2.4 Systems identification: subspace and nonlinear methods

Dr. Riccardo Bacci di Capaci

University of Pisa, Italy
riccardo.bacci@unipi.it

GRICU PhD School 2021
Digitalization Tools for the Chemical and Process Industries
March 12, 2021

Outline

What is systems identification ?

Conventional systems identification methods - ***not our focus (see 2.3 Linear Methods)***

- FIR models via least-squares
- ARX models via least-squares

Advanced systems identification methods - our main focus

- Prediction Error Methods for input-output systems
- Iterative Least-Squares
- Recursive Least-Squares
- Nonlinear Models
- An industry example: valve stiction
- Subspace Identification Methods

Complementaries: Input design, data collection, model performance - **very quickly**

- Conventional data collection via step tests
- Advanced data collection methods: OL & CL data collection
- Information criteria & Model validation

Identification software packages - **to do some practice**

- The Literature & our SIPPY
- Exercises: solved and proposed

Conclusions

Objectives and main ingredients of systems identification

Objectives

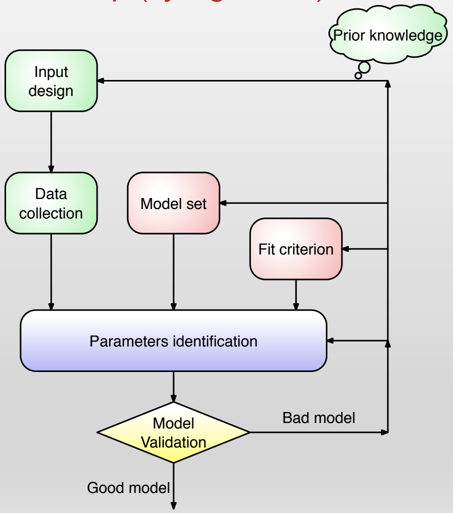
Systems identification is concerned with the **determination of a dynamic model** of the considered process given experimental (input and output) **data**

Three fundamental ingredients

1. **Data set**: Input (manipulated MVs or disturbance DVs variables) and output (CVs) can be collected during **specific** identification campaigns or during normal plant operation
2. **Model set**: A family of candidate dynamic models among which the **optimal** model will be selected
3. **Identification algorithm**: A numerical method to calculate the model parameters and obtain the optimal model

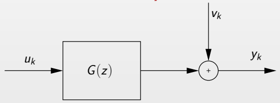
The typical procedure

The system identification loop (Ljung, 1999)



Some preliminary definitions

The reference model (in discrete-time)



Disturbance model

- By definition, the disturbance sequence $\{v\}$ is **not predictable** a priori
- Often we assume that:

$$v_k = H(z)e_k$$

in which:

- $H(z)$ is a stable Transfer Function (TF)
- e_k is zero mean, white noise with variance λ: $\lambda = \mathcal{E}\left(e_k^2\right)$

The basic relation

$$y_k = G(z)u_k + H(z)e_k$$

we want to determine two TFs: $G(z)$ and $H(z)$ given measured sequences $\{u\}$ and $\{y\}$

Some useful definitions and results

Probability quantities

- Expected Value of a random variable v:

$$\mathcal{E}\left(v_k\right) = \mathcal{E}\left(H(z)e_k\right) = \sum_{j=0}^{\infty} h_j \mathcal{E}(e_{k-j}) = 0$$

in which $H(z) = \sum_{j=0}^{\infty} h_j z^{-j}$

- Auto-correlation of v:

$$R_v(\tau) = \mathcal{E}\left(v_k v_{k-\tau}\right) = \mathcal{E}\left(\sum_{j=0}^{\infty} h_j z^{-j} e_k \sum_{j=0}^{\infty} h_j z^{-j} e_{k-\tau}\right)$$

$$= \mathcal{E}\left(\sum_{j=\tau}^{\infty} h_j h_{j-\tau} z^{-j} e_k^2\right) = \sum_{j=\tau}^{\infty} h_j h_{j-\tau} \mathcal{E}(e_{k-j}) = \lambda \sum_{j=\tau}^{\infty} h_j h_{j-\tau}$$

- When the autocorrelation function does **not depend on** k, the signal v is said to be **stationary**

Input/output models: Linear vs. Nonlinear methods

Model structures, black-boxes and possible identification methods

Model structure	Polynomials in z		Id. Method
	$G(z)$	$H(z)$	
FIR	$B(z)$	1	Linear (e.g. **LLS**); but also NL
ARX	$A^{-1}(z)B(z)$	$A^{-1}(z)$	
ARMAX	$A^{-1}(z)B(z)$	$A^{-1}(z)C(z)$	**Nonlinear - Advanced (e.g. PEM, ILLS, RLLS)**
ARMA	1	$A^{-1}(z)C(z)$	
ARARX	$A^{-1}(z)B(z)$	$A^{-1}(z)D^{-1}(z)$	
ARARMAX	$A^{-1}(z)B(z)$	$A^{-1}(z)D^{-1}(z)C(z)$	
OE	$F^{-1}(z)B(z)$	1	
BJ (Box-Jenkins)	$F^{-1}(z)B(z)$	$D^{-1}(z)C(z)$	
GEN (Generalized)	$A^{-1}(z)F^{-1}(z)B(z)$	$A^{-1}(z)D^{-1}(z)C(z)$	

Linear Methods: FIR model for SISO systems

Ideal and practical Finite Impulse Response model

- The **ideal convolution** model in discrete time is:

$$y_k = \sum_{j=0}^{\infty} h_j u_{k-j}$$

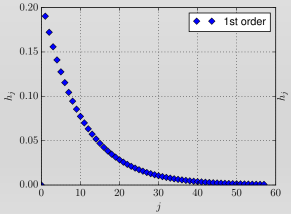
  with $\{h_j\}$ coefficients of the finite impulse response

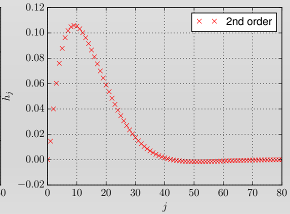
- For **open-loop stable** systems, it follows that: $\lim_{j\to\infty} h_j = 0$

- The **practical FIR** model is limited:

$$y_k = \sum_{j=0}^{M} h_j u_{k-j}$$

  where $M > 0$ is the **model horizon**





FIR model: identification via least-squares

Linear predictor construction

- Assume N **input** and **output data** are available: $[u_0, \ldots, u_N]$, $[y_0, \ldots, y_N]$
- For **each** $k \geq M$, write (note that usually $h_0 = 0$):

$$y_k = h_1 u_{k-1} + h_2 u_{k-2} + \cdots + h_M u_{k-M} + e_k = \varphi_k \theta + e_k$$

where: $\varphi_k = \begin{bmatrix} u_{k-1} & u_{k-2} & \cdots & u_{k-M} \end{bmatrix}$, and $\theta = \begin{bmatrix} h_1 & h_2 & \cdots & h_M \end{bmatrix}^\top$
are **regressor** and **parameter** vectors, respectively

- Stack all terms for $k = M, \ldots, N$:

$$\begin{bmatrix} y_M \\ y_{M+1} \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} \varphi_M \\ \varphi_{M+1} \\ \vdots \\ \varphi_N \end{bmatrix} \theta + \begin{bmatrix} e_M \\ e_{M+1} \\ \vdots \\ e_N \end{bmatrix} \Rightarrow y = \phi\theta + e$$

Least-squares problem and solution

- **Mean Square Error** (MSE) loss function:

$$V_{LS}(\theta) = \frac{1}{N} \sum_{k=M}^{N} e_k^2 = \frac{1}{N} (y - \phi\theta)^\top (y - \phi\theta)$$

- Well known **solution** (with **pseudo-inverse**): $\theta = (\phi^\top \phi)^{-1} \phi^\top y = \phi^+ y$

Multivariable FIR model

Extension to Multiple Input Multiple Output (MIMO) systems

- Consider a system with m **inputs** $(u^{(1)}, u^{(2)}, \dots, u^{(m)})$ and p **outputs** $(y^{(1)}, y^{(2)}, \dots, u^{(p)})$
- For **each output** $^{(}i)$, a Multiple Input Single Output (**MISO**) approach is used:

$$
\begin{aligned}
y_k^{(i)} &= \sum_{j=1}^{M} h_j^{(i1)} u_{k-j}^{(1)} + \sum_{j=1}^{M} h_j^{(i2)} u_{k-j}^{(2)} + \cdots + \sum_{j=1}^{M} h_j^{(im)} u_{k-j}^{(m)} + e_k^{(i)} \\
&= \varphi_k^{(1)} \theta^{(i1)} + \varphi_k^{(2)} \theta^{(i2)} + \cdots + \varphi_k^{(m)} \theta^{(im)} + e_k^{(i)}
\end{aligned}
$$

