MATLAB Software for Nonlinear and Delayed Recursive Identification– Revision 2

Torbjörn Wigren

Division of Systems and Control, Department of Information Technology, Uppsala University, SE-75105 Uppsala, SWEDEN. E-mail: torbjorn.wigren@it.uu.se

April 2020

Abstract

This report is the user's manual for a package of MATLAB™ scripts and functions, developed for recursive prediction error identification of nonlinear state space systems. The identified state space model incorporates delay, which allows a treatment of general nonlinear networked identification, as well as of general nonlinear systems with delay. The core of the package is an implementation of two output error identification algorithms. The algorithms are based on a continuous time, structured black box state space model of a nonlinear system. The present revision adds a new algorithm, where also the output is determined via a parameterized measurement equation in the states and inputs. The software can only be run off-line, i.e. no true real time operation is possible. The algorithms are however implemented so that true on-line operation can be obtained by extraction of the main algorithmic loop. The user must then provide the real time environment. The software package contains scripts and functions that allow the user to either input live measurements or to generate test data by simulation. The scripts and functions for the setup and execution of the identification algorithms are somewhat more general than what is described in the references. The functionality for display of results include scripts for plotting of e.g. data, parameters, prediction errors, eigenvalues and the condition number of the Hessian. The estimated model obtained at the end of a run can be simulated and the model output plotted, alone or together with the data used for identification. Model validation is supported by two methods apart from the display functionality. First, a calculation of the RPEM loss function can be performed, using parameters obtained at the end of an identification run. Secondly, the accuracy as a function of the output signal amplitude can be assessed.

Keywords: Delay, Identification, MATLAB™, Networked identification, Nonlinear systems, Ordinary differential equation, Output Equation, RPEM, Recursive algorithm, Sampling, Software, State space model.

Prerequisites

This report only describes the parts of [1], [2], [3], [4] and [5] that are required for the description of the software. Hence the user is assumed to have a working knowledge of the algorithms of these publications and of MATLAB™, see e.g. [7]. This, in turn, requires that the user has a working knowledge of system identification and in particular of recursive identification methods as described in e.g. [8].

Revisions

Revision 1: This is the first revision of the software. The SW is originally derived from the software package [6] that implements the algorithm of [1] – [4], together with some other variants. The present revision builds directly on the first revision of this software package [10].

Installation

The file SW.zip is copied to the selected directory and unzipped. MATLAB is opened and a path is set up to the selected directory using the path browser. The software is then ready for use.
Note: This report is written with respect to the software, as included in the SW.zip file. It may therefore be advantageous to store the originally supplied software for reference purposes.

Error reports

When errors are found, these may be reported in an e-mail to:
torbjorn.wigren@it.uu.se.

1. Introduction

Identification of nonlinear systems is an active field of research today. There are several reasons for this. First, many practical systems show strong nonlinear effects. This is e.g. true for high angle of attack flight dynamics, many chemical reactions and electromechanical machinery of many kinds, see e.g. [1], [2], [3], [4], [5] and the references therein for further examples. Another important reason is perhaps that linear methods for system identification are quite well understood today, hence it is natural to move the focus to more challenging problems.

Today, the wireless internet is being expanded to new low delay standards, commonly denoted the fifth generation (5G) wireless systems. There, a round trip delay close to 1 ms will be offered to users, a fact that is expected to lead to a fast expansion into new networked control use cases like the tactile internet and wireless industrial control. However, in many situations non-constant network delays will remain significant, which requires new nonlinear recursive system identification algorithms that are capable of identifying the delay together with the nonlinear dynamics, as in [5]. The present software implements the algorithm proposed in [5], which is also applicable to delayed systems which are not necessarily networked controlled.

This report thus focuses on software that implements new nonlinear recursive system identification methods, that also incorporates identification of delay, integer as well as fractional [5]. These black box method estimate continuous time parameters in a general state space model, both with known and unknown nonlinear measurement equations. The identification methods belong to the class of recursive prediction error methods (RPEMs) and the methods are of output error type. Advantages include the fact that the stability of the estimated model is checked by a projection algorithm, at each iteration step.

The nonlinear identification algorithms are based on a continuous time black box state space model, with delay modeled by the output equation. This model is structured in that only one right hand side component of the ordinary differential equation (ODE) model is parameterized as an unknown function. As shown in [1] and [2] this avoids over-parametrization. The restriction imposed on the model structure may seem restrictive. However, it is motivated in [1] and [2] that the selected structure can always (locally in the states) model systems with more general right hand sides, a fact that extends the applicability of the methods significantly. The selected parameterization of the right hand side function of the ODE is a linear-in-the-parameters multi-variate polynomial in the states and input signals. The covariance matrix of the measurement disturbances is estimated on-line.

In case the nonlinear measurement equation needs to be identified as well, a model with a similar parameterization is used. However, note that the static differential gain of either the ODE model or the measurement equation needs to be fixed, in order to avoid over-parameterization. Therefore, the output equation is restricted to have a constant gain in the middle of the region of support, with the outer parts of the region of support being modeled by multi-variate polynomial in the states and inputs. These models contain a specific factor to enforce continuity of the overall output equation. The algorithm with nonlinear measurement equation identification re-uses most of the code of the previous SW package, however a number of scripts have been modified and are marked with the inclusion of “Output” in the name of the m-files.

Recursive system identification is a software dependent technology. Hence, when publishing new methodology in this field, it is relevant to also provide useful software for application of the presented

algorithms. This facilitates a quick practical exploitation of new ideas. The development of the present MATLAB™ software package is motivated by this fact.

The present software package is developed and tested using MATLAB 6.5 and MATLAB 8.2. The software package does not rely on any MATLAB toolboxes. It consists of a number of scripts and functions. Briefly, the software package consists of scripts for setup, scripts for generation or measurement of data, scripts for execution of the RPEMs and scripts for generation and plotting of results. There is presently no GUI, the scripts must be run from the command window. Furthermore, input parameters need to be configured in one or several of the setup scripts, as well as when running the scripts. In case of data generation by simulation, the ODE that defines the data generating system must be specified in standard MATLAB style. The software can only be run off-line, i.e. there is no support for execution in a real time environment. The major parts of the algorithmic loop can however easily be extracted for such purposes.

The report is organized according to the flow of tasks a user encounters when applying the scripts of the package. Before the software is described, some basic facts about the ODE model are reviewed.

2. Model

The nonlinear MIMO model to be defined here is used for estimation of an unknown parameter vector $\begin{pmatrix}\theta_T & \boldsymbol{\theta}_S^T\end{pmatrix}^T$ from measured inputs $\mathbf{u}(t)$ and outputs $\mathbf{y}_m(t)$, given by

$$\begin{aligned}\mathbf{u}(t) &= \begin{pmatrix}u_1(t) & \dots & u_1^{(n_1)}(t) & \dots & u_k(t) & \dots & u_k^{(n_k)}(t)\end{pmatrix}^T \\ \mathbf{y}_m(t) &= \begin{pmatrix}y_{m,1}(t) & \dots & y_{m,p}(t)\end{pmatrix}^T\end{aligned} \qquad (1)$$

The superscript $^{(k)}$ denotes differentiation k times. The continuous time model is given by the following n:th order state space ODE

$$\begin{aligned}\begin{pmatrix}x_1^{(1)} \\ \vdots \\ x_{n-1}^{(1)} \\ x_n^{(1)}\end{pmatrix} &= \begin{pmatrix}x_2 \\ \vdots \\ x_n \\ f\left(x_1,\dots,x_n,u_1,\dots,u_1^{(n_1)},\dots,u_k,\dots,u_k^{(n_k)},\boldsymbol{\theta}_S\right)\end{pmatrix} \\ \begin{pmatrix}y_1 \\ \vdots \\ y_p\end{pmatrix} &= \begin{pmatrix}c_{11} & \dots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{p1} & \dots & c_{pn}\end{pmatrix}\begin{pmatrix}x_1(t-\theta_T) \\ \vdots \\ x_n(t-\theta_T)\end{pmatrix},\end{aligned} \qquad (2)$$

where $\mathbf{x} = \begin{pmatrix}x_1 & \dots & x_{n-1} & x_n\end{pmatrix}^T$ is the state vector. The following polynomial parameterization of the right hand side function of (2) is used

$$f\left(x_1,\dots,x_n,u_1,\dots,u_1^{(n_1)},\dots,u_k,\dots,u_k^{(n_k)},\boldsymbol{\theta}_S\right)$$

$$= \sum_{i_{x_1}=0}^{I_{x_1}} \cdots \sum_{i_{x_n}=0}^{I_{x_n}} \sum_{i_{u_1}=0}^{I_{u_1}} \cdots \sum_{i_{u_1^{(n_1)}}=0}^{I_{u_1^{(n_1)}}} \cdots \sum_{i_{u_k}=0}^{I_{u_k}} \cdots \sum_{i_{u_k^{(n_k)}}=0}^{I_{u_k^{(n_k)}}} \theta_{S,i_{x_1}\ldots i_{x_n} i_{u_1}\ldots i_{u_1^{(n_1)}}\ldots i_{u_k}\ldots i_{u_k^{(n_k)}}} (x_1)^{i_{x_1}} \ldots (x_n)^{i_{x_n}} (u_1)^{i_{u_1}} \ldots$$

$$\ldots \left(u_1^{(n_1)}\right)^{i_{u_1^{(n_1)}}} \ldots (u_k)^{i_{u_k}} \ldots \left(u_k^{(n_k)}\right)^{i_{u_k^{(n_k)}}} = \varphi^T(\mathbf{x},\mathbf{u})\boldsymbol{\theta}_S . \qquad (3)$$

Here

$$\boldsymbol{\theta}_S = \Big(\theta_{S,0\ldots0} \quad \ldots \quad \theta_{S,0\ldots I_{u_k^{(n_k)}}} \quad \theta_{S,0\ldots010} \quad \ldots \quad \theta_{S,0\ldots01I_{u_k^{(n_k)}}} \quad \ldots$$

$$\theta_{S,0\ldots0I_{u_k^{(n_k-1)}}0} \quad \ldots \quad \theta_{S,0\ldots0I_{u_k^{(n_k-1)}}I_{u_k^{(n_k)}}} \quad \ldots \quad \theta_{S,I_{X_1}\ldots I_{u_k^{(n_k)}}} \Big)^T$$

$$\varphi = \Big(1 \quad \ldots \quad \left(\left(u_k^{(n_k)}\right)^{I_{u_k^{(n_k)}}}\right) \quad u_k^{(n_k-1)} \quad \ldots \left(u_k^{(n_k-1)} \left(u_k^{(n_k)}\right)^{I_{u_k^{(n_k)}}} \right) \ldots \quad \left(u_k^{(n_k-1)}\right)^{I_{u_k^{(n_k-1)}}} \quad \ldots$$

$$\left(\left(u_k^{(n_k-1)}\right)^{I_{u_k^{(n_k-1)}}} \left(u_k^{(n_k)}\right)^{I_{u_k^{(n_k)}}}\right) \quad \ldots \left(\left(x_1\right)^{I_{x_1}} \ldots \left(x_n\right)^{I_{x_n}} \left(u_1\right)^{I_{u_1}} \ldots \left(u_k^{(n_k)}\right)^{I_{u_k^{(n_k)}}}\right)\Big)^T . \qquad (4)$$

In order to obtain a discrete time model, the Euler integration method is applied to (2). In addition, a multiple model approach together with interpolation is used for discretization of the delay θ_T. This results in a capability to estimate large integer delays together with a fractional part, which results in a very high modeling precision. The details are quite complicated and discussed at length in [5].

3. Software package overview

The software package is command driven, i.e. no GUI is available. It consists of a number of MATLAB scripts and functions. These are described in the next subsection. They are all marked with a string Delay at the end, a fact which allow them to be stored together with the files of [6] without any ambiguity.

3.1 Scripts, functions and command flow

Roughly, the scripts and functions can be divided into five groups:

- *Live data measurements*. The two scripts of this group set up and perform clocked live data measurements. The scripts are **SetupLive.m** and **MeasurementMain.m**.
- *Simulated data generation*. The four scripts of this group define a dynamic system that is then used for generation of simulated data. The scripts and functions are **SetupDataDelay.m**, **f.m**, **h.m** and **GenerateDataDelay.m**.
- *Recursive identification*. The five scripts of this group perform the actual identification tasks, supporting user interaction. The scripts and functions are **SetupRPEMDelay.m**, **RPEMDelay.m**, **h_m.m** and **dhdx_m.m**.

- *Supporting functions - not called by the user*. The four functions of this group are called by scripts of the previous group. They are all related to the implementation of the RHS model of the identified ODE. The scripts are **GenerateIndices.m**, **f_m.m**, **dfdx_m**.**m** and **dfdtheta_m**.**m**.
- *Preparation and display of results*. There are eleven scripts in this group. They all prepare, compute and display results of the identification process. The scripts are **PlotDataDelay.m, SimulateModelOutputDelay.m**, **PlotParametersDelay.m**, **PlotPredictionErrorsDelay.m**, **PlotEigenvaluesDelay.m**, **PlotConditionDelay.m**, **ComputeRPEMLossFunctionDelay.m**, **PlotModelOutputDelay.m**, **PlotSystemAndModelOutputDelay.m**, **PlotResidualErrorsDelay.m** and **MeanResidualAnalysisDelay.m**.

These groups of scripts and functions need to be operated in a particular order to make sense. This order of execution between scripts and functions is similar to the one of Figure 1 of [6].

There are three major ways to exploit the five groups of scripts and functions.

1. In case the user has input and output signals available, the first step is to define and run the script **SetupLive.m.** This sets basic parameters like the sampling period. The user can then proceed directly to use the groups *Recursive Identification* and *Preparation and display of results.* The data, which can be simulated or live, should be stored in the (row) matrices **u** and **y** .
2. In case the user is to perform live measurements, all the steps of the *Live data measurement* group should be executed first. The user can then proceed directly to use the groups *Recursive Identification* and *Preparation and display of results.*
3. In case the user intends to use simulated data, this data can be generated by execution of the scripts and functions of the group *Simulated data generation.* The user can then proceed directly to use the groups *Recursive Identification* and *Preparation and display of results.*

4. Data input

The generation of data begins the section where the actual software is described. Since the user has access to all source files, the descriptions below do not describe code related issues and internal variables. Only the parts that are required for the use of the software package are covered. When m-files are reproduced, only the relevant parts are included, the reader should be aware that more information can be found in the source code. Note that the setup files are to be treated as templates, the user is hence required to modify right hand sides only - no addition or deletion of code should be used in the normal use of the package.

