System Identification Toolbox

For Use with MATLAB®

Lennart Ljung

Computation

Visualization

Programming

User's Guide

How to Contact The MathWorks:

508-647-7000	Phone

508-647-7001	Fax

The MathWorks, Inc. 24 Prime Park Way Natick, MA 01760-1500	Mail

`http://www.mathworks.com`	Web
`ftp.mathworks.com`	Anonymous FTP server
`comp.soft-sys.matlab`	Newsgroup

`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`subscribe@mathworks.com`	Subscribing user registration
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

System Identification Toolbox User's Guide

Printing History: April 1988 First printing
July 1991 Second printing
May 1995 Third printing
August 1995 Reprint

Contents

1 The System Identification Problem

2 The Graphical User Interface

4 Command Reference

1

The System Identification Problem

1. Basic Questions About System Identification

What is System Identification?

System Identification allows you to build mathematical models of a dynamic system based on measured data.

How is that done?

Essentially by adjusting parameters within a given model until its output coincides as well as possible with the measured output.

How do you know if the model is any good?

A good test is to take a close look at the model's output compared to the measured one on a data set that wasn't used for the fit ("Validation Data").

Can the quality of the model be tested in other ways?

It is also valuable to look at what the model couldn't reproduce in the data ("the residuals"). This should not be correlated with other available information, such as the system's input.

What models are most common?

The techniques apply to very general models. Most common models are difference equations descriptions, such as ARX and ARMAX models, as well as all types of linear state-space models.

Do you have to assume a model of a particular type?

For parametric models, you have to specify the structure. However, if you just assume that the system is linear, you can directly estimate its impulse or step response using Correlation Analysis or its frequency response using Spectral Analysis. This allows useful comparisons with other estimated models.

What does the System Identification Toolbox contain?

It contains all the common techniques to adjust parameters in all kinds of linear models. It also allows you to examine the models' properties, and to check if they are any good, as well as to preprocess and polish the measured data.

Isn't it a big limitation to work only with linear models?

No, actually not. Most common model nonlinearities are such that the measured data should be nonlinearly transformed (like squaring a voltage input if you think that it's the power that is the stimuli). Use physical insight about the system you are modeling and try out such transformations on models that are linear in the new variables, and you will cover a lot!

How do I get started?

If you are a beginner, browse through Chapter and then try out a couple of the data sets that come with the toolbox. Use the graphical user interface (GUI) and check out the built-in help functions to understand what you are doing.

Is this really all there is to System Identification?

Actually, there is a huge amount written on the subject. Experience with real data is the driving force to understand more. It is important to remember that any estimated model, no matter how good it looks on your screen, has only picked up a simple reflection of reality. Surprisingly often, however, this is sufficient for rational decision making.

2. Common Terms Used in System Identification

This section defines some of the terms that are frequently used in System Identification.

- **Estimation Data** is the data set that is used to fit a model to data. In the GUI this is the same as the **Working Data**.
- **Validation Data** is the data set that is used for model validation purposes. This includes simulating the model for these data and computing the residuals from the model when applied to these data.
- **Model Views** are various ways of inspecting the properties of a model. They include looking at zeros and poles, transient and frequency response, and similar things.
- **Data Views** are various ways of inspecting properties of data sets. A most common and useful thing is just to plot the data and scrutinize it. So-called *outliers* could be detected then. These are unreliable measurements, perhaps arising from failures in the measurement equipment. The frequency contents of the data signals, in terms of periodograms or spectral estimates, is also most revealing to study.
- **Model Sets** or **Model Structures** are families of models with adjustable parameters. **Parameter Estimation** amounts to finding the "best" values of these parameters. The System Identification problem amounts to finding both a good model structure and good numerical values of its parameters.
- **Parametric Identification Methods** are techniques to estimate parameters in given model structures. Basically it is a matter of finding (by numerical search) those numerical values of the parameters that give the best agreement between the model's (simulated or predicted) output and the measured one.
- **Nonparametric Identification Methods** are techniques to estimate model behavior without necessarily using a given parametrized model set. Typical nonparametric methods include **Correlation analysis**, which estimates a system's impulse response, and **Spectral analysis**, which estimates a system's frequency response.
- **Model Validation** is the process of gaining confidence in a model. Essentially this is achieved by "twisting and turning" the model to scrutinize all aspects of it. Of particular importance is the model's ability to reproduce the behavior of the Validation Data sets. Thus it is important to inspect the properties of the residuals from the model when applied to the Validation Data.

3. Basic Information About Dynamic Models

System Identification is about building **Dynamic Models**. Some knowledge about such models is therefore necessary for successful use of the toolbox.The topic is treated in several places in the Chapter and there is a wide range of textbooks available for introductory and in-depth studies. For basic use of the toolbox, it is sufficient to have quite superficial insights about dynamic models. This section describes such a basic level of knowledge.

The Signals

Models describe relationships between measured signals. It is convenient to distinguish between **input** signals and **output** signals. The outputs are then partly determined by the inputs. Think for example of an airplane where the inputs would be the different control surfaces, ailerons, elevators, and the like, while the outputs would be the airplane's orientation and position. In most cases, the outputs are also affected by more signals than the measured inputs. In the airplane example it would be wind gusts and turbulence effects. Such "unmeasured inputs" will be called **disturbance** signals or **noise**. If we denote inputs, outputs, and disturbances by **u**, **y**, and **e**, respectively, the relationship can be depicted in the following figure.

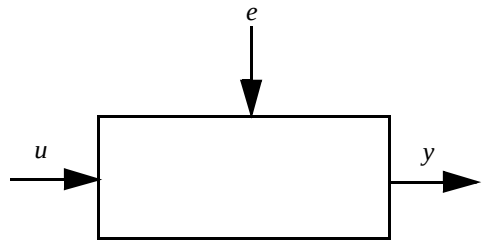


Figure 1-1: Input Signals *u*, Output Signals *y*, and Disturbances *e*

All these signals are functions of time, and the value of the input at time *t* will be denoted by *u(t)*. Often, in the identification context, only discrete-time points are considered, since the measurement equipment typically records the signals just at discrete-time instants, often equally spread in time with a **sampling interval** of *T* time units. The modeling problem is then to describe how the three signals relate to each other.

The Basic Dynamic Model

The basic relationship is the **linear difference equation**. An example of such an equation is the following one.

$$y(t)-1.5y(t-T)+0.7y(t-2T)=0.9u(t-2T)+0.5u(t-3T)+e(t)\quad(ARX)$$

Such a relationship tells us, for example, how to compute the output *y(t)* if the input is known and the disturbance can be ignored:

$$y(t)=1.5y(t-T)-0.7y(t-2T)+0.9u(t-2T)+0.5u(t-3T)$$

The output at time *t* is thus computed as a linear combination of past outputs and past inputs. It follows, for example, that the output at time *t* depends on the input signal at many previous time instants. This is what the word **dynamic** refers to. The identification problem is then to use measurements of *u* and *y* to figure out

- The coefficients in this equation (i.e., -1.5, 0.7, etc.)

  How many delayed outputs to use in the description (two in the example: $y(t-T)$ *and* $y(t-2T)$)
- The **time delay** in the system is (*2T* in the example: you see from the second equation that it takes *2T* time units before a change in *u* will affect *y*) and
- How many delayed inputs to use (two in the example: *u(t-2T)* and *u(t-3T)*)

Variants of Model Descriptions

The model given above is called an **ARX model**. There are a handful of variants of this model known as **Output-Error** (OE) models, **ARMAX** models, **FIR** models, and **Box-Jenkins** (BJ) models. These are described later on in the manual. At a basic level it is sufficient to think of them as variants of the ARX model allowing also a characterization of the properties of the disturbances *e*.

General linear models can be described symbolically by

$$y=Gu+He$$

which says that the measured output *y(t)* is a sum of one contribution that comes from the measured input *u(t)* and one contribution that comes from the noise *He*. The symbol *G* then denotes the dynamic properties of the system, that is, how the output is formed from the input. For linear systems it is called the transfer function from input to output. The symbol *H* refers to the noise

properties, and is called the noise model. It describes how the disturbances at the output are formed from some standardized noise source *e(t)*.

State-space models are common representations of dynamical models. They describe the same type of linear difference relationship between the inputs and the outputs as in the ARX model, but they are rearranged so that only one delay is used in the expressions. To achieve this, some extra variables, the **state variables**, are introduced. They are not measured, but can be reconstructed from the measured input-output data. This is especially useful when there are several output signals, i.e., when *y(t)* is a vector. Chapter gives more details about this. For basic use of the toolbox it is sufficient to know that the **order** of the state-space model relates to the number of delayed inputs and outputs used in the corresponding linear difference equation. The state-space representation looks like

$$x(t+1) = Ax(t) + Bu(t) + Ke(t)$$

$$y(t) = Cx(t) + Du(t) + e(t)$$

Here *x(t)* is the vector of state variables. The matrix *K* determines the noise properties. Notice that if *K = 0*, then the noise source *e(t)* affects only the output, and no specific model of the noise properties is built. This corresponds to *H = 1* in the general description above, and is usually referred to as an *Output-Error model*. Notice also that *D = 0* means that there is no direct influence from *u(t)* to *y(t)*. Thus the effect of the input on the output all passes via *x(t)* and will thus be delayed at least one sample. The first value of the state variable vector *x(0)* reflects the initial conditions for the system at the beginning of the data record. When dealing with models in state-space form, a typical option is whether to estimate *D*, *K*, and *x(0)* or to let them be zero.

How to Interpret the Noise Source

In many cases of system identification, the effects of the noise on the output are insignificant compared to those of the input. With good signal-to-noise ratios (SNR), it is less important to have an accurate noise model. Nevertheless it is important to understand the role of the noise and the noise source *e(t)*, whether it appears in the ARX model or in the general descriptions given above.

There are three aspects of the noise that should be stressed:

- understanding white noise
- interpreting the noise source
- using the noise source when working with the model

These aspects are discussed one by one.

How can we understand white noise? From a formal point of view, the noise source *e(t)* will normally be regarded as *white noise*. This means that it is entirely unpredictable. In other words, it is impossible to guess the value of *e(t)* no matter how accurately we have measured past data up to time *t*-1.

How can we interpret the noise source? The actual noise contribution to the output, *H e(t)*, has real significance. It contains all the influences on the measured *y*, known and unknown, that are not contained in the input *u*. It explains and captures the fact that even if an experiment is repeated with the same input, the output signal will typically be somewhat different. However, the noise source *e(t)* need not have a physical significance. In the airplane example mentioned earlier, the noise effects are wind gusts and turbulence. Describing these as arising from a white noise source via a transfer function *H*, is just a convenient way of capturing their character.

How can we deal with the noise source when using the model? If the model is used just for simulation, i.e., the responses to various inputs are to be studied, then the noise model plays no immediate role. Since the noise source *e(t)* for new data will be unknown, it is taken as zero in the simulations, so as to study the effect of the input alone (a noise-free simulation). Making another simulation with *e(t)* being arbitrary white noise will reveal how reliable the result of the simulation is, but it will not give a more accurate simulation result for the actual system's response.

The need and use of the noise model can be summarized as follows:

- It is, in most cases, required to obtain a better estimate for the dynamics, *G*.
- It indicates how reliable noise-free simulations are.
- It is required for reliable predictions and stochastic control design.

Terms to Characterize the Model Properties

The properties of an input-output relationship like the ARX model follow from the numerical values of the coefficients, and the number of delays used. This is however a fairly implicit way of talking about the model properties. Instead a number of different terms are used in practice:

Impulse Response

The impulse response of a dynamical model is the output signal that results when the input is an impulse, i.e., $u(t)$ is zero for all values of t except $t=0$, where $u(0)=1$. It can be computed as in the equation following (ARX), by letting t be equal to 0, 1, 2, ... and taking $y(-T)=y(-2T)=0$ and $u(0)=1$.

Step Response

The step response is the output signal that results from a step input, i.e., $u(t)$ is zero for negative values of t and equal to one for positive values of t. The impulse and step responses together are called the model's **transient response**.

Frequency Response

The frequency response of a linear dynamic model describes how the model reacts to sinusoidal inputs. If we let the input $u(t)$ be a sinusoid of a certain frequency, then the output $y(t)$ will also be a sinusoid of this frequency. The amplitude and the phase (relative to the input) will however be different. This frequency response is most often depicted by two plots; one that shows the amplitude change as a function of the sinusoid's frequency and one that shows the phase shift as function of frequency. This is known as a Bode plot.

Zeros and Poles

The zeros and the poles are equivalent ways of describing the coefficients of a linear difference equation like the ARX model. The poles relate to the "output-side" and the zeros relate to the "input-side" of this equation. The number of poles (zeros) is equal to number of sampling intervals between the most and least delayed output (input). In the ARX example in the beginning of this section, there are consequently two poles and one zero.

4. The Basic Steps of System Identification

The System Identification problem is to estimate a model of a system based on observed input-output data. Several ways to describe a system and to estimate such descriptions exist. This section gives a brief account of the most important approaches.

The procedure to determine a model of a dynamical system from observed input-output data involves three basic ingredients:

- The input-output data
- A set of candidate models (the model structure)
- A criterion to select a particular model in the set, based on the information in the data (the identification method)

The identification process amounts to repeatedly selecting a model structure, computing the best model in the structure, and evaluating this model's properties to see if they are satisfactory. The cycle can be itemized as follows:

1. Design an experiment and collect input-output data from the process to be identified.
2. Examine the data. Polish it so as to remove trends and outliers, select useful portions of the original data, and apply filtering to enhance important frequency ranges.
3. Select and define a model structure (a set of candidate system descriptions) within which a model is to be found.
4. Compute the best model in the model structure according to the input-output data and a given criterion of fit.
5. Examine the obtained model's properties
6. If the model is good enough, then stop; otherwise go back to Step 3 to try another model set. Possibly also try other estimation methods (Step 4) or work further on the input-output data (Steps 1 and 2).

The System Identification Toolbox offers several functions for each of these steps.

For Step 2 there are routines to plot data, filter data, and remove trends in data.

For Step 3 the System Identification Toolbox offers a variety of nonparametric models, as well as all the most common black-box input-output and state-space structures, and also general tailor-made linear state-space models in discrete and continuous time.

For Step 4 general prediction error (maximum likelihood) methods as well as instrumental variable methods and sub-space methods are offered for parametric models, while basic correlation and spectral analysis methods are used for nonparametric model structures.

To examine models in Step 5, many functions allow the computation and presentation of frequency functions and poles and zeros, as well as simulation and prediction using the model. Functions are also included for transformations between continuous-time and discrete-time model descriptions and to formats that are used in other MATLAB toolboxes, like the Control System Toolbox and the Signal Processing Toolbox.

5. A Startup Identification Procedure

There are no standard and secure routes to good models in System Identification. Given the number of possibilities, it is easy to get confused about what to do, what model structures to test, and so on. This section describes one route that often works well, but there are no guarantees. The steps refer to functions within the GUI, but you ca6n also go through them in command mode. See Chapter for the basic commands.

Step 1 Looking at the Data

Plot the data. Look at them carefully. Try to see the dynamics with your own eyes. Can you see the effects in the outputs of the changes in the input? Can you see nonlinear effects, like different responses at different levels, or different responses to a step up and a step down? Are there portions of the data that appear to be “messy” or carry no information. Use this insight to select portions of the data for estimation and validation purposes.

Do physical levels play a role in your model? If not, detrend the data by removing their mean values. The models will then describe how changes in the input give changes in output, but not explain the actual levels of the signals. This is the normal situation.

The default situation, with good data, is that you detrend by removing means, and then select the first half or so of the data record for estimation purposes, and use the remaining data for validation. This is what happens when you apply **Quickstart** under the pop-up menu **Preprocess** in the main **ident** window.

Step 2 Getting a Feel for the Difficulties

Apply **Quickstart** under pop-up menu **Estimate** in the main **ident** window. This will compute and display the spectral analysis estimate and the correlation analysis estimate, as well as a fourth order ARX model with a delay

estimated from the correlation analysis and a default order state-space model computed by `n4sid`. This gives three plots. Look at the agreement between the

- Spectral Analysis estimate and the ARX and state-space models' frequency functions
- Correlation Analysis estimate and the ARX and state-space models' transient responses
- Measured Validation Data output and the ARX and state-space models' simulated outputs

If these agreements are reasonable, the problem is not so difficult, and a relatively simple linear model will do a good job. Some fine tuning of model orders, and noise models have to be made and you can proceed to Step 4. Otherwise go to Step 3.

Step 3 Examining the Difficulties

There may be several reasons why the comparisons in Step 2 did not look good. This section discusses the most common ones, and how they can be handled:

Model Unstable

The ARX or state-space model may turn out to be unstable, but could still be useful for control purposes. Change to a 5- or 10-step ahead prediction instead of simulation in the **Model Output View**.

Feedback in Data

If there is feedback from the output to the input, due to some regulator, then the spectral and correlations analysis estimates are not reliable. Discrepancies between these estimates and the ARX and state-space models can therefore be disregarded in this case. In the **Model Residuals View** of the parametric models, feedback in data can also be visible as correlation between residuals and input for negative lags.

Noise Model

If the state-space model is clearly better than the ARX model at reproducing the measured output, this is an indication that the disturbances have a substantial influence, and it will be necessary to model them carefully.

Model Order

If a fourth order model does not give a good **Model Output** plot, try eighth order. If the fit clearly improves, it follows that higher order models will be required, but that linear models could be sufficient.

Additional Inputs

If the **Model Output** fit has not significantly improved by the tests so far, think over the physics of the application. Are there more signals that have been, or could be, measured that might influence the output? If so, include these among the inputs and try again a fourth order ARX model from all the inputs. (Note that the inputs need not at all be control signals, anything measurable, including disturbances, should be treated as inputs).

Nonlinear Effects

If the fit between measured and model output is still bad, consider the physics of the application. Are there nonlinear effects in the system? In that case, form the nonlinearities from the measured data. This could be as simple as forming the product of voltage and current measurements, if you realize that it is the electrical power that is the driving stimulus in, say, a heating process, and temperature is the output. This is of course application dependent. It does not take very much work, however, to form a number of additional inputs by reasonable nonlinear transformations of the measured ones, and just test if inclusion of them improves the fit.

Still Problems?

If none of these tests leads to a model that is able to reproduce the Validation Data reasonably well, the conclusion might be that a sufficiently good model cannot be produced from the data. There may be many reasons for this. The most important one is that the data simply do not contain sufficient information, e.g., due to bad signal to noise ratios, large and nonstationary disturbances, varying system properties, etc. The reason may also be that the system has some quite complicated nonlinearities, which cannot be realized on physical grounds. In such cases, nonlinear, black box models could be a solution. Among the most used models of this character are the Artificial Neural Networks (ANN).

Otherwise, use the insights of which inputs to use and which model orders to expect and proceed to Step 4.

Step 4 Fine Tuning Orders and Noise Structures

For real data there is no such thing as a "correct model structure." However, different structures can give quite different model quality. The only way to find this out is to try out a number of different structures and compare the properties of the obtained models. There are a few things to look for in these comparisons:

Fit Between Simulated and Measured Output

Keep the **Model Output View** open and look at the fit between the model's simulated output and the measured one for the Validation Data. Formally, you could pick that model, for which this number is the lowest. In practice, it is better to be more pragmatic, and also take into account the model complexity, and whether the important features of the output response are captured.

Residual Analysis Test

You should require of a good model, that the cross correlation function between residuals and input does not go significantly outside the confidence region. A clear peak at lag *k* shows that the effect from input *u(t-k)* on *y(t)* is not properly described. A rule of thumb is that a slowly varying cross correlation function outside the confidence region is an indication of too few poles, while sharper peaks indicate too few zeros or wrong delays.

Pole Zero Cancellations

If the pole-zero plot (including confidence intervals) indicates pole-zero cancellations in the dynamics, this suggests that lower order models can be used. In particular, if it turns out that the orders of ARX models have to be increased to get a good fit, but that pole-zero cancellations are indicated, then the extra poles are just introduced to describe the noise. Then try ARMAX, OE, or BJ model structures with an A or F polynomial of an order equal to that of the number of noncanceled poles.

What Model Structures Should be Tested?

Well, you can spend any amount of time to check out a very large number of structures. It often takes just a few seconds to compute and evaluate a model in a certain structure, so that you should have a generous attitude to the testing. However, experience shows that when the basic properties of the system's behavior have been picked up, it is not much use to fine tune orders in absurdum just to press the fit by fractions of percents.

Many ARX models: There is a very cheap way of testing many ARX structures simultaneously. Enter in the **Orders** text field many combinations of orders, using the colon (":") notation. When you select **Estimate**, models for all combinations (easily several hundreds) are computed and their (prediction error) fit to Validation Data is shown in a special plot. By clicking in this plot the best models with any chosen number of parameters will be inserted into the Model Board, and evaluated as desired.

Many State-space models: A similar feature is also available for black-box state-space models, estimated using `n4sid`. When a good order has been found, try the PEM estimation method, which often improves on the accuracy.

ARMAX, OE, and BJ models: Once you have a feel for suitable delays and dynamics orders, if is often useful to try out ARMAX, OE, and/or BJ with these orders and test some different orders for the noise transfer functions (C and D). Especially for poorly damped systems, the OE structure is suitable.

There is a quite extensive literature on order and structure selection, and anyone who would like to know more should consult the references.

Multivariable Systems

Systems with many input signals and/or many output signals are called *multivariable*. Such systems are often more challenging to model. In particular systems with several outputs could be difficult. A basic reason for the difficulties is that the couplings between several inputs and outputs lead to more complex models. The structures involved are richer and more parameters will be required to obtain a good fit.

Available Models

The System Identification Toolbox as well as the GUI handles general, linear multivariable models. All earlier mentioned models are supported in the single output, multiple input case. For multiple outputs ARX models and state-space models are covered. Multi-output ARMAX and OE models are covered via state-space representations: ARMAX corresponds to estimating the K-matrix, while OE corresponds to fixing K to zero. (These are pop-up options in the GUI model order editor.)

Generally speaking, it is preferable to work with state-space models in the multivariable case, since the model structure complexity is easier to deal with. It is essentially just a matter of choosing the model order.

Working with Subsets of the Input Output Channels

In the process of identifying good models of a system, it is often useful to select subsets of the input and output channels. Partial models of the system's behavior will then be constructed. It might not, for example, be clear if all measured inputs have a significant influence on the outputs. That is most easily tested by removing an input channel from the data, building a model for how the output(s) depends on the remaining input channels, and checking if there is a significant deterioration in the model output's fit to the measured one. See also the discussion under Step 3 above.

Generally speaking, the fit gets better when more inputs are included and worse when more outputs are included. To understand the latter fact, you should realize that a model that has to explain the behavior of several outputs has a tougher job than one that just must account for a single output. If you have difficulties obtaining good models for a multi-output system, it might be wise to model one output at a time, to find out which are the difficult ones to handle.

Models that are just to be used for simulations could very well be built up from single-output models, for one output at a time. However, models for prediction and control will be able to produce better results if constructed for all outputs simultaneously. This follows from the fact that knowing the set of all previous output channels gives a better basis for prediction, than just knowing the past outputs in one channel.

Some Practical Advice

The GUI is particularly suited for dealing with multivariable systems since it will do useful bookkeeping for you, handling different channels. You could follow the steps of this agenda:

- Import data and create a data set with all input and output channels of interest. Do the necessary preprocessing of this set in terms of detrending, prefiltering, etc., and then select a Validation Data set with all channels.
- Then select a Working Data set with all channels, and estimate state-space models of different orders using `n4sid` for these data. Examine the resulting model primarily using the **Model Output** view.
- If it is difficult to get a good fit in all output channels or you would like to investigate how important the different input channels are, construct new data sets using subsets of the original input/output channels. Use the pop-up menu **Preprocess > Select Channels** for this. Don't change the Validation

Data. The GUI will keep track of the input and output channel numbers. It will “do the right thing” when evaluating the channel-restricted models using the Validation Data. It might also be appropriate to see if improvements in the fit are obtained for various model types, built for one output at a time.

- If you decide for a multi-output model, it is often easiest to use state-space models. Use `n4sid` as a primary tool and try out `pem` when a good order has been found. Note that `n4sid` does not provide confidence intervals for the model views.

Reading More About System Identification

There is substantial literature on System Identification. The following textbook deals with identification methods from a similar perspective as this toolbox, and also describes methods for physical modeling.

- Ljung L. and T. Glad. *Modeling of Dynamic Systems,* Prentice Hall, Englewood Cliffs, N.J. 1994.

For more details about the algorithms and theories of identification:

- Ljung L.. *System Identification - Theory for the User*, Prentice Hall, Englewood Cliffs, N.J. 1987.
- Söderström T. and P. Stoica. *System Identification,* Prentice Hall International, London. 1989.

For more about system and signals:

- Oppenheim J. and A.S. Willsky. *Signals and Systems,* Prentice Hall, Englewood Cliffs, N.J. 1985.

The following textbook deals with the underlying numerical techniques for parameter estimation.

- Dennis, J.E. Jr. and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall, Englewood Cliffs, N.J. 1983.

2

The Graphical User Interface

1. The Big Picture

The System Identification Toolbox provides a graphical user interface (GUI) . The GUI covers most of the toolbox's functions and gives easy access to all variables that are created during a session. It is started by typing

```
ident
```

in the MATLAB command window.

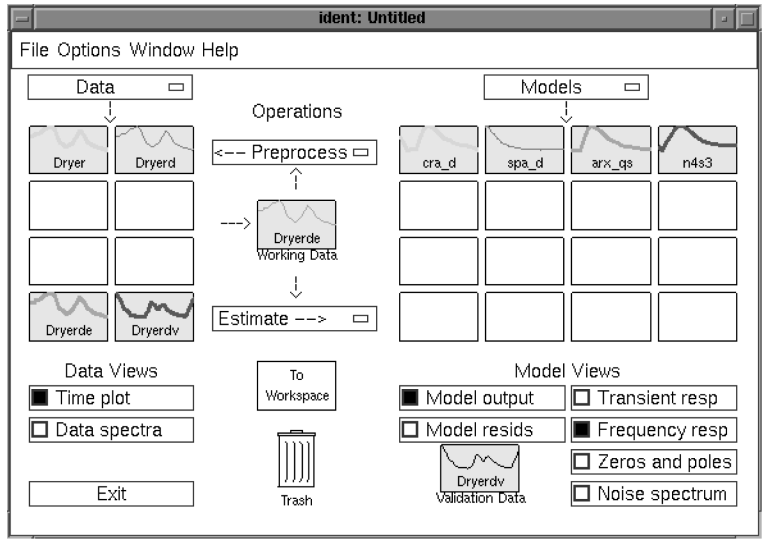


Figure 2-1: The Main ident Information Window

The Model and Data Boards

System Identification is about data and models and creating models from data. The main information and communication window **ident**, is therefore dominated by two tables:

- A table over available data sets, each represented by an icon.
- A table over created models, each represented by an icon.

These tables will be referred to as the “Model Board” and the “Data Board” in this chapter. You enter data sets into the Data Board by

- Opening earlier saved sessions.
- Importing them from the MATLAB workspace.
- Creating them by detrending, filtering, selecting subsets, etc., of another data set in the Data Board.

Imports are handled under the pop-up menu **Data** while creation of new data sets is handled under the pop-up menu **Preprocess**. “Handling Data” on page 2-7 deals with this in more detail.

The models are entered into the summary board by

- Opening earlier saved sessions.
- Importing them from the MATLAB workspace.
- Estimating them from data.

Imports are handled under the pop-up menu **Models**, while all the different estimation schemes are reached under the pop-up menu **Estimate**. More about this in “Estimating Models” on page 2-14.

The Data and Model Boards can be rearranged by dragging and dropping. More boards open automatically when necessary or when asked for (under menu **Options**).

The Working Data

All data sets and models are created from the Working Data set. This is the data that is given in the center of the **ident** window. To change the Working Data set drag and drop any data set from the Data Board on the Working Data icon.

The Views

Below the Data and Model Boards are buttons for different views. These control what aspects of the data sets and models you would like to examine, and are described in more detail in “Handling Data” on page 2-7 and in “Examining Models” on page 2-27. To select a data set or a model, so that its properties are displayed, click on its icon. A selected object is marked by a thicker line in the icon. To deselect, click again. An arbitrary number of data/model objects can be examined simultaneously. To have more information about an object, double-click on its icon.

The Validation Data

The two model views **Model Output** and **Model Residuals** illustrate model properties when applied to the Validation Data set. This is the set marked in the box below these two views. To change the Validation Data, drag and drop any data set from the Data Board on the Validation Data icon.

It is good and common practice in identification to evaluate an estimated model's properties using a "fresh" data set, that is, one that was not used for the estimation. It is thus good advice to let the Validation Data be different from the Working Data, but they should of course be compatible with these.

The Work Flow

You start by importing data (under pop-up menu **Data**); you examine the data set using the **Data Views.** You probably remove the means from the data and select subsets of data for estimation and validation purposes using the items in the pop-up menu **Preprocess**. You then continue to estimate models, using the possibilities under the pop-up menu **Estimate**, perhaps first doing a quickstart. You examine the obtained models with respect to your favorite aspects using the different **Model Views**. The basic idea is that any checked view shows the properties of all selected models at any time. This function is "live" so models and views can be checked in and out at will in an online fashion. You select/deselect a model by clicking on its icon.

Inspired by the information you gain from the plots, you continue to try out different model structures (model orders) until you find a model you are satisfied with.

Management Aspects

Diary: It is easy to forget what you have been doing. By double-clicking on a data/model icon, a complete diary will be given of how this object was created, along with other key information. At this point you can also add comments and change the name of the object and its color.

Layout: To have a good overview of the created models and data sets, it is good practice to try rearranging the icons by dragging and dropping. In this way models corresponding to a particular data set can be grouped together, etc. You can also open new boards (**Options** menu **Extra model/data boards**) to further rearrange the icons. These can be dragged across the screen between different windows. The extra boards are also equipped with notepads for your comments.

Sessions: The Model and Data Boards with all models and data sets together with their diaries can be saved (under menu item **File**) at any point, and reloaded later. This is the counterpart of save/load workspace in the command-driven MATLAB. The four most recent sessions are listed under **File** for immediate open.

Cleanliness: The boards will hold an arbitrary number of models and data sets (by creating clones of the board when necessary). It is however advisable to clear (delete) models and data sets that no longer are of interest. Do that by dragging the object to the **Trash Can**. (Double-clicking on the trash can will open it up, and its contents can be retrieved.)

Window Culture: Dialog and plot windows are best managed by the GUI's `close` function (submenu item under **File** menu, or select **Close**, or check/uncheck the corresponding View box). They may also be"quitted" by the specific window system's quit/close function. This does no harm, but "quit" will not be properly acknowledged by the GUI, and the window will have to be re-created next time it is to be used.

It is generally not suitable to iconify the windows – the GUI's handling and window management system is usually a better alternative.

Workspace Variables

The models and data sets created within the GUI are normally not available in the MATLAB workspace. Indeed, the workspace is not at all littered with variables during the sessions with the GUI. The variables can however be exported at any time to the workspace, by dragging and dropping the object icon on the **To Workspace** box. They will then carry the name in the workspace that marked the object icon at the time of export. You can work with the variables in the workspace, using any MATLAB commands, and then perhaps import modified versions back into the GUI. Note that models have a specific internal structure and should be dealt with using the MATLAB commands `present`, `th2ff`, `th2ss`, etc. See "Model Conversions" on page 4-5 of the "Command Reference" chapter.

The GUI's names of data sets and models are suggested by default procedures. Normally, you can enter any other name of your choice at the time of creation of the variable. Names can be changed (after double-clicking on the icon) at any time. Unlike the workspace situation, two GUI objects can carry the same name (i.e., the same string in their icons).

The GUI produces a handful of global workspace variables for management purposes. They all start with the prefix `XID`.

NOTE: Do not `clear all` or `clear global` during a GUI session! This would mean that you lose control over the objects that you have created. The same disaster occurs if you do `clg` or `quit` the main **ident** window. It is however safe to `clear` (without adding `all` or `global`) the workspace at any time.

Help Texts

The GUI contains some 100 help texts that are accessible in a nested fashion, when required. The main **ident** window contains general help topics under the **Help** menu. This is also the case for the various plot windows. In addition, every dialog box has a **Help** push button for current help and advice.

2. Handling Data

Data Representation

In the System Identification Toolbox (SITB), signals and observed data are represented as column vectors, e.g.,

$$u = \begin{bmatrix} u(1) \\ u(2) \\ \dots \\ \dots \\ u(N) \end{bmatrix}$$

The entry in row number k, i.e., $u(k)$, will then be the signal's value at sampling instant number k. It is generally assumed in the toolbox that data are sampled at equidistant sampling times, and the sampling interval T is supplied as a specific argument.

We generally denote the input to a system by the letter u and the output by y. If the system has several input channels, the input data is represented by a matrix, where the columns are the input signals in the different channels:

$$u = \begin{bmatrix} u1 & u2 & \dots & um \end{bmatrix}$$

The same holds for systems with several output channels.

The observed input-output data record is represented in the SITB by a matrix, where the first column(s) is the output, followed by the input column(s):

```
z = [y u];
```

When you work with the GUI, you only need to think of these representation issues when you insert the data set into the summary board. The GUI will then handle the data representation automatically.

Getting Data into the GUI

The information about a data set that should be supplied to the GUI is as follows:

1. The input and output signals
2. The name you give to the data set
3. The starting time
4. The sampling interval
5. Data notes

NOTE: Items 3 and 4 are used only to ensure correct time and frequency scales when you plot data and model characteristics.

These are notes for your own information and bookkeeping that will follow the data and all models created from them.

As you select the pop-up menu **Data** and choose the item **Import...**, a dialog box will open, where you can enter the information items 1 - 5, just listed. This box has six fields for you to fill in:

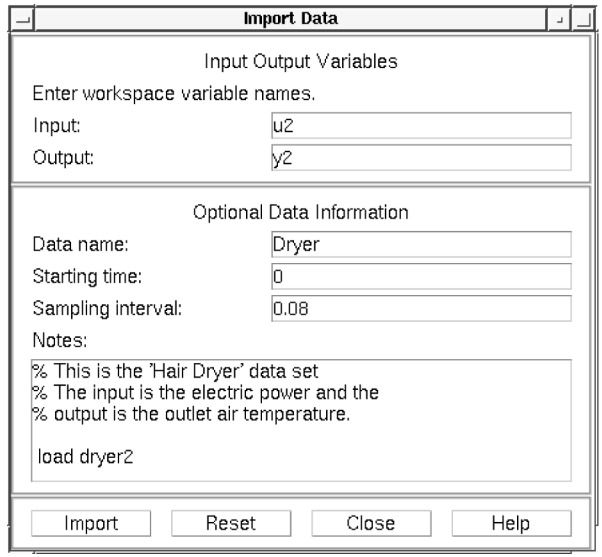


Figure 2-2: The Dialog for Importing Data into the GUI

Input and **Output**: Enter the variable names of the input and output respectively. These should be variables in your MATLAB workspace, so you may have to load some disk files first.

Actually, you can enter any MATLAB expressions in these fields, and they will be evaluated to compute the input and the output before inserting the data into the GUI.

Data name: Enter the name of the data set to be used by the GUI. This name can be changed later on.

Starting time and Sampling interval: Fill these out for correct time and frequency scales in the plots.

Note that you can enter any text you want to accompany the data for bookkeeping purposes.

Finally, select **Import** to insert the data into the GUI. When no more data sets are to be inserted, select **Close** to close the dialog box. **Reset** will empty all the fields of the box.

Taking a Look at the Data

The first thing to do after having inserted the data set into the Data Board is to examine it. By checking the **Data View** item **Plot Data**, a plot of the input and output signals will be shown for the data sets that are selected. You select/deselect the data sets by clicking on them. For multivariable data, the different combinations of input and output signals are chosen under menu item **Channel** in the plot window. Using the zoom function (drawing rectangles with the left mouse button down) different portions of the data can be examined in more detail.

To examine the frequency contents of the data, check the **Data View** item **Data Spectra**. The function is analogous to **Plot Data**, but the signals' spectra are shown instead. By default the periodograms of the data are shown, i.e., the absolute square of the Fourier transforms of the data. The plot can be changed to any chosen frequency range and a number of different ways of estimating spectra, by the **Options** menu item in the spectra window.

The purpose of examining the data in these ways is to find out if there are portions of the data that are not suitable for identification, if the information contents of the data is suitable in the interesting frequency regions, and if the data have to be preprocessed in some way, before using them for estimation.

Preprocessing Data

Detrending

Detrending the data involves removing the mean values or linear trends from the signals (the means and the linear trends are then computed and removed from each signal individually). This function is accessed under the pop-up menu **Preprocess**, by selecting item **Remove Means** or **Remove Trends**. More advanced detrending, such as removing piecewise linear trends or seasonal variations cannot be accessed within the GUI. It is generally recommended that you always remove at least the mean values of the data before the estimation phase, unless physical insight involving actual signal levels is built into the models.

Selecting Data Ranges

It is often the case that the whole data record is not suitable for identification, due to various undesired features (missing or "bad" data, outbursts of disturbances, level changes etc.), so that only portions of the data can be used. In any case, it is advisable to select one portion of the measured data for estimation purposes and another portion for validation purposes. The pop-up menu item **Preprocess > Select Range...** opens a dialog box, which facilitates the selection of different data portions, by typing in the ranges, or marking them by drawing rectangles with the mouse button down.

For multivariable data it is often advantageous to start by working with just some of the input and output signals. The menu item **Preprocess > Select Channels...** allows you to select subsets of the inputs and outputs. This is done in such a way that the input/output numbering remains consistent when you evaluate data and model properties, for models covering different subsets of the data.

Prefiltering

By filtering the input and output signals through a linear filter (the same filter for all signals) you can focus the model's fit to the system to specific frequency ranges. This is done by selecting the pop-up menu item **Preprocess > Filter...** in the main window. The dialog is quite analogous to that of selecting data ranges in the time domain. You mark with a rectangle in the spectral plots the intended passband or stop band of the filter, you select a button to check if the filtering has the desired effect, and then you insert the filtered data into the GUI's Data Board.

Prefiltering is a good way of removing high frequency noise in the data, and also a good alternative to detrending (by cutting out low frequencies from the pass band). Depending on the intended model use, you can also make sure that the model concentrates on the important frequency ranges. For a model that will be used for control design, for example, the frequency band around the intended closed-loop bandwidth is of special importance.

Resampling

If the data turn out to be sampled too fast, they can be decimated, i.e., every k-th value is picked, after proper prefiltering (antialias filtering). This is obtained from menu item **Preprocess > Resample**.

You can also resample at a faster sampling rate by interpolation, using the same command, and giving a resampling factor less than one.

Quickstart

The pop-up menu item **Preprocess > Quickstart** performs the following sequence of actions: It opens the Time plot Data view, removes the means from the signals, and it splits these detrended data into two halves. The first one is made Working Data and the second one becomes Validation Data. All the three created data sets are inserted into the Data Board.

Checklist for Data Handling

- Insert data into the GUI's Data Board.
- Plot the data and examine it carefully.
- Typically detrend the data by removing mean values.
- Possibly prefilter the data to enhance and suppress various frequency bands.
- Select portions of the data for Estimation and for Validation. Drag and drop these data sets to the corresponding boxes in the GUI.

Simulating Data

The GUI is intended primarily for working with real data sets, and does not itself provide functions for simulating synthetic data. That has to be done in command mode, and you can use your favorite procedure in SIMULINK, the Signal Processing Toolbox, or any other toolbox for simulation and then insert the simulated data into the GUI as described above.

The System Identification Toolbox also has several commands for simulation. You should check `idinput`, `idsim`, `poly2th`, `modstruc`, and `ms2th` in Chapter 4, "Command Reference," for details. The following example shows how the ARMAX model

$$y(t)-1.5y(t-1)+0.7y(t-2)=u(t-1)+0.5u(t-2)+e(t)-e(t-1)+0.2e(t-2)$$

is simulated with a binary random input u:

```
model1 = poly2th([1 -1.5 0.7],[0 1 0.5],[1 -1 0.2]);
u = idinput(400,'rbs',[0 0.3]);
e = randn(400,1);
y = idsim([u e],model1);
```

The input, u, and the output, y, can now be imported into the Graphical User Interface as data, and the various estimation routines can be applied to them. By also importing the simulation model, `model1`, into the GUI, its properties can be compared to those of the different estimated models.

To simulate a continuous-time state-space model:

$$\dot{x} = Ax + Bu + Ke$$

$$y = Cx + e$$

with the same input, and a sampling interval of 0.1 seconds, do the following in the System Identification Toolbox:

```
A=[-1 1;-0.5 0]; B=[1; 0.5]; C=[1 0]; D=0; K=[0.5;0.5];
model2=ms2th(modstruc(A,B,C,D,K),'c');
model2=sett(model2,0.1);
y=idsim([u e],model2);
```

3. Estimating Models

The Basics

Estimating models from data is the central activity in the System Identification Toolbox. It is also the one that offers the most variety of possibilities and thus is the most demanding one for the user.

