Introduction

What is CPS?

Cyber Physical Systems, which embed software into the physical world (for example, in smart grids, robotics, intelligent manufacture and medical monitoring), have proved resistant to modeling due to the intrinsic complexity arising from (a) the combination of physical components and cyber counterparts and (b) the interaction between systems.

What is IHYDE?

Identification of hybrid dynamical systems (IHYDE) is a open-source Matlab toolbox for automating the mechanistic modeling of hybrid dynamical systems from observed data and without prior knowledge. IHYDE has much lower computational complexity comparing with genetic algorithms, enabling its application to real-world CPS problems. IHYDE implements the clustering-based algorithms described in the Data-driven Discovery of Cyber Physical Systems. It can also be used, positively, for the creation of guidelines for designing new CPSs.

Licensing

IHYDE is free for use in both academic and commercial settings.

Installation

IHYDE uses routines of the CVX toolbox for constructing and solving disciplined convex programs (DCPs).The latest version of CVX toolbox can be download at this website .For installing it , you can follow these steps.

Download the latest version of IHYDE toolbox from the website . Decompress the .zip file in a directory and add its path (and the path of the subdirectories) to the Matlab path.

The IHYDE toolbox consists of the following directories.

Directories	Description
/IHYDE	main functions and examples
/IHYDE/docs	pdfand html documentation pdf and html documentation
/IHYDE/data	datasets we used in the paper
/IHYDE/tools	functions for IHYDE

Theroy of IHYDE

Hybrid dynamical systems

To give a formal definition for hybrid dynamical systems, we adopt the notation from (20). Each hybridautomataH,isdefinedasatuple,H =(W,M,F,T)withthefollowingdefinitions: To determine the actual form of subsystems from data directly, we collect time-course input-output data (y(t),u(t)) uniformly sampled at a number of discrete time indices 1,2,...,M. Let

- W defines a continuous space for input-output variables u, y;
- M definesacountable,discretesetofmodesinwhichonlyasinglemode,m∊{1,2,...,K}, is occupied at a given time;
- F defines a countable discrete set of first-order difference equations :

$$y(t+1) = f_k(y(t), u(t)) = I_k(y(t)) + h_k(u(t));$$

- T defines a countable discrete set of transitions, where Ti→i' denotes a Boolean expression that represents the condition to transfer from mode k to k'.

Under this definition, we can construct a mathematical model for hybrid dynamical systems.

$$m(t+1) = \mathcal{T}(m(t), u(t))$$

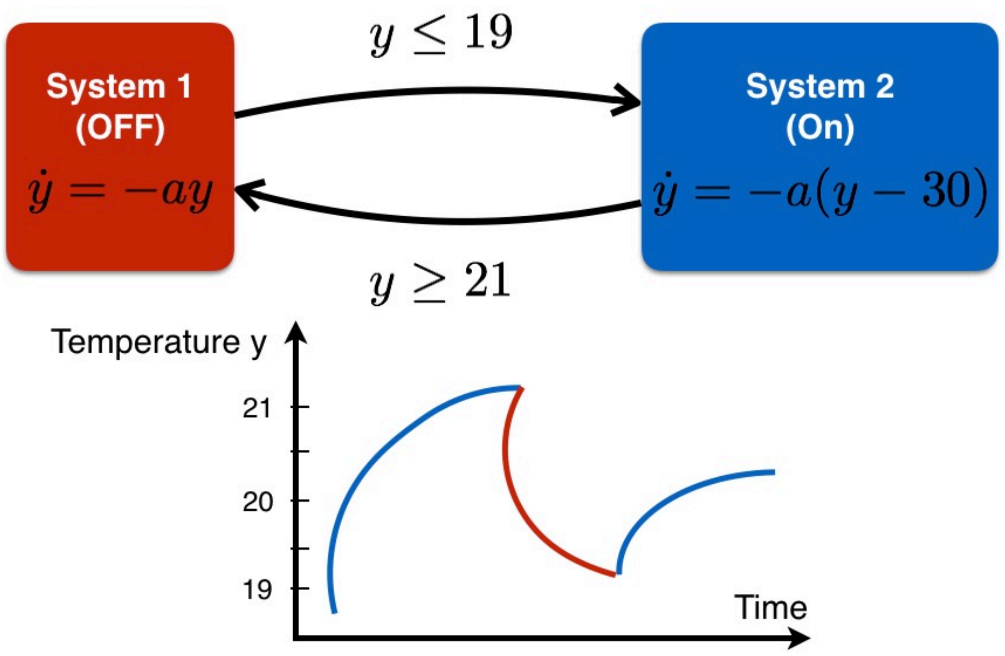
$$y(t+1) = \mathbf{f}(m(t), y(t), u(t)) = \begin{cases} \mathbf{f}_1(y(t), u(t)), & \text{if } m(t) = 1 \\ \vdots, & \vdots \\ \mathbf{f}_K(y(t), u(t)), & \text{if } m(t) = K \end{cases}$$

