# Frequency Domain System Identification Toolbox

For Use with MATLAB®

*István Kollár*

**Computation**

**Visualization**

**Programming**

**Copyright page**

Please note that support and maintenance is provided by the *toolbox authors*, and no longer by The MathWorks.

### How to Contact the Authors

| | |
|---|---|
| http://elecwww.vub.ac.be/fdident/ | Toolbox Web site, with general information and details on new versions |
| fdident@vub.ac.be | Technical support and product enhancement suggestions |
| fdident@gamax.hu | Sales, pricing, and general information |

---

# Frequency Domain System Identification Toolbox User's Guide



Printing History:

| | |
|---|---|
| January 1994 | First printing (The MathWorks) |
| August 1995 | Reprint (The MathWorks) |
| January 2001 | Electronic version |

# This Manual

This manual describes the basic functionalities of the toolbox, and contains a guide to the backward compatible command-line calls of the functions (compatible with toolbox version 2.0.3, distributed until November 2000, with Matlab 5.x).

Each function also has a new, simple calling form, based on the objects. To learn about these, type 'help <function>' or 'usage <function>' in the command line.

There is a new, easy-to-use graphical user interface. Start it by typing fdtool . Further information is available from http://elecwww.vub.ac.be/fdident/.

Most of the data files have been transformed to MAT-files, containing objects. Learn how to use these by typing

```
help fiddata
help(fiddata)
helpc(fiddata)
get(fiddata)
set(fiddata)
help tiddata
help(tiddata)
helpc(tiddata)
help fidmodel
help(fidmodel)
helpc(fidmodel)
etc.
```

# Contents

## 1 Before You Begin



## 2 Tutorial

## 3 Reference

# Appendix A

# About the Author

## István Kollár

István Kollár received his M.S. degree in Electrical Engineering from the Technical University of Budapest, Hungary, in 1977, and his Cand. Sci. and Dr. Acad. degrees from the Hungarian Academy of Sciences, Budapest, in 1985 and in 1998, respectively. He is professor of Electrical Engineering at the Technical University of Budapest.

Dr. Kollár is a Fellow of the IEEE. He has published over 70 technical papers and a textbook in the areas of signal processing and system identification. He is co-author of the book, Technology of Electrical Measurements, Schnell, L. ed., Wiley, 1993.

1

# Before You Begin

Modern system identification methods heavily use matrix calculations, usually based on complex numbers. Therefore, algorithms for system identification can be very effectively implemented in MATLAB. Its interactive environment and the graphics possibilities offer an easy-to-use and flexible tool for specific applications and for further development of the implemented methods. This toolbox is a collection of frequency domain system identification procedures, covering the wholeidentification process from excitation signal design, through data preprocessing, parameter estimation, graphics presentation of the results, to model verification.

## Installation of the Toolbox

Installation instructions come from the distributor or along with your electronically downloaded software. Demonstration are most easily available through the graphical user interface. Start this by typing fdtool, the demonstrations are accessible under the help menu.
The demonstrations recalculate several of the results given in the book of Schoukens and Pintelon, *Identification of Linear Systems: A Practical Guideline for Accurate Modelin*g, London, Pergamon Press, 1991.

# Introduction

Identification means determining models of physical systems from noisy measured data. Since the modeling of nature is the basis of our understanding of the world, identification methods have applications in virtually every field of sciences, especially technical ones.

One of the most often used models for dynamic physical systems is an ordinary linear differential equation with constant coefficients. This model appears in the same mathematical form in very different fields. Thus, the common properties of these equations, and the measurement and estimation procedures of their coefficients can be treated independently of the dimensions of the physical quantities. This is done in the frame of linear system theory.

Linear dynamic systems have two equivalent descriptions: in the time domain (differential equations), and in the frequency domain (transfer functions in the f-domain or in the s-domain). Also the discrete-time descriptions exhibit this duality: difference equations are equivalent to the *z*–domain transfer functions.

Time domain and frequency domain have different advantages and disadvantages, they are in several respects complementary to each other. Engineers often prefer frequency domain descriptions, because of the following reasons:

- While the solution of a differential equation needs convolution in the time domain, this convolution is substituted by simple multiplication in the frequency domain. This often makes it possible to explain system behavior in a visual way.
- It is often possible to decompose signals/noises into different frequency bands. Signal-to-noise ratios can often be improved in the frequency domain by choosing appropriate bands, and an occasional dc offset can also be easily removed. Moreover, the energy (power) of periodic signals is concentrated to discrete points in the frequency domain. Thus, frequency domain is very selective with respect to periodic components.
- By selecting the Fourier coefficients of the appropriate frequency band only, significant data reduction can be achieved.

- It is often easier to calculate accurately a good model of a (continuous-domain) physical system in the *s*-domain, using a digital computer, than in the time domain. Continuous-time linear systems can be modeled without systematic errors by difference equations, if the excitation is *piecewise constant* (Ljung, 1987). If this assumption is not met, modeling errors are introduced, resulting in a limited accuracy of the results.
- The attainable dynamic range is usually significantly larger in the frequency domain than in the time domain.
- Slight nonlinearities are easier detected and measured by frequency domain methods.
- FFT is a very powerful tool for fast time domain to frequency domain conversion, and thus, e. g., the evaluation of a correlation function is much quicker via the frequency domain, than directly in the time domain.

On the other hand, *time domain* has also important advantages:

- It is very “natural” to deal with time domain signals (although after some practice, even the two-sided frequency axis becomes very visual, too).
- Recursive methods often provide on-line calculation possibilities.
- Time varying systems are easier modeled in the time domain.
- The transient behavior of systems can be directly measured in the time domain.
- Time domain digital methods are not very sensitive to the type of the signal, while frequency domain methods suffer from leakage effects when non-periodic signals are processed.
- Certain nonlinearities (clipping, slew rate etc.) are easier detected in the time domain.

Although the two descriptions are basically equivalent to each other, the formulation of the identification problem leads to very different methods in the two domains. Thus, time domain identification methods and frequency domain identification methods form two distinct groups.

Time domain methods are covered by several books, and a comprehensive System Identification Toolbox is available for MATLAB. However, frequency domain parametric methods are usually not treated, and these are not incorporated in the System Identification Toolbox.

This Frequency Domain System Identification Toolbox has been written to fill this gap. It is based mostly on the book of Schoukens and Pintelon (1991), and covers the whole identification procedure from excitation signal design through data preprocessing and system parameter estimation to model validation. It is also possible to convert identified parameters to the System Identification Toolbox and vice versa.

The *Tutorial* chapter contains the essentials for the use of these methods, thus the toolbox can be used in its own right. However, for a more profound understanding of the methods, the book of Schoukens and Pintelon is highly recommended.

MATLAB proved to be an ideal frame for all these methods, because most algorithms are based on complex vector/matrix calculations and array manipulations, and since interactive graphical checking of the results is often essential. MATLAB also provides an easy-to-use environment for the embedding of the prepared routines into a dedicated program, written for use in a given field. Thus, the use of the toolbox may be twofold: it can be used as a tool for the design and evaluation of experiments, and also as a frame to check ideas before writing a lengthy special-purpose measurement and data processing program.

The demonstrations accompanying the toolbox are not merely illustrations to the use and the power of the functions, but also work with the measured data used in the examples of Schoukens and Pintelon. Thus, the toolbox is an ideal supplement to the book.

The author of this toolbox is very much indebted to Johan Schoukens and Rik Pintelon for long and fruitful discussions on the implemented methods, and also to the Vrije Universiteit Brussel and to the National Fund for Scientific Research of Belgium for providing the conditions of this project.

Thanks are due also to Yves Rolain, Patrick Guillaume, Hugo Van hamme, Béla Pataki, Tadeusz Dobrowiecki, Frank Louage, Françoise Renneboog and Johan Top for their useful suggestions and remarks.

Budapest, Aug. 23, 1993.

*István Kollár*

# Tutorial

# Frequency Domain Formulation and Solution

## Basic Concepts

A general model used in the frequency domain identification of dynamic linear systems is shown in Figure 2-1. The system is represented by its transfer function H(Ω), where $\Omega = s = j\omega = j2\pi f$ in the Laplace-domain, or $\Omega = z^{-1} = \exp(-j\omega T_s)$ in the $z$–domain, respectively, and $H$ is a rational form, eventually extended by a delay term, (see Equation (1).

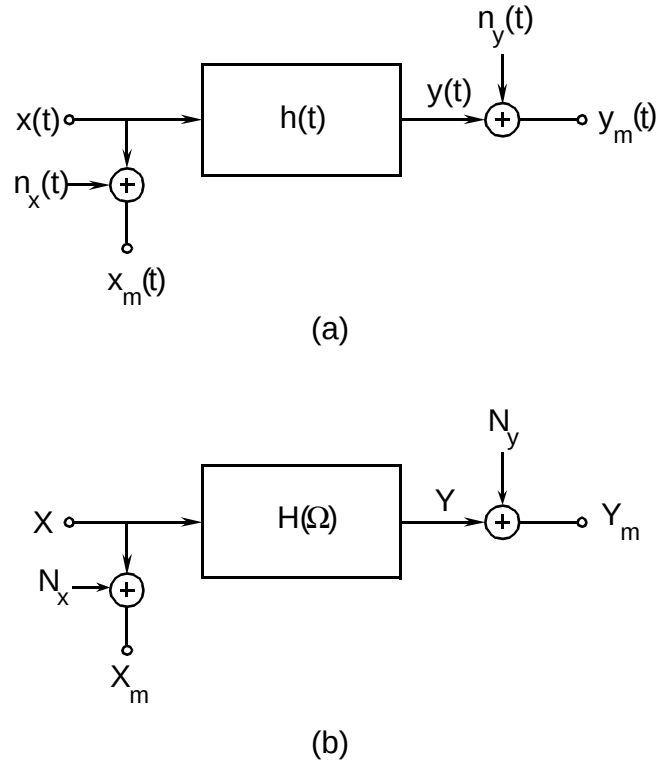



**Figure 2-1: The general model used in frequency domain system identification.**

$$H(\Omega) = e^{-j\omega T_d} \frac{b_0\Omega^0 + b_1\Omega^1 + \ldots b_{nn}\Omega^{nn}}{a_0\Omega^0 + a_1\Omega^1 + \ldots a_{nd}\Omega^{nd}} \quad \textbf{(1)}$$

The excitation signal has complex amplitudes $X_k$ at angular frequencies $\omega_k$, the response of the system is $Y_k = H(\Omega_k)X_k$. The measured input and output complex amplitudes are both corrupted by noises $N_x$ and $N_y$ (errors-in-variables model), which are usually assumed to be Gaussian, uncorrelated between input and output, and also uncorrelated between different frequency points. Input-output correlation may also be considered, see Equation (14) later in this chapter.

Measurements are made at angular frequencies $\omega_k$, $k = 1...F$, the measured complex input and output amplitudes are $X_{mk}$ and $Y_{mk}$, respectively. The unknown parameters are those of the transfer function (vector $\mathbf{P}$), and the complex input and output amplitudes (vectors $\mathbf{X}$ and $\mathbf{Y}$). The basic equations can be written as

$$Y_k = H(\Omega_k,\mathbf{P})X_k, \quad k = 1,2\ldots F, \quad \textbf{(2)}$$

and

$$Y_{mk} = H(\Omega_k,\mathbf{P})(X_{mk} - N_{xk}) + N_{yk}, \quad k = 1,2\ldots F, \quad \textbf{(3)}$$

Assuming that the noise on the complex amplitudes is Gaussian and uncorrelated, its joint probability density function can be written as

$$\begin{aligned} p(\mathbf{N}_x,\mathbf{N}_y) &= \\ &= \prod_{k=1}^{F} \frac{1}{2\pi\sigma_{xk}^2} \exp\left(-\frac{N_{Rxk}^2 + N_{Ixk}^2}{2\sigma_{xk}^2}\right) \prod_{k=1}^{F} \frac{1}{2\pi\sigma_{yk}^2} \exp\left(-\frac{N_{Ryk}^2 + N_{Iyk}^2}{2\sigma_{yk}^2}\right) \\ &= \prod_{k=1}^{F} \frac{1}{2\pi\sigma_{xk}^2} \exp\left(-\frac{N_{xk}\overline{N_{xk}}}{2\sigma_{xk}^2}\right) \prod_{k=1}^{F} \frac{1}{2\pi\sigma_{yk}^2} \exp\left(-\frac{N_{yk}\overline{N_{yk}}}{2\sigma_{yk}^2}\right) \end{aligned} \quad \textbf{(4)}$$

where $N_{Rxk}$, $N_{Ixk}$, $N_{Ryk}$ and $N_{Iyk}$ are the real and imaginary parts of the input and the output noise samples, respectively. $\overline{N}$ is the complex conjugate of $N$, $\sigma_{xk}$ and $\sigma_{yk}$ are the corresponding standard deviations, $N_x$ and $N_y$ denote the

vectors formed of $N_{xk}$, $N_{yk}$, respectively. Here the input and output noises are assumed to be uncorrelated. Correlatedness will be considered later in this section.

By expressing the noise variables in (2.3) by $X_{mk}$ and $X_k$ ($N_{xk} = X_{mk}$-$X_k$, $N_{Rxk} = X_{Rmk}$-$X_{Rk}$ etc.), and taking the logarithm, with the assumption that $\sigma_x$ and $\sigma_y$ are known from a preceding noise analysis, the log-likelihood function is obtained:

$$\ln(L(X, Y, P)) = \\ = const - \sum_{k=1}^{F}\left(\frac{(X_{mk} - X_k)\overline{(X_{mk} - X_k)}}{2\sigma_{xk}^2}\right) - \sum_{k=1}^{F}\left(\frac{(Y_{mk} - Y_k)\overline{(Y_{mk} - Y_k)}}{2\sigma_{yk}^2}\right) \quad \textbf{(5)}$$

**X**, **Y** and **P** are not independent of each other, since Equation (2) must be fulfilled.

The maximization of Equation (5) is equivalent to the minimization of

$$C_{LS}(X, Y, P) = \sum_{k=1}^{F}\left(\frac{(X_{mk} - X_k)\overline{(X_{mk} - X_k)}}{2\sigma_{xk}^2}\right) + \sum_{k=1}^{F}\left(\frac{(Y_{mk} - Y_k)\overline{(Y_{mk} - Y_k)}}{2\sigma_{yk}^2}\right) \quad \textbf{(6)}$$

subject to the constraints

$$Y_k = H(\Omega_k, \mathbf{P})X_k, \quad k = 1,2\ldots F \quad \textbf{(7)}$$

The constraints can be substituted into Equation (6), to eliminate **Y**. The result is a nonlinear weighted least squares problem. Since generally we are not interested in **X** either, a better way of the minimization of Equation (5) with the constraints Equation (7) is to use the Lagrange multiplier technique to eliminate both **X** and **Y**. Fortunately, **X**, **Y** and the multipliers can really be eliminated, and the following expression is obtained for minimization:

$$C(\mathbf{P}) = \frac{1}{2}\sum_{k=1}^{F}\frac{\left|e^{-j\omega_k T_d}N(\Omega_k, \mathbf{P})X_{mk} - D(\Omega_k, \mathbf{P})Y_{mk}\right|^2}{\sigma_{yk}^2\left|D(\Omega_k, \mathbf{P})\right|^2 + \sigma_{xk}^2\left|N(\Omega_k, \mathbf{P})\right|^2} \quad \textbf{(8)}$$

where $N(\Omega,\mathbf{P})$ and $D(\Omega,\mathbf{P})$ are the numerator and the denominator of the transfer function, respectively.

The cost function in Equation (8) may look somewhat strange. However, a quite simple explanation of the underlying idea can be given as follows. This is by no means a proof of the maximum likelihood nature of the estimate, but can help to understand its structure.

A “natural” way of developing an appropriate estimate of the transfer function is to minimiz

$$C_{tf}(\mathbf{P}) = \frac{1}{2}\sum_{k=1}^{F} W_{tfk}\left|e^{-j\omega_k T_d}\frac{N(\Omega_k,\mathbf{P})}{D(\Omega_k,\mathbf{P})} - \frac{Y_{mk}}{X_{mk}}\right|^2 \tag{9}$$

This is a weighted least squares type cost function. The weights have to be chosen equal to the variances of the terms whose absolute values are taken, in order to have an approximately chi-squared cost function. (The chi-squared cost function is the one usually obtained in maximum likelihood estimations for Gaussian data.)

The problem is that the terms between the absolute value signs are not any more Gaussian distributed because of the division by $X_{mk}$, moreover, the division makes the distribution asymmetric, introducing a bias. So this is not a proper way to obtain a cost function that provides high accuracy.

An alternative formulation for avoiding division is to investigate the terms in

$$C_{WLS}(\mathbf{P}) = \frac{1}{2}\sum_{k=1}^{F} W_{tfk}\left|e^{-j\omega_k T_d}X_{mk}N(\Omega_k,\mathbf{P}) - Y_{mk}D(\Omega_k,\mathbf{P})\right|^2 \tag{10}$$

For zero noise, and a proper model, this expression equals zero. The terms are Gaussian, and independent of each other, so if the proper weights $W_k$ can be found to form a chi-squared cost function, minimization of this may provide a good estimate.

Since

$$e^{-j\omega_k T_d}X_k N(\Omega_k,\mathbf{P}) - Y_k D(\Omega_k,\mathbf{P}) = 0 \tag{11}$$

we are looking for the variances of the remaining terms

$$e_k = e^{-j\omega_k T_d}N_{xk}N(\Omega_k,\mathbf{P}) - N_{yk}D(\Omega_k,\mathbf{P}) \tag{12}$$

The variances are equal to

$$\begin{aligned}\mathrm{var}\{e_k\} &= E\{\overline{N_{xk}}N_{xk}\}\left|N(\Omega_k,\mathbf{P})\right|^2 + E\{\overline{N_{yk}}N_{yk}\}\left|D(\Omega_k,\mathbf{P})\right|^2 \\ &= 2\sigma_{xk}^2\left|N(\Omega_k,\mathbf{P})\right|^2 + 2\sigma_{yk}^2\left|D(\Omega_k,\mathbf{P})\right|^2\end{aligned} \quad \textbf{(13)}$$

for independent $N_{xk}$ and $N_{yk}$ . Setting $W_k$ =1/var{$e_k$}, the cost function Equation (8) of ELiS is obtained. For correlated input-output noise, a more general weighting can be developed (see Equation (14)).

Since Equation (8) is a sum of quadratic terms (though nonlinear in $\mathbf{P}$ because of the denominators), $C(\mathbf{P})$ can be minimized using powerful numerical techniques developed for nonlinear least squares problems (Newton-Gauss method, Levenberg-Marquardt method). This can be done in complex terms, paying attention to maintain that the elements of $\mathbf{P}$ are real, or alternatively, Equation (8) can be written as a sum of squared real terms. Because both $N$ and $D$ are linear in $\mathbf{P}$, the cost function is insensitive to the multiplication of $\mathbf{P}$ by a scalar. Therefore, an additional constraint has to be introduced in order to obtain a well-defined solution: e. g., the norm of the vector $\mathbf{P}$, or the value of at least one nonzero parameter can be fixed.

In order to obtain starting values for the iteration, the sum of the numerators in Equation (8) can be minimized. This is an ordinary linear least squares problem, having a unique solution, and can be readily solved by standard procedures. However, this LS step does not provide any information on the delay $T_d$, thus an initial value has to be given by the user.

In the following sections the above described estimator will be referred to as ELiS (Estimator for Linear Systems).

## Input-Output Correlation

An even more general approach is to assume that the input and output noises, belonging to the same frequencies, may be mutually correlated. This correlation can be the result of different sources:

- The input signal of the system can be noisy. In this case the noise is no more an observation noise, since it excites the system under test, and the output noise is at least partly produced from the input noise. A preceding noise analysis, assuming observation noises only, gives too large input and output variances, and without introducing a correction, leads to an erroneous cost function. Considering input/output noise covariance, this error source can be corrected for.

- When the system under test is inside a feedback loop, the process noise is at least partly led back to the input, resulting in a noisy excitation[1].
- The excitation signal generator may produce slightly unstable amplitudes, a phenomenon that can be considered by an additive noise at the input.
- Slight synchronization imperfections of measurements may virtually increase the instability of Fourier coefficients, an effect that can also be well compensated for by considering the input/output correlation. (Strictly speaking, this is a kind of phase noise that may violate the complex Gaussian assumption. ELiS will not be a maximum likelihood estimate any more, but since it is also a least squares estimator, robust with respect to the noise distribution (see the "Key Features of ELiS" on page 2-10), it will still perform well.)

It can be shown that for consideration of the covariances, the cost function must be modified as follows:

$$C(\mathbf{P}) = \frac{1}{2}\sum_{k=1}^{F}\frac{\left|e^{-j\omega_k T_d}N(\Omega_k,\mathbf{P})X_{mk} - D(\Omega_k,\mathbf{P})Y_{mk}\right|^2}{\sigma_{yk}^2\left|D(\Omega_k,\mathbf{P})\right|^2 + \sigma_{xk}^2\left|N(\Omega_k,\mathbf{P})\right|^2 - 2\,\mathrm{real}\{CND_k\}} \quad \textbf{(14)}$$

with

$$CND_k = c_{xyk}e^{j\omega_k T_d}N(\Omega_k,\mathbf{P})D(\Omega_k,\mathbf{P}) \quad \textbf{(15)}$$

and

$$c_{xyk} = 0.5\,\mathrm{cov}\{N_{xk}, N_{yk}\} = 0.5E\{\overline{N_{xk}}N_{yk}\} \quad \textbf{(16)}$$

1. R. Pintelon, P. Guillaume, Y. Rolain and F. Verbeyst, "Identification of Linear Systems Captured in a Feedback Loop," *IEEE Trans. on Instrumentation and Measurement*, Vol. 41, No. 6, pp. 747-754, Dec. 1992.

## Covariance of the Estimate

The above estimator is a maximum likelihood one in the Gaussian case. Practice shows that the covariance matrix is usually close to the corresponding Cramér-Rao lower bound. Let us maintain the constraints Equation (7) during the derivations. In this case the Cramér-Rao bound is

$$\operatorname{cov}\{[\mathbf{X}_{\mathrm{ML}},\mathbf{P}_{\mathrm{ML}}]\} \geq E\left\{\left(\frac{\partial \mathrm{C}_{\mathrm{LS}}(\mathbf{X},\,\mathbf{Y}(\mathbf{X},\mathbf{P}),\,\mathbf{P})}{\partial[\mathbf{X},\mathbf{P}]}\right)^{\mathrm{T}}\left(\frac{\partial \mathrm{C}_{\mathrm{LS}}(\mathbf{X},\,\mathbf{Y}(\mathbf{X},\mathbf{P}),\,\mathbf{P})}{\partial[\mathbf{X},\mathbf{P}]}\right)\right\}^{-1} \tag{17}$$

where T denotes the transpose of the row vector, and $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{P}$ denote the true (exact) values.

Equation (17) has an alternative form:

$$\operatorname{cov}\{[\mathbf{X}_{\mathrm{ML}},\mathbf{P}_{\mathrm{ML}}]\} \geq \mathbf{E}\left\{\frac{\partial^2 \mathrm{C}_{\mathrm{LS}}(\mathbf{X},\,\mathbf{Y}(\mathbf{X},\mathbf{P}),\,\mathbf{P})}{\partial[\mathbf{X},\mathbf{P}]^2}\right\}^{-1} \tag{18}$$

which gives the very same values as Equation (16). The lower bound is asymptotically approximated when the number of observations is large, or when the signal-to-noise ratio is large.

Since we are interested in the covariance matrix of $\mathbf{P}_{\mathrm{ML}}$ only, it is desirable to bring Equation (17) or Equation (18) into a form that does not contain the covariances of $\mathbf{X}_{\mathrm{ML}}$. After a long derivation, the following expression can be obtained:

$$\operatorname{cov}\{\mathbf{P}_{ML}\} \geq \left(\frac{\partial^2 C(\mathbf{P})}{\partial \mathbf{P}^2}\right)^{-1}_{\mathbf{X}_m = \mathbf{X},\, \mathbf{Y}_m = \mathbf{Y},\, \mathbf{P} = \mathbf{P}_{true}} \tag{19}$$

An alternative form can be given as follows. The cost function can be expressed as

$$C(\mathbf{P}) \;=\; \frac{1}{2}\sum_{\mathrm{k}\,=\,1}^{\mathrm{F}} |\mathbf{EScm}(\mathrm{k})|^2 \;=\; \frac{1}{2}\mathbf{EScm}^{\mathrm{T}}\overline{\mathbf{EScm}} \tag{20}$$

with the elements of the error vector **EScm** being,

$$\mathrm{ESc}m(k) = \frac{e^{-j\omega_k T_d/2} N(\Omega_k,\mathbf{P}) X_{mk} - e^{j\omega_k T_d/2} D(\Omega_k,\mathbf{P}) Y_{mk}}{\sqrt{\sigma_{yk}^2 \left|D(\Omega_k,\mathbf{P})\right|^2 + \sigma_{yk}^2 \left|N(\Omega_k,\mathbf{P})\right|^2 - 2\,\mathrm{real}\{CND_k\}}} \tag{21}$$

see Equation (8) and Equation (14), and the second derivative is

$$\frac{\partial^2 C(\mathbf{P})}{\partial \mathbf{P}^2} = \left(\frac{\partial \mathbf{ESc}m}{\partial \mathbf{P}}\right)^T \left(\frac{\partial \mathbf{ESc}m}{\partial \mathbf{P}}\right) + \sum_{k=1}^{F} \mathrm{real}\left\{\frac{\partial^2 ESc m(k)}{\partial \mathbf{P}^2} \overline{ESc m(k)}\right\} \tag{22}$$

With the substitutions $\mathbf{X}_m = \mathbf{X}$, $\mathbf{Y}_m = \mathbf{Y}$ made, we will denote the noise-free error vector as **ESce**. This still depends on the parameter vector **P**. For $\mathbf{P}=\mathbf{P}_{\mathrm{true}}$ the elements of **ESce** are all zero, since the true values of the parameters satisfy the system equation, so the second sum in Equation (22) disappears, and Equation (19) becomes:

$$\mathrm{cov}\{\mathbf{P}_{ML}\} \geq \left(\left(\frac{\partial \mathbf{ESce}}{\partial \mathbf{P}}\right)^T \left(\frac{\partial \overline{\mathbf{ESce}}}{\partial \mathbf{P}}\right)\right)^{-1}_{\mathbf{P}=\mathbf{P}_{true}} \tag{23}$$

The problem with Equation (19) or Equation (23) is that the exact values $\mathbf{P}_{\mathrm{true}}$, $\mathbf{X}$ and $\mathbf{Y}$ are not known in practice. When the noise on the measured complex amplitudes is small, these expressions can be well approximated by leaving $\mathbf{X}_m$ and $\mathbf{Y}_m$ alone, and substituting $\mathbf{P}_{\mathrm{ML}}$ for **P**.

The above statements are true when the system under test can be perfectly modeled by the given model structure (rational form in *s*-domain or in *z*-domain, with given orders, maybe with a given delay). If this is not true (which is the case in a few practical cases when an approximate model is identified, e. g., for the approximate description of a distributed system), the estimate is not a maximum likelihood one any more, and the above variance expressions cannot be used. In such a case the approximate covariance matrix can be given as follows.

The uncertainty of the estimated parameters is due to the noise on the measurements. Random deviations from the mean values are due to the fact that the parameters are selected for each noise record to minimize the actual cost function *C*. Therefore, the variations of the parameter values directly depend on the variations of the complex amplitudes (through the cost function *C*), caused by the noise. For large signal-to-noise ratio let us develop the cost

function into Taylor series around the true values **X**, **Y** and the expected value of $\mathbf{P}_{\mathrm{ML}}$, in terms of the noise and in terms of the parameters, respectively. By comparing the two series up to the second-order terms (the first series is second-order, anyway), it is obtained that

$$\mathrm{cov}\{\mathbf{P}_{\mathrm{ML}}\} \approx \left(\mathbf{Q}^{-1}\left(\frac{\partial \mathbf{ESce}}{\partial \mathbf{P}}\right)^{T}\left(\frac{\partial \overline{\mathbf{ESce}}}{\partial \mathbf{P}}\right)\mathbf{Q}^{-1}\right)_{\mathbf{P}\,=\,\mathrm{E}\{\mathbf{P}_{\mathrm{ML}}\}} \tag{24}$$

with

$$\mathbf{Q} \;=\; \left(\frac{\partial \mathbf{ESce}}{\partial \mathbf{P}}\right)^{\mathrm{T}}\left(\frac{\partial \mathbf{ESce}}{\partial \mathbf{P}}\right) + \sum_{k\,=\,1}^{\mathrm{F}} \mathrm{real}\left\{\frac{\partial^2 \mathrm{ESce}(k)}{\partial \mathbf{P}^2}\overline{\mathrm{ESce}(k)}\right\} \tag{25}$$

The approximation of this expression is implemented in `elis`. Since the exact values are not known, $\mathbf{X}_m$, $\mathbf{Y}_m$, and the estimate $\mathbf{P}_{\mathrm{ML}}$ are used instead. The second term of Equation (25) will usually be small if no systematic modeling errors are present, thus it will not introduce a serious error, especially since the substitutions $\mathbf{Y} \sim \mathbf{Y}_m$ , etc., also mean approximations in the same order of magnitude. A heuristic argument is that E{**ESce**m} equals zero, and **ESce**m is small and more or less independent from the second derivative, so the summation effectively averages out the random terms.

Serious model errors will cause a significant difference between Equation (24) (`Cp`) and the approximation of Equation (19) (`CR`), this is why both quantities are calculated in `elis`.

## Key Features of ELiS

ELiS has been developed to handle the most important practical situations. First of all, it takes into consideration both input and output observation noise. These noises are present in every situation when both the input and the output signals are measured, e. g., quantization noise can be taken into consideration by ELiS very easily. Disregarding the input noise — a common practice in identification — can lead to severe modeling errors.

However, if the input observation noise is for some reason negligible, this can also be taken into consideration by setting $\sigma_{xk} = 0$.

The use of measured input and output signals has the advantage that only *relative* calibration of the measurement channels is necessary: as long as the transfer functions of the two channels are close to each other, their imperfections do not cause additional errors.

In the maximum likelihood approach it was assumed that the frequency domain noise is Gaussian. This is not a severe restriction, because signals are usually measured in the time domain, and it is easy to see that the DFT, which is the commonly used procedure to obtain complex amplitudes, leads to approximately Gaussian frequency domain noise, even if the time domain noise is not Gaussian. Moreover, as shown in Pintelon and Schoukens (1991), the properties of the estimate are robust with respect to the noise distribution[2].

ELiS as an estimate has further attractive properties. The estimator is asymptotically normally distributed, and it converges very well even for rather small signal-to-noise values. An important factor of the good convergence behavior is the well chosen initial LS step.

As a frequency domain method, ELiS is based on band-limited measurements of the input and the output signals, and can directly identify *s*-domain transfer functions. These measurements are quite easy to do with commercial measurement devices, and by avoiding the need of intermediate discrete-time identification, the systematic errors can be kept at a low level[3].

An important cause of small estimation errors is the improved signal-to-noise ratio in selected bands. There is however one important requirement: the frequency domain data must not exhibit systematic errors, otherwise the estimated transfer function can be biased. An important source of such distortions is *leakage*: because of the always limited time record length, the calculated digital spectrum is "smeared", unless the signal consists of integer periods of sine waves, or it is limited in time (transient signal). However, transient signals have usually worse signal-to-noise ratio, and the Fourier transform is subject to aliasing (if the signal is of limited duration, the spectrum theoretically cannot be band-limited). Because of this fact, ELiS should be used whenever possible with *periodic excitation signals,* in order to exploit its accuracy. Nevertheless, aperiodic signals can also be used for excitation, but the transfer function will be accurate in the order of leakage and aliasing only. With sufficiently high sampling frequency and record length, or by using advanced signal processing techniques (interpolated FFT and so on) both effects can be reduced to an acceptable level.

2. See also: I. Kollár, "On Frequency Domain Identification of Linear Systems," *IEEE Trans. on Instrumentation and Measurement*, Vol. 42, No. 1, pp. 2-6, Feb. 1993.

3. J. Schoukens and R. Pintelon, "Identification — Why do we need it, how to use it?" *Conference Record of the Instrumentation and Measurement Technology Conference IMTC/93*, 93CH3292-0, Irvine, Orange County, CA, May 18-20, 1993. pp. 246-251.

The “Excitation Signals for Identification in the Frequency Domain” on page 2-20 discusses the possibilities to design optimal excitation signals for the frequency domain identification procedure.

## Imposing Constraints on the Estimates

Sometimes certain properties of the transfer function are known. In these cases you may want to impose these properties as constraints to the estimate. In this section some possibilities for this will be discussed, as:

- Fixing some parameters
- Fixing some of the poles/zeros
- Maintaining known partial transfer function
- Fixing the value of the transfer function at certain frequencies
- Looking for special forms, like allpass or linear phase

### Fixing Some Parameters

In a few situations, some of the parameters can be set a priori to certain values. For example, in the *s*-domain, when the bandpass or highpass nature of the transfer function is known, some of the trailing coefficients of the numerator (belonging to $s^0$, $s^1$ etc.) should be set equal to zero. (The leading coefficients need not be set equal to zero for the bandpass/lowpass case, because the order on the numerator can simply be chosen lower.) In `elis`, the value of selected parameters can be fixed, using the input argument `fixp`.

### Fixing Some Poles/Zeros

This is a somewhat more complicated case, since poles/zeros can be fixed by means of the coefficients only if the whole numerator or denominator can be fixed. Therefore, the only solution is to *precompensate* the complex amplitudes to be fitted (and also the variance vectors) by the term formed of the fixed poles/zeros: instead of the equations

$$Y_{mk} = H_{\mathrm{fix}}(\Omega_k,\mathbf{P})H_{\mathrm{est}}(\Omega_k,\mathbf{P})(X_{mk} - N_{xk}) + N_{yk} \qquad \textbf{(26)}$$

the modified model

$$\frac{Y_{mk}}{H_{fix}(\Omega_k,\mathbf{P})} = H_{\mathrm{est}}(\Omega_k,\mathbf{P})(X_{mk} - N_{xk}) + \frac{N_{yk}}{H_{fix}(\Omega_k,\mathbf{P})} \qquad \textbf{(27)}$$

can be identified. The routine `modifyfv` generates the modified complex amplitudes and variances.

### Maintaining Known Partial Transfer Function

Sometimes a multiplicative term of the transfer function is a priori known, perhaps from earlier identification, or because it represents a well-designed building block of the system. In such cases the effect of this term can be removed from the measured data by using `modifyfv`. This is essentially the same case as fixing some poles and zeros.

### Fixing the Value of the Transfer Function at Certain Frequencies

The simplest case is when the lowpass, highpass or bandpass character of an *s*-domain transfer function is known: see "Fixing Some Parameters." However, in the *z*-domain this is not a solution any more. There may be other situations as well, when one would like to fix, e. g., the transfer function at dc to a given value. Using `elis`, there is a very simple solution for this: artificially set complex amplitude(s) are to be added to the Fourier vector, with the input and output variances set equal to a small value (e. g., `eps` or even smaller). If any of the variance values is set to a larger value, the constraint will be approximately followed; thus, for example, the maximum absolute value of the transfer function in a non-measured band can be controlled by introducing artificial zero transfer function points with large variance.

### Looking for Special Forms

Sometimes the transfer function is sought in a special form, like allpass, linear phase, imaginary (differentiator/integrator) etc. This happens when `elis` is used for filter design, as in the allpass design.

The allpass constraint in the *z*-domain is a built-in service of the basic `elis` routine. However, the other cases enumerated above will usually be well approximated, if the amplitude vector is given in the appropriate form. For example, linear phase FIR filters, fitted in LS sense, can be designed by setting the delay to (order/2), and defining real amplitudes for fitting.

## Solutions for Some Special Cases

It may happen that the available measurement setup does not completely correspond to the assumptions of ELiS, or you experience difficulties fitting your data. The purpose of this section is to provide some hints for such cases.

### Dealing with Data from a Network Analyzer (Dynamic Signal Analyzer)

When input-output measurements are made by a digitizer, the input-output Fourier amplitudes can be determined and used for estimation. However, you may already have an analyzer that provides the measured transfer function points ($Y_{mk}/X_{mk}$) at given frequencies, usually on a linear grid from zero to about half of the sampling frequency, and a parametric model of the system is sought.

Theoretically, if you have to rely on the above transfer function estimate points only, the noise on $X_{mk}$ may introduce an annoying bias through the division. Variance can be reduced by simple averaging, but this will not remove this kind of bias. You may consider taking the complex geometric mean of estimates (see `gmean`), which has a smaller bias.

Having assured a possibly small bias of the transfer function points, you may try to use `elis`, accepting limited accuracy. The input amplitudes can be all set to ones, and the output ones can be set to the complex transfer function points.

A more or less acceptable result can be achieved by setting zero input and uniform output variances, but without a correct absolute variance value the actual value of the cost function will provide no information about the quality of the fit and about the bias actually introduced. A plot of the complex residuals (see `rdueelis`) can give some idea about the trends in the fitting error. Strongly correlated complex error is an indication of modeling errors.

However, you can do somewhat better. If repeated measurements can be done, you should consider variance analysis using `varanal`, even if only a few data sets are available. In such a case the cost function and the mean model error can be used for characterization of the bias.

If repeated measurements are not available, but you have some knowledge about the input and output noises, `stdtfm` can be used. For this, you should know not only the forms of the spectra of the input and output noises, which are often white if quantization noise dominates, but also their absolute levels or at least their levels relative to each other, to be able to combine them into

`stdAm`. Proper noise variances may improve the accuracy of your estimates (with the above limitations due to the bias of the transfer function points).

Extensive bias can cause that `elis` tends to "concentrate" poles/zeros to given frequency bands. Quite often, people incline to increase emphasis to other frequency parts by repeating transfer function points or by artificially increase weighting in these bands, decreasing the variance values passed to the estimation algorithm. This is not a good practice. The general recipe is to make better measurements by designing optimized excitation signals, improvement of the signal-to-noise ratio, by provision of input-output Fourier amplitudes instead of transfer function points, and so on. Or, if all these are not practicable for some reason, partial subband fits can be tried out, as described below.

