UNIVERSITY OF PISA, DEPARTMENT OF CIVIL AND INDUSTRIAL ENGINEERING

Systems Identification Package for Python: User Guide

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

July 5, 2018

This document is intended to give a guideline to the user who has to perform his own identification with collected data.

The program is written both for Python 2.7 and 3.6 (supported on Windows \ Linux \ Mac). The following packages must be installed: NumPy, SciPy, control, math, Slycot. In order to make the code compatible to both Python 2.7 and 3.6, 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`, or searching for `future` among non installed packages in Anaconda Navigator (more information can be found at https://anaconda.org/anaconda/anaconda-navigator).

The Identification_code is distributed as packed file Identification_code.zip that contains the following items:

- sys_identification.py: main file containing the function that has to be recalled to perform the identification

- lib/functionset.py: file containing most of the functions used by the identification functions and other useful functions (See Section 6).

- lib/functionsetSIM.py: additional functions used by the Subspace identification functions and other useful functions for state space models (See Section 6).

- Examples/*.py: folder containing various examples, stressing different aspects of the package.

In the folder "lib/" there are other files ".py", that are called by the main file, so the user has not to use them.

1

The example file created by the user can be in the same folder as `sys_identification.py` of can be in a level down folder e.g. the folder "Example". In order to run the example, few `import` lines must be inserted, e.g. take as reference the example files already in the zip file. For instance, the following line is needed to import the main function "system_identification":

```
from sys_identification import *
```

The line used to perform the identification is:

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

The input arguments depend on the used identification method, while the output is an object containing the attributes that define the model (see Section 5).

1 NECESSARY PARAMETERS TO BE DEFINED

Using the following notation:
L: number of sampling times of the collected data,
n_y: number of the controlled variables,
n_u: number of the manipulated variables.
Being $L >> \max(n_y, n_u)$.
The three necessary input arguments to define 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. Identification method: a string.

The identification method can be chosen between: 'ARX', 'ARMAX', 'N4SID', 'MOESP', 'CVA', 'PARSIM-P', 'PARSIM-S' or 'PARSIM-K'. For instance:

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

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

2 OPTIONAL PARAMETERS COMMON TO ALL METHODS

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

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

- the initial condition, in this case: `centering='InitVal'`;

By default `centering='None'`.
The user can use one of the implemented Information Criteria to select the model order:

- 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 IC='None'.
The optional argument `tsample` is the sampling time used to build (see in Python Library the functions control.tf and control.ss) the transfer functions of the model (see Section 5). By default `tsample=1`.

3 OPTIONAL ARGUMENTS IF IC='*None*'

If IC='None', the user has to define the model orders. Anyway, there is a default option. The optional arguments for each model set are discussed below.

3.1 'ARX' AND 'ARMAX' METHODS

The user has to define:

- n_a: the n_y output orders in a list containing n_y **integer** numbers;

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

- n_c (only for 'ARMAX'): the n_y error model orders in a list containing n_y **integer** numbers;

- θ: the $n_y \times n_u$ input delays in a list containing n_y lists, in each of these lists there are n_u **integer** numbers;

For the ARX identification, add: `ARX_orders=`$[n_a, n_b, \theta]$.
For the ARMAX identification, add: `ARMAX_orders=`$[n_a, n_b, n_c, \theta]$.
The alternative is a list containing integer numbers:

- for the ARX case, by default: `ARX_orders=[1,1,0]`;

- for the ARMAX case, by dafault: `ARMAX_orders=[1,1,1,0]`

The respective lists (n_a, n_b, n_c and θ) will contain numbers equal to the one specified.
The ARMAX identification has an iterative procedure and the number of maximum iterations can be specified, by default: `ARMAX_max_iterations=100`.
Denoting z^{-1} as the backward shift operator and z as the forward shift operator, **for each output** the identified (ARMAX) structure is:

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

$$(1 + a_1 z^{-1} + ... + a_{n_a} z^{-n_a})y(t) = (b_{1,1} z^{-(1+\theta_1)} + ... + b_{1,n_{b1}} z^{-(n_{b1}+\theta_1)})u_1(t)+$$
$$+ (b_{2,1} z^{-(1+\theta_2)} + ... + b_{2,n_{b2}} z^{-(n_{b2}+\theta_2)})u_2(t)+ \qquad (3.1)$$
$$+ ... + (1 + c_1 z^{-1} + ... + c_{n_c} z^{-n_c})e(t)$$

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. In the ARX structure $n_c = 0$.

