HIT
Hybrid Identification Toolbox

Giancarlo Ferrari-Trecate
DIS, Università degli Studi di Pavia
Via Ferrata 1, 27100 Pavia, Italy
giancarlo.ferrari@unipv.it

1 Introduction

The Hybrid Identification Toolbox (HIT) is a free MatLab toolbox for regression with PieceWise Affine (PWA) maps and identification of PieceWise AutoRegressive eXogenous (PWARX) models. A key feature of HIT is the capability of reconstructing discontinuous models. HIT implements the clustering-based algorithms described in the papers [4, 3, 2, 5]. In addition, HIT provides facilities for plotting and validating the identified models.

2 Installation

HIT uses routines of the MPT toolbox [9] for handling polytopes and solving Linear Programming (LP) and Quadratic Programming (QP) problems.

The latest version of HIT can be downloaded at the webpage

`http://www-rocq.inria.fr/who/Giancarlo.Ferrari-Trecate/HIT_toolbox.html`

For installing it, decompress the .zip file in a directory and add its path (and the path of the subdirectories) to the `matlabpath`.

HIT is also shipped as part of MPT and it is automatically installed with a release of MPT grater than 2.5. The latest version of the MPT toolbox is available at the webpage

`http://control.ee.ethz.ch/~mpt/`

The HIT toolbox consists of the following directories

`/hit`	main functions
`/hit/analysis`	functions for validating models
`/hit/auxiliary`	auxiliary functions
`/hit/clustering`	clustering algorithms
`/hit/docs`	pdf and html documentation
`/hit/examples`	examples demonstrating the functionalities of HIT
`/hit/pattern_rec`	pattern recognition algorithms
`/hit/plotting`	plotting functions

For familiarizing quickly with HIT, examples of the most typical identification experiments are supplied in the directory `/hit/examples`. These .m files can also be used as templates for your personal experiments.

3 An introduction to hybrid identification

The aim of this section is to provide a concise overview on hybrid identification and to introduce the jargon used in HIT. The basic problem solved by HIT is the reconstruction of a Piece-Wise Affine (PWA) map from a finite number of noisy data points. A PWA map $f : \mathcal{X} \mapsto \mathbb{R}$ is defined by the equations

$$f(x) = f_i(x) \quad \text{if} \quad \lambda(x) = i \tag{1}$$

$$f_i(x) = \begin{bmatrix} x^T & 1 \end{bmatrix} \theta_i \tag{2}$$

together with a bounded polyhedron $\mathcal{X} \subset \mathbb{R}^n$ $\{\mathcal{X}_i\}_{i=1}^s$ and a polyhedral partition $\{\mathcal{X}_i\}_{i=1}^s$ of $\mathcal{X}$ in s regions[1]. The map $\lambda(x)$ is the *switching function* defined as $\lambda(x) = i \Leftrightarrow x \in \mathcal{X}_i$, the vectors $\theta_q \in \mathbb{R}^{n+1}$ are *Parameter Vectors (PVs)* and $\mathcal{X}$ is termed the *regressor set*. Therefore, a PWA map is composed of s affine *modes* defined by the pairs $(\theta_i, \mathcal{X}_i)$.

The dataset $\mathcal{N}$ collects the samples $(x(k), y(k))$, $k = 1, \dots, N$ generated by the model

$$y(k) = f(x(k)) + \eta(k) \tag{3}$$

where $\eta(k)$ are noise samples corrupting the measurements, $x(k)$ are called *regressors* and $y(k)$ are termed *output samples*.

Assume that all modes are represented in the data. The aim of PWA regression is to estimate s, the PVs and the regions by using the information provided by $\mathcal{N}$. Note that usually the regressor set is known since it can be deduced from physical bounds on the regressors or it can be set equal to the smallest hyperrectangle containing all regressors.

When considering hybrid systems, an input/output description of a PWA system with inputs $u(k) \in \mathbb{R}^m$ and outputs $y(k) \in \mathbb{R}$ is provided by PieceWise ARX (PWARX) models that are defined by equation (3) where k is now the time index and the vector of regressors $x(k)$ is given by

$$x(k) = \begin{bmatrix} y(k-1) & y(k-2) & \dots & y(k-n_a) & u^T(k-1) & u^T(k-2) \dots & u^T(k-n_b) \end{bmatrix}^T. \tag{4}$$

It is apparent that, if the orders n_a and n_b are known, the identification of a PWARX model amounts to a PWA regression problem.

3.1 Clustering-based algorithms

The basic tasks that most procedures for hybrid identification perform are (not necessarily in this order):

a. The estimation of the *switching sequence* $\lambda(x(k))$ and the construction of the *mode data sets* $\mathcal{F}_i = \{(x(k), y(k)) : \lambda(x(k)) = i\}$;

b. The estimation of the PVs θ_i, $i = 1, \dots, s$;

c. The reconstruction of the regions $\mathcal{X}_i$.

[1]Each set $\mathcal{X}_i$ is a (not necessarily closed) convex polyhedron s.t. $\mathcal{X}_i \bigcap \mathcal{X}_j = \emptyset$, $\forall i \neq j$, $\bigcup_{i=1}^s \mathcal{X}_i = \mathcal{X}$.

In particular, step (a) amounts to *classify the data*, i.e. assign each data point to the mode that most likely generated it.

