

Advanced DYNAMICS and CONTROL

Finn Haugen
TechTeach

August 2010

ISBN 978-82-91748-17-7

Contents

I	CONTINUOUS-TIME SYSTEMS THEORY	13
1	State-space models	15
1.1	Introduction	15
1.2	A general state-space model	15
1.3	Linear state-space models	18
1.4	Linearization of non-linear models	19
1.4.1	Introduction	19
1.4.2	Deriving the linearization formulas	21
2	Frequency response	25
2.1	Introduction	25
2.2	How to calculate frequency response from sinusoidal input and output	26
2.3	How to calculate frequency response from transfer functions .	28
2.4	Application of frequency response: Signal filters	33
2.4.1	Introduction	33
2.4.2	First order lowpass filters	34
3	Frequency response analysis of feedback control systems	39

3.1	Introduction	39
3.2	Definition of setpoint tracking and disturbance compensation	40
3.3	Definition of characteristic transfer functions	41
3.3.1	The Sensitivity transfer function	41
3.3.2	The Tracking transfer function	44
3.4	Frequency response analysis of setpoint tracking and disturbance compensation	45
3.4.1	Introduction	45
3.4.2	Frequency response analysis of setpoint tracking . . .	45
3.4.3	Frequency response analysis of disturbance compensation	50
4	Stability analysis of dynamic systems	55
4.1	Introduction	55
4.2	Stability properties and impulse response	56
4.3	Stability properties and poles	58
4.4	Stability properties of state-space models	64
5	Stability analysis of feedback systems	67
5.1	Introduction	67
5.2	Pole-based stability analysis of feedback systems	68
5.3	Nyquist's stability criterion	70
5.4	Stability margins	77
5.4.1	Stability margins in terms of gain margin and phase margin	77
5.4.2	Stability margins in terms of maximum sensitivity amplitude	79

	5
5.5	Stability analysis in a Bode diagram 79
5.6	Robustness in term of stability margins 82
II	DISCRETE-TIME SYSTEMS THEORY 87
6	Discrete-time signals 89
7	Difference equations 91
7.1	Difference equation models 91
7.2	Calculating responses from difference equation models 92
8	Discretizing continuous-time models 93
8.1	Simple discretization methods 93
8.2	Discretizing a simulator of a dynamic system 95
8.3	Discretizing a signal filter 97
8.4	Discretizing a PID controller 99
8.4.1	Computer based control loop 99
8.4.2	Development of discrete-time PID controller 100
8.4.3	Some practical features of the PID controller 102
8.4.4	Selecting the sampling time of the control system . . . 103
9	Discrete-time state space models 105
9.1	General form of discrete-time state space models 105
9.2	Linear discrete-time state space models 106
9.3	Discretization of continuous-time state space models 106
9.3.1	Discretization of non-linear continuous-time state-space models 106

9.3.2	Discretization of linear continuous-time state-space models	107
10	The z-transform	111
10.1	Definition of the z -transform	111
10.2	Properties of the z -transform	112
10.3	z -transform pairs	112
10.4	Inverse z -transform	113
11	Discrete-time (or z-) transfer functions	115
11.1	Introduction	115
11.2	From difference equation to transfer function	115
11.3	From transfer function to difference equation	116
11.4	Calculating time responses for discrete-time transfer functions	117
11.5	Static transfer function and static response	118
11.6	Poles and zeros	119
11.7	From s -transfer functions to z -transfer functions	119
12	Frequency response of discrete-time systems	123
13	Stability analysis of discrete-time dynamic systems	129
13.1	Definition of stability properties	129
13.2	Stability analysis of transfer function models	130
13.3	Stability analysis of state space models	133
14	Analysis of discrete-time feedback systems	137

III STOCHASTIC SIGNALS 143

15 Stochastic signals 145

15.1 Introduction	145
15.2 How to characterize stochastic signals	145
15.2.1 Realizations of stochastic processes	145
15.2.2 Probability distribution of a stochastic variable	146
15.2.3 The expectation value and the mean value	147
15.2.4 Variance. Standard deviation	148
15.2.5 Auto-covariance. Cross-covariance	149
15.3 White and coloured noise	151
15.3.1 White noise	151
15.3.2 Coloured noise	153
15.4 Propagation of mean value and co-variance through static systems	155

IV ESTIMATION OF PARAMETERS AND STATES 159

16 Estimation of model parameters 161

16.1 Introduction	161
16.2 Parameter estimation of static models with the Least squares (LS) method	162
16.2.1 The standard regression model	162
16.2.2 The LS problem	164
16.2.3 The LS solution	165
16.2.4 Criterion for convergence of estimate towards the true value	168

16.2.5	How to compare and select among several candidates?	168
16.3	Parameter estimation of dynamic models	169
16.3.1	Introduction	169
16.3.2	Good excitation is necessary!	170
16.3.3	How to check that a model is good?	172
16.3.4	Estimation of differential equation models using the LS-method	174
16.3.5	Estimation of black-box models using subspace methods	175
17	State estimation with observers	185
17.1	Introduction	185
17.2	How the observer works	187
17.3	How to design observers	188
17.3.1	Deriving the estimation error model	188
17.3.2	Calculation of the observer gain	191
17.4	Observability test of continuous-time systems	197
17.5	Discrete-time implementation of the observer	200
17.6	Estimating parameters and disturbances with observers	201
17.7	Using observer estimates in controllers	207
17.8	Using observer for increased robustness of feedback control at sensor failure	210
18	State estimation with Kalman Filter	215
18.1	Introduction	215
18.2	Observability of discrete-time systems	216
18.3	The Kalman Filter algorithm	218

18.3.1	The basic Kalman Filter algorithm	218
18.3.2	Practical issues	227
18.3.3	Features of the Kalman Filter	229
18.4	Tuning the Kalman Filter	230
18.5	Estimating parameters and disturbances with Kalman Filter .	231
18.6	Using the Kalman Filter estimates in controllers	239
18.7	Using the Kalman Filter for increased robustness of feedback control at sensor failure	239
V	MODEL-BASED CONTROL	241
19	Testing robustness of model-based control systems with sim- ulators	243
20	Feedback linearization	245
20.1	Introduction	245
20.2	Deriving the control function	246
20.2.1	Case 1: All state variables are controlled	246
20.2.2	Case 2: Not all state variables are controlled	251
21	LQ (Linear Quadratic) optimal control	257
21.1	Introduction	257
21.2	The basic LQ controller	258
21.3	LQ controller with integral action	268
21.3.1	Introduction	268
21.3.2	Including integrators in the controller	269
21.3.3	Discrete-time implementation of the LQ controller . .	271

22 Model-based predictive control (MPC)	273
23 Dead-time compensator (Smith predictor)	285
A Model-based PID tuning with Skogestad's method	289
A.1 The principle of Skogestad's method	289
A.2 The tuning formulas in Skogestad's method	292
A.3 How to find model parameters from experiments	292
A.4 Transformation from serial to parallel PID settings	293
A.5 When the process has no time-delay	294

Preface

This book covers estimation of model parameters and states and model-based control, and the systems theory required. I have selected the topics among many possible topics by considering what I assume are the most relevant topics in an application oriented course about these topics.

It is assumed that you – the reader – has basic knowledge about complex number, differential equations, and Laplace transform based transfer functions (s -transfer functions), and also that you have knowledge about basic control methods, i.e. PID control, feedforward control and cascade control. A reference for these topics is my book *Basic Dynamics and Control* [5].

Supplementary material, as tutorials, ready-to-run simulators, and instructional videos are available at techteach.no.

This book is available for sale only from the web site <http://techteach.no>.

It is not allowed to make copies of the book.

If you want to know about my background, please visit my home page <http://techteach.no/adm/fh>.

Finn Haugen, MSc

TechTeach

Skien, Norway, August 2010

Part I

**CONTINUOUS-TIME
SYSTEMS THEORY**

Chapter 1

State-space models

1.1 Introduction

A *state-space model* is just a structured form or representation of the differential equations for a system.

State-space models are useful in many situations:

- **Linearization of non-linear models**
- **Calculation of time-responses** – both analytically and numerically
- **Using simulation tools:** MATLAB, LabVIEW, Octave, and Scilab have simulation functions that assumes state-space models.
- **Analysis of dynamic systems**, e.g. stability analysis
- **Analysis and design of advanced controllers and estimators:** Controllability and observability analysis; Design of LQ optimal controllers, Model-based predictive control, and Feedback linearization control; Design of state estimators (Kalman filters).

1.2 A general state-space model

In general, an n 'th order state-space model consists of n *first order* differential equations characterized by having the *time-derivatives alone on*

the left side. A general n 'th order state-space model is

$$\dot{x}_1 = f_1() \quad (1.1)$$

$$\vdots$$

$$\dot{x}_n = f_n() \quad (1.2)$$

where $f_1(), \dots, f_n()$ are functions (given by the model equations, of course). The variables which have their time-derivatives in the state-space model are the *state-variables* of the model.¹ Thus, in the model above the x -variables are state-variables. x is a common name for state-variable, but you can use any name. The initial state (at $t = 0$) are defined by the values $x_1(0), \dots, x_n(0)$.

Some times you want to define the *output variables* of a state-space model. y is a common name for output variable. A model with m output variables can be written

$$y_1 = g_1() \quad (1.3)$$

$$\vdots$$

$$y_m = g_m() \quad (1.4)$$

where $g_1(), \dots, g_m()$ are functions. We can use the name state-space model for (1.1) – (1.2) and for (1.1) – (1.4).

The following example shows an example of a state-space model. A second order differential equation (which here is the model of a mass-spring-damper-system) will be written as a second order state-space model. We do this by defining a variable for each of the variables which have their time-derivative in the differential equation. These new variables becomes the state-variables of the state-space model. The same principle is used also for developing state-space models from higher order differential equations.

Example 1.1 *Mass-spring-damper-model written as a state-space model*

Figure 1.1 shows a mass-spring-damper-system. z is position. F is applied force. D is damping constant. K is spring constant. It is assumed that the damping force F_d is proportional to the velocity, and that the spring force

¹The name *state-space* model is because the values of the state-variables $x_1(t), \dots, x_n(t)$ defines the *state* of the at any instant of time. These values can be regarded as points in the *state-space*.

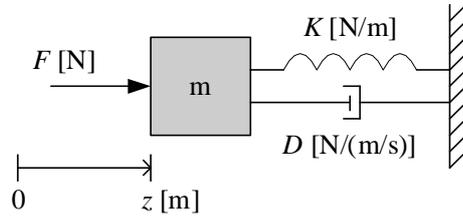


Figure 1.1: Mass-spring-damper

F_s is proportional to the position of the mass. The spring force is assumed to be zero when y is zero. Force balance (Newton's 2. Law) yields

$$\begin{aligned} m\ddot{y}(t) &= F(t) - F_d(t) - F_s(t) \\ &= F(t) - D\dot{y}(t) - Ky(t) \end{aligned} \quad (1.5)$$

which is a second order differential equation.

We define the following new variables: x_1 for position z , x_2 for speed \dot{z} and u for force F . Then the model (1.5) can be written as the following equivalent set of two first order differential equations:

$$\dot{x}_1 = x_2 \quad (1.6)$$

$$m\dot{x}_2 = -Dx_2 - Kx_1 + u \quad (1.7)$$

which can be written on the standard form (1.1), (1.2):

$$\dot{x}_1 = \underbrace{x_2}_{f_1} \quad (1.8)$$

$$\dot{x}_2 = \underbrace{\frac{1}{m}(-Dx_2 - Kx_1 + u)}_{f_2} \quad (1.9)$$

Let us regard the position x_1 as the output variable y :

$$y = \underbrace{x_1}_g \quad (1.10)$$

The initial position, $x_1(0)$, and the initial speed, $x_2(0)$, define the initial state of the system.

(1.8) and (1.9) and (1.10) constitute a second order state-space model which is equivalent to the original second order differential equation (1.5).

[End of Example 1.1]

(1.1) – (1.2) can be written on a *matrix-vector* model form:

$$\dot{x} = f() \quad (1.11)$$

where $\dot{x} = [\dot{x}_1, \dot{x}_2, \dots, \dot{x}_n]^T$ and $f = [f_1, f_2, \dots, f_n]^T$. Similarly, (1.3) – (1.4) can be written

$$y = g() \quad (1.12)$$

These compact model forms are convenient when deriving formulas for numerical calculation of time responses (i.e. developing simulation algorithms) and linearization of differential equation models.

1.3 Linear state-space models

Linear state-space models are a special case of the general state-space model (1.1) – (1.2). Many methods for analysis of differential equation models, as stability analysis, response calculation and model transformations, are based on linear state-space models. Let us study a general second order linear state-space model to see how linear state-space models are defined. The model has two state-variables, x_1 and x_2 , and two input variables, u_1 and u_2 :

$$\dot{x}_1 = a_{11}x_1 + a_{12}x_2 + b_{11}u_1 + b_{12}u_2 \quad (1.13)$$

$$\dot{x}_2 = a_{21}x_1 + a_{22}x_2 + b_{21}u_1 + b_{22}u_2 \quad (1.14)$$

where the a 's and b 's are parameters (constants).

(1.13), (1.14) can be written on matrix-vector form as follows:

$$\underbrace{\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}}_A \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_x + \underbrace{\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}}_B \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_u \quad (1.15)$$

or, more compact:

$$\dot{x} = Ax + Bu \quad (1.16)$$

where x is the state vector and u is the input vector. A is called the system-matrix, and is square in all cases.

Let us assume that the system has two output variables, which generally can be functions of both the state variables and the input variables. The output function can then be written on the form

$$y_1 = c_{11}x_1 + c_{12}x_2 + d_{11}u_1 + d_{12}u_2 \quad (1.17)$$

$$y_2 = c_{21}x_1 + c_{22}x_2 + d_{21}u_1 + d_{22}u_2 \quad (1.18)$$

which can be written on matrix-vector form as follows:

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}}_y = \underbrace{\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}}_C \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_x + \underbrace{\begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix}}_D \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_u \quad (1.19)$$

or, more compact:

$$y = Cx + Du \quad (1.20)$$

Example 1.2 *Mass-spring-damper model written on state-space form*

The state-space model (1.8), (1.9), (1.10) is linear. We get

$$\underbrace{\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\frac{K_f}{m} & -\frac{D}{m} \end{bmatrix}}_A \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_x + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}}_B u \quad (1.21)$$

$$y = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_C \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_x + \underbrace{\begin{bmatrix} 0 \end{bmatrix}}_D u \quad (1.22)$$

[End of Example 1.2]

1.4 Linearization of non-linear models

1.4.1 Introduction

In many cases the mathematical model contains one or more non-linear differential equations. If the mathematical model is non-linear, there may be good reasons to *linearize* it, which means to develop a local linear model which approximates the original model about a given operating point. The reasons may be the following:

- We want to study the behavior of the system *about an operating point*, which is one specified state where the system can be. It is then the *deviations* from this operating point we study. Examples of such operating points are the level of 8.7 m in a tank, the temperature of 50 degrees Celcius in a heat exchanger, etc. It can be shown (and we will do it soon) that a model which describes the behavior of the deviations about the operating point, is approximately linear.

- We can use the large number of the methods which are available for analysis and design of linear systems, e.g. for stability analysis, frequency response, controller design and signal filter design. The number of methods for linear models are much larger than for non-linear models.

Note: If you have a non-linear model of a (physical) system, do *not use the linearized model for simulation* unless you have a good reason for using it. In stead, use the (original) non-linear model since it gives a more accurate representation of the system.

Figure 1.2 illustrates the relation between the original non-linear system and the local linear system (model). The input variable which excites the

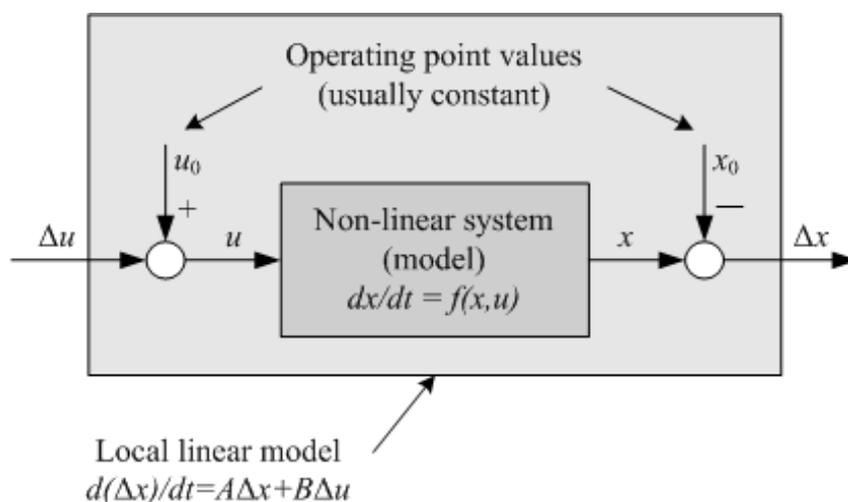


Figure 1.2: Illustration of the relation between the original non-linear system and the local linear system (model)

non-linear system is assumed to be given by

$$u = u_0 + \Delta u \quad (1.23)$$

where u_0 is the value in the operating point and Δu is the deviation from u_0 . Similarly,

$$x = x_0 + \Delta x \quad (1.24)$$

If you are going to experiment with the system to develop or adjust a linear model about the operating point, you must adjust Δu and observe the corresponding response in Δx (or in the output variable Δy).

1.4.2 Deriving the linearization formulas

We assume that the model is a non-linear state-space model:

$$\dot{x} = f(x, u) \quad (1.25)$$

Suppose that the system is in an *operating point* defined by

$$\dot{x}_0 = f(x_0, u_0) \quad (1.26)$$

If the input variable u is changed by Δu from the operating point value u_0 , the state-variable x is changed by Δx from x_0 . (1.26) can then be written

$$\frac{d(x_0 + \Delta x)}{dt} = f(x_0 + \Delta x, u_0 + \Delta u) \quad (1.27)$$

$$\downarrow$$

$$\dot{x}_0 + \Delta \dot{x} \approx f(x_0, u_0) + \left. \frac{\partial f}{\partial x} \right|_0 \Delta x + \left. \frac{\partial f}{\partial u} \right|_0 \Delta u \quad (1.28)$$

On the left side of (1.27) we have applies the summation rule of differentiation and on the right side we have used a first order Taylor series expansion of $f(\cdot)$. The expression $\left. \frac{\partial f}{\partial x} \right|_0$ means the partial time-derivative of f with respect to x , calculated in the operating point, that is, with x_0 and u_0 inserted into $\frac{\partial f}{\partial x}$. The same applies to $\left. \frac{\partial f}{\partial u} \right|_0$. Now we will exploit the fact that \dot{x}_0 is equal $f(x_0, u_0)$, cf. (1.26). This implies that these two terms are cancelled against each other in (1.28). (1.28) then becomes

$$\Delta \dot{x} = \underbrace{\left. \frac{\partial f}{\partial x} \right|_0}_{A} \Delta x + \underbrace{\left. \frac{\partial f}{\partial u} \right|_0}_{B} \Delta u \quad (1.29)$$

$$= A \Delta x + B \Delta u \quad (1.30)$$

or, in more detail,

$$\begin{bmatrix} \Delta \dot{x}_1 \\ \Delta \dot{x}_2 \\ \vdots \end{bmatrix} = \underbrace{\begin{bmatrix} \left. \frac{\partial f_1}{\partial x_1} \right|_0 & \left. \frac{\partial f_1}{\partial x_2} \right|_0 & \cdots \\ \left. \frac{\partial f_2}{\partial x_1} \right|_0 & \left. \frac{\partial f_2}{\partial x_2} \right|_0 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}}_A \cdot \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \end{bmatrix} + \underbrace{\begin{bmatrix} \left. \frac{\partial f_1}{\partial u_1} \right|_0 & \left. \frac{\partial f_1}{\partial u_2} \right|_0 & \cdots \\ \left. \frac{\partial f_2}{\partial u_1} \right|_0 & \left. \frac{\partial f_2}{\partial u_2} \right|_0 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}}_B \cdot \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \\ \vdots \end{bmatrix} \quad (1.31)$$

which is the local linear model. A and B becomes Jacobi-matrices (which are partial derivative matrices) which generally are functions of the operating point. If the operating point is *constant*, A and B will be constant matrices, which can be calculated once and for all.

Similarly, linearization of the output equation

$$y = g(x, u) \quad (1.32)$$

gives

$$\Delta y = \underbrace{\left. \frac{\partial g}{\partial x} \right|_0}_{C} \Delta x + \underbrace{\left. \frac{\partial g}{\partial u} \right|_0}_{D} \Delta u = C \Delta x + D \Delta u \quad (1.33)$$

or

$$\begin{bmatrix} \Delta y_1 \\ \Delta y_2 \\ \vdots \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}}_C \cdot \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \end{bmatrix} + \underbrace{\begin{bmatrix} \frac{\partial g_1}{\partial u_1} & \frac{\partial g_1}{\partial u_2} & \cdots \\ \frac{\partial g_2}{\partial u_1} & \frac{\partial g_2}{\partial u_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}}_D \cdot \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \\ \vdots \end{bmatrix} \quad (1.34)$$

If the operating point is a static *equilibrium* point, all variables have constant values and all time-derivatives are zero. Thus,

$$\dot{x}_0 = 0 = f(x_0, u_0) \quad (1.35)$$

The values of the model variables in the static operating point can be found by solving the algebraic equation (1.35) with respect to the unknown variables.