- **Stacking** all terms for $k = M, \dots, N$,
  with $\theta^{(i)} = \begin{bmatrix} \theta^{(i1)} & \dots & \theta^{(im)} \end{bmatrix}^{\top}$ and $\varphi_k = \begin{bmatrix} \varphi_k^{(1)} & \cdots & \varphi_k^{(m)} \end{bmatrix}$

$$
y^{(i)} = \phi \theta^{(i)} + e^{(i)} \Rightarrow \theta^{(i)} = \phi^{+} y^{(i)}
$$

Input and output relations

- The user defines **which inputs affect** the response of each output $y^{(i)}$
- This input/output relations are **decided** using **preliminary tests**

Comments of the FIR model

Good features of FIR models

- Very **little prior knowledge** is required, except which **input/output** coefficients need to be determined
- It is statistically **unbiased** and **consistent**

Bad features of FIR models

- It is **over-parameterized**, and can be **noise sensitive** because the regressor matrix ϕ is often **ill-conditioned**
- It is a (very) **high-order** model: **order reduction** may be necessary

Extension to measurable disturbances

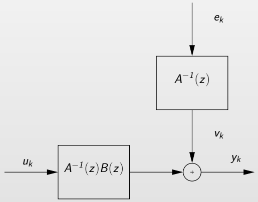
- Measurable disturbances d are treated as **additional inputs** of the MISO structure:

$$y_k^{(i)} = \varphi_k^{(1)}\theta^{(i1)} + \varphi_k^{(2)}\theta^{(i2)} + \cdots + \varphi_k^{(m)}\theta^{(im)} + \cdots + \varphi_k^{(m+n_d)}\theta^{(i(m+n_d))} + e_k^{(i)}$$

ARX model for SISO systems: description

ARX (**A**uto**R**egressive with e**X**ternal Inputs) model

- Scheme:



- Model:

$$A(z)y_k = B(z)u_k + e_k$$

- Polynomials (**SISO** case):

$$A(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_{n_a} z^{-n_a}$$
$$B(z) = b_1 z^{-(\ell+1)} + b_2 z^{-(\ell+2)} + \cdots + b_{n_b} z^{-(\ell+n_b)}$$

- $[n_a,\ n_b]$ are the **model orders** and ℓ is the time-delay, all **user defined**

ARX model for SISO systems: identification

"Equation error model"

The error e_k enters directly the difference equation:

$$y_k + a_1 y_{k-1} + \cdots + a_{n_a} y_{k-n_a} = b_1 u_{k-\ell-1} + \cdots + b_{n_b} u_{k-\ell-n_b} + e_k$$

Regressor and LS Solution

- **Linear regressor** ϕ:

$$\hat{y}_k = \underbrace{\begin{bmatrix} -y_{k-1} & \cdots & -y_{k-n_a} & u_{k-\ell-1} & \cdots & u_{k-\ell-n_b} \end{bmatrix}}_{\varphi_k} \underbrace{\begin{bmatrix} a_1 \\ \vdots \\ a_{n_a} \\ b_1 \\ \vdots \\ b_{n_b} \end{bmatrix}}_{\theta}$$

- Stack all terms for $k = n_m, \ldots, N$ with $n_m = \max(n_a, n_b + \ell) \Rightarrow y = \phi\theta + e$
- **LS** solution as for the FIR model: $\theta = (\phi^\top \phi)^{-1} \phi^\top y = \phi^+ y$

ARX identification for MIMO systems

Multi-variable "equation error" model

- Model (for simplicity time-delay ℓ is omitted):

$$y_k + A_1 y_{k-1} + \cdots + A_{n_a} y_{k-n_a} = B_1 u_{k-1} + \cdots + B_{n_b} u_{k-n_b} + e_k$$

- where $A_i \in \mathbb{R}^{p\times p}$ (for $i = 1, \ldots, n_a$) and $B_i \in \mathbb{R}^{p\times m}$ (for $i = 1, \ldots, n_b$); y, u, e are all vectors, e.g. $u_{k-1} = \left[u_{k-1}^{(1)}, u_{k-1}^{(2)}, \ldots, u_{k-1}^{(m)}\right]^T \in \mathbb{R}^m$

Regressor and LS Solution

- **Linear** regressor

$$\hat{y}_k = \underbrace{\begin{bmatrix} A_1 & \cdots & A_{n_a} & B_1 & \cdots & B_{n_b} \end{bmatrix}}_{\Theta^\top} \underbrace{\begin{bmatrix} -y_{k-1} \\ \vdots \\ -y_{k-n_a} \\ u_{k-1} \\ \vdots \\ u_{k-n_b} \end{bmatrix}}_{\varphi_k^\top}$$

- As for **FIR** model, **MISO** approach and **LS** solution:

$$y^{(i)} = \phi\theta^{(i)} + e^{(i)} \Rightarrow \theta^{(i)} = \phi^+ y^{(i)}$$

Advanced (Prediction Error) Methods: preliminaries

Motivation

- Linear methods and ARX/FIR models can be too **simplistic**, in terms of **noise description**
- **Advanced model**: increase flexibility by describing equation error $H(z)$ with a proper dynamics of the white noise e_k

Features (of PEMs)

- Given the output model $\hat{y}_k$, define the **prediction error** $\epsilon_k = y_k - \hat{y}_k$
- Often the prediction error **is filtered**: $\epsilon_k^F = L(z)\epsilon_k$, where $L(z)$ is a suitable TF which acts as a frequency filter
- Given data for N sampling times, **the loss function** is:

$$V_N = \frac{1}{N}\sum_{k=1}^{N} \mathcal{L}(\epsilon_k^F)$$

where $\mathcal{L}(\cdot)$ is a non-negative scalar function

Prediction Error Methods: details

The optimization problem

- Chosen the **model structure**, the predictor has the form: $\hat{y}_k = \varphi_k^T(\theta)\theta$
- The coefficient vector (θ) has a **nonlinear effect** in the regressor vector φ_k so that linearity is lost
- **PEM** solve an **optimization problem**:

$$\hat{\theta}_N = \arg\min_{\theta \in \mathcal{D}} V_N$$

- Depending on the choice of function $\mathcal{L}(\cdot)$ and of the model structure, the above problem can be a simple Quadratic Program (**QP**) or a more involved **NLP**
- For specific classes of model, **ad hoc** optimization algorithms were developed
- Note that these NL problems maybe **very large** for MIMO systems with evident computational issues
- Anyway, identification of input-output systems is always **MISO**

Prediction Error Methods: Model Validation

Predictors

- Given the general model: $y_k = G(z)u_k + H(z)e_k$
- the prediction error: $e_k := \epsilon_k = y_k - \hat{y}_k$

we can define:

1-step-ahead predictor

$$y_k = G(z)u_k + H(z)(y_k - \hat{y}_k) \quad \Rightarrow \quad H^{-1}(z)\hat{y}_k = G(z)u_k + (H(z) - 1)y_k$$

$$\hat{y}_k = H^{-1}(z)G(z)u_k + (1 - H^{-1}(z))y_k$$

k-step-ahead predictor

$$\hat{y}_k = W_k(z)G(z)u_k + (1 - W_k(z))y_k$$

where:

$$W_k(z) = \bar{H}_k(z)H^{-1}(z); \quad \bar{H}_k(z) = \sum_{j=0}^{k-1} h_j z^{-j}$$

being $\{h_j\}$ the coefficients of the finite impulse response of TF $H(z)$

ARMAX model

AutoRegressive Moving Average with eXternal inputs model

- The difference equation is:

$$y_k + a_1 y_{k-1} + \cdots + a_{n_a} y_{k-n_a} = b_1 u_{k-\ell-1} + \cdots + b_{n_b} u_{k-\ell-n_b} + e_k + c_1 e_{k-1} + \cdots + c_{n_c} e_{k-n_c}$$

- **noise model**: the error e_k has its own dynamics (as **moving average**)
- Polynomial form:

$$A(z) y_k = B(z) u_k + C(z) e_k$$

with:

$$C(z) = 1 + c_1 z^{-1} + c_2 z^{-2} + \cdots + c_{n_c} z^{-n_c}$$

- Observation: in this model $G(z)$ **and** $H(z)$ **have same poles**, given that:

$$G(z) = A^{-1}(z) B(z), \qquad H(z) = A^{-1}(z) C(z)$$

- ARMAX can be identified via various **nonlinear** methods, PEMs (see later...)

