Chapter 2

System Identification from I/O Data

2.1 Introduction

The purpose of this note is to provide some background and working knowledge on the subject of system identification from input-output (I/O) data. This basic and practically interesting problem can be defined as follows:

Given I/O data $(u(t), y(t))$, generated by a system G, $y = G[u]$, find a system $\hat{G}$ that approximates G and provide an estimate of the size of the approximation error.

The issues associated with I/O system identification can be classified in four main categories:

- System Approximation
- System Parametrization
- Parameter Estimation
- Implementation

System approximation deals with the sense in which the assumed class of system models (e.g. LTI systems of finite dimension) approximates the actual system. This is an important issue in order to clarify and make precise the theoretical framework on which we build our study. At this point we should emphasize that, without additional assumptions, any number of I/O data can never validate an approximation of a system but only invalidate one. (Still, we can produce estimates of an upper bound on the approximation/identification error.) To overcome this difficulty, one can make assumptions based on physical principles and some knowledge about the system, or establish results in a probabilistic framework; here, for simplicity, a complete development is not pursued.

System parametrization deals with the manner adjustable parameters enter the description of the class of system models under consideration. For example, such parameters can be the poles and zeros of a transfer function, or the coefficients of the numerator and denominator polynomials, etc. Among other properties, the parametrization approach can have a dramatic effect on the amount of required computations, in order to obtain the identified system. (Parametric models are required for the so-called "parametric system identification;" however, non-parametric methods can be employed as well.)

The issue of parameter estimation arises after having decided on the structure of a parametric model and the number of adjustable parameters. At this point, we are interested in constructing a suitable algorithm that, given u, y, defines the computations we must perform in order to obtain the parameter estimates. These estimates are, in turn, substituted in our parametric model to define the identified system ($\hat{G}$).

Finally, under "implementation" we consider various practical issues such as selection of test inputs, fine-tuning of the estimation algorithm (problem-dependent), computational trade-offs etc.

2.2 System Parametrization

A useful way to parametrize LTI systems is the so-called "equation error" approach. We illustrate this approach with a simple example. Let $G(s;\theta)$ denote the family of second-order proper transfer functions parametrized by the vector θ as follows:

$$G(s;\theta) = \frac{N(s;\theta)}{D(s;\theta)} = \frac{\theta_1 s^2 + \theta_2 s + \theta_3}{s^2 + \theta_4 s + \theta_5}$$

Clearly any second-order proper transfer function belongs to this family. Now suppose that u, y denote the I/O pair of an LTI system whose transfer function is $G(s;\theta_*)$ for some fixed but unknown θ_*. Then,

$$D(s;\theta_*)\hat{y}(s) = N(s;\theta_*)\hat{u}(s)$$

Let $D_F(s)$ be a monic Hurwitz polynomial (all roots in the open left-half plane) of the same degree as $D(s;\theta_*)$, i.e., $D_F(s) = s^2 + f_1 s + f_2$. Dividing both sides with $D_F(s)$ we get

$$\frac{D(s;\theta_*)}{D_F(s)}\hat{y}(s) = \frac{N(s;\theta_*)}{D_F(s)}\hat{u}(s)$$

which can be expressed as

$$\hat{y}(s) = \frac{N(s;\theta_*)}{D_F(s)}\hat{u}(s) + \frac{D_F(s) - D(s;\theta_*)}{D_F(s)}\hat{y}(s)$$

Isolating the unknown θ_* in the right hand side, we obtain

$$\hat{y}(s) = \left[\frac{s^2\hat{u}(s)}{D_F(s)}, \frac{s\hat{u}(s)}{D_F(s)}, \frac{\hat{u}(s)}{D_F(s)}, \frac{s\hat{y}(s)}{D_F(s)}, \frac{\hat{y}(s)}{D_F(s)}\right] \begin{bmatrix} \theta_{1*} \\ \theta_{2*} \\ \theta_{3*} \\ f_1 - \theta_{4*} \\ f_2 - \theta_{5*} \end{bmatrix} \tag{2.1}$$

The important observation here is that the unknown parameters satisfy a so-called "linear model" that is of the form

$$\phi^\top x = b$$

Here b corresponds to the output $\hat{y}(s)$ and ϕ is a vector of signals that can be obtained after filtering the I/O pair with suitable filters; x itself is not the vector of unknown parameters but θ_* can be obtained from x after some straightforward computations. Thus, all we need to do in our parameter estimation section is to devise an algorithm that solves the above equation for θ. Before we proceed, some comments are necessary to help in the interpretation and generalization of our linear model.

- The form of equation (2.1) remains essentially the same regardless of the order of the system model.
- Equation (2.1) remains valid if all signals are converted to time-domain. In this case, the multiplication by a transfer function translates into a convolution with an impulse response. This is a useful observation since it indicates that the identification computations can be performed directly in the time-domain where, afterall, the measurements are obtained.
- For practical problems where the order of the actual system is unknown or very high, $G(s;\theta_*)$ should be viewed as a desirable low order approximation of the actual system. In this case, we should add a correction term in (2.1) to get

$$\hat{y}(s) = \frac{N(s;\theta_*)}{D_F(s)}\hat{u}(s) + \frac{D_F(s) - D(s;\theta_*)}{D_F(s)}\hat{y}(s) + \hat{e}(s) \tag{2.2}$$

Here, e (or its Laplace transform $\hat{e}$) represents the error between the actual system and its approximation.[1] If the actual system admits a "good" LTI approximation, for sufficiently high-order $G(s;\theta_*)$ then the error term e should be small relative to u and y. Thus, a meaningful approach to estimate θ_* is to minimize the size of e in some sense, e.g., minimize its energy. Alternatively, in terms of our linear model, we would like to minimize the energy of the difference $\phi^\top x - b$.