### Wide-Band Model Fitting

When a system has to be modeled in a very wide band (the frequency range stretching to several orders of magnitude), the cost function can have a very complicated surface, and the iteration algorithms may iterate to unacceptable local minima. In such cases it is often possible to select subbands of the frequency range where fitting by a lower-order subsystem seems to be reasonable. Such "subfits" have to be made with care, because the effect of such subsystems can be quite intensive in neighboring bands, so the combinations of independent subsystems can provide a very poor overall fit. A better strategy is to use `modifyfv` after each fit, removing the effect of the subsystems already obtained. By this strategy a more or less acceptable compound system can be obtained with poles/zeros in every band of importance. Finally, the subsystems can be refitted cyclically using the residuals of the other fits, or the whole compound transfer function can be polished using the combination as a starting value, and using Levenberg-Marquardt iteration in order to assure decreasing steps of the cost function during iterations. In very wide bands, however, the evaluation of the transfer functions, based on `polyval`, may become inaccurate, so all the results should be used with precaution.

## Numerical Stability and Speed of the Procedures

When performing simple calculations, MATLAB has virtually infinite precision. However, for larger equation sets, as for the frequency domain identification of systems of order 15-20 or higher, finite precision may play an important role, and the results may become unreliable if `elis` is not applied with proper care.

The speed of calculations has been dramatically increased with the development of modern CPU-s. However, the calculation time of certain iterations is still not negligible. By proper use of the built-in possibilities of the functions of this toolbox, the speed can be remarkably increased.

This subsection discusses the above aspects of the usage of the toolbox.

## Numerical Stability

In `elis`, often a large set of linear equations is solved in every iteration step, in order to find the necessary correction of the parameter vector. In the $s$-domain the parameters are coefficients of polynomials of the variable $s$. It is easy to see that when having roots around $10^3$ Hz, the coefficients of higher powers of $s$ will be very small, typically in the order of $10^{-3n}$ for the coefficient of $s^n$. From the numerical point of view, to handle parameters of different orders of magnitude in parallel is very disadvantageous.

A measure of the numerical behavior of the equations is the condition number of the matrix in the basic equation to be solved. The condition number is the ratio of the maximum and minimum singular values of this matrix, and it should not reach the order of magnitude of `1/eps` on the given machine.

In the toolbox functions, and especially in `elis`, an attempt is made to avoid numerical problems. If this cannot be maintained, warning messages are sent, describing the concrete cause. It is therefore advisable to take each warning message seriously, and try to find a way of better conditioning. Some hints are given below.

In some cases `elis` cannot "catch" the problems in time, and the internal routines of MATLAB will send messages like

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.617346e-17
```

or

```
Warning: Divide by zero
```

These messages usually warn about bad conditioning.

In order to avoid numerical problems, *scaling* is introduced in `elis` for $s$-domain calculations. According to experience, the equations behave the best when the majority of the poles/zeros have their magnitudes around 1.

However, when starting identification, no usable information is at hand about poles/zeros. This is why in `elis` a simple rule of thumb is used, based on the excitation band, which is available from the Fourier data. The center radian frequency of the excitation band:

$$f_s = \frac{\omega_{\min} + \omega_{\max}}{2} \tag{28}$$

is applied, supposing that a reasonable excitation will inject the energy in the band of interest, where most of poles and zeros are located.

After having made a more or less acceptable fit, a better guess of the optimum scaling frequency can be made. `elis` will display a suggestion for a new scaling frequency after the last iteration, if the ratio of the actual one and the suggestion is larger than 2 or smaller than 0.5. In the case of a system of order 20, a change of factor 2 in the scaling frequency may mean a change of $2^{20} \approx 10^6$ in the condition number! However, the decision is left to the user; the new scaling frequency can be set for `elis` in a repeated run. The suggestion is made either on the basis of the mean of the absolute values of all the poles and zeros, or if these are not calculated in `elis`, on the basis of the ad hoc formula in `exppar` (see in the *Reference* chapter). This latter value is accessible in the workspace after the run of `elis` as `pvect(2)`.

Bad conditioning can result from a few other causes, too. Its basic meaning is that for the algorithm no sufficient information is provided about the parameters, and/or the machine precision is not sufficient for the treatment of the given problem. This can mean, for example, that the measurement bandwidth is not wide enough, and we try to let `elis` extrapolate the behavior of the system far beyond the band limit of our measurement. In such a case, no better advice can be given than repeating the measurements in a wider band. The treatment of high-order, wide-band systems may sometimes also require high machine precision, even with proper measurement design.

Bad signal-to-noise ratio also results in bad conditioning: if this is the cause, averaging can be suggested, or the redesign of the whole experiment.

A further cause can be overmodeling: the algorithm has no information for the placement of the “nonsense” poles/zeros, and this fact is reflected in bad conditioning. Decrease of the order may immediately improve the condition number. Even undermodeling might cause bad conditioning, if the model is far

too simple even to roughly describe the system. Improper fixing of nonzero parameters can also increase the condition number: the default setting (no fixed parameters) is usually optimal with respect to the condition number.

`elis` provides three condition numbers in the vector `fit`.

The most relevant information about the conditioning of the solution is provided as "the condition number of the matrix actually decomposed or inverted in the last iteration." The description is somewhat general, but this is the way to have a number relevant for all the algorithms. For reliable results, this number should be well below `1/eps`.

However, the above condition number may depend on the applied iteration algorithm. Newton-Raphson usually has the worst condition number, so if bad conditioning is a danger, it should be avoided. The other extreme is Levenberg-Marquardt (without svd): if lambda is large enough (the value 0.1 is sufficient in general), the equations are well conditioned, but the iteration may be painfully slow.

Most of the iteration schemes operate on the Jacobian **J** of the error vector. The condition number of **J** is therefore relevant to the problem itself. (It should be mentioned, however, that also scaling and measurement band selection influence conditioning.) Should the condition number of **J** be too large, it is better to rescale or to make new measurements.

The so-called approximate covariance matrix (`Cp`) is obtained by inverting the approximate Hessian (see Equation (24)). Since this matrix is usually close to $\mathbf{J}^T\mathbf{J}$, its condition number equals approximately the square of the condition number of **J**. As a consequence, the approximate covariance matrix may be unreliable even if the solution is good. In such cases, when no or small modeling error is anticipated, the approximate Cramér-Rao bound can be used instead (`CR`), because this is calculated by singular value decomposition of **J**.

## Speed of the Procedures

The speed of the calculations depends on two important factors: the number of iterations performed, and the time spent on each iteration.

The number of iterations can be influenced by the starting values, the iteration method, and the stop criteria, but depends also on the investigated problem and the selected model order.

Most iterating functions in this toolbox have a good default setting of the *starting values,* thus these should be changed with care. An example for such a justified decision is when a more or less reasonable estimate is at hand, and iteration is used primarily for polishing these values. This can happen when a very wide-band fit (covering several orders of magnitude) is being done. In such cases it can be extremely difficult to find the global minimum (or a reasonable local minimum) of the cost function. A strategy could be to fit certain bands separately with low-order subsystems, modify the Fourier and variance data after each partial fit using `modifyfv`, and at the end, "polish" the compound system in a global fit.

There is a wide choice of *iteration methods* in `elis`. As a general rule it can be stated that the Newton-Gauss type iteration schemes (Newton-Gauss and singular value decomposition) have about the same rate of convergence. Newton-Gauss is quicker, but numerically less robust than svd. Near to the optimum, Newton-Raphson is usually the quickest, but far from the optimum it behaves worse than other algorithms. The Levenberg-Marquardt algorithms are the most robust, but they converge rather slowly. With proper setting of `rpalg` an attempt can be made to set lambda to zero (switch to the Newton-Gauss algorithm) after a few successful iteration steps, which can accelerate convergence. If the Newton-Gauss step is not successful, the Levenberg-Marquardt iteration continues.

The setting of the *stop criteria* can also be decisive. According to experience, the default setting (relative change of the cost function is smaller than $10^{-6}$) is appropriate for most cases. A too small prescribed value may prevent stopping, since because of numerical roundoff the derivatives will never exactly equal zero.

The selection of the model order can also seriously influence the number of necessary iteration steps. A too high model order means "nonsense" poles or zeros, for which the measured data do not provide information. The iteration often wanders around rather flat portions of the cost function surface.

The cycle time is also an important factor of calculation speed. In identification, much depends on your skill, and we made an attempt to provide you with as much graphics information as possible. But the sophisticated graphics procedures may cost a lot of time. In most iterative algorithms the plots can be made more rarely than in every cycle, or can even be completely suppressed. This can dramatically improve the speed of `elis`, `dibs`, `msinclip` and `optexcit`, but total elimination of plots can be extremely dangerous, since this will mean "blind" acceptance of the results of numerical methods that might sometimes even diverge.

# Excitation Signals for Identification in the Frequency Domain

As mentioned previously, frequency domain methods work best if periodic excitation is applied. Nevertheless, there is still a lot of freedom left. One can select the harmonic frequencies where the system is to be excited, and also the amplitude and the phase of the sinusoids. Thus, criteria are necessary for the correct decision.

The final aim is clear: the parameters of the transfer function are to be estimated with as small error as possible. However, practical limitations may be present: an arbitrary waveform generator is at hand or not, the actuator in the system may allow continuous amplitude changes, or it may be an on-off relay, the measurement equipment usually has limited amplitude resolution, etc.

Roughly speaking, as much energy is to be injected at the "interesting" frequencies, as it is possible under the given restrictions. Thus, the first task is to select these frequencies, and determine the optimal power distribution[4,5,6].

If at least a rough model of the system is at hand, the volume of the information matrix can be maximized, by distributing the energy among the frequency points. This can be done by using the routine `optexcit`. However, this maximization is to be done with care, since after some iteration steps the information only slightly increases, but the spectrum becomes spiky, and the possibility of crest factor minimization (see the following "Multisine" on page 2-21) becomes limited.

A less systematic approach is to try to localize the "sensitive" frequency bands (where the transfer function has a large value, or it changes rapidly, or it has its important poles/zeros). Here users are left to their own skill and knowledge about the system.

4. J. Schoukens and R. Pintelon, *Identification of Linear Systems: a Practical Guideline for Accurate Modeling,* London, Pergamon Press, 1991.

5. K. R. Godfrey, ed.: *Perturbation Signals for System Identification*. Englewood Cliffs, Prentice-Hall, 1993.

6. F. Delbaen, "Optimizing the Determinant of a Positive Definite Matrix," *Bulletin Société Mathématique de Belgique — Tijdschrift Belgisch Wiskundig Genootschap*, Vol. 42, No. 3, pp. 333-346.

While in MATLAB it is easy to define a linear grid, is not that simple to design a quasi-logarithmic one (near to logarithmic, on the linear grid of the FFT). The routines `lin2qlog` and `log2qlog` serve this purpose.

The following subsections deal with the two most important cases: the first one is when there is no restriction imposed on the time domain signal form, i. e., a general multisine can be used, the other one is the case of a binary excitation signal.

## Multisine

Having determined the desired power distribution, users can still choose the phases. The phases of the components of a multisine have important influence on the time domain signal shape: by proper choice of the phases the maximal peak can be significantly compressed, allowing larger energy to be injected for the given input range of the measurement device, or keeping the system in the linear working region. This procedure is called *crest factor minimization*. The crest factor is defined as the ratio of the maximal absolute peak value to the effective value of the signal in the frequency band of interest:

$$cr = \frac{\max\{\mathrm{abs}(x(t))\}}{V_{\mathrm{RMSeff}}} = \frac{\max\{\mathrm{abs}(x(t))\}}{V_{\mathrm{RMS}}\sqrt{\frac{\text{in-band power}}{\text{total power}}}} \quad \textbf{(29)}$$

The achievable decrease in the crest factor is a factor of 2-3 for nearly uniform spectra, compared to the crest factor of a random phase sine wave.

Crest factor minimization can be performed on a single signal, or on two signals related to each other by a linear system (input-output optimization). Both tasks are implemented in the routine `msinclip`. This routine is based on consecutive transforms from the time domain to the frequency domain and vice versa, applying a mild clipping in the time domain, and combining the new phases with the desired amplitudes in the frequency domain[7].

The optimized signal can be applied to the system, using an arbitrary waveform generator. This usually works with a zero-order hold (a D/A converter), which introduces some amplitude distortion:

7. E. van der Ouderaa, J. Schoukens and J. Renneboog, “Peak Factor Minimization, Using Time—Frequency Domain Swapping Algorithm,” *IEEE Trans. on Instrumentation and Measurement*, 1988, Vol. 37, No. 1, pp. 144-147.

$$H_{zoh}(f) = \frac{\sin(\pi f \Delta t)}{(\pi f \Delta t)} \tag{30}$$

where $\Delta t$ is the reciprocal of the clock frequency. This distortion increases towards the half of the clock frequency: here the value of the transfer function is approximately $2/\pi \approx 0.6366$, instead of 1 (see Figure 2-2).

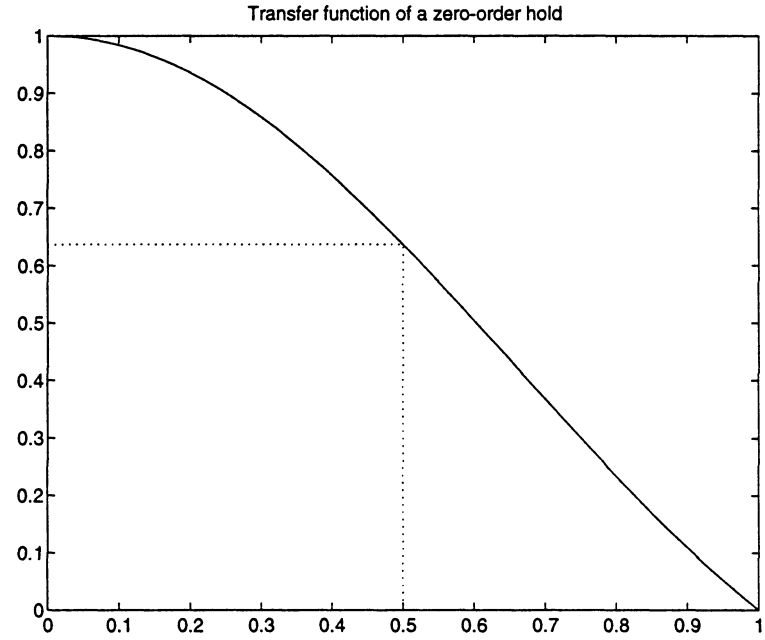



**Figure 2-2: The amplitude distortion introduced by a zero-order hold**

The routine `msinprep`, which generates a time series for the arbitrary signal generator, can perform a *precompensation* for this distortion. This prepares the data vector to be downloaded in to the arbitrary signal generator, producing the desired waveform (if no other amplitude or phase distortion is introduced in the chain).

## Binary Excitation Signals

There are situations when only binary excitation signals can be used. Theoretically the state transition instants can be anywhere within the signal period, but in practice these signals are produced using a high-frequency clock.

Thus, users can choose the clock periods where the state is inverted (discrete interval binary sequence). This does not achieve exactly the desired power distribution, but the approximation is generally usable.

In order to approximate the desired spectrum, a similar algorithm, implemented in `msinclip`, can be applied. The difference is that instead of clipping, a comparator is used in the time domain, resulting in a binary signal[8]. This algorithm, implemented in the routine `dibs`, converges rapidly, but in the result there may be frequencies where only a fraction of the desired power is present. To improve this property, a systematic search can be performed using the routine `dibsimpr`, which tries to maximize the minimum relative power (calculated with respect to the desired power at a given frequency)[9].

If an approximately white spectrum is acceptable, the so-called maximum length binary sequences (or pseudorandom binary sequences, PRBS)[10] can be also used (routine `mlbs`). These sequences are produced using binary shift registers with appropriate feedback. Since `mlbs` can be considered as a pseudorandom sequence applied to a zero-order hold, it is easy to see that the amplitude spectrum is modified again by the transfer function depicted in Figure 2-2.

This spectrum is valid for the continuous case; however, because of the rich overharmonic contents, the use of an anti-aliasing filter in the measurement setup is advisable to avoid undesirable spectral distortions due to aliasing.

The length of the sequence is $N = 2^n$-1, where $n$ is the register length, thus with synchronized sampling the length of the time record is not a power of two. Consequently, instead of standard base-2 FFTs, some special DFT algorithm has to be used (see the section “Transformation into the Frequency Domain”).

However, there is also another possibility: by careful adjustment of the sampling frequency, the number of samples in a period of the excitation signal can be set to a power of two, thus a standard FFT can be applied.

8. A. van den Bos and R. G. Krol, “Synthesis of Discrete-Interval Binary Signals with Specified Fourier Amplitude Spectra,” *International Journal of Control*, 1979, Vol. 30, No. 5, pp. 871-884.

9. K.-D. Paehlike and H. Rake, “Binary Multifrequency Signals — Synthesis and Application,” *Proc. 5th IFAC Symposium on Identification and System Parameter Estimation*, Darmstadt, FRG, Sept. 24-28, 1979. Vol. 1, pp. 589-596.

10. K. R. Godfrey, ed.: *Perturbation Signals for System Identification*. Englewood Cliffs, Prentice-Hall, 1993.

# Preprocessing of Data

Before starting the estimation procedure, the measured data have to be prepared. Though ELiS works well for rather small signal-to-noise ratios, the quality of the estimate can be improved by decreasing the measurement noise. Thus, for precise estimations averaging is usually recommended. Moreover, the noise often has to be analyzed in order to determine the necessary variance values; variance analysis can be done in parallel with averaging.

According to experiences with ELiS, the measurement of a few periods is often sufficient for obtaining a usable estimate. By segmenting the records into equal-length parts which contain full periods of the periodic excitation signal, an approximate noise analysis can be performed, and then the estimation can be performed on the averaged data.

## Preprocessing in the Time Domain

Averaging can be done directly in the time domain. For this, the measurements have to be started in synchronization with the excitation signal. To achieve perfect synchronization, the clock of the measurement equipment and that of the signal generator have to be synchronized (it is not enough to have very stable clocks, because they may slip slowly with respect to each other). With no proper synchronization, time domain averaging and noise analysis are hopeless; however, as shown in the next subsection, in the frequency domain there is a quite effective way to “synchronize” the measurement records even if the measurements themselves were not synchronized.

The time domain variance can be determined in parallel with the averaging. Since the noise on the signals is assumed to be ergodic, the following formula can be applied:

$$\hat{\sigma}_{xt}^2 = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{M-1}\sum_{m=1}^{M}\left(x_m(n\Delta t) - x_{av}(n\Delta t)\right)^2$$

$$= \frac{1}{N}\sum_{n=1}^{N}\frac{1}{M-1}\sum_{m=1}^{M}\left(x_m^2(n\Delta t) - Mx_{av}^2(n\Delta t)\right) \qquad \textbf{(31)}$$

where $M$ is the number of measurements, N is the number of measured points in each measurement, and $x_{av}$ denotes the result of averaging over different

measurements. This estimate is c2 distributed, with r = N(M-1) degrees of freedom. The confidence interval for the confidence limit a can be obtained using the corresponding c2-values (see also Table 2-1):

$$P\left(\frac{N(M-1)}{\chi^2 N(M-1);\alpha/2}\hat{\sigma}_{xt}^2 < \sigma_{xt}^2 < \frac{N(M-1)}{\chi^2 N(M-1);1-\alpha/2}\hat{\sigma}_{xt}^2\right) = \alpha \quad \textbf{(32)}$$

The $\chi^2$-values can be taken from statistical tables.

For $r = N(M-1) > 30$ and small $\alpha$, a useful approximation[11] for the terms in Equation (32) is:

$$\frac{r}{\chi_{r;\beta}^2} = \frac{1}{\left(1 - \frac{2}{9r} + u_\beta\sqrt{\frac{2}{9r}}\right)^3} \quad \textbf{(33)}$$

where $\beta = \alpha/2$ or $\beta = 1-\alpha/2$, and $u_\beta$ is the corresponding abscissa of the standard normal distribution.

**Table 2-1: Confidence intervals for the variances (multipliers of the empirical variance)**

| r | 90% | | 95% | | 98% | |
|---|---|---|---|---|---|---|
| 10 | 0.55 | 2.54 | 0.48 | 3.12 | 0.43 | 3.91 |
| 20 | 0.64 | 1.84 | 0.58 | 2.10 | 0.53 | 2.42 |
| 30 | 0.69 | 1.62 | 0.63 | 1.80 | 0.59 | 2.01 |
| 40 | 0.72 | 1.51 | 0.67 | 1.64 | 0.63 | 1.81 |
| 50 | 0.74 | 1.44 | 0.70 | 1.55 | 0.66 | 1.69 |
| 100 | 0.80 | 1.28 | 0.77 | 1.35 | 0.74 | 1.43 |
| 200 | 0.85 | 1.89 | 0.83 | 1.23 | 0.80 | 1.28 |

11. G. A. Korn and T. M. Korn, *Mathematical Handbook for Scientists and Engineers: Definitions, Theorems and Formulas for Reference and Review*, 2nd ed, London, McGraw-Hill, 1968.

**Table 2-1: Confidence intervals for the variances (multipliers of the empirical variance)**

| r | 90% | | 95% | | 98% | |
|---|---|---|---|---|---|---|
| 500 | 0.90 | 1.11 | 0.89 | 1.14 | 0.87 | 1.16 |
| 1000 | 0.93 | 1.08 | 0.92 | 1.09 | 0.90 | 1.11 |

If you assume that the noise spectrum is approximately white, the time domain variance data can be directly used for the calculation of the variances of the real and of the imaginary parts of the complex amplitudes:

$$\hat{\sigma}_x^2 = \frac{N}{2}\hat{\sigma}_{xt}^2 \tag{34}$$

However, when the noise spectrum is to be determined, and/or the input-output noise covariances have to be calculated, the auto- and cross-correlation functions have to be estimated. This is usually done via the frequency domain, using FFTs; in this case it is better to do averaging and noise analysis in the frequency domain.

## Transformation into the Frequency Domain

The standard procedure of time domain to frequency domain conversion is the fast Fourier transform (FFT). In the case of periodic signals this will not introduce spectral leakage if the time record contains integer numbers of periods of each component. Leakage can be detected by throwing a glance at the frequency spectrum, if the signal contains a few spectral lines only: it appears as “skirts” around spectral lines (Figure 2-3).

```
N = 100; t = [0:N-1]; %N = 100 points
%two sine waves:
xt = sin(2*pi*10/N*t)+0.5*sin(2*pi*20/(N-1)*t);
Xfa = abs(fft(xt)); %spectrum
Xsh = [zeros(1,N);Xfa;zeros(1,N)];
Xsh = [0;Xsh(:);0]; %prepare for “bar” plot
tsh = [t;t;t]/N; tsh = [0;tsh(:);1]; plot(tsh,Xsh)
```

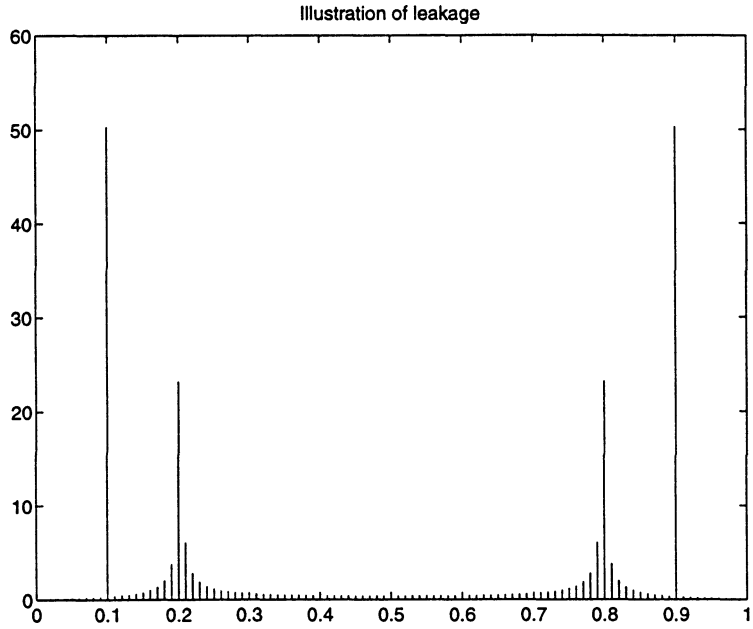



**Figure 2-3: Absolute value of the DFT of two sine waves**

The first sine wave ($f_1$ = 10/100 = 0.1) exhibits no leakage because exactly ten periods have been measured, while the second one ($f_2$ = 20/99 ≈ 0.2) shows considerable leakage.

The FFT is base $N$, where $N$ is the record length. If $N$ is not a power of two, special techniques (like the chirp $z$-transform[12]) have to be applied. MATLAB's `fft` routine can effectively transform sequences of arbitrary length. The applied formula is

$$X_k = \sum_{i=1}^{N-1} x_i e^{-j2\pi\frac{ki}{N}}$$

12. L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Englewood Cliffs, NJ, Prentice-Hall, 1975, pp. 393-399.

## Preprocessing in the Frequency Domain

When the measured input and output signals have been transformed into the frequency domain, averaging and noise analysis (including the estimation of the input/output covariances) can be performed using the routine `varanal`. This routine provides the results in the form necessary for `elis`, and even calculates the 95% confidence intervals of the variances. As a rule of thumb, input-output covariances should be calculated and passed to `elis`. If they are small, they make no harm; if they are large, they have to be considered.

### Post-Measurement Synchronization

The routine `varanal` can, if required, make an attempt to "synchronize" the measurements to the first one, by looking for the delay which results in the smallest phase differences. The routine works well even for as small as 3 dB frequency domain signal-to-noise ratios. If the synchronization attempt fails, it sends an error message.

The synchronization algorithm in `varanal` is based on an approximation of the maximum likelihood estimation of the delay[13]. It is worth mentioning that for the maximum likelihood estimation of the delay between records `elis` can also be used, identifying a hypothetical pure delay transfer function between the two records, considering them as input and output.

A rough guess of the shape of the transfer function can be quickly obtained by averaging the transfer function estimates, calculated as the ratio of the measured output and input complex amplitudes. Since the bias is much smaller for the geometric mean of the transfer function estimates than for the simple arithmetic one[14,15], the geometric mean is to be used for averaging. The routine `gmean` calculates this mean value for complex numbers, without suffering from phase wrapping problems.

13. I. Kollár, "Signal Enhancement of Non-Synchronized Measurements for Frequency Domain System Identification," *IEEE Trans. on Instrumentation and Measurement*, Vol. 41, No. 1, pp. 156-159, Feb. 1992.

14. J. Schoukens and R. Pintelon, "Measurement of Frequency Response Functions in Noisy Environments," *IEEE Trans. on Instrumentation and Measurement*, Vol. 39, No. 6, pp. 905-909, Dec. 1990.

15. P. Guillaume, R. Pintelon and J. Schoukens, "Nonparametric Frequency Response Function Estimates Based on Nonlinear Averaging Techniques," *IEEE Trans. on Instrumentation and Measurement*, Vol. 41, No. 6, pp. 739-746, Dec. 1990.

### A Priori Known Partial Transfer Function

Sometimes a term of the transfer function is already known (e. g., from earlier identification or because it represents a block which was carefully installed to meet strict specifications). It is not worthwhile to identify this term again, since this may deteriorate the variance of the estimates of other parameters. The routine `modifyfv` can be used to precompensate Fourier and variance data.

## Presentation of the Results

Identification is a science and an art at the same time. Much depends on the skill and imagination of the person, who identifies a given system. This toolbox seeks to provide as much illustrative information as it is possible during the whole procedure.

Therefore, during the most important procedures of the Frequency Domain System Identification Toolbox, it is possible to follow the iterations on graphs. Though this "movie" can be switched off, to accelerate the run, we recommend that the user follows the iterations, and intervenes if necessary. Nothing is more dangerous than to blindly believe the results of programs, even if they are well tested and reliable.

In addition to the graphs, there are two plotting programs for the illustration of the result of `elis`.

`ploteltf` plots magnitudes and phases of model data. It can plot the transfer function of two parameter sets and, in addition, the data given in a Fourier file, in order to facilitate comparisons. Moreover, on one of the transfer functions confidence intervals can also be plotted, to give an insight into the reliability of the estimation results.

The uncertainty of the estimated transfer function is usually much smaller than the scattering of the $Y_{mk}/X_{mk}$ points. This has to be so, since the estimated transfer function is based on all measured data, and like that, its variance is roughly inversely proportional to the number of measurement points, while the variance of the points $Y_{mk}/X_{mk}$ depends only on the input and output amplitudes and measurement noises. On the other hand, almost all of the confidence intervals of the nonparametric transfer function estimates $Y_{mk}/X_{mk}$ should cover the parametric transfer function estimates calculated from `pdat1` or `pdat2`.

The confidence interval plots are made by internally using the results of the routines `stdtf` and `stdtfm`. When nonstandard forms of plots are preferred, these routines can be directly used for the calculation of the standard deviations of the magnitudes and phases.

If possible, we suggest you plot not just one amplitude set, but the result of averages. If for some reason synchronization failed even with the routine `varanal`, it is recommended to average the ratios of output and input amplitudes via the routine `gmean` (see Section 4.3).

`plotelpz` plots pole/zero patterns of identification results, along with the uncertainty ellipses. This is an often studied plot of dynamic systems.

However, this plot does not provide information on the coupling of poles and zeros. The standard deviations and covariances of the poles and zeros can be directly calculated by the routine `stdpz` for the preparation of special-form plots. Coupling between poles and zeros can be detected from high-value cross-correlation terms in the output argument `rzp`.

Both plotting routines have a number of optional arguments. Besides the usual purposes, they can be used as sophisticated statements in script files, preparing graphs for documentation, archiving and illustration of papers.

Only a small part of the demands can be covered by standard routines. Since the Frequency Domain System Identification Toolbox uses the standard MATLAB representations of *s*- and *z*-domain polynomials, script files can be easily written to show other possible plots, as Nyquist, Nichols diagrams etc.

To obtain high-quality illustrations for scientific papers, etc., the plots made in MATLAB can be saved in PostScript form. Data vectors can be stored into ASCII files, using the routine `expvect`, for export to sophisticated graphing programs.

# Model Validation and Simulations

A basic rule for scientists and engineers using powerful computers is to never believe the results of any numerical procedure, unless these can be checked in an independent way. This is especially true for identification, where even the right order of the system model is unknown. Therefore, model validation is crucial. In this section the validation possibilities will be summarized, in the Frequency Domain System Identification Toolbox and in the environment of MATLAB.

## The First Quick Checks

The best thing one can do is to observe the internal iteration steps of the procedures on the presented graphs, and to plot the transfer function and the pole/zero pattern of the result with the uncertainties (see the "Presentation of the Results" on page 2-30). From these plots rough errors (instability, roughly bad fit, outliers, etc.) can be easily detected.

## Stability and the Choice of the Proper Delay

Instability of the fitted models is often a problem. The studied physical systems are usually stable (otherwise measurements cannot be done, except if the unstable system is stabilized by a feedback loop), thus for our purpose unstable models are usually not acceptable.

Unstable identified model of a stable system can be due to several causes, like

- Overmodeling or undermodeling (too large or too small model orders)
- Improperly chosen value of the delay
- Outliers
- Too small signal-to-noise ratio
- Local minimum of the cost function

Overmodeling and undermodeling are going to be discussed in the next section.

Improper choice of the delay is a common source of instability. The reason is that a too large or a too small delay value drastically changes the phase behavior of the identified rational form, and thus can easily deteriorate the fitting.

One possibility is to let also the delay be estimated. This is an attractive idea, however, especially for high-order systems, the convergence region of the delay is narrow, thus the estimated value will strongly depend on the starting one.

One can make fits with different delay values around the a priori guess. Usually the fits become better when approaching the true value.

Another possibility is the study of the estimation results. In the $z$–domain too small delay values usually result in small values of the leading coefficients of the numerator[16]. In both domains, a too small delay value often gives unstable models. The gradual increase of the delay can lead to the good estimate.

The rest of the causes can usually be recognized by looking at the transfer function or pole/zero plots, estimation with different (eventually more effectively averaged) measurements, or eventually by estimations on simulated data.

## Detection of Undermodeling and Overmodeling

*Overmodeling* means that the model order is too high, while undermodeling means that it is too small. Both should be avoided. However, in practical situations two important limitations should be considered. First, a linear transfer function is often an approximate model only of the physical system: some small nonlinearities, some distributed parameters, etc., may always be present. Therefore, we usually have to content ourselves with a limited modeling error of our identification result. Second, it cannot be expected to decrease modeling errors significantly below the random errors, unless new measurements with smaller noise (more averaging, etc.) are done.

*Undermodeling* is treated in detail in Chapter 5 of the book of Schoukens and Pintelon (1991). Here only the most important statements will be summarized.

The best indicator of a bad model is the too large value of the cost function. As it was established in Section 2, in the case of a good model the double of the cost function is a random variable of $\chi^2$-distribution, with $2F–n_p$ degrees of freedom. Consequently, the standard $\chi^2$-test can be applied: if the value is too large, modeling errors are present.

However, not only undermodeling can result in a large value of the cost function. A wrong value of the delay, or incorrectly small variance values can also increase the value of it.

16. L. Ljung, *System Identification Toolbox for Use with Matlab. User's Guide,* July, 1991. The MathWorks, Inc. p. 1-64.

If modeling errors are present, it is often desirable to obtain some information about the extent of these errors. The so-called *mean model error* may provide a rough measure. For an approximately constant transfer function value $H_e$ (the subscript e refers to the exact values), approximately constant excitation signal amplitudes, and approximately flat noise spectra with variances of the real and the imaginary parts $\sigma_x^2$ and $\sigma_y^2$

the mean model error is

$$|h_{\text{mean}}|^2 \approx \frac{2C-(2F-n_p)}{F|X|^2}(|H_e|^2\sigma_x^2+\sigma_y^2-2\text{real}\{c_{xy}\overline{H_e}\}) \tag{35}$$

where $h = H - H_e$ is the complex modeling error of the transfer function, $C$ is the value of the cost function, $F$ is the number of frequencies, and $n_p$ is the number of estimated parameters. For non-uniform $H_e$, the mean modeling error can be averaged over the frequencies used in the estimation, as it is done in `elis`, but the result has to be interpreted with precaution.

Using $h_{\text{mean}}$, special care has to be taken to use correct variance values, since these directly scale the value of $C$. Too large variance values lead to a too small cost function (and maybe to an imaginary $h_{\text{mean}}$), too small variance values result in a too large cost function and a large mean model error.

A confidence interval plot of the magnitude and phase of the estimated transfer function may provide important information. The standard deviations of the magnitude and phase are also accessible using `stdtf`.

When *overmodeling* occurs, the estimation procedure is forced to find additional poles/zeros which are not relevant to the measured data. This will increase the condition number of the normal equations (the condition number is accessible as an output argument of the routine `elis`). Moreover, the variance of the estimated parameters will also increase, which can be detected easily if additional estimations are done on different measurement data.

For a too large number of parameters, much more iteration steps are necessary to reach the minimum of the cost function.

It is even easier to detect overmodeling using the pole/zero plot, preferably with the uncertainty ellipses. The “superfluous” poles and zeros will have a large uncertainty, poles and zeros often appear in pairs (nearly canceling each other), and at different locations in identifications of different measurement data. They are often located outside the measured band. (Note that sometimes out-of-the-band poles and zeros do belong to the desired model, far from the

imaginary axis in the Laplace-domain, or far from the unit circle in the *z*-domain.)

The standard deviations of the poles and zeros are also accessible using `stdpz`.

Unnecessary pole/zero pairs can also be detected by studying the correlation coefficients between poles and zeros (see the output argument `rpz` of `stdpz`).

Pole/zero uncertainties can be studied by calculating several sets of roots using random perturbations of the parameters, according to the covariance matrix, and by plotting all the pole/zero sets using `plotelpz`.

A further possibility is to evaluate the *Akaike criterion* for identification results of different orders, and find the one with the smallest AIC value:

$$\mathrm{AIC} = 2(C+n_p) \tag{36}$$

When two or more independent data sets are available, *cross validation* can indicate overmodeling (see also "Cross Validation with Another Set of Measured Data" on page 2-40). A too high-order model tends to follow the noise patterns. The cost function for this model, evaluated with an independent data set, is therefore significantly larger than with the data used for modeling.

Finally, the so-called *F-test* (Söderström and Stoica, 1989) can also be used. If two identified models are available with cost function values $C_1$ and $C_2$, and numbers of estimated parameters $n_{p1} < n_{p2}$, where second model contains the first one as a special case, and both models are good, the expression

$$\Phi = \frac{C_1 - C_2}{C_2} \cdot \frac{2F - n_{p2}}{n_{p2} - n_{p1}} \tag{37}$$

is $F$-distributed (this has nothing to do with the number of frequencies $F$!), with the degrees of freedom ($n_{p2}$-$n_{p1}$,$2F$-$n_{p2}$), and can be tested using standard statistical tables. If the hypothesis can be accepted, the lower-order model is to be chosen.

For large values of F the relative variance of $C_2$ becomes negligible, thus $C_2 \approx 2F\text{-}n_{p2} \approx 2F$, and the variable

$$x = 2F\frac{C_1 - C_2}{C_2} \tag{38}$$

is approximately $\chi^2$-distributed with $n_{p2}$-$n_{p1}$ degrees of freedom, allowing the use of the $\chi^2$-test.

## Study of the Residuals

As shown in statistics, whenever a good model of the system is obtained from the maximum likelihood formulation for Gaussian noise, the residuals (the deviations of the data from the estimated values) will exhibit some characteristic properties. They will be approximately independent and normally distributed with zero mean and the given variances. These properties can be tested.

The routine `rdueelis` calculates the residuals for the model given in Equation (3). However, care should be taken, because estimates of the so-called *nuisance parameters* ($\mathbf{X}$ and $\mathbf{Y}$) are *not consistent*: with the increase of the number of measured points, their variance remains finite. Their estimation error can only be decreased by processing several data sets (several experiments) in parallel. As an effect, though the input and output residuals will have the above approximate properties, they will be correlated with each other at each point. Therefore, the routine calculates also the *equation error* vector:

$$r_{\mathrm{eq}k} = \frac{Y_{mk}}{X_{mk}} - \hat{H}(\Omega_k) \tag{39}$$

The two terms in Equation (39) are positively correlated, since a deviation of $Y_m/X_m$ from its theoretical value will "pull" the estimates in the direction of the deviation. It can be shown that

$$\mathrm{var}\{r_{\mathrm{eq}k}\} = \mathrm{var}\left\{\frac{Y_{mk}}{X_{mk}}\right\} - \mathrm{var}\left\{\hat{H}(\Omega_k)\right\} \tag{40}$$

The second variance in Equation (40) is usually much smaller than the first one, since the estimated parametric transfer function is a kind of average over the noisy Fourier amplitudes. It is calculated by `stdtf`.

When the above two variances are in the same order of magnitude, the variance of the residual may become very small. This small variance should be interpreted with care: it usually means that the model follows the noise, so the averaging effect of the model fitting does not work at the given frequency. The best check of this is to compare the variance with the result of `stdtf` and `stdtfm`. Theoretically, Equation (40) is non-negative, if evaluated with the same variance values as used for the estimation. However, because of the approximations, sometimes small negative values may appear in Equation (40).

The first variance in Equation (40), or rather its half, that is, the variances of the real and imaginary parts of $Y_m/X_m$, can be calculated as follows:

$$\mathrm{var}\{\mathrm{Re}(r_{\mathrm{eq}k})\} \approx \mathrm{var}\{\mathrm{Im}(r_{\mathrm{eq}k})\} \approx \left|\hat{H}(\Omega_k)\right|^2\left(\frac{\sigma_{xk}^2}{\left|\hat{X}_k\right|^2} + \frac{\sigma_{xy}^2}{\left|\hat{Y}_k\right|^2} - 2\,\mathrm{real}\left\{\frac{c_{xy}}{\hat{X}_k\hat{Y}_k}\right\}\right) \quad \textbf{(41)}$$

using the substitution

$$\hat{H}(\Omega_k) \approx \frac{Y_k}{X_k} \quad \textbf{(42)}$$

which usually has a smaller error than the estimated value of $Y_k/X_k$.

These variance values result from the linear approximation of the division, and Equation (41) uses the inconsistent estimates of $Y_k$ and $X_k$, but for large signal-to-noise ratios the approximation is good.

A very powerful method is *cross validation* (see "Cross Validation with Another Set of Measured Data" on page 2-40). Overmodeling is indicated by large residuals. Undermodeling means modeling errors that appear in the residuals for any data set. Statistical analysis of residuals of a series of data sets can quickly reveal such patterns.

## Simulations

An important tool for the study of the properties of the estimates and of the behavior of the whole procedure is the simulation of data vectors. The Frequency Domain System Identification Toolbox provides two possibilities for this purpose.

`simfou` generates simulated *frequency domain* data from the system model and the given variance vectors. This means that the *preprocessed* data are simulated.

It is also possible to simulate *time domain* data, using the routine `simtime`. This simulation has the advantage that the whole data processing chain is checked, from the actual measurements to the estimates. Moreover, by using this facility, frequency domain identification can be compared directly to time domain methods. `simtime` can generate both the transient response and the steady-state response of the system.

There is just one difficult point in this simulation. `elis` uses a nonparametric representation of the noise variances, defined only at the frequencies of interest. However, for the time domain simulation full information on the correlation function or the power spectral density of the noise is required. The routine `simtime` will read in the parameters of a filter, which produces the colored system noise from a white input sequence.

If the noise is approximately white, there is no such problem, the transfer function of this filter is constant. However, for colored noise a solution still needs to be found.

In ELiS, the noise characteristics are "hidden" in the variance vectors. In principle, an additional identification step would be necessary to obtain the values of the parameters of the noise shaping filters. The difficulty is that amplitude-only information is available, thus this identification cannot be directly done by using `elis`. A correct solution is to construct an estimate for this purpose, but this is rather involved[17,18]. Fortunately, the required accuracy allows the use of approximate methods, since the exact modeling of the noise spectrum is usually not critical in the evaluation of the identification procedures.

The knowledge of the physical system may often help. If the noise source is located in the system under test, and you have some idea about the path of the noise to the measurement point, a rough approximation can be given.

The phase of the transfer function can be freely chosen. Thus, filter design methods which approximate a given magnitude response can be used. In MATLAB, the Signal Processing Toolbox routine `yulewalk` can be used for IIR filter approximation with orders *n*/*n*, or the routine `remez` for linear phase FIR approximation. However, care should be taken with both methods, because the variance values usually have stochastic nature, and this can lead to gross errors. Furthermore, `yulewalk` fits the autocorrelation function in the time domain, and this may result in unexpected bias, especially because the built-in Hamming window[19]. For these reasons, the above possibilities should be considered as rough approximations of the noise spectrum.

17. Y. Rolain, R. Pintelon and J. Schoukens, "Amplitude-Only versus Amplitude-Phase Estimation," *IEEE Trans. on Instrumentation and Measurement*, Vol. IM-39, No. 6, pp. 818-823, Dec. 1990.

18. Y. Rolain, *Identification of Linear Systems from Amplitude Information Only,* Ph.D Thesis, Vrije Universiteit Brussel, Dienst ELEC, Brussels, Belgium, 1993.

19. The built-in Hamming window can be switched off, as it has been done in the routine ywalk.

Figure 2-4 shows a simulation example of the approximation of a second-order noise variance shape, where the variance values were estimated from 20 measurements, using the expression of the empirical variance. The simulation has been performed using the following statements:

```
num = 1; r = 0.7; phi = pi/4;
den = real(poly(r*[exp(j*phi),exp(-j*phi)])); %denominator
F = 50; fv = [0:F]'/F/2; %frequency points
%Exact transfer function:
ea = j*fv*2*pi; N = 20;
tf = polyval([0,0,num],exp(ea))./polyval(den,exp(ea));
%simulated random variables:
tfn = abs(tf).*sum(randn(2*N-2,length(fv)).^2)'/(2*N-2);
%transfer function by no windowing version of yulewalk:
[num2,den2] = ywalk(2,2*fv,tfn);
tf2 = polyval([0,0,num2],exp(ea))./polyval(den2,exp(ea));
plot(fv,abs(tf),':w',fv,abs(tf2),'-w',fv,tfn,'+w')
title('Magnitude')
```

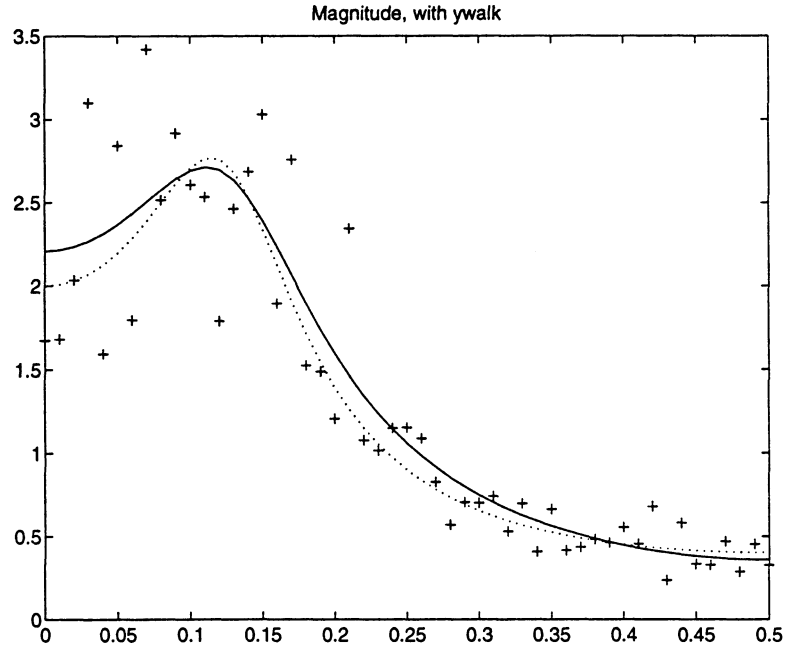



**Figure 2-4: Result of the approximation of the noise spectrum, using ywalk**

The “+” signs denote the measurements, the continuous line shows the result of `ywalk`, while the dotted line marks the exact shape.

Simulations can even be performed starting from pole-zero models. The function `fdcovpzp` converts these models with their uncertainties into transfer function data, used by `simfou` and `simtime`.

## Cross Validation with Another Set of Measured Data

A final test of the quality of the identified model is *cross validation*. This means that the model is evaluated using a different measured data set. The residuals and the cost function of a model can be evaluated for different data sets (see “Detection of Undermodeling and Overmodeling” on page 2-33 and the “Study of the Residuals” on page 2-36). A further possibility is to make a new fit, with the LS starting values or to start from the previous model. The identified parameters should be equal to the previously obtained ones within the uncertainty bounds; large deviations may indicate overmodeling.

# Model Conversions from/to the System Identification Toolbox

The basic aim of the Frequency Domain System Identification Toolbox is the same as of the System Identification Toolbox: to identify linear systems from measured data. The main difference is that the Frequency Domain System Identification Toolbox works in the frequency domain, while the System Identification Toolbox works in the time domain. Thus, they are complementary to each other, and both can sometimes be used to solve a given problem. In this case, their results should be directly compared. This comparison can be easily done if the identified models are in the same format, because the results (pole/zero pattern, transfer function etc.) can be presented in a similar form. This section discusses the possibilities of conversion between the two toolboxes.

In the Frequency Domain System Identification Toolbox the investigated system model is as follows:

$$Y_k = \delta \frac{N(\Omega_k)}{D(\Omega_k)} X_k \tag{43}$$

where $X_k$, $Y_k$ ($k = 1,2...F$) are the complex input and output amplitudes, and $\delta$ is a delay operator:

$$\delta = e^{-sT_d} \qquad \text{or} \qquad \delta = e^{-j2\pi f_s T_d} = \iota^{-f_s T_d} \tag{44}$$

in the $s$- or the $z$-domain, respectively, and $T_d$ is the delay, $N(\Omega_k)$ and $D(\Omega_k)$ are polynomials of $\Omega_k$, and $\Omega_k = s_k = j\omega_k$ in the $s$-domain, or $\Omega_k = z_k = \exp(j\omega_k T)$ in the $z$-domain. The observation equations are

$$Y_{mk} = Y_k + N_{yk} \quad and \quad X_{mk} = X_k + N_{xk} \tag{45}$$

where $X_{mk}$ and $Y_{mk}$ are the measured complex amplitudes, while $N_{xk}$ and $N_{yk}$ are the measurement noises.

Substituting Equation (45) into Equation (43), the following equation is obtained:

$$Y_{mk} = \delta\frac{N(\Omega_k)}{D(\Omega_k)}(X_{mk} - N_{xk}) + N_{yk} \tag{46}$$

In Equation (46) the statistical properties of the noises are assumed to be known at the beginning of the identification of $N(\Omega_k)$, $D(\Omega_k)$ and eventually $\delta$. Thus, the noise analysis is to be done separately, using the available measured data.

In the time domain System Identification Toolbox, identification is performed in the discrete time domain. The general model of the system (with the exception of the nonparametric methods) is in $z$-notations:

$$A(z)Y(z) = \frac{B(z)}{F(z)}z^{-nk}U(z) + \frac{C(z)}{D(z)}E(z) \tag{47}$$

where $A(z)$, $B(z)$, $C(z)$, $D(z)$ and $F(z)$ are polynomials of $z$, $Y(z)$ is the measured signal, $nk$ is the delay (an integer value), $U(z)$ is the input signal, and $E(z)$ is white noise. The orders of the polynomials are na, nb and so on. The methods included into the System Identification Toolbox treat special cases of the general model:

- ARX: $nc = nd = nf = 0$ (instrumental variables can be used)
- ARMAX: $nd = nf = 0$
- ARARX: $nc = nf = 0$ (generalized least squares model)
- ARARMAX: $nf = 0$ (extended matrix model)
- OE: $na = nc = nd = 0$ (output error model)
- BJ: $na = 0$ (Box-Jenkins model)

The methods usually minimize the prediction error; for the ARX case the instrumental variable method can also be used.

The parameters of the model Equation (47) are stored in the so-called theta-format. In what follows, this model will be referred to as *theta*.

When comparing the two models given in Equation (46) and Equation (47), the following significant differences can be noticed.

**Table 2-2: Comparison of system model**

| ELiS | theta |
|---|---|
| $s$- and $z$-domain | $z$-domain |
| input and output noise | one noise source (output noise) |
| nonparametric noise model | parametric noise model |
| also fractional delay | delay integer multiple of $1/f_s$ |
| single input (presently) | also multiple input |

Because of the above important differences, the models can only be converted into each other with restrictions. Also, there are some less important differences: the *theta-format* can handle integer delays only, and the leading coefficients of $A(z)$, $C(z)$, $D(z)$ and $F(z)$ must be ones, while in the Frequency Domain System Identification Toolbox there are no such restrictions.

Conversions are facilitated by the routines `elis2tha` and `tha2elis` in this toolbox.

## Conversion from ELiS to the theta-Format

It is clear from Equation (46) and Equation (47) that for $z$-domain models the conversion is more or less straightforward, while $s$-domain models need special considerations.

### Conversion of Discrete Time Models

The main model parameters have direct equivalents:

$$N(z) \Rightarrow B(z), D(z) \Rightarrow F(z), \tau f_s \Rightarrow nk \tag{48}$$

With $na$ set equal to 0, the only remaining task is to define the equivalents of the noises. Here troubles arise again, since on the one hand ELiS does not use a parametric model of the noise, and on the other hand, the theta-format does not model the input noise.

Let us assume first that the *input noise is negligible*, and the *variance of the output noise is approximately constant along the frequency axis*. In this case, since $na = 0$, $\{N_{yk}\}$ will correspond to a white time domain noise, thus simply $nc = nd = 0$ can be chosen in Equation (47) ($C(z) = 1$, $D(z) = 1$), and the variance value can be passed.

For the comparison of identification results, the properties of the estimates of the *transfer function itself* are important. This is the suggested procedure even if the above assumptions do not hold.

**Approximations for Colored Noise and input/output Noise.** If the results of ELiS are to be used for simulations of time domain data, and the variance is significantly different at different frequencies, the above described solution for model conversion may not be acceptable. In this case, the user has to provide a parametric description of the noise spectrum. However, rather than trying to convert the model into the theta-format, we suggest using the `simtime` simulation routine, which generates time domain data that can be used by both toolboxes. (Be aware of the fact that `simtime` also requires a parametric description of the noise spectrum.)

For obtaining a usable (but approximate) parametric model of the noise, solutions are suggested in the "Simulations" section.

If the input noise cannot be neglected, the situation is even more complex. `simtime` can simulate both input *and* output noises, thus this routine can be used to get realistic simulation data.

Here the question arises, what will the time domain methods identify in the case of input-output noises. Since they will consider the input data as the exact input sequence, it is easy to see that in the result they will *reduce the input noise to the output of the system*. The resulting noise they identify in the model is in the prediction error method

$$N_{\mathrm{red}}(z) = -H(z)N_x(z) + N_y(z) \tag{49}$$

Since the frequency domain noises are assumed to have a rotationally symmetric two-dimensional probability distribution at each frequency:

$$\mathrm{var}\{\mathrm{Re}(N_x)\} = \mathrm{var}\{\mathrm{Im}(N_x)\}, \mathrm{cov}\{\mathrm{Re}(N_x), \mathrm{Im}(N_x)\} = 0 \tag{50}$$

the variance of the combined noise will be

$$\mathrm{var}\{N_{\mathrm{red}}\} = |H(z)|^2 \mathrm{var}\{N_x\} + \mathrm{var}\{N_y\} - 2\overline{H(z)}\,\mathrm{cov}\{N_x, N_y\} \quad \textbf{(51)}$$

with

$$\mathrm{cov}\{N_x, N_y\} = 0.5\mathrm{E}\{\overline{N_x}, N_y\}$$

$C(z)/D(z)$ will approximate the spectrum of this reduced noise.

**Conversion of the Covariances.** The routine `elis2tha` will also accept the covariance matrix of the estimated parameters, and pass it to the theta-format. However, since the leading coefficient of $F(z)$ must be equal to 1 in the theta-format, the polynomial needs to be scaled by the first coefficient. If this coefficient has been estimated (i. e., its variance is not zero), the scaling will change the variances of the denominator coefficients. `elis2tha` will make an attempt to calculate the new covariance matrix by linear approximation. If the linear approximation cannot be applied, it sends a warning message, and passes a covariance matrix full of zeros.

### Conversion of Continuous-Time Models

Continuous-time and discrete-time systems should be transformed into each other with extreme care[20]. Since ELiS is based on band-limited measurements, the best recipe is to repeat the identification in the $z$-domain with the same data. Those who like adventures can also try the standard $s$–domain to $z$–domain mapping methods (bilinear transform, impulse invariant transform etc.[21]), but generally the results will be much worse.

## Conversion from the theta-Format to ELiS

In this direction there are fewer difficulties. Let us divide Equation (47) by $A(z)$:

$$Y(z) = \frac{B(z)}{A(z)F(z)} z^{-nk} U(z) + \frac{C(z)}{A(z)D(z)} E(z) \quad \textbf{(52)}$$

20. J. Schoukens and R. Pintelon, “Identification — Why do we need it, how to use it?” *Conference Record of the Instrumentation and Measurement Technology Conference IMTC/93, 93CH3292-0*, Irvine, Orange County, CA, May 18-20, 1993. pp. 246-251.

21. see e.g. L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*, Englewood Cliffs, NJ, Prentice-Hall, 1975.

Most of the parameters in Equation (46) can be immediately recognized:

$$B(z) \Rightarrow N(z), A(z)F(z) \Rightarrow D(z), nk \Rightarrow f_s\tau \tag{53}$$

The input noise is equal to zero. The variance vector can be easily generated from the coefficients of $E(z)$ in Equation (52). If the model in the theta-format is for multiple inputs, in ELiS an individual model will correspond to each input.

## Conversion of the Covariances

The routine `tha2elis` will try to pass the covariance matrix of the estimated parameters to ELiS. However, since the denominator of the transfer function is a product of $A(z)$ and $F(z)$, if the coefficients of $A(z)$ have been estimated, i. e., their variance is not zero, the covariance matrix of the result of this multiplication will be a nonlinear function of the original covariances. `tha2elis` will make an attempt to calculate the new covariance matrix by linear approximation. If the linear approximation cannot be applied, it sends a warning message, and passes a covariance matrix full of zeros.

# Data Formats and File Handling

For the Frequency Domain System Identification Toolbox, standard data and file formats are defined. The functions of the toolbox exchange data with each other via variables in the workspace, and these variables generally have a straightforward definition. There are some compound variables like the ones comprising the parameters of an estimated transfer function, the delay, the sampling frequency and the domain ($s$ or $z$), to be used by different functions. Such variables must have a standard form. Moreover, since the purpose of the toolbox is working on real measurement problems, we attempted to facilitate *data exchange* between it and other programs, like the ones for the control of the measurement setup, data logging and preprocessing, etc., by providing a *standard archive format*. The data and file formats cover

- *Time* vectors and files containing measured or simulated time domain data
- *Fourier* vectors and files containing measured, simulated or calculated frequency domain complex amplitudes or input-output point pairs
- *Variance* vectors and files containing variances of the real parts (that is, also of the imaginary parts) of the measured frequency amplitudes
- *Parameter* vectors and files containing parameters of the transfer functions
- *Covariance* vectors and files, containing the covariance matrix of the estimated parameters

There are also so-called *report* files, containing all relevant information about a run, in textual format. These files are ASCII files, and they are not standardized.

Data vectors usually contain different kinds of data, belonging to the same measurement or estimation procedure, in a long vector form. The data vectors can be "taken apart" by using the `imptim`, `impfou`, `impvar`, `imppar`, and `impcov` functions. This allows direct access to the numerator and denominator coefficients, so that the user does not need to worry about the internal structure of the vector. The data vector formats are defined in Appendix A1.

Each file type listed above (but not the report file) has an ASCII and a MATLAB binary version as described below. The ASCII and binary versions can be generated from MATLAB using the `exptim`, `expfou`, `expvar`, `exppar`, and `expcov` functions, respectively. Both the ASCII files and the binary files can be read into MATLAB using the `imptim`, `impfou`, `impvar`, `imppar`, and `impcov` functions.

The binary files can also be directly loaded into (or saved from) MATLAB, if the conventions are followed (see Appendix A). This allows much quicker data transfer. However, this must be done properly, because the variable names are fixed in direct load/save, and the checkings otherwise performed by the export/import routines are bypassed.

The *full* ASCII files can also be read in MATLAB, using the above import functions, but this may be a rather slow procedure. The read-in is based on the function `loadasc`, which ignores comments (preceded by a % mark in a line) in an ASCII file, and generates a vector with a user-defined name, containing all the numbers present in the file.

The different file formats can be converted to each other using the file conversion function `elisfcnv`.

The file types and formats are recognized by their extensions. The first character of the extension refers to the contents of the file (Fourier, variance, etc.), the second and third to the file format: `nt` is reserved for ASCII files with no text (flat ASCII files), and `bn` is reserved for binary files. The standard extensions are:

| | | | |
|---|---|---|---|
| Time files: | `.tim`, | `.tnt`, | `.tbn` |
| Fourier files: | `.fou`, | `.fnt`, | `.fbn` |
| Variance files: | `.var`, | `.vnt`, | `.vbn` |
| Parameter files: | `.par`, | `.pnt`, | `.pbn` |
| Covariance files: | `.cov`, | `.cnt`, | `.cbn` |

The complete definition of the file formats is given in Appendix A.

An extra facility, incorporated into the toolbox, allows the export of vectors into flat ASCII files (`expvect`), for data transfer to graphing programs.

From toolbox version 3.0, data and model objects can be generated (tiddata, fiddata, fidmodel), and saved directly to MAT-files. This makes the above defined files mostly unnecessary (except importing data or exporting models through text files).

# A Typical Identification Session

This section illustrates the frequency domain identification procedure for the identification of a flexible robot arm. The procedures described below include a few typical steps, applicable under a variety of circumstances. However, every identification task has its own flavor. For systems other than this robot arm, other procedures may be adequate. Therefore, while this section serves as an example, and suggests a series of good solutions, it should not be followed blindly for other systems.

The procedures described below are implemented in the M-file `rarmdemo`. The code fragments do not necessarily constitute a full program; their purpose is rather to illustrate a simple way of calculation.

The behavior of a flexible robot arm was measured by applying controlled torque to the vertical axis at one end of the arm, and measuring the tangential acceleration of the other end. The excitation signal was a multisine, generated with frequency components at `[1:2:199]*df`, with `df` = 500/4096 Ý 0.122 Hz; that is, the frequency range was about 0.12 Hz – 24 Hz. The originally flat multisine was distorted by the nonlinear behavior of the actuator. The odd harmonic frequencies provided that components produced by a squaring nonlinearity would not disturb the identification. The input and output signals were sampled with sampling frequency fs = 500 Hz. Sampling was synchronized to the excitation signal so that 4096 samples were taken from each period. The data records contain 40960 points; that is, 10 periods were measured. The data are available in the file `robotarm.mat`[22]. The time series are scaled to 16-bit integers in order to reduce the file size.

22. This measurement was made at the Department of Mechanical Engineering, Catholic University of Leuven (KUL), in cooperation with Department ELEC, Vrije Universiteit Brussel (VUB), as a part of the Belgian program "Interuniversity Attraction Poles (IUAP50: Robotics and Industrial Automation)" initiated by the Belgian State, Prime Minister's Office, Science Policy Programming. These data belong to the public domain and can be freely used by anyone.

## Investigation of the Time Domain Data

First the time domain data and the autocorrelation function can be investigated. Let us have a look at the time domain data.

```
load robotarm.mat
     %xt = scaled input time record, 40960x1
     %yt = scaled output time record, 40960x1
     %fs = sampling frequency, 500 Hz
     %ascale = scaling factor of the time records
     %N = number of points in a period (4096)
     %freqind = index numbers of sine waves in DFT of
                %a period, [1:2:199]'
xt = xt*ascale; yt = yt*ascale; dt = 1/fs; df = fs/N;
Nl = length(xt); expno = Nl/N;
T = Nl*dt; timevtot = [1:Nl]'*dt; freqindtot = freqind*expno;
clf, hold off
subplot(2,1,1), plot(timevtot,xt,'-w')
title(sprintf(['Input data (torque), number of points:',...
        ' .0f'],Nl))
xlabel('Time, s')
subplot(2,1,2), plot(timevtot,yt,'-w')
title(sprintf(['Output data (acceleration), ',...
         'number of points:  .0f'],Nl))
xlabel('Time, s')
```

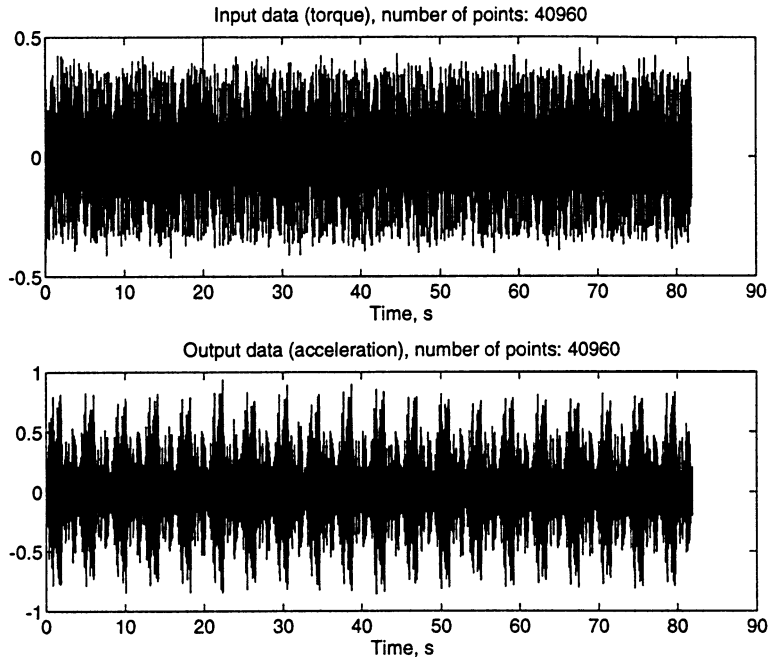



**Figure 2-5: Time domain data**

Not much can be stated on the basis of the time functions; not even the period length can be read. The input signal apparently has a smaller crest factor than the output one, but that's about all we can see. More can be determined from the autocovariance function. We will evaluate the so-called circular correlation, which is the inverse Fourier transform of the periodogram, `(1/N1)*abs(X).^2`. We will suppress the dc component to get the autocovariance. In order to have immediate information about the periodicity, we will connect every 4096th point by a dotted line.

```
%Calculations:
X = fft(xt); dcx = X(1); X(1) = 0; Sx = (1/Nl)*abs(X).^2;
Cx = real(ifft(Sx));
%Plotting:
clg
subplot(2,1,1)
axv = [0,max(timevtot),1.1*min(Cx),1.1*max(Cx)];
plot(timevtot,Cx,'-',timevtot([1:4096:Nl,Nl]),...
     Cx([1:4096:Nl,Nl]),':')
axis(axv)
title('Circular autocovariance function'), xlabel('time, s')
```

From the autocovariance function (see below) several conclusions can be drawn. First of all, there is indeed a periodicity of 4096*dt = 8.192 s. The autocovariance function corresponds to a bandlimited white spectrum; the negative peaks verify the use of odd harmonics at `[1:2:199]*df`. The signal was oversampled by a factor of 2048/199 = 10, that is, from the `sin(x)/x` shaped main lobes of the autocovariance function, about 20 points are sampled.

```
p1h = axes('Position',[0.1300,0.1100,0.3175,0.3375]);
pv = 1*4096+1+[-15:15];
axv = [min(timevtot(pv))-dt,max(timevtot(pv))+dt,...
           1.2*min(Cx(pv)),1.1*max(Cx(pv))];
plot(timevtot(pv),Cx(pv),'o',timevtot(pv),Cx(pv),':')
axis(axv)
mpv = median(pv); hold on
plot(timevtot(mpv)*[1,1],[0,Cx(mpv)],':w',...
          axv(1:2),[0,0],':w')
title('Lag No. 1')
hold off
p2h = axes('Position',[0.5825,0.1100,0.3175,0.3375]);
pv = 5*4096+1+[-15:15];
axv = [min(timevtot(pv))-dt,max(timevtot(pv))+dt,...
         1.2*min(Cx(pv)),1.1*max(Cx(pv))];
plot(timevtot(pv),Cx(pv),'o',timevtot(pv),Cx(pv),':')
axis(axv)
mpv = median(pv); hold on
plot(timevtot(mpv)*[1,1],[0,Cx(mpv)],':w',...
         axv(1:2),[0,0],':w')
title('Lag No. 5'), hold off
```

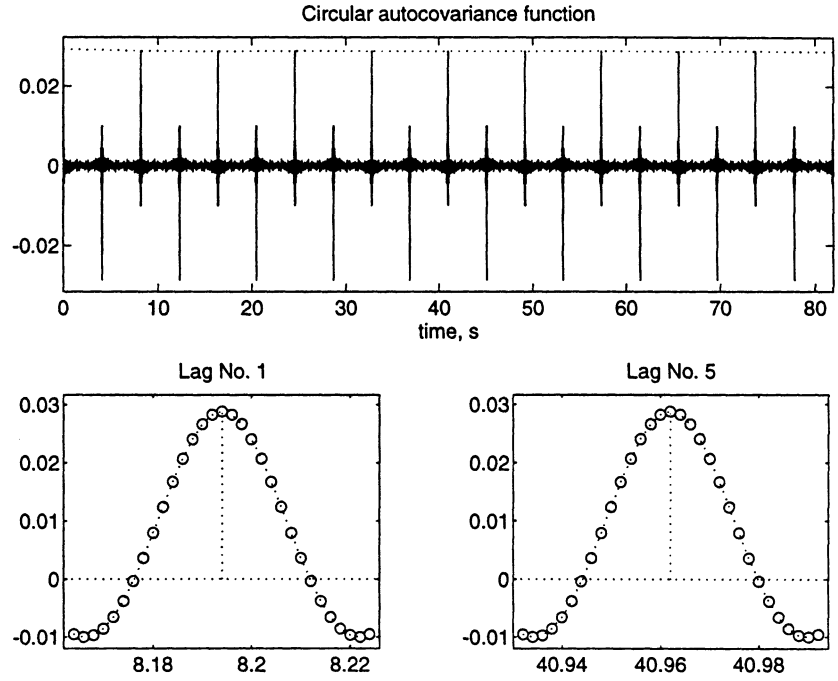



**Figure 2-6: The autocovariance function**

It is obvious from the enlarged peaks that synchronization is very good between the excitation signal and the sampling clock. This can also be verified in the frequency domain. If there was a slip, it could not be more than about 0.003% (dt/2 in a time of 4*4096*dt).

## Examination of the Signal-to-Noise Ratios

The autocovariance function can be used for approximate determination of the signal-to-noise ratio; the power of the periodic components is given by one of the peaks at nonzero lag, the total power is given by the covariance value at zero.

```
PtotxC = Cx(1); PperxC = mean(Cx([1:9]*N+1));
%
fprintf(['Cx(0) = %.3g, Cx(k*Tp) = %.3g, SNRx = %.1f',...
    ' dB\n'],Cx(1),Cx(N+1),10*log10(PperxC/(PtotxC-PperxC)) )
Cx(0) = 0.0293, Cx(k*Tp) = 0.0288, SNRx = 17.3 dB
```

However, this is not exactly what we need. The useful signal has power at the given frequencies only; the rest are spurious components, produced by the nonlinearities.

```
%Calculations:
freqvtot = [0:Nl/2-1]'/Nl*fs;
Y = fft(yt); dcy = Y(1); Y(1) = 0;
Sy = (1/Nl)*abs(Y).^2; clear Y
%Plotting:
clg, hold off, subplot(2,1,1)
plot(freqvtot,abs(X(1:Nl/2)) )
title('Input amplitudes'), xlabel('Frequency, Hz')
subplot(2,1,2), plot(freqvtot,abs(Y(1:Nl/2)) )
title('Output amplitudes'), xlabel('Frequency, Hz')
```

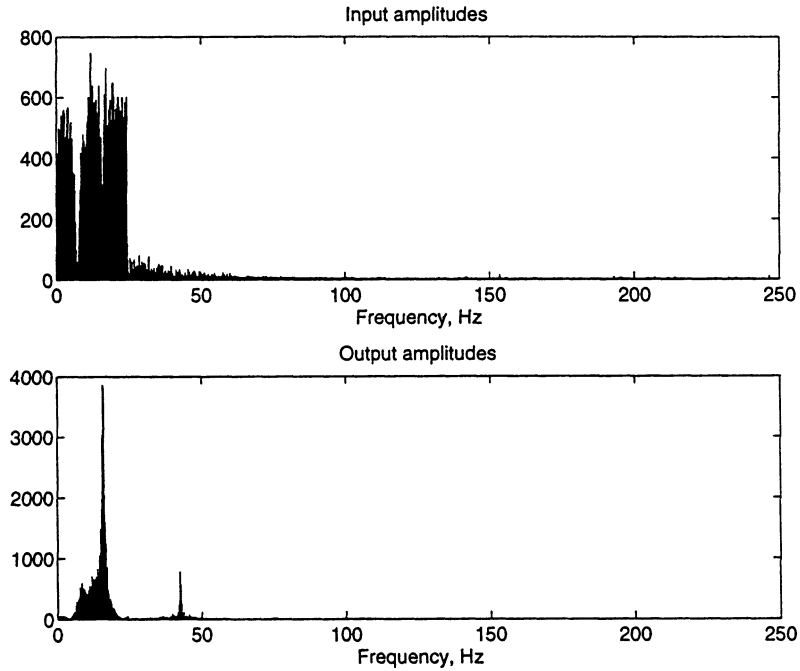



**Figure 2-7: Input and output Fourier amplitudes**

The input amplitude spectrum is not really flat. The cause is most probably the nonflat transfer function of the system composed of the actuator and the device under test. The two dips at 7.2 Hz and 15.7 Hz in the input spectrum correspond to resonance points of the system, where the actuator is not capable of maintaining the signal level. This is not a serious problem since the frequency range of interest is sufficiently covered by nonzero excitation amplitudes. The powers of the useful signal, of the harmonic components, and of the noise, can be calculated for both the input and the output signals.

```
%Calculations:
Pux = 2*sum(Sx(freqindtot+1))/Nl; Ptotx = sum(Sx)/Nl;
Pperx = real(exp(j*2*pi*N*[0:Nl-1]/Nl)*Sx)/Nl;
Puy = 2*sum(Sy(freqindtot+1))/Nl; Ptoty = sum(Sy)/Nl;
Ppery = real(exp(j*2*pi*N*[0:Nl-1]/Nl)*Sy)/Nl;
%Display results:
fprintf('  Input signal:\n')
fprintf(['    Total power: %.3g, useful power: %.3g, ',...
     'noise power: %.3g\n'],Ptotx,Pux,Ptotx-Pperx)
fprintf(['    Power of spurious periodic components: ',...
     '%.3g\n'],Pperx-Pux)
fprintf(['    SNR: %.1f dB, for useful components only:',...
     ' %.1f dB\n'],10*log10(Pperx/(Ptotx-Pperx)),...
     10*log10(Pux/(Ptotx-Pux)) )
fprintf('  Output signal:\n')
fprintf(['    Total power: %.3g, useful power: %.3g,',...
     ' noise power: %.3g\n'],Ptoty,Puy,Ptoty-Ppery)
fprintf(['    Power of spurious periodic components: ',...
     '%.3g\n'],Ppery-Puy)
fprintf(['    SNR: %.1f dB, for useful components only: ',...
     '%.1f dB\n'],10*log10(Ppery/(Ptoty-Ppery)),...
     10*log10(Puy/(Ptoty-Puy)) )

Input signal:
    Total power: 0.0293
       useful power: 0.0286, noise power: 0.000508
    Power of spurious periodic components: 0.000218
    SNR: 17.5 dB, for useful components only: 16.0 dB
  Output signal:
    Total power: 0.0828
       useful power: 0.0797, noise power: 0.00139
    Power of spurious periodic components: 0.00171
    SNR: 17.7 dB, for useful components only: 14.1 dB
```

The signal-to-noise ratios are quite good, although the nonlinearities produce significant components. Since the noise is larger than the nonlinearity products, these will hopefully not deteriorate the identification significantly. Let us select the excitation lines `[1:2:199]` only, so the SNR will be improved by a factor of about 20 (13 dB).

## Conversion to Frequency Domain

Now periodwise conversion to frequency domain follows. Ten periods were measured. We will treat each period as a separate experiment, thus noise analysis can be performed. The time vector of a period (one experiment) is shorter than the measurement data vectors `xt` and `yt`. Therefore, the easiest way for passing the data to `tim2fou` is to use `exptim`.

```
timevect = [1:N]'*dt; freqv = freqind*df;
F = length(freqind);
[x,y] = tim2fou(exptim(timevect,xt,yt),freqv);
```

## Variance Analysis

Frequency domain noise analysis can be performed using `varanal`. In parallel with noise analysis, a test of synchronization could be performed, making use of the post-measurement synchronization possibility. This would take considerable time. We already know that the measurements were made with good synchronization, so testing of synchronization will not be done.