Other "Advanced" input/output models

Equation-Error-Type

Different **error models**:

- "**ARMA**" model:
  $$A(z)y_k = C(z)e_k$$
  i.e. $G(z) = 1, \quad H(z) = A^{-1}(z)C(z)$
  Moving Average, but no eXternal part
- "**ARARX**" model:
  $$A(z)y_k = B(z)u_k + D(z)^{-1}e_k$$
  i.e. $G(z) = A^{-1}(z)B(z), \quad H(z) = A^{-1}(z)D^{-1}(z)$
  a specific AutoRegressive
- "**ARARMAX**" model:
  $$A(z)y_k = B(z)u_k + D(z)^{-1}C(z)e_k$$
  i.e. $G(z) = A^{-1}(z)B(z), \quad H(z) = A^{-1}(z)D^{-1}(z)C(z)$
  a specific ARMAX structure

Other “Advanced” input/output models

Output-Error-Type

When $G(z)$ and $H(z)$ are parametrized **independently**, no AutoRegressive part ($A(z)$) is used:

- “**Output-Error**” (OE) model:

$$y_k = F^{-1}(z)B(z)u_k + e_k$$

i.e. $G(z) = F^{-1}(z)B(z), \quad H(z) = 1$

- “**Box-Jenkins**” (BJ) model:

$$y_k = F^{-1}(z)B(z)u_k + D^{-1}(z)C(z)e_k$$

i.e. $G(z) = F^{-1}(z)B(z), \quad H(z) = D^{-1}(z)C(z)$
$G(z)$ and $H(z)$ have **different poles**

General model:

$$A(z)y_k = F^{-1}(z)B(z)u_k + D^{-1}(z)C(z)e_k$$

i.e. $G(z) = A^{-1}(z)F^{-1}(z)B(z), \quad H(z) = A^{-1}(z)D^{-1}(z)C(z)$

Iterative Least-Squares for ARMAX

Preliminaries (SISO case)

- The parameter vector is: $\theta = [a_1\ a_2\ \dots\ a_{n_a}\ b_1\ \dots\ b_{n_b}\ c_1\ \dots\ c_{n_c}]^T$, where the orders n_a, n_b, n_c are defined by the user, as for the time-delay ℓ
- The predictor is of the form: $\hat{y}_k(\theta) = \frac{B(z)}{C(z)} u_k + \left[1 - \frac{A(z)}{C(z)}\right] y_k$
- which can be rewritten as: $\hat{y}_k(\theta) = B(z)u_k + [1 - A(z)]y_k + [C(z) - 1][y_k - \hat{y}_k(\theta)]$
- Being the predictor error: $\epsilon_k = y_k - \hat{y}_k$
- the regressor vector is: $\varphi_k = [-y_{k-1}\ \dots\ -y_{k-n_a}\ u_{k-1-\ell}\ \dots\ u_{k-n_b-\ell}\ \epsilon_{k-1}\ \dots\ \epsilon_{k-n_c}]^T$
- Hence, the predictor becomes: $\hat{y}_k = \varphi_k^T(\theta)\theta$
- which is not a linear regression: the terms ϵ_k can be computed only once θ is known, i.e., φ_k depends on θ. Note: we call this *pseudo-linear regression*
- An **iterative procedure** is built to get the "best" parameters
- Easy extension to MIMO ARMAX by using a MISO approach

Iterative Least-Squares for ARMAX

The procedure (1/2)

- The whole output vector y is: $[y_M \; y_{M+1} \; \dots \; y_N]$

  with $M = \max(n_a, n_b + \ell, n_c)$
- Regressor matrix is obtained by stacking terms for $k = M, \dots, N$:
$$\phi = [\varphi_M \; \varphi_{M+1} \; \dots \; \varphi_N]^T$$
- To compute parameter vector θ, ϵ sequence must be already known
- Start with an **ARX** identification and compute the first prediction error: $\epsilon = y - \phi\theta$
- Use ϵ sequence to update matrix ϕ and get a new θ by using standard **LLS**
- Go on updating the error sequence and matrix ϕ, so that, **Iterative LLS** is built
- At each step, a **norm** is computed; e.g.:
$$V_N(\theta) = \frac{1}{2(N - M + 1)} \sum_k \epsilon_k^2$$

Iterative Least-Squares for ARMAX

The procedure (2/2)

- If $V_N(\theta_{new}) < V_N(\theta_{old})$, then θ_{new} is taken to update matrix ϕ
- Otherwise a **re-evaluation** of θ is performed (*line search* method):

$$\theta^* = \lambda_s \theta_{new} + (1 - \lambda_s)\theta_{old}$$

  where $\lambda_s = \frac{1}{2^s}$, being $s = 1, 2, 3, \cdots$ the s-th step of re-evaluation
- At each step of re-evaluation:
  - norm V_N is calculated
  - if $V_N(\theta^*) < V_N(\theta_{old})$, then θ^* is taken as the next parameter vector
  - otherwise s is updated and a new re-evaluation is performed
  - when $\frac{1}{2^s}$ becomes less than $eps^{(*)}$ (the smallest representable positive number such that $1.0 + eps \neq 1.0$), procedure is stopped and θ_{old} is taken

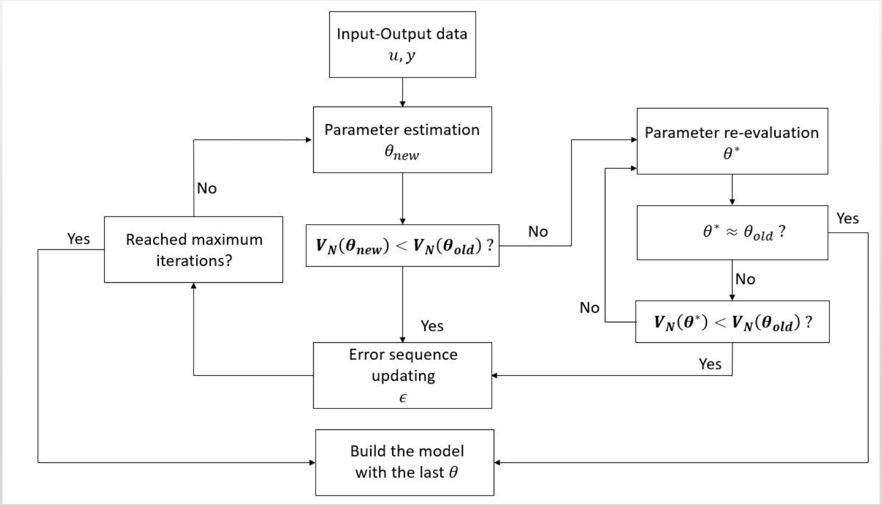
Finally, the procedure is stopped when:

1. the method finds a minimum of $V_N(\theta)$;
2. the maximum number of iterations, user defined, is reached

$^{(*)}$: $eps = 10^{-7}$ in Python 2.7, using 32 bit NumPy

Iterative Least-Squares for ARMAX

The scheme



Input-Output data u, y
Parameter estimation θ_{new}
Parameter re-evaluation θ^*
No
Yes
Reached maximum iterations?
$V_N(\theta_{new}) < V_N(\theta_{old})$?
No
$\theta^* \approx \theta_{old}$?
Yes
No
No
$V_N(\theta^*) < V_N(\theta_{old})$?
Yes
Error sequence updating ϵ
Yes
Build the model with the last θ

NL Methods

Other Classical NL methods

- **Recursive Least-Squares**
  - Classical Linear Least-Squares can be **recursive**
  - a time-variant estimator allows a uniform variation of the model parameters along the identification horizon
  - A Gain Estimator, a Covariance matrix and a **Forgetting Factor** are required
  - Details are here omitted, but an example MATLAB code for ARMAX is provided
- **Nonlinear Optimization**
  - Advanced Input-Output model may benefit from NLP
  - NL models require NL Programming
  - Also mixed methods are possible: e.g. **grid search** + LLS/PEM
  - See a later example

Nonlinear Models - EARMAX (Karra and Karim, 2009)

Extended AutoRegressive Moving Average model

- An **extended** ARMAX structure:

$$y_k = \frac{B(z)}{A(z)} u_k + \frac{C(z)}{A(z)} e_k + \frac{1}{A(z)} \eta_k$$

- not only the noise term e_k (stochastic disturbance with zero mean)
- but also a deterministic input disturbance η_k which is a **time variant** bias term representing any external non-stationary disturbances
- The model to identify is: $\hat{y}_k = \varphi_k^T(\theta)\theta + \hat{\eta}_k$
- $\{\hat{\eta}\}$ is therefore intended as a parameter which **varies slowly** over time
- Identification method must be **recursive**
- Necessary condition: build an estimator that allows a non-uniform variation of the model parameters, separating LTI part from time variant disturbance
- Parameter update with **different forgetting factors** between the two components
- An example MATLAB code is provided

Optimization Problem to Identify NL Models

Very useful implementation tools

- **Python**
  Amply validated, fast, easy-to-use, open-source, customizable
- **CasADi**
  Open-source symbolic calculation through algorithmic differentiation, numeric optimization oriented
- **IPOPT**
  Standard in the class of open-source nonlinear programming (NLP) solvers

Nonlinear Block Models

In Discrete-Time

2 Blocks Models

Hammerstein

$$\begin{tikzcd} {} \arrow[r, "u_k"] & \boxed{f(u)} \arrow[r, "x_k"] & \boxed{G(z)} \arrow[r, "y_k"] & {} \end{tikzcd}$$

static nonlinear block followed by dynamic linear block (TF)

