# Experimental modelling of dynamic systems

Morten Knudsen

Department of Control Engineering, Aalborg University. 2004

# Contents

# Preface

This lecture note is written for the course 'Simulation and Experimental Modelling' (Simulering og Eksperimentel Modelbestemmelse, PR6-5), a special course at the 6th semester of the line Control Engineering.

Besides the lecture note, the material for the course consists of SENSTOOLS, a Matlab toolkit written specifically to implement the sensitivity approach applied in the lecture note. SENSTOOLS contain main programs for parameter estimation, input design, accuracy verification and supporting functions and tools. So all the user has to program is the simulation program for his particular process.

It is recommended that SENSTOOLS is included as a Matlab toolbox (File - Set Path - Add to Path, or by the Matlab command: addpath (e.g. /staff/mk/public_html/ExpMod/SENSTOOLS (UNIX)).

SENSTOOLS and all other course material can also be downloaded from the course home page http://www.control.auc.dk/~mk/ExpMod/

This version 0.2 of the lecture note is the preliminary version 0.1 with a number of typing errors corrected, an overview of examples added in Chapter 1, and an updated appendix B: Problems.

Version 0.2. January 2004

# Chapter 1

# Introduction

*This lecture note is an attempt to introduce experimental modelling in a way that is easy to learn and apply in practise.*

*In classical system identification a stochastic framework is used for estimating model parameters from noisy measurements. However, the theory for stochastic processes is normally considered difficult, and as the noise is only of limited importance, a new approach has been developed, focussing on the deterministic model instead of the noise - where possible.*

*This chapter gives an introduction to and en overview of the lecture note.*

## 1.1 Models, modelling and system identification.

Mathematical models are the foundation of most scientific and engineering methods. Models can be obtained by either a theoretical approach based on physical laws, or an experimental approach based on obtained measurements from the system.

System identification deals with the problem of building mathematical models of dynamic systems based on observed data from the system, and is thus an experimental modelling method.

In classical system identification a discrete-time model is obtained. Such models can be useful e. g. for simulation and prediction or for design of digital control systems. A number of high quality textbooks in system identification are available [Eykhoff 1974],[Goodwin 1977], [Knudsen T. 1993], [Ljung 1987], [Schoukens 1991], [Söderström 1989], [Unbehauen 1987], and Matlab Toolboxes have been developed.

In spite of that, system identification is not a generally known tool outside the control engineering area.

## 1.2 Why this lecture note ?

Reasons why system identification is not as widely known and applied as it deserves may be, that it is considered complicated and difficult to learn and apply in practise.

This lecture note is an attempt to introduce experimental modelling in a way that is easy to learn and apply in practise. A method has been developed, where physical parameters of continuous-time models are estimated directly. While system identification apply a stochastic framework focussing on the noise, this is toned down in this note. The reasons are that it is considered difficult and requires assumptions about the noise that are normally not available in practice, and that the noise is of limited importance when long sequences of measurements are available.

The merits of the method and the presentation in this lecture note are:

- a simple fundamental approach, illustrated graphically
- continuous-time models with physically significant parameters
- any model structure is appropriate, linear and non-linear, distributed and lumped parameters, time delay etc.
- stochastic aspects are reduced to a minimum
- robustness to violations of theoretical assumptions and approximations
- a sensitivity approach useful for choice of model structure, for experiment design, and for accuracy verification
- all in all, compatibility with physical insight
- the method and the presentation is developed with generally preferred learning styles in mind

## 1.3 SENSTOOLS

SENSTOOLS is a Matlab toolkit written specifically to the sensitivity approach applied in the lecture note. SENSTOOLS contains main programs for parameter estimation, input design and accuracy verification and supporting functions and tools. So all the user has to program is the simulation program for his particular process - and for this there are plenty of helpful guidelines and examples.

Besides, SENSTOOLS contains the data files needed for the problems, and the programs for the examples. This means that the examples can be executed live in Matlab - which is much more illustrative, than just reading them.

## 1.4 Procedure for experimental modelling

A procedure for experimental modelling - and specifically direct estimation of physical parameters in dynamic systems - can be divided into five parts:

1. Model structure determination:

   The model structure is determined from basic physical laws and empirical considerations. A simulation program is constructed. (Chapter 4).

2. Experiment design:

   In particular, a good input signal is important. It is either chosen from common-sense considerations, or designed as a signal optimizing characteristic parameter sensitivity measures. (Chapter 8).

3. Experiment:

   The process is excited by the input signal, and corresponding sequences of input and output signal values are sampled and stored in a computer.

4. Parameter estimation:

   The simulation model parameters are adjusted until minimum of the deviation between the sampled system output and the simulated model output (the model error). As a measure for this deviation is used a performance function containing the sum of the squared deviations at the sampling instants. This is illustrated in Fig.1.1. (Chapter 3 + 5).

5. Model validation:

   The correctness of the model structure and the accuracy of the parameter estimates are evaluated, based on the model fit and characteristic parameter sensitivity measures. (Chapter 6).

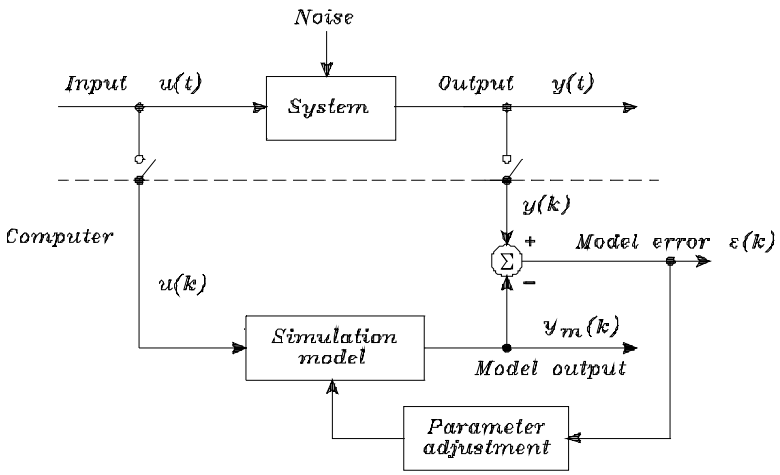


Figure 1.1 Parameter estimation principle. The model parameters are adjusted until minimum of the model error, i.e. the deviation between the sampled system output and the simulated model output.

This procedure appears very similar to the one used for estimating the discrete-time parameters in classical system identification. The main differences are that the model is a continuous-time, possibly non-linear simulation model, and the procedure for minimizing the performance function is operating directly on the physical/continuous-time parameters.

## 1.5 Organization of the lecture note

The chapters, where the subject of the procedure steps are treated, are indicated above. It appears, that it has not been appropriate to use the same succession in the lecture note as in the procedure.

*Chapter 2* (Graphical model fitting) describes how a model can be fitted to experimental data manually by graphical methods, in the time domain (step response) as well as in the frequency domain (Bode plots). The purpose is twofold: the method is applicable - even in this computer age, and it gives a pedagogical introduction to the computer based parameter estimation principle of Fig. 1.1.

*Chapter 3* (System Identification) deals with computer fitting of models to measured input-output data - i.e. system identification. It is shown how the problem of model fitting is transformed into an optimization problem, and numerical methods for the minimum search are developed - the Gauss Newton algorithm in particular.

*Chapter 4* (Modelling, model description and simulation) gives some fundamental concepts in modelling and simulation, mathematical models and discretization methods. Methods for simulation of linear and nonlinear models with Matlab is treated and illustrated with examples. These examples may be handy, when simulation programs are constructed in step 1 of the parameter estimation procedure.

*Chapter 5* (SENSTOOLS for parameter estimation) gives an introduction to Senstools, in particular how parameter estimation with Senstools is done. The procedure is illustrated by examples.

*Chapter 6* (Parameter accuracy and sensitivity) shows how information about the model fit, combined with so-called sensitivity measures, can give valuable information about the accuracy of the obtained model. Physically understandable expressions for errors of the parameter estimate, caused by noise and undermodelling are developed.

*Chapter 7* (Frequency-domain considerations) demonstrates how parameter fitting in the time domain is equivalent to fitting in the frequency domain. It is also shown how frequency-domain considerations can be very valuable supplement for a better understanding of the time domain fitting.

*Chapter 8* (Input signal design) deals with conditions for the experiment, in particular the input signal used. An optimal input can be designed using the characteristic sensitivity measures.

*Appendix A* (SENSTOOLS) is a brief manual for Senstools, describing the facilities, and giving an overview of Senstools files and a listing of the most important ones.

*Appendix B* (Problems) is a collection of problems and exercises, some apply real life data stored in Senstools.

*Appendix C* (Proof of Theorem 1) gives a prof for the expression used for determination of the minimum sensitivity.

*Appendix D* (List of the symbols used) is a list of the most important symbols.

## 1.6 Examples overview

| Chapter | Examples | Programs, data |
|---|---|---|
| 3. Sys ident | 1. P($\tau$)-plot | |
| | 2. P(K,$\tau$)-plot | |
| | 3. Newton iter. v=a/x+bx$^3$ | |
| 4. Modelling | 4. Sim K/(1+s*$\tau$) in 3 ways | simktauloop, simktaufilt and simktau |
| | 5. Sim saturation and ktau | simsatktau.m |
| | 6. Loudsp: physical modelling | |
| | 7. Loudsp: linear simulation | simsplf, testsimsplf |
| | 8. Loudsp: nonlinear simulation | simspn, testsimspnst |
| 5. Senstools | 9. mainest: 3 vays | simkutau, progprogkutau, simktau, measkutau, measdcml |
| | 10. DC-motor, 2-variable: mainest | simdcml, measdcml |
| 6. Par acc | 11. Sensitivity ellipse and measures | sellip |
| | 12. Sens: sine input, k and tau, ellipses | psinf, sens, sellip4sinusDiary |
| | 13. Overparamterization A, B and C | simABC, measABC, |
| | 14. Parameter spread, noise + underm | |
| | 15. Equiv par error, Loudsp, sellip | pacr.m |
| 7. Freq dom | 16. Freq. Domaine simulation for est | simfktau.m |
| 8. Expdesign | 17. maininp: Opt. inp square wave (default) | |
| | 18. Opt. Inp. Inpsqram: nonlinear, kutau | progproginpkutau, progdatainpkutau, simkutau |

DC-motor DEMO

| | | |
|---|---|---|
| Linear | clear, process='dcml', no='8', mainest | simdcml.m, measdcml8.mat progdatadcml8.mat |
| Nonlinear | clear, process ='dcmn', no='8', mainest | simdcmn.m, measdcmn8.mat progdatadcmn8.mat |

## 1.7 Applications

Senstools and the sensitivity approach for experimental modelling has been applied in numerous research and student projects. Examples are: Ships and marine systems [Blanke 1998], [Blanke 1999], wind turbines, loudspeakers [Knudsen 1996], induction motors [Børsting 1994], DC-motors [Knudsen 1995], heat exchangers, human tissue for hyperthermia cancer therapy [Knudsen 1986], kidney and cerebellar blood flow.

# Chapter 2

# Graphical model fitting

*Experimental modelling consists in adjusting the parameters of a chosen model structure in order to fit the model output to measured system output data. The fitting can be done either manually by graphical methods, or by having a computer perform an optimization procedure.*

In this chapter a method for graphical fitting of a model to time response or frequency response data is first described. A simple model based on transient response or frequency response data is often valuable for controller design or as a basis for more advanced modelling. In addition the graphical methods give a valuable understanding of model fitting in general.

## 2.1 Models from step response data

Step response data are, in particular, quick and relatively easy to obtain, and they give obvious information of fundamental dynamic and static qualities. To obtain a good signal-to-noise ratio, the transient response must be highly noticeable, but such a severe disturbance is not always acceptable.

A typical step response is shown in Fig. 2.1. Notice that the measurement is contaminated with high frequent noise.

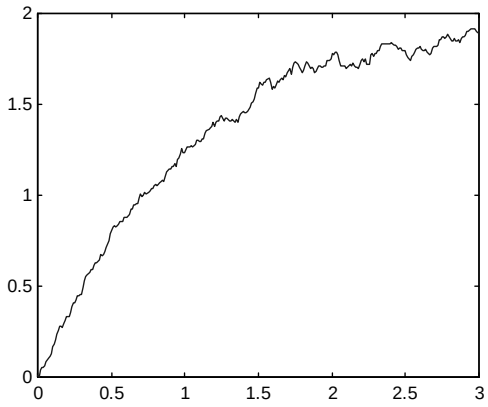


Figure 2.1 Measured step response

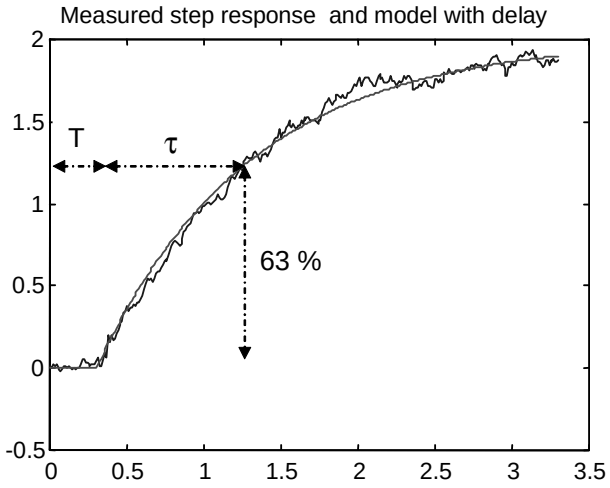


Figure 2.1: Figure 2.2 Graphical determination of K and $\tau$ from step response

It appears from Fig. 2.1 that a first order model might fit the data well, so we choose the model *structure*

$$G_m(s) = \frac{K}{1 + s\tau} \tag{2.1}$$

For a step input of magnitude a:

$$U(s) = \frac{a}{s} \tag{2.2}$$

the response is

$$Y(s) = \frac{aK}{s(1 + s\tau)} \tag{2.3}$$

or in the time-domain

$$y(t) = Ka(1 - e^{-t/\tau}) \tag{2.4}$$

$$t \to \infty : \quad y(\infty) = Ka \implies K = \frac{y(\infty)}{a} \tag{2.5}$$

$$t = \tau : \quad y(\tau) = Ka(1 - e^{-1}) = Ka \cdot 0.63 \tag{2.6}$$

It is now possible to determine K and $\tau$ in the following way:
Sketch a smooth exponential curve through the measured output data (graphical model fitting).
The steady-state gain K is now determined as the steady-state value of the model output $y(\infty)$ divided by the step magnitude a.
The time constant $\tau$ is determined as the time it takes the model output to reach 63 % of the steady-state value, see Fig. 2.2.

In control systems a time delay is crucial, and in many processes there is a true transport delay. Besides, a system with one large time constant and a number of small time constants can

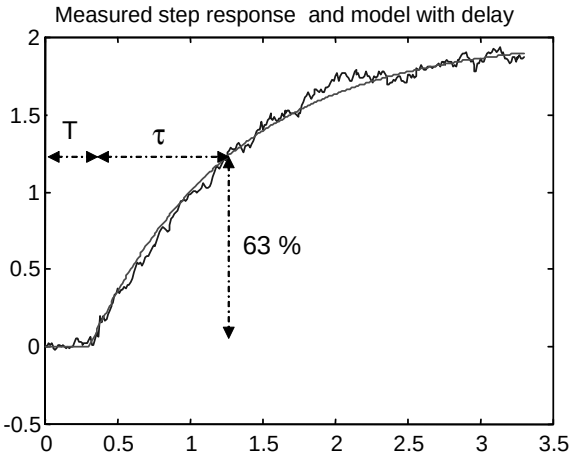


Figure 2.2: Figure 2.3 Graphical determination delay T and time constant $\tau$ from step response

be approximated by the large time constant plus a time delay equal to the sum of the small time constants. Consequently, a model structure with one time constant and one time delay is generally more adequate:

$$G_m(s) = \frac{K}{1 + s\tau} e^{-sT} \tag{2.7}$$

For this model the three parameters, time delay T, time constant $\tau$ and gain K, can be determined by graphical model fitting, see Fig. 2.3.

## 2.2 Models from frequency response data

Frequency response data are more complicated and substantially more time consuming to obtain than transient response information. This is especially so if the time constants of the process are large. But the model obtained is generally more accurate because of better noise filtering ability of the frequency response method.

The first order model with delay, equation (2.7) can be fitted to the frequency response data in a Bode plot as shown in Fig. 2.4

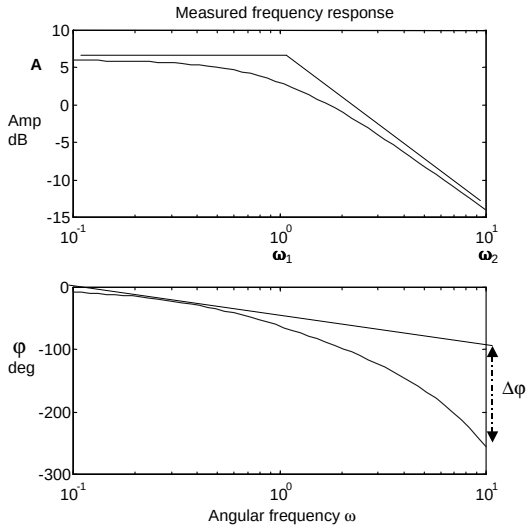


Figure 2.4 Graphical determination of $A$, $\omega_1$ , and $\Delta\varphi$ from frequency response data

First the asymptotes of the amplitude characteristics are sketched with slopes 0 and -20 dB/decade. The time constant $\tau$ can then be determined as the inverse of the frequency at the intersection of the asymptotes, $\omega_1$, and K is determined from the amplitude, A= 20 log K of the 0-slope asymptote:

$$\tau = \frac{1}{\omega_1} \tag{2.8}$$

$$20 \log K = A \implies K = \log^{-1}(\frac{A}{20}) \tag{2.9}$$

Asymptotic phase characteristic for the time constant alone are then sketched as

$$\omega \leq \omega_1/10 \qquad \varphi = 0 \tag{2.10}$$

$$\omega_1/10 < \omega < 10\omega_1 \qquad \varphi = -45^0(1 + \log(\omega/\omega_1)) \tag{2.11}$$

$$\omega > 10\omega_1 \qquad \varphi = -90^0 \tag{2.12}$$

The extra phase $\Delta\varphi$, due to the delay T at a frequency $\omega_2$ is then determined as the phase deviation between the asymptotic phase characteristic of the time constant alone and the phase characteristic for the entire model with delay as indicated in Fig. 2.4. The corresponding delay T is calculated as:

$$-\Delta\varphi = \angle e^{-j\omega_2 T} = -\frac{180}{\pi}\omega_2 T \quad \implies \quad T = \frac{\pi}{180}\frac{\Delta\varphi}{\omega_2} \tag{2.13}$$

## 2.3 Resumé

Two model structures, a model with one time constant or a model with a time constant and a delay, can easily be fitted to step response data or frequency response data.

The models obtained may be sufficient, e.g. for controller design. If a more accurate model is required the simple models may be useful for experiment design and for starting values of parameter iteration in system identification methods, as we shall see in the following chapters.

# Chapter 3

# System Identification

*System identification is fitting models to measured input-output data from dynamic systems by having a computer perform an optimization procedure.*

Computer fitting has several advantages compared to manual fitting: It is much less dependent on special input signals, it can provide very detailed and accurate models, and it may be less time consuming. In this IT-era, computer fitting of models, i.e. system identification, is generally preferred, and it is evidently the main subject of this lecture note.

In this chapter the fundamental principle for system identification is first described, and three main classes of system identification methods are defined.

Next is shown in more details how the problem of model fitting is reduced to an optimization problem, and numerical methods for minimum search are described and illustrated by examples.

The chapter is concluded by a condensed procedure for direct estimation of physical parameters using a Gauss-Newton algorithm.

## 3.1 Fundamental principle

The fundamental principle for estimation of the parameters in dynamic systems is illustrated in fig.2.1.

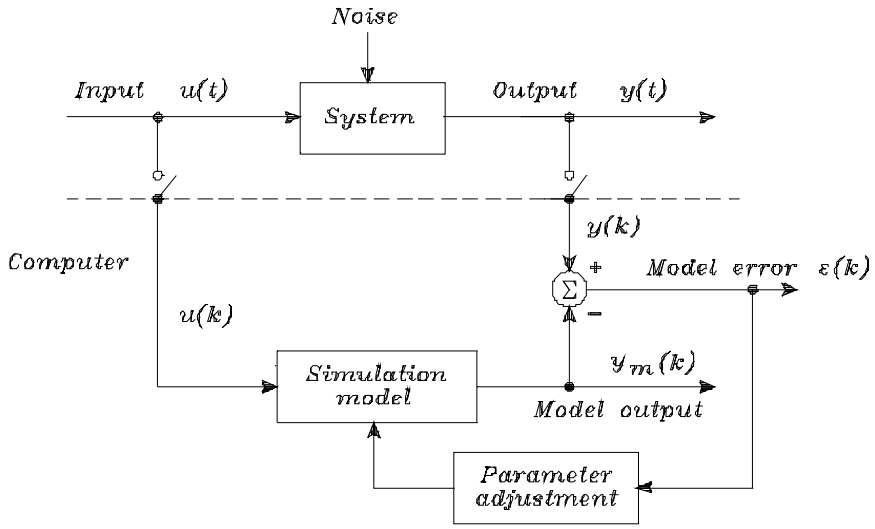


Figure 3.1 Parameter estimation principle

- The process (system) is excited by a suitable input signal, and corresponding sequences of input and output signal values are measured and stored in a computer.
- The system is simulated by the computer with guessed initial values of the unknown parameters, and the simulation model is excited by the sampled input sequence.
- The model parameters are adjusted until minimum of the deviation between the sampled system output and the simulated model output (the model output error $\varepsilon$). As a measure for this deviation is used a performance function $P(\theta)$, typically containing the sum of the squared model output error at the sampling instants.

This principle applies for all system identification and parameter estimation methods. Various methods are characterized by different model types:

**A Linear discrete-time model**

This is classical system identification, useful when a linear model is sufficient and the physical structure and parameters are of no special interest. The obtained model is a discrete-time transfer function, useful for simulation or digital control design.

**B Neural network**

This method is useful for strongly nonlinear systems with a complicated structure. Neural networks provide a general nonlinear model structure, that has no relation to the actual physical structure, and a large number of parameters with no relation to the physical parameters.

**C General simulation model**

This is any mathematical model, that can be simulated e.g. with Matlab. It requires a realistic physical model structure, typically developed by theoretical modelling, and will then give accurate values of the physical parameters.

This method, direct estimation of physical parameters, is the one employed in this lecture note.

## 3.2 Computer fitting by minimization

Consider again the example in Chapter 2, where the step response

$$y_m(t) = Ka(1 - e^{-t/\tau}) \tag{3.1}$$

of the model structure

$$G_m(s) = \frac{K}{1 + s\tau} \tag{3.2}$$

should be graphically fitted to measured output $y(t)$ in Fig. 3.2

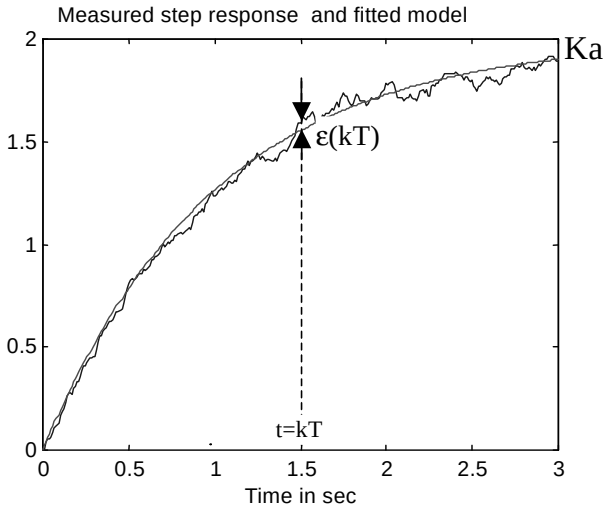


Figure 3.2 Measured and model response. Notice the interpretation of the model error $\varepsilon(kT)$

If instead we apply the computer fitting approach, the model output of equation 3.1 shall be fitted to the measured output by adjusting the parameters K and $\tau$ until minimum of the performance function

$$P(\theta) = \frac{1}{2N} \sum_{k=1}^{N} \varepsilon^2(kT, \theta) \tag{3.3}$$

where $\varepsilon(kT, \theta)$ is the model error at the time $t = kT$ , see Fig. 3.2

$$\varepsilon(kT, \theta) = y(kT) - y_m(kT, \theta) \tag{3.4}$$

$T$ is the sampling time, $\theta = [K, \tau]^T$ is a vector containing the model parameters, and $N$ is the number of samples, $k = 1.....N$.