- Choosing the order of the system too high is not recommended since it causes several problems of both numerical and theoretical nature. In particular, for high order approximants some of the parameters may attempt to "identify" the noise in the data, leading to meaningless results.

- The system identification problem can be formulated in a similar way even if a different approach is used to parametrize the family $G(s;\theta_*)$ or if the error to be minimized is different, e.g., the more natural error $\hat{y}(s) - G(s;\theta_*)\hat{u}(s)$. Unfortunately, most of these variants result in nonlinear optimization problems that are difficult to solve.

- A key feature of the equation error approach is in the way the actual system is approximated. This requires an additional feedback uncertainty term and cannot be captured by the usual multiplicative or additive uncertainty models but. Such an approximation may not be as good for long term prediction of the system behavior. Nevertheless, under some mild assumptions, it can be shown that if the identification error is small enough then a controller that stabilizes the identified system also stabilizes the actual one. In other words, this approach yields system approximations that are suitable for controller design.

- The polynomial $D_F(s)$ is a degree of freedom left to the designer. Although its selection is not important in the "ideal" case where the model mismatch is zero ($e = 0$), it has significant effects when $e \neq 0$. As a loose "rule of thumb," the bandwidth of transfer function $1/D_F(s)$ should contain the frequency range of interest. In addition, when the I/O data are sampled, the bandwidth of $1/D_F(s)$ should be sufficiently smaller (by about a decade) than the Nyquist frequency, in order to minimize the perturbation introduced by the sampling process.

2.3 Parameter Estimation

In the previous section we obtained a linear model relating the unknown system parameters with signals that are either measured directly, or they can be computed by filtering measured signals. That is, converting (2.2) in time domain, we need to address the problem:

Estimate the unknown constant vector x, given measurements $\phi(t), b(t)$ that satisfy

$$b(t) = \phi(t)^\top x + e(t)$$

where e is an unmeasured "noise" signal.

Here, for simplicity, we only consider the sampled-data version of this problem where the values of $\phi(t), b(t)$ are available at discrete time instants $t_k, k = 1, 2, \ldots, M$. This is motivated by the eventual implementation of our algorithm in a digital computer environment. Furthermore, in order to avoid a detailed analysis of discretization issues, we assume that the sampling frequency is sufficiently high so that the sampled signals provide an adequate description of their continuous-time counterparts.

Thus, our problem translates to estimating the unknown x from $b(t) = \phi(t)^\top x + e(t)$ given M measurements $\phi(t_k), b(t_k)$. Alternatively, to simplify the notation, we may concatenate the linear equations in a matrix form to obtain

$$\Phi x = B - E$$

where Φ is an $M \times m$ matrix (m being the number of elements in x) whose rows are the vectors $\phi(t_k)^\top$ and B, E are $M \times 1$ vectors with elements $b(t_k)$ and $e(t_k)$ respectively. Of course, since E is unknown, this equation cannot be solved in a strict sense. However, seeking the "best" model (defined by x) that

[1] The effect of initial conditions can also be included in e. Also, notice that e depends on the choice of parameters θ.

describes our data, it makes sense to select x so as to minimize the size of the vector E. This process can be interpreted as finding the model that minimizes the mismatch error between measured and predicted values of the signals. Here, we use a Eclidean distance to quantify the size of E, namely $\|E\|_2 = \sqrt{E^\top E}$.

After the last transformation, our parameter estimation problem has been converted into the standard least squares minimization problem

$$\min_{x \in \mathbf{R}^m} \|\Phi x - B\|$$

for which extensive literature is available dealing with theoretical and computational issues. Assuming that the matrix $\Phi^\top \Phi$ is invertible, the least-squares solution can be written in a simple form:

$$x_{LS} = (\Phi^\top \Phi)^{-1} W^\top B \tag{2.1}$$

$$\min_{x \in \mathbf{R}^m} \|\Phi x - B\| = \|\Phi x_{LS} - B\|$$

With the least-squares solution formula (2.1) in our disposal, we may now establish a system identification procedure for SISO systems.

- Select a test input sequence (u)
- Collect experimental data (u, y).
- Form the matrix Φ and vector B by processing the I/O data.
- Compute the LS solution
- Compute the transfer function of the identified system
- Compute the residual error and obtain an estimate of the uncertainty bound.

Of course, our algorithm has limitations and we must make sure that the experiment design and data processing are performed in an appropriate manner, consistent with the limitations of the algorithm. Furthermore, we should be careful in interpreting the results to correctly assess their validity and potential usefulness (or lack thereof). For this purpose, some implementation issues are discussed in the following section.

2.4 Implementation and Miscellaneous Issues

In this section we briefly discuss several important issues that are necessary for the "correct" implementation of the presented system identification algorithm and the appropriate interpretation of the results.

2.4.1 Input Selection

The test input used in the identification procedure should be such that it reveals all the important attributes of the system. In order to obtain sufficient information about the system, the input spectrum should contain enough frequencies that excite all system modes that are of interest. Even simple inputs may satisfy this qualitative criterion but may result in parameter estimates that emphasize unimportant charactersitics of the response. For example, a step input contains energy at all frequencies but puts excessive emphasis on the steady state response. For controller design purposes, we are interested in obtaining a good match around the eventual crossover frequency of the closed loop system. Notice that test inputs that produce sufficient excitation are also required in order to ensure the invertibility of the matrix $W^\top W$, entering our least squares formula.