4.1 Simulated data

The generation of simulated data requires that the user

1. Modifies the underlying ODE model, as given by **f.m** and **h.m**. The function **f.m** implements the RHS of the ODE, using a conventional MATLAB function call. Note that the built in ODE solvers

of MATLAB are not used. Instead an Euler algorithm is implemented. The reason is that this allows the generation of simulated data that can be exactly described by the applied model, should this be desired. The function **h.m** implements the (possibly nonlinear) measurement equation. The functions allow for addition of systems noise and measurement noise.

2. Provides further input data in the script **SetupDataDelay.m**. The parameters that define the data generation are directly written into this script. These parameters define the sampling period, the data length, the dimensions of the system, the type and parameters of the input signal, the type and parameters of the disturbances, the delay, as well as the initial value of the ODE.
3. Executes **SetupDataDelay.m**. This loads the necessary parameters into the MATLAB workspace.
4. Generates data by execution of **GenerateDataDelay.m**. After the execution of this script, variables with sampling instances, input signals and output signals are available in the MATLAB workspace.

Example 1: This and the following examples illustrates the use of the software package for identification of the system

$$\begin{aligned} \dot{x}(t) &= -x(t) + u(t-T) + 0.5u(t-T)x(t) \\ y(t) &= x(t) + e(t). \end{aligned} \tag{5}$$

This system is also used in [5]. Note that the delay does not enter exactly as defined in (2). This is intentional since such situations are common in practical situations and the location where the delay is modeled does not matter for time-invariant non-linear systems.

The relevant parts of the files **f.m** and **h.m** become

f.m

```
function [f]=f(t,x,u,w)

f(1,1)=-x(1,1)+u(1,1)+0.5*x(1,1)*u(1,1);

end
```

h.m

```
function [h]=h(t,x,u,e);
   h=x(1,1)+e(1,1);
end
```

Data is to be generated by simulation using a sampling period of $T_S = 0.10s$. 3000 input-output samples are to be generated. The input signal is to be selected with a uniform distribution in amplitude, with a mean of 0, a range $[-1,1]$ and a clock period 3.0s. The measurement disturbance is to be white, zero mean with a standard deviation of 0.1. The delay is to be $T_d = 1.22s$.

The setup script that performs this task is **SetupDataDelay.m**

```
% dimensions...

nu=1; % Input signal dimension
nx_0=1; % State dimension
ny=1; % Output dimension, normally 1

% Input signal related...Type may be selected among:
%
% InputType=[
%    'PRBS       ';
%    'Gaussian   ';
%    'UniformPRBS';
%    'SineWave   ';
%    'Custom     '];

InputType=[
   'UniformPRBS'];
uAmplitude=[
   1.0];
uMean=[
   0];
uFrequency=[
   0.2];
ClockPeriod=[
   30];% Clock period vector in terms of sampling time

% System disturbance related...Type may be selected among:
%
% DisturbanceTypeSystem=[
%   'WGN     ';
%   'SineWave';
%   'Custom  '];

DisturbanceTypeSystem=[
   'WGN     '];
wSigma=[
   0.0]; % Gaussian system noise standard deviation (discrete time)
wMean=[
   0];
wSineAmplitude=[
```

```
   0];
wSineFrequency=[
   0];

% Measurement disturbance related... Type may be selected among:
%
% DisturbanceTypeMeasuremet=[
%   'WGN     ';
%   'SineWave';
%   'Custom  '];

DisturbanceTypeMeasurement=[
   'WGN     '];
eSigma=0.1; % Gaussian measurement noise standard deviation (discrete time)
eMean=[
   0];
eSineAmplitude=[
   0];
eSineFrequency=[
   0];

% sampling time, data length and delay

Ts=0.1; % Sampling time in seconds
N=3000; % Number of data points
SamplingInstances=(Ts:Ts:N*Ts);
Td=1.22; % Delay in seconds

% ODE related...

x0=[0.5]'; % Initial values
```

The final step of the data generation is to execute the files **SetupDataDelay.m** and **GenerateDataDelay.m**. This is done in the MATLAB command window as follows

» SetupDataDelay

» GenerateDataDelay

…

percentReady =

```
  100
»
```

4.2 Live data

The generation of simulated data requires that the user

1. Is connected to the system via MATLAB. The connection must be such that commands to control DA-converters that generate input signals can be issued from within MATLAB. Similarly, commands that read AD-converters that sample output signals must be available from within MATLAB. The script **MeasurementMain.m** probably needs modification in a few parts in order to interface correctly to the AD- and DA-converters of the system of the user.
2. Generates an input signal, that is stored in the matrix (row vector in the one-dimensional input signal case) **u** .
3. Provides further data in the script **SetupLive.m**. The parameters that define the data generation are directly written into this script. These parameters defines the sampling period, the data length and the dimensions of the system.
4. Executes **SetupLive.m**. This loads the necessary parameters into the MATLAB workspace.
5. Generates data by execution of **MeasurementMain.m**. After the execution of this file, variables with input signals and output signals are available in the MATLAB workspace. This script operates as a loop that continuously polls the MATLAB real time clock, waiting for the next sampling instance. This means that it may not be possible to use the computer for other tasks during the data collection session. The reason for this solution is that it avoids the need for a real time OS connection. Note also that the calls to AD- and DA-converters may be different on other systems. This script is hence likely to require some modification.

   Example 2: The setup script file **SetupLive.m** becomes (empty since simulated data is used here)

```
% dimensions...Note that nx is not really relevant,
% it is however required in the RPEM setup so it is set in this file

nu=[]; % Input signal dimension
nx=[]; % State dimension
ny=[]; % Output dimension, normally 1

% sampling time and data length

Ts=[]; % Sampling time in seconds
N=[]; % Number of data points
SamplingInstances=(Ts:Ts:N*Ts);
```

The measurement process is started by typing

» SetupLive

» MeasurementMain

…

in the MATLAB command window. During the measurement session, the script continuously displays the time, the inputs as well as the measured outputs, as commanded to DA-converters and read by AD-converters. After termination all data that is needed for identification is available in the MATLAB workspace.

4.3 Display of data

After execution of either one of the chain of actions of section 4.1 or section 4.2, data can be plotted.

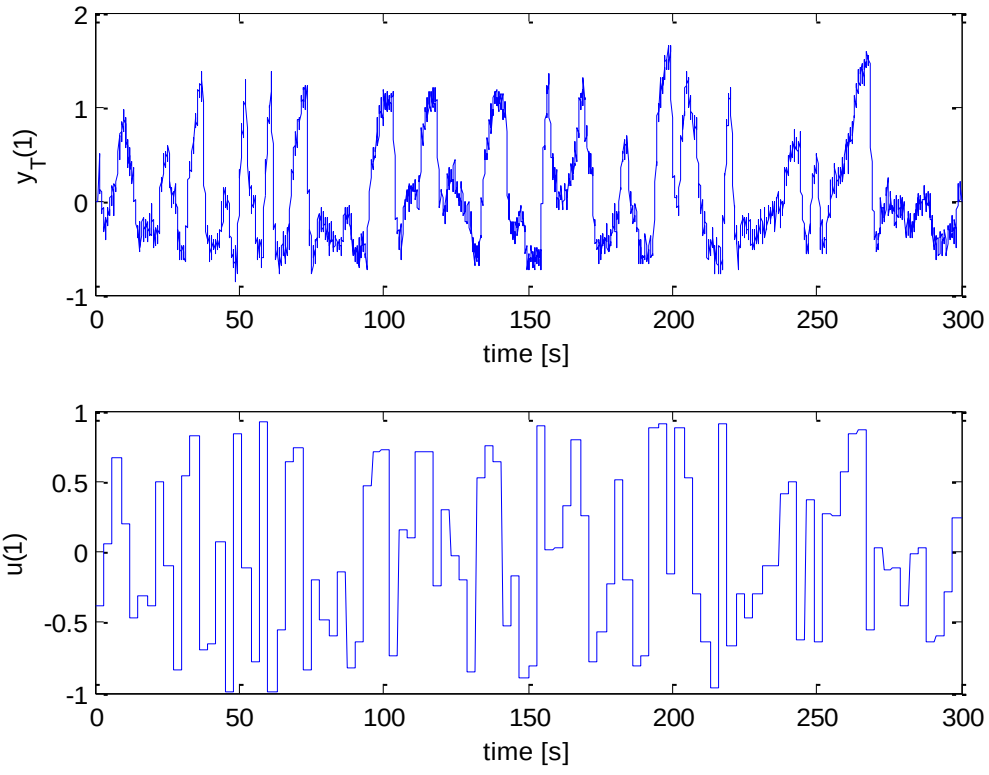
1. The **PlotDataDelay.m** script that is executed in the MATLAB command window. This script makes use of the dimensions of the system in order to divide the plot into several sub-windows, and in order to provide the axis text.

   Example 3: The MATLAB command window command is

» PlotDataDelay

»

The following plots are generated



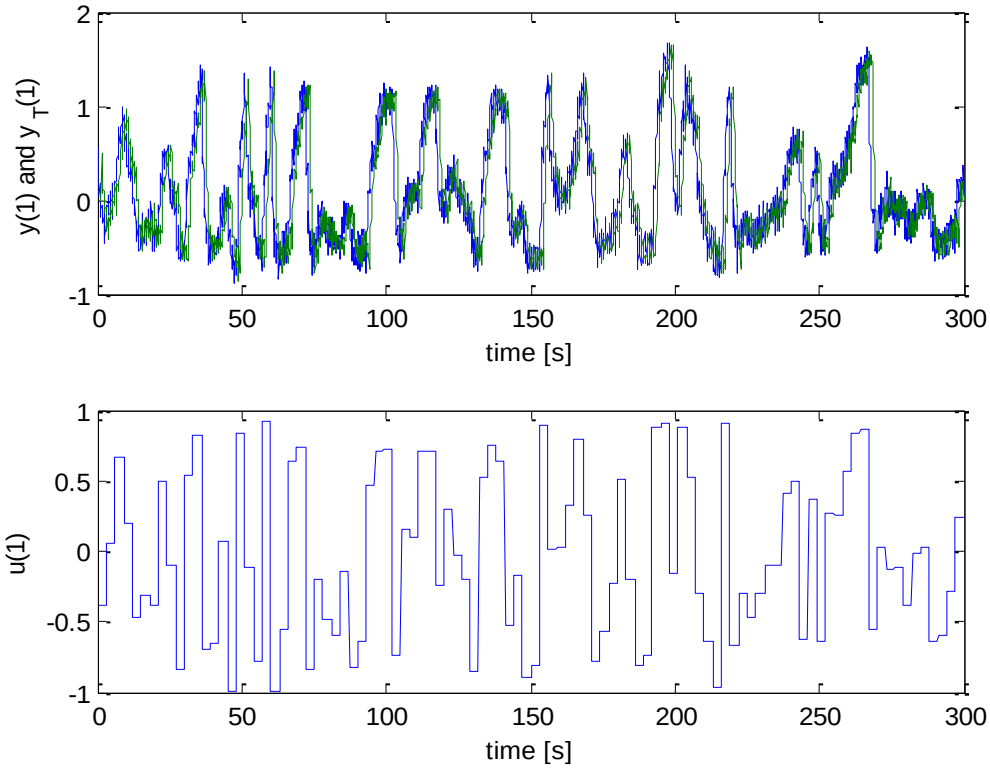


Figure 2: The results of a PlotDataDelay command. Note that both the delayed and un-delayed output is depicted.

5. Recursive Identification

At this point everything is in place for a first identification run.

5.1 RPEM setup

The preparation for the identification run requires that the user

1. Modifies the output equation and the corresponding derivative of underlying ODE model, as given by **h_m.m** and **dhdx_m.m**. The function **h_m.m** implements the output equation of the model. Note that this function is allowed to be a nonlinear function of the state and input. The function is not allowed to be dependent on the estimated parameters, it must be known a priori. Note also that the derivative of the function, with respect to the estimated state, needs to be supplied in the function **dhdx_m.m**.
2. Provides further input data in the script **SetupRPEMDelay.m**. The parameters that define the data generation are directly written into this script. These parameters define the dimension of the system, the initial value used in the ODE model, the gain sequence $\mu(t)/t$, the size of the initial value of the $\mathbf{R}$ -recursion, the initial value of the measurement covariance matrix $\Lambda(t)$, the stability limit applied by the projection algorithm, the allowed delay range, as well as the down-sampling

period used to avoid too large logs during long runs with high degree models. The reader is referred to [1] - [5] for details on these parameters, as well as on their use.

3. Executes **SetupRPEMDelay.m**. This loads the necessary parameters into the MATLAB workspace.

Example 4: The system is to be identified with a delayed first order model. The projection algorithm is to use a stability radius of 0.975 and the allowed delay range is to be [0.0 s, 2.0 s]. The initial value of the measurement covariance matrix is selected equal to 0.1. The initial value of the $\mathbf{R}$ - recursion (its inverse affecting the initial algorithmic gain) is selected equal to 10, unless for the component corresponding to the identified delay which is set to 1. The selection of the gain sequence $\mu(t)/t$ is a little more complicated, see [5] for details.

The functions **h_m.m** and **dhdx_m.m** become

h_m.m

```
function [h_m]=h_m(x_m,u);
   h_m=x_m(1,1);
end
```

dhdx_m.m

```
function [dhdx_m]=dhdx_m(x,u);
    dhdx_m=[1];
end
```

The setup script **SetupRPEMDelay.m** becomes

```
%  State initial value

nx=1;
x_m_0=[0.5]';
nu_m=nu;
u_m_0=u(:,1);
dudt_m_0=dudt(:,1);

% Remaining initial values

muFactor=300; % To stabilize Gamma and to reduce the gain
if exist('theta_0_new')
   muFactor=1000;
end
mu_0=10;
```

```
mu0=0.9995;
y_m_0=0;
initialNoiseVariance=0.1; % Initial value for the prediction error variance
scaleFactorR=10; % the size of the initial diagonal approximation of the
Hessian
scaleFactorRDelay=1; % the size of the initial diagonal approximation of
the Hessian

% Parameters

stabilityLimit=0.975; % The linearized pole radius used for stability
checking and projection
downSampling=1; % The downsampling factor used when data from the run is
saved
scalingTs=1; % The scaling factor with which the sampling period is
multiplied during identification
Td_min=0.0; % The minimum delay allowed during identification
Td_max=2.00; % The maximum delay allowed during identification
nd=ceil((Td_max-Td_min)/Ts)+1; % The number of samples covering the allowed
delay span
```

Finally the user executes **SetupRPEMDelay.m** in the MATLAB command window

```
» setupRPEMDelay
»
```

5.2 RPEM command window control and identified parameters

In order to perform an identification run the user is required to execute and provide input to the script **RPEMDelay.m**. The execution of this script makes use of four additional functions, implementing the polynomial model applied for modeling of the RHS of the ODE. These functions are **f_m.m**, **dfdx_m.m**, **dfdtheta_m.m** and **GenerateIndices.m**. The latter function generates the exponents of all factors of all terms of the polynomial expansion. The generation of these indices involves nested loops. They are therefore calculated in advance and used in repetitive calls in the form of a table.

