An integrated System Identification toolbox for linear and non-linear models

Lennart Ljung, Qinghua Zhang, Peter Lindskog, Anatoli Iouditski, Rajiv Singh

Division of Automatic Control
E-mail: ljung@isy.liu.se, zhang@irisa.fr,
peter.lindskog@niradynamics.se,
Anatoli.Iouditski@inrialpes.fr,
rajiv.singh@mathworks.com

14th June 2007

Report no.: LiTH-ISY-R-2786
Accepted for publication in Proc 14th IFAC Symposium on System Identification, Newcastle, Australia

Address:
Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

WWW: http://www.control.isy.liu.se

Technical reports from the Automatic Control group in Linköping are available from http://www.control.isy.liu.se/publications.
Abstract
The paper describes additions to the MATLAB System Identification Toolbox, that handle also the estimation of nonlinear models. Both structured grey-box models and general, flexible black-box models are covered. The idea is that the look and feel of the syntax, and the graphical user interface should be as close as possible to the linear case.

Keywords: identification, software
Abstract

The paper describes additions to the MATLAB System Identification Toolbox, that handle also the estimation of nonlinear models. Both structured grey-box models and general, flexible black-box models are covered. The idea is that the look and feel of the syntax, and the graphical user interface should be as close as possible to the linear case.

Keywords: nonlinear system identification, neural networks, nonlinear models

1. INTRODUCTION

The next version of the MathWorks System Identification Toolbox (SITB), (Ljung, 2003) will integrate techniques for non-linear and linear models. The basic idea is to do this integration in a transparent manner, so that the rather complex problem of estimating and analyzing non-linear models will appear simple and natural. It is also desirable that it can be done with a syntax which has the same look and feel as for linear models. In this paper it is assumed that the reader is already familiar with the syntax for linear models of the SITB which is described in (Ljung, 2003).

The non-linear model structures supported by the toolbox are

- **idnlgrey:** grey-box models corresponding to arbitrary non-linear state-space equations in continuous or discrete time. The user supplies code in the form of MATLAB m-file or C-mex file that defines the right hand side of these state-space equations.
- **idnlarx:** non-linear ARX models: system output is modeled as a nonlinear regression of past inputs and past outputs.
- **idnlhw:** these are non-linear block-oriented models of Hammerstein-Wiener type.

Like the linear models of the SITB, each nonlinear model is implemented as a MATLAB object. A model thus has a certain number of properties, like all MATLAB objects. Once a model is created, its properties can be accessed by the com-
mands get/set following the standard syntax. Property/Value pairs can also be used in other method functions. For example, when a nonlinear model object \( m \) has been created, the estimation syntax is the same as for linear models:

\[
m = \text{pem}(\text{data}, m, 0, P_1, V_1, \ldots, P_n, V_n)
\]

where \( P_k, V_k \) are optional Property/Value pairs, and \( \text{data} \) is an \text{iddata} object.

The quality of the estimated models \( m_1, m_2, \ldots, m_n \) can also be evaluated similarly to the linear case by commands like

\[
\text{compare}(\text{data}, m_1, m_2, \ldots, m_n)
\]

\[
\text{resid}(\text{data}, m_1)
\]

\[
\text{sim}(\text{data}, m_1)
\]

\[
\text{predict}(\text{data}, m_1)
\]

for model simulation error comparison, residual analysis, simulation and prediction, respectively. The nonlinearities in estimated models are basically plotted by

\[
\text{plot}(m_1, m_2, \ldots, m_n)
\]

For an \text{idnlgrey} model, the model structure must be specified in a m-file or C-mex file before being estimated with data.

For \text{idnlarx} and \text{idnlhw} models, a simple command can be used to specify a model structure and to estimate the model with data. For example:

\[
m = \text{nlarx}(\text{data}, [2 2 1], 'sigmoidnet')
\]

\[
m = \text{nlhw}(\text{data}, [2 2 1], \ldots, 'OutputNonLinearity', 'wavenet')
\]

where ‘sigmoidnet’ and ‘wavenet’ indicate the types of nonlinearity estimators.