The driving idea of clustering-based procedures is that PWA maps are locally linear. Then, if the local models around two data points are similar, it is likely that the data points belong to the same mode. More in details, clustering-based methods are structured in the following steps:

1. Associate to each data point a local affine model;

2. Aggregate local models with similar features into clusters;

3. Classify in the same way data points corresponding to local models in the same cluster;

4. Estimate the PVs and the regions.

Step 1. For $j = 1, \dots, N$, we build a Local Dataset (LD) $\mathcal{C}_j$ collecting $(x(j), y(j))$ and its $c-1$ neighboring data points. Examples of LDs are reported in Fig. 1(a). The cardinality c of an LD is a parameter of the algorithm, and it is assumed that $c \geq n+1$. For each LD $\mathcal{C}_j$, compute a vector ξ_j representing the features of a local affine model built using only the data in $\mathcal{C}_j$. Possible choices are

- ξ_j is the Local Parameter Vector (LPV) obtained by fitting an affine model on the LD $\mathcal{C}_j$ through least squares [3]

- ξ_j is a Feature Vector (FV) collecting the LPV associated to $\mathcal{C}_j$ and a measure of the spatial localization of the local model [4];

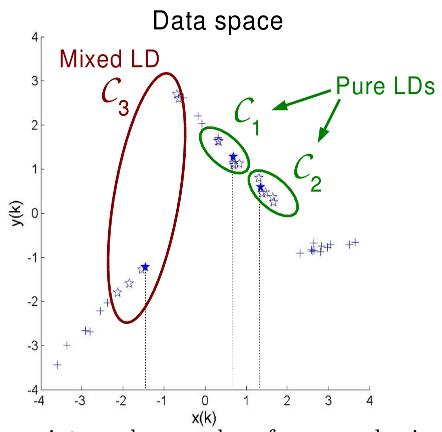
As discussed in [4, 8], FVs are crucial for reconstructing modes having virtual intersections or characterized by the the same PVs but associated to distinct regions. In both cases, we associate to ξ_j also a *matrix and a scalar* related to quality of the local model. LPVs for the example in Fig. 1(a) are represented in Fig. 1(b).

We refer to $\mathcal{C}_j$ as a *pure LD* if it collects only data points associated to a single mode (and we say that $\mathcal{C}_j$ is *associated* to this mode). Otherwise the LD is termed *mixed*, see Fig. 1(a).

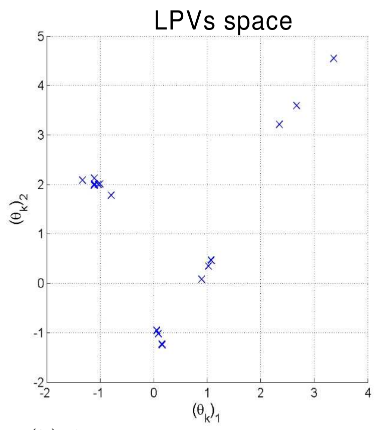
Intuitively, a pure ξ-point (i.e. associated to pure LDs) carries information about one of the modes composing the system while mixed ξ-point provide spurious information about the true modes. Clustering-based algorithms are expected to be effective if the ratio between pure and mixed ξ-points is sufficiently high and if pure ξ-points provide an accurate enough representation of the PVs. Since the bigger c, the bigger number of mixed ξ-points, one would like to keep c as low as possible. On the other hand, one would like to increase c in order to counteract the effect of noise on the accuracy of local model. A good tuning of c always results from this trade-off. Fro more details, we defer the reader to [4].

$\square$

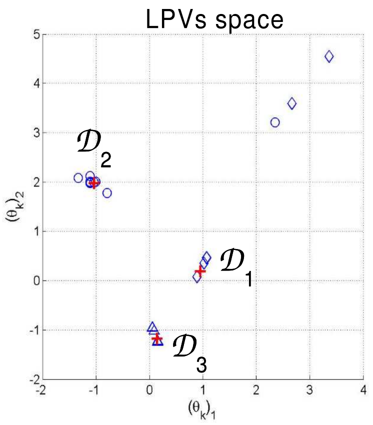
Step 2. The goal is to find s dense clusters $\{\mathcal{D}_i\}_{i=1}^{s}$ together with their centers $\{\mu_i\}_{i=1}^{s}$ that partition the set of all ξ-points. This is usually done via *clustering algorithms* exploiting either the matrix- or scalar-valued quality measures computed in step 1. Two alternatives are possible: supervised clustering methods (like Kmeans), that need s as input, or unsupervised clustering methods (like single-linkage) that do not require the knowledge of s but need other parameters



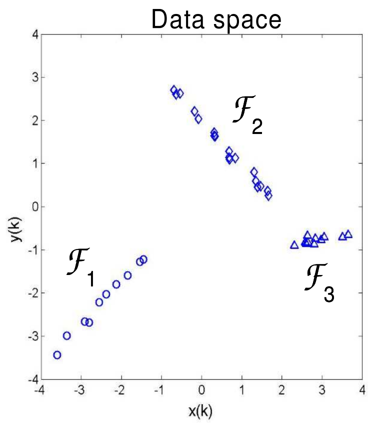
(a) Data points and examples of pure and mixed LDs. Vertical lines mark the data point associated to each LDs.



(b) ξ-points representing LPVs



(c) Clusters of ξ-points.



(d) Mode data sets.

Figure 1: Data-based reconstruction of a PWA map with three modes.

that influence the number of clusters [6]. An example of clusters is represented in Fig. 1(c). Some clustering procedures perform the detection of *outliers* (that are likely to be mixed ξ-points) that will be not attributed to clusters. For this reason, in the sequel we will call *inliers* ξ-points that are attributed to clusters. As it will be clear in step 4, just clusters that contain a sufficiently high number of ξ-points can be used for estimating the modes. Then, if a cluster contains few points, it is discarded and its ξ-points are marked as outliers. This operation also reduce the number of modes s.

□

Step 3. Since each cluster is expected to collect all local models with similar features, data points are classified according to the rule $\lambda(x(j)) = i \Leftrightarrow \xi_j \in \mathcal{D}_i$, for all inliers ξ_j. Mode data sets are built accordingly. Note that data point corresponding to outliers are not classified and hence ignored in the next steps. The mode dataset for the example in Fig. 1(a) are plot in Fig. 1(d).

□

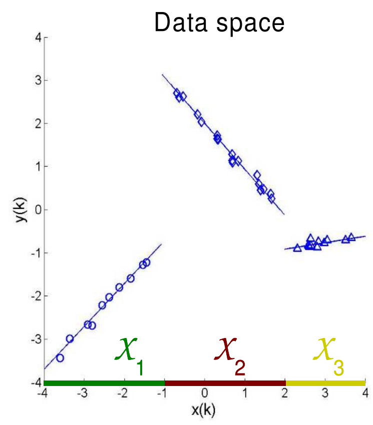


Figure 2: Identified PWA map from the data in Fig. 1(a).