To identify the system, the user is required to

1. Execute the script **RPEMDelay.m**
2. Provide the degrees of the polynomial model (***polynomialOrders)*** when prompted. The ***polynomialOrders*** variable is a column vector with the first element corresponding to the maximal degree of x_1, the second element corresponding to the maximal degree of x_2 and so on. The last element corresponds to the maximum degree of the derivative of highest degree of the last input

signal component. In the present example, ***polynomialOrders*** = [1 3]' would mean that the highest degree term of the polynomial expansion is $\theta_{13}x_1u^3$.

3. Provide a list of indices that are not to be used (***notUsedIndices***) by the algorithm. The indices exclude terms in the polynomial expansions. Providing an empty matrix ([]) indicates that no terms shall be excluded. The list of not used indices are to be provided as rows in a matrix , where the number of rows equals the number of terms that are to be excluded from the model. In the present example ***notUsedIndices*** = [0 0; 1 1] would mean that the terms θ_{00} and $\theta_{11}x_1u$ are to be excluded from the model.
4. Provide the initial parameter vector of the state space model (***theta_0***). Note that this parameter vector needs to correspond to a linearized system with all poles within the stability radius indicated by the script **SetupRPEMDelay.m**. If the initial parameter vector does not meet this criterion the user is prompted for ***theta_0*** again.
5. Provide the initial delay parameter (***Td_0***). If the given value is not within the allowed range a re-newed prompt is obtained.

   Note: A good strategy is to initialize the algorithm with a model that has time constants and a static gain that are similar to those of the system.

   Example 5: The algorithm is in this example initialized with

$$\begin{aligned}\hat{\boldsymbol{\theta}}_S(0) &= (0.0000 \quad 1.5000 \quad -0.5000 \quad 0.0000)^T \\ \theta_T &= 0.2000\end{aligned} \quad . \qquad (6)$$

The command sequence applied in the MATLAB command window i

```
>> RPEMDelay

ans =

Input polynomialOrders and notUsedIndices

K>> polynomialOrders=[1 1]'

polynomialOrders =

    1
    1

K>> notUsedIndices=[];
K>> return
```

```
allIndices =

     0     0
     0     1
     1     0
     1     1


ans =

Input theta_0

K>> theta_0=[0 1.5 -0.5 0]'

theta_0 =

         0
    1.5000
   -0.5000
         0

K>> return

LinearizedPoleRadii =

    0.9500


ans =

Input Td_0

K>> Td_0=0.2

Td_0 =
```

```
    0.2000

K>> return…
percentReady =

   100

ans =

    1.2206
   -0.0023
    1.0109
   -1.0065
    0.4975
```

Note that the exact result depends on the generated input signal. This may differ between systems and execution occasions since the seed for the random number generator may differ. Hence, slight variations of the estimated parameters are normal.

6. **Display of results**

The display of results is straightforward. By a study of the source code, users should be able to tailor available scripts and also write own ones when needed.

6.1 Parameters

In order to plot the parameters the user is required to

1. Execute the script **PlotParametersDelay.m**. The components of the parameter vector are then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

   Example 6: The command in the MATLAB command window is

```
» PlotParametersDelay
»
```

The following plot is then generated

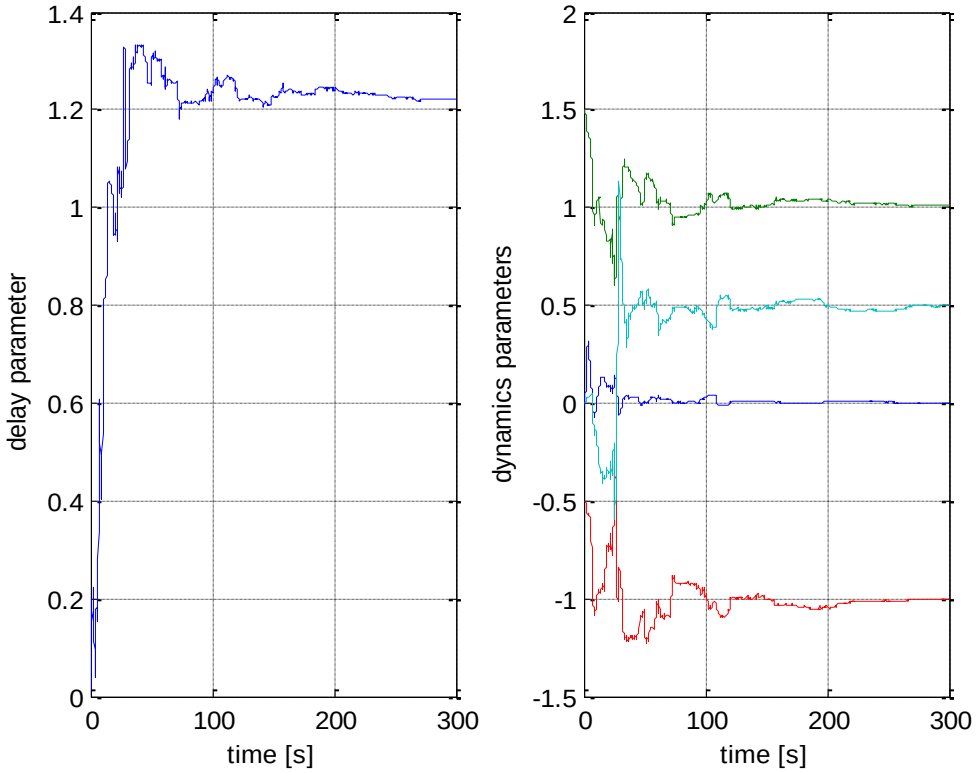


Figure 3: The result of a PlotParametersDelay command.

6.2 Prediction errors

In order to plot the parameters the user is required to

1. Execute the script **PlotPredictionErrorsDelay.m**. The prediction errors are then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

   Example 7: The MATLAB command window command is

» PlotPredictionErrorsDelay

»

The following plot is generated

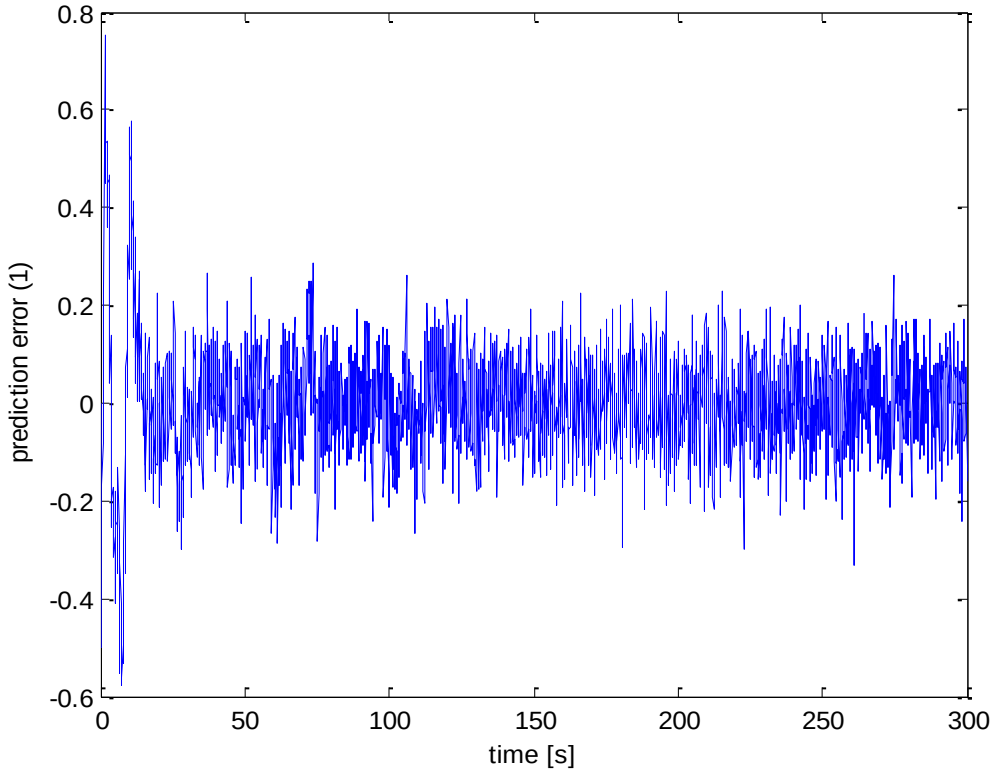


Figure 4: The result of a PlotPredictionErrorsDelay command.

6.3 Eigenvalues

In order to plot the convergence of the eigenvalues over time, the user is required to

1. Execute the script **PlotEigenvaluesDelay.m**. The eigenvalues of the Hessian are then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

   Example 8: The MATLAB command window command is

» PlotEigenvaluesDelay

»

The following plot is generated

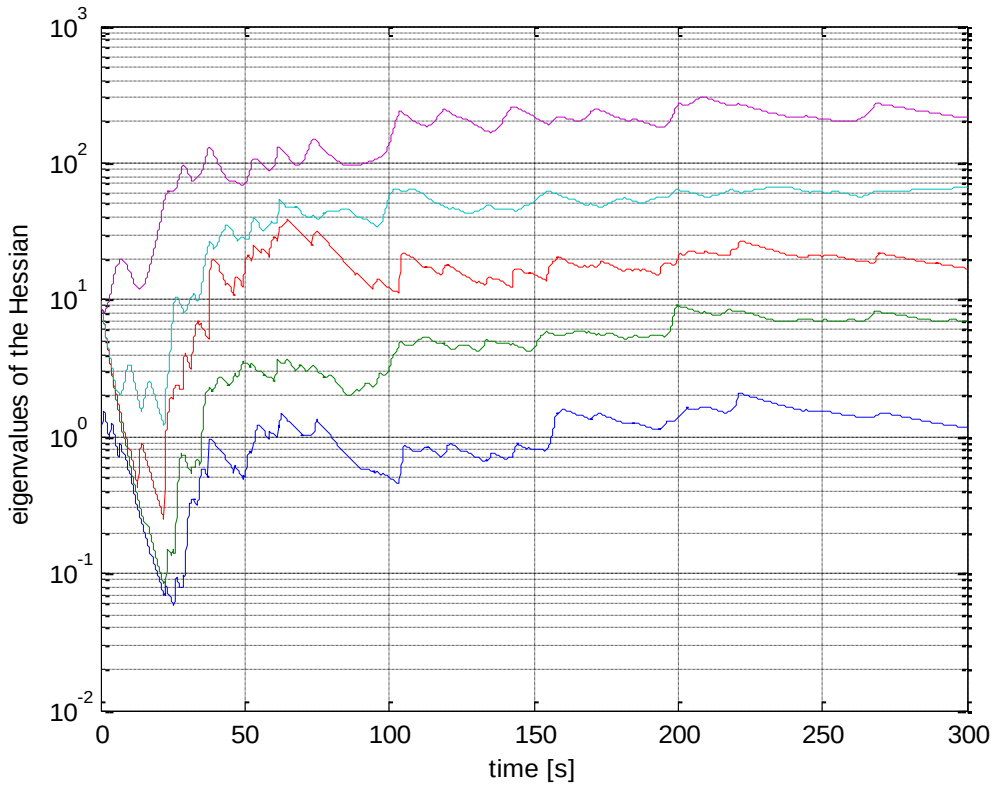


Figure 5: The result of a PlotEigenvaluesDelay command.

6.4 Condition number

In order to plot the convergence of the condition number over time, the user is required to

1. Execute the script **PlotConditionDelay.m**. The condition number is then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

   Example 9: The MATLAB command window command is

» PlotConditionDelay

»

The following plot is generated

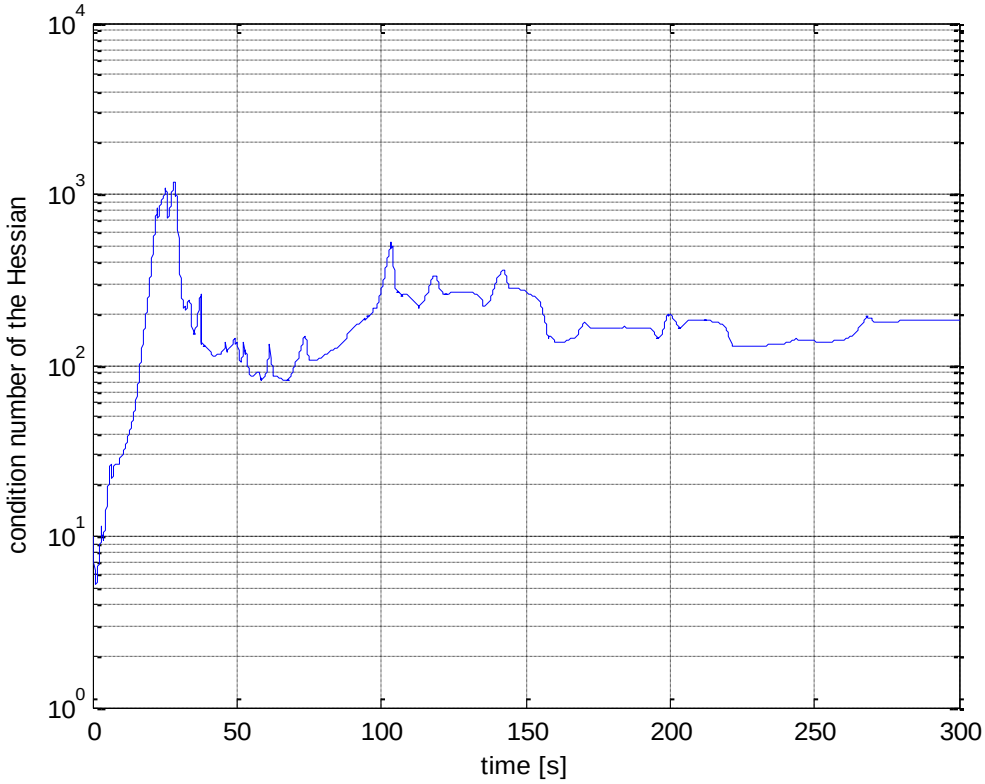


Figure 6: The result of a PlotConditionDelay command.