2. NON-LINEAR GREY-BOX MODELS

\text{idnlgrey} is a complement to the linear grey-box modeling framework provided by \text{idgrey} of SITB. Using \text{idnlgrey} it is possible to estimate unknown parameters in model structures written in explicit state-space form of output error type:

\[
x'(t) = f(x(t), u(t), \theta)
\]

\[
y(t) = h(x(t), u(t), \theta) + e(t)
\]

where the dynamic equations are either time-continuous \((x'(t) = \frac{d}{dt}x(t))\) or time-discrete \((x'(t) = x(t + T))\), \( f(\cdot) \) and \( h(\cdot) \) are here nonlinear functions of the states \( x(t) \), the inputs \( u(t) \) (optionally), the parameters \( \theta \) and, optionally, of the time \( t \).

Given measurements of the output(s) \( y(t) \) and input(s) \( u(t) \), the unknown parameters \( \theta \) can be estimated using several different approaches, including Gauss-Newton and Levenberg-Marquardt methods. For time-continuous systems, the user can choose any of the 10 Ordinary Differential Equations’ (ODE) solvers currently supported within \text{SIMULINK}. This ODE suit involves standard as well as stiff ODE solvers, fixed as well as adaptive step length solvers, and so forth.

The first step in \text{idnlgrey} modeling is to specify the model structure, or rather the state \( F = f(\cdot) \) and the output equations \( H = h(\cdot) \). This information is either specified in an m-file or (preferably, due to the improved computational speed) in a C-mex-file with the following calling syntax

\[
[F, H] = \text{ModelFile}(t, x, p, u);
\]

In order to facilitate this model structure entering phase, templates and special means for writing C-mex-files are available.

The next step is to construct an \text{idnlgrey} object:

\[
m = \text{idnlgrey}('ModelFile', \text{ParameterVector}, [nx ny nu]);
\]

\text{ParameterVector} is here the initial guessed values of the parameters and \([nx ny nu]\) a vector specifying the number of states, outputs and inputs of the model structure.

With this information entered, simulation and prediction of the model given input data can be done using \text{sim} and \text{predict}, respectively. Parameter estimation is carried out through \text{pem}.

3. NON-LINEAR BLACK-BOX MODELS

Compared to grey-box models, black-box models are built with much less physical knowledge about the underlying system. A general introduction to non-linear black-box models can be found in (Sjöberg et al., 1995; Juditsky et al., 1995).

In the toolbox, each non-linear black-box model is specified in two aspects: the dynamics structure and the nonlinearity estimators. If the choice of dynamics structures may sometimes by guided by a priori knowledge, the choice of nonlinearity estimators is very often arbitrary and needs trials before getting a satisfactory result.

Two dynamics structures are implemented in the forthcoming version of SITB: \text{idnlarx} and \text{idnlhw}.

The available nonlinearity estimators in this version are \text{tree}, \text{wavenet}, \text{sigmoidnet}, \text{pwlinear}, and user-defined simple neural networks. It is also possible to use general multiple-layer neural networks based on the Matlab Neural Networks Toolbox.
4. THE IDNLARX DYNAMICS STRUCTURE

If we denote the input-output data by
\[ Z^t = \{ y(1), u(1), \ldots, y(t), u(t) \} \]
an idnlarx model has the structure
\[ \hat{y}(t) = F(x(t)), \quad x(t) = h(Z^{t-1}) \]
where \( x(t) \) are the regressors, which in principle can be arbitrary functions of past measured input-output data, \( F \) is the non-linear regression function approximated by one of the nonlinearity estimators which will be introduced later, and \( \hat{y}(t) \) is the model output corresponding to a prediction of the system output at time instant \( t \).

The toolbox handles such models within the idnlarx model object. The model can be created with the constructor idnlarx or directly estimated from data by the command nlarx, as in
\[ m = nlarx(data, \text{arg1, arg2, \ldots}) \]
where the arguments define the particular choice of non-linear structure.

To completely define such a structure, three things must be specified:

- the regressors: how \( x(t) \), depends on \( Z^{t-1} \), i.e. the function \( h(Z^{t-1}) \).
- the non-linear structure: How \( x(t) \) enters in the function \( F \).
- the non-linearity: The character of the function \( F \).