Step 4. Conceptually, this is the easiest step. The data points in each set $\mathcal{F}_i$ can be used for estimating the PVs of each submodel through weighted least squares exploiting the scalar quality measures computed in step 1. It is apparent that $\mathcal{F}_i$ must contain at least $\min\{\alpha(n+1), c\}$ data points in order to estimate a PV, where $\alpha \in \mathbb{N}$, $\alpha \geq 1$ is a *discarding factor* that must be supplied by the user. Indeed, a violation of this bound means either that there are too few data points, compared to the number of scalar parameters composing a PV, or that no pure LD is associated to the mode. In a noiseless setting, $\alpha = 1$ would be enough for estimating PVs, but when noise is present, one wants to use more than a single data point for each scalar parameter, and hence $\alpha > 1$ is more appropriate

Since the cardinalities of $\mathcal{F}_i$ and $\mathcal{D}_i$ do coincide, only clusters with more than $\min\{\alpha(n+1), c\}$ ξ-points are retained in step 2.

Also the regions $\{\mathcal{X}_i\}_{i=1}^{s}$ can be found on the basis of the mode data sets by resorting to *pattern recognition* algorithms. At this stage, it suffices to recall the basic problem solved by these algorithms. Since all the sets $\mathcal{X}_i$ are polyhedral, for each pair $(\mathcal{X}_i,\ \mathcal{X}_j)$, with $i \neq j$, a separating hyperplane exists, described by the equation $M'_{ij}x = m_{ij}$ and leading to $M'_{ij}x \leq m_{ij}$, if $x \in \mathcal{X}_i$, and to $M'_{ij}x > m_{ij}$, if $x \in \mathcal{X}_j$. Let $\bar{\mathcal{F}}_i = \{x(k) : \lambda(x(k)) = i\}$. The vector M_{ij} and the constant m_{ij} can be estimated by finding an hyperplane that separates the regressors in $\bar{\mathcal{F}}_i$ from those in $\bar{\mathcal{F}}_j$. In the case of *perfect classification* (i.e. the switching sequence is reconstructed without errors) and when data are generated by (3), it is guaranteed that the sets $\bar{\mathcal{F}}_i$ and $\bar{\mathcal{F}}_j$, $i \neq j$ are *linearly separable* i.e. separable by an hyperplane. In presence of classification errors, pattern recognition algorithms usually look for the separating hyperplanes that minimizes some performance measure related to the number of misclassified points. The final results obtained from the data in Fig. 1(a) are represented in Fig. 2.

□

The previous steps can be complemented by an optional post-processing articulated into two sub-steps. The first one is to detect, among inliers, ξ-points that are suspected to be mixed. Inspired by the theory developed in [5], a point $\xi_j \in \mathcal{F}_i$ is suspected to be mixed if $\mathcal{C}_j$ is not contained in $\mathcal{F}_i$ (i.e. the local dataset does not belong completely to the reconstructed mode data set). The second sub-step is to re-classify suspected points and outliers, that is to attribute them to the most likely mode of operation. After re-attribution, mode data sets with less than $\min\{\alpha(n+1), c\}$ points are discarded, s is updated accordingly and step 4 is performed again.

4 Learning HIT through examples

This section is based on some of the examples provided with HIT and illustrate the use of HIT for typical identification experiments. The most important variables influencing the behavior of HIT are also described.

4.1 Simple PWA regression

In this experiment we will reconstruct a PWA with 3 modes and a 2D regressor set. We first initialize the HIT and MPT toolboxes. This operation is necessary for setting the global variables `idpar, plotpar` controlling the behavior of HIT and the global variable `mptOptions` used by MPT. All these variables are structure arrays. Initializations are done by typing:

```
  hit_init
```

In order to make the global variables visible in the workspace, type

```
global idpar plotpar
```

Next, we define the PWA map used for generating the data. To this purpose:

- we define the PVs and collect them in a cell array:

  ```
  th_1 =[4 2 3];
  th_2 = [-6  6 -5];
  th_3=[4 -2 -2];
  Theta={th_1,th_2,th_3};
  ```

- we set the regressor set $\mathcal{X}$ as the square $[-1,1] \times [-1,1]$:

  ```
  R=[1 0;-1 0;0 1; 0 -1];
  r=[1;1;1;1];
  Regressor_set=polytope(R,r);
  ```

  Note the use of the command `polytope` of MPT for defining a *polytope object.* Try:

  ```
  help polytope
  ```

  for more details.

- define a cake-like partition of $\mathcal{X}$ into 3 regions, stored as elements of a *polytope array*:

```
    Regions=[polytope([0 -1;-1 -1/sqrt(3);R],[0;0;r]),...
            polytope([1 1/sqrt(3);1 -1/sqrt(3);R],[0; 0;r]),...
            polytope([-1 1/sqrt(3);0 1;R],[0;0;r])];
```

Now, we generate 60 noisy samples of the PWA map:

```
Nid=60;
% Sample the regressor set
Xid=hit_sample_rectangles({{[-1,1],[-1,1],Nid}},Regressor_set);
% Create the corresponding (noisy) output samples
yid=[];
for k=1:Nid
  point=[Xid(k,1) Xid(k,2)];
  [val,ind]=hit_pwa(Theta,Regions,point);
  yid(k)=val+0.01*randn(1); % add noise to the output
end
yid=yid(:);
```

The previous lines use the HIT functions `hit_sample_rectangles` and `hit_pwa` for generating randomly regressors in a rectangular domain and for evaluating the PWA map on the regressors, respectively. Regressors are stored as *rows* of the matrix `Xid` and output samples as elements of the *column vector* `yid`.