Example 1.3 *Linearization of a non-linear tank model*

Figure 1.3 shows a liquid tank with inlet via a pump and outlet via a valve with fixed opening. The outflow is assumed to be proportional to the square root of the pressure drop over the valve, and this pressure is assumed to be equal to the hydrostatic pressure ρgh at the outlet. The mass balance becomes

$$\begin{aligned} \rho A_t \dot{x}(t) &= \rho q_i(t) - \rho q_u(t) \\ &= \rho K_p u(t) - \rho K_u \sqrt{\rho g x(t)} \end{aligned}$$

which can be written

$$\dot{x}(t) = \frac{K_p}{A_t} u(t) - \frac{K_u}{A_t} \sqrt{\rho g x(t)} \equiv f(x, u) \quad (1.36)$$

We will find a local linear model about the static operating point given by $u = u_0$ (constant) and $x = x_0$ (constant). Let us first find the relation

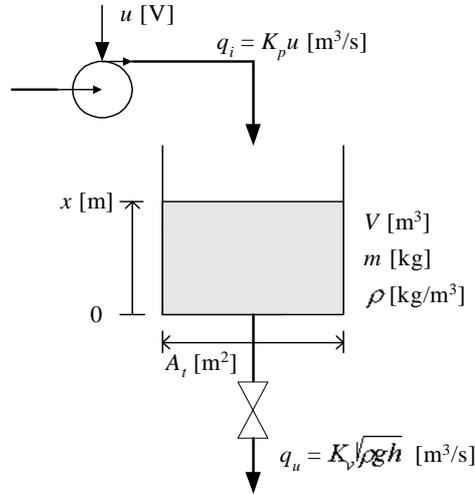


Figure 1.3: Liquid tank with non-linear mathematical model

between u_0 and x_0 from the static model which get by setting $\dot{x}(t) = 0$ in the dynamic model (1.36):

$$0 = \frac{K_p}{A_t} u_0 - \frac{K_u}{A_t} \sqrt{\rho g x_0} \equiv f(x_0, u_0) \quad (1.37)$$

Assume that x_0 is specified. The corresponding u_0 is found by solving (1.37) with respect to u_0 . The result is

$$u_0 = \frac{K_u}{K_p} \sqrt{\rho g x_0} \quad (1.38)$$

Now that we have found the static operating point we can derive the local linear model. Linearization of (1.36) yields

$$\begin{aligned} \Delta \dot{x} &= \left. \frac{\partial f}{\partial x} \right|_0 \Delta x + \left. \frac{\partial f}{\partial u} \right|_0 \Delta u \\ &= \underbrace{-\frac{K_u}{A_t} \frac{1}{2\sqrt{\rho g x_0}}}_{A} \Delta x + \underbrace{\frac{K_p}{A_t}}_B \Delta u \\ &= A \Delta x + B \Delta u \end{aligned} \quad (1.39)$$

[End of Example 1.3]

Chapter 2

Frequency response

2.1 Introduction

The *frequency response* of a system is a frequency dependent function which expresses how a sinusoidal signal of a given frequency on the system input is transferred through the system. Time-varying signals – at least periodical signals – which excite systems, as the reference (setpoint) signal or a disturbance in a control system or measurement signals which are inputs signals to signal filters, can be regarded as consisting of a sum of *frequency components*. Each frequency component is a sinusoidal signal having a certain amplitude and a certain frequency. (The Fourier series expansion or the Fourier transform can be used to express these frequency components quantitatively.) The frequency response expresses how each of these frequency components is transferred through the system. Some components may be amplified, others may be attenuated, and there will be some phase lag through the system.

The frequency response is an important tool for analysis and design of signal filters (as lowpass filters and highpass filters), and for analysis, and to some extent, design, of control systems. Both signal filtering and control systems applications are described (briefly) later in this chapter.

The definition of the frequency response – which will be given in the next section – *applies only to linear models*, but this linear model may very well be the local linear model about some operating point of a non-linear model.

The frequency response can found experimentally or from a transfer function model. It can be presented graphically or as a mathematical function.

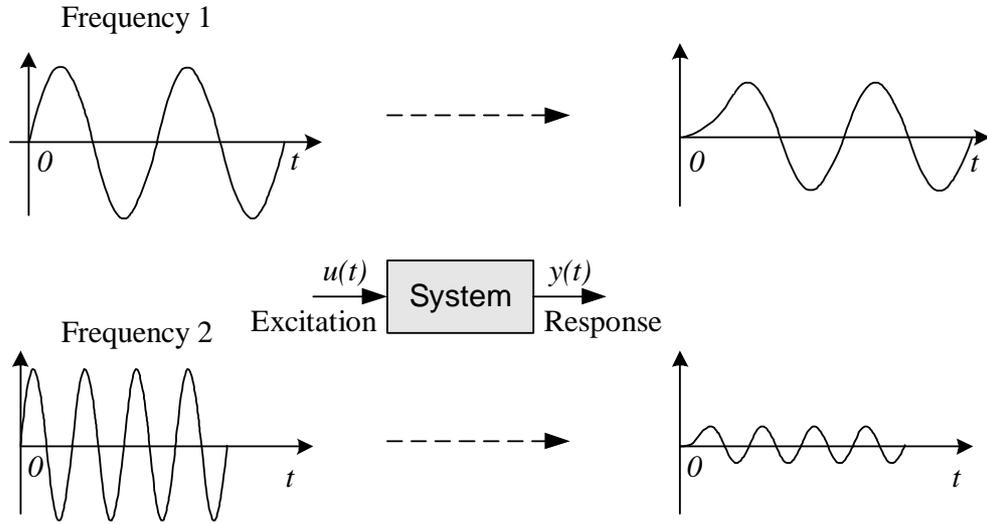


Figure 2.1: Sinusoidal signals in the input and the resulting responses on the output for two different frequencies

2.2 How to calculate frequency response from sinusoidal input and output

We can find the frequency response of a system by exciting the system with a sinusoidal signal of amplitude U and frequency ω [rad/s] and observing the response in the output variable of the system.¹

Mathematically, we set the input signal to

$$u(t) = U \sin \omega t \quad (2.1)$$

See Figure 2.1. This input signal will give a transient response (which will die, eventually) and a *steady-state* response, $y_s(t)$, in the output variable:

$$y_s(t) = Y \sin(\omega t + \phi) \quad (2.2)$$

$$= \underbrace{UA}_Y \sin(\omega t + \phi) \quad (2.3)$$

Here A is the (*amplitude*)*gain*, and ϕ (phi) is *the phase lag* in radians. The frequency of $y_s(t)$ will be the same as in $u(t)$. Figure 2.2 shows in detail $u(t)$ and $y(t)$ for a simulated system. The system which is simulated is

$$y(s) = \frac{1}{s+1} u(s) \quad (2.4)$$

¹The correspondance between a given frequency ω in rad/s and the same same frequency f in Hz is $\omega = 2\pi f$.

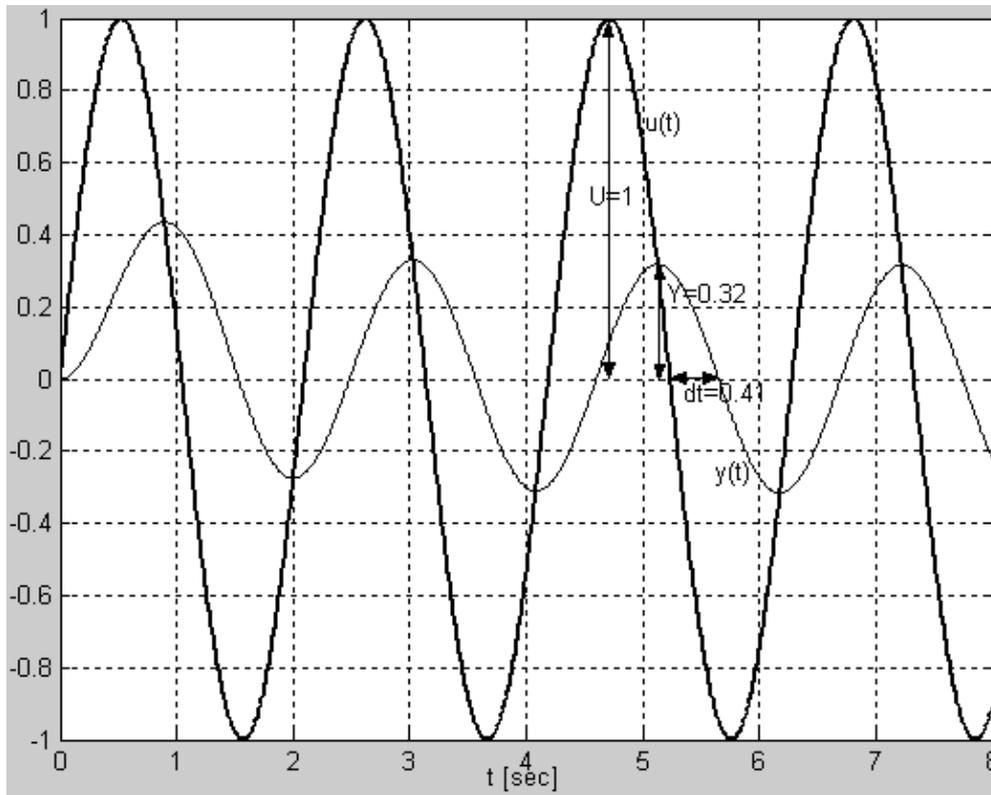


Figure 2.2: The input signal $u(t)$ and the resulting (sinusoidal) response $y(t)$ for a simulated system. $u(t)$ has frequency $\omega = 3$ rad/s and amplitude $U = 1$. The system is given by (2.4).

(a first order system with gain 1 and time-constant 1). The input signal $u(t)$ has frequency $\omega = 3$ rad/s and amplitude $U = 1$.

A is the ratio between the amplitudes of the output signal and the input signal (in steady-state):

$$A = \frac{Y}{U} \quad (2.5)$$

For the signals shown in Figure 2.2,

$$A = \frac{Y}{U} = \frac{0.32}{1} = 0.32 \quad (2.6)$$

ϕ can be calculated by first measuring the time-lag Δt between $u(t)$ and $y_s(t)$ and then calculating ϕ as follows:

$$\phi = -\omega \Delta t \quad [\text{rad}] \quad (2.7)$$

In Figure 2.2 we find $\Delta t = 0.41$ sec, which gives

$$\phi = -\omega\Delta t = -3 \cdot 0.41 = -1.23 \text{ rad} \quad (2.8)$$

The gain A and the phase-lag ϕ are functions of the frequency. We can use the following terminology: $A(\omega)$ is the *gain function*, and $\phi(\omega)$ is the *phase shift function* (or more simply: phase function). We say that $A(\omega)$ and $\phi(\omega)$ expresses the *frequency response* of the system.

Bode diagram

It is common to present $A(\omega)$ and $\phi(\omega)$ graphically in a *Bode diagram*, which consists of two subdiagrams, one for $A(\omega)$ and one for $\phi(\omega)$, where the phase values are usually plotted in degrees (not radians). Figure 2.3 shows a Bode diagram of the frequency response of the system given by (2.4). The curves may stem from a number of A -values and ϕ -values found in experiments (or simulations) with an sinusoidal input signal of various frequencies. The curves may also stem from the transfer function of the system, as described in Section 2.3. The frequency axes usually show the 10-logarithm of the frequency in rad/s or in Hz.

Actually, the system (2.4) is used to generate $u(t)$ and $y(t)$ shown in Figure 2.2. We have earlier in this chapter calculated $A(3) = 0.32 = -10.2$ dB (the dB-unit is described below) and phase lag $\phi(3) = -1.23 \text{ rad} = -72$ degrees. This gain value and phase lag value are indicated in the Bode diagram in Figure 2.3.

The $A(\omega)$ -axis is usually drawn with decibel (dB) as unit. The decibel value of a number x is calculated as

$$x \text{ [dB]} = 20 \log_{10} x \quad (2.9)$$

Table 2.1 shows some examples of dB-values.

2.3 How to calculate frequency response from transfer functions

In Section 2.2 we saw how to find the frequency response from experiments on the system. No model was assumed. However, if we know a transfer function model of the system, we can *calculate the frequency response from the transfer function*, as explained below.

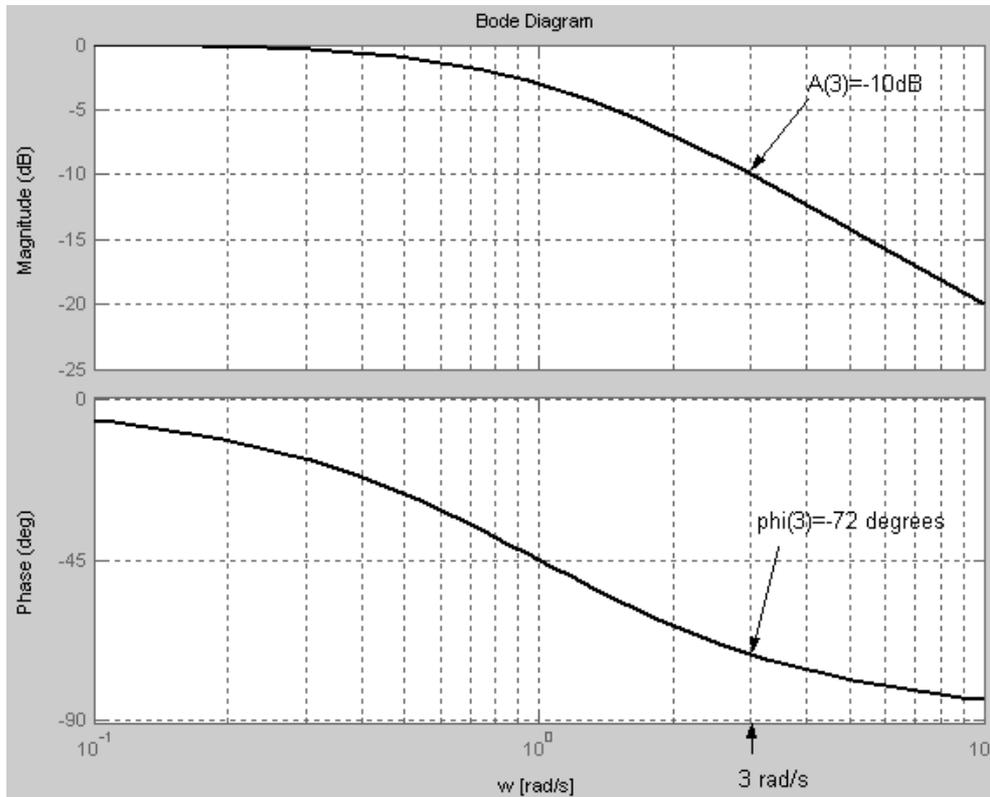


Figure 2.3: The frequency response of the system given by (2.4) presented in a Bode diagram

Suppose that system has the transfer function $H(s)$ from input u to output y , that is,

$$y(s) = H(s)u(s) \quad (2.10)$$

By setting

$$s = j\omega \quad (2.11)$$

(j is the imaginary unit) into $H(s)$, we get the complex quantity $H(j\omega)$, which is the *frequency response* (function). The *gain function* is

$$A(\omega) = |H(j\omega)| \quad (2.12)$$

and the *phase shift function* is the angle or argument of $H(j\omega)$:

$$\phi(\omega) = \arg H(j\omega) \quad (2.13)$$

(The formulas (2.12) and (2.13) can be derived using the Laplace transform.)

0	=	$-\infty$ dB
0.01	=	-40dB
0.1	=	-20dB
0.2	=	-14dB
0.25	=	-12dB
0.5	=	-6dB
$\frac{1}{\sqrt{2}}$	=	-3dB
1	=	0dB
$\sqrt{2}$	=	3dB
2	=	6dB
$\sqrt{10}$	=	10dB
4	=	12dB
5	=	14dB
10	=	20dB
100	=	40dB

Table 2.1: Some dB-values

Example 2.1 *Frequency response calculated from a transfer function*

We will find the frequency response for the transfer function

$$H(s) = \frac{K}{Ts + 1} \quad (2.14)$$

The frequency response becomes

$$H(j\omega) = H(s)|_{s=j\omega} = \frac{K}{Tj\omega + 1} = \frac{K}{\underbrace{1}_{\text{Re}} + j\underbrace{T\omega}_{\text{Im}}} \quad (2.15)$$

which we write on polar form:

$$H(j\omega) = \frac{K}{\sqrt{1^2 + (T\omega)^2} e^{j \arctan(\frac{T\omega}{1})}} \quad (2.16)$$

$$= \frac{1}{\sqrt{1 + (T\omega)^2}} e^{j[-\arctan(T\omega)]} \quad (2.17)$$

$$= |H(j\omega)| e^{j \arg H(j\omega)} \quad (2.18)$$

Thus, the gain function is

$$|H(j\omega)| = \frac{K}{\sqrt{1 + (T\omega)^2}} \quad (2.19)$$

and the phase function is

$$\arg H(j\omega) = -\arctan(T\omega) \quad [\text{rad}] \quad (2.20)$$

Figure 2.4 shows the curves of $|H(j\omega)|$ and $\arg H(j\omega)$ drawn in a Bode diagram. The numerical values along the axes assume $K = 1$ and $T = 1$. (The asymptotes indicated in the figure are not explained in this document.)

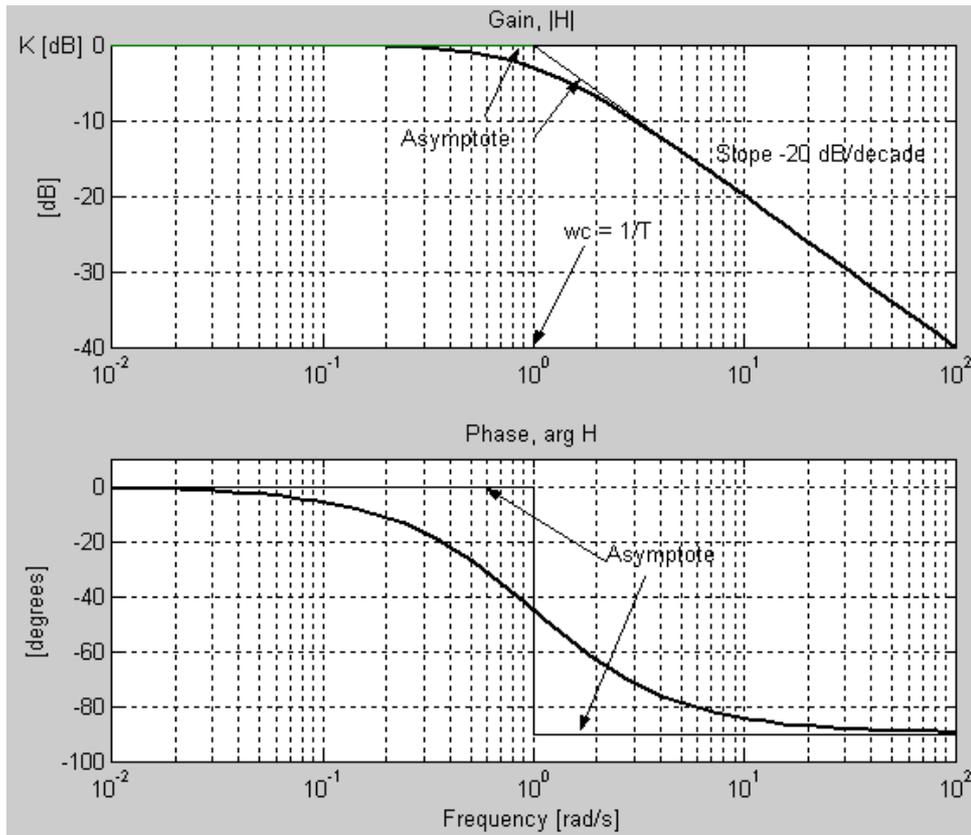


Figure 2.4: Bode diagram for the frequency response of the first order system (2.14). The asymptotes are not explained in this document.

To illustrate the use of (2.19) and (2.20), let us calculate the gain and phase lag values for the frequency $\omega = 3$ rad/s. We assume that $K = 1$ and $T = 1$. (2.19) gives

$$|H(j3)| = \frac{1}{\sqrt{1+3^2}} = \frac{1}{\sqrt{10}} = 0.316 = -20 \log_{10} \left(\frac{1}{\sqrt{10}} \right) = -10.0 \text{ dB} \quad (2.21)$$

(2.20) gives

$$\arg H(j3) = -\arctan(3) = -1.25 \text{ rad} = -71.6 \text{ degrees} \quad (2.22)$$

[End of Example 2.1]

The next example shows how the frequency response can be found of a transfer function which consists of several factors in the numerator and/or the denominator.

Example 2.2 *Frequency response of a (more complicated) transfer function*

Given the transfer function

$$H(s) = K \frac{T_1 s + 1}{(T_2 s + 1) s} e^{-\tau s} \quad (2.23)$$

(The term $e^{-\tau s}$ represents a time-delay of τ sec.) We set $s = j\omega$ in $H(s)$ and then sets the individual factors on *polar form*. Finally, we combine these factors so that we end up with a polar form of $H(j\omega)$:

$$H(j\omega) = K \frac{T_1 j\omega + 1}{(T_2 j\omega + 1) j\omega} e^{-\tau j\omega} \quad (2.24)$$

$$= K \frac{\sqrt{1^2 + (T_1\omega)^2} e^{j \arctan\left(\frac{T_1\omega}{1}\right)}}{\left[\sqrt{1^2 + (T_2\omega)^2} e^{j \arctan\left(\frac{T_2\omega}{1}\right)} \right] \left[\sqrt{0^2 + \omega^2} e^{j \frac{\pi}{2}} \right]} e^{-\tau j\omega} \quad (2.25)$$

$$= \underbrace{\frac{K \sqrt{1 + (T_1\omega)^2}}{\sqrt{1 + (T_2\omega)^2} \omega}}_{|H(j\omega)|} e^{j \underbrace{\left[\arctan(T_1\omega) - \arctan(T_2\omega) - \frac{\pi}{2} - \tau\omega \right]}_{\arg H(j\omega)}} \quad (2.26)$$

So, the amplitude gain function is

$$A(\omega) = |H(j\omega)| = \frac{K \sqrt{1 + (T_1\omega)^2}}{\sqrt{1 + (T_2\omega)^2} \omega} \quad (2.27)$$

and the phase shift function is

$$\phi(\omega) = \arg H(j\omega) = \arctan(T_1\omega) - \arctan(T_2\omega) - \frac{\pi}{2} - \tau\omega \quad (2.28)$$

[End of Example 2.2]

2.4 Application of frequency response: Signal filters

2.4.1 Introduction

A *signal filter* – or just *filter* – is used to attenuate (ideally: remove) a certain frequency interval of frequency components from a signal. These frequency components are typically noise. For example, a lowpass filter is used to attenuate high-frequent components (low-frequent components passes).