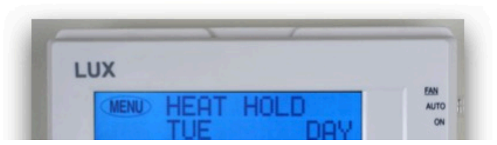
For example, a temperature control system consisting of a heater and a thermostat.Variables that are included in a model of such a system are the room temperature andthe operating mode of the heater (on or off). The goal of this study is to infer both thesubsystems and transition logic from time-series data. The two subsystems have the following form

$$\begin{aligned} &\text{System 1}: \ \dot{y} = -ay \\ &\text{System 2}: \ \dot{y} = -a(y-30). \end{aligned}$$

The transition logics are

$$\begin{aligned} &\mathcal{T}_{1\to 2}: \ y \leq 19 \\ &\mathcal{T}_{2\to 1}: \ y \geq 21. \end{aligned}$$





The IHYDE Algorithm

Inferring subsystems

When a hybrid system has a single mode, i.e., $K = 1$, it becomes a time-invariant nonlinear dynamical system. We shall briefly review the identification for nonlinear systems in Pan and Brunton , Pan2012cdc; this would be beneficial for the reader to understand the basic idea for later IHYDE we propose. To determine the form of the function $\mathbf{f}$ from data, we collect time-course input-output data $(y(t), u(t))$ sampled at a number of instances in time $\delta t, 2\delta t, \dots, M\delta t$. These data are then arranged into the following matrix: (to avoid notational complexity, we dropout M in the subscript of Y and let $\delta t = 1$ without loss of generality)

$$Y = [\, y(1) \quad y(2) \quad \dots \quad y(M) \,]^T, \; U = [\, u(1) \quad u(2) \quad \dots \quad u(M) \,]^T.$$

Next, we construct an augmented library $\Phi(Y, U)$ consisting of candidate nonlinear functions of the columns of Y and U. For example, $\Phi(Y, U)$ may consist of constant, polynomial and trigonometric terms:

$$\Phi(Y, U) = [\ \mathbf{1} \quad Y \quad Y^{P2} \quad \cdots \quad U \quad U^{P2} \quad \cdots \ ].$$

Here, higher polynomials are denoted as Y^{P_2}, Y^{P_3}, etc. For example, Y^{P_2} denotes the quadratic nonlinearities in the state variable Y, given by:

$$Y^{P_2} = \begin{bmatrix} y_1^2(1) & y_1(1)y_2(1) & \cdots & y_n^2(1) \\ y_1^2(2) & y_1(2)y_2(2) & \cdots & y_n^2(2) \\ \vdots & \vdots & \ddots & \vdots \\ y_1^2(M) & y_1(M)y_2(M) & \cdots & y_n^2(M) \end{bmatrix}.$$