Since the PWA map has a 2D domain, it can be visualized. For plotting the map and the data points try the following lines:

```
% x_grid, y_grid: grids for plotting the pwa function
x_grid=-1:.05:1;
y_grid=-1:.05:1;
% plot the mode hyperplanes and the data in figure 2
minz=hit_pwa_plot2d(Theta,Regions,[],x_grid,y_grid,2,plotpar.color_surface);
hold on
for i=1:Nid
  hp(i) = plot3(Xid(i,1),Xid(i,2),yid(i),'ob');
  hold on
  set(hp(i), 'MarkerSize', 8,'LineWidth',1.5);
end
axis square
grid on
ylabel('{x_2}','FontSize',16)
xlabel('{x_1}','FontSize',16)
zlabel('{y}','FontSize',16)
set(gca,'FontSize',13)
% plot the mode regions
hit_plot_regions3d(Regions,minz,plotpar.color_regions);
% save the axis for later purposes
axis_saved=axis;
title('True PWA model');
hold off
```

In the previous lines, the function `hit_pwa_plot2d` plots the map on a grid. The color of the map is stored in `plotpar.color_surface` that has been initialized in `hit_init`. In general, the fields of `plotpar` (PLOTting PARameters) define all variables influencing the plots in HIT.

Note also that plotted regions are numbered in order to allow the user to easily recognize which is the the first, second and third mode (as declared in `Theta` and `Regions`).

The variable `minz` stores the minimum of the gridded map and is used in `hit_plot_regions3d` for plotting the regions below the map. The colors of the regions are stored in the matrix `plotpar.color_regions` that has been initialized in `hit_init`.

Now, we reconstruct the PWA map from the data. All parameters influencing the PWA regression algorithm are stored in fields of the structure `idpar` (IDentification PARameters) that has been initialized in `hit_init`. Comments in `hit_init.m` describe all fields of `idpar`. The only fields that must be supplied by the user are the size of local datasets c, the regressor set and the number of modes s (because, by default, HIT uses Kmeans as a clustering algorithm and Kmeans is a supervised method).

```
% number of modes
idpar.s=3;
% size of Local Datasets
idpar.c=6;
% regressor_set
idpar.Regressor_set=Regressor_set;
```

Since few data points are available, we can choose MRLP (Multicategory Robust Linear Programming) [1] as a pattern recognition algorithm. MRLP is the most precise algorithm for reconstructing the regions among those in HIT, but also the most computationally expensive.

```
idpar.patt_rec_algo='mrlp';
```

The reconstruction of the PWA map is done by invoking `hit_regression`

```
[idmodes,F,xi,LDs,inliers]=hit_regression(Xid,yid);
```

During its running, `hit_regression` displays various messages describing the operations that are performed at each step. Usually, error messages provide also hints about how to circumvent the problem. `hit_regression` plots also the first 2 or 3 coordinates of the clustered ξ-points, by default.

Let us have a look at the outputs of `hit_regression`. The structure array `idmodes` (IDentified MODES) describes the reconstructed PWA map and stores also measures related to the correctness of clustering and pattern recognition. The parameters describing the PWA map can be displayed by typing:

```
% Display the number of modes found after identification. It can be different
% from idpar.s
idmodes.s
% Display the PVs stored in the cell array idmodes.par
idmodes.par{:}
% Display the covariances of PVs, stored in the cell array idmodes.cov
idmodes.cov{:}
% Display the regions stored in the polytope array idmodes.regions
idmodes.regions(1); idmodes.regions(2); idmodes.regions(3)
```

Note that the identified modes *might be a permutation* of the original ones. As an example, idmodes.par{1} it is not necessarily an estimate of Theta{1} but might be an estimate of Theta{2} or Theta{3}. This is due to the fact that the regression algorithm has no knowledge about how modes have been ordered in the model used for generating the data.

The remaining outputs describe the mode data sets (structure F) the ξ-points (matrix xi) the LDs (structure LDs) and the inliers (vector inliers). For a precise description of all these variables, just type help hit_regression.

For plotting the identified model, type

```
% plot the model in figure 3
hit_plot_idmodes(Xid, yid,inliers,idmodes,3);
hold on
% put the same axis as in the plot of the original model
axis(axis_saved)
title('Identified PWA model');
hold off
```

Since regions appearing in the plot are numbered, a comparison with the plot of the map generating the data help reconstruct the modes permutation.

We conclude the example by commenting some facilities provided by HIT for validating the identified model.

We first check the the global Mean Squared Error (MSE) and the MSE of each mode. These quantities, that helps in isolating modes that have been badly identified, can be computed by typing:

```
[mse,mse_mode]=hit_mse(Xid,yid,idmodes);
```

The structure array idmodes.pattern_rec_valid contains measures of the quality of pattern recognition. Some of its entries depend on the pattern recognition algorithm used. However, all methods produce the field idmodes.pattern_rec_valid.correctness that is a matrix whose ij-entry, $i > j$, measures the correctness in separating regressors of the i-th mode from regressors of the j-th mode. More precisely, idmodes.pattern_rec_valid.correctness(i,j) is the percentage of the points in $\bar{\mathcal{F}}_i \cup \bar{\mathcal{F}}_j$ correctly separated. A correctness of 100 means perfect separation.

Similarly, idmodes.pattern_rec_valid stores algorithm-dependent measures of the clustering quality. In our experiment we used Kmeans, that initializes randomly the cluster centers. For this reason, HIT runs Kmeans a number of times, specified by

```
idpar.clustalgo.kmeans.repetitions
```

and keep only the best clustering result, i.e. the results in the run that gave the minimal clustering cost. The field idmodes.clust_valid.costs stores the cost at each run. It is wise to check if Kmeans converged to the minimal cost in many different runs, otherwise, either clusters are not well-separated in the ξ-space (in this case it is recommended to increase idpar.c) or Kmeans started from a bad initialization in all runs. In the latter case, a remedy is to increase the number of runs.

