)$ we will have:

$$y(k) = \left[\underbrace{-a_1 y(k-1) - ... - a_n y(k-n)}_{Auto-regressive\ part}\right] + \left[\underbrace{b_0 u(k) + b_1 u(k-1) + ... + b_m u(k-m)}_{Exogenous\ inputs}\right] + n(k)$$

That's why is called ARX (**A**uto **R**egressive with e**X**ogenous inputs).

We can notice that $[-a_1 y(k-1) - ... - a_n y(k-n)]$ is equal to $[1 - A(z)]$, so we can rewrite the predictive form found in a more compact way:

$$\boxed{y(k) = [1 - A(z)]\, y(k) + B(z)u(k) + n(k)} \quad \text{(ARX Pred)}$$

We can also find this last form starting from the general predictive form, considering $C(z) = A(z)$ and $D(z) = 1$:

$$y(k) = \left[1 - \frac{C(z)}{D(z)}\right] y(k) + \left[\frac{C(z)B(z)}{D(z)A(z)}\right] u(k) + n(k)$$

$$y(k) = [1 - A(z)]\, y(k) + \left[\frac{A(z)B(z)}{A(z)}\right] u(k) + n(k)$$

So, we will have the previously found predictive form:

$$y(k) = [1 - A(z)]\, y(k) + B(z)u(k) + n(k)$$

We can also write it in a third way, considering the following column vectors. The first one is the parameters vector:

$$\boldsymbol{\theta} = [a_1, a_2, ..., a_n, b_0, b_1, ..., b_m]^T$$

The second one is called the **regressor**:

$$\boldsymbol{\varphi}(k) := [-y(k-1), -y(k-2), ..., -y(k-n), u(k), u(k-1), ..., u(k-m),]^T$$

The predictive ARX form can the be rewritten, considering the regressor at time k, in the following way:

$$\boxed{y(k) = \boldsymbol{\varphi}(k)^T \cdot \boldsymbol{\theta} + n(k)} \quad \text{(ARX Pred with } \boldsymbol{\varphi}\text{)}$$

→ ARX is **linear** in $\boldsymbol{\theta}$!

Considering the second case of

$$\epsilon(k) = \begin{cases} n(k) \sim W_N(0, \lambda^2) \\ D(z)n(k) = n(k) + d_1 n(k-1) + ... + d_q n(k-q) \end{cases}$$

→ let's develop the equations for $\epsilon(k) = D(z)n(k)$:

$$A(z)y(k) = B(z)u(k) + D(z)n(k)$$

$$y(k) = \underbrace{[1 - A(z)]y(k)}_{Auto-regressive} + \underbrace{B(z)u(k)}_{Exogenous} + D(z)n(k)$$

$$y(k) = \left[\underbrace{-a_1 y(k-1) - ... - a_n y(k-n)}_{Auto-regressive} \right] + \left[\underbrace{b_0 u(k) + b_1 u(k-1) + ... + b_m u(k-m)}_{Exogenous} \right] +$$

$$+ \left[\underbrace{n(k) + d_1 n(k-1) + ... + d_q n(k-q)}_{Moving\ average} \right]$$

This is the **ARMAX** (AutoRegressive with Moving Average end eXogenous inputs), also called **coloured equation error** because the error is not white.

Starting from the general black-box predicting form:

$$y(k) = \left[z \left(\frac{D(z) - C(z)}{D(z)} \right) \right] y(k-1) + \left[\frac{C(z)B(z)}{D(z)A(z)} \right] u(k) + n(k)$$

the ARMAX ($C = A$) in predictive form we will be:

$$y(k) = \left[1 - \frac{A(z)}{D(z)} y(k) \right] + \frac{B(z)}{D(z)} u(k) + n(k)$$

Output error model, ARX and ARMAX are the three most used models, and we will basically use only them along the course. Sometimes you can hear about **ARMA** models (like for weather forecasting), for cases where there is not a control action (you can't control the weather).

4 Least-squares methods

4.1 Identification of the best model

Let's consider the general Black Box hypothesis for Discrete LTI systems (Hp.1). So for the model, we will have the perfect values for $A(z), B(z), C(z), D(z)$ but not the perfect values for the parameters $\boldsymbol{\theta}$. But we don't know even the dimensions, so we have to, in some way:

- identify the set of the meta-parameters n, m, l, q, the so-called **complexity** p of the model
- identify the parameters $\hat{\boldsymbol{\theta}}_p = \left[\hat{a_1}, \hat{a_2}, ..., \hat{a_n}, \hat{b_0}, \hat{b_1}, ..., \hat{b_m}\right]^T$ whose dimension depends on the complexity p.

We will start with the identifiication of the parameters $\hat{\boldsymbol{\theta}}_p$, as if the meta-parameters where fixed.

4.2 Parameter identification: one-shot solution

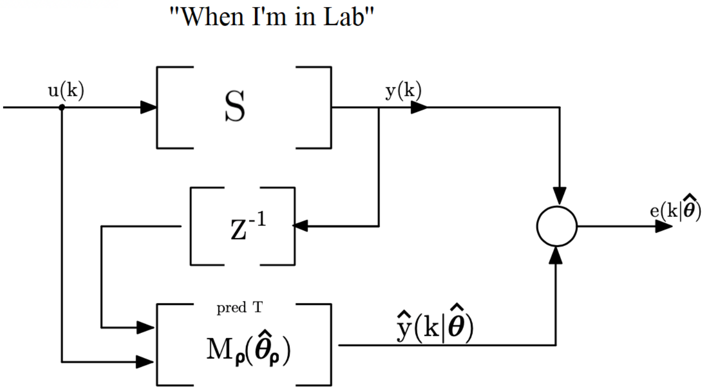
For a given fixed complexity $p = \{n, m, l, q\}$ we will have to identify a certain parameter vector $\hat{\boldsymbol{\theta}}_p^*$ so that:

$$\hat{\boldsymbol{\theta}}_p^* = \{\text{best (w.r.t. the performance index) value for } \hat{\boldsymbol{\theta}}_p\}$$

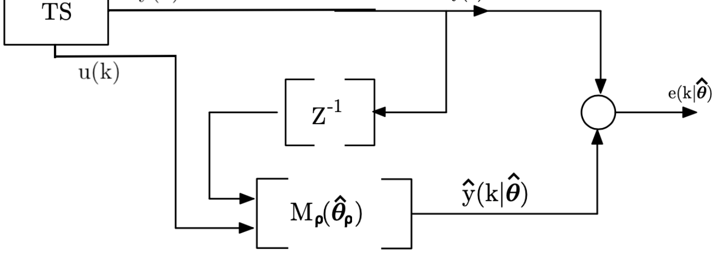
In order to do that we will use the standard choice for performance index, that is the **Least Square Cost**. In order to do that we will need a **dataset** used for the identification process that is called **training set**, that is, the results of the experiments made on the real system:

$$\langle u(k), y(k) \rangle, \qquad k = 0, 1, ..., N$$

So, when we are in laboratory we make experiments and we register the training set data:



Then, when we are not anymore in the laboratory (when "we're home"), when the experiments are done, we have the already registered values $y(k)$:



The Least-squares Cost will be:

$$J_N(\hat{\boldsymbol{\theta}}) = \frac{1}{N}\sum_{k=1}^{N} e^2(k|\hat{\boldsymbol{\theta}})$$

So the identification process will be choose the optimal value $\hat{\boldsymbol{\theta}}^*$ among the possible $\hat{\boldsymbol{\theta}}$ values that will minimize this cost index:

$$\hat{\boldsymbol{\theta}}^* = \arg\min_{\hat{\boldsymbol{\theta}}} J_N(\hat{\boldsymbol{\theta}})$$

We already seen that, for ARX model, $Hp.1$ holds but not $Hp.2$:

$$ARX \quad \rightarrow \quad \hat{y}(k|\hat{\boldsymbol{\theta}}) = [1-\hat{A}(z)]y(k) + \hat{B}(z)u(k) = \boldsymbol{\varphi}(k)^T\boldsymbol{\theta} + n(k)$$

$$ARX\ Hp.1 \quad \rightarrow \quad y(k) = \boldsymbol{\varphi}(k)^T\boldsymbol{\theta} + n(k)$$

$$\boldsymbol{\theta} \rightarrow \hat{\boldsymbol{\theta}} \quad \rightarrow \quad \hat{y}(k|\hat{\boldsymbol{\theta}}) = \boldsymbol{\varphi}(k)^T\boldsymbol{\theta}$$

So we can rewrite the cost function:

$$J_N(\hat{\boldsymbol{\theta}}) = \frac{1}{N}\left[e(1|\hat{\boldsymbol{\theta}})e(1|\hat{\boldsymbol{\theta}}) + e(2|\hat{\boldsymbol{\theta}})e(2|\hat{\boldsymbol{\theta}}) + ... + e(N|\hat{\boldsymbol{\theta}})e(N|\hat{\boldsymbol{\theta}})\right]$$

$$\boldsymbol{e}_N(\hat{\boldsymbol{\theta}}) = \begin{bmatrix} e(1|\hat{\boldsymbol{\theta}}) \\ e(2|\hat{\boldsymbol{\theta}}) \\ \cdots \\ e(N|\hat{\boldsymbol{\theta}}) \end{bmatrix} \rightarrow J_N(\hat{\boldsymbol{\theta}}) = \frac{1}{N}\left[\boldsymbol{e}_N^T(\hat{\boldsymbol{\theta}})\boldsymbol{e}_N(\hat{\boldsymbol{\theta}})\right]$$

So we have:

$$\boldsymbol{e}_N(\hat{\boldsymbol{\theta}}) = \begin{bmatrix} e(1|\hat{\boldsymbol{\theta}}) \\ e(2|\hat{\boldsymbol{\theta}}) \\ \cdots \\ e(N|\hat{\boldsymbol{\theta}}) \end{bmatrix} = \begin{bmatrix} y(1) - \boldsymbol{\varphi}^T(1)\hat{\boldsymbol{\theta}} \\ y(2) - \boldsymbol{\varphi}^T(2)\hat{\boldsymbol{\theta}} \\ \cdots \\ y(N) - \boldsymbol{\varphi}^T(N)\hat{\boldsymbol{\theta}} \end{bmatrix} = \begin{bmatrix} y(1) \\ y(2) \\ \cdots \\ y(N) \end{bmatrix} - \begin{bmatrix} \boldsymbol{\varphi}^T(1)\hat{\boldsymbol{\theta}} \\ \boldsymbol{\varphi}^T(2)\hat{\boldsymbol{\theta}} \\ \cdots \\ \boldsymbol{\varphi}^T(N)\hat{\boldsymbol{\theta}} \end{bmatrix}$$

We can compact the view with two definitions

$$\boldsymbol{y}_N = \begin{bmatrix} y(1) \\ y(2) \\ \cdots \\ y(N) \end{bmatrix} \qquad \boldsymbol{\Phi}_N^T = \begin{bmatrix} \boldsymbol{\varphi}^T(1) \\ \boldsymbol{\varphi}^T(2) \\ \cdots \\ \boldsymbol{\varphi}^T(N) \end{bmatrix} \quad \rightarrow \quad \boldsymbol{e}_N(\hat{\boldsymbol{\theta}}) = \boldsymbol{y}_N - \boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}}$$

The regressor is:

$$\boldsymbol{\varphi}(k) := [-y(k-1), -y(k-2), ..., -y(k-n),\ u(k), u(k-1), ..., u(k-m)]^T$$

$$\dim(\boldsymbol{\varphi}(k)) = (n+m+1) \times 1 = p \times 1$$

So, as well:

$$\dim(\hat{\boldsymbol{\theta}}) = (n+m+1) \times 1 = p \times 1$$

And we can have all the dimensions (**dimension check**)[6]:

$$\boldsymbol{y}_N = \begin{bmatrix} y(1) \\ y(2) \\ \cdots \\ y(N) \end{bmatrix} \qquad \boldsymbol{\Phi}_N^T = \begin{bmatrix} \boldsymbol{\varphi}^T(1) \\ \boldsymbol{\varphi}^T(2) \\ \cdots \\ \boldsymbol{\varphi}^T(N) \end{bmatrix} \quad \rightarrow \quad \underset{N\times 1}{\boldsymbol{e}_N(\hat{\boldsymbol{\theta}})} = \underset{N\times 1}{\boldsymbol{y}_N} - \underset{N\times p}{\boldsymbol{\Phi}_N^T}\ \underset{p\times 1}{\hat{\boldsymbol{\theta}}}$$

Then:

$$J_N(\hat{\boldsymbol{\theta}}) = \frac{1}{N}\left[\boldsymbol{y}_N - \boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}}\right]^T\left[\boldsymbol{y}_N - \boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}}\right] = \frac{1}{N}\left[\boldsymbol{y}_N^T\boldsymbol{y}_N - \hat{\boldsymbol{\theta}}^T\boldsymbol{\Phi}_N\boldsymbol{y}_N - \boldsymbol{y}_N^T\boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}} + \hat{\boldsymbol{\theta}}^T\boldsymbol{\Phi}_N\boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}}\right]$$

[6] So, the actual definition of $\boldsymbol{\Phi}_N$ is

$$\boldsymbol{\Phi}_N = [\boldsymbol{\varphi}(1)\ \ |\boldsymbol{\varphi}(2)\ \ |\cdots\ \ |\boldsymbol{\varphi}(N)] \in \mathbf{R}^{p\times N}$$

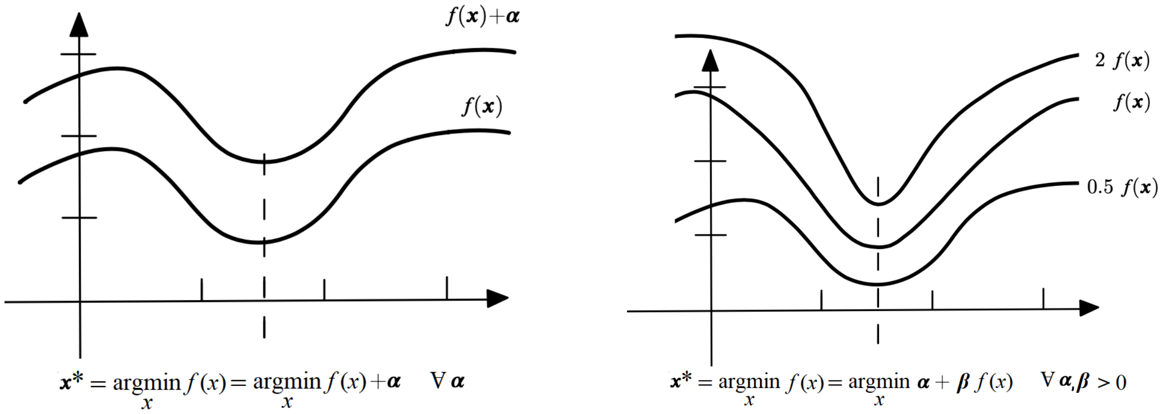
where we used the property $(MN)^T = N^T M^T$.

$\hat{\boldsymbol{\theta}}^T \boldsymbol{\Phi}_N \boldsymbol{y}_N$ and $\boldsymbol{y}_N^T \boldsymbol{\Phi}_N^T \hat{\boldsymbol{\theta}}$ are scalars, hence they are equal, then:

$$J_N(\hat{\boldsymbol{\theta}}) = \frac{1}{N}\boldsymbol{y}_N^T\boldsymbol{y}_N + \frac{2}{N}\left[\frac{1}{2}\hat{\boldsymbol{\theta}}^T\boldsymbol{\Phi}_N\boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}} - \boldsymbol{y}_N^T\boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}}\right]$$

$$\frac{1}{N}\boldsymbol{y}_N^T\boldsymbol{y}_N \rightarrow \text{Constant value not dependent on } \hat{\boldsymbol{\theta}}$$

As we can see from the following graphic examples, we don't need the constant part to compute $\arg\min_{\boldsymbol{\theta}}$, neither the scalar multiplication factor:


$f(\boldsymbol{x}){+}\boldsymbol{\alpha}$
$f(\boldsymbol{x})$
$\boldsymbol{x}^* = \underset{x}{\text{argmin}}\, f(x) = \underset{x}{\text{argmin}}\, f(x) + \boldsymbol{\alpha} \quad \forall\, \boldsymbol{\alpha}$
$2\, f(\boldsymbol{x})$
$f(\boldsymbol{x})$
$0.5\, f(\boldsymbol{x})$
$\boldsymbol{x}^* = \underset{x}{\text{argmin}}\, f(x) = \underset{x}{\text{argmin}}\, \boldsymbol{\alpha} + \boldsymbol{\beta}\, f(x) \quad \forall\, \boldsymbol{\alpha},\boldsymbol{\beta} > 0$

$$\hat{\boldsymbol{\theta}}^* = \arg\min_{\boldsymbol{\theta}}\left[J_N(\hat{\boldsymbol{\theta}})\right] = \arg\min_{\boldsymbol{\theta}}\left[\frac{1}{2}\hat{\boldsymbol{\theta}}^T\boldsymbol{\Phi}_N\boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}} - \boldsymbol{y}_N\boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}}\right]$$

But is it better to have big N or small N? And which p is better to have? **Complexity should be traded off with** N. But first, let's think about N alone, what is it? N is **the cardinality of the dataset**, the number of examples stored. The bigger the data, the better is. That is not the same with complexity: you have to be able to handle it:

$$N \rightarrow \infty \quad \text{Good} \qquad p \rightarrow \infty \quad \text{Too much w.r.t. computation power}$$

Now, let's **minimize the desired function**:

$$f(\boldsymbol{x}) = \frac{1}{2}\hat{\boldsymbol{\theta}}^T\boldsymbol{\Phi}_N\boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}} - \boldsymbol{y}_N\boldsymbol{\Phi}_N^T\hat{\boldsymbol{\theta}}$$

This is a **quadratic form**! In fact:

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T M\boldsymbol{x} - \boldsymbol{C}^T\boldsymbol{x} \qquad f:\mathbb{R}^p \rightarrow \mathbf{R} \quad M \in \mathbb{R}^{p\times p} \quad C \in \mathbb{R}^{p\times 1}$$

So how to minimize a quadratic form?

- Assume $f(\boldsymbol{x}) \in \mathcal{C}^1$ (first order derivable)
- $\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) \Rightarrow \left.\dfrac{\partial}{\partial \boldsymbol{x}} f(\boldsymbol{x})\right|_{\boldsymbol{x}=\boldsymbol{x}^*} = \mathbf{0}$ (condition necessary but not sufficient)

! Be careful! The gradient is a column vector with the same direction of the argument:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(\boldsymbol{x}) \\ \frac{\partial}{\partial x_2} f(\boldsymbol{x}) \\ \cdots \\ \frac{\partial}{\partial x_p} f(\boldsymbol{x}) \end{bmatrix}$$

While the partial derivative is a row vector, not a column:

$$\frac{\partial}{\partial \boldsymbol{x}} f(\boldsymbol{x}) = \left[\frac{\partial}{\partial x_1} f(\boldsymbol{x})\,, \frac{\partial}{\partial x_2} f(\boldsymbol{x})\,, \cdots\,, \frac{\partial}{\partial x_p} f(\boldsymbol{x})\right] = \nabla_{\boldsymbol{x}} f(\boldsymbol{x})^T$$

- In the quadratic form

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T M\boldsymbol{x} - \boldsymbol{C}^T\boldsymbol{x} \qquad f : \mathbb{R}^p \to \mathbf{R} \quad M \in \mathbb{R}^{p\times p}$$

If M is a **positive definite** matrix, the function has a unique minimum at $\boldsymbol{x}^*$:

$$\boxed{\underset{p\times p}{M} > 0} \Leftrightarrow \forall \boldsymbol{x} \in \mathbb{R}^p \text{ s.t } \boldsymbol{x} \neq \mathbf{0} \to \boxed{\boldsymbol{x}^T M\boldsymbol{x} > 0} \Rightarrow \exists!\,\text{min at } \boldsymbol{x}^*$$

- So, let's compute the partial derivative:

$$\frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{C}^T\boldsymbol{x} = \frac{\partial}{\partial \boldsymbol{x}}\left[c_1x_1 + c_2x_2 + ... + c_px_p\right] = [c_1\; c_2 \;...\; c_p] = \boldsymbol{C}^T$$

<u>Note</u>: in the following computation keep in mind that $\boldsymbol{x}^T M = a^T(\boldsymbol{x})$ is a function (because a is defined as $a(\boldsymbol{x}) = M^T\boldsymbol{x}$):

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{x}^T M\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}}[a(\boldsymbol{x})\cdot \boldsymbol{x}] = (\boldsymbol{x}^T M)\cdot 1 + M\cdot(\boldsymbol{x}) = \underset{1\times p}{\boldsymbol{x}^T}\;\underset{p\times p}{M} + \underset{p\times p}{M}\;\underset{p\times 1}{\boldsymbol{x}}$$

But it is wrong at a dimensional check, but we can easily fix it with a transposition:

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{x}^T M\boldsymbol{x}) = \underset{1\times p}{\boldsymbol{x}^T}\;\underset{p\times p}{M} + \underset{1\times p}{\boldsymbol{x}^T}\;\underset{p\times p}{M^T} = \boldsymbol{x}^T(M + M^T)$$

So, if we have a **symmetrical matrix** $M = M^T$ we have:

$$\frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{x}^T M\boldsymbol{x} = 2\boldsymbol{x}^T M$$

And in this case (we will deal with symmetrical matrix, so our case), we have finally:

$$\frac{\partial}{\partial \boldsymbol{x}}\left[\frac{1}{2}\boldsymbol{x}^T M\boldsymbol{x} - \boldsymbol{C}^T\boldsymbol{x}\right] = \boldsymbol{x}^T M - \boldsymbol{C}^T$$

So we have computed the partial derivative for every M-symmetric-matrix quadratic form!

- Now, if M is a symmetrical positive defined matrix ($M = M^T > 0$):

$$\frac{\partial}{\partial \boldsymbol{x}}\left[\frac{1}{2}\boldsymbol{x}^T M\boldsymbol{x} - \boldsymbol{C}^T\boldsymbol{x}\right] = \boldsymbol{x}^T M - \boldsymbol{C}^T = 0 \to \boldsymbol{x}^{T^*} M = \boldsymbol{C}^T$$

- In order to finish we just have to demonstrate the existence of M^{-1}. But we know that:

$$M > 0 \Rightarrow \exists M^{-1}$$

- So we can conclude:

$$\boldsymbol{x}^{T^*} M = \boldsymbol{C}^T \to M^T\boldsymbol{x}^* = \boldsymbol{C} \to M\boldsymbol{x}^* = \boldsymbol{C} \to$$
$$\to \boldsymbol{x}^* = M^{-1}\boldsymbol{C}$$

Now we can apply this to our case **assuming that $\boldsymbol{\Phi}_N\boldsymbol{\Phi}_N^T$ is positive defined**, and this is true if and only if $rank(\boldsymbol{\Phi}_N) = p$ (it is already symmetric):

$$\hat{\boldsymbol{\theta}} \to \boldsymbol{x} \qquad \boldsymbol{\Phi}_N\boldsymbol{\Phi}_N^T \to M \qquad \boldsymbol{y}_N\boldsymbol{\Phi}_N^T \to \boldsymbol{C}$$

$$\Rightarrow \boxed{\hat{\boldsymbol{\theta}}^* = [\boldsymbol{\Phi}_N\boldsymbol{\Phi}_N^T]^{-1}\boldsymbol{\Phi}_N\boldsymbol{y}_N}$$

This is the so-called **one shot solution** to the least squares.