Wiener

$$\begin{tikzcd} {} \arrow[r, "u_k"] & \boxed{G(z)} \arrow[r, "x_k"] & \boxed{h(x)} \arrow[r, "y_k"] & {} \end{tikzcd}$$

linear block (TF) followed by **static nonlinear** block

3 Blocks Models

Hammerstein - Wiener

$$\begin{tikzcd} {} \arrow[r, "u_k"] & \boxed{f(u)} \arrow[r, "x_k"] & \boxed{G(z)} \arrow[r, "w_k"] & \boxed{h(w)} \arrow[r, "y_k"] & {} \end{tikzcd}$$

Wiener - Hammerstein

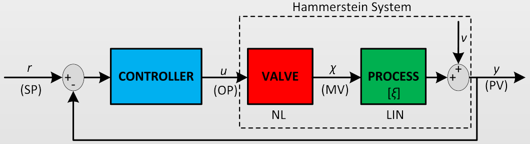
$$\begin{tikzcd} {} \arrow[r, "u_k"] & \boxed{G_1(z)} \arrow[r, "x_k"] & \boxed{f(x)} \arrow[r, "w_k"] & \boxed{G_2(z)} \arrow[r, "y_k"] & {} \end{tikzcd}$$

An example of NL block model: control loop with valve stiction

Extended Hammerstein SISO system

Control valve followed by the process dynamics:

- χ: valve stiction output, that is, process input
- y: process output
- u: output of a generic controller (PID or MPC)
- v: white Gaussian output noise



Whole plant dynamics

- **nonlinear** dynamics for the sticky valve, $\varphi(\cdot)$, here also **NON static**
- **linear** block for the process, SS form (**A**, **B**, **C**)

$$z_{k+1} = \begin{bmatrix} \chi_k \\ \xi_{k+1} \end{bmatrix} = \begin{bmatrix} \varphi(\chi_{k-1}, u_k) \\ \mathbf{A}\xi_k + \mathbf{B}\varphi(\chi_{k-1}, u_k) \end{bmatrix}$$

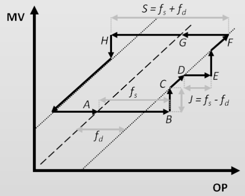
$$y_k = \mathbf{C}\xi_k + v_k$$

χ: 1st component of the state vector

Stiction Modeling

Generalities

- Stiction description (Garcia, 2008):
  - detailed physical models
  - empirical (data-driven) models
- **Data-driven** models are useful: **few parameters** and relatively **simple algebra**
- Most established models use 2 parameters:
  - Choudhury et al. (2005), Kano et al. (2004): stickband + deadband (S) and stick-slip jump (J)
  - **He et al. (2007)**: dynamic (f_D) and static (f_S) friction



A proven model - (He et al., 2007)

- **Reproduce** valve response obtained with **physical stiction models** without involving computationally intensive numerical integration
- **Fast response** from the valve is assumed, i.e. transient dynamics ignored
- Only the stationary-state values of stem position are considered

Discontinuous Valve Model

Data-driven stiction model (He et al., 2007) - Standard formulation

The sticky valve has a **nonlinear** dynamics $\chi_k = \varphi(\chi_{k-1}, u_k)$:

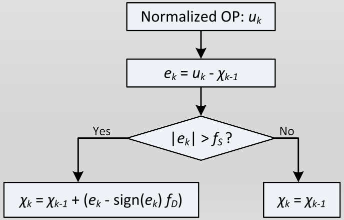
$$\chi_k = \begin{cases} \chi_{k-1} + [e_k - \text{sign}(e_k) f_D] & \text{if } |e_k| > f_S \\ \chi_{k-1} & \text{if } |e_k| \leq f_S \end{cases}$$

- f_S, f_D: static and dynamic friction parameters, $f_S \geq f_D$
- $e_k = u_k - \chi_{k-1} \sim$ valve position error

Rewritten as:

$$\chi_k = \begin{cases} u_k - f_D & \text{if} \quad u_k - \chi_{k-1} > f_S \\ u_k + f_D & \text{if} \quad u_k - \chi_{k-1} < -f_S \\ \chi_{k-1} & \text{if} \quad |u_k - \chi_{k-1}| \leq f_S \end{cases}$$

$\varphi(\cdot)$: $\sim$ a switching "**three-mode**" **discontinuous** model



NL Valve Model

Smoothing function $\varphi_S(\cdot) \simeq \varphi(\cdot)$ - (Bacci di Capaci et al., 2017)

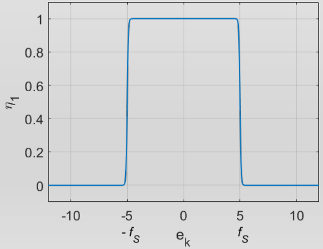
$$\chi_k = \eta_1(e_k)\chi_{k-1} + (1 - \eta_1(e_k))u_k + \eta_2(e_k)f_D$$

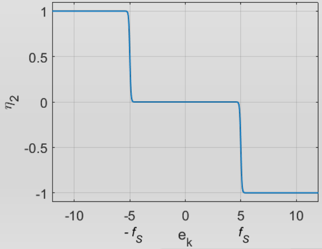
where

$$\eta_1(e_k) = \frac{1}{2}\tanh(\tau(e_k + f_S)) + \frac{1}{2}\tanh(\tau(-e_k + f_S))$$

$$\eta_2(e_k) = \frac{1}{2}\tanh(-\tau(e_k + f_S)) + \frac{1}{2}\tanh(\tau(-e_k + f_S))$$

- being $e_k = u_k - \chi_{k-1} \sim$ valve position error
- Tuning parameter τ: $\simeq 10^4 \Rightarrow \varphi_S(\cdot) \simeq \varphi(\cdot)$ higher value, sharper functions





Identification Problem (1/4)

Defining the Hammerstein model

Linear Process: ARX structure in discrete-time form

$$A(z)y_k = B(z)\chi_{k-\ell} + e_k$$

- $A(z), B(z)$: polynomials in backward shift operator z^{-1} (i.e. $\chi_k = z^{-1}\chi_{k+1}$)

$$A(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_{n_a} z^{-n_a}$$
$$B(z) = b_1 z^{-1-\ell} + b_2 z^{-2-\ell} + \ldots + b_{n_b} z^{-n_b-\ell}$$

- ℓ: input time-delay
- (n_a, n_b): orders on the auto-regressive and exogenous terms

Non Linear Valve: the aforesaid smoothed stiction model $\varphi_S(\cdot)$

Optimization variables X

- static and dynamic friction parameters $(\hat{f}_S, \hat{f}_D)$
- $n_a + n_b$ coefficients of ARX process model

$$X = [\hat{f}_S, \hat{f}_D, \hat{\theta}] \quad \text{with} \quad \hat{\theta} = [a_1, \ldots, a_{n_a}, b_1, \ldots, b_{n_b}]$$

GRUPPO DI INGEGNERIA CHIMICA DELL'UNIVERSITÀ GRICU GET INNOVATION

Identification Problem (2/4)

One-stage nonlinear optimization problem

$$X^* = \arg\min_{f_S, f_D, \theta} \text{SE}(y, \hat{y})$$

subject to:

$$f_{\min} \leq f_S, f_D \leq f_{\max}$$
$$f_S \geq f_D$$
$$\sigma^2(\hat{\chi}) \geq \sigma^2_{\min}$$

where

- $\hat{y} = \Phi\theta$: identified process output
- $\Phi \in \mathbb{R}^{N \times n_a + n_b}$: regressor matrix of the measurements (u, y)
- N: number of data points
- $\hat{\chi}$: identified valve position

Remarks

- Square Error (SE) objective function: $\text{SE}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^T(y - \hat{y})$
- Constraint on the variance $\sigma^2(\hat{\chi})$: **valve is forced to oscillate** due to the presence of stiction, e.g. a safe choice $\sigma^2_{\min} = 0.1\sigma^2(u)$
- Time-delay ℓ and model orders are assumed as parameters

Identification Problem (3/4)

Initialization

Suitable initial point X_0:

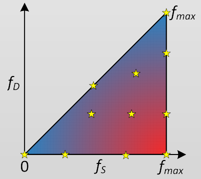
- $\hat{\theta}_0$: ARX model identification, **valve stiction free**

$$\hat{\theta}_0 = (\phi_0)^+ y = [\phi_0^T \phi_0]^{-1} \phi_0^T y$$

ϕ_0: initial regressor matrix computed by stacking linear regressor vectors $\varphi_{0,k}$ $\forall$ time sample k

$$\varphi_{0,k} = [-y_{k-1}, \dots, -y_{k-n_a}, u_{k-1-t_d}, \dots, u_{k-n_b-\ell}]$$