4.2 Identification of a PWARX model

We consider the problem of identifying a PWARX model with scalar inputs/outputs, orders $n_a = n_b = 1$ and 3 modes. As in the previous section, we first initialize HIT and MPT, define PVs and regions of the PWARX model and collect the data.

```
% initialize the HIT and MPT toolboxes
hit_init
% define idpar and plotpar as global to make them visible
% in the workspace
global idpar plotpar

%
% Definition of the PWARX model
%

% Specify the PVs. Note that two of them are equal
th_1 =[0.6 0.3 0];
th_2 = [-.6 -0.3 0];
th_3=[0.6 0.3 0];
% store the PVs in the cell array Theta
Theta={th_1,th_2,th_3};
% Define the regressor set equal to the square [-2,2]*[-2 2]
Regressor_set=polytope([1 0;-1 0;0 1; 0 -1],[2;2;2;2]);
% Regions: polytope array specifying the true mode regions
Regions=[polytope([0.2 1],[-.8]),...
         polytope([-0.2 1; -0.2 -1],[.8; .8]), ...
         polytope([0.2 -1],[-.8])];
% intersect Regions with the regressor set
for i=1:3
    Regions(i)=Regions(i)&Regressor_set;
end

%
% Generate input/output samples
%

% N: number of data
N=60;
% standard deviation and variance of the noise corrupting
% the output samples
sigma_sq=0.01;
sigma=sqrt(sigma_sq);
% create N samples of the scalar input u(k) unifomly distributed in [-2,2]
u=hit_sample_intervals({{[-2,2],N}}, polytope([1;-1],[2; 2]));
% initialize the vector of output samples
% using y(1)=0.5 as initial state
y=[0.5 zeros(1,N)];
% simulate the PWARX system to obtain the output samples
for k=1:N
    point=[y(k) u(k)]';
    [val,ind]=hit_pwa(Theta,Regions,point);
    y(k+1)=val+sigma*randn(1); % add noise to the data
end
% Build regressors and outputs for the PWARX model of orders n_a=1, n_b=1
% Xid(i,:) is the i-th regressors
% yid(i) is the i-th output sample
[Xid,yid]=hit_pwarx_format_data(u,y,1,1);
```

A few comments on the new commands appearing in the previous lines of code:

- The polyhedra stored in `Regions` at the beginning of the code are unbounded. To make them bounded, they are intersected with the regressor set through the command

  ```
  Regions(i)=Regions(i)&Regressor_set
  ```

  that uses the `&` operator of MPT;

- The command `hit_sample_intervals` randomly samples the interval [-2 ,2];

- `hit_pwarx_format_data` takes inputs, outputs and orders of the PWARX model and create the regressors and outputs as in (4).

Now, we plot modes and data in figure 2:

```
minz=hit_pwa_plot2d(Theta,Regions,[],x_grid,y_grid,2,plotpar.color_surface);
hold on
for i=1:Nid
    hp(i) = plot3(Xid(i,1),Xid(i,2),yid(i),'ob');
    hold on
    set(hp(i), 'MarkerSize', 8,'LineWidth',1.5);
end
axis square
grid on
hold on
ylabel('{u(k-1)}','FontSize',16)
xlabel('{y(k-1)}','FontSize',16)
zlabel('{y(k)}','FontSize',16)
set(gca,'FontSize',13)
% plot the mode regions
hit_plot_regions3d(Regions,minz,plotpar.color_regions);
% save the axis
axis_saved=axis;
title('True PWA model');
hold off
```

The identification is done exactly as in the example of Section 4.1, the only difference being that we use SVC (Support Vector Classification) [10] for reconstructing the regions. SVC is faster but less accurate than MRLP. Most importantly, if the regressor set has dimension strictly greater than one, it is not guaranteed that the union of regions will cover the regressor set, i.e. some subregions of $\mathcal{X}$ not associated to any mode might arise (see [4] for more details).

```
%
% Setup of the fields of idpar.
%

% number of modes
```

```
idpar.s=3;
% size of Local Datasets
idpar.c=6;
% regressor_set
idpar.Regressor_set=Regressor_set;
% pattern_recognition algorithm
idpar.patt_rec_algo='svc';

%
% Identify the PWARX model
%

[idmodes,F,xi,LDs,inliers]=hit_regression(Xid,yid);

%
% Plot the identified PWARX model
%

% Setup of the fields of plotpar
plotpar.x_grid=x_grid;
plotpar.x_grid=y_grid;

% plot the model in figure 3
hit_plot_idmodes(Xid, yid,inliers,idmodes,3);
hold on
% use the same axis as in the plot of the original model
axis(axis_saved)
title('Identified PWARX model');
ylabel('{u(k−1)}','FontSize',16)
xlabel('{y(k−1)}','FontSize',16)
zlabel('{y(k)}','FontSize',16)
set(gca,'FontSize',13)
hold off
```

As for the validation of the identified model, one can follow the guidelines provided in Section 4.1. In addition, one can check if SVC has left some "holes" by typing

```
  H=hit_holes(idmodes,idpar.Regressor_set)
```

where `H` is a polytope array partitioning the hole. An empty polytope means that no hole is present.

4.3 Identification with a large number of data points

The next example shows how to use HIT for reconstructing a PWARX model (with scalar inputs/outputs, orders $n_a = 0$, $n_b = 1$ and 5 modes) from 1000 data points. In particular, we will show how to speed up the identification procedure. Again, we initialize HIT and MPT, define PVs and regions of the PWARX model, generate the data and plot the results.

```
hit_init
global idpar plotpar

%
```

```
% Generate input and output samples
%

% N: number of datapoints
N=1000;
% specify the PVs. Note that two modes have a continuous junction.
th_1 =[1 .2];
th_2 = [-1 2] ;
th_3 = [1 -1];
th_4 = [0 2];
th_5 = [2 -10];
Theta={th_1,th_2,th_3,th_4,th_5};
% Define the regressor set equal to the interval [-4,8]
Regressor_set=polytope([1 ;-1 ],[8;4]);
% Regions: polyhedra array specifying the true regions.
% They are the intervals [-inf,-1], [-1,2], [2,4], [4,6], [6,inf]
Regions=[polytope([1],[-1]),polytope([1;-1],[2; 1]);...
         polytope([-1;1],[-2;4]);polytope([-1;1],[-4;6]);...
         polytope([-1],[-6])];

% standard deviation and variance of the noise corrupting
% the output samples
sigma_sq=0.01;
sigma=sqrt(sigma_sq);
% create N samples of the scalar input u(k) unifomly distributed in Regressor_set
V=extreme(Regressor_set);
u=hit_sample_intervals({{V,N}}, Regressor_set);
% initialize the vector of output samples  with zeros.
% This means that, for computing y(2), the 'initial state' y(1)=0 is used
y=zeros(1,N+1);
% simulate the PWA system for obtaining the output samples
for k=1:N
    point=[u(k)];
    y(k+1)=hit_pwa(Theta,Regions,point)+sigma*randn(1); % add noise to the data
end
% Build the vectors of datapoints that will be used for identification
na=0;
nb=1;
[Xid,yid]=hit_pwarx_format_data(u,y,na,nb);

% plot the true model and the data in figure 1
hit_pwa_plot1d(Theta,Regions,Regressor_set,1,'u(k-1)','y(k)');
hold on
plot(Xid(:,1),yid,'+','MarkerSize',8);
% save the axis
axis_saved=axis;
title('True model and data points')
hold off
```

Note that we used the command `extreme` of MPT for finding the starting and ending point of the regressor set.

