# A MANUAL FOR SYSTEM IDENTIFICATION

**Lennart Andersson, Ulf Jönsson, Karl Henrik Johansson, and Johan Bengtsson**

## 0. Introduction

The main purpose of this manual is to describe how to use the Matlab System Identification Toolbox. The most important toolbox commands are described briefly and in the order they are normally executed during a system identification session. See the toolbox manual and the Matlab built-in help function for further information about the commands. With a web browser (for example, Firefox) it is possible to read the manual at

`http://www.mathworks.com/access/helpdesk/help/toolbox/ident/`

Figure 1 shows an algorithm for modeling and system identification. The presentation in this manual follows this algorithm. System identification is an iterative process and it is often necessary to go back and repeat earlier steps. This is illustrated with arrows in the figure. Notice that the order of the blocks in the algorithm does not only describe the chronological order the tasks are performed, but also how they influence each other. For example, a certain model structure can be proposed by the physical model, the amount of data limits the model complexity etc. Support for control design is often the purpose for system identification in this course, as illustrated by the last block in Figure 1.

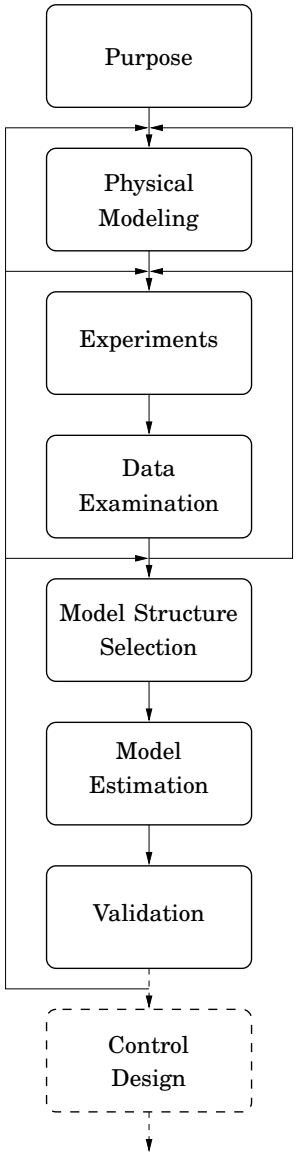


**Figure 1** Algorithm for modeling and system identification.

## 1. Purpose

It is important to state the purpose of the model as a first step in the system identification procedure. There are a huge variety of model applications, for example, the model could be used for control, prediction, signal processing, error detection or simulation. The purpose of the model affects the choice of identification methods and the experimental conditions, and it should therefore be clearly stated. It is, for example, important to have an accurate model around the desired crossover frequency, if the model is used for control design.

## 2. Physical Modeling

It is sometimes possible to derive a model directly from physical laws. This model will most often, however, contain unknown parameters to be estimated. If some parameters are known and some are unknown, it is sometimes (but not always) possible to perform an estimation using the values of the known parameters. We commonly use only the structure of the model derived from the physical model. This structure can be

in terms of model order, known pole locations (an integrator or dominating resonance), static nonlinearities etc. If there are no knowledge about the considered system, we use the term *black-box identification*. It is called *grey-box identification*, if some part of the system is known.

## 3. Experiments

The experiments are done in two steps. In the first step, preliminary experiments such as impulse and step responses are performed to gain primary knowledge about important system characteristics such as stationary gain, time delay and dominating time constants. It should be possible to draw conclusions from these experiments on whether or not the system is linear and time invariant and if there are disturbances acting on the system. The information obtained from the preliminary experiments are then used to determine suitable experimental conditions for the main experiments, which will give the data to be used in the System Identification Toolbox.

### 3.1 Preliminary experiments

Some system characteristics simple preliminary experiments are discussed in this subsection.