4.3 Parameter identification: dataset

From a training set $\{u(k), y(k)|k = 1, ..., N\}$ we can define a matrix including all the regressors that we need in order to compute the one-shot solution:

$$\boldsymbol{\Phi}_N := [\boldsymbol{\varphi}(1) \mid \boldsymbol{\varphi}(2) \mid ... \mid \boldsymbol{\varphi}(N)] = \begin{bmatrix} -y(0) & | & -y(1) & | & ... & | & -y(N-1) \\ \vdots & | & \vdots & | & ... & | & \vdots \\ -y(1-n) & | & -y(2-n) & | & ... & | & -y(N-n) \\ u(1) & | & u(2) & | & ... & | & u(N) \\ u(0) & | & u(1) & | & ... & | & u(N-1) \\ \vdots & | & \vdots & | & ... & | & \vdots \\ u(1-m) & | & u(2-m) & | & ... & | & u(N-m) \end{bmatrix}$$

But of these quantities the only available values are $y(1), ..., u(1)$ what we can do?

- We can put the other values of the matrix (the "past values") as 0
- We can put "1" such that all the regressors are "full".

  → consider the following example with $n = 3$ and $m = 2$:

  Normally is not a big problem because $n, m \ll N$, so that there will not be too much "past values" w.r.t. all the matrix. If you have $N = 1000$ data, the first $n = 3$ columns are not so influential, they don't have a big weight and so this will not affect the estimation.

Remember that our notation implies this regressor "with the minuses":

$$\boldsymbol{\varphi}(k) := \begin{bmatrix} -y(k-1) \\ -y(k-2) \\ \vdots \\ -y(k-n) \\ \\ u(k) \\ u(k-1) \\ ... \\ u(k-m) \end{bmatrix}$$

(here he made an excursus about what if we used the regressor defined with the pluses instead of the minuses and its effects on ARX and on the parameters)

Recall that the hypotheses on ARX model is that $\hat{u}(k|\hat{\theta}) = \boldsymbol{\varphi}^T(k|\hat{\theta})$ is **quadratic** and **linear in** $\hat{\theta}$.

→ this ***LQ* hypothesis implies** an explicit solution: the **one-shot solution** (see later for the demonstration that *LQ* for ARX model):

$$\text{unique solution} \Leftrightarrow \boldsymbol{\Phi}_N \boldsymbol{\Phi}_N^T > 0 \Leftrightarrow rank(\boldsymbol{\Phi}_N) = p = n + m + 1 \text{ (full-rank matrix)}^{7}$$

Φ_N is a $p \times N$. You should have at least $N \geq p$ data, but normally should be $N \ggg p$, so Φ_N will be "larger than taller", so the maximum rank will be p.

Since $\boldsymbol{\Phi}_N = [\boldsymbol{\varphi}(1) \ \ |\boldsymbol{\varphi}(2) \ \ | \cdots \ \ |\boldsymbol{\varphi}(N)] \in \mathbf{R}^{p \times N}$, we have to find p linearly independent rows or columns, but it is difficult since they are the regressors. Let's consider then the rows:

[7] MM^T is always semi-positively defined (as the square $m^2 \geq 0$ in the scalar case):

$$MM^T \Leftrightarrow \boldsymbol{v}^T(MM^T)\boldsymbol{v} \geq 0 \rightarrow \boldsymbol{v}^T(MM^T)\boldsymbol{v} = \boldsymbol{v}^T M \ M^T \boldsymbol{v} = \boldsymbol{w}^T \boldsymbol{w} = ||\boldsymbol{w}|| \geq 0$$

Moreover $||\boldsymbol{w}|| = 0 \Leftrightarrow \boldsymbol{w} = M^T \boldsymbol{v} = 0$, so that:

$$\boldsymbol{w} = M^T \boldsymbol{v} = 0 \Leftrightarrow \boldsymbol{v} \in Ker(M^T) \qquad Ker(M^T) = \mathbf{0} \Leftrightarrow rank(M) \text{ is } max$$

So, $\Phi_N \Phi_N^T > 0$ (positive defined) if in our case we ask $rank(\boldsymbol{\Phi}_N) = p$

$$\boldsymbol{\Phi}_N = \left[\begin{array}{cccc} -y(0) & -y(1) & \dots & -y(N-1) \\ \vdots & \vdots & \dots & \vdots \\ -y(1-n) & -y(2-n) & \dots & -y(N-n) \\ \hline u(1) & u(2) & \dots & u(N) \\ u(0) & u(1) & \dots & u(N-1) \\ \vdots & \vdots & \dots & \vdots \\ u(1-m) & u(2-m) & \dots & u(N-m) \end{array}\right] = \left[\begin{array}{c} \boldsymbol{\Phi}_y \\ \hline \boldsymbol{\Phi}_u \end{array}\right]$$

A trivial case would be the case of null input: in this case $u(1-m) \dots u(N) = 0 \rightarrow \boldsymbol{\Phi}_u = \mathbf{0}$, and that is obvious for a linear system with null initial condition. Take a coin and a smartphone on a desk: the only way to distinguish then from their input-output behaviour is not to leave them still (null input), but to apply a certain force → applying an input is important!

We can recall that for first order systems we used to make the "step response" analysis in order to fully determine parameters. In this case though, if we have a constant input as a step signal:

- $u(k) = 0 \rightarrow$ No good (null input)
- $u(k) = A \ \forall k \rightarrow$ the matrix $\boldsymbol{\Phi}_N$ is filled with constant values equal to $A \rightarrow rank(\boldsymbol{\Phi}_N) = 1$. In this case, if $n \neq 0$ or $m \neq 0$, we would just have:

  $$y(k) = b_0 u(k) + n(k)$$

  So, in the case of **no dynamics** (pure amplificational system) with a constant input we can detect only one parameter b_0
- $\{u(k)\} = \{-1, 1, -1, 1, \dots\} \rightarrow$ we will have:

  $$\boldsymbol{\Phi}_N = \begin{bmatrix} 0 & 1 & 0 & 1 & \dots \\ 1 & 0 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}$$

  We will have then $rank(\boldsymbol{\Phi}_N) = 1$.

⇓ So, the bigger is the matrix, the more irregular has to be the input. And what is the most irregular input that we know? The **white noise**

- $u(k) = WN \rightarrow$ **persistently exciting input**.

But how it is possible that for a 1st order system like a RC circuit we just need a constant step input in order to identify two parameters $\{\tau, k\}$? Recall that we need the transient to identify τ, not only the regime (that we use to identify K).

We make a strong use of the regime, if we arrive at regime we have a constant input, so the output will be basically white noise around $K \rightarrow$ it will be impossible to identify τ. But when we use $\boldsymbol{\Phi}_N$ we want a big matrix, so N will be big, and the transient will be only in the first samples, so it will have a very low weight on the "averaging process" that is this kind of identification.

In this case, the best thing to do is to "keep shaking the system" with a white noise:

> The first phase will be open loop random control with white noise in order to identify the parameters. Then there will be the closed loop control with a certain designed input.

Inputs, so a sequence $\{u(k)\}$ can allow you to identify (for $N \rightarrow \infty$):

$u(k) = 0 \Rightarrow 0$ parameters	$PE(0)$
$u(k) = 1 \Rightarrow 1$ parameters	$PE(0)$
$\vdots$	$\vdots$
$u(k) = WN \Rightarrow \infty$ parameters	$PE(\infty)$

$PE(k)$ is “Persistently Exciting input of order k”, that will allow you to identify k parameters. Given the following definitions, we can then state the corresponding theorem:

- Probabilistic autocorrelation function: $R_{uu}(\tau) = Exp\{u(k)u(k-\tau)\}$
- Empirical autocorrelation function: $R_{uu}(\tau) = \frac{1}{N}\sum_{k=1}^{\infty} u(k)u(k-\tau)$
- Power Spectrum $S_{uu} = \frac{1}{2\pi}\sum_{\tau=\infty}^{\infty} R_{uu}(\tau)e^{-j\omega\tau}$

Theorem: Order of persistence of excitation (number of parameters that can be identified when $N \to \infty$) corresponds to the values of frequency for which the Power Spectrum $S_{uu}(e^{j\omega}) \neq 0$

In fact, if we recall Fourier analysis of spectrum signals, how many δ in S_{uu} we have for each signal?

null → 0
constant → 1
sin → 2
cos → 2
⋮
$WN \to \infty \quad S_{uu} = \frac{1}{2\pi}\lambda^2$

4.4 Parameter identification for black box models

Summing up what we saw before, given a training set

$$T.S. = \{\ u(k), y(k)\ \mid\ k \in [1, ..., N]\ \}$$

we have

$\hat{\boldsymbol{\theta}}^* = [(\boldsymbol{\Phi}_N\boldsymbol{\Phi}_N^T)^{-1}\boldsymbol{\Phi}_N\boldsymbol{y}_N]$ explicit one-shot solution ⇐ LQ hypothesis
$L \to$ model linear in $\hat{\boldsymbol{\theta}}$
$Q \to$ cost is a quadratic form in the model

Consider the general Black Box case:

$$\hat{y}(k|\hat{\boldsymbol{\theta}}) = \left[z\left(\frac{D(z|\hat{\boldsymbol{\theta}}) - C(z|\hat{\boldsymbol{\theta}})}{D(z|\hat{\boldsymbol{\theta}})}\right)\right]y(k-1) + \left[\frac{C(z|\hat{\boldsymbol{\theta}})B(z|\hat{\boldsymbol{\theta}})}{D(z|\hat{\boldsymbol{\theta}})A(z|\hat{\boldsymbol{\theta}})}\right]u(k)$$

Let’s see if our black box models (OutputError, ARX and ARMAX) respect the LQ hypothesis.

ARX Is the estimation $\hat{y}(k|\hat{\boldsymbol{\theta}})$ linear w.r.t. $\hat{\boldsymbol{\theta}}$?

$\hat{D}(z) = 1$, $\hat{C}(z) = \hat{A}(z)$ → all denominators desappear

"No denominator"

$$\hat{y}(k|\hat{\theta}) = \underbrace{\underbrace{[1 - \hat{A}(z)]\,y(k)}_{} + \underbrace{\hat{B}(z)\,\mu(k)}_{}}_{}$$

linear combination of finite number of the output $y(k-1)$ -- $y(k-N)$

↳ lin. comb. of the input $\mu(k)$ --- $\mu(k-n)$

linear combination of the regressor $\varphi(k)$

↓

in fact we can say : $\hat{y}(k|\hat{\theta}) = \hat{\theta}^T\varphi(k)$

but also <u>linearity w.r.t. $\hat{\theta}$</u>.

Hence, we can use one-shot solution for ARX models.

Output Error Is the estimation $\hat{y}(k|\hat{\boldsymbol{\theta}})$ linear w.r.t. $\hat{\boldsymbol{\theta}}$ for OE models?

O.E: output ERROR $\quad y(k) = \frac{B(z)}{A(z)}\mu(k) + \eta(k)$

Is the O.E. linear in $\underline{\hat{\theta}}$?

$$\hat{y}(k|\underline{\hat{\theta}}) = \frac{\hat{B}(z)}{\hat{A}(z)}\mu(k)$$

At the moment there is a denominator

⇝ division between polinomi ⇝ not linear

We have parameters $(\hat{a}_1 \cdots \hat{a}_n)$ at denominator

⇝ not linear in $\hat{\underline{\theta}}$

$$\text{Assume: } \hat{G}(z) = \frac{\hat{B}(z)}{\hat{A}(z)} = g_0 + \underbrace{g_1 z^{-1} + g_2 z^{-2} \ldots}_{} \text{ infinite}$$

if $A \neq 1$

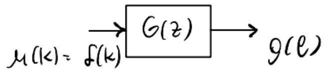
$$= \boxed{\sum_{\ell=0}^{\infty} g_\ell z^{-\ell}}$$

infinite long division

z-transform of the sequence $g_0, g_1, \ldots$

$g(0) = g_0,\quad g(1) = g_1 \quad \ldots \quad g(\ell) = g_\ell$

impulse response



g_0, g_1, g_2 ----- (∞ number)

$$\begin{aligned}\hat{y}(k|\underline{\hat{\theta}}) &= \hat{G}(z)\mu(k) = \\ &= [g_0 + g_1 z^{-1} + g_2 z^{-2} \ldots]\mu(k) = \\ &= g_0\mu(k) + g_1\mu(k-1) + \ldots = \\ &= [g_0, g_1, g_2 \ldots]^T[\mu(k), \mu(k-1) \ldots]\end{aligned}$$

it's again a linear combination in the equivalent parameter vector $[g_0, g_1 \ldots]^T$ that is infinitely long. ⇝ not useful

We <u>don't have linearity wrt $\hat{\underline{\theta}}$</u>.

Hence, there is not an explicit one-shot solution for OE models.

ARMAX For the ARMAX is even easier because it is impossible to reduce it to the needed form:

impossible to write

$$\hat{y}(k|\underline{\theta}) = \hat{\varphi}^T(k)\,\underline{\hat{\theta}}$$

depends on ∞ parameters ⇝ no one shot sol.

In this case $C = A$, $D \neq 1$:

$$\hat{y}(k|\underline{\hat{\theta}}) = \left[1 - \frac{\hat{A}(z)}{\hat{D}(z)}\right] y(k) + \frac{\hat{B}(z)}{\hat{D}(z)} u(k)$$

it has denominator ⇝ dividing by parameters ⇝ $\underline{\text{not linear in } \underline{\hat{\theta}}}$

Conclusions Do we prefer ARX or the output-error model? It depends: in terms of description output-error model is better, in terms of identification of the parameters, the ARX is better because of the one-shot solution (→ very fast and ready-to-use identification procedure, with some warnings like being aware that we have to inverse a matrix, that is not a easy and simple task).

5 Mathematical programming methods

As we saw in the previous section, having $\hat{\boldsymbol{\theta}}^* = \arg\min_{\boldsymbol{\theta}} J(\hat{\boldsymbol{\theta}})$ with ARX (**L**inearity hp.) with Least squared cost (**Q**uadratic Hp.) leads to one-shot solution. But what if the LQ hypothesis does not holds?

Suppose we have a function like:

$$f(\boldsymbol{x}) : \mathbb{R}^p \to \mathbb{R}$$

What are looking for is:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

What if we have multiple minima, like some local minima and an absolute minimum $f(\boldsymbol{x})$? How we can avoid of "falling into the local minimum"?

5.1 First order information exploitation: gradient descent

Consider the initial, tentative solution $\boldsymbol{x}^{(1)}$. For $l = 0, 1, 2, ...$ we have:

$$\boldsymbol{x}^{(l+1)} = \boldsymbol{x}^{(l)} - \alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}^{(l)}}$$

with α that is the slope of the gradient algorithm. A stop condition is missing:

```
if stopping condition hold true → STOP
```

We need two things to decide:

- Initialization
  - a "reasonable" value
- Stopping condition:
  - Max number of iterations or Max computational time
  - $f(\boldsymbol{x}^{(l)}) - f(\boldsymbol{x}^{(l-1)}) < \epsilon_f$
  - $\boldsymbol{x}^{(l)} - \boldsymbol{x}^{(l-1)} < \epsilon_f$
  - $\alpha \nabla_{\boldsymbol{x}} f(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}^l} < \epsilon_\Delta$

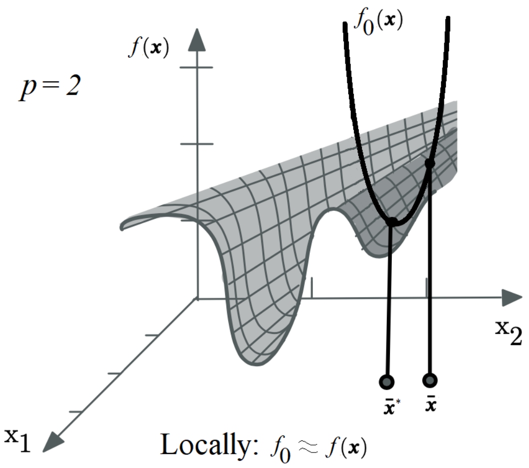
These and the other kind of methods like this one are called **mathematical programming algorithms**, they are different among the others because of the maths they are using, they imply an Hp. on $f(\boldsymbol{x})$ and their depends on the quantities that are available:

Type 0	Random movement		$f(\boldsymbol{x})$	(see Machine Learning course "Optimization" part on Type 0, 1, 1.5 and 2)
Type 1	Gradient algorithms	$\partial/\partial x$	$f(\boldsymbol{x})$	
Type 2	?	$\partial^2/\partial x^2$	$f(\boldsymbol{x})$	

5.2 Second order information exploitation: Newton-Raphson algorithm

How to exploit second order information? Assume $f(\boldsymbol{x}) \in \mathcal{C}^2$. At $\boldsymbol{x} = \bar{\boldsymbol{x}}$, so, a local approximation of the function would be this quadratic approximation $f_q(\boldsymbol{x})$ as we can see from the Taylor expansion and the following plotted example:

$$f(\boldsymbol{x}) \simeq f(\bar{\boldsymbol{x}}) + \frac{\partial}{\partial x} f(\boldsymbol{x})\bigg|_{\boldsymbol{x}=\bar{\boldsymbol{x}}} (\boldsymbol{x} - \bar{\boldsymbol{x}}) + \frac{1}{2}(\boldsymbol{x} - \bar{\boldsymbol{x}})^T \frac{\partial^2}{\partial x^2} f(\boldsymbol{x})\bigg|_{\boldsymbol{x}=\bar{\boldsymbol{x}}} (\boldsymbol{x} - \bar{\boldsymbol{x}})$$



But what if the actual 3D function would be exactly the quadratic function that we called "local approximation"? We would have an explicit exact solution (the **one-shot solution again!**):

$$f(\boldsymbol{x}) \simeq f(\bar{\boldsymbol{x}}) + \boldsymbol{C}^T(\boldsymbol{x}-\bar{\boldsymbol{x}}) + \frac{1}{2}(\boldsymbol{x}-\bar{\boldsymbol{x}})^T M(\boldsymbol{x}-\bar{\boldsymbol{x}}) \qquad if \;\; M > 0 \Rightarrow \bar{\boldsymbol{x}}^* = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) = M^{-1}\boldsymbol{C}.$$

If we're not in this case, we will minimize $f_{\bar{\boldsymbol{x}}}(\boldsymbol{x}-\bar{\boldsymbol{x}})$ (problems here and later with the inversion, what is this and why does it have the inversion $^{-1}$?):

$$\boldsymbol{z}^* = (\boldsymbol{x}-\bar{\boldsymbol{x}})^* = (\bar{\boldsymbol{x}}-\boldsymbol{x}) = \left[\left.\frac{\partial^2 f(\boldsymbol{x})}{\partial x^2}\right|_{\boldsymbol{x}=\bar{\boldsymbol{x}}} \right]^{-1} \left[\left.\frac{\partial f(\boldsymbol{x})}{\partial x}\right|_{\boldsymbol{x}=\bar{\boldsymbol{x}}} \right]^T$$

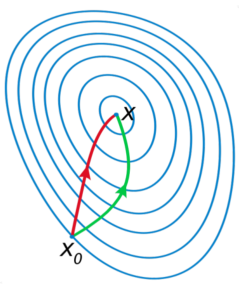
$$\bar{\boldsymbol{x}}^* = \bar{\boldsymbol{x}} - H(\bar{x})\, \nabla_{\boldsymbol{x}} f(\boldsymbol{x})|_{\boldsymbol{x}=\bar{\boldsymbol{x}}}$$

At $\boldsymbol{x}^{(l)}$ we will have:

$$\boldsymbol{x}^{(l+1)} = \boldsymbol{x}^{(l)} - \left[\left.\frac{\partial^2 f(\boldsymbol{x})}{\partial x^2}\right|_{\boldsymbol{x}=\boldsymbol{x}^{(l)}} \right]^{-1} \left[\left.\frac{\partial f(\boldsymbol{x})}{\partial x}\right|_{\boldsymbol{x}=\boldsymbol{x}^{(l)}} \right]^T$$

This, in optimization, is the **Newton's method** generalized for higher dimensions[8]

Here in figure we see a comparison of gradient descent (green) and Newton's method (red) for minimizing a function (with small step sizes). Newton's method uses curvature information (i.e. the second derivative) to take a more direct route:



But this is not complete. In fact here, we're assuming $H(\bar{\boldsymbol{x}}) > 0$, but this is not always true, and without it we can't have a one-shot solution. What we have seen is that in this context is that when we have a function $f(\boldsymbol{x})$, we have:

$$\boldsymbol{x}^{(l+1)} = \boldsymbol{x}^{(l)} - [H(x^{(l)})]^{-1}\, \nabla f(\boldsymbol{x})|_{\boldsymbol{x}^{(l)}}$$

[8] In one dimension, Newton's method performs the iteration

$$x_{k+1} = x_k + t = x_k - \frac{f'(x_k)}{f''(x_k)}$$

The hypothesis $H(\boldsymbol{x}^{(l)}) > 0$ means that the **local approximation function is a convex function**. But what if this **does not hold**?

$$\text{if } H(\boldsymbol{x}^{(l)}) > 0 \qquad \text{then } \boldsymbol{x}^{(l+1)} = \boldsymbol{x}^{(l)} - [H(x^{(l)})]^{-1} \left.\nabla_{\boldsymbol{x}} f(\boldsymbol{x})\right|_{\boldsymbol{x}^{(l)}}$$
$$\text{else } \boldsymbol{x}^{(l+1)} = \boldsymbol{x}^{(l)} - \alpha^{(l)} \left.\nabla_{\boldsymbol{x}} f(\boldsymbol{x})\right|_{\boldsymbol{x}^{(l)}}$$

So, instead of using the Hessian matrix, when this is **not invertible** we use an identity matrix. More in general, I can modify, **perturb the Hessian matrix** in some way M^l such that $\left[H(\boldsymbol{x}^{(l)}) + M^l\right] > 0$:

$$\boldsymbol{x}^{(l+1)} = \boldsymbol{x}^{(l)} - \alpha^{(l)} \left[H(\boldsymbol{x}^{(l)}) + M^l\right] \left.\nabla_{\boldsymbol{x}} f(\boldsymbol{x})\right|_{\boldsymbol{x}^{(l)}}$$

For example, the **descent algorithm** we saw can be seen a specific case of this algorithm, using:

$$M^{(l)} = I - H(\boldsymbol{x}^{(l)})$$

To avoid the $H(\bar{\boldsymbol{x}}) \leq 0$ issue, we can use:

$$M^{(l)} = \begin{cases} 0 & \text{if } H(\boldsymbol{x}^{(l)}) > 0 \\ I - H(\boldsymbol{x}^{(l)}) & \text{else} \end{cases}$$

Another possibility can be:

$$M^{(l)} = \delta^{(l)} \qquad \delta^{(l)} \in \mathbb{R} \qquad \text{s.t.} \quad H(\boldsymbol{x}^{(l)}) + \delta^{(l)} > 0 \rightarrow \ \text{(it always exists!)}$$

$$\boxed{H(\boldsymbol{x}^{(l)}) + \delta^{(l)} > 0}$$

Notice that ,in the last case:

$$\text{for } \delta \rightarrow \infty \qquad \frac{1}{\delta^{(l)}}[H + M] \text{ is the identity matrix}$$

Note also that, if $H(\boldsymbol{x}^{(l)}) > 0$ then:

$$\forall \delta \qquad [H(\boldsymbol{x}^{(l)}) + \delta^{(l)}] > 0$$

We can demonstrate it in the following way:

$$H \geq 0 \quad \boldsymbol{x}^T H \boldsymbol{x} \geq 0 \qquad M > 0 \quad \boldsymbol{x}^T M \boldsymbol{x} > 0 \quad x \neq 0$$
$$\boldsymbol{x}^T [H + M] \boldsymbol{x} > 0$$

Hence the algorithm, at step l:

- Compute $H(\boldsymbol{x}^{(l)})$
- Choose a tentative value for $\delta^{(l)}$
```
    if $H(\boldsymbol{x}^{(l)}) + \delta^{(l)} I > 0$ ok
    else increase
```

This method does not save us from local minima → simulated annealing and other stochastic minimization algorithms are used to avoid it (but in these cases yes, you'll find the general minima with probability 1, but it can takes infinite time: you don't have guaranteed that you'll reach it in a finite time).