We see, that the graphical model fitting approach of Chapter 2 and the parameter estimation (computer fitting, system identification) approach of this chapter have essential similarities. The obvious difference is the fitting method. This difference gives rise to the following advantages of the computer fitting approach:

- any input signal may be used, not just transient and sine waves

- complicated model structures with a high number of parameters may be used
- better accuracy can be obtained
- time-consuming graphical constructions are avoided

The main disadvantage is, that it does not automatically give the same physical understanding and immediate feeling of model accuracy. This has, however been build on the method.

## 3.3 Finding the minimum

The problem of model fitting is now reduced to determination of the parameter value $\theta_o$ minimizing the performance function $P(\theta)$ in equation (3.3).

Before we introduce the minimization methods, let us see a graphical picture of the problem.

**Example 1** *One model parameter.*

Assume we have measured the input u and the output $y_0$ of an unknown system, see Fig.3.3, and we shall fit a model to these data.

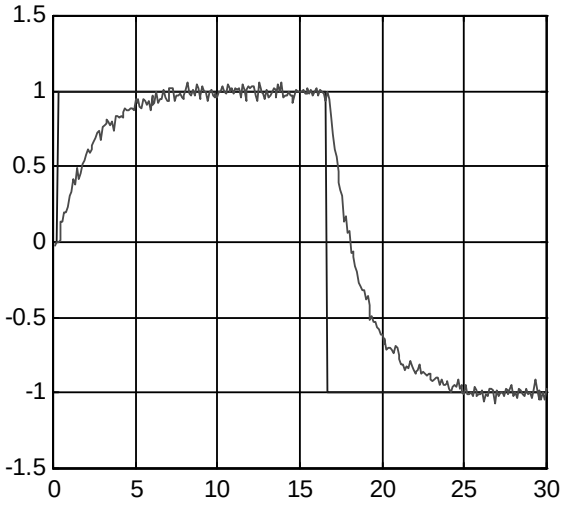


Figure 3.3 Square wave input and corresponding output

First we chose a model with only one unknown parameter, the time constant $\tau$, i.e., $\theta = \tau$ and the model structure is

$$\frac{Y_m(s)}{U(s)} = \frac{1}{1+s\tau} \tag{3.5}$$

For a number of different $\tau$-values we next calculate the model output $y_m(t,\tau)$ and the corresponding value of the performance function equation (3.3)

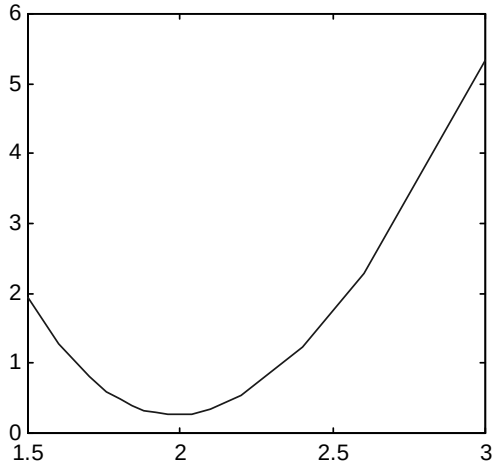


Figure 3.4 Performance function $P(\theta)$ as a function of $\theta$.One dimensional case $\theta = \tau$.

In Fig. 3.4 $P(\theta)$ is plotted as a function of $\theta$.

The parameter value minimizing the performance function is seen to be $\tau = 2$.

The minimum value of the performance function $\mathrm{P}(\theta)$ is seen to be $\mathrm{P}(\theta_0) > 0$. The model output can not possibly fit the measured output exactly because of noise on the measured output.

■

**Example 2** *Two model parameters*

Next we chose a model with two unknown parameters, gain and time constant, i.e. $\theta = [K \quad \tau]^T$, and the model structure is

$$\frac{Y_m(s)}{U(s)} = \frac{K}{1 + s\tau} \tag{3.6}$$

Now the performance function as a function of $K$ and $\tau$ is a two dimensional surface, depicted by level curves in Fig. 3.5.

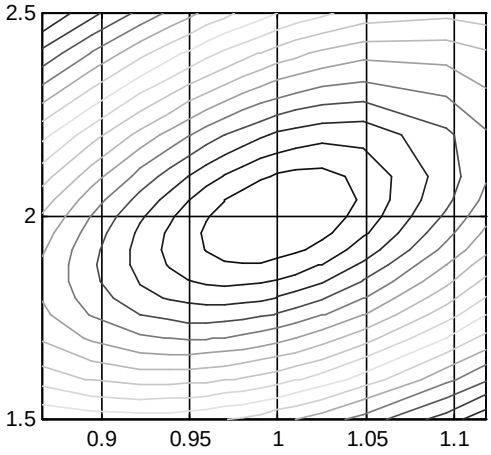


Figure 3.5 Performance function $P(\theta)$ as a function of $\theta$. Two dimensional case $\theta = [K, \tau]^T$

It is seen that the parameter values minimizing $P(\theta)$ are approximately $K = 1, \tau = 2$.
■

### 3.3.1 Minimum of a function

The condition for minimum $\theta = \theta_0$ of a multi variable function

$$P(\theta) = \frac{1}{2N}\sum_{k=1}^{N}(y(kT) - y_m(kT,\theta))^2 \tag{3.7}$$

is, that the first partial derivatives are zero, i.e. the gradient vector is the zero vector

$$G(\theta_0) = \frac{\partial P(\theta)}{\partial \theta} \mid_{\theta=\theta_0} = 0 \tag{3.8}$$

and that the second derivative, i.e. the Hessian,

$$H(\theta) = \frac{\partial^2 P(\theta)}{\partial \theta \partial \theta^T} \mid_{\theta=\theta_0} \tag{3.9}$$

is positive definite. This requires that

$$v^T H v > 0 \tag{3.10}$$

for any vector $v$.

### 3.3.2 Numerical methods for minimum search.

In general the equation 3.8 does not have an explicit solution, so the parameter set $\theta = \theta_0$ minimizing the performance function 3.7 must be determined by numerical search methods.

The idea is to start at some point $\theta = \theta_i$ and move on stepwise to points at which P is smaller, i.e.

$$\theta_{i+1} = \theta_i + s_{i+1} r_{i+1} \quad \text{so that} \quad P(\theta_{i+1}) < P(\theta_i) \tag{3.11}$$

where $r_{i+1}$ is a vector determining the direction of the step, and $s_{i+1}$ is a scalar determining the step length. The two most preferred methods for determination of $r_{i+1}$ and $s_{i+1}$ are

#### 1) Steepest descent

Here we choose

$$r_{i+1} = -G(\theta_i) \tag{3.12}$$

taking a step in direction of the negative gradient, i.e. the direction of maximum decrease (hence the name). The step length $s_{i+1}$ can initially be chosen to a small value. If the step results in a smaller P-value, we can try to increase s in the next step. If a larger P-value is obtain by the step, it means that the step has been too large - we have stepped over the valley - and we must go back and try a smaller step:

$$\text{if} \quad P(\theta_{i+1}) \quad \leq \quad P(\theta_i) \quad \text{then} \quad s_{i+2} = 2s_{i+1} \tag{3.13}$$

$$\text{if} \quad P(\theta_{i+1}) \quad > \quad P(\theta_i) \quad \text{then} \quad s_{i+2} = 0.5s_{i+1} \tag{3.14}$$

The method of steepest descent is reliable, as it will find a minimum for sufficiently small s-values, but the convergence rate is slow close to the minimum, where the gradient approach zero.

**2) The Newton method**

Newton's method for solving equation (3.8) is to use a second order Taylor approximation $\widetilde{P}$ of $P$ at the starting point, $\theta = \theta_i$

$$\begin{aligned} P(\theta) &\approx \widetilde{P}(\theta) \triangleq P(\theta_i) + (\theta - \theta_i)^T G(\theta_i) + \frac{1}{2}(\theta - \theta_i)^T H(\theta_i)(\theta - \theta_i) && (3.15)\\ \frac{\partial \widetilde{P}(\theta)}{\partial \theta} &= 0 + G(\theta_i) + H(\theta_i)(\theta - \theta_i) && (3.16)\\ \frac{\partial \widetilde{P}(\theta)}{\partial \theta} &= 0 \Rightarrow && (3.17)\\ \theta &= \theta_{i+1} = \theta_i - H^{-1}(\theta_i)G(\theta_i) && (3.18) \end{aligned}$$

where $G(\theta)$ and $H(\theta)$ are the first and second derivatives of $P(\theta)$, see equations (3.8) and (3.9). The minimum $\theta_{i+1}$ of $\widetilde{P}$ is determined by setting the derivative equal to the zero vector. In the next iteration $\theta_{i+1}$ is used as a new starting point.

As stop criterion can be used a minimum value of the relative parameter update or maximum number of iterations. For complex problems it is recommended to use both.

**Example 3** *Newton iteration*

(Run the example in Matlab: example3.m and watch how the iteration progresses!)

A Newton procedure is illustrated with a simple one-dimensional example.

Determine the minimum of the nonlinear function:

$$v = \frac{a}{x} + bx^3 \tag{3.19}$$

for $a = 100,\ \ b = 0.0001$

As the function is known, it is easy to calculate G and H as the first and second derivative (see the Matlab implementation example33.m for the details).

In Fig. 3.6 the function v(x) is shown. We select a starting point fairly far from the minimum: $x = 80$. Fig. 3.7 shows the Taylor approximations for each iteration. The minimum of the first Taylor approximation is $x = 40.6$, and after the 4th iteration, the stop criterion is reached, as relative update is less than 0.1 %. The minimizing value is $x = 24.0281$.

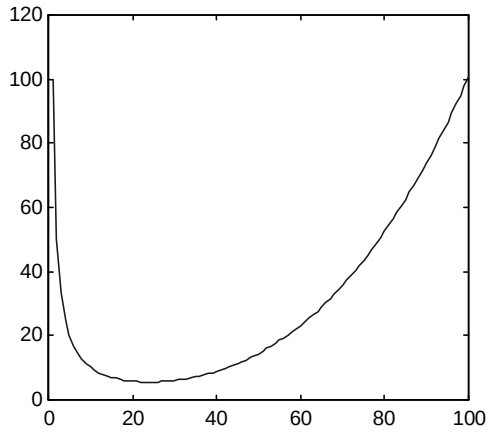


Figure 3.6 The function v(x)

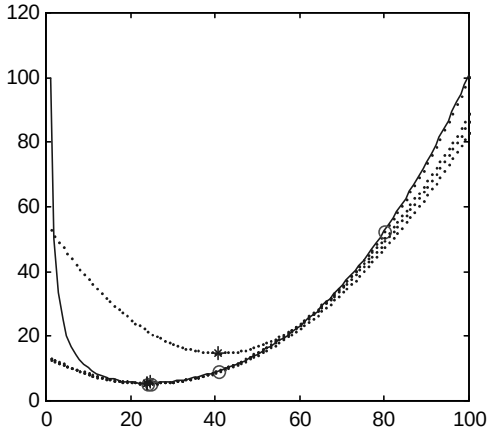


Figure 3.7 Taylor approximations in each iteration

■

### 3.3.3 The Gauss-Newton method.

Newton's method require determination of the first and second derivatives $G(\theta)$ and $H(\theta)$ of the performance function $P(\theta)$. In general they cannot be determined analytically. However, Newton has shown how approximated derivative can be found for a function of the type

$$P(\theta) = \frac{1}{2N}\sum_{k=1}^{N}(y(kT) - y_m(kT,\theta))^2 \tag{3.20}$$

The gradient $G(\theta)$ is found as

$$G(\theta) = \frac{\partial P(\theta)}{\partial \theta} = \frac{1}{N}\sum_{k=1}^{N}(y(kT) - y_m(kT,\theta))(-\frac{\partial y_m(kT,\theta)}{\partial \theta}) = -\frac{1}{N}\sum_{k=1}^{N}\varepsilon(kT,\theta)\psi(kT,\theta) \tag{3.21}$$

where $\psi(kT,\theta)$ is denoted the model gradient

$$\psi(kT,\theta) \triangleq \frac{\partial y_m(kT,\theta)}{\partial \theta} \tag{3.22}$$

The Hessian $H(\theta)$ is found as

$$H(\theta) = \frac{\partial^2 P(\theta)}{\partial\theta\partial\theta^T} = \frac{\partial G(\theta)}{\partial\theta} = \frac{1}{N}\sum_{k=1}^{N}\psi(kT,\theta)\psi^T(kT,\theta) - \frac{\partial\psi(kT,\theta)}{\partial\theta}\varepsilon(kT,\theta) \tag{3.23}$$

$$H(\theta) \approx \widetilde{H}(\theta) \triangleq \frac{1}{N}\sum_{k=1}^{N}\psi(kT,\theta)\psi^T(kT,\theta) \tag{3.24}$$

The approximation of equation (3.24) corresponds to assuming that the model gradient $\psi(kT,\theta)$ is independent of $\theta$, i.e. the model output $y_m(kT,\theta)$ is a linear function of $\theta$ (the model is linear in the parameters). This is a similar approximation as the 2nd order Taylor approximation in the Newton method, as a model linear in the parameters gives a quadratic performance function. The approximation is also augmented by the fact that the model error $\varepsilon(kT,\theta)$ is small for $\theta$ close to the minimum.

For determination of the gradient $G$ and the approximated Hessian $\widetilde{H}$ it is, consequently, necessary to calculate the model gradient $\psi$.

## 3.4 Direct estimation of physical parameters

The methods for estimating model parameters by a minimum search procedure is now in place, and we shall in condensed form give the procedure for direct estimation of physical parameters using a Gauss-Newton algorithm:

A model in form of a simulation program can determine the model output $y_m$ for a given input and given values of the model parameters:

$$y_m(k) = F(u_N, \theta) \tag{3.25}$$

where $\theta$ is a vector containing the physical parameters, and $u_N$ is the input sequence with N samples. F is a, normally nonlinear, function of $u_N$ and $\theta$.

The quadratic performance function $P(\theta)$ to be minimized is

$$P(\theta) = \frac{1}{2N}\sum_{k=1}^{N}\varepsilon^2(k,\theta) \tag{3.26}$$

where the model output error is

$$\varepsilon(k) = y(kT) - y_m(kT,\theta) \tag{3.27}$$

and the parameter estimate of $\theta$ based on N input-output data points, $u_N$ and $y_N$ is then the value $\theta_N$ minimizing $P(\theta)$

$$\theta_N = \arg\min_{\theta} P(u_N, y_N, \theta) \tag{3.28}$$

Using a Gauss-Newton algorithm for the minimization, the estimate $\theta_N$ of the unknown parameters can thus be calculated. The parameter updating algorithm is

$$\theta_{i+1} = \theta_i - \widetilde{H}(\theta_i)^{-1}G(\theta_i)$$

The gradient G and the approximated Hessian $\widetilde{H}$ can be determined from the model gradient

$$\psi(k,\theta) = \frac{\partial y_m(k,\theta)}{\partial\theta} \tag{3.29}$$

as

$$G(\theta) = -\frac{1}{N}\sum_{k=1}^{N}\varepsilon(k,\theta)\psi(k,\theta) \tag{3.30}$$

and

$$\widetilde{H}(\theta) = \frac{1}{N}\sum_{k=1}^{N}\psi(k,\theta)\psi^T(k,\theta) \tag{3.31}$$

The model gradient $\psi$ (dxN matrix) can be determined by a numerical differentiation. The elements of the j'th row $\psi_j$ is calculated by giving the j'th parameter a small deviation $\Delta\theta_j = d\cdot\theta_j, (d = 10^{-4}$ for example) and calculating the corresponding output of the simulation model,

$$\psi_j(k) \approx \frac{\Delta y_m(k,\theta_j)}{\Delta\theta_j} = \frac{y_m(k,\theta_j+\Delta\theta_j) - y_m(k,\theta_j)}{\Delta\theta_j} \tag{3.32}$$

The numerical differentiation and the Gauss-Newton algorithm are both implemented in Senstools, as *psinf.m* and *gausnewt.m* respectively.

## 3.5 Resumé

The - extremely important - fundamental principle for system identification is illustrated in Fig. 3.1. This principle converts the model fitting problem to the problem of minimizing a performance function

$$P(\theta) = \frac{1}{2N} \sum_{k=1}^{N} (y(kT) - y_m(kT, \theta))^2 \tag{3.33}$$

as the optimal parameter values $\theta_N$ are the ones minimizing $P(\theta)$.

The minimum can be determined by an iterative Gauss-Newton procedure, requiring calculation of the first and second derivatives G and H. Both of these can be determined from the model gradient $\psi(kT, \theta)$.

To proceed with the computer fitting approach based on a general simulation model, we will need: simulation models (!). This is the subject of the following chapter.

# Chapter 4

# Modelling, model description and simulation

*Some fundamental concepts and terms in models, modelling, and simulation are briefly defined and discussed. Next mathematical model description and discretization methods are reviewed. Finally simulation of linear and nonlinear dynamic models with Matlab is treated with many examples.*

This chapter is a mixture of some soft stuff: concepts necessary to understand the fundamentals of models, modelling, and simulation, some harder stuff: mathematical model description, and some very-much-to-the-point-stuf: making Matlab simulation programs. The latter is absolutely essential for practical application of the parameter estimation method.

Mathematical model description and discretization is treated in details in [Franklin 1994], [Franklin 1998].

## 4.1 Models and modelling.

Some of the fundamental concepts and terms in modelling of dynamic systems are uniquely defined, but for others rigorous definitions are not required, and may not exist. In Table 4.1 is a list of the main characterizations of models and modelling, arranged as alternatives.

The concepts most relevant in this lecture note are marked with bold types. In the following the most important concepts and characterizations and their relationships will be discussed.

The main concepts in this lecture note is *model* which we will define as: *a representation - in a usable form - of the essential aspects of a system.* Two other important concepts are modelling and system identification. The term *modelling* shall be used in a wide sense, as a method to derive a model. It is subdivided into theoretical (or physical) modelling - i.e. developing models from basic physical laws, and experimental modelling - i.e. developing models from experimental results.

*System identification* deals with the problem of building mathematical models of dynamic systems based on observed data from the system. From this it appears that system identification is essentially experimental modelling. It is, however, feasible to put a little more information into this term. System identification can be subdivided into structure determination and parameter estimation. While the parameter estimation is based on measured data, the structure can be determined from experiments as well as physical laws. Consequently system identification may also include elements of theoretical modelling. Typically, system identification is further characterized by:

1. many measurement data are obtained as sampled values of the inputs and outputs
2. a computer is used for processing the data
3. model parameters are estimated by minimization of an error criterion.

Models:
**mathematical** - other
**parametric** - nonparametric
**continuous-time - discrete-time**
**input/output** - state-space
**linear - nonlinear**
**dynamic** - static
**time-invariant** - time-varying
**SISO - MIMO**
Modelling / system identification:
**theoretical (physical) - experimental**
white-box - **grey-box** - black-box
**structure determination - parameter estimation**
**time-domain - frequency-domain**
**non-recursive** - recursive
**direct** - indirect

Table 4.1. Concepts and characterizations of models and modelling methods.

Now the choices of relevant concepts indicated in Table 4.1 shall be commented. As the topic of this lecture note is estimation of physical parameters, parametric models must be applied. Physical parameters belong in continuous-time models, but as discrete-time models are the foundation of system identification they shall be used as well. In general, physical insight can more easily be incorporated into state-space models. However, for the models considered here - of low order and with few inputs and outputs - a physical parametrization of input/output models is most straight-forward. Furthermore estimation of input/output models is simpler and the theories more developed. Observing the TSTF-principle (Try Simple Things First), linear models are employed first; later potential improvements from inclusion of static nonlinearities are investigated.

Estimating physical parameters the model structure must clearly be determined by theoretical modelling. The structure determination is particularly crucial in this case, as the accuracy of the estimated parameter values depends entirely on the correctness of the structure. A combination of theoretical and experimental modelling is denoted the grey-box approach, while black-box refers to purely experimental and white-box to purely theoretical modelling. Direct methods indicate that the continuous-time parameters are directly estimated, while indirect methods calculate the continuous-time parameters from estimated discrete-time parameters.

## 4.2 Physical parameters.

First it shall be emphasized that physical parameters are coefficients in the chosen model structure - they are not nature given. In fact, systems and processes do not have parameters, only models have. It is, however, customary to accept the illusion of a true system, containing the true parameters. This is really a model of the physical reality (the process), but to minimize the confusion, we shall denote it "the (true) system". The physical meaning as well as the value of the parameters is determined by the model structure.

Next the peculiarity of physical parameters shall be addressed. Physical parameters are model parameters with an apparent physical meaning or significance. Typically they are the coefficients in basic physical laws, e.g. Newtons, Hooks, Ohms, and Kirchoffs laws. Some corresponding physical parameters are

- mechanical parameters: mass, friction coefficients, stiffness
- electrical parameters: resistance, inductance, capacitance
- thermal parameters: thermal resistance, specific heat

The parameters: static gain, time constant, natural frequency and damping ratio are very characteristic for the properties of first and second order transfer function. They shall also be considered physical parameters. Even parameters describing physically given nonlinearities, e.g. the coefficients of low order polynomial approximations, will be considered physical parameters.

Obviously physical significance depends on the use of the model, and in particular on the user. Many electronic engineers, for example, find it easier to interpret a mechanical system in terms of its analogous electric network, even though the parameters lose their immediate physical meaning.

## 4.3 Simulation

Simulation is in short: *experimenting with a model instead of the real system.* The main benefit is, that it is much faster and less costly. The purpose of simulating may be:

- To obtain an understanding of the system
- Prediction of future system output
- Design and test of control systems
- Optimization of constructions
- Training of system operators on real time simulators
- Model parameter estimation from experiments

Early simulators were based on the principle of analogy, i.e. using another system with the same differential equations, or on scaling, i.e. using small scale models of e.g. ships. Today computer simulation is totally dominating. A computer simulation requires:

1. a discrete-time model
2. means for performing experiments on the model, e.g. giving specific inputs signals and parameter values
3. graphical tools for presenting the results.

## 4.4 Mathematical Models

Mathematical models of dynamic systems are descriptions of relations between system variables, normally in form of differential equations, see Fig. 4.1

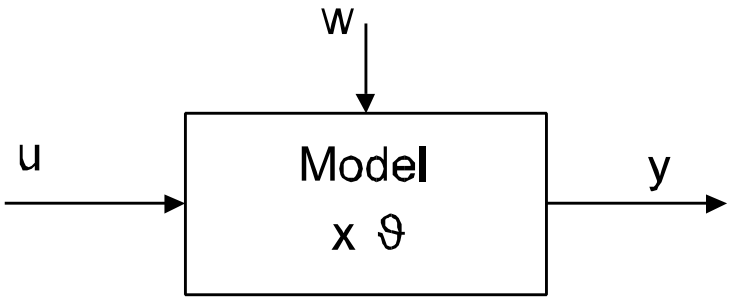


Figure 4.1 Block diagram of mathematical model with input, output and disturbance.

Model parameters $\theta$ are constants (or slowly varying quantities). Variables and signals are time-varying quantities in the model, e.g.

- output signal y: a variable of particular interest
- external signal: a variable affecting the system without being affected by other variables
- input signal u: an external signal for which we can freely choose the time evolution
- disturbance signal w: an external variable we cannot affect
- internal variable x: a variable that is not an output or an external signal

## 4.5 Model description

### 4.5.1 Block diagram notation

In block diagrams with linear and nonlinear blocks the variables between blocks must be time functions - not Laplace or Z-transforms:, see Fig. 4.2 and 4.3:

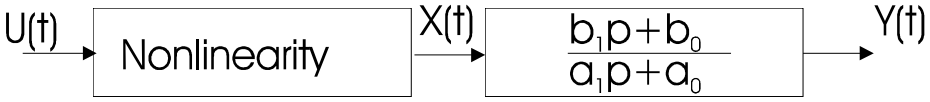


Figure 4.2 Nonlinear continuos-time model with time functions and p-operator in linear dynamic block.

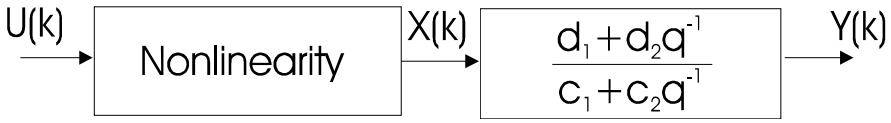


Figure 4.3 Nonlinear discrete-time model with time functions and q-operator in linear dynamic block

In this lecture note we use the following notation for continuous-time and discrete-time models:

Complex Laplace variable $s$ and differential operator $p$: $\dot{x}(t) = \frac{\partial x(t)}{\partial t} \triangleq px(t)$
Complex $Z$-transform variable $z$ and shift operator $q$: $x(k+1) = qx(k)$

When the effect of initial conditions are not considered, $s$ and $p$ are directly interchangeable and so are $z$ and $q$.