***Time-invariance*** Most identification methods assume time-invariant models. Measurements or experiments at different times indicate if the system is time invariant or not. A system may appear to be time varying even when this is not the case. The reason is that slow dynamics may appear as a time variation when considered on a short time scale. If the purpose is to identify fast dynamics, then the slow dynamics can be removed by trend elimination. Time variations, which do not depend on slow dynamics may be identified by use of recursive algorithms.

***Linearity*** The System Identification Toolbox contains routines only for identifying linear systems. Linearity can be checked by

- investigating system responses for various input signal amplitudes,
- examining if the responses to a square wave are symmetric, and
- deriving the coherence spectrum.

Most real systems are nonlinear. A linearized model can then be obtained from data if the input signal has a small amplitude. Sometimes it is possible by a proper choice of state-transformation to rewrite a nonlinear system into a linear, see Chapter 5 in Johansson (1993).

A common nonlinearity is friction. The influence of friction can be reduced or eliminated by performing the identification experiment with a bias on the input. For example, a DC motor model should be identified using a bias such that the motor runs in only one direction during the experiment. Furthermore, large input signals (contrary to the linearization prerequisite above) can reduce the problems with friction.

***Transient response analysis*** Step- and impulse-response analysis give information on dominating time constant, time delay, and stationary gain. It is also possible to recognize non-minimum phase properties of the system from these experiments. An indication of the disturbances acting on the system may also be obtained.

***Frequency response analysis*** Frequency response analysis according to the correlation method in Chapter 2 of Johansson (1993) gives an estimated transfer function for the system. The obtained Bode plot gives information on location of poles and zeros, stationary gain, and time delay.

### 3.2 Main experiments

This subsection treats the main experiments where data are collected to be used in the System Identification Toolbox. In particular, the choice of input signal is discussed.

The identification gives an accurate model at the frequencies where the input signal contains much energy. We say that the input signal has good excitation at these frequencies. The frequency content of the input should therefore be concentrated to frequencies where small estimation errors are desired. A pseudo-random binary sequence (PRBS) is a common choice of input signal, since it has a large energy content in a large frequency range. In the course projects, we use the program `logger` for generation of PRBS, see Gustafsson (1989). The PRBS signal in `logger` is defined in terms of

- sampling interval (`h`),
- number of sampling intervals between PRBS shift register updates (`M`),
- number of data points collected (`N`), and
- amplitude of the excitation signal (`A`).

A rule of thumb is to let $1/(\mathtt{h} \cdot \mathtt{M})$ be equal to the bandwidth of the system. To avoid aliasing, `M` should be at least 2–10. The closed loop system step response should be sampled 5-10 times during it's rise time. The experiment duration should be chosen to get good parameter estimates. A rule of thumb is to make it

5-10 times longer than the longest interesting time constant.

The data can be divided into three parts

- data with transients
- identification data
- validation data

The amplitude should be chosen as large as possible in order to achieve a good signal-to-noise ratio and to overcome problems with friction. However, the amplitude may not be chosen larger than the range in which the linearity assumption holds. (See the section on preliminary experiments above.) Typically saturations give an upper bound on the amplitude of the input signal. The mean value is in many cases non-zero in order to reduce friction problems or to give a linearized model around a stationary point with $u^0 \neq 0$.

## 4. Data Examination

Assume that an experiment has been performed and that we have an input sequence and an output sequence represented as column vectors `u` and `y`, respectively, in Matlab. It is suitable to split the data into two sets, one for identification and one for validation:

```
>> zi=iddata(y(1:N1),u(1:N1),h);
>> zv=iddata(y(N1+1:N),u(N1+1:N),h);
```

Start by checking the data manually via plots. Look for

- outliers,
- aliasing effects, and
- trends and non-stationarity.

Outliers should simply be removed from the data series.

If there are aliasing effects in the data, the experiment set-up has to be changed: either the sampling rate should be increased or an appropriate anti-aliasing filter should be used.