5.3 Parameter identification

Given the already-seen Least-squares Cost $J_N(\hat{\boldsymbol{\theta}}_p)$, we want to find the values $\hat{\boldsymbol{\theta}}_p$ such that:

$$J_N \leftarrow f \qquad \hat{\boldsymbol{\theta}} \leftarrow \boldsymbol{x} \qquad \hat{\boldsymbol{\theta}}^* \leftarrow \boldsymbol{x}^*$$
$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J_N(\hat{\boldsymbol{\theta}})$$

Hence, with a given training set:

$$\{u(k), y(k) | k = 1, ..., N\}$$

we have:

$$J_N(\hat{\boldsymbol{\theta}}) = \frac{1}{N}\sum_{k=0}^{N} e^2(k|\hat{\boldsymbol{\theta}}) = \frac{1}{N}\sum_{k=0}^{N}[y(k) - \hat{y}(k|\hat{\theta})]^2 = \frac{1}{N}\sum_{k=0}^{N} e(k|\hat{\theta})^2$$

So, applying the general mathematical programming algorithm we just introduced:

$$\hat{\boldsymbol{\theta}}^{(l+1)} = \hat{\boldsymbol{\theta}}^{(l)} - \alpha^{(l)} \left[\left.\frac{\partial^2 J_N(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} + M^l \right]^{-1} \left[\left.\frac{\partial J_N(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} \right]$$

The conditions that we use to make consistency checks are the dimensions of the matrices/vectors:

$$\frac{\partial J_N(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}} = \frac{1}{N}\sum_{k=0}^{N} 2e(k|\hat{\boldsymbol{\theta}})\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}$$

$\frac{\partial J_N(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}$ is a row vector $(1 \times p)$:

$$\left[\frac{\partial J_N(\hat{\boldsymbol{\theta}})}{\partial \hat{\theta}_1}, \ \frac{\partial J_N(\hat{\boldsymbol{\theta}})}{\partial \hat{\theta}_2}, \ ..., \ \frac{\partial J_N(\hat{\boldsymbol{\theta}})}{\partial \hat{\theta}_p} \right]$$

$e(k|\hat{\theta})$ is a scalar and $\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}$ is a $(1 \times p)$.
So we have:

$$\frac{\partial J_N^2(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2} = \frac{\partial}{\partial \hat{\boldsymbol{\theta}}}\frac{\partial}{\partial \hat{\boldsymbol{\theta}}} J_N(\hat{\boldsymbol{\theta}}) = \frac{2}{N}\sum_{k=0}^{N}\left[e(k|\hat{\boldsymbol{\theta}})\frac{\partial^2}{\partial \hat{\boldsymbol{\theta}}^2} e(k|\hat{\boldsymbol{\theta}}) + \frac{\partial}{\partial \hat{\boldsymbol{\theta}}} e(k|\hat{\boldsymbol{\theta}})\frac{\partial}{\partial \hat{\boldsymbol{\theta}}} e(k|\hat{\boldsymbol{\theta}}) \right]$$

Another dimensional check:

$\frac{\partial J_N^2(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2}$ is a $(p \times p)$ matrix as well as $\frac{\partial^2 e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2}$, while $e(k|\hat{\boldsymbol{\theta}})$ is still a scalar

$\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}$ is a $(1 \times p)$, so $\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}$ is **not consistent**! We have to transpose it

If we transpose the second one we have a $(1 \times p) \cdot (p \times 1) = (1 \times 1)$, and it is still **not correct** (careful! MATLAB will not detect this error!). If we transpose the first one we have a $(p \times 1) \cdot (1 \times p) = (p \times p)$, and **now is correct**:

$$\boxed{\frac{\partial J_N^2(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2} = \frac{2}{N}\sum_{k=0}^{N}\left[e(k|\hat{\boldsymbol{\theta}})\frac{\partial^2 e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2} + \frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}^T \frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}} \right]}$$

So, what we need?

$$e(k|\hat{\boldsymbol{\theta}}) \qquad \frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}} \qquad \frac{\partial^2 e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2} \qquad \forall k \in [1, N]$$

In the identification process, $e(k|\hat{\boldsymbol{\theta}})$ is easy to compute, and $\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}$ is quite easy to compute as well.

The problem arises with $\frac{\partial^2 e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2}$, that is not so easy to do. Can we not compute it? If we need another reason to not do it, we can notice that $\frac{\partial J_N^2(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2}$ is the Hessian matrix. Then, if we consider only the second term of the sum

$$\frac{\partial J_N^2(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2} = \frac{2}{N}\sum_{k=0}^{N}\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}^T \frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}$$

and we define a vector $\boldsymbol{v} = \frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}^T$ that has $\dim(\boldsymbol{v}) = (p \times 1)$, we will have a matrix in the form

$$\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}} \frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}^T = \boldsymbol{v} \cdot \boldsymbol{v}^T \geq 0 \quad \text{(is \textbf{always semi-positive defined}!)}$$

Then:

$$\forall \hat{\boldsymbol{\theta}} \neq 0 \qquad \left[\hat{\boldsymbol{\theta}} \boldsymbol{v}(k)\right] \left[\boldsymbol{v}(k)^T \hat{\boldsymbol{\theta}}\right] \geq 0 \qquad\qquad \frac{2}{N} \sum_{k=0}^{N} \frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}^T \frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}} \geq 0$$

We can rewrite then the general mathematical programming algorithm as follows:

$$\hat{\boldsymbol{\theta}}^{(l+1)} = \hat{\boldsymbol{\theta}}^{(l)} - \alpha^{(l)} \left[\left.\frac{\partial^2 J_N(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} + M^l \right]^{-1} \left[\left.\frac{\partial J_N(\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} \right] \longrightarrow$$

$$\longrightarrow \hat{\boldsymbol{\theta}}^{(l+1)} = \hat{\boldsymbol{\theta}}^{(l)} - \alpha^{(l)} \underbrace{\left[\frac{2}{N} \sum_{k=0}^{N} \left.\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}^T\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} \left.\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} + \delta^l I \right]^{-1}}_{\forall \delta^{(l)} \text{is positive defined } (>0)} \left[\frac{1}{N} \sum_{k=0}^{N} e(k|\hat{\boldsymbol{\theta}}) \left.\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} \right]$$

In this case then we **perturbed** our Hessian matrix in the following way:

$$H(\hat{\boldsymbol{\theta}})^{(l)} + M^{(l)} = H(\hat{\boldsymbol{\theta}})^{(l)} - \frac{2}{N} \sum_{k=0}^{N} e(k|\hat{\boldsymbol{\theta}}) \left.\frac{\partial^2 e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{\boldsymbol{\theta}}^2}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} + \delta^l I$$

But we don't have to compute it! **The second order partial derivative is in the perturbation because we don't want to compute it!**

So, we have to:

- Choose a tentative value for $\hat{\boldsymbol{\theta}}^{(0)}$
- $l = (0, 1, 2, .. \to *)$ `stop`

  `if` *max* iterations reached $l = L$

  `or` $||\ \hat{\boldsymbol{\theta}}^{(l+1)} - \hat{\boldsymbol{\theta}}^{(l)}\ || < \epsilon_{\boldsymbol{\theta}}$

  `or` $|\ J_N(\hat{\boldsymbol{\theta}}^{(l+1)}) - J_N(\hat{\boldsymbol{\theta}}^{(l)})\ | < \epsilon_J$
- Choose $\alpha^{(l)}$

  $\alpha^{(l)} = 1$ (you don't use it at all)

  $\alpha^{(l)} = const. < 1$ you are more cautious as lower is the value → α small is too slow

  $\alpha^{(l)} = const. > 1$ you are more fast as higher is the value → α big leads to too many jumps

  another possibility is to use two scalars in order to have a dynamic behaviour, reducing time by time the effect:

  $$\alpha^{(l)} = \frac{c_1}{c_2 + l}$$

  be careful about the tuning of the two constants c_1, c_2, they don't have to be too high w.r.t. the l values, if not it would behave just as a constant value.

5.4 Parameter identification for ARMAX

Let's recall now the ARMAX model (general case when $C(z) = A(z)$) in input/output form:

$$y(k) = \frac{B(z)}{A(z)} u(k) + \frac{D(z)}{A(z)} n(k)$$

and in predictive form:

$$y(k) = \left[1 - \frac{A(z)}{D(z)} y(k)\right] + \frac{B(z)}{D(z)} u(k) + n(k)$$

$$\hat{y}(k|\hat{\boldsymbol{\theta}}) = \left[1 - \frac{\hat{A}(z)}{\hat{D}(z)} y(k)\right] + \frac{\hat{B}(z)}{\hat{D}(z)} u(k)$$

it allows us to predict a value of $y(\bar{k})$ relying on $y(k < \bar{k})$ and $u(k \leq \bar{k})$.

As we saw, what we need for the l-th iteration:

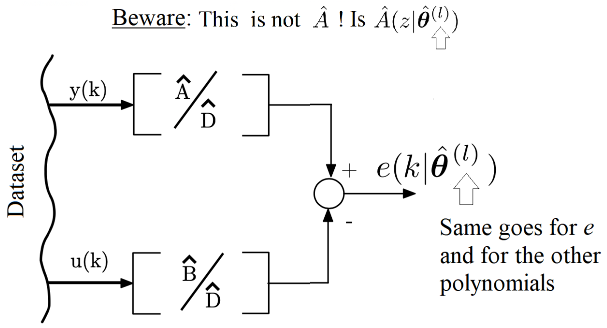
$$e(k|\hat{\boldsymbol{\theta}}^{(l)}) \in \mathbb{R} \quad \text{with } k \in [1, k], \qquad \frac{\partial e(k|\hat{\boldsymbol{\theta}}^{(l)})}{\partial \hat{\boldsymbol{\theta}}} \in \mathbb{R}^{(\not{k} \times 1)} \quad \text{with } k \in [1, k]$$

$$\hat{\boldsymbol{\theta}}^{(l)} = \left[\hat{a_1}^{(l)}, ..., \hat{a_n}^{(l)} \ , \ \hat{b_0}^{(l)}, ..., \hat{b_n}^{(l)} \ , \ \hat{d_1}^{(l)}, ..., \hat{d_n}^{(l)}\right]^T \qquad \dim(\hat{\boldsymbol{\theta}}^{(l)}) = p = n + (1+m) + q$$

From a $\hat{\boldsymbol{\theta}}^{(0)}$ available we compute the steps $e(k|\hat{\boldsymbol{\theta}}^{(0)})$ and $\left.\frac{\partial e(k|\hat{\boldsymbol{\theta}}^{(l)})}{\partial \hat{\boldsymbol{\theta}}^{(l)}}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(0)}}$ and so on.

$$e(k|\hat{\boldsymbol{\theta}}^{(l)}) := y(k) - \hat{y}(k|\hat{\boldsymbol{\theta}}^{(l)}) = \frac{\hat{A}(z)}{\hat{D}(z)} y(k) - \frac{\hat{B}(z)}{\hat{D}(z)} u(k)$$

We are hence in this situation:

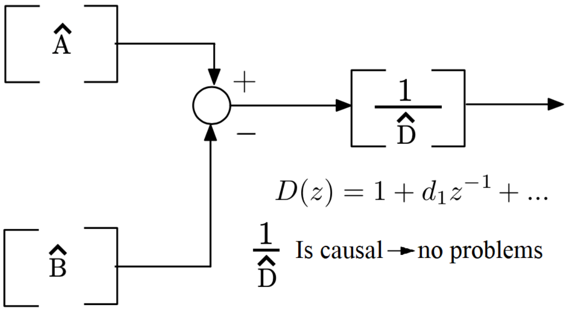


Beware: This is not $\hat{A}$! Is $\hat{A}(z|\hat{\boldsymbol{\theta}}^{(l)})$

$e(k|\hat{\boldsymbol{\theta}}^{(l)})$

Same goes for e and for the other polynomials

As we can see from the figure, we need a dataset in order to have $y(k)$ and $u(k)$ values. We can move $\hat{D}$ "outside":



$D(z) = 1 + d_1 z^{-1} + ...$

$\frac{1}{\hat{D}}$ Is causal → no problems

$$\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)}) \ e(k|\hat{\boldsymbol{\theta}}^{(l)}) = \hat{A}(z|\hat{\boldsymbol{\theta}}^{(l)}) y(k|\hat{\boldsymbol{\theta}}^{(l)}) - \hat{B}(z|\hat{\boldsymbol{\theta}}^{(l)}) u(k|\hat{\boldsymbol{\theta}}^{(l)})$$

We have then:

$$\frac{\partial e(k|\hat{\boldsymbol{\theta}}^{(l)})}{\partial \hat{\boldsymbol{\theta}}^{(l)}} = \left[\frac{\partial e(k|\hat{\boldsymbol{\theta}}^{(l)})}{\partial \hat{a_1}}, \ ... \ \frac{\partial e(k|\hat{\boldsymbol{\theta}}^{(l)})}{\partial \hat{a_n}}, \ ... \ ... \ \frac{\partial e(k|\hat{\boldsymbol{\theta}}^{(l)})}{\partial \hat{d_q}}\right]$$

We have then to compute:

$$\boxed{\frac{\partial e(k|\hat{\boldsymbol{\theta}}^{(l)})}{\partial \hat{a_i}} =? \qquad \frac{\partial e(k|\hat{\boldsymbol{\theta}}^{(l)})}{\partial \hat{b_i}} =? \qquad \frac{\partial e(k|\hat{\boldsymbol{\theta}}^{(l)})}{\partial \hat{d_i}} =?}$$

How much $\hat{B}$ changes if i change ${\hat{a}_i}^{(l)}$? (In the following, we will imply the $^{(l)}$ notation where not explicit) It does not, so the partial derivative w.r.t. a_i is 0. $\hat{A}$ changes instead, so:

$$\hat{A}(z)y(k|\hat{\boldsymbol{\theta}}^{(l)}) = \left[1 + \hat{a_1}z^{-1} + ... + \hat{a_n}z^{-n}\right] y(k) = y(k) + \hat{a_1}y(k-1) + ... + \hat{a_n}y(k-n)$$

$$\frac{\partial}{\partial \hat{a_i}}\left[\hat{A}(z|\hat{\boldsymbol{\theta}}^{(l)})y(k)\right] = y(k-i) = z^{-i}\ y(k)$$

The $y(k)$ term is written in our dataset, it is not a prediction! So, we don't have to pass to the time domain, we can directly write (and then ⇒ for the other matrices):

$$\frac{\partial}{\partial \hat{a_i}}\left[\hat{A}(z|\hat{\boldsymbol{\theta}}^{(l)})\right] = z^{-i} \qquad \Rightarrow \qquad \left(\frac{\partial}{\partial \hat{b_i}}\left[\hat{B}(z|\hat{\boldsymbol{\theta}}^{(l)})\right] = z^{-i} \qquad \frac{\partial}{\partial \hat{d_i}}\left[\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})\right] = z^{-i}\right)$$

How much $\hat{D}$ changes if i change $\hat{a}_i$? It does not, so only A changes are interested by the derivative:

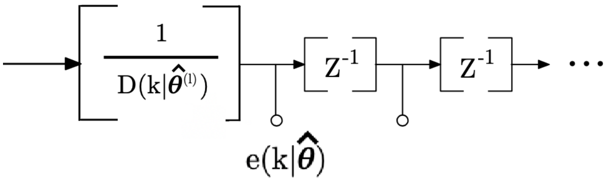
$$\frac{\partial}{\partial \hat{a_i}}\left[\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})\ e(k|\hat{\boldsymbol{\theta}}^{(l)})\right] = \frac{\partial}{\partial \hat{a_i}}\left[\hat{A}(z|\hat{\boldsymbol{\theta}}^{(l)})y(k|\hat{\boldsymbol{\theta}}^{(l)}) - \hat{B}(z|\hat{\boldsymbol{\theta}}^{(l)})u(k|\hat{\boldsymbol{\theta}}^{(l)})\right]$$

$$\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})\frac{\partial}{\partial \hat{a_i}}e(k|\hat{\boldsymbol{\theta}}^{(l)}) = z^{-i}y(k) - 0$$

So:

$$\boxed{\left.\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{a_i}}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} = \frac{z^{-i}}{\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})}y(k), \quad i = 1, ..., n}$$

We can then notice that we can have the following block representation:



Let's compute now the $\partial \hat{b_i}$ part:

$$\frac{\partial}{\partial \hat{b_i}}\left[\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})\ e(k|\hat{\boldsymbol{\theta}}^{(l)})\right] = \frac{\partial}{\partial \hat{b_i}}\left[\hat{A}(z|\hat{\boldsymbol{\theta}}^{(l)})y(k|\hat{\boldsymbol{\theta}}^{(l)}) - \hat{B}(z|\hat{\boldsymbol{\theta}}^{(l)})u(k|\hat{\boldsymbol{\theta}}^{(l)})\right]$$

$$\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})\frac{\partial}{\partial \hat{b_i}}e(k|\hat{\boldsymbol{\theta}}^{(l)}) = 0 + z^{-i}u(k)$$

So:

$$\boxed{\left.\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{b_i}}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} = \frac{-z^{-i}}{\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})}u(k), \quad i = 0, 1, ..., m} \leftarrow \left(\begin{matrix}\text{Be} \\ \text{aware} \\ \text{of } \mathbf{0}!\end{matrix}\right)$$

Let's compute now the $\partial \hat{d_i}$ part:

$$\frac{\partial}{\partial \hat{d_i}}\left[\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})\ e(k|\hat{\boldsymbol{\theta}}^{(l)})\right] = \frac{\partial}{\partial \hat{d_i}}\left[\hat{A}(z|\hat{\boldsymbol{\theta}}^{(l)})y(k|\hat{\boldsymbol{\theta}}^{(l)}) - \hat{B}(z|\hat{\boldsymbol{\theta}}^{(l)})u(k|\hat{\boldsymbol{\theta}}^{(l)})\right]$$

$$\frac{\partial}{\partial \hat{d_i}}\left[\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})\right]e(k|\hat{\boldsymbol{\theta}}^{(l)}) + \hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})\frac{\partial}{\partial \hat{d_i}}\left[e(k|\hat{\boldsymbol{\theta}}^{(l)})\right] = 0$$

$$z^{-i}e(k|\hat{\boldsymbol{\theta}}^{(l)}) + D(z|\hat{\boldsymbol{\theta}}^{(l)})\frac{\partial}{\partial \hat{d_i}}\left[e(k|\hat{\boldsymbol{\theta}}^{(l)})\right] = 0$$

So:

$$\boxed{\left.\frac{\partial e(k|\hat{\boldsymbol{\theta}})}{\partial \hat{d_i}}\right|_{\hat{\boldsymbol{\theta}}=\hat{\boldsymbol{\theta}}^{(l)}} = \frac{-z^{-i}}{\hat{D}(z|\hat{\boldsymbol{\theta}}^{(l)})}e(k|\hat{\boldsymbol{\theta}}^{(l)}), \quad i = 1, ..., q}$$

5.5 Parameter Identification for Output Error

... do it as a homework ...

6 Model Validation

In the previous sections we saw the *offline parameter identification*, that is, given a fixed complexity and a fixed dataset → identification of $\hat{\boldsymbol{\theta}}$. Now, let's see how to **identify the metaparameters**, that is, the complexity of the system, and validate it.

6.1 Complexity identification of n, m for ARX

Talking about the ARX, n, m are the complexity metaparameters: when you choose them, you "freeze" the complexity of the model. In fact, we use the following notation:

$$ARX_{n,m} \qquad \text{(e.g. } ARX_{4,3} \to p = \dim(\boldsymbol{\theta}) = n + m + 1 = 8)$$

Notice that $ARX_{4,3}$ and $ARX_{3,4}$ shares the same p but they have very different dynamics. Sometimes we refer to p as the "complexity", but this is not completely correct: the n, m numbers are the complexity: the **real complexity** is the **vector of metaparameters**, not just p.

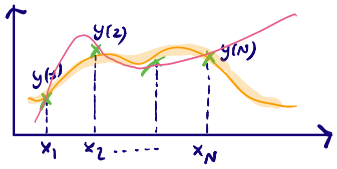
$ARX_{0,0}$	$p = 1$
$ARX_{0,1}$ $ARX_{1,0}$	$p = 2$
$ARX_{1,1}$ $ARX_{0,2}$ $ARX_{2,1}$	$p = 3$
...	...