Knowledge about filtering functions is crucial in signal processing, but it is useful also in control engineering because control systems can be regarded as filters in the sense that the controlled process variable can follow only a certain range or interval of frequency components in the reference (setpoint) signal, and it will be only a certain frequency range of process disturbances that the control system can compensate for effectively. Furthermore, knowledge about filters can be useful in the analysis and design of physical processes. For example, a stirred tank in a process line can act as a lowpass filter since it attenuates low-frequent components in the inflow to the tank.

In this section we will particularly study *lowpass filters*, which is the most commonly used filtering function, but we will also take a look at *highpass filters*, *bandpass filters* and *bandstop filters*.

Figure 2.5 shows the gain function for ideal filtering functions and for practical filters (the phase lag functions are not shown). The *passband* is the frequency interval where the gain function has value 1, ideally (thus, frequency components in this frequency interval passes through the filter, unchanged). The *stopband* is the frequency interval where the gain function has value 0, ideally (thus, frequency components in this frequency interval are stopped through the filter).²

It can be shown that transfer functions for ideal filtering functions will have infinitely large order. Therefore, ideal filters can not be realized, neither with analog electronics nor with a filtering algorithm in a computer program.

²It is a pity that lowpass filters were not called highstop filters in stead since the main purpose of a lowpass filter is to stop high-frequency components. Similarly, highpass filters should have been called lowstop filters, but it is too late now...

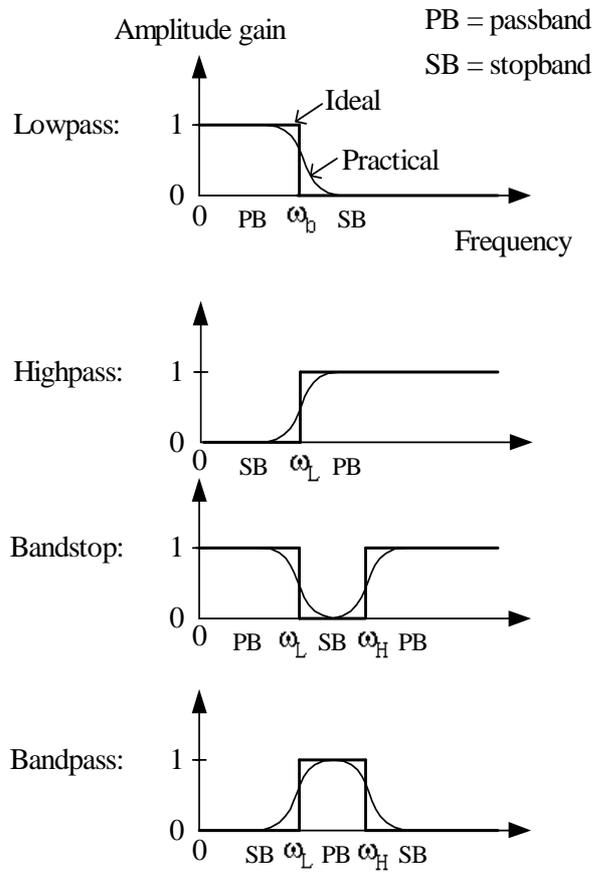


Figure 2.5: The gain functions for ideal filters and for practical filters of various types.

2.4.2 First order lowpass filters

The most commonly used signal filter is the first order lowpass filter. As an example, it is the standard measurement filter in a feedback control system.

The transfer function of a first order lowpass filter with input variable u and output variable y is usually written as

$$H(s) = \frac{1}{\frac{s}{\omega_b} + 1} \quad (2.29)$$

where ω_b [rad/s] is the *bandwidth* of the filter. This is a first order transfer

function with gain $K = 1$ and time-constant $T = 1/\omega_b$. The frequency response is

$$H(j\omega) = \frac{1}{\frac{j\omega}{\omega_b} + 1} \quad (2.30)$$

$$\begin{aligned} &= \frac{1}{\sqrt{\left(\frac{\omega}{\omega_b}\right)^2 + 1} e^{j \arctan \frac{\omega}{\omega_b}}} \\ &= \frac{1}{\sqrt{\left(\frac{\omega}{\omega_b}\right)^2 + 1}} e^{j\left(-\arctan \frac{\omega}{\omega_b}\right)} \end{aligned} \quad (2.31)$$

The gain function is

$$|H(j\omega)| = \frac{1}{\sqrt{\left(\frac{\omega}{\omega_b}\right)^2 + 1}} \quad (2.32)$$

and the phase lag function is

$$\arg H(j\omega) = -\arctan \frac{\omega}{\omega_b} \quad (2.33)$$

Figure 2.4 shows exact and asymptotic curves of $|H(j\omega)|$ and $\arg H(j\omega)$ drawn in a Bode diagram. In the figure, $K = 1$ and $\omega_b = \omega_c$.

The bandwidth defines the upper limit of the passband. It is common to say that the bandwidth is the frequency where the filter gain is $1/\sqrt{2} = 0.71 \approx -3$ dB (above the bandwidth the gain is less than $1/\sqrt{2}$). This bandwidth is therefore referred to as the “ -3 dB-bandwidth”. Now, what is the -3 dB-bandwidth of a first order lowpass filter? It is the ω -solution of the equation

$$|H(j\omega)| = \frac{1}{\sqrt{\left(\frac{\omega}{\omega_b}\right)^2 + 1}} = \frac{1}{\sqrt{2}} \quad (2.34)$$

The solution is $\omega = \omega_b$. Therefore, ω_b [rad/s] given in (2.29) is the -3 dB-bandwidth in rad/s. In Hertz the bandwidth is

$$f_b = \frac{\omega_b}{2\pi} \quad (2.35)$$

Figure 2.6 shows the front panel of a simulator of a first order filter where the input signal consists of a sum of two sinusoids or frequency components of frequency less than and greater than, respectively, the bandwidth. The simulation shows that the low frequent component (0.5 Hz) passes almost unchanged (it is in the passband of the filter), while the high-frequent component (8 Hz) is attenuated (it lies in the stopband).

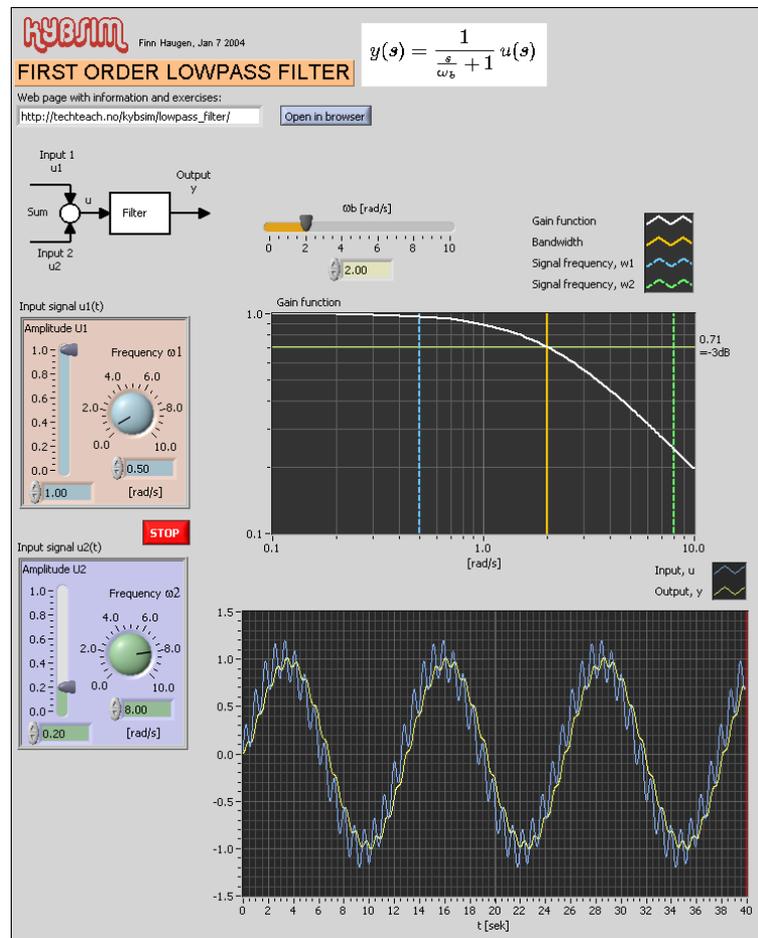


Figure 2.6: Simulator for a first order lowpass filter where the input signal consists of a sum of two frequency components

Example 2.3 The RC-circuit as a lowpass filter

Figure 2.7 shows an RC-circuit (the circuit contains the resistor R and the capacitor C). The RC-circuit is frequently used as an analogue lowpass filter: Signals of low frequencies passes approximately unchanged through the filter, while signals of high frequencies are approximately filtered out (stopped). v_1 is the signal source or input voltage to be filtered, while v_2 is the resulting filtered output voltage.

We will now find a mathematical model relating v_2 to v_1 . First we apply the Kirchhoff's voltage law in the circuit which consists the input voltage terminals, the resistor, and the capacitor (we consider the voltage drops to

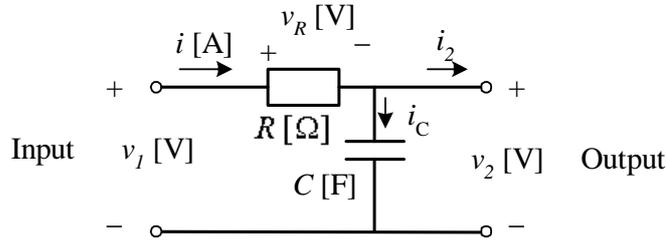


Figure 2.7: RC-circuit

be positive clockwise direction):

$$-v_1 + v_R + v_2 = 0 \quad (2.36)$$

(v_2 equals the voltage drop over the capacitor.) In (2.36) v_R is given by

$$v_R = Ri \quad (2.37)$$

We assume that there is no current going through the output terminals. (This is a common assumption, and not unrealistic, since it is typical that the output terminals are connected to a subsequent circuit which has approximately infinite input impedance, causing the current into it to be approximately zero. An operational amplifier is an example of such a load-circuit.) Therefore,

$$i = i_C = C\dot{v}_2 \quad (2.38)$$

The final model is achieved by using i as given by (2.38) in (2.37) and then using v_R as given by (2.37) for v_R in (2.36). The model becomes

$$RC\dot{v}_2(t) = v_1(t) - v_2(t) \quad (2.39)$$

The transfer function from the input voltage v_1 to the output voltage v_2 becomes

$$H_{v_2, v_1}(s) = \frac{1}{RCs + 1} = \frac{1}{\frac{s}{\omega_b} + 1} \quad (2.40)$$

Thus, the RC-circuit is a first order lowpass filter with bandwidth

$$\omega_b = \frac{1}{RC} \text{ rad/s} \quad (2.41)$$

If for example $R = 1 \text{ k}\Omega$ and $C = 10 \text{ }\mu\text{F}$, the bandwidth is $\omega_b = 1/RC = 100 \text{ rad/s}$. (2.41) can be used to design the RC-circuit (calculate the R- and C-values).

[End of Example 2.3]

Chapter 3

Frequency response analysis of feedback control systems

3.1 Introduction

With frequency response – using Bode plots – we can analyze dynamic properties of feedback control systems. These properties refers to

- dynamic setpoint tracking, and
- dynamic disturbance compensation.

By definition, in frequency response analysis all signals in the system are assumed to be sinusoids. This seems to limit the usefulness of such analysis because in real systems signals are rarely sinusoids. Still, the frequency response analysis provides useful insight about the dynamic properties of a control system because signals are varying more or less, i.e. the signals can be said to have certain frequency components.

In this chapter the frequency response analysis will be based on the following two transfer functions of the control system:

- *Tracking transfer function, $T(s)$*
- *Sensitivity transfer function, $S(s)$*

This analysis assumes a *linear* model of the control system. However, practical control systems are nonlinear due to phenomena as saturation,

hysteresis, stiction, nonlinear signal scaling etc. Such nonlinearities can influence largely the dynamic behaviour of the control system. To perform “linear” analysis of a non-linear model, this model must be linearized about some operating point. Thus, the results of the analysis will be valid at or close to the operation point where the linearization was made. This fact limits the usefulness of a theoretical analysis of a given nonlinear control system using linear systems methods, but the results may still be useful, particularly if the system most of the time operates close to the chosen or specified operating point.

Although a “linear” analysis of a given nonlinear control system may have limited value, you will get much general understanding about the behaviour of control systems through analysis of examples of linear control systems.

Note: Once you have a mathematical model of a given control system, you should definitely *run simulations* as a part of the analysis. This applies for both linear and nonlinear control systems. Actually, you may get all the answers you need by just running simulations. The types of answers may concern response-time, static control error, responses due to process disturbances and measurement noise, and effects of parameter variations.

3.2 Definition of setpoint tracking and disturbance compensation

Figure 3.1 shows a principal block diagram of a control system. There are

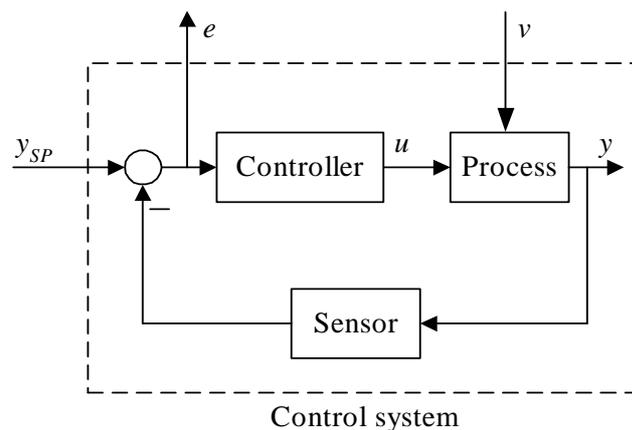


Figure 3.1: Principal block diagram of a control system

two input signals to the control system, namely the setpoint y_{SP} and the disturbance v . The value of the control error e is our primary concern (it should be small, preferably zero). Therefore we can say that e is the (main) output variable of the control system. The value of e expresses the *performance* of the control system: The less e , the higher performance. e is influenced by y_{SP} and v . Let us therefore define the following two properties of control systems:

- The **setpoint tracking property** of the control system concerns the relation between y_{SP} and e .
- The **disturbance compensation property** of the control system concerns the relation between v and e .

Totally, the setpoint tracking and disturbance compensation properties determine the *performance* of the control system.

3.3 Definition of characteristic transfer functions

3.3.1 The Sensitivity transfer function

We assume that the control system has a transfer function-based block diagram as shown in Figure 3.2. In the block diagram $U_0(s)$ represents the Laplace transform of the nominal control variable u_0 . In the analysis we will set u_0 to zero, because we will focus on the responses of the control system due to the setpoint and disturbance inputs.

We regard the setpoint y_{SP} and the disturbance v as input variables and the control error e as the output variable of the system. Thus, we will derive the transfer function from y_{SP} to e and the transfer function from v to e . From the block diagram we can write the following expressions

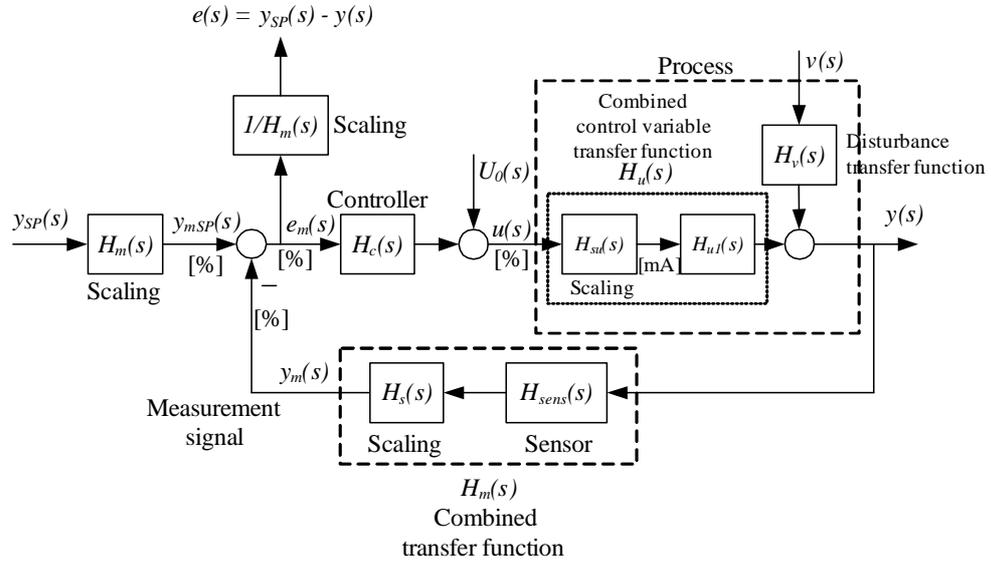


Figure 3.2: Transfer function based block diagram of a control system. (The units, e.g. %, are typical examples of units.)

for $e(s)$:

$$e(s) = \frac{1}{H_m(s)} e_m(s) \quad (3.1)$$

$$= \frac{1}{H_m(s)} [y_{m,SP}(s) - y_m(s)] \quad (3.2)$$

$$= \frac{1}{H_m(s)} [H_m(s)y_{SP}(s) - H_m(s)y(s)] \quad (3.3)$$

$$= y_{SP}(s) - y(s) \quad (3.4)$$

$$= y_{SP}(s) - [H_v(s)v(s) + H_u(s)H_c(s)e_m(s)] \quad (3.5)$$

$$= y_{SP}(s) - [H_v(s)v(s) + H_u(s)H_c(s)H_m(s)e(s)] \quad (3.6)$$

In (3.6), $e(s)$ appears at both the left and the right side. Solving for $e(s)$

gives

$$\begin{aligned} e(s) &= \frac{1}{1 + H_c(s)H_u(s)H_m(s)} [y_{SP}(s) - H_v(s)v(s)] \\ &= \frac{1}{\underbrace{1 + L(s)}_{S(s)}} [y_{SP}(s) - H_v(s)v(s)] \end{aligned} \quad (3.7)$$

$$= S(s) [y_{SP}(s) - H_v(s)v(s)] \quad (3.8)$$

$$= \underbrace{S(s)y_{SP}(s)}_{e_{SP}(s)} - \underbrace{S(s)H_v(s)v(s)}_{e_v(s)} \quad (3.9)$$

$$= e_{SP}(s) + e_v(s) \quad (3.10)$$

which is a transfer functions based model of the control system. $S(s)$ is the *sensitivity transfer function*:

$$S(s) = \frac{1}{1 + L(s)} \quad (3.11)$$

where

$$L(s) \equiv H_c(s)H_u(s)H_m(s) \quad (3.12)$$

is the *loop transfer function* which is the product of the transfer functions in the loop. From (3.9) we can calculate the control error for any setpoint signal, any disturbance signal and any nominal control signal (assuming we know their Laplace transform).

In the following we discuss the various terms in (3.10).

- **The response in the error due to the setpoint:** The response in the control error due to the *setpoint* is

$$e_{SP}(s) = S(s)y_{SP}(s) = \frac{1}{1 + L(s)}y_{SP}(s) \quad (3.13)$$

which gives a quantitative expression of the *tracking property* of the control system. The *static* tracking is given by static error when y_{SP} is constant. This error can be calculated as follows:¹

$$e_{SP} = \lim_{t \rightarrow \infty} e_{SP}(t) = \lim_{s \rightarrow 0} s \cdot e_{SP}(s) \quad (3.14)$$

$$= \lim_{s \rightarrow 0} s \cdot S(s)y_{SP}(s) = \lim_{s \rightarrow 0} s \cdot S(s) \frac{y_{SP_s}}{s} = S(0)y_{SP_s} \quad (3.15)$$

Roughly speaking that the tracking property of the control system are good if the sensitivity function N has small (absolute) value – ideally zero.

¹Here the Final Value Theorem of the Laplace transform is used.

- **The response in the error due to the disturbance:** The response in the control error due to the disturbance is

$$e_v(s) = -S(s)H_v(s)v(s) = -\frac{H_v(s)}{1+L(s)}v(s) \quad (3.16)$$

which expresses the *compensation property* of the control system.

The *static* compensation property is given by

$$e_{v_s} = \lim_{t \rightarrow \infty} e_v(t) = \lim_{s \rightarrow 0} s \cdot e_v(s) \quad (3.17)$$

$$= \lim_{s \rightarrow 0} s \cdot [-S(s)H_v(s)v(s)] \quad (3.18)$$

$$= \lim_{s \rightarrow 0} s \cdot \left[-S(s)H_v(s) \frac{v_s}{s} \right] \quad (3.19)$$

$$= -S(0)H_v(0)v_s \quad (3.20)$$

From (3.20) we see that the *compensation property is good if the sensitivity function S has a small (absolute) value (close to zero)*.

3.3.2 The Tracking transfer function

The *tracking transfer function* $T(s)$ – or simply the tracking function – is the transfer function from the setpoint y_{SP} to the process output variable y :

$$y(s) = T(s)y_{SP}(s) \quad (3.21)$$

From the block diagram in Figure 3.2, or by setting

$e_{y_{SP}}(s) \equiv y_{SP}(s) - y(s)$ for $e_{y_{SP}}(s)$ in (3.13), we can find the tracking function $T(s)$ as the transfer function from y_{SP} to y :

$$\frac{y(s)}{y_{SP}(s)} = T(s) = \frac{H_c(s)H_u(s)H_m(s)}{1 + H_c(s)H_u(s)H_m(s)} = \frac{L(s)}{1 + L(s)} = 1 - S(s) \quad (3.22)$$

The *static* tracking property is given by the static tracking ratio $T(0)$:

$$y_s = \lim_{t \rightarrow \infty} y(t) = \lim_{s \rightarrow 0} s \cdot y(s) \quad (3.23)$$

$$= \lim_{s \rightarrow 0} s \cdot T(s)y_{SP}(s) = \lim_{s \rightarrow 0} s \cdot T(s) \frac{y_{SP_s}}{s} \quad (3.24)$$

$$= T(0)y_{SP_s} \quad (3.25)$$

The *tracking property is good if the tracking function T has (absolute) value equal to or close to 1* (since then y will be equal to or close to y_{SP}).