```
%Variance analysis:
[vx,vy,cxy,mx,my,Na,Np,cfl,dv,sd] = ...
               varanal(expfou(freqv,x,y));
%Calculation of the SNR's:
PNx = 0; Px = 0; PNy = 0; Py = 0;
Px = 2*sum(abs(mx).^2/N)/N; Py = 2*sum(abs(my).^2/N)/N;
PNx = 2*sum(2*vx/N)/N; PNy = 2*sum(2*vy/N)/N;
%Plotting:
subplot(2,2,1)
plot(freqv,vx,'+'), title('Input variances')
xlabel('Frequency, Hz')
ylabel(sprintf('SNR = %.1f dB',10*log10(Px/PNx)))
subplot(2,2,2)
plot(freqv,vy,'+'), title('Output variances')
xlabel('Frequency, Hz')
ylabel(sprintf('SNR = %.1f dB',10*log10(Py/PNy)))
subplot(2,2,3)
plot(freqv,abs(cxy),'+'), title('I/O covariances'),
xlabel('Frequency, Hz')
subplot(2,2,4)
plot(freqv,abs(cxy)./sqrt(vx.*vy),'+')
title('I/O corr. coefficients'), xlabel('Frequency, Hz')
```

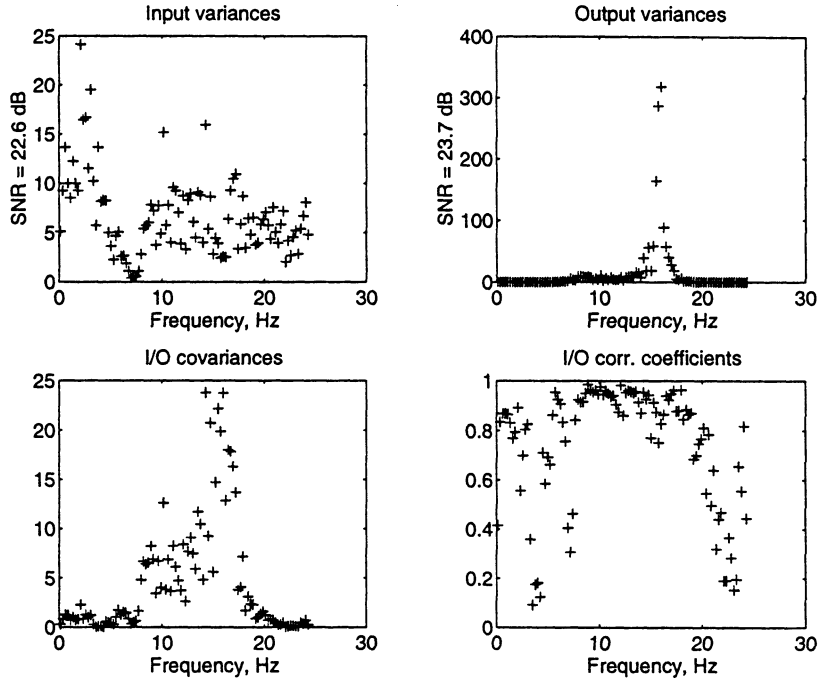



**Figure 2-8: The results of noise analysis**

The total SNR is increased by the selection of the points of interest:

```
fprintf('SNRinp = %.1f dB, SNRoutp = %.1f dB\n',...
        10*log10(Px/PNx),10*log10(Py/PNy))
SNRinp = 22.6 dB, SNRoutp = 23.7 dB
```

The increase of about 5-6 dB is due to the oversampling and selection of points of interest only. It is less than expected, probably because the noise has less power at higher frequencies than in the lower frequency band.

The input-output covariances are quite large so they can not be ignored. This could mean that a part of the noise goes through the system, that is, the estimation is corrupted by less noise than calculated above. This can be verified by plotting cxy./vx. If an important part of the noise goes through the system, this plot will have a shape similar to the transfer function:

$$cxy = 0.5E\{\overline{N}_x N_y\} \sim E\{\overline{N}_x N_x tf\} = vx*tf$$

We will make this plot when the approximate shape of the transfer function is plotted.

A rough guess about the gain in SNR can be obtained by calculating the SNR of the nonparametric estimate of the transfer function. The SNR of this nonparametric estimate is smaller than those above, because the division `ym./xm` amplifies the noise significantly where `xm` is small. In `elis`, this division is not used, as it can be seen from the cost function.

```
[tfm,stdAm] = stdtfm([freqv,mx,my],[vx,vy]);
[tfmc,stdAmc] = stdtfm([freqv,mx,my],[vx,vy,cxy]);
SNRtfm = 10*log10(sum(abs(tfm).^2)/sum(2*stdAm.^2));
SNRtfmc = 10*log10(sum(abs(tfmc).^2)/sum(2*stdAmc.^2));
fprintf(['SNRtfm without covariance: %.1f dB, with ',...
     'covariance: %.1f dB\n'],SNRtfm,SNRtfmc)
SNRtfm without covariance: 13.6 dB, with covariance: 14.2 dB
```

The SNR can also be studied at the selected points by calculating it both for the input and the output. However, it is much quicker to plot the nonparametric transfer function estimates with uncertainties using `ploteltf`.

```
ploteltf('','',[freqv,x(1:F),y(1:F)],'','',[vx,vy,cxy])
```

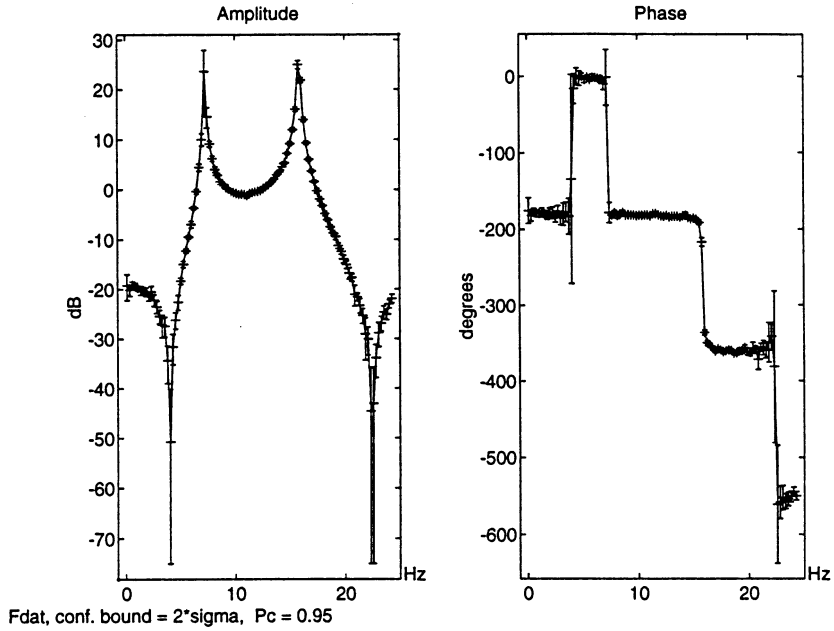



**Figure 2-9: Uncertainties of the nonparametric transfer function estimate**

The SNR is quite good indeed. Now `cxy./vx` will be plotted to check that a significant part of the noise goes through the system.

```
ploteltf('','',[freqv,ones(100,1),cxy./vx])
```

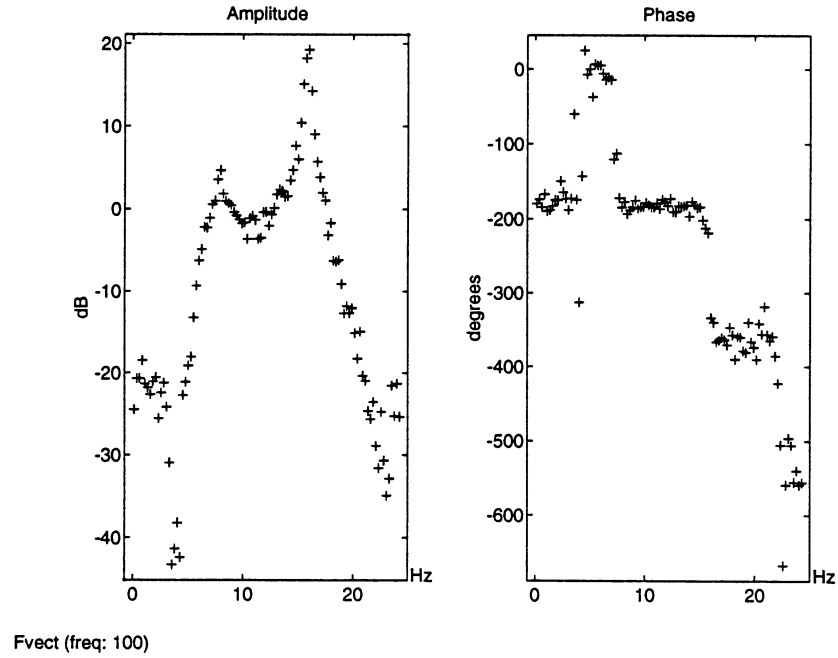



**Figure 2-10: The shape of cyx./vx, showing a pattern similar to the transfer function**

`cxy./vx` has a shape very similar to the transfer function. A large part of the noise goes through the system.

## Identification

We can now proceed with identification. Since the experiments are well synchronized, the average of the complex amplitudes, `mx` and `my`, can be used. Since these averaged quantities have smaller variances, `vx`, `vy` , and `cxy` have to be divided by the number of averaged experiments, `Na`. For the run of `elis`, the numerator and denominator orders of the transfer function have to be given. From the nonparametric plot it is obvious that at least two complex pole pairs and two complex zero pairs will be necessary. So, let us start with a system 4/4.

```
Fdat = [freqv,mx,my]; vdat = [vx,vy,cxy]/Na;
[pv,fit,Cp] = elis(Fdat,vdat,['s',4,4],[],'',10);
```

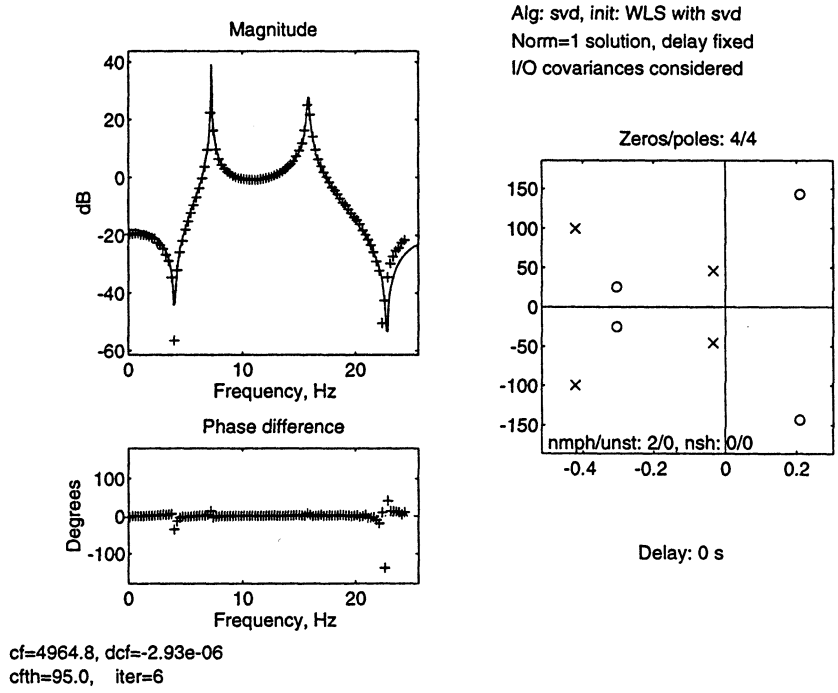



**Figure 2-11: Result of the 4/4 fit**

The fit is quite good, but the cost function is still large, and there is an apparent mismatch at the higher frequency band. It seems reasonable to increase the orders. Let us try a 6/6 system.

```
[pv66,fit66,Cp66] = elis(Fdat,vdat,['s',6,6],[],'',10);
```

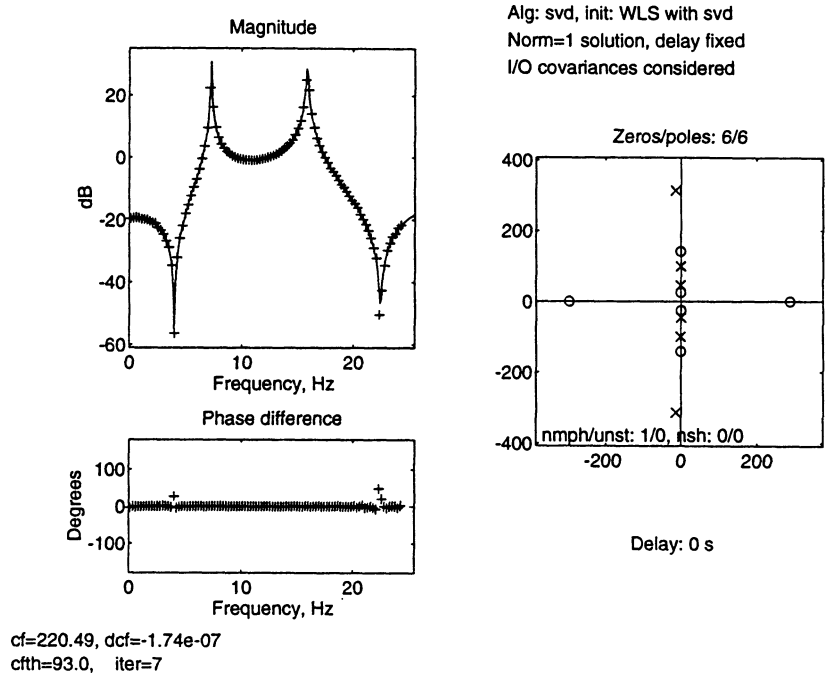



**Figure 2-12: Result of the 6/6 fit**

The fit is much better. The cost function got quite close to the theoretical value, however, it is still larger than the theoretical value by about a factor of 2.5, so there are probably still small modeling errors. A model of order 8/8 can still be tried.

```
[pv,fit88,Cp] = ...
              elis(Fdat,vdat,['s',8,8],[],[NaN,NaN,100],25);
```

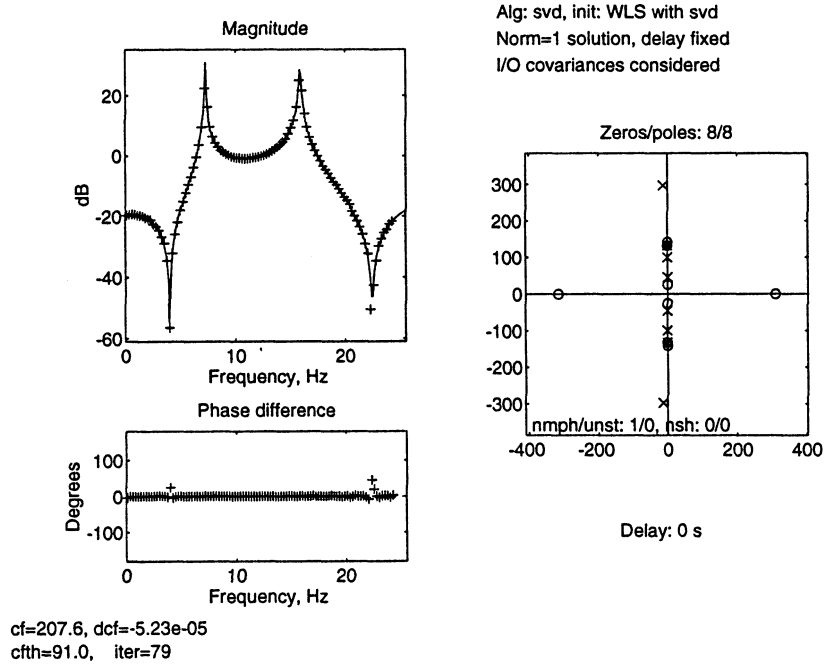



**Figure 2-13: Result of the 8/8 fit**

The 8/8 model is not much better than the 6/6 one. The cost function decreased from 220.5 to 207.6 only and the theoretical values are 91 and 93, respectively. The large number of necessary iterations is also an indicator of probable overmodeling. Now other attempts can be made with different numerator and denominator orders, but none of them is successful finding a better fitting stable model than the 6/6 one. The modeling error is probably due to nonlinearities. The order need not be further increased.

However, there is still a chance that a lower-order system can be as good as the 6/6 one. Let us make a pole-zero uncertainty plot of the 6/6 model.

```
plotelpz(pv66,[],Cp66,2)
```

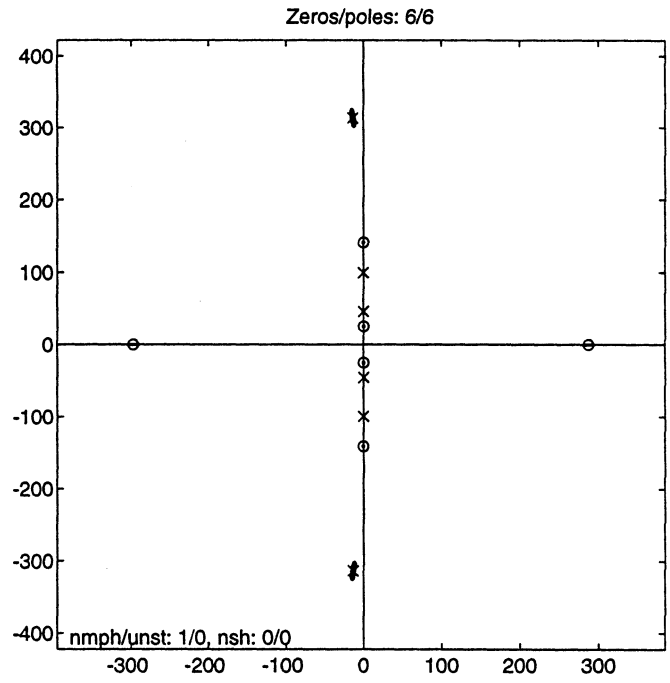



**Figure 2-14: Uncertainty ellipses of the 6/6 fit**

The confidence ellipses are quite small, so they are of no use in this case. What can be seen is that the one zero pair and one pole pair have larger variance than the rest. However, we can speculate that the real zero pair far from the imaginary axis plays no important role, so it is reasonable to decrease the numerator order by 2. This will also allow the transfer function to decrease for higher frequencies, the usual behavior of physical systems. The two poles may correspond to the resonance around 42 Hz, shown in the complex output amplitude plot. At this frequency there was no excitation applied, however, the nonlinearities produced enough overharmonics to show this resonance. For a proper identification of it, the complex input/output amplitudes around this frequency should also be used, and a broader excitation signal should have been applied. We are not going to specifically deal with this resonance, but will proceed with the above used data. But before making the 4/6 fit, let us have a closer look at the uncertainties of the important poles and zeros.

```
plotelpz(pv66,[-0.8,0,-200,200],Cp66)
```


**Figure 2-15: Uncertainties of the important zeros and poles**

The dominant error is in the damping of the poles and zeros; their frequencies are well determined.

Let us do now a 4/6 fit. For a thorough study of this fit, a report file can be generated.

```
[pv46,fit46,Cp46] = elis(Fdat,vdat,['s',4,6],[],'',10,[],...
'robotarm.rep');
```

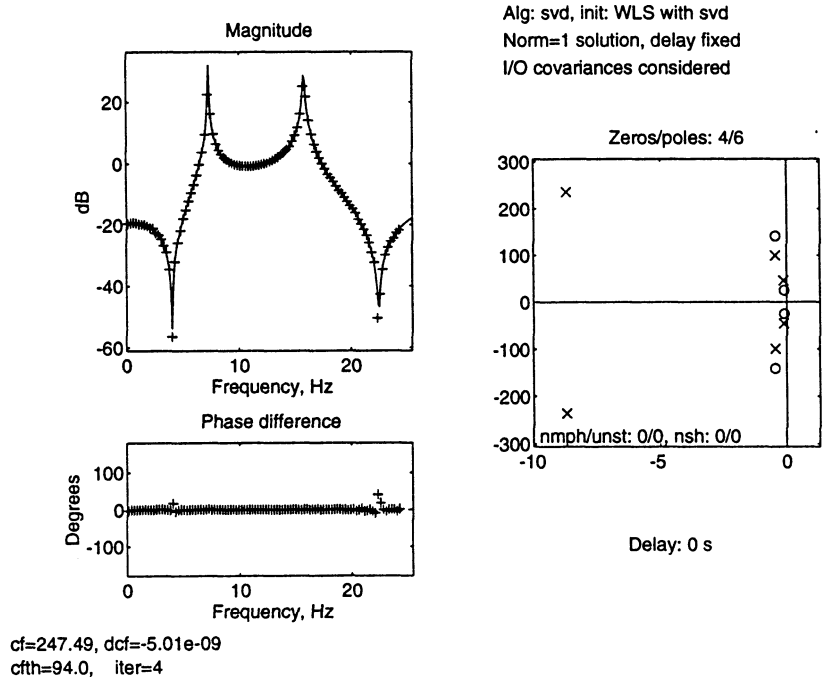



**Figure 2-16: Result of the 4/6 fit**

## Model Validation

```
type robotarm.rep
FREQUENCY DOMAIN SYSTEM IDENTIFICATION TOOLBOX FOR MATLAB
ELiS RUN PROTOCOL, date and time: 13-Dec-93, 17:54:48
Report file: robotarm.rep
Default run parameters, modified in command line

Fourier data given in command line
Experiment: 1, number of frequencies: 100
Variance data given
Input-output covariances are taken into account
Fit in s-domain, frequencies normalized internally
   by omegasc = 76.699 rad*Hz
   suggestion: omegasc = 109.48 rad*Hz
Orders: 4/6
No fixed nonzero parameters (norm=1 solution)
Fixed value of the delay:  0 s
Algorithm: Singular value decomposition
Initial value setting: WLS, by singular value decomposition

Allowed maximum number of iterations: 50,
   iterations performed: 4
Total run time: 0.43 min, time used for plots: 0.28 min
   time used for pole/zero calculations: 0.00 min
Stop if relative change of cost function is smaller than
   1.00e-06
   last relative variation: -2.03e-11
Stop if maximum relative change of parameters is smaller than
   0.00e+00
   last max. rel. variation: +1.09e-06
Condition number of the decomposed or inverted matrix:
                      70.001, reciprocal: 0.014285
Condition number of J: 70.001, reciprocal: 0.014285
Condition number of Q=d^2C/dp^2: 4496.6,
   reciprocal: 0.00022239
eps = 2.220e-16
Value of the cost function: 247.49, its double: 494.99
Theoretical value of the cost function: 94.0
Degrees of freedom of the chi-square value (2*cfth): 188
```

```
5%-95% points of the theoretical distribution of the cf:
   75.959, 113.93
Approximate mean model error: 0.2232
Mean absolute value of the transfer function: 1.3037
Approximate relative mean model error: 0.17121
Akaike criterion: 518.99
Number of free parameters: 12

Values of the parameters (with standard deviations, calculated
from the approximate covariance matrix):
Numerator:
s^0    2.573756993444797e-02    std: 2.5441e-05  (0.098848%)
s^1    9.551589736680904e-06    std: 4.4310e-07  (4.639%)
s^2    4.165357328394967e-05    std: 1.7078e-08  (0.041%)
s^3    2.261458387371986e-09    std: 4.5396e-11  (2.0074%)
s^4    2.026311165334144e-09    std: 1.3721e-12  (0.067713%)
Denominator:
s^0   -2.311031465095138e-01    std: 2.0162e-04  (0.087241%)
s^1   -1.209269285803730e-04    std: 2.4367e-06  (2.0151%)
s^2   -1.379252065522331e-04    std: 1.5389e-07  (0.11158%)
s^3   -5.572203126063595e-08    std: 1.0717e-09  (1.9233%)
s^4   -1.355803200968540e-08    std: 2.7230e-11  (0.20084%)
s^5   -3.740349830798937e-12    std: 7.8835e-14  (2.1077%)
s^6   -2.006123724224281e-13    std: 1.3813e-15  (0.68853%)
Delay:
0 s    fixed

Zeros (rad*Hz):
-4.5424e-01 -1.4113e+02*j
-4.5424e-01 +1.4113e+02*j
-1.0378e-01 -2.5252e+01*j
-1.0378e-01 +2.5252e+01*j

Poles (rad*Hz):
-8.7516e+00 -2.3556e+02*j
-8.7516e+00 +2.3556e+02*j
-4.4728e-01 -9.9527e+01*j
-4.4728e-01 +9.9527e+01*j
-1.2343e-01 -4.5749e+01*j
-1.2343e-01 +4.5749e+01*j
```

```
First nonzero numerator coefficient is b(4)=2.0263e-09
First nonzero denominator coefficient is a(6)=-2.0061e-13
b(4)/a(6)=-1.0101e+04
Static gain: -1.1137e-01, -19.1 dB

Parameter file to save data: -
Covariance file to save data: -
%%%%% End of report file robotarm.rep %%%%%
```

The fit seems to be very reasonable. We have a good identified model.

The 4/6 model can be verified using the standard techniques of the toolbox. We will not do all the possible tests, but will do some of the typical ones. One of the most important indicators of the quality of the fit is the value of the cost function, already discussed above. It is also important to examine visually the quality of the fit on the plot of `elis`. In this case the error is too small to be easily detected on the plots. The phase errors at the zeros are not really important, since here the phase information of the measurements is small. The confidence interval plots using `ploteltf` could also be used, but the confidence intervals have to be magnified for visual checking.

```
ploteltf(pv46,[],expfou(freqv,x(1:F),y(1:F)),'','',Cp46,100)
```

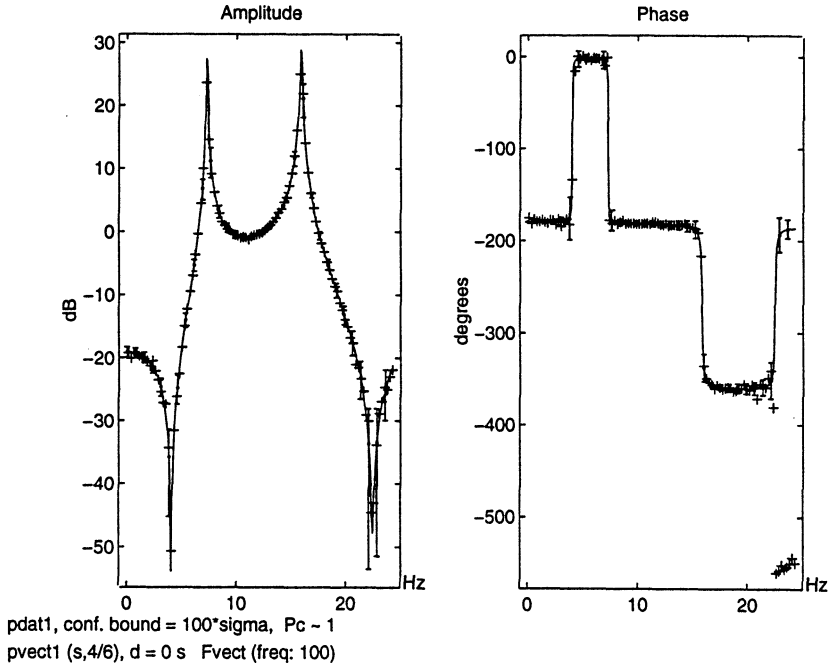



**Figure 2-17: Magnified confidence bounds of the 4/6 fit**

Even with a bound of 100*sigma, not much can be seen. It is better to look for other tests. We think that there are modeling errors, so let us check the approximate mean model error.

```
fprintf('Approximate mean model errors:\n')
fprintf('4/6 model   6/6 model   8/8 model\n')
fprintf('  %.2f        %.2f        %.2f\n',...
     fit46(10),fit66(10),fit88(10))
fprintf('Mean absolute value of the transfer function:\n')
fprintf('  %.2f        %.2f        %.2f\n',...
     fit46(11),fit66(11),fit88(11))

Approximate mean model errors:
4/6 model   6/6 model   8/8 model
  0.22        0.21        0.20
Mean absolute value of the transfer function:
  1.30        1.31        1.31
```

These values illustrate that the modeling error is not negligible, and is in the same order of magnitude for all three fits. Because of the modeling error, the Akaike criterion cannot be used. For closer investigation of the quality of the fit, the residuals can be calculated.

```
[rx,ry,ryx,vryx,xe,ye] = ...
          rdueelis(pv46,Cp46,expfou(freqv,x,y),[vx,vy,cxy]);
```

`rx`, `ry` and `ryx` can be studied for normal distribution and whiteness. We will do a few tests for `ryx`. First, it has to be standardized, dividing the residuals at each frequency point by the standard deviation.

```
il = [1:F]'; il = il(:,ones(1,expno)); il = il(:);
ryxn = ryx./sqrt(vryx(il));
```

If the fit is good, the standardized residuals have to exhibit circular standard normal distribution at each point, and they have to be independent. These properties will be checked by simple tests. First, let us draw the histograms of the real and imaginary parts. The dotted lines show the standard normal probability density function, scaled up to the histogram, which is made of 1000 points, with `dx = 0.2`.

```
%Calculations:
dx = 0.2; X = [-3.8:dx:3.8];
Nhr = hist(real(ryxn),X); Nhi = hist(imag(ryxn),X);
fX = 1/sqrt(2*pi)*exp(-X.^2/2);
np = F*expno;
%Plotting:
clg
subplot(121), bar(X,Nhr), hold on, plot(X,fX*np*dx,':g'), hold
off
title('Histogram of real part'), xlabel('real(ryxn)')
subplot(122), bar(X,Nhi), hold on, plot(X,fX*np*dx,':g'), hold
off
title('Histogram of imaginary part'), xlabel('imag(ryxn)')
```

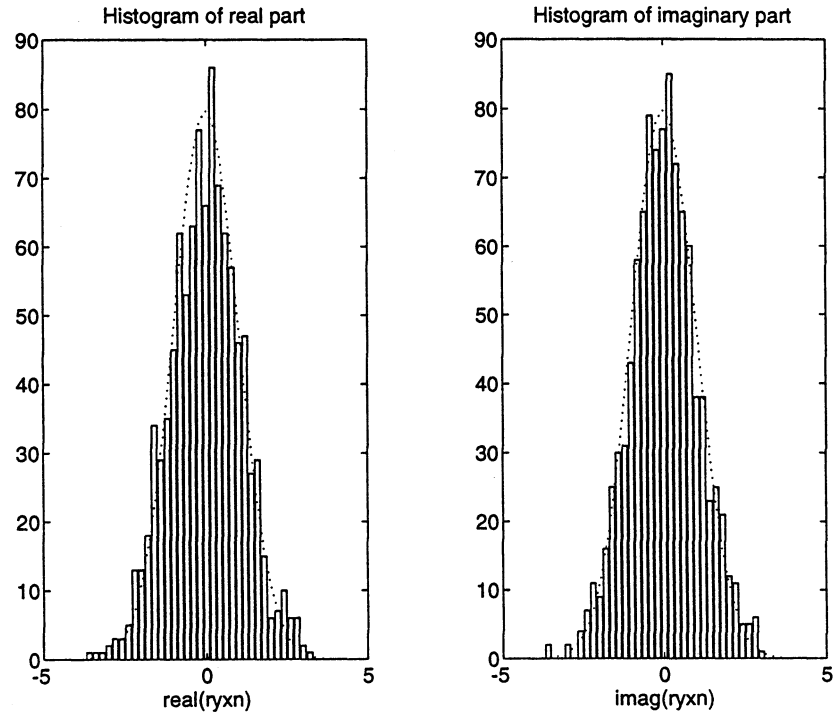



**Figure 2-18: Histograms of the real and imaginary parts of the complex residual of y./x**

The fit is good. The chi-squared value can also be evaluated for both histograms, simply by using the approximate probabilities.

```
i = find((X >= -2)&(X <= 2)); NPiv = fX(i)*dx*np;
chir = sum(((Nhr(i)-NPiv).^2)./NPiv);
chii = sum(((Nhi(i)-NPiv).^2)./NPiv);
fprintf(['E{chi^2} = %.0f, chi^2_real: %.1f,',...
     '  chi^2_imag: %.1f\n'],length(i)-1,chir,chii)

E{chi^2} = 20, chi^2_real: 22.5,  chi^2_imag: 10.4
```

The test shows no significant deviation from the standard normal distribution.

As a last test, let us plot the autocorrelation function of the frequency domain residual series.

```
Cf = real(fft(1/np*abs(ifft(ryxn)).^2));
clg, plot(Cf), title('Frequency domain correlation of ryxn')
xlabel('Indices (through all experiments)')
```

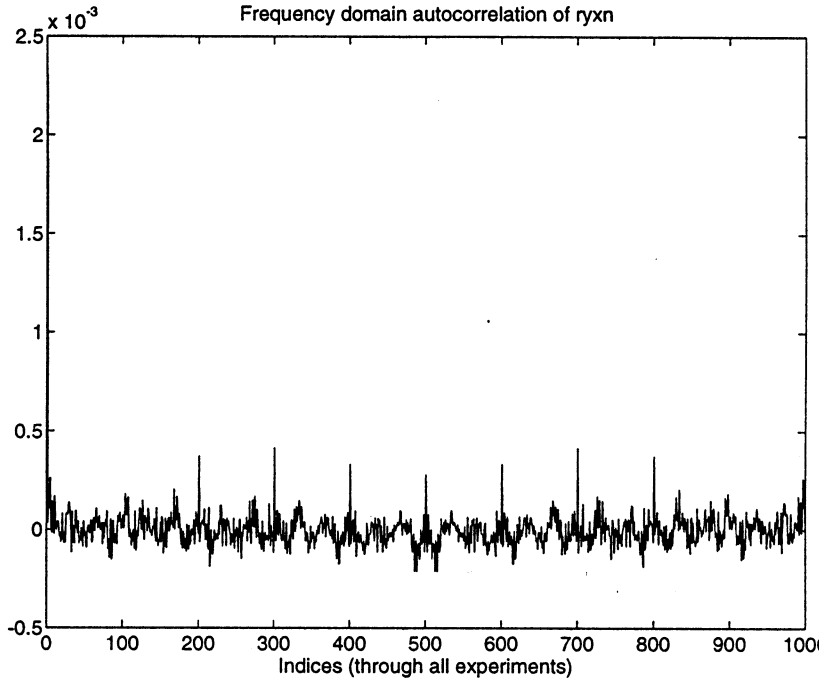



**Figure 2-19: Frequency domain autocorrelation of the residual of the non-parametric estimate y./x**

The autocorrelation function has a dominant peak at zero, an indication of the approximate uncorrelatedness of the residuals. The repeated smaller peaks are at lag distances of experiment lengths each, which indicates a small modeling error again, since it corresponds to a repetitive pattern in the residuals.

A last thing we will try in this session is to make a fit using the input and output variances, but not the covariances, in order to explore what happens in this case.