For the reconstruction of the regions, MRLP/SVC would be too slow unless one has access to professional LP/QP solvers like CPLEX (we highlight that MPT provides an easy-to-use interface to some commercial solvers). Indeed, MRLP/SVC are based on LP/QP problems whose number of variables scales linearly with the number of data points. Therefore we resort to the fastest pattern recognition algorithm provided with HIT: Proximal SVC (PSVC) [7]. PSVC is not optimization-based, but it might be also less accurate than MRLP or SVC. Moreover, as for SVC, if the regressor set has dimension strictly greater than one, it is not guaranteed that the union of regions will cover the regressor set.

Be aware that the execution of the next lines of code might take a couple of minutes, depending on the speed of your machine.

```
% number of modes
idpar.s=5;
% size of Local Datasetsidpar.c=8;
% use PSVC as pattern_recognition algorithm
idpar.patt_rec_algo='psvc';
% Define the regressor set
idpar.Regressor_set=Regressor_set;
% Define the regressor set *for simulation* equal to the interval [-10,10]
idpar.Regressor_set_sim=polytope([1 ;-1 ],[10;10]);

%
% Identify the PWARX model
%

[idmodes,F,xi,LDs,inliers]=hit_regression(Xid,yid);

%
% Plot the identified PWARX model
%

% Setup of the fields of plotpar
plotpar.x_grid=x_grid;
plotpar.x_grid=y_grid;

% plot the model in figure 3
hit_plot_idmodes(Xid, yid,inliers,idmodes,3);
hold on
% use the same axis as in the plot of the original model
axis(axis_saved)
title('Identified PWARX model');
ylabel('{u(k-1)}','FontSize',16)
xlabel('{y(k-1)}','FontSize',16)
zlabel('{y(k)}','FontSize',16)
set(gca,'FontSize',13)
hold off
```

In the previous lines, we used a *regression set for simulation* specified by the optional field `idpar.Regressor_set_sim`. This option is relevant for the following reason: if the identified model must be used (e.g. simulated) on a polytope $\bar{\mathcal{X}}$ that is bigger than $\mathcal{X}$, it is convenient to let `hit_regression` intersect the hyperplanes separating the regions with $\bar{\mathcal{X}}$. In this case, regions for simulation $\{\bar{\mathcal{X}}_i\}_{i=1}^{s}$ are computed and stored in the field `idmodes.regions_sim`.

This procedure has a drawback: if the dimension of regressors is strictly greater than one, the regions for simulation might not cover $\bar{\mathcal{X}}$, even if they cover $\mathcal{X}$. In other words, some holes may appear due to the fact that $\mathcal{X} \subset \bar{\mathcal{X}}$. However, the presence of holes can be checked with the command `hit_holes` as shown in Section 4.2.

4.4 PWA approximation of a nonlinear function

We discuss now how to approximate a sinusoidal function with a PWA map. We initialize HIT and MPT, sample the function and plot the data.

```
% initialize the HIT and MPT toolboxes
hit_init
% define idpar and plotpar as global to make them visible
% in the workspace
global idpar plotpar

%
% Sample the function

% N: number of generated datapoints
N=60 ;
% create a grid in the function domain
Xid=[-pi:2*pi/N:pi];
% function samples
yid=sin(Xid);
% Regressor_set: interval [-pi,pi]
idpar.Regressor_set=polytope([1 ;-1 ],[pi;pi]);
% plot the data in figure 1
figure(1);clf;
hold on
plot(Xid,yid,'+','MarkerSize',8);
hold on
baseline=axis;
title('Function samples')
hold off
```

For computing the approximation, we identify a PWA map with 5 modes by using the samples. Assume that we want to obtain a *continuous* PWA approximation. This can be achieved by setting `idpar.continuity='c'`.

```
% Number of modes of the PWA approximator
idpar.s=5;
% Size of Local Datasets
idpar.c=8;
% pattern_recognition algorithm
idpar.patt_rec_algo='svc';
% we want a continuous PWA approximation
idpar.continuity='c';

%
% PWA regression
%

% put Xid in column format
Xid=Xid(:);
[idmodes,F,xi,LDs,inliers]=hit_regression(Xid,yid);

%
% If the regressor set is 1- or 2-dimensional
% the PWA model can be plotted
%

% avoid plotting datapoints
plotpar.datapoints_yn='N';
hit_plot_idmodes(Xid, yid,inliers,idmodes,2);

% plot the original function on the same figure
figure(2);
hold on
plot(Xid,yid,'r');
title('True function and PWA approximation');
hold off
```

In the previous lines, we prevented `hit_plot_idmodes` from plotting the data by setting

```
plotpar.datapoints_yn='N'
```