### 4.5.2 Linear continuos-time models

A linear model of a dynamic systems, i.e. an ordinary n'th order differential equation, can be a state space description or a transfer function.

The linear state space model is a set of n first order differential equations, or - in other words - a first-order matrix differential equation:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{4.1}$$

where x is the state vector of length n, and u is the input. The output y is a linear combination of the state and the input

$$y(t) = Cx(t) + Du(t) \tag{4.2}$$

A transfer function is obtained by the Laplace transformation

$$\frac{Y(s)}{U(s)} = G(s) = \frac{b_m s^m + .... + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + .... + a_1 s + a_0} \tag{4.3}$$

where the denominator polynomial is of order $n, n \geq m$.

In Matlab the symbols $ns$ and $ds$ are used for the polynomial coefficients:
$ns = [b_m ... b_1 b_0]$ and $ds = [a_n ... a_1 a_0]$.

### 4.5.3 Nonlinear continuos-time models

A nonlinear state space description is:

$$\dot{x} = f(x(t), u(t)) \tag{4.4}$$
$$y(t) = h(x(t), u(t)) \tag{4.5}$$

From the corresponding block diagram Fig. 4.4 it appears that the model consists of linear dynamic elements - integrators - and nonlinear static elements.

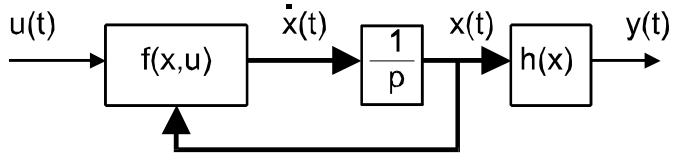


Figure 4.4 Block diagram of nonlinear state-space model. Input and output are scalars, the state variable is a vector (bold lines).

In many cases, it is convenient to use a more general block diagram model representation consisting of nonlinear static elements and linear higher order transfer functions. An example is seen in Fig. 4.5

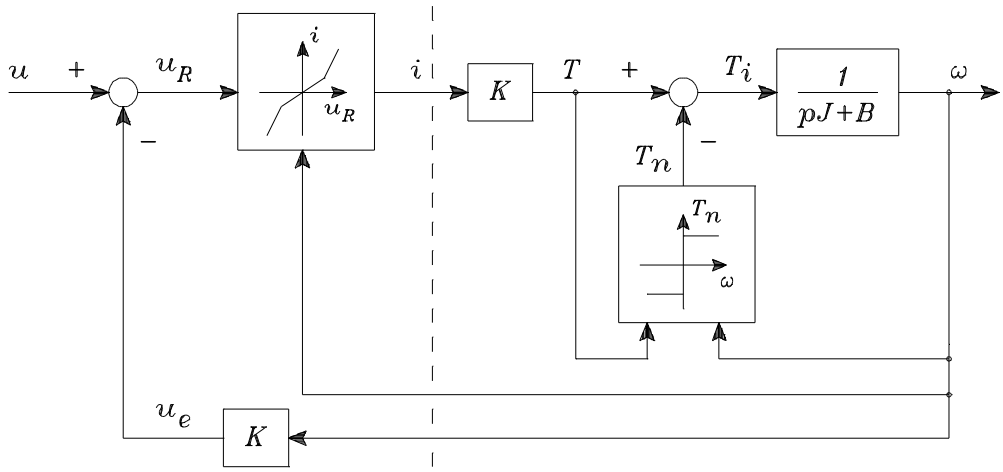


Figure 4.5 Blockdiagram of nonlinear system: DC-motor with nonlinear static elements and linear dynamic block

### 4.5.4 Linear discrete-time models

For simulation discrete-time equivalents of the continuous-time models, equations (4.1), (4.2) and (4.3) must be developed

$$x(k+1) = A_d x(k) + B_d u(k) \tag{4.6}$$
$$y(k) = C_d x(k) + D_d u(k) \tag{4.7}$$

and

$$\frac{Y(z)}{U(z)} = G_d(z) = \frac{b_{d0} + b_{d1} z^{-1} + ...b_{dn} z^{-n}}{1 + a_{d1} z^{-1} + ...a_{dn} z^{-n}} \tag{4.8}$$

How to transfer the continuous-time models to discrete-time models is the subject of section 4.6.

### 4.5.5 Nonlinear discrete-time models

Like nonlinear continuous-time models, nonlinear discrete-time models can be described in two ways. As a nonlinear state-space representation

$$x(k+1) = f_d(x(k), u(k)) \tag{4.9}$$
$$y(k) = h(x(k), u(k)) \tag{4.10}$$

or as a general block diagram model representation consisting of nonlinear static elements and linear higher order discrete-time transfer functions.

## 4.6 Discretization methods

Some of the most important methods for determination of discrete-time equivalents of linear continuous-time models are listed in Table 4.2 below, and characterized by name(s), 'transformation', and assumption/approximation.

A fixed time-step size, T is assumed.

| Name | Algorithm | Characteristics |
|---|---|---|
| Eulers forward rule | $s \rightarrow \frac{z-1}{T}$ | $\dot{x}(t)$ constant over the period $kT < t \leq (k+1)T$ |
| Tustin (Bilinear Transformation) | $s \rightarrow \frac{2}{T}\frac{z-1}{z+1}$ | $\dot{x}(t)$ varies linearly over the period |
| Step Invariant (ZOH equivalent) | $G_d(z) = (1-z^{-1})Z\{\frac{1}{s}G(s)\}$ | $u(t)$ constant over the period |
| Ramp Invariant (Tr H equivalent) | $G_d(z) = \frac{(1-z^{-1})^2}{z^{-1}T}Z\{\frac{1}{s^2}G(s)\}$ | $u(t)$ varies linearly over the period |
| Pole-zero mapping | $z_0 = e^{s_0 T}$ | |

Table 4.2: Discretization methods

**Eulers forward rule** is very simple to understand, as it approximates a differential quotient by a difference quotient:

$$\dot{x}(t)_{t=kT} \approx \frac{1}{T}(x(k+1) - x(k)) = \frac{1}{T}(q-1)x(k) \tag{4.11}$$

$$\dot{x}(t) = px(t) \implies p \approx \frac{1}{T}(q-1) \text{ or } s \approx \frac{1}{T}(z-1) \tag{4.12}$$

A stable pole in the s-plane $s_0$, $re(s_0) < 0$ is transferred to a pole in the z-plane $z_0$, $re(z_0) < 1$ i.e. not necessarily a stable pole (inside the unit circle). That means that numeric instability might occur, in particular for too large T-values.

Eulers rule is therefore mostly used, where a simple relationship between continuous-time coefficients and discrete-time coefficients is of importance.

**Tustins method** is also called the Trapeze rule, because an integral is approximated by the area of a trapeze:

$$\begin{aligned} x(k+1) &= x(k) + \int_{kT}^{(k+1)T} \dot{x}(t)dt \approx x(k) + \frac{T}{2}(\dot{x}(k) + \dot{x}(k+1)) \\ qx(k) &= x(k) + \frac{T}{2}(q+1)\dot{x}(k) \implies \dot{x}(k) = \frac{2}{T}\frac{q-1}{q+1}x(k) \\ p &= \frac{2}{T}\frac{q-1}{q+1} \quad \text{or} \quad s = \frac{2}{T}\frac{z-1}{z+1} \end{aligned}$$

The method can be recommended for most applications, in particular for frequency domain considerations.

**Pole-zero mapping**. It is an obvious idea to map s-domain poles according to $z = e^{sT}$ as this is the way the s-plane is mapped into the z-plane. The problem is, what to do with the zeroes, as there are normally more zeroes in the discrete-time transfer function than in the continous-time transfer function.

In invariance transformation, the poles are automatically mapped as $z = e^{sT}$.

### 4.6.1 Invariance transformations

The invariance transformations are particularly useful, so they shall be given a special treatment here.

The simulation problem is:

*Given an analog system $G(p)$. Determine the transfer function $G_d(q)$ of a digital system (the model) so the outputs are equal at the sampling times $t = kT$:*

$$y_d(k) = y(kT) \tag{4.13}$$

see Figure 4.6

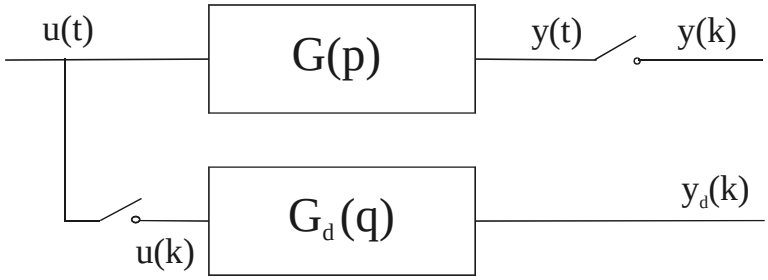


Figure 4.6 The simulation problem, $G_d(q)$ is a digital model of the system $G(p)$.

For input $u(t)$, the system and model outputs are respectively

$$Y(s) = G(s)U(s) \tag{4.14}$$
$$Y_d(z) = G_d(z)U(z) \tag{4.15}$$

The requirement that model output is equal to system output at sampling times 4.13 means that

$$Y_d(z) = Z\{Y(s)\} = Z\{G(s)U(s)\} \tag{4.16}$$

and

$$G_d(z) = \frac{Z\{G(s)U(s)\}}{U(z)} \tag{4.17}$$

In general $Z\{G(s)U(s)\} \neq Z\{G(s)\}U(z)$, and the result (4.17) consequently means, that the digital model depends on the input signal. This makes sense, as the system output depends on the input signal in between sampling times, while the model does not have this information. In other words, the requirement (4.13) can only be met, for a specific input.

We shall consider (4.17) for 3 input cases:

**Input impulse** $U(s) = 1$ From z-transform tables we have $U(z) = 1$

$$G_d(z) = Z\{G(s)\} \tag{4.18}$$

This is the impulse invariant transformation. It has only theoretical interest and should not be used for practical applications.

**Input step** $U(s) = \frac{1}{s}$ From z-transform tables we have $U(z) = \frac{1}{1-z^{-1}}$

$$G_d(z) = (1 - z^{-1})Z\{\frac{1}{s}G(s)\} \tag{4.19}$$

This is the step invariant transformation. It should be used whenever the input signal is a step or a series of steps, e.g. coming from a D/A converter with Zero Order Hold - hence the name ZOH-equivalent. For most other inputs it is quite useful as well.

**Input ramp** $U(s) = \frac{1}{s^2}$ From z-transform tables we have $U(z) = \frac{Tz^{-1}}{(1-z^{-1})^2}$

$$G_d(z) = \frac{(1 - z^{-1})^2}{z^{-1}T}Z\{\frac{1}{s^2}G(s)\} \tag{4.20}$$

This is the ramp invariant transformation or Triangular Hold equivalent - in Matlab it is denoted First Order Hold (FOH) equivalent. It is particularly useful for slowly varying input signals.

## 4.7 Simulating linear and nonlinear systems with Matlab

Matlab with relevant toolboxes (control toolbox in particular) has many efficient tools for simulation. Discretization can be accomplished with the Matlab function sysd=c2d(sysc,h,Method), where h is the time step. The default is 'zoh' (Zero Order Hold) when 'Method' is omitted.

Linear systems can even be simulated directly by the function y=lsim(sys,u,t), where sys is the continous-time system description.

This could lead to believe, that it is not necessary to be familiar with the subjects in the previous sections of this chapter, but that is absolutely not true. In real life applications, unexpected things will almost always happen, and to understand why and what to do, a fundamental knowledge of the principles behind the methods is essential.

In the following some methods for simulating linear and nonlinear systems will be given, and illustrated by simple examples. For use in Senstools the simulation program shall be a Matlab function of the form

y = simprocess(u,t,par)

where u is the input, t the time vector, par the model parameters, and y the output.

**Example 4** *Simulation of linear system*

For the simple first order system

$$\frac{Y(s)}{U(s)} = \frac{K}{1 + s\tau} \tag{4.21}$$

we shall now make a simulation program in three different ways:

**a) state space and for-loop:** A discrete time-state space description, equations (4.6) and (4.7), can directly be implemented as a loop. The discrete-time state space description is determined from the continuous-time state space description

$$\dot{x}(t) = -\frac{1}{\tau}x(t) + \frac{K}{\tau}u(t) \tag{4.22}$$

$$y(t) = x(t) \tag{4.23}$$

is discretisized using c2d

```
function y=simktauloop(u,t,par)
% y=simktauloop(u,t,par) simulates K/(1+s*tau)
% using ss (xdot=-1/tau*x+k/tau*u, y=x) c2d and a for loop.
% par=[K tau]
%
% 22/10-02,MK
h=t(2)-t(1);
nn=length(u);
x=0;
K=par(1); tau=par(2);
sysc=ss(-1/tau,K/tau,1,0); sysd=c2d(sysc,h); [ad,bd,cd,dd] = ssdata(sysd);
for jj=1:nn
 x1=ad*x + bd*u(jj);
 y(jj)=cd*x1;
 x=x1;
end
```

The program can be tested by:

```
>> par=[2 3];
>> h=par(2)/40;
>> t=[1:100]*h;
>> u=[0 ones(1,99)];   % input step
>> y=simktauloop(u,t,par);
>> plot(t,y)
```

**b) Using 'filter':**

The Matlab function: y=filter(nz,dz,u) calculates the output based on the discrete-time transfer function nz/dz. So the core of the simulation program is:

```
sysctf=tf(par(1),[par(2) 1]);
sysdtf=c2d(sysctf,h);
[nz,dz]=tfdata(sysdtf,'v');     % NB: 'v' for vector format - not cell
y=filter(nz,dz,u);
```

The program is **function y=simktaufilt(u,t,par)** and it can be tested as in case a).

**c) Using 'lsim':** The Matlab function; y=lsim(sysc,u,t) calculates the output based on the continuous-time system description, e.g. the continuous-time transfer function nc/dc. So the core of the simulation program is:

```
nc=K;  dc=[tau 1];
t=[0 t(1:length(t)-1)];    % lsim requires that t starts with 0
y=lsim(nc,dc,u,t);
```

The program is **function y=simktau(u,t,par)** and it can be tested as in case a).

Of the three methods a) is slow (loops in Matlab shall be avoided if possible), and should be used only for nonlinear systems. b) and c) are similar in performance, but c) is the easiest and chooses the discretization method for you.

**Example 5** *Simulation of nonlinear systems*

A nonlinear system can be described as a linear dynamic part and a nonlinear static part. The linear dynamic part can be simulated by discretization, as shown in Example 4, but it is normally necessary to use a loop construction to calculate the nonlinear function for each time step.

We shall now make a simulation program for the system in Fig.4.7

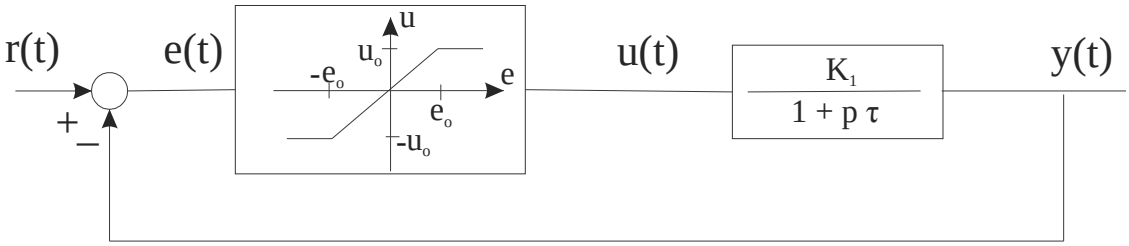


Figure 4.7 System with saturation

The saturation can be described by:

$$e < -e_0 : \quad u = -u_0 \tag{4.24}$$

$$-e_0 \leq e \leq e_0 : \quad u = ke \tag{4.25}$$

$$e_0 < e : \quad u = u_0 \tag{4.26}$$

where $k = u_0/e_0$

This can be implemented in Matlab as

```
v=k*e;
if e>e0,  u=u0;  end
if e<e0,  u=-u0; end
```

Using logical operations where a false statement has the value 0, it can be expressed more efficient as:

```
v=k*e-(abs(e)>e0)*k*(e-sign(e)*e0);
```

as $v_0 = ke_0$ can be written as $v_0 = ke - k(e - e_0)$ and $-v_0 = ke - k(e + e_0)$.

The simulation program can then written as

```
function y=simsatktau(u,t,par)
% y=simsatktau(u,t,par) simulates a simple nonlinear example:
% first order linear transfer funktion with saturation.
% par=[e0 v0 tau]
```

```
% Example: par=[.5 1 5] (inp: f1=.03).
%
% 29/10-02,MK
h=t(2)-t(1);
nn=length(u);
e0=par(1), v0=par(2), tau=par(3); k=v0/e0
sysc=ss(-1/tau,5/tau,1,0);
sysd=c2d(sysc,h);
[ad,bd,cd,dd] = ssdata(sysd);
x=0; y=[0 0];
for jj=1:nn-1
 e=u(jj)-y(jj);
v=k*e-(abs(e)>e0)*k*(e-sign(e)*e0);
 x1=ad*x + bd*v;
 y(jj+1,:)=[cd*x1 v];
 x=x1;
end
```

■

### 4.7.1 Modelling and simulation of loudspeaker

In the following a major example will be studied. It includes physical modelling and simulation of an electrodynamic loudspeaker. Linear as well as nonlinear models will be considered. The example is very useful for solving Problem 4.4, simulation of DC-motor.

The study is split up into 3 examples:

Example 6: Physical modelling of loudspeaker

Example 7: Simulation of linear loudspeaker model

Example 8: Simulation of nonlinear loudspeaker model

**Example 6** *Physical modelling of loudspeaker*

A sectional view of an electrodynamic loudspeaker is shown in Fig. 4.7

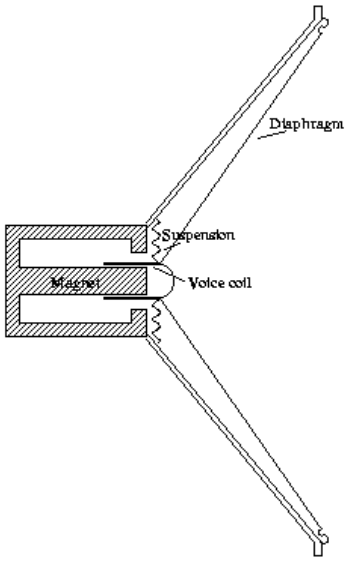


Figure 4.7 Loudspeaker with magnet, suspension, diaphragm and voice coil.

The symbols used:

| | | |
|---|---|---|
| u | applied voltage | [V] |
| i | current in voice coil | [A] |
| x | displacement of voice coil | [m] |
| R | resistance of voice coil | [Ohm] |
| Bl | force factor | [N/A] |
| m | mass of moving system | [kg] |
| r | viscous friction coefficient | [Ns/m] |
| k | suspension stiffness | [N/m] |

The linear dynamic properties are described by two differential equations, one for the electrical and one for the mechanical part.

$$u(t) = Ri(t) + Bl\frac{dx(t)}{dt} \tag{4.27}$$

$$Bl \cdot i(t) = m\frac{d^2x(t)}{dt^2} + r\frac{dx(t)}{dt} + kx(t) \tag{4.28}$$

and the corresponding block diagram is derived by Laplace transformation

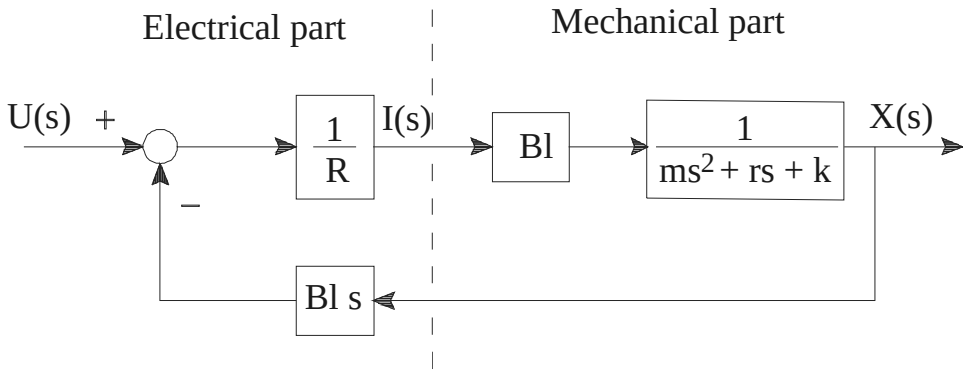


Figure 4.8 Block diagram of linear loudspeaker model

As we shall see later, all 5 parameters cannot be estimated from measurements of input u and output x alone, it is necessary also to measure the current i. Consequently we have a system with one input and two outputs. The corresponding transfer functions are derived from the bloc diagram

$$\frac{I/s)}{U(s)} = \frac{(ms^2+rs+k)/R}{ms^2+(\frac{Bl^2}{R}+r)+k} \tag{4.29}$$

$$\frac{X/s)}{U(s)} = \frac{Bl/R}{ms^2+(\frac{Bl^2}{R}+r)+k} \tag{4.30}$$

We shall also need the transfer function for a current driven loudspeaker

$$\frac{X/s)}{I(s)} = \frac{Bl}{ms^2+r+k} \tag{4.31}$$

A state space description can be developed by inverse Laplace transformation of (4.30) as we choose the state vector and output vector to

$$x_v = \left\{ \begin{array}{c} x \\ \dot{x} \end{array} \right\} \qquad y = \left\{ \begin{array}{c} x \\ i \end{array} \right\} \tag{4.32}$$

State space description of linear loudspeaker model

$$\dot{x}_v = \left\{ \begin{array}{cc} 0 & 1 \\ -\frac{k}{m} & -\frac{r}{m}-\frac{Bl^2}{mR} \end{array} \right\} x_v + \left\{ \begin{array}{c} 0 \\ \frac{Bl}{mR} \end{array} \right\} u \tag{4.33}$$

$$y = \left\{ \begin{array}{cc} 1 & 0 \\ 0 & -\frac{Bl}{R} \end{array} \right\} x_v + \left\{ \begin{array}{c} 0 \\ \frac{1}{R} \end{array} \right\} u \tag{4.34}$$

A state space description of the current driven loudspeaker can be developed by inverse Laplace transformation of (4.31)

$$\dot{x}_v = \left\{ \begin{array}{cc} 0 & 1 \\ -\frac{k}{m} & -\frac{r}{m} \end{array} \right\} x_v + \left\{ \begin{array}{c} 0 \\ Bl/m \end{array} \right\} i \tag{4.35}$$

$$y = x = \left\{ \begin{array}{cc} 1 & 0 \end{array} \right\} x_v \tag{4.36}$$

A major *nonlinearity* is the displacement dependency of the force factor, which can be modelled as

$$Bl(x) = Bl \cdot e^{-c_1(x+c_2)^2} \tag{4.37}$$

The corresponding nonlinear block diagram is

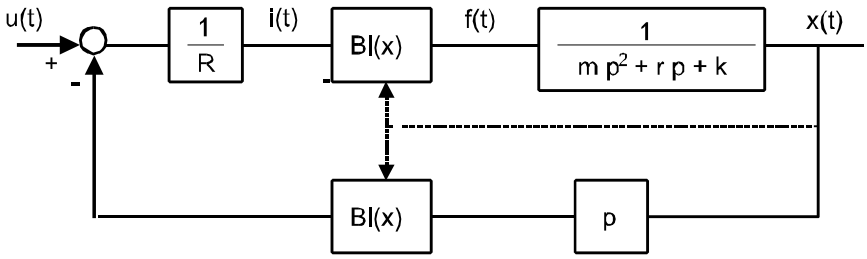


Figure 4.9 Block diagram of nonlinear loudspeaker model with displacement dependent force factor Bl(x)

The parameter values to be used in the simulations are [m, r, k, R, Bl,$c_1$,$c_2$] = [$12.5{\cdot}10^{-3}$, 5.2, $12{\cdot}10^{3}$, 5, 10, $10^{6}$, $2{\cdot}10^{-4}$].

**Example 7** *Simulation of linear loudspeaker*

The linear voltage driven loudspeaker model is simulated, using transfer functions and the Matlab function 'filter'. The simulation program is in the Matlab function form
y = simprogram(u,t,par) for use in Senstools.