6.5 End of run model simulation

The above plot commands make use of logged signals, covering the transient part of the identification process. This is not always desired. To compare the identified model to the measured data it is e.g. more appropriate to simulate the model, using the parameters obtained at the end of the identification run.

Hence, to prepare for the remaining plot commands the user is required to

1. Execute the script **SimulateModelOutputDelay.m**.

Example10: The MATLAB command window command is

```
» SimulateModelOutputDelay
…
percentReady =
  100
»
```

The remaining plot commands and model validation commands can now be executed.

6.6 Simulated model output

In order to plot the simulated model output signal over time, the user is required to

1. Execute the script **PlotModelOutputDelay.m**. The output signals of the model are then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

   Example 11: The MATLAB command window command is

```
» PlotModelOutputDelay
»
```

The following plot is generated

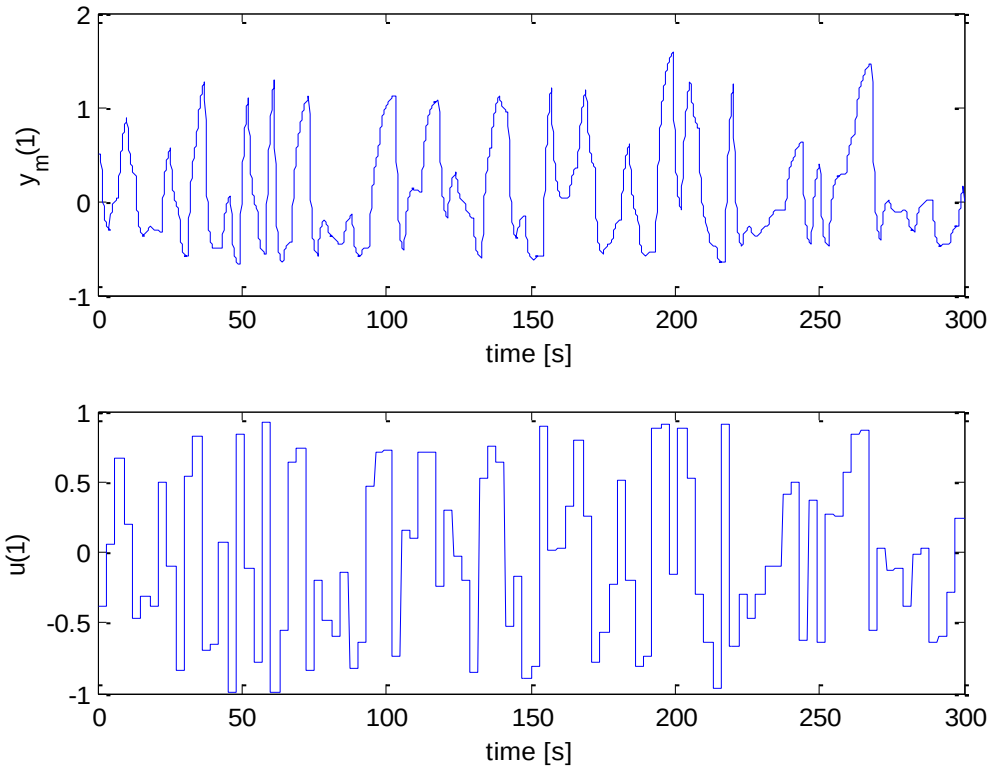


Figure 7: The result of a PlotModelOutputDelay command.

6.7 Simulated model output together with data

In order to plot the simulated model output signal over time, together with the corresponding measured data, the user is required to

1. Execute the script **PlotSystemAndModelOutputDelay.m**. The output signals of the model and the system are then plotted as a function of time, in the same plots. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

   Example 12: The MATLAB command window command is

```
» PlotSystemAndModelOutputDelay
»
```

The following plot is generated

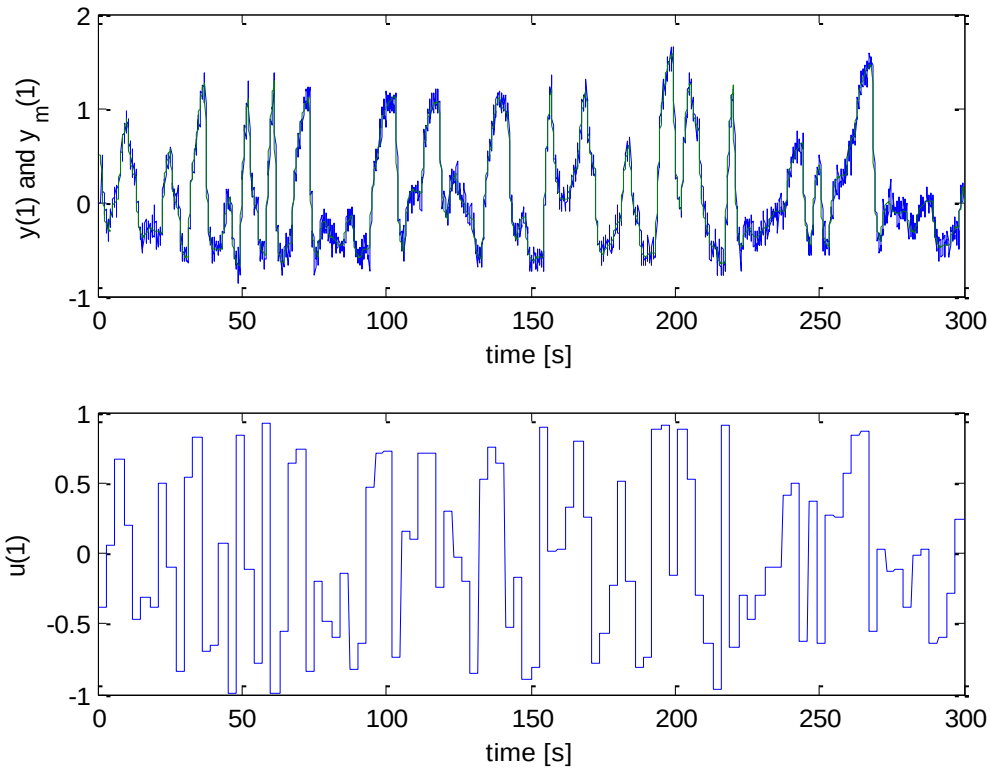


Figure 8: The result of a PlotSystemAndModelOutput command.

6.8 Residual errors

In order to plot the prediction errors, obtained from the simulated model output signal using parameters at the end of the identification run, the user is required to

1. Execute the script **PlotResidualErrorsDelay.m**. The residual errors are then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

Example 13: The MATLAB command window command is

» PlotResidualErrorsDelay

»

The following plot is generated

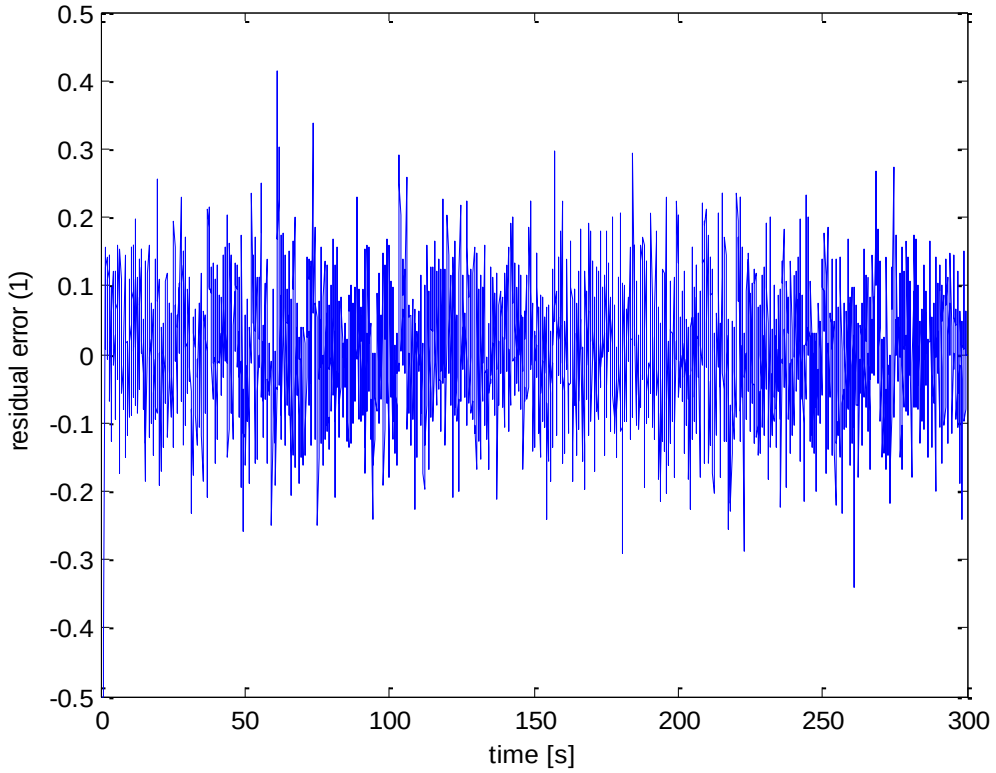


Figure 9: The result of a PlotResidualErrorsDelay command.

7. Model validation

Two scripts are provided for model validation purposes (in addition to the commands of section 6). The first method studies the value of the loss function that is minimized by the RPEM-algorithm. Since the measurement covariance matrix is estimated, the loss function contains an additional term on top of the sample average of the squared prediction errors.

7.1 RPEM loss function

The RPEM loss function that is computed is given by

$$V\left(\hat{\boldsymbol{\theta}}\left(t_0+NT_S\right)\right)=\frac{1}{2}\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\varepsilon}\left(t_0+iT_S,\hat{\boldsymbol{\theta}}\left(t_0+NT_S\right)\right)\boldsymbol{\varepsilon}^T\left(t_0+iT_S,\hat{\boldsymbol{\theta}}\left(t_0+NT_S\right)\right)+\frac{1}{2}\log\det\left(\hat{\boldsymbol{\Lambda}}\left(t_0+NT_S\right)\right) \qquad (7)$$

The user is referred to [1], [5] and the references therein for further details. In order to compute the loss function, using parameters at the end of the identification run, the user is required to

1. Execute the script **ComputRPEMLossFunctionDelay.m**. The loss function is then computed.

Example 14: The MATLAB command window command is

» ComputeRPEMLossFunctionDelay

…

percentReady =

100

V =

-1.8220

>>

Note: Another relevant measure to use is the sum of the squared prediction errors.

7.2 Mean residual analysis

Mean residual analysis is a method that evaluates the obtained static characteristics of an identified model of any kind. It operates by sorting residual errors into bins, the bin being decided by the value of the measured output signal with the same time index as the residual error. The mean of the residuals are then computed, in each bin, and plotted against the range of the output signal. The number of samples of each bin is also plotted. The user is referred to [9] for further details. In order to perform mean residual analysis, the user is required to

1. Execute the script **MeanResidualAnalysisDelay.m**.
2. Provide the intervals used by the method when prompted for ***intervals***.

Example 15: This example performs mean residual analysis using about 40 intervals, each with an output amplitude width of 0.1. The MATLAB command window command is

» meanResidualAnalysisDelay

ans =

Input intervals for division into bins

K» intervals=(-2:0.1:2);

K» return

»

The following plot results

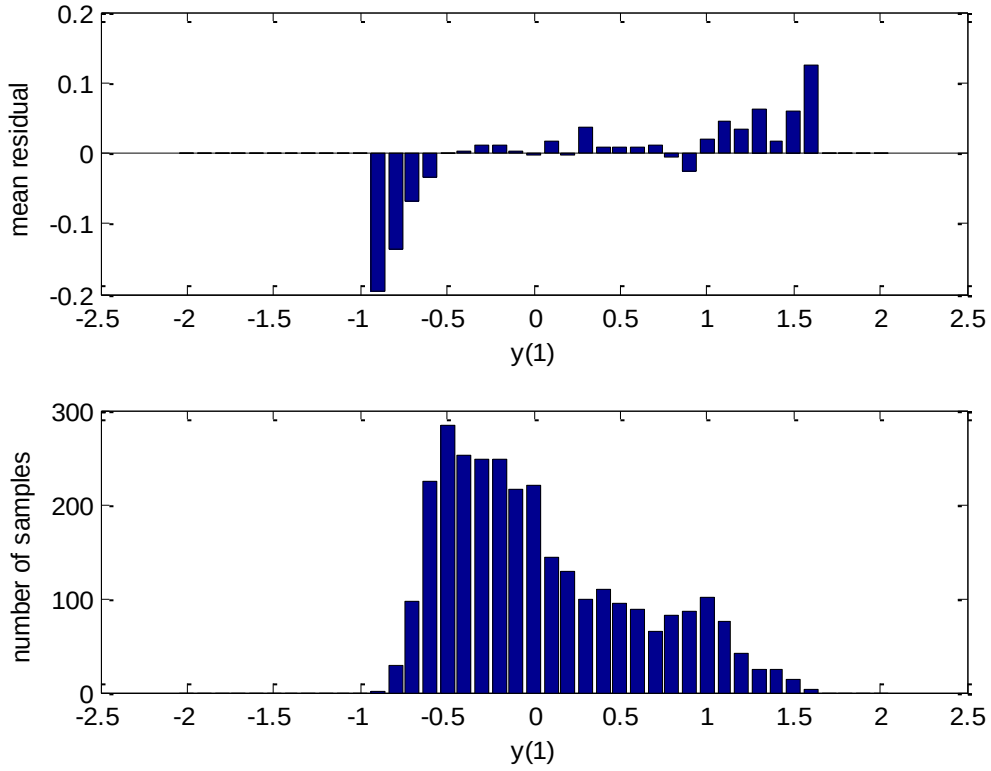


Figure 10: The result of a MeanResidualAnalysisDelay command.