It is better to have a fixed increment method in order to **not** consider **all the possible values**. That is why **sometimes** it is better to refer to p to reduce the cases, and use this **conventional limitation**:

$p = 1$	$ARX_{0,0}$	
$p = 2$	$ARX_{1,0}$	$(n \uparrow)$
$p = 3$	$ARX_{1,1}$	$(m \uparrow)$
$p = 4$	$ARX_{2,1}$	$(n \uparrow)$
$p = 5$	$ARX_{2,2}$	$(m \uparrow)$
$p = 6$	$ARX_{3,2}$	$(n \uparrow)$
...	...	

That is why in the future we will refer as p as the "complexity" of the model. Now, how to choose p?

- how we choose it
- is there an optimal value of p?



$$y(k) = f(x_k) + \eta$$

- original function
- my measurations

$f = ?$ regression/interpolation problem

Polynomial approximation

$$g(x|\theta) = g_0 + g_1 x + g_2 x^2 + \dots + g_p x^p$$

degree of the polinomial = complexity of pol.

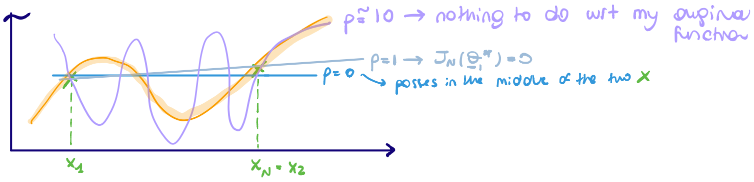
$\underline{\theta}_p = [g_0 \dots g_p]^T$ we want to find $\underline{\theta}_p^*$

s.t. $\underline{\theta}_p^* = \text{argmin } J_N(\underline{\theta}_p)$

$$J_N(\underline{\theta}_p) = \frac{1}{N} \sum \left[y(k) - g(x_k | \underline{\theta}_p) \right]^2$$ least-square cost

↳ p just to make clear that I'm using a θ of dimension p

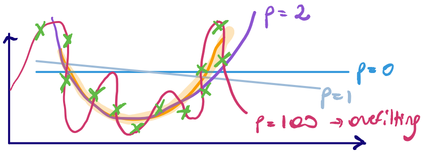
As p increase → $J_N(\underline{\theta}_p^*)$ decrease



$p \tilde{=} 10$ → nothing to do wrt my original function
$p=1$ → $J_N(\underline{\theta}_1^*) = 0$
$p=0$ → passes in the middle of the two ✗
x_1
$x_N = x_2$

→ $N = 2$

If p is too big wrt N → <u>overfitting problem</u>

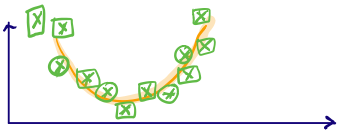


Consider that we don't know how ● is. In this case it's a parabola so the best value of p would be 2 but we don't know that is a parabola, what we know are ✗

It passes trough all the points, but am I happy?

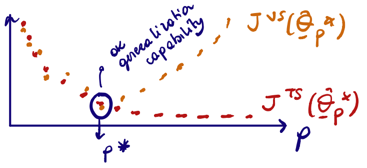
What can I do is divide the ✗ in two sets:

$\begin{cases} \square \text{ training set} \\ \bigcirc \text{ validation set} \end{cases}$



$$\underline{\hat{\theta}}_p^* = \underset{\underline{\hat{\theta}}_p}{\operatorname{argmin}} \; J_N^{TS}(\underline{\hat{\theta}}_p)$$

$$J_N^{TS}(\underline{\hat{\theta}}_p) = \frac{1}{N_{TS}} \sum_{k \in TS} [y(k) - \hat{y}(k \mid \underline{\hat{\theta}}_p)]^2$$



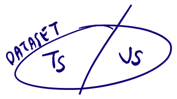
What I have to do is:

- Optimize $\underline{\hat{\theta}}$ over TS $\rightarrow \underline{\hat{\theta}}_p^*$
- Compute the cost over the validation set

$$J^{VS}(\underline{\hat{\theta}}_p^*) = \frac{1}{N_{VS}} \sum_{k \in VS} [y(k) - \hat{y}(k \mid \underline{\hat{\theta}}_p^*)]^2$$

Algorithm

1) Data set cardinality = N $\nearrow$ TS N_{TS} $\searrow$ VS N_{VS}



TS and VS must be independent

2) for $p = 1 \dots N_{TS}$ $\longrightarrow$ I will not use more parameters than examples

$$\underline{\hat{\theta}}_p^* = \underset{\underline{\hat{\theta}}_p}{\operatorname{argmin}} \; J^{TS}(\underline{\hat{\theta}}_p) \rightarrow \underline{\hat{\theta}}_p^* = (\Phi_p \Phi_p^T)^{-1} \Phi_p \, y_{N_{TS}}$$

One shot solution (in ARMAX case instead I have to do a complete minimization problem)

$$J_p^{VS} = J^{VS}(\underline{\hat{\theta}}_p^*)$$

end

$$\boxed{p^* = \underset{p}{\operatorname{argmin}} \; J_p^{VS}}$$

$$\rightsquigarrow \; p^* = \underset{p = 1 \dots N_{TS}}{\operatorname{argmin}} \; J^{VS}\left[\underset{\underline{\hat{\theta}}_p}{\operatorname{argmin}} \; J^{TS}(\underline{\hat{\theta}}_p)\right]$$

Final Results of my optimization:

$$(\underline{\hat{\theta}}_{p^*}^*, \; p^*)$$

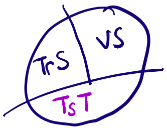
parameters meta-parameters

How good is my optimilation?

$$J^{test} = \frac{1}{N} \sum_{k \in \text{test set}} \left[y(k) - \hat{y}(k \mid \hat{\underline{\theta}}_p *) \right]^{p}$$

but which values of k I have to use?

I can't use k of TS and of VS because I used both for training and for validation → I need a third set test set



Special cases:

1) I have p^*, it's given.

I have a data set like this

Which part of the data set I use to obtain $\hat{\underline{\theta}}_p^*$? I use all ●. ⇝ I obtain $\hat{\underline{\theta}}_{p^*}^{**}$

$$\hat{\underline{\theta}}_{p^*}^{**} = \underset{\hat{\underline{\theta}}_{p^*}^{*}}{\operatorname{argmin}} \; J^{TS \cup VS} (\hat{\underline{\theta}}_{p^*})$$ best model for a given p^*

2) Other case: I have p^* and my client doesn't ask how good is the model → I can use all dataset to train $\hat{\underline{\theta}}_p$

$$\hat{\underline{\theta}}_{p^*}^{***} = \underset{\hat{\underline{\theta}}_{p^*}}{\operatorname{argmin}} \; J^{TS \cup VS \cup TestS} (\hat{\underline{\theta}}_{p^*})$$

but I'm no more able to tell how good is the model.

6.2 Other ways to chose complexity/validate the model

We made two hypothesis:

- (Hp.1) $\mathcal{S} \equiv \mathcal{M}(\boldsymbol{\theta})$
- (Hp.2) $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}$

We have some additional validation methods for the model:

- Whiteness test or Anderson test (error sequence is white with a certain confidence α)
- Statistical corrections of J_N
- Akaike Information Criterion (AIC)
- Minimum Description Length (MDL)
- ...

6.3 Whiteness/Anderson test (validation between 2 models)

Other way to choose the complexity

Black box general case prediction form:

$$\hat{y}(k|\hat{\underline{\theta}}) = \left[1 - \frac{\hat{C}(z)}{\hat{D}(z)}\right] y(k) - \frac{\hat{C}(z)\hat{B}(z)}{\hat{D}(z)\hat{A}(z)}\mu(k)$$

Which hypotesis lead to the best model?

1) There exists a structure of my System like this:

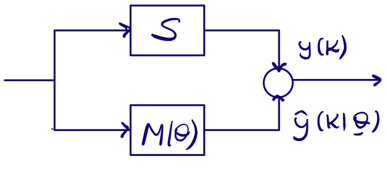
$$y(k) = \left[1 - \frac{C(z|\theta)}{D(z|\theta)}\right] y(k) + \frac{C(z|\theta)B(z|\theta)}{D(z|\theta)A(z|\theta)}\mu(k) + \eta(k)$$

if my sys is exactly made like this so $\underline{\theta}$ are my true parameters

$S \in \underbrace{M(\underline{\theta})}_{\text{class of the models I'm considering}} + \text{noise}$

2) $\hat{\underline{\theta}} = \underline{\theta}$

$$1) + 2) \rightsquigarrow \hat{y}(k|\hat{\underline{\theta}} = \underline{\theta}) = \left[1 - \frac{C(z|\underline{\theta})}{D(z|\underline{\theta})}\right] y(k) + \frac{C(z|\underline{\theta})B(z|\underline{\theta})}{D(z|\underline{\theta})A(z|\underline{\theta})}\mu(k)$$



$e(k|\hat{\underline{\theta}} = \underline{\theta}) = \eta(k)$

best possible value of the error

Even if I'm in my best case the error ≠ 0

$$E\, e^2(k|\hat{\theta}) > \lambda^2$$

variance of the error (expectation of the error)

variance of the noise

if I'm in hypotesis 1 + 2 → so it's exacty λ^2

empirical approximation of λ^2:

$$\lambda^2 \simeq \frac{1}{N_{test}} \sum e^2(k|\hat{\underline{\theta}}^*_{p*})$$

NB: 1) and 2) are quite never true

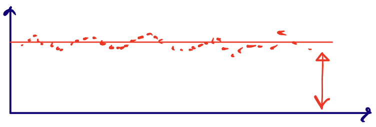
how can I understand if my sys respects 1 and 2?

I compute $e(k|\hat{\underline{\theta}}^*)$

$e(k|\hat{\underline{\theta}}^*) \sim WN(0, \lambda^2)$ → should be a white noise

if this is true → sys respects 1 and 2.

es:



this is not white → it has not zero mean + it has some sinusoidal trend → not white (the points should not be correlated)

So if the error sequence is not white it means that hypotesis 1 and 2 do not hold → I have made some errors before

In order to perform whiteness test we need the set-correlation:

$$R_{ee}(\tau) = Exp\{e(k)e(k+\tau)\} = \langle e(k)e(k+\tau)\rangle$$

This is a probabilistic definition. We don't have R_{ee} available, because we does not have the probability density function (so then we can't integrate it). But we can say, considering white noise approximation:

$$R_{ee}(\tau) = \langle e(k)e(k+\tau)\rangle \overset{(\text{if } WN(0,\lambda^2)\ )}{=} \begin{cases} \lambda^2 & \tau = 0 \\ 0 & \tau \neq 0 \end{cases}$$

It's good to recall that we have only N samples, so our R will be an approximated one:

$$\hat{R}_{ee}(\tau) = \cancel{\frac{1}{N}\sum_{k=1}^{N} e(k)e(k+\tau)} \overset{\text{not enough samples}}{\rightarrow} \frac{1}{N-\tau}\sum_{k=1}^{N-\tau} e(k)e(k+\tau)$$

$R_{ee}(\tau)$ is the self-correlation of $e(k)$, $\hat{R}_{ee}(\tau)$ is the **empirical** self-correlation: $\langle e(k)e(k+\tau)\rangle$ is an expectation (we don't need the time, we can "freeze" it), while $\hat{R}_{ee}(\tau)$ computation is an **ergodic** process, it proceed in time while computing the value.
Two facts can be proven:

$$\hat{R}^N_{ee}(\tau) \overset{N\to\infty}{\longrightarrow} R_{ee}(\tau) \qquad\qquad \hat{R}^N_{ee}(\tau) \overset{N \text{ big}}{\sim} N\left(0, \frac{\lambda^2}{N}\right)$$

Considering $\hat{R}^N_{ee}$, we can express a **normalized** self-correlation:

$$\chi(\tau) = \frac{\hat{R}^N_{ee}(\tau)}{\hat{R}^N_{ee}(0)} \qquad\qquad \chi(\tau) \overset{N \text{ big}}{\sim} N\left(0, \frac{1}{N}\right)$$

Equivalently:

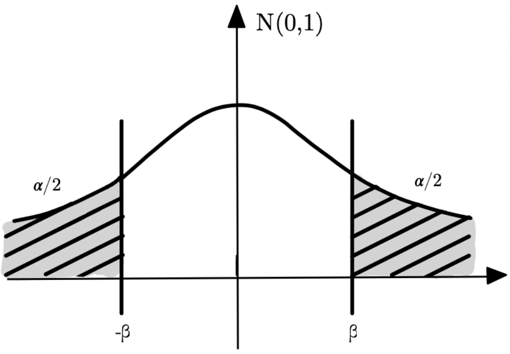
$$\sqrt{N}\chi(\tau) \sim N(0,1)$$

and $N(0,1)$ is the **normal Gaussian distribution**!

→ the whiteness/Anderson test is testing if $\sqrt{N}\chi(\tau)$ has a normal standard distribution:

$$\longrightarrow \boxed{\sqrt{N}\chi(\tau) \sim N(0,1)}$$

For the test we choose the α = confidence $\in (0,1)$ (the smaller it is, the more sure we are). The usual value is 0.05.
After that, the value β has to be obtained, how? Consider a normalized Gaussian distribution $N(0,1)$:



→ the β values have to cut tails for $\alpha/2$ probability (the outside areas cut by β are $\alpha/2$) and we will consider **the values inside the interval** (we cancel every bias, hence is a test on variance, checking that it is actually a variance of that kind).

Then, we have to compute $\chi(\tau)$, with $\tau \in (1, N_\tau) \subset \mathbb{N}$ considering **a correlation window of $N_\tau \ll N$ samples**. Beware, we need $N - N_\tau \gg 1$ in order to have statistically meaningful empirical averages, because the window has to be small w.r.t. the full dataset:

$$\begin{bmatrix} \chi(1) = \dfrac{\hat{R}^N_{ee}(1)}{\hat{R}^N_{ee}(0)} = \dfrac{\frac{1}{N-1}\sum_{k=1}^{N-1} e(k)e(k+1)}{\frac{1}{N}\sum_{k=1}^{N} e(k)e(k)} \\ \vdots \\ \chi(N_\tau) = \dfrac{\hat{R}^N_{ee}(N_\tau)}{\hat{R}^N_{ee}(0)} = \dfrac{\frac{1}{N-N_\tau}\sum_{k=1}^{N-N_\tau} e(k)e(k+N_\tau)}{\frac{1}{N}\sum_{k=1}^{N} e(k)e(k)} \end{bmatrix}$$

$$\left[\underbrace{e(0)\ \ e(1)\ \ e(2)\ \ ...\ \ e(N_\tau)}_{\text{"small"}\ N_\tau << N} \middle| \underbrace{e(N_\tau+1)\ \ ...\ \ e(N)}_{\text{"big"}\ N >> N_\tau} \right]$$

N_τ is "how much in the past you see for each sample when you compute the correlation between two samples". N_{out} is the number of times $\sqrt{N}\chi(\tau)$ is outside the interval $(-\beta, \beta)$:

$$if\ \frac{N_{out}}{N_\tau} < \alpha \Rightarrow \text{ OK} \qquad \begin{bmatrix} \text{I can say "}e(k)\text{ is a }WN\text{"} \\ \text{with confidence } \alpha \end{bmatrix}$$

Hypothesis evaluation How can we use α and β to evaluate how good are our hypotheses?

- (Hp.1) $\mathcal{S} \equiv \mathcal{M}(\boldsymbol{\theta})$
- (Hp.2) $\hat{\boldsymbol{\theta}}) = \boldsymbol{\theta}$

Let's start from (Hp.2). Consider two models $\mathcal{M}(\boldsymbol{\theta}')$ and $\mathcal{M}(\boldsymbol{\theta}'')$. Which of the two is better? If we observe the following results (be careful!!! The **test** is just a OK/notOK! It does **not** give us a **quantitative** index in order to evaluate the model):

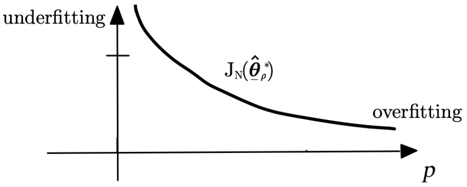
	$\mathcal{M}(\boldsymbol{\theta}')$	$\mathcal{M}(\boldsymbol{\theta}'')$	Outcome
$\alpha = 0.1$	OK	OK	Both OK → reduce the confidence
$\alpha = 0.05$	OK	OK	Both OK → $\alpha \downarrow$
$\alpha = 0.01$	notOK	notOK	Both notOK → $\alpha \uparrow$
$\alpha = 0.02$	notOK	OK	$\mathcal{M}(\boldsymbol{\theta}'')$ is better

So, changing α allows us to choose between the models → α is useful not only to validate them, but to compare them as well.

6.4 Statistical corrections of J_N (FPE)

For this method we can use **all the database**, so we can avoid the arbitrary fold-sectioning in training set an test set[9]. **Especially if we have the feeling that N is "too small" to be divided** in training-test sets that are "large enough", we should use this technique → we use that all the N samples for the training (though N by itself should be "big enough" at least to train the model).

While choosing the complexity p of the system we are then in front of a trade-off that can be clearly exemplified by the following qualitative graph (w.r.t. the graph, we want to "stay in the middle" to avoid overfitting):



Since we don't have any validation set, we need a different index to evaluate our model. This index is the **Final Prediction Error**, and regarding the complexity of the model, we can anticipate that is defined as follows:

$$FPE(p) = \frac{N+p}{N-p} J_N(\hat{\boldsymbol{\theta}}^*_p)$$

Now, let's find this expression. Let's say we're in the case in which N is "big enough" to train the model but "not big enough" to be splitted in train and test it. We want then to predict the final expected value of J_N (the Least-squared Cost):

$$J_N \underset{(N\to\infty)}{\longrightarrow} E\{J_N\}$$

$$E\{J_N\} = E\left\{\frac{1}{N}\sum_{k=1}^{N} e^2(k|\hat{\boldsymbol{\theta}})\right\} \underset{(\text{Hp.1, Hp.2})}{=} E\left\{\frac{1}{N}\sum_{k=1}^{N} n^2(k)\right\} \underset{(N\to\infty)}{\longrightarrow} \lambda^2$$

In general, we don't consider Hp.2 as an actual rule so, given the cardinality N of the dataset and the complexity p of the system we have to add:

$$E\{J_N(\hat{\boldsymbol{\theta}}^*)\} \simeq \lambda^2\left(1+\frac{p}{N}\right) = \lambda^2\left(\frac{N+p}{N}\right)$$

To have an unbalanced variance it has to converge to 0 with $N \to \infty$.

From statistics we know that the empirical approximation of λ^2 is:

$$\lambda^2 \simeq \frac{1}{N}\sum_{k=1}^{N} e^2(k|\hat{\boldsymbol{\theta}^*}_p)$$

But this is a biased statistical value, if we want an **unbiased** (average mean = 0) estimate of λ^2 we need to add a $-p$ to the denominator:

$$\lambda^2 \simeq \frac{1}{N-p}\sum_{k=1}^{N} e^2(k|\hat{\boldsymbol{\theta}^*}_p)$$

So, we have:

$$Exp\{J_N(\hat{\boldsymbol{\theta}}^*)\} \simeq \frac{1}{N-p}\frac{N+p}{N}\sum_{k=1}^{N} e^2(k|\hat{\boldsymbol{\theta}}_p) = \frac{N+p}{N-p}\left(\frac{1}{N}\sum_{k=1}^{N} e^2(k|\hat{\boldsymbol{\theta}}_p)\right) = \frac{N+p}{N-p}J_N(\hat{\boldsymbol{\theta}}^*_p)$$

[9] Normally they both have to be "large enough", and this is very difficult to determine, standard thumb rule are $train : test = 70\% : 30\%$, but this is not an actual criterion

which is the result already anticipated. So, we will have

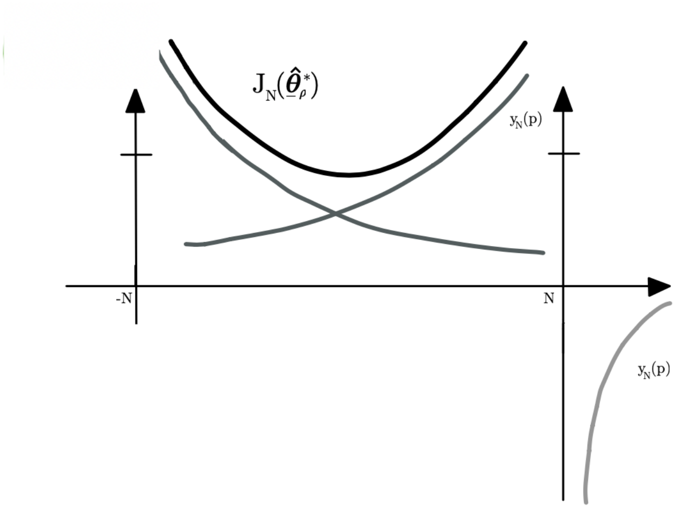
$$p^* = \arg\min_p \left(\frac{N+p}{N-p} J_N(\hat{\boldsymbol{\theta}}^*_p) \right)$$

→ we basically added a multiplicative part $\dfrac{N+p}{N-p}$, used when N is not so big. If we analyze it:

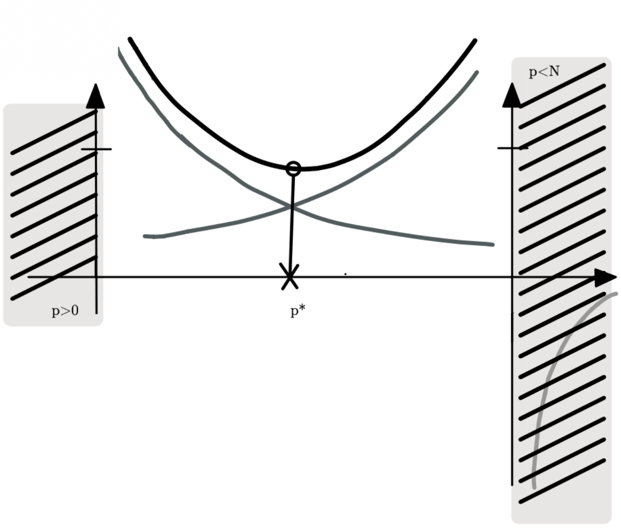
$$\boxed{FPE(p) = \underbrace{\frac{N+p}{N-p}}_{g_N(p)} J_N(\hat{\boldsymbol{\theta}}^*_P) \qquad g_N(p) = \frac{\frac{N}{p}+1}{\frac{N}{p}-1} \xrightarrow[p\to\pm\infty]{} -1}$$

$g_N(p)$ is the **statistical correction of the cost**

We can notice that, following this graph:



we should take $p < N$, and also $p > 0$. So the actual graph will be:



Then, we have $FPE(p)$ and we should take the value p^* of p that minimize it.