4.5 Selection of c and s through cross-validation

When one is uncertain about the proper values of the parameters c and s, a possibility is to tune them with cross-validation. We first generate identification and validation data points by sampling a PWA map with 5 modes:

```
hit_init

% define idpar and plotpar as global to make them visible
% in the workspace
global idpar plotpar

%
% Generate regression and validation datapoints
%

% N: number of datapoints
N=150 ;
% Nv: number of validation datapoints
Nv=N/2 ;

% specify the  PVs:
th_1 =[1 .2];
th_2 = [-1 2] ;
th_3 = [1 .2];
th_4 = [3 -10];
th_5 = [-2 15];

% store the  PVs in the cell array Theta
Theta={th_1,th_2,th_3,th_4,th_5};

% Regions: polyhedra array specifying the true regions.
% They are the intervals [-inf,-1], [-1,2], [2,4], [4,6], [6,inf]
Regions=[polytope([1],[-1]),polytope([1;-1],[2; 1]);...
         polytope([-1;1],[-2;4]);polytope([-1;1],[-4;6]);...
         polytope([-1],[-6])];

% Define the regressor set equal to the interval [-4,8]
idpar.Regressor_set=polytope([1 ;-1 ],[8;4]);

% standard deviation and variance of the noise corrupting
% the output samples
sigma_sq=0.1;
sigma=sqrt(sigma_sq);

% create input samples u(k) unifomly distributed in Regressor_set
V=extreme(idpar.Regressor_set);
u=hit_sample_intervals({{V,N}}, idpar.Regressor_set);

% create input samples for cross-validation
uv=hit_sample_intervals({{V,Nv}}, idpar.Regressor_set);

% create placeholders for output samples
y=zeros(1,N);
yv=zeros(1,Nv);

% sample the PWA map to obtain output samples
for k=1:N
    point=[u(k)];
    y(k)=hit_pwa(Theta,Regions,point)+sigma*randn(1);
end

% sample the PWA map to obtain output samples for cross-validation
for k=1:Nv
    point=[uv(k)];
    yv1(k)=hit_pwa(Theta,Regions,point)+sigma*randn(1);
end

% Regressors and outputs for identification
Xid=u(:);
yid=y(:);

% Regressors and outputs for cross-validation
Xv=uv(:);
yv=yv1(:);

% plot the true model and the data in figure 1
hit_pwa_plot1d(Theta,Regions,idpar.Regressor_set,1,'x(k)','y(k)');
hold on
plot(Xid(:,1),yid,'+','MarkerSize',8);
hold on
baseline=axis;
title('True model and datapoints')
hold off

% plot the true model and the validation data in figure 2
hit_pwa_plot1d(Theta,Regions,idpar.Regressor_set,2,'x(k)','y(k)');
hold on
plot(Xv(:,1),yv,'+','MarkerSize',8);
hold on
baseline=axis;
```

```
title('True model and validation datapoints')
hold off
```

Next, we store the candidate values of c and s in two vectors and call `hit_estimate_cs` that identify a model for each combination of c and s and select the values minimizing the MSE on validation data. In order to speed up the procedure, we prevent HIT from plotting the classified ξ-points by setting

```
plotpar.plot_class_xi_points_yn='N';
```

and choose the fastest pattern recognition algorithm (PSVC):

```
c_test=[6 8];
s_test=[4 5 6];

idpar.patt_rec_algo='psvc';
plotpar.plot_class_xi_points_yn='N';
[best_s,best_c,mse_m]=hit_estimate_cs(Xid,yid,Xv,yv,c_test,s_test);
```

We conclude by re-identifying the model with the best c and s. To achieve better results we use SVC for reconstructing the regions.

```
% PWA regression with the best Local Dataset size and the best number of
% modes
idpar.c=best_c;
idpar.s=best_s;
% Plot classified xi-points during the running of hit_regression
plotpar.plot_class_xi_points_yn='y';
% Choose SVC as pattern_recognition algorithm
idpar.patt_rec_algo='svc';
[idmodes,F,xi,LDs,inliers]=hit_regression(Xid,yid);

% Plot the identified model in figure 3
hit_plot_idmodes(Xid, yid,inliers,idmodes,3);
xlabel('x(k)')
ylabel('y(k)')
```

5 Global variables used in HIT

As seen in the previous examples, the behavior of `hit_regression` depends on the field structures `idpar` and `plotpar`. In this Section, we discuss just the fields of `idpar`. The fields of `plotpar` are self-explanatory and can be easily learned from the comments in `hit_init.m`. For field values that are strings, we highlight that HIT is *not* case-sensitive.

The following variables

```
idpar.s
idpar.c
idpar.Regressor_set
idpar.Regressor_set_sim
```

have been already discussed in Section 4. Here, we just mention the fact that if `idpar.Regressor_set` is not supplied by the user, it is automatically set as the smallest hyper-rectangle containing all regressors.

The user selects if a continuous or discontinuous PWA/PWARX model must be reconstructed through the variable

```
idpar.continuity= 'C' | 'D'
```

We stress that clustering-based procedures are tailored to *discontinuous* models and continuity is just imposed during the final estimation of the PVs.

The post-processing phase discussed in Section 3 is activated according to the value of the field

```
 idpar.mix_detect= 'Y' | 'N'
```

The factor $\alpha \geq 1$, used for discarding clusters, is stored in the field

```
  idpar.discard_threshold_factor
```

In a noisy setting, it is wise to set α equal to (at least) 2 or 3. Indeed, as a rule of thumb, if data are noisy, one needs at least 2 or 3 data points for each parameter to be estimated.

HIT can offer an option to stop right after the clustering step. This is useful when many clusters have been found (through Single-linkage clustering, for instance) and MRLP has been selected for finding the regions. In this case, it might take a *very* long time to get the MRLP solution and one may want to skip the pattern-recognition phase. HIT offers this option according to the value of the field

```
idpar.YNquestions='Y' | 'N'
```

5.1 Clustering algorithms

Two clustering algorithms are included in HIT: weighted Kmeans and single-linkage. The desired algorithm is chosen according to the field

```
idpar.clustalgo.name= 'KMEANS' | 'SL'
```

Moreover, each algorithm clusters ξ-points that are either LPVs or FVs. To choose the nature of ξ-points, one has to set the field

```
idpar.what_to_clust= 'LPVS' |  'FVS'
```

Finally, one should note that two or more LDs can contain the *same* data points thus providing the same ξ-point. Duplicates of ξ-points are removed in clustering according to the field

```
idpar.clustalgo.remove_duplicates='Y' | 'N'
```

This is useful in single-linkage for discarding clusters containing ξ-points that are associated to less than $\min\{\alpha(n+1), c\}$ *distinct* data points.