- $\hat{f}_{S,0}, \hat{f}_{D,0}$: define a **triangular shape domain**
  - $f_{\max} \geq f_S \geq f_D \geq f_{\min} = 0$
  - $f_{\max} = \Delta u$, oscillation span of controller output
  - $f_S + f_D = \Delta u$, square-shaped signal



Multiple starts (M) to avoid to be stuck in a local minimum
Start from the domain boundaries, step $\Delta f_S = \Delta f_D = 0.5$
Choose the **best** solution in terms of **objective function and infeasibility**

Subspace Identification Methods (SIM): introduction

Motivations

- Multivariable input-output systems identification requires **prior knowledge** or **trial-and-error** to determine the system orders (note: **Information Criteria** can be used; see complements...)
- Input-output systems identification is always **MISO**, whereas in some cases it would desirable to **directly** identify **MIMO** models
- Identification of **advanced multivariable** models (e.g., ARMAX, OE, etc.) may require solution of large **nonconvex nonlinear programming** problems

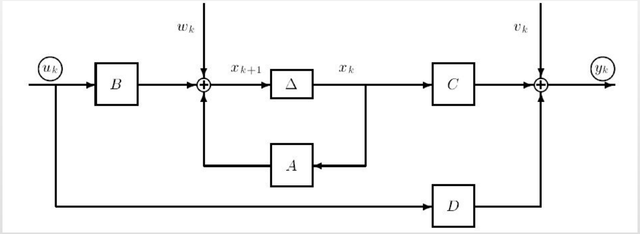
Features

- **Direct** identification of a DLTI state-space model
- Applicable to both **MIMO** and MISO approaches
- **Compact** multivariable state-space representation
- Very **little prior knowledge** required (an **upper bound** to the order)
- Based on **reliable linear algebra** decompositions

State-space systems: basic definitions

DLTI system: **process** form

$$x_{k+1} = Ax_k + Bu_k + w_k$$
$$y_k = Cx_k + Du_k + v_k$$



- dimensions: $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $y \in \mathbb{R}^p$
- note: n often **unknown**; m, p known; usually $D = 0$, for physical systems
- **noise** term: split into *measurement* v_k and *process* w_k noise
- Assumption: $\{w\}$, $\{v\}$ are sequences of independent random variables with zero mean and covariances:

$$\mathcal{E}(w_k w_k^T) = Q \quad \mathcal{E}(w_k v_k^T) = S \quad \mathcal{E}(v_k v_k^T) = R$$

State-space systems: basic definitions

DLTI system: **innovation** and **predictor** forms

Innovation form:

$$x_{k+1} = Ax_k + Bu_k + Ke_k$$
$$y_k = Cx_k + e_k$$

Predictor form ($A_K = A - KC$):

$$x_{k+1} = A_K x_k + Bu_k + Ky_k$$
$$y_k = Cx_k + e_k$$

where K is the steady-state **Kalman filter gain**, obtained from Algebraic Riccati Equation

Main assumptions

- (A, B) controllable; (A, C) observable; $A_K = A - KC$ **strictly Schur** ($|\lambda_i(A_k)| < 1, \forall i$)
- The innovation $\{e_k\}$ is a **stationary**, zero mean, white noise process:
  $$\mathcal{E}(e_j e_j^\top) = R_e, \qquad \mathcal{E}(e_i e_j^\top) = 0 \quad \text{for } i \neq j$$
- Input $\{u_k\}$ and output $\{y_k\}$ data sequences are available for $k = 0, \ldots, N$

Indirect routes to get this LTI model

- They can be obtained **via realization** of input-output models (TF $\Rightarrow$ SS)
- Often the obtained **order** is quite **high**, with **no** perceivable **advantages**

SIM: classification (Qin, 2006)

Traditional - N4SID, MOESP, CVA

- later grouped into a unifying theorem (Overschee and Moor, 1995)
- estimate system matrices $(A, B, C, (D))$ from the **process** form
- seen as a **singular value decomposition** (SVD) of a suitable weighted matrix
- basic assumption: u_k and w_k, v_k are uncorrelated $\Rightarrow$ may **fail on closed-loop** data
- **do not enforce** any block-triangular matrix and then casual models

Parsimoniuos - PARSIM-S, PARSIM-P, PARSIM-K, PARSIM-E

- estimate system matrices $(A, B, C, (D), K)$ from the **innovation/predictor** form
- remove non-causal terms in the linear projections by enforcing **causal models**; in particular, enforce the lower triangular structure of matrix H_r^u
  - sequential **PARSIM-S** (Qin et al., 2005) and parallel **PARSIM-P** (Qin and Ljung, 2003)
    - oldest and most common algorithms
    - basic assumption: u_k and e_k are uncorrelated $\Rightarrow$ **still may fail on CL** data
  - **PARSIM-K** (Pannocchia and Calosi, 2010) and **PARSIM-E** algorithms (Hou et al., 2015):
    - specifically consistent with CL data

Basic SIM: algorithm derivation

An r-step prediction model

- For each k, define an r-step prediction model (here, innovation form):

$$y_k = Cx_k + e_k; \quad y_{k+1} = Cx_{k+1} + e_{k+1} = CAx_k + CBu_k + CKe_k + e_{k+1}; \quad y_{k+2} = \dots$$

$$\underbrace{\begin{bmatrix} y_k \\ y_{k+1} \\ y_{k+2} \\ \vdots \\ y_{k+r-1} \end{bmatrix}}_{\bar{y}_k} = \underbrace{\begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{r-1} \end{bmatrix}}_{\Gamma_r} x_k + \underbrace{\begin{bmatrix} 0 & \cdots & \cdots & 0 \\ CB & 0 & \cdots & 0 \\ CAB & CB & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{r-2}B & CA^{r-3}B & \cdots & 0 \end{bmatrix}}_{H_r^u} \underbrace{\begin{bmatrix} u_k \\ u_{k+1} \\ u_{k+2} \\ \vdots \\ u_{k+r-1} \end{bmatrix}}_{\bar{u}_k}$$

$$+ \underbrace{\begin{bmatrix} I & \cdots & \cdots & 0 \\ CK & I & \cdots & 0 \\ CAK & CK & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{r-2}K & CA^{r-3}K & \cdots & I \end{bmatrix}}_{H_r^e} \underbrace{\begin{bmatrix} e_k \\ e_{k+1} \\ e_{k+2} \\ \vdots \\ e_{k+r-1} \end{bmatrix}}_{\bar{e}_k}$$

- **Repeat** for $k \in \{r, \dots, M = N - r + 1\}$ and **concatenate horizontally**:

$$\underbrace{\begin{bmatrix} \bar{y}_r & \cdots & \bar{y}_M \end{bmatrix}}_{Y} = \Gamma_r \underbrace{\begin{bmatrix} x_r & \cdots & x_M \end{bmatrix}}_{X} + H_r^u \underbrace{\begin{bmatrix} \bar{u}_r & \cdots & \bar{u}_M \end{bmatrix}}_{U} + H_r^e \underbrace{\begin{bmatrix} \bar{e}_r & \cdots & \bar{e}_M \end{bmatrix}}_{E}$$

Basic SIM: algorithm - extended observability matrix

The basic relation

- The previous relation is written **compactly** as:

$$Y = \Gamma_r x + H_r^u U + H_r^e E$$

- Γ_r is called **extended observability matrix**
- H_r^u and H_r^e are **lower triangular block Toeplitz** matrices

Computing the extended observability matrix

- Note: the **different methods** basically **separate here**.
  E.g., in PARSIM-K, express:

$$x = \begin{bmatrix} x_r & \cdots & x_M \end{bmatrix} = A_K^r \begin{bmatrix} x_0 & \cdots & x_{M-r} \end{bmatrix} + \begin{bmatrix} A_K^{r-1}B & \cdots B \end{bmatrix} U_p + \begin{bmatrix} A_K^{r-1}K & \cdots K \end{bmatrix} Y_p$$

$$\approxeq \underbrace{\begin{bmatrix} A_K^{r-1}B & \cdots B & A_K^{r-1}K & \cdots K \end{bmatrix}}_{L_z} \underbrace{\begin{bmatrix} U_p \\ Y_p \end{bmatrix}}_{Z_p}$$

- **Solve** the basic relation: $Y = \Gamma_r L_z Z_p + H_r^u U + H_r^e E$ to obtain $(\Gamma_r L_z)$ from **LS**
- Compute Γ_r, having n columns with $n < r$, from a **truncated SVD** of $(\Gamma_r L_z)$

Basic SIM: algorithm - compute (A, C)

Computing (A, C): another LS problem

- After Γ_r is obtained, we can easily compute C (MATLAB notation):

$$C = \Gamma_r(1 : p, :)$$

that is, first p rows of Γ_r, being p the output number

- The matrix A is instead obtained from the **LS problem**, by using shift-invariance property of Γ_r:

$$\Gamma_r(p + 1 : p \times r, :) = \Gamma_r(1 : p \times (r - 1), :)A$$

- Hence, the solution is:

$$A = \Gamma_r(1 : p \times (r - 1), :)^{+}\Gamma_r(p + 1 : p \times r, :)$$

where $\{\cdot\}^{+}$ is the pseudo-inverse matrix; $\Gamma_r(1 : p \times (r - 1), :)$ and $\Gamma_r(p + 1 : p \times r, :)$ are Γ_r without the last and the first p rows, respectively