6.5 Other methods

We have several other methods, for example:

- Akaike Information Criterion (AIC)

$$AIC(p) = 2 * \frac{2}{N} + \ln(J_N(\hat{\boldsymbol{\theta}}_p^*))$$

- Minimum Description Length

$$MDL(p) = \frac{\ln(N)}{N} p + \ln(J_N(\hat{\boldsymbol{\theta}}_p^*))$$

Statistically, all the methods seen and the others converge to the same results for $N \to \infty$.

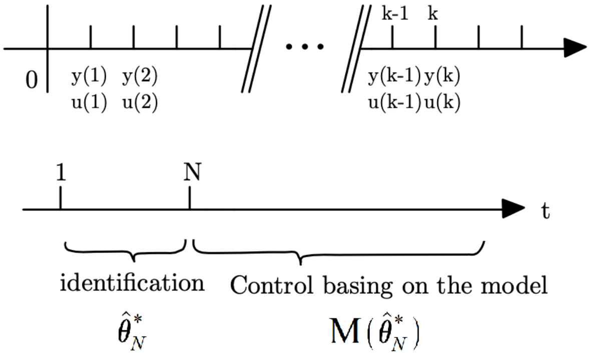
7 Recursive Online Identification

Now, we want to identify the parameters, but while collecting the data (before we saw only *offline* methods for identification, that relied on already-collected data). Given a current estimated model and a new observation, how should we update this model in order to take this new piece of information into account? Hence, we need to perform:

1. Data collection $k = 0, .., N$
2. Parameter identification & model validation

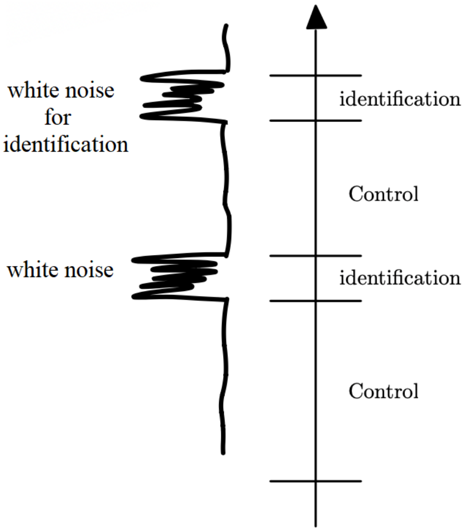
In many cases it is beneficial to have a model of the system available online while the system is in operation. The model should then be based on the observations up till the current time. A naive way to go ahead is to use all observations up to k to compute an estimate $\hat{\boldsymbol{\theta}}(k)$ of the system parameters. In recursive identification methods, the parameter estimates are computed recursively over time: suppose we have an estimate $\hat{\boldsymbol{\theta}}(k-1)$ at iteration $t-1$, then recursive identification aims to compute a new estimate $\hat{\boldsymbol{\theta}}(k)$ by a "simple modification" of $\hat{\boldsymbol{\theta}}(k-1)$ when a new observation becomes available at iteration k. The counterpart to online methods are the so-called offline or batch methods in which all the observations are used simultaneously to estimate the model.

What does it means having a time varying system? Normally, we don't have $\boldsymbol{\theta}$ but $\boldsymbol{\theta}(k)$ (temporal dependencies). What if, also, we have the necessity of an estimation of $\boldsymbol{\theta}$ or $\boldsymbol{\theta}(k)$ while collecting the data?



As shown in figure, we need a certain time in order to collect and identify the parameters. But imagine you driving a car: you need to drive it almost instantly (→ control task), but you also have to understand how and how to make it (→ identification): in the beginning you have a very uncertain model for "driving the car", so, you will probably need a robust control, in order to control the car even with big uncertainties (like when you learn to drive a car). **Robust control** is like moving object with strong constraining forces and no compliance → not gives a lot of information about the system. While if I want to identify the system, I have to solicitate it in order to see how it behaves ⇒ **conflicting goals**: control & identification.

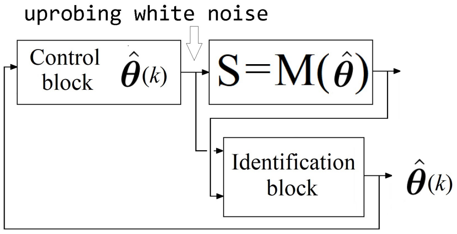
So, for example, we can identify for very few seconds, then control, then identify again and so on... as we can see in the following figure:



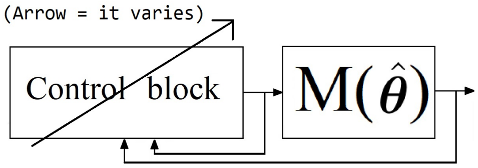
But this is not always valuable, it depends on the application: for a pick-and-place application with a serial manipulator, to "shake" a little the manipulator (adding some white noise to the input) in order to identify the parameters we have to move it back to the safe position. How much we "shake" the system is given by some parameter like this α:

$$u(k) = \underbrace{\alpha u_c(k)}_{control} + \underbrace{(1-\alpha)u_f(k)}_{identif.(probing)}$$

The control then should not be robust, but should comply to the uncertainty. The so-called adaptive control allows this. One possibile scheme for adapting control is the **"indirect" adapting control one**:


uprobing white noise
Control block
$\hat{\boldsymbol{\theta}}(k)$
$S=M(\hat{\boldsymbol{\theta}})$
Identification block
$\hat{\boldsymbol{\theta}}(k)$

Another one is the **"direct" adaptive control**:



7.1 Recursive Least-squares with ARX

At instant N we will have

$$\hat{\boldsymbol{\theta}}^* = \arg\min_{\hat{\boldsymbol{\theta}}} J_N(\hat{\boldsymbol{\theta}}) = \arg\min_{\hat{\boldsymbol{\theta}}} J_K(\hat{\boldsymbol{\theta}}) = \frac{1}{N}\sum_{i=1}^{N}[y(i) - \hat{y}(i|\hat{\boldsymbol{\theta}})]^2$$

(important to use i instead of k, because we will use k later)

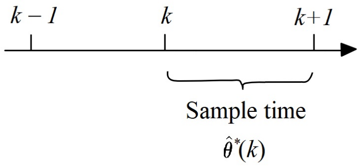
$\hat{\boldsymbol{\theta}}^*$ after N steps: $\hat{\boldsymbol{\theta}}^*(N)$ is the estimation of $\boldsymbol{\theta}(N)$ based on $u(i)y(i), i \in [1, N]$

At instant k we will have:

$$\hat{\boldsymbol{\theta}}^* = \arg\min_{\hat{\boldsymbol{\theta}}} J_k(\hat{\boldsymbol{\theta}}) = \arg\min_{\hat{\boldsymbol{\theta}}} J_k(\hat{\boldsymbol{\theta}}) = \frac{1}{k}\sum_{i=1}^{k}[y(i) - \hat{y}(i|\hat{\boldsymbol{\theta}})]^2$$

Assume we have an ARX model ($\hat{y}(k|\hat{\boldsymbol{\theta}}) = \boldsymbol{\varphi}^T(k)\hat{\boldsymbol{\theta}}$) we will have:

$$\hat{\boldsymbol{\theta}}^* = \left(\Phi_k\Phi_k^T\right)^{-1}\Phi_k\boldsymbol{y}_k \quad \text{with} \quad \Phi_k := [\boldsymbol{\varphi}(1)|...|\boldsymbol{\varphi}(k)] \quad \text{and} \quad \boldsymbol{y}_k = [y(1)|...|y(k)]^T$$



The computation time depends also on the value of k. But we have a problem here: Φ_k is a $p \times k$ vector while $\boldsymbol{y}_k$ is $k \times k$ → **the dimension increases with time** → at one point it will be unbearable to compute. Hence, at one point, i can **stop memorizing** old samples or i can **reduce the complexity**:

$$J_K(\hat{\boldsymbol{\theta}}) = \frac{1}{k}\sum_{i=0}^{k}\left[y(i) - \hat{y}(i|\hat{\boldsymbol{\theta}})\right]^2$$

To consider time-varying parameters I put then some **forgetting factor** $\mu \in (0, 1]$ ($\mu = 1$ is "no forgetting"):

$$J_K(\hat{\boldsymbol{\theta}}) = \frac{1}{k}\sum_{i=0}^{k}\mu^{k-1}\left[y(i) - \hat{y}(i|\hat{\boldsymbol{\theta}})\right]^2$$

... redo all the computations but with μ this time ...

We did it considered ($k = N$, $\mu = 1$):

$$J_N(\hat{\boldsymbol{\theta}}) = \frac{1}{N}\boldsymbol{e}_N^T(\hat{\boldsymbol{\theta}})\boldsymbol{e}_N(\hat{\boldsymbol{\theta}}) = \frac{1}{N}(\boldsymbol{y}_N - \Phi_N\hat{\boldsymbol{\theta}})^T(\boldsymbol{y}_N - \Phi_N\hat{\boldsymbol{\theta}})$$

... and all the other computations ...

If you want to do it for $\mu \neq 0$ you'll have:

$$J_N(\hat{\boldsymbol{\theta}}) = \frac{1}{N}\boldsymbol{e}_N^T(\hat{\boldsymbol{\theta}})W_N\boldsymbol{e}_N(\hat{\boldsymbol{\theta}})$$

You will work with a W_N that is a $N \times N$ matrix with columns like:

$$W_N = \begin{bmatrix} \mu^{N-1} & & & & \\ & \mu^{N-2} & & & \\ & & \mu^{N-3} & & \\ & & & \ldots & \\ & & & & \mu^{N-N=0} \end{bmatrix}$$

With this, the **normal equation** will be :

$$\nabla_{\boldsymbol{\theta}} J_N(\boldsymbol{\theta})|_{\hat{\boldsymbol{\theta}}(k)} \iff \Phi_N W_N \Phi_N^T\hat{\boldsymbol{\theta}}^* = \Phi_N W_N \boldsymbol{y}_N$$

$$\hat{\boldsymbol{\theta}}^*(N) = \arg\min\{J_N(\hat{\boldsymbol{\theta}})\} = \frac{1}{N}\sum_{i=0}^{N}\mu^{N-1}\left[y(i) - \hat{y}(i|\hat{\boldsymbol{\theta}})\right]^2 = (\Phi_N W_N \Phi_N^T)^{-1}\Phi_N W_N \boldsymbol{y}_N$$

$$\hat{\boldsymbol{\theta}}^*(k) = (\Phi_k W_k \Phi_k^T)^{-1}\Phi_k W_k \boldsymbol{y}_k$$

$$W_k = \begin{bmatrix} \mu^{k-1} & & & & \\ & \mu^{k-2} & & & \\ & & \mu^{k-3} & & \\ & & & ... & \\ & & & & \mu^{k-k=0} \end{bmatrix}$$

This equation has some interesting properties:

- In general we have:

$$[\boldsymbol{a}_1|...|\boldsymbol{a}_n] \begin{bmatrix} \boldsymbol{b}_1^T \\ \boldsymbol{b}_2^T \\ ... \\ \boldsymbol{b}_k^T \end{bmatrix} = \sum_{i=1}^{k} \boldsymbol{a}_i \boldsymbol{b}_i^T$$

with $\boldsymbol{a}_i$ that is a $n \times 1$, $\boldsymbol{b}_i^T$ that is a $1 \times m$ and $\boldsymbol{a}_i\boldsymbol{b}_i^T$ that is a $n \times m$.

- We define

$$\mathcal{S}_k := \Phi_k W_k \Phi_k^T = [\boldsymbol{\varphi}(1)|...|\boldsymbol{\varphi}(k)] \begin{bmatrix} \mu^{k-1} & & & & 0 \\ & \mu^{k-2} & & & \\ & & \mu^{k-3} & & \\ & & & ... & \\ 0 & & & & \mu^0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\varphi}(1) \\ \vdots \\ \boldsymbol{\varphi}(k) \end{bmatrix} =$$

$$= \mathcal{S}_k = [\boldsymbol{\varphi}(1)|...|\boldsymbol{\varphi}(k)] \begin{bmatrix} \mu^{k-1}\boldsymbol{\varphi}(1) \\ \vdots \\ \mu^0\boldsymbol{\varphi}(k) \end{bmatrix}$$

That we can rewrite as:

$$\mathcal{S}_k = \boldsymbol{\varphi}(k)\boldsymbol{\varphi}^T(k) + \sum_{i=1}^{k-1} \mu^{k-1}\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T = \boldsymbol{\varphi}(k)\boldsymbol{\varphi}^T(k) + \mu \sum_{i=1}^{k-1} \mu^{(k-1)-i}\boldsymbol{\varphi}(i)\boldsymbol{\varphi}(i)^T =$$
$$= \boldsymbol{\varphi}(k)\boldsymbol{\varphi}^T(k) + \mu\mathcal{S}_{k-1}$$

$$\mathcal{S}_{k-1} = \frac{1}{\mu}\mathcal{S}_k - \frac{1}{\mu}\boldsymbol{\varphi}(k)\boldsymbol{\varphi}(k)^T$$

- We define also

$$\mathcal{Q}_k = \Phi_k W_k \boldsymbol{y}_k = [\boldsymbol{\varphi}(1)|...|\boldsymbol{\varphi}(k)] \begin{bmatrix} \mu^{k-1} & & & & 0 \\ & \mu^{k-2} & & & \\ & & \mu^{k-3} & & \\ & & & ... & \\ 0 & & & & \mu^0 \end{bmatrix} \begin{bmatrix} y(1) \\ \vdots \\ y(k) \end{bmatrix} =$$

$$= \mathcal{Q}_k = [\boldsymbol{\varphi}(1)|...|\boldsymbol{\varphi}(k)] \begin{bmatrix} \mu^{k-1}y(1) \\ \vdots \\ \mu^0 y(k) \end{bmatrix}$$

That we can rewrite as:

$$\mathcal{Q}_k = \sum_{i=1}^{k} \boldsymbol{\varphi}(i)\mu^k y(i)^T = \boldsymbol{\varphi}(k)\boldsymbol{y}(k) + \mu\mathcal{Q}_{k-1}$$

- The normal equation for $k-1$ is:

$$\mathcal{S}_k \hat{\boldsymbol{\theta}}^* = \mathcal{Q}_k$$

So:

$$\hat{\boldsymbol{\theta}}^* = \mathcal{S}_k^{-1}\mathcal{Q}_k = \mathcal{S}_k^{-1}[\boldsymbol{\varphi}(k)\boldsymbol{y}(k) + \mu\mathcal{Q}_{k-1}] =$$

$$= \mathcal{S}_k^{-1}[\boldsymbol{\varphi}(k)\boldsymbol{y}(k) + \mu(\frac{1}{\mu}\mathcal{S}_k - \frac{1}{\mu}\boldsymbol{\varphi}(k)\boldsymbol{\varphi}^T(k))\hat{\boldsymbol{\theta}}^*(k-1)]$$

$$\hat{\boldsymbol{\theta}}^*(k) = \underset{Prep.}{\hat{\boldsymbol{\theta}}^*(k-1)} + \underset{Gain}{\mathcal{S}_k^{-1}\boldsymbol{\varphi}(k)}\underset{Innovation}{[y(k) - \boldsymbol{\varphi}^T(k)\hat{\boldsymbol{\theta}}(k-1)]}$$

$$\boxed{\hat{\boldsymbol{\theta}}^*(k) = \hat{\boldsymbol{\theta}}^*(k-1) + \gamma(k)[y(k) - \hat{y}(k|\hat{\boldsymbol{\theta}}(k-1))]}$$

this means that we can take the last value that we have, the output of the value instant k, and compare it with what we can predict with the information up to instant $k-1$.

7.2 Recursive LS Algorithm

$$\hat{\underline{\theta}}^*(k) = \underset{\hat{\theta}}{\text{argmin}}\left[\frac{1}{k}\sum_{i=1}^{k}[y(i) - \hat{y}(i|\hat{\underline{\theta}})]^2\mu^{k-i}\right]$$

ARX: $\hat{y}(k|\hat{\underline{\theta}}) = \varphi^T(k)\,\underline{\theta}$

$$\hat{\underline{\theta}}^*(k) = \hat{\underline{\theta}}^*(k-1) + \underline{\gamma}(k)\left[y(k) - \hat{y}(k|\hat{\underline{\theta}}^*(k-1))\right]$$

At instant k things available: $\hat{\underline{\theta}}^*(k-1)$, S_{k-1}, $y(1) \dots y(k-1)\ y(k)$
$\mu(1) \cdots \mu(k-1)\ \mu(k)$

complete algorithm

$S_k = \varphi(k)\varphi^T(k) + \mu S_{k-1}$ we can compute this, we have everything

$\gamma(k) = S_{k-1}^{-1}\varphi(k)$ same here

$\hat{y}(k|\hat{\underline{\theta}}^*(k-1)) = \varphi^T(k)\hat{\underline{\theta}}^*(k-)$ same here [prediction fae]
y

$\hat{\underline{\theta}}^*(k) = \hat{\underline{\theta}}^*(k-1) + \underline{\gamma}(k)\left[y(k) - \hat{y}(k|\hat{\underline{\theta}}^*(k-1))\right]$ same here

7.3 Algorithm initialization

How to start: (Initialisation)

$S_0 = ?$

$\hat{\underline{\theta}}^*(0) = ?$

Recall

$$\underbrace{S_k = \sum_{i=1}^{k}\varphi(i)\varphi^T(i)\mu^{k-1} = \Phi_k W_k \Phi_k^T = S_k^T}_{\text{it's a symmetrik matrix}}$$

① Standard initialisation:

$S_0 = I$ pxp (identity matrix)

it's a symmetrix matrix no other reason

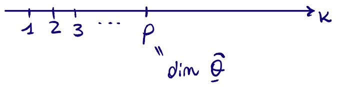
$\underline{\hat{\theta}}^*(0) = \underline{0}$ (vector of zeros)

If you start with wrong values ⇝ does not matter

they are forgotten (there is the forget factor)

Initial values have NO meaning

② Exact initialisation



$$\underset{p \times p}{\text{Rank}\,(S_k)} = \text{rank}\,(\underset{p \times p}{\Phi_k W_k \Phi_k^T}) = p$$

in ARX we needed $\text{Rank}\,(\Phi_N \Phi_N^T) = p$

$\Updownarrow$

$\text{rank}\,(\Phi_N) = p$

we need it $\Leftrightarrow \text{rank}\,(\Phi_k) = p$

If $\text{Rank} = p \rightsquigarrow \underline{\hat{\theta}}^*(k) = S_k^{-1} Q_k$

one-shot solution

If $\text{rank}\ \Phi_k = p \rightsquigarrow S_{\tilde{k}} = \Phi_{\tilde{k}} W_{\tilde{k}} \Phi_{\tilde{k}}^T$

Wait until $\tilde{k}$ such that $(\text{rank}\ \Phi_k = p)$ this condition holds true. I'm filling the matrix. $\tilde{k} \geq p$

$$\Phi_k = [\varphi(1) \mid \varphi(2) \mid \varphi(3) \mid \dots\ \varphi(p) \mid \dots\ \varphi(\tilde{k})]$$

only at this moment I can have Rank = p

So:

Wait until $(\tilde{k} \geq p)$ $\text{Rank}\ \Phi_k = p$:

$$\begin{cases} S_{\tilde{k}} = \Phi_{\tilde{k}} W_{\tilde{k}} \Phi_k^T \\ \underline{\hat{\theta}}^*(\tilde{k}) = S_{\tilde{k}}^{-1} Q_{\tilde{k}} \end{cases}$$

Which is better? ① or ②? <u>It depends</u> : why?

① starts immediately at instant 1

↳ it starts with the wrong values but we start (but probably arrived in $k = \tilde{k}$ we will have still wrong values)

② we go in blind way until $\tilde{k}$, but when arrived in $\tilde{k}$ we have the exact values.

<u>Best choice</u> : mixing the two.

Starts with ① until $k = \tilde{k}$ and then use ②

We will probably have a jump in S but we can smooth it.

7.4 Algorithm matrix inversion

INVERSION PROBLEM

$$S_k = \varphi(k)\varphi^T(k) + \mu S_{k-1}$$

If we started with a positive-definite S_k, S_k will remain positive-definite

we need it to be p.d. in order to be inverted

$$\gamma(k) = \underbrace{S_k^{-1}}_{V_k} \varphi(k) =$$

$$= V_k \varphi(k)$$

I don't see any inverse here

we are only multipling for a scalar and adding a positive-definite

$$V_k = S_k^{-1} = [\underbrace{\mu V_{k-1}^{-1}}_{F} + \underbrace{\overset{p\times1}{\varphi(k)}}_{G} \underbrace{\overset{1}{I}}_{H} \underbrace{\overset{1\times p}{\varphi^T(k)}}_{K}]^{-1} =$$

identity → does not change anything

it's a 1

$$= [F + GHK]^{-1}$$

we don't have to recall it by heart.

Matrix inversion lemma $\forall$ FGHK

F = square matrix dimension must match

$$(F + GHK)^{-1} = F^{-1} - F^{-1}G(H^{-1} + KF^{-1}G)^{-1}KF^{-1}$$

$$V_k = S_k^{-1} = \frac{1}{\mu}\underset{V_{k-1}}{\cancel{S_{k-1}^{-1}}} - \frac{1}{\mu}\underset{V_{k-1}}{\cancel{S_{k-1}^{-1}}}\varphi(k)\underbrace{\left[1 + \frac{\varphi^T(k)\overset{V_{k-1}}{\cancel{S_{k-1}^{-1}}}\varphi(k)}{\mu}\right]^{-1}}\varphi^T(k)\frac{1}{\mu}\underset{V_{k-1}}{\cancel{S_{k-1}^{-1}}}$$

No invertion needed, but this one ● but this is a scalar

We don't have no more to invert a pxp matrix, just a scalar to invert.