All estimation routines are accessed from the pop-up menu **Estimate** in the **ident** window. The models are always estimated using the data set that is currently in the **Working Data** box.

One can distinguish between two different types of estimation methods:

- Direct estimation of the Impulse or the Frequency Response of the system. These methods are often also called nonparametric estimation methods, and do not impose any structure assumptions about the system, other than that it is linear.
- Parametric methods. A specific model structure is assumed, and the parameters in this structure are estimated using data. This opens up a large variety of possibilities, corresponding to different ways of describing the system. Dominating ways are state-space and several variants of difference equation descriptions.

Direct Estimation of the Impulse Response

A linear system can be described by the impulse response g_t, with the property that

$$y(t) = \sum_{k=1}^{\infty} g_k u(t-k)$$

The name derives from the fact that if the input $u(t)$ is an impulse, i.e., $u(t)=1$ when $t=0$ and 0 when $t>0$ then the output $y(t)$ will be $y(t)=g_t$. For a multivariable system, the impulse response g_k will be a p by m matrix, where p is the number of outputs and m is the number of inputs. Its i-j element thus described the behavior of the i-th output after an impulse in the j-th input.

By choosing menu item **Estimate > Correlation Model** and then selecting **Estimate** in the dialog window that opens, impulse response coefficients are estimated directly from the input/output data using so called *correlation*

analysis. The actual method is described under the command `cra` in the *Command Reference* chapter. An option that determines the order of a prewhitening filter can also be set in the dialog window. It is not very sensitive in most cases, and the default choice is often good enough. To obtain the default choice without opening the dialog window, you can also just type the letter `c` in the **ident** window. This is the "hotkey" for correlation analysis.

The resulting impulse response estimate is placed in the Model Board, under the default name `cra_d`. (The name can be changed by double-clicking on the model icon and then typing in the desired name in the dialog box that opens.)

The best way to examine the result is to select the **Model View Transient Response**. This gives a graph of the estimated response. This view offers a choice between displaying the Impulse or the Step response. For a multivariable system, the different channels, i.e., the responses from a certain input to a certain output, are selected under menu item **Channel**.

The number of lags for which the impulse response is estimated, i.e., the length of the estimated response, is determined as one of the options in the Transient Response View.

Direct Estimation of the Frequency Response

The frequency response of a linear system is the Fourier transform of its impulse response. This description of the system gives considerable engineering insight into its properties. The relation between input and output is often written

$$y(t) = G(z)u(t) + v(t)$$

where G is the transfer function and v is the additive disturbance. The function

$$G(e^{i\omega T})$$

as a function of (angular) frequency ω is then the frequency response or frequency function. T is the sampling interval. If you need more details on the different interpretations of the frequency response, consult See "The System Identification Problem" on page 3-8. in the *Tutorial* or any textbook on linear systems.

The system's frequency response is directly estimated using *Spectral Analysis* by the menu item **Estimate > Spectral Model**, and then selecting the **Estimate** button in the dialog box that opens. The result is placed on the Model

Board under the default name spa_d. The best way to examine it is to plot it using the **Model View Frequency Response**. This view offers a number of different options on how to graph the curves. The frequencies for which to estimate the response can also be selected as an option under the **Options** menu in this **View** window.

The Spectral Analysis command also estimates the spectrum of the additive disturbance *v(t)* in the system description. This estimated disturbance spectrum is examined under the **Model View** item **Noise Spectrum**.

The Spectral Analysis estimate is stored in the SITB's `freqfunc` format. If you need to further work with the estimates, you can export the model to the MATLAB workspace and retrieve the responses by the command `getff`. See `freqfunc` and `getff` in Chapter 4, "Command Reference," for more information. (A model is exported by dragging and dropping it over the **To Workspace** icon.)

Two options that affect the spectral analysis estimate can be set in the dialog box. The most important choice is a number, M, (the size of the lag window) that affects the frequency resolution of the estimates. Essentially, the frequency resolution is about $2\pi/M$ radians/(sampling interval). The choice of M is a trade-off between frequency resolution and variance (fluctuations). A large value of M gives good resolution but fluctuating and less reliable estimates. The default choice of M is good for systems that do not have very sharp resonances and may have to be adjusted for more resonant systems.

The options also offer a choice between the Blackman-Tukey windowing method spa (which is default) and a method based on smoothing direct Fourier transforms, `etfe`. `etfe` has an advantage for highly resonant systems, in that it is more efficient for large values of M. It however has the drawbacks that it requires linearly spaced frequency values, does not estimate the disturbance spectrum, and does not provide confidence intervals. The actual methods are described in more detail in Chapter 4, "Command Reference," under spa and `etfe`. To obtain the spectral analysis model for the current settings of the options, you can just type the hotkey s in the **ident** window.

Estimation of Parametric Models

The SITB supports a wide range of model structures for linear systems. They are all accessed by the menu item **Estimate > Parametric Models...** in the **ident** window. This opens up a dialog box **Parametric Models**, which contains the basic dialog for all parametric estimation as shown on the following page

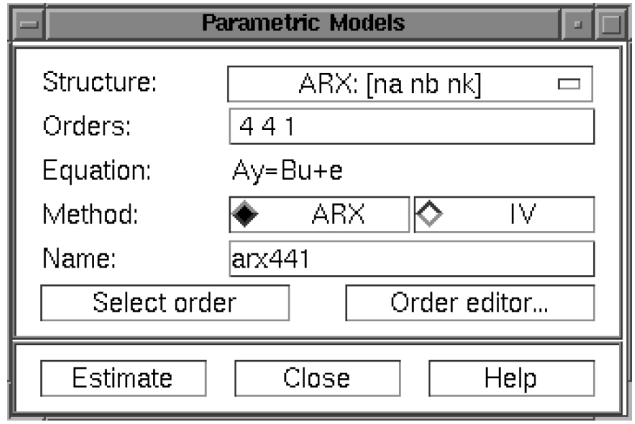


Figure 2-3: The Dialog Box for Estimating Parametric Models

The basic function of this box is as follows:

As you select **Estimate**, a model is estimated from the Working Data. The structure of this model is defined by the pop-up menu **Structure** together with the edit box **Orders**. It is given a name, which is written in the edit box **Name**.

The GUI will always suggest a default model name in the **Name** box, but you can change it to any string before selecting the **Estimate** button. (If you intend to export the model later, avoid spaces in the name.)

The interpretation of the model structure information (typically integers) in the **Order** box, depends on the selected **Structure** in the pop-up menu. This covers, typically, six choices:

- ARX models
- ARMAX model
- Output-Error (OE) models
- Box-Jenkins (BJ) models
- State-space models
- Model structure defined by Initial Model (User defined structures)

These are dealt with one by one shortly.

You can fill out the **Order** box yourself at any time, but for assistance you can select **Order Editor...** This will open up another dialog box, depending on the chosen **Structure**, in which the desired model order and structure information can be entered in a simpler fashion.

You can also enter a name of a MATLAB workspace variable in the order edit box. This variable should then have a value that is consistent with the necessary orders for the chosen structure.

NOTE: For the state-space structure and the ARX structure, several orders and combination of orders can be entered. Then all corresponding models will be compared and displayed in a special dialog window for you to select suitable ones. This could be a useful tool to select good model orders. This option is described in more detail later in this section. When it is available, a button **Order selection** is visible.

Estimation Method

A common and general method of estimating the parameters is the *prediction error approach*, where simply the parameters of the model are chosen so that the difference between the model's (predicted) output and the measured output is minimized. This method is available for all model structures. Except for the ARX case, the estimation involves an iterative, numerical search for the best fit.

To obtain information from and interact with this search, select **Iteration control...** This also gives access to a number of options that govern the search process. (See auxvar in the Chapter 4, "Command Reference,".)

For some model structures (the ARX model, and black-box state-space models) methods based on correlation are also available: Instrumental Variable (IV) and Sub-space (N4SID) methods. The choice between methods is made in the **Parametric Models** dialog box.

Resulting Models

The estimated model is inserted into the GUI's Model Board. You can then examine its various properties and compare them with other models' properties using the **Model View** plots. More about that in "Examining Models" on page 2-27.

To take a look at the model itself, double-click on the model's icon (middle/right mouse button or alt- double-click). The **Data/Model Info** window that then opens gives you information about how the model was estimated. You can then also select **Present** button, which will list the model, and its parameters with estimated standard deviations in the MATLAB command window.

If you need to work further with the model, you can export it by dragging and dropping it over the **To Workspace** icon, and then apply any MATLAB and toolbox commands to it. (See, in particular, the commands `th2ss`, `th2tf`, `th2par`, and `thd2thc` in Chapter 4, "Command Reference,".)

How to Know Which Structure and Method to Use

There is no simple way to find out "the best model structure"; in fact, for real data, there is no such thing as a "best" structure. Some routes to find good and acceptable model are described in "A Startup Identification Procedure" on page 1-12 in the introductory chapter. It is best to be generous at this point. It often takes just a few seconds to estimate a model, and by the different validation tools described in the next section, you can quickly find out if the new model is any better than the ones you had before. There is often a significant amount of work behind the data collection, and spending a few extra minutes trying out several different structures is usually worth while.

ARX Models

The Structure

The most used model structure is the simple linear difference equation

$$y(t) + a_1 y(t-1) + \ldots + a_{na} y(t-na) = b_1 u(t-nk) + \ldots + b_{nb} u(t-nk-nb+1)$$

which relates the current output *y(t)* to a finite number of past outputs *y(t-k)* and inputs *u(t-k)*.

The structure is thus entirely defined by the three integers `na`, `nb`, and `nk`. `na` is equal to the number of poles and `nb-1` is the number of zeros, while `nk` is the pure time-delay (the dead-time) in the system. For a system under sampled-data control, typically `nk` is equal to 1 if there is no dead-time.

For multi-input systems `nb` and `nk` are row vectors, where the i-th element gives the order/delay associated with the i-th input.

Entering the Order Parameters

The orders `na`, `nb`, and `nk` can either be directly entered into the edit box **Orders** in the **Parametric Models** window, or selected using the pop-up menus in the **Order Editor**.

Estimating Many Models Simultaneously

By entering any or all of the structure parameters as vectors, using MATLAB's colon notation, like `na=1:10`, etc., you define many different structures that correspond to all combinations of orders. When selecting **Estimate**, models corresponding to all of these structures are computed. A special plot window will then open that shows the fit of these models to Validation Data. By clicking in this plot, you can then enter any models of your choice into the Model Board.

Multi-input models: For multi-input models you can of course enter each of the input orders and delays as a vector. The number of models resulting from all combinations of orders and delays can however be very large. As an alternative, you may enter one vector (like `nb=1:10`) for all inputs and one vector for all delays. Then only such models are computed that have the same orders and delays from all inputs.

Estimation Methods

There are two methods to estimate the coefficients a and b in the ARX model structure:

Least Squares: Minimizes the sum of squares of the right-hand side minus the left-hand side of the expression above, with respect to a and b. This is obtained by selecting ARX as the **Method**.

Instrumental Variables: Determines a and b so that the error between the right- and left- hand sides becomes uncorrelated with certain linear combinations of the inputs. This is obtained by selecting IV in the **Method** box.

The methods are described in more detail in Chapter 4, "Command Reference," under arx and iv4.

Multi-Output Models

For a multi-output ARX structure with NY outputs and NU inputs, the difference equation above is still valid. The only change is that the coefficients *a* are NY by NY matrices and the coefficients *b* are NY by NU matrices.

The orders [NA NB NK] define the model structure as follows:

NA: an NY by NY matrix whose i-j entry is the order of the polynomial (in the delay operator) that relates the j-th output to the i-th output

NB: an NY by NU matrix whose i-j entry is the order of the polynomial that relates the j-th input to the i-th output

NK: an NY by NU matrix whose i-j entry is the delay from the j-th input to the i-th output

The **Order Editor** dialog box allows the choices

```
NA = na*ones(NY,NY)
NB = nb*ones(NY,NU)
NK = nk*ones(NY,NU)
```

where na, nb, and nk are chosen by the pop-up menus.

For tailor-made order choices, construct a matrix [NA NB NK] in the MATLAB command window and enter the name of this matrix in the **Order** edit box in the **Parametric Models** window.

Note that the possibility to estimate many models simultaneously is not available for multi-output ARX models.

See “Defining Model Structures” on page 3-29 for more information on multi-output ARX models.

ARMAX, Output-Error and Box-Jenkins Models

There are several elaborations of the basic ARX model, where different noise models are introduced. These include well known model types, such as ARMAX, Output-Error, and Box-Jenkins.

The General Structure

A general input-output linear model for a single-output system with input u and output y can be written:

$$A(q)y(t) = \sum_{i=1}^{nu} [B_i(q)/F_i(q)]u_i(t-nk_1) + [C(q)/D(q)]e(t)$$

Here u_i denotes input #i, and A, B_i, C, D, and F_i, are polynomials in the shift operator (z or q). (Don't get intimidated by this: It is just a compact way of writing difference equations; see below.)

The general structure is defined by giving the time-delays nk and the orders of these polynomials *(i.e.*, the number of poles and zeros of the dynamic model from u to y, as well as of the noise model from e to y).

The Special Cases

Most often the choices are confined to one of the following special cases:

ARX: A(q) y(t) = B(q) u(t-nk) + e(t)

ARMAX: A(q) y(t) = B(q) u(t-nk) + C(q) e(t)

OE: y(t) = [B(q)/F(q)] u(t-nk) + e(t) (Output-Error)

BJ: y(t) = [B(q)/F(q)] u(t-nk) + [C(q)/D(q)] e(t) (Box-Jenkins)

The "shift operator polynomials" are just compact ways of writing difference equations. For example the ARMAX model in longhand would be:

$$y(t)+a_1(t\text{-}1)+\ldots+a_{na}y(t\text{-}na)=b_1u(t\text{-}nk)+\ldots b_{nb}u(t\text{-}nk\text{-}nb+1)+e(t)$$
$$+c_1e(t\text{-}1)$$
$$+\ldots+e_{nc}e(t\text{-}nc)$$

Note that *A(q)* corresponds to poles that are common between the dynamic model and the noise model (useful if noise enters the system "close to" the input). Likewise *F(q)* determines the poles that are unique for the dynamics from input # i, and *D(q)* the poles that are unique for the noise.

The motivation for introducing all these model variants is to provide for flexibility in the noise description and to allow for common or different poles (dynamics) for the different inputs.

Entering the Model Structure

Use the **Structure** pop-up menu in the **Parametric Models** dialog to choose between the ARX, ARMAX, Output-Error, and Box-Jenkins structures. Note that if the Working Data set has several outputs, only the first choice is available. For time series (data with no input signal) only AR and ARMA are available among these choices. These are the time series counterparts of ARX and ARMAX.

The orders of the polynomials are selected by the pop-up menus in the **Order Editor** dialog window, or by directly entering them in the edit box **Orders** in the **Parametric Models** window. When the order editor is open, the default orders, entered as you change the model structure, are based on previously used orders.

Estimation Method

The coefficients of the polynomials are estimated using a prediction error/ Maximum Likelihood method, by minimizing the size of the error term “e” in the expression above. Several options govern the minimization procedure. These are accessed by activating **Iteration Control** in the **Parametric Models** window, and selecting **Option**.

The algorithms are further described in Chapter 4, "Command Reference," under `armax`, `auxvar`, `bj`, `oe`, and `pem`. See also “Parametric Model Estimation” on page 3-22 and “Defining Model Structures” on page 3-29.

NOTE: These model structures are available only for the scalar output case. For multi-output models, the state-space structures offer the same flexibility. Also note that it is not possible to estimate many different structures simultaneously for the input-output models.

State-Space Models

The Model Structure

The basic state-space model in innovations form can be written

$$x(t+1) = A\,x(t) + B\,u(t) + K\,e(t)$$
$$y(t) = C\,x(t) + D\,u(t) + e(t)$$

The SITB supports two kinds of parametrizations of state-space models: black-box, free parametrizations, and parametrizations tailor-made to the application. The latter is discussed below under the heading "User Defined Model Structures." First we will discuss the black-box case.

Entering Black-Box State-Space Model Structures

The most important structure index is the model order, i.e., the dimension of the state vector *x*.

Use the pop-up menu in the **Order Editor** to choose the model order, or enter it directly into the **Orders** edit box in the **Parametric Models** window. Using the other pop-up menus in the **Order Editor**, you can further affect the chosen model structure:

- Fixing K to zero gives an Output-Error method, i.e., the difference between the model's simulated output and the measured one is minimized. Formally, this corresponds to an assumption that the output disturbance is white noise.
- Fixing D to zero means that there is a delay of (at least) one sample from the input to the output. For physical systems that are driven by piece-wise constant inputs, this is a natural assumption.
- Fixing the initial value x(0) to zero means that the initial state is not estimated. Otherwise this initial state is estimated from the data. (Note that the estimated value of x(0) is something that relates to the Working Data, and is not necessarily a good value of x(0) for the Validation Data.)

Estimating Many Models Simultaneously

By entering a vector for the model order, using MATLAB's colon notation, (such as "1:10") all indicated orders will be computed using a preliminary method. You can then enter models of different orders into the Model Board by clicking in a special graph that contains information about the models.

Estimation Methods

There are two basic methods for the estimation:

`PEM`: Is a standard prediction error/maximum likelihood method, based on iterative minimization of a criterion. The iterations are started up at parameter values that are computed from `n4sid`. The parametrization of the matrices A, B, C, D, and K follows a default canonical form. The search for minimum is controlled by a number of options. These are accessed from the **Option** button in the **Iteration Control** window.

`N4SID`: Is a subspace-based method that does not use iterative search. The quality of the resulting estimates may significantly depend on an auxiliary order (like a prediction horizon). This auxiliary order can be given within parenthesis after the order in the edit box **Orders** (like “4 (9)”). If the auxiliary order is not given, a default value is used.

If the auxiliary order is given as a vector (like “4 (5:17)”), models for all these auxiliary orders are computed and evaluated using the Working Data set. The model that gives the best fit is chosen. A figure is shown that illustrates the fit as a function of auxiliary order. (The fit is a simulation error fit if K=0 is chosen, otherwise a one-step ahead prediction error fit.)

See `n4sid` and `pem` in Chapter 4, "Command Reference," for more information.

User Defined Model Structures

State-Space Structures

The SITB supports user-defined linear state-space models of arbitrary structure. Using the command `ms2th`, known and unknown parameters in the A, B, C, D, K, and X0 matrices can be easily defined both for discrete- and continuous-time models. The command `mf2th` allows you to use a completely arbitrary structure, defined by an M-file. `fixpar` and `unfixpar` can be used to manipulate structures. See Chapter 4, "Command Reference," and “Defining Model Structures” on page 3-29

To use these structures in conjunction with the GUI, just define the appropriate structure in the MATLAB command window. Then use the **Structure** pop-up menu to select **By Initial Model** and enter the variable name of the structure in the edit box **Initial Model** in the **Parametric Models** window and select **Estimate**.

Any Model Structure

Arbitrary model structures can be defined using several commands in the System Identification Toolbox:

- `poly2th`: Creates Input-output structures for single-output models
- `ms2th`: Creates Linear State-space models with arbitrary, free parameters
- `mf2th`: Creates completely arbitrary parametrizations of linear systems
- `arx2th`: Creates multivariable ARX structures
- `fixpar`/`unfixpar`: Modifies structures by fixing or unfixing parameters

In addition, all estimation commands create model structures in terms of the resulting models.

Enter the name of any model structure in the box **Orders (or Initial model)** in the window **Parametric Models** and then select **Estimate**. Then the parameters of the model structure are adjusted to the chosen Working Data set. The method is a standard prediction error/maximum likelihood approach that iteratively searches for the minimum of a criterion. Options that govern this search are accessed by the **Option** button in the **Iteration Control** window.

The name of the initial model must be a variable either in the workspace or in the Model Board. In the latter case you can just drag and drop it over the **Orders/Initial model** edit box.

4. Examining Models

Having estimated a model is just a first step. It must now be examined, compared with other models, and tested with new data sets. This is primarily done using the six **Model View** functions, at the bottom of the main **ident** window:

- Frequency response
- Transient response
- Zeros and poles
- Noise spectrum
- Model output
- Model residuals

In addition, you can double-click on the model's icon to get **Text Information** about the model. Finally, you can export the model to the MATLAB workspace and use any commands for further analysis and model use.

Views and Models

The basic idea is that if a certain **View** window is open (checked), then *all models* in the Model Summary Board that are selected will be represented in the window. The curves in the **View** window can be clicked in and out by selecting and deselecting the models in an online fashion. You select and deselect a model by clicking on its icon. An selected model is marked with a thicker line in its icon.

On color screens, the curves are color coded along with the model icons in the Model Board. Before printing a plot it might be a good idea to separate the line styles (menu item under **Style**). This could also be helpful on black and white screens.

Note that models that are obtained by spectral analysis only can be represented as frequency response and noise spectra, and that models estimated by correlation analysis only can be represented as transient response.

The Plot Windows

The six views all give similar plot windows, with several common features. They have a common menu bar, which covers some basic functions.

First of all, note that there is a zoom function in the plot window. By dragging with the left mouse button down, you can draw rectangles, which will be enlarged when the mouse button is released. By double-clicking, the original axis scales are restored. For plots with two axes, the x-axes scales are locked to each other. A single click on the left mouse button zooms in by a factor of two, while the middle button zooms out. The zoom function can be deactivated if desired. Just select the menu item **Zoom** under **Style**.

Second, by pointing to any curve in the plot, and pressing the right mouse button, the curve will be identified with model name and present coordinates.

The common menu bar covers the following functions:

File

File allows you to copy the current figure to another, standard MATLAB figure window. This might be useful, e.g., when you intend to print a customized plot. Other **File** items cover printing the current plot and closing the plot window.

Options

Options first of all covers actions for setting the axes scaling. This menu item also gives a number of choices that are specific for the plot window in question, like a choice between step response or impulse response in the **Transient response** window.

A most important option is the possibility to show confidence intervals. Each model property has some uncertainty. This can also be estimated from data. By checking **Show confidence intervals**, a confidence region around the nominal curve (model property) will be marked (by dash-dotted lines). The level of confidence can also be set under this menu item.

NOTE: Confidence intervals are supported for most models and properties, except models estimated using `n4sid` and `etfe`, and the k-step ahead prediction-property.

Style

The style menu gives access to various ways of affecting the plot. You can add gridlines, turn the zoom on and off, and change the linestyles. The menu also covers a number of other options, like choice of units and scale for the axis.

Channel

For multivariate systems, you can choose which input-output channel to examine. The current choice is marked in the figure title.

Help

The **Help** menu has a number of items, which explain the plot and its options.

Frequency Response and Disturbance Spectra

All linear models that are estimated can be written in the form

$$y(t) = G(z)u(t) + v(t)$$

where *G(z)* is the (discrete-time) transfer function of the system and *v(t)* is an additive disturbance. The frequency response or frequency function of the system is the complex-valued function $G(e^{i\omega T})$ viewed as a function of angular frequency ω.

This function is often graphed as a Bode diagram, i.e., the logarithm of the amplitude (the absolute value) $G(e^{i\omega T})$ as well as the phase arg (the argument) $G(e^{i\omega T})$ are plotted against the logarithm of frequency ω in two separate plots. These plots are obtained by checking the **Model View Frequency Response** in the main **ident** window.

The estimated spectrum of the disturbance *v* is plotted as a power spectrum by choosing the **Model View Noise Spectrum**.

If the data is a time series *y* (with no input *u*), then the spectrum of *y* is plotted under **Noise Spectrum**, and no frequency functions are given.

Transient Response

Good and simple insight into a model's dynamic properties is obtained by looking at its step response or impulse response. This is the output of the model when the input is a step or an impulse. These responses are plotted when the **Model View Transient Response** is checked.

It is quite informative to compare the transient response of a parametric model, with the one that was estimated using correlation analysis. If there is good agreement between the two, you can be quite confident that some essentially correct features have been picked up. It is useful to check the confidence intervals around the responses to see what "good agreement" could mean quantitatively.

Many models provide a description of the additive disturbance *v(t)*:

$$v(t) = H(z)e(t)$$

Here *H(z)* is a transfer function that describes how the disturbance *v(t)* can be thought of as generated by sending white noise *e(t)* through it. To display the properties of *H*, you can choose channels (in the **Channel** menu) that have noise components as inputs.

Poles and Zeros

The poles of a system are the roots of the denominator of the transfer function *G(z)*, while the zeros are the roots of the numerator. In particular the poles have a direct influence on the dynamic properties of the system.

The poles and zeros of *G* (and *H*) are plotted by choosing the **Model View Poles and Zeros**.

It is useful to turn on the confidence intervals in this case. They will clearly reveal which poles and zeros could cancel (their confidence regions overlap). That is an indication that a lower order dynamic model could be used.

For multivariable systems it is the poles and zeros of the individual input/output channels that are displayed. To obtain the so called transmission zeros, you will have to export the model, extract a state-space description by `th2ss`, and then apply the command `tzero` from the Control System Toolbox.

Compare Measured and Model Output

A very good way of obtaining insight into the quality of a model is to simulate it with the input from a fresh data set, and compare the simulated output with the measured one. This gives a good feel for which properties of the system have been picked up by the model, and which haven't.

This test is obtained by checking the **Model View Model Output**. Then the data set currently in the **Validation Data** box will be used for the comparison.

The fit will also be displayed. This is computed as the root of the mean square value of the difference between measured and simulated output.

If the model is unstable, or has integration or very slow time constants, the levels of the simulated and the measured output may drift apart, even for a model that is quite good (at least for control purposes). It is then a good idea to evaluate the model's predicted output rather than the simulated one. With a *prediction horizon* of k, the k-step ahead predicted output is then obtained as follows:

The predicted value *y(t)* is computed from all available inputs $u(s)(s \le t)$ (used according to the model) and all available outputs up to time *t-k*, $y(s)(s \le t-k)$. The simulation case, where no past outputs at all are used, thus formally corresponds to $k=\infty$. To check if the model has picked up interesting dynamic properties, it is wise to let the predicted time horizon (kT, T being the sampling interval) be larger than the important time constants.

Note here that different models use the information in past output data in their predictors in different ways. This depends on the noise model. For example, so called Output-Error models (obtained by fixing K to zero for state-space models and setting `na=nc=nd=0` for input output models, see the previous section) do not use past outputs at all. The simulated and the predicted outputs, for any value of k, thus coincide.

Residual Analysis

In a model

$$y(t) = G(z)u(t) + H(z)e(t)$$

the noise source *e(t)* represents that part of the output that the model could not reproduce. It gives the "left-overs" or, in Latin, the *residuals*. For a good model, the residuals should be independent of the input. Otherwise, there would be more in the output that originates from the input and that the model has not picked up.

To test this independence, the cross-correlation function between input and residuals is computed by checking the **Model View Model Residuals**. It is wise to also display the confidence region for this function. For an ideal model the correlation function should lie entirely between the confidence lines for positive lags. If, for example, there is a peak outside the confidence region for lag *k*, this means that there is something in the output *y(t)* that originates from *u(t-k)* and that has not been properly described by the model. The test is carried

out using the Validation Data. If these were not used to estimate the model, the test is quite tough. See also “Model Structure Selection and Validation” on page 3-49.

For a model also to give a correct description of the noise properties (i.e., the transfer function *H*), the residuals should be mutually independent. This test is also carried out by the view **Model Residuals**, by displaying the auto-correlation function of the residuals (excluding lag zero, for which this function by definition is one). For an ideal model, the correlation function should be entirely inside the confidence region.

Text Information

By double-clicking (middle/right mouse button or alt-double-click) on the model icon, a **Data/model Info** dialog box opens, which contains some basic information about the model. It also gives a diary of how the model was created, along with the notes that originally were associated with the estimation data set. At this point you can do a number of things:

Present

Selecting the **Present** button displays details of the model in the MATLAB command window. The model's parameters along with estimated standard deviations are displayed, as well as some other notes.

Modify

You can simply type in any text you want anywhere in the **Diary and Notes** editable text field of the dialog box. You can also change the name of the model just by editing the text field with the model name. The color, which the model is associated with in all plots, can also be edited. Enter RGB-values or a color name (like 'y') in the corresponding box.

Further Analysis in the MATLAB Workspace

Any model and data object can be exported to the MATLAB workspace by dragging and dropping its icon over the **To Workspace** box in the **ident** window.

Once you have exported the model to the workspace, there are lots of commands by which you can further transform it, examine it, and convert it to other formats for use in other toolboxes. Some examples of such commands are

`thd2thc`	Transform to continuous time.
`th2ss`	Convert to state-space representation.
`th2tf`	Convert to transfer function form.
`th2poly`	Convert to polynomial input/output form.

Also, if you need to prepare specialized plots that are not covered by the **Views**, all the SITB commands for computing and extracting simulations, frequency functions, zeros and poles, etc., are available. See Chapter 3, "Tutorial," and Chapter 4, "Command Reference,".

5. Some Further GUI Topics

This section discusses a number of different topics.

Mouse Buttons and Hotkeys

The GUI uses three mouse buttons. If you have fewer buttons on your mouse, the actions associated with the middle and right mouse buttons are obtained by shift-click, alt-click or control-click, depending on the computer.

The Main ident Window

In the main **ident** window the mouse buttons are used to drag and drop, to select/deselect models and data sets, and to double-click to get text information about the object. You can use the left mouse button for all of this. A certain speed-up is obtained if you use the left button for dragging and dropping, the right one for selecting models and data sets, and the middle one for double-clicking. On a slow machine a double-click from the left button might not be recognized.

The **ident** window also has a number of hotkeys**.** By pressing a keyboard letter when it is the current window, some functions can be quickly activated. These are

- s: Computes **Spectral Analysis Model** using the current options settings. (These can be changed in the dialog window that opens when you choose **Spectral Model** in the **Estimate** pop-up menu.)
- c: Computes **Correlation Analysis Model** using the current options settings. (These can be changed in the dialog window that opens when you choose **Correlation Model** in the **Estimate** pop-up menu.)
- q: Computes the models associated with the **Quickstart**.
- d: Opens a dialog window for importing **Data** that are in the SITB data format.

Plot Windows

In the various plot windows the action of the mouse buttons depends on whether the zoom is activated or not:

Zoom Active: Then the left and middle mouse buttons are associated with the zoom functions as in the standard MATLAB zoom. Left button zooms in and the middle one zooms out. In addition, you can draw rectangles with the left

button, to define the area to be zoomed. Double-clicking restores the original plot. The right mouse button is associated with special GUI actions that depend on the window. In the **View** plots, the right mouse button is used to identify the curves. Point and click on a curve, and a box will display the name of the model/data set that the curve is associated with, and also the current coordinate values for the curve. In the **Model Selection** plots the right mouse button is used to inspect and select the various models. In the **Prefilter** and **Data Range** plots, rectangles are drawn with this mouse button down, to define the selected range.

Zoom not active: The special GUI functions just mentioned are obtained by any mouse button.

The zoom is activated and deactivated under the menu item **Style.** The default setting differs between the plots. Don't activate the zoom from the command line! That will destroy the special GUI functions. (If you happen to do so anyway, "quit" the window and open it again.)

Troubleshooting in Plots

The function **Auto-range**, which is found under the menu item **Options**, sets automatic scales to the plots. It is also a good function to invoke when you think that you have lost control over the curves in the plot. (This may happen, for example, if you have zoom in a portion of a plot and then change the data of the plot).

If the view plots don't respond the way you expect them to, you can always "quit" the window and open it again. By quit here we mean using the underlying window system's own quitting mechanism, which is called different things in the different platforms. This action will not be acknowledged by the corresponding check box in the **ident** window, so you will have to first uncheck, and then check it. The normal way to close a window is to use the **Close** function under the menu item **File**, or to uncheck the corresponding check box.

Layout Questions and idprefs.mat

The GUI comes with a number of preset defaults. These include the window sizes and positions, the colors of the different models, and the default options in the different **View** windows.

The window sizes and positions, as well as the options in the plot windows, can of course be changed during the session in the standard way. If you want the

GUI to start with your current window layout and current plot options, select menu item

```
Options > Save preferences
```

in the main **ident** window. This saves the information in a file `idprefs.mat`. This file also stores information about the four most recent sessions with **ident**. This allows the session **File** menu to be correctly initialized. The session information is automatically stored upon exit. The layout and preference information is only saved when the indicated option is selected.

The file `idprefs.mat` is created the first time you open the GUI. It is by default stored in the same directory as your `startup.m` file. If this default does not work, a warning is issued. This can be ignored, but then session and preference information cannot be stored.

To change or select a directory for `idprefs.mat`, use the command `midprefs`. See Chapter 4, "Command Reference," for details.

To change model colors and default options to your own customized choice, make a copy of the M-file `idlayout.m` to your own directory (which should be *before* the basic `ident` directory in the `MATLABPATH`), and edit it according to its instructions.

Customized Plots

If you need to prepare hardcopies of your plots with specialized texts, titles and so on, make a copy of the figure first, using **Copy Figure** under the **File** menu item. This produces a copy of the current figure in a standard MATLAB figure format.

For plots that are not covered by the **View** windows, (like, e.g., Nyquist plots), you have to export the model to the MATLAB workspace and construct the plots there.

Import from and Export to Workspace

As you export a model or a data set to the MATLAB workspace by dropping its icon over the To Workspace icon, it will become a workspace variable with the same name as in **ident**, say `mymodel`. At the same time another workspace variable will be created with the name `mymodel_info`. This contains all diary and note information. This will be quite useful if you want to manipulate the

model using MATLAB commands and then import it back into ident. The use is illustrated by the following example:

Export model with name `mymodel` to workspace. Execute the following at the MATLAB command line:

```
mymodelc=thd2thc(mymodel);
     % converting the model to continuous time
mymodelc_info=str2mat(mymodel_info,...
     'mymodelc=thd2thc(mymodel)');
     % adding the info of how the new variable was created
```

Now select pop-up menu **Model>Import**, enter the variable name `mymodelc`, and select **Import**.

This will enter the continuous-time model into **ident** along with the relevant information.

What Cannot be Done Using the GUI

The GUI covers primarily everything you would like to do to examine data, estimate models and evaluate and compare models. It does not cover

- Generation (simulation) of data sets
- Model creation (other than by estimation)
- Model manipulation and conversions
- Recursive (on-line) estimation algorithms

Check Chapter 3, "Tutorial," as well as the headings **Simulation and Prediction**, **Model Structure Creation**, **Manipulating Model Structures**, **Model Conversions**, and **Recursive Parameter Estimation** in the beginning of Chapter 4, "Command Reference," to see what M-files are available in the toolbox for these functions.

Note that at any point you can export a data set or a model to the MATLAB workspace (by dragging and dropping its icon on the **To Workspace** icon). There you can modify and manipulate it any way you want and then import it back into **ident**. You can, for example, construct a continuous-time model from an estimated discrete-time one (using `thd2thc`) and then use the model views to compare the two.

3

Tutorial

This chapter has two purposes. It describes the commands of the System Identification Toolbox, their syntax and use. If you use the graphical user interface (GUI), you will not have to bother about these aspects. The chapter also describes the underlying methods and algorithms used. The commands that are not reached from the GUI, i.e., the recursive algorithms and more advanced model structure definitions, are also treated here.

1. The Toolbox Commands

It may be useful to recognize several *layers* of the System Identification Toolbox. Initially concentrate on the first layer of basic tools, which contains the commands from the System Identification Toolbox that any user must master. You can proceed to the next levels whenever an interest or the need from the applications warrants it. The layers are described in the following paragraphs:

Layer 1: Basic Tools. The first layer contains the basic tools for correlation and spectral analysis, and the estimation of so-called ARX models using the least-squares method. It also contains the basic tools for examining the obtained models. The commands are:

```
ar, arx, bodeplot, compare, cra, dtrend, ffplot
idsim, present, resid, spa, th2arx, th2ff, th2zp, zpplot
```

The corresponding background is given in the following sections: 3,4,5,6,7 and 8 of this chapter.

Layer 2: Model Structure Selection. The second layer of the toolbox contains some useful techniques to select orders and delays, and to estimate more general input-output model structures. The commands are:

```
armax, arxstruc, bj, etfe, idfilt, iv4, n4sid
oe, poly2th, selstruc, th2poly
```

Note that the commands `arx` and `iv4` also apply to multi-input, multi-output systems. The corresponding background is given in the sections 3, 5, 6 ,8 and 9 of this chapter.

Layer 3: More Methods To Examine Models and Multi-Input Systems .
This third layer contains transformations between continuous and discrete time, and functions for estimating completely general multi-input model structures. The commands are:

```
pe, pem, predict, th2ss, th2tf, thc2thd, thd2thc
```

The corresponding background is given in section 7 of this chapter.

Layer 4: Recursive Identification. Recursive (adaptive, online) methods of parameter estimation are covered by the commands:

```
rarmax, rarx, rbj, roe, rpem, rplr
```

The corresponding background is given “Recursive Parameter Estimation” on page 3-61.

Layer 5: State-Space Modeling. To build models in state-space form (both continuous and discrete time), and to handle more advanced multi-output structures, there are the commands:

```
canstart, fixpar, mf2th, modstruc, ms2th, unfixpar
```

The corresponding theory is covered in sections 3, 5, 6, and 8 of this chapter.

(See the beginning of the Chapter 4, "Command Reference," for a complete list of available functions.)

2. An Introductory Example to Command Mode

A demonstration M-file called `iddemo.m` provides several examples of what might be typical sessions with the System Identification Toolbox. To start the demo, execute `iddemo` from inside MATLAB.

Before giving a formal treatment of the capabilities and possibilities of the toolbox, this example is designed to get you using the software quickly. This example is essentially the same as demo #2 in `iddemo`. You may want to invoke MATLAB at this time, execute `iddemo1`, and follow along.

Data have been collected from a laboratory scale process. (Feedback's Process Trainer PT326; see page 440 in Ljung, 1987 and "Reading More About System Identification" on page 1-18 for references.) The process operates much like a common hand-held hair dryer. Air is blown through a tube after being heated at the inlet to the tube. The input to the process is the power applied to a mesh of resistor wires that constitutes the heating device. The output of the process is the air temperature at the outlet measured in volts by a thermocouple sensor.

One thousand input-output data points were collected from the process as the input was changed in a random fashion between two levels. The sampling interval is 80 ms. The data were loaded into MATLAB in ASCII form and are now stored as the vectors `y2` (output) and `u2` (input) in the file `dryer2.mat`.

First load the data:

```
load dryer2
```

This example selects the first 300 data points for building a model. For convenience, the input-output vectors are merged into a matrix:

```
z2 = [y2(1:300) u2(1:300)];
```

Take a look at the data,

```
idplot(z2)
```

The toolbox makes frequent use of default arguments. Default values are used when trailing arguments are omitted. In the case above, by default, all data points are graphed and the sampling interval is one time unit.

You can select the values between sample numbers 200 and 300 for a close-up, and at the same time obtain correct time scales, with

```
idplot(z2,200:300,0.08)
```

Remove the constant levels and make the data zero mean with

```
z2 = dtrend(z2);
```

Now, fit to the data a model of the form:

$$y(t) + a_1 y(t-T) + a_2 y(t-2T) = b_1 u(t-3T) + b_2 u(t-4T) \qquad (2.1)$$

where T is the sampling interval (here 0.08 seconds). This model, known as an ARX model, tries to explain or compute the value of the output at time t, given previous values of y and u. Later on, especially in Section 8, are discussions of how to decide upon suitable model orders (structures).

The best values of the coefficients a_1, a_2, b_1 and b_2 can be computed with

```
th = arx(z2,[2 2 3]);
```

The numbers in the second argument tell `arx` to find a model (2.1) with two *a*-parameters, two *b*-parameters, and three delays. The result is stored in the matrix `th` in a somewhat coded form. To specify the actual sampling interval, enter

```
th = sett(th,0.08);
```

There are several ways to display and illustrate the computed model. With

```
present(th)
```

the coefficient values of (2.1) and their estimated standard deviations are presented on the screen.

Next, you might ask how to evaluate how well the model fits the data. A simple test is to run a simulation whereby real input data is fed into the model, and compare the simulated output with the actual measured output. For this, select a portion of the data that was not used to build the model, for example, from sample 700 to 900:

```
u = dtrend(u2(700:900));
y = dtrend(y2(700:900));
```

```
ysim = idsim(u,th);
plot([y(100:200) ysim(100:200)])
```

Note that the function compare does this sequence of commands more efficiently.

You see that the model is quite capable of describing the system, even for data that were not used in calculating the fit.

To compute and graph the poles and zeros of the model, use

```
zpth = th2zp(th);
zpplot(zpth)
```

If you want to know the frequency response, you can compute the frequency function of the model and present it as a Bode plot by entering

```
gth = th2ff(th);
bodeplot(gth)
```

Compare this transfer function with a transfer function obtained from a nonparametric, spectral analysis method. Such an estimate is obtained directly from the data

```
gs = spa(z2);
gs = sett(gs,0.08);
```

The sampling interval, 0.08, is set by the second command in order to obtain correct frequency scales. The function spa also allows you to select window sizes, frequency ranges, etc. All these have here been given default values.