```
[pv,fit,Cp] = elis(Fdat,[vx,vy]/Na,['s',4,6],[],'',10);
```

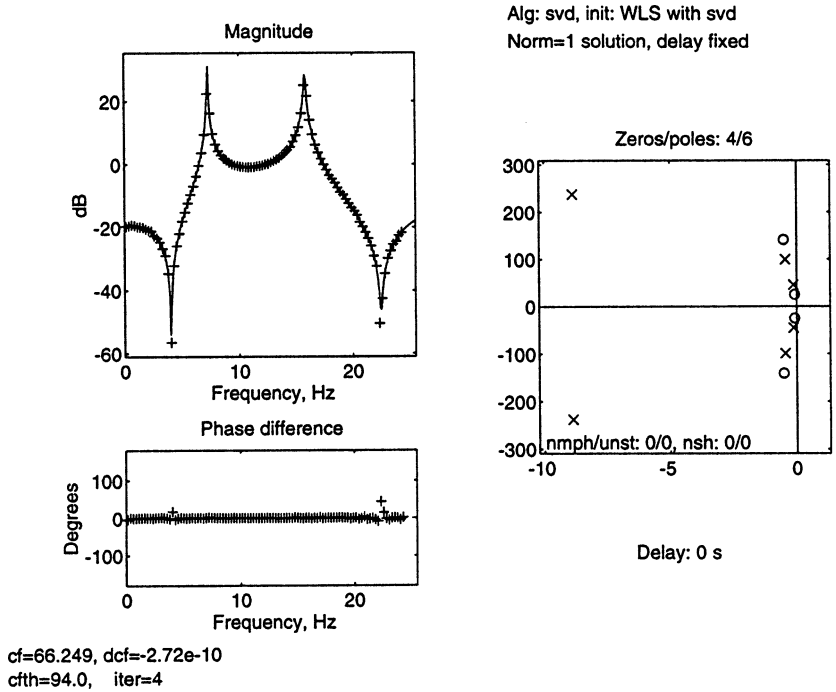



**Figure 2-20: A 4/6 fit without taking input-output covariance into account**

The fit seems to be as good as with the covariances and the cost function is even smaller. But this small cost function is wrong, and has to be avoided. This is one reason why it is advisable to use the covariance values whenever possible; the cost function will only have a reasonable value by application of a correct noise model. But the most important reason for using the covariances is to utilize the available information correctly, with more emphasis on the bands where the amplitudes are measured with smaller error.

# Bibliography

The basic source of the algorithms and ideas used in the Frequency Domain System Identification Toolbox is the book of Schoukens and Pintelon. This book provides a more comprehensive treatment of the methods than this brief tutorial. On general questions of identification — and especially time domain identification — the books of Eykhoff, Godfrey, Goodwin and Payne, Ljung, Norton, Söderström and Stoica are recommended. For the numerical methods, the book of Press et al. is excellent.

P. Eykhoff, *System Identification*, London, John Wiley and Sons, 1974.

K. R. Godfrey, ed., *Perturbation Signals for System Identification*. Englewood Cliffs, Prentice-Hall, 1993.

G. C. Goodwin and R. L. Payne, *Dynamic System Identification: Experiment Design and Data Analysis*, New York, Academic Press, 1977.

L. Ljung, *System Identification: Theory for the User*, Englewood Cliffs, Prentice-Hall, 1987.

J. P. Norton, *An Introduction to Identification*, London, Academic Press, 1986.

W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge, Cambridge University Press, 1986.

J. Schoukens and R. Pintelon, *Identification of Linear Systems: A Practical Guideline for Accurate Modeling*, London, Pergamon Press, 1991.

T. Söderström and P. Stoica, *System Identification*, Englewood Cliffs, Prentice-Hall, 1989.

# Reference

This section contains detailed descriptions of all Frequency Domain System Identification Toolbox functions. It begins with a list of functions grouped by subject area and continues with the reference entries in alphabetical order. Information is also available through the online Help facility.

# Function Tables

Most functions have several default arguments. Under the Syntax heading, the function is listed first with all necessary input arguments, then with all possible input arguments. The functions can also be used with fewer input arguments. Missing trailing arguments are given default values. Default values are also obtained by entering the arguments as empty arrays or strings.

In MATLAB, all output arguments do not need to be specified. Unspecified output arguments are not returned.

## Excitation Signal Design

| Function | Purpose |
|---|---|
| `dibs` | Discrete interval binary sequence design |
| `dibsimpr` | Discrete interval binary sequence improvement |
| `lin2qlog` | Quasi-logarithmic frequency set from linear grid |
| `log2qlog` | Quasi-logarithmic frequency set from log grid |
| `mlbs` | Maximum length binary sequence (PRBS) |
| `msinclip` | Crest factor minimization of multisine |
| `msinprep` | Time domain multisine for download |
| `optexcit` | Excitation signal with optimum power spectrum |

## Preprocessing of Data

| Function | Purpose |
|---|---|
| `modifyfv` | Data prefiltering by inverse of known partial “tf” |
| `tim2fou` | Time domain to frequency domain conversion |
| `varanal` | Variance analysis and averaging of data |

## Estimation

| Function | Purpose |
|---|---|
| `elis` | Iterative minimization of the LS cost function |
| `elisqa` | Generate run parameter settings for `elis` |
| `elrpf2v` | List run parameter file or convert to run parameter vectors |
| `elrpv2f` | List run parameter vectors or convert to run parameter file |
| `gmean` | Geometric mean of complex vectors and numbers |

## Presentation of the Results

| Function | Purpose |
|---|---|
| `expvect` | Export vectors to ASCII files for plotting |
| `gmean` | Geometric mean of complex vectors and numbers |
| `ploteltf` | Plot transfer functions and confidence intervals |
| `plotelpz` | Plot pole/zero patterns with uncertainty ellipses |

## Model Validation

| Function | Purpose |
|---|---|
| **Function** | **Purpose** |
| `fdcovpzp` | Pole-zero model to transfer function conversion |
| `rdueelis` | Calculate residuals after identification |
| `simfou` | Generate simulated frequency domain data |
| `simtime` | Generate simulated time domain data |
| `stdpz` | Calculate zero/pole uncertainties |
| `stdtf` | Calculate transfer function uncertainties |
| `stdtfm` | Calculate uncertainties of $Y_m/X_m$ points |

## Model Conversions

| Function | Purpose |
|---|---|
| `idmodel` | ELiS to SITB model object conversion (Matlab 6.x) |
| `fidmodel` | SITB to ELiS conversion (all Matlab's) |
| `elis2tha` | ELiS to theta format conversion (Matlab 5.x) |
| `tha2elis` | Theta format to ELiS conversion (Matlab 5.x) |

## Data Vector and File Read/Write

| Function | Purpose |
|---|---|
| `expfou` | Write data to Fourier vector or file |
| `exppar` | Write data to parameter vector or file |
| `exptim` | Write data to time domain data vector or file |
| `expvar` | Write data to variance vector or file |
| `expvect` | Export vectors to ASCII files for plotting etc. |
| `impcov` | Read data from covariance vector or file |
| `impfou` | Read data from Fourier vector or file |
| `imppar` | Read data from parameter vector or file |
| `imptim` | Read data from time domain data vector or file |
| `impvar` | Read data from variance vector or file |
| `loadasc` | Load contents of ASCII file into variable |
| `fiddata` | Generate frequency domain data object |
| `tiddata` | Generate time domain data object |
| `fidmodel` | Generate model object |

## Other

| Function | Purpose |
|---|---|
| `fdiddemo` | Demonstrations for the toolbox |
| `fnamanal` | Analysis of filenames |
| `gmean` | Geometric mean of complex vectors and numbers |
| `loadvar` | Load value of single variable from MAT-file |
| `pairs` | Find closest point pairs in two complex vectors |
| `savevar` | Save variable into existing MAT-file |
| `yesinput` | “Intelligent” input function with default value |
| `ywalk` | `yulewalk` without windowing |

# dibs, dibsimpr

## Purpose

Discrete interval binary sequence design.

## Syntax

```
bitser = dibs(N,dt,freqv,ampv)
[bitser,ampopt,Puf,Ptot] = dibs(N,dt,freqv,ampv,trialno,graphmod)
[bitser,ampoptn] = dibsimpr(bits0,dt,freqv,ampv)
[bitser,ampoptn,Puf,Ptot] = ...
   dibsimpr(bits0,dt,freqv,ampv,itno,graphmod)
```

## Description

`dibs` generates a zero-mean discrete interval binary sequence of length `N`, with interval size `dt`, approximating the power spectrum given in `ampv` for the frequency points `freqv`. The algorithm is started `trialno` times from random starting values. The iteration can be followed on the screen, unless `graphmod` is given with a value `'nograph'`.

The bit sequence (values ±1) is returned in `bitser`, and the complex amplitudes of the generated sequence at the points `freqv` in `ampopt`. The complex amplitudes are scaled in such a way that the total power of the designed signal equals the total power `Ptot`, defined by `ampv`.

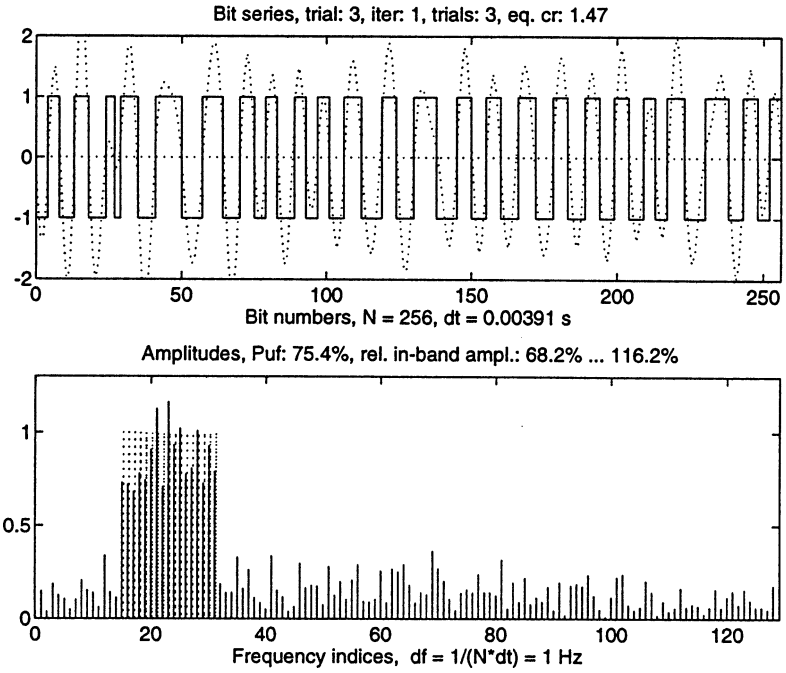
In order to have a measure of the quality of the design, a so-called "equivalent crest factor" is calculated (it is shown in the plots). The basic idea is as follows. The crest factor of a zero-mean binary signal is 1. However, in our case the spectrum is not equal to the desired one, it only approximates it. If the smallest relative amplitude is increased to 1, by amplification of the binary signal, in order to assure that the system is excited at each frequency at least at the desired level, the peak value is multiplied by the reciprocal of the smallest relative amplitude: `eqcr = max(abs(ampv./ampopt))`.

`Puf` gives the useful power (the sum of the power at the desired lines), as a fraction of the total signal power, that is, the theoretical maximum of `Puf` is 1. `Ptot` is the total signal power, calculated from `ampv`.

`dibsimpr` attempts to improve the properties of a discrete interval binary sequence given in `bits0`: it maximizes the smallest relative amplitude of the actual amplitude vector, normalized by `ampv`. The bit series is searched for improvements by changing the sign of a pair of bits: this search is started by `itno` times.

`dibsimpr` may take quite a long time. To make it possible to follow how it proceeds, each already processed bit is marked by a dot on the screen.

A typical example of the plot of `dibs` is shown in the next figure. *trial* is the serial number of the actual trial (or at the end of the iterations the serial number of the one in which the optimum was found), *iter* is the number of the iteration cycles in the given trial, *trials* is the total number of the trials (given in `trialno`), *eq. cr* is the "equivalent crest factor" (see above), *N* is the length of the bit series, and *dt* is the length of the sampling interval.




In the frequency domain plot the desired amplitudes are given by dotted lines, the actual ones by solid lines. *Puf* is the part of the total signal power, which is concentrated at the given frequencies. The minimum and maximum values of the relative amplitudes (actual amplitudes vs. the desired ones) are also given in percents. For these numbers the actual amplitudes are scaled to have the same total signal power as prescribed by `ampv`.

The frequency axis is scaled in “frequency indices,” that is, the unit is *df*, the reciprocal of the period length.

The labels of the plots of `dibsimpr` are similar.

## Default Argument Values

```
ampv = ones(size(freqv)), trialno = 25, graphmod = 'graph', itno = 1.
```

## Examples

Let us assume that a system is to be excited at uniformly distributed frequency points between 500 Hz and 2 kHz, with 100 Hz resolution. A good choice for the sampling frequency is four times the highest harmonic defined in `freqv`. A possible program is as follows:

```
freqv = [500:100:2e3];
fs = 4*2e3; N = round(fs/100); %N = 80
bits0 = dibs(N,1/fs,freqv,[],10);
bitser = dibsimpr(bits0,1/fs,freqv);
```

`bitser` can be directly used for the control of a relay, a thyristor, etc.

## Diagnostics

The sizes of `freqv` and `ampv` must be the same, otherwise an error message is generated:

```
freqv and ampv must have the same size
```

Since the discrete interval binary sequence is periodic, the frequency components must be at the points $k/T$, where $T$ is the period length. If `N`, `dt` and `freqv` are inconsistent, the error message is:

```
a df value in freqv is smaller than 1/(N*dt)
```

The condition of the sampling theorem must be fulfilled, further the elements of `freqv` must be non-negative, otherwise an error message is sent:

```
freqv is out of range
```

`dibsimpr` iterates until the maximum iteration number is reached, or no further improvement is found. In this latter case an information message is sent:

```
dibsimpr cannot further improve signal
```

## Algorithm

`dibs` is based on [1], with the modification that the returned signal is the one with the largest minimum relative amplitude. The algorithm generates a multisine with the amplitudes in `ampv`, takes the sign of the time function, combines the obtained phases with the given amplitudes, generates a new multisine, and so on. The mean value of the binary signal will be kept equal to zero if `N` is even, or will be equal to `±1/N` if `N` is odd.

`dibsimpr` changes the sign of two intervals of length `dt` at a time, observing the change in the minimum relative amplitude [2].

## See Also

`mlbs`

## References

[1] A. van den Bos and R. G. Krol, "Synthesis of Discrete-Interval Binary Signals with Specified Fourier Amplitude Spectra," *International Journal of Control*, 1979, Vol. 30, No. 5, pp. 871-884.

[2] K.-D. Paehlike and H. Rake, "Binary Multifrequency Signals — Synthesis and Application," *Proc. 5th IFAC Symposium on Identification and System Parameter Estimation*, Darmstadt, FRG, Sept. 24-28, 1979. Vol. 1, pp. 589-596.

[3] K. R. Godfrey, ed.: *Perturbation Signals for System Identification*. Englewood Cliffs, Prentice-Hall, 1993.

# elis, elisqa, elrpf2v, elrpv2f

## Purpose

Basic iteration routine to calculate parametric estimate of linear transfer functions (`elis`); treat run parameters (`elisqa`, `elrpf2v`, `elrpv2f`).

## Syntax

```
elis(Fdat,vdat,rppar)
[pvect,fit,Cp,CR,cfv] = ...
     elis(Fdat,vdat,rppar,fixp,rpalg,rppl,initp,rpfs)

elisqa
rpfout = elisqa(rpf,defaults)
elrpf2v(rpfile)
[rppar,fixp,rpalg,rppl,initp,rpfs] = elrpf2v(rpfile)
elrpv2f(rpfile,rppar)
elrpv2f(rpfile,rppar,fixp,rpalg,rppl,initp,rpfs)
```

## Description

`elis` is the routine which performs the desired nonlinear least squares iteration to obtain the parameter estimates. `elisqa` is an optional question/answer routine. `elis` is rather complex, and the possibilities may generally need some explanation; in `elisqa` such explanations are given, and run parameters may be set. `elrpf2v` and `elrpv2f` perform conversions between run parameter vectors and a run parameter file.

The run parameters of `elis` can be set by its input arguments, or they can be passed through a so-called run parameter file, generated by `elisqa`. Most run parameters have default values, thus it is not necessary to give the values of all parameters for every run.

`Fdat` contains the input and output Fourier amplitudes: it may be defined as `[freqv,x,y]` (*N*-by-3 array), where `freqv` is the frequency vector, `x` is the complex input vector, and `y` is the complex output vector; or it may be a compound Fourier vector, generated by `expfou`; or it may be the name of a Fourier file.

`vdat` sets the variance values. It may be an *N*-by-2 or *N*-by-3 array, `[varx,vary]` or `[varx,vary,covxy]`; an 1-by-2 or 1-by-3 row vector, containing the constant variances; a variance vector (see `expvar`); or it may be a string with the name of a variance file.

Instead of either `Fdat` or `vdat`, the name of a run parameter file may also be given: this must be a string which ends by `'.ebn'`. It may not contain any other

period, and the given run parameter file must define the Fourier or the variance data, respectively.

`rppar`, `rpalg`, `rppl`, `rpfs` are run parameter vectors, each composed of a set of run parameters. These prescribe the running of the algorithm, by elements that are either letters or numbers. It does not matter whether the vectors are given as strings or a numerical vectors, `elis` will take care of the proper meanings. The vectors may be shorter than their maximum length, or may contain `NaN` elements; in such a case the default values will be assigned to the corresponding run parameters. Some parameters may be influenced in different ways (e. g., the setting of starting values by `rpalg(2)` and by `initp`); in such cases the last setting will be valid, e. g., the one defined by `initp` in the above case.

`rppar` is the vector of the most often changed run parameters, associated with the model structure.

`rppar(1)` = domain: `'s'` or `'z'`

`rppar(2)` = numord: order of the numerator

`rppar(3)` = denomord: order of the denominator

`rppar(4)` = `fs`: scaling angular frequency (in *s*-domain) or sampling frequency (in *z*-domain)

`rppar(5)` = `'a'` for allpass design.

In the absence of a run parameter file, at least `rppar(1:3)` must be given.

`fixp` defines the fixed parameters. If it is given as an *M*-by-2 array, the first column must contain the serial numbers of the fixed parameters in the vector `[num,denom,delay]'`, and the second column the values. For example, if the numerator order is 4 and the denominator order is 8, the parameter vector has (4+1+8+1+1=15) elements. The first denominator coefficient and the delay can be fixed by `fixp = [6,1;15,0]`. The delay alone can be fixed by `fixp = [15,0]`.

If `fixp` is empty, previously set fixed parameters (by default or by a run parameter file) will not be changed.

A special form is when `fixp` is just a one-character string: `'n'` for no fixed parameters at all (the delay is variable, too); `'f'` for fixed delay, or `'v'` for

variable delay (for these two values the fixing of numerator and denominator coefficients remains untouched). Another special form is `fixp = 'O'`, for which the delay is fixed (to zero, or to the value defined by `initp`) and the zero-order coefficient of the denominator is set to 1.

`rpalg` provides control over the iteration possibilities.

`rpalg(1)` is the type of the iteration algorithm

- `'g'` (Newton-Gauss),
- `'l'` (Levenberg-Marquardt),
- `'m'` (LM with svd),
- `'s'` (singular value decomposition),
- `'r'` (Newton-Raphson)

`rpalg(2) = initset`, way of setting the initial values:

- `'l'` (ordinary least squares),
- `'w'` (weighted least squares),
- `'s'` (WLS with svd),
- `'f'` (file or parameter vector, see `initp`),
- `'e'` (equation error method of the Signal Processing Toolbox)

`rpalg(3) = itmax`: maximum number of iterations

`rpalg(4) = rcostvar`: stop if relative variation of cost function is smaller than this value.

`rpalg(5) = rparvar`: stop if relative variation of all parameters is smaller than this value.

Levenberg-Marquardt settings:

`rpalg(6) = lambdadecr`: number of consecutive decreases of the cost function before trial with `lambda = 0`

`rpalg(7) = lambda`: initial value of lambda

`rpalg(8) = lambdalim`: minimum value of lambda which allows stopping of iteration for small variations of the cost function or of the parameters.

`rppl` controls the plots during the iterations:

`rppl(1) = plotdens`: make plots after every `plotdens` cycle and after the initial setting and the last cycle if `plotdens` is finite; if it is `inf`, no plots will be made at all; if it is negative, the procedure is the same as for a positive number, except that the result of the initial setting will not be plotted.

`rppl(2:5) = [fmin,fmax,amin,amax]`: axis vector for the magnitude plots

`rppl(6) = plotmode`: type of frequency axis, `'i'` for linear, `'o'` for logarithmic

`rppl(7) = calcrnum`: calculate zeros from numerator polynomial, `'c'` or `'n'`

`rppl(8) = calcrdenom`: calculate poles from denominator polynomial, `'c'` or `'n'`

`rppl(9:12)`: axis vector for zero/pole plots. Special meanings (the strings are always 4-character long, with appropriately set trailing spaces): `[n,n,n,n]` (four identical numbers): limit plotted zeros/poles by `n*2*π*max(freqv)` in the *s*-domain, or by just `n` in the *z*-domain; `'a'` to show all zeros and poles; `'p'` to show all zeros and poles, with the same scaling on the two axes.

`rppl(13) = pzfollown`: follow movement of zeros and poles by plotting several zero/pole sets, the number given by `pzfollown`.

`initp` may have different meanings: it may be the parameter vector of the initial parameter values; or a parameter filename; if it is a scalar number, it is the new value of the delay.

`rpfs` may contain the names of files to be generated, in the form of a string array, with trailing spaces if the filenames have different lengths. Possibilities: `*.rep` for `elis` a report file, `*.par`, `*.pbn` or `*.pnt` for generation of a parameter file, and `*.cov`, `*.cbn` or `*.cnt` for a covariance file. Other extensions are not allowed in `rpfs`.

If no argument is given for `elis`, `elisqa` will be started, with the name `elisrpar.ebn`.

`elisqa` offers default answers to the questions, these can be accepted by simply pressing **Enter** or **Return**.

The output arguments of `elis` are as follows.

`pvect` is the calculated parameter vector (see `exppar`).

`fit` is a vector containing information relevant to the results of the fit:

```
fit(1) = cost function
fit(2) = theoretical value of the cost function (c2/2)
fit(3) = 95% point of the theoretical distribution of the cf
fit(4) = number of frequencies
fit(5) = number of free parameters
fit(6) = performed iterations
fit(7) = last change of the cost function
fit(8) = last maximum of the relative changes of parameters
fit(9) = last lambda (for Levenberg-Marquardt, otherwise NaN)
fit(10) = approximate mean model error
fit(11) = mean absolute value of the transfer function
fit(12) = Akaike criterion
fit(13) = condition number of the matrix actually decomposed
          or inverted in the last iteration step
fit(14) = condition number of J
fit(15) = condition number of Q=d^2C/dP^2, inverted when
          calculating the approximate covariance matrix
          (see Eq. (2.23)).
fit(16) = scaling frequency for internal calculations
```

The meaning and significance of the above condition numbers is explained in "Numerical Stability and Speed of the Procedures" on page 2-15.

The approximate mean model error is calculated according to "Detection of Undermodeling and Overmodeling" on page 2-33. The value of $h_{\mathrm{mean}}$ is given, which will be imaginary if the cost function is too small (e. g., because the variances were overestimated, or simply because the cost function may be somewhat smaller than its expected value). This can be best checked from the information given in the report file. $H_{ek}$ and $X_k$ are usually not even nearly constant. In these cases the square of the mean model error is averaged over all the frequencies given.

The mean model error can be compared to the mean absolute value of the transfer function, computed from the values at the same frequency points as above.

`Cp` is the approximate covariance matrix of the parameters (see "Covariance of the Estimate" on page 2-8). `CR` is the approximate Cramér-Rao lower bound for the covariance matrix of the estimates. `Cp` and `CR` are calculated from the

Jacobian of the last iteration (therefore at least one iteration step is necessary to calculate them). The covariances are usable only if the algorithm has converged.

`cfv` is the vector of the values of the cost function in each cycle (the initial cycle included). In the case of the Levenberg-Marquardt method `cfv` has a second column: the values of `lambda` are also given.

`elis` is a rather complex function, and the results often need careful documentation and studying. Therefore, a so-called report file can be requested with extensive textual information on the run (see `rpfs` above).

`elisqa` can be run separately, in order to set run parameters in a file for `elis`. The default run parameters are taken from the file `defaults`, or if this is not given, from the file `rpf`, or if neither of them is given, from an internal table (see below). If `rpf` is not given, the name of the file to be generated will be `elisrpar.ebn`.

The run parameter files are MATLAB binary files, with the extension `'.ebn'`. filenames given without extension will be extended in `elisqa` by `'.ebn'`. In principle, these run parameter files might be modified directly (e. g., by the routine `savevar`), but this is not recommended, since it is easy to make a mistake when doing this. `elisqa` offers a safe and easy way for such modifications.

The user of the routines usually need not bother about the internal names and values of the run parameters. However, the internal run parameters are listed below with their default values.

Sometimes it may be useful to use the same parameters in vector form, as given in a run parameter file, or vice versa. `elrpf2v` and `elrpv2f` serve this purpose. The meanings of the input and output arguments are explained above.

When `elrpf2v` has no output argument, or `rpfile` is empty when invoking `elrpv2f`, the values of the run parameters will be displayed on the screen.

The following paragraphs give a short description of the possibilities and solutions of `elis`, providing more detailed information than was possible in the description of the run parameter vectors.

The measurement data are supposed to be given by `Fdat`, or maybe in a Fourier file. For the internal calculations the frequency vector is scaled by the sampling

frequency in the *z*-domain, or by a scaling frequency in the *s*-domain. The default setting for the scaling frequency is $(\omega_{min} + \omega_{max})/2$.

When defining the model to be fitted, the domain must be specified (*s* or *z*), the orders of the numerator and the denominator must to be given, and the fixed parameters have to be defined (in the transfer function something must be fixed, since multiplication of each coefficient by the same constant gives the same transfer function). If no fixed nonzero parameters are given, `elis` will set the norm (the square root of the sum of squares) of the scaled coefficients (but not the delay) to 1. Another possibility is to set at least one coefficient of either the numerator or the denominator to a fixed value.

The allpass filter is treated as a special case of parameter fixing (the parameters of the denominator are the same as those of the numerator, but in reverse order), however, in this case at least one parameter of the numerator has to be fixed.

You can also specify whether the roots of the numerator and the denominator are to be calculated. When dealing with high order systems (>30), the iteration speed can be increased by sacrificing the pole/zero plot. However, the most significant acceleration can be achieved by completely suppressing plots (`plotdens` set to `inf`), but this has the risk of missing something that could be seen from the plot. A possible compromise is to set `plotdens` to a high value; in this case the starting values and the result of the last cycle will be plotted (independently of the actual serial number of the last cycle).

In the transfer function an extra delay term can be present. The value of this delay can be either fixed or estimated in the iteration procedure. However, a guess of the delay has to be given even if it is to be estimated, since the initial value may seriously influence the convergence properties.

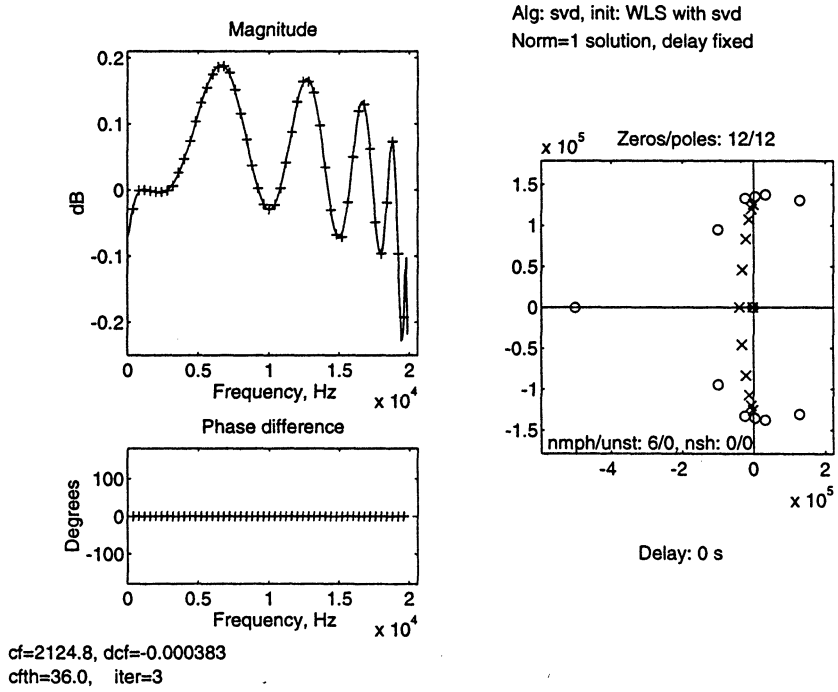
The numerical methods solving the nonlinear LS problem (see “Basic Concepts” on page 2-2) are standard methods in numerical analysis, including Newton-Gauss, Newton-Raphson, singular value decomposition (for the Newton-Gauss formulation), Levenberg-Marquardt and Levenberg-Marquardt with singular value decomposition. In the Levenberg-Marquardt algorithm an identity matrix, multiplied by `lambda` and by the Frobenius norm of $\mathbf{J}^T\mathbf{J}$, is added to $\mathbf{J}^T\mathbf{J}$ before inversion; if a better fit is found, `lambda` is divided by 2, otherwise it is multiplied by 10 and the previous parameter values are restored. After `lambdadecr` successive divisions, an attempt is made with `lambda = 0`. (This corresponds to a Newton-Gauss step).

The singular value decomposition is always applied to **J**, even if it is combined with Levenberg-Marquardt, in order to make full use of the power of the singular value decomposition algorithm.

There are a few different possibilities to set the initial values for iteration. The first one is the standard initial setting procedure of ELiS: the linear least squares fitting. This can be modified to a weighted LS problem (weighting by the variances of the complex output amplitudes), and can also be solved by singular value decomposition. There are two more possibilities: the initial values can be taken from a parameter vector or file (e. g., the result of an earlier fit or of another method can be used), and also the *equation error method* of the Signal Processing Toolbox can be applied for initial value determination for `elis` (see `invfreqz` or `invfreqs`). In this latter case the input noise is transformed to the output as if a compound output noise was present.

For the control of the iterations, the maximum number of iteration cycles, the minimum relative variations of the cost function and of the estimated parameters can be set. The iteration will be terminated when the maximum number of iteration cycles is reached, or when any of the absolute values of the above variations is less than the corresponding minimum value. In the case of the Levenberg-Marquardt method the value of lambda is also considered: the iteration is terminated because of small variations only if the value of `lambda` is also smaller than `lambdalim`. The default value (`1e-10`) corresponds to 30 halvings of the default starting value 0.1.

A typical plot of the function `elis` is shown in the figure.




The left-hand side illustrates the fitting of the transfer function. The diagram consists of two parts. The upper part shows the absolute values of the ratios of the complex output and input amplitudes (+ marks), along with the magnitude of the estimated transfer function. The lower part shows the phase *errors* between the ratios of the measured complex amplitudes and the estimated transfer function. The plot is scaled vertically to (–180°,180°).

As a default, the pole/zero plot is also displayed at the right-hand side. The numbers of poles and zeros, the numbers of non-minimal phase zeros and unstable poles are also given, along with the number of zeros/poles occasionally not shown in the plot.

When `elis` is terminated, the current `axes` is the pole/zero plot. This can be manually rescaled if desired, but the text about unstable poles etc., has to be deleted first:

```
h = get(gca,'ch'); delete(h(1))
```

A somewhat simpler solution is to reapply `plotelpz` with a given axis vector:

```
plotelpz(pvect,axv,'','','nomsg')
```

If desired, even the uncertainty ellipses can be added:

```
plotelpz(pvect,axv,Cp,2,'nomsg')
```

The screen contains some further information concerning the run: the value of the cost function, the last change of the cost function, the theoretical expected value of the cost function (number of frequencies minus half of the number of free parameters), cycle number, iteration algorithm and way of setting the initial conditions, and the value of the delay.

A report file, that can be requested when running `elis`, will contain something like this:

```
FREQUENCY DOMAIN SYSTEM IDENTIFICATION TOOLBOX FOR MATLAB
ELiS RUN PROTOCOL, date and time: 1-Nov-93, 18:55:1
Report file: elisrpar.rep
Default run parameters, modified in command line

Fourier data given in command line
Experiment: 1, number of frequencies: 30
Input and output variances:
   5e-07   1.4754e-07
Input-output covariances are not given
Fit in s-domain, frequencies normalized internally
   by omegasc = 2.0452 rad*Hz
   suggestion: omegasc = 1.4408 rad*Hz
Orders: 1/3
No fixed nonzero parameters (norm=1 solution)
Fixed value of the delay:  0 s
Algorithm: Singular value decomposition
Initial value setting: WLS, by singular value decomposition

Allowed maximum number of iterations: 50,
   iterations performed: 4
Total run time: 0.56 min, time used for plots: 0.51 min
   time used for pole/zero calculations: 0.00 min
Stop if relative change of cost function is smaller than
   1.00e-06,  last relative variation: -1.59e-14
Stop if maximum relative change of parameters is smaller
   than 0.00e+00,  last max. rel. variation: +2.33e-11
Condition number of the decomposed or inverted matrix:
                       54.176, reciprocal: 0.018459
Condition number of J: 54.176, reciprocal: 0.018459
Condition number of Q=d^2C/dp^2:
                       2932.8, reciprocal: 0.00034097
                                     eps = 2.220e-16
Value of the cost function: 30.785, its double: 61.57
Theoretical value of the cost function: 27.0
Degrees of freedom of the chi-square value (2*cfth): 54
5%-95% points of the theoretical distribution of the cf:
```

```
   17.789, 38.098
Approximate mean model error: 0.00027292
Mean absolute value of the transfer function: 0.40298
Approximate relative mean model error: 0.00067726
Akaike criterion: 73.57
Number of free parameters: 6

Values of the parameters (with standard deviations,
   calculated from the approximate covariance matrix):
Numerator:
s^0   -7.035487766480046e-02    std: 1.7580e-04  (0.24987%)
s^1   -7.038203864830019e-02    std: 5.6617e-05  (0.080443%)
Denominator:
s^0   -2.813969892740070e-01    std: 5.4478e-04  (0.1936%)
s^1   -2.110973116935604e-01    std: 1.6580e-04  (0.07854%)
s^2   -1.407034079272249e-01    std: 1.8950e-04  (0.13468%)
s^3   -7.037131510238287e-02    std: 9.1545e-05  (0.13009%)
Delay:
0 s    fixed

Zeros (rad*Hz):
-9.9961e-01

Poles (rad*Hz):
-1.6501e+00
-1.7467e-01 -1.5469e+00*j
-1.7467e-01 +1.5469e+00*j

First nonzero numerator coefficient is b(1)=-7.0382e-02
First nonzero denominator coefficient is a(3)=-7.0371e-02
b(1)/a(3)=1.0002e+00
Static gain: 2.5002e-01, -12 dB

Parameter file to save data: -
Covariance file to save data: -

%%%%% End of report file elisrpar.rep %%%%%
```

## Default Argument Values

Here is the list of the run parameters and their default values (the possible answers are given between parentheses):

```
Ffile = ''; %name of Fourier file
vfile = ''; %name of variance file
pfile = ''; %parameter filename (maybe empty)
cfile = ''; %name of the covariance file to be generated
rfile = ''; %name of the report file to be generated
initpfile = ''; %name of param file, with initial values
algtype = 'NG'; %iteration algorithm (NG,LM,LMsvd,NR,svd)
amin = NaN; %axisvect(3) for magnitude plot
amax = NaN; %axisvect(4) for magnitude plot
calcrnum = 'c'; %calculate roots of numerator (c,n)
calcrdenom = 'c'; %calculate roots of denominator (c,n)
covxy = []; %input-output covariance
delay = 0; %(initial) value of the delay
delaytreat = 'f'; %delay fix (f) or variable (v)
denomord = 2; %order of the denominator
denomfix = []; %fixed denominator coefficients
denomfixind = []; %fixed denominator coefficient indices
domain = 's'; %domain of the model (s,z)
expi = 1; %serial number of the experiment to be used
fmin = 0; %lower bound of displayed frequencies
fmax = NaN; %max. displayed frequency in the s-domain
fs = NaN; %sampling (normalizing) frequency
initset = 'l'; %initial value setting (l,w,s,f,e)
itmax = 50; %maximum number of iteration cycles
lambda0 = .1; %starting lambda value for Levenberg-Marquardt
lambdadecr = 10; %after 10 consecutive decreases, 0 is tried
lambdalim = 1e-10; %iteration may stop below this value
numord = 1; %order of the numerator
numfix = []; %fixed numerator coefficients
numfixind = []; %fixed numerator coefficient indices
paramtreat = 'n'; %No fixed params (n), some are fixed:
   %(0, d, r) or allpass filter is designed (a)
plotdens = 1; %cycles of iteration to plot (1,2...inf)
pzfollown = 1; %pole/zero sets to be plotted on same plot
pzlimit = 'y'; %limit poles/zeros on plot to
   %pzlimitv*2*π*fmax (s) or 2 (z)
pzlimitv = 10; %limit poles/zeros on plot (see pzlimit)
```

```
rcostvar = 1e-6; %minimum relative variation of the cf
rparvar = 0; %minimum relative variation of parameters
vardef = 'u'; %variances: numbers or take from file (u,f)
varx = 1; %uniform variance value of input coefficients
vary = 1; %uniform variance value of output coefficients
```

Default values of the run parameter vectors:

```
rppar = [NaN,NaN,NaN,NaN];
rpalg = ['g','w',50,1e-6,0,10,0.1,1e-10];
rppl = [1,0,NaN,NaN,NaN,'i','c','c','a    ',1];
rpfs = '';
```

The `NaN` values mean that the actual values will be controlled by the Fourier data.

## Examples

```
[freqv,x,y] = impfou('bandpass.fbn',1);
elis([freqv,x,y],3.4e-4*[1,1],['s',4,6]);
elis('inpchans.ebn');
elis('inpchanz.ebn',[],['z',16,16],[35,0]); %fixing the delay

elisqa('elisqtst.ebn','inpchans.ebn');

[rppar,fixp,rpalg,rppl,initp,prfs] = elrpf2v('inpchans.ebn');
elrpf2v('inpchans.ebn')

elrpv2f('newfile.ebn',['z',12,12],[0],'r');
elrpv2f('',['z',12,12],'0','r');
```

## Diagnostics

The error and warning messages are self-explanatory. For the messages about condition numbers, see “Numerical Stability and Speed of the Procedures” on page 2-15.

## Algorithm

The algorithm and the main expressions are briefly described in Chapter 2, or in detail in [1].

## References

[1] J. Schoukens and R. Pintelon, *Identification of Linear Systems: a Practical Guideline for Accurate Modeling,* London, Pergamon Press, 1991.

**Function eliminated:  use fiddata, tiddata or fidmodel instead**

## Purpose

Model conversion from/to the theta-format of the System Identification Toolbox

## Syntax

```
theta = elis2tha(pdat,cdat,varet)
theta = elis2tha(pdat,cdat,varet,numn,denomn)
[num,denom,delay,fs] = tha2elis(theta)
[num,denom,delay,fs,vary,ccovar] = tha2elis(theta,N,freqv)
```

## Description

These are the two routines which convert models of the Frequency Domain System Identification Toolbox from/to the System Identification Toolbox. `pdat` is the parameter vector in ELiS (see `exppar`), or the name of a parameter file. `cdat` is the covariance matrix of the parameters if it is an array (the rows and columns of the fixed parameters should contain zeros), or the covariance vector (see `expcov`), or the name of the covariance file. `cdat` may be empty, if the covariances are not available.

`varet` is the variance of the time domain noise, reduced to the output of the system, without noise shaping.

If the input noise is zero, and the variances of the real and of the imaginary parts of the complex amplitudes are uniformly equal to `vary` in the $N$-point spectrum, `varet` should be calculated as: `varet = 2/N*vary`, and `numn` and `denomn` need not be given.

`numn` and `denomn` are the numerator and the denominator of the observation noise shaping filter (see "Model Conversions from/to the System Identification Toolbox" on page 2-41). If the frequency domain noise shape is fitted by the noise shaping filter:

$$\left|\frac{C(z_k)}{C(z_k)}\right|^2 \approx \mathrm{vary}(f_k),\ z_k = e^{j2\pi f_k/f_s}$$

and $C$ corresponds to `numn`, $D$ to `denomn`, the value of `varet` should be `2/N`. The sampling frequency for the noise shaping filters is the same as that of the parameter vector.

When transforming models from the theta-format of the System Identification Toolbox, `theta` is the array of the theta-format to be converted, and `N` is the number of FFT points (for the calculation of `vary` only). `vary` is a scalar if $C(z) \equiv D(z) \equiv 1$, and a vector if the noise is not white. `freqv` is the vector of

frequencies where the variances are to be calculated. If `vary` is not required, `N` and `freqv` are not necessary.

The output arguments of `tha2elis` are as follows: `num` is the numerator, `denom` is the denominator of the transfer function, `delay` is the additional delay in the model, `fs` is the sampling frequency, `vary` is the vector of frequency domain variances of the output complex amplitudes at the frequencies in `freqv` (or just the frequency domain variance if this is constant), and `ccovar` is the covariance matrix of the vector `[num,denom,delay]`.

## Default Argument Values

```
numn = 1, denomn = 1
```

## Examples

```
theta = elis2tha('inpchanz.pbn','inpchanz.cbn',2/256*1e-9);
[num,denom,delay,fs] = tha2elis(theta);
```

## Diagnostics

As it is discussed in "Model Conversions from/to the System Identification Toolbox" on page 2-41, the covariances can usually be converted only by using linear approximation of a ratio. This is done automatically by the conversion routines, but if approximation was applied, a warning message is sent:

```
WARNING: variance of denom(1) is not zero in elis2tha
```

or

```
WARNING: ccovar will be approximated in tha2elis
```

If in `elis2tha` the variance of `denom(1)` is too large, that is, larger than `0.2*denom(1)^2`, the calculated variances are useless. In this case an error message is sent:

```
var(denom(1)) is too large
```

It is checked in `tha2elis` whether the approximated covariances are plausible. If not, the warning message is:

```
WARNING: In tha2elis the approximated covariances are too large
```

`elis2tha` calls `poly2th` or `mktheta`, and `tha2EliS` calls `th2poly` or `polyform`, thus the System Identification Toolbox must be installed.

## See Also

System Identification Toolbox

## Purpose

Read/write ELiS covariance vectors and/or ELiS covariance files; furthermore, extend covariance matrix of free parameters by zero rows/columns of fixed parameters, or delete zero rows and columns, belonging to fixed parameters.

## Syntax