RBS Inputs: An easy way to generate inputs that provide sufficient excitation is using the so called Random Binary Sequence (RBS). One variation of this is generated in MATLAB by the function *prbs_1* as a sequence of pulses with amplitude switching randomly between -1 and 1. This procedure results in an input that switches between -1 and 1 and the interval between the switches is random. Its energy spectrum is ideally "flat" but it can be weighted towards middle or lower frequencies by introducing a variable that specifies a minimum time between switches. PRBS is a popular test input because of its simplicity and the ability to prescribe the amplitude of excitation. The choice of the amplitude involves a trade-off between large values that provide better signal-to-noise ratio and small values that minimize the effects of system nonlinearities.

Further, it is often desired to design the test input so that the initial and final state of the system is "at rest."[2] This can be achieved by augmenting the test sequence with leading and trailing zeros. The number of the trailing zeros can be determined easily by some initial experiment since their role is simply to allow the response to come to a "near-rest" state before terminating the data collection. The selection of leading zeros, on the other hand, could be more involved. Here, we assume that the system has been kept at rest for a sufficiently long period of time so that any transient response contributions are negligible. Hence, this initial period can be used to estimate offsets that arise from linearization and noise properties. The length of the leading zeros should, therefore, be determined so that an effective averaging of the noise can be achieved. Of course, the same result could also be obtained from a separate experiment.

2.4.2 Data Weighting

For controller design purposes, it is often desirable to put additional emphasis on the quality of approximation in a particular frequency range. This can be achieved by using a bandpass filter to "prefilter" the input and output measurements. Notice that this prefiltering need not occur during the data collection process. The use of the *lsim* MATLAB function before solving the least squares problem can accomplish the same result. This also allows experimentation with different filters without having to repeat a lengthy and possibly expensive data collection process.

2.4.3 Identification in Closed-Loop

For many applications it may be impossible (or undesirable) to perform an identification experiment in open loop, i.e., supply the RBS (or any other) test input as an input to the system. Typical examples are systems that are open-loop unstable or have very slow response. In such cases, we may use a simple controller to close the loop and stabilize the system or produce faster response. Then, the desired test input can be supplied as a *reference signal* to the controller, or added to the control input. The data collection remains the same as before, i.e., we record the system output and the system input which is now the controller output. This approach alters the effective excitation, as observed by the system. It also introduces a bias the least squares problem, in the form of a frequency-dependent weight. For controller design purposes, the effects of this bias are not necessarily detrimental to the quality of the identification since it tends to emphasize the model accuracy in the middle-frequency range (around the crossover).

2.4.4 Quality of the Approximation

As with any other approximation/estimation process, system identification is not complete until we obtain both an identified (or nominal) system and the corresponding error bounds. One way to define the uncertainty bound was described in the previous chapter, in terms of the γ_2-gain of the uncertainty. Better yet, a less conservative, frequency-dependent bound can be given in terms of a weight (transfer function) whose product with the uncertainty has γ_2-gain less than one. Unfortunately, neither one is applicable in our case. The reason is that the identification approach described here (equation error) assumes a different uncertainty structure, shown in Fig. 2.1. Nevertheless, we may still process the information contained in the residual error to determine the power spectrum in the identification error. This can be translated into sensitivity and complementary sensitivity bounds and used for controller design in a similar manner as in the multiplicativw uncertainty case. The computational procedure resembles the derivation of an equivalent multiplicative uncertainty bound but its justification is different. Here, we simply present the formulae used

[2]See also the relation between FFT and the Fourier transform.

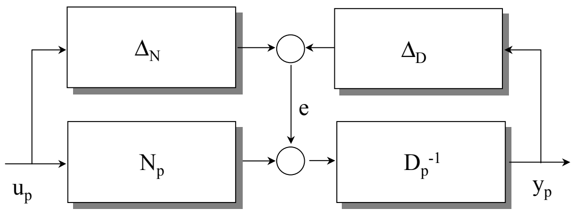

Figure 2.1: Structure of the identification uncertainty with an equation error approach. $N_p = N/D_F$, $D_p = D/D_F$.

for this computation.

$$|T(jw)| \quad < \quad \frac{|N(jw;\theta)/D_F(jw)|}{|\Delta_N|}, \ \forall w \tag{2.1}$$

$$|S(jw)| \quad < \quad \frac{|D(jw;\theta)/D_F(jw)|}{|\Delta_D|}, \ \forall w \tag{2.2}$$

where N, D are the numerator and denominator of the identified transfer function; Δ_N, Δ_D are defined as $\hat{e}(jw)/\hat{u}(jw)$, $\hat{e}(jw)/\hat{y}(jw)$, respectively, and $e(t) = \phi(t)^\top x_{LS} - b(t)$ is the residual error from the estimation. The frequency domain counterparts of the various signals can be computed via FFTs. Observe that these bounds depend only on the identified system and the error residual but are independent of the particular controller used to close the loop.