Linear trends in data should be removed. This is done by using the command `detrend`. A non-zero mean value is removed by the Matlab command

```
>> zi=detrend(zi,'constant');
```

and a linear trend are removed by the command

```
>> zi=detrend(zi);
```

The trends should, of course, also be removed from validation data `zv`.

| | |
|---|---|
| `plot` | Display input–output data |
| `detrend` | Remove trends from data |

### 4.1 Excitation

The input should provide good excitation in the frequency range where the model need to be accurate. We can check if the input excites the system appropriately by studying

- the autospectrum of `u` and `y`,
- the coherence spectrum, and
- the conditions for persistent excitation.

It is possible to concentrate the identification to interesting frequency ranges by filtering `u` and `y` as explained below. If the excitation is insufficient after filtering then the experiments must be repeated with new experimental conditions. The Matlab function `spectrum` can be used to plot the autospectrum and the coherence function as follows

```
>> yi=zi.OutputData; ui=zi.InputData;
>> spectrum(ui,yi)
```

With an output argument

```
>> S=spectrum(ui,yi);
```

the following quantities are derived:

```
S=[Suu Syy Suy Tyu Gamuy Suuc Syc Suyc]
  Suu=u-vector power spectral density
  Syy=y-vector power spectral density
  Suy=Cross spectral density
  Tuy=Complex transfer function Suy./Suu
  Gamuy=Coherence function
     (abs(Suy).^2)./(Suu.*Syy)
  Suuc,Syyc,Suyc=Confidence ranges
```

Some of these can also be derived separately, as shown in the box below.

| | |
|---|---|
| `psd` | Power spectral density |
| `csd` | Cross spectral density |
| `cohere` | Coherence function estimate |
| `tfe` | Transfer function estimate |
| `spectrum` | `psd`, `csd`, `cohere`, `tfe` combined |

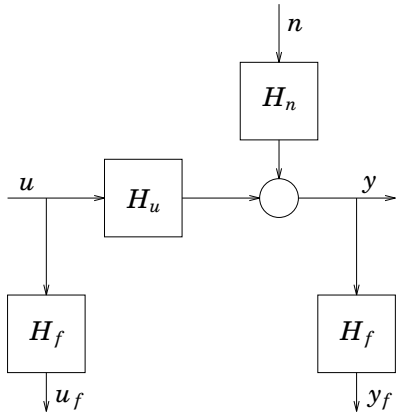


**Figure 2** Filtering of data affects the noise model

### 4.2 Filtering

It is often a good idea to filter the input signals with a lowpass or bandpass filter. Filtering concentrates the identification to the frequency range of interest and reduces the effect of high-frequency measurement noise. A Butterworth filter can be applied via the command `idfilt`:

```
>> zif=idfilt(zi,N,Wn);
```

where

```
N=Order of Butterworth filter
Wn=Cut-off frequencies in fractions
   of the Nyquist frequency.
     Wn scalar gives lowpass filter
     Wn=[Wl Wh] gives bandpass filter
```

It is also possible to use the commands `filter` and `filtfilt` from the Signal Processing Toolbox. The latter assures zero phase distortion.

It is important to note that filtering affects the noise model. Consider the setup in Figure 2. If we use the filtered data $u_f$ and $y_f$ to obtain a noise model $\hat{H}_n$, then we actually obtain the estimate $\widehat{H_f H_n}$.

| | |
|---|---|
| `idfilt` | Butterworth filter |
| `filter` | Arbitrary filter |
| `filtfilt` | Undistorted-phase filter |

## 5. Model Structure Selection

We discuss model structure selection for parametric models in this section. The model structure determines the set in which the model estimation is performed. For example, a very simple such model set is the set of static gains $K$ mapping the input to the output, that is, the input–output model $y(t) = Ku(t)$. The complexity of the model structure, of course, affects the accuracy with which the model can approximate the real process. Few dynamical systems can be well approximated by the model $y(t) = Ku(t)$.

Model estimation is treated in next section. Then, both parametric and nonparametric models are considered.