```
cvect = expcov(coeffcovar,fixpind)
[cvect,Cp] = expcov(coeffcovar,fixpind,... filename,comments,fdate)
coeffcovar = impcov(cdat)
[coeffcovar,fixpind,comments,fdate] = impcov(cdat,nofixp)
```

## Description

`expcov` and `impcov` perform conversions of different representations of the covariance matrix of parameters. The structure of the ELiS covariance vector and the file format are described in Appendix A1.

`cvect` is the vector of covariances, `Cp` is the covariance matrix of all the parameters (both estimated and fixed ones).

`coeffcovar` is an $n$-by-$n$ array, the covariance matrix of the coefficients. If `fixpind` is not given, $n$ is equal to the number of the numerator coefficients plus the number of the denominator coefficients plus one (for the delay). If `fixpind` is given, `coeffcovar` should either not contain the rows and columns belonging to fixed coefficients, or these rows and columns should consist of zeros. `fixpind` contains the indices of fixed parameters in the total parameter vector, defined as `[num,denom,delay]'`, where the coefficients are in descending order of powers of $s$ in the $s$-domain, and in ascending order of the powers of $z^{-1}$ in the $z$-domain.

If `fixpind` is given, and in `coeffcovar` there are zero rows and columns, the two notations must correspond to each other.

`filename` is the name of the covariance file. If `filename` is missing or empty, no file will be generated.

If the generation of a file is requested, the file will be created in the active subdirectory or folder. If the name has no extension, `expcov` extends it by `'.cbn'`. If the extension is `.cbn`, the result will be a binary file, otherwise an ASCII file. If the extension is `'.cnt'`, no text is sent to the ASCII file, only data.

If a file is written or read, the most important values will be displayed on the screen, unless a global variable `expimpmessages` with value `'no'` is defined.

Covariance values will be written to ASCII files by `expcov` in floating-point form, using 8 digits in the mantissas.

`comments` is a string with comments (optional), and `fdate` is the date (and time) string (also optional). If `fdate` is missing or empty, a date string will be generated.

`impcov` is the inverse of `expcov`. The data vector or the name of the file is `cdat`; the data vector contains the data in the same order as a `*.cnt` file. If `nofixp` is given with the value `'nofixp'`, the zero rows and columns in `coeffcovar` will be deleted. The default extension is `.cbn`.

## Default Argument Values

```
nofixp = ''
```

## Examples

```
coeffcovar = eye(5); expcov(coeffcovar,[],'data.cbn');
[coeffcov,fixpind] = impcov('inpchans.cbn','nofixp');
```

## Diagnostics

`expcov` checks whether `coeffcovar` is quadratic, real, finite and symmetric. Also the validity of `fixpind` is checked.

If a file already exists with the same name, `expcov` tries to delete it. This will be unsuccessful if the file is not in the active subdirectory/folder. In this case the error message is:

```
Cannot delete existing file ...
```

`impcov` checks the length of the covariance vector form: if it is not valid, the error message is sent:

```
Number of data is not n*(n+1)/2
```

## See Also

"Description of the Data Vector and File Formats" on page A-2

## Purpose

Read/write ELiS Fourier vectors and/or ELiS Fourier files.

## Syntax

```
Fvect = expfou(freqvect,x,y)
Fvect = expfou(freqvect,x,y,i,expno,...
   filename,comments,fdate,digitnum)
[freqvect,x,y] = impfou(Fdat)
[freqvect,x,y,expno,comments,fdate] = impfou(Fdat,expi)
```

## Description

`expfou` exports data of one experiment (or several experiments) to a vector in the workspace, and/or to a (perhaps already existing) Fourier file.

The output argument `Fvect` is the vector containing the same data as would be sent to a `*.fnt` file.

`freqvect` is the vector of frequency points, `x` is the complex input amplitude vector (or array for multiple inputs), and `y` is the complex output amplitude vector (or array for multiple outputs). Each amplitude vector must be a column vector. The amplitudes belonging to different experiments have to be put under each other. `x` or `y` may be empty, but both of them have to be given.

`expno` is the total number of experiments. `i` contains the number(s) of the actual experiment(s). If several experiments are given, they must be denoted by successive numbers. If `i` is empty, the default is `i = [1:length(x(:,1))/length(freqvect)]`.

The string `filename` contains the name of the output file. If this name has no extension, `expfou` extends it by `'.fbn'`. If the extension is `.fbn`, the result will be a binary file, else an ASCII file. If the extension is `.fnt`, no text is sent to the ASCII file, only data.

The file will be created in the active subdirectory or folder. If `filename` is empty, no file will be generated.

If a file is read or written, the most important values will be displayed on the screen, unless a global variable `expimpmessages` with value `'no'` is defined.

When exporting the data of the first experiment, any file with the same name will be deleted.

`comments` is a string with comments (optional), and `fdate` is the date (and time) string (also optional). If `fdate` is missing or empty, a date string will be generated.

Amplitude values will be written to ASCII files by `expfou` in floating-point form. `digitnum` is the number of digits of mantissas, sent to ASCII files, with the limits `1 ð digitnum ð 16`, and default value 7.

`impfou` reads complex amplitudes from Fourier vectors or files. The file may be an ASCII file with comments (usual extension: `.fou`), a so-called flat ASCII file without comments (`.fnt`), or a binary file (`.fbn`). The file has to be somewhere within the path of MATLAB, or the path is to be explicitly given. The default extension is `.fbn`.

The input argument `Fdat` of `impfou` is the data vector, or the name of a file, or may be an array of size *F*-by-3 in order to be able to process directly any output format of other M-files. The data vector contains the data in the same order as a `*.fnt` file, and the array is `[freqv,x,y]`.

`expi` contains the number(s) of the experiment(s) to be read (integer vector). `expi` is optional; if it is omitted or empty, all the experiments will be read.

## Default Argument Values

```
digitnum = 7,  i = [1:length(x(:,1))/length(freqvect)],
expno = max(i)
```

## Examples

```
expfou([1:20],ones(100,1),0.1*ones(100,1),...
          [1:5],5,'data.fbn');
[freqvect,x,y,expno] = impfou('data.fbn');
```

Send results of experiments in two steps:

```
Fvect = expfou([1:20],ones(60,1),0.1*ones(60,1),[1:3],5);
Fvect = [Fvect;expfou([1:20],...
     ones(40,1),0.1*ones(40,1),[4,5],5)];
[freqvect,x,y,expno] = impfou(Fvect);
```

## Diagnostics

If not experiment No. 1 is being exported to a file, `expfou` looks for an already existing file. If this is not found, the error message is

```
File has to exist already when exporting ... experiment
```

If a file already exists with the same name, and the first experiment is being exported, `expfou` tries to delete it. This will be unsuccessful if the file is not in the active subdirectory/folder. In this case the error message is:

```
Cannot delete existing file ...
```

`impfou` checks the validity of the contents of the vector (file). If it is not consistent, an error message is sent:

```
Not enough data in file ...
```

or if the vector is too long, a warning message is displayed:

```
Number of data is incorrect in file ...
```

## See Also

"Description of the Data Vector and File Formats" on page A-2

# exppar, imppar

## Purpose

Read/write ELiS parameter vectors and/or ELiS parameter files.

## Syntax

```
pvect = exppar(domain,num,denom)
pvect = exppar(domain,num,denom,delay,fs,...
      filename,comments,fdate)
[domain,num,denom,delay,fs] = imppar(pdat)
[domain,num,denom,delay,fs,comments,fdate] = imppar(pdat,fsc)
```

## Description

`exppar` writes parameters of the transfer function to a vector in the workspace, and/or to parameter files (used by `elis`).

`pvect` is the vector containing the same data as the `*.pnt` file.

domain may be `'s'` or `'z'`, depending on the domain; `num` is the numerator vector, $z$-domain coefficients in ascending order of powers of $z^{-1}$, or $s$-domain coefficients in descending order of powers of $s$. `denom` is the denominator vector, similarly to `num`. `delay` is the additional `delay`, `fs` is the sampling frequency in the $z$-domain, or the scaling frequency between the internal representation and the $s$-domain parameter vector or file, which is written in standard SI units.

The string variable `filename` is the name of the file. If the name has no extension, this function extends it by `'.pnt'`. If the extension is `.pbn`, the result will be a binary file, else an ASCII text file. If the extension is `.pnt`, no comment is sent to the ASCII file, only data.

The file will be created in the active subdirectory or folder. If a file is written or read, the most important values will be displayed on the screen, unless a global variable `expimpmessages` with value `'no'` is defined. If `filename` is empty, no file will be written.

The parameters will be written to ASCII files in floating-point form by `exppar`, with 16-digit accuracy of the mantissas.

`comments` is a string with eventual comments (optional), and `fdate` is a date (and time) string (also optional). If `fdate` is missing or empty, an actual date string will be generated.

In an $s$-domain parameter vector or file, the saved `fs` value is the so-called suggested scaling frequency, usable in a later import. The algorithm uses a

simple ad hoc formula to find a scaling frequency, which brings the roots of the numerator and denominator polynomials possibly close to 1:

$$\mathrm{fs} = \frac{2}{3}\frac{1}{nn+nd}\left(\sum_{i=0}^{nn-1}\left|\frac{b_i}{b_{nn}}\right|^{\frac{1}{nn-i}} + \sum_{i=0}^{nd-1}\left|\frac{a_i}{a_{nd}}\right|^{\frac{1}{nd-i}}\right)$$

When the numerator or the denominator is degenerate, the corresponding term is set to 1; if the formula would still give an unusable result, like `inf` or `NaN`, `fs` is set to 1.

`imppar` reads parameters from parameter vectors or files (used by `elis`). The file may be an ASCII file with comments (usual extension: `.par`), a flat ASCII file without comments (`.pnt`) or a binary file (`.pbn`). The file has to be somewhere within the path of MATLAB, or the path is to be explicitly given. The default extension is `.pnt`.

As an output argument of `imppar`, `fs` is the sampling frequency in the *z*-domain, while in the *s*-domain it is the scaling frequency between the internal representation and the parameter vector or file which is written in standard SI units. `pdat` is the data vector or the name of the file; the data vector contains the data in the same order as a `.pnt` file.

The scaling frequency between the *s*-domain internal representation and the parameter file can be set by `fsc`. If `fsc` is not given, no scaling will be performed (`fsc = 1`), if it is empty (`fsc = []`), the previously saved value in the parameter vector or file is used. In the case of *z*-domain files, `fsc` is ignored.

## Default Argument Values

```
delay = 0, fs = 1, fsc = 1
```

## Examples

```
pvect = exppar('s',[1,1],[1,2,3,4]);
exppar('z',[1,1],[4,3,2,1],0,1,'filter.pnt',...
          'First trial',date);
[domain,num,denom,delay,fs] = imppar('filter.pnt');
[domain,num,denom,delay,fsc] = imppar('inpchans.pbn',[]);
```

## Diagnostics

If a file already exists with the same name, `exppar` tries to delete it. This will be unsuccessful if the file is not in the active subdirectory/folder. In this case the error message is:

```
Cannot delete existing file ...
```

`imppar` checks the contents of the vector (file); if this is inconsistent, an error message is sent:

```
Not enough data in file ...
```

## See Also

"Description of the Data Vector and File Formats" on page A-2

## Purpose

Read/write ELiS time domain data vectors and/or ELiS time domain files.

## Syntax

```
tvect = exptim(timevect,xt,yt)
tvect  exptim(timevect,xt,yt,i,expno,...
        filename,comments,fdate,digitnum)
[timevect,xt,yt] = imptim(tdat)
[timevect,xt,yt,expno,comments,fdate] = imptim(tdat,expi)
```

## Description

`exptim` exports data of one experiment (or several experiments) to a vector in the workspace, and/or to a (perhaps already existing) time domain data file. The output argument `tvect` is the vector containing the same data as would be sent to a `*.tnt` file.

`timevect` is the vector of sampling time instants, `xt` is the input vector (or array for multiple inputs), and `yt` is the output vector (or array for multiple outputs). Each vector must be a column vector. The data belonging to different experiments have to be put under each other. `xt` or `yt` may be empty, but both of them have to be given.

`expno` is the total number of experiments, `i` contains the number(s) of the actual experiment(s). If several experiments are given, they must be denoted by successive numbers. If `i` is empty, the default is `i = [1:length(xt(:,1))/length(timevect)]`.

The string `filename` contains the name of the output file. If this name has no extension, exptim extends it by `'.tbn'`. If the extension is `.tbn`, the result will be a binary file, else an ASCII file. If the extension is `'.tnt'`, no text is sent to the ASCII file, only data.

The file will be created in the active subdirectory or folder. If `filename` is empty, no file will be written.

If a file is read or written, the most important values will be displayed on the screen, unless a global variable `expimpmessages` with value `'no'` is defined.

When exporting the data of experiment No. 1, any file with the same name will be deleted.

`comments` is a string with eventual comments (optional), and `fdate` is the date (and time) string (also optional). If `fdate` is missing or empty, an actual date string will be generated.

Amplitude values will be written to ASCII files by `exptim` in floating-point form. `digitnum` is the number of digits of the mantissas, sent to ASCII files, with the limits `1 ð digitnum ð 16`, and default value 7.

`imptim` reads time records from time domain data vectors or files (used by `elis`). The file may be an ASCII file with comments (usual extension: `.tim`), a so-called flat ASCII file without comments (`.tnt`), or a binary file (`.tbn`). The file has to be somewhere within the path of MATLAB, or the path is to be explicitly given. The default extension is `.tbn`.

The input argument `tdat` of `imptim` is the data vector or the name of the file, or maybe an array of size `tl×3`, `[timevl,xt,yt]`; the data vector contains the data in the same order as a `*.tnt` file.

`expi` contains the number(s) of the experiment(s) to be read (integer vector). `expi` is optional; if it is omitted or empty, all the experiments will be read.

## Default Argument Values

```
digitnum = 6,  i = [1:length(xt(:,1))/length(timevect)],
expno = max(i)
```

## Examples

```
tvect = exptim([1:10],ones(50,1),0.1*ones(50,1),...
     [1:5],5,'data.tbn');
[timevect,xt,yt,expno] = imptim('data.tbn');
```

## Diagnostics

If not experiment No. 1 is being exported to a file, `exptim` looks for an already existing file. If this is not found, the error message is

```
File has to exist already when exporting ... . experiment
```

If a file already exists with the same name, and the first experiment is being exported, `exptim` tries to delete it. This will be unsuccessful if the file is not in the active subdirectory/folder. In this case the error message is:

```
Cannot delete existing file ...
```

`imptim` checks the validity of the contents of the vector (file). If it is not consistent, an error message is sent:

```
Not enough data in file ...
```

or if the read vector is too long, a warning message is displayed:

```
Number of data is incorrect in file ...
```

**See Also** "Description of the Data Vector and File Formats" on page A-2

# expvar, impvar

## Purpose

Read/write ELiS variance vectors and/or ELiS variance files.

## Syntax

```
vvect = expvar(varx,vary)
vvect = expvar(varx,vary,covxy,filename,comments,fdate,Ffile)
[varx,vary] = impvar(vdat)
[varx,vary,covxy,comments,fdate] = impvar(vdat)
```

## Description

`expvar` writes variance data to variance vectors or files.

The output argument `vvect` is the vector containing the same data as the `.pnt` file.

`varx` is a column vector containing the variances of the real part (that is, also of the imaginary part) of the input Fourier coefficients; `varx` is an array for multiple inputs. `vary` is the column vector containing the variances of the real part (that is, also of the imaginary part) of the output Fourier coefficients; `vary` is an array for multiple outputs.

`covxy` is the column vector of input-output covariances, `covxy = E{conj(Nx)*Ny}`. For multiple inputs or outputs, `covxy` contains just the covariances between input(1) and output(1), or all the covariances beside each other, as `[ci1o1,ci1o2,...ci2o1...]`. `covxy` may be empty.

`filename` is the name of the output file. If the name has no extension, `expvar` extends it by `'.vbn'`. If the extension is `'.vbn'`, the result will be a binary file, else an ASCII file. If the extension is `'.vnt'`, no text is sent to the ASCII file, but data.

The file will be created in the active subdirectory or folder. If a file is written or read, the most important values will be displayed on the screen, unless a global variable `expimpmessages` with value `'no'` is defined. If `filename` is empty, no file will be written.

Variance values will be written to ASCII files by `expvar` in floating-point form, using 4 digits in the mantissas.

`comments` is a string with eventual comments (optional), `fdate` is the date (and time) string (also optional). If `fdate` is missing or empty, an actual date string will be generated.

`Ffile` is the associated Fourier vector or file (optional, for cross-checking).

`impvar` reads variances from variance vectors or files.

`vdat` is a vector, or the name of the file, or an array of size *F*-by-2 or *F*-by-3, `[varx,vary]` or `[varx,vary,covxy]`; the vector contains the data in the same order as a `*.vnt` file.

The file may be an ASCII file with comments (usual extension: `.var`), an ASCII file without comments (`.vnt`), or a binary file (`.vbn`). The file has to be somewhere within the path of MATLAB, or the path is to be explicitly given. The default extension is `.vbn`.

## Examples

```
expvar(ones(10,1),0.1*ones(10,1),zeros(10,1),'data.vbn');
[varx,vary] = impvar('data.vbn');
```

## Diagnostics

If a file already exists with the same name, `expvar` tries to delete it. This will be unsuccessful if the file is not in the active subdirectory/folder. In this case the error message is:

```
Cannot delete existing file ...
```

If `Ffile` is given, `impvar` compares the lengths of the frequency vector and the input and output variance vectors. If they are inconsistent, the error message is:

```
Data incompatible with Fourier file
```

`expvar` and `impvar` also check the variance vectors for negative, complex, infinite elements.

## See Also

"Description of the Data Vector and File Formats" on page A-2

# expvect

## Purpose

Export MATLAB vectors to ASCII files for plotting.

## Syntax

```
expvect(file,v1)
expvect(file,v1,digitnum)
expvect(file,v1,v2,v3,v4,v5,v6,v7,v8,v9,v10,v11,v12,digitnum)
```

## Description

The contents of the real vectors `v1`, `v2`, ... will be exported into a flat ASCII file (file), for further processing by plotting/graphing programs, or for transfer to other computers, or for sending by electronic mail, etc. The vectors will be written into an array, where the first line will contain the first elements of all the vectors and so on. The delimiter between the numbers in a row is tab. The last character in each line is a `CR/LF` or a `CR`.

`digitnum` defines the number of digits of the mantissa.

## Default Argument Values

```
digitnum = 7
```

## Examples

```
t = [1:100]; x = 100*ones(100,2)+rand(100,2)+j*rand(100,2);
expvect('data.txt',t,real(x(:,2)),imag(x(:,2)),8)
```

## Diagnostics

`expvect` checks the vectors for complex, infinite or `NaN` elements, and sends an error message if any element of this kind is found.

The length of the vectors must be the same, otherwise an error message is generated.

If a file with the same name exists, the routine attempts to delete it. This can only be done if this file is in the active subdirectory/folder. If this is not the case, an error message is sent:

```
Cannot delete existing file ...
```

## See Also

`save`

## Purpose

Calculate transfer function model and its covariance matrix from pole-zero model, including the standard deviations and covariances of poles and zeros and gain values.

## Syntax

```
[pvect,Cp] = fdcovpzp(zv,stdz,pv,stdp)
[pvect,Cp] = fdcovpzp(zv,stdz,pv,stdp,rzp,g,stdg)
[pvect,Cp] = fdcovpzp(zv,stdz,pv,stdp,rzp,g,stdg,...
                    domain,fs,da,dzp)
```

## Description

It may be desirable to simulate or examine systems of which a pole-zero model is available. `fdcovpzp` converts such a model into the transfer function model required by this toolbox. It is the "inverse" of `stdpz`.

The output argument `pvect` is the parameter vector of the system (see exppar) and `Cp` is the corresponding covariance matrix.

`zv` and `pv` are the column vectors of the zeros and poles, respectively.

`stdz` and `stdp` contain the corresponding uncertainties. In each row there are three elements: the standard deviation of the real part, the standard deviation of the imaginary part, and the correlation coefficient between the real part and the imaginary part of the corresponding zero or pole.

For a full description of the interrelations of different poles and zeros, a full correlation coefficient matrix of the real and imaginary parts of all zeros and poles and two gain factors can be given in `rzp`. The order of the random variables in the corresponding random vector is: real part of the first zero, imaginary part of the first zero, real part of the second zero, etc., then the real part of the first pole, etc. The last two variables are the leading coefficients of the numerator and the denominator, respectively.

`g` contains two gain factors: the leading coefficients of the numerator and the denominator, respectively. If `g` is given as a scalar, `g(2)` is set to 1. The standard deviations of the gain values and their correlation coefficient are to be given in `stdg`.

By default, an *s*-domain model is assumed. For *z*-domain models, the domain (`domain = 'z'`) and the sampling frequency (`fs`) can be given.

`fdcovpzp` allows the use of *analytical* or *numerical differentiation* in the calculation of the sensitivity matrix. By numerical differentiation it can be

checked whether variations of the poles and zeros, made in the order of magnitude of the standard deviations (in the directions of the eigenvalues of the appropriate covariance matrix), cause the same changes in the transfer function parameters, as calculated from the analytical sensitivity calculations. Numerical differentiation can be requested by setting `da` to the value `'num'`. The amount of perturbations can be influenced by the variable `dzp`. By default, its value is 1, which means perturbations equal the eigenvalues of the covariance matrix of the zeros or poles, in the directions of the eigenvectors. These step lengths are multiplied by the value of `dzp`.

## Default Argument Values

```
g = [1;1]; stdg = [0;0;0]; domain = 's'; fs = 1; da = 'num';
dzp = 1.
```

## Examples

```
[zv,stdz,pv,stdp,rzp,g,stdg] = ...
        stdpz('inpchans.pbn','inpchans.cbn');
[pvectn,Cpn] = fdcovpzp(zv,stdz,pv,stdp,rzp,g,stdg);
subplot(121)
plotelpz('inpchans.pbn',[-6,2,-4,4]*1e5,'inpchans.cbn',...
        10,'nomsg')
subplot(122)
plotelpz(pvectn,[-6,2,-4,4]*1e5,Cpn,10,'nomsg')
```

## Diagnostics

The validity of the given correlation coefficients are checked, and an error message is sent if they are not consistent.

## Algorithm

In analytical calculations first the sensitivity matrix of zeros and poles on the coefficients of the corresponding polynomials is calculated using

$$S(i,j) = \frac{r(i)^j}{\frac{df(r(i))}{dr(i)}},$$

where $r(i)$ is the $i$th root of the polynomial $f(r(i))$. The sensitivity of zeros and poles is then calculated as the pseudo inverse of this matrix. The covariance matrix of the parameters is calculated multiplying the covariance matrix of the real and imaginary parts of the poles/zeros and the gains by the sensitivity matrix from both sides.

For numerical derivation, perturbations of the zeros and poles are introduced in the direction of each eigenvector of the corresponding covariance matrix, with step sizes equal to the corresponding eigenvalue, multiplied by `dzp`. The so determined sensitivities are then used in the covariance calculations.

## See Also

`stdpz`

## References

[1] P. Guillaume, J. Schoukens and R. Pintelon, "Sensitivity of roots to errors in the coefficients of polynomials obtained by frequency-domain estimation methods," *IEEE Trans. on Instrumentation and Measurement*, Vol. 38, No. 6, pp. 1050-1056, Dec. 1989.

# fdiddemo

## Purpose

Demonstrate the Frequency Domain System Identification Toolbox.

## Syntax

```
fdiddemo
```

## Description

Invoking `fdiddemo` starts a question and answer procedure, offering the choice among different demonstrations.

Many of the demonstrations work on measured data. These data are mostly results of measurements done by Department ELEC, Vrije Universiteit Brussel, Belgium. The data are public domain; they may befreely used by anybody.

## Diagnostics

Some of the demonstrations work on the data files used in the book of Schoukens and Pintelon. For these demonstrations the appropriate data files must be available in the search path of MATLAB.

Many of the demonstrations, using measured data, need more memory than available on small machines. These demonstrations will run on larger computers (with a minimum memory of 4 MBytes) only.

## References

[1] J. Schoukens and R. Pintelon, Identification of Linear Systems: A Practical Guideline for Accurate Modeling, London, Pergamon Press, 1991.

## Purpose

Syntactic analysis of filenames, splitting filenames to names and extensions, assigning default extension to filenames. This routine is used internally by other functions of the toolbox.

## Syntax

```
nfilename = fnamanal(filename)
[nfilename,fnamsh,ext] = fnamanal(filename,stext)
```

## Description

`fnamanal` analyzes the filename given in `filename`, puts the filename without extension into `fnamsh`, and the extension to `ext`. If `stext` is given (standard extension), and the `filename` has no extension, the standard one will be appended (preceded by a period) to the filename. The complete filename is given in `nfilename`.

## Examples

```
fnam = 'inpchan'; [fnam,fnshort,ext] = fnamanal(fnam,'fbn')
fnam =
   inpchan.fbn
fnshort =
   inpchan
ext =
   fbn
```

# gmean

## Purpose

Geometric mean of complex numbers.

## Syntax

```
gm = gmean(X)
```

## Description

`gmean` makes an attempt to eliminate the effect of phase wrapping when calculating the geometric mean of complex numbers. For matrices, `gmean(X)` is a row vector containing the complex geometric mean value of each column.

The complex geometric mean has important applications in the averaging of nonparametric estimates of complex transfer function values (see [1], [2], and "Solutions for Some Special Cases" on page 2-14).

## Example

```
x = -ones(100,2) + 0.1*randn(100,2) + j*0.01*randn(100,2);
mfalse1 = prod(x).^(1/100), mfalse2 = prod(x.^(1/100)),
mx = gmean(x)
```

## Algorithm

The absolute value of the geometric mean is calculated from the arithmetic mean of the logarithms of the absolute values; the phase is the phase of the arithmetic mean value of the complex numbers.

## References

[1] .J. Schoukens and R. Pintelon, "Measurement of Frequency Response Functions in Noisy Environments," *IEEE Trans. on Instrumentation and Measurement,* Vol. 39, No. 6, pp. 905-909, Dec. 1990.

[2] R. Pintelon, J. Schoukens and J. Renneboog, "The Geometric Mean of Power (Amplitude) Spectra Has a Much Smaller Bias than the Classical Arithmetic Averaging," *IEEE Trans. on Instrumentation and Measurement,* Vol. 37, No. 2, pp. 213-218, June 1988.

## Purpose

Generate quasi-logarithmic frequency set for use with FFT and for the generation of periodic excitation signals.

## Syntax

```
fqlog = lin2qlog(freqv,rf)
[fqlog,df,cdmax,freqind] = lin2qlog(freqv,rf)
fqlog = log2qlog(freqv,mhno)
[fqlog,df,cdmax,freqind] = log2qlog(freqv,mhno)
```

## Description

Both routines calculate a subset of the linear frequency grid, where the members of this subset are more or less logarithmically distributed.

`lin2qlog` starts from a linear frequency grid, given in `freqv`, and selects a quasi-logarithmic set, providing that the ratio of successive frequencies is about `rf` (or larger, if the frequency vector is not dense enough).

`log2qlog` starts from the given logarithmic frequency set `freqv`, and rounds these frequencies to values of the linear (DFT) frequency grid. The harmonic number `mhno` will be associated to the highest frequency. Multiple frequency points are eliminated. `log2qlog` also works with a non-logarithmic input vector, and produces a close equivalent on the DFT grid.

`fqlog` is the quasi-logarithmic frequency vector, and `df` is the size of the frequency step in the corresponding full linear grid. The period length of the corresponding multisine is usually `1/df`, but if the harmonic numbers have a common divider, it may be smaller. The largest common divider is given by `cdmax`, thus the period length is in general `1/(cdmax*df)`.

The harmonic numbers in `fqlog` can be calculated as

```
harmno = round(fqlog/df);
```

`freqind` is a column vector, containing indices of the selected frequency points of `freqv`.

## Examples

```
fqlog = lin2qlog([1:128],sqrt(2));
[fqlog,df] = ...
     log2qlog(logspace(log10(1),log10(256),17),256);
```

## Diagnostics

`freqv` must be real, non-negative and strictly increasing; `mhno` must be a positive integer.

## Algorithm

`lin2qlog`: starting from the first non-zero frequency point `fqlog(1)`, the next point `fqlog(2)` will be the one in `freqv`, closest to `fqlog(1)*rf`, and larger than `fqlog(1)`. The points of `freqv` between `fqlog(1)` and `fqlog(2)` are deleted. This is repeated until the end of the file: the last point will be taken only if the last frequency is larger than or equal to `fqlog(n-1)*rf`.

`log2qlog`: the points of `fqlog` will be chosen from the points of the grid `max(freqv)*[1:mhno]/mhno`. A point will be selected if it is closer to a point in `freqv` than any other point of the linear grid.

## See Also

`logspace`

## Purpose

Load data from ASCII files into a variable.

## Syntax

```
data = loadasc(filename)
data = loadasc(filename,ASCIItype)
```

## Description

`loadasc` loads the contents of the ASCII file `filename` into data. In some sense, this is an "intelligent" version of `load`, where

- •the generated variable need not have the same name as the file,
- •the ASCII file may contain comments, not only numbers,
- •the lines in the ASCII file need not contain the same number of numbers.

`ASCIItype` may have the value `'flat'`, in this case the `load` command is directly used, or `'text'`, which means that the file contains some textual information which has to be deleted during the loading, or the number of numbers in a line may vary from line to line. The filtering of the file is done using the command `sscanf`, available in MATLAB Version 4.0 or higher. Everything after a % character in a line is ignored, and the rest is searched for numbers. `loadasc` will read the file properly into a vector even if it is not provided the same number of elements in each line if used with the option `'text'`.

The resulting variable `data` is a column vector, unless `ASCIItype` is set to `'flat'` and each line of the flat ASCII file contains more than one number (but the number of elements in each row must be the same in a flat ASCII file).

## Default Argument Values

```
ASCIItype = 'text'
```

## Examples

```
fprintf('lasctest.txt',...
     '%.0f %%N\n%.4e,  %.4e %%amp1\n',1,5,6)
type lasctest.txt, v3 = loadasc('lasctest.txt')
```

## See Also

`load`

# loadvar, savevar

## Purpose

Load single variable from MAT-file, and save variable to existing MAT-file.

## Syntax

```
variable = loadvar(matfile,varname)
savevar(matfile,varname)
savevar(matfile,varname,varvalue)
```

## Description

`loadvar` loads a single variable from a MAT-file and assigns its value to a variable. The name of the file is given in the string `matfile`, the name of the variable in the string `varname`.

`savevar` saves a single variable to an already existing MAT-file. The value of the variable is given in `varvalue`. `varvalue` may be any valid expression in MATLAB. If `varvalue` is not given, the variable will be deleted from the MAT-file.

If no extension is given, the default extension (`.mat`) is appended to the filename.

The MAT-file should be in the active subdirectory/folder if `savevar` is used, or the path has to be explicitly given.

Note: using `savevar`, the variable `donotusethisnamemf` will also appear in the MAT-file, with the name of the file as string value.

## Examples

The value of a variable can be checked easily without destroying the complete workspace by loading the complete binary file

```
itlimit = loadvar('inpchans.ebn','itmax')
```

The value of a variable in a previously saved workspace can be changed, and/or a new variable can be added to the saved workspace:

```
x = 1; save savevtst.mat x
savevar('savevtst.mat','x',10)
savevar('savevtst.mat','y',20)
```

## Algorithm

Both `loadvar` and `savevar` are function M-files, which assign the input arguments to hopefully not used long variable names (`donotusethisnamev`, `donotusethisnamevv`), load the MATLAB binary file, and create the desired variable. `savevar` also saves the MAT-file again. `loadvar` and `savevar` will behave in an unpredictable way when either of these two long variable names are used in the MAT-file.

## Purpose

Generate maximum length binary sequence (pseudo-random binary sequence).

## Syntax

```
bitseries  = mlbs(log2N)
[bitseries,nextstnum] = mlbs(log2N,bitno,startnum)
```

## Description

`mlbs` generates a maximum length binary sequence (column vector `bitseries`), using a shift register of length `log2N`. The minimum value of the argument `log2N` is 2, the maximum value is 30. The length of the generated sequence is given by `bitno`, that is, a partial sequence can also be generated. The default value of `bitno` is `2^log2N-1`, that is, the full length of the PRBS with the given register length.

The generation is based on a binary shift register with modulo 2 feedback. The starting value of the register is `startnum`. The register contents can be obtained via `nextstnum`, which can be used for the continuation of the sequence generation.

## Default Argument Values

```
bitno = 2^log2N-1, startnum = 2^log2N-1
```

## Examples

```
bitseries = mlbs(10); %Length: N = 1023 = 2^10-1
```

## Diagnostics

The register length must be an integer number between the allowed minimum and maximum, otherwise an error message is sent:

```
log2N is not integer
```

or

```
log2N = ... is not allowed
```

`startnum` must be between 1 and $2^{log2N}$-1, otherwise the error message is

```
startnum out of range
```

## Algorithm

The feedback shift register is implemented in a MATLAB vector, according to the definition (see [1]).

## See Also

`dibs`

## References

[1] K. R. Godfrey, ed.: *Perturbation Signals for System Identification*. Englewood Cliffs, Prentice-Hall, 1993.

# modifyfv

## Purpose

`modifyfv` prefilters measured data by the inverse of a known part of the transfer function.

## Syntax

```
Fdatm = modifyfv(pdat,Fdat)
[Fdatm,vdatm] = modifyfv(pdat,Fdat,vdat,plotmode)
```

## Description

If a multiplicative term of the transfer function is known, it is disadvantageous to estimate this part from the measured data, since this increases the variance of the other term, too. A possible solution is to prefilter the input data by the inverse of this term.

The parameter vector is given in `pdat` (a string if the filename is given).

`Fdat` contains the Fourier data: it is an array: `[freqv,x,y]`, or a Fourier vector (see `expfou`), or the name of the Fourier file.

The filtered Fourier data are given in `Fdatm`, as an array if `Fdat` is an array, or as a Fourier vector if `Fdat` is a vector or a filename.

If the known partial transfer function has significant dynamics in the band of interest, the variance data also have to be changed, since the modification preserves the *relative variance* of the Fourier amplitudes, and in ELiS the absolute variances are given.

`vdat` is a variance array, `[varx,vary]` or `[varx,vary,covxy]`; or a variance vector (see `expvar`); or the name of the variance file. The filtered variance data are given in `vdatm`, as an array if `vdat` is an array, or as a variance vector if `vdat` is a vector or a filename.

`plotmode` defines the form of the plots of the results:

`'lin'`, `'linpb'` stands for linear frequency scale,

`'log'`, `'logpb'` stands for logarithmic frequency scale,

where `pb` requests plot in the passband only (the points where the transfer function is zero will be excluded from the plot).

If `plotmode` has any other value, no plot will be shown.

## Default Argument Values

```
plotmode = 'linpb'
```

## Examples

```
Fdatm = modifyfv('inpchanz.pbn','inpchan.fbn');
```

## Algorithm

The output Fourier amplitudes are divided by the transfer function values of the known term, and output variances by their absolute square values; the covariances are divided again by the transfer function values.

# msinclip

## Purpose

Minimize crest factor of multisine using the clipping algorithm.

## Syntax

```
cx = msinclip(freqv,ampv)
[cx,crx,crxmax,cry,crymax] = ...
   msinclip(freqv,ampv,tf,gmod,itno,ovs,N,clO)
```

## Description

`msinclip` iterates towards an optimum set of phases for which the crest factor of the multisine with the given amplitudes (or the larger of the crest factors of the two multisines at the input and the output of a linear system) is minimum.

`freqv` is the vector of frequencies where the nonzero amplitudes are given. The elements must be integer multiples of a `df` value, and the minimum number of sines is 2. `freqv` must be monotonously increasing.

`ampv` contains the absolute values of the desired nonzero complex amplitudes at the corresponding frequencies (halves of the real amplitudes). If any element of `ampv` is complex, the phases of `ampv` will be used as starting values, otherwise the Schroeder multisine is used.

`tf` contains the complex transfer function values at the given frequencies. If `tf` is given, input-output optimization will be performed.

If `gmod` is given with the value `'nograph'`, iteration results will not be plotted, if with the value `'lastgraph'`, the result of the last iteration only, if with the value `'graph10'` or `'graph100'`, the result of every $10^{th}$ or $100^{th}$ iteration, if with `'graph'`, the result of every iteration will be plotted.

When `gmod` contains the string `'ZOH'` (e. g., its value is `'graphZOH'`, or just `'ZOH'`), a zero-order hold multisine will be designed, instead of the band-limited one. This means that a stepwise function will be designed (prepared for a D/A converter), the number of samples will not be rounded up to the next power of two, and the maximum overshoots between samples will not be calculated.

`itno` contains the maximum number of iteration cycles. If `itno = 0`, the starting values will be returned (the Schroeder multisine or the one externally given).

`ovs` determines the minimum resolution of the peak factor calculation; the sampling frequency will chosen according to `fs Š ovs*2*fmax`. If the resolution is small, and a band-limited design is requested (that is, not a zero-order hold design), the grid is often not dense enough to "catch" the maximum values with certainty. The default value of `ovs = 16` usually provides errors less than 1%.

`N` offers a direct control of the length of the time series used for crest factor calculation, often necessary in ZOH design. The point number in the time series will be chosen equal to `N` in the ZOH design if the above condition for `fs` can be fulfilled when generating just one period, otherwise the condition will determine the point number. In the BL design the point number will be rounded up to the next power of 2 to use a base-2 FFT, and the condition has to be fulfilled again.

`N` must be an even number for `msinclip`.

If `N` is not given, it will be chosen to provide that each frequency be an integer multiple of `fs/N`.

The last input argument, `cl0`, lets the user set the initial clipping level of the algorithm between (0,1).

The output arguments are as follows.

The vector `cx` contains the complex amplitudes of the multisine (their absolute values are equal to the halves of the real amplitudes).

`crx` is the crest factor of the generated multisine, calculated with the given oversampling factor. However, the true value of the crest factor may be somewhat larger because the true peak value is usually slightly larger than the peak value *on the grid*.

`crxmax` is the upper limit of the crest factor of the multisine, calculated for the maximum possible peak value. This calculation is based on the determination of the maximum curvature of the time function:

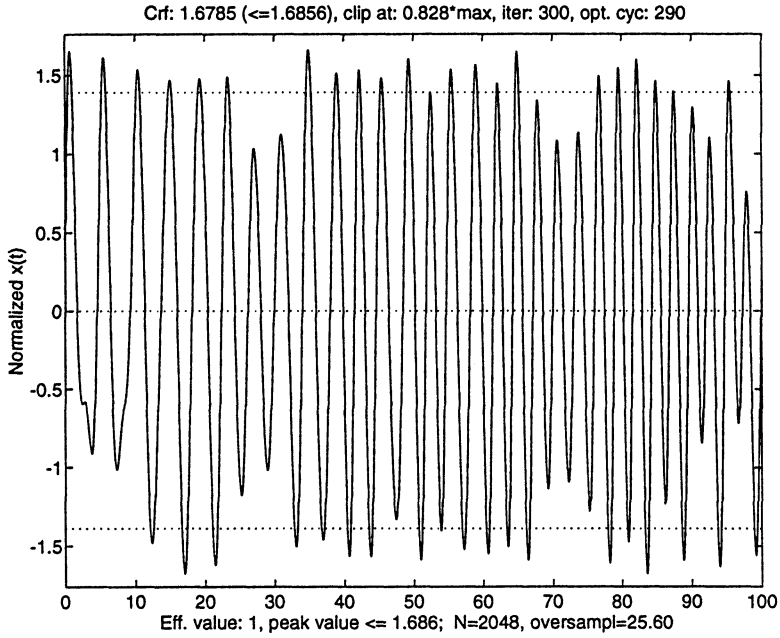
$$x(t) = \sum_{i=1}^{N} a_i \cos(2\pi f_i t)$$

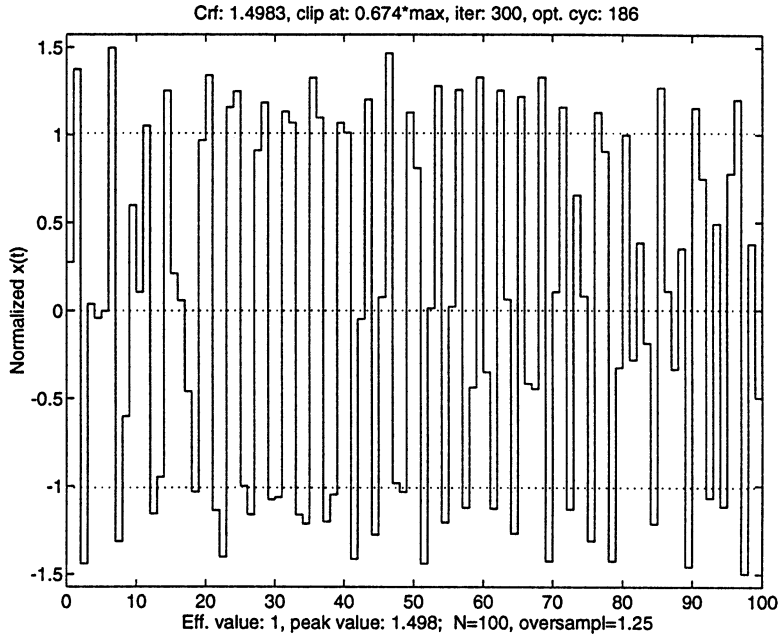
$$\frac{d^2 x(t)}{dt^2} = \sum_{i=1}^{N} a_i (2\pi f_i)^2 \cos(2\pi f_i t) \geq \sum_{i=1}^{N} -|a_i| (2\pi f_i)^2$$

$$\Delta x_{\max} \leq \sum_{i=1}^{N} (2\pi f_i)^2 \frac{\Delta t^2}{8}$$

`cry` is the crest factor of the multisine, calculated with the given oversampling factor, at the output of the linear system. `crymax` contains the worst-case crest factor of the output multisine.

Typical plots of the results of `msinclip` are shown in the figures. The first one illustrates a usual multisine design, the second one a zero-order hold one.






*crf* denotes the calculated crest factor of the multisine, followed by the worst case value. The actual clipping level is given next, and is shown by two dotted lines on the plot. In cycle 0 or if the iteration has converged, the clipping level is set equal to 1. The next number gives the number of performed iterations, and "*opt. cyc.*" denotes the cycle in which the smallest crest factor was found (the last plot of a run shows this multisine).

## Default Argument Values