The first inequality must be satisfied at middle-to-high frequencies while the second describes a middle-to-low frequency constraint. Their intersection is a constraint for robust stability only; additional objectives can then be added so that the loop sensitivities meet the required performance specifications. Notice that, since $S + T = 1$, these bounds should have an overlapping region where they are both greater than one and in which the eventual crossover frequency will be contained; otherwise the control problem has no solution. If such a region does not exist or if it is too small (less than about one decade), then it is recommended that the identification is repeated, either by adjusting the various filters or by repeating the experiment with a different excitation input. Of course, since these bounds are only estimates (and in fact quite noisy), their violation does not necessarily imply that the control law will destabilize the closed-loop system. They do, however, indicate that any "gross" violations are highly likely to result in an unstable system. In a simplistic, first-pass design, the T and S bounds can be interpreted as constraints on the closed-loop maximum and minimum bandwidth.

Finally, it should be kept in mind that this identification procedure may not result in a good "long term" approximation of the actual system. That is, if the response of the identified system is simulated with the same test input, its output may not yield a good match with the actual output for large time intervals. They should, however, stay close in short time intervals. Such simulations provide one way to validate the results of the identification. Other types of validation include experimentation with different test inputs and computation of the residual error that should remain close to its previous level, without updating the parameter estimates.

2.4.5 Recursive Least Squares

An important feature of the least squares approach is that the computation of the least squares solution can be performed recursively.[3] In a recursive form, the storage requirements are fairly low since the update requires only the current I/O measurements. Its principal application is for on-line estimation but it can also be thought as a cheap way to circumvent computer memory problems (at the expense of speed).

There are many variants of the recursive least squares algorithm that are designed in an attempt to enhance its properties with respect to time-varying parameters and/or exponential forgetting of old data that may be corrupted by transient noise. Most of them fit under the general recursion decribed below.

[3]In such a case the convergence of the estimates to the LS solution is only asymptotic.

For the linear model $\phi_k^\top x = b_k$, $k = 1, 2, \ldots$, the estimate of x, say $\hat{x}_k$ is updated as follows:

$$\hat{x}_{k+1} = \hat{x}_k - \frac{\lambda P_k \phi_k (\phi_k^\top \hat{x}_k - b_k)}{1 + \lambda \phi_k^\top P_k \phi_k} \tag{2.3}$$

$$P_{k+1} = \frac{1}{\alpha} P_k - \frac{\lambda^2 P_k \phi_k \phi_k^\top P_k}{1 + \lambda \phi_k^\top P_k \phi_k} \tag{2.4}$$

$$P_{k+1}^{-1} = \alpha P_k^{-1} + \lambda \phi_k \phi_k^\top + (1 - \alpha) Q \tag{2.5}$$

with the initialization $P_1 = \rho I, \rho \gg 1$ (e.g. $\rho = 10^5$), $x_1 =$ "your best guess" (e.g. the zero vector). The rest of the parameters are selected according to the following guidelines:

- When (2.3) is used with (2.4), $\lambda = 1/\alpha \geq 1$; values strictly greater than one introduce a forgetting factor with time constant (number of significant data points) $1/(1-\alpha)$; $\lambda = \alpha = 1$ is the standard recursive LS algorithm.
- When (2.3) is used with (2.5), $0 < \alpha \leq 1$ has the same forgetting factor interpretation, $\lambda > 0$ is the algorithm gain, and $Q = \epsilon I$, $\epsilon > 0$ is a safeguard against pathological cases (a typical choice is $\epsilon = 10^{-4}$). Notice that this variant requires the inversion of an $m \times m$ matrix at every point.
- (2.3) also works by itself by setting $P_k = I$, $\forall k$. This is a standard gradient algorithm that is very simple to implement, fast to compute, but slow to converge.
- If the excitation is too low, the update law (2.4) cannot prevent P_k from exhibiting unacceptable growth. In our case, this problem can be avoided by a proper design of the test input (amplitude and bandwidth).

2.4.6 Solution of Nonlinear Equations and Nonlinear Optimization

The so-called "Newton" algorithm provides a useful tool in the solution of simultaneous nonlinear equations in many variables. This problem arises in many situations including nonlinear optimization where a necessary condition for extreme points is that the first derivative is zero. The following two Newton-type algorithms are derived based on similar procedures that determine the adjustment of the unknowns by solving (or minimizing) the first term(s) of the Taylor series expansion.

Newton's Algorithm to find a simple root of the system of nonlinear equations $F(x) = 0$: (Here, $x \in \mathbf{R}^m$ and ∇F denotes the gradient of F.)

$$x_{k+1} = x_k - a_k S_k F(x_k)$$

$$S_k = (\nabla F(x_k))^\top [\epsilon_k I + \nabla F(x_k)(\nabla F(x_k))^\top]^{-1}$$

$a_k > 0$, is a step-control parameter, usually less than one; $\epsilon_k = \delta - \lambda_k$ if $\lambda_k < \delta$ and zero otherwise, where $\lambda_k = \min \operatorname{eig}[\nabla F(x_k)(\nabla F(x_k))^\top]$ and $\delta > 0$ is a small design parameter, say 10^{-4}.

Newton's Algorithm to find a local minimum of a scalar function $f(x)$: (Here, $x \in \mathbf{R}^m$ and ∇f, $\nabla^2 f$ denote the gradient and Hessian of f, respectively; $\nabla^2 f$ is assumed to be positive definite at least in a neighborhood of the minimizer.)

$$x_{k+1} = x_k - a_k S_k [\nabla f(x_k)]^\top$$

$$S_k = [\epsilon_k I + \nabla^2 f(x_k)]^{-1}$$

where $\lambda_k = \min \operatorname{eig}[\nabla^2 f(x_k)]$ and a_k, ϵ_k and δ are as before.

