UNIVERSITY OF PISA, DEPARTMENT OF CIVIL AND INDUSTRIAL ENGINEERING,
CHEMICAL PROCESS CONTROL LABORATORY (CPCLAB)

Systems Identification Package for PYthon (SIPPY): User Guide

Riccardo Bacci di Capaci, Marco Vaccari,
Giuseppe Armenise, Gabriele Pannocchia

April 23, 2021

CONTENTS

1 INTRODUCTION

This document is intended to give a guideline to the user of SIPPY - Systems Identification Package for PYthon (SIPPY) - who wants to perform system identification from his own MIMO (or SISO) collected data.
The program is written both for Python 2.7 and 3.7 (supported on Windows \ Linux \ Mac).
The following Python packages must be installed: `NumPy`, `SciPy`, `Control` (version ≥ 0.8.2), `Math`, `Slycot`, `CasADi`.
In order to make the code compatible with both Python 2.7 and 3.7, the package `future` has to be installed (for more details see: http://python-future.org/index.html).
This package can be installed via `pip install`, e.g. by using the user Terminal, or by searching for `future` among non installed packages in Anaconda Navigator (more information can be found at https://anaconda.org/anaconda/anaconda-navigator).

1.1 INSTALLATION E PACKAGE CONTENT

In order to make the installation easier, the user can simply use the quick command, on his/her own Terminal or on the `Anaconda Prompt`, by editing:

```
python setup.py install
```

in order to collect the required packages together.
The program SIPPY is distributed as a packed file SIPPY.zip that includes the following items:

- setup.py: which runs the installation for the whole package.
- sippy/__init__.py: the main file containing the function that has to be recalled to perform the system identification;
- sippy/functionset.py: the file containing most of the functions used by the identification functions and other useful functions (See Section 7).
- sippy/functionset_OPT.py: the file containing the nonlinear optimization problem used by some of the identification functions (See Section 7).
- sippy/functionsetSIM.py: the file with additional functions used by the Subspace Identification Methods functions and other useful functions for state space models (See Section 7).
- Examples/*.py: the folder containing various simulation examples, stressing different aspects of the identification package.

Every other file not mentioned above and inside the folder "sippy/", i.e. sippy/*.py, is called by the main file, hence the user has not to directly use them.
Any new example file written by the user can be executed within the same folder as `sippy/_init_.py`, or can be put in a level down folder, e.g. within the folder "Example". In order to run the example, few `import` lines must be inserted, lines which can be found at the beginning of the example files included in the zip file. For instance, the following line is needed to import the main function "system_identification":

```
from sippy import *
```

The line used to perform the identification is:

```
Identified_system = system_identification(INPUT_ARGUMENTS)
```

The specific input arguments depend on the used identification method, while the output is an object containing the attributes that define the identified model (see Section 6).

2 NECESSARY PARAMETERS

The following notation is used:

- L: is the number of samples of the collected data;
- n_u: the number of input variables to the system;
- n_y: the number of output variables.

Note that $L >> \max(n_y, n_u)$ to identify a significant and robust model of any type.
The three necessary input arguments to be defined are:

1. Output data: y ($\in \mathbb{R}^{n_y \times L}$ or $\in \mathbb{R}^{L \times n_y}$);
2. Input data: u ($\in \mathbb{R}^{n_u \times L}$ or $\in \mathbb{R}^{L \times n_u}$);
3. The structure of the identified model: a string.

The **linear** model to be identified - that is, the string to define - can be chosen between:

- an **input-output structure**, with 'FIR', 'ARX', 'ARMAX', 'ARMA', 'ARARX', 'ARARMAX', 'OE', 'BJ', 'GEN', as possible choices;
- a **state-space structure**, with 'N4SID', 'MOESP', 'CVA', 'PARSIM-P', 'PARSIM-S' or 'PARSIM-K', as different methods.

For instance, set:

```
Identified_system = system_identification(y,u,'ARMAX')
```

The optional parameters are all keyword arguments to be added after the necessary parameters and their name cannot be changed by the user. These arguments are written in the following Sections in `typographical style`.