```
ampv = ones(length(freqv),1), tf = [],
gmod = 'graph10' %plot every 10th result
```

The number of iterations: `itno = 250`.

`ovs = 16` for band-limited design, `ovs = 1` for zero-order hold design. In the case of band-limited design, the sampling frequency will be

```
2*ovs*2*fmax > fs Š ovs*2*fmax
```

or higher, when the given value of `N` prescribes it.

The clipping level will be chosen depending on the crest factor: for small crest factors (around 1.5) as 0.9, and smaller if the crest factor is larger.

## Examples

Multisine design:

```
[cx,crestx] = msinclip([1:15]',ones(15,1));
```

Multisine design, iteration started from random phases:

```
cx = msinclip(4:15,ones(1,12).*exp(j*2*pi*rand(1,12)));
```

Calculation of a Schroeder multisine:

```
[cx,crx] = msinclip([1:15]',[],[],'',0);
```

Calculation of a zero-order hold multisine:

```
[cx,crx] = msinclip([0.2:0.01:0.4]',ones(21,1)/sqrt(42),...
   [],'ZOH',250,1,100);
```

## Diagnostics

`ovs` must be at least 1, and the frequency vector must increase strictly monotonously. If a dc value is also given (`freqv(1) = 0`), the corresponding amplitude must not be zero.

The frequency resolution (`df`) is also calculated from `freqv` (the smallest common divider). If the maximum harmonic number is found to be larger than 1023 (which may well mean that the frequency vector was given erroneously), a warning message is sent:

```
WARNING: maximum harmonic index found in 'msinclip' is ...
```

## Algorithm

The algorithm is based on swapping between time domain and frequency domain: the time domain waveform is clipped, then transformed to the frequency domain, and the amplitudes are restored to the desired values. The clipping level is slowly adjusted according to the evolution of the crest factor: when the result is improved, the clipping level is decreased, otherwise it is increased.

If input-output minimization is performed, the worse crest factor is minimized.

## See Also

`msinprep`, `dibs`

## References

[1] E. van der Ouderaa, J. Schoukens and J. Renneboog, “Peak Factor Minimization, Using Time—Frequency Domain Swapping Algorithm,” *IEEE Trans. on Instrumentation and Measurement*, 1988, Vol. 37, No. 1, pp. 144-147.

[2] K. R. Godfrey, ed.: *Perturbation Signals for System Identification*. Englewood Cliffs, Prentice-Hall, 1993.

## Purpose

Generation of time domain multisine from complex amplitudes, and preparation for downloading into an arbitrary waveform generator.

## Syntax

```
xtim = msinprep(freqv,cx)
[xtim,df] = msinprep(freqv,cx,N,fs,dev)
```

## Description

`msinprep` generates time series from a set of complex amplitudes (multisine). The complex amplitudes are usually produced by `msinclip`. It is assumed that the frequencies are harmoniously related and an integer number of periods is to be generated. The algorithm can introduce a predistortion for a zero-order hold.

`xtim` is the generated time series, `df` is the calculated common divider of the given frequencies. `freqv` is the frequency vector, where the complex amplitudes are given, `cx` is the vector of complex amplitudes, and `N` is the length of the time series. If `N` is not given or is empty, it will be defined as `N = fs/df`, where `df` is the maximal common divider of the frequencies in `freqv`.

`fs` is the sampling frequency. If it is not given, it will be chosen as `fs = N*df`. If `N` is not given, either, `fs` will be chosen as `fs = df*2*(max(freqv)/df+1)`.

`dev` defines the device for which the series is prepared. Use `dev = 'screen'` for plotting, etc., with no modification of the Fourier series, or use `dev = 'DAC'` for D/A converter. In the latter case, the amplitudes will be multiplied by the inverse transfer function of the zero-order hold.

## Default Argument Values

`dev = 'DAC'`; `fs` and `N` as defined above.

## Examples

Generate a multisine and prepare it for downloading:

```
[cx,crx,crxmax] = msinclip([1:15]'/256,ones(15,1));
arbitgen = msinprep([1:15]'/256,cx,512,1,'DAC');
```

## Diagnostics

`freqv` must be monotonously increasing, and must not exceed the half of the given or above defined `fs`. `N` must also be large enough to have at least one period of each sine. If the length of the time series is smaller than the period length, the error message will be sent:

```
N (...) must be at least ... for one period
```

If the length of the time series is not equal to an integer multiple of the period length, a warning message will be sent:

```
WARNING: the N (...) samples cover ... periods, this is not an
integer
```

## Algorithm

The amplitudes are divided by the transfer function of the zero-order hold. The time function is calculated by inverse FFT.

## See Also

`msinclip`, `optexcit`

## Purpose

Generate the optimum input power spectrum for transfer function measurements.

## Syntax

```
X = optexcit(pdat,freqv)
[X,CR,fsv,vXwdev,Fiw] = ...
   optexcit(pdat,freqv,vdat,fixpind,XO,Ncyc,Fiw,pd)
```

## Description

`optexcit` iterates towards the optimum power spectrum of the input signal in the sense that it minimizes the volume of the uncertainty ellipsoid of the estimated parameters of the linear system. The transfer function is given by `pdat` (the vector of all the parameters, see `imppar`; or a filename), the frequencies where the optimum spectrum is looked for are given in the vector `freqv`. The input and output variance vectors are given by `vdat`. If this is an 1-by-2 or 1-by-3 vector, its elements are taken as constant input and output variances, and may be the input-output covariance. If this is an *N*-by-2 or *N*-by-3 array, the variance vectors are formed from the first two columns, and the covariance vector from the third one; if this is a vector, the variance vectors and the covariance vector are obtained using `impvar`; if it is a string, the variance file is looked for.

`fixpind` defines the fixed parameters in the following way: if the parameters (numerator, denominator and the delay) are put together into a vector as: `[num,denom,delay]'`, the elements of `fixpind` are the indices of the fixed parameters in this vector. (Here `num` and `denom` are row vectors defined in the usual way: in descending order of powers of $s$ in the $s$-domain, or in ascending order of powers of $z^{-1}$ in the $z$-domain.) When `fixpind` is given as just `fixpind = 0`, the zero-order coefficient of the denominator and the delay will be fixed. `fixpind = 'n'` means that there are no fixed parameters.

`XO` contains the starting values of the amplitudes, and `Ncyc` gives the number of the iteration cycles. If `Ncyc = 0`, a "partial run" is performed, and the returned amplitude vector equals `XO`, but the Cramér-Rao bound `CR` is properly calculated. `Fiw` is a large array of partial information matrices, exported in a previous run for the same system for acceleration of the subsequent runs.

`pd` is the density of plots: plotting occurs if `rem(cycle,pd) = 0`, and in the last cycle. `pd = inf` will totally suppress plotting.

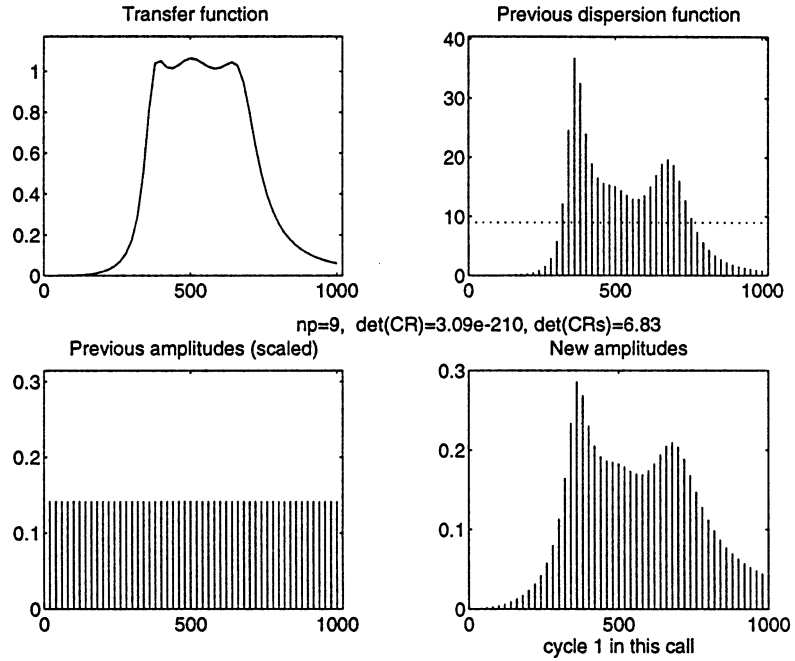
`X` is the generated amplitude vector. `CR` is the Cramér-Rao lower bound of the covariance matrix of the estimated parameters. For higher orders this matrix is usually very badly scaled in the *s*-domain, thus for the calculation of the determinant, scaling is advisable. `fsv` is the suggested scaling vector `(CRscaled = CR.*(fsv*fsv'))`.

`vXwdev` is an 1-by-2 vector, containing the minimum and maximum deviations of the dispersion function from its limiting value, that is, the number of the free parameters.

`Fiw` is the above mentioned array. If `Fiw` is too large to be stored on the computer, it will be returned as empty array, and recalculated in every iteration cycle.

Important: The delay may also be estimated. Consequently, if it is fixed, it is to be given in `fixpind`.

A typical plot of `optexcit` is shown in the figure.



The figure consists of four subplots. The first one shows the absolute value of the transfer function (linear scale), the one below it represents the previous amplitudes. The *previous dispersion function* shows the dispersion function calculated from the previous amplitudes. The result of the actual calculation cycle, the new amplitude set, is shown in the fourth subplot.

Below the dispersion function the number of free parameters is displayed (also represented by the dotted horizontal line in the plot). This is the theoretical final value of the dispersion function. The determinant of the covariance matrix, and the determinant of its scaled version are also shown.

## Default Argument Values

```
vdat = [1,1].
```

Index vector of the fixed parameters: if `fixpind` is not given, in the $s$-domain

```
fixpind = [nn+nd,nn+nd+1],
```

that is, the coefficient of $s^0$ in the denominator and the delay are fixed, while in the $z$-domain

```
fixpind = [nn+1,nn+nd+1]
```

the coefficient of $z^0$ in the denominator and the delay are fixed. `nn` is the number of parameters in the numerator, `nd` is the same in the denominator. Note that `fixpind = []` means that no fixed parameters are given (even the delay is variable); the default value can be given with `fixpind = 0`.

```
X0 = ones(F,1)/sqrt(F)
```

where `F` is the length of the frequency vector,

```
Ncyc = 1.
```

## Examples

(see `optexdem`)

1. Let us reproduce the results given in [1], Subsection 4.3.5, for a bandpass filter. After each power spectrum optimization step the crest factor is minimized, and the volume of `CR` is multiplied by an appropriate power of the crest factor.

Some remarks concerning the program: `optexcit` is performed consecutively, but some rearrangement of the results was necessary because the routine produces the *new* amplitudes and the *previous* covariance matrix. Thus, crest factor minimization is performed on `Xold`. Moreover, `msinclip` is used in two steps: experience shows that it is advantageous to optimize first the input crest factor, and then use this result for input-output optimization.

```
num = [3.2010e-17,5.5155e-12,8.973e-10,0,0];
denom = [1.0131e-21,2.5351e-18,3.6031e-14,...
       5.5550e-11,3.5869e-7,2.5017e-4,1];
F = 50; fv = 20*[1:F]'; pd = exppar('s',num,denom);
fixpar = [4;5;12;13]; np = 9; Fiw =  '';
Xold = ones(F,1)/sqrt(F); Nold = 0;
for N = [1,2,3,4,10,11,100,101]
  [X,CR,fsv,vXwdev,Fiw] = ...
       optexcit(pd,fv,[1,1],fixpar,Xold,N-Nold,Fiw);
  tf = polyval(num,sqrt(-1)*freqv*2*pi)./...
       polyval(denom,sqrt(-1)*freqv*2*pi);
  [cx,crx,crxmax] = msinclip(fv,Xold, [],'lastgraph');
  [cx,crx,crxmax,cry,crymax] = ...
       msinclip(fv,cx,tf,'lastgraph');
  X = Xold; N = Nold;
  cyc = N-1, dCR = det(CR), crx, dCRs = dCR*crx^(2*np)
end %for N
```

**2** Let us check the theoretical results of the example given in [1] (Section 4.1, Example 2). A first order system is excited at just one frequency. The Cramér-Rao lower bound can be calculated in closed form (`CRtheor`), thus it can be compared to the covariance matrix given by the routine `optexcit`.

```
b0 = 1; b1 = 1; num = [1]; denom = [b1,b0]; f = 2;
parvect = exppar('s',num,denom);
cf = b0^2+(2*pi*f*b1)^2;
CRtheor = [cf*(1+cf)/(2*pi*f)^{2,0;0,cf*(1+cf)];
[X,CR] = optexcit(parvect,f,[1,1],[1,4],1,1);
format long e, CR, CRtheor
```

## Diagnostics

The sizes of `X0` and `freqv` must be equal, otherwise an error message is sent:

```
X0 has not the same size as freqv
```

If `fixpind` is given with value 0, a warning message is generated:

```
The denominator coefficient of s^0 and the delay will be fixed in
optexcit
```

or

```
WARNING: the coefficient of z^0 and the delay will be fixed in
optexcit
```

The variance vectors given by `vdat` must have the same length as `fvect`, otherwise an error message is sent:

```
The length of varx is ... instead of ...
```

`Fiw` is generated only if the computer can store it in full size. If this is not the case, `Fiw` will not be generated (this results in a longer run time. If `Fiw` is requested by defining it as an output argument, a warning message is sent:

```
WARNING! Fiw would be too large, it cannot be generated
```

and the output argument will be returned as an empty variable.

## Algorithm

The algorithm is the one described in [1], [2] and [3]. First the partial information matrices are generated in each cycle for all the frequencies, then the dispersion function is determined, and the new power distribution is calculated.

## See Also

`msinclip`, `dibs`, `msinprep`

## References

[1] J. Schoukens and R. Pintelon, *Identification of Linear Systems: a Practical Guideline for Accurate Modeling,* London, Pergamon Press, 1991.

[2] F. Delbaen, “Optimizing the Determinant of a Positive Definite Matrix,” *Bulletin Société Mathématique de Belgique —Tijdschrift Belgisch Wiskundig Genootschap*, Vol. 42, No. 3, pp. 333-346.

[3] K. R. Godfrey, ed.: *Perturbation Signals for System Identification*. Englewood Cliffs, Prentice-Hall, 1993.

# pairs

## Purpose

Find the closest point pairs in two complex vectors.

## Syntax

```
indab = pairs(a,b)
[indab,cycle,digits] = pairs(a,b,p,maxcycle,digitsreq,D)
```

## Description

pairs looks for a permutation of the elements of the complex vector b to minimize

$$\sum_{i=1}^{n_a} \left| a_i - b_{i_{\text{perm}}} \right|^p$$

or if the array D is given,

$$\sum_{i=1}^{n_a} D(i, i_{\text{perm}})$$

The appropriate indices of `b` are returned in `indab`. The vector `a` may not be longer than `b`. If each element of `a` is paired to an element of `b`, `indab` will contain positive indices only; if not (the algorithm did not converge in the allowed number of iterations), `indab` will contain at least one zero.

`cycle` gives the number of iterations of the Hungarian method. If it is returned with the value zero, the simple nearest neighbors search gave the optimum. The Hungarian method may converge very slowly in some rare cases. Therefore, the maximum number of allowed iterations may be given in `maxcycle`. If `maxcycle` is given with a finite value, the internal cost values of the Hungarian method (the complements of the powers of distances, with respect to the maximum value) will be rounded to a reasonable number of digits (convergence is assured for integer numbers). Such a rounding will be indicated by a value of digits, smaller than `floor(log10(1/eps))`, which is 15 on most platforms. The number of used digits can also be adjusted at the beginning of the iteration (`digitsreq`).

## Default Argument Values

```
p = 2, maxcycle = inf, digitsreq = floor(log10(1/eps))
```

## Examples

```
indab = pairs(roots([1,2,3,4]),roots([1.01,2,3,4]),2);
pind = pairs([-0.1-1.5*j,-0.1+1.5*j],roots([1,2,3,4]));
```

If `cycle == 0`, it may be assumed that the new positions of the roots of the perturbed polynomial `p1` have been found. If `cycle>0`, and `all(indab)>0`, the optimal indices are found, but there is a chance that the roots are not paired properly, since the nearest neighbors cannot all be paired to each other.

## Diagnostics

The vectors are checked for infinite or `NaN` elements. If `a` or `b` is empty or is an array, an error message will be sent. `p` is also checked for positivity.

If the closest pairs do not give the best permutation, a warning message is sent:

```
WARNING: nearest neighbors do not give optimum in PAIRS
```

The algorithm may converge very slowly. In this case the reciprocals of the distances will be rounded, and a warning message will be sent:

```
Number of digits is set to ... in PAIRS
```

The algorithm may not converge if the number of allowed iterations is small. If this happens, a warning message is sent:

```
WARNING: PAIRS did not converge
```

and `indab` will contain at least one element that is equal to zero.

## Algorithm

The so-called Hungarian method ([1], [2]) is used: it is based on looking for alternating paths in bipartite graphs. The convergence of the method is assured for rational distances only; the algorithm implemented here makes an attempt to find a solution without rounding; when it fails, the complements of the distances to the maximum distance are rounded. By this the smallest distances, which are the most interesting, will be distorted the least.

## See Also

`plotelpz`

## References

[1] H. W. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Res. Logist. Quart.*, Vol. 2, 1955, pp. 83-97.

[2] B. Andrásfai, *Graph Theory: Flows, Matrices,* Budapest, Akadémiai Kiadó; Bristol, UK, Adam Hilger, 1991.

# plotelpz

## Purpose

Plot pole/zero pattern (maybe with confidence ellipses) of estimated transfer function.

## Syntax

```
plotelpz(pdat)
[rnum,rdenom] =...
    plotelpz(pdat,axv,cdat,Pc,ntx,parr,zarr,domain,da,dp,plm)
```

## Description

`plotelpz` plots pole/zero pattern of the transfer function defined by the parameter vector `pdat` or by the named file if `pdat` is a string.

The scaling may be modified by `axv`: if this is a four-element vector, it will be passed through `axis`; if this is `'z'`, the statement `axis([-2,2,-2,2])` will be executed; and if this is `'p'`, the plot will show all poles and zeros, and the vertical and the horizontal scaling will be the same. If `axv` is empty or missing, the plot will be scaled to show every pole and zero.

`cdat` is the covariance matrix (array), or the covariance vector (see `expcov`), or the name of the file containing the covariance matrix of the estimated parameters. It is used for the plot of the uncertainty ellipses.

The standard deviations of poles and zeros are calculated using `stdpz`.

The routine plots the "one-σ" contours as uncertainty ellipses. This can be modified by `Pc` which determine the multiplier of σ for the contours. It is easy to see that, supposing two-dimensional nondegenerate normal distribution, the confidence limit for the event that the pole (zero) falls inside the contour is

$$p = 1 - e^{-sc^2/2}$$

with sc being the multiplier of σ. This expression gives the following values:

| sc | p | degenerate p |
|---|---|---|
| 1.0 | 0.39 | 0.68 |
| 1.5 | 0.68 | 0.866 |
| 2.0 | 0.86 | 0.954 |
| 2.5 | 0.96 | 0.988 |
| 3.0 | 0.989 | 0.9973 |
| 3.5 | 0.9978 | 0.99953 |
| 4.0 | 0.99966 | 0.999937 |

if `0 < Pc < 1`, `Pc` will be interpreted as the confidence level *p*. If `Pc ≥ 1`, its value will be assigned to *sc*.

In the degenerate cases (one-dimensional distribution, as for real poles and zeros, or for certain constraints) the ellipses reduce to straight lines, and are represented by narrow "strips" on the plots. In such cases the probabilities can be calculated from the normal distribution (see third column).

The plot of uncertainty ellipses does not provide information about coupling of poles and zeros. Such couplings can be explored using the routine `stdpz`.

For multiple zeros/poles the analytical sensitivity calculations give infinite standard deviations: in such cases dotted rectangulars are plotted instead of ellipses around the multiple zeros/poles.

By the help of the string argument `ntx`, the plot style can be modified. If it is given with the value `'notext'`, no text at all will be written onto the plot; if with the value `'nomsg'`, the warning messages will be suppressed only.

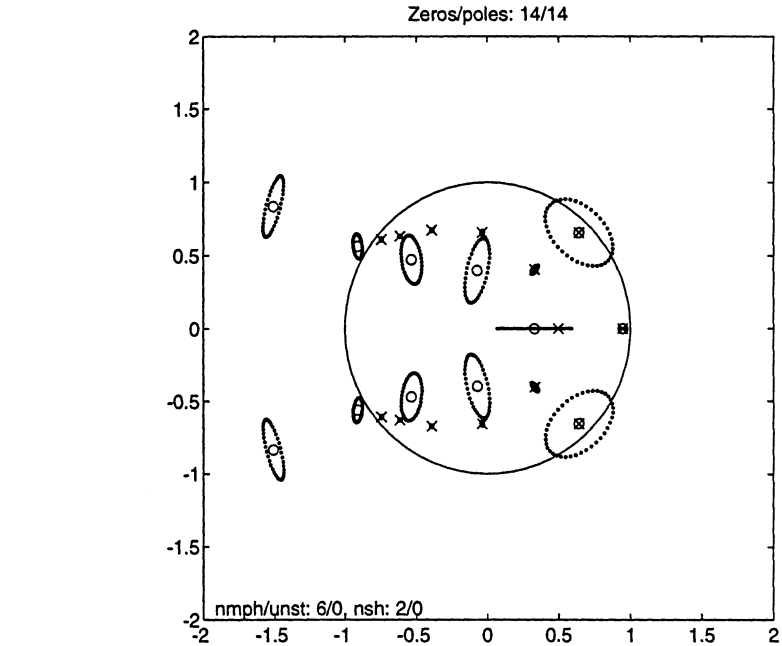
A set of poles and zeros each can also be given in the arrays `parr` and `zarr`, accompanied by the domain (`'s'` or `'z'`). These sets can also be plotted in addition to or instead of the zeros and poles given in `pdat`.

`plotelpz` can pass some arguments to `stdpz: da`, `dp` and `plm` can influence the calculation mode (see `stdpz`).

The output arguments `rnum` and `rdenom` are the row vectors of the poles and zeros, calculated from `pdat`. They are exported from `plotelpz` to avoid the necessity of repeated use of `roots`, if the order is large.

`plotelpz` can be used as a building block of complex M-files, especially with the `'notext'` option. It does not even change the state of the graphics window, previously set by the `subplot` or `axes` statement, if `ntxt` is given as `'notext'` or `'nomsg'`.

A typical plot of `plotelpz` is shown in the figure.




The numbers of poles and zeros, the numbers of non-minimal phase zeros and unstable poles are given on the plot, along with the number of zeros/poles not shown because of the axis scaling applied. If any of the uncertainty ellipses is so large that less than 10 points of the dotted line can be shown on the plot, a warning message appears at the lower left corner:

```
WARNING: fully or partly not visible ellipse
```

## Default Argument Values

```
axv = [], Pc = 0.39 (sc = 1), ntx = '', da = 'anal', dp = 1,
plm = ''.
```

## Examples

```
plotelpz('inpchanz.pbn','z');
[rnum,rdenom] = plotelpz('inpchans.pbn',[],'inpchans.cbn');
```

## Algorithm

The calculation of the poles/zeros is done by `roots`. The determination of the uncertainty ellipses is rather involved. The covariances of the real and imaginary parts of the poles/zeros are calculated by linear transformation from the covariance matrix of the parameters, using the sensitivity matrix (see `stdpz`).

## See Also

`stdpz`, `roots`, `ploteltf`

## References

[1] P. Guillaume, J. Schoukens and R. Pintelon, “Sensitivity of Roots to Errors in the Coefficient of Polynomials Obtained by Frequency-Domain Estimation Methods,” *IEEE Trans. on Instrumentation and Measurement*, Vol. 38, No. 6, pp. 1050-1056, Dec. 1989.

# ploteltf

## Purpose

Plot transfer function (may be with confidence bounds), and/or ratio of output and input Fourier amplitudes.

## Syntax

```
ploteltf(pdat1)
ploteltf(Fdat)
[ha1,ha2,fsc] = ploteltf(pdat1,pdat2,Fdat,fscale,msc,...
                         cdat1,cc,cn,rec,expi,ntx)
```

## Description

`ploteltf` plots transfer functions. `pdat1` and `pdat2` may be parameter vectors (see `exppar`), or parameter filenames. The first transfer function is plotted by mark `'–'`, the second one by `'.'`, at 256 equally distributed frequency points (equal distribution is understood here in either linear or logarithmic scaling). Also the ratio of output and input Fourier amplitudes can be plotted with mark `'+'`: these can be given in the Fourier vector `Fdat`, or in a file given by this string.

`fscale` defines the scaling of the frequency axis. Possible values:

`'lin'`: linear, from 0 to either the half of the sampling frequency in any of the parameter sets, or the maximal frequency in Fourier frequency vector (whichever is larger). If no Fourier data were given, and *s*-domain parameter set(s) are investigated, the maximum frequency will be the double of the suggested scaling frequency, calculated by `exppar` for the given transfer function.

`'linF'`: linear, from 0 to maximal frequency in the Fourier data.

`'lin',f1,f2]`: linear from `f1` to `f2`.

`'log'`: logarithmic, between `1e-3*fs/2 to fs/2` (`fs` is the higher sampling frequency in the parameter sets, if there is any), or from `1e-3*maxf` to `maxf` (maximal frequency in the Fourier data). For *s*-domain parameter sets with no Fourier file, plot from `1e-2*fscale` to `10*fscale`.

`'logF'`: logarithmic, from minimal nonzero frequency to maximal frequency in the Fourier data.

`'log',f1,f2]`: logarithmic from `f1` to `f2`.

The scaling of the amplitude plot can be influenced by `msc`. Its possible values are as follows: `'full'` to scale to all points, `'passb'` to scale to passband only (defined by the minimum and maximum of frequencies, for which `y` is not zero

in Fdat). If a '−' is appended to this parameter, only the phase is plotted, if a '+', only the amplitude.

For confidence interval plots of pdat1, the covariance matrix of the estimated parameters of pdat1 can be given by cdat1 as an array, or as a covariance vector (see expcov), or as the name of the covariance file. The $\pm 2\sigma$ bounds are plotted, usually at 32 points of the estimated magnitude and phase values, or at each frequency in Fdat. The number of points can be changed by cn between 1 and 256. The bounds can be changed by a multiplying factor cc. Its default value is 2.

The value of cc and the approximate confidence limit (Pc) are displayed under the plot, assuming normal distribution.

It is also possible to add confidence bounds to the nonparametric transfer function estimate points, defined by Fdat. For this purpose, the variance data have to be given in the place of cdat1, as a variance array [vx,vy] or [vx,vy,cxy], or as a variance vector (see expvar), or as a variance filename. ploteltf will recognize the type of the covariance/variance data; in an ambiguous case (long vector form) a covariance matrix of pdat1 will be assumed as given.

One is interested sometimes in the reciprocal of the transfer function, or in the reciprocal of the measured transfer function in the Fourier data (equalization). You can plot the reciprocal of any of the functions. rec = 'abc', where the letters refer to pdat1, pdat2, Fdat, respectively: 's' means straight (no reciprocal building is necessary), 'r' means reciprocal before plotting.

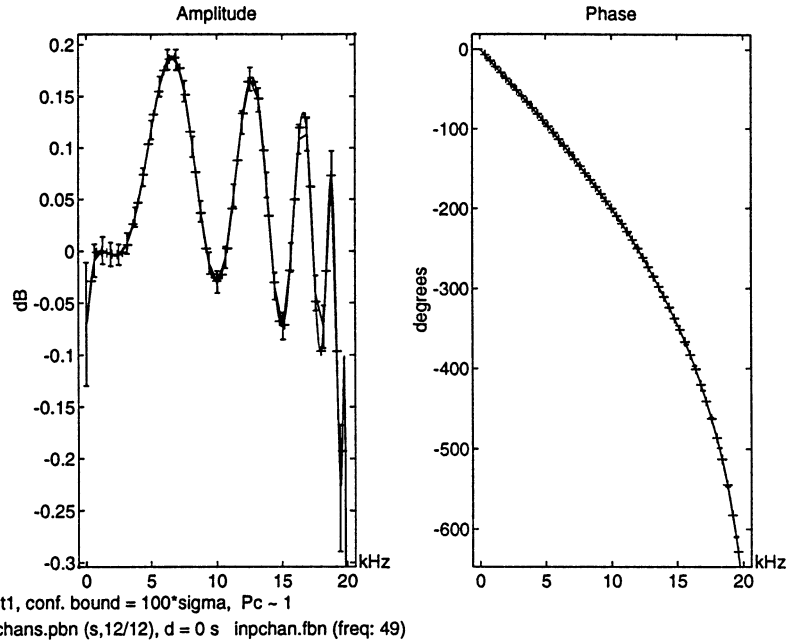
expi is the number of the experiment(s) in Fdat to be plotted. If expi is empty, all the experiments in Fdat will be plotted.

ploteltf puts some textual information to the plots. If ntx is given with the value 'notext', the filenames will not be shown in the plot.

For the purpose of further plots on the screen, the handles of the magnitude and phase plots (the axis vectors in MATLAB 3.5) are exported to ha1 and ha2. To avoid unwanted exponents below the frequency axis, the frequency vector is often scaled internally (to kHz, MHz or mHz). The scaling frequency is exported in fsc.

The magnitude plots are scaled in dB. To avoid annoying downscaling, the zeros of the transfer function are substituted by -100 dB values.

A typical plot is shown in the figure.




## Default Argument Values

```
Fdat = '', fscale = 'lin', msc = 'full', cdat1 = '', cc = 2,
cn = 32, rec = 'sss', expi = [], ntx = ''.
```

If only `Fdat` is given, this must be a Fourier filename, otherwise the program cannot distinguish it from a parameter vector.

## Examples

```
ploteltf('inpchans.pbn','','inpchan.fbn','linF','full+');
num = [1.1,1]; denom = [4,3,2,1];
[ha1,ha2,fsc] = ploteltf(exppar('z',num,denom,0));
```

## Algorithm

The confidence bounds are calculated using `stdtf` or `stdtfm`.

## See Also

`stdtf`, `stdtfm`

## Purpose

Calculate residuals of an ELiS fit.

## Syntax

```
[rx,ry] = rdueelis(pdat,cdat,Fdat,vdat,expi)
[rx,ry,ryx,vryx,xe,ye] = ...
         rdueelis(pdat,cdat,Fdat,vdat,expi,inp,outp)
```

## Description

`rdueelis` calculates the complex residuals of a fit of `elis` (see "Study of the Residuals" on page 2-36). `pdat` is the parameter vector of the fit (see `exppar`), or the name of the parameter file. `cdat` is the covariance array, or the covariance vector, or the name of the covariance file of `pdat`. If `cdat` is empty, the variance of the parametric estimate of the transfer function will not be considered.

`Fdat` is the Fourier vector, or the array `[freqvect,x,y]`, or the name of the Fourier file. `expi` contains the number(s) of the experiment(s) in `Fdat`, for which the residuals are to be calculated. `vdat` is a 1-by-2 vector if the variances are constant (or a 1-by-3 vector if the covariances are also given), or an *N*-by-2 array if the variances are given point by point (or an *N*-by-3 array if the covariances are also given), or a long variance vector if the variances are put together by `expvar`, or a string if the variance file is referred to.

`inp` and `outp` select the serial numbers of the input and of the output port in the Fourier file.

The residuals are given in `rx` (complex residuals of the input amplitude vector), `ry` (complex residuals of the output amplitude vector), `ryx` (complex residuals of `ym./xm` vs. the estimated transfer function).

`vryx` contains the variance vector of the real and of the imaginary parts of `ryx`. If the distributions of the complex `ryx` values are circularly symmetric, this equals the halves of the variances of `ryx`.

`vryx` is the difference of two quantities (see "Study of the Residuals" on page 2-36). Theoretically, this is never negative if the proper data are given (the same variance vector as used in `elis`; `pdat` and `cdat` that belong to this run of `elis`). However, because of the approximations, small negative values may occur: these usually indicate very small variances.

`xe` and `ye` are the estimated complex input and output amplitude vectors. These estimates are not consistent for one experiment. However, processing of several experiments at the same time does decrease their estimation error.

**Default Argument Values**

```
expi = [], inp = 1, outp = 1.
```

**Examples**

```
[rx,ry,ryx,vryx] = rdueelis('inpchans.pbn','inpchans.cbn',...
                            'inpchan.fbn',[9.61e-12,9.61e-10]);
```

**Diagnostics**

The validity of the variance values and the common length of the variance and frequency vectors is checked. If negative variance values are obtained, a warning message is sent.

**Algorithm**

The complex input and output amplitudes are estimated via weighted LS fitting, having the estimated transfer function parameters fixed.

**References**

[1] I. Kollár, "On Frequency Domain Identification of Linear Systems," *IEEE Trans. on Instrumentation and Measurement,* Vol. 42, No. 1, pp. 2–6, Feb. 1993.

## Purpose

Generate simulated Fourier amplitudes from parameters and variances.

## Syntax

```
[x,y] = simfou(pdat,freqv,x0,vdat)
[x,y] = simfou(pdat,freqv,x0,vdat,expno)
```

## Description

`simfou` generates simulated Fourier amplitudes (and perhaps also a Fourier file) for `elis`. The parameters of the transfer function are given in the vector `pdat` (see `exppar`), or in a file if `pdat` is a string. The frequencies are given in the vector `freqv`. The input amplitudes are given in the vector `x0`.

The variances are defined by `vdat`: if this is an 1-by-2 or 1-by-3 vector, its elements are taken as constant input and output variances (and perhaps the input-output covariance). If this is an *N*-by-2 or *N*-by-3 array, the variance vectors (and perhaps the covariance vector) are formed from the two (or three) columns. If this is a vector, the variance and covariance vectors are obtained using `impvar`. If it is a string, the variance file is looked for.

`expno` is the number of experiments to be generated.

## Default Argument Values

```
expno = 1
```

## Examples

The convergence properties of `elis` can be checked by running it several times on data simulated using a known transfer function:

```
freqv = [100:50:1000]';
num = 1; denom = [1e-6,1e-3,1];
pdat = exppar('s',num,denom);
vdat = [0.01,0.001];
[x,y] = simfou(pdat,freqv,[],vdat);
ploteltf(pdat,'',[freqv,x,y])
for k = 1:5
   [x,y] = simfou(pdat,freqv,[],vdat);
    Fdat = [freqv,x,y];
    [pvect,fit,Cp] = elis(Fdat,vdat,['s',0,2],[],'',100);
    pause
end
```

## Diagnostics

The routine checks the validity of the variance values and their compatibility with the frequency vector, and sends an error message when incompatibility is found.

When the system is unstable (the real part of a root is ≥0 in the *s*-domain or the absolute value of a root is ≥1 in the *z*-domain), a warning message is sent:

```
WARNING! The given system is unstable in simfou
```

but the simulated values will be calculated in the frequency domain.

## Algorithm

`simfou` calculates the input and output amplitude vectors, and adds zero-mean complex Gaussian noise with the given variances and properly set covariance to them.

## See Also

`simtime`

## Purpose

Generate simulated input and output time series from a transfer function.

## Syntax

```
[xt,yt] = simtime(pdat,u,numi,denomi,numo,denomo)
[xt,yt] = simtime(pdat,u,numi,denomi,numo,denomo,typ)
```

## Description

`simtime` generates simulated input and output time series for `elis` from a transfer function parameter set. The parameters of the transfer function are given in the vector `pdat` (see `exppar`, `imppar`), or in a file if `pdat` is a string. The excitation time series is given in `u`. The additive observation noise is generated from white Gaussian noise of variance 1 by the noise shaping filters, defined by `numi`, `denomi` and `numo`, `denomo`.

The input and output noises are independent. For the generation of correlated noise, the inverse Fourier transform of the outputs of `simfou` can be used, after having been called with a frequency vector `[0:N/2-1]*df`.

The resulting time series are returned in `xt` and `yt`.

`typ` chooses between two basic possibilities. If its value is `periodic`, `u` is considered as just one period of a periodic excitation, and one period of the steady-state system response is calculated, while the value `'transient'` makes `simtime` produce the transient response, starting from energyless state.

If `pdat` defines an *s*-domain model, only steady-state simulation is allowed. In such cases, `typ` must contain the sampling frequency in Hz.

## Default Argument Values

```
typ = 'periodic'
```

## Examples

```
f = 400*[1:49];
u = msinprep(f,msinclip(f,[],[],'',0),256,51200);
[xt,yt] = simtime('inpchanz.pbn',u,3e-5,1,3e-5,1);
[xtr,ytr] = simtime('inpchanz.pbn',u,...
     3e-5,1,3e-5,1,'transient');
```

## Diagnostics

The leading coefficient of the denominator should not be close to zero (this would approximate a predictor). If a too small leading coefficient is detected (its

absolute value is smaller than $10^{-10}$ times the largest coefficient), a warning message is sent:

```
WARNING: the leading coefficient of the denominator is very small
in simtime (maxdenom/denom(1) = ...)
```

In the transient case, the delay is realized by time shifts. The value must not be negative (this would mean a predictor). If the value of the delay is negative, an error message is sent:

```
The delay must not be negative (predictor cannot be simulated)
```

For the time shifting, the value of the delay is rounded to the nearest integer. When the value of the delay is changed, a warning message is sent:

```
WARNING: the delay has been rounded in simtime
```

The system defined by `pdat` must be stable. If this is not true, an error message is sent:

```
System is not stable
```

The noise shaping filters must be stable. If the absolute value of any of their poles is larger than $1\text{-}10^{-10}$, an error message is sent:

```
Input noise shaping filter is not sufficiently stable
```

or

```
Output noise shaping filter is not sufficiently stable
```

## Algorithm

The transient response is calculated via the function `filter` of MATLAB, the steady-state one by inverse Fourier transform of the frequency domain response.

## See Also

`simfou`

## Purpose

Calculate standard deviations and covariances of poles and zeros of an identified transfer function, using the covariance matrix of the parameters.

## Syntax

```
[zv,stdz,pv,stdp] = stdpz(pdat,cdat)
[zv,stdz,pv,stdp,rzp] = stdpz(pdat,cdat,zv0,pv0)
[zv,stdz,pv,stdp,rzp,g,stdg,dps] = ...
      stdpz(pdat,cdat,zv0,pv0,da,plm,dp,axv)
```

## Description

If the covariance matrix of a set of estimated parameters is given, the uncertainties of the zeros and poles can be calculated. `pdat` is the parameter vector, usually generated by `elis`, or the name of a parameter file (see `exppar`). `cdat` is the covariance matrix, corresponding to `pdat`, in the form of an array, as generated by `elis`, or in the form of a vector (see `expcov`), or the name of a covariance file.

`zv` and `pv` are the column vectors of the zeros and poles, respectively, and `stdz` and `stdp` contain the corresponding uncertainties. In each row there are three elements: the standard deviation of the real part, the standard deviation of the imaginary part, and the correlation coefficient between the real part and the imaginary part of the corresponding zero or pole.

For multiple zeros or poles the standard deviations cannot be determined by analytical differentiation; in such a case the corresponding standard deviations are given as `NaN`.

For the investigation of the interrelations of different poles and zeros, a correlation coefficient matrix of the real and imaginary parts of all zeros and poles and the two gain values can also be obtained in `rzp`. The order of the random variables in the corresponding random vector is: real part of the first zero, imaginary part of the first zero, real part of the second zero, etc., then the real part of the first pole, etc. The last two elements are the leading coefficients of the numerator and the denominator.

`g` contains the leading coefficients of the numerator and the denominator. The standard deviations of the gain values and their correlation coefficient are given in `stdg`.

Sometimes it is desirable to prescribe the order of the poles and zeros in `zv` and `pv`. This can be done by giving `zv0` and `pv0`. Any of these input arguments may be given as an empty variable, if the order need not be prescribed.

`stdpz` allows the use of *numerical differentiation* in the calculation of the zero/pole sensitivities to variations of parameters. It can check whether variations of the parameters, in the order of the standard deviations, cause the same changes in the positions of zeros and poles, as calculated from the analytical sensitivity calculations. The numerical differentiation can be requested by giving `da` with the value `'num'`.

The amount of perturbations can be influenced by `dp`. By default, its value is 1, which means perturbations equal the eigenvalues of the covariance matrix, in the directions of the eigenvectors. These steps can be multiplied by a given value `dp`. When the perturbations are too large, the zeros and poles cannot all be paired to their nearest neighbors in the perturbed sets, and a suggested value of `dp`, for which all zeros and poles can be paired to their nearest neighbors, will be given in `dps`; otherwise `dps` will be equal to the actual value of `dp`.

The perturbed sets of zeros and poles can be plotted on the screen. If `plm` is given with value `'mc'`, the perturbed sets and the pairing will be shown on the screen, one after the other. For the value `'mp'`, a statement pause will be executed after each pairing. The axis vector for these plots may be given in `axv`.

## Default Argument Values

```
zv0 = [], pv0 = [], da = 'anal', dp = 1, plm = ''.
```

## Examples

```
[pvect,fit,Cp] = elis('inpchans.ebn');
[zv,stdz,pv,stdp,rzp] = stdpz(pvect,Cp);
```

## Diagnostics

The standard deviations cannot be calculated by analytical derivation for multiple zeros or poles; in such cases the corresponding standard deviations will be given as `NaN`.

## Algorithm

The sensitivity matrix of zeros and poles on the coefficients of the corresponding polynomials is calculated using

$$S(i,j) = \frac{r(i)^j}{\frac{df(r(i))}{dr(i)}},$$

where $r(i)$ is the $i$th root of the polynomial $f(r(i))$. The covariance matrix of the real and imaginary parts of the poles/zeros is calculated multiplying the covariance matrix of the parameters by the sensitivity matrix from both sides.

For numerical derivation, perturbations of the parameter vector are introduced in the direction of each eigenvector of the covariance matrix, with step sizes equal to the corresponding eigenvalue, multiplied by `dp`. The original and perturbed sets are paired to each other by using `pairs`, with `p = 1`. This means that even for the multiple zeros or poles a standard deviation will be calculated, which will be in the order of the actual movements caused by the uncertainty of the parameters.

## See Also

`plotelpz`

## References

[1] P. Guillaume, J. Schoukens and R. Pintelon, “Sensitivity of roots to errors in the coefficients of polynomials obtained by frequency-domain estimation methods,” *IEEE Trans. on Instrumentation and Measurement*, Vol. 38, No. 6, pp. 1050-1056, Dec. 1989.

# stdtf

## Purpose

Calculate standard deviations of amplitude and phase values of an identified transfer function, using the covariance matrix of the parameters.

## Syntax

```
[tf,stda,stdph,stdri,rtf] = stdtf(freqv,pdat,cdat)
```

## Description

The approximate standard deviations of the amplitude and phase values of estimated transfer functions are calculated from the covariance matrix of the estimated parameters. `freqv` is the vector of frequencies at which the standard deviations are to be evaluated. `pdat` is the parameter vector, usually generated by `elis`, or the name of a parameter file (see `exppar`). `cdat` is the covariance matrix, corresponding to `pdat`, in the form of an array, as generated by `elis`, or in the form of a vector (see `expcov`), or the name of a covariance file.

`tf` is the column vector of the complex values of the transfer function, calculated from `pdat` at the given frequencies, `stda` is the column vector of the standard deviations of the amplitudes, and `stdph` is the column vector of the standard deviations of the phases, in radians.

If the distribution of the complex error of `tf` is circularly symmetric, `stda` also equals the standard deviations of the real and imaginary parts, but these can also be obtained in `stdri`, which has three columns: standard deviations of the real parts and of the imaginary parts of `tf`, furthermore the correlation coefficients of the real and imaginary parts of the corresponding `tf` value.

`rtf` is the correlation coefficient matrix of the vector of the real and imaginary parts of all the points: the real and imaginary parts of the first point are the first two elements, those of the second point are the third and the fourth, and so on.

The variance of the transfer function can be calculated as

```
vartf = stda^2 + (abs(tf).*stdph).^2
```

or

```
vartf = stdri(:,1).^2 + stdri(:,2).^2
```

## Examples

```
[pvect,fit,Cp] = elis('inpchans.ebn');
[tf,stda,stdph,stdri,rtf] = stdtf([400:400:19600],pvect,Cp);
```

## Algorithm

The sensitivity matrices of the amplitudes and phases on the estimated parameters, furthermore of the real and imaginary parts are calculated by direct derivation of the corresponding expressions. The appropriate covariance matrices of the amplitudes and phases are calculated multiplying the covariance matrix of the parameters by the sensitivity matrices from both sides.

## See Also

`fdcovpzp`, `ploteltf`

# stdtfm

## Purpose

Calculate empirical standard deviations of nonparametric transfer function estimates.

## Syntax

```
[tfm,stdAm,stdphm] = stdtfm(Fdat,vdat)
```

## Description

For model validation purposes you may wish to calculate the nonparametric transfer function estimate `tfm = ym./xm`, and check the uncertainties of these values.

`Fdat` contains the Fourier data. It is either an array: `[freqv,xm,ym]`, or a Fourier vector (see `expfou`), or the name of the Fourier file. The variances are defined by `vdat`. If this is an 1-by-2 or 1-by-3 vector, its elements are taken as constant input and output variances (and perhaps the input-output covariance). If this is an *N*-by-2 or *N*-by-3 array, the variance vectors (and perhaps the covariance vector) are formed from the two (or three) columns. If this is a vector, the variance and covariance vectors are obtained using `impvar`. If it is a string, the variance file is looked for.

`tfm` is the nonparametric estimate of the transfer function: `tfm = ym./xm`.

`stdAm` contains the approximate standard deviations of the absolute values of the transfer function. These are approximately equal to the standard deviations of the real and of the imaginary parts of `tfm`, if the distribution of the complex noises is circularly symmetric, as it is in the usual case.

`stdphm` returns the standard deviations of the phases in radians.

## Examples

```
[tfm,stdAm,stdphm] = stdtfm('emachine.fbn','emachine.vbn');
```

## Diagnostics

The routine checks the validity of the variance values and their compatibility with the frequency vector, and sends an error message when incompatibility is found.

## Algorithm

stdtfm calculates the standard deviations by evaluating the formula

$$\frac{\sqrt{\sigma_{xk}^2\left|\frac{Y_{mk}}{X_{mk}}\right|^2+\sigma_{yk}^2-2\,\mathrm{real}\left(c_{xyk}\left(\frac{\overline{Y_{mk}}}{X_{mk}}\right)\right)}}{\left|X_{mk}\right|}$$

$$=\left|\frac{Y_{mk}}{X_{mk}}\right|\sqrt{\frac{\sigma_{xk}^2}{\left|X_{mk}\right|^2}+\frac{\sigma_{yk}^2}{\left|Y_{mk}\right|^2}-2\,\mathrm{real}\left(\frac{c_{xyk}}{\overline{X_{mk}}Y_{mk}}\right)}$$

which gives the standard deviations of the real and of the imaginary parts of `tfm`. The output argument `stdphm` is obtained as `stdAm./abs(tfm)`.

If the standard deviations of the estimate `tfav = my./mx` are sought, using the results of `varanal`, these can be obtained by `vx/Na`, `vy/Na`, and `cxy/Na`, respectively.

## See Also

`ploteltf`

# tim2fou

## Purpose

Convert time domain data to frequency domain data for `elis`.

## Syntax

```
[x,y] = tim2fou(tdat,freqv)
[x,y,fv] = tim2fou(tdat,freqv,expi)
```

## Description

The time domain data vector is given in `tdat` (see `exptim`); or this is an array `[tvl,xt,yt]`, where the column vector `tvl` contains the time instants, repeated with the same values for each experiment; or this is the name of the time domain data file, with obligatory extension `.tbn`, `.tim`, or `.tnt`.

`tdat` can also be the name of an M-file. For example, if `tdat = 'gettim'`, the call `[tv,xt,yt] = gettim(i)` must return the results of experiment `i` for the values contained in `expi`.

`freqv` is the desired frequency vector; possibly each element of `freqv` should be a divider of the sampling frequency, but the algorithm works even if this is not true. If `freqv` is empty or missing, all possible frequencies in the FFT grid will be used from 0 to `fs/2`.

The generated input and output vectors (or arrays for multiple inputs or outputs) of complex amplitudes are `x` and `y`. Multiple experiments can also be processed. The actually used frequency vector is returned in `fv`.

## Default Argument Values

`freqv = [0:N/2-1]'/N*fs`, where `N` is the length of the time vectors, and `fs = 1/dt` is the sampling frequency.

## Examples

```
[x,y,fv] = tim2fou([[0:63]',cos([0:63]'/64*2*pi*3)]);
```

## Diagnostics

The time vector should consist of equidistant points, otherwise an error message is sent:

```
Samples are not equidistant
```

`fs = 1/dt` should be larger than the maximum of `freqv`, otherwise an error message is generated:

```
Maximum of freqv is larger than fs/2
```

Every element of `freqv` has to be a divider of the sampling frequency, otherwise a warning message is sent:

```
'WARNING: frequencies are not FFT points, leakage will appear
```

## Algorithm

The usual expression of the DFT is evaluated:

$$X_k = \sum_{i=1}^{N-1} x_i e^{-j2\pi\frac{ki}{N}}$$

If the frequency points in `freqv` are on the FFT grid, the DFT of the time domain vectors is calculated via FFT, and the corresponding complex amplitudes are selected. Otherwise the DFT is evaluated for the given frequency points.

## See Also

`exptim`

# varanal

## Purpose

Averaging and variance analysis of multiple experiments.

## Syntax

```
[vx,vy,cxy] = varanal(Fdat,expi)
[vx,vy,cxy,mx,my,Na,Np,cfl,dv,sd] = ...
      varanal(Fdat,expi,synch,T,inp,outp)
```

## Description

`varanal` averages frequency domain amplitudes as results of experiments, and calculates the empirical variances and input-output covariances of complex amplitudes. The results of the individual experiments are given either in just one Fourier vector `Fdat` (see `expfou`); or an array `[freqvlong,x,y]`, where `freqvlong` contains the frequency vector, repeated as many times as the number of experiments (the first frequency may not be repeatedly present in the short frequency vector); or in a file with name given in `Fdat`. A further possibility is when the string `Fdat` is the name of a user-defined function M-file, which can be called using `eval`. If the serial number of a given experiment is `i`, and `Fdat` is `'getfou'`, then `[freqv,x,y] = getfou(i)` must return the data of the $i^{th}$ experiment.

`expi` is the vector of the serial numbers of the experiments to be processed.

Averaging can be easily performed if the measurements were made in a synchronized way. However, when this is not the case, the routine can also be asked to try to "synchronize" the input vectors (assuming that they resulted from FFTs of the same length), minimizing the weighted phase differences of the complex amplitudes in the frequency domain by introducing a delay. Such synchronization can be requested by giving the argument `synch` with the value `'synch'`. For this synchronization attempt, the maximum value of the delay (the period length for periodic signals) is advisable to be given in `T`, although `varanal` makes an attempt to find a reasonable period length if `T` is not given. Since `T` specifies the range of rough search, a value of `T` slightly larger than the period length is preferred to a smaller one.

For synchronization, the elements of the frequency vector have to be given in `Fdat` with sufficient accuracy. The best solution is to generate them in MATLAB from the harmonic numbers. Inaccurate frequency values can be prohibitive for appropriate phase fitting.

If there are several input and/or output ports in the measurement, the user may choose from among them by defining the scalars (or vectors) `inp` and `outp`.

If any of them is not given, or it is given as an empty vector, each port present in the Fourier data will be used.

The variances are returned in the vectors (arrays) `vx` and `vy`, the input/output covariances

$$0.5\mathrm{E}\{\overline{N_x}N_x\}$$

in `cxy`. In the case of MIMO data, `cxy` will be an array, containing the covariances beside each other, as `[cx1y1,cx1y2,...cx2y1...]`.

The averaged input and output amplitudes are returned in `mx` and `my`.

The variances and covariances of the *averaged* complex amplitudes `mx` and `my` can be obtained as `vx/Na`, `vy/Na`, `cxy/Na`, respectively.

If the nonparametric estimate `Ym./Xm` is calculated, its approximate standard deviation can be determined from `vx`, `vy`, and `cxy`, using `stdtfm`. The standard deviations are to be scaled by `1/sqrt(Na)` when `my/mx` is used.

The number of averaged experiments is returned in `Na`, while the total number of processed experiments is `Np`. The two numbers may differ because when a synchronization attempt fails, the given experiment is not averaged to the others.

The multiplicative factors to obtain the lower and higher bounds of the 95% confidence intervals of the variances in the Gaussian case are returned in the 1×2 vector `cfl`. The confidence limits of the real and imaginary parts of the covariance values can be obtained by calculating `(cfl-1)*sqrt(varx(k)*vary(k))`.

If synchronization is requested, the vector of all the determined delay values is returned in `dv`. For the values of the delay the inequalities `-T/2<delay<=T/2` hold. If the synchronization failed, the corresponding element of `dv` is `NaN`.

The delays obtained by `varanal` can be used for restoring synchronization: if the value `dv(i-1)` is obtained for the complex amplitude set `Xi`, `exp(j*2*pi*freqv*dv(i-1)).*Xi` will give the amplitudes which correspond in phase to the reference set.

`sd` is the Cramér-Rao bound of the delay values. It is calculated from `vx` and `mx` (see [1]):

$$sd = \sqrt{\frac{2}{\sum_{k=1}^{F}\left(\frac{\sum_{ii=1}^{inpno}\left|mx_{k,ii}\right|^2}{vx_k}(2\pi f_k)^2\right)}}$$

Using the value of `sd`, a 95% significance level test is performed in order to determine if the estimated delays can all be zero (synchronized experiments). If there is no significant deviation from the hypothesis that the experiments are synchronized, a warning message is sent:

```
The estimated delays are small. With this SNR,
the statistical test shows no significant desynchronization.
```

If the signal-to-noise ratio is small, the synchronization attempt may fail. This is detected by observing for the best fit a phase deviation higher than $\pi/2$ at least at one frequency. When this happens, the experiment will not be averaged to the others.

## Default Argument Values

```
synch = '', inp = [], outp = []
```

## Examples

```
[vx,vy,cxy,mx,my] = varanal('bandpass.fbn',[1:6],'synch');
[vx,vy,cxy] = varanal('lowpass1.fbn',[1:5],'synch');
```

## Diagnostics

Synchronization can only be done if the Fourier amplitudes turn around by integer multiples of $2\pi$ with a time shift of `T`. If this condition is not met, a warning message is sent:

```
WARNING! Not all frequency vector elements are integer multiples
of 1/T
Maximum relative deviation is larger than 1e-6 in varanal.
```

varanal makes an attempt to find the period length of the signals from the frequency vector, and compares this to the given value T. if there is a deviation, one of the following warning messages is sent:

```
WARNING! T does not cover full period length found by varanal: T
= ..., Tp = ...
The proper delay may not be found.
```

or

```
WARNING! T is larger than period length found by varanal:
T = ..., Tp = ...
Search time may be unnecessarily long.
```

The synchronization procedure assumes that the signal-to-noise ratio is not very small, that is, in synchronized state the phase differences of the complex amplitudes in the different experiments, at the frequencies given in freqv, are smaller than $\pi/2$. Otherwise the experiment is skipped (and the delay value NaN appears in dv), and a warning message is sent:

```
Synchrnization is not successful for experiment ...
```

## Algorithm

Averaging of every quantity is done in a recursive way.

$$mx_k = \frac{k-1}{k} mx_{k-1} + \frac{1}{k} x_k, mx_1 = x_1$$

$$vx_k = s^{*2}_{xk} = \frac{k-2}{k-1} vx_{k-1} + \frac{k}{2(k-1)^2} \left| x_k - mx_k \right|^2, vx_1 = 0$$

$$cxy_k = \frac{k-2}{k-1} cxy_{k-1} + \frac{k}{2(k-1)^2} \overline{(x_k - mx_k)} (y_k - my_k), cxy_1 = 0$$

The confidence limit factors are calculated from the approximation of the $\chi^2$ distribution:

$$cfl = \frac{2(N_a - 1)}{\chi^2_{2N_a - 2,\ \alpha}} = \frac{1}{\left(1 - \frac{2}{9(2(N_a - 1))} \pm 1.96 \sqrt{\frac{2}{9(2(N_a - 1))}}\right)^3}$$

The degrees of freedom equal `2*(Na-1)`, because both the real and imaginary parts are used in the estimation.

The synchronization attempt starts with a scan through the possible values of the delay in steps of $T_{pmin}/6$, where $T_{pmin}$ is the smaller value of `T` and the period length of the largest frequency in the vector `freqv`. Using the best delay value, an additional, appropriately weighted LS fit of the phases is performed (see [1]). The resulting delay estimate is a maximum likelihood one, if the noises are Gaussian.

## See Also

`tim2fou`

## References

[1]I. Kollár, “Signal Enhancement Using Non-synchronized Measurements,” *IEEE Trans. on Instrumentation and Measurement*, Vol. 41, No. 1, pp. 156-159, Feb. 1992.

## Purpose

Intelligent input routine.

## Syntax

```
answer = yesinput(question,default)
answer = yesinput(question,default,possib)
```

## Description

`yesinput` is an "intelligent" version of the statement `input`. It displays a message as input usually does (string `question`), offers a default answer (defined by `default`) which can be accepted by pressing **Return** or **Enter**, and checks the validity of the answer using the optional argument `possib`. The type of the returned answer (string or number) is determined by the type of `default`. If the type of the desired `answer` is string, `possib` may be either a string array, where the rows contain the acceptable answers, or a string containing the acceptable answers, separated by `('|')` characters. If a number is desired, `possib` may be an 1×2 vector, containing the lower and higher limits for the input:

```
possib(1) ð answer ð possib(2).
possib(1) may be -inf,  possib(2) may be inf.
```

If a number is to be typed in, any valid MATLAB expression may be given, e. g., `2*pi/128`.

If the answer is not acceptable, the user is prompted again for a new answer.

For testing purposes, `yesinput` can be forced to accept the default answers and not to wait for the keyboard, by defining the global variable `yesinpacceptdef` with value `'yes'`.

## Examples

```
order = yesinput('Order of the filter',10,[0,12]);
color = yesinput('Color of plot','red','red|blue|green');
```

## See Also

`input`

# ywalk

## Purpose

Fit absolute values of frequency response points by a linear $z$-domain transfer function.

## Syntax

```
function [B,A] = ywalk(na,ff,aa,npt,lap)
```

## Description

`ywalk` is a slightly modified version of the routine `yulewalk` of the Signal Processing Toolbox; the built-in Hamming window is switched off. For more details, see `yulewalk`.

A

# Appendix

# Description of the Data Vector and File Formats

This section describes the following data vectors and file types:

- *time* vectors and files containing measured or simulated time domain data
- *Fourier* vectors and files containing measured, simulated or calculated frequency amplitudes or input-output point pairs
- *variance* vectors and files containing variances of the real parts (that is, also of the imaginary parts) of the measured frequency amplitudes, sometimes along with complex input/output covariances
- *parameter* vectors and files containing parameters of the transfer functions
- *covariance* vectors and files, containing the covariance matrix of the estimated parameters
- *report* files, containing textual documentation of the estimation procedure

The data vectors contain the same data in the same order as the flat ASCII files described below. Thus, data vectors will not be treated separately.

## Common Conventions for Every Data Type

### Vectors and ASCII Files

ASCII files may be *full* files (with comments), or *flat* files (containing numbers only).

The individual lines of full ASCII files may contain

- comments (lines beginning with %)
- a number, sometimes followed by explanation after the percent sign
- several numbers, sometimes followed by explanation after the percent sign

For the purpose of easier interfacing to MATLAB, the ASCII files may be filtered (ASCII flat file format): in this case they contain no text, the comment lines are deleted, and numbers that were originally in the same row are put into separate rows. For example, the rows

```
1.0000  2.0000  3.0000  −4.0000
1.1000  2.1000
```

may be transformed into the form

```
1.0000
                2.0000
                3.0000
               -4.0000
1.1000
                2.1000
```

This flat format corresponds to the data vectors used by the toolbox in the workspace.

The data given in a file are identified by their position among the numbers in the file; the comments and explanations are for readability only. However, we strongly recommend you to use the standard explanations in full ASCII files, because this makes possible to verify the contents of the files. The file structure is also slightly redundant, thus programs can also verify — to some extent — the correctness of files.

Full ASCII files can be directly read by the functions `impfou`, `imppar`, `impvar`, `impcov`, and `imptim`, or just loaded into a MATLAB vector by using the function `loadasc`, which is based on the C routine `sscanf`, accessible from MATLAB 4.0.

**Further Conventions for the ASCII Files.** A number in a line may be preceded by spaces or tabs, but by no other character; separators between numbers are spaces or tabs, sometimes commas. The numbers may be integers, or they may be given in scientific notation, with the usual restrictions adopted (they may not begin with a period — the valid beginning is a digit, maybe preceded by a sign, e. g., 1.0e-5, -1.2, etc). The number of digits is determined by the individual programs, taking into account the relevant features of the data (noise, uncertainty, etc.). Do not use much more digits than necessary to keep the information, because this may decrease the speed of loading data from the file.

The first lines of each file should be comment lines. The first comment line should contain the filename (for printouts), file type with a version number of the format, and the date (and perhaps also the time). Further comment lines may contain text necessary for identification of the data.

The filenames consist of a name (names not longer than 8 characters are preferred, to provide the easy transfer to MS-DOS), and an extension preceded by a period (.). The standard extensions are for “full” ASCII files: `.tim`, `.fou`,

`.var`, `.par`, `.cov`, and `.rep`, but the programs usually can handle files with other extensions. The names themselves must not contain periods or commas.

The extensions of the names of the filtered (flat ASCII) files end in `.nt` (which stands for no text): the standard extensions are `.tnt`, `.fnt`, `.vnt`, `.pnt`, and `.cnt`.

Files containing no text can be "filled in" with the appropriate texts; this can currently be done by using the toolbox function `elisfcnv`.

### Binary Files

Data read/write is much quicker through binary files. To maintain easy access from MATLAB, use the `.mat` binary format (see the online *MATLAB Reference Guide*, `load` and `save` statements). In binary files, each number or group of numbers is identified by a standard name. The string variables `comments`, `fdate`, and `ftype` may contain the necessary data for identification. The comments string need not contain % marks (but the % mark is allowed in the string). It may contain `cr` (Macintosh), `lf` (UNIX) or `cr/lf` (IBM-PC) characters marking multiple lines of comments, but this is not recommended if the transfer of binary files between different platforms is foreseen.

The standard extensions of the names of binary files are: `.tbn`, `.fbn`, `.vbn`, `.pbn`, and `.cbn`.

An ASCII file can be converted into binary format or vice versa by using the `elisfcnv` function of the toolbox.

## Conventions for Individual Data and File Types

Examples for the individual file types are given in the next section.

### Time Vector and File

The sample ASCII file shown in this Appendix is self-explanatory. However, some remarks can be made concerning special problems:

1. The start time and the stop time are redundant data beside the time vector. They are given for ease of manual check only, and are not used (though may be checked) in programs. Usually the time vector is not important, either, since sampling is usually equidistant; however, this convention was adopted to be able to handle non-uniformly sampled data.

2. The time instants given must be the same for each experiment in a file.

The binary file (`.tbn`) contains the following standard variables:

`comments` — eventual identification string (optional)

`ftype` — `'time Vx.x'`

`fdate` — date (and time) string (optional)

`expno` — number of experiments

`timevect` — column vector of time instants

`xt` — column vector of input amplitudes, or input amplitude array which has as many columns as the number of inputs; the results of different experiments are just put below each other.

`yt` — column vector of output amplitudes, or output amplitude array which has as many columns as the number of outputs; the results of different experiments are just put below each other.

## Fourier Vector and File

The sample ASCII file shown in this Appendix is self-explanatory. However, some remarks can be made concerning special problems:

1. The start frequency and the stop frequency are redundant data beside the frequency vector, they are given for ease of manual check only, and are not used (though may be checked) in programs.

2. The frequencies given must be the same for each experiment in a file.

3. If the data come from amplitude-only measurements, the imaginary parts are set to zero, which is a clear indication of the nature of the data.

The binary file (`.fbn`) contains the following standard variables:

`comments` — eventual identification string (optional)

`ftype` — `'Fourier Vx.x'`

`fdate` — date (and time) string (optional)

`expno` — number of experiments

`freqvect` — column vector of frequency points

`x` — column vector of complex input amplitudes, or input amplitude array which has as many columns as the number of inputs; the results of different experiments are just put below each other.

`y` — column vector of complex output amplitudes, or output amplitude array which has as many columns as the number of outputs; the results of different experiments are just put below each other.

## Variance Vector and File

These vectors and files contain the variances and perhaps covariances associated with complex amplitudes given in Fourier files. The variances are those of the real parts (that is, also of the imaginary parts) of the measured frequency amplitudes, in other words, these variances are the halves of the variances of the complex amplitudes. The covariances are defined as follows:

$$c_{xyz} = 0.5\mathrm{cov}\{N_{xk}, N_{yk}\} = 0.5\mathrm{E}\{\overline{N_{xk}} N_{yk}\}$$

Since variances and covariances have a meaning with the corresponding Fourier file only, you should give the same name to both files. The variance files contain 2 or 4 or `(2+2*inputno*outputno)` columns, the first two being the variances of the real (and of the imaginary) part of the corresponding complex amplitude vector, respectively, and the last columns being the real and imaginary parts of the complex covariances. For MIMO systems, the order of covariances is: `ci1o1, ci1o2,...,ci2o1,...`

The binary file (`.vbn`) contains the following standard variables:

`comments` — eventual identification string (optional)

`ftype` — `'variance Vx.x'`

`fdate` — date (and time) string (optional)

`varx` — column vector (or array for multiple inputs) of the input variances

`vary` — column vector (or array for multiple outputs) of the output variances

`covxy` — column vector (or array) of complex covariances: for MIMO systems, it may contain the covariances between just input1 and output1, or all the covariances beside each other, as `[c11,c12,...,c21...]`

## Parameter Vector and File

The protocol of runs of `elis` may be rather complicated, since a lot of data relevant to the run may be required. Thus, the protocol is provided in the report file. However, there is a standard basic set of data that should be provided in each case. In the following, this general block is defined.

The standard set of data is as follows:

- Domain
- Sampling frequency (in *z*-domain) or suggested scaling frequency (in *s*-domain)
- Order of the numerator
- Order of the denominator
- Coefficients of the numerator
- Coefficients of the denominator
- Delay

Notice that in the sample files along with the textual definition of the domain, a number is also given (0 for the *s*-domain and 1 for the *z*-domain). This provides that the domain information is contained also by flat ASCII files.

The above data may be followed by additional information, usually not used by programs. However, more information about the run of `elis` is provided in the vector `fit`, and in the report file.

The binary file (`.pbn`) contains the following standard variables:

`comments` — eventual identification string (optional)

`ftype` — `'parameter Vx.x'`

`fdate` — date (and time) string (optional)

`domain` — `'s'` or `'z'`, domain of the fit

`fs` — sampling frequency (in $z$-domain); suggested scaling frequency (in $s$-domain)

`num` — coefficients of the numerator (in ascending order of powers of $z^{-1}$ or in descending order in powers of s)

`denom` — coefficients of the denominator (in ascending order of powers of $z^{-1}$ or in descending order in powers of $s$)

`delay` — time delay associated to the transfer function (usually zero or a negative number)

## Covariance Vector and File

This file gives the covariance matrix of the coefficients (that is, coefficients of the numerator, coefficients of the denominator, and the delay). For the fixed coefficients, the rows and columns contain just zeros. The data set given in the ASCII file is just the upper right triangle of the covariance matrix. The order of the coefficients is numerator, denominator, delay. Numerator and denominator coefficients are ordered as in the coefficient vector, that is, in ascending order of powers of $z^{-1}$, or in descending order of powers of $s$. Each row of the triangle is given in a separate line; if the lines were too long, the continuations of a line may be put into a separate line, preceded by five spaces.

The binary file (`.cbn`) contains the following standard variables:

`comments` — eventual identification string (optional)

`ftype` — `'covariance Vx.x'`

`fdate` — date (and time) string (optional)

`coeffcovar` — covariance matrix of the parameters (zeros for the fixed parameters)

## Report File

This ASCII file gives the information relevant to a run of `elis`, the format is for direct use as text. The contents are:

- Date, time, comments
- Domain
- Sampling frequency ($z$-domain) or suggested scaling frequency ($s$–domain)
- Order of the numerator
- Order of the denominator
- Coefficients of the numerator
- Coefficients of the denominator
- Delay
- Fixed numerator coefficients
- Fixed denominator coefficients
- Iteration algorithm, number of iterations
- Way of setting the start values
- Condition numbers
- Total run time
- Value of the cost function, and last variation
- Degree of freedom
- Theoretical cost function value and the associated confidence limit
- Maximum of absolute deviations
- Poles and zeros
- etc.

A sample report file is given in the *Reference* chapter, in the description of `elis`.

# Examples for ASCII Files

## Sample Time File

```
%filename: data.tim, ftype: time V1.0, 06-Nov-1993, 11:51
%XH2 Synchronous machine identification
2 %Number of inputs
1 %Number of outputs
2 %Number of experiments
5 %Number of samples
0 %Start time (s)
1000 %Stop time (s)
%
%                        Measurement results
%                  Time        input                output
1 %Experiment no.
                   0.00        1.00       2.00      0.0125
                   250.00      1.00       3.00      0.2345
                   500.00      1.00       4.00      2.456
                   750.00      1.00       5.00      15.63
                   1000.0      1.00       6.00      5.678
%
2 %Experiment no.
                   0.00        1.00       2.00      0.0125
                   250.00      1.00       3.00      0.2545
                   500.00      1.00       4.00      2.876
                   750.00      1.00       5.00      16.63
                   1000.0      1.00       6.00      4.678
%
%End of time file
```

## Sample Fourier File

```
%filename: data.fou, ftype: Fourier V1.0, 06–Nov–1993, 11:51
%XH2 Synchronous machine identification
1 %Number of inputs
1 %Number of outputs
2 %Number of experiments
5 %Number of frequencies
0 %Start frequency (Hz)
1000 %Stop frequency (Hz)
%
%                      Measurement results
%Frequency         input                output
1 %Experiment no.
                   0.00      1.00       0.00      0.0125    –1.03e–4
                   250.00    1.00       0.00      0.2345    0.05412
                   500.00    1.00       0.00      2.456     –0.8455
                   750.00    1.00       0.00      15.63     –1.684
                   1000.0    1.00       0.00      5.678     0.3566
%
2 %Experiment no.
                   0.00      1.00       0.00      0.0123    –1.08e–3
                   250.00    1.00       0.00      0.2873    –0.03712
                   500.00    1.00       0.00      2.736     –0.8733
                   750.00    1.00       0.00      13.68     1.687
                   1000.0    1.00       0.00      3.678     0.8366
%
%End of Fourier file
```

## Sample Variance File

```
%filename: data.var, ftype: variance V1.0, 06–Nov–1993, 11:51
%XH2 Synchronous machine identification
1 %Number of inputs
1 %Number of outputs
1 %Number of covariances (0, 1 or  inputno×outputno)
%Variances
%                  Input     Output    Covariance: real, imag
                   1.0e–5    5.2e–4    3.9e–5     –2.6e–4
                   1.0e–5    5.2e–4    3.9e–5     –2.6e–4
                   1.0e–5    5.2e–4    3.9e–5     –2.6e–4
                   1.0e–5    5.2e–4    3.9e–5     –2.6e–4
%End of variance file
```

## Sample Parameter Files

```
%filename: data.par, ftype: parameter V1.0, 06-Nov-1993, 11:51
%XH2 Synchronous machine identification
%
0 %s-domain
1e3 %Suggested frequency scaling factor
0 %Order of the numerator
2 %Order of the denominator
%Coefficients of the numerator
%Powers of s       coefficients
                   0          1.324678532345477e-2
%
%Coefficients of the denominator
%Powers of s       coefficients
                   0          1.000000000000000
                   1          1.596753424395043e+3
                   2          1.896563012344548e+6
0.0000 %Delay
%
%End of parameter file
```

```
%filename: data.par, ftype: parameter V1.0, 06-Nov-1993, 11:51
%XH2 Synchronous machine identification
%
1 %z-domain
1e3 %Sampling frequency
0 %Order of the numerator
2 %Order of the denominator
%Coefficients of the numerator
%Powers of z^-1    coefficients
                   0          1.324678532345477e-2
%
%Coefficients of the denominator
%Powers of z^-1     coefficients
                   0          1.000000000000000
                   1          1.596753424395043e+3
                   2          1.896563012344548e+6
%
0.0000 %Delay, normalized to 1/fs
%
%End of parameter file
```

## Sample Covariance File

```
%filename: data.cov, ftype: covariance V1.0, 06–Nov–1993, 11:51
%Second–order mechanical system
%
                    1.18e–2     1.62e–3     1.93e–3
                    1.24e–2     1.456e–3
                    2.19e–2
%End of covariance file
```