Initialisation:

Now we need V_0

Standard initialisation: $V_0 = I$

Exact initialisation: $V_{\tilde{k}} = [\Phi_{\tilde{k}} W_{\tilde{k}} \Phi_{\tilde{k}}^T]^{-1}$

there is an inverse, but we have to do it just once

7.5 One-step prediction form

We have seen always the prediction form but it exits also the "One-step prediction form"

$$\hat{y}(k|\hat{\underline{\theta}}) = \varphi^T(x)\,\hat{\underline{\theta}} \underset{\substack{\uparrow \\ \text{gen. case}}}{=} \left[1+\frac{\hat{C}}{\hat{D}}\right] y(k) + \frac{\hat{C}\hat{B}}{\hat{D}\hat{A}}\,\mu(k)$$

At instant k:

$\hat{y}(k+1|\hat{\underline{\theta}})$ we need --- $y(k-1)\ y(k)$
--- $\mu(k-1)\ \mu(k)\ \mu(k+1)$

$\hat{y}(k+2|\hat{\underline{\theta}})$ we need --- $y(k-1)\ y(k)$
--- $\mu(k-1)\ \mu(k)\ \mu(k+1)\ \mu(k+2)$

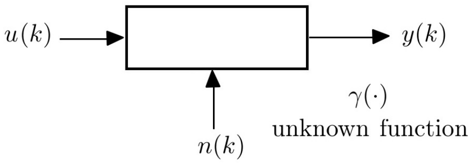
We are predicting the future

8 Identification for non-linear systems

We can extend our treatise on non-linear system starting with an hypothesis, that is, our nonlinear system has a structure similar to ARX (Nonlinear ARX):

$$Hp.NARX \quad y(k) = \boldsymbol{\varphi}^T(k)\boldsymbol{\theta} + n(k) \qquad \text{becomes} \qquad y(k) = \gamma[\boldsymbol{\varphi}(k)] + n(k)$$

$$\text{with} \quad n(k) \sim WN(0, \lambda^2) \quad \text{and} \quad \boldsymbol{\varphi} = [y(k-1), ..., y(k-n), u(k), ..., u(k-m)]^T$$



Augmenting n, m increases the representation capability.

For our case: $\boldsymbol{\varphi}(k)$ is a-priori decided, we are **not** using p for its dimension (there is a fixed $p_{\boldsymbol{\varphi}}$), we're gonna use p for the dimension of a parameter vector.

We don't know what is the shape of the unknown function $\gamma(\cdot)$. The problem, qualitatively speaking, consists in finding the best γ^* to model the system. Modeling the system in this case means having a model able to predict the behavior of the system. This is a **predictive** description:

$$y(k) = \gamma[\boldsymbol{\varphi}(k)] + n(k)$$

We will have:

$$\hat{y}(k|\gamma^*) = \gamma^*(\boldsymbol{\varphi}(k))$$

$$\text{If we know } \ \gamma(\cdot) \Rightarrow \hat{y}(k|\gamma) = \gamma[\boldsymbol{\varphi}(k)] + \cancel{n(k)}$$

Notice that in $y(k) = \boldsymbol{\varphi}^T(k)\boldsymbol{\theta} + n(k)$ the term $\boldsymbol{\varphi}^T(k)\boldsymbol{\theta}$ can be rewritten as

$$\boldsymbol{\varphi}^T(k)\boldsymbol{\theta} = -a_1 y(k-1) - ... - a_n y(k-n) - b_0 u(k) - ... - b_n u(k-n)$$

This is a **linear difference equation**, so

$$\gamma^*(\boldsymbol{\varphi}(k))$$

is a **non-linear difference equation**, and what we are doing can be seen as trying to solve it in an indirect way, using γ^* instead of actually solving it.

The unknown γ is a function , we need to map any possible argument with the output! Is an ∞-dimensional problem, because the function is an infinite (dense) input-output mapping. But, from another side, it is easy if we think that the function is a scalar → is (formally) easily extended to the MIMO case:

$$\boldsymbol{\varphi}(k) \rightarrow [\boldsymbol{y}(k-1)^T, \ ..., \ \boldsymbol{y}(k-n)^T, \ \boldsymbol{u}(k)^T, \ \boldsymbol{u}(k-m)^T]^T$$

Approximation → we have methods for solving (in some way at some quality) parametrical problems:

$$\hat{J}(\hat{\boldsymbol{\theta}}) = \arg\min_{\hat{\boldsymbol{\theta}}} \hat{\boldsymbol{\theta}}$$

→ we saw the one-shot solution or the mathematical programming methods.

Cost functions Which cost function we will use here? We hypothesized that is N-ARX and now we compute the approximation:

$$Hp.NARX \rightarrow y(k) = \gamma(\boldsymbol{\varphi}(k)) + n(k)$$

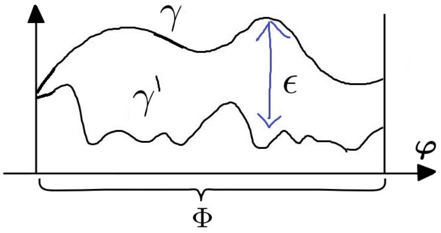
- If $\hat{\gamma}$ is approx. solution:

$$\epsilon_2 = \int_{\Phi} ||\gamma(\boldsymbol{\varphi}) - \hat{\gamma}(\boldsymbol{\varphi})||^2 \, p_{\boldsymbol{\varphi}} \, d\boldsymbol{\varphi}$$

  Φ is the set of all the possible $\boldsymbol{\varphi}$. This is the so-called 2-norm of the function.

- Another cost function could be:

$$\epsilon_\infty = \max_{\boldsymbol{\varphi}\in\Phi} ||\gamma(\boldsymbol{\varphi})\hat{\gamma}(\boldsymbol{\varphi})||$$



Using the 2-norm, we can reformulate the problem as follows:

$$\texttt{Find}\quad \gamma^*(\cdot)\ \ \texttt{s.t.}\qquad \tilde{\gamma}^* = \arg\min_{\tilde{\gamma}} (\epsilon_2(\tilde{\gamma}))$$

Regarding the parameters:

$$\tilde{\gamma}(\cdot) \to \hat{\gamma}[\cdot, \hat{\boldsymbol{\theta}}]$$

$$\boldsymbol{\varphi}(k) = \hat{\gamma}[\cdot, \hat{\boldsymbol{\theta}}]$$

where $\hat{\gamma}$ is a function with a fixed structure with a fixed number of parameters.

Reduce the search to a certain set of parametrized functions.

We constrain the solution to have some point structure:

$$\gamma(\boldsymbol{\varphi}) \sim\!\!\to \hat{\gamma}(\boldsymbol{\varphi}, \hat{\theta})$$

Example:

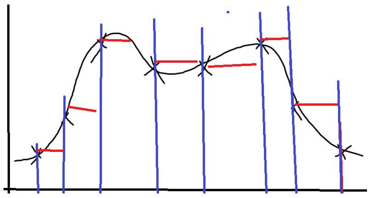
$$\hat{\gamma}(\boldsymbol{\varphi}, \hat{\theta}) \longrightarrow \boldsymbol{\varphi}^T\hat{\boldsymbol{\theta}}$$

linear combination of the elements of $\boldsymbol{\varphi} \Rightarrow$ ARX! Linear in $\boldsymbol{\varphi}$, linear in $\boldsymbol{\theta}$.

8.1 Function estimation

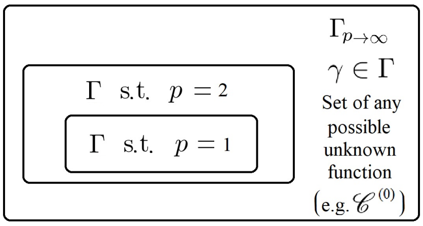
Machine Learning How we estimate functions? Did we already seen it? Yes, when in Machine Learning we perform regression, we operate a estimation of a function.

Rectangles We can also sample it, and use rectangular, or trapezoidal parts in order to interpolate it:



With L rectangles we will have an overall complexity that is $p = 2L + 1$ (is it correct?)

For continuous functions $(p \to \infty) \Rightarrow (\epsilon_2 \to 0)$. We have then the following situation:

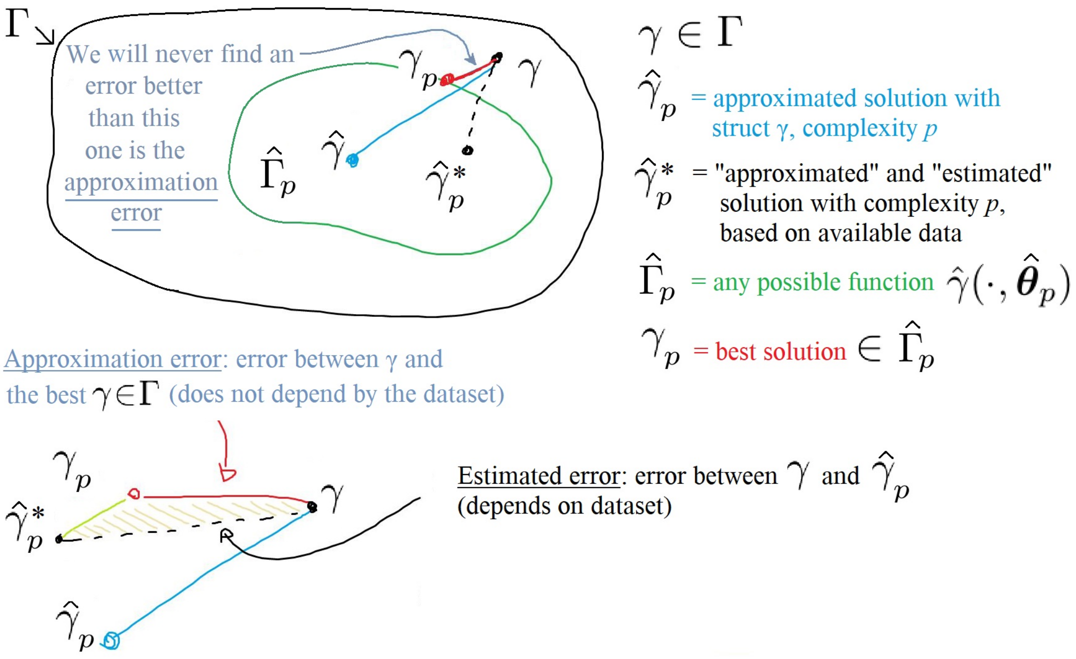


$$\text{For a complexity } p \Rightarrow \boxed{\hat{\gamma}(\cdot, \boldsymbol{\theta}_p)} \Rightarrow \Gamma_p$$

Γ is dense in Γ_p for $p \to \infty \Rightarrow$ **universal representation property**: rectangles can represent every function (universal) with an adequate p ($p \to \infty$ for continuous functions)

Other methods Not only rectangles or regression, also polynomials, Neural networks, splines, etc... are other approximation methods.

Approximation and estimation errors In general, regarding any functional approximation problem (γ is called functional because it is a function that takes as argument a function) we will have the following situation:


Γ
We will never find an error better than this one is the approximation error
$\hat{\Gamma}_p$
γ_p
γ
$\hat{\gamma}$
$\hat{\gamma}_p^*$
$\gamma \in \Gamma$
$\hat{\gamma}_p$ = approximated solution with struct γ, complexity p
$\hat{\gamma}_p^*$ = "approximated" and "estimated" solution with complexity p, based on available data
$\hat{\Gamma}_p$ = any possible function $\hat{\gamma}(\cdot, \hat{\boldsymbol{\theta}}_p)$
γ_p = best solution $\in \hat{\Gamma}_p$
Approximation error: error between γ and the best $\gamma \in \Gamma$ (does not depend by the dataset)
γ_p
$\hat{\gamma}_p^*$
γ
$\hat{\gamma}_p$
Estimated error: error between γ and $\hat{\gamma}_p$ (depends on dataset)

8.2 Approximation properties

We have talked about the possibility of using an approximated structure function for our model:

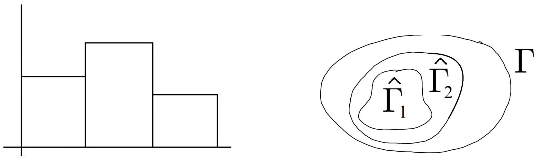
$$\gamma(\cdot) \sim \hat{\gamma}[\cdot, \hat{\boldsymbol{\theta}}]$$

$$Hp.1 \quad \to \quad y(k) = \gamma[\boldsymbol{\varphi}(k)] + n(k) \qquad \gamma_i \text{ unknown}$$

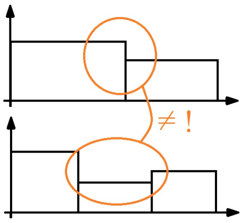
$$\hat{y}(k|\hat{\boldsymbol{\theta}}) = \hat{\gamma}[\boldsymbol{\varphi}(k), \hat{\boldsymbol{\theta}}]$$

p is the complexity, and it can be "confused" without error (as we stated before) with the dimension of $\hat{\boldsymbol{\theta}}$ since the dimension of $\boldsymbol{\varphi}$ is fixed.

Now, let's analyze some properties of the **rectangles approximation** case. Given the number of rectangles $p = 3$ we can say that:



But it is not true that Γ_3 includes Γ_2, because:



Talking about **approximation properties**:

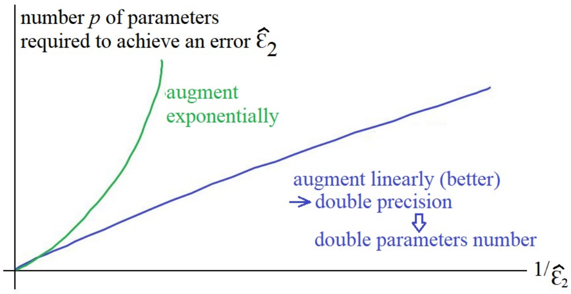
1. Chose any $\hat{\epsilon_2} > 0$

   ⇒ a number of parameters exists such that the approximation error between $\hat{\Gamma_p}$ and γ (i.e. a suitable $\hat{\gamma}(\cdot, \hat{\boldsymbol{\theta}}_p)$) exists, so that:

   $$\epsilon_2(\gamma, \hat{\Gamma_p}) < \hat{\epsilon_2}$$

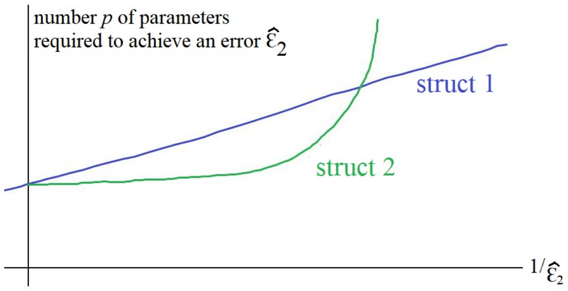
   This is the **universal approximation property** that we already saw: sums of rectangles is an universal approximator for continuous functions, as well polynomials and neural networks. But nothing says how much we have to increase p or the dimension of data! This works only in principle.

2. The previous property stated that "we will succeed someday", but when? How does p increases if the required error ϵ_2 decreases? What we prefer?



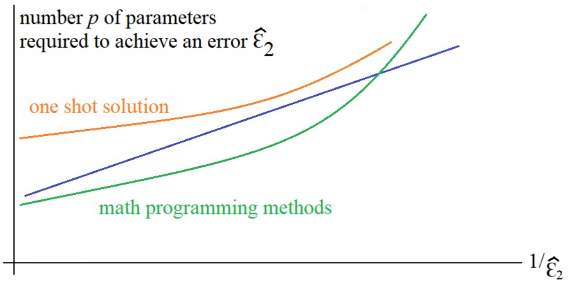
   We would appreciate if the number of parameters p to obtain a certain error increases moderately when the error decreases.

   There are different mathematical proofs about p behavior (e.g. Barron theorem) but, as for the first case, they are **asymptotic** theoretical results. Consider this case:



   Which structure is better? It really depends on what we want to obtain. You have to ask yourself what is the **required precision**, which one is a required precision.

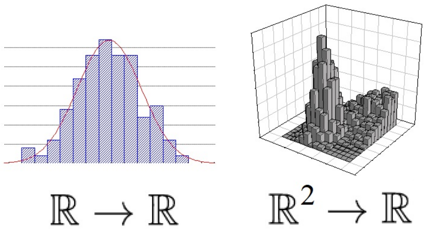
   Consider another case:



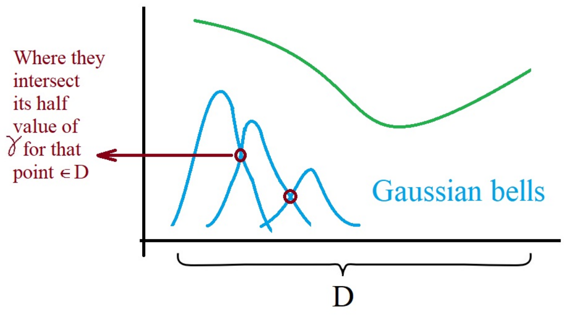
In this case, there is another tradeoff: if the one-shot solution has not so much additional parameters, probably it will be better to use the one-shot solution instead of using a complex mathematical programming solution.

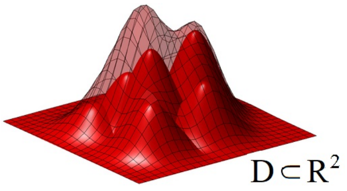
8.3 Examples of approximated structures

- Sum of **rectangles** (or, more in general, sum of hyper rectangles):



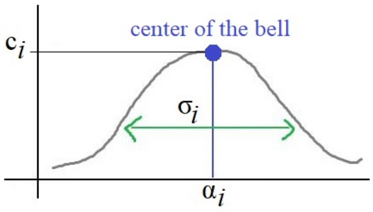
- Generalizing the rectangles approximation we find the methods based on **nearest neighbors algorithms**
- *RBF* (**Radial Basis Functions**): assume $\gamma : D \subset \mathbb{R} \rightarrow C \subset \mathbb{R}$ (restricted only on a limited domain D):


Where they intersect its half value of γ for that point ∈ D
Gaussian bells
D



How many parameters are needed for *RBF*? In the scalar case:

$$\hat{\gamma}(\varphi, \hat{\theta}) = \sum_{i=1}^{L} c_i e^{\left[-\frac{(\varphi - \alpha_i)^2}{\sigma_i^2}\right]}$$



In the general case:

$$\hat{\gamma}(\boldsymbol{\varphi}, \hat{\boldsymbol{\theta}}) = \sum_{i=1}^{L} c_i e^{\left[-\frac{||\boldsymbol{\varphi} - \boldsymbol{\alpha}_i||^2}{\sigma_i^2}\right]}$$

A structure like this has spherical symmetry, α_i represents the center of the i-th bell, c_i it's max height and the σ_i the width.

Then, the parameter vector that will characterize this function will be:

$$\hat{\boldsymbol{\theta}} = col\{c_i, \sigma_i, \boldsymbol{\alpha}_i | i \in [1, L]\}$$

- $FFNN$ (**Feed-forward neural networks**)

$$\hat{\gamma}(\boldsymbol{\varphi}, \hat{\boldsymbol{\theta}}) = \sum_{i=1}^{L} c_i \cdot f_{sigmoidal}(\boldsymbol{\varphi}^T \boldsymbol{\alpha}_i + b_i)$$

There are many possibilities of sigmoidal function (e.g Heaviside step, sigmoid, ReLU etc...) as shown in Machine Learning course.

$f_{sigmoidal}(\boldsymbol{\varphi}^T \boldsymbol{\alpha}_i + b_i)$ is indeed a function depending on $(\boldsymbol{\varphi}, \boldsymbol{w}_i)$, where $\boldsymbol{w}_i$ is a set of vectors

$$\boldsymbol{w}_i = \begin{pmatrix} \boldsymbol{\alpha}_i \\ b_i \end{pmatrix}$$

while for RBF was:

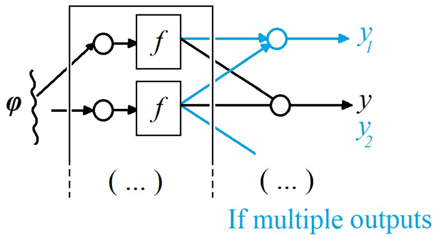
$$\boldsymbol{w}_i = \begin{pmatrix} \boldsymbol{\alpha}_i \\ \sigma_i \end{pmatrix}$$

- So we can just generalize the expression for both FFNN and RBF with:

$$\hat{\gamma}(\boldsymbol{\varphi}, \hat{\boldsymbol{\theta}}) = \sum_{i=1}^{L} c_i \cdot f(\boldsymbol{\varphi}^T, \boldsymbol{w}_i)$$

→ if f is sigmoidal, it is a $FFNN$, if f a exponential is RBF. f is called **basis function**

These are all called neural networks, both $FFNN$ and RBF, because they share the same schematic representation:



If $\boldsymbol{w}_i, i \in [1, L]$ (internal parameters of the basis function f) are chosen a-priori, one for all and then not anymore optimized, then we can write f as a function of $\boldsymbol{\varphi}$ only!

$$f(\boldsymbol{\varphi}^T, \boldsymbol{w}_i) \longrightarrow f_i(\boldsymbol{\varphi}^T)$$

So, the structure becomes:

$$\hat{\gamma}(\boldsymbol{\varphi}, \hat{\boldsymbol{\theta}}) = \sum_{i=1}^{L} c_i \cdot f_i(\boldsymbol{\varphi}^T)$$

L in the original representation was meaning, for example in RBF, the number of bells → is not the number of parameters! The dimension of parameters vector is different! For RBF there was $\hat{\boldsymbol{\theta}} = col\{...\}$ as parameter vector. For this particular case instead:

$$\hat{\boldsymbol{\theta}} = \begin{pmatrix} c_1 \\ \vdots \\ c_L \end{pmatrix}$$

$$\boldsymbol{f}(\boldsymbol{\varphi}) = \begin{pmatrix} f_1(\boldsymbol{\varphi}) \\ \vdots \\ f_L(\boldsymbol{\varphi}) \end{pmatrix}$$

Then:

$$\hat{\gamma}[\boldsymbol{\varphi}, \hat{\boldsymbol{\theta}}] = [\boldsymbol{f}(\boldsymbol{\varphi})]^T \hat{\boldsymbol{\theta}}$$
$$y(k|\hat{\boldsymbol{\theta}}) = [\boldsymbol{f}(\boldsymbol{\varphi})]^T \hat{\boldsymbol{\theta}}$$

In the ARX we saw:

$$y(k|\hat{\boldsymbol{\theta}}) = \boldsymbol{\varphi}(k)^T \hat{\boldsymbol{\theta}}$$

So, $\boldsymbol{f}$ plays the role of $\boldsymbol{\varphi}(k)$ in the ARX!!