3 Optional parameters common to all models and methods

The user can center to zero the collected input-output data with respect to:

- the mean value; in this case add: `centering = 'MeanVal'`;
- the initial value; in this case: `centering = 'InitVal'`;

By default, no centering is used: `centering = 'None'`.
In addition, the user can employ one of the implemented **Information Criteria** to find the best model orders when they are not selected or not known a-priori:

- *Akaike Information Criterion*, in this case add `IC='AIC'`;
- *Corrected Akaike Information Criterion*, in this case add: `IC='AICc'`;
- *Bayesian Information Criterion*, in this case: `IC='BIC'`.

By default, no information criterion is used: `IC='None'`.
The optional argument `tsample` is the sampling time used to build (compare in Python Library the functions `control.tf` and `control.ss`) the discrete-time linear model (see Section 6). By default, `tsample=1`.

4 Optional arguments if IC='*None*'

If `IC='None'`, no information criteria is used, and the user has to manually **define the various model orders**. Anyway, there is a default option for every model structure.
The optional arguments for each model set (input-output or state-space type) are discussed below.

4.1 Input-Output Models and Methods

Every identified input-output model is returned according to the following structure:

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

where $G(z)$ and $H(z)$ are transfer function matrices of polynomials in z, which is the forward shift operator (see Figure 4.1).
The underlying generalized model structure is:

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

The user can choose between the various structures reported in Table 4.1.
For each structure, the corresponding orders of the various blocks have to be set, since a specific parameter vector (Θ) and a corresponding regressor matrix (ϕ, in which the various φ_k are stacked) have to be defined [Ljung, 1999]. For example, in the case of a SISO ARMAX model, we have:

Table 4.1: Input-Output model structures and corresponding black-boxes to define.

Model structure	Polynomials in z used	
	$G(z)$	$H(z)$
FIR	$B(z)$	1
ARX	$A^{-1}(z)B(z)$	$A^{-1}(z)$
ARMAX	$A^{-1}(z)B(z)$	$A^{-1}(z)C(z)$
ARMA	$A^{-1}(z)$	$A^{-1}(z)C(z)$
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)	$A^{-1}(z)F^{-1}(z)B(z)$	$A^{-1}(z)D^{-1}(z)C(z)$

$$\Theta = [a_1\ a_2 \dots a_{n_a}\ b_1 \dots b_{n_b}\ c_1 \dots c_{n_c}]^T$$
$$\varphi_k = [-y_{k-1} \dots -y_{k-n_a}\ u_{k-1-\theta} \dots u_{k-n_b-\theta}\ \epsilon_{k-1} \dots \epsilon_{k-n_c}]^T$$

where $\epsilon_k = y_k - \hat{y}_k$ is the *prediction error*, being $\hat{y}_k$ the output model, and θ is the time-delay. In details, for the various structure of Table 4.1, the user has to set:

- n_a, orders of $A(z)$: the order of n_y outputs in a list containing n_y **integer** numbers;
- n_b, orders of $B(z)$: the orders of $n_y \times n_u$ input-output relations in a list containing n_y lists, in each list there are n_u **integer** numbers. Note that if an input has no influence on an output, set that order equal to 0;

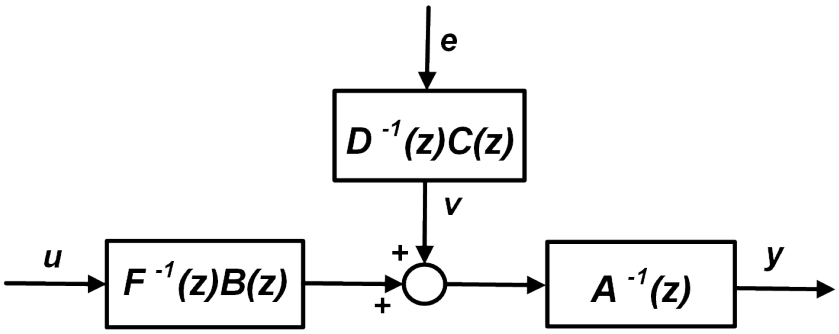

Figure 4.1: The structure of the generalized input-output model.