You can compare the estimate of the transfer function obtained by spectral analysis and the one obtained from the model (2.1) with

```
bodeplot([gs gth])
```

The agreement is quite good.

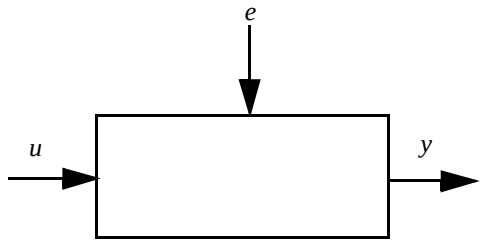
Finally, plot the step response of the model. The model comes with an estimate of its own uncertainty. Ten different step responses are computed and graphed. They correspond to “possible” models, drawn from the distribution of the true system (according to our model):

```
step = ones(30,1);
idsimsd(step,th)
```

3. The System Identification Problem

This section discusses different basic ways to describe linear dynamic systems and also the most important methods for estimating such models.

Impulse Responses, Frequency Functions, and Spectra



The basic input-output configuration is depicted in the figure above. Assuming unit sampling interval, there is an input signal

$$u(t);\ t=1,2,\ldots\ldots,N$$

and an output signal

$$y(t);\ t=1,2,\ldots\ldots,.N$$

Assuming the signals are related by a linear system, the relationship can be written

$$y(t) = G(q)u(t) + v(t) \tag{3.1}$$

where q is the shift operator and $G(q)u(t)$ is short for

$$G(q)u(t) = \sum_{k=1}^{\infty} g(k)u(t-k) \tag{3.2}$$

and

$$G(q) = \sum_{k=1}^{\infty} g(k)q^{-k}; \qquad q^{-1}u(t) = u(t-1) \tag{3.3}$$

The numbers $\{g(k)\}$ are called the *impulse response* of the system. Clearly, $g(k)$ is the output of the system at time k if the input is a single (im)pulse at time zero. The function $G(q)$ is called the *transfer function* of the system. This function evaluated on the unit circle ($q = e^{i\omega}$) gives the *frequency function*

$$G(e^{i\omega}) \tag{3.4}$$

In (3.1) $v(t)$ is an additional, unmeasurable disturbance (noise). Its properties can be expressed in terms of its (auto) spectrum

$$\Phi_v(\omega) \tag{3.5}$$

which is defined by

$$\Phi_v(\omega) = \sum_{\tau=-\infty}^{\infty} R_v(\tau)e^{-iwt} \tag{3.6}$$

where $R_v(\tau)$ is the covariance function of $v(t)$:

$$R_v(\tau) = Ev(t)v(t-\tau) \tag{3.7}$$

and E denotes mathematical expectation. Alternatively, the disturbance $v(t)$ can be described as filtered white noise:

$$v(t) = H(q)e(t) \tag{3.8}$$

where $e(t)$ is white noise with variance λ and

$$\Phi_v(\omega) = \lambda\left|H(e^{i\omega})\right|^2 \tag{3.9}$$

Equations (3.1) and (3.8) together give a *time domain description* of the system:

$$y(t) = G(q)u(t) + H(q)e(t) \tag{3.10}$$

while (3.4) and (3.5) constitute a *frequency domain description*:

$$G(e^{i\omega}); \qquad \Phi_v(\omega) \tag{3.11}$$

The impulse response (3.3) and the frequency domain description (3.11) are called *nonparametric model descriptions* since they are not defined in terms of a finite number of parameters. The basic description (3.10) also applies to the multivariable case; i.e., to systems with several (say *nu*) input signals and several (say *ny*) output signals. In that case $G(q)$ is an *ny* by *nu* matrix while $H(q)$ and $\Phi_v(\omega)$ are *ny* by *ny* matrices.

Polynomial Representation of Transfer Functions

Rather than specifying the functions *G* and *H* in (3.10) in terms of functions of the frequency variable ω, you can describe them as rational functions of q^{-1} and specify the numerator and denominator coefficients in some way.

A commonly used parametric model is the ARX model that corresponds to

$$G(q) = q^{-nk}\ \frac{B(q)}{A(q)}; \qquad H(q) = \frac{1}{A(q)} \tag{3.12}$$

where *B* and *A* are polynomials in the delay operator q^{-1}:

$$A(q) = 1 + a_1 q^{-1} + \ldots\ldots + a_{na} q^{-na} \tag{3.13}$$

$$B(q) = b_1 + b_2 q^{-1} + \ldots\ldots + b_{nb} q^{-nb+1} \tag{3.14}$$

Here, the numbers *na* and *nb* are the orders of the respective polynomials. The number *nk* is the number of delays from input to output. The model is usually written

$$A(q)y(t) = B(q)u(t-nk) + e(t) \tag{3.15}$$

or explicitly

$$y(t) + a_1 y(t-1) + \ldots\ldots + a_{na} y(t-na) = \tag{3.16}$$
$$b_1 u(t-nk) + b_2 u(t-nk-1) + \ldots\ldots + b_{nb} u(t-nk-nb+1) + e(t)$$

Note that (3.15) -(3.16) apply also to the multivariable case, where $A(q)$ and the coefficients a_i become ny by ny matrices, $B(q)$ and the coefficients b_i become ny by nu matrices.

Another very common, and more general, model structure is the ARMAX structure

$$A(q)y(t) = B(q)u(t-nk) + C(q)e(t) \tag{3.17}$$

Here, $A(q)$ and $B(q)$ are as in (3.13)-(3.14), while

$$C(q) = 1 + c_1 q^{-1} + \ldots + c_{nc} q^{-nc} \tag{3.18}$$

An *Output-Error* (OE) structure is obtained as

$$y(t) = \frac{B(q)}{F(q)} u(t-nk) + e(t) \tag{3.19}$$

with

$$F(q) = 1 + f_1 q^{-1} + \ldots + f_{nf} q^{-nf} \tag{3.20}$$

The so-called *Box-Jenkins* (BJ) model structure is given by

$$y(t) = \frac{B(q)}{F(q)} u(t-nk) + \frac{C(q)}{D(q)} e(t) \tag{3.21}$$

with

$$D(q) = 1 + d_1 q^{-1} + \ldots + d_{nd} q^{-nd} \tag{3.22}$$

All these models are special cases of the general parametric model structure:

$$A(q)y(t) = \frac{B(q)}{F(q)}u(t-nk) + \frac{C(q)}{D(q)}e(t) \tag{3.23}$$

The variance of the white noise $\{e(t)\}$ is assumed to be λ.

Within the structure of (3.23), virtually all of the usual linear black-box model structures are obtained as special cases. The ARX structure is obviously obtained for $nc = nd = nf = 0$. The ARMAX structure corresponds to $nf = nd = 0$. The ARARX structure (or the "generalized least-squares model") is obtained for $nc = nf = 0$, while the ARARMAX structure (or "extended matrix model") corresponds to $nf = 0$. The Output-Error model is obtained with $na = nc = nd = 1$, while the Box-Jenkins model corresponds to $na = 0$. (See Section 4.2 in Ljung (1987) for a detailed discussion.)

The same type of models can be defined for systems with an arbitrary number of inputs. They have the form

$$A(q)y(t) = \frac{B_1(q)}{F_1(q)}u_1(t-nk_1) + \ldots + \frac{B_{nu}(q)}{F_{nu}(q)}u_{nu}(t-nk_{nu}) + \frac{C(q)}{D(q)}e(t)$$

State-Space Representation of Transfer Functions

A common way of describing linear systems is to use the *state-space form*:

$$x(t+1) = Ax(t) + Bu(t) \tag{3.24a}$$

$$y(t) = Cx(t) + Du(t) + v(t) \tag{3.24b}$$

Here the relationship between the input $u(t)$ and the output $y(t)$ is defined via the nx-dimensional *state vector* $x(t)$. In transfer function form (3.24) corresponds to (3.1) with

$$G(q) = C(qI_{nx} - A)^{-1}B + D \tag{3.25}$$

Here I_{nx} is the nx by nx identity matrix. Clearly (3.24) can be viewed as one way of parametrizing the transfer function: Via (3.25) $G(q)$ becomes a function of the elements of the matrices A, B, C, and D.

To further describe the character of the noise term $v(t)$ in (3.24) a more flexible *innovations* form of the state-space model can be used:

$$x(t+1) = Ax(t) + Bu(t) + Ke(t) \tag{3.26a}$$

$$y(t) = Cx(t) + Du(t) + e(t) \tag{3.26b}$$

This is equivalent to (3.10) with $G(q)$ given by (3.25) and $H(q)$ by

$$H(q) = C(qI_{nx} - A)^{-1}K + I_{ny} \tag{3.27}$$

Here ny is the dimension of $y(t)$ and $e(t)$.

It is often possible to set up a system description directly in the innovations form (3.26). In other cases, it might be preferable to describe first the nature of disturbances that act on the system. That leads to a stochastic state-space model:

$$x(t+1) = Ax(t) + Bu(t) + w(t) \tag{3.28a}$$

$$y(t) = Cx(t) + Du(t) + e(t) \tag{3.28b}$$

where $w(t)$ and $e(t)$ are stochastic processes with certain covariance properties. In stationarity and from an input-output view, (3.28) is equivalent to (3.26) if the matrix K is chosen as the steady-state *Kalman gain*. How to compute K from (3.28) is described in the Control System Toolbox.

Continuous-Time State-Space Models

It is often easier to describe a system from physical modeling in terms of a continuous-time model. The reason is that most physical laws are expressed in continuous time as differential equations. Therefore, physical modeling typically leads to state-space descriptions like

$$\dot{x}(t) = Fx(t) + Gu(t) \tag{3.29a}$$

$$y(t) = Hx(t) + Du(t) + v(t) \tag{3.29b}$$

Here, $\dot{x}$ means the time derivative of x. If the input is piece-wise constant over time intervals $kT \le t < (k+1)T$, then the relationship between $u[k] = u(kT)$ and $y[k] = y(kT)$ can be exactly expressed by (3.24) by taking

$$A = e^{FT}; B = \int_{\tau=0}^{T} e^{F\tau} G d\tau; C = H \tag{3.30}$$

and associate $y(t)$ with $y[t]$, etc.

If you start with a continuous-time innovations form

$$\dot{x}(t) = Fx(t) + Gu(t) + \tilde{K}e(t) \tag{3.31a}$$

$$y(t) = Hx(t) + Du(t) + e(t) \tag{3.31b}$$

the discrete-time counterpart is given by (3.26) where still the relationships (3.30) hold. The exact connection between $\tilde{K}$ and K is somewhat more complicated, though. An *ad hoc* solution is to use

$$K = \int_{\tau=0}^{T} e^{F\tau} \tilde{K} d\tau; \tag{3.32}$$

in analogy with G and B. This is a good approximation for short sampling intervals T.

Estimating Impulse Responses

Consider the description (3.1). Suppose that the input $u(t)$ is white noise; i.e., its covariance function is

$$R_u(\tau) = Eu(t+\tau)u(t) = \begin{cases} \lambda & if \quad \tau = 0 \\ 0 & else \end{cases}$$

Then, the cross covariance function between the input and the output is

$$R_{yu}(\tau) = Ey(t+\tau)u(t) = \lambda g(\tau)$$

where $g(\tau)$ is the impulse response of the system. In this case, it can easily be estimated as

$$\hat{g}(\tau) = \frac{1}{\lambda N}\sum_{t=1}^{N} y(t+\tau)u(t) \tag{3.33}$$

based on the observed input-output data.

If the input is not white, a whitening filter $L(q)$ can be determined for it, so that the filtered sequence

$$u_F(t) = L(q)u(t) \tag{3.34}$$

is approximately white. By filtering the output sequence through the same filter and computing (3.33) based on the filtered data, a natural estimate of the impulse response coefficients $g(k)$ is obtained. This procedure is known as *correlation analysis*.

Estimating Spectra and Frequency Functions

This section describes methods that estimate the frequency functions and spectra (3.11) directly. The cross-covariance function $R_{yu}(\tau)$ between *y(t)* and *u(t)* was defined above. Its Fourier transform, the cross spectrum, $\Phi_{yu}(\omega)$ is defined analogously to (3.6). Provided that the input $u(t)$ is independent of $v(t)$, the relationship (3.1) implies the following relationships between the spectra:

$$\Phi_y(\omega) = \left|G(e^{i\omega})\right|^2 \Phi_u(\omega) + \Phi_v(\omega)$$

$$\Phi_{yu}(\omega) = G(e^{i\omega})\Phi_u(\omega)$$

By estimating the various spectra involved, the frequency function and the disturbance spectrum can be estimated as follows:

Form estimates of the covariance functions (as defined in (3.7)) $\hat{R}_y(\tau)$, $\hat{R}_{yu}(\tau)$, and $\hat{R}_u(\tau)$, using

$$\hat{R}_{yu}(\tau) = \frac{1}{N}\sum_{t=1}^{N} y(t+\tau)u(t) \tag{3.35}$$

and analog expressions for the others. Then, form estimates of the corresponding spectra

$$\hat{\Phi}_y(\omega) = \sum_{\tau=-M}^{M} \hat{R}_y(\tau)W_M(\tau)e^{-i\omega\tau} \tag{3.36}$$

and analogously for Φ_u and Φ_{yu}. Here $W_M(\tau)$ is the so-called *lag window* and M is the width of the lag window. The estimates are then formed as

$$\hat{G}_N(e^{i\omega}) = \frac{\hat{\Phi}_{yu}(\omega)}{\hat{\Phi}_u(\omega)}; \qquad \hat{\Phi}_v(\omega) = \hat{\Phi}_y(\omega) - \frac{\left|\hat{\Phi}_{yu}(\omega)\right|^2}{\hat{\Phi}_u(\omega)} \tag{3.37}$$

This procedure is known as *spectral analysis*. (See Chapter 6 in Ljung (1987).)

Estimating Parametric Models

Given a description (3.10) and having observed the input-output data u, y, the (prediction) errors $e(t)$ in (3.10) can be computed as:

$$e(t) = H^{-1}(q)[y(t) - G(q)u(t)] \tag{3.38}$$

These errors are, for given data y and u, functions of G and H. These in turn are parametrized by the polynomials in (3.15)-(3.23) or by entries in the state-space matrices defined in (3.29)–(3.32). The most common parametric identification method is to determine estimates of G and H by minimizing

$$V_N(G, H) = \sum_{t=1}^{N} e^2(t) \tag{3.39}$$

that is

$$[\hat{G}_N, \hat{H}_N] = argmin \sum_{t=1}^{N} e^2(t) \tag{3.40}$$

This is called a *prediction error method*. For Gaussian disturbances it coincides with the maximum likelihood method. (See Chapter 7 in Ljung (1987).)

A somewhat different philosophy can be applied to the ARX model (3.15). By forming filtered versions of the input

$$N(q)s(t) = M(q)u(t) \tag{3.41}$$

and by multiplying (3.15) with $s(t-k)$, $k = 1, 2, \ldots, na$ and $u(t-nk+1-k)$, $k = 1, 2, \ldots, nb$ and summing over t, the noise in (3.15) can be correlated out and solved for the dynamics. This gives the *instrumental variable* method, and $s(t)$ are called the instruments. (See Section 7.6 in Ljung (1987).)

Subspace Methods for Estimating State-Space Models

The state-space matrices *A, B, C, D,* and *K* in (3.26) can be estimated directly, without first specifying any particular parametrization by efficient *subspace methods*. The idea behind this can be explained as follows: If the sequence of state vectors *x(t)* were known, together with *y(t)* and *u(t),* eq (3.26b) would be a linear regression, and *C* and *D* could be estimated by the least squares method. Then *e(t)* could be determined, and treated as a known signal in (3.26a), which then would be another linear regression model for *A, B* and *K*. (One could also treat (3.24) as a linear regression for *A, B, C,* and *D* with *y(t)* and *x(t+1)* as simultaneous outputs, and find the joint process and measurement noises as the residuals from this regression. The Kalman gain *K* could then be computed from the Riccati equation.) Thus, once the states are known, the estimation of the state-space matrices is easy.

How to find the states *x(t)*? Appendix 4.A of Ljung(1987) shows that all states in representations like (3.26) can be formed as linear combinations of the k-step ahead predicted outputs (k=1,2,...,n). It is thus a matter of finding these predictors, and then selecting a basis among them. The subspace methods form an efficient and numerically reliable way of determining the predictors by projections directly on the observed data sequences. For more details, see the references under `n4sid` in Chapter 4, "Command Reference" .

4. Nonparametric Model Estimation

This and the following sections give an introduction to the basic functions in the System Identification Toolbox. Not all of the options available when using the functions are described here; see the *Command Reference* chapter and the online Help facility.

Data Representation

The observed output and input signals, $y(t)$ and $u(t)$, are represented in the System Identification Toolbox as *column vectors* `y` and `u`. Row `k` corresponds to sample number k. For multivariable systems, each input (output) component is represented as a column vector, so that `u` becomes an N by nu matrix (N = number of sampled observations, nu = number of input signals). The output-input data is collectively represented as a matrix whose first column(s) is the output and next column(s) is the input:

```
z = [y u]
```

The data can be graphed by the command `idplot`:

```
idplot(z)
```

This gives plots of all combinations of inputs and outputs for visual inspection.

Some of the functions in the System Identification Toolbox apply only to single output data ($ny = 1$) and some only to single-input-single-output data ($ny = 1$, $nu = 1$). Chapter 4, "Command Reference" gives details about this.

Correlation Analysis

The correlation analysis procedure described in “The System Identification Problem” in Chapter 3 is implemented in the function `cra`:

```
ir = cra(z)
```

This function returns the estimated impulse response in the vector `ir`. It also optionally plots the estimated covariance functions for *y* and *u*, so that the success of the whitening filter (3.34) can be checked. From the estimated impulse response it is easy to form the step response:

```
ir = cra(z)
sr = cumsum(ir)
plot(sr)
```

Spectral Analysis

The function `spa` performs spectral analysis according to the procedure in (3.35)–(3.37).

```
[G,PHIV] = spa(z)
```

Here `z` contains the output-input data as above. `G` and `PHIV` are matrices that contain the estimated frequency function G_N and the estimated disturbance spectrum Φ_v in (3.37). They are coded into a special format, the *freqfunc format*, which allows you to plot them using the function `bodeplot` or `ffplot`:

```
[G,PHIV] = spa(z)
bodeplot(G)
bodeplot(PHIV)
```

`bodeplot` gives logarithmic amplitude and frequency scales (in rad/sec) and linear phase scale, while `ffplot` gives linear frequency scales (in Hz). The details of the `freqfunc` format are given in Chapter 4, "Command Reference" and by typing `help freqfunc`. To obtain correct frequency scales you can operate with the command `sett` on `G` or give the sampling interval as an optional argument to `spa`.

By omitting the argument `PHIV`, only the transfer function estimate `G` is computed:

```
G = spa(z)
```

When PHIV is included, the estimated standard deviations of G and PHIV are also returned by spa and included in the freqfunc format.

If z = y is a time series, spa returns an estimate of the spectrum of that signal:

```
PHIY = spa(y)
ffplot(PHIY)
```

In the computations (3.35)-(3.37), spa uses a Hamming window for $W(\tau)$ with a default length M equal to the minimum of 30 and a tenth of the number of data points. This window size M can be changed to an arbitrary number by

```
[G,PHIV] = spa(z,M)
```

The rule is that as M increases, the estimated frequency functions show sharper details, but are also more affected by random disturbances. A typical sequence of commands that test different window sizes is

```
g10 = spa(z,10)
g25 = spa(z,25)
g50 = spa(z,50)
bodeplot([g10 g25 g50])
```

An empirical transfer function estimate is obtained as the ratio of the output and input Fourier transforms with

```
G = etfe(z)
```

This can also be interpreted as the spectral analysis estimate for a window size that is equal to the data length. For a single signal etfe gives the *periodogram* as a spectral estimate. The function also allows some smoothing of the crude estimate; it can be a good alternative for signals and systems with sharp resonances. See Chapter 4, "Command Reference" for more information.

5. Parametric Model Estimation

The System Identification Toolbox contains several functions for parametric model estimation. They all share the same command structure

```
th = function([y u],ths)
```

Input variables y and u are column vectors that contain the output and input data sequences, while the matrix ths specifies the particular structure of the model to be estimated. The resulting estimated model is contained in th. It is coded into the *theta format*. This is the basic format for representing models in the System Identification Toolbox. It collects information about the model structure and the orders, delays, parameters, and estimated covariances of estimated parameters into a matrix. The details of this representation are found in Chapter 4, "Command Reference" or by typing `help theta`. The `theta` format can be translated to other useful model representations. To just display the model information, use the command `present`:

```
th = function(z,ths)
present(th)
```

The following sections assume that you have formed an array z that consists of the output y and the input u:

```
z = [y u]
```

ARX Models

To estimate the parameters a_i and b_i of the ARX model (3.15), use the function `arx`:

```
th = arx(z,[na nb nk])
```

Here na, nb, and nk are the corresponding orders and delays in (3.16) that define the exact model structure. The function `arx` implements the least-squares estimation method, using the MATLAB "/" operator for overdetermined linear equations.

An alternative is to use the Instrumental Variable (IV) method described in connection with (3.41). This is obtained with

```
th = iv4(z,[na nb nk])
```

which gives an automatic (and approximately optimal) choice of the filters N and M in (3.41). (See the procedure (15.21)-(15.26) in Ljung (1987).)

Both `arx` and `iv4` are applicable to arbitrary multivariable systems. If you have ny outputs and nu inputs, the orders are defined accordingly: `na` is an ny by ny matrix whose i -j entry gives the order of the polynomial that relates past values of y_j to the current value of y_i (i.e., past values of y_j up to $y_j(t-na(i,j))$are used when predicting $y_i(t)$). Similarly the i -j entries of the ny by nu matrices `nu` and `nk`, respectively, give the order and delay from input number j when predicting output number i. (See “Defining Model Structures” on page 3-29 and Chapter 4, "Command Reference" for exact details.)

AR Models

For a single signal $y(t)$ the counterpart of the ARX model is the AR model:

$$A(q)y(t) = e(t) \tag{5.1}$$

The `arx` command also covers this special case

```
th = arx(y,na)
```

but for scalar signals more options are offered by the command

```
th = ar(y,na)
```

which has an option that allows you to choose the algorithm from a group of several popular techniques for computing the least-squares AR model. Among these are Burg's method, a geometric lattice method, the Yule-Walker approach, and a modified covariance method. (See Chapter 4, "Command Reference" for details.) The counterpart of the `iv4` command is

```
th = ivar(y,na)
```

which uses an instrumental variable technique to compute the AR part of a time series.

General Polynomial Black-Box Models

Based on the prediction error method (3.40), you can construct models of basically any structure. For the general model (3.23), there is the function

```
th = pem(z,nn)
```

where nn gives all the orders and delays

```
nn = [na nb nc nd nf nk]
```

The pem command covers all cases of black-box linear system models. More efficient routines are available, however, for the common special cases

```
th = armax(z,[na nb nc nk])
th = oe(z,[nb nf nk])
th = bj(z,[nb nc nd nf nk])
```

These handle the model structures (3.17), (3.19) and (3.21), respectively.

All the routines also cover single-output, multi-input systems of the type

$$A(q)y(t) = \frac{B_1(q)}{F_1(q)}u_1(t-nk_1) + \ldots + \frac{B_{nu}(q)}{F_{nu}(q)}u_{nu}(t-nk_{nu}) + \frac{C(q)}{D(q)}e(t) \qquad (5.2)$$

where nb, nf, and nk are row vectors of the same lengths as the number of input channels containing each of the orders and delays

```
nb = [nb1 ...  nbnu];
nf = [nf1 ...  nfnu];
nk = [nk1 ...  nknu];
```

These parameter estimation routines require an iterative search for the minimum of the function (3.39). This search uses a special start-up procedure based on least squares and instrumental variables (the details are given as equation (10.75) in Ljung (1987)). From the initial estimate, a Gauss-Newton minimization procedure is carried out until the norm of the Gauss-Newton direction is less than a certain tolerance. See Ljung (1987), Section 11.2 or Dennis and Schnabel(1983) for details. See also the entry on optional variables associated with the search at the end of this section.

The routines pem, armax, oe, and bj can also be started at any initial value thi specified in the theta format by replacing nn by thi. For example

```
th = pem(z,thi)
```

While the search is typically initialized using the built-in start-up procedure giving just orders and delays (as described above), the ability to force a specific initial condition is useful in several contexts. Some examples are mentioned in “Some Special Topics” on page 3-68.

If a *last* argument `'trace'` is added, as in

```
th = armax(z,nn,'trace')
```

these routines give status reports to the screen during the minimization, containing current and previous estimates, the associated loss functions, and the Gauss-Newton update vector and its norm. The estimates are then represented by a column vector, listing the parameters in alphabetical order.

The number then flickering in the upper left corner shows how many times the suggested update direction has been bisected during the search. It is a healthy sign if this number is small.

These routines return the estimated covariance matrix of the estimated parameter vector as part of `th`. This reflects the reliability of the estimates. The covariance matrix estimate is computed under the assumption that it is possible to obtain a "true" description in the given structure.

State-Space Models

Black-box, discrete-time, parametrizations. Suppose first that there is no particular knowledge about the internal structure of the discrete-time state-space model (3.16). Any linear model of a given order is sought. Then there are essentially two different ways to estimate state-space models, like (3.26). The simplest one is to use `n4sid`:

```
th = n4sid(z,order);
```

which creates a model in state-space form (coded as usual in the theta format) in a realization that is automatically chosen to be reasonably well conditioned. The basic idea behind this subspace algorithm was described in "The System Identification Problem" on page 3-8.

The other alternative is to use a prediction error method, minimizing the criterion (5.3) below. Initializing the search at the `n4sid` model is then a good idea. This is obtained by

```
thi = canstart(z,order,nu)
th = pem(z,thi)
```

Arbitrarily parameterized models in discrete and continuous time. For state-space models of given structure, most of the effort involved relates to defining and manipulating the structure. This is discussed in "Defining Model Structures" on page 3-29. Once the structure is defined in the theta format as `ths`, you can estimate its parameters with

```
th = pem(z,ths)
```

When the systems are multi-output, the following criterion is used for the minimization:

$$min \quad det \sum_{t=1}^{N} e(t)e^{T}(t) \tag{5.3}$$

which is the maximum likelihood criterion for Gaussian noise with unknown covariance matrix.

The numerical minimization of the prediction error criterion (3.39) or (5.3) can be a difficult problem for general model parametrizations. The criterion, as a function of the free parameters, can define a complicated surface with many local minima, narrow valleys and so on. This may require substantial interaction from the user, in providing reasonable initial parameter values, and also by freezing certain parameter values while allowing others to be free. Note that `pem` easily allows the freezing of any parameters to their nominal values. The functions `fixpar` and `unfixpar` are also useful. A procedure that is often used for state-space models is to allow the noise parameter in the K matrix free, only when a reasonable model of the dynamic part has been obtained. See also "Some Special Topics" on page 3-68.

Optional Variables

The functions `pem`, `armax`, `oe`, and `bj` can accept three additional input options that affect the minimization, as well as one extra output argument. For example,

```
[th, iter_info] = bj(z,nn,maxiter,tol,lim)
```

The format is the same for the other functions. If these variables are not given, default values are assigned to them.

`maxiter`: This variable determines the maximum number of iterations performed in the search for the minimum. The default value is 10. Setting the value of `maxiter` to zero results in the return of the estimate from the start-up procedure.

`tol`: The iterations are continued until the norm of the Gauss-Newton update vector is less than `tol`. The default value is 0.01. The iterations also terminate when maxiter is reached or when no decrease in the criterion can be found along the search direction after it has been bisected 10 times. (After the 10 first bisections, the search direction is changed to the gradient one, and 10 new bisections are tried.)

The parameters associated with the noise dynamics, (*C* and *D*), are often more difficult to estimate than the dynamics parameters, *A*, *B*, and *F*. Major contributions to the norm of the update direction then come from the entries corresponding to the noise parameters. The Gauss-Newton vector can be monitored to see if it is reasonable to increase `tol` on this ground.

`lim`: The quadratic criterion (3.39) gives a considerable relative weight to large prediction errors. You may want to limit their influence, using *robust estimation techniques*. In the estimation functions `ar`, `bj`, `max`, `oe`, and `pem`, a prediction error that is larger than `lim*` (estimated standard deviation of *e*) carries a linear, rather than a quadratic, weight.

The default value of `lim` is `1.6`. `lim=0` disables the robust estimation and applies a purely quadratic criterion. The standard deviation is estimated robustly as the median of the absolute deviations from the median, divided by 0.7. (See eq (15.9)-(15.10) in Ljung (1987).)

Note that the loss function displayed by `present(th)` is quadratic and that the data-dependent delimiter is computed only once, before the minimization starts.

`iter_info`: This a a row vector with three entries that give information about the iteration process. The first entry is the number of used iterations. The second one is the improvement of the fit over the last iteration, and the third variable is the norm of the Gauss-Newton search vector. If the number of used iterations is less than `maxiter`, at the same time as this norm is larger then `tol`, it means that the iterations have been aborted because no better fit could be obtained along the indicated search direction.

For the spectral analysis estimate, you can compute the frequency functions at arbitrary frequencies. If the frequencies are specified in a row vector `w`, then

```
[G,PHIV] = spa(z,M,w)
```

results in `G` and `PHIV` being computed at these frequencies. This computation is somewhat slower, however, than using the default frequencies.

You can generate logarithmically spaced frequencies using the MATLAB `logspace` function. For example

```
w = logspace(-3,pi,128)
```

The optional trailing arguments can be omitted, in which case default values are used. Entering them as empty matrices, [ ], also causes functions to rely upon defaults.

6. Defining Model Structures

Since the System Identification Toolbox handles a wide variety of different model structures, it is important that these can be defined in a flexible way. In the previous section you saw how model structures that are special cases of the general model (3.23) can be defined by specifying the orders and delays in the various estimation routines `arx`, `iv4`, `oe`, `bj`, `armax`, and `pem`. This section describes how model structures and models can be directly defined. For black-box models (3.23) this may be required, for example, when creating a model for simulation. Commands for creating state-space model structures are discussed in this section.

The general way of representing models and model structures in the System Identification Toolbox is the *theta format*. This section introduces the commands (apart from the parametric estimation functions themselves) that create models in the theta format.

Polynomial Black-Box Models

The general input-output form (3.23):

$$A(q)y(t) = \frac{B(q)}{F(q)}u(t-nk) + \frac{C(q)}{D(q)}e(t) \tag{6.1}$$

is defined by the five polynomials $A(q)$, $B(q)$, $C(q)$, $D(q)$, and $F(q)$. These are represented in the standard MATLAB format for polynomials. Polynomial coefficients are stored as row vectors ordered by descending powers. For example, the polynomial

$$A(q) = 1 + a1q^{-1} + a2q^{-2} + \ldots + anq^{-n}$$

is represented as

```
A = [1 a1 a2 ... an]
```

Delays in the system are indicated by leading zeros in the $B(q)$ polynomial. For example, the ARX model

$$y(t) - 1.5y(t-1) + 0.7y(t-2) = 2.5u(t-2) + 0.9u(t-3) \tag{6.2}$$

is represented by the polynomials

```
A = [1 –1.5 0.7]
B = [0 0 2.5 0.9]
```

The theta format representation of (6.1) is now created by the command

```
th = poly2th(A,B,C,D,F,lam,T)
```

`lam` is here the variance of the white noise source $e(t)$ and `T` is the sampling interval. Trailing arguments can be omitted for default values. The system (6.2) can for example be represented by

```
th = poly2th([1 –1.5 0.7],[0 0 0.5 0.7])
```

In the multi-input case (5.2) `B` and `F` are matrices, whose row number `k` corresponds to the `k`-th input. The command `poly2th` can also be used to define time-continuous systems. See the *Command Reference* chapter for details.

Multivariable ARX Models

A multivariable ARX model is given by

$$A(q)y(t) = B(q)u(t) + e(t) \tag{6.3}$$

Here $A(q)$ is an ny by ny matrix whose entries are polynomials in the delay operator q^{-1}. You can represent it as

$$A(q) = I_{ny} + A_1 q^{-1} + \ldots + A_{na} q^{-na} \tag{6.4}$$

as well as the matrix

$$A(q) = \begin{bmatrix} a_{11}(q) & a_{12}(q) & \cdots & a_{1ny}(q) \\ a_{21}(q) & a_{22}(q) & \cdots & a_{2ny}(q) \\ \vdots & \vdots & \ddots & \vdots \\ a_{ny1}(q) & a_{ny2}(q) & \vdots & a_{nyny}(q) \end{bmatrix}$$

where the entries a_{kj} are polynomials in the delay operator q^{-1}:

$$a_{kj}(q) = \delta_{kj} + a_{kj}^{1}q^{-1} + \ldots + a_{kj}^{na_{kj}}q^{-na_{kj}} \tag{6.6}$$

This polynomial describes how old values of output number j affect output number k. Here δ_{kj} is the Kronecker-delta; it equals 1 when $k = j$, otherwise, it is 0. Similarly $B(q)$ is an ny by nu matrix

$$B(q) = B_0 + B_1q^{-1} + \ldots B_{nb}q^{-nb} \tag{6.7}$$

or

$$B(q) = \begin{bmatrix} b_{11}(q) & b_{12}(q) & \cdots & b_{1nu}(q) \\ b_{21}(q) & b_{22}(q) & \cdots & b_{2nu}(q) \\ \vdots & \vdots & \ddots & \vdots \\ b_{ny1}(q) & b_{ny2}(q) & \vdots & b_{nynu}(q) \end{bmatrix}$$

with

$$b_{kj}q = b_{kj}^{1}q^{-nk_{kj}} + b_{kj}^{-nb}q^{-nk_{ij}-nb_{ij}+1}$$

The delay from input number j to output number k is nk_{kj}. To link with the structure definition in terms of `[na nb nk]` in the `arx` and `iv4` commands, note that `na` is a matrix whose kj-element is na_{kj}, while the kj-elements of `nb` and `nk` are nb_{kj} and nk_{kj} respectively.

The theta format representation of the model (6.3) can be created by

```
th = arx2th(A,B,ny,nu)
```

where `ny` and `nu` are the number of outputs and inputs, respectively, and `A` and `B` are matrices that define the matrix polynomials (6.4) and (6.7):

```
A = [eye(ny) A1 A2 .. Ana]
B = [B0 B1 .. Bnb]
```

Consider the following system with two outputs and three inputs:

$$y_1(t) - 1.5y_1(t-1) + 0.4y_2(t-1) + 0.7y_1(t-2) = 0.2u_1(t-4) + 0.3u_1(t-5) + 0.4u_2(t) - 0.1u_2(t-1) + 0.15u_2(t-2) + e_1(t)$$

$$y_2(t) - (0.2)y_1(t-1) - 0.7_{y2}(t-2) + 0.01_{y1}(t-2) = u_1(t) + 2u_2(t-4) + 3u_3(t-1) + 4u_3(t-2) + e_2(t)$$

This system is defined and simulated for a certain input u, and then estimated in the correct ARX structure by the following string of commands:

```
A0 = eye(2);
A1 = [-1.5 0.4; -0.2 0];
A2 = [0.7 0 ; 0.01 -0.7];
B0 = [0 0.4 0;1 0 0];
B1 = [0 -0.1 0;0 0 3];
B2 = [0 0.15 0;0 0 4];
B3 = [0 0 0;0 0 0];
B4 = [0.2 0 0;0 2 0];
B5 = [0.3 0 0;0 0 0];
A = [A0 A1 A2];
B = [B0 B1 B2 B3 B4 B5];
th0 = arx2th(A,B,2,3);
e = randn(200,2);
u = [idinput(200),idinput(200),idinput(200)];
y = idsim([u e],th0);
na = [2 1;2 1];
nb = [2 3 0;1 1 2];
nk = [4 0 0;0 4 1];
the = arx([y u],[na nb nk]);
[Ae, Be] = th2arx(the);
```

You can use the two commands `fixpar` and `unfixpar` to manipulate ARX structures, so that certain parameters in the structure are fixed to certain known values and not estimated. See Chapter 4, "Command Reference"for more information.

State-Space Models with Free Parameters

The basic state-space models are the following ones: (See also "The System Identification Problem" on page 3-8.)

Discrete-Time Innovations Form

$$x(kT + T) = Ax(kT) + Bu(kT) + Ke(kT) \quad (6.8a)$$

$$y(kT) = Cx(kT) + Du(kT) + e(kT) \quad (6.8b)$$

$$x(0) = x0 \quad (6.8c)$$

Here T is the sampling interval, $u(kT)$ is the input at time instant kT and $y(kT)$ is the output at time kT. (See Ljung (1987) page 87.)

System Dynamics Expressed in Continuous Time

$$\dot{x}(t) = Fx(t) + Gu(t) + \tilde{K}w(t) \quad (6.9a)$$

$$y(t) = Hx(t) + Du(t) + w(t) \quad (6.9b)$$

$$x(0) = x0 \quad (6.9c)$$

(See Ljung (1987) page 82.) It is often easier to define a parametrized state-space model in continuous time because physical laws are most often described in terms of differential equations. The matrices F, G, H, and D contain elements with physical significance (for example, material constants). The numerical values of these may or may not be known. To estimate unknown parameters based on sampled data (assuming T is the sampling interval) first transform (6.9) to (6.8) using the formulas (3.30). The value of the Kalman-gain matrix K in (6.8) or $\tilde{K}$ in (6.9) depends on the assumed character of the additive noises $w(t)$ and $e(t)$ in (3.28), and its continuous-time counterpart. Disregard that link and view K in (6.8) (or $\tilde{K}$ in (6.9)) as the basic tool to model the noise properties. This gives the *directly parametrized innovations form*. (See Ljung (1987) page 88.) If the internal noise structure is important, you could use user-defined structures as in Example 6.4.

The System Identification Toolbox allows you to define arbitrary parameterizations of the matrices in (6.8) or (6.9). When defining the structure, the known elements in the matrices are entered with their numerical values,

while the unknown elements are entered as `NaN`. Use the commands `modstruc` and `ms2th`.

The Black-Box, Discrete-Time Case

For a discrete-time model, like (6.8) without any particular internal structure, it is easier and more natural to use `canstart` or `n4sid` to estimate the model, without any prior specification of the parametrization.

Example 6.1: A Discrete-Time Structure. Consider the discrete-time model

$$x(t+1) = \begin{bmatrix} 1 & \theta_1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} \theta_2 \\ \theta_3 \end{bmatrix} u(t) + \begin{bmatrix} \theta_4 \\ \theta_5 \end{bmatrix} e(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) + e(t)$$

$$x(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

with five unknown parameters θ_i, i=1, ...,3. This structure is defined in the theta format by

```
A = [1 NaN; 0 1];
B = [NaN;NaN];
C = [1 0];
D = 0;
K = [NaN;NaN];
x0 = [0;0];
ms1 = modstruc(A,B,C,D,K,x0);
th1 = ms2th(ms1,'d')
```

The definition thus follows in two steps. First the actual structure (known and unknown parameter values) is coded in the matrix `ms1` and then a theta format matrix is created by `ms2th`. The argument `'d'` denotes that the structure is a discrete-time structure. Other optional arguments allow definition of guessed values of the unknown parameters, as well as specification of the covariance matrix of $e(t)$ and of the sampling interval T. You can also specify unknown elements in the initial value vector $x(0)$.

For models that are parametrized as canonical forms, use `canform` instead of `modstruc` to define the model structure.

Example 6.2: A Continuous-Time Model Structure. Consider the following model structure

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & \theta_1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \theta_2 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(t) + e(t)$$

$$x(0) = \begin{bmatrix} \theta_3 \\ 0 \end{bmatrix}$$

This corresponds to an electrical motor, where $y_1(t) = x_1(t)$ is the angular position of the motor shaft and $y_2(t) = x_2(t)$ is the angular velocity. The parameter $-\theta_1$ is the inverse time constant of the motor and $-\theta_2/\theta_1$ is the static gain from the input to the angular velocity. (See page 84 in Ljung (1987).) The motor is at rest at time 0 but at an unknown angular position. Suppose that θ_1 is around -1 and θ_2 is around 0.25. If you also know that the variance of the errors in the position measurement is 0.01 and in the angular velocity measurements is 0.1, you can then define a theta structure using

```
F = [0 1; 0 NaN]
G = [0 ;NaN]
H = eye(2)
D = zeros(2,1)
K = zeros(2,2)
x0 = [NaN;0]
ms2 = modstruc(F,G,H,D,K,x0);
thguess=[-1, 0.25, 0]
noisevar = [0.01 0; 0 0.1]
th2 = ms2th(ms2,'c',thguess,noisevar)
```

The structure `th2` can now be used to estimate the unknown parameters θ_i from observed data `z = [y1 y2 u]` by

```
model = pem(z,th2)
```

The iterative search for minimum is then initialized at thguess. The structure can also be used to simulate the system above with sampling interval T=0.1 for input u and noise realization e:

```
e=randn(300,2)
u=idinput(300);
th2 = sett(th2,0.1)  % setting the sampling interval
y = idsim([u e],th2)
```

The nominal parameter values thguess are then used, and the noise sequence is scaled according to the matrix noisevar.