8. Model – ODE with unknown measurement equation

8.1 ODE model with scaling

The nonlinear MIMO model to be defined here is the same as described in Section 2 of this report. It is used for estimation of the unknown parameter vector $\begin{pmatrix}\theta_T & \boldsymbol{\theta}_S^T\end{pmatrix}^T$ from measured inputs $\mathbf{u}(t)$ and outputs $\mathbf{y}_m(t)$, given by

$$\begin{aligned}\mathbf{u}(t) &= \begin{pmatrix}u_1(t) & \dots & u_1^{(n_1)}(t) & \dots & u_k(t) & \dots & u_k^{(n_k)}(t)\end{pmatrix}^T \\ \mathbf{y}_m(t) &= \begin{pmatrix}y_{m,1}(t) & \dots & y_{m,p}(t)\end{pmatrix}^T.\end{aligned} \qquad (8)$$

The superscript $^{(k)}$ denotes differentiation k times. The continuous time model is given by the following n:th order state space ODE

$$\begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_{n-1}^{(1)} \\ x_n^{(1)} \end{pmatrix} = \begin{pmatrix} x_2 \\ \vdots \\ x_n \\ f\left(x_1,\ldots,x_n,u_1,\ldots,u_1^{(n_1)},\ldots,u_k,\ldots,u_k^{(n_k)},\boldsymbol{\theta}_S\right) \end{pmatrix}, \tag{9}$$

where $\mathbf{x} = \begin{pmatrix} x_1 & \ldots & x_{n-1} & x_n \end{pmatrix}^T$ is the state vector. The following polynomial parameterization of the right hand side function of (2) is used

$$\begin{aligned} & f\left(x_1,...,x_n,u_1,...,u_1^{(n_1)},...,u_k,...,u_k^{(n_k)},\boldsymbol{\theta}_S\right) \\ & = \sum_{i_{x_1}=0}^{I_{x_1}} \ldots \sum_{i_{x_n}=0}^{I_{x_n}} \sum_{i_{u_1}=0}^{I_{u_1}} \ldots \sum_{i_{u_1^{(n_1)}}=0}^{I_{u_1^{(n_1)}}} \ldots \sum_{i_{u_k}=0}^{I_{u_k}} \ldots \sum_{i_{u_k^{(n_k)}}}^{I_{u_k^{(n_k)}}} \theta_{S,i_{x_1}\ldots i_{x_n} i_{u_1}\ldots i_{u_1^{(n_1)}}\ldots i_{u_k}\ldots i_{u_k^{(n_k)}}} \left(x_1\right)^{i_{x_1}} \ldots \left(x_n\right)^{i_{x_n}} \left(u_1\right)^{i_{u_1}} \ldots \\ & \ldots \left(u_1^{(n_1)}\right)^{i_{u_1^{(n_1)}}} \ldots \left(u_k\right)^{i_{u_k}} \ldots \left(u_k^{(n_k)}\right)^{i_{u_k^{(n_k)}}} = \varphi^T(\mathbf{x},\mathbf{u})\boldsymbol{\theta}_S . \end{aligned} \tag{10}$$

Here

$$\begin{aligned} \boldsymbol{\theta}_S = \Big(& \theta_{S,0\ldots0} \quad \ldots \quad \theta_{S,0\ldots I_{u_k^{(n_k)}}} \quad \theta_{S,0\ldots010} \quad \ldots \quad \theta_{S,0\ldots01I_{u_k^{(n_k)}}} \quad \ldots \\ & \theta_{S,0\ldots0I_{u_k^{(n_k-1)}}0} \quad \ldots \quad \theta_{S,0\ldots0I_{u_k^{(n_k-1)}}I_{u_k^{(n_k)}}} \quad \ldots \quad \theta_{S,I_{X_1}\ldots I_{u_k^{(n_k)}}} \Big)^T \end{aligned}$$

$$\begin{aligned} \boldsymbol{\varphi} = \Big(& 1 \quad \ldots \quad \left(\left(u_k^{(n_k)}\right)^{I_{u_k^{(n_k)}}}\right) \quad u_k^{(n_k-1)} \quad \ldots \left(u_k^{(n_k-1)}\left(u_k^{(n_k)}\right)^{I_{u_k^{(n_k)}}}\right) \ldots \quad \left(u_k^{(n_k-1)}\right)^{I_{u_k^{(n_k-1)}}} \quad \ldots \\ & \left(\left(u_k^{(n_k-1)}\right)^{I_{u_k^{(n_k-1)}}}\left(u_k^{(n_k)}\right)^{I_{u_k^{(n_k)}}}\right) \quad \ldots \left(\left(x_1\right)^{I_{x_1}} \ldots \left(x_n\right)^{I_{x_n}} \left(u_1\right)^{I_{u_1}} \ldots \left(u_k^{(n_k)}\right)^{I_{u_k^{(n_k)}}}\right)\Big)^T . \end{aligned} \tag{11}$$

In order to obtain a discrete time model, the Euler integration method is applied to (9). In addition, a multiple model approach together with interpolation is used for discretization of the delay θ_T. This results in a capability to estimate large integer delays together with a fractional part, which results in a very high modeling precision. The details are quite complicated and discussed at length in [5].

8.2 Output model

To describe the output model it is noted that ODE model and the output equation form a cascade system. In the static case, with a constant input signal a steady state value (operating point) is then reached in case the system is e.g. exponentially stable. A small variation around the constant input then results in a small variation around the constant output. Using the chain rule on the formal relation

$$\boldsymbol{y} = \boldsymbol{h}\left(\boldsymbol{x}(\boldsymbol{u})\right), \tag{12}$$

results in

$$\Delta \boldsymbol{y} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{u}} \Delta \boldsymbol{u}. \tag{13}$$

Here the delay has been set to 0 to simplify the notation. This means that the differential gain is the product of the differential gains of the ODE and the output equation. Obviously, this creates an ambiguity for the identification algorithm, since the static differential gain can be arbitrarily distributed between the two blocks without changing the input to output relation. One solution, also used in [11] is to fix the differential static gain in the output equation.

Here the output equation is restricted to be the scalar function $h(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o)$, of the states, the inputs and an unknown parameter vector $\boldsymbol{\theta}_o$, where the subscript denotes output. The output model can then be formulated as:

$$\begin{aligned} y(t,\boldsymbol{\theta}_S,\theta_T,\boldsymbol{\theta}_o) &= h(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o) \\ &= \begin{cases} h^+(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o^+), & x_1(t-\theta_T,\boldsymbol{\theta}_S) > x^+ \\ k_0 x_1(t-\theta_T,\boldsymbol{\theta}_S) + \theta_{o,0}, & x_1^- \le x_1(t-\theta_T,\boldsymbol{\theta}_S) \le x_1^+ \\ h^-(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o^-) & x_1(t-\theta_T,\boldsymbol{\theta}_S) < x^- \end{cases} . \end{aligned} \tag{14}$$

Exactly as in [11] k_o is the static gain of the output equation, and $\theta_{o,0}$ is a free bias parameter that needs to be identified. It is noted that the independent variable is selected to be the first state vector component, to make sure that all dynamics is observable in the measured output.

It remains to find parameterizations for the two functions $h^-(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o^-)$ and $h^+(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o^+)$. The choice here is to re-use the multivariate polynomial model that is used to parameterize the ODE. However, continuity constraints need to be built into these models to ensure that

$$\begin{aligned} \lim_{x_1(t-\theta_T,\boldsymbol{\theta}_S)\to x^+} h^+(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o^+) &= k_0 x^+ + \theta_{o,0} \\ \lim_{x_1(t-\theta_T,\boldsymbol{\theta}_S)\to x^-} h^-(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o^-) &= k_0 x^- + \theta_{o,0} \end{aligned} \tag{15}$$

This means that $x_1(t-\theta_T,\boldsymbol{\theta}_S) - x^-$ needs to be a factor in the parameterized nonlinear part of $h^-(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o^-)$ and similarly for $h^+(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o^+)$. Therefore, these functions are selected as

$$\begin{aligned} h^-(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o^-) &= (x_1(t-\theta_T,\boldsymbol{\theta}_S) - x^-)\varphi_-^T\big(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T)\big)\boldsymbol{\theta}_o^- + \theta_{o,0} \\ h^+(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T),\boldsymbol{\theta}_o^+) &= (x_1(t-\theta_T,\boldsymbol{\theta}_S) - x^+)\varphi_+^T\big(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T)\big)\boldsymbol{\theta}_o^+ + \theta_{o,0} . \end{aligned} \tag{16}$$

In summary

$$y(t,\boldsymbol{\theta}_S,\theta_T,\boldsymbol{\theta}_o)$$

$$= \begin{cases} (x_1(t-\theta_T,\boldsymbol{\theta}_S)-x^+)\varphi_+^T\big(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T)\big)\boldsymbol{\theta}_o^+ + \theta_{o,0}, & x_1(t-\theta_T,\boldsymbol{\theta}_S) > x^+ \\ k_0 x_1(t-\theta_T,\boldsymbol{\theta}_S) + \theta_{o,0}, & x_1^- \le x_1(t-\theta_T,\boldsymbol{\theta}_S) \le x_1^+ \\ (x_1(t-\theta_T,\boldsymbol{\theta}_S)-x^-)\varphi_-^T\big(\boldsymbol{x}(t-\theta_T,\boldsymbol{\theta}_S),\boldsymbol{u}(t-\theta_T)\big)\boldsymbol{\theta}_o^- + \theta_{o,0} & x_1(t-\theta_T,\boldsymbol{\theta}_S) < x^- \end{cases} . \quad (17)$$

Thus, $\theta_{o,0}$ becomes a common parameter over the whole range of $x_1(t-\theta_T,\boldsymbol{\theta}_S)$.

9. Software package overview – unknown measurement equation

9.1 Introduction

The scripts and functions handling the identification of the measurement equation case are all marked with a string DelayOutput at the end, a fact which allow them to be stored together with the other files of the SW package without any ambiguity.

9.2 Scripts and functions

The command flow is the same as without measurement equation identification. Roughly, the output equation scripts and functions can be divided into three groups since live data measurement and supporting function parts do not require any change:

- *Simulated data generation*. The four scripts of this group define a dynamic system that is then used for generation of simulated data. The scripts and functions are **SetupDataDelayOutput.m**, **f.m (**unchanged), **hOut.m** and **GenerateDataDelayOutput.m**.
- *Recursive identification*. The five scripts of this group perform the actual identification tasks, supporting user interaction. The scripts and functions are **SetupRPEMDelayOutput.m**, **RPEMDelayOutput.m**, **hOutput_m.m, dhdThetaOutput_m.m, dhduOutput_m.m, phiThetaOutput_m,** and **dhdxOutput_m.m**.
- *Preparation and display of results*. There are eleven scripts in this group. They all prepare, compute and display results of the identification process. The scripts are **PlotDataDelayOutput.m, SimulateModelOutputDelayOutput.m**, **PlotParametersDelayOutput.m**, **PlotPredictionErrorsDelayOutput.m**, **PlotEigenvaluesDelayOutput.m**, **PlotConditionDelayOutput.m**, **ComputeRPEMLossFunctionDelayOutput.m**, **PlotModelOutputDelayOutput.m**, **PlotSystemAndModelOutputDelayOutput.m**, **PlotResidualErrorsDelayOutput.m** and **MeanResidualAnalysisDelay.m** (unchanged).

These groups of scripts and functions need to be operated in a particular order to make sense. This order of execution between scripts and functions is similar to the one of Figure 1 of [6].

There are three major ways to exploit the five groups of scripts and functions.

1. In case the user has input and output signals available, the first step is to define and run the script **SetupLive.m.** This sets basic parameters like the sampling period. The user can then

proceed directly to use the groups *Recursive Identification* and *Preparation and display of results.* The data, which can be simulated or live, should be stored in the (row) matrices **u** and **y** .

2. In case the user is to perform live measurements, all the steps of the *Live data measurement* group should be executed first. The user can then proceed directly to use the groups *Recursive Identification* and *Preparation and display of results.*
3. In case the user intends to use simulated data, this data can be generated by execution of the scripts and functions of the group *Simulated data generation.* The user can then proceed directly to use the groups *Recursive Identification* and *Preparation and display of results.*

10. Data input

10.1 Simulated data

The generation of simulated data requires that the user

1. Modifies the underlying ODE model, as given by **f.m** and **hOut.m**. The function **f.m** implements the RHS of the ODE, using a conventional MATLAB function call. Note that the built in ODE solvers of MATLAB are not used. Instead an Euler algorithm is implemented. The reason is that this allows the generation of simulated data that can be exactly described by the applied model, should this be desired. The function **hOut.m** implements the (possibly nonlinear) measurement equation. The functions allow for addition of systems noise and measurement noise.
2. Provides further input data in the script **SetupDataDelayOutput.m**. The parameters that define the data generation are directly written into this script. These parameters define the sampling period, the data length, the dimensions of the system, the type and parameters of the input signal, the type and parameters of the disturbances, the delay, as well as the initial value of the ODE.
3. Executes **SetupDataDelayOutput.m**. This loads the necessary parameters into the MATLAB workspace.
4. Generates data by execution of **GenerateDataDelayOutput.m**. After the execution of this script, variables with sampling instances, input signals and output signals are available in the MATLAB workspace.

Example 16: This and the following examples illustrates the use of the software package for identification of the system

$$\dot{x}(t) = -x(t) + u(t-T) + 0.5u(t-T)x(t)$$
$$y(t) = \begin{cases} x(t) + e(t), & x(t) < 0.6 \\ 1.12x(t) - 0.2x^2(t) + e(t), & x(t) \geq 0.6 \end{cases} \quad (18)$$

Note that the delay does not enter exactly as defined in (2). This is intentional since such situations are common in practical situations and the location where the delay is modeled does not matter for time-invariant non-linear systems, see [5].

The relevant parts of the files **f.m** and **h.m** become

f.m

```
function [f]=f(t,x,u,w)
    f(1,1)=-x(1,1)+u(1,1)+0.5*x(1,1)*u(1,1);
end
```

hOut.m

```
function [h]=h(t,x,u,e)
   if (x(1,1)<0.6)
       h=x(1,1)+e(1,1);
   else
       h=1.12*x(1,1)-0.2*x(1,1)*x(1,1)+e(1,1);
   end
end
```

Note that **h.m** and **hOut.m** are both called by h, the file name difference is to facilitate output data definitions for the two types of systems simultaneously. **h.m** could therefore be used as well.

Data is to be generated by simulation using a sampling period of $T_S = 0.10s$. 10000 input-output samples are to be generated. The input signal is to be selected with a uniform distribution in amplitude, with a mean of 0, a range $[-1,1]$ and a clock period 3.0s. The measurement disturbance is to be white, zero mean with a standard deviation of 0.01. The delay is to be $T_d = 0.33s$.