Each of these items is associated with object properties, which we now discuss.

4.1 The Regressors

To define the regressors, i.e. the function \( h(Z^{t-1}) \) two possibilities are offered in the toolbox:

- A standard set of delayed outputs and inputs
  \[ y(t - np), y(t - np - 1), \ldots, y(t - np - na + 1) \]
  \[ u(t - nk), u(t - nk - 1), \ldots, u(t - nk - nb + 1) \]
  is described by the properties ‘na’, ‘nb’, ‘nk’, ‘np’ as in
  \[ m = nlarx(data, \text{'na'}, 2, \text{'nb'}, 3, \ldots \text{'nk'}, 4, \text{'np'}, 5, \ldots) \]
  or
  \[ m = nlarx(data, [2 3 4 5], \ldots) \]
  This describes a delay of \( nk \) from the input and prepares for direct estimation of \( np \)-step ahead predictors. For multi-input, multi-output systems \( na, nb, etc. \) are matrices of appropriate dimensions.
- Custom regressors: any function of past input-output data defined by the user as arbitrary MATLAB expressions. The corresponding property is called CustomRegressor and are defined by function handles or cell arrays of strings as in
  \[ m = nlarx(data, \text{CustomRegressor'}, \ldots \{ \text{'u1(t-1)^3'}, \text{sin(y1(t-3))^2'}, \ldots \text{abs(y1(t-3)*u3(t)')}, \ldots \} \]
Custom regressors give a powerful way to incorporate a priori knowledge, making the black-box model less “dark”. For example, if \( u1 \) and \( u2 \) are voltage and current measurements and it is known that the electrical power may play an important role in the system, then it is wise to define ‘\( u1(t-1)*u2(t-1) \)’ as a custom regressor. Of course, standard and custom regressors can co-exist in a model.

4.2 The Structure of the Nonlinearity

By the structure of the nonlinearity is meant how \( x \) enters \( F \): The function \( F \) may be linear in some components of \( x \), and non-linear in others. The property LinearRegressor lists those regressors in \( x \) (by name or number) that enter \( F \) linearly. Similarly the property NonLinearRegressor lists the regressors that enter \( F \) nonlinearly (as a function described below). If the sets or regressors are denoted by \( x_\ell \) and \( x_n \), respectively, we have formally
\[ F(x) = f(x_n) + L^T x_\ell \quad (1) \]
where \( f \) is a nonlinear function defined below and \( L \) is a vector describing how the linear regressors affect \( F \).

The nonlinear and the linear regressors can be defined independently of each other. If \( x = x_n = x_\ell, \) – which is the default choice – (1) means that we have a parallel linear model. If some regressors are part neither of \( x_n \) nor \( x_\ell \), the model simply does not depend on these.

It is important to appropriately choose linear and non-linear regressors in a model, so that only significant non-linear regressors are incorporated in the estimation of the non-linear function \( f(x_n) \). When the NonLinearRegressor property is set to ‘auto’, an exhaustive search is made to choose the best non-linear regressors.

4.3 The Nonlinearity

The final step to describe the nonlinear dynamic model is to specify the function \( F \).

A common structure of \( F \) is
\[ F(x) = \sum_{k=1}^d \alpha_k \kappa(\beta_k(x - \gamma_k)) \quad (2) \]
where \( \kappa \) is the unit non-linear function and \( \alpha_k, \beta_k, \gamma_k \) are parameters of the estimator. The
values of these parameters are typically estimated by the \texttt{pem} or \texttt{nlarx} command.

The toolbox offers the following choices, which all basically correspond to values of the property \texttt{NonLinearity}.

- **Tree**: a binary partition tree. During the estimation procedure, the data points are split into two subsets, and each subset is recursively split into smaller subsets. The entire data set is thus partitioned into a binary tree. For each data subset (at different levels of the binary tree), a linear regression is computed. When a new input value is presented to the tree estimator, it chooses the best local linear regression in the binary tree in an adaptive manner to predict the corresponding output value.