When estimating models, you can try a number of "neighboring" structures, like "what happens if I fix this parameter to a certain value" or "what happens if I let loose these parameters." Rather than going through the whole procedure in redefining the structure, you can use the commands fixpar and unfixpar. For example, to free the parameter $x_2(0)$ (perhaps the motor wasn't at rest after all) you can use

```
th = unfixpar(th2,'x0',[2,1])
```

To manipulate initial conditions, the function thinit is also useful.

State-Space Models with Coupled Parameters

In some situations you may want the unknown parameters in the matrices in (6.8) or (6.9) to be linked to each other. Then the NaN feature is not sufficient to describe these links. Instead you need to write an M-file that describes the structure. The format is

```
[A,B,C,D,K,x0] = mymfile(par,T,aux);
```

where mymfile is the user-defined name for the M-file, par contains the parameters, T is the sampling interval, and aux contains auxiliary variables. The latter variables are used to house options, so that some different cases can be tried out without having to edit the M-file. The matrices A, B, C, D, K, and x0 refer to the discrete-time description (6.8). If the underlying parameterization is in continuous time, as in (6.9), it is desirable to write the M-file so that a negative value of T inhibits the sampling. This means that A, B, C and D should be returned as the matrices in (6.9) when mymfile is called with a negative T. To obtain the same structure as in the Example 6.2 , you can do the following:

```
function [A,B,C,D,K,x0] = mymfile(par,T,aux)
F = [0 1; 0 par(1)];
G = [0;par(2)];
C = eye(2);
D = zeros(2,2);
K = zeros(2,1);
x0 =[par(3);0];
if T>0, [A,B] = c2d(F,G,T); else A=F; B=G; end
```

Here `c2d` is the sampling function from the Control System Toolbox, which consequently is required to run this example.

Once the M-file has been written, the corresponding theta structure is defined by the command `mf2th` (M-file to theta):

```
th = mf2th('mymfile','c',par,aux);
```

where `par` contains the nominal (initial) values of the corresponding entries in the structure. `'c'` signals that the underlying parametrization is continuous time, and that the M-file is equipped with sampling inhibition (for $T < 0$). `aux` contains the values of the auxiliary parameters.

From here on, estimate models and evaluate results as for any other model structure. Note, though that the features `fixpar` and `unfixpar` are not available for these general structures. Some further examples of user-defined model structures are given below.

State-Space Structures: Initial Values and Numerical Derivatives

It is sometimes difficult to find good initial parameter values at which to start the numerical search for a minimum of (3.38). It is always best to use physical insight, whenever possible, to suggest such values with `thguess`. For random initialization, the command `thinit` is a useful alternative and complement to `thguess`. Since there is always a risk that the numerical minimization may get stuck in a local minimum, it is advisable to try several different initialization values for θ.

In the search for the minimum, the gradient of the prediction errors $e(t)$ is computed by numerical differentiation. The step-size is determined by the M-file `nuderst`. In its default version the step-size is simply 10^{-4} times the absolute value of the parameter in question (or the number 10^{-7} if this is

larger). When the model structure contains parameters with different orders of magnitude, try to scale the variables so that the parameters are all roughly the same magnitude. In any case, you may need to edit the M-file nuderst to address the problem of suitable step sizes for numerical differentiation.

Some Examples of User-Defined Model Structures

With user-defined structures, you have complete freedom in the choice of models of linear systems. This section gives two examples of such structures.

Example 6.3: Heat Diffusion. Consider a system driven by the heat-diffusion equation (see also Example 4.3 in Ljung (1987)).

This is a metal rod with heat-diffusion coefficient κ which is insulated at the near end and heated by the power u (W) at the far end. The output of the system is the temperature at the near end. This system is described by a partial differential equation in time and space. Replacing the space-second derivative by a corresponding difference approximation gives a time-continuous state-space model (6.9), where the dimension of x depends on the step-size in space used in the approximation. This is described by the following M-file:

```
function [A,B,C,D,K,x0] = heatd(pars,T,aux)
N = aux(1); % Number of points in the
            % space–discretization
L = aux(2); % Length of the rod
temp = aux(3); % Assuming uniform initial
               % temperature of the rod
deltaL = L/N; % Space interval
kappa = pars(1); % The heat–diffusion coefficient
K = pars(2); % Heat transfer coefficient at
            % far end of rod
F=zeros(N,N);
for kk=2:N–1
F(kk,kk–1)=1;
F(kk,kk)=–2;
F(kk,kk+1)=1;
end
F(1,1)=–1; F(1,2)=1; % Near end of rod
                     % insulated
F(N,N–1)=1; F(N,N)=–1;
F=F*kappa/deltaL/deltaL;
```

```
G=zeros(N,1);
G(N,1)=K/deltaL;
C=zeros(1,N);
C(1,1)=1;
D=0;
K=zeros(N,1);
x0=temp*ones(N,1);
if T>0 [A,B]=c2d(F,G,T);else A=F;B=G;end
```

You can then get started with the theta format with

```
th = mf2th('heatd','c',[0.27 1],[10,1,22],[],10)
```

for a 10th order approximation of a heat rod one meter in length, with an initial temperature of 22 degrees and a sampling interval of 10 seconds. The initial estimate of the heat conductivity is 0.27, and of the heat transfer coefficient is 1. The covariance matrix of the prediction errors is given a default value.

Example 6.4: Parametrized Noise Models. Consider a discrete-time model

$$x(t+1) = Ax(t) + Bu(t) + w(t)$$
$$y(t) = Cx(t) + e(t)$$

where w and e are independent white noises with covariance matrices $R1$ and $R2$, respectively. Suppose that you know the variance of the measurement noise $R2$, and that only the first component of $w(t)$ is nonzero. This can be handled by the following M-file:

```
function [A,B,C,D,K,x0] = mynoise(par,T,aux)
R2 = aux(1); % The assumed known measurement
             % noise variance
A = [par(1) par(2);1 0];
B = [1;0];
C=[par(3) par(4)];
D=0;
R1= [par(5) 0;0 0];
K = A*dlqe(A,eye(2),C,R1,R2); % from the
              % Control System Toolbox
x0=[0;0];
```

7. Examining Models

Once you have estimated a model, you need to investigate its properties. You have to simulate it, test its predictions, and compute its poles and zeros and so on. You thus have to transform the model from the theta format to other ways of representing and presenting it.

Theta Format: th

The basic format for representing models in the System Identification Toolbox is called the *theta format*. It stores all relevant information about the model structure used, including the values of the estimated parameters, the estimated covariances of the parameters, and the estimated innovations variance and so on. It also contains some information about how and when the model was created. The details of the representation can be found in the Chapter 4, "Command Reference" or by typing `help theta` (for state-space based structures type `help thss`). The representation should, however, be thought of as an internal one; unless you are an advanced user, you do not need to worry about the details. “Defining Model Structures” on page 3-29 describes how to create models in the theta format using several different commands.

The information in the theta format is displayed by the `present` command:

```
present(th)
```

This spells out the information in `th`. Depending on the character of the underlying model structure, the information is given in terms of the polynomials in (6.1), in terms of the matrices in (6.8) or (6.9), or in terms of the ARX polynomials in (6.3). Estimated standard deviations for the parameters are always supplied. For multivariable ARX models and for state-space matrices, the standard deviations are given as imaginary numbers added to the parameters. For the polynomials they are given as a second row. For example, the following printout from `present`

```
The polynomial coefficients and their standard deviations are
B =
0.0000 0.0000 1.7345
0.0000 0.0000 0.0563

A =
1.0000 –1.5312 0.6983
0.0000 0.0214 0.0022
```

is interpreted as $A(q) = 1 - 1.5312q^{-1} + 0.6983q^{-2}$ and the standard deviation of "1" is zero (naturally enough , since it is not estimated). The standard deviation of a_1 is 0.0214. Note that leading zeros in the *B* polynomial indicate the delay, as defined in "Defining Model Structures" on page 3-29.

The actual parameters can be extracted from the theta format by

```
[par,P] = th2par(th)
```

Here `par` is a row vector containing the parameters, and `P` is the estimated covariance matrix of the parameters. The parameters are listed in "alphabetical order." See "Some Special Topics" on page 3-68. For the polynomial model (6.1) the coefficients of the *A*, *B*, *C*, *D*, and *F* polynomials are given in the order of increasing powers of q^{-1}. (See for example (6.2).) Note that leading 0's of *B* and the leading 1 in the other polynomials are omitted since they do not correspond to estimated parameters. For the state-space model (6.8) the parameters are given in the order obtained as the matrix *A* is first searched for estimated parameters row by row, then *B* and then *C*, etc. The same is true for the continuous model (6.9). For user-defined structures, the ordering in `par` is the same as defined in the corresponding M-file. For multivariable ARX models, the ordering of parameters is somewhat tricky and it is safer use `th2arx`. (See "The ARX Format: arx" on page 3-45.)

The sampling interval can be extracted from the theta format by

```
T = gett(th)
```

A negative `T` means that `th` represents a time-continuous system. Then system `abs(T)` indicates the sampling interval for the data for which the model was estimated. It is also the sampling interval that applies when `th` is used for simulation or prediction.

Frequency Function Format: ff

The frequency function and the disturbance spectrum corresponding to a model `th` in the theta format is computed by

```
[G,PHIV] = th2ff(th)
```

This gives *G* and Φ_v in (3.11) along with their estimated standard deviations, which are translated from the covariance matrix of the estimated parameters. If `th` corresponds to a time-continuous model, the frequency functions are computed accordingly.

The resulting matrices G and PHIV are given in the *freqfunc format*. This format stores the frequency functions and spectra (3.11) and their standard deviations as column vectors. The first row is reserved for administrative information, giving details about input and output numbers. The frequency values can be chosen arbitrarily, but default to 128 values equally spaced between 0 (excluded) and the Nyquist frequency. For time-continuous models, the default frequency range extends to a decade over the Nyquist frequency for the sampled data upon which the model is based. Typing `help freqfunc` describes the details of the format, which, however, is not important for basic use of the toolbox.

The function th2ff has options that allow the selection of specific input-output pairs, and the computation of frequency functions and spectra at arbitrary frequencies.

If th corresponds to a model of a time series, its spectrum is obtained as

```
PHIY = th2ff(th)
```

Models in the freqfunc format are also obtained by the nonparametric estimation routines spa and etfe, as described in “Nonparametric Model Estimation” on page 3-19.

Three functions offer graphic display of the frequency functions and spectra: bodeplot, ffplot, and nyqplot.

```
bodeplot(G)
```

plots the Bode diagram of G (logarithmic scales and frequencies in rad/sec). If G is a spectrum, only an amplitude plot is given. Several curves in the same diagram are obtained with

```
bodeplot([G1 G2 ... Gn])
```

If information about standard deviations is included in G, confidence intervals corresponding to sd standard deviations are graphed with

```
bodeplot(G,sd)
```

The command ffplot has the same syntax as bodeplot but works with linear frequency scales and Hertz as the unit. The command nyqplot also has the same syntax, but produces Nyquist plots; i.e., graphs of the frequency function in the complex plane.

For the creation of own frequency plots, the command getff extracts the relevant portions of the freqfunc format. (See Chapter 4, "Command Reference"for more information.)

Zero-Pole Format: zp

The command th2zp computes the zeros, poles, and static gains of the system associated with th:

```
[zepo,K] = th2zp(th)
```

The matrix zepo now contains information about the zeros and the poles of the model, along with their estimated standard deviations. It also contains (in the first row) administrative information about input and output numbers, which is used for plotting. Similarly, K contains the static gains and their standard deviations. The best way of using the information in zepo is to graph the zeros and poles by

```
zpplot(zepo)
```

Confidence regions (lines and ellipsoids) corresponding to sd standard deviations are illustrated by

```
zpplot(zepo,sd)
```

Comparisons between several models are obtained by

```
zp1 = th2zp(th1)
zp2 = th2zp(th2)
zpplot(zpform(zp1,zp2))
```

zpplot has several options that determine how information about different models and different input-output pairs is depicted. It also keeps track of whether the underlying model is in discrete or continuous time and draws the complex plane accordingly. The command th2zp also allows the selection of specific input-output subsystems in the case of multivariable models.

The function getzp allows you to extract the zeros and the poles explicitly from the zero-pole format.

State-Space Format: ss

The command

```
[A,B,C,D,K,X0] =  th2ss(th)
```

computes the matrices of a state-space representation of th. The representation is in discrete time as (6.8) or in continuous time as (6.9) depending on the status of th. If the underlying model structure is a

state-space one, the matrices A, B, etc. are given in the same basis as originally defined. If the underlying model is defined in terms of input-output polynomials, the state-space representation is in observer canonical form. The state-space matrices can be used for further analysis and design in the Control System Toolbox and the Signal Processing Toolbox.

The command `th2ss` can also compute the standard deviations of the elements in state-space matrices.

Transfer Function Format: tf

The transfer functions associated with the model `th` are computed by

```
[NUM,DEN] = th2tf(th)
```

`NUM` and `DEN` are given in the same format that is used in the Control System Toolbox and the System Processing Toolbox. The common denominator `DEN` as a row matrix and the `k`-th row of `NUM` gives the numerator polynomial associated with output number k. If the model `th` has several inputs, specify the input number as an argument. The polynomials are represented as row vectors in descending powers of `s` (in continuous time) or `z` or `q` (in discrete time). It thus differs from the polynomial format. (See below.) For example, the continuous-time system

$$\frac{1}{s(s+2)}$$

is represented as

```
NUM = [0 0 1] (or just 1)
DEN = [1 2 0]
```

while the discrete-time system

$$\frac{q^{-3}}{1-1.5q^{-1}+0.7q^{-2}} = \frac{1}{q^3-1.5q^2+0.7q} \qquad (7.1)$$

is represented as

```
NUM = 1
DEN = [1 -1.5 0.7 0]
```

Polynomial Format: poly

In the polynomial format, the polynomials A, B, C, D, and F in (3.23) are given as follows: Polynomial coefficients are stored in row vectors, ordered by ascending powers in the delay operator q^{-k}, $k = 0, 1, \ldots$, always starting with $k = 0$. This means that (7.1) is represented by

```
B = [0 0 0 1]
A = [1 -1.5 0.7]
```

Delays are thus denoted by leading zeros in `B`. Note the difference with the transfer function representation, in that leading zeros must be explicitly given. Only when the lengths of `A` and `B` are the same do the formats coincide. See also Section 6. The polynomials corresponding to `th` are extracted by

```
[A,B,C,D,F] = th2poly(th)
```

This is consequently the "inverse" of `poly2th` described in "Defining Model Structures" on page 3-29.

Continuous-time systems can also be represented in the polynomial format. Then the polynomials are given as row vectors in descending powers of the Laplace variable s just as in the transfer function case.

The ARX Format: arx

To obtain an ARX description from a model `th` that corresponds to such a description (i.e., it has originally been generated by `arx`, `iv4`, or `arx2th`), use

```
[A,B] = th2arx(th)
```

The matrices `A` and `B` then contain the ARX polynomials as described in Section 6. `th2arx` is thus an inverse of `arx2th`.

Transformations Between Discrete and Continuous Models

Continuous-Time Models

A model in the theta format can be internally represented as a *continuous-time model*. This occurs if the structure was originally defined as a state-space structure with `ms2th` or `mf2th` using the argument `'c'`. (See “Defining Model Structures” on page 3-29 and the Chapter 4, "Command Reference".) The model then remains a continuous-time one during the estimation phases with `pem` even though the estimation is performed using sampled data. A continuous-time model of the type (6.1) in polynomial form also results when `poly2th` is used with a negative value of the sampling interval. This structure cannot, however, be used for further estimation. Finally, continuous-time models are obtained by the command `thd2thc`. (See below.)

Continuous-time models are indicated by a negative sampling interval. A quick check can thus be given by

```
gett(th)<0
```

All the model transformations `th2ff`, `th2zp`, `th2ss`, `th2tf`, and `th2poly` result in the corresponding characteristics in continuous time for models `th` that are indicated as continuous. For simulation and prediction, however, the model is first transformed to discrete time.

The absolute value of the sampling interval of a continuous-time model is the sampling interval of the data for which the model was estimated. It is also used to select suitable default frequency intervals in `th2ff` (the default being from zero up to $10 * \pi / |T|$ rad/sec). The absolute value of T also applies when the model is used for simulation and prediction.

Discrete-Time Models

These models are obtained from the estimation functions `ar`, `armax`, `arx`, `bj`, `ivar`, `iv4`, `n4sid`, and `oe`. They are also obtained from `pem` when applied to (6.1) and to state-space structures generated by `ms2th` and `mf2th` with the argument `'d'`. Discrete-time models also result from `poly2th` with positive or default values of the sampling interval. Finally discrete-time models are obtained with the command `thc2thd`. (See below.)

The sampling interval T is obtained by `T = gett(th)`. It indicates the sampling interval of the model as well as of the data to which the model was adjusted. It is also used for appropriate frequency scales in the freqfunc format.

Transformations

Transformations between continuous-time and discrete-time model representations are achieved by `thc2thd` and `thd2thc`. The corresponding uncertainty measure (the estimated covariance matrix of the internal parameters) is also transformed in most cases. The syntax is

```
thc = thd2thc(thd)
thd = thc2thd(thc,T)
```

If the discrete-time model has some pure time delays ($nk > 1$) the default command removes them before forming the continuous-time model. They should then be appended as an extra dead time. This is done automatically by `th2ff`. `thd2thc` offers as an option to approximate the dead time by a finite dimensional system. Note that the noise properties are translated by the somewhat questionable formula (3.32). The covariance matrix is translated by Gauss' approximation formula using numerical derivatives. The M-file `nuderst` is then invoked. You may want to edit it for applications where the parameters have very different orders of magnitude. See the comments in "Defining Model Structures" on page 3-29.

Here is an example which compares the Bode plots of an estimated model and its continuous-time counterpart:

```
th = armax(z,[2 3 1 2]);
gd = th2ff(th);
thc = thd2thc(th);
gc = th2ff(thc);
bodeplot([gd gc])
```

Simulation and Prediction

Any model, when given in the theta format, can be simulated with

```
y = idsim([u e],th)
```

where `u` and `e` are column vectors (or matrices) containing the input and noise sequences. The output is returned in a column vector `y` of the same length. Notice that `th` contains information about the noise variance so `e` should be a

zero-mean, unit-variance sequence. If `e` is omitted, a noise-free simulation is obtained. Here is a typical string of commands:

```
A = [1 –1.5 0.7];
B = [0 1 0.5];
th0 = poly2th(A,B,[1 –1 0.2]);
u = idinput(400,'rbs',[0 0.3]);
e = randn(400,1);
y = idsim([u e],th0);
plot(y)
```

The "inverse model" (3.38), which computes the prediction errors from given input-output data, is simulated with

```
e = pe([y u],th)
```

To compute the k-step ahead prediction of the output signal based on a model `th`, the procedure is as follows:

```
yhat = predict([y u],th,k)
```

The predicted value $\hat{y}(t|t-k)$ is computed using the information in $u(s)$ up to time $s = t$ and information in $y(s)$ up to time $s = t-k$. The actual way that the information in past outputs is used depends on the noise model in `th`. For example, an output error model $A = C = D = 1$ maintains that there is no information in past outputs, therefore, predictions and simulations coincide.

`predict` can evaluate how well a time series model is capable of predicting future values of the data. Here is an example, where `y` is the original series of, say, monthly sales figures. A model is estimated based on the first half, and then its ability to predict half a year ahead is checked out on the second half of the observations:

```
idplot(y)
y1 = y(1:48), y2 = y(49:96)
th = ar(y1,4)
yhat = predict(y2,th4,6)
plot([y2 yhat])
```

The command `compare` is useful for any comparisons involving `idsim` and `predict`.

8. Model Structure Selection and Validation

After you have been analyzing data for some time, you typically end up with a large collection of models with different orders and structures. You need to decide which one is best, and if the best description is an adequate model for your purposes. These are the problems of *model validation*.

Model validation is the heart of the identification problem, but there is no absolute procedure for approaching it. It is wise to be equipped with a variety of different tools with which to evaluate model qualities. This section describes the techniques you can use to evaluate model qualities using the System Identification Toolbox.

Comparing Different Structures

It is natural to compare the results obtained from model structures with different orders. For state-space models this is easily obtained by using a vector argument for the order in `n4sid`:

```
th = n4sid(z,1:10)
```

This invokes a plot from which a "best" order is chosen.

For models of ARX type, various orders and delay can be efficiently studied with the command `arxstruc`. Collect in a matrix `NN` all of the ARX structures you want to investigate, so that each row of `NN` is of the type

```
[na nb nk]
```

With

```
V = arxstruc(ze,zv,NN)
```

an ARX model is fitted to the data set `ze` for each of the structures in `NN`. Next, for each of these models, the sum of squared prediction errors is computed, as they are applied to the data set `zv`. The resulting loss functions are stored in `V` together with the corresponding structures.

To select the structure that has the smallest loss function for the validation set `zv`, use

```
nn = selstruc(V,0)
```

Such a procedure is known as *cross validation* and is a good way to approach the model selection problem.

It is usually a good idea to visually inspect how the fit changes with the number of estimated parameters. A graph of the fit versus the number of parameters is obtained with

```
nn = selstruc(V)
```

This routine prompts you to choose the number of parameters to estimate, based upon visual inspection of the graph, and then it selects the structure with the best fit for that number of parameters.

The command `struc` helps generate typical structure matrices `NN` for single-input systems. A typical sequence of commands is

```
V = arxstruc(ze,zv,struc(2,2,1:10));
nn = selstruc(V,0);
nk = nn(3);
V = arxstruc(ze,zv,struc(1:5,1:5,nk-1:nk+1));
nn = selstruc(V)
```

where you first establish a suitable value of the delay `nk` by testing second order models with delays between one and ten. The best fit selects the delay, and then all combinations of ARX models with up to five *a* and *b* parameters are tested with delays around the chosen value (a total of 75 models).

If the model is validated on the same data set from which it was estimated; i.e., if `ze = zv`, the fit always improves as the model structure increases. You need to compensate for this automatic decrease of the loss functions. There are several approaches for this. Probably the best known technique is Akaike's Final Prediction Error (FPE) criterion and his closely related Information Theoretic Criterion (AIC). Both simulate the cross validation situation, where the model is tested on another data set.

The FPE is formed as

$$FPE = \frac{1 + n/N}{1 - n/N} * V$$

where n is the total number of estimated parameters and N is the length of the data record. V is the loss function (quadratic fit) for the structure in question. The AIC is formed as

$$AIC \approx \log[(1 + 2n/N)*V]$$

(See Section 16.4 in Ljung (1987).)

According to Akaike's theory, in a collection of different models, choose the one with the smallest FPE (or AIC). The FPE values are displayed by the `present` command and are also given as the entry (2,1) of the `th` matrix. The structure that minimizes the AIC is obtained with

```
nn = selstruc(V,'AIC')
```

where `V` was generated by `arxstruc`.

A related criterion is Rissanen's minimum description length (MDL) approach, which selects the structure that allows the shortest over-all description of the observed data. This is obtained with

```
nn = selstruc(V,'MDL')
```

If substantial noise is present, the ARX models may need to be of high order to describe simultaneously the noise characteristics and the system dynamics. (For ARX models the noise model $1/A(q)$ is directly coupled to the dynamics model $B(q)/A(q)$.) An alternative is to compute the dynamics model only, using an IV technique, and to compute the fit between the model's simulated output and the output in the validation data set `zv`. This is accomplished with

```
V = ivstruc(ze,zv,NN)
```

The information in `V` can then be handled as described above. In this case, `V` also contains the condition number of the matrix from which the IV estimates are solved. Poor conditioning of this matrix indicates unnecessarily high model orders. (See page 415 in Ljung, 1987).

Checking Pole-Zero Cancellations

A near pole-zero cancellation in the dynamics model is an indication that the model order may be too high. To judge if a "near" cancellation is a real cancellation, take the uncertainties in the pole- and zero-locations into

consideration. The function th2zp computes confidence regions for the poles and zeros, which are graphed by

```
zpplot(th2zp(th),1)
```

where the 1 indicates how many standard-deviations wide the confidence interval is. If the confidence regions overlap, try lower model orders.

This check is especially useful when the models have been generated by arx. As mentioned under "Comparing Different Structures," the orders can be pushed up due to the requirement that $1/A(q)$ describe the noise characteristics. Checking cancellations in $B(q)/A(q)$ then gives a good indication of which orders to chose from model structures like armax, oe, and bj.

Residual Analysis

The residuals associated with the data and a given model, as in (3.38), are ideally white and independent of the input for the model to correctly describe the system. The function

```
e = resid(z,th)
```

computes the residuals e and performs whiteness and independence analyses. The auto correlation function of e and the cross correlation function between e and u are computed and displayed for up to lag 25. Also displayed are 99% confidence intervals for these variables, assuming that e is indeed white and independent of u.

The rule is that if the correlation functions go significantly outside these confidence intervals, do not accept the corresponding model as a good description of the system. Some qualifications of this statement are necessary:

- Model structures like the OE structure (3.19) and methods like the IV method (3.41) focus on the dynamics G and less about the noise properties H. If you are interested primarily in G, focus on the independence of e and u rather than the whiteness of e.
- Correlation between e and u for negative lags, or current $e(t)$ affecting future $u(t)$, is an indication of output feedback. This is not a reason to reject the model. Correlation at negative lags is of interest, since certain methods do not work well when feedback is present in the input-output data, (see

"Dealing with Data" on page 3-58), but concentrate on the positive lags in the cross-correlation plot for model validation purposes.

- When using the ARX model (3.15), the least-squares procedure automatically makes the correlation between $e(t)$ and $u(t-k)$ zero for $k = nk, nk+1, \ldots nk+nb-$, for the data used for the estimation.

As part of the validation process, you can graph the residuals using

```
plot(e)
```

for a simple visual inspection of irregularities and outliers. (See also "Dealing with Data" on page 3-58.)

Noise-Free Simulations

To check whether a model is capable of reproducing the observed output when driven by the actual input, you can run a simulation:

```
yh = idsim(u,th)
plot([y yh])
```

The same result is obtained by

```
compare([y,u],th)
```

It is a much tougher and revealing test to perform this simulation, as well as the residual tests, on a fresh data set `[y u]` that was not used for the estimation of the model `th`. This is called *cross validation*. (Note that `ivstruc` forms the squared difference of `y` and `yh` for IV-computed models `th`.)

Assessing the Model Uncertainty

The estimated model is always uncertain, due to disturbances in the observed data, and due to the lack of an absolutely correct model structure. The variability of the model that is due to the random disturbances in the output is estimated by most of the estimation procedures, and it can be displayed and illuminated in a number of ways. This variability answers the question of how different can the model be if I repeat the identification procedure, using the same model structure, but with a different data set that uses the same input sequence. It does not account for systematic errors due to an inadequate choice of model structure. There is no guarantee that the "true system" lies in the confidence interval.

The uncertainty in the frequency response is calculated with

```
[g,phiv] = th2ff(th)
```

and can subsequently be graphed with `bodeplot`. The uncertainty in the time response is displayed by

```
idsimsd(u,th)
```

Ten possible models are drawn from the asymptotic distribution of the model `th`. The response of each of them to the input `u` is graphed on the same diagram.

The uncertainty of these responses concerns the external, input-output properties of the model. It reflects the effects of inadequate excitation and the presence of disturbances.

You can also directly get the standard deviation of the simulated result by

```
[ysim,ysimsd] = idsim(u,th)
```

The uncertainty in internal representations is manifested in the covariance matrix of the estimated parameters,

```
present(th)
```

and in the standard deviations of the pole- and zero-locations, which are computed with

```
zepo = th2zp(th)
```

and displayed using `zpplot`. Large uncertainties in these representations are caused by excessively high model orders, inadequate excitation, or bad signal-to-noise ratios.

Comparing Different Models

It is a good idea to display the model properties in terms of quantities that have more physical meaning than the parameters themselves. Bode plots, zero-pole plots, and model simulations all give a sense of the properties of the system that have been picked up by the model.

If several models of different characters give very similar Bode plots in the frequency range of interest, you can be fairly confident that these must reflect features of the true, unknown system. You can then choose the simplest model among these.

A typical identification session includes estimation in several different structures, and comparisons of the model properties. Here is an example:

```
a1 = arx(z,[1 1]);
[gs,nss] = spa(z);
[ga1,nsa1] = th2ff(a1);
bodeplot([gs ga1])
bodeplot([nss nsa1])
zpa1 = th2zp(a1);
am2 = armax(z,[2 2 2 1]);
[gam2,nsam2] = th2ff(am2);
bodeplot([gs gam2])
zpam2 = th2zp(am2);
zpplot(zpform(zpa1,zpam2))
```

Conditioning of the Prediction Error Gradient

When the model orders in (3.23) are overestimated, so that a pole-zero cancellation can take place in the model, the prediction error gradient theoretically becomes rank deficient. The matrix that is inverted when estimating the covariance matrix becomes singular, and the Gauss-Newton search direction becomes ambiguous.

Due to the high numeric precision in MATLAB, you are unlikely to encounter messages about singular and rank deficient matrices. Instead, overestimated model orders usually show up as large estimated parameter variances and large norms of the Gauss-Newton vector (although only minor decreases in the criterion value result). Consequently, these are good indicators that you have gone too far in increasing the model orders. See also “Some Special Topics” on page 3-68.

Selecting Model Structures for Multivariable Systems

Multivariable systems are often more difficult to model. Some basic aspects and advice were given in “The Basic Steps of System Identification” on page 1-10. The graphical user interface (GUI) offers particularly useful support for the multivariable case. In this section we give some more technical comments about the identification of multivariable systems.

The basic comments given so far in this section also apply to such systems. There are, however, a large number of possible structures for systems with several outputs, and systematic tests of the many possibilities are impractical. This problem is a difficult one, and for a more comprehensive treatment, refer to Appendix 4.A of Ljung (1987).

First of all, physical insight is more important than anything else. Whenever there is prior knowledge sufficient to form physically parametrized state-space models, you should test that. Even lacking more precise knowledge, some physical insight might help you to come up with first suggestions of orders and delays.

For multi-output black-box models, it is easiest to first try state-space models using `n4sid` for the model estimation.

Multivariable ARX models are also easy to deal with. When building such models, a simple start-up procedure is to try `arx` with a structure that is filled with parameters. Then consider those estimates that are of the same magnitude as their standard deviations, and try orders and delays that automatically set them to zero. Note that for `arx` each row is estimated independently of the others. Changing orders and delays associated with output number *i* (i.e. the `i`-th row of `na`, `nb`, and `nk`) does not change the parameter estimates associated with the other rows. When a reasonable structure has been found, try `iv4` with it and evaluate the models in the usual ways.

If the signal to noise level is not good, and it is important to have models that describe the noise characteristics, try state-space models. (These are equivalent to multivariable ARMAX models.) Again, it is easier to estimate state-space models directly without specifying the particular structure. This is done using `n4sid`.

An alternative is to apply the prediction error method using canonical state-space forms. These are obtained by the command `canstart`. This defines a canonical form structure from the pseudo-observability indices. These indices form a vector with the same number of entries as the number of outputs. Loosely speaking, index number k describes how many delayed values of y_k affect the current value of y_k. The sum of the indices is equal to the order of the

system. The number of possible pseudo-observability indices for a system of order n with p outputs is

$$\binom{n}{p}$$

It is, however, reassuring to know that almost every system of order n can be described in a structure corresponding to any set of pseudo-observability indices whose sum is n. You can estimate the order of the system (or, rather, try several different orders) and pick any set of pseudo-observability indices corresponding to that order. A default choice of indices can be made by `canstart` if only the order is specified. Only if the minimization in `pem` shows signs of poor conditioning of the involved matrices do you need to try other indices.

Note that the canonical form parametrizations cannot handle input delays. To deal with specific delays from the input(s), shift the input sequences accordingly.

Also note that with `fixpar` any canonical form parameterization can be transformed to a corresponding output-error structure, by fixing the matrix K to zero.

9. Dealing with Data

Extracting information from data is not an entirely straightforward task. In addition to the decisions required for model structure selection and validation, the data may need to be handled carefully. This section gives some advice on handling several common situations.

Offset Levels

When the data have been collected from a physical plant, they are typically measured in physical units. The levels in these raw inputs and outputs may not match in any consistent way. This will force the models to waste some parameters correcting the levels.

Offsets are easily dealt with; always subtract the mean levels from the input and output sequences before the estimation. It is best if the mean levels correspond to a physical equilibrium, but if such values are not known, use the sample means:

```
z = dtrend(z);
```

Section 14.6 in Ljung (1987) discusses this in more detail. With the `dtrend` command, you can also remove piece-wise linear trends.

Outliers

Real data are also subject to possible bad disturbances; an unusually large disturbance, a temporary sensor or transmitter failure, etc. It is important that such outliers are not allowed to affect the models too strongly.

The robustification of the error criterion (described under `lim` in “Optional Variables” on page 3-26) helps here, but it is always good practice to examine the residuals for unusually large values, and to go back and critically evaluate the original data responsible for the large values. If the raw data are obviously in error, they can be smoothed, and the estimation procedure repeated.

Filtering Data

Depending upon the application, interest in the model can be focused on specific frequency bands. Filtering the data before the estimation, through filters that enhance these bands, improves the fit in the interesting regions. This is used in the System Identification Toolbox function `idfilt`. For

example, to enhance the data in the frequency band between $0.02 * \pi$ and $0.1 * \pi$, execute

```
zf = idfilt(z,5,[0.02 0.1]);
```

This computes and uses a fifth order Butterworth bandpass filter with passband between the indicated frequencies. Chapter 13 in Ljung (1987) discusses the role of filtering in more detail.

The SITB contains other useful commands for related problems. For example, if you want to lower the sampling rate by a factor of 5, use

```
z5 = idresamp(z,5);
```

Feedback in Data

If the system was operating in closed loop (feedback from the past outputs to the current input) when the data were collected, some care has to be exercised.

Basically, all the prediction error methods work equally well for closed-loop data. Note, however, that the Output-Error model (3.19) and the Box-Jenkins model (3.21) are normally capable of giving a correct description of the dynamics *G*, even if *H* (which equals 1 for the output error model) does not allow a correct description of the noise properties. This is no longer true for closed-loop data. You then need to model the noise properties carefully.

The spectral analysis method and the instrumental variable techniques (with default instruments) give unreliable results when applied to closed-loop data. These techniques should be avoided when feedback is present.

To detect if feedback is present, use the basic method of applying `cra` to estimate the impulse response. Significant values of the impulse response at negative lags is a clear indication of feedback. When a parametric model has been estimated and the `resid` command is applied, a graph of the correlation between residuals and inputs is given. Significant correlation at negative lags again indicates output feedback in the generation of the input. Testing for feedback is, therefore, a natural part of model validation.

Delays

The selection of the delay `nk` in the model structure is a very important step in obtaining good identification results. You can get an idea about the delays in the system by the impulse response estimate from `cra`.

Incorrect delays are also visible in parametric models. Underestimated delays (nk too small) show up as small values of leading b_k estimates, compared to their standard deviations. Overestimated delays (nk too large) are usually visible as a significant correlation between the residuals and the input at the lags corresponding to the missing b_k terms.

A good procedure is to start by using arxstruc to test all feasible delays together with a second-order model. Use the delay that gives the best fit for further modeling. When you have found an otherwise satisfactory structure, vary nk around the nominal value within the structure, and evaluate the results.

10. Recursive Parameter Estimation

In many cases it may be necessary to estimate a model on line at the same time as the input-output data is received. You may need the model to make some decision on line, as in adaptive control, adaptive filtering, or adaptive prediction. It may be necessary to investigate possible time variation in the system's (or signal's) properties during the collection of data. Terms like *recursive identification*, *adaptive parameter estimation*, *sequential estimation*, and *online algorithms* are used for such algorithms. Chapter 11 in Ljung (1987) deals with such algorithms in some detail.

The Basic Algorithm

A typical recursive identification algorithm is

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \hat{y}(t)) \tag{10.1}$$

Here $\hat{\theta}(t)$ is the parameter estimate at time t, and $y(t)$ is the observed output at time t. Moreover, $\hat{y}(t)$ is a prediction of the value $y(t)$ based on observations up to time $t-1$ and also based on the current model (and possibly also earlier ones) at time $t-1$. The gain $K(t)$ determines in what way the current prediction error $y(t) - \hat{y}(t)$ affects the update of the parameter estimate. It is typically chosen as

$$K(t) = Q(t)\psi(t) \tag{10.2}$$

where $\psi(t)$ is (an approximation of) the gradient with respect to θ of $\hat{y}(t|\theta)$. The latter symbol is the prediction of $y(t)$ according the model described by θ. Note that model structures like AR and ARX that correspond to linear regressions can be written as

$$y(t) = \psi^T(t)\theta_0(t) + e(t) \tag{10.3}$$

where the *regression vector* $\psi(t)$ contains old values of observed inputs and outputs, and $\theta_0(t)$ represents the true description of the system. Moreover, $e(t)$ is the noise source (the innovations). Compare with (3.15). The natural prediction is $\hat{y}(t) = \psi^t(t)\theta(t-1)$ and its gradient with respect to θ becomes exactly $\psi(t)$.

For models that cannot be written as linear regressions, you cannot recursively compute the exact prediction and its gradient for the current estimate $\theta(t-1)$. Then approximations $\hat{y}(t)$ and $\psi(t)$ must be used instead. Section 11.4 in Ljung (1987) describes suitable ways of computing such approximations for general model structures.

The matrix $Q(t)$ that affects both the adaptation gain and the direction in which the updates are made, can be chosen in several different ways. This is discussed in the following section.

Choosing an Adaptation Mechanism and Gain

The most logical approach to the adaptation problem is to assume a certain model for how the "true" parameters θ_0 change. A typical choice is to describe these parameters as a random walk:

$$\theta_0(t) = \theta_0(t-1) + w(t) \tag{10.4}$$

Here $w(t)$ is assumed to be white Gaussian noise with covariance matrix

$$Ew(t)w^T(t) = R_1 \tag{10.5}$$

Suppose that the underlying description of the observations is a linear regression (10.3). An optimal choice of $Q(t)$ in (10.1)-(10.2) can then be computed from the Kalman filter, and the complete algorithm becomes

$$\hat{\theta}(t) = \hat{\theta}(t-1) + K(t)(y(t) - \hat{y}(t)) \tag{10.6a}$$

$$\hat{y}(t) = \psi^T(t)\hat{\theta}(t-1) \tag{10.6b}$$

$$K(t) = Q(t)\psi(t) \tag{10.6c}$$

$$Q(t) = \frac{P(t-1)}{R_2 + \psi(t)^T P(t-1)\psi(t)} \tag{10.6d}$$

$$P(t) = P(t-1) + R_1 - \frac{P(t-1)\psi(t)\psi(t)^T P(t-1)}{R_2 + \psi(t)^T P(t-1)\psi(t)} \tag{10.6e}$$

Here R_2 is the variance of the innovations $e(t)$ in (10.3): $R_2 = Ee^2(t)$ (a scalar). The algorithm (10.6) will be called the Kalman Filter (KF) Approach to adaptation, with *drift matrix* R_1. See eq (11.66)-(11.67) in Ljung (1987). The algorithm is entirely specified by $R_1, R_2, P(0), \theta(0)$, and the sequence of data $y(t), \psi(t)$, $t = 1, 2, \ldots$. Even though the algorithm was motivated for a linear regression model structure, it can also be applied in the general case where $\hat{y}(t)$ is computed in a different way from (10.6b).

Another approach is to discount old measurements exponentially, so that an observation that is τ samples old carries a weight that is λ^τ of the weight of the most recent observation. This means that the following function is minimized rather than (3.39):

$$\sum_{k=1}^{t} \lambda^{t-k} e^2(k) \tag{10.7}$$

at time t. Here λ is clearly is a positive number ≤ 1. The measurements that are older than $\tau = 1/(1-\lambda)$ carry a weight in (10.7) that is less than about 0.3. Think of $\tau = 1/(1-\lambda)$ as the *memory horizon* of the approach. Typical values of λ are in the range 0.97– 0.995.

The criterion (10.7) can be minimized exactly in the linear regression case giving the algorithm (10.6abc) with the following choice of $Q(t)$:

$$Q(t) = P(t) = \frac{P(t-1)}{\lambda + \psi(t)^T P(t-1)\psi(t)} \tag{10.8a}$$

$$P(t) = \left(P(t-1) - \frac{P(t-1)\psi(t)\psi(t)^T P(t-1)}{\lambda + \psi(t)^T P(t-1)\psi(t)} \right) / \lambda \tag{10.8b}$$

This algorithm will be called the Forgetting Factor (FF) Approach to adaptation, with the *forgetting factor* λ. See eq (11.63) in Ljung (1987). The algorithm is also known as RLS, *recursive least squares* in the linear regression case. Note that $\lambda = 1$ in this approach gives the same algorithm as $R_1 = 0, R_2 = 1$ in the Kalman filter approach.