The setup script that performs this task is **SetupDataDelayOutput.m**

```
% dimensions...

nu=1; % Input signal dimension

nx_0=1; % State dimension

ny=1; % Output dimension, normally 1

% Input signal related...Type may be selected among:

%

% InputType=[

%    'PRBS       ';

%    'Gaussian   ';

%    'UniformPRBS';

%    'SineWave   ';

%    'Custom     '];
```

```
InputType=[
   'UniformPRBS'];
uAmplitude=[
   1.0];
uMean=[
   0];
uFrequency=[
   0.2];
ClockPeriod=[
   30];% Clock period vector in terms of sampling time

% System disturbance related...Type may be selected among:
%
% DisturbanceTypeSystem=[
%   'WGN     ';
%   'SineWave';
%   'Custom  '];

DisturbanceTypeSystem=[
   'WGN     '];
wSigma=[
   0.0]; % Gaussian system noise standard deviation (discrete time)
wMean=[
   0];
wSineAmplitude=[
   0];
wSineFrequency=[
   0];
% Measurement disturbance related... Type may be selected among:
%
% DisturbanceTypeMeasuremet=[
```

```
%   'WGN     ';
%   'SineWave';
%   'Custom  '];
DisturbanceTypeMeasurement=[
   'WGN     '];
eSigma=0.01; % Gaussian measurement noise standard deviation (discrete
time)
eMean=[
   0];
eSineAmplitude=[
   0];
eSineFrequency=[
   0];
% sampling time, data length and delay
Ts=0.1; % Sampling time in seconds
N=10000; % Number of data points
SamplingInstances=(Ts:Ts:N*Ts);
Td=0.33; % Delay in seconds
% ODE related...
x0=[0.5]'; % Initial values
```

The final step of the data generation is to execute the files **SetupDataDelayOutput.m** and **GenerateDataDelayOutput.m**. This is done in the MATLAB command window as follows

```
» SetupDataDelayOutput
» GenerateDataDelayOutput
…
percentReady =
  100
»
```

10.2 Display of data

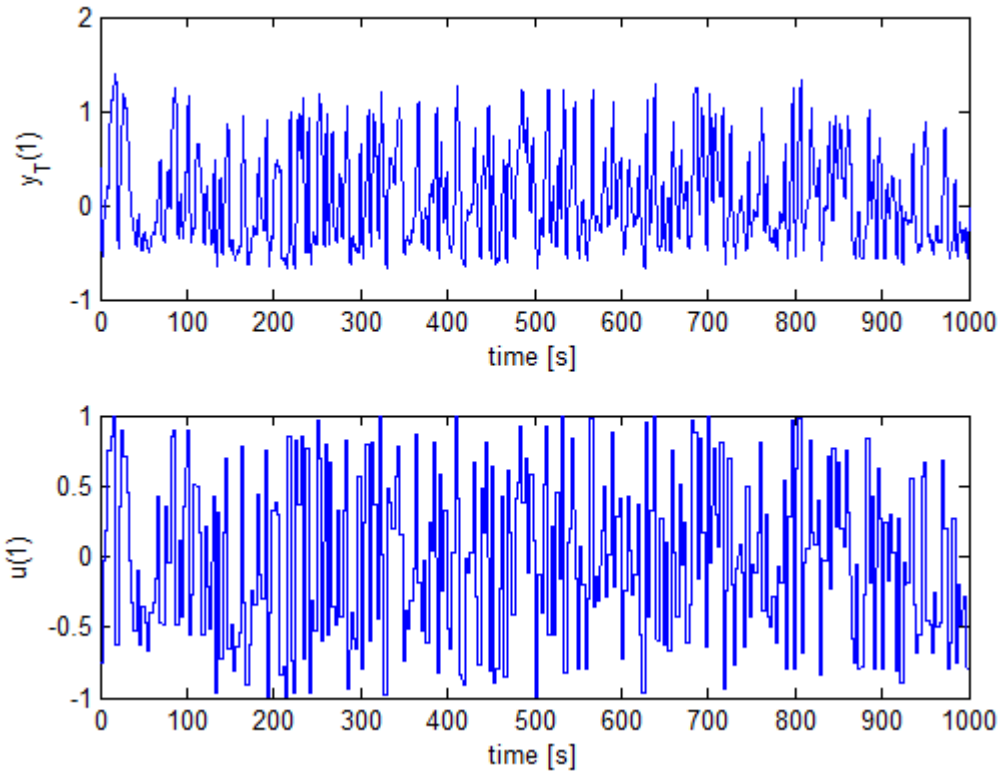
After execution, data can be plotted.

1. The **PlotDataDelayOutput.m** script that is executed in the MATLAB command window. This script makes use of the dimensions of the system in order to divide the plot into several sub-windows, and in order to provide the axis text.

Example 17: The MATLAB command window command is

```
» PlotDataDelayOutput
»
```

The following plots are generated



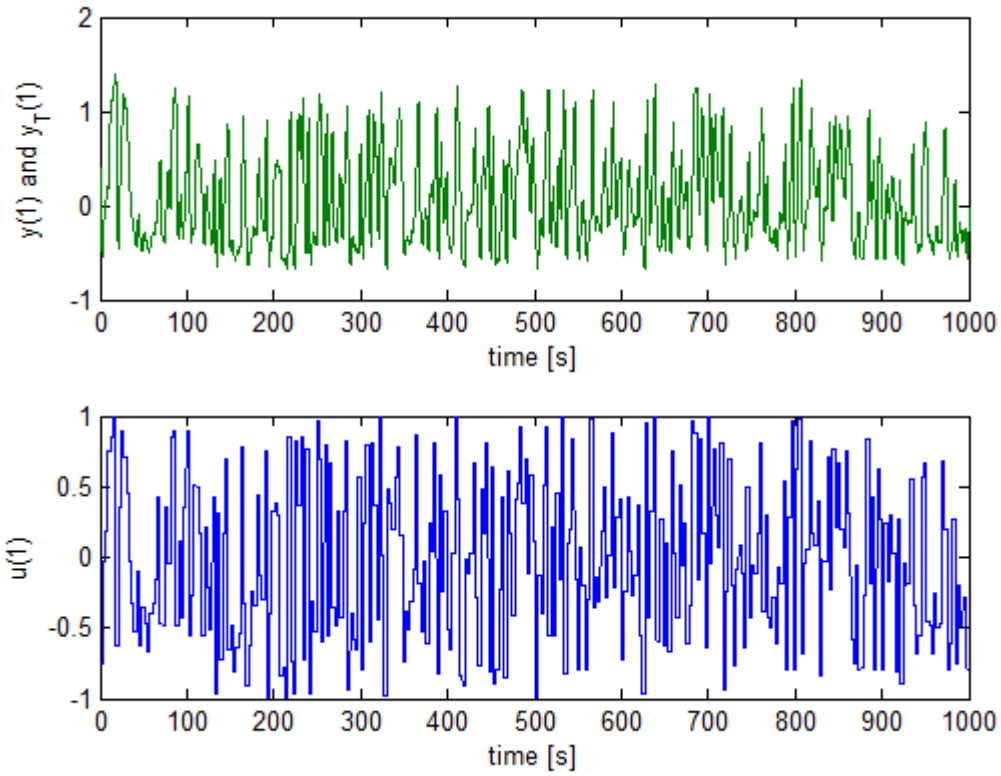


Figure 11: The results of a PlotDataDelayOutput command. Note that both the delayed and un-delayed output is depicted.

11. Recursive Identification

At this point everything is in place for a first identification run.

11.1 RPEM setup

The preparation for the identification run requires that the user

1. Provides further input data in the script **SetupRPEMDelayOutput.m**. The parameters that define the data generation are directly written into this script. These parameters define the dimension of the system, the initial value used in the ODE model, the gain sequence $\mu(t)/t$, the size of the initial value of the **R** -recursion, the initial value of the measurement covariance matrix $\Lambda(t)$, the stability limit applied by the projection algorithm, the allowed delay range, as well as the down-sampling period used to avoid too large logs during long runs with high degree models. The reader is referred to [1] - [5] for details on these parameters, as well as on their use.

4. Executes **SetupRPEMDelayOutput.m**. This loads the necessary parameters into the MATLAB workspace.

Example 18: The system is to be identified with a delayed first order model. The projection algorithm is to use a stability radius of 0.975 and the allowed delay range is to be [0.00 s, 0.50 s]. The

initial value of the measurement covariance matrix is selected equal to 0.1. The initial value of the $\mathbf{R}$ -recursion (its inverse affecting the initial algorithmic gain) is selected equal to 10, unless for the component corresponding to the identified delay which is set to 1. The selection of the gain sequence $\mu(t)/t$ is a little more complicated, see [5] for details.

The setup script **SetupRPEMDelayOutput.m** becomes

```
%  State initial value

nx=1;
x_m_0=[0.51]';
nu_m=nu;
u_m_0=u(:,1);
dudt_m_0=dudt(:,1);

% Remaining initial values

muFactor=300; % To stabilize Gamma and to reduce the gain
if exist('theta_0_new')
   muFactor=1000;
end
mu_0=10;
mu0=0.9995;
y_m_0=0;
initialNoiseVariance=0.1; % Initial value for the prediction error variance
scaleFactorR=10; % the size of the initial diagonal approximation of the
Hessian
scaleFactorRDelay=1; % the size of the initial diagonal approximation of
the Hessian

% Parameters

stabilityLimit=0.975; % The linearized pole radius used for stability
checking and projection
downSampling=1; % The downsampling factor used when data from the run is
saved
scalingTs=1; % The scaling factor with which the sampling period is
multiplied during identification
Td_min=0.0; % The minimum delay allowed during identification
```

```
Td_max=0.50; % The maximum delay allowed during identification
nd=ceil((Td_max-Td_min)/Ts)+1; % The number of samples covering the allowed
delay span
```

Finally the user executes **SetupRPEMDelay.m** in the MATLAB command window

» SetupRPEMDelay

»

11.2 RPEM command window control and identified parameters

In order to perform an identification run the user is required to execute and provide input to the script **RPEMDelayOutput.m**. The execution of this script makes use of a number of additional functions, implementing the polynomial models applied for modeling of the RHS of the ODE, and the measurement equation model.

To identify the system, the user is required to

1. Execute the script **RPEMDelay.m**
2. Provide the degrees of the polynomial model (***polynomialOrders)*** when prompted. The ***polynomialOrders*** variable is a column vector with the first element corresponding to the maximal degree of x_1, the second element corresponding to the maximal degree of x_2 and so on. The last element corresponds to the maximum degree of the derivative of highest degree of the last input signal component. In the present example, ***polynomialOrders*** = [1 3]' would mean that the highest degree term of the polynomial expansion is $\theta_{13}x_1u^3$.
3. Provide a list of indices that are not to be used (***notUsedIndices***) by the algorithm. The indices exclude terms in the polynomial expansions. Providing an empty matrix ([]) indicates that no terms shall be excluded. The list of not used indices are to be provided as rows in a matrix , where the number of rows equals the number of terms that are to be excluded from the model. In the present example ***notUsedIndices*** = [0 0; 1 1] would mean that the terms θ_{00} and $\theta_{11}x_1u$ are to be excluded from the model.
4. Provide the initial parameter vector of the state space model (***theta_0***). Note that this parameter vector needs to correspond to a linearized system with all poles within the stability radius indicated by the script **SetupRPEMDelay.m**. If the initial parameter vector does not meet this criterion the user is prompted for ***theta_0*** again.
5. Provide the initial delay parameter (***Td_0***). If the given value is not within the allowed range a re-newed prompt is obtained.
6. Provide the measurement equation user choices related to the middle interval, ***k_0***, ***x_minus*** and ***x_plus***.
7. Provide the initial value for the bias parameter of the measurement equation, ***thetaBias_0***.

8. Provide the model structure information for the lower interval of the measurement equation, ***polynomialOrdersOutputLow*** and ***notUsedIndicesOutputLow.*** The definition is the same as for the polynomial nonlinearity of the RHS of the ODE.
9. Provide initial values for the parameter vector of the lower interval of the measurement equation, ***thetaOutputLow_0***.
10. Provide the model structure information for the upper interval of the measurement equation, ***polynomialOrdersOutputHigh*** and ***notUsedIndicesOutputHigh.*** The definition is the same as for the polynomial nonlinearity of the RHS of the ODE.
11. Provide initial values for the parameter vector of the lower interval of the measurement equation, ***thetaOutputHigh_0***.

Note: A good strategy is to initialize the algorithm with a model that has time constants and a static gain that are similar to those of the system. Another recommendation is to exclude the constant of the ODE RHS in case that is consistent with prior knowledge, since this appears to improve the convergence properties.

Example 19: The algorithm is in this example initialized with

$$\begin{gathered}\hat{\boldsymbol{\theta}}_S(0) = (0.5 \quad -0.5 \quad 0.0)^T \\ \hat{\theta}_T(0) = 0.1 \\ \hat{\theta}_{o,0}(0) = 0.0 \\ \hat{\theta}_o^-(0) = 0.5 \\ \hat{\boldsymbol{\theta}}_o^+(0) = (1.0 \quad 0.0 \quad 0.0 \quad 0.0)^T.\end{gathered} \tag{19}$$

The command sequence applied in the MATLAB command window is

```
>> RPEMDelayOutput

ans =

Input polynomialOrders and notUsedIndices

K>> polynomialOrders=[1 1]'

polynomialOrders =

    1
    1

K>> notUsedIndices=[0 0]
```

```
notUsedIndices =

     0     0

K>> return

allIndices =

     0     1
     1     0
     1     1


ans =

Input theta_0

K>> theta_0=[0.5 -0.5 0]'

theta_0 =

    0.5000
   -0.5000
         0

K>> return

LinearizedPoleRadii =

    0.9500


ans =

Input Td_0

K>> Td_0=0.1;
```

K>> return

ans =

Input k_0, x_minus and x_plus for middle interval of output equation

K>> k_0=1

k_0 =

    1

K>> x_minus=0.0

x_minus =

    0

K>> x_plus=0.6;
K>> return

ans =

Input thetaBias_0

K>> thetaBias_0=0

thetaBias_0 =

    0

K>> return

ans =

Input polynomialOrdersOutputLow and notUsedIndicesOutputLow for output equation

K>> polynomialOrdersOutputLow=[1 1]'

polynomialOrdersOutputLow =

1
1

K>> notUsedIndicesOutputLow=[0 1;1 0;1 1]

notUsedIndicesOutputLow =

0 1
1 0
1 1

K>> return

allIndicesOutputLow =

0 0

ans =

Input thetaOutputLow_0 - First parameter nonzero and remaining parameters zero gives initialization with line

K>> thetaOutputLow_0=[0.5]'

thetaOutputLow_0 =

0.5000

K>> return

ans =

Input polynomialOrdersOutputHigh and notUsedIndicesOutputHigh for output equation

K>> polynomialOrdersOutputHigh=[1 1]'

polynomialOrdersOutputHigh =

1
1

K>> notUsedIndicesOutputHigh=[]

notUsedIndicesOutputHigh =

[]

K>> return

allIndicesOutputHigh =

0 0
0 1
1 0
1 1

ans =

Input thetaOutputHigh_0 - First parameter nonzero and remaining parameters zero gives initialization with line

K>> thetaOutputHigh_0=[1 0 0 0]'

thetaOutputHigh_0 =

1
0
0

```
     0

K>> return

percentReady =

     1
.
.
.
percentReady =

   100


ans =

    0.3291    0.3291
    1.0035    1.0035
   -1.0006   -1.0006
    0.4936    0.4936
    0.9924    0.9924
   -0.0004   -0.0004
    1.0717    1.0717
   -0.1016   -0.1016
   -0.2589   -0.2589
    0.0875    0.0875.
```

This result is very close to the true parameters. The top parameter is the identified delay, the following three parameters are those of the identified ODE, thereafter follow the single parameter of the lower interval of the measurement equation, the bias parameter and finally the four parameters of the upper interval of the measurement equation. Note that the exact result depends on the generated input signal. This may differ between systems and execution occasions since the seed for the random number generator may differ. Hence, slight variations of the estimated parameters are normal.