Note that in their present simple form, the algorithms may diverge if started "too far" from the solution. Partial remedies do exist but, if required, you should use a standard numerical package instead of re-writing existing software.

2.5 Examples

In the previous sections we briefly presented the basic principles of system identification with several "user-oriented" remarks, intended to supply some working knowledge on the subject. Needless to say, in such a compact presentation of an extensive theory, every comment counts but most of them are soon forgotten without the help of many examples. In the following, we aim to provide a demonstration of the theory with a few simple examples. In order to gain better understanding, t he reader is strongly encouraged to use these examples as a guideline and study the behavior of the corresponding algorithms in several other cases as well.

2.5.1 System Identification

To demonstrate an application of system identification from I/O data let us suppose that we have access to system G in the sense that we can supply inputs and measure the corresponding outputs. As with most practical situations the system is only partially known; for example let us suppose that G describes a furnace temperature that we would like to control. Starting with first principles we may observe that the process could be approximately described by a second order system. One of the time constants serves to describe the slow (convection) dynamics of the bulk temperature (about 30 min) while the other describes the much faster dynamics of the heating elements (about 15 sec). Of course, the actual system is much more complicated but additional effects (distributed parameters, nonlinearities) are ignored for simplicity. (Nevertheless, a higher-than-second-order system was used to produce the simulations). We are also aware of an output disturbance being present of magnitude about 1/2 (degrees). Furthermore, our test input (designed as a perturbation around the system steady state) should be such that the system output variation is about 10 degrees around its nominal value (so that the behavior is approximately linear). Such a constraint limits the achievable signal-to-noise ratio (SNR) in the identification process. Our eventual control objective is to control the temperature with a closed-loop time constant of about 1 min.

Based on this initial knowledge, we decide to design our identification experiment so that we obtain a good estimate of the system transfer function at frequencies around 1 rad/min. We also select to sample the data every 0.05 min. This sampling interval is sufficiently small so that discretization effects will not cause any major problems; however, it is not small enough so that they can be completely ignored. Having decided on the preliminaries, we may now apply our I/O identification procedure. We illustrate two of our options, open-loop and closed-loop identification. Also, for simplicity, we do not use any data prefiltering.

Test Input We generate our test input r in MATLAB as

```
r=prbs_1(10,20,10,200);
```

which, after performing a quick FFT, has sufficient power in the frequency range of interest.

Data Collection We apply the input r to the system, setting $u = r$, (u is the power to the heating elements) and measure the system output (see Fig. 2.2).

For the closed-loop identification, we design a simple controller (pure gain of 0.5) that "speeds-up" the system response considerably. Here we set $u = 4r - y$ and measure the system input and output (see Fig. 2.3).

Remark: Due to the high system gain at low frequencies, it is difficult to select the input in the open-loop-id. case so that the output variation stays within the desired limits (some trial and error may be required). Also, notice that different sequences of test inputs with the same PRBS parameters will in general produce different output variations depending on the specific switching times.

On the other hand, this step is considerably simplified in the closed-loop id. case where the range of output variation is very predictable. Here, maintaining the same PRBS sequence as in the open-loop case, we adjust its amplitude by a factor of four, so that the SNR remains at about the same level.

Parameter Estimation In both cases, we execute the *ioid* MATLAB script file with inputs:

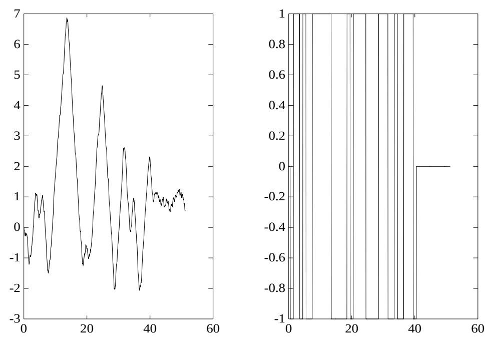
Figure 2.2: Open Loop Identification: I/O measurements.

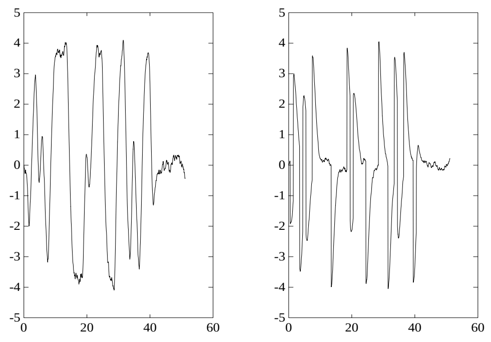
Figure 2.3: Closed Loop Identification: I/O measurements.

```
time step          0.05
identification (1) or validation (0)  1
Model type (0=strictly proper,-1=biproper)  0
filter order       2
filter cutoff      3
prefilter zeros  1
prefilter poles  1
```

I/O id. results (open-loop-id.) The identified transfer function has poles at -1.7, -0.08, a zero at -34 (mostly noise) and DC-gain 22.7. Its frequency response, the FFT-estimated response and the S and T bounds are shown in Fig. 2.4–2.6.

I/O id. results (closed-loop-id.) The identified transfer function has poles at -2.26, -0.075, a zero at +21 (mostly noise) and DC-gain 23.5 Its frequency response, the FFT-estimated response and the S and T bounds are shown in Fig. 2.7–2.9.

Remarks: Both cases produced similar results[4] with the closed-loop id. favoring, by nature, slightly higher frequencies and the open-loop id. favoring slightly lower frequencies (see S and T-Bounds). In both cases, the uncertainty bounds indicate that it should be possible to design a controller that meets the mentioned bandwidth specification.