8.4 Parameters optimization

$$\hat{y}(k|\hat{\underline{\theta}}) = \hat{\gamma}[\varphi(k), \hat{\underline{\theta}}]$$

$$\varphi(k) = \begin{bmatrix} y(k-1) \\ \vdots \\ y(k-n) \\ u(k) \\ u(k-1) \\ \vdots \\ u(k-m) \end{bmatrix}$$ regressor : $\dim(\varphi(k)) = n+m+1$

$p = \dim \hat{\underline{\theta}}$

How to optimize the parameters?

internal parameters

$$\hat{\gamma}[\varphi(k), \hat{\underline{\theta}}] = \sum_{i=1}^{L} c_i f[\varphi(k), \underline{w}_i]$$

one hidden layer

f is not a linear function (es. sigmoid ..)

if $\underline{w}_i$ a priori fixed ⇝ $\sum_{i=1}^{p=L} c_i f_i(\varphi(k))$

So we have two possibilities

Ⓐ $\underline{w}_i$ fixed $\hat{\underline{\theta}} = \begin{bmatrix} c_1 \\ \vdots \\ c_L \end{bmatrix}$ ⇝ $p = L$

Ⓑ also $\underline{w}_i$, $i = 1 \dots L$ to be optimized $\hat{\underline{\theta}} = \text{col}[c_i, \underline{w}_i \,;\, i = 1..N]$

↓ es RBF $\underline{w}_i = \begin{bmatrix} \alpha_i \\ \sigma_i \end{bmatrix}$

If fixed internal parameters, less parameters

But which is better? Fixed or not fixed?

- fixed : $\hat{\gamma}[\varphi(k), \underline{\theta}] = f^T[\varphi(k)]\, \hat{\underline{\theta}}$

$$f^T[\varphi(k)] = \begin{bmatrix} f_1(\varphi(k)) \\ \vdots \\ f_L(\varphi(k)) \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_L \end{bmatrix}$$

It's a scalar product ⇝ it's linear in the parameters! Great!

one shot solution available

- not fixed : not linear in $\hat{\underline{\theta}}$

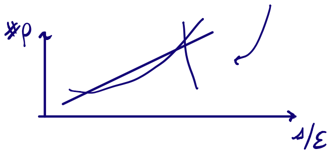
In both case f (the model) is NOT LINEAR, but with fixed is LINEAR in the PARAMETERS

$\downarrow$

no superposition of effects

But often non linear in $\underline{\theta}$ approximation benefit of "PROP 2"

but asympt. results



(A) Lin in $\underline{\theta}$ $\rightsquigarrow$ one shot solution available:

$$\underline{\hat{\theta}}^* = \left(\tilde{\Phi}_N \tilde{\Phi}_N^T\right)^{-1} \tilde{\Phi}_N \underline{y}_N$$ very rapid solution

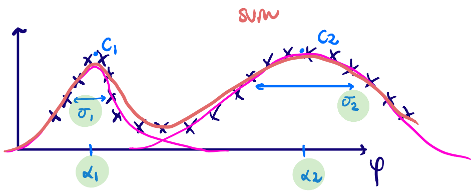
also recursive solution:

$$\underline{\hat{\theta}}^*(k) = \underline{\hat{\theta}}^*(k-1) + \dots$$ also with forgetting factor

Ideal for real-time applications

(B) Maybe better if we have lot of time and small errors.

Mathematical programming



I could think 2 bells are enough

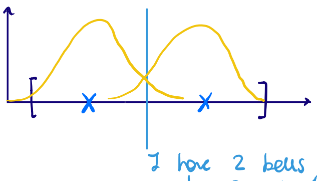
(B) $L = 2$ $\left\{ \begin{array}{l} c_1 \; c_2 \\ \alpha_1 \; \alpha_2 \\ \sigma_1 \; \sigma_2 \end{array} \right.$ good solution

(A) $\left. \begin{array}{l} \alpha_1 \; \alpha_2 \\ \sigma_1, \sigma_2 \end{array} \right\}$ a priori decided

$\downarrow$ If a priori I decide the parameters like the 🟩 I obtain the same good solution, so I can think that fixed parameters are better. But how can I choose the same parameters without using the data x?

(A) I don't have the x.

Where do I put the centers?



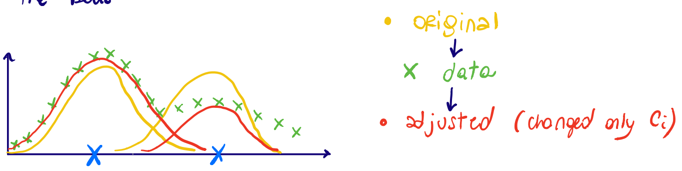
I have 2 bells, I divide the domain in two parts. One bell covers the first half of the domain, the other the other part

x in the middle of their domain

How I choose σ_i?

The bells have to overlap in a reasonable way ●

Having as free parameters the c_i, I can adjust a posteriori the bells ● (having the ×)

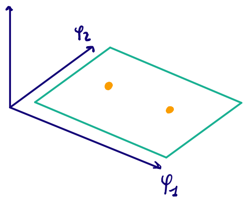


This was for $p = 2$.

But if I have to put 2 bells in this domain?

And if the bells are 7?

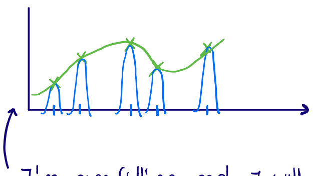
⇝ pseudo-random



Very most popular way to put the centers:

⇝ one data = one center

A bell for each data ⇝ very high risk of overfitting the data. We have to made very accurate choice of the width



I'm overfitting and I will not have a good approx.

Optimization of $\underline{\theta}$

$$J_N(\underline{\hat{\theta}}) = \frac{1}{N}\sum_{k=1}^{N}\underbrace{\left[y(k) - \hat{y}(k|\underline{\hat{\theta}})\right]^2}_{e^2(k|\underline{\hat{\theta}})}$$

$$\hat{y}(k|\underline{\hat{\theta}}) = \hat{f}\left[\underline{\varphi}(k), \underline{\hat{\theta}}\right]$$

$$\underline{\hat{\theta}}^* = \arg\max_{\underline{\hat{\theta}}} J_N(\underline{\hat{\theta}})$$

$$\text{Ⓐ} \quad y(k|\underline{\hat{\theta}}) = \sum_{i=1}^{L} c_i f_i\left[\underline{\varphi}(k)\right] = \underline{f}^T\left[\underline{\varphi}(k)\right]\underline{\hat{\theta}}$$

$$\begin{bmatrix} f_1\left[\underline{\varphi}(2)\right] \\ \vdots \\ f_L\left[\underline{\varphi}(2)\right] \end{bmatrix}$$

$$\leadsto \quad \tilde{\Phi}_N = \left[\, \underline{f}\left[\underline{\varphi}(1)\right] \,\middle|\, \underline{f}\left[\underline{\varphi}(2)\right] \,\middle|\, \dots \,\middle|\, \underline{f}\left[\underline{\varphi}(N)\right] \,\right]$$

$$\boxed{\hat{\underline{\theta}}^* = \left(\tilde{\Phi}_N \tilde{\Phi}_N^T\right)^{-1} \tilde{\Phi}_N \, y_N}$$

NON LINEAR ARX Ⓐ

compare with ARX:

$$\hat{y}(k|\hat{\underline{\theta}}) = \varphi^T(k)\,\hat{\underline{\theta}}$$

$$\Phi_N = \underbrace{\left[\varphi(1) \,\middle|\, \cdots \,\middle|\, \varphi(N)\right]}_{N} \Big\} \, (n+m+1) = p \underset{\text{ARX}}{\uparrow}$$

$$\hat{\underline{\theta}}^* = \left(\Phi_N \Phi_N^T\right)^{-1} \Phi_N \, \underline{y}_N$$

same structure but with $f[\varphi(k)] = \varphi(k)$

Ⓑ $$\hat{y}(k|\hat{\underline{\theta}}) = \hat{f}\left[\varphi(k), \hat{\underline{\theta}}\right] = \sum_{i=1}^{L} c_i \, f\left[\varphi(k), \underline{w}_i\right]$$

↑ one hidden layer (O.H.L)

NO one shot solution!

↓

mathematical programming

$\hat{\underline{\theta}}^{(0)}$ temptative solution

$\ell = 0, 1 \dots$

$\hat{\underline{\theta}}^{(\ell+1)} =$ function of ($\hat{\underline{\theta}}^{(\ell)}$, local info)

$$\left.\frac{\partial J_N(\hat{\underline{\theta}})}{\partial \hat{\underline{\theta}}}\right|_{\hat{\theta} = \hat{\theta}^{(\ell)}}$$

$$e(k|\hat{\underline{\theta}}) \quad , \quad \left.\frac{\partial e(k|\hat{\underline{\theta}})}{\partial \hat{\underline{\theta}}}\right|_{\hat{\underline{\theta}}^{(\ell)}} \quad k = 1 \dots N$$

$$e(k|\hat{\underline{\theta}}^{(\ell)}) = y(k) - \hat{y}(k|\hat{\underline{\theta}}^{(\ell)}) = y(k) - \hat{f}\left[\varphi(k), \hat{\underline{\theta}}^{(\ell)}\right]$$

$$\left.\frac{\partial e(k|\hat{\underline{\theta}})}{\partial \hat{\underline{\theta}}}\right|_{\hat{\underline{\theta}} = \hat{\underline{\theta}}^{(\ell)}} \rightsquigarrow \text{includes : } \frac{\partial e(k|\hat{\underline{\theta}})}{\partial c_i} \, , \, \frac{\partial e(k|\hat{\underline{\theta}})}{\partial \underline{w}_i}$$

$$\left.\frac{\partial e(k|\hat{\underline{\theta}})}{\partial \hat{\underline{\theta}}}\right|_{\hat{\underline{\theta}} = \hat{\underline{\theta}}^{(\ell)}} = -\frac{\partial}{\partial \hat{\underline{\theta}}} \hat{f}\left[\varphi(k), \hat{\underline{\theta}}\right]\Big|_{\hat{\underline{\theta}} = \hat{\underline{\theta}}^{(\ell)}}$$

$$\rightarrow \left.\frac{\partial}{\partial c_i} \hat{f}\left[\varphi(k), \hat{\theta}\right]\right|_{\hat{\theta}^{(\ell)}}$$

$$\left.\frac{\partial}{\partial \underline{w}_i} \hat{f}\left[\varphi(k), \hat{\underline{\theta}}\right]\right|_{\hat{\underline{\theta}}^{(\ell)}}$$

$$\left.\frac{\partial \hat{\gamma}[\varphi(k),\underline{\hat\theta}]}{\partial c_i}\right|_{\underline{\hat\theta}=\underline{\hat\theta}^{(\ell)}} = f[\varphi(k),\underline{w}_i]\Big|_{\underline{\hat\theta}=\underline{\hat\theta}^{(\ell)}} = f[\varphi(k),\underline{w}_i^{(\ell)}]$$

$$\underline{\hat\theta}^{(\ell)} = \text{col}\,[c_i^{(\ell)}, \underline{w}_i^{(\ell)}, i=1..L]$$

$$\left.\frac{\partial \hat{\gamma}[\varphi(k),\underline{\hat\theta}]}{\partial \underline{w}_i}\right|_{\underline{\hat\theta}=\underline{\hat\theta}^{(\ell)}} = c_i^{(\ell)} \left.\frac{\partial f[\varphi(k),\underline{w}_i]}{\partial \underline{w}_i}\right|_{\underline{\hat\theta}=\underline{\hat\theta}^{(\ell)}}$$

→ depends case by case on the particular function I'm using

8.5 N-ARX case

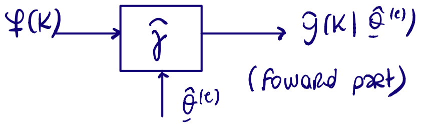
N-ARX

↳ Neural
Non-linear → we prefer this

(A) linearity in $\underline{\hat\theta}$ → one shot solution

(B) non lin. in $\underline{\hat\theta}$: $\left.\boxed{\frac{\partial \hat\gamma[\varphi(k),\underline{\hat\theta}]}{\partial \underline{\hat\theta}}}\right|_{\underline{\hat\theta}=\underline{\hat\theta}^{(\ell)}}$

computed with
Back Propagation (BP) → a lot used in machine learning



$\varphi(k)$
$\hat\gamma$
$\hat y(k|\underline{\hat\theta}^{(\ell)})$
$\underline{\hat\theta}^{(\ell)}$

(foward part)

now go back and compute the derivatives

$\hat y(k|\underline{\hat\theta})$ one step estimation of $y(k)$ given past values.

↓

in some book : $\hat y(k\,|\,k-1)$

y = quantity we want to estimate
$\hat{}$ = we are estimating

↳ information up to instant k-1

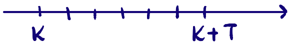
$$\hat y(k|\underline{\hat\theta}) = \left[1-\frac{\hat C(z)}{\hat D(z)}\right] y(k) + \frac{\hat B(z)\hat C(z)}{\hat A(z)\hat D(z)}\,u(k)$$

$$\hat y(k+1|\underline{\hat\theta}) = \left[1-\frac{\hat C(z)}{\hat D(z)}\right] y(k+1) + \frac{\hat B(z)\hat C(z)}{\hat A(z)\hat D(z)}\,u(k+1)$$

At istant k-1 we make the prediction of y at time k
$y(k\,|\,k-1)$

At istant k we make the prediction of y at time k+1
$y(k+1\,|\,k)$

$\hat{y}(k+T|k) \rightsquigarrow$ I want to estimate now at time k what $y(k+T)$ will be. Predict the future.



In this case I will need: $\mu(k), \mu(k+1) \dots \mu(k+T)$

9 State (and Parameter) Estimation

9.1 State equations

IDENTIFICATION

CONTROL

ESTIMATION

(Non lin.) sys in state equation:

$$\underline{x}(k+1) = \underline{f}_k[\underline{x}(k), \underline{\mu}(k)]$$

if you perfectly know this function ⇝ deterministic sys

- $\underline{x}(k+1) = \underline{f}_k[\underline{x}(k), \underline{\mu}(k), \underline{\theta}, \underline{\xi}(k)]$

"KSAI"

$\underline{\theta}$ = constant parameters

$\underline{\xi}(k)$ = random process noise

- $\underline{y}(k) = \underline{g}_k[\underline{x}(k), \underline{\theta}, \underline{\eta}(k)]$

Often the noise in $\underline{x}$ and in $\underline{y}$ have a different meaning. $\underline{\xi}(k) \neq \underline{\eta}(k)$

Simplified:

$$\begin{cases} \underline{x}(k+1) = \underline{f}_k(\underline{x}(k), \underline{\mu}(k), \underline{\theta}) + \underline{\xi}(k) \\ \underline{y}(k) = \underline{g}_k(\underline{x}(k), \underline{\theta}) + \underline{\eta}(k) \end{cases}$$

Linear case:

$$\begin{cases} x(k+1) = \overset{n_x \times n_x}{F}\, x(k) + \overset{n_x \times n_\mu}{W}\, \mu(k) \\ y(k) = \underset{n_y \times n_x}{G}\, x(k) \end{cases}$$

polinomi

matrici

⚠ NOT A, B, C, D

to differentiate from:

$$y(k) = \frac{B(z)}{A(z)}\mu(k) + \frac{D(z)}{C(z)} n(k)$$

$$T(z) \longrightarrow F, W, G$$

transfer function

$$F = \begin{bmatrix} 0 & & I & \\ \vdots & & & \\ 0 & & & \\ -a_0 & \dots & & -a_{n-1} \end{bmatrix} \quad W = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \quad G = [b_0 \ \dots \ b_{n-1}]$$

Not all the matrices are made of unknown parameters.

Most easy situation:

$$\begin{cases} x(k+1) = f\,x(k) + w\,\mu(k) \\ y(k) = g\,x(k) \end{cases}$$

scalar in this case

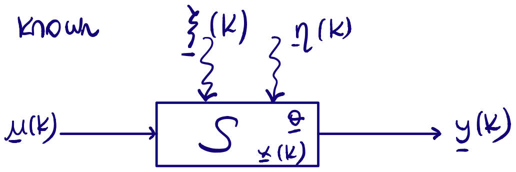
Suppose w, g known

$$\rightsquigarrow \quad x(k+1) = \underbrace{\theta\,x(k)}_{} + w\,\mu(k)$$

↓ non linearity in state and parameters (product is not linear)

How to estimate $\underline{\theta}$ (and $\underline{x}(k)$)

Assume f_k perfectly known



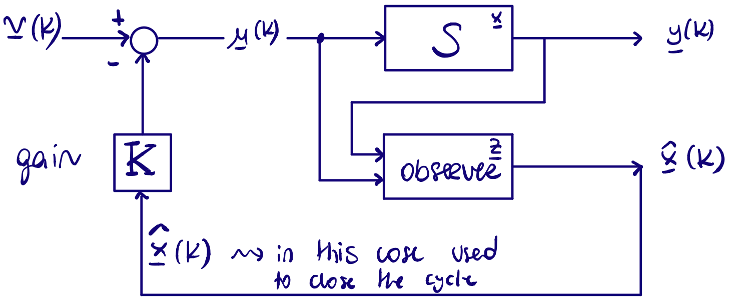
9.2 Observers

Methods to estimate the state of a system?

Luenberger observer, Kalman

Observer

$$\begin{cases} \underline{x}(k+1) = F\,\underline{x}(k) + W\,\underline{\mu}(k) \\ \underline{y}(k) = G\,\underline{x}(k) \end{cases}$$



9.3 Linear Luenberger

Luenberger:

$$\begin{cases} z(k+1) = F\, z(k) + W\, \mu(k) + \underset{\downarrow}{L} \left[y(k) - G\, z(k) \right] \\ \hat{x}(k) = z(k) \end{cases}$$

Luenberger gain introduced to correct the difference between z and x

z wants to estimate x

$$\left[y(k) - \overbrace{G\, z(k)}^{\hat{y}(k)} \right] = \text{innovation}$$

difference between output and estimated output

structure very similar to:

$$\underline{\theta}^*(k+1) = \hat{\underline{\theta}}(k) + \gamma(k) \left[y(k) - \hat{y}(k \mid \hat{\underline{\theta}}^*(k-1)) \right]$$

$$\underline{e}(k) \triangleq \underline{x}(k) - \hat{\underline{x}}(k) = x(k) - z(k)$$ error

$$\underline{e}(k+1) = \underline{x}(k+1) - z(k+1)$$

$$\begin{cases} z(k+1) = \tilde{F}\, z(k) + \tilde{W}\, \mu(k) + L\, y(k) \\ \hat{x}(k) = \cancel{\tilde{G}}_{I}\, z(k) \end{cases}$$ most general structure

It's an observer if the error goes to ∅.

$$e(k+1) = \underbrace{F\, x(k) + W\, \mu(k)}_{x(k+1)} - \underbrace{\tilde{F}\, z(k) - \tilde{W}\, \mu(k) - L\, y(k)}_{z(k+1)} =$$

$\tilde{W} = W$

$$= F\, x(k) - \tilde{F}\, z(k) - L G\, x(k) =$$

$\tilde{F} = F - LG$

$$= (F - LG)\, e(k)$$

So if L s.t. (F - LG) asy. stable (all the eigenvalues $|\cdot| < 1$)

$$\rightsquigarrow e(k+1) = (F - LG)\, e(k) \quad \forall\, x(0), \quad \boxed{e(k) \xrightarrow[k \to \infty]{} 0}$$

conditions

$$z(k+1) = (F - LG)\, z(k) + W \mu(k) + L\, y(k) =$$

$$= F\, z(k) + W\, \mu(k) + L \left[y(k) - G\, z(k) \right]$$

It's exactly what we have assumed.

9.4 Nonlinear Luenberger

The system is assumed to be described as (x and the others variables are all vectors $\boldsymbol{x}$ since the end of this part, but we used a lighter notation):

$$\mathcal{S}_{lin} : \begin{cases} x(k+1) & = Fx(k) + Wu(k) \\ y(k) & = Gx(k) \end{cases}$$

We have, as already seen, the Luenberger observer:

$$\mathcal{L}uen_{Obs} : \begin{cases} z(k+1) & = Fx(k) + Wu(k) + L[y(k) - Gz(u)] \\ \hat{x}(k) & = z(k) \end{cases}$$

For the non-linear case:

$$\mathcal{S}_{nonlin} : \begin{cases} x(k+1) & = f\left[x(k), u(k)\right] \\ y(k) & = g\left[x(k)\right] \end{cases}$$

We said that, in $\mathcal{L}uen_{Obs}$, $[Fx(k) + Wu(k)]$ can be interpreted as a prediction. If you have the possibility to measure the system but then you can't anymore, "you break the measurement tool" ($y(k) \rightarrow \nexists$) you use only $Gz(k) \rightarrow$ odometry: "were do I believe to be?". We propagate the "belief" on the robot position in time, we used the estimated position to predict it → no correction, but this is quite obvious, this is how we predict stuff. But if we propagate believes without innovation we're propagating also the errors (that's why $Gz(k) = \widehat{y(k)}$).

We are able to predict the next state given the true state and the exact control input in this purely theoretical framework. But if we have the predicted state $\hat{x}(k)$ instead of $x(k)$ we will not go in $x(k+1)$, but in the $\hat{x}(k+1|k)$.

Hence, the prediction $z(k+1)$ will be given by this "nonlinear Luenberger observer":

$$\mathcal{L}uen_{NonlinObs} : \begin{cases} z(k+1) & = f\left[z(k), u(k)\right] + L\left[y(k) - g(z(k))\right] \\ \hat{x}(k) & = z(k) \end{cases}$$

But how to choose the gain matrix L in the non-linear case? What some people do is to linearize the system at k:

$$F_k = \left.\frac{\partial}{\partial x} f(x, u)\right|_{x=z(k), u=u(k)}$$

$$W_k = \left.\frac{\partial}{\partial u} f(x, u)\right|_{x=z(k), u=u(k)}$$

$$G_k = \left.\frac{\partial}{\partial x} g(x)\right|_{x=z(k)}$$

But if you have the nonlinear model, why we do we have to linearize it?

In this case, one possibility is to use L at instant k as L_k such that $F_k - L_k G_k$ is asymptotically stable → maybe you can use the linearization for k, this is reasonable: we do not linearize first and then apply the linear observer, but we linearize the observer when we don't have other ideas.

9.5 Kalman filter

But now, let's consider a more general case, where matrices F, W, G can vary in time and where we have noises as well:

$$\mathcal{S}_{lin} : \begin{cases} x(k+1) & = F_k x(k) + W_k u(k) + \xi(k) \\ y(k) & = G_k x(k) + \eta(k) \end{cases}$$

In a **probabilistic content** you can describe the noises based by their probability density functions. We assume a gaussian distribution:[10]

$$\xi(k) \sim N(0, Q_k) \qquad \text{(Gaussian p.d.f. mean=0, variance } = Q_k)$$

Which are the unknown quantities? Since when we start the estimation we start from $x(0)$ even before any measurement, And what is the p.d.f. of $x(0)$?

$$x(0) \sim N(\alpha, \Sigma) \qquad \text{("where do you expect to be the measurement before actually doing it")}$$

We will introduce an already seen notation then: $\hat{v}$ will be the estimate of v, but the estimate referring to the instant k_1 made with all the information available at k_2:

$$\hat{v}(k_1|I(k_2)) \qquad I(k_2) = \text{ information available at instant } k_2 = \begin{Bmatrix} u(0), ..., u(k) \\ y(0), ..., y(k) \end{Bmatrix}$$

Notice that, while $\hat{x}(k|k)$ means something, $\hat{y}(k|k)$ does not mean anything! That is because there is no sense in estimating a y that is already $\in I(k)$.