Basic SIM: algorithm - compute B, (D), and x_0

Obtaining B and x_0 via LS

- From: $x_{k+1} = Ax_k + Bu_k + x_0; \quad \hat{y}_k = Cx_k$
- Being $x_{k+1} = zx_k$, obtain: $x_k = (zI - A)^{-1}(Bu_k + x_0)$
- With (A, C) known, we can write a **linear predictor**:

$$\hat{y}_k = C(zI - A)^{-1}Bu_k + C(zI - A)^{-1}x_0 = \varphi_k\theta, \qquad \text{with } \theta = \begin{bmatrix} \text{Vec}(B) \\ x_0 \end{bmatrix}$$

  where $\text{Vec}(B)$ is vectorized B matrix along the rows (easy extension when $D \neq 0$)
- repeating until N and stacking: $$Y = \phi\theta$$

  where

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \qquad \phi = \begin{bmatrix} \varphi_1 \\ \vdots \\ \varphi_N \end{bmatrix}$$

- Solve an **LS problem**: $$\theta = (\phi^\top\phi)^{-1}\phi^\top Y$$

Input Design and Data collection: objectives and practice

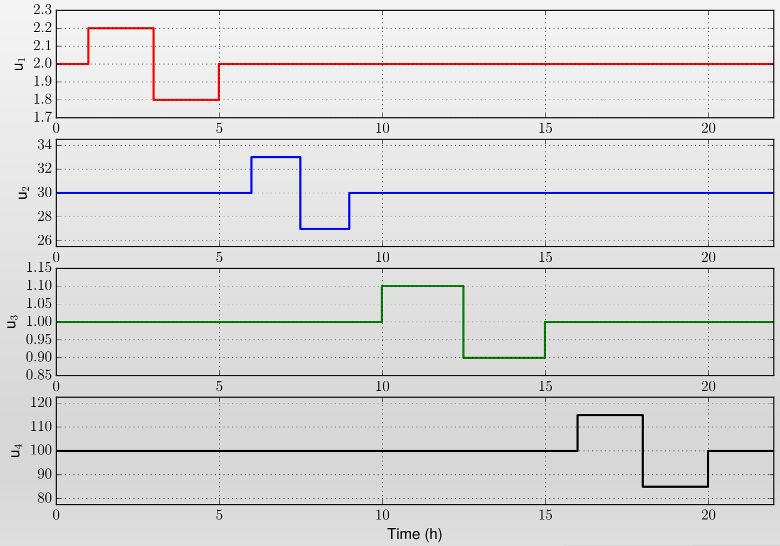
Objectives

- Identification data is (generally) collected during specific **campaigns**
- Test **duration** should be **minimized**, but data should be **informative**

Functional design

- The following **lists of variables** are compiled:
  - **MV**: manipulated variables
  - **CV**: controlled variables (measurable)
  - **DV**: disturbance variables (measurable)
- **Instrumentation** (sensors and actuators) may undergo into maintenance before testing
- **Prior knowledge** and/or preliminary tests are used to decide:
  - **Amplitude** of each MV variation (on the basis of static gains)
  - **Duration** of each MV variation (on the basis of settling times)

Traditional open-loop step tests



Limitations of step tests

The frequency content of an input signal:

- **Autocorrelation** function of a **stationary** stochastic variable $\{u(k)\}$:

$$R_u(\tau) = \mathcal{E}\left(u(k)u(k-\tau)\right)$$

- **Power spectrum** or spectral density

$$\Phi_u(\omega) = \sum_{\tau=-\infty}^{\infty} R_u(\tau)e^{-i\tau\omega}$$

Signals requirements

- Identification signals must have a sufficiently **high power spectrum** in mid and low frequency range
- A related property of signals is called **persistent excitation**
- Step signals have **limited** frequency content and do not excite the plant significantly in all frequency ranges

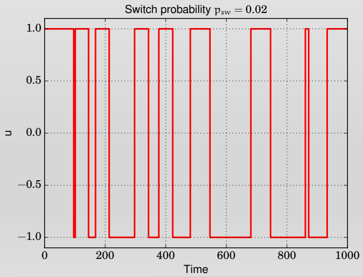
Beyond the step tests

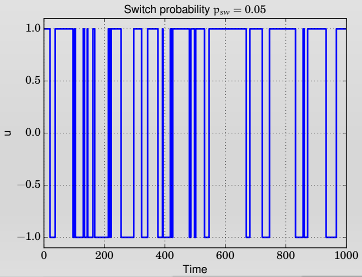
GBN and PRBS

- Generalized Binary Noise (**GBN**) signals are very effective (Zhu, 2001)
- A GBN signal has **two possible values** $\{+a, -a\}$
- Let $p_{sw} \in (0, 1)$ be the **switching probability**. The signal obeys:

$$\begin{cases} P\left[u_k = -u_{k-1}\right] = p_{sw} \\ P\left[u_k = u_{k-1}\right] = 1 - p_{sw} \end{cases}$$

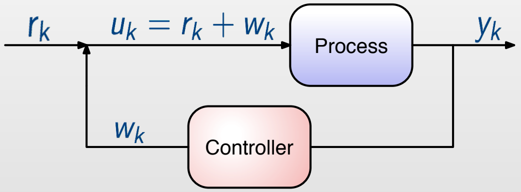
- **PRBS** are similar but **periodic**





Closed-loop tests: basic idea

Scheme



Basic relations

- **Feedback** is used:

$$w_k = -F(y_k)$$

- **Independent** "setpoints" r_k (GBN, PRBS) are added to w_k:

$$u_k = r_k + w_k = r_k - F(y_k)$$

- Setpoints are used to **improve excitations** at higher frequencies

Closed-loop vs open-loop tests

Advantageous features of open-loop signals

- There is **no need** to have a working **controller**
- Identification algorithms are **always applicable** to open-loop data
- **Input variations** (amplitude and duration) defined by the **user**
- Dynamic responses more **easily understood**
  (no correlation exist between input and process noise)

Advantageous features of closed-loop signals

- Variations of **outputs** can be **controlled**
- Variations of inputs are **simultaneous**
- Many **studies** report that "closed-loop data are better suited for **controller design**"

Multivariable data collection

Motivations

- Multivariable signals are more **informative** and excite the system in **several directions**
- The **nonlinearity** is better understood by multivariable signals

Recommended practice

- **Open-loop** data collection: use **independent** GBN **inputs**
- **Closed-loop** data collection: use **independent** GBN **setpoints**

The information criteria

Scope – find the most appropriate model orders

Basic idea: balance between:

- complexity $\Rightarrow$ higher orders and longer computational times
- simplicity $\Rightarrow$ faster and more robust model-based controllers

Three common examples– for both I-O and SIMs

- **AIC** Akaike Information Criterion (Akaike, 1974)
- **AICc** Correction of Akaike Information Criterion (Sugiura, 1978)
- **BIC** Bayesian Information Criterion (Schwarz, 1978)

Features: – find the minimum of a specific function

including a **likelihood function** and a **penalty term** on the number of model parameters

$$AIC = -2\log(\hat{\mathcal{L}}) + 2K_{IC}$$

$$AICc = AIC + \frac{2K_{IC}(K_{IC}+1)}{N_{IC}-K_{IC}-1}$$

$$BIC = -2\log(\hat{\mathcal{L}}) + K_{IC}\log(N_{IC})$$

where $\hat{\mathcal{L}}$: maximized likelihood function
(e.g., $\log(\hat{\mathcal{L}}) = -\frac{N_{IC}}{2}\log\hat{\sigma}^2 = -\frac{N_{IC}}{2}\log\frac{\sum_{k=1}^{N_{IC}}\epsilon_k^T\epsilon_k}{pN_{IC}}$)
K_{IC}: number of independent model parameters
N_{IC}: number of data points used to compute the **variance** $\hat{\sigma}^2$ of the model residuals ϵ_k

Model analysis

Mean and Variance

For each output y, we compute:

$$\bar{y} = \frac{1}{N}\sum_{i=k}^{N} y_k, \quad \sigma_y^2 = \frac{1}{N}\sum_{k=1}^{N}(\bar{y} - y_k)^2$$

to work with deviation normalized variables

Explained Variance

The **variance** that is **explained** by a model, a.k.a. **correlation coefficient**

$$EV = R^2 = 1 - \frac{\frac{1}{N}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2}{\sigma_y^2}$$

Cross-validation

Often, the Explained Variance is computed over a data **set not used** for computing the model parameters (in the identification stage)

Model validation

Maximum error estimate

- Advanced Identification methods provide an **estimate** on the maximum error of the model
- If the model is deemed **not suitable**, typical attempts are:
  - Change of model **orders** in input-output models or the **maximum order** in SIM alghoritms
  - Improve **scaling** of input and output variables

...If everything fails

- **Collect new data**, choosing different identification signals with **better frequency content**
- Identify the models from (steady-state or dynamic) **rigorous simulations** (i.e., UniSim Design, Aspen HYSYS)

Identification software packages

Commercial packages

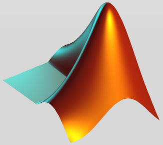
Many solutions from commercial vendors, covering the various identification methods

- **System Identification Toolbox ™** in **MATLAB** (MathWorks, 2021): the most famous and consolidated package
- Other options:
  - ISIAC software (Tona and Bader, 2006)
  - NI LabVIEW System Identification Toolkit (Instruments, 2021)

Open-source packages

Many examples, written on different programming languages, as the various **MATLAB-based** toolboxes:

- UNIT (Ninness et al., 2013)
- CONTSID (Garnier et al., 2012)
- CAPTAIN (Young and Taylor, 2012)
- ITSIE (Guzmán et al., 2012)



SIPPY: System Identification Package for PYthon

Main features

One of **most complete** open-source package for **Python** – to the best of our knowledge ...