[4]Such a close agreement should not be expected in general.

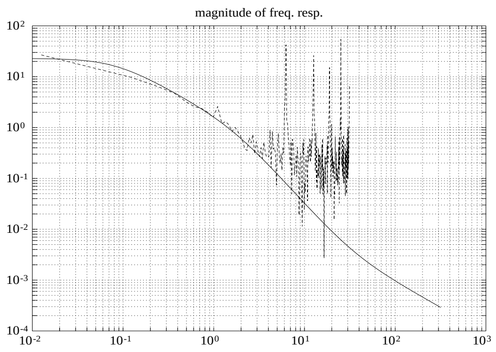

Figure 2.4: Open Loop Identification: Frequency response magnitude.

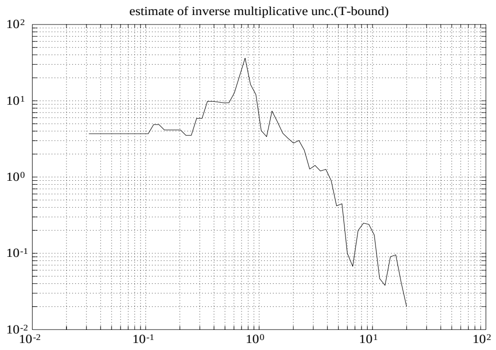

Figure 2.5: Open Loop Identification: Complementary Sensitivity bound.

The non-parametric FFT estimate of the frequency response should be used only as a guideline since it can be very noisy and biased. (There are ways to remedy this problem but they are not considered here.)

Comparison with the actual system So far, we treated our example just like an actual problem. But at this point we have the luxury of knowing the actual system and we can gain some further insight on the identification results.

Fig. 2.10 shows the magnitudes of the frequency response of the actual and identified transfer functions (precision would require the same comparison for the phases). Observe the good match of the responses in the mid-frequencies but the poor match at low and high frequencies.

Even more transparent is the comparison of step responses, shown in Fig. 2.11. As previously mentioned, the identified model could be a very poor predictor of the long-term system response. However, as shown in the detail-plot, they offer an excellent prediction of the initial stage of the step response (for about 5 min.). This figure also illustrates their value for controller design with an intuitive argument: Since the eventual closed-loop system will have a time-constant of about 1 min, only the first few minutes of the step response provide useful information for the prediction of the closed-loop behavior; the large mismatch obtained at, say 15-30 min., is almost irrelevant, since the controller would have taken corrective action long before that time.

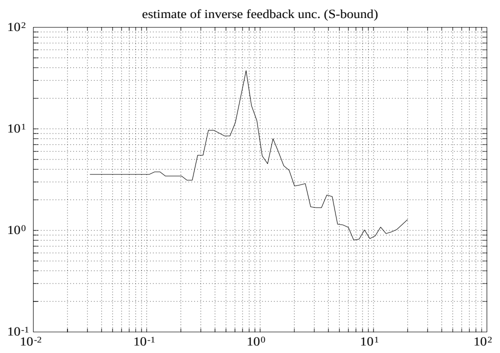

Figure 2.6: Open Loop Identification: Sensitivity bound.

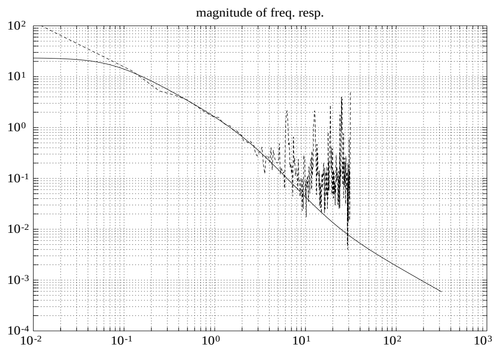

Figure 2.7: Closed Loop Identification: Frequency response magnitude.

2.5.2 Newton Algorithm Examples

Solution of a scalar equation In this example we want to find a solution of $f(\zeta) = \exp(-\pi\zeta/\sqrt{1-\zeta^2}) - 0.5 = 0$, $\zeta \in (0,1)$. For this case, the Newton recursion becomes

$$\zeta_{k+1} = \zeta_k - a_k \frac{\exp(-\pi\zeta/\sqrt{1-\zeta^2}) - 0.5}{\nabla f(\zeta_k)}$$

where the derivative of f is given by the following expression

$$\nabla f(\zeta) = \frac{df}{d\zeta}(\zeta) = -\frac{\pi \exp(-\pi\zeta/\sqrt{1-\zeta^2})}{\sqrt{1-\zeta^2}} \left[1 + \frac{\zeta^2}{1-\zeta^2}\right]$$

Here we used the simplification $\epsilon_k \equiv 0$ since $\nabla f(\zeta)$ vanishes only at 1 and this will not be a source of problems (the reason why requires further analysis).