In some contexts it is useful to be aware that the sum of the tracking function and the sensitivity function is always 1:

$$T(s) + S(s) = \frac{L(s)}{1 + L(s)} + \frac{1}{1 + L(s)} \equiv 1 \quad (3.26)$$

3.4 Frequency response analysis of setpoint tracking and disturbance compensation

3.4.1 Introduction

Frequency response analysis of control systems expresses the tracking and compensation property under the assumption that the setpoint and the disturbance are *sinusoidal signals* or *frequency components* of a compound signal. The structure of the control system is assumed to be as shown in Figure 3.2. The Laplace transformed control error is given by (3.9), which is repeated here:

$$e(s) = \underbrace{S(s)y_{SP}(s)}_{e_{SP}(s)} - \underbrace{S(s)H_v(s)v(s)}_{e_v(s)} \quad (3.27)$$

where $S(s)$ is the sensitivity function which is given by

$$S(s) = \frac{1}{1 + L(s)} \quad (3.28)$$

where $L(s)$ is the loop transfer function. In the following we will study both $S(s)$ and the tracking ratio $T(s)$ which is given by

$$T(s) = \frac{L(s)}{1 + L(s)} = \frac{y(s)}{y_{SP}(s)} \quad (3.29)$$

3.4.2 Frequency response analysis of setpoint tracking

From (3.27) we see we that the response in the control error due to the setpoint is

$$e_{SP}(s) = S(s)y_{SP}(s) \quad (3.30)$$

By plotting the *frequency response* $S(j\omega)$ we can easily calculate how large the error is for a given frequency component in the setpoint: Assume that the setpoint is a sinusoid of amplitude Y_{SP} and frequency ω . Then the steady-state response in the error is

$$e_{SP}(t) = Y_{SP} |S(j\omega)| \sin[\omega t + \arg S(j\omega)] \quad (3.31)$$

Thus, the error is small and consequently the tracking property is good if $|S(j\omega)| \ll 1$, while the error is large and the tracking property poor if $|S(j\omega)| \approx 1$.

The tracking property can be indicated by the tracking function $T(s)$, too. The response in the process output due to the setpoint is

$$y(s) = T(s)y_{SP}(s) \quad (3.32)$$

Assume that the setpoint is a sinusoid of amplitude Y_{SP} and frequency ω . Then the steady-state response in the process output due to the setpoint is

$$y(t) = Y_{SP} |T(j\omega)| \sin[\omega t + \arg T(j\omega)] \quad (3.33)$$

Thus, $|T(j\omega)| \approx 1$ indicates that the control system has good tracking property, while $|T(j\omega)| \ll 1$ indicates poor tracking property.

Since both $S(s)$ and $T(s)$ are functions of the loop transfer function $L(s)$, cf. (3.28) and (3.29), there is a relation between $L(s)$ and the tracking property of the control system. Using (3.28) and (3.29) we can conclude as follows:

$$\text{Good setpoint tracking: } |S(j\omega)| \ll 1, |T(j\omega)| \approx 1, |L(j\omega)| \gg 1 \quad (3.34)$$

$$\text{Poor setpoint tracking: } |S(j\omega)| \approx 1, |T(j\omega)| \ll 1, |L(j\omega)| \ll 1 \quad (3.35)$$

Figure 3.3 shows typical Bode plots of $|S(j\omega)|$, $|T(j\omega)|$ and $|L(j\omega)|$. Usually we are interested in the amplitude gains, not the phase lags. Therefore plots of $\arg S(j\omega)$, $\arg T(j\omega)$ and $\arg L(j\omega)$ are not shown nor discussed here. The bandwidths indicated in the figure are defined below.

The *bandwidth* of a control system is the frequency which divides the frequency range of good tracking and poor tracking. From (3.34) and (3.35) and Figure 3.3 we can list the following three candidates for a definition of the bandwidth:

- ω_t , which is the frequency where the amplitude gain of the tracking function has value $1/\sqrt{2} \approx 0.71 = -3$ dB. This definition is in accordance with the usual bandwidth definition of lowpass filters. The ω_t bandwidth is also called the -3 dB bandwidth $\omega_{-3\text{dB}}$.
- ω_c , which is the frequency where the amplitude gain of the loop transfer function has value $1 = -0$ dB. ω_c is called the *crossover frequency* of L .
- ω_s , which is the frequency where the amplitude gain of the sensitivity function has value $1 - 1/\sqrt{2} \approx 1 - 0.71 \approx 0.29 \approx -11$ dB. This definition is derived from the -3 dB bandwidth of the tracking function: Good tracking corresponds to tracking gain between $1/\sqrt{2}$

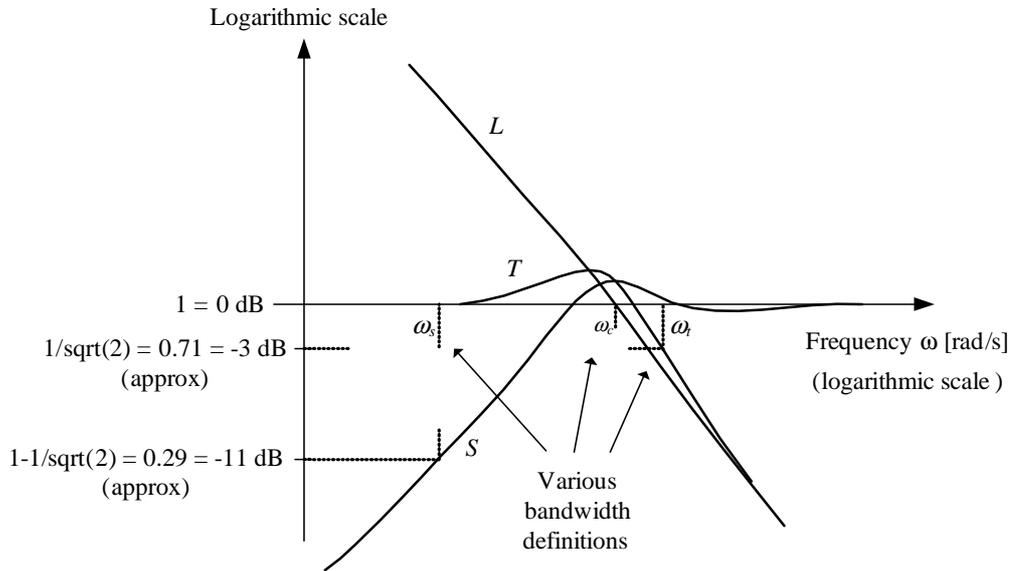


Figure 3.3: Typical Bode plots of $|S(j\omega)|$, $|T(j\omega)|$ and $|L(j\omega)|$

and 1. Now recall that the sensitivity function is the transfer function from setpoint to control error, cf. (3.30). Expressed in terms of the control error, we can say that good tracking corresponds to sensitivity gain $|S|$ less than $1 - 1/\sqrt{2} \approx -11 \text{ dB} \approx 0,29$. The frequency where $|S|$ is -11 dB is denoted the *sensitivity bandwidth*, ω_s .

Of the three bandwidth candidates defined above the sensitivity bandwidth ω_s is most closely related to the control error. Therefore ω_s may be claimed to be the most convenient bandwidth definition as far as the tracking property of a control system concerns. In addition ω_s is a convenient bandwidth related to the compensation property of a control system (this will be discussed in more detail soon). However, the crossover frequency ω_c and the -3 dB bandwidth are the commonly used bandwidth definitions.

As indicated in Figure 3.3 the numerical values of the various bandwidth definitions are different (this is demonstrated in Example 3.1).

If you need a (possibly rough) estimate of the *response time* T_r of a control system, which is time it takes for a step response to reach 63% of its steady-state value, you can use

$$T_r \approx \frac{k}{\omega_t} \text{ [s]} \quad (3.36)$$

where ω_t is the -3 dB bandwidth in rad/s.² k can be set to some value between 1.5 and 2.0, say 2.0 if you want to be conservative.

Example 3.1 Frequency response analysis of setpoint tracking

See the block diagram in Figure 3.2. Assume the following transfer functions:

PID controller:

$$H_c(s) = K_p \left(1 + \frac{1}{T_i s} + \frac{T_d s}{T_f s + 1} \right) \quad (3.37)$$

Process transfer functions (second order with time delay):

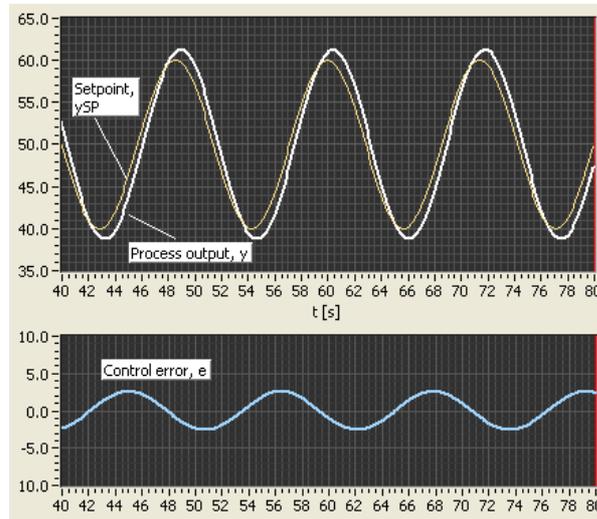


Figure 3.4: Example 3.1: Simulated responses of the control system. The setpoint y_{SP} is sinusoid of frequency $\omega_1 = 0.55$ rad/s.

$$H_u(s) = \frac{K_u}{(T_1 s + 1)(T_2 s + 1)} e^{-\tau s} \quad (3.38)$$

$$H_v(s) = \frac{K_v}{(T_1 s + 1)(T_2 s + 1)} e^{-\tau s} \quad (3.39)$$

Sensor with scaling:

$$H_s(s) = K_s \quad (3.40)$$

The parameter values are $K_p = 4.3$, $T_i = 1.40$, $T_d = 0.35$, $T_f = 0.1T_d = 0.035$, $K_u = 1$, $K_d = 1$, $T_1 = 2$, $T_2 = 0.5$, $\tau = 0.4$, $K_s = 1$.

²How can you find the exact value of the response time? Simulate!

(The PID parameter values are calculated using the Ziegler-Nichols' closed loop method.) The operation point is at setpoint value 50%, with disturbance $v = 10\%$ (constant), and nominal control signal = 40%.

Figure 3.4 shows simulated responses in the process output y and in the the control error $e = y_{SP} - y$ when the setpoint y_{SP} is a sinusoid of amplitude 10% (about a bias of 50%) and frequency $\omega_1 = 0.55\text{rad/s}$. The frequency of the sinusoidal is chosen equal to the sensitivity bandwidth ω_s . The amplitude of the control error should be $0.29 \cdot 10\% = 2.9\%$, and this is actually in accordance with the simulation, see Figure 3.4.

Figure 3.5 shows Bode plots of $|S(j\omega)|$, $|T(j\omega)|$ and $|L(j\omega)|$.

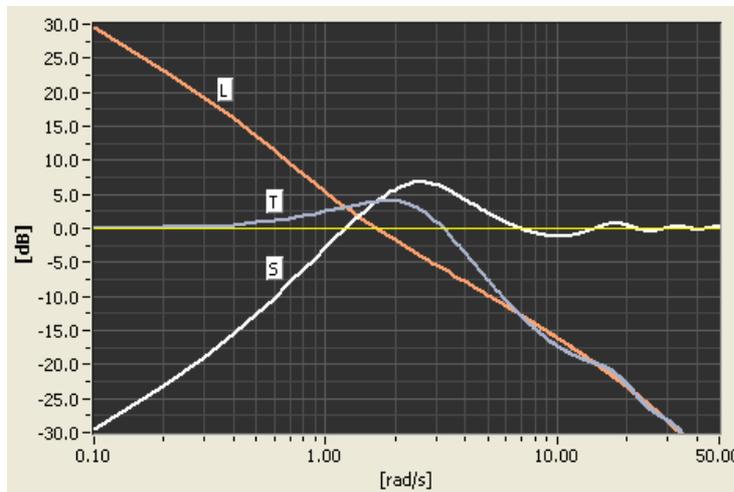


Figure 3.5: Example 3.1: Bode plots of $|L(j\omega)|$, $|T(j\omega)|$ and $|S(j\omega)|$

Let us compare the various bandwidth definitions. From Figure 3.5 we find

- -3 dB bandwidth: $\omega_t = 3.8$ rad/s
- Crossover frequency: $\omega_c = 1.7$ rad/s
- Sensitivity bandwidth: $\omega_s = 0.55$ rad/s

These values are actually quite different. (As commented in the text above this example, it can be argued that the ω_s bandwidth gives the most expressive measure of the control system dynamics.)

Finally, let us read off the response time T_r . Figure 3.6 shows the response in y due to a step in y_{SP} . From the simulation we read off $T_r \approx 1.1$ s. The

estimate (3.36) with $k = 2$ gives $T_r \approx 2/\omega_t = 2/3.8 = 0.53$, which is about half the value of the real (simulated) value.

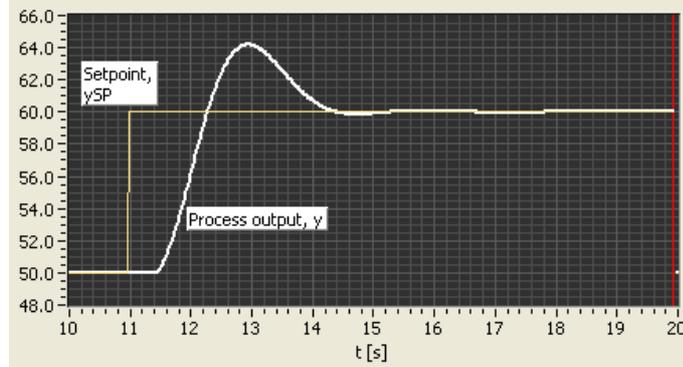


Figure 3.6: Example 3.1: Step response in process output y after a step in setpoint y_{SP}

[End of Example 3.1]

3.4.3 Frequency response analysis of disturbance compensation

(3.27) gives the response in the control error due to the disturbance. It is repeated here:

$$e_v(s) = -S(s)H_v(s)v(s) \quad (3.41)$$

Thus, the sensitivity function $S(s)$ is a factor in the transfer function from v til e for the control system. However, $S(s)$ has an additional meaning related to the compensation of a disturbance, namely it expresses the degree of the reduction of the control error due to using closed loop control. *With* feedback (i.e. closed loop system) the response in the control error due to the disturbance is $e_v(s) = -S(s)H_v(s)v(s)$. *Without* feedback (open loop) this response is $e_v(s) = -H_v(s)v(s)$. The ratio between these responses is

$$\frac{e_v(s)_{\text{with feedback}}}{e_v(s)_{\text{without feedback}}} = \frac{-S(s)H_v(s)v(s)}{-H_v(s)v(s)} = S(s) \quad (3.42)$$

Assuming that the disturbance is sinusoidal with frequency ω rad/s, (3.42) with $s = j\omega$, that is $S(j\omega)$, expresses the ratio between sinusoidal responses.

Again, effective control, which here means effective disturbance compensation, corresponds to a small value of $|S|$ (value zero or close to zero), while ineffective control corresponds to $|S|$ close to or greater than 1. We can define the *bandwidth* of the control system with respect to its compensation property. Here are two alternate bandwidth definitions:

- The bandwidth ω_s – the sensitivity bandwidth – is the upper limit of the frequency range of effective compensation. One possible definition is

$$|S(j\omega_s)| \approx 0.29 \approx -11 \text{ dB} \quad (3.43)$$

which means that the amplitude of the error *with* feedback control is less than 29% of amplitude *without* feedback control. The number 0.29 is chosen to have the same bandwidth definition regarding disturbance compensation as regarding setpoint tracking, cf. page 46.

- The bandwidth ω_c is the crossover frequency of the loop transfer functions ω_c , that is,

$$|L(j\omega_c)| = 0 \text{ dB} \approx 1 \quad (3.44)$$

Note: The feedback does not reduce the control error due to a sinusoidal disturbance if its frequency is above the bandwidth. But still the disturbance may be well attenuated through the (control) system. This attenuation is due to the typical inherent lowpass filtering characteristic of physical systems (processes). Imagine a liquid tank, which attenuates high-frequent temperature variations existing in the inflow fluid temperature or in the environmental temperature. This inherent lowpass filtering is *self regulation*.

Example 3.2 *Frequency response analysis of disturbance compensation*

This example is based on the control system described in Example 3.1 (page 48).

Figure 3.7 shows simulated responses in the process output y due to a sinusoidal disturbance v of amplitude 10% (with bias 10%) and frequency $\omega_1 = 0.55 \text{ rad/s}$. This frequency is for illustration purpose chosen equal to the sensitivity bandwidth of the control system, cf. Figure 3.5. The setpoint y_{SP} is 50%. The control error can be read off as the difference between y_{SP} and y . In the first 40 seconds of the the simulation the PID controller is in manual mode, so the control loop is open. In the following

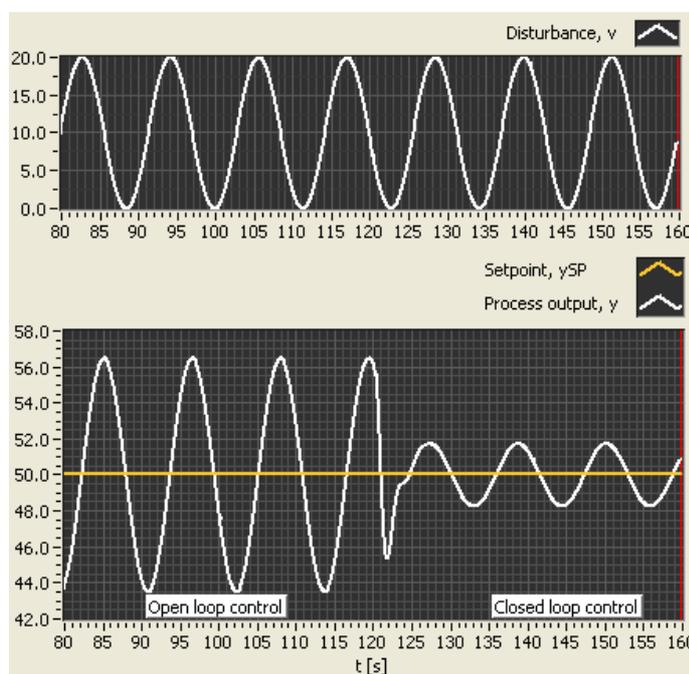


Figure 3.7: Example 3.2: Simulated responses of the control system. The disturbance v is sinusoidal with frequency $\omega_1 = 0.55$ rad/s. The PID-controller is in manual mode (i.e. open loop control) the first 40 seconds, and in automatic mode (closed loop control) thereafter.

40 seconds the PID controller is in automatic mode, so the control loop is closed. We clearly see that the feedback control is effective to compensate for the disturbance at this frequency (0.55 rad/s). The amplitude of the control error is 6.6 without feedback and 1.9 with feedback. Thus, the ratio between the closed loop error and the open loop error is $1.9/6.6 = 0.29$, which is in accordance with the amplitude of the sensitivity function at this frequency, cf. Figure 3.5.

Figure 3.8 shows the same kind of simulation, but with disturbance frequency $\omega_1 = 1.7$ rad/s, which is higher than the sensitivity bandwidth, which is 0.55 rad/s. From the simulations we see that closed loop control at this relatively high frequency, 1.7 rad/s, does not compensate for the disturbance — actually the open loop works better. This is in accordance with the fact that $|S(j\omega)|$ is greater than 1 at $\omega = 1.7$ rad/s, cf. the Bode plot in Figure 3.5.

Finally, let us compare the simulated responses shown in Figure 3.8 and in Figure 3.4. The amplitude of the control error is less in Figure 3.8, despite

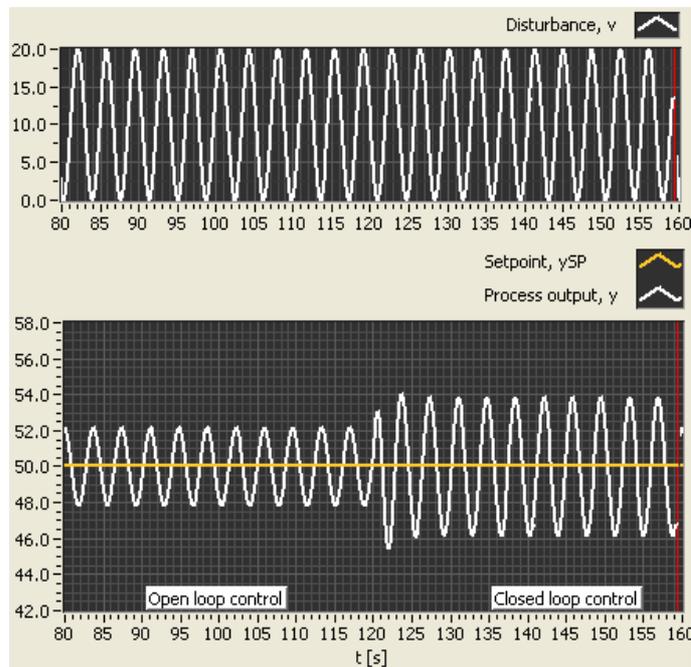


Figure 3.8: Example 3.2: Simulated responses of the control system. The disturbance v is sinusoidal with frequency $\omega_1 = 1.7$ rad/s. The PID-controller is in manual mode (i.e. open loop control) the first 40 seconds, and in automatic mode (closed loop control) thereafter.

the fact that the closed loop or feedback control is not efficient (at frequency 1.7 rad/s). The relatively small amplitude of the control error is due to the self regulation of the process, which means that the disturbance is attenuated through the process, whether the process is controlled or not.