The program is using transfer functions (4.29) and (4.30) , and 'filter'.:

```
function y=simsplf(u,t,par)
% y=simsplf(i,t,par) simulates a linear loudspeaker
% voltage input and outputs x and i
% Uses closed loop transfer functions and 'filter'
% x/u = Bl/R/(m*s^2+(Bl^2/R+r)*s+k)
% I/u = (m*s^2+r*s+k)/R/(m*s^2+(Bl^2/R+r)*s+k)
% par=[m r k R Bl], (length 5)
% 14/1-03,MK
h=t(2)-t(1); ns=length(u);
m=par(1); r=par(2); k=par(3); R=par(4); Bl=par(5);
% Closed-loop transfer functions from u to x and i:
nx = Bl/R; dx = [m (Bl^2/R+r) k];
ni = [m r k]/R; di = dx;
% Discretization:
sysxc=tf(nx,dx); sysxd=c2d(sysxc,h); [nxd,dxd] = tfdata(sysxd);
sysic=tf(ni,di); sysid=c2d(sysic,h); [nid,did] = tfdata(sysid);
% Output calculation:
x = filter(nxd{1},dxd{1},u);  % Notice {} to obtain cell content
i = filter(nid{1},did{1},u);
y = [x(:), i(:)];
```

The program can be tested with:

```
% testsimsplf: Data for test of simsplf, example ?
% Stepresponse
% 14/1-03,MK
par = [.0125 5.2 12000 5 10];
h = 2e-5;
n=500;
am = 4;
[u,t] = instepa(1,am,n,h);
y = simsplf(u,t,par);
plot(t,1000*y(:,1),t,y(:,2))
title('Loudspeaker displacement and current')
```

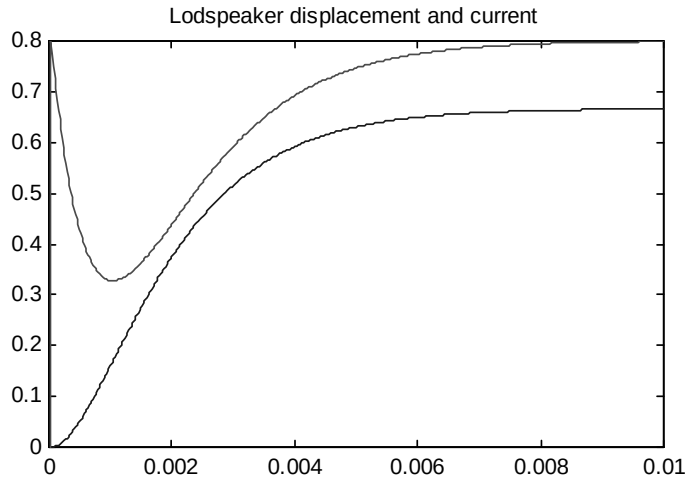


Figure 4.10 Loudspeaker step response - displacement and current.

The result in Fig. 4.10 appear to have the correct stationary values:
$i_\infty = u/R = 4/5 = 0.8$ and $x_\infty = i_\infty Bl/k = 0.8 \cdot 10/12000 = 0.67 \cdot 10^{-3}$
■

**Example 8** *Simulation of nonlinear loudspeaker model*

The program uses a state space description of the linear dynamics from the force f=Bl·i to the displacement x. The effect of the displacement dependent force factor (4.37) is implemented in a for-loop.

```
function y=simspn(u,t,par)
% y=simspn(i,t,par) simulates a nonlinear loudspeaker
% with voltage input and outputs x and i
% Displacement dependent force factor Bl(x)=Bl*e^(-c1(x+c2)^2)
% Uses state space description of mechanical part from f=Bl*i to x
```

```
% par=[m r k R Bl c1 c2], (length 7)
% 14/1-03,MK
h=t(2)-t(1); nn=length(u);
m=par(1); r=par(2); k=par(3); R=par(4); Bl=par(5); c1=par(6); c2=par(7);
% State space description from f to x:
ac=[0 1;-k/m -r/m]; bc=[0;1/m];
cc=[1 0]; dc=0;
syscss=ss(ac,bc,cc,dc);
% Descretization:
sysdss=c2d(syscss,h);
[ad,bd,cd,dd] = ssdata(sysdss);
xv=[0 0]';
for jj=1:nn
 Blx=Bl*exp(-c1*(xv(1)+c2)^2);
 i=(u(jj)-Blx*xv(2))/R;
 f=Blx*i;
 xv1=ad*xv+bd*f;   % input ss equation
 x = cd*xv;      % output ss equation
 yv(jj,:)=[x i]; % yv=[x i]
 xv=xv1;    % update
end
y=yv;
```

The function is first tested with c1 = 0, i.e. constant Bl, and a step input (test program testsimspn.m). The result is as expected similar to Fig. 4.10.

Next $[c_1\ c_2] = [10^6,\ 2{\cdot}10^{-4}]$ is applied, giving a Bl(x) as shown in Fig. 4.11.

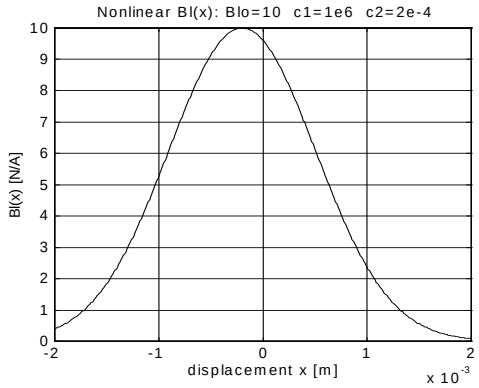


Figure 4.11 Displacement dependent force factor Bl(x)

To emphasize the effect of the nonlinearity, a staircase input voltage is applied, and the test program is:

```
% testsimspnst: Data for test of simspn, example 8
% Input staircase
% 14/1-03,MK
par = [.0125 5.2 12000 5 10 1e6 2e-4];
h = 2e-5;
n=2500;
```

```
am = 8;
ns = 4;
[u,t] = inpstair(ns,am,n,h);
y = simspn(u,t,par);
plot(t,1000*y(:,1),t,y(:,2))
title('Nonlinear loudspeaker with input staircase')
grid
```

Fig. 4.12 shows clearly how the damping decreases with increasing amplitude, because of decreasing feed-back via Bl.

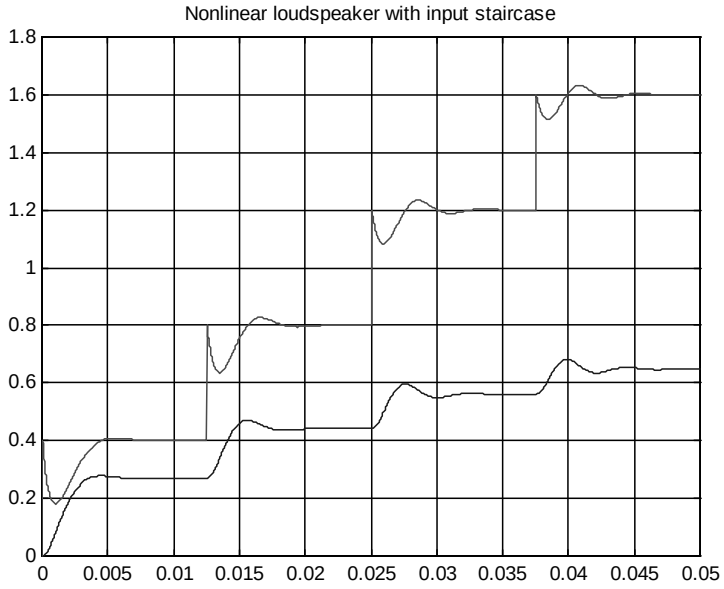


Figure 4.12 Loudspeaker with displacement dependent Bl(x) - displacement and current with staircase input

■

Other methods could have been used for simulating the loudspeaker, for example could 'lsim' have been used for the linear model as in Example 4c, but the methods used are normally efficient for most linear and nonlinear models.

## 4.8 Resumé

Some fundamental concepts were defined and discussed, e.g. that experimental modelling consists of model structure determination and parameter estimation. Various mathematical models in state-space and block diagram form was described, and the most important discretization methods explained.

The last part of the chapter consists of examples of Matlab simulation programs for linear and nonlinear models - in the function form y = simprocess(u,t,par) required by Senstools. A major

example, modelling and simulation of loudspeaker may be helpful for solving the problem 4.4, modelling and simulation of DC-motor.

# Chapter 5

# SENSTOOLS for parameter estimation

*Senstools is a collection of Matlab programs, implementing the sensitivity approach for direct parameter estimation, experiment design and model validation. The programs may conveniently be organized as a Matlab Toolbox.*

In this chapter parameter estimation with Senstools is considered, corresponding to the methods addressed in the previous chapters. In later chapters more aspects of the sensitivity approach, such as experiment design and model validation, are addressed, and corresponding Senstools implementations are described in Appendix A.

As with all Matlab code, it is fairly easy for an experienced Matlab user to understand the detailed function of the programs, and if other functions or information is required, it is easy to make minor alterations (Open Source).

The programs in Senstools are organized as main programs (script files) calling sub programs (functions) and using data (mat-files).

## 5.1 Name conventions

The program and data file names contain information of the program type and of the actual process name.

The initial letters designate the type:

**main** main program (script file)

**sim** simulation of process (function file)

**meas** input/output measurement data (mat-file)

**prog** program data. **progdata** (mat-file) are created by **progprog** (program data program, script file). Minimum program data are process name and initial parameter values.

Names of files being particular for a certain process will contain the process name. For example, for a process with the name *'motor'* the name of the simulation program is *simmotor.m*, and the measured data are stored in *measmotor.mat*

The program names also contain information of the function they perform. For example the main program for parameter estimation is called *mainest.m*.

## 5.2 What to do - parameter estimation

The user of Senstools must do 4 things for parameter estimation of a process named xxx:

1. **Make the simulation program as a Matlab function**:
   y = simxxx(u,t,par) as described in Chapter 4
2. **Save the measured data t, u and y** (column vectors of equal length for SISO (single input single output) systems; for MIMO systems see next section) in a mat-file named measxxx.mat. This is done by the instructions:
   >> u=u(:); y=y(:); % to make sure they are column vectors (SISO only!)
   >> save measxxx t u y % creates measxxx.mat
3. **Enter required program data**. This can be done in 3 different ways:
   a) Entering the values in the work space one by one. Minimum program data are process name and initial parameter values, e.g.:
   >> process='xxx';
   >> par0=[1 2];
   This may be convenient for smaller problems with few parameters
   b) Loading a mat-file (progdataxxx.mat) with the required program data values. This is done automatically if a progdata-file for the particular process exists, and the work space is cleared. The progdata-file is created and saved by a progprogxxx.m file. It is recommended that a progprog file is made for more extensive problems, and that data changes are performed in these m-files.
   c) If the progdata are not entered in the work space and there is no progdata.mat file, default values specified in the main programs are used. This is mostly for running demo examples - and to ensure that something other than an error message is the result of running mainest.m
4. **Run the main program mainest.m** for parameter estimation
   >> mainest

The procedure above is further illustrated by studying a listing of mainest:

```
% mainest is the main program for parameter estimation
% From input/output data (u, y and t) stored in meas'process','no'.mat,
% the parameters pare of 'process' are estimated.
% The sensitivity measures and the parameter deviations are calculated as well.
%
% Must be specified (e.g. by progprog'process'.m, i.e. from
% progdata'process'.mat): see default values below.
% Subprograms used:
% sim'process', gausnewt, psinf, and sens.
%
% 26/11-02,MK
% Default values:
if ~exist('process'), process='ktau'; end              % Process name
if ~exist('no'), no=''; end                            % Measurement number
if exist(['progdata',process,no,'.mat'])==2 & ~exist('par0')
```

```
  load(['progdata',process,no]), end                    % progdata loaded
if exist(['meas',process,no,'.mat'])==2, load(['meas',process,no]),
else
 disp(['data: meas',process,no,'.mat missing !']), break,
end                                          % measdata loaded or error message
if ~exist('ploty'), ploty=2;  end
if ~exist('par0'), par0=[1.5 3]; end
simmod=['sim',process];                        % Name of simulation program
if ploty>0, plot(t,y), ylabel('y'), xlabel('time'),pause(4), end
[pare,errn,Hrn]=gausnewt(simmod,par0,u,y,t);            % Parameter estimation
% (Some instructions for model verification are left out here)
pare              % The parameter estimates are written in the command window
errn              % The relative model error in per cent is written
```

Comments to mainest:

- the 'if ~exist' construction is entering a default value if a variable value does not exist in work space. If a progdata file exist for the particular process, program data from this will be used. If not, default values of par0 will be used.

- 'no' gives a possibility for numbering data sets, and thus for using several measurement data sets for the same process

- if no measxxx.mat file exists, an error message is given

- the parameter estimation is performed by the gausnewt function

**Example 9** *Parameter estimation with mainest*

The parameter estimation procedure with Senstools will be illustrated, using all three methods for entering program data - 3.a, 3.b and 3.c - described above. The required measurement-data (actually obtained by simulation, adding noise to the output) are already stored together with Senstools

It is highly recommended to execute the example in Matlab !

**Re 3a)** Entering program data in work space manually.

The process named *kutau* is a first order nonlinear system:

$$G_m(s) = \frac{K(u)}{1 + s\tau} \quad where \quad K(u) = k_0(1 + k_1/(0.5 + u^2)) \tag{5.1}$$

The simulation program simkutau.m and the measurement data measkutau.mat are in Senstools. The process name and the start values for the parameters par=[ko k1 tau] are entered in the command window, and mainest is called:

» process='kutau';

» par0=[1 2 3];

» mainest

In the Matlab figure window we see measured system and model output, and how the model output approaches the system output. This is a very ardent illustration of the iteration. Fig. 5.2 shows the situation after the first approximation. Notice the relative model error, ernn; its precise definition is given in the next chapter.

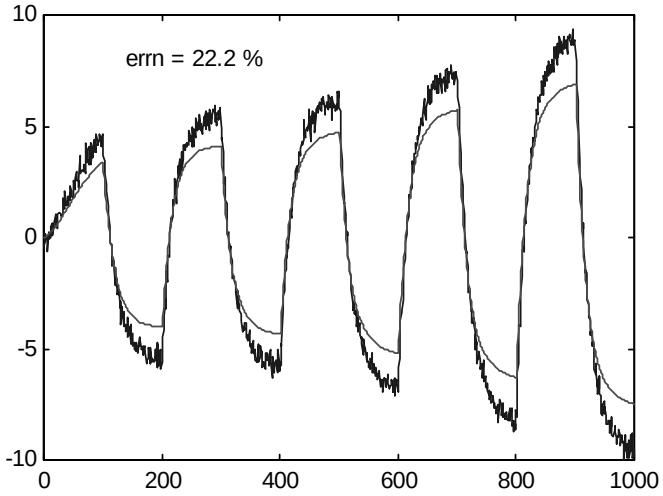


Figure 5.2 Measured system output (noisy) and model output for process kutau

After only tree iterations, we are close to the minimum. The parameter estimates and other relevant information is displayed in the command window.

**Re 3b)** Using a progprog program and a progdata file.

For more complicated problems with a larger number of parameters, it is recommended to have a more permanent set of program data. The simple process kutau will however be used again, just to illustrate the method.

First make and run the progprog program:

```
% progprogkutau.m creates program data for mainest with process kutau
%
% 27/11-02,MK
clear
process='kutau';
par0=[.85 1.8 2.24];
save progdatakutau process par0     % creates progdatakutau.mat
```

Now the program data are permanently stored in progdatakutau.mat. You only have to specify the process name, and mainest will load the remaining program data from progdatakutau.mat:

» process='kutau';

» mainest

and the parameter estimation runs as in case 3.a above.

**Re 3.c.** Using default data (demo example)

If we clear the workspace, the default values in mainest will be used, i.e. the process name is *ktau*, a simple linear first order system with parameters $K$ and $\tau$. Enter in Matlab command widow:

>> clear

>> mainest

and the demo example starts. In Fig, 5.3 we see the situation after the first iteration.

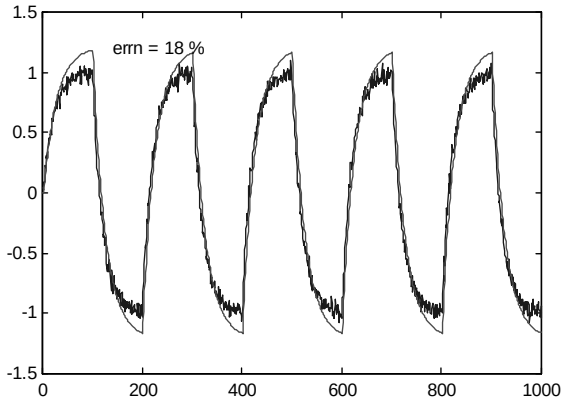


Figure 5.3 Measured system output ( noisy) and model output for process ktau

In the command window the model error errn and the relative parameter update is displayed for each iteration. Already after 3 iterations the minimum value of errn (with 5 digits) is reached, and the relative parameter update is below 0.003%. The parameter estimates are given as

pare =
1.0007
2.0020

Because the output data are obtained by simulation a correct set of model parameter values exists (this obviously not the case for real life systems). They are k=1 and tau=2. The small deviation in the estimates are due to noise.

■

**Multivariable systems** So far we have considered SISO (single input single output) systems only, but MIMO (multiple input multiple output systems can be handled as well. The signals u and y must then be matrices, with the individual signals as columns.

**Example 10** *DC-motor with two outputs*

Consider a SIMO system, a DC-motor with voltage u as input and current i and velocity w as output. The two-variable output is then: y = [i w], where i and w are the sampled values, organized in column vectors.

We have the simulation program simdcml, the input/output data in measdcml.mat and the program data in progdatadcml.mat.

Lets us first take a look at the data in measdcml using the command *whos*:

```
>> load measdcml
>> whos
Name Size Bytes Class
process 1x4 8 char array
t 1x1000 8000 double array
u 1000x1 8000 double array
y 1000x2 16000 double array
```

We note, that they have the proper format. Then the estimation is performed:

» mainest

Fig. 5.4 shows the measured output

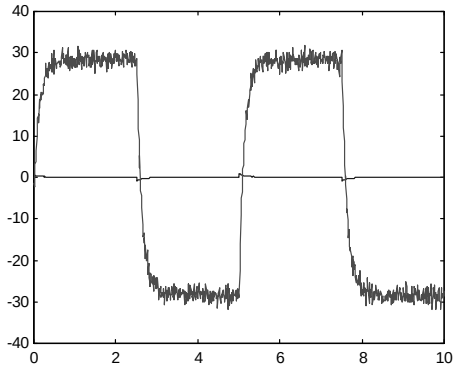


Figure 5.4 Measured output from dcml, current i and velocity w (large noisy).

For Multiple outputs systems, the normed measured and model output can be shown during the iteration in two different ways. For the prog data parameter, ploty =1, the outputs are plotted together, see Fig. 5.5.

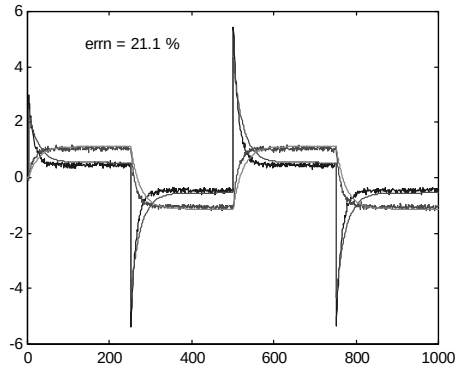


Figure 5.5 Normed measured and model output for ploty = 1

For ploty = 2 the two outputs are displayed in extension, see Fig. 5.6. This gives a better view of the fit, and ploty = 2 is consequently used as default value.

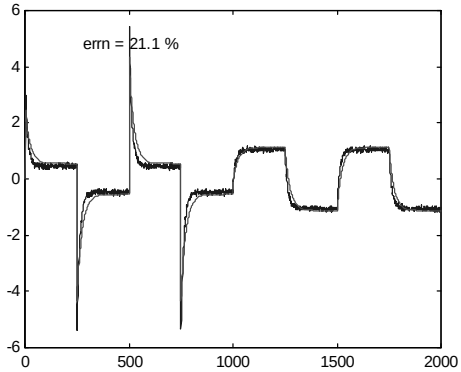


Figure 5.6 Normed measured and model output for ploty = 2

## 5.3 Résumé

An introduction to Senstools was given, describing how parameter estimation is done.

A set of measurement input/output data from the process (xxx) must be available in advance, and then the user must:

- Make the simulation program as a Matlab function y = simxxx(u,t,par)
- Save the measured data t, u and y in a mat-file named measxxx.mat.
- Enter required program data e.g. process name, xxx, and initial parameter values, par0, or make a program data program progprogxxx.m to create program data file progdataxxx.mat
- Run the main program mainest.m for parameter estimation

A number of simple examples illustrated the procedure and comments on the results in the command window and the graphic window were given.

A complete description of Senstools, including other aspects than parameter estimation is given in Appendix A.

We are now able to estimate model parameter values from measured input/output data, but if the model shall be of use we must know something about the accuracy of the obtained model. This is the subject of the next chapter, where we are back to theory.

# Chapter 6

# Parameter accuracy and sensitivity

*A good fit between the model output and the measured system output is an indication of an accurate model. But it is a necessary not a sufficient condition for accurate parameter estimates. Consequently sensitivity considerations are introduced for evaluation of parameter accuracy.*

Model verification - i.e. a reliable measure of the accuracy of the obtained model - is of utmost importance. It is generally recognized that successful application of system identification, and model verification in particular, requires a thorough understanding of the theory, combined with physical insight, intuition and common sense.

In this chapter we shall introduce methods and measures suitable for evaluating model accuracy and for incorporating physical insight, intuition and common sense. A small thing with a large effect is to deal with relative parameter values and normed signal values, as it is difficult to compare quantities with different units and of different order of magnitudes.

Parameter sensitivities and so called characteristic sensitivity measures are introduced, and a graphical interpretation is given. These sensitivity measures are very appropriate for model verification. If they indicate, that an accurate model cannot be expected, they also tell what might be wrong, for example an inadequate input signal or model structure. Besides they can be used for determination of the accuracy of the parameter estimates.

## 6.1 Evaluation of the model fit

In Chapter 3 we defined the parameter estimate

$$\theta_N = \arg\min_{\theta} P(u_N, y_N, \theta) \tag{6.1}$$

i.e. the value minimizing the performance function

$$P(\theta) = \frac{1}{2N} \sum_{k=1}^{N} \varepsilon^2(k, \theta) \tag{6.2}$$

For evaluating how well the model fits the measurements the minium value $P(\theta_N)$ is, however not very informative. Instead we introduce the *normed root mean square output error*, errn

$$errn = \sqrt{\frac{\sum_{k=1}^{N}(y(k) - y_m(k,\theta_N))^2}{\sum_{k=1}^{N} y(k)^2}} \cdot 100 \quad [\%] \tag{6.3}$$

For a single number, errn is very expressive about the fit. It is, in particular, suited for comparing different model structures. Experience show, that for well modelled electromechanical systems with low-noise measurements, errn-values of 5-8 % should be obtainable, while for more complex systems, for example physiological, one can do no better than 20-25 %. Obviously, errn has its minimum for the same $\theta-$value as $P$.

The model fit is, however, best evaluated by plotting the model output together with the measured system output. This plot gives excellent indication of where the model may be deficient, and very often ideas how to make improvements in the model structure.

A good fit, i.e. a small value of errn, only indicates that the model structure is adequate for expressing the system behavior for a particular input signal. Whether a good fit also implies accurate parameter estimates is the subject of the next section.

## 6.2 Parameter sensitivity

It is intuitively comprehensible that an accurate estimate of a parameter $\theta_i$ requires $P(\theta)$ to be sensitive to $\theta_i$, and that - roughly - the most sensitive parameters will be estimated most accurately. Because of the quadratic nature of $P(\theta)$, it is unfit for sensitivity calculations - as we shall see. In stead the root mean square of a so called parameter dependent error is introduced.

The model error, $\varepsilon(k)$ may be split up into two parts: the minimum value $\varepsilon_0 = \varepsilon(k,\theta_N)$ caused by noise and imperfect model structure, and a part, $\varepsilon_p(k,\theta)$ caused by $\theta$ being different from the optimal estimate $\theta_N$. The latter part, $\varepsilon_p$, we shall denote *the parameter dependent model error.*

By a first order Taylor expansion around $\theta_N$ we obtain

$$\varepsilon(k,\theta) = \varepsilon_0(k) + \varepsilon_p(k.\theta) \tag{6.4}$$

$$\varepsilon_p(k.\theta) = y_m(k,\theta) - y_m(k,\theta_N) \approx \psi(k,\theta_N)^T(\theta - \theta_N) \tag{6.5}$$

where $\psi$ is the model gradient

$$\psi(k,\theta) = \frac{\partial y_m(k,\theta)}{\partial \theta} \tag{6.6}$$

The root mean square of $\varepsilon_p(k,\theta)$ is next introduced

$$\varepsilon_{p,RMS}(\theta) = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\varepsilon_p^2(k,\theta)} \approx \sqrt{(\theta-\theta_N)^T\widetilde{H}(\theta_N)(\theta-\theta_N)} \tag{6.7}$$

where we have used equation (6.5) and $\widetilde{H}(\theta) = \frac{1}{N}\sum_{k=1}^{N}\psi(k,\theta)\psi^T(k,\theta)$ from chapter 3.