- covering a wide range of identification methods
- possibility to identify **multivariable** systems
- focused only on **linear** models
- excluded nonlinear systems
  – see other software, e.g. NL-ARX (MathWorks, 2021) or Hammerstein-Wiener models (Ninness et al., 2013)

Models & Data

- identifies both **input-output** and **state space** models
- uses input-output data (Open and Closed Loop)
  for a general multivariable system with m inputs and p outputs:

$$\mathbf{u} = [u_0 \; u_1 \; u_2 \; \dots \; u_{N-1}], \quad \mathbf{y} = [y_0 \; y_1 \; y_2 \; \dots \; y_{N-1}]$$

where N: number of samples

SIPPY: some details on state space models

Algorithms implemented

- **modified N4SID**: improved "*combined algorithm 2*" (Overschee and Moor, 1996)
- **original aspect** of **SIPPY**: include 3 Parsimonious methods
- **truncated SVD**:
  - retaining up to the $n-$th singular value
  - alternatively, specify a threshold value TV with the maximum order allowed
  - otherwise, employ an information criteria
- **Extensive simulation** studies:
  $\Rightarrow$ superior performance in terms of the variance of the residuals
- **lower computation load** as only a single state sequence is identified

User specifications and choices

- model orders and horizons (future and past)
- direct input-output relation, i.e., matrix $D \neq 0$
- matrix A is analyzed with a **stability test**
  for *traditional* methods, impose the stability of matrix A: $\rho(A) \equiv \max_i(|\lambda_i|) < 1$

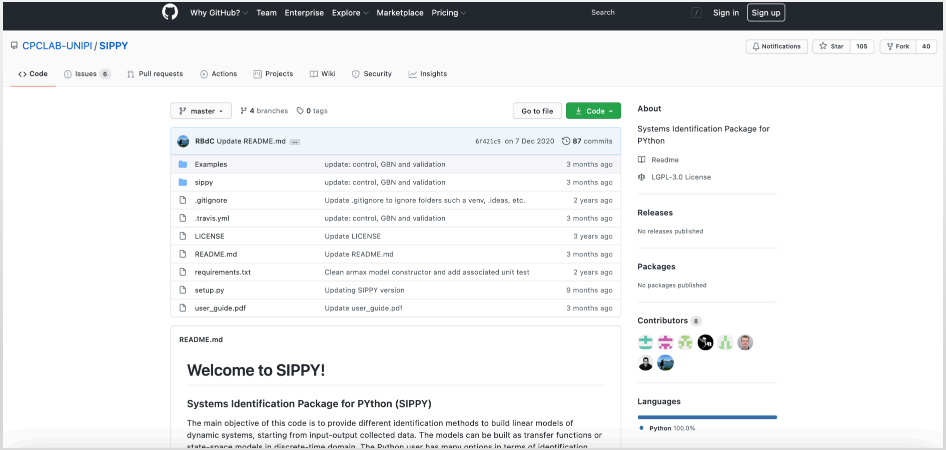
Summary about **SIPPY**

- An **open-source package for Python** with various **sys id** structures and methods
- **Input-output** models in 9 structures and 3 algorithms (*):
  - FIR, ARX $\Rightarrow$ LLS
  - ARMAX $\Rightarrow$ I-LLS
  - ARMAX, ARARX, ARARMAX, OE, BJ, GENERALIZED $\Rightarrow$ NLP (*: next release)
- **State space** models, 6 algorithms:
  - 3 standard approaches (N4SID, MOESP, and CVA)
  - 3 parsimonious (PARSIM-S, PARSIM-P, PARSIM-K)
- 3 **information criteria**: to help the choice of suitable model orders when not known a-priori
- Several **file examples** included: pure numerical and simulation
- The state-of-art **MATLAB toolbox**: taken as **reference to test** performance and accuracy
- **Comparable results** vs. MATLAB: but higher efficiency in terms of computational times

- Underlying **code**:
  - default settings: intended to be simple for beginners
  - user options: to set algorithms parameters
  - **repository**: freely available at **GITHUB**

The GitHub Repository

Open-source access: *https://github.com/CPCLAB-UNIPI/SIPPY*



Why GitHub? Team Enterprise Explore Marketplace Pricing Search Sign in Sign up

CPCLAB-UNIPI / SIPPY

Notifications Star 105 Fork 40

Code Issues 6 Pull requests Actions Projects Wiki Security Insights

master 4 branches 0 tags Go to file Code

RBdC Update README.md ... 6f421c9 on 7 Dec 2020 87 commits

Examples	update: control, GBN and validation	3 months ago
sippy	update: control, GBN and validation	3 months ago
.gitignore	Update .gitignore to ignore folders such a venv, .ideas, etc.	2 years ago
.travis.yml	update: control, GBN and validation	3 months ago
LICENSE	Update LICENSE	3 years ago
README.md	Update README.md	3 months ago
requirements.txt	Clean armax model constructor and add associated unit test	2 years ago
setup.py	Updating SIPPY version	9 months ago
user_guide.pdf	Update user_guide.pdf	3 months ago

README.md

Welcome to SIPPY!

Systems Identification Package for PYthon (SIPPY)

The main objective of this code is to provide different identification methods to build linear models of dynamic systems, starting from input-output collected data. The models can be built as transfer functions or state-space models in discrete-time domain. The Python user has many options in terms of identification

About

Systems Identification Package for PYthon

Readme

LGPL-3.0 License

Releases

No releases published

Packages

No packages published

Contributors 8

Languages

Python 100.0%

Simulation Examples – solved #1

The plant: a MIMO ARMAX system ($m = 4, p = 3$)

	Output 1	Output 2	Output 3
Input 1	$g_{11} = \frac{4z^3+3.3z^2}{z^5-0.3z^4-0.25z^3-0.021z^2}$	$g_{21} = \frac{-85z^2-57.5z-27.7}{z^4-0.4z^3}$	$g_{31} = \frac{0.2z^3}{z^4-0.1z^3-0.3z^2}$
Input 2	$g_{12} = \frac{10z^2}{z^5-0.3z^4-0.25z^3-0.021z^2}$	$g_{22} = \frac{71z+12.3}{z^4-0.4z^3}$	$g_{32} = \frac{0.821z^2+0.432z}{z^4-0.1z^3-0.3z^2}$
Input 3	$g_{13} = \frac{7z^2+5.5z+2.2}{z^5-0.3z^4-0.25z^3-0.021z^2}$	$g_{23} = \frac{-0.1z^3}{z^4-0.4z^3}$	$g_{33} = \frac{0.1z^3}{z^4-0.1z^3-0.3z^2}$
Input 4	$g_{14} = \frac{-0.9z^3-0.11z^2}{z^5-0.3z^4-0.25z^3-0.021z^2}$	$g_{24} = \frac{0.994z^3}{z^4-0.4z^3}$	$g_{34} = \frac{0.891z+0.223}{z^4-0.1z^3-0.3z^2}$
Error model	$h_1 = \frac{z^5+0.85z^4+0.32z^3}{z^5-0.3z^4-0.25z^3-0.021z^2}$	$h_2 = \frac{z^4}{z^4-0.4z^3}$	$h_3 = \frac{z^4+0.7z^3+0.485z^2+0.22z}{z^4-0.1z^3-0.3z^2}$

orders: p for output and error, $p \times m$ for inputs and time-delays:

$$n_a = [3\ 1\ 2], \quad n_b = \begin{bmatrix} 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & 1 \\ 1 & 2 & 1 & 2 \end{bmatrix}, \quad n_c = [2\ 0\ 3], \quad \Theta = \begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix}$$