In general:

$$v(k_1|k_2) = Exp\{v(k_1)|I\{k_2\}\} = \int v(k_1)pdf(v(k_1))dv(k_1)???$$

Gaussian Hp.: A-priori knowledge on $x(0)$ is $\hat{x}(0|-1)$

$$\begin{aligned} x(0|-1) &\sim N(\alpha, \Sigma) \\ \xi(k) &\sim N(0, Q_k) \\ \eta(k) &\sim N(0, R_k) \end{aligned}$$

These gaussian hypothesis with the system's linearity hypothesis allows us to say:

$$\begin{aligned} x(k|k-1) &\sim N\left[\hat{x}(k|k-1),\ \Sigma(k|k-1)\right] \\ x(k|k) &\sim N\left[\hat{x}(k|k),\ \Sigma(k|k)\right] \\ y(k|k-1) &\sim N\left[\hat{y}(k|k-1),\ S(k|k-1)\right] \end{aligned}$$

Similarly, we could have rewritten, instead α and Σ:

$$x(0|-1) \sim N(\alpha, \Sigma) = N[\hat{x}(0|-1), \Sigma(0|-1)]$$

The complete filter will be formed then by these equations:

$$\begin{cases} \hat{x}(k|k) = ? \\ \hat{x}(k|k-1) = ? \\ \hat{y}(k|k-1) = ? \\ H(z) = ? \\ \Sigma(k|k) = ? \\ \Sigma(k|k-1) = ? \\ S(k|k-1) = ? \end{cases}$$

How to compute $\hat{x}(k|k)$?

$$\hat{x}(k|k) = E\left\{ x(k) \middle| I(k) = \begin{Bmatrix} u(0), ..., u(k) \\ y(0), ..., y(k) \end{Bmatrix} \right\}$$

[10] In other context we can make some other kind of assumptions:

$$||\xi(k)|| \leq \varepsilon_\xi \qquad \textbf{("deterministic assumption on } \xi(k)\text{")}$$

This is called "deterministic" because there is no probabilistic connotation → is a bound on measurement uncertainty (like a measurement's tool sensibility).

Without any demonstration, let's see what is the shape of the estimator:

$$\hat{x}(k|k) = \underbrace{\hat{x}(k|k-1)}_{Prediction} + \underbrace{H(k)}_{Gain}[\underbrace{y(k) - \hat{y}(k|k-1)}_{Innovation}]$$

You can demonstrate that this is not optimal, but **true**, if the hypothesis hold.

How we compute the gain? With the equations structure we just saw (the covariances are needed, because not only the mean value is important, but how uncertainty is spread through the space):

$$\begin{cases} \hat{x}(k|k) & = \hat{x}(k|k-1) + H(k)[y(k) - \hat{y}(k|k-1)] \\ \hat{x}(k|k-1) & = \ ? \\ \hat{y}(k|k-1) & = \ ? \\ H(z) & = \ ? \\ \Sigma(k|k) & = \ ? \\ \Sigma(k|k-1) & = \ ? \\ S(k|k-1) & = \ ? \end{cases}$$

How to compute $\hat{x}(k|k-1)$?

$$x(k) = F_{k-1}\, x(k-1) + W_{k-1}\, u(k-1) + \xi(k-1)$$

$$\hat{x}(k|k-1) = E\left\{x(k)\,|I(k-1)\right\} =$$
$$= E\left\{F_{k-1}x(k-1|k-1) + W_{k-1}u(k-1) + \xi(k-1)\Big|I(k-1)\right\}$$

Since $u(k-1)$ is deterministic and the mean value of the noise ξ is 0 we have:

$$\hat{x}(k|k-1) = F_{k-1} \cdot E\left\{x\left(k-1\Big|I(k-1)\right)\right\} + W_{k-1}u(k-1) + \cancel{E\left\{\xi(k-1)|I(k-1)\right\}}$$

$$\hat{x}(k|k-1) = F_{k-1} \cdot \hat{x}\,(k-1|k-1) + W_{k-1}u(k-1)$$

Hence:

$$\begin{cases} \hat{x}(k|k) & = \hat{x}(k|k-1) + H(k)\big[y(k) - \hat{y}(k|k-1)\big] \\ \hat{x}(k|k-1) & = F_{k-1} \cdot \hat{x}\,(k-1|k-1) + W_{k-1}u(k-1) \\ \hat{y}(k|k-1) & = \ ? \\ H(z) & = \ ? \\ \Sigma(k|k) & = \ ? \\ \Sigma(k|k-1) & = \ ? \\ S(k|k-1) & = \ ? \end{cases}$$

$\Sigma(k|k-1)$ is the covariance of $x(k)$ computed w.r.t. the information $I(k-1)$. Recalling the Gaussian hypothesis we made:

$$\begin{aligned} x(0|-1) &\sim N(\alpha, \Sigma) \\ \xi(k) &\sim N(0, Q_k) \\ \eta(k) &\sim N(0, R_k) \end{aligned}$$

The prediction goes together with the uncertainty:

$$x(k|k-1) \sim N\Big(\hat{x}(k|k-1), \Sigma(k|k-1)\Big)$$

$$\Sigma(k|k-1) = cov\Big\{x(k)\Big|I(k-1)\Big\} =$$
$$= E\left\{\,[x(k) - \hat{x}(k|k-1)]\,[x(k) - \hat{x}(k|k-1)]^T\,\Big|\,I(k-1)\right\} =$$

$$\begin{aligned} &= E\Big\{ \left[F_{k-1}x(k-1) + W_{k-1}u(k-1) + \xi(k-1) - F_{k-1}\hat{x}(k-1|k-1) - W_{k-1}u(k-1)\right] \cdot \\ &\quad \cdot \Big[F_{k-1}x(k-1) + W_{k-1}u(k-1) + \xi(k-1) - F_{k-1}\hat{x}(k-1|k-1) - W_{k-1}u(k-1)]^T \Big| \, I(k-1) \Big\} \end{aligned}$$

$$\begin{aligned} = E\left\{ \left[F_{k-1}\Big(x(k-1) - \hat{x}(k-1|k-1)\Big)\Big(x(k-1) - \hat{x}(k-1|k-1)\Big)^T\right] F_{k-1}^T \middle| \, I(k-1) \right\} &+ \\ +E\left\{ F_{k-1}\Big(x(k-1) - \hat{x}(k-1|k-1)\Big)\xi^T(k-1) \middle| \, I(k-1) \right\} &+ \\ +E\left\{ \xi(k-1)\Big(x(k-1) - \hat{x}(k-1|k-1)\Big)^T F_{k-1}^T \middle| \, I(k-1) \right\} &+ \\ +E\left\{ \left(\xi(k-1) - 0\right)\left(\xi(k-1) - 0\right)^T \middle| \, I(k-1) \right\} &= \end{aligned}$$

Since F_{k-1} is a linear operator and ξ has zero mean:

$$\begin{aligned} = F_{k-1}\, E\left\{ \left[\Big(x(k-1) - \hat{x}(k-1|k-1)\Big)\Big(x(k-1) - \hat{x}(k-1|k-1)\Big)^T\right] \middle| \, I(k-1) \right\} F_{k-1}^T &+ \\ F_{k-1}\, E\left\{ \Big(x(k-1) - \hat{x}(k-1|k-1)\Big) \middle| \, I(k-1) \right\} \cancel{\left\{\xi^T(k-1)\middle| I(k-1)\right\}} &+ \\ +\cancel{\left\{\xi^T(k-1)\middle| I(k-1)\right\}} E\left\{ \Big(x(k-1) - \hat{x}(k-1|k-1)\Big)^T \middle| \, I(k-1) \right\} &+ \\ +E\left\{ \left(\xi(k-1) - 0\right)\left(\xi(k-1) - 0\right)^T \middle| \, \cancel{I(k-1)} \right\} &= \end{aligned}$$

We have $\cancel{I(k-1)}$ because $\xi(k-1)$ and $I(k-1)$ are **uncorrelated**. Moreover, the previous second and third line are removed because $\left\{\xi^T(k-1)\middle| I(k-1)\right\} = 0$. We notice also that the last row is the covariance Q of ξ noise:

$$Q(k-1|k-1) = Q_{k-1} = E\left\{ \left(\xi(k-1) - 0\right)\left(\xi(k-1) - 0\right)^T \middle| \right\}$$

$$\textbf{!!!} \quad \begin{bmatrix} \text{Notation for covariance matrices:} & A(k-1|k-1) & = & A_{k-1} \\ & A(k|k) & = & A_k \end{bmatrix}$$

Hence we have $\Sigma(k|k-1) =$

$$\begin{aligned} = F_{k-1}\, \underbrace{E\left\{ \left[\Big(x(k-1) - \hat{x}(k-1|k-1)\Big)\Big(x(k-1) - \hat{x}(k-1|k-1)\Big)^T\right] \middle| \, I(k-1) \right\}}_{\Sigma_{k-1}} F_{k-1}^T \ &+ \\ +\ 0\ &+ \\ +\ 0\ &+ \\ +\ Q_{k-1}& \end{aligned}$$

Let's talk about what $E\left\{\xi^T(k-1)\middle| I(k-1)\right\} = 0$ implies. Since we cancel out ξ from the estimate, even if we have a very small covariance at the start, the disturbance will propagate, because it will not be possible to cancel out Q_{k-1}:

$$\Sigma(k|k-1) = F_{k-1}\Sigma\left(k-1\middle|k-1\right) F_{k-1}^T + Q_{k-1}$$

So now we have:

$$\begin{cases} \hat{x}(k|k) & = \hat{x}\left(k\middle|k-1\right) + H(k)\Big[y(k) - \hat{y}\left(k\middle|k-1\right)\Big] \\ \hat{x}(k|k-1) & = F_{k-1}\hat{x}\left(k-1\middle|k-1\right) + W_{k-1}u(k-1) \\ \hat{y}(k|k-1) & = \ ? \\ H(z) & = \ ? \\ \Sigma(k|k) & = \ ? \\ \Sigma(k|k-1) & = F_{k-1}\Sigma(k-1|k-1)F_{k-1}^T + Q_{k-1} \\ S(k|k-1) & = \ ? \end{cases}$$

Let's focus now on the predicted output:

$$\hat{y}(k|k-1) = E\Big\{y(k)\Big|I(k-1)\Big\} = E\Big\{G_k x(k)+\eta(k)\Big|I(k-1)\Big\} = G_k\ E\Big\{x(k)\Big|I(k-1)\Big\}+G_k\ \cancel{E\Big\{\eta(k)\Big|I(k-1)\Big\}} =$$

Hence

$$\hat{y}(k|k-1) = G_k\ \ \hat{x}(k|k-1)$$

And the covariance is

$$S(k|k-1) = cov\{y(k)|I(k-1)\} = E\Big\{\ [y(k) - \hat{y}(k|k-1)]\,[y(k) - \hat{y}(k|k-1)]^T\ \Big|I(k-1)\Big\}$$

Then, avoiding the computations that are analogue to the previous case (R is the covariance of η noise:):

$$S(k|k-1) = G_k\Sigma(k|k-1)G_k^T + R_k$$

Now the situation is the following

$$\begin{cases} \hat{x}(k|k) & = \hat{x}\left(k\middle|k-1\right) + H(k)\Big[y(k) - \hat{y}\left(k\middle|k-1\right)\Big] \\ \hat{x}(k|k-1) & = F_{k-1}\ \hat{x}\left(k-1\middle|k-1\right) + W_{k-1}u(k-1) \\ \hat{y}(k|k-1) & = G_k\ \ \hat{x}(k|k-1) \\ H(z) & = \ ? \\ \Sigma(k|k) & = \ ? \\ \Sigma(k|k-1) & = F_{k-1}\Sigma(k-1|k-1)F_{k-1}^T + Q_{k-1} \\ S(k|k-1) & = G_k\ \ \ \Sigma(k|k-1)\ \ G_k^T\ \ + R_k \end{cases}$$

We have also, for the gain H of the Kalman filter:

$$H(z) = \Sigma(k|k-1)G_k^T S(k|k-1)^{-1}$$

this means, if we have a good prediction ($\Sigma(k|k-1)$ small) we need a lower gain. Also, if we trust more the new measurements ($S(k|k-1)$ small) we will have a bigger gain, while increasing R_k increases the uncertainty on new measurement $S(k|k-1)$, hence the gain $H(k)$ decreases.

How to recall the dimensions of $H(z)$ expression? → see MobRo course!

Let's check what is missing:

$$\begin{cases} \hat{x}(k|k) & = \hat{x}\left(k\middle|k-1\right) + H(k)\Big[y(k) - \hat{y}\left(k\middle|k-1\right)\Big] \\ \hat{x}(k|k-1) & = F_{k-1}\ \hat{x}\left(k-1\middle|k-1\right) + W_{k-1}u(k-1) \\ \hat{y}(k|k-1) & = G_k\ \ \hat{x}(k|k-1) \\ H(z) & = \Sigma(k|k-1)G_k^T S(k|k-1)^{-1} \\ \Sigma(k|k) & = \ ? \\ \Sigma(k|k-1) & = F_{k-1}\Sigma(k-1|k-1)F_{k-1}^T + Q_{k-1} \\ S(k|k-1) & = G_k\ \ \ \Sigma(k|k-1)\ \ G_k^T\ \ + R_k \end{cases}$$

We're missing $\Sigma(k|k)$. This can be computed as:

$$\Sigma(k|k) = \Sigma(k|k-1) - H(k)S(k|k-1)H(k)^T$$

This result is quite intuitive. In fact, what makes me decrease the covariance? The measurements, if not, there is a problem in my system.

Initialization If you are "the angry wife waiting for the husband to enter the house", the first position you'll look will be the door. The analogy with the initialization holds by means of the following expressions:

$$\hat{x}(0|-1) = \alpha \qquad \Sigma(0|-1) = \Sigma$$

Then, we will have

$$\begin{cases} \hat{x}(0|0) & = \alpha + H(0)[y(0) - \hat{y}(0|-1)] \\ \hat{x}(0|-1) & = \alpha \\ \hat{y}(k|k-1) & = G_0 \ \alpha \\ H(z) & = \Sigma G_0^T S_0^{-1} \\ \Sigma(0|0) & = \Sigma - H(0) S_0 H(0)^T \\ \Sigma(0|-1) & = \Sigma \\ S_0 & = G_0 \ \Sigma \ G_0^T \ + R_0 \end{cases}$$

Hence, the **kalman filter** complete structure is (put here in executive order):

$$\boxed{\begin{cases} & \{F_k \ , \ W_k \ , \ G_k \ , \ Q_k \ , \ R_k\} \\ \hat{x}(0|-1) & = \alpha \\ \hat{\Sigma}(0|-1) & = \Sigma \\ \hat{x}(k|k-1) & = F_{k-1} \ \hat{x}\left(k-1|k-1\right) + W_{k-1} u(k-1) \\ \Sigma(k|k-1) & = F_{k-1} \Sigma(k-1|k-1) F_{k-1}^T + Q_{k-1} \\ \hat{y}(k|k-1) & = G_k \quad \hat{x}(k|k-1) \\ S(k|k-1) & = G_k \quad \Sigma(k|k-1) \ \ G_k^T \ + R_k \\ H(z) & = \Sigma(k|k-1) G_k^T S(k|k-1)^{-1} \\ \hat{x}(k|k) & = \hat{x}\left(k|k-1\right) + H(k)\Big[y(k) - \hat{y}\left(k|k-1\right)\Big] \\ \Sigma(k|k) & = \Sigma(k|k-1) - H(k) S(k) H(k)^T \end{cases}}$$

The quantities $\hat{x}(k|k-1)$ and $\hat{x}(k|k)$ have to be computed online, but it is ok to compute them offline if F_k, W_k, G_k, Q_k, and R_k are *a priori* known (e.g. in a time invariant framework)

9.6 Extended Kalman filter

Consider a non-linear system:

$$\mathcal{S}_{nonlin} : \begin{cases} x(k+1) & = f\left[x(k), u(k)\right] + \xi(k) \\ y(k) & = g\left[x(k)\right] + \eta(k) \end{cases}$$

$$\text{apply Kalman F.} \rightarrow \hat{x}(k|k) = \hat{x}(k|k-1) + H(z)[y(k) - \hat{y}(k|k-1)]$$

Assume $\hat{x}(k-1|k-1)$ given $x(k+1) = f\left[x(k), u(k)\right] + \xi(k)$, we will have these approximated results

$$\begin{cases} \hat{x}(k|k-1) & = f_{k-1}\left[\hat{x}(k-1|k-1), u(k)\right] \\ \hat{y}(k|k-1) & = g_k\left[\hat{x}(k|k-1)\right] \\ H(k) & = \Sigma(k|k-1) G_k^T S(k|k-1)^{-1} \end{cases}$$

We take the point and we linearize it around the best value (that is the prediction $\hat{x}(k|k-1)$):

$$G_k \ := \frac{\partial}{\partial x} g_k(x) \Bigg|_{x=\hat{x}(k|k-1)} \qquad \Sigma(k|k-1) = F_{k-1}\Sigma(k-1|k-1)F_{k-1}^T + Q_{k-1}$$

$$F_{k-1} := \frac{\partial}{\partial x} f_{k-1}(x)\Bigg|_{x=\hat{x}(k-1|k-1),\ u=u(k-1)} \qquad S(k|k-1) = G_k \Sigma(k|k-1) G_k^T + R_k$$

$$\Sigma(k|k) = \Sigma(k|k-1) - H(k)S(k)H(k)^T$$

Gaussianity **no more true** (because of the nonlinearity of the system) → not anymore the zero mean and only reasonable approximated covariances.

For sure the quantities has to be computed online, and the matrices as well!

$$\Sigma(k|k) = \text{(approximate) measurement of uncertainty}$$

Hence, the **Extended Kalman Filter** structure is:

$$\begin{cases} & \{W_k\ ,\ Q_k\ ,\ R_k\} \\ \hat{x}(0|-1) & = \alpha \\ \hat{\Sigma}(0|-1) & = \Sigma \\ \hat{x}(k|k-1) & = f_{k-1}\left[\hat{x}(k-1|k-1), u(k)\right] \\ F_{k-1} & = \frac{\partial}{\partial x} f_{k-1}(x)\big|_{x=\hat{x}(k-1|k-1),\ u=u(k-1)} \\ \Sigma(k|k-1) & = F_{k-1}\Sigma(k-1|k-1)F_{k-1}^T + Q_{k-1} \\ \hat{y}(k|k-1) & = g_k\left[\hat{x}(k|k-1)\right] \\ G_k & = \frac{\partial}{\partial x} g_k(x)\big|_{x=\hat{x}(k|k-1)} \\ S(k|k-1) & = G_k \quad \Sigma(k|k-1)\ \ G_k^T\ \ + R_k \\ H(k) & = \Sigma(k|k-1)G_k^T S(k|k-1)^{-1} \\ \hat{x}(k|k) & = \hat{x}\left(k\middle|k-1\right) + H(k)\Big[y(k) - \hat{y}\left(k\middle|k-1\right)\Big] \\ \Sigma(k|k) & = \Sigma(k|k-1) - H(k)S(k)H(k)^T \end{cases}$$

In **linear case**, the kalman filter is **BLUE** (Best Linear Unbiased Estimator → you can't find a better estimator in term of minimum varianc in the error), this **also without gaussian hypothesis**.

For **non-linear cases**, the Extended Kalman Filter is in general no more optimal but even not BLUE (is not a linear filter). The only thing that we can say is that under very specific hypotheses it converges, but in general it is just a **reasonable estimator** ("in general it works but you don't have any guarantee of convergence, perhaps it diverges").

If some parameter is unknown but the functions are known:

$$\text{grey-box} : \begin{cases} x(k+1) & = f\left[x(k), u(k), \theta\right] + \xi(k) \\ y(k) & = g\left[x(k), \theta\right] + \eta(k) \end{cases}$$

If even the functions are unknown:

$$\text{black-box} : \begin{cases} f_k \longrightarrow \hat{\gamma}_k \\ g_k \longrightarrow \hat{\gamma}_k \end{cases}$$

Consider the grey-box case. Now, let's change the **notation**, using $\tilde{x}$ instead of x and let's do it as well for the functions

$$\text{grey-box (new notation)} : \begin{cases} \tilde{x}(k+1) & = \tilde{f}\left[x(k), u(k), \theta\right] + \xi(k) \\ y(k) & = \tilde{g}\left[x(k), \theta\right] + \eta(k) \end{cases}$$

Let's perform now a state redefinition (**state augmentation**):

$$x(k) := \begin{bmatrix} \tilde{x}(k) \\ \theta(k) \end{bmatrix} \Rightarrow \begin{cases} \begin{bmatrix} \tilde{x}(k+1) \\ \theta(k+1) \end{bmatrix} & = \begin{bmatrix} \tilde{f}\left[x(k), u(k), \theta\right] \\ \theta(k) \end{bmatrix} + \begin{bmatrix} \xi(k) \\ 0 \end{bmatrix} \\ y(k) & = \tilde{g}\left[x(k), \theta\right] + \eta(k) \end{cases}$$

Hence we can assign the "old" notation to these new matrices:

$$\begin{cases} x(k+1) = \begin{bmatrix} \tilde{x}(k+1) \\ \theta(k+1) \end{bmatrix} & = f\left[x(k), u(k), \theta\right] + \xi(k) \\ y(k) & = \tilde{g}\left[x(k), \theta\right] + \eta(k) \end{cases}$$

Using the EKF

$$\hat{x}(k|k) = \begin{bmatrix} \tilde{x}(k|k) \\ \hat{\theta}(k|k) \end{bmatrix} \qquad \Sigma(k|k) = \begin{bmatrix} \Sigma_{xx}(k|k) & \Sigma_{x\theta}(k|k) \\ \Sigma_{\theta x}(k|k) & \Sigma_{\theta\theta}(k|k) \end{bmatrix}$$

Consider also a linear system:

$$\begin{cases} \tilde{x}(k+1) & = a\tilde{x}(k) + bu(k) + \xi(k) \\ \tilde{y}(k) & = c\tilde{x}(k) + \eta(k) \end{cases}$$

If we augment the state, we obtain a nonlinear system → hence all the BLUE assumptions are lost.