- n_c, orders of $C(z)$: the order of n_y error models (numerator in $H(z)$) in a list containing n_y **integer** numbers;
- n_d, orders of $D(z)$: the order of n_y error models (denominator in $H(z)$) in a list containing n_y **integer** numbers;
- n_f, orders of $F(z)$: the order of n_y undisturbed output model in a list containing n_y **integer** numbers;
- θ: the $n_y \times n_u$ input-output time-delays in a list containing n_y lists, in each of these lists there are n_u **integer** numbers.

For example, in case of:

- FIR model, add: `FIR_orders = [`n_b`,`θ`]`;
- ARX model, add: `ARX_orders = [`n_a`,`n_b`,`θ`]`;
- ARMAX, add: `ARMAX_orders = [`n_a`,`n_b`,`n_c`,`θ`]`;
- OE, add: `OE_orders = [`n_b`,`n_f`,`θ`]`;
- BJ, add: `BJ_orders = [`n_b`,`n_c`,`n_d`,`n_f`,`θ`]`;
- GEN, add: `GEN_orders = [`n_a`,`n_b`,`n_c`,`n_d`,`n_f`,`θ`]`;

By default, we have for:

- FIR case, `FIR_orders = [1,0]`;
- ARX case, `ARX_orders = [1,1,0]`;
- ARMAX, `ARMAX_orders = [1,1,1,0]`;
- OE, `OE_orders = [1,1,0]`;
- BJ, `ARMAX_orders = [1,1,1,1,0]`;
- GEN, `GEN_orders = [1,1,1,1,1,0]`.

Here below, some details on the **identification methods** implemented in SIPPY to obtain the desired model structures are given.

- To identify a FIR or an ARX model, a simple *linear least-square regression* is adopted through the computation of the psuedo-inverse of the regressor matrix (ϕ).
- The identification of an ARMAX model requires a pseudo-linear regression, that is, an indirect approach. SIPPY implements an iterative procedure, with *iterative least-square regression*. The parameter to set is ARMAX_mod = 'ILLS'. The number of maximum iterations can be specified by the user; by default its value is: `ARMAX_max_iterations=100`.

- All the other input-otput model structures (MODEL-TAG = ARMA, ARARX, ARARMAX, OE, BJ, GEN) imply a more involved *Prediction Error Method* and a nonlinear regression, due to the nonlinear effect of the parameter vector (Θ) to be identified in the regressor vector $\phi(\Theta)$. These structures are identified by solving a *NonLinear Program* by the use of the **CasADi** optimization tool [Andersson et al., 2019].
  The maximum iterations for the NLP can be specified by the user; anyway, by default its value is: `MODEL-TAG_max_iterations=200`.

- Note that ARMAX model can be identified also by the use of NLP: in this case, the user has to set ARMAX_mod = 'OPT'. By default, SIPPY identifies ARMAX structure with iterative least-square regression, so that, when not specified, ARMAX_mod = 'ILLS'.

- Finally, note that FIR, ARX, ARMAX, and OE models can be identified also by using a *recursive least-square regression*, in which a gain estimator, a covariance matrix and a suitable *Forgetting Factor* are computed. In this case, the user has to set MODEL-TAG = 'RLLS'.

Here an example for an ARMAX structure for a MIMO system (4 inputs × 3 outputs) is illustrated. Denoting z^{-1} as the backward shift operator, a MISO approach is employed, that is, **for each** of the three output the identified structure is:

$$\begin{aligned}(1 + a_1 z^{-1} + ... + a_{n_a} z^{-n_a}) y_k = {} & (b_{1,1} z^{-(1+\theta_1)} + ... + b_{1,n_{b1}} z^{-(n_{b1}+\theta_1)}) u_k^{(1)} + \\ & + (b_{2,1} z^{-(1+\theta_2)} + ... + b_{2,n_{b2}} z^{-(n_{b2}+\theta_2)}) u_k^{(2)} \\ & + (b_{3,1} z^{-(1+\theta_3)} + ... + b_{3,n_{b3}} z^{-(n_{b3}+\theta_3)}) u_k^{(3)} + \\ & + (b_{4,1} z^{-(1+\theta_4)} + ... + b_{4,n_{b4}} z^{-(n_{b4}+\theta_4)}) u_k^{(4)} + \\ & + ... + (1 + c_1 z^{-1} + ... + c_{n_c} z^{-n_c}) e_k\end{aligned} \tag{4.1}$$