As the values of the individual parameters may be very different, a comparison of absolute sensitivities has little implication. Relative parameter sensitivities are more meaningful, and they can be obtained by introducing *relative parameters* (with index r), $\theta_r$

$$\theta_r = L^{-1}\theta \qquad L = diag(\theta_N) \tag{6.8}$$

where $diag(\theta_N)$ is a square matrix with $\theta_N$ as the diagonal and zeroes outside, and the corresponding relative model gradient $\psi_r$ and relative Hessian $H_r$

$$\psi_r(k,\theta) = L\psi(k,\theta) \quad \text{and} \quad H_r = LHL \tag{6.9}$$

To make the values of the individual sensitivities meaningful and to be able to handle multiple output systems, also *normed signals* (with index n) are introduced, e.g.

$$y_{mn}(k,\theta) = y_{RMS}^{-1}\ y_m(k,\theta) \tag{6.10}$$

and

$$\varepsilon_{p,RMSn} = y_{RMS}^{-1}\ \varepsilon_{p,RMS} \tag{6.11}$$

where

$$y_{RMS} = \sqrt{\frac{1}{N}\sum_{k=1}^{N} y^2(k)} \tag{6.12}$$

Note that $\varepsilon_{RMSn} = errn/100$ and $\varepsilon_{0,RMSn} = \min(errn/100)$.
The relative normed model gradient is then

$$\psi_{rn}(k,\theta) = y_{RMS}^{-1} L\psi(k,\theta) \tag{6.13}$$

and the relative normed Hessian consequently (3.31)

$$H_{rn} = y_{RMS}^{-1} LHL \tag{6.14}$$

The normed root mean square parameter dependent error can then be expressed by relative normed quantities (from equation 6.7)

$$\varepsilon_{p,RMSn}(\theta_r) = \sqrt{\frac{1}{N}\sum_{k=1}^{N} \varepsilon_{pn}^2(k,\theta_r)} \approx \sqrt{(\theta_r - 1_v)^T \widetilde{H}_{rn}(\theta_{Nr})(\theta_r - 1_v)} \tag{6.15}$$

as $\widetilde{\theta}_r = \theta_r - 1_v$ where $1_v$ is a $d \times 1$ vector $1_v = \{1\ 1\ 1...1\}^T$.

We shall now introduce the *sensitivities* of $\varepsilon_{p,RMSn}$ with respect to the parameters, as the derivatives.

The sensitivity with respect to one relative parameter $\theta_{ri}$ - while the remaining relative parameters are equal to 1 - is then

$$S_i = \frac{\partial \varepsilon_{p,RMSn}}{\partial \theta_{ri}} = \sqrt{h_{rnii}} \quad \text{where} \quad h_{rnii} = \{\widetilde{H}_{rn}(\theta_{rN}\}_{ii} \tag{6.16}$$

that is, the sensitivities are simply the square root of the diagonal elements of the relative, normed Hessian.

To interpret this sensitivity in the one-dimensional case, see Fig. 6.1

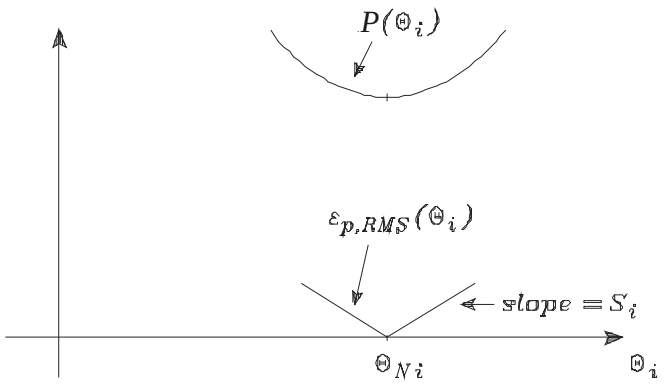


Figure 6.1 Performance function $P(\theta_i)$ and RMS parameter dependent error $\varepsilon_{p,RMS}(\theta_i)$ as a function of parameter $\theta_i$

To obtain accurate parameter estimates, large sensitivities of the individual parameters is a necessary requirement. It is not sufficient, however, as the sensitivity of a combination of two or more parameters may be much smaller than the individual sensitivities, indicating a high correlation between the parameters. This problem can be illustrated in the two-dimensional parameter space, where iso-curves expressed by $\varepsilon_{p,RMSn}(\theta_r) = constant$, are ellipses cf. equation (6.15).

For the special case $\varepsilon_{p,RMSn}(\theta_r) = 1$ we will use the name, sensitivity ellipse, as the distance from the centre to any point on the ellipse represent the reciprocal sensitivity in that direction - see Figure 6.2 and example 11.

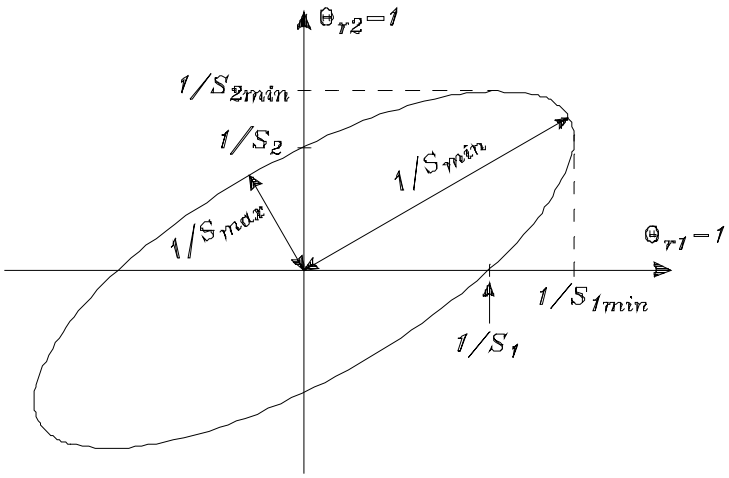


Fig. 6.2 Sensitivity ellipse: $\varepsilon_{p,RMSn}(\theta_r) \approx \sqrt{(\theta r - 1_v)^T \widetilde{H}_{rn}(\theta_{Nr})(\theta_r - 1_v)} = 1$ with some characteristic sensitivities.

Various characteristic sensitivities are indicated on Fig.6.2:

| | |
|---|---|
| $S_i$ | sensitivity of $\theta_i$ alone |
| $S_{i\,\min}$ | minimum sensitivity of $\theta_i$ |
| $S_{\min}$ | minimum sensitivity in any direction |
| $S_{\max}$ | maximum sensitivity in any direction |

A very elongated ellipse, where

$$R = \frac{S_{\max}}{S_{\min}} \tag{6.17}$$

i.e. the ratio of the major half axis over the minor, is much larger than 1, indicate large differences of sensitivities in various directions in the parameter space. If the directions of the ellipse axis differ from the parameter axis, the sensitivity is minimal for a linear combination of the parameters, $\theta_{r2} - \alpha\theta_{r1} = 0$, where $\alpha$ is the slope of the major axis. This indicates that the parameters are correlated. As it appears from Fig.6.2, the correlation is high when the ratio

$$R_i = S_i / S_{i\,\min} \tag{6.18}$$

is much larger than 1. The ratios R and Ri are, accordingly, characteristic for the shape of the ellipse and hence for the sensitivities. These sensitivities and ratios can all be calculated from $H_{rn}$ - which is already available from the Gauss-Newton parameter estimation algorithm - as

$$S_{\max} = \sqrt{\lambda_{\max}} \qquad S_{\min} = \sqrt{\lambda_{\min}} \tag{6.19}$$

where $\lambda$ is an eigenvalues of $\widetilde{H}_{rn}$ , and, according to Theorem 1, Appendix 2, the minimum (as a function of the remaining parameters) sensitivity of $\theta_i$ is

$$S_{i\,\min} = \sqrt{\{H_{rn}^{-1}(\theta_N)\}_{ii}^{-1}} \tag{6.20}$$

**Example 11**

Consider the ellipse

$$x^T H x = 1 \quad \text{where } x = \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} \text{ and } H = \begin{Bmatrix} 4 & -5 \\ -5 & 9 \end{Bmatrix} \tag{6.21}$$

shown in Figure 6.3

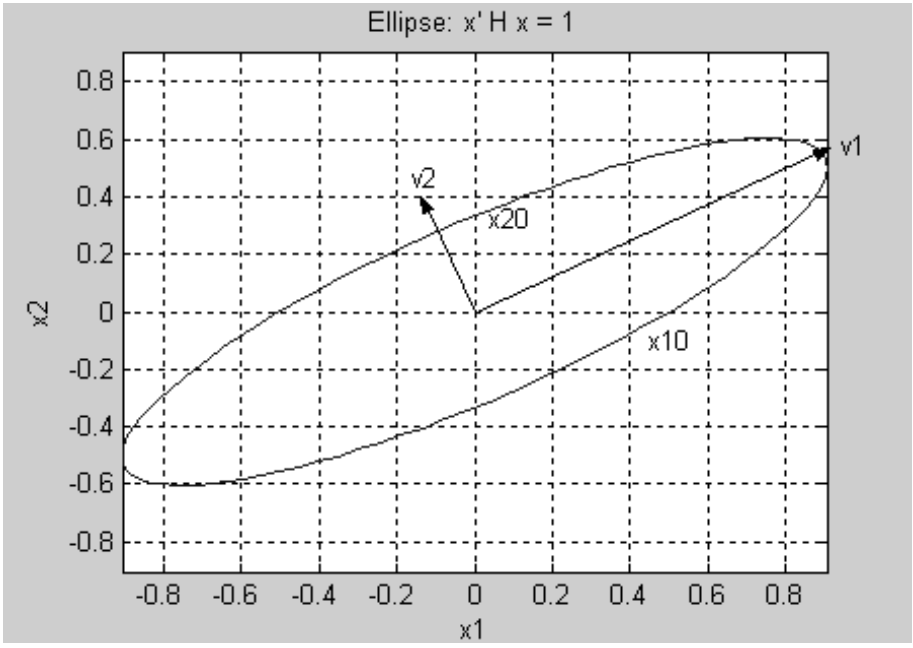


Figure 6.3 Ellipse x' H x = 1

The distance from origo to the intersections with the $x_1$ axis is easily found by setting $x_2 = 0$ to $x_{10} = \pm 1/\sqrt{h_{11}} = \pm 0.5$, and similarly $x_{20} = \pm 1/\sqrt{h_{22}} = \pm 0.333$. According to (6.16) these distances are inverse proportional to the sensitivities in these directions

$$S_i = \frac{1}{|x_{i0}|} \tag{6.22}$$

Now suppose that an orthogonal transformation $v = Vx$ is performed. The sensitivities of the new parameters $v_1$ and $v_2$ are then represented by the reciprocal distances from origo to the intersections on the new axis. That means, that the result (6.22) can be generalized to any direction, i.e. to any linear combination of the parameters $x_1$ and $x_2$.

The special transformation to the directions of the main axes, i.e. to diagonal form, is obtained by using a transformation matrix V containing the eigenvectors of H as columns

$$v = Vx \qquad H_d = V^T H V \quad \text{for} \quad V = \begin{Bmatrix} -0.85 & -0.53 \\ -0.53 & 0.85 \end{Bmatrix} \quad H_d = \begin{Bmatrix} 0.91 & 0 \\ 0 & 12.1 \end{Bmatrix} \tag{6.23}$$

The distances to the intersection with the $v_1$ and $v_2$ axes are then found to be $v_{10} = \pm 1/\sqrt{hd_{11}} = \pm 1.05$, and similarly $v_{20} = \pm 1/\sqrt{h_{d22}} = \pm 0.29$. For this diagonal form the diagonal elements of $H_d$ are equal to the eigenvalues, so the formulas (6.16) and (6.19) coincide.

The distances corresponding to the reciprocal of $S_{i\ \min}$ (6.20) are found to be $v_{1\ \max} = 0.91$ and $v_{2\ \max} = 0.60$.

■

So far, we have considered the two-parameter case, because we then have a two-dimensional graphical representation. If the number of parameters, d>2 the ellipses become d-dimensional ellipsoids, but the sensitivity measures still apply. Indeed, the measures characterizing the shape of the ellipsoid is even more essential, as a graphical presentation is not possible.

The four most characteristic sensitivity measures are summarized in Table 6.1, together with an indication of their best values for accurate parameter estimation. It may be emphasized that as the shape of the ellipse is determined by $H(\theta_N)$ (6.14), the values of the sensitivity measures are dependent on the input signal.

| | |
|---|---|
| $S_{\min}$ | minimum sensitivity, inverse of major half axis - as large as possible |
| $S_{i\,\min}$ | minimum sensitivity of $\theta_i$ - as large as possible |
| $R = S_{\max}/S_{\min}$ | ratio of maximum and minimum sensitivity in any direction or ratio of half axis - as close to 1 as possible |
| $R_i = S_i/S_{i\,\min}$ | ratio of sensitivity of $\theta$i alone and minimum sensitivity of $\theta$i - as close to 1 as possible. $R_i >> 1$ indicates correlation between two or more parameters |

Table 6.1 Characteristic parameter sensitivity measures

If the values of the characteristic parameter sensitivity measures are not satisfactory, the most obvious actions would be to try another input signal or another parametrization - but much more about that later.

The use of the characteristic sensitivity measures shall now be illustrated by two simple examples.

**Example 12** *Parameter sensitivity and correlation*

The parameters $K$ and $\tau$ of the model $G_m(s) = \frac{K}{1+s\tau}$ shall be estimated using a sine input $u = \sin(\omega t)$, and we shall determine a good value of the frequency $\omega$ - first by common sense and next by sensitivity considerations.

$$s = j\omega: \quad G_m(j\omega) = \frac{K}{1+j\omega\tau} \tag{6.24}$$

$$\omega < <1/\tau: \quad G_m(j\omega) \approx K \tag{6.25}$$

$$\omega > >1/\tau: \quad G_m(j\omega) \approx \frac{1}{j\omega}\frac{K}{\tau} \tag{6.26}$$

This means, that for very low frequencies only the gain K can be estimated, and for very high frequencies only the ratio $K/\tau$ can be estimated accurately. A good value is probably a compromise, $\omega \approx 1/\tau$.

Now, let us look at the sensitivity ellipses and the sensitivity measures for the same cases.

For $K = 1$, $\tau = 2$ and $\omega = 0.04$, 0.2, 0.5, 4, the Hessian is calculated using [psi,Hrn]=psinf(u1,t,par0,simmod), the sensitivity measures using [Smin,Simin,R,Ri]=sens(Hrn), and the corresponding sensitivity ellipses are drawn with sellip4sinus - all 3 m-files are in Senstools.

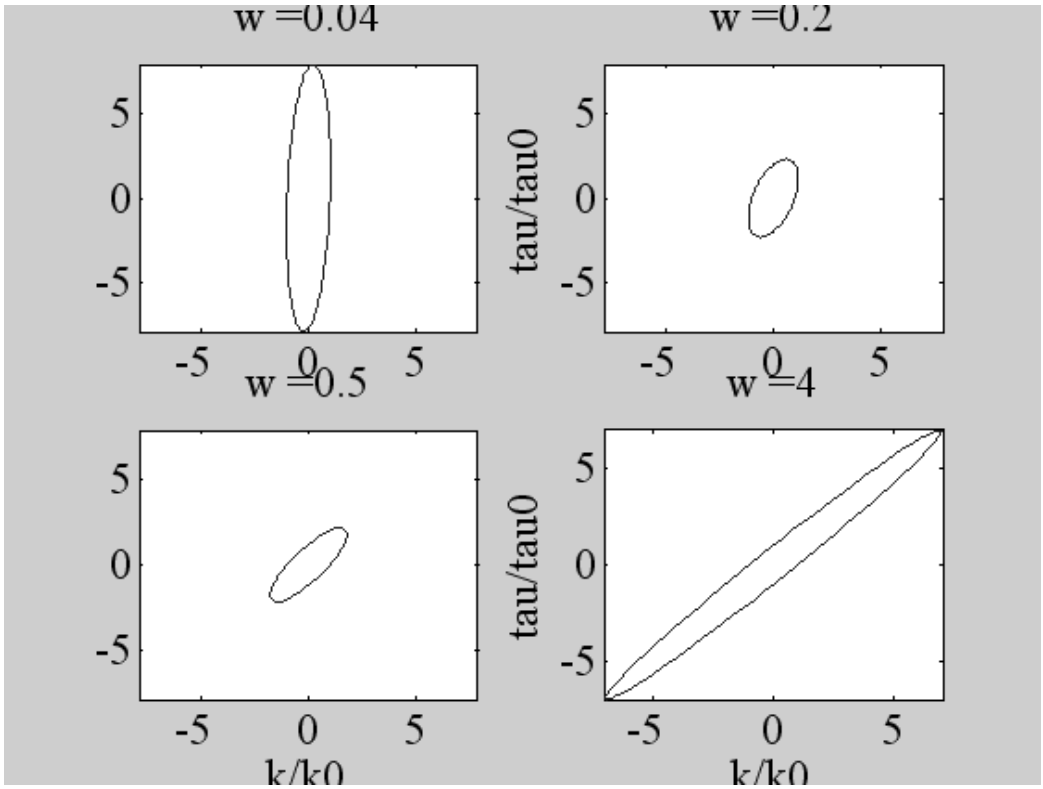


Figure 6.4 Sensitivity ellipses for sinus input with different frequences w

Fig. 6.4 clearly shows, that for a low frequency $\omega = 0.04$ the ellipse is elongated in the direction of $\tau$, indicating a low sensitivity to $\tau$, i.e. only K can be estimated accurately. For a high frequency $\omega = 4$ the ellipse is elongated in a direction different from the axes, indicating low sensitivity for a linear combination of the parameters, that is strong correlation between the parameters. The best ellipse is the smallest and closest to a circle, that is for $\omega = 0.2$.

This is confirmed by the values of the sensitivity measures in Table 6.2. The optimal input frequency can be determined as the one, giving the lowest R-value, $\omega = 0.2$. A low input frequency gives a large difference in the $S_{i\,\min}$ values for $K$ and $\tau$, while a high frequency result in large $R_i$-values, indicating high correlation between $K$ and $\tau$.

| $\omega$ | 0.04 | 0.2 | 0.5 | 4 |
|---|---|---|---|---|
| $S_{\min}$ | 0.13 | 0.42 | 0.37 | 0.10 |
| $S_{i\,\min}$ | 0.98/0.13 | 0.86/0.44 | 0.56/0.46 | 0.14/0.14 |
| $R$ | 7.9 | 2.5 | 3.3 | 13.8 |
| $R_i$ | 1.03/1.03 | 1.2/1.2 | 1.8/1.8 | 6.9/6.9 |

Table 6.2 Characteristic sensitivity measures for different input frequencies

■

For complex systems it can be difficult to ensure that the chosen model structure has a minimum number of parameters, in particular in state-space descriptions. Too many parameters, overparameterization, means 100% correlation between two or more parameters, so a unique set of estimated parameter values cannot be obtained.

The sensitivity measures are very useful to disclose overparameterization.

**Example 13** *Overparameterization*

The model

$$G_m(s) = \frac{A}{1 + s\frac{B}{C}} \tag{6.27}$$

is obviously overparameterized as only the ratio between $B$ and $C$ matters, not the individual values - $B/C$ could be replaced by a single parameter. But let us try to estimate all three parameters and see what happens. mainest.m and the data measktau.mat are used.

It appears that the minimum search is problematic. Because the sensitivity in the direction $B = C$ is zero, huge updates for B and C are attempted. When the Gauss-Newton algorithm stops after maximum numbers of iterations, the parameter estimates are:

$A = 1.002,\ \ B = 2.02 \cdot 10^{23},\ \ C = 1.11 \cdot 10^{23}$, that is a correct A-value, completely random B- and C-values, but a reasonable $B/C-$ratio.

The sensitivity measures tell the story:

$R = 7.01 \cdot 10^4$ ($>> 1$) reveals that something is seriously wrong, and

$Simin = [0.418 \quad 0.000 \ 0.000]$ and $Ri = [0.0002 \quad 2.26 \ 2.26] \cdot 10^4$ clearly indicate, that the problem is correlation between the two last parameters.

■

The formulas for single input single output (SISO) systems above are easily expanded to multivariable systems (MIMO). Multiple inputs make no difference for application of the formulas. For M outputs, M > 1, the output signals consisting of M column vectors, $y_1....y_M$, with each N values are normed individually as equation (6.10) into one normed column vector with N·M elements

$$y_n = \{y_{1n}^T \ y_{2n}^T \ ....y_{Mn}^T\}^T \tag{6.28}$$

The corresponding performance function, gradient, and Hessian are then

$$P_{nM}(\theta) = \frac{1}{2}\frac{1}{NM}\sum_{k=1}^{NM}\varepsilon_n(k,\theta)^2 = \frac{1}{M}(P_{1n} + P_{2n} + ... + P_{Mn}) \tag{6.29}$$

$$G_{rnM} = \frac{1}{M}(G_{1rn} + G_{2rn} + ... + G_{Mrn}) \tag{6.30}$$

$$H_{rnM} = \frac{1}{M}(H_{1rn} + H_{2rn} + ... + H_{Mrn}) \tag{6.31}$$

that is, means of the values for each individual output.

## 6.3 Parameter accuracy

In this section it is shown how the characteristic sensitivity measures can be used for evaluation of the accuracy of the parameter estimates.

The model fit expressed by the minimum value of the performance function, $P(\theta)$ or the normed model error, errn can be related to the parameter estimation accuracy by means of a sensitivity measure. The model error can be split up into three components

$$\varepsilon(k,\theta) = y(k) - y_m(k,\theta) = \varepsilon_x(k) + \varepsilon_m(k) + \varepsilon_p(k,\theta) \tag{6.32}$$

where $\varepsilon_x$ is caused by noise, $\varepsilon_m$ by undermodelling, and the parameter dependent error $\varepsilon_p$ by $\theta$ being different from the optimal value $\theta_N = \arg\min(P(\theta))$. As the first two error components are independent, the minimum value of the performance function is

$$P(\theta_N) = \frac{1}{2}(\varepsilon^2_{x,RMS} + \varepsilon^2_{m,RMS}) \tag{6.33}$$

We now consider the effect of the two error components on the parameter estimate separately.

**Noise only.**

First we assume that the model structure is correct $S \in M$, i.e. $\varepsilon_m(k) = 0$, and $errn = 100{\cdot}\varepsilon_{x,RMSn}$, and that the error $\varepsilon_x(k)$ is white and Normal distributed. According to [Ljung 1987] an estimate of the parameter covariance is

$$cov\widehat{\theta}_N = \frac{2}{N} P(\theta_N) H^{-1}(\theta_N) \tag{6.34}$$

and the spread of the i-th parameter $\theta_i$ consequently

$$\sigma_{\theta i} = \sqrt{\frac{2}{N} P(\theta_N)(H^{-1}(\theta_N))_{ii}} \tag{6.35}$$

Inserting 6.20 and 6.33 the estimated relative parameter spread can be expressed by physical significant quantities:

$$\sigma_{r,\theta i} = \frac{\varepsilon_{x,RMSn}}{S_{i\,\min}} \frac{1}{\sqrt{N}} \quad \text{or} \quad \sigma_{r,\theta i\%} = \frac{errn}{S_{i\,\min}} \frac{1}{\sqrt{N}}\ [\%] \tag{6.36}$$

This important result says that *the relative spread, caused by noise, of the estimate of $\theta_i$ is proportional to the normed root mean square output error (6.3), inverse proportional to the parameter sensitivity (6.20) and inverse proportional to the square root of the number of data points* - which makes sense, as the effect of noise on an estimate is smoothed out with $\sqrt{N}$.