[End of Example 3.2]

In Example 3.2 I did not choose the disturbance frequency, 1.7 rad/s, by random. 1.7 rad/s is actually the loop transfer function crossover frequency of the control system. Thus, the example demonstrates that the crossover frequency may give a poor measure of the performance of the control system. The sensitivity bandwidth is a better measure of the performance.

Chapter 4

Stability analysis of dynamic systems

4.1 Introduction

In some situations we want to determine if a dynamic system is stable or unstable. Particularly in the control theory, stability analysis is important, since feedback control systems may become unstable if the controller parameters have been given erroneous values. In this chapter, different stability properties will be defined in terms of the placement of the pole or eigenvalues of the system in the complex plane.

Traditionally Routh's stability criterion has been taught as a tool for stability analysis. Routh's criterion is a method for determining the correspondence between stability properties and values of system (model) parameters, *without* calculating the actual poles or eigenvalues. But, since it is my clear impression and experience that Routh's stability criterion does not play an important role in practical stability analysis of dynamic systems, so I have decided not to include the method in this book. (One reference is [1].)

There exists a special graphical method — the Nyquist's stability criterion — for stability analysis of feedback systems (as control systems) based on the frequency response. This method is presented in Chapter 5.

4.2 Stability properties and impulse response

This section defines the different stability properties that a dynamic system can have in terms of impulse response of the system. Then, the corresponding transfer function pole locations in the complex plane are derived. In Section 5.2, and subsequent sections, these results are applied to feedback (control) systems, which are a special case of dynamic systems.

The different stability properties can be defined in several ways. I have chosen to use the *impulse response* of the system as the basis for definition of the stability properties. The impulse response is the time-response in output variable of the system due to an impulse on the input. Using the impulse response makes it relatively simple to relate stability properties to the *poles* of the system (this is because the impulse response and the poles are closely connected), as you will see soon.

Some words about the impulse signal: It is a time-signal which in principle has infinitely short duration and infinite amplitude, but so that the integral of the signal – the integral is equal to the area under the time-function of the signal – is finite. This area is also called the *strength* of the impulse. An impulse of strength one is called a unit impulse, $\delta(t)$. The square pulse in Figure 4.1 approaches an impulse function as Δ goes to zero.

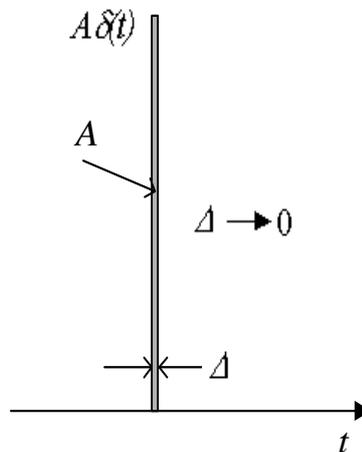


Figure 4.1: The square pulse approaches an impulse function as Δ goes to zero.

Here are the stability definitions: A dynamic system has one of the following stability properties:

- **Asymptotically stable system:** The stationary impulse response, $h(t)$, is zero:

$$\lim_{t \rightarrow \infty} h(t) = 0 \quad (4.1)$$

- **Marginally stable system:** The stationary impulse response is different from zero, but limited:

$$0 < \lim_{t \rightarrow \infty} h(t) < \infty \quad (4.2)$$

- **Unstable system:** The stationary impulse response is unlimited:

$$\lim_{t \rightarrow \infty} h(t) = \infty \quad (4.3)$$

A system is *stable* if it is either asymptotically stable or marginally stable. Figure 4.2 depicts the different stability properties.

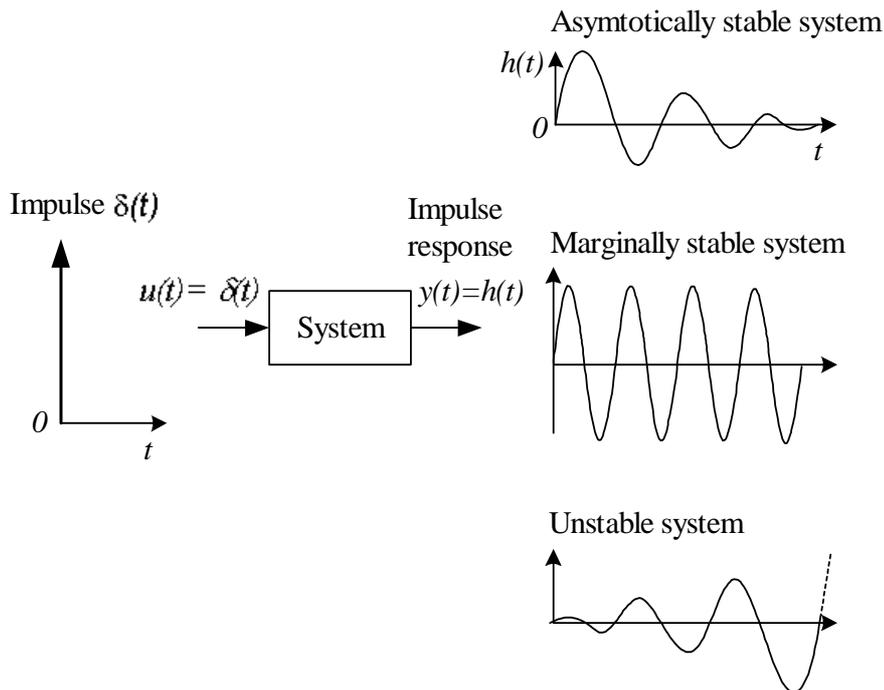


Figure 4.2: Different stability properties

In the definition above it is assumed that the impulse in the input u starts from zero and that the impulse response in the output y has zero initial value. In practice these initial values may be different from zero if the

system initially is some other operating point than the “zero” operating point.

One problem with the ideal impulse function is it can not be generated fully in practice, but in practice there is hardly ever a need to perform impulse response experiments to determine stability properties. It is more useful as a conceptual definition of stability, cf. the next section.

4.3 Stability properties and poles

In many cases it would be quite impractical if the only way to determine the stability property of a system was to do experiments or to run simulations, to obtain the impulse response (or the step response). Fortunately, we can conclude about the stationary part of the impulse response, and hence conclude about the stability property, by just analyzing the mathematical model of the system. This is because the stationary impulse response is a function of the *poles* of the system. The connection between the impulse response and the poles is derived in the following.

Let us assume that the mathematical model of the system is a transfer function, $H(s)$, from input signal u to output signal y :

$$y(s) = H(s)u(s) \quad (4.4)$$

The input u will be a unit impulse, that is, $u(t) = \delta(t)$. It can be shown that the Laplace transform of $\delta(t)$ is 1. Let us denote the impulse response $h(t)$. The Laplace transform of $h(t)$ is

$$h(s) = H(s)\mathcal{L}\{\delta(t)\} = H(s) \cdot 1 = H(s) \quad (4.5)$$

Thus, *the Laplace transform of the impulse response equals the transfer function of the system.* We need the impulse response time-function $h(t)$ since it is the basis for the definitions of the different stability properties.

With results from Laplace transform theory, $h(t)$ can be calculated using the following formula:

$$h(t) = \sum_i \lim_{s \rightarrow p_i} \frac{1}{(m-1)!} \left\{ \frac{d^{m-1}}{ds^{m-1}} \left[(s-p_i)^m \underbrace{H(s)}_{h(s)} e^{st} \right] \right\} \quad (4.6)$$

$$= \sum_i \lim_{s \rightarrow p_i} \left\{ (s-p_i) \underbrace{H(s)}_{h(s)} e^{st} \right\} \text{ if } m = 1 \quad (4.7)$$

$\{p_i\}$ is the set of poles in $H(s)$, and hence the roots of the denominator of $H(s)$. m is the multiplicity of the poles (so-called simple poles have $m = 1$). The denominator polynomial of $H(s)$ is called the *characteristic polynomial*, $a(s)$. The poles are the roots of $a(s)$. Consequently, the poles are the solutions of the *characteristic equation*

$$a(s) = 0 \quad (4.8)$$

An example: Transfer function

$$H(s) = \frac{1}{(s+2)(s+3)} \quad (4.9)$$

has the poles $p_1 = -2$ and $p_2 = -3$, and the characteristic polynomial is

$$a(s) = (s+2)(s+3) = s^2 + 5s + 6 \quad (4.10)$$

We will now use (4.6) and (4.7) to connect the different stability properties to the pole placement in the complex plane. Let us first assume that the poles of $H(s)$ are simple. Then $m = 1$, and $h(t)$ is given by (4.7). A pole is generally a complex number:

$$p_i = a_i + jb_i \quad (4.11)$$

where a_i is the real part and b_i is the imaginary part of the pole. (4.7) implies that $h(t)$ is equal to the sum of the i partial responses of the total impulse response:

$$h(t) = \sum_i h_i(t) \quad (4.12)$$

where

$$h_i(t) = k_i e^{p_i t} = k_i e^{(a_i + jb_i)t} = k_i e^{a_i t} e^{jb_i t} \quad (4.13)$$

Here k_i is some constant. The term $e^{jb_i t}$ is a complex number on the unity circle and therefore it has absolute value equal to 1.¹ Thus, it is the term $e^{a_i t}$ which determines the steady-state ($t \rightarrow \infty$) absolute value of the partial response $h_i(t)$ according to the following analysis:

- Suppose that the real part, a_i , of the pole is strictly negative, that is, $a_i < 0$, which means that the pole lies in the left half plane. This implies $e^{a_i t} \rightarrow 0$, and therefore $h_i(t) \rightarrow 0$ as $t \rightarrow \infty$.

¹If a pole has a imaginary part b different from zero, there must be a complex conjugate pole with imaginary part $-b$. However, this fact does not influence the conclusions of the analysis we are performing.

- Suppose that the real part, a_i , of the pole is zero, that is $a_i = 0$, which means that the pole lies on the imaginary axis. This implies $e^{a_i t} = 1$, and therefore $h_i(t)$ goes towards a constant value different from zero as $t \rightarrow \infty$.
- Suppose that the real part, a_i , of the pole is strictly positive, that is, $a_i > 0$, which means that the pole lies in the right half plane. This implies $e^{a_i t} \rightarrow \infty$, and therefore $h_i(t) \rightarrow \infty$ as $t \rightarrow \infty$.

From the above analysis we can conclude as follows for transfer functions having pole multiplicity one: (1) If each of the poles lies in the left half plane, the system is *asymptotically stable*, because then each of the partial impulse response terms, $h_i(t)$, goes towards 0 as $t \rightarrow \infty$. (2) If one pole lies on the imaginary axis while the rest of the poles lies on the left half plane, the system is *marginally stable*, because then one of the $h_i(t)$ -terms goes towards a constant value different from 0 as $t \rightarrow \infty$. (3) If at least one of the poles lies in the right half plane, the system is *unstable*, because then at least one term $h_i(t)$ goes to ∞ as $t \rightarrow \infty$.

Multiple poles: It would have been nice to conclude about stability and pole placement now, but we have to look closer at the case of *multiple* poles of $H(s)$. The impulse response $h(t)$ is given by (4.6). Suppose that the multiplicity of the pole p_i is $m = 2$. The corresponding partial impulse response becomes

$$h_i(t) = \lim_{s \rightarrow p_i} \left\{ \frac{d}{ds} \left[(s - p_i)^2 H(s) e^{st} \right] \right\} \quad (4.14)$$

Here, the term $\frac{d}{ds}(e^{st})$ is equal to te^{st} , which means that $h_i(t)$ will contain terms as $te^{p_i t}$. By performing the same analysis as for simple poles, we will find the following: (1) $h_i(t) \rightarrow 0$ for a pole with negative real part (since $te^{p_i t}$ goes towards zero because $e^{p_i t}$ decreases faster than t increases). (2) $h_i(t) \rightarrow \infty$ for a pole on the imaginary axis ($te^{p_i t}$ equals t). (3) $h_i(t) \rightarrow \infty$ for a pole having positive real part. We will get the same results if the multiplicity m is greater than two.

Now we can conclude by stating the following correspondence between stability and pole placement:

- **Asymptotically stable system:** Each of the poles of the transfer function lies strictly in the left half plane (has strictly negative real part).
- **Marginally stable system:** One or more poles lies on the imaginary axis (have real part equal to zero), and all these poles are distinct. Besides, no poles lie in the right half plane.

- **Unstable system:** At least one pole lies in the right half plane (has real part greater than zero). Or: There are multiple poles on the imaginary axis.

Figure 4.3 gives a illustration of the relation between stability property and pole placement.

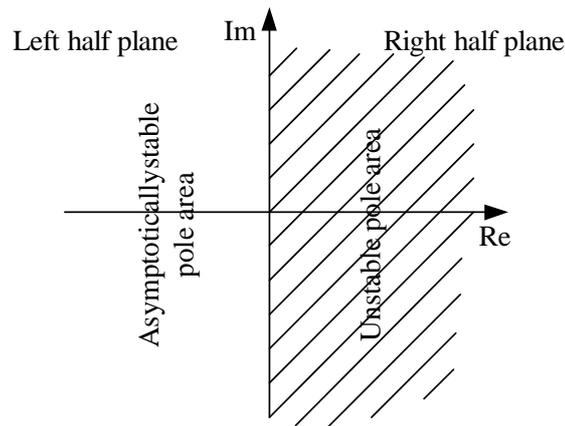


Figure 4.3: The relation between stability property and pole placement

Example 4.1 *Stability property of some simple dynamic systems*

The first order transfer function

$$H_1(s) = \frac{1}{s+1} \quad (4.15)$$

has pole $p = -1$ which lies in the left half plane. Thus, the system is asymptotically stable.

The transfer function

$$H_2(s) = \frac{1}{s} \quad (4.16)$$

(which is the transfer function of an integrator) has pole $p = 0$, which lies on the imaginary axis and has multiplicity one. So, the system is marginally stable.

The transfer function

$$H_3(s) = \frac{1}{s^2} \quad (4.17)$$

have poles $p = 0$, which is on the imaginary axis with multiplicity two. The system is therefore unstable.

The transfer function

$$H_A(s) = \frac{1}{s - 1} \quad (4.18)$$

has pole $p = +1$, which lies in the right half plane. The system is therefore unstable.

[End of Example 4.1]

Example 4.2 *Stability property of a mass-spring-damper*

Figure 4.4 shows a mass-spring-damper-system. y is position. F is applied

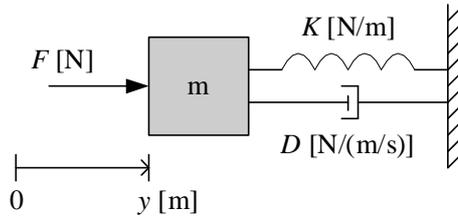


Figure 4.4: Mass-spring-damper

force. D is damping constant. K is spring constant. Newton's 2. Law gives the following mathematical model:

$$m\ddot{y}(t) = F(t) - D\dot{y}(t) - Ky(t) \quad (4.19)$$

The transfer function from the force F to position y is

$$H(s) = \frac{y(s)}{F(s)} = \frac{1}{ms^2 + Ds + K} \quad (4.20)$$

Assume that $m = 20$ kg, $D = 4$ N/(m/s), and $K = 2$ N/m. What is the stability property of the system? The characteristic polynomial becomes

$$a(s) = ms^2 + Ds + K = 20s^2 + 4s + 2 \quad (4.21)$$

which has roots

$$p_1, p_2 = \frac{-4 \pm \sqrt{4^2 - 4 \cdot 20 \cdot 2}}{2 \cdot 20} = -0.1 \pm j0.3 \quad (4.22)$$

which are the poles of $H(s)$. Both these poles have strictly negative real parts (-0.1). The system is therefore asymptotically stable. Figure 4.5

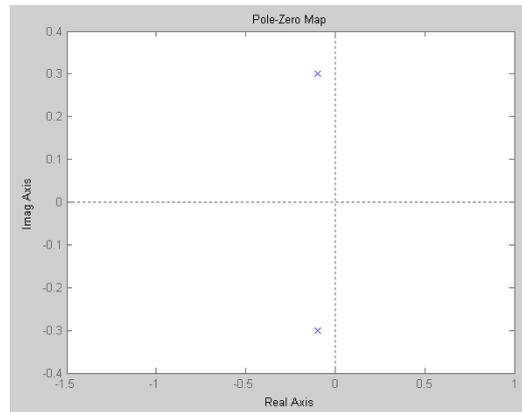


Figure 4.5: The poles of a mass-spring-damper plotted in the complex plane. The poles are $p_{1,2} = -0.1 \pm j0.3$.

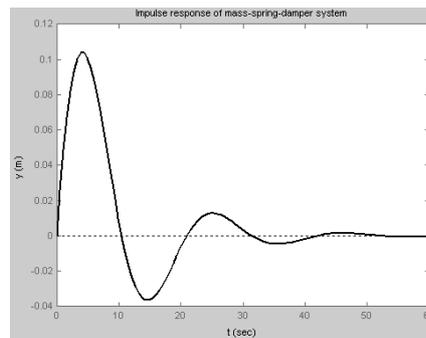


Figure 4.6: The impulse response for a mass-spring-damper with $m = 20$ kg, $D = 4$ N/(m/s) and $K_f = 2$ N/m. The system is asymptotically stable.

shows the poles (marked as crosses) in the complex plane. Figure 4.6 shows the impulse response.

Assume now that the damper is removed so that $D = 0$. Then the characteristic polynomial is

$$a(s) = ms^2 + K \quad (4.23)$$

and the poles are

$$p_1, p_2 = \pm j\sqrt{\frac{K}{m}} = \pm j0.32 \quad (4.24)$$

which lies on the imaginary axis, and they have multiplicity one. The system is then marginally stable. Figure 4.7 shows the impulse response.

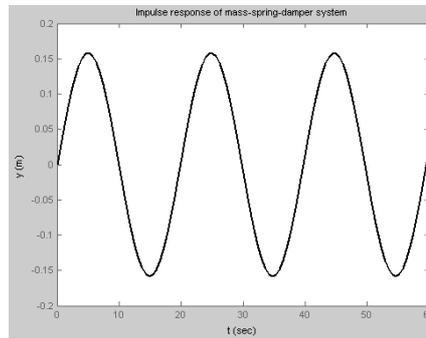


Figure 4.7: The impulse response of the mass-spring-damper with $m = 20$ kg, $D = 0$ N/(m/s) og $K_f = 2$ N/m. The system is marginally stable.

[End of Example 4.2]

4.4 Stability properties of state-space models

In Section 4.3 the different stability properties were related to the poles of the transfer function of a system. If the model originally is given as a state-space model,

$$\dot{x} = Ax + Bu \quad (4.25)$$

$$y = Cx + Du \quad (4.26)$$

we can determine the stability by finding the corresponding transfer function from u to y . We can derive the transfer function as follows: Take the Laplace transform of (4.25) – (4.26) to get (I is the identity matrix of equal dimension as of A)

$$sIx(s) - x_0 = Ax(s) + Bu(s) \quad (4.27)$$

$$y(s) = Cx(s) + Du(s) \quad (4.28)$$

We neglect x_0 . Solving (4.27) for $x(s)$ gives

$$x(s) = (sI - A)^{-1}Bu(s) \quad (4.29)$$

Inserting this $x(s)$ into (4.28) gives

$$y(s) = [C(sI - A)^{-1}B + D] u(s) \quad (4.30)$$

So the transfer function is

$$H(s) = \frac{y(s)}{u(s)} = C(sI - A)^{-1}B + D \equiv C \frac{\text{adj}(sI - A)}{\det(sI - A)} B + D \quad (4.31)$$

The stability property can now be determined from the poles of this transfer function. The poles are the roots of the characteristic equation:

$$a(s) = \det(sI - A) = 0 \quad (4.32)$$

But (4.32) defines the eigenvalues of A ². So, *the poles of the transfer function are the same as the eigenvalues of A* . Consequently, we can just calculate these eigenvalues and *conclude about the stability from the placement of the eigenvalues in the complex plane*. Just substitute pole by eigenvalue in the criteria for asymptotic stability, marginal stability and instability on page 4.3.

Here is a small modification: In some state-space models factors of type $(s - p_i)$ in the denominator can be cancelled against factors $(s - z_i)$ in the numerator of the transfer function. Such pole/zero-cancellations implies that some of the poles (and zeros) “disappears” from the transfer function. Consequently, the set of poles will then be just a subset of the set of eigenvalues. Thus, there may exist eigenvalues which are not poles, so that stability analysis based on eigenvalues placement (in the complex plane) may give a different result than stability analysis based on pole placement.

²In mathematics literature it is more common to use the symbol λ in stead of s for eigenvalues.

Chapter 5

Stability analysis of feedback systems

5.1 Introduction

A control system must be *asymptotically stable*. Two of the most relevant methods to determine the stability property of a control system are:

- **Pole-based stability analysis**, which is based on calculating the poles of the control system. The correspondence between the stability property and the poles placement in the complex plane is developed in Chapter 4.
- **Nyquist's stability criterion** which is based on the frequency response of the loop transfer function¹, $L(j\omega)$.

Both of these methods are described in the present chapter.

There is an algebraic analysis method named Routh's stability criterion [1] which is based on the coefficients of the characteristic polynomial of the control system. This method is not described in this book since it has limited practical importance, and the mathematical operations become quite complicated except for the simplest models.