A third approach is to allow the matrix $Q(t)$ to be a multiple of the identity matrix:

$$Q(t) = \Upsilon I \tag{10.9}$$

It can also be normalized with respect to the size of ψ:

$$Q(t) = \frac{\Upsilon}{|\psi(t)|^2} I \tag{10.10}$$

See eqs (11.45) and (11.46), respectively in Ljung (1987). These choices of $Q(t)$ move the updates of $\hat{\theta}$ in (10.1) in the negative gradient direction (with respect to θ) of the criterion (3.39). Therefore, (10.9) will be called the Unnormalized Gradient (UG) Approach and (10.10) the Normalized Gradient (NG) Approach to adaptation, with gain Υ. The gradient methods are also known as LMS, *least mean squares* in the linear regression case.

Available Algorithms

The System Identification Toolbox provides the following functions that implement all common recursive identification algorithms for model structures in the family (6.1): `rarmax`, `rarx`, `rbj`, `rpem`, `rplr`, and `roe`. They all share the following basic syntax:

```
[thm,yh] = rfcn(z,nn,adm,adg)
```

Here `z` contains the output-input data as usual. `nn` specifies the model structure, exactly as for the corresponding offline algorithm. The arguments `adm` and `adg` select the adaptation mechanism and adaptation gain listed above.

```
adm = 'ff'; adg = lam
```

gives the forgetting factor algorithm (10.8), with forgetting factor `lam`.

```
adm = 'ug'; adg = gam
```

gives the unnormalized gradient approach (10.9) with gain `gam`. Similarly,

```
adm = 'ng'; adg = gam
```

gives the normalized gain approach (10.10). To obtain the Kalman filter approach (10.6) with drift matrix R1 enter

```
adm = 'kf'; adg = R1
```

The value of R_2 is always 1. Note that the estimates $\hat{\theta}$ in (10.6) are not affected if all the matrices R_1, R_2 and $P(0)$ are scaled by the same number. You can therefore always scale the original problem so that R_2 becomes 1.

The output argument `thm` is a matrix that contains the current models at the different samples. Row `k` of `thm` contains the model parameters, in alphabetical order at sample time k, corresponding to row `k` in the data matrix `z`. The ordering of the parameters is the same as `th2par` would give when applied to the theta format of a corresponding offline model.

The output argument `yh` is a column vector that contains, in row `k`, the predicted value of $y(k)$, based on past observations and current model. The vector `yh` thus contains the adaptive predictions of the outputs, and is useful also for noise cancelling and other adaptive filtering applications.

The functions also have optional input arguments that allow the specification of $\theta(0), P(0)$, and $\psi(0)$. Optional output arguments include the last value of the matrix P and of the vector ψ.

Now, rarx is a recursive variant of arx; similarly rarmax is the recursive counterpart of armax and so on. Note, though that rarx does not handle multi-output systems, and rpem does not handle state-space structures.

The function rplr is a variant of rpem, and uses a different approximation of the gradient ψ. It is known as the *recursive pseudo-linear regression approach*, and contains some well known special cases. See equation (11.57) in Ljung (1987). When applied to the output error model (nn=[0 nb 0 0 nf nk]) it results in methods known as HARF ('ff'–case) and SHARF ('ng'–case). The common *extended least squares* (ELS) method is an rplr algorithm for the ARMAX model (nn=[na nb nc 0 0 nk]).

The following example shows a second order output error model, which is built recursively and its time varying parameter estimates plotted as functions of time:

```
thm = roe(z,[2 2 1],'ff',0.98);
plot(thm)
```

A second order ARMAX model is recursively estimated by the ELS method, using Kalman filter adaptation. The resulting static gains of the estimated models are then plotted as a function of time:

```
[N,dum]=size(z);
thm = rplr(z,[2 2 2 0 0 1],'kf',0.01*eye(6));
nums = sum(thm(:,3:4)')';
dens = ones(N,1)+sum(thm(:,1:2)')';
stg = nums./dens;
plot(stg)
```

So far, the examples of applications where a batch of data is examined cover studies of the variability of the system. The algorithms are, however, also prepared for true online applications, where the computed model is used for some online decision. This is accomplished by storing the update information in $\theta(t-1)$, $P(t-1)$ and information about past data in $\phi(t-1)$ (and $\psi(t-1)$) and using that information as initial data for the next time step. The following example shows the recursive least-squares algorithm being used on line (just to plot one current parameter estimate):

```
%Initialization, first i/o pair y,u (scalars)
[th,yh,P,phi] = rarx([y u],[2 2 1],'ff',0.98);
axis([1 50 -2 2])
plot(1,th(1),'*'),hold
```

```
%The online loop:
for k = 2:50
% At time k receive y,u
[th,yh,P,phi] = ...
rarx([y u],[2 2 1],'ff',0.98,th',P,phi);
plot(k,th(1),'*')
end
```

Executing `iddemo 6` illustrates the recursive algorithms.

Segmentation of Data

Sometimes the system or signal exhibits abrupt changes during the time when the data is collected. It may be important in certain applications to find the time instants when the changes occur and to develop models for the different segments during which the system does not change. This is the *segmentation problem*. Fault detection in systems and detection of trend breaks in time series can serve as two examples of typical problems.

The System Identification Toolbox offers the function `segment` to deal with the segmentation problem. The basic syntax is

```
thm = segment(z,nn)
```

with a format like `rarx` or `rarmax`. The matrix `thm` contains the piecewise constant models in the same format as for the algorithms described earlier in this section.

The algorithm that is implemented in `segment` is based on a model description like (10.4), where the change term $w(t)$ is zero most of the time, but now and then it abruptly changes the system parameters $\theta_0(t)$. Several Kalman filters that estimate these parameters are run in parallel, each of them corresponding to a particular assumption about when the system actually changed. The relative reliability of these assumed system behaviors is constantly judged, and unlikely hypotheses are replaced by new ones. Optional arguments allow the specification of the measurement noise variance R_2 in (10.3), of the probability of a jump, of the number of parallel models in use, and also of the guaranteed lifespan of each hypothesis. See `iddemo 7` in Chapter 4, "Command Reference" also.

11. Some Special Topics

This section describes a number of miscellaneous topics. Most of the information here is also covered in other parts of the manual, but since manuals seldom are read from the beginning, you can also check if a particular topic is brought up here.

Time Series Modeling

When there is no input present, the general model (6.1) reduces to the ARMA model structure:

$$A(q)y(t) = C(q)e(t) \tag{11.1}$$

With $C(q) = 1$ you have an AR model structure.

Basically all commands still apply to these time series models, but with natural modifications. They are listed as follows:

```
th = poly2th(A,[],C,1,[],lam)
y = idsim(e,th)
```

Spectral analysis (`etfe` and `spa`) and the `th2ff` function now return the spectral estimate of *y* as their first output arguments:

```
PHIY = th2ff(th)
PHIY = spa(y)
PERIOD = etfe(y)
```

Note that `etfe` gives the *periodogram* estimate of the spectrum.

`armax` and `arx` work the same way, but need no specification of `nb` and `nk`:

```
th = arx(y,na)
th = armax(y,[na nc])
```

Note that `arx` also handles multivariate signals. State-space models of time series can be built simply by specifying `B = []`, `D = []` in `modstruc`, `mf2th`, and `ms2th`. `resid` works the same way for time series models, but does not provide any input-residual correlation plots:

```
e = resid(y,th)
```

In addition there are two commands that are specifically constructed for building scalar AR models of time series. One is

```
th = ar(y,na)
```

which has an option that allows you to choose the algorithm from a group of several popular techniques for computing the least-squares AR model. Among these are Burg's method, a geometric lattice method, the Yule-Walker approach, and a modified covariance method. See Chapter 4, "Command Reference" for details. The other command is

```
th = ivar(y,na)
```

which uses an instrumental variables technique to compute the AR part of a time series.

Finally, when no input is present, the functions `bj`, `iv`, `iv4`, and `oe` are not defined.

Here is an example where you can simulate a time series, compare spectral estimates and covariance function estimates, and also the predictions of the model.

```
ts0 = poly2th([1 –1.5 0.7],[]);
  % The true spectrum:
spe0 = th2ff(ts0);
ir = idsim([1;zeros(24,1)],ts0);
  % The true covariance function:
Ry0 = conv(ir,ir(25:–1:1));
e = randn(200,1);
y = idsim(e,ts0);
plot(y)
per = etfe(y);
speh = spa(y);
ffplot([per,speh,spe0])
  % A second order AR model:
ts2 = ar(y,2);
sp2 = th2ff(ts2);
ffplot([speh,sp2,spe0,3])
  % The covariance function estimates:
Ryh = covf(y,24);
ir2 = idsim([1;zeros(24,1)],ts2);
Ry2 = conv(ir2,ir2(25:–1:1));
```

```
plot([-24:24]*ones(1,3),[Ryh,Ry2,Ry0])
  % The prediction ability of the model:
compare(y,th2,5)
```

The Sampling Interval

The System Identification Toolbox assumes a normalized sampling interval of $T = 1$, unless you specify otherwise. This means that the frequency unit becomes "radians per sampling interval," and that all the transformations between continuous- and discrete-time models give numerical parameter values corresponding to the unit "... per sampling interval" rather than "... per second." To obtain correct physical units in these cases, the sampling interval must be specified accordingly.

All models in the theta format contain the sampling interval T as the `(1,2)` element. It can be read by

```
T = gett(th)
```

and it is changed by

```
thnew = sett(thold,T)
```

Note that the value of T is just changed; no transformation of parameters is carried out by `sett`.

A negative value of T indicates that the corresponding model is given in continuous time. The absolute value of T denotes, in that case, the sampling interval of the data, for which this model was estimated. This is also the sampling interval used when the model is used for simulation or prediction. All other transformations (to the frequency function, to poles and zeros, to transfer functions and to state-space matrices) will be in terms of the continuous-time model. In the case of `th2ff`, `abs(T)` is used for the default choice of frequency range.

You can specify the sampling interval as an optional argument in all the functions that create theta structures. It is generally the last argument. Note that in `ms2th` and `mf2th`, it is the sampling interval of the data that you need to specify, even if the model itself in parametrized in continuous time. If the initial structure in `pem`, `armax`, `bj`, and `oe` is given by a matrix `th` in the theta format, the default value of T is the one given by `th`. An explicitly given sampling interval (as an argument to `pem`, for example) overrides the default value in `th`.

The sampling interval *T* is also used by `etfe`, `spa`, and `th2ff` to determine default frequency ranges. For discrete-time systems, this default is `[1:128]/128*pi/T` and for continuous-time systems, it is

logspace(log10(pi/abs(T)/100),log10(10*pi/abs(T)),128).

Discrete-time default frequency ranges can be changed by `sett`, like in

```
g = spa(z);
g = sett(g,T);
```

Out of Memory

If you run out of memory on computers with memory limitations, clear unnecessary variables and use the `pack` function. When you are using the iterative procedures `armax`, `bj`, `oe`, and `pem`, it is a good idea to use `pack` to clean up after the start-up estimation procedure using, for example

```
th = armax(z,nn,0);
pack
th = armax(z,th);
```

All the relevant System Identification Toolbox functions have an optional variable `MAXSIZE` that controls the memory/speed trade-off and is set automatically, based upon the best values for the machine you are using. The routines do not allow any matrix of size larger than `MAXSIZE` to form. Instead they go into for-loops to handle the situation.

The default value of `MAXSIZE` is set in the M-file `idmsize`. Theoretically, `MAXSIZE` is 8188 on the smaller machines, but since many matrices may form, it best to set the default value for these machines to `MAXSIZE = 4096`.

If you have memory problems, you can use the argument `MAXSIZE`, or edit `idmsize`, with values lower than the default.

If you have a large amount of input-output data, so that the memory delimiter gives you a slower response, use a portion of the data to calculate a good initial condition for the minimization, for example

```
th = armax(z(1:500,:),[na nb nc],10,0.1);
th = armax(z,th);
```

On a small machine, if you have collected more than 8188 data points so that you cannot form `z=[y u]`, no memory help is offered. You then have to build separate models for separate parts of the data and reconcile the models.

Memory-Speed Trade-Offs

On machines with no formal memory limitations, it is still of interest to monitor the sizes of the matrices that are formed. The typical situation is when an overdetermined set of linear equations is solved for the least-squares solution. The solution time depends, of course, on the dimensions of the corresponding matrix. The number of rows corresponds to the number of observed data, while the number of columns corresponds to the number of estimated parameters. The argument `MAXSIZE` used with all the relevant M-files, guarantees that no matrix with more than `MAXSIZE` elements is formed. Larger data sets and/or higher order models are handled by for-loops. For-loops give linear increase in time when the data record is increased, plus some overhead (and somewhat worse numerical properties).

If you regularly work with large data sets and/or high order models, it is advisable to tailor the memory and speed trade-off to your machine by choosing the default value of `MAXSIZE` in the M-file `idmsize` carefully. Note that this value is allowed to depend on the number of rows and columns of the matrices formed.

Regularization

The Gauss-Newton direction is usually defined as

$$gn = (\Psi' * \Psi)^{-1} * \Psi' * e$$

where Ψ is the $N \times n$ gradient matrix of the predictions with respect to the parameters, and e is the $N \times 1$ vector of residuals.

When the inverted matrix is ill-conditioned, you can add a small fraction of the identity matrix to it. This is called *regularization*. See Ljung (1987), Section 10.2.

The routines in the System Identification Toolbox compute the Gauss-Newton direction with

```
g=psi\e
```

which uses the MATLAB mechanism for dealing with underdetermined, overdetermined systems of equations. An advanced version of regularization is thus automatically included.

Because of the high precision of MATLAB, regularization is not invoked easily. You may see some very large `norm of gn-vector` values before you see the MATLAB message that the matrix is rank deficient.

A large value for the `gn-vector` suggests that you decrease the model orders. However, if you are interested in how the regularization works, you can decrease the value of the machine epsilon. For example, set

```
eps = 0.000001
```

and run the identification again.

Local Minima

The iterative search procedures in `pem`, `armax`, `oe`, and `bj` lead to theta values corresponding to a local minimum of the criterion function (3.39). Nothing guarantees that this local minimum is also a global minimum. The start-up procedure for black-box models in these routines is, however, reasonably efficient in giving initial estimates that lead to the global minimum.

If there is an indication that a minimum is not as good as you expected, try starting the minimization at several different initial conditions, to see if a smaller value of the loss function can be found.

Initial Parameter Values

When only orders and delays are specified, the functions `armax`, `bj`, `oe`, and `pem` use a start-up procedure to produce initial values. The start-up procedure goes through two to four least squares and instrumental variables steps. It is reasonably efficient in that it usually saves several iterations in the minimization phase. Sometimes it may, however, pay to use other initial conditions. For example, you can use an `iv4` estimate computed earlier as an initial condition for estimating an output-error model of the same structure:

```
th1 = iv4(z,[na nb nk]);
[a,b] = th2poly(th1);
th2 = poly2th(1,b,1,1,a);
th3 = oe(z,th2);
```

Another example is when you want to try a model with one more delay (for example, three instead of two) because the leading *b*-coefficient is quite small:

```
th1 = armax(z,[3 3 2 2]);
[a,b,c] = th2poly(th1);
b(3) = 0;
th2 = poly2th(a,b,c);
th3 = armax(z,th2);
```

If you decrease the number of delays, remember that `poly2th` strips away leading zeros. Suppose you go from three to two delays in the above example:

```
th1 = armax(z,[3 3 2 3]);
[a,b,c] = th2poly(th1);
b(3) = 0.00001;
th2 = poly2th(a,b,c);
th3 = armax(z,th2);
```

Note that when constructing home-made initial conditions, the conditions must correspond to a stable predictor (*C* and *F* being Hurwitz polynomials), and that they should not contain any exact pole-zero cancellations.

For state-space and multi-output models, you must provide the initial parameter values either when defining the structure in `ms2th` or `mf2th`, or with the special initialization function `thinit`. The basic approach is to use physical insight to choose initial values of the parameters with physical significance, and try some different (randomized) initial values for the others. For models parametrized in canonical state-space form, use `canstart` to get started.

Linear Regression Models

A linear regression model is of the type

$$y(t) = \theta^T \varphi(t) + e(t) \tag{11.2}$$

where $y(t)$ and $\varphi(t)$ are measured variables and $e(t)$ represents noise. Such models are very useful in most applications. They allow, for example, the inclusion of nonlinear effects in a simple way. The System Identification Toolbox function `arx` allows an arbitrary number of inputs. You can therefore handle arbitrary linear regression models with `arx`. For example, if you want to build a model of the type

$$y(t) = b_0 + b_1 u(t) + b_2 u^2(t) + b_3 u^3(t) \qquad (11.3)$$

call

```
z = [y ones(u) u u.^2 u.^3];
th = arx(z,[0 1 1 1 1 0 0 0 0])
```

This is formally a model with one output and four inputs, but all the model testing in terms of `compare`, `idsim`, and `resid` operate in the natural way for the model (11.2), once the data matrix `z` is defined as above.

Note that when `pem` is applied to linear regression structures, by default a robustified quadratic criterion is used. The search for a minimum of the criterion function is carried out by iterative search. Normally, use this robustified criterion. If you insist on a quadratic criterion, then set the argument `lim` in `pem` to zero. Then `pem` also converges in one step.

Spectrum Normalization and the Sampling Interval

In the function `spa` the spectrum estimate is normalized with the sampling interval T as

$$\Phi_y(\omega) = T \sum_{k=-M}^{M} \hat{R}_y(kT) e^{-i\omega T} W_M(k) \qquad (11.4)$$

where

$$\hat{R}_y(kT) = \frac{1}{N} \sum_{l=1}^{N} y(lT - kT) y(lT)$$

(See also (3.3)). The normalization in `etfe` is consistent with (11.4). This normalization means that the unit of $\Phi_y(\omega)$ is "power per radians/time unit" and that the frequency scale is "radians/time unit." You then have

$$Ey^2(t) = \frac{1}{2\pi} \int_{-\pi/T}^{\pi/T} \Phi_y(w) dw \qquad (11.5)$$

In MATLAB language, therefore, you have $S1 \approx S2$ where

```
sp = spa(y); sp = sett(sp,T);
[om,PHIY] = getff(sp);
S1 = sum(PHIY)/length(PHIY)/T;
S2 = sum(y.^2)/length(y);
```

Note that `PHIY` contains $\Phi_y(\omega)$ between $\omega = 0$ and $\omega = \pi/T$ with a frequency step of $\pi / T /$ (`length`($PHIY$)). The sum $S1$ is, therefore, the rectangular approximation of the integral in (11.5). The spectrum normalization differs from the one used by `spectrum` in the Signal Processing Toolbox, and the above example shows the nature of the difference.

The normalization with T in (11.4) also gives consistent results when time series are decimated. If the energy above the Nyquist frequency is removed before decimation (as is done in `idresamp`), the spectral estimates coincide; otherwise you see folding effects.

Try the following sequence of commands:

```
th0 = poly2th(1,[],[1 1 1 1]);
       % 4th order MA-process
y = idsim(randn(2000,1),th0);
g1 = spa(y);
g2 = spa(y(1:4:2000));g2 = sett(g2,4);
ffplot([g1 g2]) % Folding effects
g3 = spa(idresamp(y,4)); % Prefilter applied
g3 = sett(g3,4);
ffplot([g1 g3]) % No folding
```

For a parametric noise (time series) model

$$y(t) = H(q)e(t) \qquad Ee^2(t) = \lambda$$

the spectrum is computed as

$$\Phi y(w) = \lambda \cdot T \cdot \left|H(e^{iwt})\right|^2 \tag{11.6}$$

which is consistent with (11.4) and (11.5). Think of λT as the spectral density of the white noise source $e(t)$.

When a parametric noise model is transformed between continuous time and discrete time and/or resampled at another sampling rate, the functions

thc2thd and thd2thc in the System Identification Toolbox use formulas that are formally correct only for piecewise constant inputs. (See (3.32)). This approximation is good when T is small compared to the bandwidth of the noise. During these transformations the variance λ of the innovations $e(t)$ is changed so that the spectral density $T\ \lambda \cdot$ remains constant. This has two effects:

- The spectrum scalings are consistent, so that the noise spectrum is essentially invariant (up to the Nyquist frequency) with respect to resampling.
- Simulations with noise using idsim has a higher noise level when performed at faster sampling.

This latter effect is well in line with the standard description that the underlying continuous-time model is subject to continuous-time white noise disturbances (which have infinite, instantaneous variance), and appropriate low-pass filtering is applied before sampling the measurements. If this effect is unwanted in a particular application, scale the noise source appropriately before applying idsim.

Note the following cautions relating to these transformations of noise models. Continuous-time noise models must have a white noise component. Otherwise the underlying state-space model, which is formed and used in thc2thd and thd2thc, is ill-defined. Warnings about this are issued by poly2th and these functions. Modify the *C*-polynomial accordingly. Make the degree of the monic *C*-polynomial in continuous time equal to the sum of the degrees of the monic *A*- and *D*-polynomials; i.e., in continuous time

```
length(C)-1 = (length(A)-1)+(length(D)-1).
```

Interpretation of the Loss Function

The value of the quadratic loss function is given as element 1,1 in the theta format, and also displayed by present. For multi-output systems, this is equal to the determinant of the estimated covariance matrix of the innovations.

For most models the estimated covariance matrix of the innovations is obtained by forming the corresponding sample mean of the prediction errors, computed (using pe) from the model with the data for which the model was estimated.

Note the discrepancy between this value and the values shown during the minimization procedure (in `pem`, `armax`, `bj`, or `oe`, since these are the values of the *robustified* loss function (see under `lim` in Section 5).

Be careful when comparing loss function values between different structures that use very different noise models. An Output-Error model may have a better input-output fit, even though it displays a higher value of the loss function than, say, an ARX model.

Note that for ARX models computed using `iv4`, the covariance matrix of the innovations is estimated using the provisional noise model that is used to form the optimal instruments. The loss function therefore differs from what would be obtained if you computed the prediction errors using the model directly from the data. It is still the best available estimate of the innovations covariance. In particular, it is difficult to compare the loss function in an ARX model estimated using `arx` and one estimated using `iv4`.

Enumeration of Estimated Parameters

In some cases the parameters of a model are given just as an ordered list. This is the case for `th2par`, and also when online information from the minimization routines are given. Furthermore, in `ms2th`, you are asked to give a list of nominal/initial parameter values. In these cases it is important to know in what order the parameter are listed. The basic three cases are as follows:

- For the input-output model (3.23) or its multi-input variant (5.2), you have the following alphabetical ordering

$$pars = [a_1, \ldots, a_{na}, b_1^1, \ldots, b_{nb1}^1, b_1^2, \ldots b_{nb2}^2, \ldots$$
$$b_1^{nu}, \ldots, b_{nbnu}^{nu}, c_1, \ldots, c_{nc}, d_1, \ldots, d_{nc},$$
$$f_1^1, \ldots f_{nf1}^1, \ldots, f_1^{nu}, \ldots, f_{nfnu}^{nu}]$$

Here superscript refers to the input number.

- For a state-space structure, defined by `ms2th`, the parameters in `pars` are obtained in the following order. The *A* matrix is first scanned row by row for free parameters. Then the *B* matrix is scanned row by row, and so on for the *C*, *D*, *K*, and *X*0 matrices. (See `ms2th` in the *Command Reference* chapter.)
- For a state-space matrix that is defined by `mf2th`, the ordering of the parameters is the same as in the user-written M-file.

Multivariate ARX models are internally represented in state-space form. The parameter ordering follows the one described above. The ordering of the parameters may, however, not be transparent so it is better to use `th2arx` and `arx2th` instead of `th2par` and `ms2th`.

Complex-Valued Data

Some applications of system identification work with complex valued data, and thus create complex-valued models. Most of the routines in the SITB support complex data and models. This is true for the estimation routines `ar`, `armax`, `arx`, `bj`, `covf`, `ivar`, `iv4`, `oe`, `pem`, and `spa`, but (currently) not for `canstart` and `n4sid`. The transformation routines, like `th2ff`, `th2zp`, etc. also work for complex-valued models, but no pole-zero confidence regions are given. Note also that the parameter variance-covariance information then refers to the complex valued parameters, so no separate information about the accuracy of the real and imaginary parts will be given. Some display functions like `compare` and
`idplot` do not work for the complex case. Use `idsim` and plot real and imaginary parts separately.

Strange Results

Strange results can of course be obtained in any number of ways. We only point out two cautions: It is tempting in identification applications to call the residuals "eps." *Don't do that.* This changes the machine ε, which certainly will give you strange results.

It is also natural to use names like `step`, `phase`, etc., for certain variables. Note though that these variables take precedence over M-files with the same name so be sure you don't use variable names that also are names of M-files.

4

Command Reference

This chapter contains detailed descriptions of all of the functions in the System Identification Toolbox. It begins with a list of functions grouped by subject area and continues with the entries in alphabetical order. Information is also available through the online Help facility.

By typing a function name without arguments, you also get immediate syntax help about its arguments.

For ease of use, most functions have several default arguments. The Synopsis first lists the function with the necessary input arguments and then with all the possible input arguments. The functions can be used with any number of arguments between these extremes. The rule is that missing, trailing arguments are given default values, as defined in the manual. Default values are also obtained by entering the arguments as the empty matrix [ ].

MATLAB does not require that you specify all of the output arguments; those not specified are not returned. For functions with several output arguments in the System Identification Toolbox, missing arguments are, as a rule, not computed, in order to save time.

	The Graphical User Interface
`ident`	Open the interface.
`midprefs`	Set directory where to store start-up information.

	Simulation and Prediction
`idinput`	Generate input signals.
`idsim`	Simulate a general linear system.
`pe`	Compute prediction errors.
`poly2th`	Create a model structure for input-output models defined as numerator and denominator polynomials.
`predict`	Compute predictions according to model.

	Data Manipulation
`dtrend`	Remove trends from data.
`idfilt`	Filter data.
`idresamp`	Resample data.

	Nonparametric Estimation
`covf`	Estimate covariance function.
`cra`	Estimate impulse response and covariance functions using correlation analysis.
`etfe`	Estimate spectra and transfer functions using direct Fourier techniques.
`spa`	Estimate spectra and transfer functions using spectral analysis.

	Parameter Estimation
`ar`	Estimate AR model.
`armax`	Estimate ARMAX model.
`arx`	Estimate ARX model using least squares.
`bj`	Estimate Box-Jenkins model.
`canstart`	Estimate multivariate models in canonical state-space form.
`ivar`	Estimate AR model using instrumental variable methods.
`ivx`	Estimate ARX model using general instruments.
`iv4`	Estimate ARX model using four-stage instrumental variable method.

	Parameter Estimation
`oe`	Estimate Output-Error model.
`n4sid`	Estimate state-space model using subspace method.
`pem`	Estimate general linear model.

	Model Structure Creation
`arx2th`	Define (multivariate) ARX structures.
`canform`	Generate canonical forms.
`mf2th`	Create arbitrary linear model structure via an M-file that you write.
`modstruc`	Define state-space models with known and unknown parameters.
`ms2th`	Create model structure for linear state-space models with known and unknown parameters.
`poly2th`	Create a model structure for input-output models defined as numerator and denominator polynomials.

	Manipulating Model Structures
`fixpar`	Fix parameters in structures to given values.
`sett`	Set the sampling interval.
`ss2th`	Transform a state-space model to a parametrized canonical form.
`thinit`	Select or randomize initial parameter values.
`unfixpar`	Allow certain earlier fixed parameters be estimated.

	Model Conversions
idmodred	Reduce a model to lower order.
thc2thd	Transform from continuous to discrete time.
thd2thc	Transform from discrete to continuous time.
th2arx	Theta to ARX parameters.
th2ff	Theta to frequency functions and spectra.
th2par	Theta to estimated parameters and variances.
th2poly	Theta to transfer function polynomials.
th2ss	Theta to state-space matrices.
th2tf	Theta to transfer functions.
th2zp	Theta to zeros, poles, and static gains.

	Model Presentation
bodeplot	Plot Bode diagrams.
ffplot	Plot frequency functions and spectra.
idplot	Display input-output data.
nyqplot	Plot Nyquist diagrams.
present	Display model on screen.
zpplot	Plot zeros and poles.

	Information Extraction
`getff`	Extract the frequency functions from the freqfunc format.
`gett`	Extract the sampling interval from the theta format.
`getmfth`	Extract the M-file name that defines the model structure.
`getncap`	Extract from the theta format the number of data upon which model is based.
`getzp`	Extract the zeros and poles from the zepo format.
`th2par`	Extract estimated parameters and variances from the theta format.

	Model Validation
`compare`	Compare model's simulated or predicted output with actual output.
`idsim`	Simulate a model.
`pe`	Compute prediction errors.
`predict`	Predict future outputs.
`resid`	Compute and test model residuals.

	Assessing Model Uncertainty
`idsimsd`	Simulate responses from several possible models.
`th2ff`	Compute frequency function and its standard deviation.
`th2zp`	Compute zeros, poles, static gains, and their standard deviations.

	Model Structure Selection
`arxstruc`	Compute loss functions for sets of ARX model structure.
`ivstruc`	Compute loss functions for sets of output error model structures.
`selstruc`	Select structure.
`struc`	Generate sets of structures.

	Recursive Parameter Estimation
`rarmax`	Estimate ARMAX or ARMA models recursively.
`rarx`	Estimate ARX or AR models recursively.
`rbj`	Estimate Box-Jenkins models recursively.
`roe`	Estimate Output-Error models (IIR-filters) recursively.
`rpem`	Estimate general input-output models using a recursive prediction error method.
`rplr`	Estimate general input-output models using a recursive pseudo-linear regression method.
`segment`	Segment data and estimate models for each segment.

Purpose

Estimate the parameters of an AR model for scalar time series.

Syntax

```
th = ar(y,n)
[th,refl] = ar(y,n,approach,win,maxsize,T)
```

Description

The parameters of the AR model structure

$$A(q)y(t) = e(t)$$

are estimated using variants of the least-squares method.

Column vector `y` contains the time series data. Scalar `n` specifies the order of the model to be estimated (the number of *A* parameters in the AR model).

Note that the routine is for scalar time series only. For multivariate data use `arx`.

The estimate is returned in `th` and stored in theta format. For the two lattice-based approaches, `'burg'` and `'gl'` (see below), variable `refl` is returned containing the reflection coefficients in the first row, and the corresponding loss function values in the second. The first column is the zero-th order model, so that the (2,1) element of `refl` is the norm of the time series itself.

Variable `approach` allows you to choose an algorithm from a group of several popular techniques for computing the least-squares AR model. Available methods are as follows:

`approach = 'fb'`: The forward-backward approach. This is the default approach. The sum of a least-squares criterion for a forward model and the analogous criterion for a time-reversed model is minimized.

`approach = 'ls'`: The least-squares approach. The standard sum of squared forward prediction errors is minimized.

`approach = 'yw'`: The Yule-Walker approach. The Yule-Walker equations, formed from sample covariances, are solved.

`approach = 'burg'`: Burg's lattice-based method. The lattice filter equations are solved, using the harmonic mean of forward and backward squared prediction errors.

`approach = 'gl'`: A geometric lattice approach. As in Burg's method, but the geometric mean is used instead of the harmonic one. .

The computation of the covariance matrix can be suppressed in any of the above methods by ending the `approach` argument with 0 (zero), for example, `'burg0'`.

Windowing, within the context of AR modeling, is a technique for dealing with the fact that information about past and future data is lacking. There are a number of variants available:

`window = 'now'`: No windowing. This is the default value, except when `approach = 'yw'`. Only actually measured data are used to form the regression vectors. The summation in the criteria starts only at time `n`.

`window = 'prw'`: Pre-windowing. Missing past data are replaced by zeros, so that the summation in the criteria can be started at time zero.

`window = 'pow'`: Post-windowing. Missing end data are replaced by zeros, so that the summation can be extended to time N + `n`. (N being the number of observations.)

`window = 'ppw'`: Pre- and post-windowing. This is used in the Yule-Walker approach.

The combinations of approaches and windowing have a variety of names. The least-squares approach with no windowing is also known as the *covariance method*. This is the same method that is used in the `arx` routine. The MATLAB default method, forward-backward with no windowing, is often called the *modified covariance method*. The Yule-Walker approach, least-squares plus pre- and post-windowing, is also known as the *correlation method*.

`ar` only handles scalar time series. For multivariate series, use `arx`.

See `auxvar` for an explanation of the input arguments `maxsize` and `T`.

Examples

Compare the spectral estimates of Burg's method with those found from the forward-backward nonwindowed method, given a sinusoid in noise signal:

```
y = sin([1:300]') + 0.5*randn(300,1);
thb = ar(y,4,'burg');
thfb = ar(y,4);
sgb = th2ff(thb);
sfb = th2ff(thfb);
bodeplot([sgb sfb])
```

See Also

auxvar, arx, etfe, ivar, spa, theta

References

Marple, Jr., S. L. *Digital Spectral Analysis with Applications*, Prentice Hall, Englewood Cliffs, 1987, Chapter 8.

armax

Purpose

Estimate the parameters of an ARMAX or ARMA model.

Syntax

```
th = armax(z,nn)
th = armax(z,nn,'trace')
[th, iter_info] = armax(z,nn,maxiter,tol,lim,maxsize,T,'trace')
```

Description

The parameters of the ARMAX model structure

$$A(q)y(t) = B(q)u(t-nk) + C(q)e(t)$$

are estimated using a prediction error method.

Matrix `z` contains the output-input data `z = [y u]` where `y` and `u` are column vectors (`u` is a matrix in the multi-input case). `nn` can be given either as

```
nn = [na nb nc nk]
```

or as

```
nn = thi
```

In the former case `na`, `nb`, and `nc` are the orders of the ARMAX model, and `nk` is the delay. In the latter case `thi` is an initial value, given in theta format. See Section 3 in the *Tutorial* for an exact definition of the orders.

For multi-input systems, `nb` and `nk` are row vectors, such that the `k`-th entry corresponds to the order and delay associated with the `k`-th input.

If `z = y` and `nn = [na nc]`, `armax` calculates an ARMA model for `y`:

$$A(q)y(t) = C(q)e(t)$$

`th` is returned with the resulting parameter estimates, together with estimated covariances, stored in theta format.

`armax` does not support multi-output models. Use state-space model for this case (see `canstart`, `n4sid`, and `pem`)

If a last argument `'trace'` is supplied, information about the progress of the iterative search for the model will be furnished to the MATLAB command window.

The optional auxiliary variables `iter_info`, `lim`, `maxiter`, `tol`, `maxsize`, and `T` are explained under `auxvar`.

Algorithm

A robustified quadratic prediction error criterion is minimized using an iterative Gauss-Newton algorithm. The Gauss-Newton vector is bisected up to 10 times until a lower value of the criterion is found. If no such value is found, a gradient search direction is used instead, and the procedure is repeated. The iterations are terminated when `maxiter` is reached, when the Gauss-Newton vector has a norm less than `tol`, or when a lower value of the criterion cannot be found.

The initial conditions for the iterative search, if not specified in `nn`, are constructed in a special four-stage LS-IV algorithm.

The cut-off value for the robustification is based on the parameter `lim` as well as on the estimated standard deviation of the residuals from the initial parameter estimate. It is not recalculated during the minimization. The value returned in element `th(1,1)` is the nonrobustified, quadratic criterion.

A stability test of the predictor is performed, so as to assure that only models corresponding to stable predictors are tested. Generally, both $C(q)$ and $F_i(q)$ (if applicable) must have all their zeros inside the unit circle. Note that if an initial parameter estimate is given in `nn`, its predictor stability is taken for granted (not tested).

Information about the minimization is furnished to the screen in case the argument `'trace'` is specified.. Current and previous parameter estimates (in column vector form, listing parameters in alphabetical order) as well as the values of the criterion function are given. The Gauss-Newton vector and its norm are also displayed. The number in the upper left corner is the number of times the search vector has been bisected.

See Also

`arx`, `auxvar`, `bj`, `oe`, `pem`, `theta`

References

Ljung (1987), equations (10.41), (10.42), (10.46), (10.75)

arx

Purpose

Estimate the parameters of an ARX or AR model.

Syntax

```
th = arx(z,nn)
th = arx(z,nn,maxsize,T)
```

Description

The parameters of the ARX model structure

$$A(q)y(t) = B(q)u(t-nk) + e(t)$$

are estimated using the least-squares method.

Matrix `z` contains the output-input data `z = [y u]` where `y` and `u` are column vectors. `nn` is given as

```
nn = [na nb nk]
```

defining the orders and delay of the ARX model. See Section 6 in the *Tutorial* for exact definitions of the orders and delays.

`th` is returned as the least-squares estimates of the parameters, stored in theta format.

With `z = y` and `nn = na`, an AR model of order `na` for `y` is computed:

$$A(q)y(t) = e(t)$$

Models with several inputs

$$A(q)y(t) = B_1(q)u_1(t-nk_1) + \ldots B_{nu}u_{nu}(t-nk_{nu}) + e(t)$$

are handled by allowing `u` to contain each input as a column vector,

```
u = [u1 ... unu]
```

and by allowing `nb` and `nk` to be row vectors defining the orders and delays associated with each input.

Models with several inputs and several outputs are handled by allowing `nn` to contain one row for each output number. See "Defining Model Structures" on page 3-29 for exact definitions.

The optional auxiliary parameters `maxsize` and `T` are explained under `auxvar`.

When the true noise term $e(t)$ in the ARX model structure is not white noise and na is nonzero, the estimate does not give a correct model. It is then better to use armax, bj, iv4, or oe.

Examples

Here is an example that generates and estimates an ARX model:

```
A = [1  -1.5  0.7]; B = [0 1 0.5];
th0 = poly2th(A,B);
u = idinput(300,'rbs');
y = idsim([u,randn(300,1)],th0);
z = [y,u];
th = arx(z,[2 2 1]);
```

Algorithm

The least-squares estimation problem is an overdetermined set of linear equations that is solved using the MATLAB \ operator.

The regression matrix is formed so that only measured quantities are used (no fill-out with zeros). When the regression matrix is larger than maxsize, the normal equations are formed in a for-loop and subsequently solved.

See Also

auxvar, ar, iv, iv4, theta

arxstruc

Purpose

Compute loss functions for a set of different model structures of single-output ARX type.

Syntax

```
v = arxstruc(ze,zv,NN)
v = arxstruc(ze,zv,NN,maxsize)
```

Description

`NN` is a matrix that defines a number of different structures of the ARX type. Each row of `NN` is of the form

```
nn = [na nb nk]
```

with the same interpretation as described for `arx`. See `struc` for easy generation of typical `NN` matrices for single-input systems.

Each of `ze` and `zv` are matrices containing output-input data `[y u]`. For multi-input systems, `u` has the corresponding number of columns, while for time series, no `u` is present. Models for each of the model structures defined by `NN` are estimated using the data set `ze`. The loss functions (normalized sum of squared prediction errors) are then computed for these models when applied to the validation data set `zv`. The data sets, `ze` and `zv`, need not be of equal size. They could, however, be the same sets, in which case the computation is faster.

Note that `arxstruc` is intended for single-output systems only.

`v` is returned with the loss functions in its first row. The remaining rows of `v` contain the transpose of `NN`, so that the orders and delays are given just below the corresponding loss functions. The last column of `v` contains the number of data points in `ze`. The selection of a suitable model structure based on the information in `v` is best done using `selstruc`. See Section 8 in the *Tutorial* for advice on model structure selection, cross-validation, and the like.

See `auxvar` for an explanation of `maxsize`.

Examples

Compare first to fifth order models with one delay using cross-validation on the second half of the data set, and then select the order that gives the best fit to the validation data set:

```
NN = struc(1:5,1:5,1);
V = arxstruc(z(1:200,:),z(201:400,:),NN);
nn = selstruc(V,0);
th = arx(z,nn);
```

See Also

`arx, ivstruc, selstruc, struc`

Purpose

Construct theta format matrix from ARX polynomials.

Syntax

```
th = arx2th(A,B,ny,nu)
th = arx2th(A,B,ny,nu,lam,T)
```

Description

`arx2th` creates a matrix containing parameters that describe the general multi-input, multi-output model structure of ARX type:

$$y(t) + A_1 y(t-1) + A_2 y(t-2) + \ldots + A_{na} y(t-na) =$$

$$B_0 u(t) + B_1 u(t-1) + \ldots + B_{nb} u(t-nb) + e(t)$$

Here A_k and B_k are matrices of dimensions *ny* by *ny* and *ny* by *nu*, respectively (*ny* is the number of outputs, i.e., the dimension of the vector $y(t)$ and *nu* is the number of inputs). "Defining Model Structures" on page 3-29.

The arguments `A` and `B` are matrices that contain the `A` matrices and the `B` matrices of the model:

```
A = [I A1 A2 ... Ana]
B = [B0 B1 ... Bnb]
```

Note that `A` always starts with the identity matrix, and that delays in the model are defined by setting the corresponding leading entries in `B` to zero. For a multivariate time series take `B = []`.

The arguments `ny` and `nu` denote the number of outputs and inputs, respectively.