The most general parametric model structure used in the System Identification Toolbox is given by

$$A(q)y(t) = \frac{B(q)}{F(q)}u(t-n_k) + \frac{C(q)}{D(q)}e(t) \qquad (1)$$

where $y$ and $u$ is the output and input sequences, respectively, and $e$ is a white noise sequence with zero mean value. The polynomials $A, B, C, D, F$ are defined in terms of the backward shift operator[1]:

$$\begin{aligned}
A(q) &= 1 + a_1 q^{-1} + \ldots + a_{na} q^{-na} \\
B(q) &= b_1 + b_2 q^{-1} + \ldots + b_{nb} q^{-nb+1} \\
C(q) &= 1 + c_1 q^{-1} + \ldots + c_{nc} q^{-nc} \\
D(q) &= 1 + d_1 q^{-1} + \ldots + d_{nd} q^{-nd} \\
F(q) &= 1 + f_1 q^{-1} + \ldots + f_{nf} q^{-nf}
\end{aligned}$$

Rarely, we use the general structure (1) but some special forms, where one or more polynomial are set to identity:

- AR model
$$A(q)y(t) = e(t) \qquad (2)$$
which is a time-series model with no exogenous input (no input $u$).
- ARX model
$$A(q)y(t) = B(q)u(t-n_k) + e(t) \qquad (3)$$
- ARMAX model
$$A(q)y(t) = B(q)u(t-n_k) + C(q)e(t) \qquad (4)$$
- Output-error (OE) model
$$y(t) = \frac{B(q)}{F(q)}u(t-n_k) + e(t) \qquad (5)$$
- Box-Jenkins (BJ) model
$$y(t) = \frac{B(q)}{F(q)}u(t-n_k) + \frac{C(q)}{D(q)}e(t) \qquad (6)$$

The choice of model structure depends on the noise sequence: how well is it possible to estimate the noise? It is not at all necessary that a model with more parameters or more freedom (more polynomials) is better. Finding the best model is a matter of choosing a suitable structure in combination with the number of parameters.

[1] Notice that this parametrization is *not* the same we are used to from the course Computer-Controlled Systems.

| | Manipulation of models |
|---|---|
| idgrey | Fix parameters (grey-box identification) |
| model.Ts=h | Set sampling interval to h |

| | Information extraction |
|---|---|
| present | Presentation of the model |
| ssdata | convert model to state space |

## 6. Model Estimation

System identification is the procedure of deriving a model from data and *model estimation* is the procedure of fitting a model with a specific model structure. We have linear models and parametric models of a specific structure—*e.g.*, physical models, ARMAX models. In addition to parametric linear models, a linear model may consist of a weighting function or a transfer function in the form of a frequency response.

Using the Matlab System Identification Toolbox., we study how transfer function models can be estimated from data. Also when system identification aims towards a specific parametric model, it makes sense to estimate an input-output relationship in the form of a transfer function. Note that a transfer function estimate may also give hints to model complexity and model structure.

### 6.1 Estimation of Linear Models

A major advantage of linear model estimation methods such as spectrum analysis and covariance analysis is that they require no prior specification of the model structure in terms of model structure, model order etc.

***Correlation analysis*** There are two commands for correlation analysis: `cra` and `covf`. The covariance function $C_{yu}$ is estimated as

$$\hat{C}_{yu}(\tau) = \frac{1}{N}\sum_{k=1}^{N} y(k+\tau)u(k) \tag{7}$$

and $C_{uu}$ and $C_{yy}$ similarly. An estimate of the impulse response can then be derived using the relationship

$$\hat{C}_{yu}(k) = \sum_{l=0}^{\infty} h(l)\hat{C}_{uu}(k-l)$$

If $u$ is assumed to be a white noise sequence, this expression is simplified and gives

$$\hat{h}(k) = \frac{1}{\sigma_u^2}\hat{C}_{yu}(k) \tag{8}$$

If $u$ is not a white noise sequence, it is possible to use a whitening filter $H_w$ such that $u_f = H_w u$ is approximately white. The command `cra` uses filtered signals $u_f$ and $y_f$ to estimate the impulse response as in (8):

```
>> ir=cra(zi)
```