¹the product of all the transfer functions in the control loop, cf. (3.12)

5.2 Pole-based stability analysis of feedback systems

Which is the transfer function to be used to determine the stability analysis of *feedback systems, e.g. control systems*? Let's start with Figure 3.2 which shows a block diagram of a feedback control system. We must select a transfer function from one of the input signals to the closed loop to one of the output signals from the loop. Let us select the transfer function from the setpoint y_{mSP} to the process measurement y_m . This transfer function is

$$\frac{y_m(s)}{y_{mSP}(s)} = \frac{H_c(s)H_u(s)H_m(s)}{1 + H_c(s)H_u(s)H_m(s)} = \frac{L(s)}{1 + L(s)} = T(s) \quad (5.1)$$

which is the *tracking transfer function* of the control system. (If we had selected some other transfer function, for example the transfer function from the disturbance to the process output variable, the result of the analysis would have been the same.) $L(s)$ is the loop transfer function of the control system:

$$L(s) = H_c(s)H_u(s)H_m(s) \quad (5.2)$$

Figure 5.1 shows a compact block diagram of a control system. The transfer function from y_{mSP} to y_m is the tracking function:

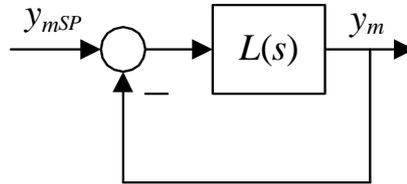


Figure 5.1: Compact block diagram of a control system with setpoint y_{mSP} as input variable and process measurement Y_m as output variable

$$T(s) = \frac{L(s)}{1 + L(s)} = \frac{\frac{n_L(s)}{d_L(s)}}{1 + \frac{n_L(s)}{d_L(s)}} = \frac{n_L(s)}{d_L(s) + n_L(s)} \quad (5.3)$$

where $n_L(s)$ and $d_L(s)$ are the numerator and denominator polynomials of $L(s)$, respectively. The *characteristic polynomial* of the tracking function is

$$c(s) = d_L(s) + n_L(s) \quad (5.4)$$

The stability of the control system is determined by the placement of the roots of (5.4) in the complex plane.

$K_p = 1$ (asympt. stable)	$K_p = 2$ (marg. stable)	$K_p = 4$ (unstable)
$p_1 = -1.75$	$p_1 = -2$	$p_1 = -2.31$
$p_2 = -0.12 + j0.74$	$p_2 = j$	$p_2 = 0.16 + j1.31$
$p_3 = -0.12 - j0.74$	$p_3 = -j$	$p_3 = 0.16 - j1.31$

Table 5.1: Poles of the tracking transfer function for various K_p -values**Example 5.1 Stability of a feedback control system**

See Figure ?? which shows a block diagram (transfer function based) of a control system. We assume that the individual transfer functions with parameter values are as follows:

$$H_{sm}(s) = K_{sm} = 1 \quad (5.5)$$

$$H_s(s) = K_s = 1 \quad (5.6)$$

$$H_c(s) = K_p \text{ (proportional controller)} \quad (5.7)$$

$$H_u(s) = \frac{1}{(s+1)^2 s} \quad (5.8)$$

$$H_d(s) = \frac{-1}{(s+1)^2 s} \quad (5.9)$$

The stability property of the control system can be determined from the placement of the poles of the tracking transfer function, $T(s)$, which is the transfer function from the reference y_{SP} to the process output variable y . The tracking transfer function is

$$T(s) = \frac{y(s)}{y_{SP}(s)} = \frac{H_{sm}(s)H_c(s)H_u(s)}{1 + H_s(s)H_c(s)H_u(s)} \quad (5.10)$$

Inserting (5.5) – (5.9) gives

$$H_{y_r,y}(s) = \frac{K_p}{s^3 + 2s^2 + s + K_p} \quad (5.11)$$

The characteristic polynomial of $T(s)$ is

$$a(s) = s^3 + 2s^2 + s + K_p \quad (5.12)$$

Table 5.1 shows the poles for three different K_p -values.²

Figure 5.2 shows the step response in y for the three K_p -values (it is a unit step in r).

[End of Example 5.1]

²The poles can be calculated using the MATLAB-function `roots` or `pzmap` or `pole` or using the LabVIEW-function `Complex Polynomial Roots`.

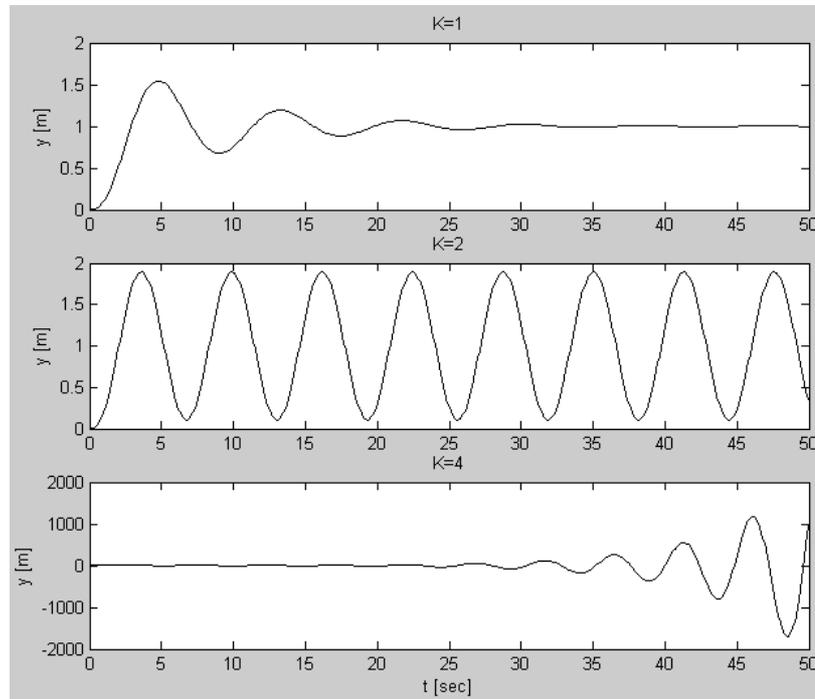


Figure 5.2: Example 5.1: Step response in the process output variable y for three different K_p -values

5.3 Nyquist's stability criterion

The Nyquist's stability criterion will now be derived. We start with a little rewriting: The roots of (5.4) are the same as the roots of

$$\frac{d_L(s) + n_L(s)}{d_L(s)} = 1 + \frac{n_L(s)}{d_L(s)} = 1 + L(s) = 0 \quad (5.13)$$

which, therefore, too can be denoted the characteristic equation of the control system. (5.13) is the equation from which the Nyquist's stability criterion will be derived. In the derivation we will use the Argument variation principle:

Argument variation principle: Given a function $f(s)$ where s is a complex number. Then $f(s)$ is a complex number, too. As with all complex numbers, $f(s)$ has an angle or argument. If s follows a closed contour Γ (gamma) in the complex s -plane which encircles a number of poles and a number of zeros of $f(s)$, see Figure 5.3, then

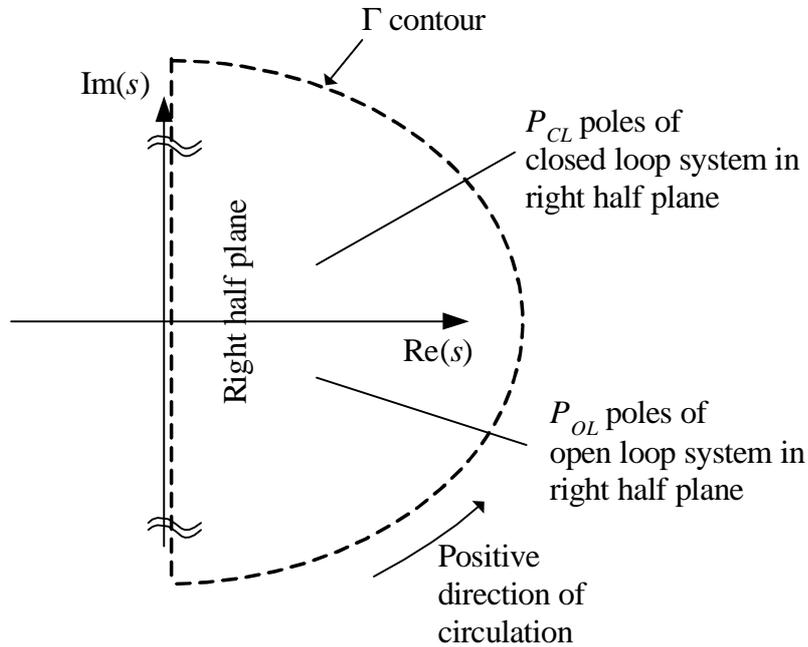


Figure 5.3: s shall follow the Γ contour once in positive direction (counter clockwise).

the following applies:

$$\arg_{\Gamma} f(s) = 360^{\circ} \cdot (\text{number of zeros minus number of poles of } f(s) \text{ inside } \Gamma) \quad (5.14)$$

where $\arg_{\Gamma} f(s)$ means the change of the angle of $f(s)$ when s has followed Γ once in positive direction of circulation (i.e. clockwise).

For our purpose, we let the function $f(s)$ in the Argument variation principle be

$$f(s) = 1 + L(s) \quad (5.15)$$

The Γ contour must encircle the entire right half s -plane, so that we are certain that all poles and zeros of $1 + L(s)$ are encircled. From the Argument Variation Principle we have:

$$\arg_{\Gamma}[1 + L(s)] = \arg_{\Gamma} \frac{d_L(s) + n_L(s)}{d_L(s)} \quad (5.16)$$

$$= 360^\circ \cdot (\text{number of roots of } (d_L + n_L) \text{ in RHP} \\ \text{minus number roots of } d_L \text{ in RHP}) \quad (5.17)$$

$$= 360^\circ \cdot (\text{number poles of closed loop system in RHP} \\ \text{minus number poles of open system in RHP}) \\ = 360^\circ \cdot (P_{CL} - P_{OL}) \quad (5.18)$$

where RHP means right half plane. By “open system” we mean the (imaginary) system having transfer function $L(s) = n_L(s)/d_L(s)$, i.e., the original feedback system with the feedback broken. The poles of the open system are the roots of $d_L(s) = 0$.

Finally, we can formulate the Nyquist’s stability criterion. But before we do that, we should remind ourselves what we are after, namely to be able to determine the number poles P_{CL} of the closed loop system in RHP. It those poles which determines whether the closed loop system (the control system) is asymptotically stable or not. *If $P_{CL} = 0$ the closed loop system is asymptotically stable.*

Nyquist’s stability criterion: Let P_{OL} be the number of poles of the open system in the right half plane, and let $\arg_{\Gamma} L(s)$ be the angular change of the vector $L(s)$ as s have followed the Γ contour once in positive direction of circulation. Then, the number poles P_{CL} of the closed loop system in the right half plane, is

$$P_{CL} = \frac{\arg_{\Gamma} L(s)}{360^\circ} + P_{OL} \quad (5.19)$$

If $P_{CL} = 0$, the closed loop system is asymptotically stable.

Let us take a closer look at the terms on the right side of (5.19): P_{OL} are the roots of $d_L(s)$, and there should not be any problem calculating these roots. To determine the angular change of the vector $1 + L(s)$. Figure 5.4 shows how the vector (or complex number) $1 + L(s)$ appears in a *Nyquist diagram* for a typical plot of $L(s)$. A Nyquist diagram is simply a Cartesian diagram of the complex plane in which L is plotted. $1 + L(s)$ is the vector from the point $(-1, 0j)$, which is denoted the *critical point*, to the Nyquist curve of $L(s)$.

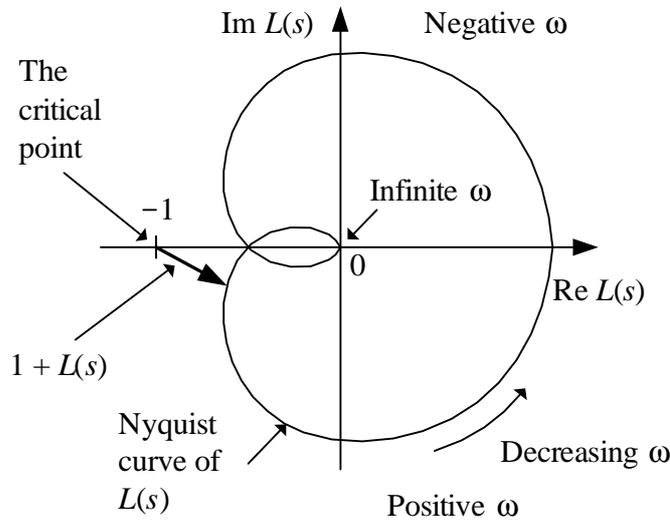


Figure 5.4: Typical Nyquist curve of $L(s)$. The vector $1 + L(s)$ is drawn.

More about the Nyquist curve of $L(j\omega)$

Let us take a more detailed look at the Nyquist curve of L as s follows the Γ contour in the s -plane, see Figure 5.3. In practice, the denominator polynomial of $L(s)$ has higher order than the numerator polynomial. This implies that $L(s)$ is mapped to the origin of the Nyquist diagram when $|s| = \infty$. Thus, the whole semicircular part of the Γ contour is mapped to the origin.

The imaginary axis constitutes the rest of the Γ contour. How is the mapping of $L(s)$ as s runs along the imaginary axis? On the imaginary axis $s = j\omega$, which implies that $L(s) = L(j\omega)$, which *is the frequency response of $L(s)$* . A consequence of this is that we can in principle determine the stability property of a feedback system by just looking at the frequency response of the open system, $L(j\omega)$.

ω has negative values when $s = j\omega$ is on the negative imaginary axis. For $\omega < 0$ the frequency response has a mathematical meaning. From general properties of complex functions,

$$|L(-j\omega)| = |L(j\omega)| \quad (5.20)$$

and

$$\angle L(-j\omega) = -\angle L(j\omega) \quad (5.21)$$

Therefore the Nyquist curve of $L(s)$ for $\omega < 0$ will be identical to the Nyquist curve of $\omega > 0$, but mirrored about the real axis. Thus, we only

need to know how $L(j\omega)$ is mapped for $\omega \geq 0$. The rest of the Nyquist curve then comes by itself! Actually we need not draw more of the Nyquist curve (for $\omega > 0$) than what is sufficient for determining if the critical point is encircled or not.

We must do some extra considerations if some of the poles in $L(s)$, which are the poles of the open loop system, lie in the origin. This corresponds to pure integrators in control loop, which is a common situation in feedback control systems because the controller usually has integral action, as in a PI or PID controller. If $L(s)$ contains integrators, the Γ contour must go *outside* the origin. But to the left or to the right? We choose to the right, see Figure 5.5. (We have thereby decided that the origin belongs to the left

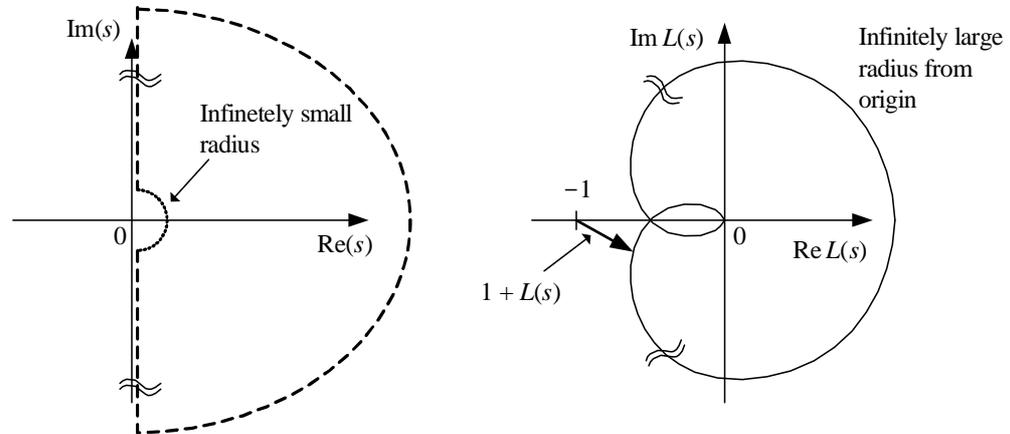


Figure 5.5: Left diagram: If $L(s)$ has a pole in origin, the Γ contour must pass the origin along an arbitrarily small semicircle to the right. Right diagram: A typical Nyquist curve of L .

half plane. This implies that P_{OL} does not count these poles.) The radius of the semicircle around origin is arbitrarily small. The Nyquist curve then becomes as shown in the diagram to the right in the same figure. The arbitrarily small semicircle in the s -plane is mapped to an infinitely large semicircle in the L -plane. This is because as $s \rightarrow 0$, the loop transfer function is approximately

$$L(s) \approx \frac{K}{s}$$

(if we assume one pole in the origin). On the small semicircle,

$$s = re^{j\theta} \quad (5.22)$$

which gives

$$L(s) \approx \frac{K}{r} e^{-j\theta} \quad (5.23)$$

When $r \rightarrow 0$ and when simultaneously θ goes from $+90^\circ$ via 0° to -90° , the Nyquist curve becomes an infinitely large semicircle, as shown.

The Nyquist's stability criterion for non-rational transfer functions

The Nyquist's stability criterion gives information about the poles of feedback systems. So far it has been assumed that the loop transfer function $L(s)$ is a rational transfer function. What if $L(s)$ is irrational? Here is one example:

$$L(s) = \frac{1}{s} e^{-\tau s} \quad (5.24)$$

where $e^{-\tau s}$ represents time delay. In such cases the tracking ratio $T(s)$ will also be irrational, and the definition of poles does not apply to such irrational transfer functions. Actually, the Nyquist's stability criterion can be used as a graphical method for determining the stability property on basis of the frequency response $L(j\omega)$.

Nyquist's special stability criterion

In most cases the open system is stable, that is, $P_{OL} = 0$. (5.19) then becomes

$$P_{CL} = \frac{\arg_{\Gamma}[L(s)]}{360^\circ} \quad (5.25)$$

This implies that the feedback system is asymptotically stable if the Nyquist curve does not encircle the critical point. This is the *Nyquist's special stability criterion* or the *Nyquist's stability criterion for open stable systems*.

The Nyquist's special stability criterion can also be formulated as follows: *The feedback system is asymptotically stable if the Nyquist curve of L has the critical point on its left side for increasing ω .*

Another way to formulate Nyquist's special stability criterion involves the *amplitude crossover frequency* ω_c and the *phase crossover frequency* ω_{180} . ω_c is the frequency at which the $L(j\omega)$ curve crosses the unit circle, while ω_{180} is the frequency at which the $L(j\omega)$ curve crosses the negative real axis. In other words:

$$|L(j\omega_c)| = 1 \quad (5.26)$$

and

$$\arg L(j\omega_{180}) = -180^\circ \quad (5.27)$$

See Figure 5.6. Note: The Nyquist diagram contains no explicit frequency axis. We can now determine the stability properties from the relation

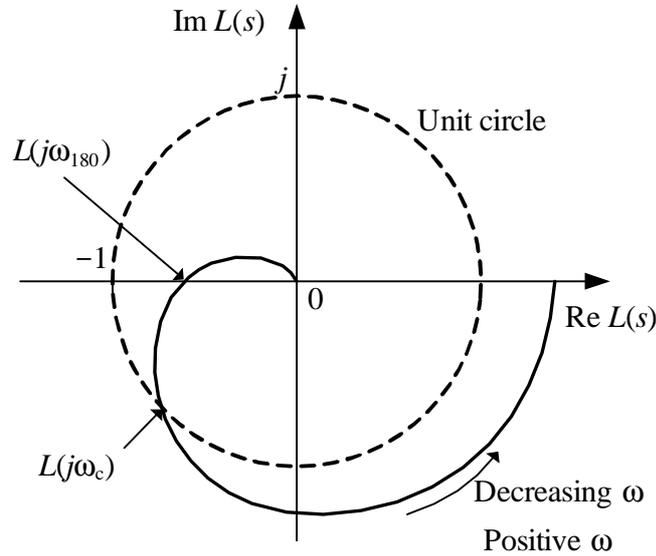


Figure 5.6: Definition of amplitude crossover frequency ω_c and phase crossover frequency ω_{180}

between these two crossover frequencies:

- Asymptotically stable closed loop system: $\omega_c < \omega_{180}$
- Marginally stable closed loop system: $\omega_c = \omega_{180}$
- Unstable closed loop system: $\omega_c > \omega_{180}$

The frequency of the sustained oscillations

There are sustained oscillations in a marginally stable system. *The frequency of these oscillations is $\omega_c = \omega_{180}$.* This can be explained as follows: In a marginally stable system, $L(\pm j\omega_{180}) = L(\pm j\omega_c) = -1$. Therefore, $d_L(\pm j\omega_{180}) + n_L(\pm j\omega_{180}) = 0$, which is the characteristic equation of the closed loop system with $\pm j\omega_{180}$ inserted for s . Therefore, the system has $\pm j\omega_{180}$ among its poles. The system usually have additional poles, but they lie in the left half plane. The poles $\pm j\omega_{180}$ leads to sustained sinusoidal oscillations. Thus, ω_{180} (or ω_c) is the frequency of the sustained oscillations in a marginally stable system.

5.4 Stability margins

5.4.1 Stability margins in terms of gain margin and phase margin

An asymptotically stable feedback system may become marginally stable if the loop transfer function changes. The *gain margin* GM and the *phase margin* PM [radians or degrees] are *stability margins* which in their own ways express how large parameter changes can be tolerated before an asymptotically stable system becomes marginally stable. Figure 5.7 shows the stability margins defined in the Nyquist diagram. GM is the

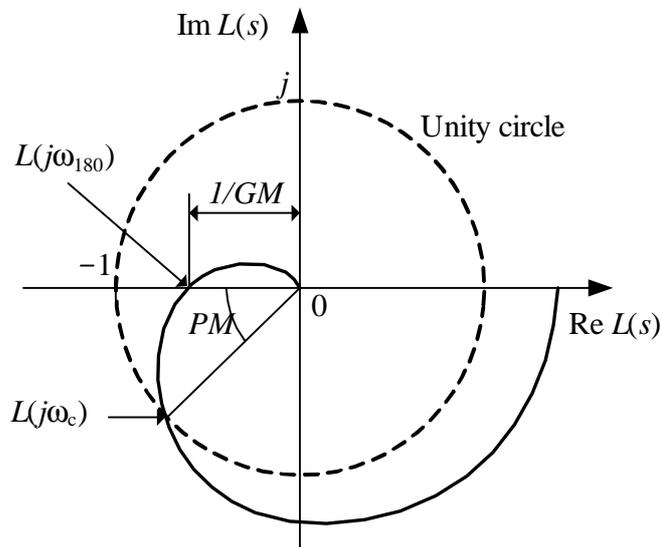


Figure 5.7: Gain margin GM and phase margin PM defined in the Nyquist diagram

(multiplicative, not additive) increase of the gain that L can tolerate at ω_{180} before the L curve (in the Nyquist diagram) passes through the critical point. Thus,

$$|L(j\omega_{180})| \cdot GM = 1 \quad (5.28)$$

which gives

$$GM = \frac{1}{|L(j\omega_{180})|} = \frac{1}{|\operatorname{Re} L(j\omega_{180})|} \quad (5.29)$$

(The latter expression in (5.29) is because at ω_{180} , $\operatorname{Im} L = 0$ so that the amplitude is equal to the absolute value of the real part.)