The optional argument `lam` sets the covariance matrix of the driving noise source $e(t)$ in the model above. The default value is the identity matrix.

The optional argument `T` defines the sampling interval (Default 1).

`th` is returned as a model structure in the theta format. See `theta`.

The use of `arx2th` is twofold. You can use it to create models that are simulated (using `idsim`) or analyzed (using `th2ff`, `th2zp`, etc.). You can also use it to define initial value models that are further adjusted to data (using `pem`). The free parameters in the structure are consistent with the structure of `A` and `B`, i.e., leading zeros in the rows of `B` are regarded as fixed delays, and trailing zeros in `A` and `B` are regarded as a definition of lower order polynomials. These

zeros are fixed, while all other parameters are free. The nominal values of these free parameters are set equal to the values in A and B. The free parameters can be changed by thinit and the structure can be manipulated by fixpar and unfixpar.

For a model with one output, arx2th is compatible with poly2th. The internal representation is however different, and only a model structure that has been defined by arx2th can be manipulated by fixpar and unfixpar.

Examples

Simulate a second order ARX model with one input and two outputs, and then estimate a model using the simulated data :

```
A1 = [-1.5 0.1;-0.2 1.5];
A2 = [0.7 -0.3;0.1 0.7];
B1 = [1;-1];
B2 = [0.5;1.2];
th0 = arx2th([eye(2) A1 A2],[[0;0],B1 B2],2,1);
u = idinput(300);
e = randn(300,2);
y = idsim([u e],th0);
th = arx([y u],[[2 2;2 2],[2;2],[1;1]]);
```

See Also

arx, fixpar, poly2th, th2arx, unfixpar

Purpose

Describe auxiliary variables `iter_info`, `lim`, `maxiter`, `maxsize`, `tol`, and `T`.

Syntax

```
help auxvar
```

Description

Most of the functions have an optional argument `maxsize` that allows a trade-off between memory usage and speed. Several of the functions allow the sampling interval `T` to be specified. The iterative search procedures in `armax`, `bj`, `oe`, and `pem` are controlled by the three parameters `lim`, `maxiter`, and `lim`.

`maxsize`: No matrix formed by the function is allowed to contain more than `maxsize` elements. Instead, the algorithms split the calculations into for-loops, which are slower. The default value of `maxsize` is set in the M-file `idmsize`. On small machines, it is `maxsize=4096`. The main use of `maxsize` is to limit variable sizes when the algorithms run out of memory. See "Some Special Topics" on page 3-68 for more information.

`T`: Specifying the sampling interval `T` gives correct frequency scales on frequency function plots, and correct time scales when transforming to continuous time using `thd2thc`. The default value is `T=1`.

`maxiter`: This variable determines the maximum number of iterations performed during a search for a minimum. The default value is `maxiter=10`. `maxiter=0` returns the results of the special startup procedure.

`tol`: The iterations are continued until the norm of the Gauss-Newton update vector is less than `TOL`. The iterations also terminate when the algorithm fails to find a lower value of the criterion and when the maximum number of iterations are reached. The default value is `tol=0.01`.

`lim`: This variable determines how the criterion is modified from quadratic to one that gives linear weight to large errors. See "Parametric Model Estimation" on page 3-22 for a more precise definition. The default value of `lim` is 1.6. `lim=0` disables the robustification and leads to a purely quadratic criterion.

Default values of these parameters are obtained either by omitting trailing arguments or by entering them as the empty matrix [ ].

`iter_info`: This output argument from the iterative numerical search algorithms `armax`, `bj`, `oe`, and `pem`, supplies information about the iterations. It is a row vector

```
iter_info = [last_iteration#, last_fit_improvement,
norm_of_last_search_vector]
```

containing the indicated information. If the norm of the last search vector is larger than `tol`, and the number of the last iteration is less than `maxiter`, then the iterations were stopped since no smaller value of the criterion could be found along the search direction.

See Also

`armax`, `bj`, `oe`, `pem`

Purpose

Estimate the parameters of a Box-Jenkins model.

Syntax

```
th = bj(z,nn)
th = bj(z,nn,'trace')
[th, iter_info]= bj(z,nn,maxiter,tol,lim,maxsize,T,'trace')
```

Description

The parameters of the Box-Jenkins model structure

$$y(t) = \frac{B(q)}{F(q)}u(t-nk) + \frac{C(q)}{D(q)}e(t)$$

are estimated using a prediction error method.

Matrix z contains the output-input data `z = [y u]` where `y` and `u` are column vectors. In the multi-input case, `u` is a matrix containing the different inputs as columns. `nn` can be given either as

```
nn = [nb nc nd nf nk]
```

or as

```
nn = thi
```

In the former case, `nb`, `nc`, `nd`, and `nf` are the orders of the Box-Jenkins model and `nk` is the delay. In the latter case, `thi` is an initial value, given in theta format. See"The System Identification Problem" on page 3-8 for exact definitions of the model orders.

`th` is returned with the resulting parameter estimates and estimated covariances, stored in theta format.

The optional variables `iter_info`, `lim`, `maxiter`, `maxsize`, `tol`, and `T` are explained under `auxvar`.

For multi-input systems, `nb`, `nf`, and `nk` are row vectors with as many entries as there are input channels. Entry number i then described the orders and delays associated with the `i`-th input.

`bj` does not support multi-output models. Use state-space model for this case (see `canstart`, `n4sid`, and `pem`)

If a last argument 'trace' is supplied, information about the progress of the iterative search for the model will be furnished to the MATLAB command window.

Examples

Here is an example that generates data and stores the results of the startup procedure separately:

```
B = [0 1 0.5];
C = [1 -1 0.2];
D = [1 1.5 0.7];
F = [1 -1.5 0.7];
th0 = poly2th(1,B,C,D,F,0.1);
e = randn(200,1);
u = idinput(200);
y = idsim([u e],th0);
z = [y u];
thi = bj(z,[2 2 2 2 1],0);
th = bj(z,thi);
present(th)
```

Algorithm

bj uses essentially the same algorithm as armax with modifications to the computation of prediction errors and gradients.

See Also

armax, auxvar, oe, pem, theta

Purpose

Plot frequency functions in Bode diagram form.

Syntax

```
bodeplot(g)
bodeplot([g1 g2 ... gn])
bodeplot(g,sd,C,mode)
```

Description

`g` contains the frequency data to be graphed. See `freqfunc` for the format. The frequency functions do not have to be specified at the same frequencies, but have to have the same number of values.

If the frequency functions are generated by `th2ff` or `spa`, and `sd` is specified as a number larger than zero, confidence intervals for the functions are added to the graph as dash-dotted curves (of the same color as the estimate curve). They indicate the confidence regions corresponding to `sd` standard deviations.

On amplitude plots, the logarithm of the absolute value, plus and minus the standard deviation you indicate, is graphed. The latter value can sometimes be negative, which results in an error message from the plotting routine. The resulting plot is still meaningful, however.

By default, amplitude and phase plots are shown simultaneously for each input (noise spectrum) present in `g`. For spectra, phase plots are omitted. Pressing the **Return** key advances the plot from one input to the next.

To show amplitude plots only, use `C = 'A'`. For phase plots only, use `C = 'P'`. The default is `C = 'B'` for both plots.

To obtain all plots on the same diagram use `mode = 'same'`.

Note that if `g` contains information about several outputs, these plots are always given separately.

See Also

`etfe`, `ffplot`, `freqfunc`, `getff`, `nyqplot`, `spa`, `th2ff`

canform

Purpose

Define multivariable state-space canonical form model structures.

Syntax

```
ms = canform(orders,nu)
ms = canform(orders,nu,dkx)
```

Description

`canform` is, like `modstruc`, a function that defines model parameterizations in state-space form, which are used in `ms2th` to create model structures in the theta format. The only use of the resulting matrix `ms` is as an input to `ms2th`.

The model considered is in state-space form:

$$\dot{x}(t) = A(\theta)x(t) + B(\theta)u(t) + K(\theta)e(t)$$

$$y(t) = C(\theta)x(t) + D(\theta)u(t) + e(t)$$

The function applies both to the continuous- and discrete-time cases; which one is determined only when the structure is formed with `ms2th`.

`orders`: The (pseudo-observability) indices `orders` define which matrix elements are fixed (to zero or one) and which are left free as parameters. `orders` is a row vector with as many entries as there are outputs. Element `k` of `orders` describes how many delayed values of the output are required to appropriately predict the `k`-th component of the output. The sum of the order indices is the order of the system (the dimension of x):

```
n = sum(orders)
```

The exact structure is defined in Appendix 4.A of Ljung (1987). Briefly, the A matrix contains $p * n$ parameters, where p is the number of outputs and n is the number of states. The C matrix contains only zeros and ones, while the B matrix is filled with parameters.

`nu`: The number of inputs.

`dkx`: The argument `dkx` determines some additional structure of the matrices of the state-space model to be estimated. It is a row vector with three entries:

```
dkx = [d, k, x]
```

The entries refer to the matrices K, D, and the initial state x(0) of the state-space model given above.

`k = 1` indicates that the K-matrix in the model (the Kalman Gain) will be fully parameterized, while `k = 0` means that this matrix will be fixed to zero. This will give a so-called output error model.

`d = 1` indicates that D-matrix in the model (the direct term from input to output) will be fully parametrized, while `d = 0` means that this matrix will be fixed to zero. This also implies that there will be a delay of (at least) one sample between the input and the output.

`x = 1` indicates that the initial state x(0) will be parameterized, while `x = 0` means that the initial state will be fixed to zero.

Default is

```
dkx = [0, 1, 0]
```

An alternative to `canform` is the function `canstart`. It also provides good initial estimates for the free parameters.

Examples

Write out the state-space matrices for a sixth order system with three outputs and two inputs with `NaN` denoting free parameters:

```
ps = [2 1 3];
ms = canform(ps,2);
th = ms2th(ms,'c',ones(1,18+12+18)*NaN);
[A,B,C,D,K] = th2ss(th)
```

See Also

`canstart`, `fixpar`, `modstruc`, `ms2th`, `unfixpar`

References

Ljung (1987), Appendix 4.A.

canstart

Purpose

Define and initialize state-space canonical form model structures.

Syntax

```
th = canstart(z,orders,nu)
th = canstart(z,orders,nu,dkx)
```

Description

`orders,nu,dkx`: These arguments define the state-space model parameterization exactly as for the function `canform`. In the present case `orders` can also be taken as a scalar, giving the model order (dimension of state vector). Then a default choice of parameterization is made.

The output `th` is a matrix in the theta format. It defines a state-space model parameterization according to the arguments `orders`, `nu` and `dkx`. The values of the parameters in `th` are estimated from the data matrix

```
z = [y u]
```

where `y` is the matrix of output signals, one column for each output, and `u` is the matrix of input signals, again one column for each input.

Choosing the order indices for many systems is not critical in the sense that most n-th order systems can be described by any set of order (pseudo-observability) indices whose sum is *n*. See "Model Structure Selection and Validation" on page 3-49 for more information.

The model `th` could be further refined by using `pem`.

Algorithm

The state-space model is first estimated using `n4sid`, and then transformed to the chosen canonical form using `ss2th`.

Examples

A system with two inputs and two outputs is estimated with a third order model:

```
th = canstart(z,3,2);
th = pem(z,th);
resid(z,th);
```

See Also

`canform`, `pem`

Purpose

Compare measured outputs with model outputs.

Syntax

```
compare(z,th);
[yh,fit] = compare(z,th,k,sampnr,leveladj)
```

Description

`z` is the output-input data in the usual format

```
z = [y u]
```

where `y` is a matrix whose `r`-th column is the `r`-th output signal and correspondingly for the input `u`. `compare` computes the output `yh` that results when the model `th` is simulated with the input `u`. The result is plotted (yellow/solid) together with the corresponding measured output `y` (magenta/dashed). The mean square fit

```
fit = norm(yh - y)/sqrt(length(y))
```

is also computed and displayed. For multi-output systems this is done separately for each output. Pressing the **Return** key advances the plots.

The argument `k` computes the k-step ahead prediction of `y` according to the model `th` instead of the simulated output. In the calculation of $yh(t)$, the model can use outputs up to time $t-k$: $y(s), s = t-k, t-k-1, \ldots$ (and inputs up to the current time t). The default value of `k` is `inf`, which obtains a pure simulation from the input only.

The argument `sampnr` indicates that only the sample numbers in this row vector are plotted and used for the calculation of the fit. The whole data record is used for the simulation/prediction, though. If the optional argument `leveladj` is set to `'yes'`, the simulated/predicted output and the measured output are level adjusted so that they both start at level zero. (This applies also to the calculation of the fit.) This allows for discounting of drift phenomena in the data.

Examples

Split the data record into two parts. Use the first one for estimating a model and the second one to check the model's ability to predict six steps ahead:

```
ze = z(1:250,:);
zv = z(251:500,:);
th = armax(ze,[2 3 1 0]);
compare(zv,th,6);
```

See Also

`idsim`, `predict`

covf

Purpose

Estimate time series covariance functions.

Syntax

```
R = covf(z,M)
R = covf(z,M,maxsize)
```

Description

`z` is an N by nz matrix and `M` is the maximum delay -1 for which the covariance function is estimated.

`R` is returned as an nz^2 x M matrix with entries

$$R(i+(j-1)nz,\ k+1) = \hat{R}_{ij}(k) = \frac{1}{N}\sum_{t=1} z_i(t)z_j(t+k)$$

where z_j is the `j`-th column of `z` and missing values in the sum are replaced by zero.

The optional argument `maxsize` controls the memory size as explained under `auxvar`.

The easiest way to describe and unpack the result is to use

```
reshape(R(:,k+1),nz,nz) = E z(t)*z'(t+k)
```

Here `'` is complex conjugate transpose, which also explains how complex data are handled. The expectation symbol `E` corresponds to the sample means.

Algorithm

When nz is at most two, and when permitted by `maxsize`, a fast Fourier transform technique is applied. Otherwise, straightforward summing is used.

See Also

`spa`

Purpose

Perform correlation analysis and estimate impulse response.

Syntax

```
cra(z);
[ir,R,cl] = cra(z,M,na,plot);
cra(R);
```

Description

The output-input data are given as

```
z = [y u]
```

with `y` as the output column vector and `u` as the input column vector. The routine only handles single-input-single-output data pairs. (For the multivariate case, apply `cra` to two signals at a time.) `cra` prewhitens the input sequence, i.e., filters `u` through a filter chosen so that the result is as uncorrelated (white) as possible. The output `y` is subjected to the same filter, and then the covariance functions of the filtered `y` and `u` are computed and graphed. The cross correlation function between (prewhitened) input and output is also computed and graphed. Positive values of the lag variable then corresponds to an influence from `u` to later values of `y`. In other words, significant correlation for negative lags is an indication of feedback from `y` to `u` in the data.

A properly scaled version of this correlation function is also an estimate of the system's impulse response `ir`. This is also graphed along with 99% confidence levels. The output argument `ir` is this impulse response estimate, so that its first entry corresponds to lag zero. (Negative lags are excluded in `ir`.)

The output argument `R` contains the covariance/correlation information as follows: The first column of `R` contains the lag indices. The second column contains the covariance function of the (possibly filtered) output. The third column contains the covariance function of the (possibly prewhitened) input, and the fourth column contains the correlation function. The plots can be redisplayed by `cra(R)`.

The output argument `cl` is the 99% confidence level for the impulse response estimate.

The optional argument `M` defines the number of lags for which the covariance/correlation functions are computed. These are from –`M` to `M`, so that the length of `R` is 2`M`+1. The impulse response is computed from 0 to `M`. The default value of `M` is 20.

For the prewhitening, the input is fitted to an AR model of order na. The third argument of cra can change this order from its default value na = 10. With na = 0 the covariance and correlation functions of the original data sequences are obtained.

plot: plot = 0 gives no plots. plot = 1 (default) gives a plot of the estimated impulse response together with a 99% confidence region. plot = 2 gives a plot of all the covariance functions.

Examples

Compare a second order ARX model's impulse response with the one obtained by correlation analysis:

```
ir = cra(z);
th = arx(z,[2 2 1]);
imp = [1;zeros(19,1)];
irth = idsim(imp,th);
subplot(211)
plot([ir irth])
title('impulse responses')
subplot(212)
plot([cumsum(ir),cumsum(irth)])
title('step responses')
```

See Also

covf, spa

Purpose

Remove trends from output-input data.

Syntax

```
zd = dtrend(z)
zd = dtrend(z,o,brkp)
```

Description

`z` is a matrix, with data organized in column vectors. `dtrend` removes the trend from each column and returns the result in `zd`.

The default (`o = 0`) removes the zero-th order trends, i.e., the sample means are subtracted.

With `o = 1`, linear trends are removed, after a least-squares fit. With `brkp` not specified, one single line is subtracted from the entire data record. A continuous piecewise linear trend is subtracted if `brkp` contains breakpoints at sample numbers given in a row vector.

Note that `dtrend` differs somewhat from `detrend` in the Signal Processing Toolbox.

Examples

Remove a V-shaped trend from the output with its peak at sample number 119, and remove the sample mean from the input:

```
zd(:,1) = dtrend(z(:,1),1,119);
zd(:,2) = dtrend(z(:,2));
```

etfe

Purpose

Estimate empirical transfer functions and periodograms.

Syntax

```
g = etfe(z)
g = etfe(z,M,N,T)
```

Description

`etfe` estimates the transfer function `g` of the general linear model

$$y(t) = G(q)u(t) + v(t)$$

The matrix `z` contains the output-input data `z = [y u]`, where `y` and `u` are column vectors. The routine works only for single-input, single-output systems.

For a time series, `z = y`. Then `g` is returned as the periodogram of `y`.

`g` is given in frequency function format (see `freqfunc`), with the estimate of $G(e^{i\omega})$ at the frequencies

```
w = [1:N]/N*pi/T
```

The default values of `N` and `T` are 128 and 1, respectively. `N` must be a power of two.

When `M` is specified other than the default value `M = [ ]`, a smoothing operation is performed on the raw spectral estimates. The effect of `M` is then similar to the effect of `M` in `spa`. This can be a useful alternative to `spa` for narrowband spectra and systems, which otherwise require large values of `M`.

When `etfe` is applied to time series, the corresponding spectral estimate is normalized in the way that is defined in "Some Special Topics" on page 3-68. Note that this normalization may differ from the one used by `spectrum` in the Signal Processing Toolbox.

Examples

Compare an empirical transfer function estimate to a smoothed spectral estimate:

```
ge = etfe(z);
gs = spa(z);
bodeplot([ge gs])
```

Algorithm

The empirical transfer function estimate is computed as the ratio of the output Fourier transform to the input Fourier transform, using `fft`. The periodogram

is computed as the normalized absolute square of the Fourier transform of the time series.

The smoothed versions (`M` less than the length of `z`) are obtained by applying a Hamming window to the output FFT times the conjugate of the input FFT, and to the absolute square of the input FFT, respectively, and subsequently forming the ratio of the results. The length of this Hamming window is equal to the number of data points in `z` divided by `M`, plus one.

See Also

`freqfunc`, `spa`

ffplot

Purpose

Plot frequency functions and spectra.

Syntax

```
ffplot(g)
ffplot([g1 g2 ... gn])
ffplot(g,sd,C,mode)
```

Description

This function has exactly the same syntax as `bodeplot`. The only difference is that it gives graphs with linear frequency scales and Hz as the frequency unit.

See Also

`bodeplot`, `freqfunc`, `getff`

Purpose

Fix parameters in structures defined by `ms2th` and `arx2th`.

Syntax

```
thn = fixpar(tho,matrix)
thn = fixpar(tho,matrix,elements,parval)
```

Description

The `fixpar` function produces a new model structure `thn` in the theta format from an old one `tho`, by fixing certain parameters to certain values. The matrix `tho` must be originally defined by `arx`, `arx2th`, `canstart`, `iv4`, or `ms2th` but may have been modified later on by `fixpar`, `pem`, `thinit`, or `unfixpar`.

To modify a state-space structure

$$\dot{x}(t) = A(\theta)x(t) + B(\theta)u(t) + K(\theta)e(t)$$

$$y(t) = C(\theta)x(t) + D(\theta)u(t) + e(t)$$

the argument `matrix` is set equal to one of `'A'`, `'B'`, `'C'`, `'D'`,`'K'`, or `'x0'`. The argument `elements` is a matrix with two columns, in which each row specifies the indices of the element in the chosen system matrix that need to be fixed. For example, to fix the 1,2-element and the 3,5-element of the A matrix use

```
thn = fixpar(tho,'A',[1,2;3,5]);
```

If the argument `elements` is omitted (or entered as the empty matrix) all elements of the indicated matrix will be fixed. In this case `parval` can be given a scalar number, and all the indicated elements will be fixed to that number.

The default is that the elements are fixed to their nominal (initial or currently estimated) value. To fix them to something else use the fourth input argument `parval`. The `r`-th entry of this vector contains the value corresponding to the element defined by the `r`-th row of `elements`.

To modify an ARX model

$$y(t) + A_1y(t-1) + A_2y(t-2) + \ldots + A_{na}y(t-na) =$$

$$B_0u(t) + B_1u(t-1) + \ldots + B_{nb}u(t-nb) + e(t)$$

the argument `matrix` is set equal to one of `'A1'`, `'A2'`, ..., `'B0'`, `'B1'`, ... The argument `elements` is then a matrix with two columns, where each row gives

the indices of the element in the chosen ARX matrix that need to be fixed. The role of `parval` is the same as for the state-space case above.

The routine does not apply to special model structures that you have originally defined using `mf2th`, or to black-box input-output models other than ARX models. Fixing certain parameters can, in those cases, be achieved by using the third argument in `pem` during the estimation phase.

Examples

Converting a state-space structure with all elements in the Kalman gain matrix K free to an output error structure in which K is fixed to zero.

```
thn = fixpar(thn,'K',[],0)
```

Fixing the b_2 parameter to 1 in a scalar ARX model, and then estimating the remaining parameters:

```
th = arx(z,[2 3 0]);
th = fixpar(th,'B2',[1,1],1);
th = pem(z,th);
```

See Also

`theta`, `thinit`, `unfixpar`

Purpose

Describe the frequency-function format.

Syntax

```
help freqfunc
```

Description

Frequency functions are created by `etfe`, `spa`, and `th2ff`, and used by `bodeplot`, `ffplot`, and `nyqplot`. The internal format of the freqfunc format is intended to be transparent. The basic way to display the information is to use the plot commands. You can also retrieve the information from the format by the function `getff`. This entry gives details of the internal representation, but this information is not necessary for normal use of the System Identification Toolbox.

The freqfunc format contains information about frequency values, amplitudes, and phases, as well as their standard deviations. These are all given as columns. The first row of a `freqfunc` matrix contains integers, that, in coded form, describe the significance of the column in question. The interpretation of these integers is as follows:

For transfer functions and spectra associated with output number 1:

`n = 0` : The column is a spectrum.

`n = 50`: The column contains standard deviations of a spectrum.

`n = 100`: The column contains frequencies for the spectrum.

`n = k`, where `k` is value between 1 and 19: The column contains amplitude values for the transfer function associated with input number `k`.

`n = k + 20`: The column contains phase values (in degrees) for input number `k`.

`n = k + 50`: The column contains amplitude standard deviations for input number `k`.

`n = k + 70`: The column contains phase standard deviations for input number `k`.

`n = k + 100`: The column contains the frequency values for input number `k`.

For the same quantities associated with output number ky, add $(ky-1)*1000$ to the numbers above.

The specified frequencies are for a discrete-time model, which is by default, equally spaced from 0 (excluded) to π/T over 128 values. Here T is the sampling interval (default = 1). For a continuous-time model, the frequencies are 128 values, logarithmically spread over three decades, up to a decade over the underlying Nyquist frequency (see `th2ff`).

Examples

You can compute and graph the frequency functions at arbitrary frequency `w` (a row vector with an arbitrary number of elements) with

```
g = spa(z,M,w)
g = th2ff(th,ku,w)
```

The MATLAB function `logspace` is useful for creating such frequency vectors.

See Also

`bodeplot`, `etfe`, `ffplot`, `getff`, `nyqplot`, `sett`, `spa`, `th2ff`

Purpose

Retrieve frequency functions and spectra from the freqfunc format.

Syntax

```
[w,amp,phas] = getff(g)
[w,amp,phas,sdamp,sdphas] = getff(g,ku,ky)
```

Description

This function extracts information from the freqfunc format for plotting as an alternative to `bodeplot`, `ffplot`, and `nyqplot`. Results in the freqfunc format are obtained by `etfe`, `spa`, and `th2ff`.

The argument `g` is the frequency function or spectrum given in the freqfunc format. `ku` is the chosen input number (only one) and `ky` is the chosen output number. The noise source is counted as input number 0, so that `ku = 0` will extract spectral information.

`w` is a vector containing the frequency values (in radians/second). `amp` contains the values of the magnitude (amplitude) of the frequency function and `phas` contains the phase values (in degrees). The output arguments `sdamp` and `sdphas` contain the corresponding standard deviations. All this information is for input number `ku` and output number `ky`. If several entries in `g` correspond to the same input-output pair, then `w`, `amp`, `phas`, `sdamp`, and `sdphas` have the corresponding number of columns. The default values of `ku` and `ky` are both 1, unless `g` contains only spectra. In that case `getff` extracts information about the spectrum corresponding to output number `ky`.

Examples

Make your own plot of the periodogram with linear scales:

```
per = etfe(y);
[w,amp] = getff(per);
plot(w,amp)
title('Periodogram of seismic data')
```

getzp

Purpose

Extract zeros and poles from the zepo format.

Syntax

```
[ze,po] = getzp(zepo)
[ze,po] = getzp(zepo,ku,ky)
```

Description

The basic use of `getzp` is to extract the poles and zeros of the coded format that `th2zp` results in. `zepo` contains this information and is typically the output of `th2zp`. `ku` contains the input number (just one) and `ky` the output number. The noise source number k is here counted as input number $-k$.

`ze` contains the zeros and `po` the poles of the dynamics associated with input number `ku` and output number `ky`. The default values of `ku` and `ky` are both 1.

Note that for the noise dynamics, `zepo` normally just contains information about the zeros and poles from noise source k to output number k, (no cross terms). To extract this information enter

```
[ze,po] = getzp(zepo,0,k)
```

See Also

`th2zp`, `zpform`, `zpplot`

Purpose

Extract information from the theta format.

Syntax

```
mymfile = getmfth(th)
N = getncap(th)
T = gett(th)
```

Description

These functions retrieve some information that is coded into the theta format. `mymfile` is the name of the M-file that you write to define a model structure created by `mf2th`.

`N` is the number of data, from which a certain model `th` is estimated. If the model is not estimated `N` is returned as `[]`.

`T` is the sampling interval of the model `th`. If `T` is negative, the model is a continuous-time one that has been estimated from data with the sampling interval `abs(T)`.

See Also

`mf2th`, `sett`

ident

Purpose

Open the graphical user interface

Syntax

```
ident
ident(session,directory)
```

Description

`ident` by itself opens the main interface window, or brings it forward if it is already open.

`session` is the name of a previous session with the graphical interface, and typically has extension `.sid`. `directory` is the complete path for the location of this file. If the session file is on the `MATLABPATH`, `directory` can be omitted.

When the session is specified, the interface will open with this session active. Typing `ident(session,directory)` on the MATLAB command line, when the interface is active, will load and open the session in question.

For more information about the graphical user interface, see Chapter 2 of this manual.

Examples

```
ident('iddata1.sid')
ident('mydata.sid','\matlab\data\cdplayer\')
```

Purpose

Filter data using Butterworth filters.

Syntax

```
zf = idfilt(z,ord,Wn)
[zf,thf] = idfilt(z,ord,Wn,hs)
```

Description

`idfilt` computes a Butterworth filter of order `ord` and filters all columns of the data matrix

```
z = [y u]
```

through this filter.

If `hs` is not specified and `Wn` contains just one element, a low pass filter with cutoff frequency `Wn` (measured as a fraction of the Nyquist frequency) is obtained. If `hs =' high'` a high pass filter with this cutoff frequency is obtained instead.

If `Wn = [Wnl Wnh]` is a vector with two elements, a filter (of order `2*ord`) with passband between `Wnl` and `Wnh` is obtained is `hs` is not specified. If `hs = 'stop'` a bandstop filter with stop band between these two frequencies is obtained instead.

The output argument `thf` is the filter given in the theta format.

It is common practice in identification to select a frequency band where the fit between model and data is concentrated. Often this corresponds to bandpass filtering with a pass band over the interesting breakpoints in a Bode diagram.

If `ord` is a positive integer, a non-causal, zero-phase filter is used for the filtering. If ord is a negative integer, a causal filter (of order `abs(ord)`) is used instead.

Algorithm

The used filter is the same as `butter` in the Signal Processing Toolbox would give. Also, the zero-phase filter is equivalent to `filtfilt` in that toolbox.

References

Ljung (1987), Chapter 13.

idinput

Purpose

Generate signals, typically to be used as inputs for identification.

Syntax

```
u = idinput(N)
u = idinput(N,type,band,levels)
u = idinput(N,'sine',band,levels,auxvar)
```

Description

`idinput` generates input signals of different kinds, that are typically used for identification purposes. Only scalar inputs are generated.

`N` is the number of data points generated, i.e., the length of `u`.

`type` defines the type of input signal to be generated. This argument takes one of the following values:

- `type = 'rs'` : This gives a random, Gaussian signal.
- `type = 'rbs'`: This gives a random, binary signal.
- `type = 'prbs'`: This gives a pseudo-random, binary signal.
- `type = 'sine'`: This gives a signal which is a sum of sinusoids.

Default is `type = 'rbs'`.

The frequency contents of the signal is determined by the argument `band`. For the choices `type = 'rs'`, `'rbs'`, and `'sine'`, this argument is a row-vector with two entries

```
band = [wlow, whigh]
```

that determine the lower and upper bound of the pass-band. The frequencies `wlow` and `whigh` are expressed in fractions of the Nyquist frequency. A white noise character input is thus obtained for `band = [0 1]`, which also is the default value.

For the choice `type = 'prbs'` we have

```
band = [twologp, M]
```

where the periodicity of the generated PRBS is `2^twologp -1`, and `M` is such that the signal is constant over intervals of length `1/M`. `twologp = 0` gives the maximum length PRBS, corresponding to `twologp = 18`. Also in this case the default is `band = [0 1]`.

The argument `levels` defines the input level. It is a row vector

```
levels = [minu, maxu]
```

such that the signal u will always be between the values `minu` and `maxu` for the choices `type = 'rbs'`, `'prbs'` and `'sine'`. For `type = 'rs'`, the signal level is such that `minu` is the mean value of the signal, minus one standard deviation, while `maxu` is the mean value plus one standard deviation. Gaussian white noise with zero mean and variance one is thus obtained for `levels = [-1, 1]`, which is also the default value.

For the option `type = 'sine'`, there is a fourth argument

```
auxvar = [no_of_sinusoids, no_of_trials]
```

determining the number of sinusoids to be used in the input signal. The variable `no_of_trials` determines how many trials to be made to minimize the signal amplitude by assigning random phases to the different sinusoids. Default is `auxvar = [10, 10]`.

Algorithm

Very simple algorithms are used. The frequency contents is achieved for `'rs'` by an eighth order Butterworth, non-causal filter, using `idfilt`. This is quite reliable. The same filter is used for the `'rbs'` case, before making the signal binary. This means that the frequency contents is not guaranteed to be precise in this case.

For the `'sine'` case, the frequencies are selected to be equally spread over the chosen pass band, and each sinusoid is given a random phase. A number of trials are made, and the phases that give the smallest signal amplitude are selected. (The amplitude is then scaled so as to satisfy the specifications of `levels`.)

See Also

The Frequency Domain System Identification Toolbox contains several commands for input design that utilize more sophisticated algorithms.

Reference

For PRBS, see, e.g., Söderström and Stoica (1989), Chapter C5.3.

idmodred

Purpose

Reduce the order of a model in theta format.

Syntax

```
THRED = idmodred(TH)
THRED = idmodred(TH,ORDER,OE)
```

Description

This function reduces the order of any model `TH` given in the theta format. The resulting reduced order model `THRED` is also in the theta format. This reduced model is always represented internally in state-space form with no free parameters, regardless of the nature of the model `TH`.

The function requires several routines in the Control Systems Toolbox.

`ORDER`: The desired order (dimension of the state-space representation). If `ORDER = []`, which is the default, a plot will show how the diagonal elements of the observability and controllability Gramians decay with the order of the representation. You will then be prompted to select an order based on this plot. The idea is that such a small element will have a negligible influence on the input-output behavior of the model. It is thus suggested that an order is chosen, such that only large elements in these matrices are retained.

`OE`: If the argument `OE` has the value `'oe'`, then an output error model `THRED` is produced, that is, one with the Kalman gain equal to zero (see (3.27) and (3.31) in Chapter 3, "Tutorial"). Otherwise (default), also the noise model is reduced.

The function will recognize whether `TH` is a continuous- or discrete-time model and perform the reduction accordingly. The resulting model `THRED` will be of the same kind in this respect as `TH`.

Algorithm

The functions `(d)balreal` and `(d)modred` from the Control Systems Toolbox are used. The plot, in case `ORDER = []`, shows the vector `g` as returned from `(d)balreal`.

Examples

Build a high order multivariable ARX model, reduce its order to 3 and compare the frequency responses of the original and reduced models:

```
TH = arx([y u],[4*ones(3,3),4*ones(3,2),ones(3,2)]);
THRED = idmodred(TH,3);
bodeplot([trf(TH),trf(THRED)])
```

Use the reduced order model as initial condition for a third order state-space model:

```
THI = ss2th(THRED);
THSS = pem([y u],THI);
```

See Also

ss2th

idplot

Purpose

Plot input-output data.

Syntax

```
idplot(z)
idplot(z,int,T,ny,pc)
```

Description

`z` is the output-input data `z = [y u]` to be graphed. A split plot is obtained with the output on top and the inputs, one by one if several, at the bottom. Pressing the **Return** key advances the plot.

The data points specified in the row vector `int` are graphed. The default value of `int` is all the data. You can use the sampling interval `T` (default `T = 1`) to give correct time axes.

The argument `ny` is the number of outputs in the data matrix. Default is `ny = 1`. The input is piecewise constant between sampling points, and it is graphed accordingly. If you prefer linear interpolation between input data points, use `pc = 'li'`. The default value is `pc = 'pc'`.

Examples

Plot only a portion of the data points:

```
idplot(z,100:200)
```

Purpose

Resample data by interpolation and decimation.

Syntax

```
zr = idresamp(z,R)
[zr, R_act] = idresamp(z,R,filter_order,tol)
```

Description

`z` : The data to be resampled. Each column of `z` contains a signal.

`zr` : The resampled data. The columns of `zr` correspond to those of `z`.

`R` : The resampling factor. The new data record will correspond to a new sampling interval of `R` times the original one. `R > 1` thus corresponds to decimation and `R < 1` corresponds to interpolation. Any positive real number for `R` is allowed, but it will be replaced by a rational approximation (`R_act`).

`R_act` : The actually achieved resampling factor.

`filter_order`: The order of the presampling filters used before interpolation and decimation. Default is 8.

`tol` : The tolerance in the rational approximation of `R`. Default is 0.1.

Algorithm

The resampling factor is first approximated by a rational number by `[num,den] = rat(R,tol)`. The data are then interpolated by a factor `den` and then decimated by a factor `num`. The interpolation and decimation are preceded by prefiltering, and follow the same algorithms as in the routines `interp` and `decimate` in the Signal Processing Toolbox.

Caution

For signals that have much energy around the Nyquist frequency (like piece-wise constant inputs), the resampled waveform may look “very different,” due to the prefiltering effects. The frequency and information contents for identification is, however, not mishandled.

Example

Resample by a factor 1.5 and compare the signals.

```
plot(t,u)
[ur,ra] = idresamp(u,1.5);
plot(t,u,ra*t,ur)
```

idsim

Purpose

Simulate systems specified in theta format.

Syntax

```
y = idsim([u e],th)
[y, ysd] = idsim(u,th)
```

Description

`th` describes an arbitrary model in the theta format. `idsim` returns `y` containing the simulated output, corresponding to the input sequence `u` (one column for each input) and the noise `e`. If `e` is omitted, a noise-free simulation is obtained.

The noise-sequence `e` is scaled by $\sqrt{\lambda}$, where λ is the noise variance (loss function) as specified by `th`. To achieve the correct noise effect, give `e` zero mean and unit variance.

For multi-output systems simulated with noise, `e` should have as many columns as the numbers of outputs. The noise is again scaled using the noise covariance matrix in `th`.

The second output argument is the standard deviation of the simulated output. This option is however not available for state-space models.

If `th` is a continuous-time model, it is first converted to discrete time with sampling interval `abs(T)`. See "Some Special Topics" on page 3-68

Examples

Simulate a given system `th0` (for example created by `poly2th`):

```
e = randn(500,1);
u = idinput(500,'prbs');
y = idsim([u e],th0);
```

Validate a model by comparing a measured output `y` with one simulated using an estimated model `th`:

```
yh = idsim(u,th);
plot([y yh])
```

Algorithm

In case the model is of input-output type, `idsim` uses the MATLAB `filter` function. For state-space models, it uses `ltitr`.

See Also

`idsimsd`, `poly2th`

Purpose

Simulate theta format system with uncertainty.

Syntax

```
idsimsd(u,th)
idsimsd(u,th,N,noise)
```

Description

`u` is a column vector (matrix) containing the input(s). `th` is a model given in the theta format (see `theta`). `N` random models are created, according to the covariance information given in `th`. The responses of each of these models to the input `u` are computed and graphed in the same diagram. If `noise = 'noise'`, noise is added to the simulation, in accordance with the noise model of `th`, and its own uncertainty.

The default values are

```
N = 10
noise = 'nonoise'
```

Examples

Plot the step response of the model `th` and evaluate how it varies in view of the model's uncertainty:

```
step1 = [zeros(5,1); ones(20,1)];
idsimsd(step1,th)
```

See Also

`idsim`, `th2ff`, `th2zp`

ivar

Purpose

Estimate the parameters of an AR model using an approximately optimal choice of instrumental variable procedure.

Syntax

```
th = ivar(y,na)
th = ivar(y,na,nc,maxsize,T)
```

Description

The parameters of an AR model structure

$$A(q)y(t) = v(t)$$

are estimated using the instrumental variable method. `y` is the signal to be modeled, entered as a column vector. `na` is the order of the A polynomial (the number of A parameters). The resulting estimate is returned as `th`, in theta format. The routine is for scalar signals only.

In the above model, $v(t)$ is an arbitrary process, assumed to be a moving average process of order `nc`, possibly time varying. (Default is `nc = na`.) Instruments are chosen as appropriately filtered outputs, delayed `nc` steps.

The optional arguments `maxsize` and `T` are explained under `auxvar`.

Examples

Compare spectra for sinusoids in noise, estimated by the IV method and estimated by the forward-backward least-squares method:

```
y = sin([1:500]'*1.2) + sin([1:500]'*1.5)...
 + 0.2*randn(500,1);
thiv = ivar(y,4);
thls = ar(y,4);
giv = th2ff(thiv);
gls = th2ff(thls);
bodeplot([giv gls])
```

See Also

`ar`, `etfe`, `spa`

References

Stoica, P. et al., *Optimal Instrumental variable estimates of the AR-parameters of an ARMA process*, IEEE Trans. Autom. Control, Vol AC-30, 1985, pp. 1066-1074.

Purpose

Compute fit between simulated and measured output for a group of model structures.

Syntax

```
v = ivstruc(ze,zv,NN)
v = ivstruc(ze,zv,NN,p,maxsize)
```

Description

`NN` is a matrix that defines a number of different structures of the ARX type. Each row of `NN` is of the form

```
nn = [na nb nk]
```

with the same interpretation as described for `arx`. See `struc` for easy generation of typical `NN` matrices for single-input systems.

Each of `ze` and `zv` are matrices containing output-input data `[y u]`. For multi-input systems, `u` has the corresponding number of columns. Models for each model structure defined in `NN` are estimated using the instrumental variable (IV) method on data set `ze`. The estimated models are simulated using the inputs from data set `zv`. The normalized quadratic fit between the simulated output and the measured output in `zv` is formed and returned in `v`. The rows below the first row in `v` are the transpose of `NN`, and the last row contains the logarithms of the condition numbers of the IV matrix

$$\sum \zeta(t)\varphi^{T}(t)$$

A large condition number indicates that the structure is of unnecessarily high order (see page 415 in Ljung (1987)).

The information in `v` is best analyzed using `selstruc`.

If `p` is equal to zero, the computation of condition numbers is suppressed. For the use of `maxsize`, see `auxvar`.

The routine is for single-output systems only.

IMPORTANT: The IV method used does not guarantee that the obtained models are stable. The output-error fit calculated in `v` may then be misleading.

Examples

Compare the effect of different orders and delays, using the same data set for both the estimation and validation:

```
v = ivstruc(z,z,struc(1:3,1:2,2:4));
nn = selstruc(v)
th = iv4(z,nn);
```

Algorithm

A maximum order ARX model is computed using the least-squares method. Instruments are generated by filtering the input(s) through this model. The models are subsequently obtained by operating on submatrices in the corresponding large IV matrix.