Each column of $\Phi(Y, U)$ represents a candidate function for $\mathbf{f}$. There is very considerable freedom of choice in constructing the entries in this dictionary matrix. Since only a few of these nonlinearities are active in each row of $\mathbf{f}$, we may set up a sparse regression problem to determine the sparse vectors of coefficients $W = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}$ that determine which nonlinearities are active . This results in the overall model

$$\bar{Y} \triangleq \begin{bmatrix} y_1(2) & \cdots & y_1(M+1) \\ y_2(2) & \cdots & y_2(M+1) \\ \vdots & \ddots & \vdots \\ y_n(2) & \cdots & y_n(M+1) \end{bmatrix} = \Phi(Y, U)W.$$

When $K > 1$, we can use the identical formulation above however, the challenge is that there no single W that can fit all the data due to hybridity of the dynamical system. Next, we introduce a new method to tackle such a challenge. Define $Z = \bar{Y} - \Phi W - \xi$, where ξ is zero-mean i.i.d. Gaussian measurement noise with covariance matrix $\sigma^2 I$. The goal is to find a Z^* as sparse as possible, i.e., $Z^* = \arg\min_Z \|Z\|_0$. Solution to this optimization, the index of the zero entries of Z^* corresponds to the index for input-output that is from one subsystem. We assume the dictionary matrix Φ is full rank without loss of generality, then we can define the transformation matrix Θ such that $\Theta\Phi = \mathbf{0}$. It furthermore follows that $\Theta\bar{Y} = \Theta Z + \Theta\xi$. We then transform the optimization problem as follows using Lagrange Multiplier:

$$\min_Z \frac{1}{2}(\tilde{\bar{Y}} - \Theta Z)^\top \Sigma^{-1}(\tilde{\bar{Y}} - \Theta Z) + \tilde{\lambda}\|Z\|_{\ell_0},$$

where $\tilde{\bar{Y}} \triangleq \Theta\bar{Y}$. However, this problem is known computationally expensive, we may use the following convex relaxation

$$\min_Z \frac{1}{2}(\tilde{\bar{Y}} - \Theta Z)^\top \Sigma^{-1}(\tilde{\bar{Y}} - \Theta Z) + \tilde{\lambda}\|Z\|_{\ell_1}.$$

Once solved, we take the index set $\mathbf{seq} = \{k| \, |Z^*[i]| \le \epsilon_z\}$ and further identify sparse coefficients W^* using the following optimization

$$W^* = \arg\min_W \frac{1}{2}\|\bar{Y}[\text{seq}] - \Phi[\text{seq}, :]W\|_2^2 + \lambda_w\|W\|_{\ell_1}.$$

W^* is the coefficient of the identified subsystem. Define $\text{error} = abs(\bar{Y} - \Phi W^*)$. Then we set the ith element of $\bar{Y}$: $\bar{Y}[i] = 0$ and the ith row of Θ: $\Theta[i, :] = 0$ for which the ith element of $error$ is less than ϵ_w. We can solve the same problem with the rest of is using the exact procedure. The number of iterations this problem is solved gives the minimum number of subsystems.

A variant of Inferring alogorithm

One of the issues in inferring subsystems, according to the experiments presented in the next Results section, is sensitive to noise and redundant dictionary matrix. The main reason is that the proposed algorithm has a sequence of optimizations, and the error in the solution to earlier optimization propagates to later ones, leading to large final errors. We hereby propose a new variant algorithm, our implementation is available from the website . The key idea is to select the longest continuous sequence as the subsystem identified by the current loop. Using the identified model, we then group other data points that fits well by the model.

Distributed version

When solving $Z^k = \arg\min_Z \frac{1}{2}(\tilde{\bar{Y}} - \Theta Z)^\top \Sigma^{-1} (\tilde{\bar{Y}} - \Theta Z) + \tilde{\lambda}\|Z\|_{\ell_1}$, it is desirable to

decompose the large optimization to a number of smaller problems. By decomposition, we can reduce the computation complexity in the overall problem.