Where n_a denotes the output order, n_{bi} denotes the order of the *i-th* input, θ_i denotes the delay of the *i-th* input, n_c denotes the error model order, a_j denotes the *j-th* coefficient of the output, $b_{i,j}$ denotes the *j-th* coefficient of the *i-th* input and c_j denotes the *j-th* coefficient of the error model.
Table 4.2 shows the considered transfer functions for the ARMAX model, in which:

$$\begin{gathered}n_a = [3,\ 1,\ 2] \\ n_b = [[2,\ 1,\ 3,\ 2], [3,\ 2,\ 1,\ 1], [1,\ 2,\ 1,\ 2]] \\ n_c = [2,\ 2,\ 3] \\ \theta = [[1,\ 2,\ 2,\ 1], [1,\ 2,\ 0,\ 0], [0,\ 1,\ 0,\ 2]]\end{gathered}$$

Table 4.2: Example of MIMO ARMAX system.

	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	$h_1 = \frac{z^5+0.85z^4+0.32z^3}{z^5-0.3z^4-0.25z^3-0.021z^2}$	$h_2 = \frac{z^4+0.4z^3+0.05z^2}{z^4-0.4z^3}$	$h_3 = \frac{z^4+0.7z^3+0.485z^2+0.22z}{z^4-0.1z^3-0.3z^2}$

4.2 State-Space Models and Subspace Identification Methods: 'N4SID', 'MOESP', 'CVA', 'PARSIM-P', 'PARSIM-S' and 'PARSIM-K'

Subspace IDentification (SID) Methods identify directly a state space model structure, that can be represented in three forms, recalled below.
Process form:

$$\begin{cases} x_{k+1} = Ax_k + Bu_k + w_k \\ y_k = Cx_k + Du_k + v_k \end{cases} \tag{4.2}$$

where: $y_k \in \mathbb{R}^{n_y}$, $x_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^{n_u}$, $w_k \in \mathbb{R}^n$ and $v_k \in \mathbb{R}^{n_y}$ are the system output, state, input, state noise and output measurement noise respectively (the subscript "k" denotes the $k-th$ sampling time); $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times n_u}$, $C \in \mathbb{R}^{n_y\times n}$, $D \in \mathbb{R}^{n_y\times n_u}$ are the system matrices.
Innovation form:

$$\begin{cases} x_{k+1} = Ax_k + Bu_k + Ke_k \\ y_k = Cx_k + Du_k + e_k \end{cases} \tag{4.3}$$

Predictor form:

$$\begin{cases} x_{k+1} = A_K x_k + B_K u_k + Ky_k \\ y_k = Cx_k + Du_k + e_k \end{cases} \tag{4.4}$$

where the following relations hold:

$$A_K = A - KC$$
$$B_K = B - KD$$

where K is the steady-state *Kalman filter gain*, obtained from Algebraic Riccati Equation.
The user has to define the future and past horizons (`SS_f` and `SS_p` respectively).
For *traditional* methods, that is, N4SID', 'MOESP' and 'CVA' methods, the future and past horizons are equal, set by default `SS_f=20` (integer number).
For *parsimonious* methods, that is, 'PARSIM-P', 'PARSIM-S' and 'PARSIM-K' methods, the future and past horizons can be set, by default: `SS_f=20`, `SS_p=20` (integer numbers).
After performing the singular value decomposition (SVD) scheduled for the identification,

which allows building the suitable subspace from the original data space, the corresponding model order n (that is, the number of states) can be chosen by the settings:

- a *threshold value* (`SS_threshold`, between 0.0 and 1.0) and the maximum order (`SS_max_order`, integer number);
- a *fixed order* (`SS_fixed_order`, integer number).

The threshold value implies that all singular values such that $\frac{\sigma_i}{\sigma_{max}} <$ `SS_threshold` are discarded, where σ_{max} is the largest singular value, and σ_i is i-th singular value.
The maximum order value limits the order of the model to that value.
Using a fixed order value, the maximum order and the threshold value are not taken into account.
By default: `SS_threshold=0.1`, while the other parameters are not specified.
The other optional arguments are by default:

- `SS_D_required=False`;
- `SS_A_stability=False` (only for 'N4SID', 'MOESP' and 'CVA').