12. Display of results

The display of results is straightforward. By a study of the source code, users should be able to tailor available scripts and also write own ones when needed.

12.1 Parameters

In order to plot the parameters the user is required to

1. Execute the script **PlotParametersDelayOutput.m**. The components of the parameter vector are then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

Example 20: The command in the MATLAB command window is

```
» PlotParametersDelayOutput
»
```

The following plot is then generated

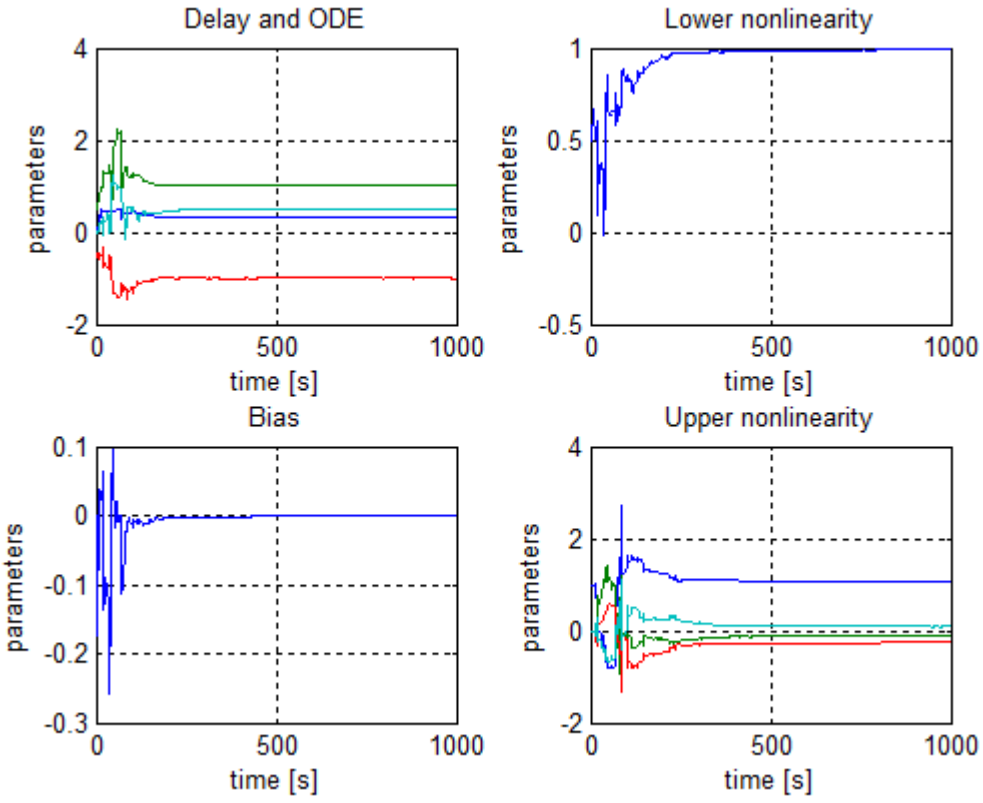


Figure 12: The result of a PlotParametersDelayOutput command.

12.2 Prediction errors

In order to plot the parameters the user is required to

1. Execute the script **PlotPredictionErrorsDelayOutput.m**. The prediction errors are then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

Example 21: The MATLAB command window command is

» PlotPredictionErrorsDelayOutput

»

The following plot is generated

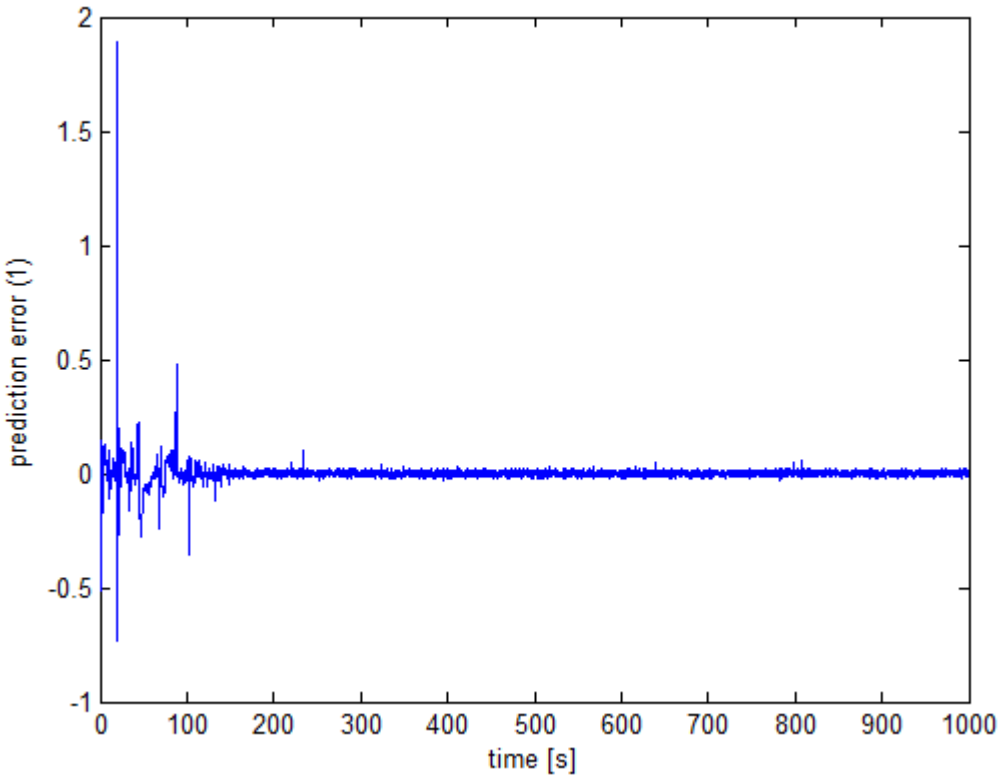


Figure 13: The result of a PlotPredictionErrorsDelayOutput command.

12.3 Eigenvalues

In order to plot the convergence of the eigenvalues over time, the user is required to

1. Execute the script **PlotEigenvaluesDelayOutput.m**. The eigenvalues of the Hessian are then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

Example 22: The MATLAB command window command is

» PlotEigenvaluesDelayOutput

»

The following plot is generated

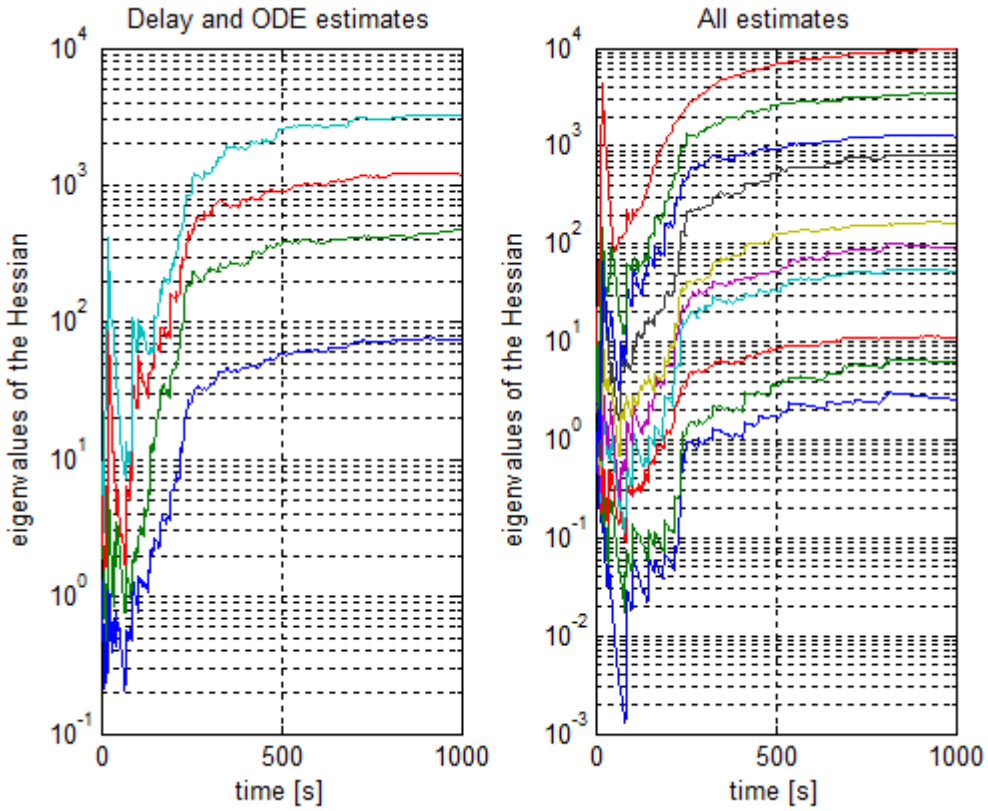


Figure 14: The result of a PlotEigenvaluesDelay command.

12.4 Condition number

In order to plot the convergence of the condition number over time, the user is required to

1. Execute the script **PlotConditionDelayOutput.m**. The condition number is then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

Example 23: The MATLAB command window command is

» PlotConditionDelayOutput

»

The following plot is generated

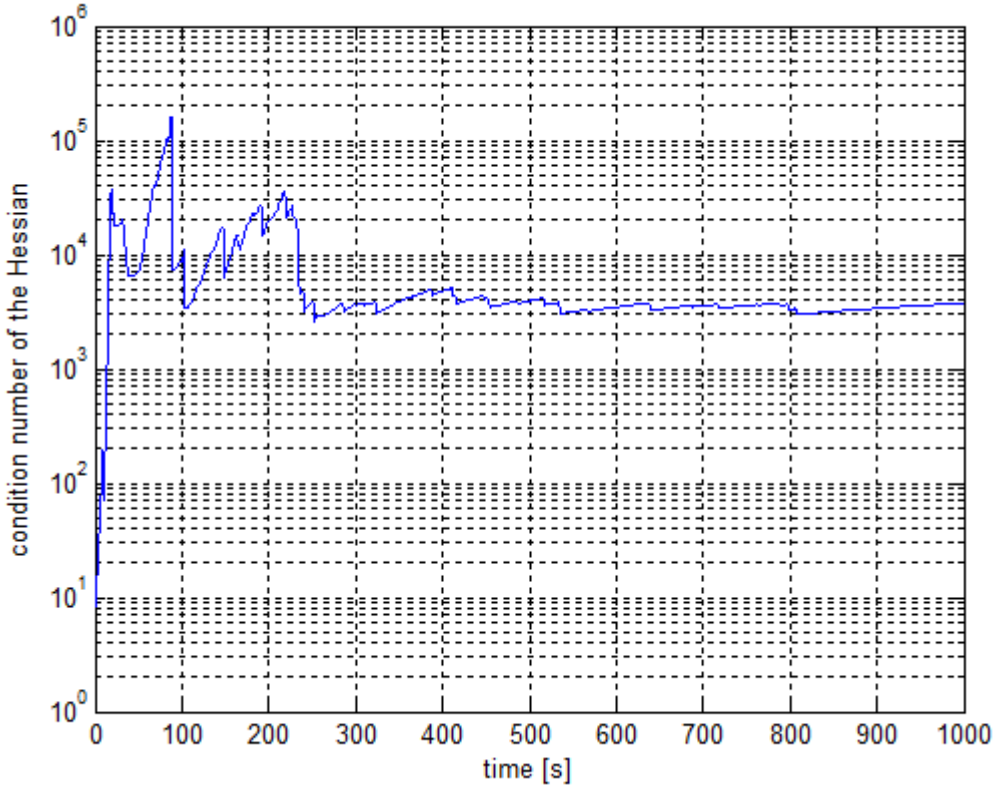


Figure 15: The result of a PlotConditionDelay command.

12.5 End of run model simulation

The above plot commands make use of logged signals, covering the transient part of the identification process. This is not always desired. To compare the identified model to the measured data it is e.g. more appropriate to simulate the model, using the parameters obtained at the end of the identification run.

Hence, to prepare for the remaining plot commands the user is required to

1. Execute the script **SimulateModelOutputDelayOutput.m**.

Example24: The MATLAB command window command is

```
» SimulateModelOutputDelayOutput
…
percentReady =
  100
»
```

The remaining plot commands and model validation commands can now be executed.