We test the algorithm starting with initial condition $\zeta_0 = 0.1$ and we find that it converges to 0.2155, within four significant digits, in 3 iterations. Next, we replace the constant 0.5 in the function by 0.1 and repeat the process. The algorithm now requires 5 iteration to converge to the solution 0.5912 within four significant digits. However, if we try the same examples starting with initial condition $\zeta_0 = 0.9$ the algorithm diverges. The reason is that in the first step the required correction is grossly overestimated resulting in a large negative value for ζ that causes the square root to become imaginary. To correct this problem we could use the step-control parameter a_k to prevent the updates from drifting outside the interval (0,1) where the expression makes sense afterall. A partial code for such a modification of the algorithm is given below:

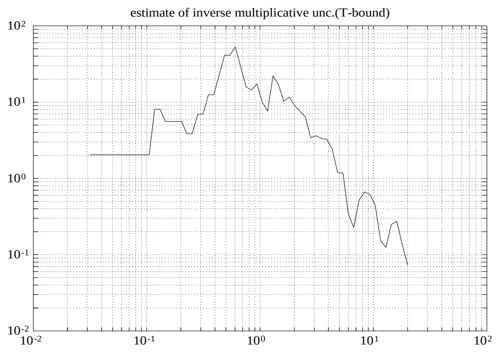

Figure 2.8: Closed Loop Identification: Complementary Sensitivity bound.

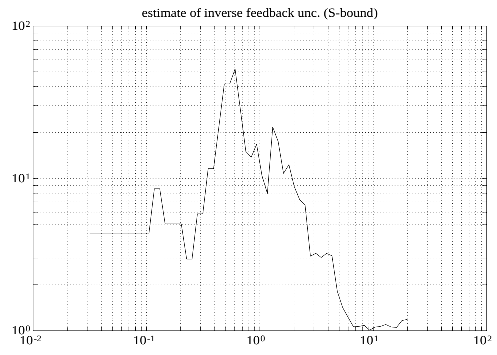

Figure 2.9: Closed Loop Identification: Sensitivity bound.

```
% step 1: compute the basic step size SkFk
ak=1;
   while zk-ak*SkFk <= 0 | zk-ak*SkFk >= 1
        ak=ak/2;
   end
xk=xk-ak*SkFk;
% return to step 1
```

Unfortunately, however, problems cannot always be fixed in such a simple way.

Solution of multiple equations Let us now suppose that we want to find $x \in \mathbf{R}^2$ to satisfy the following two equations, simultaneously:

$$\sin(x_1 + x_2) = 0 \quad ; \quad \cos(x_1^2 + x_2^2) = 0$$

Again, we compute the gradient of f (which is now a 2-by-2 matrix) and apply Newton's algorithm. The MATLAB script file that performs the necessary computations is listed in Section 6.4. Its execution indicates that with the initial condition $x_0 = [1, -0.5]'$ the algorithm converges to a solution [0.8862, -0.8862] within four significant digits in three steps. However, different initial conditions can cause the algorithm to diverge (e.g. $x_0 = [1; 1]$).

Notice that for more complicated/high-dimension problems, the analytical computation of gradients could be very time-consuming, if at all possible. For such problems, a careful numerical computation of the gradient will preserve the algorithm properties, at the expense of speed of execution.

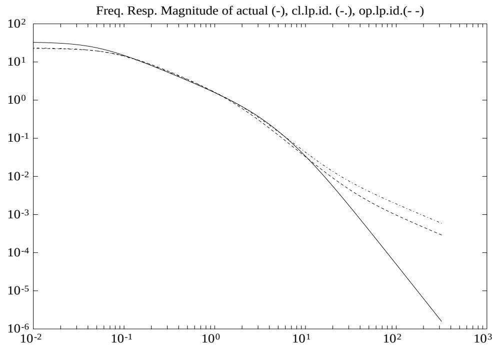

Figure 2.10: Frequency response comparison with the actual system.

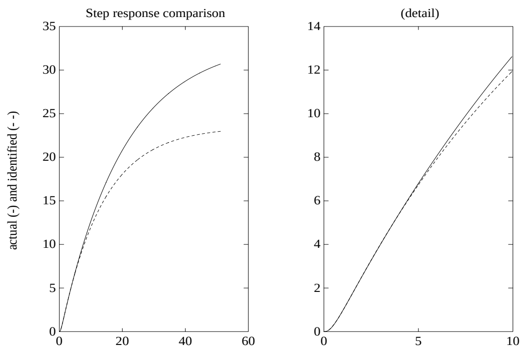

Figure 2.11: Step response comparison with the actual system.