If `SS_D_required=False`, the state space model will have the matrix D filled with zeros; add `SS_D_required=True` to normally compute the matrix D (remind that $D = 0$ is typical for *causal* systems, e.g. physical/chemical systems).
For 'N4SID', 'MOESP' and 'CVA' methods, setting `SS_A_stability=True` will guarantee the stability of the matrix A. After the normal evaluation of the matrix A, if the computed A is not stable, the stability is forced and a warning message is shown: "Forcing A stability". Note that this can lead to significant errors in the identified models.
In addition, only for PARSIM-K method, a revaluation step for matrix B and initial state x_0 can be required by the user, adding `SS_PK_B_reval=True`. This revaluation step may lead to better performance of the model in the process form.

5 Optional arguments if IC='AIC', 'AICc' or 'BIC'

By using an Information Criterion (IC), the user does not need to the define the exact model orders, but he/she has to assign a **range of orders** and let the IC find the best combination.
Note that, when an input-output model structure is selected, the information criteria are implemented only for SISO systems. Therefore, for MIMO cases, the user has to defined every model orders or can use the default values, as discussed in Section 4.1. In the case of state-space models, information criteria are implemented for both SISO and MIMO systems.

5.1 Input-Output Models and Methods

As just said, the user has to define the range of the various model orders, according to the selected SISO structure. In details:

- for the blocks $A(z)$, $C(z)$, $D(z)$, $F(z)$, define the order range in a list with two integer numbers, by default: `na_ord=[0,5]`, `nc_ord=[0,5]`, `nd_ord=[0,5]`, `nf_ord=[0,5]`;

- for $B(z)$, that is, for the input order range, define a list with two integer numbers, by default: `nb_ord=[1,5]`;
- for θ, that is, for the input time-delay range, assign a list with two integer numbers, by default: `delays=[0,5]`;

Finally, note that the other input arguments, as the ones concerning the maximum iteration number (e.g., `MODEL-TAG_max_iterations`), are the same as described in Section 4.1.

5.2 SUBSPACE IDENTIFICATION METHODS: 'N4SID', 'MOESP', 'CVA', 'PARSIM-P', 'PARSIM-S' AND 'PARSIM-K'

The following input arguments are the same described in Section 4.2: `SS_f, SS_p, SS_D_required, SS_A_stability`. When an information criterion is selected, the model order n is selected in the defined range `SS_orders`, a list with two integer numbers; by default: `SS_orders=[1,10]`.
In the case of PARSIM-K method, by adding `SS_PK_B_reval=True`, the revaluation step for matrix B and initial state x_0 is performed externally to the evaluation of the best model according to the information criterion; therefore, this is only a final step.

6 ATTRIBUTES OF THE MODEL

Depending on the selected identification model/method, the object returned by the function "system_identification" has different attributes. For example, to recall the attribute `G` of the object `Identified_system`, the command line is:

```
Identified_system.G
```

6.1 INPUT-OUTPUT MODELS AND METHODS

The attributes for these methods are:

- `na`, `nb`, `nc`, `nd`, `nf`, `theta`: the order list for the block $A(z)$, $B(z)$, $C(z)$, $D(z)$, $F(z)$, respectively, and the order list of input time-delay θ.
- `G`: the transfer function matrix that relate inputs with outputs;
- `H`: the transfer function matrix of the error model;
- `ts`: the sampling time of the discrete transfer functions;
- `NUMERATOR, DENOMINATOR`: two lists containing the coefficients in z of the numerators and the denominators of `G`, respectively. The matrix `G` can be built with the line:

```
G=control.tf(NUMERATOR,DENOMINATOR,ts)
```

- `NUMERATOR_H`, `DENOMINATOR_H`: two lists containing the coefficients in z of the numerators and the denominators of `H`. The matrix `H` can be built with the line:

```
H=control.tf(NUMERATOR_H,DENOMINATOR_H,ts)
```

- `Vn`: the estimated error norm, defined as:

$$\text{Vn}= \frac{1}{2L} tr(\epsilon\epsilon^T)$$

where tr is the trace operator, ϵ is the prediction error sequence, computed as:

$$\epsilon = y - \hat{y}$$

being $\hat{y}$ the output of the identified model. This norm is calculated on the outputs normalized with respect to their standard deviation.