5.1.1 Kmeans

The number of times Kmeans is run with different initializations, is stored in

```
  idpar.clustalgo.kmeans.repetitions
```

For initializing cluster centers and update them during the execution of Kmeans, one can use either the matrix-valued or the scalar-valued measures of quality associated to ξ-points. In general, matrix-valued measures produce better results at the price of making Kmeans slower. The quality measure to be used is set through the variables

```
idpar.clustalgo.kmeans.init_centers= 'COVARIANCES' | 'SCALARS'
idpar.clustalgo.centers= 'COVARIANCES' | 'SCALARS'
```

5.1.2 Single-linkage

Single-linkage clustering does not need `idpar.s` to be specified and estimates automatically the number of clusters. However, the user must supply the expected minimal distance between clusters [2] and this can be done by setting the field

```
idpar.clustalgo.sl.guess_min_dist
```

Single-linkage is a hierarchical method that aggregates clusters in an iterative way. HIT can plot how clusters change at each step of the process, according to the field

```
  idpar.clustalgo.sl.plot_steps= 'Y' | 'N'
```

More precisely, HIT plots the first 2 or 3 components of the clustered ξ-points. The number of the figure where the plot appears is stored into the variable

```
idpar.clustalgo.sl.plot_steps.fig
```

5.2 Pattern-recognition algorithms

As discussed in Section 4, HIT offers three pattern-recognition algorithms for reconstructing the regions: MRLP, SVC and PSVC. We summarize here the pros and cons of each one:

- MRLP is the most precise algorithm but also the slowest one. This is due to the fact that MRLP tries to find the boundaries between *all regions at the same time* by solving a single LP. This guarantees that the regions are a partition of the regressor set, i.e. no hole will be left in $\mathcal{X}$. As a rule of thumb, if the number of data points is bigger than 200 and the number of modes is bigger than 3, we recommend to use it only if a fast LP solver (like CPLEX) can be used.

- SVC is less precise than MRLP, in the sense that the reconstructed regions might leave holes in $\mathcal{X}$ when the dimension of $\mathcal{X}$ is strictly greater than one [4]. This issue is due to the fact that SVC finds recursively linear boundaries between pairs of regions by solving QPs.

However, pairwise separation results in QPs with a reduced number of unknown, compared to the number of unknowns in the LP produced by MRLP. As a result, usually SVC is computationally more efficient than MRLP.

- PSVC does not involve any optimization and is extremely efficient from the computational point of view. However, it is less precise than SVC and might leave holes in $\mathcal{X}$ when the dimension of $\mathcal{X}$ is strictly greater than one.

The pattern recognition algorithm is selected by setting

```
idpar.patt_rec_algo: 'MRLP' | 'SVC' | 'PSVC'
```

5.3 Solvers for LP and QP

HIT uses functions of the MPT toolbox for solving LPs and QPs. Generally, one does not need to change the default settings because MPT checks automatically the solvers installed on a machine and uses the most convenient one. The remainder of this Section will explain how force MPT to use a specific solver.

The LP solver used by MRLP is selected through the variable:

```
  idpar.LPsolver=0 | 1 | 2 | 3,..
```

where the meaning of the integers can be visualized by typing `help mpt_solvelp`. Analogously, the LP solver used for reducing constraints defining polytopes (see the manual of MPT for an explanation of this operation), is specified through the field

```
  idpar.LPsolver_cnstr_reduction=0 | 1 | 2 | 3,..
```

HIT invokes QP solvers for solving SVC problems and for finding continuous PWA/PWARX models. The QP solver can be chosen by setting

```
idpar.QPsolver=0 | 1 | 2
```

where the meaning of the integers can be visualized by typing `help mpt_solveqp`

5.4 Experimental features

For estimating LPVs one may want to use weighted Least Squares. This can be done by specifying the weights in the vector

```
idpar.Weight_primal
```

If `Weight_primal` is empty, it is filled with ones, and the weights play no role.

Acknowledgments

The development of HIT has been supported by the EU projects HYCON (FP6-IST-511368) and HYGEIA (NEST-4995).

References

[1] E.J. Bredensteiner and K.P. Bennett. Multicategory classification by support vector machines. *Computational Optimizations and Applications*, 12(1-3):53–79, Jan 1999.

[2] G. Ferrari-Trecate and M. Muselli. Single-linkage clustering for optimal classification in piecewise affine regression. In S. Engell, H. Gueguen, and J. Zaytoon, editors, *IFAC Conference on the Analysis and Design of Hybrid Systems (ADHS 03)*. Saint-Malo, France, 16-18 june 2003.

[3] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. Identification of piecewise affine and hybrid systems. In *Proc. of the 2001 American Control Conference*, volume 5, pages 3521–3526. Arlington, VA, IEEE, Piscataway, NJ, USA, 25-27 june 2001.

[4] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. A clustering technique for the identification of piecewise affine and hybrid systems. *Automatica*, 39(2):205–217, Feb 2003.

[5] G. Ferrari-Trecate and M. Schinkel. Conditions of optimal classification for piecewise affine regression. In *Proc. 6th Int. Workshop on Hybrid Systems: Computation and Control*, volume 2623, pages 188–202. Prague, Czech, Springer-Verlag, Berlin Heidelberg 2003, 3-5 April 2003.

[6] B. Fritzke. Some competitive learning methods. Technical report, Institute for Neural Computation. Ruhr-Universit at Bochum, 1997.

[7] G. Fung and O. L. Mangasarian. Proximal support vector machine classifiers. In F. Provost and R. Srikant, editors, *Proceedings KDD-2001: Knowledge Discovery and Data Mining, August 26-29, 2001, San Francisco, CA*, pages 77–86, New York, 2001. Asscociation for Computing Machinery. `ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-02.ps`.

[8] A. Juloski, M. Heemels, G. Ferrari-Trecate, R. Vidal, S. Paoletti, and H. Niessen. Comparison of four procedures for identification of hybrid systems. In M. Morari and L. Thiele, editors, *Proc. 8th Int. Workshop on Hybrid Systems: Computation and Control*, volume 3414, pages 354–369. Zurich, Switzerland, Springer-Verlag, Berlin Heidelberg 2005, 9 - 11 March 2005.

[9] M. Kvasnica, P. Grieder, and M. Baotić. Multi-Parametric Toolbox (MPT), 2004.

[10] V. Vapnik. *Statistical Learning Theory.* John Wiley, NY, 1998.