Equivalently, we can formulate a number of sub-problems as follows: for $i = 1, \ldots, n$

$$Z^k[:, i] = \arg\min_z \frac{1}{2}(\tilde{\bar{Y}}[:, i] - \Theta z)^\top \Sigma^{-1} (\tilde{\bar{Y}}[:, i] - \Theta z) + \tilde{\lambda}\|z\|_{\ell_1} .$$

Once we solve all the sub-problems, we then obtain the solution to the original problem by merging solutions, i.e.,

$$Z^k = \left[\, Z^k[:, 1] \quad Z^k[:, 2] \quad \ldots \quad Z^k[:, n] \,\right] .$$

Application to the identification of nonhybrid dynamical systems

When there is only one subsystem, we can show that Z^k should be a all 0 matrix from the first optimization in $Z^k = \arg\min_Z \frac{1}{2}(\tilde{\bar{Y}} - \Theta Z)^\top \Sigma^{-1} (\tilde{\bar{Y}} - \Theta Z) + \lambda_z \|Z\|_{\ell_1}$.

Then $W^k = \arg\min_W \frac{1}{2}\|\bar{Y}[\text{seq}] - \Phi[\text{seq}, :]W\|_{\ell_2}^2 + \lambda_w \|W\|_{\ell_1}$ should be the same as $W^* = \arg\min \|\bar{Y} - \Phi W\|_{\ell_2}^2 + \lambda \|W\|_{\ell_1}$ since $\text{seq} = \{1, 2, \ldots, M\}$, which recovers the results for time-invariant nonlinear system identification in Pan and Brunton As a result, we provide a unified point of view to subsystems identification problem for any $K \in \{1, 2, \ldots\}$.

Inferring transition logic

Once the subsystems have been identified, we can assign every input-output data point $(u(t), y(t))$ to a specific subsystem. The next step is to identify the transition logic between different subsystems. We shall first convert the problem of identifying transition logic to a standard sparse logistic regression which can then be efficiently solved by many methods in literature.

To proceed, we define $\gamma_k(t)$ as the set membership which equals to 1 only if the subsystem k is active at discrete-time t, otherwise 0. The goal is to identify $\mathcal{T}_{k\to k'}$ between any subsystems k, k'. Mathematically, we are searching for a nonlinear function g acting on $u(t), y(t)$, such that $\mathbf{step}(g)$ (in here $\mathbf{step}(x)$ equals to 1 if $x \geq 0$, and 0 otherwise) specifies the membership. Due to non-differentiability of step functions at 0, we alternatively relax the step function to a sigmoid function, i.e., $\gamma_{k'}(t+1) = \frac{1}{1+e^{-g(y(t),u(t))}}$.

Instead of using symbolic regression in Ly2012jmlr , we can parameterize $g(y(t), u(t))$ as by a linear combination of over-determined dictionary matrix, i.e., $g(y(t), u(t)) \triangleq \Psi(y(t), u(t))v$, in which Ψ can be constructed similarly as Φ in the previous section. Thus the fitness function has the following form which shall be minimized for any k, k'

Example

To quickly familiarize with IHYDE, examples are in the directory /IHYDE/examples. These .m files can also be used as templates for other experiments. We shall use the autonomous car example to explain the code.

In the running process, the embedded camera captures the road information ahead to checkwhether the forward road is a straightaway or a curve. Depending the decision, it choosescorresponding speed control strategy. The car measures current speed by encoder and calculatesu˙ that the motor needs to change. The speed control strategy is based on increasing PI controlalgorithm and equation is shown below.

$$\dot{u}(k) = P(v_{expect} - v(k)) + I(v(k-1) - v(k))$$

where $u(k)$ represents motor's output and $v(k)$ is the velocity at time k. When the car is running on a straight line, we expect it a higher speed so we set a high expected speed vexpect. In order to prevent the car from running out of the track, so we give it a lower expectation speed while it is running on the curve. We use IHYDE to successfully reverse engineering the control strategy of the CPS.