- `Yid`: output of the identified model obtained in the identification stage ($\hat{y}$).

6.2 Subspace identification methods: 'N4SID', 'MOESP', 'CVA', 'PARSIM-P', 'PARSIM-S' and 'PARSIM-K'

The attributes for these methods are:

- `A, B, C, D`: the system matrices of (4.2).
- `A_K, B_K`: the system matrices of (4.4). These matrices for 'N4SID', 'MOESP' and 'CVA' methods can be obtained only if the Kalman filter gain `K` is calculated.
- `K`: the Kalman filter gain. This matrix for 'N4SID', 'MOESP' and 'CVA' methods can be obtained only if the package `Slycot` is full-installed.
- `ts`: sampling time of the identified discrete system `G`;
- `G`: linear state-space model that relates inputs with outputs, obtained as follows:

```
G=control.ss(A,B,C,D,ts)
```

- `x0`: initial state estimate. For 'N4SID', 'MOESP' and 'CVA' methods, `x0` is filled with zeros (that is, no estimate of the initial state is performed).
- $\texttt{Q}^{(*)}$, $\texttt{R}^{(*)}$, $\texttt{S}^{(*)}$: covariance matrices defined as follows:

$$\begin{pmatrix} Q & S \\ S^T & R \end{pmatrix} = \mathbf{E}[\begin{pmatrix} \rho_w \\ \rho_v \end{pmatrix} (\rho_w^T \, \rho_v^T)]$$

where ρ_w and ρ_v are the estimated sequences of the state noise and output measurement noise (considering the outputs normalized with respect to their standard deviations). **E** denotes the expected value operator.
(*) Note: these attribute exist only if the selected method is 'N4SID', 'MOESP' or 'CVA'.

- `Vn`: the same `Vn` of subsection 6.1. This norm for PARSIM methods is calculated using the predictor form (4.4), while for 'N4SID', 'MOESP' and 'CVA' methods using the process form (4.2), i.e., two values of this norm can be compared only if obtained from the same class of methods.
  Note that for PARSIM-K method, if the user selected `SS_PK_B_reval=True`, the norm is calculated in the process form.

7 USEFUL FUNCTIONS

To use the functions contained in the file "functionset.py", the following line is needed:

```
from sippy import functionset as fset
```

The functions that can be useful for the user are:

- `fset.GBN_seq`: this function returns a Generalized Binary Noise (GBN) sequence [Zhu, 2001] . The input arguments are:
  1. `N`: data sequence length (total number of samples);
  2. `p_swd`: desired probability of switching (0: no switch, 1: always switch);
  3. `Nmin` (optional): minimum number of samples between two switches (by default `Nmin=1`);
  4. `Range` (optional): upper and lower values of the sequence, i.e.: `Range=[3.,15.]` (by default `Range=[-1.0,1.0]`);
  5. `Tol` (optional): tolerance on switching probability relative error (by default `Tol = 0.01`);
  6. `nit_max` (optional): maximum number of iterations to get the desired switching probability (by default `nit_max = 30`).

  For example, by using the command line:

  ```
  [gbn, p_swv, Nsw] = GBN_seq(1000, 0.1, Nmin = 20, Range = [-10, 5])
  ```

  - `gbn`: is an array of length equal to 1000, 10% of switch probability, switching at least every 20 samples, between -10 or 5;
  - `p_swv`: is the actual probability of switching (which may differ a little from `p_swd`, according to the set tolerance);
  - `Nsw`: is the number of switches in the selected GBN sequence.

- `fset.RW_seq`: this function returns a random signal sequence (a random walk from a normal distribution). The input arguments are:
  1. `N`: data sequence length (total number of samples);
  2. `InValue`: Initial value;
  3. `sigma` (optional): standard deviation (by default, sigma = 1).

E.g., using the command line:

```
rn=fset.RW_seq(100,10, sigma = 0.01)
```

`rw` is an array with size=100, 10 as initial value, and standard deviation of 0.01.