The transfer functions of the system are returned as fractions of polynomials in z:

$$y = G(z)u + H(z)e$$

For instance, Table 3.1 shows the identified matrices obtained for a MIMO system using the ARMAX method, in which

$$n_a = [3, 1, 2]$$
$$n_b = [[2, 1, 3, 2], [3, 2, 1, 1], [1, 2, 1, 2]]$$
$$n_c = [2, 2, 3]$$
$$\theta = [[1, 2, 2, 1], [1, 2, 0, 0], [0, 1, 0, 2]]$$

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

Subspace identification methods (SIM) identify a state space model structure, that can be represented in three forms, recalled below.

Process form:

$$\begin{cases} x_{k+1} = Ax_k + Bu_k + w_k \\ y_k = Cx_k + Du_k + v_k \end{cases} \qquad (3.2)$$

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:

$$\begin{cases} x_{k+1} = Ax_k + Bu_k + Ke_k \\ y_k = Cx_k + Du_k + e_k \end{cases} \tag{3.3}$$

Predictor form:

$$\begin{cases} x_{k+1} = A_K x_k + B_K u_k + Ky_k \\ y_k = Cx_k + Du_k + e_k \end{cases} \tag{3.4}$$

where the following relations hold:

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

The user has to define the future and past horizons (`SS_f` and `SS_p` respectively). For 'N4SID', 'MOESP' and 'CVA' methods, the future and past horizons are equal, set by default `SS_f=20` (integer number).

For '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, the model order n is 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 imply that all singular values such that:
$\frac{\sigma_i}{\sigma_{max}} <$ `SS_threshold` are discarded.
σ_{max}: largest singular value, σ_i: 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 ones 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 have the matrix D calculated.
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 calculated A is not stable, the stability is forced and a warning message is shown: "Forcing A stability". This can

lead to significant errors in the identified models.

Only in PARSIM-K method, a revaluation step for B and x_0 can be required by the user, adding `SS_PK_B_reval=True`. This revaluation step may lead to better performances of the model in the process form.

4 OPTIONAL ARGUMENTS IF IC='AIC', 'AICC' OR 'BIC'

Using an information criterion, the user has to define a range for the model orders.

4.1 'ARX' AND 'ARMAX' METHODS

The `ARMAX_max_iterations` argument is the same described in Section 3.1.

The user has to define the ranges of the model orders:

- The output order range in a list with two integer numbers, by default: `na_ord=[0,5];`

- The input order range in a list with two integer numbers, by default: `nb_ord=[1,5];`

- The error order range (only for ARMAX) in a list with two integer numbers, by default: `nc_ord=[0,5];`

- The input delays range in a list with two integer numbers, by default: `delays=[0,5];`

For ARX and ARMAX identification, the information criteria are implemented only in the SISO case.

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

The following arguments are the same described in Section 3.2: `SS_f`, `SS_p`, `SS_D_required`, `SS_A_stability`. The order is selected in the defined range `SS_orders`, a list with two integer numbers, by default: `SS_orders=[1,10]`.

Adding `SS_PK_B_reval=True`, the revaluation step for B and x_0 is performed externally to the evaluation of the best model according to the information criterion and therefore only as a final step.

5 ATTRIBUTES OF THE MODEL

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

$$Identified\_system.G$$

5.1 'ARX' AND 'ARMAX' METHODS

The attributes for these methods are:

- `na`, `nb`, `nc`[(*)], `theta`: respectively output orders, input orders, error model orders, input delays;

- `G`: transfer functions that relate inputs with outputs;

- `H`: transfer functions of the error models;

- `ts`: sampling time of the transfer functions;

- `NUMERATOR, DENOMINATOR`: lists containing coefficients respectively of the numerators and of the denominators of `G`. The `G` is built with the line:

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

- `NUMERATOR_H`[(*)], `DENOMINATOR_H`[(*)]: lists containing coefficients respectively of the numerators and of the denominators of `H`. The `H` is built with the line:

$$H=control.tf(NUMERATOR\_H,DENOMINATOR\_H,ts)$$

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

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

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

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

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

In the subsection 5.1, the superscript (∗) means that the attribute can be found only in the ARMAX structure.