Here is the code. First, we load the data.

```
clear all,close all,clc
addpath('./tools');
addpath('./data');
basis_function.work='off';
data=load('normal_car.mat');  %%  load Data
index = 1000:1400;
flag = data.flag(index);   % 1:straghtway   0:curve
dy = data.dy(index);
v = data.v(index)/10;
```

Use the data to built a library by function library.

```
memory = 4; %The data covers the time from k-4 to k
polyorder = 4; % The highest order of the polynomial is 4 order
usesine  = 0;%no sin and cos
A= library(v,polyorder,usesine,memory,basis_function);%make a library
A = A(memory+2:end,:);
dy = dy(memory+2:end); %dpwm_{k}
flag = flag(memory+2:end); %flag_{k}

v_k1 = v(memory+1:end-1,:);  % v_{k-1}
v_k2 = v(memory:end-2,:);% v_{k-2}
v_k3 = v(memory-1:end-3,:);% v_{k-3}
v = v(memory+2:end,:);  % v_{k}
```

Then, we initialize the parameters and identify the systems by function

```
parameter.lambda = [0.01 1e-5];   %  the fist element in lambda is lambda_z and
the second
%is lambda_w
parameter.MAXITER = 5;   %the iter for  the sparsesolver algorithm
parameter.max_s = 20;           % the max number of subsystems
parameter.epsilon = [100  8]; %the fist element in lambda is epsilon_z and the
second is
%epsilon_w
parameter.Phi = A;    % the library
parameter.y = dy;  %dpwm
parameter.normalize_y = 1;  % normalize:1     unnormalize:0
[result]=ihyde(parameter);     % inferring  subsystems
```

Function **ihyde** will return a preliminary identified result which contains the details of subsystems. Since we want to get a better result based on the minimum error principle, we use function **finetuning** to fine-tune the results.

```
result.epsilon = parameter.epsilon(2);% use epsilon_w as the epsilon in
finetuning
result.lambda = parameter.lambda(2);% use lambda_w as the epsilon in finetuning
result.threshold = [0.05];    %set a threshold for clustering
final_result  = finetuning(result);   % finetuning each  subsystems
sys = final_result.sys;  % get the identified subsystems
idx_sys = final_result.idx;% get the index of each subsystems
```

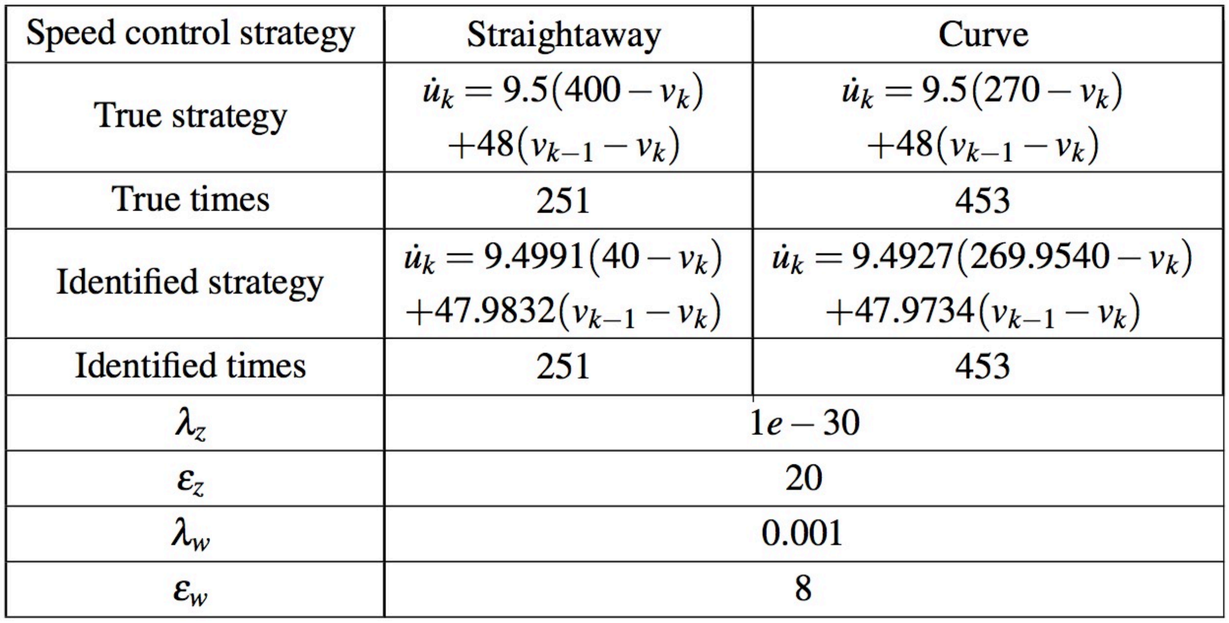
For inferring transition logic between subsystems, the code is shown above.

```
Phi2 = [ones(size(flag)) flag 1./v sin(v) cos(v) v.^2  v_k1./v_k2 v_k3.^2 ];
% library for inferring transition logic between subsystems.

para_log.idx_sys = idx_sys;
para_log.beta= 0.5;   % the tradeoff of l1-sparse logistic regression
para_log.y = dy;
para_log.Phi2 = Phi2;

[syslogic,labelMat,data] = ihydelogic(para_log);
```