See Also

`arxstruc`, `iv4`, `selstruc`, `struc`

Purpose

Estimate the parameters of an ARX model using the instrumental variable (IV) method with arbitrary instruments.

Syntax

```
th = ivx(z,nn,x)
th = ivx(z,nn,x,maxsize,T)
```

Description

`ivx` is a routine analogous to the `iv4` routine, except that you can use arbitrary instruments. These are contained in the matrix `x`. Make this the same size as the output, i.e., the first column(s) of `z`. The instruments used are then analogous to the regression vector itself, except that `y` is replaced by `x`.

Note that `ivx` does not return any estimated covariance matrix for `th`, since that requires additional information.

Use `iv4` as the basic IV routine for ARX model structures. The main interest in `ivx` lies in its use for nonstandard situations; for example when there is feedback present in the data, or when other instruments need to be tried out. Note that there is also an IV version that automatically generates instruments from certain filters you define (type `help iv`).

See Also

`iv4`, `ivar`

References

Ljung (1987), page 198.

iv4

Purpose

Estimate the parameters of an ARX model using an approximately optimal four-stage instrumental variable (IV) procedure.

Syntax

```
th = iv4(z,nn)
th = iv4(z,nn,maxsize,T)
```

Description

This routine is an alternative to arx and the use of the arguments are entirely analogous to the arx function. The main difference is that the procedure is not sensitive to the color of the noise term $e(t)$ in the model equation.

For an interpretation of the loss function (innovations covariance matrix), consult "Some Special Topics" on page 3-68.

Examples

Here is an example of a two-input one-output system with different delays on the inputs u_1 and u_2:

```
z = [y u1 u2];
nb = [2 2];
nk = [0 2];
th = iv4(z,[2 nb nk]);
```

Algorithm

The first stage uses the arx function. The resulting model generates the instruments for a second-stage IV estimate. The residuals obtained from this model are modeled as a high-order AR model. At the fourth stage, the input-output data are filtered through this AR model and then subjected to the IV function with the same instrument-filters as in the second stage.

For the multi-output case, optimal instruments are obtained only if the noise sources at the different outputs have the same color. The estimates obtained with the routine are reasonably accurate though even in other cases.

See Also

arx, oe

References

Ljung (1987), equations (15.21)-(15.26).

Purpose

Package model structures you define into the theta model format.

Syntax

```
th = mf2th(model,cd,parval)
th = mf2th(model,cd,parval,aux,lambda,T)
```

Description

`th` is returned as a model structure in the theta format. `model` is the name of an M-file that defines how the state-space matrices depend on the parameters to be estimated. The format of this M-file is given below. The argument `cd` must be assigned either the value `'c'` which designates that the underlying parameterization refers to a continuous-time model, or the value `'d'`, indicating that the model parameterization is inherently a discrete-time one.

The argument `parval` contains the nominal values of the parameters. This is a row vector of the same length as the number of free parameters in the model structure. The argument `aux` is a matrix of auxiliary variables that the M-file can use for various purposes.

`T` denotes the sampling interval of the data, for which the model is going to be estimated (and the sampling interval that is used when the model is used for simulation and prediction). Give `T` a positive value even if the underlying model is defined to be continuous time.

The model structure corresponds to the general linear state-space structure

$$x(t+T) = A(\theta)x(t) + B(\theta)u(t) + K(\theta)e(t)$$

$$x(0) = x_0(\theta)$$

$$y(t) = C(\theta)x(t) + D(\theta)u(t) + e(t)$$

The matrices in this time-discrete model can be parameterized in an arbitrary way by the vector θ. Write the format for the M-file as follows:

```
[A,B,C,D,K,x0] = mymfile(pars,T,aux)
```

Here the row vector `pars` contains the parameters θ, and the output arguments `A`, `B`, `C`, `D`, `K`, and `x0` are the matrices in the discrete-time model description that correspond to this value of the parameters.

`T` is the sampling interval, and `aux` is any matrix of auxiliary variables with which you want to work. (In that way you can change certain constants and other aspects in the model structure without having to edit the M-file.) Note that the two arguments `T` and `aux` must be included in the function head of the M-file, even if they are not utilized within the M-file.

If the underlying parameterization is a continuous-time one, it is still the discrete-time model matrices, corresponding to the sampling interval `T` that should be delivered by the M-file. However. it is desirable that if the M-file `myfile` is called with a negative value of `T`, it outputs the matrices of the corresponding continuous-time state-space model. If such a feature is included in the M-file, use `cd = 'c'`. This allows for easy transformations between continuous and discrete time using the normal functions `thc2thd` and `thd2thc`.

"Defining Model Structures" on page 3-29 contains several examples of typical M-files that define model structures.

Examples

Use the M-file `'mynoise'` given in Section 6 to obtain a physical parametrization of the Kalman gain:

```
thn = mf2th('mynoise','d',[0.1,-2,1,3,0.2],1)
th = pem(z,thn)
```

Purpose

Select a directory for `idprefs.mat`, a file that stores the graphical user interface's start-up information.

Syntax

```
midprefs
midprefs(path)
```

Description

The graphical user interface `ident` allows a large number of variables for customized choices. These include the window layout, the default choices of plot options, and names and directories of the four most recent sessions with `ident`. This information is stored in the file `idprefs.mat`, which should be placed on the user's `MATLABPATH`. The default, automatic location for this file is in the same directory as the user's `startup.m` file.

`midprefs` is used to select or change the directory where you store `idprefs.mat`. Either type `midprefs`, and follow the instructions, or give the directory name as the argument. Include all directory delimiters as in the PC case

```
midprefs('c:\matlab\toolbox\local\')
```

or in the UNIX case

```
midprefs('/home/ljung/matlab/')
```

Warning

The file `idprefs.mat` contains a variable with the directory name, which also needs to be updated. Therefore, do not just move it using the file system. Always use `midprefs` to change the directory for `idprefs.mat`.

modstruc

Purpose

Define state-space structure with unknown elements.

Syntax

```
ms = modstruc(A,B,C,D,K)
ms = modstruc(A,B,C,D,K,x0)
```

Description

`modstruc` is, like `canform`, a function that defines model parameterizations in state-space form, which are used in `ms2th` to create model structures in the theta format. The only use of the resulting matrix `ms` is as an input to `ms2th`.

The model considered is in state-space form:

$$\dot{x}(t) = A(\theta)x(t) + B(\theta)u(t) + K(\theta)e(t)$$

$$x(0) = x_0(\theta)$$

$$y(t) = C(\theta)x(t) + D(\theta)u(t) + e(t)$$

The function applies both to the continuous and discrete-time cases; which one is determined only when the structure is formed with `ms2th`.

The input arguments `A`, `B`, `C`, `D`, `K`, and `x0` are the matrices of the above state-space model. Numerical values in these matrices indicate fixed (known) entries, while the symbol `NaN` marks an element that is not known and you need to estimate.

The default value of the initial state vector `x0` is the zero vector, but it may also contain parameters you need to estimate.

Examples

Define a continuous-time model structure in diagonal form with the two (real) poles and the numerator unknown:

```
A = [NaN,0;0,NaN];
B = [NaN;NaN];
C = [1,1];
D = 0;
K = [0;0];
ms = modstruc(A,B,C,D,K)
th = ms2th(ms,'c');
```

See Also

`canform`, `fixpar`, `ms2th`, `thinit`, `unfixpar`

Purpose

Package standard state-space parameterizations into the theta model format.

Syntax

```
th = ms2th(ms)
th = ms2th(ms,cd,parval,lambda,T)
```

Description

The function returns `th` as a model structure in the theta format for further use in estimating, simulating, and analyzing models. The argument `ms` defines which parameters are fixed and which ones you need to estimate. It is typically formed by `modstruc` or `canform`.

The argument `cd` indicates whether the state-space matrices in `ms` refer to a continuous-time model (`cd = 'c'`) or a discrete-time model (`cd = 'd'`). `cd = 'd'` is the default.

For a continuous-time model there are two options for how to sample it, as it is fitted to observed sample data (in `pem`): By selecting `cd = 'czoh'` (continuous, zero order hold) the input is assumed to be piecewise constant over the sampling interval. By selecting `cd = 'cfoh'` (continuous, first order hold), the input is supposed to be piecewise linear between the samples. This means that the continuous-time input u(t) is obtained by linear interpolation between the sampled values. Use `cd = 'czoh'` (which is the default for continuous-time models) if the system has been controlled using a piece-wise constant input. Use `cd = 'cfoh'` if the input has been a continuous function during the data acquisition.

The row vector `parval` contains the nominal values of the free parameters (those that correspond to `NaN` in `ms`). These nominal values are used as initial estimates when the parameters in `th` are identified with `pem`. They are also used whenever the model `th` is simulated or analyzed. The default value of `parval` is all zeros.

The numbering of the parameters in `parval` (as well as in all other contexts in which the model parameters are listed) is as follows. The matrix *A* is first scanned, row by row, for free parameters, then the matrix *B* is scanned, again row by row, and then *C*, *D*, *K*, and *X0* each of them row by row. The order in which the free parameters are found by this scanning defines the ordering in `parval`.

Note that setting all initial estimates equal to zero is not a good starting point for the iterative search for parameter estimates in `pem`, since this case often corresponds to a nonobservable/noncontrollable model. It is better to give more realistic values of `parval` or to randomize them using `thinit`.

The argument `lambda` gives the covariance matrix of the innovation $e(t)$ for the sampling interval indicated in `T`. The default value of `lambda` is the unit matrix.

`T` denotes the sampling interval of the data, for which the model is going to be estimated (and the sampling interval that is used when the model is used for simulation and prediction). Give `T` a positive value even if the underlying model is defined to be continuous time.

Examples

Define a continuous-time model structure corresponding to

$$\dot{x} = \begin{bmatrix} \theta_1 & 0 \\ 0 & \theta_2 \end{bmatrix} x + \begin{bmatrix} \theta_3 \\ \theta_4 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 1 \end{bmatrix} x + e$$

with initial values

$$\theta = \begin{bmatrix} -0.2 & -0.3 & 2 & 4 \end{bmatrix}$$

and estimate the free parameters:

```
A = [NaN,0;0,NaN];
B = [NaN;NaN];
C = [1,1];
ms = modstruc(A,B,C,0,[0;0]);
th = ms2th(ms,'c',[-0.2,-0.3,2,4]);
th = pem(z,th);
```

See Also

`canform`, `fixpar`, `modstruc`, `pem`, `thinit`, `unfixpar`

Purpose

Select the step size for numerical differentiation.

Syntax

```
nds = nuderst(pars)
```

Description

The function `pem` uses numerical differentiation with respect to the model parameters when applied to state-space structures. The same is true for the functions `th2ff` and `th2zp` when the parameter covariance information is translated to frequency function and zero-pole accuracy (again only for state-space structures). Finally, the command `thd2thc` uses numerical differentiation when translating the covariance information for any model structure.

The step size used in these numerical derivatives is determined by the M-file `nuderst`. The output argument `nds` is a row vector whose `k`-th entry gives the increment to be used when differentiating with respect the `k`-th element of the parameter vector `pars`.

The default version of `nuderst` uses a very simple method. The step size is the maximum of 10^{-7} and 10^{-4} times the absolute value of the current parameter. You can adjust this to the actual value of the corresponding parameter by editing `nuderst`. Note that the nominal value, for example 0, of a parameter may not reflect its normal size.

nyqplot

Purpose

Plot Nyquist curve of frequency function.

Syntax

```
nyqplot(g)
nyqplot([g1 g2 ... gn])
nyqplot(g,sd,mode)
```

Description

`nyqplot` is an alternative to `bodeplot` and `ffplot` to graph frequency functions in the freqfunq format. The Nyquist diagram of a frequency function is a graph of its imaginary part against its real part.

The argument `g` is the frequency function in question, given in the freqfunc format, typically as the output of `th2ff`, `spa`, or `etfe`. Several plots are obtained in the same diagram by simply listing the different frequency functions after each other. These need not be specified at the same frequencies (although they have to be of the same length).

If the frequency function(s) contains information about several different input-output pairs, the default is that the Nyquist plot for each pair is graphed separately. Pressing the **Return** key advances from one pair to the next. With `mode = 'same'` all plots corresponding to the same input are given in the same diagram.

If `sd` is given a value larger than 0, a confidence region around the nominal Nyquist plot is marked with dash-dotted lines. This region corresponds to `sd` standard deviations. (The information in `g` does not contain the correlation between the magnitude and phase estimates. The confidence region is therefore only approximate. It is computed by plotting the two curves $g \pm sd\Delta g$, where Δg is the standard deviation (magnitude and phase) of `g`.)

Examples

```
g = th2ff(thmod)
nyplot(g,3)
```

See Also

`bodeplot`, `etfe`, `ffplot`, `freqfunc`, `spa`, `th2ff`

Purpose

Estimate state-space models using a subspace method.

Syntax

```
TH = n4sid(z)
[TH,AO] = n4sid(z,order,ny,auxord,dkx,maxsize,T,'trace')
```

Description

The function `n4sid` estimates models in state-space form, and returns them in the theta format. It handles an arbitrary number of input and outputs, including the time series case (no input). The state-space model is in the innovations form:

```
x(t+1) = A x(t) + B u(t) + K e(t)
    y(t) = C x(t) + D u(t) + e(t)
```

`TH`: The resulting model in theta format. No covariance information about the uncertainty of the model is contained in `TH`.

`AO`: See under `auxord` below.

`z`: A matrix that contains the output-input data: `z = [y u]`, where `y` and `u` are column vectors. In the multi-variable case, `u` and `y` contain one column for each output and input. In the time series case `z = y`.

`order`: The desired order of the state-space model. If `order` is entered as a row vector (like `order = [1:10]`, which is the default choice), preliminary calculations for all the indicated orders are carried out. A plot will then be given that shows the relative importance of the dimension of the state vector. More precisely, the singular values of the Hankel matrices of the impulse response for different orders are graphed. You will be prompted to select the order, based on this plot. The idea is to choose an order such that the singular values for higher orders are comparatively small. If `order = 'best'`, a model of "best" (default choice) order is computed, among the orders 1:10.

`ny`: The number of outputs in the data set `z`. Default is `ny = 1`.

`auxord`: An "auxiliary order" used by the algorithm. This can be seen as a prediction horizon, and it should be larger than the order. The default value is `auxord = 1.2*order+3`. The choice of `auxord` could have a substantial influence on the model quality, and there are no simple rules for how to choose it. If you enter `auxord` as a row vector (like `auxord = [5:15]`), models for all these values will be computed. The prediction error for each of the models are computed using the data `z`, and that value of `auxord` that minimizes the fit will be selected. This value is returned as the output argument `AO`. If the last given

argument to n4sid is 'trace', information about the different choices of auxord will be given to the screen. Note that auxord can be chosen as a vector, only if order is a given value (no vector).

dkx: The argument dkx determines some additional structure of the matrices of the state-space model to be estimated. It is a row vector with three entries:

```
dkx = [d, k, x]
```

The entries refer to the matrices K, D, and the initial state X(0) of the state-space model given above.

k = 1 indicates that the K-matrix in the model (the Kalman Gain) will be estimated, while k = 0 means that this matrix will be fixed to zero. This will give a so called output error model.

d = 1 indicates that D-matrix in the model (the direct term from input to output) will be estimated, while d = 0 means that this matrix is fixed to zero. This also implies that there will be a delay of (at least) one sample between the input and the output.

x = 1 indicates that the initial state x(0) will be estimated and stored in the model TH, while x = 0 means that the initial state will be taken as zero. Note that the initial state is something that relates to the particular data set for which the model was estimated, and may not be relevant when the model is evaluated on a new set of data.

Default is

```
dkx = [0, 1, 0]
```

The optional variables maxsize and T are explained under AUXVAR.

trace: Letting the last input argument be 'trace', gives information to the command line about the choice of auxiliary order, in case this is given as a vector.

Algorithm

The function implements the methods described in P. Van Overschee and B. De Moor: N4SID:Subspace algorithms for the identication of combined deteministic-stochastic systems. *Automatica*, Vol. 30, No 1, pp. 75-93, 1994.

The algorithm is complemented with a separate linear least-squares step to re-estimate the matrices B, D, and X(0), which enter linearly.

Examples

Build a fifth order model from data with three inputs and two outputs. Try several choices of auxiliary orders. Look at the frequency response of the model. (Note that there will be no confidence intervals!)

```
z = [y1 y2 u1 u2 u3];
th = n4sid(z,5,2,7:15,'trace');
bodeplot(trf(th))
```

Use the resulting model as initial values for estimating a state-space model of the same order using the prediction error method:

```
thi = ss2th(th);
thp = pem(z,thi);
```

See Also

auxvar, canstart, pem, theta

oe

Purpose

Estimate the parameters of an Output-Error model.

Syntax

```
th = oe(z,nn)
th = oe(z,nn,'trace')
[th, iter_info] = oe(z,nn,maxiter,tol,lim,maxsize,T,'trace')
```

Description

The parameters of the Output-Error model structure

$$y(t) = \frac{B(q)}{F(q)}u(t-nk) + e(t)$$

are estimated using a prediction error method.

Matrix `z` contains the output-input data `z = [y u]` where `y` and `u` are column vectors. `nn` can be given either as

```
nn = [nb nf nk]
```

or as

```
nn = thi
```

In the former case, `nb`, and `nf` are the orders of the Output-Error model and `nk` is the delay. In the latter case, this is an initial value for the estimate, given in theta format. See "The System Identification Problem" on page 3-8 for an exact definition of the orders and delay.

`th` is returned with the resulting parameter estimates and estimated covariances, stored in theta format.

The optional variables `iter_info`, `lim`, `maxiter`, `maxsize`, `tol`, and `T` are explained under `auxvar`.

For multi-input models, `nb`, `nf`, and `nk` are row vectors, such that entry number `i` gives the orders and delays associated with the i-th input.

`oe` does not support multi-output models. Use state-space model for this case (see `canstart`, `n4sid`, and `pem`)

If a last argument `'trace'` is supplied, information about the progress of the iterative search for the model will be furnished to the MATLAB command window.

Algorithm

oe uses essentially the same algorithm as armax with modifications to the computation of prediction errors and gradients.

See Also

armax, auxvar, bj, pem, theta

pe

Purpose

Compute the prediction errors associated with a model and a data set.

Syntax

```
e = pe(z,th)
```

Description

Matrix `z` is the output-input data set, `z = [y u]`, and `th` is a model specified in theta format. `e` is returned containing the prediction errors that result when model `th` is applied to the data,

$$e(t) = H^{-1}(q)[y(t) - G(q)u(t)]$$

See Also

`resid`, `theta`

Purpose

Estimate the parameters of general linear models.

Syntax

```
th = pem(z,nn)
th = pem(z,nn,'trace')
[th, iter_info] = pem(z,nn,index,maxiter,tol,lim,...
                          maxsize,T,'trace')
```

Description

The function pem handles all model structures, including the general multi-input-single-output structure

$$A(q)y(t) = \frac{B_1(q)}{F_1(q)}u_1(t-nk_1) + \ldots + \frac{B_{nu}(q)}{F_{nu}(q)}u_{nu}(t-nk_{nu}) + \frac{C(q)}{D(q)}e(t)$$

and general structures defined by fixpar, mf2th, ms2th, thinit, and unfixpar. Multivariable ARX structures defined by arx2th are also covered.

The matrix z contains the output-input data z = [y u], where y and u are column vectors (in the multi-variable case u and y contain one column for each input and output).

nn is given either as

```
nn = [na nb nc nd nf nk]
```

or as

```
nn = thi
```

In the former case, na, nb, nc, nd, and nf are the orders of the model and nk is the delay(s). For multi-input systems, nb, nf, and nk are row vectors giving the orders and delays of each input. (See Section 3 of the *Tutorial* for exact definitions of the orders).

In the latter case, thi defines a model structure and an initial value for the estimate, given in theta format.

th is returned with the resulting parameter estimates and estimated covariances, stored in theta format.

The optional argument index is a row vector that contains the indices of the parameters that are to be estimated. The others remain fixed to their nominal values. The ordering of the parameters is defined under th2par. The default

value of index is that all free parameters are estimated. The optional variables iter_info, lim, maxiter, maxsize, tol, and T are explained under auxvar.

If a last argument 'trace' is supplied, information about the progress of the iterative search for the model will be furnished to the MATLAB command window.

For the special cases of single-input models of Output-Error, ARMAX, and Box-Jenkins type, it is more efficient to use oe, armax, and bj.

Examples

Here is an example of a system with three inputs and two outputs. A canonical form state-space model of order 5 is sought.

```
z = [y1 y2 u1 u2 u3];
thc = canstart(z,5,3)
th = pem(z,thc);
```

Algorithm

pem uses essentially the same algorithm as armax with modifications to the computation of prediction errors and gradients.

See Also

armax, auxvar, bj, oe, theta

Purpose

Construct theta format matrix for input-output models.

Syntax

```
th = poly2th(A,B)
th = poly2th(A,B,C,D,F,lam,T)
```

Description

`poly2th` creates a matrix containing parameters that describe the general multi-input-single-output model structure:

$$A(q)y(t) = \frac{B_1(q)}{F_1(q)}u_1(t - nk_1) + \frac{B_{nu}(q)}{F_{nu}(q)}u_{nu}(t - nk_{nu}) + \frac{C(q)}{D(q)}e(t)$$

`A`, `B`, `C`, `D`, and `F` specify the polynomial coefficients.

For single-input systems, these are all row vectors in the standard MATLAB format:

```
A = [1 a1 a2 ... ana]
```

`A`, `C`, `D`, and `F` all start with 1, while `B` contains leading zeros to indicate the delays. See “Defining Model Structures” on page 3-29.

For multi-input systems, `B` and `F` are matrices with one row for each input.

For time series, `B` and `F` are entered as empty matrices:

```
B = [];   F = [];
```

`lam` is the variance of the white noise sequence $e(t)$, while `T` is the sampling interval.

A negative value of `T` indicates that the model is a continuous-time one. Then the interpretation of the arguments is that

```
A = [1 2 3 4]
```

corresponds to the polynomial $s^3 + 2s^2 + 3s + 4$ in the Laplace variable s, and so on. For continuous-time systems `lam` indicates the level of the spectral density of the innovations. (A sampled version of the model has the innovations variance `lam/T`, where `T` is the sampling interval. The continuous-time model must have a white noise component in its noise description. See “Some Special Topics” on page 3-68.

Trailing arguments `C`, `D`, `F`, `lam`, and `T` can be omitted, in which case they are taken as 1. (If `B=[]`, then `F` is taken as `[]`.)

For discrete-time models (`T>0`), note the following: `poly2th` strips any trailing zeros from the polynomials when determining the orders. It also strips leading zeros from the `B` polynomial to determine the delays. Keep this in mind when you use `poly2th` and `th2poly` to modify earlier estimates to serve as initial conditions for estimating new structures. See "Some Special Topics" on page 3-68.

Examples

To create a system of ARMAX type (the "Åström system"):

```
A = [1 -1.5 0.7];
B = [0 1 0.5];
C = [1 -1 0.2];
th0 = poly2th(A,B,C);
```

This gives a system with one delay (`nk = 1`).

Create the continuous-time model

$$y(t) = \frac{1}{s(s+1)}u_1(t) + \frac{s+3}{s^2+2s+4}u_2(t) + e(t)$$

Sample it with `T=0.1` and then simulate it without noise:

```
B=[0 1;1 3];
F=[1 1 0;1 2 4]
th=poly2th(1,B,1,1,F,1,-1)
thd=thc2thd(th,0.1)
y=idsim([u1 u2],thd);
```

See Also

`idsim`, `theta`

Purpose

Predict the output *k*-step ahead.

Syntax

```
p = predict(z,th)
[yp,thpred] = predict(z,th,k)
```

Description

`z` is the output-input data in the usual format

```
z = [y u]
```

where `y` is a matrix whose `r`-th column is the `r`-th output signal and correspondingly for the input `u`.

The argument `k` indicates that the *k*-step ahead prediction of `y` according to the model `th` (in the theta format) is computed. In the calculation of *yp(t)* the model can use outputs up to time

$t-k: y(s), s = t-k, t-k-1, \ldots$

and inputs up to the current time *t*. The default value of `k` is 1.

The output `yp` is a matrix of the same size as `y`, and its *i,j* element contains the predicted value of the corresponding element in `y`. The output argument `thpred` contains the *k*-step ahead predictor in the theta format, in the case that `th` corresponds to an input-output model. (The predictor is a system with $ny + nu$ inputs and ny outputs, ny being the number of outputs and nu the number of inputs to `th`.)

An important use of `predict` is to evaluate a model's properties in the mid-frequency range. Simulation with `idsim` (which conceptually corresponds to `k = inf`) can lead to levels that drift apart, since the low frequency behavior is emphasized. One step ahead prediction is not a powerful test of the model's properties, since the high frequency behavior is stressed. The trivial predictor $\hat{y}(t) = y(t-1)$ can give good predictions in case the sampling of the data is fast.

Another important use of `predict` is to evaluate models of time series. The natural way of studying a time series model's ability to reproduce observations is to compare its *k*-step ahead predictions with actual data.

Note that for output-error models, there is no difference between the *k*-step ahead predictions and the simulated output, since, by definition, output-error models only use past inputs to predict future outputs.

Algorithm

For a model `th` that is an input-output model, the formula (3.31) in Ljung (1987) is applied to compute the predictor. For state-space models, the state equations are simulated k-steps ahead with initial value $x(t-k) = \hat{x}(t-k)$, where $\hat{x}(t-k)$ is the Kalman filter state estimate.

Examples

Simulate a time series, estimate a model based on the first half of the data, and evaluate the four step ahead predictions on the second half:

```
th0 = poly2th([1 -0.99],[],[1 -1 0.2]);
y = idsim(randn(400,1),th0);
th = armax(y(1:200),[1 2]);
yp = predict(y,th,4);
plot([y(201:400),yp(201:400)])
```

Note that the last two commands also are achieved by

```
compare(y,th,4,201:400);
```

See Also

`compare`, `idsim`, `pe`

Purpose

Display the information in a theta matrix.

Syntax

```
present(th)
```

Description

This function displays the polynomials of the model `th`, together with their standard deviations, loss function, and Akaike's Final Prediction Error Criterion (FPE) (which essentially equals the AIC). It also displays information about how `th` was created.

Leading zeros in *B* correspond to delays; therefore, the delay is *nk* if *B* starts with *nk* exact zeros.

For input-output models, the estimated standard deviations are given just below the estimated parameters. (Note that leading zeros and ones are exact and have zero standard deviation.)

For state-space models and multivariable ARX models, the standard deviations are given as fake, imaginary parts of the parameter estimates.

See Also

`theta`

rarmax

Purpose

Estimate recursively the parameters of an ARMAX or ARMA model.

Syntax

```
thm = rarmax(z,nn,adm,adg)
[thm,yhat,P,phi,psi] = ...
  rarmax(z,nn,adm,adg,th0,P0,phi0,psi0)
```

Description

The parameters of the ARMAX model structure

$$A(q)y(t) = B(q)u(t-nk) + C(q)e(t)$$

are estimated using a recursive prediction error method.

Matrix `z` contains the output-input data `z = [y u]` where `y` and `u` are column vectors. `nn` is given as

```
nn = [na nb nc nk]
```

where `na`, `nb`, and `nc` are the orders of the ARMAX model, and `nk` is the delay. See equations (3.16)-(3.18) in Chapter 3, "Tutorial" for an exact definition of the orders.

If `z = y` and `nn = [na nc]`, `rarmax` estimates the parameters of an ARMA model for `y`:

$$A(q)y(t) = C(q)e(t)$$

Only single-input, single-output models are handled by `rarmax`. Use `rpem` for the multi-input case.

The estimated parameters are returned in the matrix `thm`. The `k`-th row of `thm` contains the parameters associated with time `k`, i.e., they are based on the data in the rows up to and including row `k` in `z`. Each row of `thm` contains the estimated parameters in the following order:

```
thm(k,:) = [a1,a2,...,ana,b1,...,bnb,c1,...,cnc]
```

See equations (3.16),(3.18) in Chapter 3, "Tutorial" for more information.

`yhat` is the predicted value of the output, according to the current model, i.e., row *k* of `yhat` contains the predicted value of `z(k,1)` based on all past data.

The actual algorithm is selected with the two arguments `adm` and `adg`. These are described under `rarx`.

The input argument `th0` contains the initial value of the parameters, a row vector, consistent with the rows of `thm`. The default value of `th0` is all zeros.

The arguments `P0` and `P` are the initial and final values, respectively, of the scaled covariance matrix of the parameters. See `rarx`. The default value of `P0` is 10^4 times the unit matrix. The arguments `phi0`, `psi0`, `phi`, and `psi` contain initial and final values of the data vector and the gradient vector, respectively. The sizes of these depend in a rather complicated way on the chosen model orders. The normal choice of `phi0` and `psi0` is to use the outputs from a previous call to `rarmax` with the same model orders. (This call could of course be a *dummy* call with default input arguments.) The default values of `phi0` and `psi0` are all zeros.

Note that the function requires that the delay `nk` be larger than 0. If you want `nk=0`, shift the input sequence appropriately and use `nk=1`.

Algorithm

The general recursive prediction error algorithm (11.44), (11.47)-(11.49) of Ljung (1987) is implemented. See “Recursive Parameter Estimation” on page 3-61 for more information.

Examples

Compute and plot, as functions of time, the four parameters in a second order ARMA model of a time series given in the vector `y`. The forgetting factor algorithm with a forgetting factor of 0.98 is applied.

```
thm = rarmax(y,[2 2],'ff',0.98);
plot(thm)
```

rarx

Purpose

Estimate recursively the parameters of an ARX or AR model.

Syntax

```
thm = rarx(z,nn,adm,adg)
[thm,yhat,P,phi] = rarx(z,nn,adm,adg,th0,P0,phi0)
```

Description

The parameters of the ARX model structure

$$A(q)y(t) = B(q)u(t-nk) + e(t)$$

are estimated using different variants of the recursive least-squares method.

Matrix `z` contains the output-input data `z = [y u]` where `y` and `u` are column vectors. `nn` is given as

```
nn = [na nb nk]
```

where `na` and `nb` are the orders of the ARX model, and `nk` is the delay. See equation (3.16) in the *Tutorial* for an exact definition of the orders.

If `z = y` and `nn = na`, `rarx` estimates the parameters of an AR model for `y`:

$$A(q)y(t) = e(t)$$

Models with several inputs

$$A(q)y(t) = B_1(q)u_1(t-nk_1) + \ldots B_{nu}u_{nu}(t-nk_{nu}) + e(t)$$

are handled by allowing `u` to contain each input as a column vector,

```
u = [u1 ... unu]
```

and by allowing `nb` and `nk` to be row vectors defining the orders and delays associated with each input.

Only single-output models are handled by `rarx`.

The estimated parameters are returned in the matrix `thm`. The `k`-th row of `thm` contains the parameters associated with time `k`, i.e., they are based on the data

in the rows up to and including row k in z. Each row of thm contains the estimated parameters in the following order:

```
thm(k,:) = [a1,a2,...,ana,b1,...,bnb]
```

See equation (3.16) in Chapter 3, "Tutorial". In the case of a multi-input model, all the *b* parameters associated with input number 1 are given first, and then all the *b* parameters associated with input number 2, and so on.

yhat is the predicted value of the output, according to the current model, i.e., row k of yhat contains the predicted value of z(k,1) based on all past data.

The actual algorithm is selected with the two arguments adg and adm. These are described in “Recursive Parameter Estimation” on page 3-61. The options are as follows:

With adm ='ff' and adg = lam the *forgetting factor* algorithm (10.6abd)+(10.8) is obtained with forgetting factor λ = lam. This is what is often referred to as Recursive Least Squares, RLS. In this case the matrix P (see below) has the following interpretation: $R_2/2 * P$ is approximately equal to the covariance matrix of the estimated parameters. Here R_2 is the variance of the innovations (the true prediction errors *e(t)* in (10.3)).

With adm ='ug' and adg = gam the *unnormalized gradient* algorithm (10.6abc)+(10.9) is obtained with gain *gamma*= gam. This algorithm is commonly known as unnormalized Least Mean Squares, LMS. Similarly adm ='ng' and adg = gam gives the *normalized gradient* or Normalized Least Mean Squares, NLMS algorithm (10.6abc) + (10.10). In these cases P is not defined or applicable.

With adm ='kf' and adg = R1 the *Kalman Filter Based* algorithm (10.6) is obtained with R_2= 1 and R_1 = R1. If the variance of the innovations *e(t)* is not unity but R_2, then $R_2 * P$ is the covariance matrix of the parameter estimates, while R_1 =R1 / R_2 is the covariance matrix of the parameter changes in (10.4).

The input argument th0 contains the initial value of the parameters; a row vector, consistent with the rows of thm. (See above.) The default value of th0 is all zeros.

The arguments P0 and P are the initial and final values, respectively, of the scaled covariance matrix of the parameters. The default value of P0 is 10^4 times the identity matrix.

The arguments `phi0` and `phi` contain initial and final values, respectively, of the data vector:

$$\varphi(t) = [y(t-1), \ldots, y(t-na), u(t-1), \ldots u(t-nb-nk+1)]$$

Then, if

```
z = [y(1),u(1); ... ;y(N),u(N)]
```

you have `phi0` $=\varphi(1)$ and `phi` $=\varphi(N)$. The default value of `phi0` is all zeros. For online use of `rarx`, use `phi0`, `th0`, and `P0` as the previous outputs `phi`, `thm` (last row), and `P`.

Note that the function requires that the delay `nk` be larger than 0. If you want `nk=0`, shift the input sequence appropriately and use `nk=1`.

Examples

Adaptive noise canceling: The signal y contains a component that has its origin in a known signal r. Remove this component by estimating, recursively, the system that relates r to y using a sixth order FIR model together with the NLMS algorithm:

```
z = [y r];
[thm,noise] = rarx(z,[0 6 1],'ng',0.1);
%noise is the adaptive estimate of the noise
%component of y
plot(y-noise)
```

If the above application is a true online one, so that you want to plot the best estimate of the signal `y - noise` at the same time as the data *y* and *u* become available, proceed as follows:

```
phi = zeros(6,1); P=1000*eye(6);
th = zeros(1,6); axis([0 100 -2 2]);
plot(0,0,'*'), hold
%The loop:
while ~abort
[y,r,abort] = readAD(time);
[th,ns,P,phi] = rarx([y r],'ff',0.98,th,P,phi);
plot(time,y-ns,'*')
time = time +Dt
end
```

This example uses a forgetting factor algorithm with a forgetting factor of 0.98. `readAD` represents an M-file that reads the value of an A/D converter at the indicated time instance.

rbj

Purpose

Estimate recursively the parameters of a Box-Jenkins model.

Syntax

```
thm = rbj(z,nn,adm,adg)
[thm,yhat,P,phi,psi] = ...
  rbj(z,nn,adm,adg,th0,P0,phi0,psi0)
```

Description

The parameters of the ARMAX model structure

$$y(t) = \frac{B(q)}{F(q)}u(t-nk) + \frac{C(q)}{D(q)}e(t)$$

are estimated using a recursive prediction error method.

Matrix `z` contains the output-input data `z = [y u]` where `y` and `u` are column vectors. `nn` is given as

```
nn = [nb nc nd nf nk]
```

where `nb`, `nc`, `nd`, and `nf` are the orders of the Box-Jenkins model, and `nk` is the delay. See equations (3.21) and (3.14),(3.18), (3.22) and (3.29) in the *Tutorial* for an exact definition of the orders.

Only single-input, single-output models are handled by `rbj`. Use `rpem` for the multi-input case.

The estimated parameters are returned in the matrix `thm`. The `k`-th row of `thm` contains the parameters associated with time `k`, i.e., they are based on the data in the rows up to and including row `k` in `z`. Each row of `thm` contains the estimated parameters in the following order:

```
thm(k,:) = [b1,...,bnb,c1,...,cnc,d1,...,dnd,f1,...,fnf]
```

(See equations (3.14),(3.18),(3.22) and (3.20) in the *Tutorial*.)

`yhat` is the predicted value of the output, according to the current model, i.e., row `k` of `yhat` contains the predicted value of `z(k,1)` based on all past data.

The actual algorithm is selected with the two arguments `adm` and `adg`. These are described under `rarx`.

The input argument `th0` contains the initial value of the parameters, a row vector, consistent with the rows of `thm`. (See above.) The default value of `th0` is all zeros.

The arguments `P0` and `P` are the initial and final values, respectively of the scaled covariance matrix of the parameters. See `rarx`. The default value of `P0` is 10^4 times the unit matrix. The arguments `phi0`, `psi0`, `phi`, and `psi` contain initial and final values of the data vector and the gradient vector, respectively. The sizes of these depend in a rather complicated way on the chosen model orders. The normal choice of `phi0` and `psi0` is to use the outputs from a previous call to `rbj` with the same model orders. (This call could, of course, be a dummy call with default input arguments.) The default values of `phi0` and `psi0` are all zeros.

Note that the function requires that the delay `nk` is larger than 0. If you want `nk=0`, shift the input sequence appropriately and use `nk=1`.

Algorithm

The general recursive prediction error algorithm (11.44) of Ljung (1987) is implemented. See also “Recursive Parameter Estimation” on page 3-61.

resid

Purpose

Compute and test the residuals (prediction errors) of a model.

Syntax

```
[e,r] = resid(z,th)
[e,r] = resid(z,th,M,maxsize)
resid(r);
```

Description

Matrix `z` contains the output-input data `z = [y u]`, where `y` and `u` are column vectors. In the multivariable case, `y` and `u` are matrices, with columns corresponding to the different inputs and outputs.

`th` is the model to be evaluated on the given data set, defined in theta format.

`e` is returned with the residuals (prediction errors) associated with the model and the data.

The autocorrelation function of `e` and the cross correlation between `e` and the input(s) `u` are computed and displayed. The 99% confidence intervals for these values are also computed and displayed as dotted (red) curves. The computation of these values is done assuming `e` to be white and independent of `u`. The functions are displayed up to lag `M`, which is 25 by default.

The correlation information is returned with `r`. The plots can then be reviewed by

```
resid(r);
```

See "Model Structure Selection and Validation" on page 3-49 for more information.

The argument `maxsize` is explained under `auxvar`.

Examples

Here are some typical model validation commands:

```
e = resid(z,th);
plot(e)
compare(z,th);
```

See Also

`auxvar`, `compare`, `pem`, `theta`

References

Ljung (1987), Section 16.5.

Purpose

Estimate recursively the parameters of an Output-Error model.

Syntax

```
thm = roe(z,nn,adm,adg)
[thm,yhat,P,phi,psi] = roe(z,nn,adm,adg,th0,P0,phi0,psi0)
```

Description

The parameters of the Output-Error model structure

$$y(t) = \frac{B(q)}{F(q)}u(t-nk)$$

are estimated using a recursive prediction error method.

Matrix `z` contains the output-input data `z = [y u]` where `y` and `u` are column vectors. `nn` is given as

```
nn = [nb nf nk]
```

where `nb` and `nf` are the orders of the Output-Error model, and `nk` is the delay. See equations (3.14) and (3.19)-(3.20) in the *Tutorial* for an exact definition of the orders.

Only single-input, single-output models are handled by `roe`. Use `rpem` for the multi-input case.

The estimated parameters are returned in the matrix `thm`. The `k`-th row of `thm` contains the parameters associated with time `k`, i.e., they are based on the data in the rows up to and including row `k` in `z`.

Each row of `thm` contains the estimated parameters in the following order:

```
thm(k,:) = [b1,...,bnb,f1,...,fnf]
```

See equations (3.14), (3.20) in the *Tutorial*.

`yhat` is the predicted value of the output, according to the current model, i.e., row `k` of `yhat` contains the predicted value of `z(k,1)` based on all past data.

The actual algorithm is selected with the two arguments `adg` and `adm`. These are described under `rarx`.

The input argument `th0` contains the initial value of the parameters, a row vector, consistent with the rows of `thm`. (See above.) The default value of `th0` is all zeros.

The arguments P0 and P are the initial and final values, respectively, of the scaled covariance matrix of the parameters. See rarx. The default value of P0 is 10^4 times the unit matrix. The arguments phi0, psi0, phi, and psi contain initial and final values of the data vector and the gradient vector, respectively. The sizes of these depend in a rather complicated way on the chosen model orders. The normal choice of phi0 and psi0 is to use the outputs from a previous call to roe with the same model orders. (This call could be a dummy call with default input arguments.) The default values of phi0 and psi0 are all zeros.

Note that the function requires that the delay nk is larger than 0. If you want nk=0, shift the input sequence appropriately and use nk=1.

Algorithm

The general recursive prediction error algorithm (11.44) of Ljung (1987) is implemented. See also “Recursive Parameter Estimation” on page 3-61.

See Also

oe, rarx, rbj, rplr,

Purpose

Estimate recursively the parameters of a general multi-input single-output linear model.