- `fset.white_noise`: this function adds a white noise to a single signal. The input arguments are:
  1. `y`: original signal
  2. `A_rel`: relative amplitude. The standard deviation of the returned white noise is the standard deviation of y multiplied by `A_rel`.

  The outputs are:
  1. `errors`: the white noise;
  2. `y_err`: the overall signal, that is, equal to errors+y

  The command line is:

```
errors,y_err=white_noise(y,A_rel)
```

- `fset.white_noise_var`: this function generates a white noise matrix (rows - the same of `y` - with zero mean). The input arguments are:
  1. `L`: size (number of columns of the returned matrix);
  2. `Var`: variance list;

  e.g. the following command line:

```
noise=white_noise_var(100,[1,2])
```

  returns a matrix with 100 columns (the data number) and two rows with variances 1 and 2, respectively.

- `fset.validation`: this function performs *k-step* ahead validation of input-output type models in the predictor form.
  In particular, when $k = 1$, *1-step-ahead* predictor is:

$$\hat{y}_k = H^{-1}(z)G(z)u_k + (1 - H^{-1}(z))y_k \tag{7.1}$$

  while general *k-step-ahead* predictor is:

$$\hat{y}_k = W_k(z)G(z)u_k + (1 - W_k(z))y_k \tag{7.2}$$

  where $W_k(z) = \bar{H}_k(z)H^{-1}(z)$, and $\bar{H}_k(z) = \sum_{j=0}^{k-1} h_j z^{-j}$, being $\{h_j\}$ the coefficients of the finite impulse response of Transfer Function $H(z)$.

  This function is very useful when the user wants to cross-validate the identified input/output model, that is, test the previously identified model on new data not used in the identification stage. The input arguments are:

1. SYS: the system to validate (that is, an identified input/output model);
2. u: input data;
3. y: output data;
4. Time: time sequence;
5. k: number of k-step ahead (by default, k = 1);
6. centering: to set how to center input-output validation data; with respect to:
   - the mean value; in this case add: `centering = 'MeanVal'`;
   - the initial value; in this case: `centering = 'InitVal'`;
   - by default, no centering is performed: `centering = 'None'`.

For example, by using the command line:

```
Yval = validation(SYS,u,y,np.linspace(0, 1000, 1001), k = 3):
```

Yval is an array with the number of outputs as rows and 1001 (the number of samples) as columns, obtained by 3-step ahead validation of SYS model.

To use the functions contained in the file "functionsetSIM.py", the following line is needed:

```
from sippy import functionsetSIM as fsetSIM
```

The functions that can be useful for the user are:

- fsetSIM.SS_lsim_process_form: this function performs a simulation in the process form, given the identified system matrices, the input sequence (an array with n_u rows and L columns) and the initial state estimate (array with n rows and one column). The state sequence and the output sequence are returned. The command line is:

```
x,y=fsetSIM.SS_lsim_process_form(A,B,C,D,u,x0)
```

- fsetSIM.SS_lsim_predictor_form: this function performs a simulation in the predictor form, given the identified system matrices, the Kalman filter gain, the real output sequence (array with n_y rows and L columns, the input sequence (an array with n_u rows and L columns) and the initial state estimate (array with n rows and one column). The state sequence and the estimated output sequence are returned. The command line is:

```
x,y_hat=SS_lsim_predictor_form(A_K,B_K,C,D,K,y,u,x0)
```

- fsetSIM.SS_lsim_innovation_form: this function performs a simulation in the innovation form. This function is analogous to the previous one, using the system matrices A and B instead of A_K and B_K:

```
x,y_hat=SS_lsim_innovation_form(A,B,C,D,K,y,u,x0)
```

In these functions, the initial state is an optional argument and if it is not given, is taken as zeros. The user can perform a simulation with these functions recalling the attributes of the `Identified_system`, e.g.:

```
x,y=fsetSIM.SS_lsim_process_form(Identified_system.A,
Identified_system.B, Identified_system.C,Identified_system.D,
u,Identified_system.x0)
```

8 METHODS SUMMARY

For each method/model, the considered parameters (excluding `centering` and `tsample`, being common to all methods) and the attributes of the system are reported in the Table 8.1.