The identified results are saved in sys , idx_sys and syslogic. Here are the result and related parameters.



Speed control strategy	Straightaway	Curve
True strategy	$\dot{u}_k = 9.5(400 - v_k)$ $+48(v_{k-1} - v_k)$	$\dot{u}_k = 9.5(270 - v_k)$ $+48(v_{k-1} - v_k)$
True times	251	453
Identified strategy	$\dot{u}_k = 9.4991(40 - v_k)$ $+47.9832(v_{k-1} - v_k)$	$\dot{u}_k = 9.4927(269.9540 - v_k)$ $+47.9734(v_{k-1} - v_k)$
Identified times	251	453
λ_z	$1e-30$	
ε_z	20	
λ_w	0.001	
ε_w	8	

API

library

```
Function [yout ]= library(yin,polyorder,usesine,memory,basis_function)
```

Input

- [ ] yin : a m*n matrix which contains time-course input-output data. m means m times, and n is the number of data's type.
- [ ] polyorder : used to construct the polynomial of the highest order (up to fifth order).
- [ ] basicfunction : add more basis functions. It can be turned off, if basis_function.work set as 'off'.

Output

- [ ] Phi : constructed dictionary matrix Φ.

ihyde

```
Function [result]=ihyde(parameter)
```

Input

- [ ] parameter.y : the ouput data.
- [ ] parameter.normalize_y : set to 1 if y need to be normalized.
- [ ] parameter.max_s: the max number of subsystems that could be identified by IHYDE.
- [ ] parameter.epsilon : a 1 by 2 vector. For finding new subsystems.
- [ ] parameter.lamdba :a 1 by 2 vector. For the identification of a new subsystems.
- [ ] parameter.MAXITER : the max number of iterations that the sparsesolver function solves.

Output

- [ ] result. idx{sys}: the index of each subsystem.
- [ ] result.sys : the model of each subsystem.
- [ ] result.theta : z of each identified subsystem.
- [ ] result.error:the fitting error.

fineturning

```
Function [ final_result ] = finetuning( result)
```

Input

- [ ] result.lambda: the trading-off parameter λ of the **sparsesolver** function. The parameter.lamdba(2) is set as the default value.
- [ ] result.epsilon: the threshold in finetuning. The parameter.epsilon(2) is set as the default value.
- [ ] result.threshold: the threshold for subsystem clustering.

Output

- [ ] final_result.idx : the index of each subsystem.
- [ ] final_result.sys : the model of each subsystem.

fineturning

```
Function [syslogic,labelMat,data,num] = ihydelogic(para_log)
```

Input

- para_log.Phi2: constructed dictionary matrix Ψ for inferring transition logic of each subsystem.
- para_log.idx_sys: the index of each subsystem.
- para_log.beta: the tradeoff parameter in the ℓ_1 sparse logistic regression.

Output

- syslogic: transition logic of each subsystem.