Syntax

```
thm = rpem(z,nn,adm,adg)
[thm,yhat,P,phi,psi] = rpem(z,nn,adm,adg,th0,P0,phi0,psi0)
```

Description

The parameters of the general linear model structure

$$A(q)y(t) = \frac{B_1(q)}{F_1(q)}u_1(t-nk_1) + \ldots + \frac{B_{nu}(q)}{F_{nu}(q)}u_{nu}(t-nk_{nu}) + \frac{C(q)}{D(q)}e(t)$$

are estimated using a recursive prediction error method.

Matrix `z` contains the output-input data `z = [y u]` where `y` and `u` are column vectors (in the multi-input case `u` contains one column for each input). `nn` is given as

```
nn = [na nb nc nd nf nk]
```

where `na`, `nb`, `nc`, `nd`, and `nf` are the orders of the model, and `nk` is the delay. For multi-input systems `nb`, `nf`, and `nk` are row vectors giving the orders and delays of each input. See equations (3.13)-(3.23) in the *Tutorial* for an exact definition of the orders.

The estimated parameters are returned in the matrix `thm`. The `k`-th row of `thm` contains the parameters associated with time `k`, i.e., they are based on the data in the rows up to and including row `k` in `z`. Each row of `thm` contains the estimated parameters in the following order:

```
thm(k,:) = [a1,a2,...,ana,b1,...,bnb,...
  c1,...,cnc,d1,...,dnd,f1,...,fnf]
```

See equations (3.13)-(3.23) in the *Tutorial*. For multi-input systems the *B* part in the above expression is repeated for each input before the *C* part begins, and also the *F* part is repeated for each input. This is the same ordering as in `th2par`.

`yhat` is the predicted value of the output, according to the current model, i.e., row *k* of `yhat` contains the predicted value of `z(k,1)` based on all past data.

The actual algorithm is selected with the two arguments adg and adm. These are described under rarx

The input argument th0 contains the initial value of the parameters;, a row vector, consistent with the rows of thm. (See above.) The default value of th0 is all zeros.

The arguments P0 and P are the initial and final values, respectively, of the scaled covariance matrix of the parameters. See rarx. The default value of P0 is 10^4 times the unit matrix. The arguments phi0, psi0, phi, and psi contain initial and final values of the data vector and the gradient vector, respectively. The sizes of these depend in a rather complicated way on the chosen model orders. The normal choice of phi0 and psi0 is to use the outputs from a previous call to rpem with the same model orders. (This call could be a dummy call with default input arguments.) The default values of phi0 and psi0 are all zeros.

Note that the function requires that the delay nk is larger than 0. If you want nk=0 , shift the input sequence appropriately and use nk=1.

Algorithm

The general recursive prediction error algorithm (11.44) of Ljung (1987) is implemented. See also “Recursive Parameter Estimation” on page 3-61.

For the special cases of ARX/AR models, and of single-input ARMAX/ARMA, Box-Jenkins, and Output-Error models, it is more efficient to use rarx, rarmax, rbj, and roe.

See Also

pem, rarmax, rarx, rbj, roe, rplr

Purpose

Estimate recursively the parameters of a general multi-input single-output linear model.

Syntax

```
thm = rplr(z,nn,adm,adg)
[thm,yhat,P,phi] = rplr(z,nn,adm,adg,th0,P0,phi0)
```

Description

This is a direct alternative to `rpem` and has essentially the same syntax. See `rpem` for an explanation of the input and output arguments.

`rplr` differs from `rpem` in that it uses another gradient approximation. See Section 11.5 in Ljung (1987). Several of the special cases are well known algorithms.

When applied to ARMAX models, (`nn = [na nb nc 0 0 nk]`), `rplr` gives the Extended Least Squares, ELS method. When applied to the output error structure (`nn = [0 nb 0 0 nf nk]`) the method is known as HARF in the `adm = 'ff'` case and SHARF in the `adm = 'ng'` case.

`rplr` can also be applied to the time series case in which an ARMA model is estimated with

```
z = y
nn = [na nc]
```

You can thus use `rplr` as an alternative to `rarmax` for this case.

See Also

`pem`, `rarmax`, `rarx`, `rbj`, `roe`, `rpem`

segment

Purpose

Segment data and estimate models for each segment.

Syntax

```
segm = segment(z,nn)
[segm,V,thm,R2e] = segment(z,nn,R2,q,R1,M,th0,P0,ll,mu)
```

Description

`segment` builds models of AR, ARX, or ARMAX/ARMA,

$$A(q)y(t) = B(q)u(t-nk) + C(q)e(t)$$

assuming that the model parameters are piece-wise constant over time. It results in a model that has split the data record into segments over which the model remains constant. The function models signals and systems that may undergo abrupt changes.

The argument `z` is the output-input data

```
z = [y u]
```

where `y` is a column vector containing the outputs and `u` is a column vector containing the inputs. If the system has several inputs, `u` has the corresponding number of columns.

The argument `nn` defines the model order. For the ARMAX model

```
nn = [na nb nc nk]
```

where `na`, `nb`, and `nc` are the orders of the corresponding polynomials. See (3.13)-(3.18) in Chapter 3, "Tutorial". Moreover `nk` is the delay. If the model has several inputs, `nb` and `nk` are row vectors, giving the orders and delays for each input.

For an ARX model (`nc = 0`) enter

```
nn = [na nb nk]
```

For an ARMA model of a time series

```
z = y
nn = [na nc]
```

and for an AR model

```
nn = na
```

The output argument segm is a matrix, whose k-row contains the parameters corresponding to time k. This is analogous to the output argument thm in rarx and rarmax. The output argument thm of segment contains the corresponding model parameters that have not yet been segmented. That is, they are not piecewise constant, and therefore correspond to the outputs of the functions rarmax and rarx. In fact, segment is an alternative to these two algorithms, and has a better capability to deal with time variations that may be abrupt.

The output argument V contains the sum of the squared prediction errors of the segmented model. It is a measure of how successful the segmentation has been.

The input argument R2 is the assumed variance of the innovations *e(t)* in the model. The default value of R2 is that it is estimated. Then the output argument R2e is a vector whose k-th element contains the estimate of R2 at time k.

The argument q is the probability that the model undergoes at an abrupt change at any given time. The default value is 0.01.

R1 is the assumed covariance matrix of the parameter jumps when they occur. The default value is the identity matrix with dimension equal to the number of estimated parameters.

M is the number of parallel models used in the algorithm (see below). Its default value is 5.

th0 is the initial value of the parameters. Its default is zero. P0 is the initial covariance matrix of the parameters. The default is 10 times the identity matrix.

ll is the guaranteed life of each of the models. That is, any created candidate model is not abolished until after at least ll time steps. The default is ll = 1. Mu is a forgetting parameter that is used in the scheme that estimates R2. The default is 0.97.

The most critical parameter for you to choose is R2. It is usually more robust to have a reasonable guess of R2 than to estimate it. Typically, you need to try different values of R2 and evaluate the results. (See the example below.) sqrt(R2) corresponds to a change in the value *y(t)* that is normal, giving no indication that the system or the input might have changed.

Algorithm

The algorithm is based on `M` parallel models, each recursively estimated by an algorithm of Kalman filter type. Each is updated independently, and its posterior probability is computed. The time varying estimate `thm` is formed by weighting together the `M` different models with weights equal to their posterior probability. At each time step the model (among those that have lived at least `ll` samples) that has the lowest posterior probability is abolished. A new model is started, assuming that the system parameters have jumped, with probability `q`, a random jump from the most likely among the models. The covariance matrix of the parameter change is set to `R1`.

After all the data are examined, the surviving model with the highest posterior probability is tracked back and the time instances where it jumped are marked. This defines the different segments of the data. (If no models had been abolished in the algorithm, this would have been the maximum likelihood estimates of the jump instances.) The segmented model `segm` is then formed by smoothing the parameter estimate, assuming that the jump instances are correct. In other words, the last estimate over a segment is chosen to represent the whole segment.

Examples

Check how the algorithm segments a sinusoid into segments of constant levels. Then use a very simple model $y(t) = b_1 * 1$, where 1 is a faked input and b_1 describes the piecewise constant level of the signal $y(t)$ (which is simulated as a sinusoid).

```
y = sin([1:50]/3)';
thm = segment([y,ones(y)],[0 1 1],0.1);
plot([thm,y])
```

By trying various values of `R2` (0.1 in the above example), more levels are created as `R2` decreases.

Purpose

Select model order (structure).

Syntax

```
[nn,vmod] = selstruc(v)
[nn,vmod] = selstruc(v,c)
```

Description

`selstruc` is a function to help choose a model structure (order) from the information contained in the matrix `v` obtained as the output from `arxstruc` or `ivstruc`.

The default value of `c` is `'plot'`. This graphs the values of the loss functions in `v` against the total number of parameters in the corresponding model structure. If `v` was generated by `ivstruc`, so that it also contains the condition numbers of the IV matrices, then these are also graphed against the number of parameters. Based on inspection of these plots, you can choose a suitable number of parameters, and `nn` returns the best structure with this number of parameters.

If `c = 'log'`, the logarithms of the loss functions are graphed instead.

`c = 'aic'` gives no plots, but returns in `nn` the structure that minimizes Akaike's Information Theoretic Criterion (AIC),

$$V_{mod} = V*(1 + 2*(d/N))$$

where V is the loss function, d is the total number of parameters in the structure in question, and N is the number of data points used for the estimation.

`c = 'mdl'` returns in `nn` the structure that minimizes Rissanen's Minimum Description Length (MDL) criterion.

$$V_{mod} = V*(1 + \log(N)*d/N)$$

When `c` equals a numerical value, the structure that minimizes

$$V_{mod} = V*(1 + c*d/n)$$

is selected.

The output argument `vmod` has the same format as `v`, but it contains the logarithms of the accordingly modified criteria.

The formal grounds for applying AIC or MDL require that v be generated by arxstruc with ze = zv. When cross-validation is used (ze different from zv), the logical choice is c = 0.

When selecting model structures, it is important to assess carefully whether the benefit of a better fit is worth the additional complexity of a higher order model. See Chapter 3, "Tutorial" for more information.

Examples

Here is a typical sequence of commands for finding a suitable model structure:

```
NN = struc(2,2,1:8);
v = arxstruc(z(1:200,:),z(201:400,:),NN);
nn = selstruc(v,0);
nk = nn(3);
NN = struc(1:4,1:4,nk);
va = arxstruc(z(1:200,:),z(201:400,:),NN);
vi = ivstruc(z(1:200,:),z(201:400,:),NN);
nna = selstruc(va);
nni = selstruc(vi);
```

See Also

arxstruc, ivstruc, struc

Purpose

Set the sampling interval directly.

Syntax

```
modn = sett(mod,T)
```

Description

All functions that create model descriptions in the theta or freqfunc format set the sampling interval `T` in their last argument. For convenience `sett` offers an alternative to set it directly. It can be applied both to the case where `mod` is a model in the theta format and to the case where `mod` is a frequency function or spectrum in the freqfunc format, with a default choice of frequencies.

```
th = armax(z,nn);
th = sett(th,T);
```

is equivalent to

```
th = armax(z,nn,[],[],[],[],T)
```

Similarly,

```
g = spa(z);
g = sett(g,T);
```

is equivalent to

```
g = spa(z,[],[],[],T)
```

Note that you *cannot* use `sett` to recalculate models to a new sampling interval. Use `thc2thd` or `thd2thc` instead.

See Also

`freqfunc`, `theta`

spa

Purpose

Estimate frequency response and spectrum by spectral analysis.

Syntax

```
[g,phiv] = spa(z)
[g,phiv,z_spe] = spa(z,M,w,maxsize,T)
```

Description

spa estimates the transfer function g and the noise spectrum phiv = Φ_v of the general linear model

$$y(t) = G(q)u(t) + v(t)$$

where $\Phi_v(\omega)$ is the spectrum of $v(t)$.

Matrix z contains the output-input data z = [y u], where y and u are column vectors. If there are several inputs, u is a matrix with one column for each input. The data may be complex-valued.

g is returned in the frequency function format (see freqfunc) with the estimate of $G(e^{i\omega})$ at the frequencies ω specified by row vector w. The default value of w is

```
w = [1:128]/128*pi/T
```

phiv is returned with the autospectrum estimate of $\Phi_v(\omega)$ at the same frequencies. Both outputs are returned with estimated standard deviations (see freqfunc).

M is the length of the lag window used in the calculations. The default value is

```
M = min(30,length(z)/10)
```

Changing the value of M exchanges bias for variance in the spectral estimate. As M is increased, the estimated functions show more detail, but are more corrupted by noise. The sharper peaks a true frequency function has, the higher M it needs. See etfe as an alternative for narrowband signals and systems.

T is the sampling interval and maxsize controls the memory-speed trade-off (see auxvar).

For time series `z = y`, `g` is returned with the estimated output spectrum and its estimated standard deviation.

IMPORTANT: For multi-output data the argument `M` must be entered as a row vector of the same length as the number of outputs. This is the way the distinction between inputs and outputs in `z` is clarified. For default window size use in the multi-output case

---

```
M = [-1, -1, ..., -1]
```

The optional third output argument `z_spe` gives directly the spectrum matrix of `z` as follows:

`reshape(z_spe(:,k),nz,nz)` = The spectrum S at frequency W(k)

where `nz` is the number of channels in the data matrix `z` and

$$S = \sum_{m=-M}^{M} Ez(t+m)z(t)'\exp(-iW(k)mT)win(m)$$

Here *win(m)* is weight at lag m of an M-size Hamming window and *W(k)* is the frequency value i rad/s. Note that ' denotes complex-conjugate transpose.

The normalization of the spectrum differs from the one used by `spectrum` in the Signal Processing Toolbox . See “Some Special Topics” on page 3-68 for a more precise definition.

Examples

With default frequencies

```
g = spa(z);
bodeplot(g)
```

With logarithmically spaced frequencies

```
w = logspace(-2,pi,128);
[g,phiv] = spa(z,[],w);
% (empty matrix gives default)
bodeplot([g phiv],3)
```

plots the estimated spectrum together with confidence intervals of three standard deviations.

Algorithm

The covariance function estimates are computed using `covf`. These are multiplied by a Hamming window of lag size `M` and then Fourier transformed. The relevant ratios and differences are then formed. For the default frequencies, this is done using FFT, which is more efficient than for user-defined frequencies. For multi-variable systems, a straightforward for-loop is used.

Note that `M = ϒ` is in Table 6.1 of Ljung (1987). The standard deviations are computed as on pages 155-156 and 264 of Ljung (1987).

See Also

`auxvar`, `bodeplot`, `etfe`, `ffplot`, `freqfunc`, `th2ff`

References

Ljung (1987), Section 6.4.

Purpose

Create a model structure parametrized in canonical form.

Syntax

```
THS = ss2th(TH)
THS = ss2th(TH,orders)
```

Description

This function converts any model in theta format to a canonically parameterized state-space model, also in theta format. It is useful when a model has been obtained in some way, and you want to use it as an initial model for prediction error estimation using `pem`.

`TH`: The given model, which can be any model in theta format.

`THS`: The resulting model, also in theta format. A canonical parametrization in observer form, based on the pseudo-observability indices `orders`.

`orders`: The pseudo-observability indices. A row vector, with as many elements as there are outputs in the model `TH`. Their sum must be equal to the order of the model `TH`. See `canform` for more details. If `orders` is omitted, a default choice of indices is made.

If the model `TH` is an output error model (its Kalman gain equal to zero), then so is `THS`. Also if there is a delay from input to output in `TH` (corresponding to a state-space representation (3.27) in the *Tutorial* with `D=0`) then `THS` will also have such a structure.

Examples

Make a parametrized state-space model from given matrices `A`, `B`, `C`, `D`, and `K` and use it as initial condition for `pem` estimation:

```
th1 = ms2th(modstruc(A,B,C,D,K),'d');
thi = ss2th(th1);
th = pem(z,thi);
```

Let the model obtained from `n4sid` be used as the initial value for prediction error estimation:

```
thn = n4sid(z,3);
thp = pem(z,ss2th(thn));
```

See Also

`canform`, `canstart`, `ms2th`, `n4sid`

struc

Purpose

Generate model structure matrices.

Syntax

```
NN = struc(NA,NB,NK)
```

Description

`struc` returns in `NN` the set of model structures comprised of all combinations of the orders and delays given in row vectors `NA`, `NB`, and `NK`. The format of `NN` is consistent with the input format used by `arxstruc` and `ivstruc`. The command is intended for single-input systems only.

Examples

The statement

```
NN = struc(1:2,1:2,4:5);
```

produces

```
NN =
 1  1  4
 1  1  5
 1  2  4
 1  2  5
 2  1  4
 2  1  5
 2  2  5
```

Purpose

Convert a model from continuous time to discrete time.

Syntax

```
thd = thc2thd(thc,T)
```

Description

`thc` is a continuous-time model in the theta format. `thd` is what is obtained when it is sampled with sampling interval `T`. If `thc` is of input-output type, the covariance matrix of the parameters is not translated.

Note that the innovations variance λ of the continuous-time model is interpreted as the intensity of the spectral density of the noise spectrum. The innovations variance in `thd` will thus be given as λ/T.

Examples

Define a continuous-time system and study the poles and zeros of the sampled counterpart:

```
thc = poly2th(1,1,1,1,[1 1 0],1,-1);
thd = thc2thd(thc,0.5);
zpplot(th2zp(thd))
```

See Also

`thd2thc`

thd2thc

Purpose

Convert a model from discrete to continuous time.

Syntax

```
thc = thd2thc(thd)
thc = thd2thc(thd,delay,NoP)
```

Description

The discrete-time model `thd`, given in the theta format, is converted to a continuous-time counterpart `thc`. The covariance matrix of the parameters in the model is also translated using Gauss' approximation formula and numerical derivatives of the transformation. The step sizes in the numerical derivatives are determined by the function `nuderst`. To inhibit the translation of the covariance matrix and save time, enter `NoP = 1`.

If the discrete-time model contains pure time delays, i.e., $nk > 1$, then these are first removed before the transformation is made. These delays should then be appended as pure time-delay (deadtime) to the continuous-time model. This is done automatically by `th2ff`. To have the time delay approximated by a finite-dimensional continuous system, enter `delay = 'del'`. The default is `delay = 'nodel'`.

If the innovations variance is λ in `thd`, and its sampling interval is T, then the continuous-time model has an indicated level of innovations spectral density equal to $T * \lambda$.

IMPORTANT: The transformation from discrete to continuous time is not unique. `thd2thc` selects the continuous-time counterpart with the slowest time constants, consistent with the discrete-time model. The lack of uniqueness also means that the transformation may be ill-conditioned or even singular. In particular, poles on the negative real axis, in the origin, or in the point 1, are likely to cause problems. Interpret the results with care.

Examples

Transform an identified model to continuous time and compare the frequency responses of the two models:

```
gd = th2ff(th);
thc = thd2thc(th);
gc = th2ff(thc);
bodeplot([gd, gc],3)
```

See Also

`nuderst`, `thc2thd`

References

See "Some Special Topics" on page 3-68.

Purpose

Describe the theta format.

Syntax

```
help theta
help thss
```

Description

`theta` is a packed matrix containing information about both a model structure and its nominal or estimated parameters. It also contains other relevant information about the identification result.

This model format is the basic format with the System Identification Toolbox. It is used by all parametric identification methods and it can be transformed to many other model representations. See the tables in the beginning of this chapter for more details.

The internal format of the theta format is intended to be transparent to the user. The basic way to display the information is to use the `present` command. Some specific information is retrieved from the format by the functions `getmfth`, `getncap`, `gett`, and `th2par`. This entry gives the details of the internal representation, but this information is not necessary for most users of the System Identification Toolbox. The formats differ whether the underlying model is in state-space form or of the input-output black box character.

I. For the *general multi-input single-output linear model structure*

$$A(q)y(t) = \frac{B_1(q)}{F_1(q)}u_1(t-nk_1) + \ldots + \frac{B_{nu}(q)}{F_{nu}(q)}u_{nu}(t-nk_{nu}) + \frac{C(q)}{D(q)}e(t)$$

A, B, C, D, and F are polynomials in the delay operator of orders na, nb, nc, nd, and nf, respectively. If the system has nu inputs, nb, nf and nk are row vectors of dimension nu containing information about the orders and delays associated with each of the inputs. In the case of a time series (no u), B and F are not defined.

Let n be the sum of all the orders (the number of estimated parameters) and let

```
r = max(n, 7, 6  + 3 nu )
```

Then theta is a (3+*n*) *r* matrix organized as follows:

- Row 1 has entries: estimated variance of *e*, sampling interval *T*, *nu*, *na*, *nb*, *nc*, *nd*, *nf*, and *nk*.
- Row 2 has entries: FPE (Akaike's Final Prediction Error), year, month, date, hour, minute and command by which the model was generated. The matrix entry (2,7) thus contains the coded number of which command generated the model. This number is less than 20 for the black-box models of the type above.
- Row 3 is the vector of estimated parameters, A, B ,C, D, and F, (excluding leading 1s and 0s).
- Rows 4 to 3+*n* contain the estimated covariance matrix.
- For continuous-time models, a possible element (4+n,1) contains a dead-time for the system. This is used when computing frequency functions in th2ff.

II. For models that are defined as state-space structures there is an underlying M-file that defines the structure. This M-file is called by

```
[A,B,C,D,K,X0] = mfname(par,T,aux)
```

(See mf2th.) For model structures that are defined by ms2th, the name of this M-file is ssmodx9 or ssmodx8 and the argument aux is the actual model structure ms created by modstruc. Notice in particular that multi-output ARX models are internally represented as state-space models with the aid of ssmodx9.

Suppose that the number of estimated parameters is *n* and that the length of the name of your M-file mfname is *r*. Suppose also that the model has *ny* outputs and that the size of the argument aux above is *nr* times *nc*. Then theta is a matrix of dimension

max(n, ny, nc, 7 + r) by 3+ n + nr + ny)

organized as follows:

- Row 1 has entries: determinant of innovations covariance, sampling interval, number of inputs, number of outputs, number of estimated parameters, number of rows of aux, number of columns of aux, and name of M-file that defines the structure.

- Row 2 contains the entries: FPE (Akaike's Final Prediction Error), year, month, date, hour, minute, and command by which the model was generated. This last one is a number larger than 20 for state-space structures.
- Entry (2,8) is interpreted as follows: "1" means that the underlying parameterization is in continuous time, using ssmodx9. "ll" means that the underlying parameterization is in continuous time, to be sampled by first-order-hold, using ssmodx8. "2" means that it is in discrete time, again using ssmodx9. '3" means that the model is a multivariate ARX model. "4" means that the underlying parameterization is in discrete time and user defined. "5" means that the model is a user-defined continuous-time parameterization, equipped with sampling inhibition when called with a negative value of T.
- Row 3 contains the estimated (nominal) values of the parameters.
- Rows 4 to 3 + *n* contain the estimated covariance matrix of the parameters.
- Rows 4 + *n* to 3 + *n* + *nr* contain the matrix aux.
- Rows 4 +*n* + *nr* to 3+*n* + *nr* + *ny* contain the covariance matrix of the innovations of the model.

thinit

Purpose

Set initial values for the parameters to be estimated.

Syntax

```
th = thinit(th0)
th = thinit(th0,R,pars,sp)
```

Description

This function randomizes initial parameter estimates for model structures `th0` in the theta format that correspond to state-space models. `th` is the same model structure as `th0`, but with a different nominal parameter vector. This vector is used as the initial estimate by `pem`.

The parameters are randomized around `pars` with variances given by the row vector `R`. Parameter number *k* is randomized as `pars(k)` + `e*sqrt(R(k))`, where `e` is a normal random variable with zero mean and a variance of 1. The default value of `R` is all ones, and the default value of `pars` is the nominal parameter vector in `th0`.

Only models that give stable predictors are accepted. If `sp = 'b'`, only models that are both stable and have stable predictors are accepted.

`sp = 's'` requires stability only of the model, and `sp = 'p'` requires stability only of the predictor. `Sp = 'p'` is the default.

A maximum of 100 trials are made by `thinit`. It may be difficult to find a stable predictor for high order systems just by trial and error. An alternative is then to compute the Kalman filter predictor for a randomized model.

See Also

`canstart`, `mf2th`, `ms2th`, `pem`

Purpose

Extract the ARX parameters from a theta format model.

Syntax

```
[A,B] = th2arx(th)
[A,B,dA,dB] = th2arx(th)
```

Description

`th` is the model in the theta format. `A` and `B` are returned as the matrices that define the ARX structure:

```
A = [I A1 A2 ... Ana]
B = [B0 B1 ... Bnb]
```

where

$$y(t) + A_1 y(t-1) + \ldots + A_{na} y(t-na) = B_0 u(t) + B_1 u(t-1) + \ldots + B_{nb} u(t-nb)$$

Delays in the system are indicated by leading zeros in the *B* matrices. See Section 6 in the *Tutorial*.

`dA` and `dB` are the standard deviations of `A` and `B`.

See Also

`arx2th`

th2ff, trf

Purpose

Compute the frequency response and standard deviations of a theta format model.

Syntax

```
[g,phiv] = th2ff(th)
[g,phiv] = th2ff(th,ku,w,ky)
```

Description

`th2ff` computes the frequency functions of `th`, where `th` is a matrix in theta format containing a general model.

`g` is returned with the transfer function estimate (see (3.4), (3.23), and (3.25) in the *Tutorial*)

$$G(e^{i\omega})$$

computed at the frequencies ω given in row vector `w`. If `th` has several inputs and outputs, `g` is computed for the input and output numbers specified in row vectors `ku` and `ky`, respectively. The format of `g` is detailed under `freqfunc`. The default values of `ku` and `ky` are all input and output pairs contained in `th`.

For a time continuous model the frequency function $G(i\omega)$ is obtained instead.

`Phiv` (Φ_v) is returned with the estimated noise spectrum for each of the outputs

$$\Phi_v(\omega) = \lambda * T * \left|H(e^{iwt})\right|^2$$

where λ is the estimated variance of *e(t)* (the loss function) specified by `th`. `phiv` is computed at the same frequencies as `g`. The normalization of the spectrum differs from the one used by `spectrum` in the Signal Processing Toolbox. See “Some Special Topics” on page 3-68, for a precise definition. Note that no cross-spectra between different outputs are computed.

The standard deviations of the frequency function(s) and the spectra are also computed, using the Gauss approximation formula. For models with complicated parameter dependencies, numerical differentiation is applied. The step sizes for the numerical derivatives are determined by `nuderst`.

The default values of the frequencies are in the discrete-time case are

```
w = [1:128]/128*pi/T
```

where `T` is the sampling interval specified by `th` (default = 1) and for the continuous-time case

```
w = logspace(log10(pi/abs(T)/100),log10(10*pi/ abs(T)),128)
```

where `abs(T)`is the "underlying" sampling interval for the continuous-time model.

The frequency functions can be graphed with `bodeplot`, `ffplot`, and `nyqplot`.

IMPORTANT: The command `trf` has the same syntax as `th2ff` but does not calculate the standard deviations, and can be considerably faster.

Examples

Compare the results from spectral analysis and an ARMAX model (input-output dynamics only).

```
th = armax(z,[2 2 2 1]);
gp = th2ff(th);
gs = spa(z);
bodeplot([gs gp])
```

Plot, but don't store, the frequency function `g` associated with `th`:

```
bodeplot(th2ff(th))
```

See Also

`bodeplot`, `etfe`, `ffplot`, `nyqplot`, `sett`, `spa`

th2par

Purpose

Extract the parameters from the theta format.

Syntax

```
[par,P,lam] = th2par(th)
```

Description

`th` is a model, defined in the theta format. `par` is returned as the nominal or estimated values of the free parameters in `th`. The covariance matrix of the parameters is obtained as `P`, and `lam` is the variance (covariance matrix) of the innovations.

The ordering for the parameters is as follows. For the general input-output model (7.2), you have

$$pars = [a_1, \ldots, a_{na}, b_1^1, \ldots, b_{nb1}^1, \ldots, b_1^2, \ldots, b_{nb2}^2, \ldots$$

$$b_1^{nu}, \ldots, b_{nbnu}^{nu}, \ldots, c_1, \ldots, c_{nc}, d_1, \ldots, d_{nc}, f_1^1, \ldots, f_{nf1}^1, \ldots, f_1^{nu}, \ldots, f_{nfnu}^{nu}]$$

Here, superscript refers to the input number.

For a state-space structure, defined by `ms2th`, the parameters in `pars` are obtained in the following order: The *A* matrix is first scanned row by row for free parameters. Then the *B* matrix is scanned row by row, and so on for the *C*, *D*, *K*, and *X0* matrices. (See `ms2th`.)

For a state-space matrix that is defined by `mf2th`, the ordering of the parameters is the same as in your M-file.

Multivariate ARX models are internally represented in state-space form. The ordering of the parameters may not be transparent in this case; it is better to use `th2arx`.

Purpose

Convert theta matrix into its component polynomials.

Syntax

```
[A,B,C,D,F,LAM,T] = th2poly(th)
```

Description

This is essentially the inverse of the `poly2th` function. It returns the polynomials of the general model

$$A(q)y(t) = \frac{B_1(q)}{F_1(q)}u_1(t - nk_1) + \ldots + \frac{B_{nu}(q)}{F_{nu}(q)}u_{nu}(t - nk_{nu}) + \frac{C(q)}{D(q)}e(t)$$

as contained by the matrix `th` in theta format. See "Examining Models" on page 3-40.

`LAM` is the noise variance and `T` is the sampling interval.

See Also

`poly2th`, `th2tf`, `theta`

th2ss

Purpose

Transform a model to state-space form.

Syntax

```
[A,B,C,D,K,X0] = th2ss(th)
[A,B,C,D,K,X0,dA,dB,dC,dD,dK,dX0] = th2ss(th)
```

Description

`th` is the model given in the theta format. `A`, `B`, `C`, `D`, `K`, and `X0` are the matrices in the state-space description

$$\tilde{x}(t) = Ax(t) + Bu(t) + Ke(t)$$

$$x(0) = x0$$

$$y(t) = Cx(t) + Dx(t) + e(t)$$

where $\tilde{x}(t)$ is $\dot{x}$ or $x(t+1)$ depending on whether `th` is a continuous or discrete-time model.

`dA`, `dB`, `dC`, `dD`, `dK`, and `dX0` are the standard deviations of the state-space matrices.

If the underlying model itself is a state-space model, the matrices correspond to the same basis. If the underlying model is an input-output model, an observer canonical form representation is obtained.

Algorithm

The computation of the standard deviations in the input-output case assumes that an A polynomial is not used together with a F or D polynomial in (7.2). For the computation of standard deviations in the case that the state-space parameters are complicated functions of the parameters, Gauss approximation formula is used together with numerical derivatives. The step sizes for this differentiation are determined by `nuderst`.

See Also

`mf2th`, `ms2th`, `nuderst`

Purpose

Transform a model to transfer function form.

Syntax

```
[num,den] = th2tf(th)
[num,den] = th2tf(th,iu)
```

Description

`th` is a model given in the theta format. `num` is a matrix whose row number `k` gives the numerator polynomial associated with the transfer function from input number `iu` to output number `k`. `den` is a row vector giving the (common) denominator polynomial for these transfer functions. The default value of `iu` is 1.

The formats of `num` and `den` are the same as those used by the Signal Processing Toolbox and the Control Systems Toolbox, both for continuous-time and discrete-time models. See "Examining Models" on page 3-40.

To obtain the transfer function from noise source number `k`, enter `iu = -k`.

Examples

For a continuous-time model

```
num = [1 2]
den = [1 3 0]
```

corresponds to the transfer function

$$G(s) = \frac{s+2}{s^2+3s}$$

For a discrete-time model

```
num = [2 4 0]
den = [1 2 3 5]
```

corresponds to the transfer function

$$H(z) = \frac{2z^2+4z}{z^3+2z^2+3z+5}$$

which is the same as

$$H(q) = \frac{2q^{-1}+4q^{-2}}{1+2q^{-1}+3q^{-2}+5q^{-3}}$$

See Also

`th2poly`

th2zp, zp

Purpose

Compute zeros, poles, and static gains of a theta format model.

Syntax

```
[zepo,k] = th2zp(th)
[zepo,k] = th2zp(th,ku,ky,thresh)
```

Description

For any model described in theta format by `th`, the poles and zeros and static gains, along with their standard deviations are computed. The poles and zeros are stored in coded form in the matrix `zepo`, while the static gains are returned in `k`.

The best way to display the information in `zepo` is to plot it using `zpplot`. The information can also be retrieved with `getzp`.

The first row of `k` contains the integer $(jy-1)*1000+ju$, where *jy* is the output number and *ju* is the input number for the gain in question. The second row of `k` contains the corresponding gains, and the third row the standard deviation of the gain in question.

NOTE: The gain here is the static gain, i.e., the steady state gain from a step in input number *ju* to output number *jy*. It is thus the same for a discrete-time and a continuous-time representation of the same model. This is different from the routines `ss2zp` and `zp2ss` in the Signal Processing Toolbox, which use the *transfer function gain*, i.e., the coefficient of the highest power in the numerator.

Row vectors `ku` and `ky` contain the indices of the inputs and outputs, respectively, for which the zeros, poles and gains are to be computed. In this context, the noise *e(t)* is counted as inputs with negative numbers. That is, noise source number *ju* (the *ju*-th component of `e(t)` is counted as input number *ju*. The value 0 in the vector `ku` is the default for "all noise sources." The default values of `ku` and `ky` are all inputs and all outputs (no noise inputs).

The optional argument `thresh` is a threshold for the computation of the zeros. Zeros at infinity may, due to the numerical procedure, end up as large, finite values. To avoid this, any zero whose absolute value is larger than `thresh` is regarded to be at infinity. The default value of `thresh` is 100000.

The procedure handles both models in continuous and discrete time.

For the general discrete-time multi-input, single-output model

$$A(q)y(t) = \frac{B_1(q)}{F_1(q)}u_1(t - nk_1) + \ldots + \frac{B_{nu}(q)}{F_{nu}(q)}u_{nu}(t - nk_{nu}) + \frac{C(q)}{D(q)}e(t)$$

the zeros are the roots of $z^{nb+nk}B(z)$ (with z replacing the forward shift operator q), and the poles are the roots of $z^{na+nf}A(z)F(z)$. The static gain is $k = B(1)/(A(1)F(1))$.

For models that are internally represented in state-space form, the poles are the eigenvalues of the system matrix. The zeros are the zeros of the corresponding transfer function. These zeros may thus differ from the *transmission zeros* associated with the multivariable system. To find the trans-mission zeros, first use `th2ss` and then apply `tzero` from the Control System Toolbox.

Note that you cannot rely on information about zeros and poles at the origin and at infinity. (This is a somewhat confusing issue anyway.)

`zepo` is returned in a format that allows easy plotting with `zpplot`. (Then zeros and poles at the origin and at infinity are ignored.) The routine `zpform` is useful when comparing different models.

The alternative routine `zp` has the same syntax as `th2zp` but does not compute standard deviations. This can be a useful alternative, when the standard deviations are not of interest, and computation time for `th2zp` is long. Moreover, `zp` uses `ss2zp` from the Signal Processing Toolbox, which may give better numerical accuracy in difficult cases.

Note: Although `zp` computes zeros and poles for all combinations of noise sources and outputs, present in the indices `ky`, `ku`, the command `th2zp` only gives information about poles and zeros from noise source number *ju* to output number *ju* (if `ku` contains the number *-ju*).

Algorithm

The standard deviations are computed using Gauss's approximation formula, using the parameter covariance matrix contained in `th`. When the transfer function depends on the parameters in a complicated way, numerical

differentiation is applied. The step sizes for the differentiation are determined in the M-file `nuderst`.

Note that Gauss's approximation formula gives infinite variance to poles and zeros that are exactly repeated.

Examples

The statement

```
zpplot(th2zp(th))
```

plots, but does not store, the poles and zeros.

To compare the zeros and poles of second and third order ARX models (input-output dynamics only), use

```
th2 = arx(z,[2 2 1]);
th3 = arx(z,[3 3 1]);
zp2 = th2zp(th2);
zp3 = th2zp(th3);
zpplot(zpform(zp2,zp3))
```

See Also

`getzp`, `theta`, `zepo`, `zpform`, `zpplot`

Purpose

Free parameters in structures defined by `ms2th` and `arx2th`.

Syntax

```
thn = unfixpar(tho,matrix)
thn = unfixpar(tho,matrix,elements)
```

Description

This function is the inverse of `fixpar`. The interpretation of the arguments is the same. `unfixpar` makes the indicated parameters free parameters to be estimated. The nominal/initial values of these parameters are taken to be equal to their actual values in the structure `tho`.

See Also

`fixpar`, `ms2th`, `theta`, `thinit`

zepo

Purpose

Describe the zeros and pole format.

Syntax

```
help zepo
```

Description

The zepo format is created by `th2zp` and `zp` and used by `zpplot`. It contains information about zeros and poles and their standard deviations. The internal format is intended to be transparent to the user. The basic way to display the information is to use the `zpplot` command. Some specific infor-mation is retrieved from the format by the function `getzp`. This entry gives the details of the internal representation, but this information is not necessary for most users of the System Identification Toolbox.

The first row of the matrix consists of integers that give information about what the column below contains. The integers are coded in the following way:

- The zeros associated with input number *ku* and output number *ky* correspond to the number (*ky* - 1)*1000 + *ku*.
- The standard deviations of these zeros correspond to the number (*ky*-1)*1000 + *ku* +60.
- The poles associated with input number *ku* and output number *ky* correspond to the number (*ky* - 1)*1000 + *ku* + 20.
- The standard deviation of these poles correspond to the number (*ky* - 1)*1000 + *ku* + 80.
- The zeros associated with noise input number *ky* and output number *ky* (only these are normally represented) correspond to the number 500 + *ky*.
- The standard deviation of these, the corresponding poles, and their standard deviations are obtained by adding 60, 20, and 80, respectively to this number.
- Positions corresponding to nonexisting zeros and poles (as well as zeros and poles at infinity) are represented by `inf`.
- If any of the above numbers is negative, it indicates that the pole or zero representation corresponds to a continuous-time model. Then the absolute value of the number has the interpretation above.

- For complex conjugated pairs, the first row in the corresponding entry for the standard deviation contains a complex number whose real and imaginary parts are the standard deviations of the real and imaginary parts, respectively, of the pole or zero in question. The next row entry (corres-ponding to the conjugate pole or zero) contains a real number that is the correlation between the real and imaginary parts.

See Also `getzp`, `th2zp`, `zpform`, `zpplot`

zpform

Purpose

Merge zero-pole information from different models.

Syntax

```
zepo = zpform(zepo1,zepo2,...,ku)
```

Description

The zeros and poles in `zepo1`, `zepo2`, ... from different models are merged into one matrix to be used by `zpplot`. `zepo1`, `zepo2`, ... have the format as produced by `th2zp`. `ku` is an optional row vector containing the input numbers to be picked out when forming `zepo`. The default value is `ku` is equal to all inputs present in `zepo1`, `zepo2`, ... A maximum of five input arguments to `zpform` is possible.

Examples

The statement

```
zpplot(zpform(zpbj2,zpbj4,zpax2,zpax4,0))
```

compares the noise characteristics from four different models.

See Also

`th2zp`, `zpplot`

Purpose

Plot zeros and poles.

Syntax

```
zpplot(zepo)
zpplot(zpform(zepo1,zepo2, ..., zepon))
zpplot(zepo,sd,mode,axis)
```

Description

The zeros and poles specified by `zepo` (see `zepo` for the format) are graphed, with `o` denoting zeros and `x` denoting poles. Poles and zeros associated with the same input, but different models, are always graphed in the same diagram, and pressing the **Return** key advances the plot from one model to the next. On color screens poles, zeros and their confidence regions, corresponding to the same model all have the same color. Poles and zeros at infinity are ignored. For discrete-time models, zeros and poles at the origin are also ignored.

If `sd` has a value larger than zero, confidence regions around the poles and zeros are also graphed. The regions corresponding to `sd` standard deviations are marked. The default value is `sd = 0`. Note that the confidence regions may sometimes stretch outside the plot, but they are always symmetric around the indicated zero or pole.

If the poles and zeros are associated with a discrete-time model, a unit circle is also drawn.

When `zepo` contains information about several different inputs, there are some options:

`mode = 'sub'` splits the screen into several plots.

`mode = 'same'` gives all plots in the same diagram. Pressing the **Return** key advances the plots.

`mode = 'sep'` erases the previous plot before the next input is treated.

The default value is `mode = 'sub'`.

`axis = [x1 x2 y1 y2]` fixes the axis scaling accordingly. `axis = m` is the same as

```
axis = [-m m -m m]
```

Examples

```
zpbj = th2zp(thbj2);
zpax = th2zp(tharmax4);
zpoe = th2zp(thoe3);
zpplot(zpform(zpoe,zpax,zpbj),3)
```

show all zeros and poles of three models along with the confidence regions corresponding to three standard deviations.

See Also

`th2zp`, `zpform`

Index

Y

Z