- **WaveNet**: a radial basis function network based on wavelets. The WaveNet estimator is similar to classical radial basis function (RBF) network, but the radial basis functions are multi-scale wavelets and scaling functions. This is an example of the structure (2), with $\kappa$ being either a wavelet function or a scaling function. The advantage of using wavelets is their multi-scale structure and their good numerical conditioning (though they do not form an orthogonal basis, they are far from linearly dependent). The estimation method is based on the orthogonal least square (OLS) algorithm, optionally refined by gradient based optimization. More technical details can be found in (Zhang, 1997).

- **SigmoidNet**: a single hidden layer neural network with sigmoid neurons. This is a well known nonlinearity estimator. The estimation method applies gradient based optimization after a random parameter initialization. This is also an example of (2), with $\kappa$ being the sigmoid $1/(1 + e^{-x})$ and the argument interpreted as a scalar product.

- **PWlinear**: piecewise linear estimator. The nonlinearity is approximated piece-wisely by local linear estimators. The connection between local linear estimators is continuous. It is implemented and estimated similarly to the sigmoidnet estimator, but the sigmoid neurons are replaced by piecewise linear functions.

- **LinearRegressors**: a linear combination of regressors. This estimator is useful for \texttt{idnlarx} models with non-linear custom regressors. In this case the nonlinearity resides in the definition of the custom regressors.

This is the same as setting \texttt{NonLinearRegressor=[]}.

- **MLnet**: A multi-layer neural network from the MATLAB Neural Network Toolbox (NNTB) (Demuth et al., 2005). This is an interface to the NNTB for creating general multi-layer neural networks. The NNTB license is required.

Estimators for saturation and deadzone nonlinearities are also provided. They are implemented as special cases of the PWlinear estimator. Typically they are used in \texttt{idnlhw} models. See below.

The user can also easily create new simple neural network estimators, similar to the SigmoidNet estimator. For this purpose, the user needs to create an m-file specifying the network unit function (defining the form of the neurons, i.e. the function $\kappa$ in (2), also known as activation function) and its derivative. For example, if a unit function named \texttt{myunit} is created, then it can be used with the syntax

\begin{verbatim}
m = nlarx(data,[na nb nk np], ...
    'custom','unitfcn','myunit')
\end{verbatim}

In all cases, the flexibility of the function is governed by the property \texttt{NumberOfUnits}, which corresponds to the number $d$ in (2), when applicable. For trees it corresponds to the number of nodes in the binary tree. When not specified, this flexibility number is given a default value.

4.4 Putting it All Together

A model of a particular structure can now be estimated from data, by simply giving all the properties as arguments in \texttt{nlarx}. The third argument always defines the nonlinearity, as defined in Section 4.3. The other properties appear in Property/Value pairs. For short they can be given by any unambiguous abbreviation (also interpreting 'nl' as 'nonlinear'). Properties not specified will be given default values. For example,

\begin{verbatim}
m = nlarx(z,[2 2 1],',sigm',,'cust',... ('y1(t-1)^2'),,'nlreg',,[1:4])
\end{verbatim}

will give a model with the structure depicted below. (obtained by \texttt{spresent(m).})
5. THE IDNLHW DYNAMICS STRUCTURE

An idnlhw model consists of a linear dynamic block connected at its input and/or output with static nonlinearity blocks. If the non-linear block is present only at the input (or output), the model is known as of Hammerstein (or Wiener) type. If both the input and output are connected with non-linear blocks, then the model is said to be of Hammerstein-Wiener type.

In the toolbox the linear block is described by a discrete time transfer function, whereas the non-linear blocks are implemented by nonlinearity estimators sigmoidnet, pwlinear, wavenet or simple nonlinearities like saturation and deadzone.

In the multiple input and/or multiple output case, the non-linear blocks are assumed to be “diagonal”, i.e., the nonlinearity connected to each input of the dynamic linear system is defined as a scalar non-linear function, independent of the other inputs. The situation is similar for the output nonlinear blocks.