**Undermodelling only.**

Next the effect of undermodelling without noise is addressed, i.e., $S \notin M$ and $\varepsilon_x(k) = 0$, i.e. $errn = 100{\cdot}\varepsilon_{m,RMSn}$. We introduce the concept of *the equivalent parameter error*, $\widetilde{\theta}_{eq}$, defined by

$$\widetilde{\theta}_{eq} = \{\widetilde{\theta}_r \in \Re^d \mid \varepsilon_{p,RMSn}(\widetilde{\theta}_r) = \varepsilon_{m,RMSn}\} \tag{6.37}$$

that is, *the relative parameter error* $\widetilde{\theta}_r$ *that would cause an output error* $\varepsilon_{p,RMSn}(\widetilde{\theta}_r)$*, equal to the actual error caused by undermodelling,* $\varepsilon_{m,RMSn}$*.*

According to (6.15) we then have

$$\widetilde{\theta}_r^T H_{rn} \widetilde{\theta}_r = \varepsilon_{m,RMSn}^2 \tag{6.38}$$

which describe an ellipsoid equal to the sensitivity ellipsoid multiplied by $\varepsilon_{m,RMSn}$. The maximum value of the equivalent error of parameter $\theta_i$ is then (Theorem 1, Appendix C)

$$\Delta_{\theta eq,i} = \frac{\varepsilon_{m,RMSn}}{S_{i\,\min}} \quad \text{or} \quad \Delta_{\theta eq,i\%} = \frac{errn}{S_{i\,\min}}\ [\%] \tag{6.39}$$

So *the maximal error of the estimate of* $\theta_i$ *caused by undermodelling is simply the normed root mean square output error (6.3) divided by the parameter sensitivity (6.20).* This expression for the equivalent error has strong similarity to the one for the spread caused by noise. The difference being, that it is independent of the number of data points N. Also this is makes sense - the effect of the deterministic undermodelling is not smoothed out by many measurements as the effect of the stochastic noise.

**Noise and undermodelling.**

In most practical applications there is both noise and undermodelling. We then define the total relative parameter uncertainty as the sum of the relative parameter spread and the relative equivalent parameter error

$$\Delta_{\theta total,i} = \sigma_{r,\theta i} + \Delta_{\theta eq,i} = \frac{1}{S_{i\,\min}}\left(\frac{\varepsilon_{x,RMSn}}{\sqrt{N}} + \varepsilon_{m,RMSn}\right) \tag{6.40}$$

A fundamental problem for using (6.40) is, how to split the output error up into the two parts caused by noise and undermodelling respectively. If $\varepsilon_{x,RMSn}$ and $\varepsilon_{m,RMSn}$ are of the same order of magnitude, the effect of undermodelling is dominating, as N is typically $> 100$, i.e. $\sqrt{N} > 10$.

A worst case parameter error can be calculated by assuming the total output error is caused by undermodelling, and a best case by assuming the total output error is caused by noise. This is done in mainest (Senstools) and denoted dpar and sigpar respectively.

A simple way to split the output error up into the two parts is to measure an output sequence with zero input and calculate $\varepsilon_{x,RMSn}$ from this, assuming the noise is independent of the input. This method is illustrated in the following example.

**Example 14** *Determination of parameter spread with noise and undermodelling.*

The purpose of this example is to illustrate how accurately the parameter spread can be estimated in case of undermodelling. The three systems

$$S1:\ \frac{1}{1+2s} \qquad S2:\ \frac{1}{(1+2s)(1+0.4s)} \qquad S3:\ \frac{1}{(1+2s)^2} \tag{6.41}$$

are all identified, using the same model structure

$$G(s) = \frac{K}{1 + s\tau} \tag{6.42}$$

A square wave is used as input, and white noise N{0, 0.02} is added to the system output. $\varepsilon_{x,RMSn}$ is determined by a zero-input experiment, and based on this, the parameter spreads spr_K_est and spr_$\tau$_est are estimated using (6.36). For comparison the actual parameter spreads spr_K_calc and spr_$\tau$_calc are calculated from 100 repeated runs. The results are presented in Table 6.3

| | spr_K_calc | spr_K_est | spr_$\tau$_calc | spr_$\tau$_est | $K$_est | $\tau$_est |
|---|---|---|---|---|---|---|
| S1 | .0012 | .0012 | .0031 | .0029 | 1.00 | 2.00 |
| S2 | .0012 | .0013 | .0023 | .0026 | 1.05 | 2.55 |
| S3 | .0018 | .0019 | .0021 | .0026 | 1.22 | 5.07 |

Table 6.3 Calculated and estimated relative parameter spread, and estimated parameter values for the three systems (6.41)

For all three systems there is very good agreement between calculated and estimated spreads of K, while for $\tau$ the estimated spread values deviate 7%, 11% and 24% for the three systems. Considering the severe structure mismatch, in particular for S3, and the fact that very accurate spread values are hardly ever required and expected, the results indicate that the method is adequate.

If the total output error $\varepsilon_{RMSn}$ , including the contribution from undermodelling, is used for estimating parameter spreads (which is normally the case in traditional system identification), the discrepancies are in the area of 200% and 400% for S1 and S2. This is generally unacceptable.

■

**Example 15** *Determination of equivalent parameter error with noise and undermodelling.*

The example is a (simulated) system consisting of a loudspeaker with time varying voice coil resistance. During the experiment the voice coil resistance $R_e$ increases from 5.0 to 8.2 Ω due to increase in temperature. The model structure, however, presumes a constant $R_e$-value. $R_e$ is the only free parameter in this example.

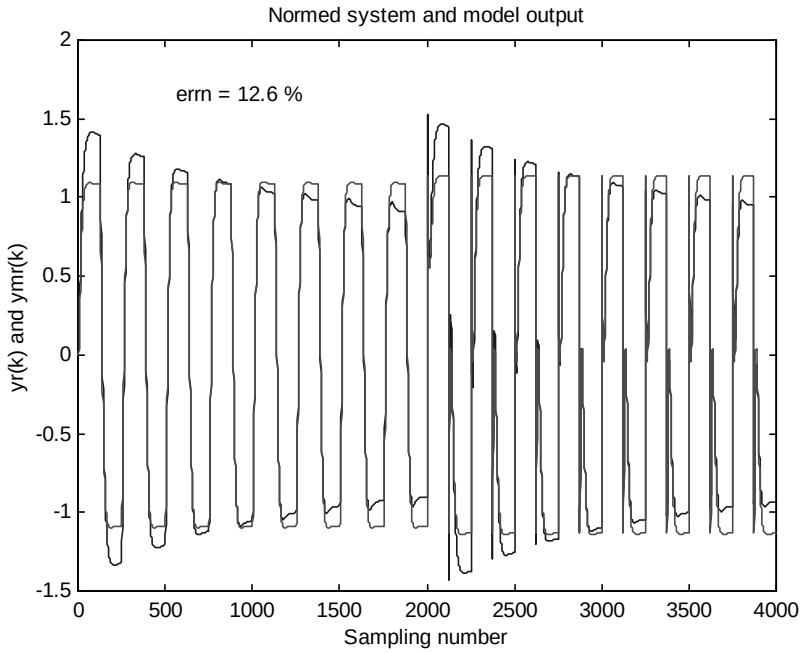


Figure 6.5 Loudspeaker output - velocity and current - and corresponding model output

From Figure 6.5 it appears, that the output error is due to undermodelling - not noise - and the total errn is therefore used in (6.39) for calculating the equivalent parameter error for $R_e$. The result is an estimated equivalent parameter error of 14.1%. In comparison the root mean square parameter error over the time for the experiment is 13.5%, and the errors corresponding to the start and final $R_e$-values are 25.6% and 21.3% respectively. The estimated $R_e$-value is 6.7 Ω which is close to the average resistance $\mathrm{R}_{e,mean} = 6.8\ \Omega$.

The conclusion of the results is, that the estimated equivalent parameter error gives a reasonable ball-park figure for the expected parameter accuracy in case of severe undermodelling.

Another important point is, that Fig. 6.4 clearly shows that the model structure is inadequate - something time varying is missing in the model. So unless a model with constant $R_e$ is required (for example for control purpose), the temperature dependence should be included in the model.

■

## 6.4 Resumé

A good model fit is a necessary but not sufficient condition for an accurate model. The model fit can be expressed by the normed root mean square output error, errn [%]. But even more informative is a plot of the model output together with the measured system output. This plot gives excellent indication of where the model may be deficient, and very often ideas how to make improvements in the model structure.

To ensure that a good fit also implies accurate parameter estimates, parameter sensitivity and characteristic sensitivity measures were introduced, see Table 6.1

| | |
|---|---|
| $S_{\min}$ | minimum sensitivity, inverse of major half axis - as large as possible |
| $S_{i\,\min}$ | minimum sensitivity of $\theta_i$ - as large as possible |
| $R = S_{\max}/S_{\min}$ | ratio of maximum and minimum sensitivity in any direction or ratio of half axis - as close to 1 as possible |
| $R_i = S_i/S_{i\,\min}$ | ratio of sensitivity of $\theta$i alone and minimum sensitivity of $\theta$i - as close to 1 as possible. $R_i$ >>1 indicates correlation between two or more parameters |

Table 6.1 Characteristic parameter sensitivity measures

If the ratio R is inadequate (R>>1), the reason can be a poor input signal or severe parameter correlation. Parameter correlation, including which parameters are correlated, is revealed by $R_i$.

Based on the sensitivity measures the accuracy of the parameter estimates can be expressed in a very informative manner.

The relative spread caused by noise is

$$\sigma_{r,\theta i} = \frac{\varepsilon_{x,RMSn}}{S_{i\,\min}} \frac{1}{\sqrt{N}} \quad \text{or} \quad \sigma_{r,\theta i\%} = \frac{errn}{S_{i\,\min}} \frac{1}{\sqrt{N}} \ [\%] \tag{6.43}$$

and the relative error caused by undermodelling is

$$\Delta_{\theta eq,i} = \frac{\varepsilon_{m,RMSn}}{S_{i\,\min}} \quad \text{or} \quad \Delta_{\theta eq,i\%} = \frac{errn}{S_{i\,\min}} \ [\%] \tag{6.44}$$

These two expressions possess wealth of intuitively understandable information.

In Chapter 8 the sensitivity measures will be used for experiment design, in particular for design of a good input signal.

# Chapter 7

# Frequency-domain considerations

*System identification is normally fitting the model to measurements in the time domain. In this chapter it is shown how the frequency domain can be a valuable supplement to the time domain.*

In real life the model structure and the system structure will never be identical. It is therefore necessary to resolve how the model should fit the measurements, to extract the most important characteristics of the system, corresponding to the most accurate parameter estimates. Often the obvious procedure is to specify the frequency range, where the model accuracy is most critical. Accordingly, frequency-domain considerations are a valuable supplement to time-domain system identification methods.

Frequency-domain methods apply for linear systems only, but for small deviations from an operating point the results are often useful for nonlinear systems as well.

In this chapter it is first demonstrated, how an optimal model fit in the time domain is equivalent to an optimal weighed fit in the frequency domain. Next it is shown how the parameters can be estimated by fitting the model to a measured frequency function. As stated in Chapter 2, frequency response data are more time consuming to obtain than transient response (time-domaine) data, but in retaliation frequency-domain simulation is much easier to perform, than time-domain simulation.

## 7.1 Time-domain fit analyzed in the frequency domain

The model output error in the time domain is - disregarding the noise:

$$\varepsilon(k,\theta) = y(k) - y_m(k,\theta) = (G_o(q) - G(q,\theta))u(k) \tag{7.1}$$

where $G_o(q)$ and $G(q,\theta)$ are transfer functions for the system and the model.
For a time sequence u(k), k=1,2,3,....,N the discrete time Fourier Transformation is

$$U_N(\omega) = \frac{1}{\sqrt{N}} \sum_{k=1}^{N} u(k) e^{-j\omega k} \tag{7.2}$$

and the inverse transformation is

$$u(k) = \frac{1}{\sqrt{N}} \sum_{l=1}^{N} U_N(\omega_l) e^{j\omega_l k} \quad \text{where} \quad \omega_l = \frac{2\pi l}{N} \tag{7.3}$$

Parsevals relationship expresses that the energy of the time-domain sequence is equal to the energy of the frequency domain spectrum

$$\sum_{k=1}^{N} u^2(k) = \sum_{l=1}^{N} \mid U_N(\omega_l) \mid^2 \tag{7.4}$$

Using this, the time-domain performance function can be expressed in the frequency domain

$$P_N(\theta) = \frac{1}{2N}\sum_{k=1}^{N}\varepsilon^2(k,\theta) = \frac{1}{2N}\sum_{l=1}^{N} \mid E_N(\frac{2\pi l}{N}) \mid^2 = \frac{1}{2N}\sum_{l=1}^{N} \mid G_o(\omega_l) - G(\omega_l,\theta) \mid^2 \mid U_N(\omega_l) \mid^2 \tag{7.5}$$

From this expression we make the important observation: *the time domain parameter estimate*

$$\theta_N = \arg\min_{\theta} P_N(\theta) \tag{7.6}$$

*is also the least squares estimate in the frequency domain with a frequency weighting*

$$Q(\omega) = \mid U_N(\omega) \mid^2 \tag{7.7}$$

Thus a good fit between the system frequency function $G_o(\omega)$ and the model frequency function $G(\omega,\theta)$ is emphasized at the frequencies where the input signal power is high.

### 7.1.1 Prefilters

Prefilters L(q) are digital filters, working on the sampled input and output sequences

$$u_f(k) = L(q)\ u(k) \quad \text{and} \quad y_f(k) = L(q)\ y(k) \tag{7.8}$$

and the corresponding model error is then obtained from (7.1)

$$\varepsilon_f(k,\theta) = y_f(k) - G(q,\theta)u_f(k) = L(q)(G_o(q) - G(q,\theta))u(k) \tag{7.9}$$

Using $\varepsilon_f(k,\theta)$ instead of $\varepsilon(k,\theta)$ in (7.5) shows, that the prefilter will give a total frequency weighting

$$Q(\omega) = \mid L(\omega) \mid^2 \mid U_N(\omega) \mid^2 \tag{7.10}$$

Consequently, if it is not possible to make an experiment with an input signal power spectrum, giving the desired frequency weighting, a prefilter can be used for shifting the frequency weighting.

For nonlinear systems and models, prefilters (7.8) should not be used. Instead the model error can be filtered before applied in the performance function

$$\varepsilon_f(k) = L(q)\ \varepsilon(k) \tag{7.11}$$

## 7.2 Parameter fitting in the frequency domain

The frequency function of the system is measured at a number of frequencies $\omega_k$ by measuring the amplitude and phase of the output y corresponding to an input $u(t) = c_k \sin(\omega_k t)$. For the corresponding complex frequency function we have

$$Y(j\omega_k) = G_{of}(j\omega_k)U(\omega_k) \tag{7.12}$$

and similar for the model

$$Y_m(j\omega_k) = G_f(j\omega_k, \theta)U(\omega_k) \tag{7.13}$$

For fitting the complex frequency function of the model to that of the system at a number, N of frequencies we use the performance function

$$P_f(\theta) = \frac{1}{2N}\sum_{k=1}^{N} \mid E_f(j\omega_k) \mid^2 = \frac{1}{2N}\sum_{k=1}^{N} \mid G_{of}(j\omega_k) - G_f(j\omega_k, \theta) \mid^2 \mid U(j\omega_k) \mid^2 \tag{7.14}$$

$$= \frac{1}{2N}\sum_{k=1}^{N}\{(\mathrm{Re}(G_{of}(j\omega_k) - G_f(j\omega_k, \theta))^2 + \mathrm{Im}(G_{of}(j\omega_k) - G_f(j\omega_k, \theta))^2\}c_k^2 \tag{7.15}$$

where $E_f(j\omega_k)$ is the complex frequency function error at $\omega_k$.

To deal with both real and imaginary parts we introduce a model output vector z for each frequency $\omega_k$

$$z_k(\omega_k, \theta) = [\mathrm{Re}(Y_m(j\omega_k)) \quad \mathrm{Im}(Y_m(j\omega_k))]^T = c_k[\mathrm{Re}(G_f(j\omega_k, \theta)) \quad \mathrm{Im}(G_f(j\omega_k, \theta))]^T \tag{7.16}$$

In analogy with the time domain case, we also introduce a frequency domain model gradient

$$\psi_{fk} = \frac{\partial z_k}{\partial \theta^T} = c_k[\frac{\partial \mathrm{Re}(G_f(j\omega_k, \theta)}{\partial \theta^T} \quad \frac{\partial \mathrm{Im}(G_f(j\omega_k, \theta)}{\partial \theta^T}]^T \tag{7.17}$$

For multiple (N) sinusoids the model output is arranged in a vector, as for multiple output systems in the time domain, i.e. the system is considered having two outputs, real and imaginary parts of the sine responses

$$z = [c_1 \mathrm{Re}(G_{f1})...c_N \mathrm{Re}(G_{fN}) \quad c_1 \mathrm{Im}(G_{f1})...c_N \mathrm{Im}(G_{fN})]^T \quad \text{where} \quad G_{fk} = G_f(j\omega_k, \theta) \tag{7.18}$$

and the corresponding model gradient matrix

$$\psi_f = \left\{ \begin{array}{ccc} c_1\frac{\partial \mathrm{Re}(G_{f1})}{\partial \theta_1} & \cdot \cdot & c_{nw}\frac{\partial \mathrm{Im}(G_{fN})}{\partial \theta_1} \\ \cdot & \cdot \cdot & \cdot \\ \cdot & \cdot \cdot & \cdot \\ c_1\frac{\partial \mathrm{Re}(G_{f1})}{\partial \theta_N} & \cdot \cdot & c_N\frac{\partial \mathrm{Im}(G_{fN})}{\partial \theta_N} \end{array} \right\}^T \tag{7.19}$$

The frequency domain Hessian is analogously

$$H_f = \psi_f^T \psi_f \tag{7.20}$$

**Calculating the parameter estimate**

The parameter estimation is performed with the Gauss-Newton procedure, and actually the Senstools programs Gausnewt.m and psinf.m (for calculating $\psi$ and H) developed in the time domain can be used directly in the frequency domain as well.

Only the simulation programs must - obviously - be special for the frequency domain. Frequency domain simulation requires much less calculation in Matlab, as it is only requires a substitution of s with the complex frequencies $j\omega_k$, and simple calculations. An example will illustrate this.

**Example 16** *Frequency-domain simulation program*

The function z=simfktau(wv,av,par) calculates the output z (defined in (7.18)) at a number of frequencies specified in the vector Wv and with amplitudes specified in av.

```
function z=simfktau(wv,av,par)
% function z=simfktau(wv,av,par) calculates real (a) and imaginary (b)
% parts of the frequency response at frequencies wv and amplitudes av
% of the transfer function par k/(1+s*tau), k=par(1) tau=par(2)
% 12/1-03,MK

s=j*wv;
y=par(1)./(1+par(2)*s); y=y*diag(av);
a=real(y); b=imag(y);
z=[a(:)' b(:)']';
```

■

## 7.3 Resumé

A least squares fit in the time domain is equal to a weighed least squares fit of the frequency function. The frequency weighting factor is

$$Q(\omega) =\mid U_N(\omega) \mid^2 \tag{7.21}$$

that is, the fit is best in the frequency range, where the input signal power is high. The fit can be further weighted by filtering the signals u(k) and y(k) by a so called prefilter L(q), giving a weighting factor

$$Q(\omega) =\mid L(\omega) \mid^2 \mid U_N(\omega) \mid^2 \tag{7.22}$$

The model parameters can alternatively be estimated by fitting the model to a measured frequency function, i.e. using a frequency-domain performance function of the type

$$P_f(\theta) = \frac{1}{2N} \sum_{k=1}^{N} \mid G_{of}(j\omega_k) - G_f(j\omega_k, \theta) \mid^2 \tag{7.23}$$

Again a frequency weighting can be obtained by choosing different input sine amplitudes.
By introducing a model output vector

$$z = [c_1 \operatorname{Re}(G_{f1}) ... c_N \operatorname{Re}(G_{fN}) \;\; c_1 \operatorname{Im}(G_{f1}) ... c_N \operatorname{Im}(G_{fN})]^T \quad \text{where} \quad G_{fk} = G_f(j\omega_k, \theta) \tag{7.24}$$

a frequency-domain model gradient $\psi_f$ and Hessian $H_f$ can easily be calculated, and the Senstools programs developed in the time domain, can be used for parameter estimation. The frequency-domain simulation model is easier to program and requires simpler calculations than the time-domain simulation program.

# Chapter 8

# Input signal design

*The experimental conditions for obtaining input–output measurements - in particular the input signal - is important for the final result of the parameter estimation. An optimal input signal is designed using the characteristic sensitivity measures.*

In Chapter 7 it vas demonstrated that the best model fit is obtained in the frequency range, where the input signal power is high. Accordingly the input signal should be chosen to have most of its power in the frequency range where the accuracy of the model is most important.

For nonlinear systems it is equally important, that the amplitude range of the input signal corresponds to the amplitude range, where the accuracy of the model is most important.

In this chapter a procedure for determination of good input signals is given and illustrated by examples.

## 8.1 Input signal design procedure

If the input signal for the experiment can be chosen freely, the following procedure for design of a good input signal is recommended:

1. *Obtain approximative parameter estimates (Chapter 2) or acquire à priori parameter values.*

2. *Choose a class of preliminary input signals with feasible frequency and amplitude distribution.*

   The input signals shall depend on few (preferably just one) input signal parameters. One parameter, at least, shall control the frequency spectrum; if the model is nonlinear, an additional input signal parameter shall control the amplitude. Use intuition and physical insight.

3. *Optimize the input signal for best possible sensitivity measures (simulation).*

   Calculate and plot some of the characteristic sensitivity measures as a function of input signal parameters, and choose best values of these according to Chapter 6.

4. *Use the determined input signal on the real system, obtaining an improved parameter estimate.*

   If necessary, repeat the procedure with the improved parameter values.

The Senstools program for optimal input design is called *maininp.m.* It has some similarities with mainest (main program for parameter estimation, described in Chapter 5), but where mainest estimates the parameters for measured input output, maininp simulates the system with different inputs and calculates and plots the sensitivity measures, to determine the optimal input within the chosen class. The optimal input is the one minimizing the sensitivity ratio $R = S_{\max}/S_{\min}$, defined in Chapter 6.

The procedure is now illustrated by two examples.

**Example 17** *Optimal input square wave for linear system.*

For identification of linear systems a square wave input is often a good choice, as it has a frequency spectrum with most energy at low frequencies, where the accuracy is normally most important. Consequently we choose the class of inputs as square wave signal, and optimize the sensitivity measures with respect to the fundamental frequency.

The model in this example is the same as in Example 9, $G_m(s) = \frac{K}{1+s\tau}$, $K = 1$, $\tau = 2$, and as this is the default case of maininp, we just have to write in the Matlab command window:

>> clear

>> maininp

Plots of model outputs corresponding to the chosen inputs are first displayed. Next a plot of the four most important characteristic measures as functions of the fundamental square wave input is shown, see Fig.8.1

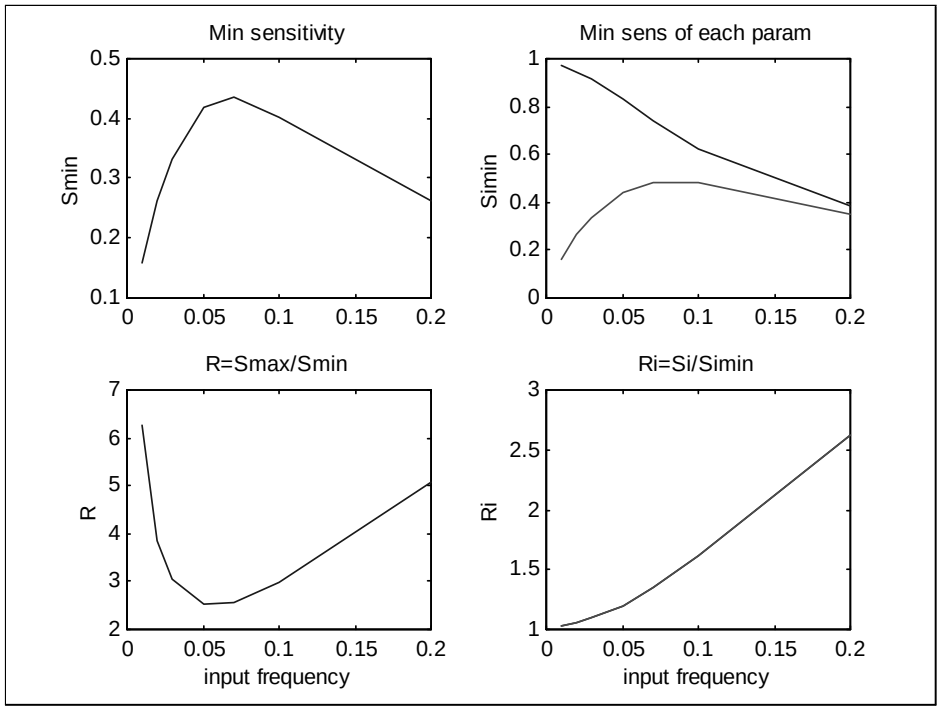


Figure 8.1 Sensitivity measures as a function of fundamental square wave frequency.