9 PROBLEMS AND SOLUTIONS

Here some typical problems that the user may meet are illustrated.

- **LinAlgError: SVD did not converge.** The user can try to solve the problem by changing `SS_f` and `SS_p` (future and past horizons) or, if this does not work, changing identification method.
- For 'N4SID', 'MOESP' and 'CVA' methods, if the message "**Kalman filter cannot be calculated**" is shown, the problem can be that the `Slycot` package is not well installed, otherwise function `control.dare` is not able to solve the Discrete Algebraic Riccati Equation. To check if `Slycot` works properly, use the state space example (attached in Identification_code.zip), where the `K` should be calculated. The `Slycot` package is available at:

  `https://pypi.org/project/slycot/`

  Alternatively, the binaries can be found at:

  `https://www.lfd.uci.edu/~gohlke/pythonlibs/`

- The **identification is not well performed**, by returning an unstable system:
  - in input/output models, when using optimization-based methods.
    The following message is displayed: *Warning: One of the identified system is not stable.* Note that the user may select to impose a stability constraint (`stab_cons = True`) on the identified models. In this case, if the stability constraint is violated, SIPPY returns the following message: *Infeasible solution.* This means that the maximum pole of the identified Transfer Functions is beyond the imposed stability margin `stab_margin`, which can be selected by the user.
    Note that `stab_margin` must be in the range $\in [0,1]$; by default, `stab_margin` = 1.0, and `stab_cons = False`.

Table 8.1: Methods and their parameters and attributes

Method	IC=	Input parameters	Output Attributes
MODEL-TAG = 'ARX', 'FIR'	'None'	MODEL-TAG_orders	G, H, na, nb, theta, ts, NUMERATOR, DENOMINATOR, Vn, Yid
	'AIC', 'AICc' or 'BIC'	na_ord, nb_ord, delays	
MODEL-TAG = 'ARMAX' 'ARMA', 'ARARX', 'ARARMAX', 'OE', 'BJ' or 'GEN'	'None'	MODEL-TAG_orders, MODEL-TAG_max_iterations	G, H, na, nb, nc, nd, nf, theta, ts, NUMERATOR, DENOMINATOR, Vn, Yid, NUMERATOR_H, DENOMINATOR_H
	'AIC', 'AICc' or 'BIC'	na_ord, nb_ord, nc_ord, nd_ord, nf_ord, delays, MODEL-TAG_max_iterations	
'N4SID', 'MOESP' or 'CVA'	'None'	SS_f, SS_threshold, SS_max_order, SS_fixed_order, SS_D_required, SS_A_stability	A, B, C, D, A_K, B_K, K, ts, G, x0, Q, R, S, Vn
	'AIC', 'AICc' or 'BIC'	SS_f, SS_orders, SS_D_required, SS_A_stability	Same as above
'PARSIM-P' or 'PARSIM-S'	'None'	SS_f, SS_p, SS_threshold, SS_max_order, SS_fixed_order, SS_D_required	A, B, C, D, A_K, B_K, K, ts, G, x0, Vn
	'AIC', 'AICc' or 'BIC'	SS_f, SS_p, SS_orders, SS_D_required	Same as above
'PARSIM-K'	'None'	SS_f, SS_p, SS_threshold, SS_max_order, SS_fixed_order, SS_D_required, SS_PK_B_reval	A, B, C, D, A_K, B_K, K, ts, G, x0, Vn
	'AIC', 'AICc' or 'BIC'	SS_f, SS_p, SS_orders, SS_D_required, SS_PK_B_reval	Same as above

  - in state space models.
    The user may try different options, changing the `centering` argument, resetting the future and past horizons, lowering the `SS_threshold` value, and so on. For PARSIM methods, a good advice can be to set `SS_p` $\geq$ `SS_f`.

- **Other Issues**. Recent and common problems are faced and solved within the GitHub at: `https://github.com/CPCLAB-UNIPI/SIPPY/issues`.

REFERENCES

J. A.E. Andersson, J. Gillis, G. Horn, J.B. Rawlings, and M. Diehl. CasADi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

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

Y. Zhu. *Multivariable System Identification For Process Control*. Elsevier Science, first edition, 2001.