The syntax for directly specifying an idnlhw model structure and for its estimation with data is illustrated by the example

\[
m = \\text{nlinhw(data,[3 2 1], ... ~\text{'InputNonLinearity','sigmoidnet', ... ~\text{'NumberOfUnits', 15, ... ~\text{'OutputNonLinearity','saturation'})}
\]

The second input argument, [2 3 1], specifies the order of the transfer function of the linear block: \(nb=2\), \(nf=3\), \(nk=1\), which is similar to the specification of a linear Output Error (OE) model. The input and output nonlinearity estimators are then specified in the following arguments. Property/Value pairs can be used after the specification of the type of a nonlinearity estimator to specify the properties of the estimator, like \(\text{NumberOfUnits}=15\) in this example.

For simplicity, 'u' and 'y' are synonymous to 'input' and 'output', so the command can also be written

\[
m = \\text{nlinhw(data,[3 2 1],'unl','sig',... ~\text{'num',15,'ynl','sat'})
\]

6. SOME IMPORTANT MODEL PROPERTIES

The property Parameter gives the values of the parameters in the model. The goal of the estimation is of course to find suitable values of these. The syntax will thus be of the kind

\[
m = \text{nlarx(data,[na nb nk np], ... ~\text{'sigmoid','numberofunits',25})}
\]

The details of the actual model are then retrieved by

\[
m\text{.parameter}
\]

which will be a structure containing all relevant information about the model.

Just typing the model name \(m\) gives a concise description of the model, while \(\text{present}(m)\) gives more details. \(\text{spresent}(m)\) gives a graphical presentation of the model structure as in the figure in section 4. Moreover, \(\text{plot}(m)\) gives a plot of the nonlinearity/nonlinearities of the model.

7. GRAPHICAL INTERFACES

The estimation and evaluation commands are also enclosed in a graphical user interface (GUI), which builds on the one for linear models. This means that linear and non-linear models from the same data can readily be compared.

A special challenge is to design a user-friendly GUI for choosing all the options that are inherent in non-linear modeling, and also provide data-dependent guidance for such choices.

8. EXAMPLE

In this example we load data from a hydraulic crane (forest machine to lift logs). The input is the pressure in the cylinder and the output is the position of the crane tip. First a linear model is estimated based on the first half of the data. The simulated output is then compared to the measured output for the whole data record. After that a Hammerstein model is tried out, with a sigmoidal neural net with 10 neurons as the input, static nonlinearity The third model is an nlarx model with a Wavenet nonlinearity affecting only the past inputs (the past outputs entering linearly).

The figures show the measured output as a thick line, and the model simulated output as a thin line. They also show the fit of the model (in terms of the percentage of the measured output variation that is reproduced by the model). Clearly, the Hammerstein model gives the best performance in this case. The physical explanation for this could be that the measured hydraulic cylinder pressure is transformed to forces acting on the mechanical parts by a non-linear static function, where-after the dynamics is described by linear mechanical motion equations.

\[
\text{load robotarm}
\]
\[
data = \text{iddata(y,u)};
\]
\[
date=data(1:512);
\]
\[
datv=data(513:end);
\]
ml = arx(date,[3 2 1]);
compare(data,ml);

mh=nlhw(date,[2 3 1],'unon','sig');
compare(data,mh);

mw=nlarx(date,[3 2 1],'wave','nlreg',[4 5]);
compare(data,mw);

9. CONCLUSIONS

There are several challenges involved in constructing a user-friendly toolbox for estimating and evaluating nonlinear models. The first one is of course to have good and reliable algorithms. The second one is to have sufficiently many options in the choices that give reasonable freedom and diversity in the toolbox. At the same time the number of choices must not be overwhelming, so that the user is stalled. This requires a gentle balance in syntax, graphical interfaces and the use of proper defaults.

The third challenge is equally important: To make the user understand that nonlinear identification is difficult. In the linear case, it may be fair to say that the linear toolbox can guide even an inexperienced user to find a reasonable model, if there is a good linear model available. This is not true, unfortunately, for non-linear models. The world of such models is very rich and somewhat complicated. Regardless of syntax and GUIs, it will require a fair amount of understanding by a user to navigate in this world.

REFERENCES


### Abstract

The paper describes additions to the *Matlab* System Identification Toolbox, that handle also the estimation of nonlinear models. Both structured grey-box models and general, flexible black-box models are covered. The idea is that the look and feel of the syntax, and the graphical user interface should be as close as possible to the linear case.

### Keywords
- identification
- software