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

The attributes for these methods are:

- `A, B, C, D`: the system matrices of (3.2).

- `A_K, B_K`: the system matrices of (3.4). 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 transfer function `G`;

- `G`: transfer function 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 (no estimate of the initial state is made).

- $Q^{(*)}$, $R^{(*)}$, $S^{(*)}$: 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). $\mathbf{E}$ denotes the expected value operator.

- `Vn`: the same `Vn` of subsection 5.1. This norm for PARSIM methods is calculated using the predictor form (3.4), while for 'N4SID', 'MOESP' and 'CVA' methods using the process form (3.2), i.e. two values of this norm can be compared only if obtained from the same class of methods. In PARSIM-K method, if the user selected `SS_PK_B_reval=True`, the norm is calculated in the process form.

In the subsection 5.2, the superscript $(*)$ means that the attribute exists only if the selected method is 'N4SID', 'MOESP' or 'CVA'.

6 USEFUL FUNCTIONS

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

```
from lib import functionset as fset
```

The functions that can be useful for the user are:

- `fset.PRBS_seq`: this function returns a pseudo-random binary sequence. The input arguments are:
  1. `length_arg`: size of the sequence;
  2. `prob_switch`: probability of switch (0:no switch, 1:always switch);
  3. `Range` (optional): Upper and lower values of the sequence, example: `Range=[3.,15.]` (by default `Range=[-1.0,1.0]`).

  E.g., using the command line:

```
prbs=fset.PRBS_seq(100,0.1,Range=[-10.0,5.0])
```

prbs is an array with size=100, 10 % of switch probability, elements that can be -10 or 5.

- `fset.white_noise`: this function adds a white noise to a 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`: equal to errors+y

  The command line is:

$$\text{errors,y\_err=white\_noise(y,A\_rel)}$$

- `fset.white_noise_var`: this function generates a white noise matrix (rows 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:

$$\text{noise=white\_noise\_var(100,[1,2])}$$

  returns a matrix that has 100 columns and two row vectors with variances 1 and 2.

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

$$\text{from lib import functionsetSIM as fsetSIM}$$

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 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:

$$\text{x,y=fsetSIM.SS\_lsim\_process\_form(A,B,C,D,u,x0)}$$

- `fsetSIM.SS_lsim_predictor_form`: this function performs a simulation in the predictor form, given the 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, the initial states are 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)
```

7 METHODS SUMMARY

For each method, the considered parameters (excluding `centering` and `tsample`, being common to all methods) and the attributes of the system are reported in the Table 7.1.

8 PROBLEMS AND SOLUTIONS

Here some problems that the user can meet are presented.

- **LinAlgError: SVD did not converge.** The user can try to solve the problem changing `SS_f` and `SS_p` or, if this does not work, changing 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 "control.dare" is not able to solve the Riccati equation. To check if Slycot works, use the state space example (attached in Identification_code.zip), where the `K` should be calculated. The Slycot package is available at:

  `https://pypi.python.org/pypi/slycot/0.2.0`

  Alternatively the binaries can be found at:

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

- The **identification is not well performed** (e.g. returning an unstable system in State space models). The user has to try different options, changing the `centering` argument, resetting the future and past horizons, lowering the `SS_threshold` value, and so on. For PARSIM methods, an advice can be to set $SS\_p \geq SS\_f$.

Table 7.1: Methods and their parameters and attributes

Method	IC=	Method parameters	Attributes
'ARX'	'None'	`ARX_orders`	`G, H, na, nb, theta, ts,` `NUMERATOR, DENOMINATOR, Vn`
	'AIC', 'AICc' or 'BIC'	`na_ord, nb_ord, delays`	Same as above
'ARMAX'	'None'	`ARMAX_orders,` `ARMAX_max_iterations`	`G, H, na, nb, nc, theta, ts,` `NUMERATOR, DENOMINATOR, Vn,` `NUMERATOR_H, DENOMINATOR_H`
	'AIC', 'AICc' or 'BIC'	`na_ord, nb_ord, nc_ord, delays,` `ARMAX_max_iterations`	Same as above
'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