The optimal frequency is determined to 0.05 Hz or 0.31 rad/sec, the value minimizing R, giving Smin = 0.42 R = 2.5. (This is the same values as for a sine input with $\omega = 0.2$ rad/sec found in Example 9).

Fig. 8.1 shows that the optimal value is close to maximum of $S_{\min}$ , and it is a sensible compromise for the individual sensitivities $S_{i,\min}$ of $K$ and $\tau$. The plot of $R_i$ clearly shows the higher fundamental frequency gives higher parameter correlation.

The output corresponding to the optimal input is shown in Fig, 8.2. Common sense indicate, that this contains ample information about the magnitudes of $K$ and $\tau$.

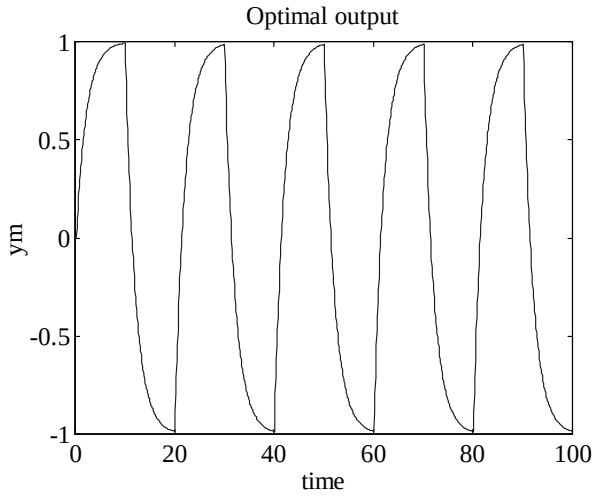


Figure 8.2 Model output corresponding to optimal input

■

**Example 18** *Optimal input for nonlinear system.*

For the nonlinear system kutau from Example 7, Chapter 5

$$G_m(s) = \frac{K(u)}{1+s\tau} \quad \text{where} \quad K(u) = k_0(1 + k_1/(0.5 + u^2)) \tag{8.1}$$

$$\text{with} \quad [k_0, k_1, \tau] = [1.5,\ 4,\ 2] \tag{8.2}$$

an optimal input shall be found. As the system is nonlinear the amplitude distribution is important, and an input class of so called square-ramp is chosen. It is a square wave signal superposed by a ramp function, see Fig. 8.4. The input signal parameter to be varied is the maximum value of the input signal.

The fundamental frequency is chosen as the optimal value in example $f = 0.05$ Hz, because the value of the time constant is the same, $\tau = 2$.

As there are many program data, we use a progprog-program to generate these:

```
% progproginpkutau creates program data for maininp:
% progdatainpkutau.mat
% 15/1-03,MK
clear
process='kutau';
inputt='inpsqramp';
par=[1.5,4,2];
```

```
h=par(3)/20;
f1v=.05;
amv=[ 2 3 5 6 8 10 15 20];
save progdatainpkutau process inputt par h f1v amv
```

and the input design is done by just entering

>> progproginpkutau
>> maininp

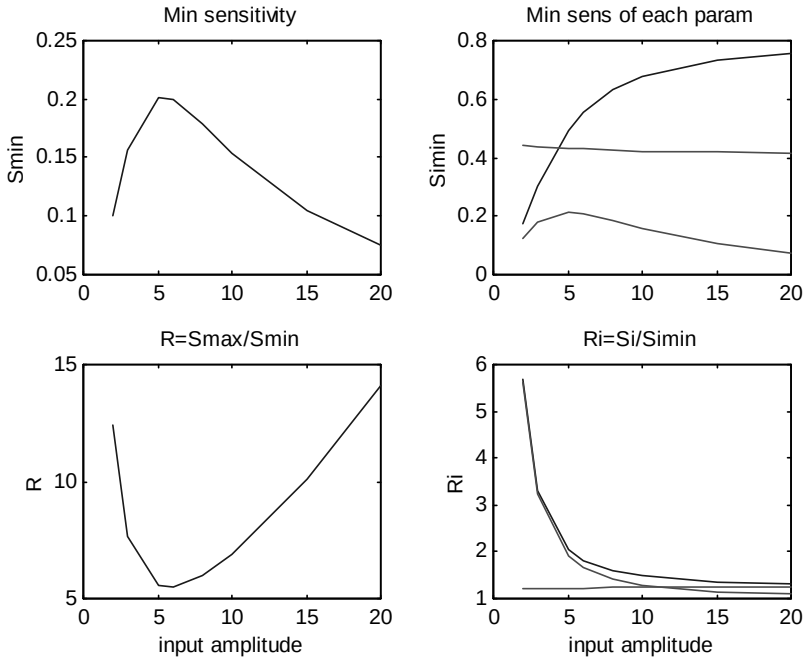


Figure 8.3 Sensitivity measures as a function of input amplitude

Fig. 8.3 shows that the ratio R displays a distinct minimum (R=5.5 for am=6), and that $S_{\min}$ has its maximum value ($S_{\min} = 0.2$) for the same am-value. It is also worth noticing, that the smallest sensitivity $S_{2,\min}$ (for $k_1$) has its maximum value for am=6.

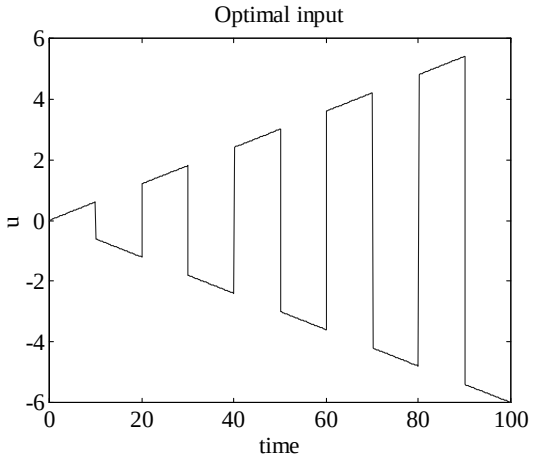


Figure 8.4 Optimal square-rampe input signal

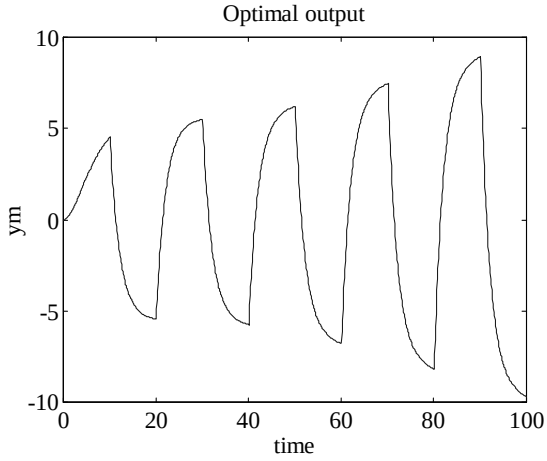


Figure 8.5 Optimal output signal

■

The two examples demonstrate the procedure for designing optimal input signals within a specified class for linear as well as nonlinear systems. Plots of the four characteristic sensitivity measures give valuable information of the qualities of the various input signals within the class.

The optimal input is chosen as the one minimizing the ratio R, and as this value also gives (close to) maximum value of $S_{\min}$ and of the smallest $S_{i,\min}$, it appears to be a good design criteria.

The examples only have one parameter characterizing the input (square wave frequency and amplitude, respectively). With two or more parameters to characterize the input, the optimization becomes an iterative, alternating process.

It is very important to use a good input signal for the experiment, but it is seldom critical if the signal is exactly optimal. In many cases common sense is sufficient for determination of the input signal, and often there are practical restrictions for what inputs can be generated and used. In these cases it may be a good idea to calculate the sensitivity measures for an optimal input, and compare to those for the actual input, to determine if the actual input is acceptable. In this relation it is worth remembering, that the parameter uncertainties are inverse proportional to $S_{i,\min}$, cf. Chapter 6.

## 8.2 Resumé

A procedure for designing an optimal input signal within a certain class was described and illustrated by two examples. For a linear system a square wave input was used, and the optimization consists in determination of the fundamental frequency, minimizing the ratio R. For a nonlinear system the amplitude distribution minimizing R was determined for an optimal signal. Minimum R appears to be a reasonable criteria for optimal input signal, as it causes near optimal values for the other sensitivity measures as well.

# Appendix A

# SENSTOOLS

*The use of Senstools for parameter estimation, accuracy verification and input design was described in Chapter 5, 6 and 8. This appendix contains a brief manual for Senstools including further facilities, an overview of Senstools files and a listing of some of the most important files.*

The programs in Senstools are organized as main programs (script files) calling sub programs (functions) and using data (mat-files).

## A.1 Name conventions

The program and data file names contain information of the program type and of the actual process name.

The initial letters designate the type:

**main** main program (script file)

**sim** simulation of process (function file)

**meas** input/output measurement data (mat-file)

**prog** program data. **progdata** (mat-file) are created by **progprog** (program data program, script file).

Other significant abbreviations:

**est** estimation

**inp** input. Programs for creating input signals or for optimal input design.

Names of files being particular for a certain process will contain the process name. For example, for a process with the name *'motor'* the name of the simulation program is *simmotor.m*, and the measured data are stored in *measmotor.mat*

## A.2 Senstools files, overview

Some of the most important Senstools files are presented in Table A.1. The files are ordered alphabetically under subgroups.

For functions the input and output arguments are shown. A process name is indicated by xxx. The help function is recommended for further information (eg. help inpstair).

| File name | Description |
|---|---|
| **Main programs** | |
| mainest * | parameter estimation by gausnewt. Requires simxxx.m and measxxx.mat |
| maininp * | optimal input design minimizing R. Requires simxxx.m |
| maininpw | optimal multisine input design in the frequency domain. Req simwxxx.m |
| mainsimest | creates simulated "measurement" data and parameter estimation |
| **Sub programs** | |
| gausnewt * | [par, errn, Hrn]=gausnewt(simxxx,par0,u,y,t). Optimal parameter estimate |
| psinf * | [psi, Hrn]=psinf(u,t,par,simxxx). Calc model gradient psi for gausnewt |
| sens * | [Smin,Simin,R,Ri]=sens(Hrn,W). Calculates sensitivity measures |
| sensplot | plots sensitivity measures versus input signal parameter |
| **Utility programs** | |
| diamainest | makes a diary for mainest |
| diamaininp | makes a diary for maininp |
| enoise | makes normal distributed filtered white noise |
| sellip | draws sensitivity ellipse for Hrn ($2 \times 2$ only) |
| **Input signals** | |
| inpstair | [u,t]=inpstair(ns,am,nu,h) generates a staircase signal with ns stairs |
| inpstep | [u,t]=inpstep(f1,am,nu,h), step of amplitude a and lengt n. f1 is a dummy. |
| inpsqw | [u,t]=inpsqw(f1,am,nu,h), square wave with fundamental frequency f1 |
| inpsqramp * | [u,t]=inpsqramp(f1,am,nu,h), sq wave superposed by rampe, final ampl am |

| **Simulation prog** | *functions of the structure y=simxxx(u,t,par)* |
|---|---|
| simABC | A/(s*B/C + 1) Overparameterized Example 12 |
| simdcml * | linear dc-motor with input u and output y=[i,w], Example 10 |
| simfktau | z=simfktau(wv,av,par), freq response at freq wv and ampl av. Ex 15 |
| simnonlex2 | saturation and 2. oder dynamics, kzetaw, Problem 4.3 |
| simktau | k/(1+s tau), Example 3 and 9 |
| simktaufilt | k/(1+s tau), Example 3 using 'filter' |
| simktauloop | k/(1+s tau), Example 3 using for-loop |
| simkzetaw | 2. oder dynamics, kzetaw, Problem 4.2 |
| simkutau | nonlinear k(u)/(1+s tau), Example 9 |
| simsatktau | saturation and 1. oder dynamics, ktau, Example 5 |
| simsplf | linear loudspeaker with input u and output y=[i,x], Example 7 |
| simspn | nonlinear loudspeaker with input u and output y=[i,x], Example 8 |
| **progprog files** | |
| progprogdcml | program data for dc-motor. Example 10 |
| progprogkutau | program data for Example 9b |
| **mat files** | |
| measABC | measurement data for Example 12 |
| measdcml | measurement data for Example 10 (sqw ) |
| measdcml1 | measurement data for Problem 5.2 (step, mk_14_11_29 data) |
| measdcml8 | measurement data for dc-motor demo |
| measktau | measurement data for Example 9c (default data for mainest) |
| measkutau | measurement data for Example 9a |
| progdatakutau | program data for Example 9b |
| progdatadcml | program data for Example 10 |

Table A.1 Overview of Senstool files
The files with an * are listed in the next section.

## A.3 Senstools files, listing

Some of the most important programs are listed: gausnewt, mainest, maininp, psinf, sens, inpsqramp, simdcml.

**gausnewt:**

```
function  [par, errn,Hrn] = gausnewt(simmod,par0,u,y,t,plottype,weight)
%   [par,errn,Hrn] = gausnewt(simmod,par0,u,y,t)  determines the physical
%  parameters  par  in a simulation model  simmod.m  using a
%  Gauss Newton algorithme. Relative gradients are used  and, for multiple
%  output (MO)  systems, a normed model output.
%  The relative model gradient ,psir is determined by psinf.m.
%
%  The function is called with:  measured input u = [u1(t)  u2(t) ...] ,
%  measured output  y = [y1(t)  y2(t) ...] , a time vector t,
%  and initial parameter values par0.
%  The system input  u1, u2 ...  and system output  y1, y2 ...  must all
%  be column vectors.
%
%  Results of the algorithm:  estimated parameter vektor par, plot of
%  measured output and model output, normed output error errn,
%  and a relative, normed Hessian, Hrn.
%
%  [par,errn,Hrn] = gausnewt(simmod,par0,u,y,t,plottype,weight)
%  For multiple system outputs the choice,  plottype = 1 plots all outputs wih
%  same time axis, while  plottype = 2  plots the outputs in succession.
%  Default is plottype = 2.
%  With  weight = [f1  f2 ...]  the individual outputs are weighted.
%  Default is weight = [1 1 ...].
%
%  20/9-1994 JGJ+MK.
%  29/1-2001, MK: updated, isempty( ) in line 144 and 145
%  26/11-02, MK: plot text adjusted

par = par0(:);
mumin = 0.01;     upddmin = 0.001;     noomax = 20;

N = length(t);    [Ry,Cy] = size(y);   [Ru,Cu] = size(u);    NM = N*Cy;

if N~=Ry
   error('The individual outputs in y must be columns of same length as t')
end

if N~=Ru
   error('The individual inputs in u must be columns of same length as t')
end

if nargin==5;  plottype = 2;  weight = ones(1,Cy);  end;
if nargin==6;  weight = ones(1,Cy);  end;
```

```
for ii=1:Cy
   NORMMATR(ii,ii) = 1/norm(y(:,ii))*weight(ii);
end

yn = y*NORMMATR;  yr = yn(:);   NORMMATR = NORMMATR*sqrt(NM)/norm(yr);
yn = y*NORMMATR;  yr = yn(:);

% Iteration algorithme:
% In an outer loop new relative parameter update directions, updd(parny)
% are calculated from a new relative model gradient psir(parny) -
% untill norm(updd)<=upddmin.
% In an inner loop the step length mu is reduced, but the direction maintained,
% untill the new errn  is smaller.
% The iteration is stopped if mu < mumin

ym = feval(simmod,u,t,par);   [Rym,Cym] = size(ym);
if N~=Rym;  error('Wrong dimension of model output');  end;
ymn = ym*NORMMATR;  ymr = ymn(:);  errn = 100*norm(yr-ymr)/sqrt(NM);

if Cy==1
   nn = 1:N;
   plot(nn,y,nn,ym);
   title('System and model output:  Start '),  pause(5)
else
   if plottype == 1
      nn = 1:N;
      plot(nn,yn,nn,ymn,':');
   else
      nn = 1:NM;
      plot(nn,yr,nn,ymr,':');
   end
   title('Normed system and model output: Start'),  pause(5)
end

mu = 0.125;  noo = 1;
updd = 1;

% Outer loop:
while norm(updd) > upddmin

  tekst = num2str(noo);
  tekst = [tekst ,'. calculation of psi of max ',num2str(noomax),' times'];
  disp(tekst);    psi = psinf(u,t,par,simmod,NORMMATR);
  psir = psi*diag(par);

  mu = 4*mu;  if mu>1,  mu=1;  end
  updd = psir\(yr-ymr);        % updd = Rr^-1*Gr = [psir'*psir]^-1*[e*psir]
  disp(' ');
  disp('Relative parameter update = ');  disp(updd);
  nyerrn = errn;               % Guarantees at least one  run through inner loop.

     % Inner loop:
```

```
    while nyerrn >= errn
      parny = par + par.*mu.*updd;
      ym = feval(simmod,u,t,parny);  ymn = ym*NORMMATR;  ymr = ymn(:);
      nyerrn = 100*norm(yr-ymr)/sqrt(NM);
      mu = mu/2;
      if mu < mumin
        disp('iteration stopped - mu < mumin'), break
      end
    end                          % End of inner loop.

  if mu < mumin,  break, end
  par = parny;   errn = nyerrn;

  % Plotting:
  figure(1)
  if Cy==1
     plot(nn,y,nn,ym);      title('System and model output');
     ylabel('y(k) and ym(k)');
  else
     if plottype == 1
        plot(nn,yn ,nn, ymn,':');  ylabel('yn(k) and ymn(k)');
     else
        plot(nn,yr,nn,ymr,':');    ylabel('yr(k) and ymr(k)');
     end
     title('Normed system and model output')
  end

  xlabel('Sampling number');

  ny = round(log10(abs(errn)));                 % Rounding errn
  aerrn = round(errn*10^(2-ny))/10^(2-ny);      %    -      -

  L = axis;    axis('auto');
  xpkt = 0.14*(L(2) - L(1)) + L(1);
  ypkt = 0.90*(L(4) - L(3)) + L(3);
  text(xpkt,ypkt,['errn = ',num2str(aerrn),' %']);  pause(2)

  disp(['Percentage errn = ',num2str(errn)]);
  disp(' ');
  disp('* - * - * - * - * - * - * - *');


  noo = noo+1;  if noo > noomax,  break,  end
end                                                 % End of outer loop.

% print gausnewt                                    % Last plot saved

Hrn = psir'*psir/Ry/Cy;

if isempty(errn);  errn=100;  end;
if isempty(Hrn);  Hrn = zeros(length(par0),length(par0));  end;
```

**mainest:**

```
% mainest is the main program for parameter estimation
% From input/output data (u, y and t) stored in meas'process','no'.mat,
% the parameters pare of 'process' are estimated.
% The sensitivity measures and the parameter deviations are calculated as well.
%
% Must be specified (e.g. by progprog'process'.m, i.e. from
% progdata'process'.mat): see default values below.
% Subprograms used:
% sim'process', gausnewt, psinf, and sens.
%
% 20/9-94,MK. 26/11-02,MK
% Default values:
if ~exist('process'), process='ktau';    end      % Process name
if ~exist('no'), no=''; end                       % Measurement number
if exist(['progdata',process,no,'.mat'])==2 & ~exist('par')
   load(['progdata',process,no]), end             % progdata loades
if exist(['meas',process,no,'.mat'])==2, load(['meas',process,no]), else
   disp(['data: meas',process,no,'.mat missing !']), break, end
if ~exist('ploty'), ploty=2; end
if ~exist('par0'), par0=[1.5 3]; end
simmod=['sim',process];
if ploty>0, plot(t,y), ylabel('y'), xlabel('time'),
 title('Measured system output'), pause(4), end
[pare,errn,Hrn]=gausnewt(simmod,par0,u,y,t,ploty);      % Parameter estimation
[Smin,Simin,R,Ri]=sens(Hrn);                           % Sensitivity measures
[Ry,Cy]=size(y);
dpar=errn./Simin;
sigpar=dpar/sqrt(Ry*Cy);                               % Parameter deviations
pare, errn
```

**maininp:**

```
% maininp is the main program for design of optimal input in the time domain.
% The system is simulated with different inputs u described by one input
% parameter (eg. fundamental frequency or amplitude of a square wave),
% the sensitivity measures are calculated and plotted versus the input parameter
% and the input  parameter value minimizing R is determined.
%
% Must be specified (e.g. by progprog'process' or from progdata'process'.mat):
% Process name, process=' '
% Input signal type, inputt=' '
% Parameter vector, par=[ ]
% Time step, h=   . Input sequence length,  nu=
% Plot of output y(t), yplot=      (>0 => plot)
% Input frequency vector, f1v=[ ] and amplitude amv=   ... or ...
% Input frequency, f1v=  and amplitude vector, amv=[ ]
% Weigh matrix for multiple outputs, W=[]
%
% Subprograms used:
```

```
% sim'process' (eg. simktau or simdcmotm), 'input' (eg.inputsq or inpsqramp),
%  psinf, sens and sensplot.
%
% 15/1-03,MK

% Default values:
if ~exist('process'),    process='ktau';                                    end
if exist(['progdatainp',process,'.mat'])==2  &  ~exist('par'),
   load(['progdatainp',process]),  end
% else  disp(['data: progdatainp',process,'.mat  missing !']),   break,    end
if ~exist('inputt'),    inputt='inpsqw';                                   end
if ~exist('par'),       par=[1 2];                                          end
if ~exist('h'),         h=par(2)/20;                                        end
if ~exist('nu'),        nu=1000;                                            end
if ~exist('ploty'),     ploty=2;                                            end
if ~exist('f1v'),       f1v=[1 2 3 5 7 10 20]/par(2)/50;                    end
if ~exist('amv'),       amv=1;                                              end
if ~exist('W'),         W=diag(ones(length(par)));                          end
simmod=['sim',process];

% Sensitivity measures:
j=1;  lenf=length(f1v);
if lenf>1, xvector=f1v; f1=f1v;  am=amv*ones(size(f1v));
   else    xvector=amv; am=amv;  f1=f1v*ones(size(amv));
end
for j=1:length(xvector)
  disp(['j = ',int2str(j)])
  [u,t]=feval(inputt,f1(j),am(j),nu,h);
  if ploty>0,
     y=feval(simmod,u,t,par); subplot, plot(t,y), figure(1)
     title(['Model output nr  ',int2str(j)]),pause(2)
  end
  [psi,Hrn]=psinf(u,t,par,simmod);
  [Smin(j),Simin(:,j),R(j),Ri(:,j),D(j)]=sens(Hrn,W);
end

% Optimal frequency or amplitude:
jmin=find(R==min(R));
if lenf>1 f1opt=f1v(jmin), else amopt=amv(jmin), end
disp(['Opt. values: Smin = ',num2str(Smin(jmin)),'  R = ',num2str(R(jmin))])

% Plotning of sensitivity measures:
if lenf>1,   xlab='input frequency';
   else,     xlab='input amplitude';
end
sensplot, pause(4)

[u,t]=feval(inputt,f1(jmin),am(jmin),nu,h);
if ploty>1
   y=feval(simmod,u,t,par); [Ry,Cy]=size(y);
   figure(2); clf,   % subplot(221),
   for ii=1:Cy, NM(ii,ii)=1/norm(y(:,ii));      end, yn=y*NM;
```

```
   if Cy>1,  plot(t,yn),  else plot(t,y),       end
   set(gca,'fontname','times'); set(gca,'fontsize',16)
   xlabel('time'), title('Optimal output')
   if Cy>1, ylabel('ymn'), else  ylabel('ym'),  end
   end
```

**psinf:**

```
function  [psi,Hrn] = psinf(u,t,par,simmod,NORMMATR)
%  psi = psinf(u,t,par,simmod,NORMMATR) calculates the model gradient
%  psi = d(ymr)/d(par)  wrt the n parameters in the parameter vector par
%  using numerical differentation.
%  u is the inputmatrix/vector and t the time vektor.
%  simmod.m is the simulation program calculating the model output ym
%  The output ymn used for calculating psi is normed by NORMMATR=diag(1/yiRMS).
%  If argument NORMMATR is omitted a NORMMATR=diag(1/ymiRMS) is used as default.
%  The relative normed Hessian Hrn is calculated as well.
%
%  30/8 - 1994 JGJ+MK. 26/11-02,MK

n    = length(par);
ymn0 = feval(simmod,u,t,par);   [Ry,Cy]=size(ymn0);
if  Cy>Ry,  ymn0=ymn0';         [Ry,Cy]=size(ymn0);                end
if nargin==4
   for i=1:Cy,  NORMMATR(i,i)=sqrt(Ry*Cy)/norm(ymn0(:,i));     end
end
dpar = par*1e-4;                              % 0.01 per cent delta values


ymn0 = ymn0*NORMMATR;     ymn0 = ymn0(:);

for ii=1:n
 % disp(['calculating  dym/dpar(',int2str(ii),') ...'])
  parii = par;
  parii(ii) = par(ii) + dpar(ii);
  ymn = feval(simmod,u,t,parii)*NORMMATR;
  psi(:,ii) = (ymn(:) - ymn0)/dpar(ii);                % dymn/dpar(ii)
end

Hn=psi'*psi/Ry/Cy;
L=diag(par);  Hrn=L*Hn*L;
```

**sens:**

```
function  [Smin,Simin,R,Ri,D] = sens(Hrn,W)
% [Smin,Simin,R,Ri,D] = sens(Hrn) calculates the sensitivity measures for Hrn,
% the relative, normed Hessian:
% Smin - min sens in any direction
% Simin - min sens of thi
% R (Smax/Smin) - ratio of major half axis to minor
% Ri=Si/Simin - ratio of sens of thi alone and min sens of thi
```

```
% D= det(Hrn) - ~the elepsoid volume
% W=[1 k1..kn] is a vector, weighting the sensitivity of the i'th
% parameter with a factor ki.
%  30/8-94,MK. 26/11-02,MK

if nargin==2, W=diag(W); Hrn=W*Hrn*W;  end
[xe,de]=eig(Hrn);  lam=diag(de)';
Smin=sqrt(min(lam));
Simin=sqrt(diag(inv(Hrn)).^(-1));
R=sqrt(max(lam)/min(lam));
Ri=sqrt(diag(Hrn))./Simin;
D=det(Hrn);
```

**inpsqramp:**

```
function [u,t]=inpsqramp(f1,am,nu,h);
% [u,t]=inpsqramp(f1,am,nu,h); generates a square wave signal -
% superposed by a rampe - with fundamental frequency  f1,
% final amplitude am, and length nu
%
% 17/1-03,MK.

np=f1*nu*h;                                     % Number of periods
t=[1:nu]*h;
u=am*sign(sin(np*2*pi*[1:nu]/nu))';  u(1:2)=[0 0]';
u=u.*[1:nu]'/nu;                                % * ramp with ampl 1
```

**simdcml:**

```
function y=simdcml(u,t,par)
% y=[i,w]=simdcml(u,t,par) simulates a  linear dc-motor with
% input u and outputs i and w.
% w/u = K/R/(J*s+B+K^2/R),  i/u=(J*s+B)/R/(J*s+B+K^2/R)
% par=[R K J B]
%
% 27/11-02,MK

h=t(2)-t(1);  par=par(:)';
t=[0 t(1:length(t)-1)];
nw=par(2)/par(1); dw=[par(3) par(4)+par(2)^2/par(1)];
ni=[par(3) par(4)]/par(1);
w = lsim(nw,dw,u,t);
i = lsim(ni,dw,u,t);
y=[i w];
```

# Appendix B

# Problems

*The first digit of the problem number is the corresponding chapter number.*

*If you have problems with the problems, you may find help in the hints at the end of the appendix.*

**Problem 2.1**

Fig. 2.1 below shows the response of a system to an input step with magnitude 2. The response data are stored in prob21data.mat in Senstools.

a) Determine by a graphical method the steady-state gain and time constant for a first order model

b) Control if the first order model structure is adequate by plotting the model step response together with the system response.