The step response can be computed and plotted as

```
>> sr=cumsum(ir)
>> plot(sr)
```

| | |
|---|---|
| cra | Impulse response from correlation analysis |
| covf | Covariance function estimate |

### 6.2 Spectral analysis from the covariance function

Filtered discrete Fourier transformations (DFT) of the covariance functions $\hat{C}_{uu}$, $\hat{C}_{yy}$, $\hat{C}_{yu}$ give the spectral estimates $\hat{S}_{uu}$, $\hat{S}_{yy}$, $\hat{S}_{yu}$. We have, for example,

$$\hat{S}_{yu}(i\omega) = \sum_{\tau=-M}^{M} \hat{C}_{yu}(\tau)W_M(\tau)e^{-i\omega\tau}$$

where $W_M$ is a window function. The accuracy of the estimates depend on the used window function.

The function `spa` estimates the transfer function and the noise spectrum $S_{nn}$ according to the following formulas

$$\hat{H}(e^{i\omega}) = \frac{\hat{S}_{yu}(i\omega)}{\hat{S}_{uu}(i\omega)}$$

$$\hat{S}_{nn}(i\omega) = \hat{S}_{yy}(i\omega) - \frac{|\hat{S}_{yu}(i\omega)|^2}{\hat{S}_{uu}(i\omega)}$$

A typical Matlab sequence could be

```
>> [H,Snn]=spa(zi,M,w);
>> bode(H)
>> bode(Snn)
```

| | |
|---|---|
| spa | Spectral analysis |

### 6.3 Spectral analysis from the DFT

The command `etfe` estimates the transfer function as the ratio of the DFT of the input and the output:

$$\hat{H}(e^{i\omega_k}) = \frac{\hat{Y}(e^{i\omega_k})}{\hat{U}(e^{i\omega_k})}$$

In Matlab, we write

```
>> H=etfe(zi);
>> bode(H)
```

As in spa, a window is used when deriving H. The estimate will vary with the choice of window, and it is far from trivial to choose a proper window.

The commands tfe and spectrum in the Signal Processing Toolbox estimates the transfer function by using the so called Welch's averaged periodogram method. The quality of the estimates depends on the choice of a number of parameters defining windows, averaging and zero padding. An estimate is derived as

```
>> H=tfe(ui,yi);
```

| | |
|---|---|
| etfe | DFT transfer function estimate |
| tfe | Welch's transfer function estimate |
| spectrum | Various power spectra using Welch's method |
| fft | Discrete Fourier transform |

### 6.4 Parametric models

Assume that one of the model structures in the section Model Structure Selection is adopted. The next step is then to choose an appropriate model order and to estimate the parameters of the polynomials. Methods for doing this is presented in this subsection.

All Matlab functions for parameter estimation have the structure

```
>> model=functionname(zi,orders)
```

where model contains the resulting estimated model, functionname is any of the command names listed below, and orders defines the model order. As an example, we show the use of the command pem, which estimates the parameters of the general parametric model structure (1):

```
>> na=1; nb=1; nc=1; nd=1; nf=1; nk=1;
>> orders=[na nb nc nd nf nk];
>> pem1=pem(zi,orders)
```

The information can be presented by the command present:

```
>> present(pem1);
```

For an ARX model, there are two methods for parameter estimation: (1) the least squares (LS) method and (2) the instrumental (IV) variable method. The parameters of the other model structures are estimated by use of a prediction error method.

| | |
|---|---|
| pem | Estimate general model |
| ar | Estimate AR model |
| ivar | IV estimate of AR model |
| iv4 | IV estimate of AR model |
| arx | LS estimate of ARX model |
| ivx | IV estimate of ARX model |
| armax | Estimate ARMAX model |
| oe | Estimate output-error model |
| bj | Estimate Box-Jenkins model |
| present | Presentation of estimated model |

### 6.5 Model reduction

Model reduction is an alternative to standard model estimation. The idea is to first estimate the parameters of a high order ARX model and then reduce the model order using suitable methods. By estimating a high order model we capture most of the information in the data. The model reduction step then extracts the most significant states of this model.