If we use decibel as the unit (like in the Bode diagram which we will soon encounter), then

$$GM \text{ [dB]} = -|L(j\omega_{180})| \text{ [dB]} \quad (5.30)$$

The phase margin PM is the phase reduction that the L curve can tolerate at ω_c before the L curve passes through the critical point. Thus,

$$\arg L(j\omega_c) - PM = -180^\circ \quad (5.31)$$

which gives

$$PM = 180^\circ + \arg L(j\omega_c) \quad (5.32)$$

We can now state as follows: The feedback (closed) system is asymptotically stable if

$$GM > 0\text{dB} = 1 \text{ and } PM > 0^\circ \quad (5.33)$$

This criterion is often denoted the *Bode-Nyquist stability criterion*.

Reasonable ranges of the stability margins are

$$2 \approx 6\text{dB} \leq GM \leq 4 \approx 12\text{dB} \quad (5.34)$$

and

$$30^\circ \leq PM \leq 60^\circ \quad (5.35)$$

The larger values, the better stability, but at the same time the system becomes more sluggish, dynamically. If you are to use the stability margins as design criterias, you can use the following values (unless you have reasons for specifying other values):

$$GM \geq 2.5 \approx 8\text{dB} \text{ and } PM \geq 45^\circ \quad (5.36)$$

For example, the controller gain, K_p , can be adjusted until one of the inequalities becomes an equality.³

It can be shown⁴ that for $PM \leq 70^\circ$, the damping of the feedback system approximately corresponds to that of a second order system with relative damping factor

$$\zeta \approx \frac{PM}{100^\circ} \quad (5.37)$$

For example, $PM = 50^\circ \sim \zeta = 0.5$.

³But you should definitely check the behaviour of the control system by simulation, if possible.

⁴The result is based on the assumption that the loop transfer function is $L(s) = \omega_0^2 / [(s + 2\zeta\omega_0)s]$ which gives tracking transfer function $T(s) = L(s) / [1 + L(s)] = \omega_0^2 / [s^2 + 2\zeta\omega_0s + \omega_0^2]$. The phase margin PM can be calculated from $L(s)$.

5.4.2 Stability margins in terms of maximum sensitivity amplitude

An alternative quantity of a stability margin is the minimum distance from the $L(j\omega)$ curve to the critical point. This distance is $|1 + L(j\omega)|$, see Figure 5.8. We can use the minimal value of $|1 + L(j\omega)|$ as a stability

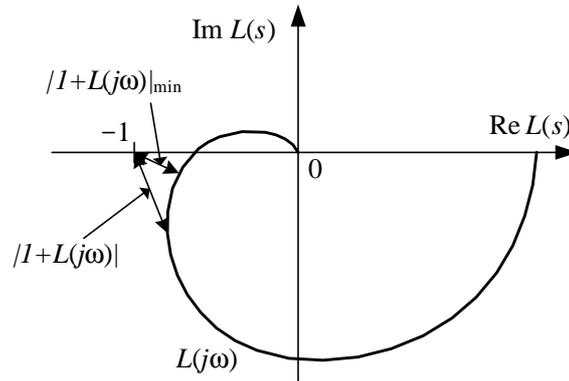


Figure 5.8: The distance between the $L(j\omega)$ curve and the critical point is $|1 + L|$. The minimum of this distance is related to the stability margin.

margin. However, it is more common to take the inverse of the distance: Thus, a stability margin is the *maximum* value of $1/|1 + L(j\omega)|$. And since $1/[1 + L(s)]$ is the sensitivity function $S(s)$, then $|S(j\omega)|_{\max}$ represents a stability margin. Reasonable values are in the range

$$1.5 \approx 3.5 \text{ dB} \leq |S(j\omega)|_{\max} \leq 3.0 \approx 9.5 \text{ dB} \quad (5.38)$$

If you use $|S(j\omega)|_{\max}$ as a criterion for adjusting controller parameters, you can use the following criterion (unless you have reasons for some other specification):

$$|S(j\omega)|_{\max} = 2.0 \approx 6 \text{ dB} \quad (5.39)$$

5.5 Stability analysis in a Bode diagram

It is most common to use a Bode diagram for frequency response based stability analysis of closed loop systems. The Nyquist's Stability Criterion says: The closed loop system is marginally stable if the Nyquist curve (of L) goes through the critical point, which is the point $(-1, 0)$. But where is the critical point in the Bode diagram? The critical point has phase (angle) -180° and amplitude $1 = 0\text{dB}$. The critical point therefore

constitutes two lines in a Bode diagram: The 0dB line in the amplitude diagram and the -180° line in the phase diagram. Figure 5.9 shows typical L curves for an asymptotically stable closed loop system. In the figure, GM , PM , ω_c and ω_{180} are indicated.

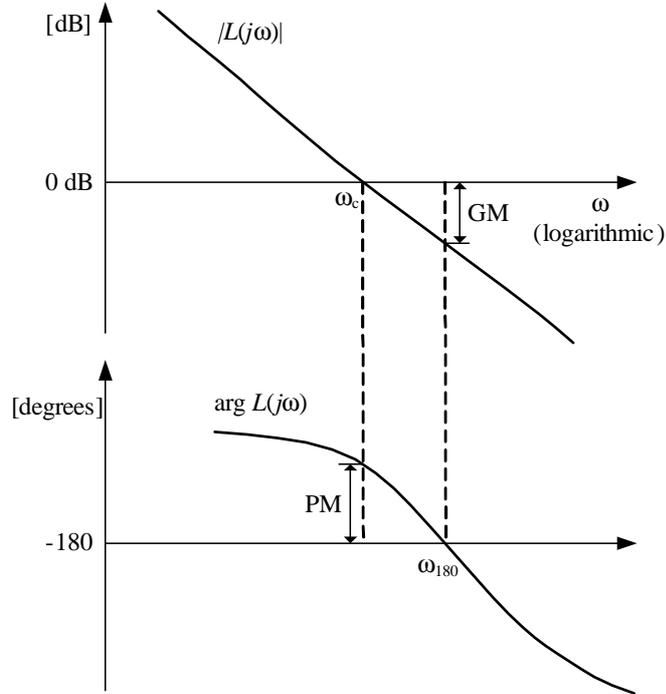


Figure 5.9: Typical L curves of an asymptotically stable closed loop system with GM , PM , ω_c and ω_{180} indicated

Example 5.2 Stability analysis of a feedback control system

Given a feedback control system with structure as shown in Figure 5.10. The loop transfer function is

$$L(s) = H_c(s)H_p(s) = \underbrace{K_p}_{H_c(s)} \underbrace{\frac{1}{(s+1)^2 s}}_{H_p(s)} = \frac{K_p}{(s+1)^2 s} = \frac{n_L(s)}{d_L(s)} \quad (5.40)$$

We will determine the stability property of the control system for different values of the controller gain K_p in three ways: Pole placement, Nyquist's Stability Criterion, and simulation. The tracking transfer function is

$$T(s) = \frac{y_m(s)}{y_{m_{SP}}(s)} = \frac{L(s)}{1+L(s)} = \frac{n_L(s)}{d_L(s) + n_L(s)} = \frac{K_p}{s^3 + 2s^2 + s + K_p} \quad (5.41)$$

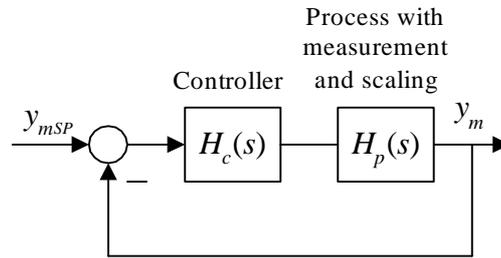


Figure 5.10: Example 5.2: Block diagram of feedback control system

The characteristic polynomial is

$$c(s) = s^3 + 2s^2 + s + K_p \quad (5.42)$$

Figures 5.11 – 5.13 show the step response after a step in the setpoint, the poles, the Bode diagram and Nyquist diagram for three K_p values which result in different stability properties. The detailed results are shown below.

- $K_p = 1$: Asymptotically stable system, see Figure 5.11. From the Bode diagram we read off stability margins $GM = 6.0\text{dB} = 2.0$ and $PM = 21^\circ$. we see also that $|S(j\omega)|_{\max} = 11\text{ dB} = 3.5$ (a large value, but it corresponds with the small the phase margin of $PM = 20^\circ$).
- $K_p = 2$: Marginally stable system, see Figure 5.12. From the Bode diagram, $\omega_c = \omega_{180}$. The L curve goes through the critical point in the Nyquist diagram. $|S|_{\max}$ has infinitely large value (since the minimum distance, $1/|S|_{\max}$, between $|L|$ and the critical point is zero).

Let us calculate the period T_p of the undamped oscillations: Since $\omega_{180} = 1.0\text{rad/s}$, the period is $T_p = 2\pi/\omega_{180} = 6.28\text{s}$, which fits well with the simulation shown in Figure 5.12.

- $K_p = 4$: Unstable system, see Figure 5.13. From the Bode diagram, $\omega_c > \omega_{180}$. From the Nyquist diagram we see that the L curve passes outside the critical point. (The frequency response curves of M and N have no physical meaning in this the case.)

[End of Example 5.2]

5.6 Robustness in term of stability margins

Per definition the stability margins expresses the robustness of the feedback control system against certain parameter changes in the loop transfer function:

- *The gain margin GM* is how much the loop gain, K , can increase before the system becomes unstable. For example, is $GM = 2$ when $K = 1.5$, the control system becomes unstable for K larger than $1.5 \cdot 2 = 3.0$.
- *The phase margin PM* is how much the phase lag function of the loop can be reduced before the loop becomes unstable. One reason of reduced phase is that the time delay in control loop is increased. A change of the time delay by $\Delta\tau$ introduces the factor $e^{-\Delta\tau s}$ in $L(s)$ and contributes to $\arg L$ with $-\Delta\tau \cdot \omega$ [rad] or $-\Delta\tau \cdot \omega \frac{180^\circ}{\pi}$ [deg]. $|L|$ is however not influenced because the amplitude function of $e^{-\tau s}$ is 1, independent of the value of τ . The system becomes unstable if the time delay have increased by $\Delta\tau_{\max}$ such that⁵

$$PM = \Delta\tau_{\max} \cdot \omega_c \frac{180^\circ}{\pi} \text{ [deg]} \quad (5.43)$$

which gives the following maximum change of the time delay:

$$\Delta\tau_{\max} = \frac{PM}{\omega_c} \frac{\pi}{180^\circ} \quad (5.44)$$

If you want to calculate how much the phase margin PM is reduced if the time delay is increased by $\Delta\tau$, you can use the following formula which stems from (5.43):

$$\Delta PM = \Delta\tau \cdot \omega_c \frac{180^\circ}{\pi} \text{ [deg]} \quad (5.45)$$

For example, assume that a given control system has $\omega_c = 0.2\text{rad/min}$ and $PM = 50^\circ$. If the time delay increases by 1min, the phase margin is reduced by $\Delta PM = 1 \cdot 0.2 \frac{180^\circ}{\pi} = 11.4^\circ$, i.e. from 50° to 38.6° .

⁵Remember that PM is found at ω_c .

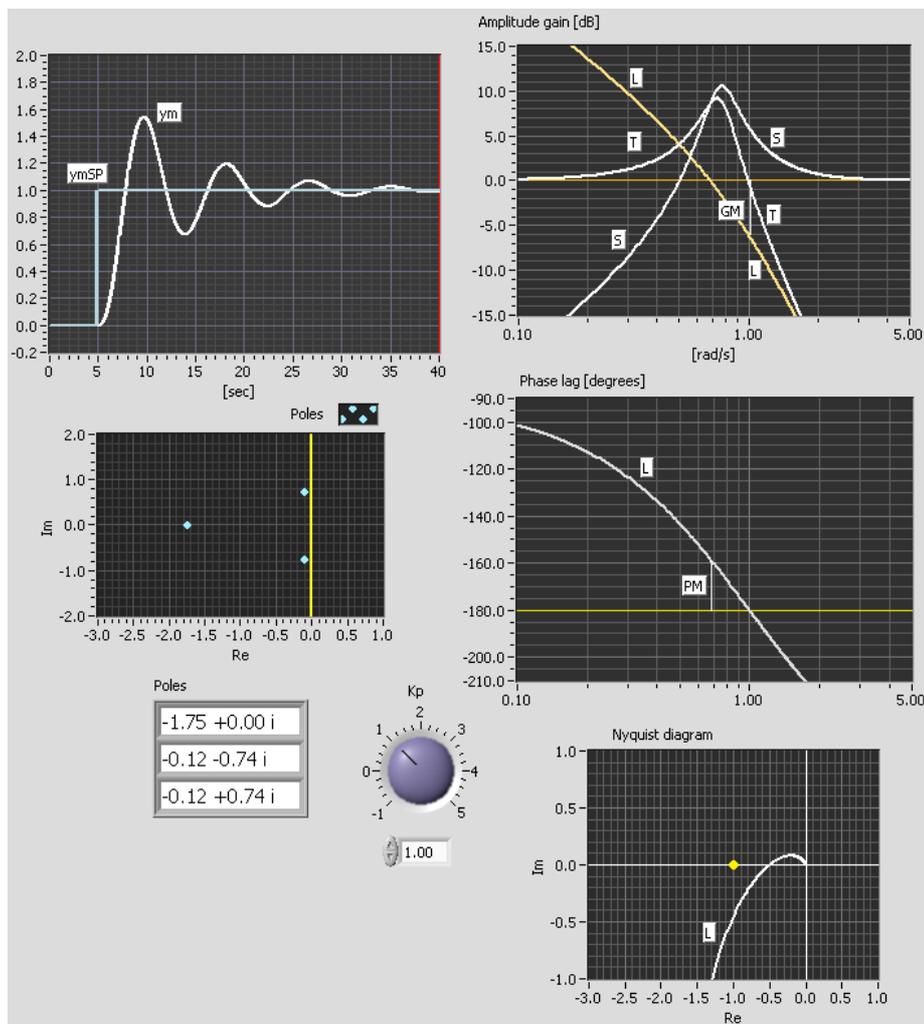


Figure 5.11: Example 5.2: Step response (step in setpoint), poles, Bode diagram and Nyquist diagram with $K_p = 1$. The control system is asymptotically stable.

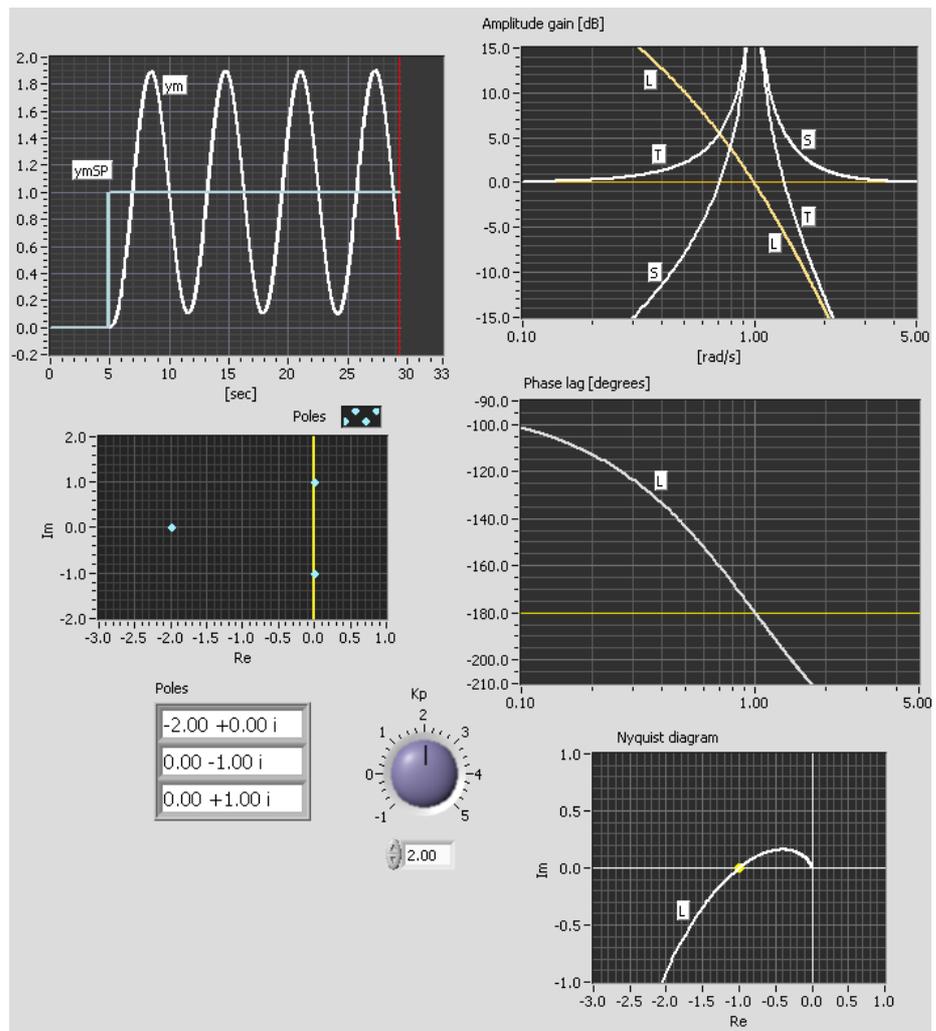


Figure 5.12: Example 5.2: Step response (step in setpoint), poles, Bode diagram and Nyquist diagram with $K_p = 2$. The control system is marginally stable.

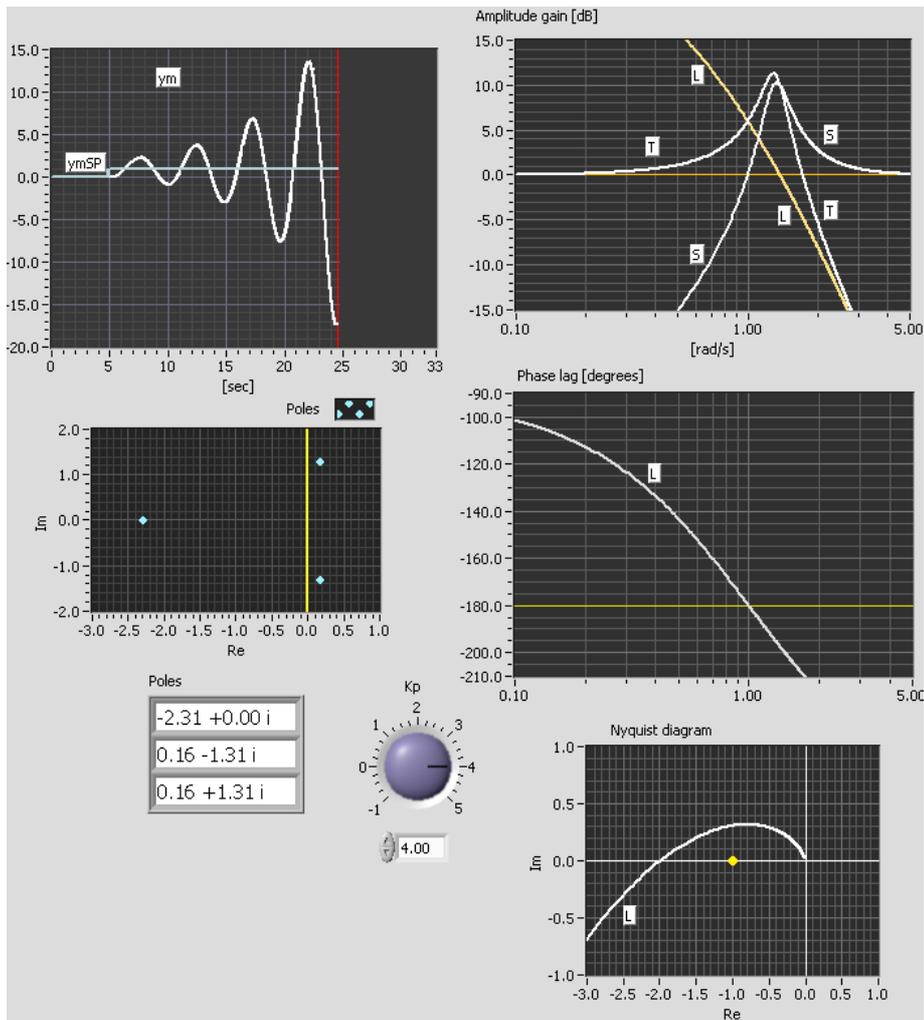


Figure 5.13: Example 5.2: Step response (step in setpoint), poles, Bode diagram and Nyquist diagram with $K_p = 4$. The control system is unstable.

Part II

**DISCRETE-TIME
SYSTEMS THEORY**

Chapter 6

Discrete-time signals

Assume that an AD-converter (analog-digital) at discrete points of time converts an analog signal $y_a(t)$, which can be a voltage signal from a temperature or speed sensor, to an equivalent digital signal, $y_d(t_k)$, in the form of a number to be used in operations in the computer, see Figure 6.1.

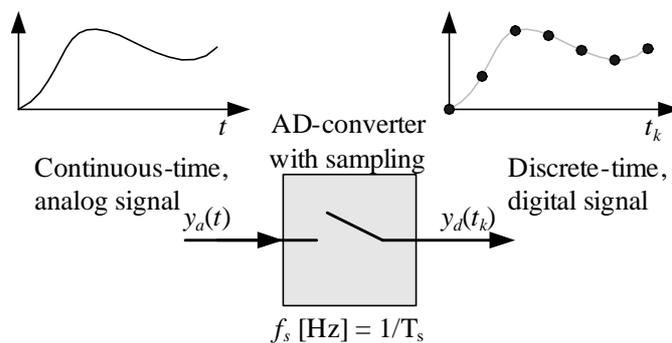


Figure 6.1: Sampling. T_s is the time step between the samplings, or the sampling interval.