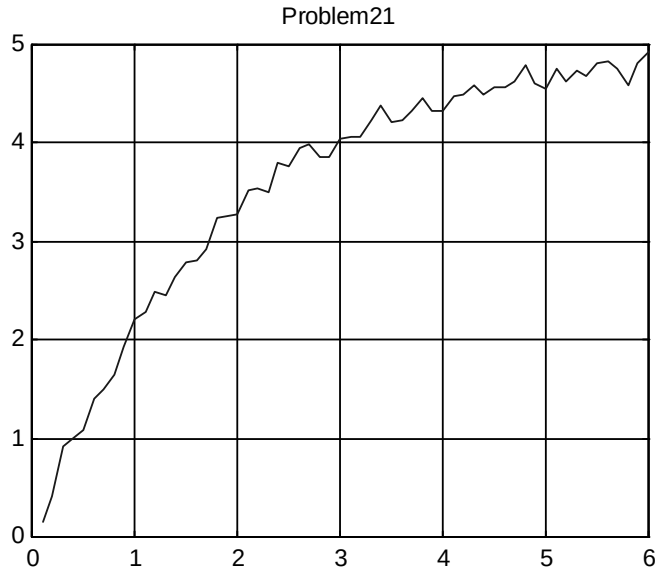


Figure 2.1 Response to a step of magnitude 2

**Problem 2.2**

Determine by graphical methods the gain, time constant and time delay for a first order model from

a) The unit step response below

b) The Bode plot below

and control if the model structure is adequate by plotting the model response together with the system response. The data prob22datan.mat are in Senstools.

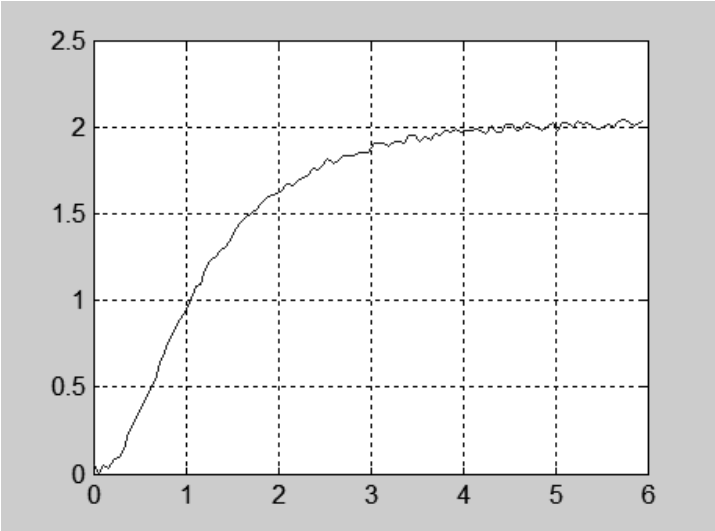


Figure 2.2a Response to unit step for sys22, with noise

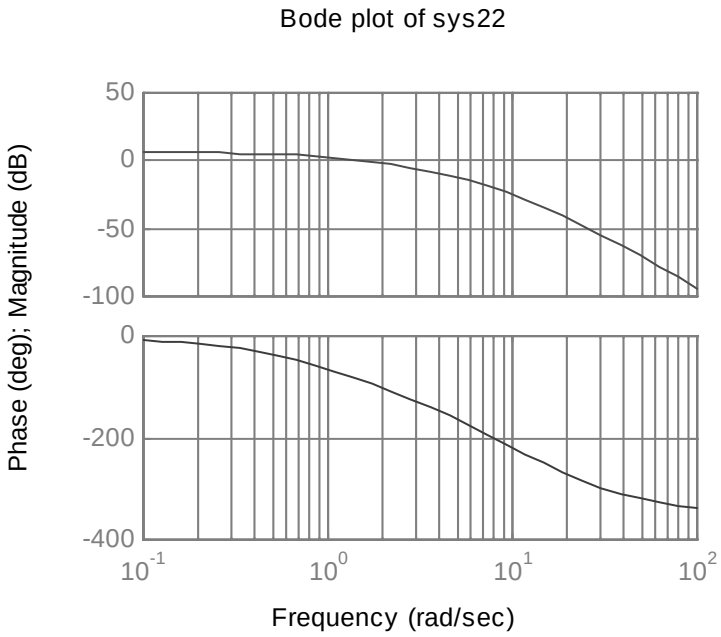


Figure 2.2b Bode plot for sys22

**Problem 3.1**

In this problem you shall try to fit two parameters, gain k and time constant tau, of a first order model to 'measured' system data.

Run manest in Matlab (manest.m and the data measmanest.mat are in Sentools).

Update the parameters until best fit according to the graph and displayed relative RMS-error. (This should motivate you to use a numeric minimization program in the future).

**Problem 3.2**

Determine the value of x minimizing v = 100/x + 0.0001x$^3$ analytically, and compare the result with the iterative solution in Example 1.

**Problem 4.1**

For the second order transfer function

$$H(s) = \frac{K}{s^2 + 2\,\zeta\,\omega\,s + \omega^2} \tag{B.1}$$

where $K = 40,\ \zeta = 0.6,\ \omega = 2$

a) determine the corresponding Z-transform using c2d (Matlab) with ZOH and Tustin. Comment on the results.

b) Determine the stationary gain of the obtained Z-transforms.

c) Plot the step response.

**Problem 4.2**

Make a simulation program for the model (B.1) of problem 4.1 in form of a Matlab function y=simkzetaw(u,t,par) for use with Senstools.

**Problem 4.3**

Make a simulation program for a nonlinear system with saturation as the one in Fig. 4.6 (Chapter 4), but with the linear dynamics equal to (B.1) of problem 4.1.

**Problem 4.4**

DC-motor simulation, linear and nonlinear.

A DC-motor, Axem F9M2, has the following data according to the data sheet

| | |
|---|---|
| motor constant: | K = 0.030 [V/rad/sec] or [Nm/A] |
| moment of inertia for motor: | J = 8 $10^{-5}$ [kg m2] |
| viscous friction: | B = 5 $10^{-5}$ [Nm/rad/s] |
| armature resistance: | R = 1.2 [ohm] |

On the motor shaft is mounted a circular aluminum disc with radius r=0.1 [m] and thickness d=0.002 [m].

**a)** Develop a linear physical model (block diagram) with input voltage u and the outputs, velocity $\omega$ and current i.

Calculate the gain $K_M$ and the time constant $\tau_M$ for the transfer function $\frac{\Omega(s)}{U(s)} = \frac{K_M}{1+s\tau_M}$

**b)** Make a simulation program for the linear motor as a Matlab Function:

y = simdcml(u,t,par)

where y=[i, $\omega$] is the output vector, t is the time vector and par is the parameter vector.

Determine and plot the response to a step input (e.g. inpstep.m with amplitude am=10, nu=1000 ans h=0.01).

Verify the final value and time constant for $\omega$, and the initial value of i.

**c**) In the motor model a nonlinear armature conductance is now introduced (see block diagram Fig. 4.4):

$i = G_0\ u_R$ for abs $(u_R) \leq u_1$

$i = G_0\ u_R + (G_1 - G_0)\ (u_R - u_1\ \text{sign}(u_R))$ for abs $(u_R) > u_1$

where:

$u_1 = 3.0$ [V] $G_0 = 0.5$ [ohm$^{-1}$] $G_1 = 1.6$ [ohm$^{-1}$]

Simulate the total motor with nonlinear armature conductance using the staircase input (e.g. inpstair(4,12,2000,0.01)), and plot the responses $\omega$ and i.

Verify how $K_M$ and $\tau_M$ depend on the input amplitude.

**d**) A nonlinear Coulomb friction is now introduced as well. The coulomb friction coefficient is Tc = 0.025 [Nm]

The total nonlinear block diagram is shown in Fig. 4.4.

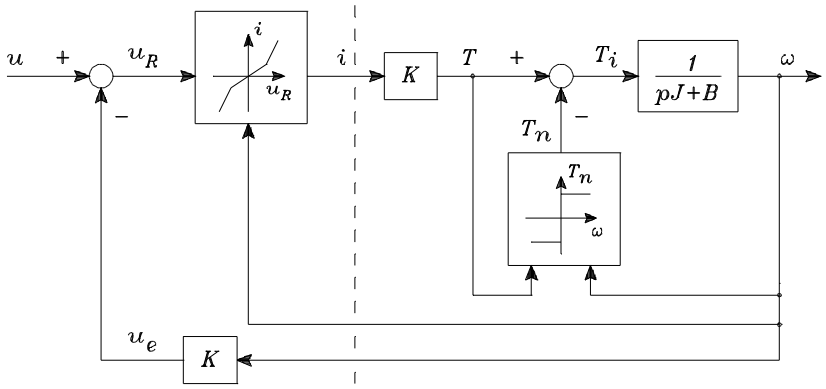


Fig.4.4 Nonlinear block diagram for DC-motor. The dotted line divides the model in the electrical part (left) and the mechanical part (right).

Make a simulation program for the nonlinear motor as a Matlab Function: y = simdcmn(u,t,par)

Determine and plot the response to a staircase input. Compare to c) and comment.

**Problem 5.1**

In Problem 3.1 you fitted the parameters manually. In this problem you shall use Senstools to fit the two parameters, gain k and time constant tau of a first order model to 'measured' system data, measmanest.mat (in Senstools).

Make a simulation program, y=simmanest(u,t,par) using 'foh', and apply the method in Example 9a (Chapter 5) to estimate k and tau.

Notice how the optimal parameter estimates are found in 2-3 iterations.

**Problem 5.2**

The parameters of the DC-motor in Problem 4.4 shall now be estimated from real measurements of input voltage, tachometer voltage and current. The tachometer constant is kt = 0.03 V/(rad/s) and the current is measured as the voltage over a 0.217 Ω resistor.

Get data already obtained from a step input experiment on the motor by running the data m-file mk_14_11_29.m in Senstools.

**a)** Determine a linear first order SISO-model with two parameters:

$$\frac{\Omega(s)}{V(s)} = \frac{K_1}{1 + s\tau} \tag{B.2}$$

from the measurements of input voltage, volt and velocity, tacho, going through the following procedure:

Make a simulation program simdcml1.m or get it from m-files.

Run the data m-file (eg. mk_14_11_29.m ) giving the samples of volt, tach and strom, (sample frequency 50 Hz) and check them by:

```
>> mk_14_11_29
>> whos
>> t=(1:length(volt))/50;
>> plot(t,volt,t,strom,t,tach)
```

Make a data mat-file:

```
>> y=tach(:)/0.03;
>> u=volt(:);
>> save measdcml1 u y t
```

and estimate the parameters $K_1$ and $\tau$ by:

```
>> process = 'dcml1';
>> par0=[31 1];   % Values from Problem 4.4
>> mainest
```

Calculate the relative deviation of the estimated parameter values with those from Problem 4.4. Comment on the fit and the results.

**b)** Determine a linear first order SIMO-model with 4 parameters, R, K, Itotal, B (cf. Problem 4.4) from measurements volt, tacho and strom (4 parameters require 2 outputs, y = [i $\omega$]). Use the simulation program developed in Problem 4.4 or simdcml2.m in problem solutions. Use a procedure similar to the one for a).

**Problem 5.3**

As in problem 5.2 a) only the measurements of input voltage and output velocity for the DC-motor are available (stored in measdcmlRB1.mat).

a) How many of the physical parameters can you estimate?

You decide to estimate the armature resistance R and viscous friction coefficient B, and use data sheet values for motor constant and moment of inertia: K=0.03 and J=6 $10^{-4}$.

b) Make a simulation program y = simdcmlRB1(u,t,par) where y is the velocity and par=[R B], e.g. by modifying simdcml.m

c) Estimate R and B, using the data measdcmlRB1, and compare the results with those obtained in problem 5.2.

**Problem 6.1**

In problem 5.2 b) print out dpar and sigpar. Calculate the sensitivity measures:

[Smin,Simin,R] = sens(Hrn) and comment

**Problem 8.1**

Determine an optimal input square wave signal for the second order system of Problem 4.1 (B.1) with the Senstool program maininp.

Use the simulation program from Problem 4.2, y=simkzetaw(u,t,par).

**Problem 10.1**

Estimate the gain, time constant and time delay for a first order model from the step response in problem 2.2 stored in prob22data.mat. Plot the model response together with the system response and comment on the fit.

**Hints**

**Re P4.1:** Stationary gain for z=1. Can be implemented in Matlab as:

```
[nz,dz]=tfdata(sysd,'v')        % NB 'v' for vector format - not cell
K=sum(nz)/sum(dz)               % DC-gain: set z=1
```

**Re P4.4:** A circular aluminum disc with radius and thickness d has the moment of inertia:

$$I_{disc} = M\frac{r^2}{2} = \frac{1}{2}\rho_{al}\pi r^4 d \qquad \text{where} \quad \rho_{al} = 2.5 \cdot 10^3 [kg/m^3] \tag{B.3}$$

**% Nonlinear friction:**

```
Ti = Tm - T0*sign(w);      % Correct if motor is moving
wny = c*x + d*Ti;          % Simulation of lin dyn part
x = a*x + b*Ti;
if (sign(wny)~=sign(w))    % If these conditions are fulfilled
if (abs(Tm)<=T0)           % the motor remains stopped
wny = 0; x = 0;
end
```

# Appendix C

# Proof of Theorem 1

Determination of the minimum sensitivity of $\theta_i$ (6.20) is equivalent to determination of maximum values of an ellipsoid in the directions of the axes.

The maximum value of $\theta_i$ on the ellipsoid

$$F(\theta) = \theta^T R\theta = c^2 \tag{C.1}$$

is

$$\theta_{i,\max} = \max \mid \theta_i \mid = c\sqrt{\frac{R_{ii}}{\det R}} = c\sqrt{(R^{-1})_{ii}} \tag{C.2}$$

R is symmetric (dxd) and positive definite

**Proof.** ■

$$E_c = \{\theta \in R^d \mid \theta^T R\theta = c^2\} \qquad c > 0 \tag{C.3}$$

$\theta_0 \in E_c$; $\tau_0$ is a tangent plane of $E_c$ in $\theta_0$ with the unit normal vector $n_0$.
The distance from origo to $\tau_0$ is (projection of $\theta_0$ on $n_0$)

$$\delta_0 = \mid n_0^T \theta_0 \mid \tag{C.4}$$

The gradient of F in $\theta_0$ is in the direction of $n_0$

$$\nabla F(\theta_0) = 2R\theta_0 = k_0 n_0 \tag{C.5}$$

where $k_0$ is a constant, and accordingly

$$\theta_0 = \frac{1}{2} k_0 R^{-1} n_0 \tag{C.6}$$

(C.6)is next inserted in (C.1)

$$F(\theta_0) = \frac{1}{4} k_0^2 n_0^T R^{-1} n_0 = c^2 \tag{C.7}$$

and in (C.4) squared

$$\delta_0^2 = \frac{1}{4} k_0^2 \mid n_0^T R^{-1} n_0 \mid = c^2 n_0^T R^{-1} n_0 \tag{C.8}$$

This an important general result.

In particular, in the direction of the i'th axis (in $R^d$), the normal vector is

$$n_{0i} = \{0...0\ 1\ 0...0\}^T \tag{C.9}$$

i.e. the i'th element is 1 and the remaining 0. Then from (C.8) we obtain the maximum value of $\theta_i$ on $E_c$

$$\theta_{i,\max} = \delta_{0i} = c\sqrt{(R^{-1})_{ii}} \tag{C.10}$$

where $(R^{-1})_{ii}$ is the i'th diagonal element of $R^{-1}$.

# Appendix D

# List of symbols used

| | |
|---|---|
| $d$ | Number of parameters in a model (dim of param vector $\theta$) |
| $errn = \frac{1}{y_{RMS}}\sqrt{\sum_{k=1}^{N}\varepsilon^2(k,\theta_N)} \cdot 100$ | Normed root mean square output error in % |
| $G(\theta) = \frac{\partial P(\theta)}{\partial \theta}$ | Gradient vector (dimension d) |
| $H(\theta) = \frac{\partial^2 P(\theta)}{\partial \theta \partial \theta^T}$ | Hessian matrix (d×d) |
| $L = diag(\theta_N)$ | d×d matrix with $\theta_N$ as the diagonal and zeroes outside |
| $N$ | Number of samples or data points in signal. |
| $P(\theta) = \frac{1}{2N}\sum_{k=1}^{N}\varepsilon^2(kT,\theta)$ | Performance function |
| $S_i = \frac{\partial \varepsilon_{p,RMSn}}{\partial \theta_{ri}} = \sqrt{h_{rnii}}$ | Sensitivity w.r.t one relative parameter $\theta_{ri}$ |
| $S_{i\min} = \sqrt{\{H_{rn}^{-1}(\theta_N)\}_{ii}^{-1}}$ | Minimum sensitivity w.r.t one relative parameter $\theta_{ri}$ |
| $R = S_{\max}/S_{\min}$ | Sensitivity measure; $R >> 1$ indicates poor input signal |
| $R_i = S_i/S_{i\min}$ | Sensitivity measure; $R_i >> 1$ indicates param correlation |
| $\Delta_{\theta eq,i} = \frac{\varepsilon_{m,RMSn}}{S_{i\min}}$ | Parameter error caused by undermodelling |
| $h_{rnii} = \{H_{rn}(\theta_N\}_{ii}$ | i-th diagonal element of relative normed Hessian |
| $u(t)$ | Input signal (continuos time) |
| $u(k) = u(kT)$ | Discrete time system input, sampl time T and N samples |
| $y(k) = y(kT)$ | Discrete time system output, sampl time T and N samples |
| $y_m(k,\theta) = y_m(kT,\theta)$ | Discrete time model output, sampl time T and N samples |
| $\varepsilon(k,\theta) = y(k) - y_m(k,\theta)$ | Discrete time model error |
| $\varepsilon_p(k)$ | Model error caused by $\theta \neq \theta_N$ |
| $\varepsilon_x(k)$ | Model error caused by noise |
| $\varepsilon_m(k)$ | Model error caused by undermodelling |
| $\theta = [\theta_1\ \theta_2\ ....\ \theta_d]^T$ | Parameter vector containing d parameters. T= transposed. |
| $\theta_0$ or $\theta_N$ | Parameter values minimizing $P(\theta)$. |
| $\theta_r = L^{-1}\theta$ | Relative parameter vector |
| $\widetilde{\theta}_r = \theta_r - 1_v$ | Relative parameter vector deviation from $\theta_0$ |
| $1_v = \{11...1\}^T$ | Unit d-vector |
| $\psi(k,\theta) = \frac{\partial y_m(k,\theta)}{\partial \theta}$ | Model gradient (d×N matrix) |
| $\sigma_{r,\theta i} = \frac{\varepsilon_{x,RMSn}}{S_{i\min}}\frac{1}{\sqrt{N}}$ | Parameter spread caused by noise |
| $Y_{RMS} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}y^2(k)}$ | Root mean square output |
| $y_{mm}(k) = \frac{1}{y_{RMS}}y_m(k)$ | Normed model output |

# Bibliography


[Blanke 1998] Blanke M. and Knudsen M., A Sensitivity Approach to Identification of Ship Dynamics from Sea Trial Data. Ifac Conference CAMS'98: Control Applications in Marine Systems. Fokuoka Japan. 1998.

[Blanke 1999] Blanke M. and Knudsen M., Optimized Experiment Design for Identification of Marine Systems. 14th Ifac World Congress; Beijing China. 1999.

[Børsting 1994] Børsting H. , Knudsen M, Rasmussen H. and Vadstrup P., Estimation of physical parameters in induction motors. 10th IFAC Symposium on System Identification, Copenhagen. 1994.

[Eykhoff 1974] Eykhoff P., System Identification. Parameter and State Estimation. John Wiley 1974.

[Franklin 1994] Franklin G., Powell J.D. and Emami-Naeni A., Feedback Control of Dynamic Systems, Addison-Wesley. 1994.

[Franklin 1998] Franklin G., Powell J.D. and Workman M., Digital Control of Dynamic Systems. Addison-Wesley. 1998.

[Goodwin 1977] Goodwin G. and Payne R., Dynamic System Identification: Experiment Design and Data Analysis. Academic Press 1977.

[Knudsen 1986] Knudsen M. and Overgaard J., Identification of Thermal Model for Human Tissue. IEEE Trans. Biomedical Engineering, 1986.

[Knudsen 1993] Knudsen M., Estimation of Physical Parameters in Linear and Nonlinear Dynamic Systems (PhD-thesis). Aalborg University, Department of Control Engineering, 1993.

[Knudsen 1994] Knudsen M., A Sensitivity Approach for Estimation of Physical Parameters, 10th IFAC Symp on System Identification, Copenhagen 1994.

[Knudsen 1995] Knudsen M. and Grue Jensen, J.,Estimation of nonlinear DC-motor models using a sensitivity approach. Third European Control Conf, ECC'95. Rome 1995

[Knudsen 1996] Knudsen M. , Loudspeaker modelling and parameter estimation. 100th AES Convention, Audio Engineering Society Preprint, Copenhagen 1996.

[Knudsen T. 1993] Knudsen T., Systemidentifikation. AUC-Control-U93-4008. 1993. (In Danish).

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

[Schoukens 1991] Schoukens J. and Pintelon R., Identification of Linear Systems, Pergamon Press, 1991.

[Söderström 1989] T.Söderström, System Identification, Prentice Hall International, 1989.

[Unbehauen 1987] Unbehauen H. and Rao G.P., Identification of Continous Systems. North-Holland, 1987.