One way to reduce the order of a linear system $H(q) = B(q)/A(q)$ is to transform it into a balanced state-space realization. The Gramian matrix corresponding to the balanced state-space realization indicates the states of importance. The next step is to reduce the order of the balanced realization by eliminating the insignificant states. Here is a typical sequence of commands:

```
>> arx1=arx(zi,[10 9 1]);
>> [A,B,C,D]=ssdata(arx1);
>> [Ab,Bb,Cb,M,T]=dbalreal(A,B,C);
```

This gives

```
M=[21 12 12 0.03 0.03 0.02 0.02 0 0 0]
```

so the last seven states can be eliminated. This is done as

```
>> [Ab,Bb,Cb,Db]=dmodred(Ab,Bb,Cb,D,4:10);
>> [b,a]=ss2tf(Ab,Bb,Cb,Db);
>> arx1red=idpoly(a,b);
```

| | |
|---|---|
| ssdata | Conversion to state-space |
| tfdata | Conversion to transfer function |
| dbalreal | Discrete balanced realization |
| balreal | Continuous balanced realization |
| dmodred | Discrete model order reduction |
| modred | Continuous model order reduction |
| ss2tf | State-space to transfer function conversion |
| idpoly | Polynomial to transfer function conversion |

# 7. Validation

The parametric models obtained in previous section can be validated in a variety of ways. Here, we discuss model validity criterion, pole–zero and Bode plots, residual analysis, and simulation and cross validation. In a standard identification session all of these are used to affirm an accurate model.

### 7.1 Model validity criterion

It is possible to get an indication of a suitable model order by studying how various criteria depend on the model order. Two such criteria are the loss function and Akaike's Final Prediction Error (FPE). These two criteria is given by the command `present`. A more sophisticated way of using model validity criteria are obtained through `arxstruc` and `ivstruc` together with `selstruc`. These commands can be used for determination of suitable model order for an ARX structure. An example:

```
>> NN=[2 1 1;3 1 1;3 2 1;4 3 1;5 4 1];
>> V=arxstruc(zi,zv,NN);
>> selstruc(V);
```

Each row of the matrix `NN` contains the order of an ARX model. The `arxstruc` command estimates the parameters and computes the corresponding losses for the ARX models defined by `NN` based on the identification data in `zi` and the validation data in `zv`. The `selstruc` command plots the loss as a function of the number of parameters in the model in a diagram. It is possible to invoke the `selstruc` command with an extra argument `'AIC'` or `'MDL'`, in which case the best model is selected according to Akaike's Information Theoretic Criterion (AIC) or Rissanen's Minimum Description Length Criterion (MDL). For example,

```
>> selstruc(V,'AIC');
```

The command `ivstruc` is used in the same way as the `arxstruc` command.

| | |
|---|---|
| `arxstruc` | Loss functions for families of ARX models |
| `ivstruc` | Output-error fit for families of ARX models |
| `selstruc` | Select model structures according to various criteria |
| `struc` | Generate typical structure matrices for `arxstruc` and `ivstruc` |

### 7.2 Pole–zero plots

A rough idea of pole–zero locations are obtained from the simple experiments discussed in Section 3 and from the nonparametric models. It should be verified that the estimated model has similar pole–zero locations. Moreover, if the noise appeared to have dominating frequencies, there should be resonant poles in the noise model with corresponding frequencies.