Parameters & Settings

- **Input Data**: 4 independent GBN with a switch probability equal to 3%
- **Models tested**: ARX and ARMAX, with **known** system orders and time-delays
- **White noise** $e^{(i)}$: 5 levels with different variances σ^2
- **Monte Carlo simulations**: $\forall$ noise level, a set of simulations with $N = 400$ data
  500 simulations for the **identification** stage and 500 for the **validation**

Simulation Examples – solved #1

Identification performance index

Performance evaluated by using the **explained variance** (**EV**):

$$EV^{(i)} = 1 - \frac{\sum_k (\hat{\epsilon}_k^{(i)})^2}{\sum_k (y_k^{(i)} - \overline{y}^{(i)})^2}$$

where $\overline{y}^{(i)}$ is the mean value of the i-th output

When a model returns $EV^{(i)} < 0$ for an output, the EV is considered equal to zero

Results & Discussion

Average explained variance $\overline{EV}$,
for identification **ID** and validation **VA** data sets

	σ_1^2	σ_2^2	σ_3^2	σ_4^2	σ_5^2
ARMAX (ID)	0.9423	0.9585	0.9513	0.9386	0.9293
ARX (ID)	0.9955	0.9918	0.9858	0.9769	0.9672
ARMAX (VA)	0.9417	0.9565	0.9474	0.9341	0.9230
ARX (VA)	0.9940	0.9890	0.9819	0.9712	0.9622

- **ARX**: shows superior performance
- **ARMAX**:

$EV < 0$ for several simulations,
which make values of $\overline{EV} \Downarrow$

whether the identification is successful,
single EV is usually greater than the one obtained by ARX

note: **this is NOT a general result !!!**

Simulation Examples – solved #2

Plant & Models

- **MIMO State-Space** system ($m = 2, p = 2, n = 3$)
- *White noise*: 4 different levels, with the same variance σ^2 for both outputs
- **Model tested**: N4SID and PARSIM-K ($f_h = p_h = 20$)

Parameters & Settings

- **Closed-loop CL mode**: data collected ($\mathbf{u}, \mathbf{y}$), with known model order
  - *reference vector* r: 2 independent GBNs with a switch probability of 2%
  - *CL input*: proportional control law $u_k = K_c(r_k - y_k)$
- **Monte Carlo simulations**: $\forall$ noise level, a set of simulations with $L = 500$
  - 500 for the *identification* stage and 500 for the *validation* stage

Identification performance index

Performance evaluated with the explained variance (EV): $EV^{(i)} = 1 - \frac{\sum_k (\hat{\epsilon}_k^{(i)})^2}{\sum_k (y_k^{(i)} - \bar{y}^{(i)})^2}$

where $\bar{y}^{(i)}$: mean value of the i-th output

when a model returns $EV^{(i)} < 0$, the EV is considered equal to zero

Simulation Example – solved #2

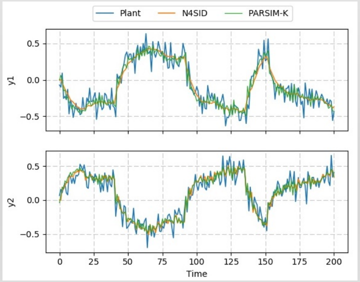
Results & Discussion

Overall: mean variance $\overline{EV}$
for identification (ID) and validation (VA) data sets

	σ_1^2	σ_2^2	σ_3^2	σ_4^2
N4SID (ID)	0.9988	0.9859	0.8793	0.4256
PARSIM-K (ID)	0.9989	0.9886	0.8825	0.4089
N4SID (VA)	0.9988	0.9871	0.8766	0.4181
PARSIM-K (VA)	0.9988	0.9881	0.8758	0.3972

PARSIM-K shows:

- better performance in the identification stage
- a slightly lower robustness to noise in the validation stage



A generic simulation example

Validation stage with noise variance σ_3^2

- both **PARSIM-K and N4SID**: excellent performance
- proper fitting of the original output
- despite being a hard case of CL data

Simulation Examples – proposed

Choose among the simulation examples included in SIPPY

- pure numerical cases: ARX SISO, ARMAX SISO, SS
- Wood-berry column
- Continuos Stirred Tank
- Continuos Stirred Tank Reactor
- Triple effect evaporators
- .. but also build your own examples and/or use your own data

Plug and Play

- test different models structures and algorithms
- test parameters sensitivity: model orders, noise levels, input design ...

Conclusions

Some observations

- Systems identification is of **primary importance** for the success (or the failure) of **advanced control**
- Since its origin in the process industries (identification of FIR models via least-squares), many advances were made:
  - The importance of data collection has been recognized and widely accepted
  - Robust and efficient identification methods have been developed (e.g., LS, ILS, RLS, QP, NLP)
  - Especially for MPC design, subspace identification methods has been very popular:
    - traditional: N4SID, MOESP, CVA;
    - parsimonious: PARSIM-S, -P, -K, -E.
- Examples with software **SIPPY**: Systems Identification Package for PYthon
  `https://github.com/CPCLAB-UNIPI/SIPPY`

References I

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723.

Bacci di Capaci, R., Vaccari, M., and Pannocchia, G. (2017). A valve stiction tolerant formulation of MPC for industrial processes. In *Proceedings of the 20th IFAC World Congress*, pages 9374–9379, Toulouse, France, 9–14 July.

Choudhury, M. A. A. S., Shah, S. L., and Thornhill, N. F. (2005). Modelling valve stiction. *Contr. Eng. Pract.*, 13:641–658.

Garcia, C. (2008). Comparison of friction models applied to a control valve. *Contr. Eng. Pract.*, 16:1231–1243.

Garnier, H., Gilson, M., Laurain, V., and Ni, B. (2012). Developments for the CONTSID toolbox. *IFAC Proceedings Volumes*, 45(16):1553 – 1558.

Guzmán, J., Rivera, D., Dormido, S., and Berenguel, M. (2012). An interactive software tool for system identification. *Advances in Engineering Software*, 45(1):115 – 123.

He, Q. P., Wang, J., Pottmann, M., and Qin, S. J. (2007). A curve fitting method for detecting valve stiction in oscillating control loops. *Ind. Eng. Chem. Res.*, 46:4549–4560.

Hou, J., Chen, F., and Liu, T. (2015). Recursive closed-loop PARSIM-E subspace identification. *IFAC-PapersOnLine*, 48(28):880 – 885.

Instruments, N. (2021). NI LabVIEW System Identification Toolkit, `https://zone.ni.com/reference/en-XX/help/372458D-01/lvsysidconcepts/sysid_help_main/`.

Kano, M., Hiroshi, M., Kugemoto, H., and Shimizu, K. (2004). Practical model and detection algorithm for valve stiction. In *Proceedings of 7th IFAC DYCOPS*, Boston, USA. Paper ID n. 54.

References II

Karra, S. and Karim, M. N. (2009). Alternative model structure with simplistic noise model to identify linear time invariant systems subjected to non-stationary disturbances. *Journal of Process Control*, 19:964–977.

Ljung, L. (1999). *System Identification: Theory for the User*. Prentice Hall Inc., Upper Saddle River, New Jersey, Second edition.

MathWorks (2021). MATLAB System Identification Toolbox, `https://it.mathworks.com/products/sysid.html`.

Ninness, B., Wills, A., and Mills, A. (2013). UNIT: A freely available system identification toolbox. *Control Engineering Practice*, 21(5):631–644.

Overschee, P. V. and Moor, B. D. (1995). A unifying theorem for three subspace system identification algorithms. *Automatica*, 31(12):1853–1864.

Overschee, P. V. and Moor, B. D. (1996). *Subspace identification for linear systems*. Kluwer Academic Publishers.

Pannocchia, G. and Calosi, M. (2010). A predictor form PARSIMonious algorithm for closed-loop subspace identification. *Journal of Process Control*, 20(4):517–524.

Qin, S. (2006). An overview of subspace identification. *Computers & Chemical Engineering*, 30(10):1502–1513.

Qin, S. J., Lin, W., and Ljung, L. (2005). A novel subspace identification approach with enforced causal models. *Automatica*, 41(12):2043–2053.

Qin, S. J. and Ljung, L. (2003). Parallel QR implementation of subspace identification with parsimonious models. *IFAC Proceedings Volumes*, 36(16):1591–1596.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464.

References III

Sugiura, N. (1978). Further analysts of the data by Akaike' s information criterion and the finite corrections. *Communications in Statistics - Theory and Methods*, 7(1):13–26.

Tona, P. and Bader, J. (2006). Efficient system identification for model predictive control with the ISIAC software. In *Informatics in Control, Automation and Robotics I*, pages 225–232, Dordrecht. Springer Netherlands.

Young, P. and Taylor, C. (2012). Recent developments in the CAPTAIN toolbox for Matlab. *IFAC Proceedings Volumes*, 45(16):1838 – 1843.

Zhu, Y. (2001). *Multivariable System Identification For Process Control*. Elsevier Science, First edition.