2.6 MATLAB Scripts

2.6.1 I/O identification

```
% script file ioid: Performs I/O system id for a SISO system.
% Requires the plant input "u" and output "y" to be in the
%    workspace.
% It is recommended that u&y have power-of-2 lengths for
%    faster computations.  To avoid excessive bias,
%    the test input should be selected so that the
%    system is approximately "at rest" in the beginning
%    and the end of the id-process.  Also, it is assumed
%    that u&y have been pre-processed to remove constant
%    bias terms (i.e., at rest means (u,y)=(0,0))
% Results: numo,deno: identified system numerator and denominator
%          frw, magno, phaso: frequency response (parametric)
%          fftfr, frpla: system frequency response (fft)
%          fftfr, fftmunc: mult.uncert. bound
%          fftfr, fftmync: feedbk unc. bound

% -------- Initialization

hold off, format short e
stp=input('time step  ');
```

```
nn=length(y);frw=logspace(-2,2.5,100)';frg=logspace(-1.5,1.3,60)';
valid=input('identification (1) or validation (0)  ');
nc=input('Model type (0=strictly proper,-1=biproper)  ');
filord=input('filter order  ');filpol=input('filter cutoff  ');
prenum=input('prefilter zeros  ');preden=input('prefilter poles  ');
t=[0:nn-1]'*stp;clg;subplot(121);plot(t,y);plot(t,u);pause

% -------- auxiliary filter definition

den=poly(-ones(1,filord)*filpol);num=[10];
[f,q,cc,dd]=tf2ss(num,real(den));
cy=eye(length(q),length(q));zq=q*0;n=length(q);

% -------- prefilter definition
nuf=poly([-prenum])*preden(1)/prenum(1); def=poly([-preden]);
   if prenum~=preden
      uf=lsim(nuf,def,u,t); yf=lsim(nuf,def,y,t);
      else, uf=u;yf=y;
   end

% -------- filter states
wu=lsim(f,q,cy,zq,uf,t);wy=lsim(f,q,cy,zq,yf,t);
disp('lsim results')
www=[wu,wy];if nc == -1,www=[www,uf]; end

% -------- parameter estimation
   if valid == 1
      thx=inv(www'*www)*www'*(yf);thx=real(thx);
      th1=thx(1:n);th2=thx(n+1:2*n);
      th3=-thx(length(thx))*min(0,nc);
   end

% -------- Display Results

disp('Estimated parameters')
[th1',th2',th3]             %rem: lth=-th3/(th2'*inv(f)*q)
[numo,deno]=ss2tf(f'+th2*q',th1+th2*th3,q',th3,1);
[magno,phaso]=bode(numo,deno,frw);
disp('Numerator'),numo
disp('Denominator'),deno
disp('Poles'),z1=roots(deno)
disp('Zeros'),z2=roots(numo)

% -------- Compute estimation error without prefiltering

wu=lsim(f,q,cy,zq,u,t);wy=lsim(f,q,cy,zq,y,t);
err=y-[wu,wy,u]*[th1;th2;th3];

% -------- Frequency domain analysis

disp('fft analysis')
fftu=fft(u);fftu=fftu(1:nn/4);fftu=fftu-u(1)/2-u(nn)/2;
ffty=fft(y);ffty=ffty(1:nn/4);ffty=ffty-y(1)/2-y(nn)/2;
fftfr=[0:nn/4-1]'*2*pi/nn/stp;fftfr(1)=fftfr(2)/10;
frpla=ffty./fftu;    %rem: fft estimate of freq.resp.

disp('uncertainty computations')
```

```
ffte=fft(err);ffte=ffte(1:nn/4);ffte=ffte-err(1)/2-err(nn)/2;
frrespo=freqs(numo,deno,fftfr);frrespn=freqs(numo,den,fftfr);
frrespd=freqs(den,deno,fftfr);
fftmunc=abs(ffte./fftu)./abs(frrespn);fftmync=abs(ffte./ffty).*abs(frrespd);
fmunc=smoothin(fftfr,fftmunc,frg);fmync=smoothin(fftfr,fftmync,frg);
% -------- PLOTS

hold off, clg,  plot(t,err), title('prediction error')
pause

clg,subplot(221),loglog(frw,magno),grid,title('frequency response')
subplot(223), semilogx(frw,phaso), grid
subplot(122), z1r=real(z1);z1i=imag(z1);
  if length(z2) > 0
    z2r=real(z2);z2i=imag(z2);
    plot(z1r,z1i,'x',z2r,z2i,'o'),grid,title('pole-zero plots')
  else
    plot(z1r,z1i,'x'), grid,title('pole-zero plots')
  end
pause

clg,loglog(frw,magno,fftfr,abs(frpla))
title('magnitude of freq. resp.'),grid,pause
loglog(frg,(fmunc.^(-1)))
title('estimate of inverse multiplicative unc.(T-bound)'),grid,pause
loglog(frg,abs(fmync.^(-1)))
title('estimate of inverse feedback unc. (S-bound)'),grid,pause
```

2.6.2 RBS Generator

```
function r=prbs_1(n,a,bl,bt);
% function r=prbs_1(n,a,bl,bt);
% n: RBS sequence of 2^n points;
% a: minimum time between switches (~1/BW)
% bl: number of leading zeros
% bt: minimum number of trailing zeros

tns=2^n;
rx=rand(round((tns-bl-bt)/a-.5),1)-.5;
rx=rx*100000;rx=max(rx,-1);rx=min(rx,1);
rt=kron(rx,ones(a,1));
r=[zeros(bl,1);rt;zeros(tns-bl-length(rt),1)];
```

2.6.3 FFT Smoothing

```
function [zsm,zstd]=smoothin(f,z,w);
% [zsm,zstd]=smoothin(f,z,w);
% A quick simple way to smooth frequency responses
% f: input frequency vector, z: frequency resp. in, w: desired frequncy vector
% zsm: averaged z at w, zstd: st. deviation at w

zsm=0*(w);zstd=0*(w);tem=log(z);
   for i=1:length(w)
      k=max(i-1,1);l=min(i+1,length(w));
      ind=find(f <= w(l) & f >= w(k));
         if length(ind) == 0
            ind=[max(find(f <= w(i)));min(find(f >= w(i)))];
         end
      zsm(i)=mean(tem(ind));zstd(i)=std(tem(ind));
   end
zsm=exp(zsm);zstd=exp(zstd);
```

2.6.4 Newton Algorithm Example

```
f=[sin(x(1)+x(2));cos(x'*x)]
df=[cos(x(1)+x(2)),cos(x(1)+x(2));-2*x(1)*sin(x'*x),-2*x(2)*sin(x'*x)];
F=df*df'; m=min(eig(F)); n=length(F);
   if m < 1.e-4, S=F+(m-1.e-4)*eye(n,n);
   else, S=F;
   end
x=x-df'*inv(S)*f
```