A pole–zero plot may indicate if the model order is too large. Then there will be poles and zeros located close together, suggesting that model reduction is possible.

```
>> pzmap(pem1);
```

| | |
|---|---|
| `zpkdata` | Zeros, poles, static gains and their standard deviations |
| `pzmap` | Plot of zeros and poles |

### 7.3 Bode diagram

The Bode plot of the estimated polynomial model should be consistent with the frequency analysis in Section 3 and the Bode plots obtained by the nonparametric methods in Section 6. Stationary gain and location of dominating poles and zeros can be checked in the Bode plot. The Nyquist plot of the estimated model can be used in the same way. The noise-spectrum is also obtained, if a second output arguments is added to `th2ff`.

```
>> H=idfrd(pem1('m'));
>> bode(H);
>> Snn=idfrd(pem1('n'));
>> bode(Snn);
```

| | |
|---|---|
| `idfrd` | Frequency function for the model |

### 7.4 Residual analysis

The parametric models described in Section 5 are of the form

$$y(t) = H_u(q)u(t) + H_e(q)e(t)$$

where $H_u(q)$ and $H_e(q)$ are rational transfer functions. The residuals are computed from input–output data as

$$\varepsilon(t) = H_e^{-1}(q)\big(y(t) - H_u(q)u(t)\big)$$

If the residuals are computed based on the identified model and the data used for the identification, then ideally the residuals should be white and independent of the input signals. If this is not the case, then, for example, the model order, the model structure, or the length of the data sequence are inappropriate.

The residuals can be tested in several ways, for example, through

- autocorrelation for the residuals,
- crosscorrelation between the residuals and the input, and
- distribution of residual zero crossings (see Johansson (1993)).

The first two quantities are plotted as

```
>> resid(pem1,zi);
```

Then, the zero crossing test can be checked by plotting the residuals:

```
>> e=resid(pem1,zi);
>> plot(e);
```

The residual analysis should be performed based on both the data sets `zi` and `zv`. The residual sequence computed from the validation data `zv` should ideally be white.

The residual tests should be used with caution. There are model structures, such as the output-error structure and identification methods like the IV methods, which focus on the input–output part $H_u$. Then, it is not realistic to expect that the residuals have white noise properties.

Note also that residual analysis is not relevant in connection with the identification method based on model reduction.

| | |
|---|---|
| `resid` | Computes and tests the residuals associated with a model |

### 7.5 Simulation and cross validation

Simulation and cross validation are methods for testing whether a model can reproduce the observed output when driven by the actual input. The identified model can be tested with the input–output data `zi` in a simulation:

```
>> ui=zi.InputData;
>> yi=zi.OutputData;
>> ys=idsim(ui,pem1);
>> plot([yi ys]);
```

A much more interesting and revealing test is the cross validation, where the validation data `zv` is used for simulation. This test gives a good indication whether the identified model captures the dominating dynamics of the true system or not.

```
>> uv=zv.InputData;
>> yv=zv.OutputData;
>> ys=idsim(uv,pem1);
>> plot([yv ys]);
```

It may be possible to achieve better correspondence between the simulated and the measured output in the cross validation test by using the residuals in the simulation. The reason is that then not only the dynamic part of the model but also the noise model is taken into consideration in the simulation.

```
>> zv=iddata(yv,uv,h);
>> ev=resid(pem1,zv);
>> yc=idsim([uv ev.OutputData],pem1);
>> plot([yv yc]);
```

Cross validation is probably the most important of the validation tests, since by definition we want our obtained model to mirror the true plant in exactly this sense.

| | |
|---|---|
| `compare` | Compare simulated predicted or output with the measured output |
| `idsim` | Simulate a given system |
| `pe` | Prediction errors |
| `predict` | M-step ahead prediction |
| `resid` | Compute and test the residuals associated with a model |

## 8. References


Gustafsson, K. (1989): "`logger`—a program for data logging." Technical Report ISRN LUTFD2/TFRT--7509--SE. Department of Automatic Control, Lund Institute of Technology.

Johansson, R. (1993): *System Modeling and Identification*. Prentice-Hall, Englewood Cliffs, New Jersey.