12.6 Simulated model output

In order to plot the simulated model output signal over time, the user is required to

1. Execute the script **PlotModelOutputDelayOutput.m**. The output signals of the model are then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

   Example25: The MATLAB command window command is

» PlotModelOutputDelayOutput

»

The following plot is generated

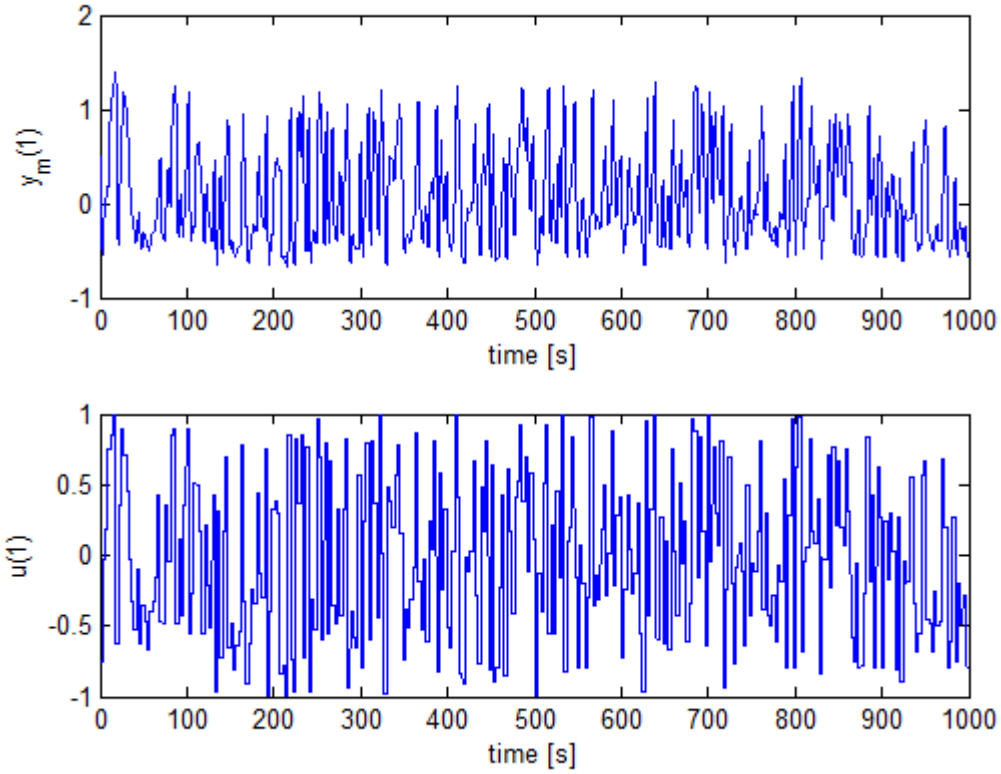


Figure 16: The result of a PlotModelOutputDelayOutput command.

12.7 Simulated model output together with data

In order to plot the simulated model output signal over time, together with the corresponding measured data, the user is required to

1. Execute the script **PlotSystemAndModelOutputDelayOutput.m**. The output signals of the model and the system are then plotted as a function of time, in the same plots. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

   Example26: The MATLAB command window command is

» PlotSystemAndModelOutputDelayOutput

»

The following plot is generated

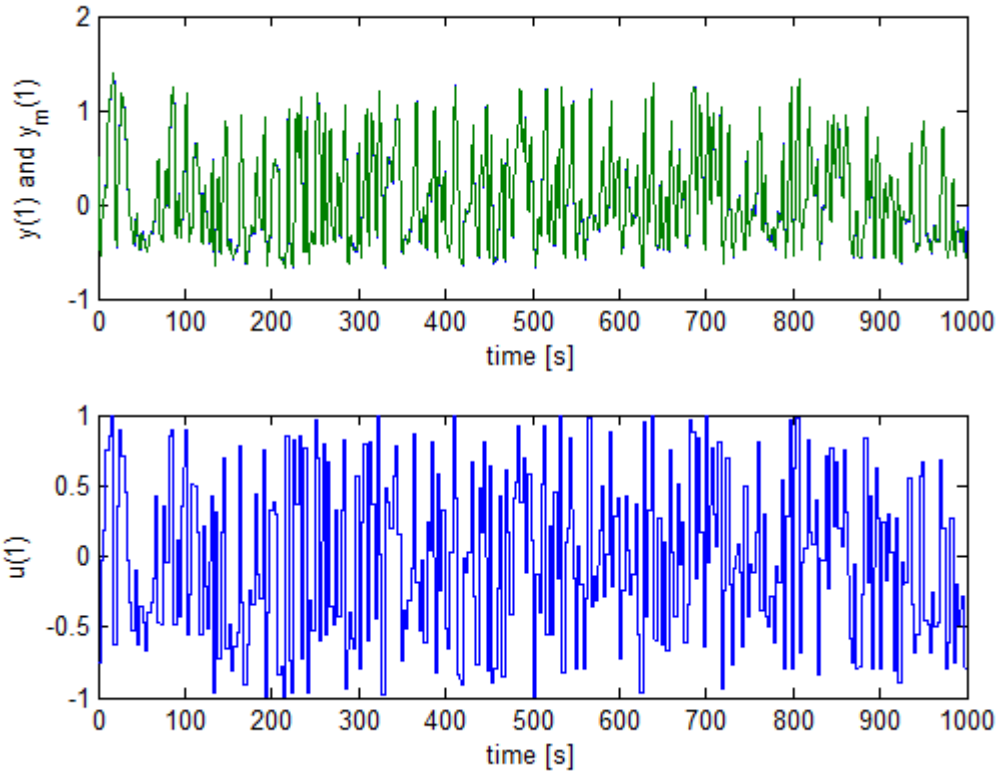


Figure 17: The result of a PlotSystemAndModelOutputDelayOutput command.

12.8 Residual errors

In order to plot the prediction errors, obtained from the simulated model output signal using parameters at the end of the identification run, the user is required to

1. Execute the script **PlotResidualErrorsDelayOutput.m**. The residual errors are then plotted as a function of time. Note that the time scale is assumed to be seconds. In case another time scale is required, the figure needs to be edited after plotting, or the script needs modification.

Example 27: The MATLAB command window command is

```
» PlotResidualErrorsDelayOutput
»
```

The following plot is generated

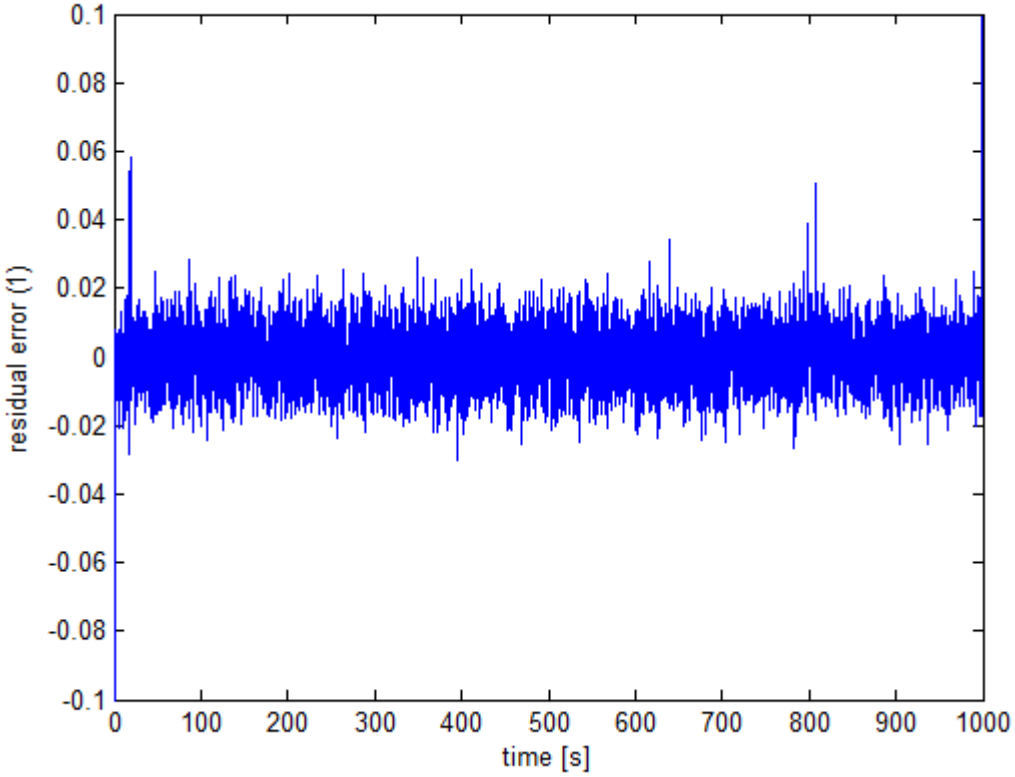


Figure 18: The result of a PlotResidualErrorsDelayOutput command.

13. Model validation

Two scripts are provided for model validation purposes (in addition to the commands of section 12). The first method studies the value of the loss function that is minimized by the RPEM-algorithm. Since the measurement covariance matrix is estimated, the loss function contains an additional term on top of the sample average of the squared prediction errors.

13.1 RPEM loss function

Exactly as in section 7, the RPEM loss function that is computed is given by

$$V\left(\hat{\boldsymbol{\theta}}\left(t_0+NT_S\right)\right)=\frac{1}{2}\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\varepsilon}\left(t_0+iT_S,\hat{\boldsymbol{\theta}}\left(t_0+NT_S\right)\right)\boldsymbol{\varepsilon}^T\left(t_0+iT_S,\hat{\boldsymbol{\theta}}\left(t_0+NT_S\right)\right)+\frac{1}{2}\log\det\left(\hat{\boldsymbol{\Lambda}}\left(t_0+NT_S\right)\right) \qquad (20)$$

In order to compute the loss function, using parameters at the end of the identification run, the user is required to

1. Execute the script **ComputRPEMLossFunctionDelayOutput.m**. The loss function is then computed.

Example 28: The MATLAB command window command is

» ComputeRPEMLossFunctionDelayOutput

…

percentReady =

100

V =

-3.6468

>>

Note: Another relevant measure to use is the sum of the squared prediction errors.

13.2 Mean residual analysis

Mean residual analysis is a method that evaluates the obtained static characteristics of an identified model of any kind. It operates by sorting residual errors into bins, the bin being decided by the value of the measured output signal with the same time index as the residual error. The mean of the residuals are then computed, in each bin, and plotted against the range of the output signal. The number of samples of each bin is also plotted. The user is referred to [9] for further details. In order to perform mean residual analysis, the user is required to

1. Execute the script **MeanResidualAnalysisDelay.m**. This is the same one as used in Section 7.
2. Provide the intervals used by the method when prompted for ***intervals***.

Example 29: This example performs mean residual analysis using about 40 intervals, each with an output amplitude width of 0.1. The MATLAB command window command is

» meanResidualAnalysisDelay

ans =

Input intervals for division into bins

K» intervals=(-2:0.1:2);

K» return

»

The following plot results

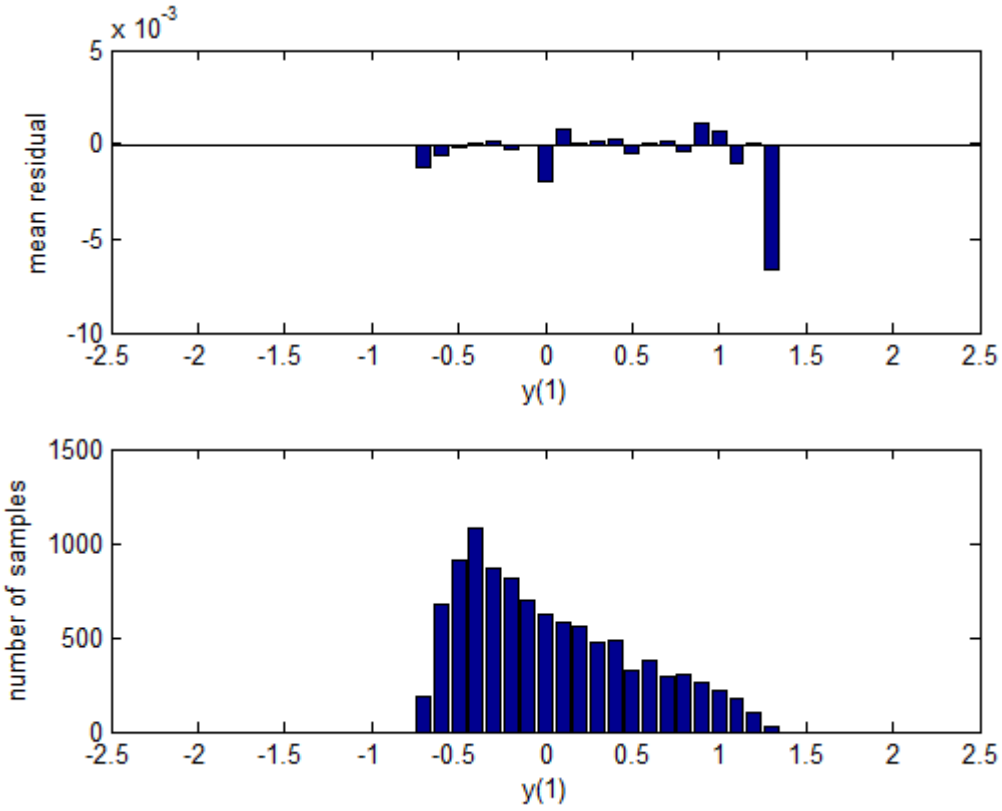


Figure 19: The result of a MeanResidualAnalysisDelay command.

14. Summary

This report describes a software package for identification of nonlinear systems. Future work in this field, that result in useful MATLAB routines, will be integrated with the presently available functionality. Updated versions of this report will then be made available.

References

[1] T. Wigren, "Recursive Prediction Error Identification of Nonlinear State Space Models", Technical Reports from the department of Information Technology 004-2004, Uppsala University, Uppsala, Sweden, January, 2004.

[2] T. Wigren, "Recursive prediction error identification and scaling of nonlinear state space models using a restricted black box parameterization", Automatica, vol. 42, no. 1, pp. 159-168, 2006.

[3] T. Wigren "Scaling of the sampling period in nonlinear system identification", in Proceedings of IEEE ACC 2005, Portland, Oregon, U.S.A., pp. 5058-5065, June 8-10, 2005.

[4] T . Wigren "Recursive identification based on nonlinear state space models applied to drum-boiler dynamics with nonlinear output equations", in Proceedings of IEEE ACC 2005, Portland, Oregon, U.S.A., pp. 5066-5072, June 8-10, 2005.

[5] T. Wigren, "Networked and delayed recursive identification of nonlinear systems", submitted, 2017.

[6] T. Wigren, "MATLAB software for recursive identification and scaling using a structured nonlinear black-box model - Revision 7", to appear in Technical Reports from the department of Information Technology, Uppsala University, Uppsala, Sweden.

[7] D. Hanselmann and B. Littlefield. *Mastering Matlab 5 - A Comprehensive Tutorial and Reference.* Upper Saddle River, NJ: Prentice Hall, 1998.

[8] L. Ljung and T. Söderström. *Theory and Practice of Recursive Identification.* Cambridge, Ma: MIT Press, 1983.

[9] T. Wigren, "User choices and model validation in system identification using nonlinear Wiener models", Prep. 13:th IFAC Symposium on System Identification, Rotterdam, The Netherlands, pp. 863-868, August 27-29, 2003. Invited session paper.

[10] T. Wigren, “MATLAB software for nonlinear and delayed recursive identification – Revision 1”, Technical Report from the department of Information Technology 2017-007, Uppsala University, Uppsala, Sweden, April, 2017.

[11] T. Wigren, “Recursive prediction error identification using the nonlinear Wiener model”, *Automatica*, vol. 29, no. 4, pp. 1011-1025, 1993.