(The AD-converter is a part of the interface between the computer and the external equipment, e.g. sensors.) As indicated in Figure 6.1 the resulting *discrete-time signal* is a sequence or a series of signal values defined in discrete points of time. T_s is the time step between the samplings, or the sampling interval. Figure 6.2 shows this signal in more detail. The discrete points of time may be denoted t_k where k is an integer *time index*. The time series can be written in various ways:

$$\{x(t_k)\} = \{x(kT_s)\} = \{x(k)\} = x(0), x(1), x(2), \dots \quad (6.1)$$

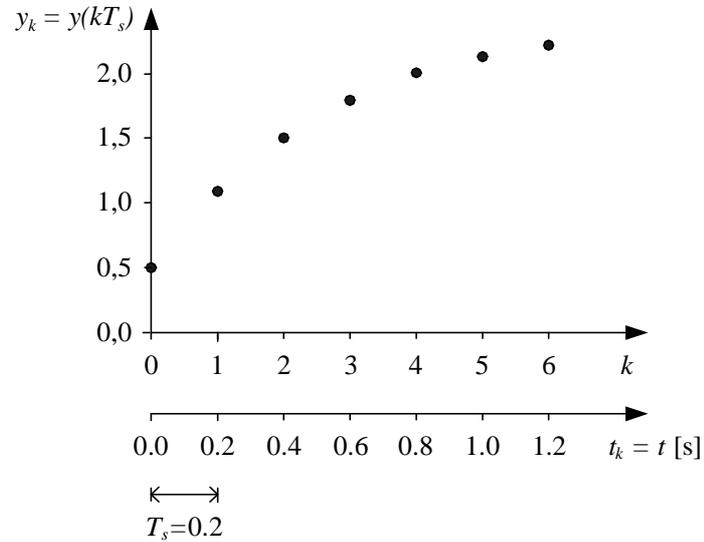


Figure 6.2: Discrete-time signal

To make the notation simple, we can write the signal in one of the following ways:

$$x(t_k) \quad (6.2)$$

$$x(kT_s) \quad (6.3)$$

$$x(k) \quad (6.4)$$

$$x_k \quad (6.5)$$

In the example above, the discrete-time signal originated from *sampling* of a continuous-time signal. However, discrete-time signals exists in many other circumstances, for example,

- the output signal from a discrete-time (computer-based) *signal filter*, for example a lowpass filter,
- the output from a discrete-time (computer-based) *controller* which controls a physical process,
- the response in a dynamic system as calculated by a (computer-based) *simulator*.

Chapter 7

Difference equations

7.1 Difference equation models

The basic model type of continuous-time dynamic systems is the differential equation. Analogously, the basic model type of discrete-time dynamic systems is the *difference equation*. Here is an example of a linear second order difference equation with u as input variable and y as output variable:

$$y(t_{k+2}) + a_1y(t_{k+1}) + a_0y(t_k) = b_0u(t_k) \quad (7.1)$$

which may be written somewhat simpler as

$$y(k + 2) + a_1y(k + 1) + a_0y(k) = b_0u(k) \quad (7.2)$$

where a_i and b_j are *coefficients* of the difference equation, or model *parameters*. Note that this difference equation has unique coefficients since the coefficient of $y(k + 2)$ is 1.

One equivalent form (7.2) is

$$y(k) + a_1y(k - 1) + a_0y(k - 2) = b_0u(k - 2) \quad (7.3)$$

where there are *no time delayed* terms (no negative time indexes), only time advanced terms (positive or zero time indexes). This form can be obtained from (7.2) by increasing each time index in (7.2) by 2.

In most cases we want to write the difference equation as a formula for the output variable. In our example the formula for the output $y(k)$ can be obtained by solving for $y(k)$ from (7.3):

$$y(k) = -a_1y(k - 1) - a_0y(k - 2) + b_0u(k - 2) \quad (7.4)$$

(7.4) says that the output $y(k)$ is given as a linear combination of the output one time step back in time, $y(k-1)$, the output two time steps back in time, $y(k-2)$, and the input two time steps back in time, $u(k-2)$.

7.2 Calculating responses from difference equation models

For example, (7.4) is a formula for calculating dynamic (time-varying) responses in the output, $y(k)$. The formula must be calculated once per time step, and it can be implemented in a While loop or a For loop in a computer program. Assume as an example that $y(1)$, $y(0)$ and $u(0)$ are zero. Then (7.4) gives

$$y(2) = -a_1y(1) - a_0y(0) + b_0u(0) \quad (7.5)$$

$$y(3) = -a_1y(2) - a_0y(1) + b_0u(1) \quad (7.6)$$

$$y(4) = -a_1y(3) - a_0y(2) + b_0u(2) \quad (7.7)$$

and so on.

The *static response* – which is the (steady-state) response of the system when all variables are assumed to have constant values – can be calculated from the static version of the difference equation. The static version is found by neglecting all time-dependencies in the difference equation, and setting $y(k) = y_s$, $y(k-1) = y_s$ etc. where subindex s is for static. For example, the static version of (7.4) is

$$y_s = -a_1y_s - a_0y_s + b_0u_s \quad (7.8)$$

The static response is

$$y_s = \frac{b_0}{1 + a_1 + a_0} u_s \quad (7.9)$$

Chapter 8

Discretizing continuous-time models

8.1 Simple discretization methods

Typical applications of difference equation models are computer-based implementations of

- simulators,
- signal filters,
- controllers.

Often the simulator model, the filter, or the controller, is originally given as a continuous-time model in the form of a differential equation or a Laplace transfer function. To obtain a corresponding differential equation (ready for implementation in a computer program), you have to discretize the continuous-time model. This approximation can be made in many ways. My experience is that in most applications it is sufficient to apply one of the following (simple) methods, which are based on approximations of the time derivatives of the differential equation:

- **(Euler's) Forward differentiation method**, which is commonly used in developing simple simulators.
- **(Euler's) Backward differentiation method**, which is commonly used in discretizing simple signal filters and industrial controllers.

The Forward differentiation method and the Backward differentiation method will be explained in detail below. But you should at least have heard about some other methods as well, see below [2]:

- *Zero Order Hold (ZOH) method*: It is assumed that the system has a zero order hold element on the input side of the system. This is the case when the physical system is controlled by a computer via a DA converter (digital to analog). Zero order hold means that the physical input signal to the system is held fixed between the discrete points of time. The discretization method is relatively complicated to apply, and in practice you will probably use a computer tool (e.g. MATLAB or LabVIEW) to do the job.
- *Tustin's method*: This method is based on an integral approximation where the integral is interpreted as the area between the integrand and the time axis, and this area is approximated with trapezoids. (The Euler's methods approximates this area with a rectangle.)
- *Tustin's method with frequency prewarping, or Bilinear transformation*: This is the Tustin's method but with a modification so that the frequency response of the original continuous-time system and the resulting discrete-time system have exactly the same frequency response at one or more specified frequencies.

Some times you want to go the opposite way – transform a discrete-time model into an equivalent continuous-time model. Such methods will however not be described in this document.

The Forward differentiation method is somewhat less accurate than the Backward differentiation method, but it is simpler to use. Particularly, with nonlinear models the Backward differentiation method may give problems since it results in an implicit equation for the output variable, while the Forward differentiation method always gives an explicit equation (the nonlinear case will be demonstrated in an example).

Figure 8.1 illustrates both the Forward and the Backward differentiation methods. The Forward differentiation method can be seen as the following approximation of the time derivative of a time-valued function which here is denoted x :

$$\text{Forward differentiation method: } \dot{x}(t_k) \approx \frac{x(t_{k+1}) - x(t_k)}{T_s} \quad (8.1)$$

T_s is the time step, i.e. the time interval between two subsequent points of time. The name “Forward differentiation method” stems from the $x(t_{k+1})$ term in (8.1).

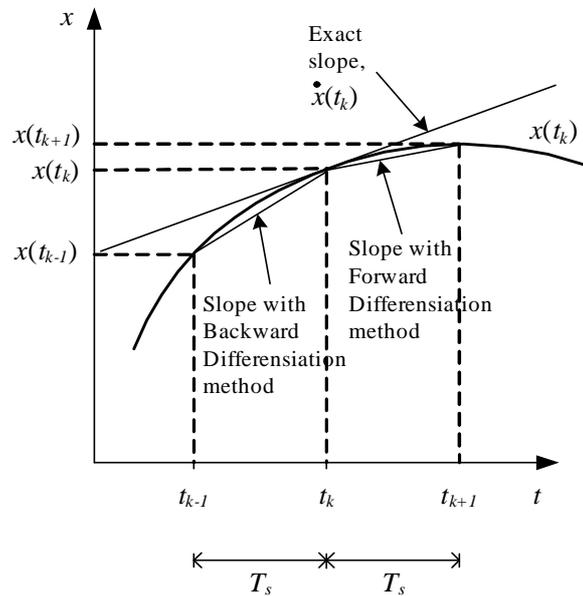


Figure 8.1: The Forward differentiation method and the Backward differentiation method

The Backward differentiation method is based on the following approximation of the time derivative:

$$\text{Backward differentiation method: } \dot{x}(t_k) \approx \frac{x(t_k) - x(t_{k-1})}{T_s} \quad (8.2)$$

The name “Backward differentiation method” stems from the $x(t_{k-1})$ term in (8.2), see Figure 8.1.

The examples in the subsequent sections demonstrate the application of the Forward differentiation method and the Backward differentiation method. It is also demonstrated how to get an equivalent differential equation from an original transfer function model or an integral equation (the time derivatives of the differential equation is then approximated with the Forward differentiation method or the Backward differentiation method).

8.2 Discretizing a simulator of a dynamic system

A simulator of a dynamic system, e.g. a motor, liquid tank, a ship etc., must of course be based on the mathematical model of the system. Typically, the model is in the form of a nonlinear differential equation

model. The Forward differentiation method may be used to discretize such nonlinear models.

As an example, let us discretize the following nonlinear model:

$$\dot{x}(t) = -K_1\sqrt{x(t)} + K_2u(t) \quad (8.3)$$

where u is the input, x is the output, and K_1 and K_2 are parameters.¹ We will now derive a simulator algorithm or formula for $x(t_k)$. Let us first try applying the Backward differentiation method with time step h to the time derivative in (8.3):

$$\frac{x(t_k) - x(t_{k-1})}{T_s} = -K_1\sqrt{x(t_k)} + K_2u(t_k) \quad (8.4)$$

$x(t_k)$ appears on both sides of (8.4). We say that $x(t_k)$ is given implicitly – not explicitly – by (8.4). Solving for $x(t_k)$ in this implicit equation is possible, but a little difficult because of the nonlinear function (the square root). (If the difference equation was linear, it would be much easier to solve for $x(t_k)$.) In other cases, nonlinear functions may cause big problems in solving for the output variable, here $x(t_k)$.

Since we got some problems with the Backward differentiation method in this case, let us instead apply the Forward differentiation method to the time derivative of (8.3):

$$\frac{x(t_{k+1}) - x(t_k)}{T_s} = -K_1\sqrt{x(t_k)} + K_2u(t_k) \quad (8.5)$$

Solving for $x(t_{k+1})$ is easy:

$$x(t_{k+1}) = x(t_k) + T_s \left[-K_1\sqrt{x(t_k)} + K_2u(t_k) \right] \quad (8.6)$$

Reducing each time index by one and using the simplifying notation $x(t_k) = x(k)$ etc. finally gives the simulation algorithm:

$$x(k) = x(k-1) + T_s \left[-K_1\sqrt{x(k-1)} + K_2u(k-1) \right] \quad (8.7)$$

In general it is important that the time-step T_s of the discrete-time function is relatively small, so that the discrete-time function behaves approximately similar to the original continuous-time system. For the Forward differentiation method a (too) large time-step may even result in

¹This can be the model of a liquid tank with pump inflow and valve outflow. x is the level. u is the pump control signal. The square root stems from the valve.

an unstable discrete-time system! For simulators the time-step T_s should be selected according to

$$T_s \leq \frac{0.1}{|\lambda|_{\max}} \quad (8.8)$$

Here $|\lambda|_{\max}$ is the largest of the absolute values of the eigenvalues of the model, which is the eigenvalues of the system matrix A in the state-space model $\dot{\underline{x}} = A\underline{x} + B\underline{u}$. For transfer function models you can consider the poles in stead of the eigenvalues (the poles and the eigenvalues are equal for most systems not having pol-zero cancellations). If the model is nonlinear, it must be linearized before calculating eigenvalues or poles.

In stead of, or as a supplementary using However, you may also use a trial-and-error method for choosing T_s (or f_s): *Reduce h until there is a negligible change of the response of the system if T_s is further reduced.* If possible, you should use a simulator of your system to test the importance of the value of T_s before implementation.

8.3 Discretizing a signal filter

A lowpass filter is used to smooth out high frequent or random noise in a measurement signal. A very common lowpass filter in computer-based control systems is the discretized first order filter – or time-constant filter. You can derive such a filter by discretizing the Laplace transfer function of the filter. A common discretization method in control applications is the Backward differentiation method. We will now derive a discrete-time filter using this method.

The Laplace transform transfer function – also denoted the continuous-time transfer function – of a first order lowpass filter is

$$H(s) = \frac{y(s)}{u(s)} = \frac{1}{T_f s + 1} = \frac{1}{\frac{s}{\omega_b} + 1} = \frac{1}{\frac{s}{2\pi f_b} + 1} \quad (8.9)$$

Here, u is the filter input, and y is the filter output. T_f [s] is the time-constant. ω_b is the filter bandwidth in rad/s, and f_b is the filter bandwidth in Hz. (In the following, the time-constant will be used as the filter parameter since this is the parameter typically used in filter implementations for control systems.)

Cross-multiplying in (8.9) gives

$$(T_f s + 1) y(s) = u(s) \quad (8.10)$$

Resolving the parenthesis gives

$$T_f s y(s) + y(s) = u(s) \quad (8.11)$$

Taking the inverse Laplace transform of both sides of this equation gives the following differential equation (because multiplying by s means time-differentiation in the time-domain):

$$T_f \dot{y}(t) + y(t) = u(t) \quad (8.12)$$

Let us use t_k to represent the present point of time – or discrete time:

$$T_f \dot{y}(t_k) + y(t_k) = u(t_k) \quad (8.13)$$

Substituting the time derivative by the Backward differentiation approximation gives

$$T_f \frac{y(t_k) - y(t_{k-1})}{T_s} + y(t_k) = u(t_k) \quad (8.14)$$

Solving for $y(t_k)$ gives

$$y(t_k) = \frac{T_f}{T_f + T_s} y(t_{k-1}) + \frac{T_s}{T_f + T_s} u(t_k) \quad (8.15)$$

which is commonly written as

$$y(t_k) = (1 - a) y(t_{k-1}) + a u(t_k) \quad (8.16)$$

with filter parameter

$$a = \frac{T_s}{T_f + T_s} \quad (8.17)$$

which has a given value once you have specified the filter time-constant T_f and the time-step T_s is given. (8.16) is the formula for the filter output. It is ready for being programmed. This filter is denoted the *exponentially weighted moving average (EWMA) filter*, but we can simply denote it a first order lowpass filter.

It is important that the filter time-step T_s is considerably smaller than the filter time-constant T_f , otherwise the filter may behave quite differently from the original continuous-time filter (8.9) from which it is derived. A rule of thumb for the upper limit of T_s is

$$T_s \leq \frac{T_f}{5} \quad (8.18)$$

8.4 Discretizing a PID controller

8.4.1 Computer based control loop

Figure 8.2 shows a control loop where controller is implemented in a computer. The computer registers the process measurement signal via an AD converter (from analog to digital). The AD converter produces a numerical value which represents the measurement. As indicated in the block diagram this value may also be scaled, for example from volts to percent. The resulting digital signal, $y(t_k)$, is used in the control function, which is in the form of a computer algorithm or program calculating the value of the control signal, $u(t_k)$.

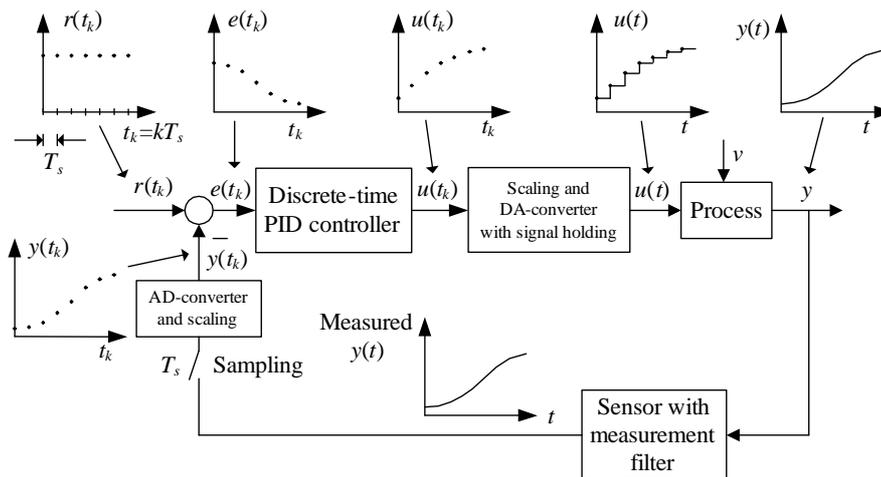


Figure 8.2: Control loop where the controller function is implemented in a computer

The control signal is scaled, for example from percent to milliamperes, and sent to the DA converter (from digital to analog) where it is held constant during the present time step. Consequently the control signal becomes a staircase signal. The time step or the sampling interval, T_s [s], is usually small compared to the time constant of the actuator (e.g. a valve) so the actuator does not feel the staircase form of the control signal. A typical value of T_s in commercial controllers is 0.1 s.

8.4.2 Development of discrete-time PID controller

The starting point of deriving the discrete-time PID controller is the continuous-time PID (proportional + integral + derivate) controller:

$$u(t) = u_0 + K_p e(t) + \frac{K_p}{T_i} \int_0^t e \, d\tau + K_p T_d \dot{e}_f(t) \quad (8.19)$$

where u_0 is the control bias or manual control value (to be adjusted by the operator when the controller is in manual mode), u is the controller output (the control variable), e is the control error:

$$e(t) = r(t) - y(t) \quad (8.20)$$

where r is the reference or setpoint, and y is the process measurement. e_f is the filtered control error. It is the output of the following lowpass filter:

$$e_f(s) = \frac{1}{T_f s + 1} e(s) \quad (8.21)$$

where T_f is the filter time-constant, which is typically selected as

$$T_f = a T_d \quad (8.22)$$

where typically $a = 0.1$.

We will now derive a discrete-time formula for $u(t_k)$, the value of the control signal for the present time step. The discretization can be performed in a number of ways. Probably the simplest way is as follows: Differentiating both sides of (8.19) gives²

$$\dot{u}(t) = \dot{u}_0 + K_p \dot{e}(t) + \frac{K_p}{T_i} e(t) + K_p T_d \ddot{e}_f(t) \quad (8.23)$$

Applying the Backward differentiation method (8.2) to \dot{u} , \dot{e} , and \ddot{e}_f gives

$$\frac{u(t_k) - u(t_{k-1})}{T_s} = \frac{u_0(t_k) - u_0(t_{k-1})}{T_s} \quad (8.24)$$

$$+ K_p \frac{e(t_k) - e(t_{k-1})}{T_s} \quad (8.25)$$

$$+ \frac{K_p}{T_i} e(t_k) \quad (8.26)$$

$$+ K_p T_d \frac{\dot{e}_f(t_k) - \dot{e}_f(t_{k-1})}{T_s} \quad (8.27)$$

²The time derivative of an integral is the integrand.

Applying the Backward differentiation method on $\dot{e}_f(t_k)$ and $\dot{e}_f(t_{k-1})$ in (8.24) gives

$$\frac{u(t_k) - u(t_{k-1})}{T_s} = \frac{u_0(t_k) - u_0(t_{k-1})}{T_s} \quad (8.28)$$

$$+ K_p \frac{e(t_k) - e(t_{k-1})}{T_s} \quad (8.29)$$

$$+ \frac{K_p}{T_i} e(t_k) \quad (8.30)$$

$$+ K_p T_d \frac{\frac{e_f(t_k) - e_f(t_{k-1})}{T_s} - \frac{e_f(t_{k-1}) - e_f(t_{k-2})}{T_s}}{T_s} \quad (8.31)$$

Solving for $u(t_k)$ finally gives the discrete-time PID controller:

$$u(t_k) = u(t_{k-1}) + [u_0(t_k) - u_0(t_{k-1})] \quad (8.32)$$

$$+ K_p [e(t_k) - e(t_{k-1})] \quad (8.33)$$

$$+ \frac{K_p T_s}{T_i} e(t_k) \quad (8.34)$$

$$+ \frac{K_p T_d}{T_s} [e_f(t_k) - 2e_f(t_{k-1}) + e_f(t_{k-2})] \quad (8.35)$$

The discrete version of the filter (8.21) can be derived as described in Section 8.3.

The discrete-time PID controller algorithm (8.32) is denoted the *absolute* or *positional* algorithm. Automation devices typically implements the *incremental* or *velocity* algorithm. because it has some benefits. The incremental algorithm is based on splitting the calculation of the control value into two steps:

1. First the *incremental control value* $\Delta u(t_k)$ is calculated:

$$\Delta u(t_k) = [u_0(t_k) - u_0(t_{k-1})] \quad (8.36)$$

$$+ K_p [e(t_k) - e(t_{k-1})] \quad (8.37)$$

$$+ \frac{K_p T_s}{T_i} e(t_k) \quad (8.38)$$

$$+ \frac{K_p T_d}{T_s} [e_f(t_k) - 2e_f(t_{k-1}) + e_f(t_{k-2})] \quad (8.39)$$

2. Then the *total or absolute control value* is calculated with

$$u(t_k) = u(t_{k-1}) + \Delta u(t_{k-1}) \quad (8.40)$$

The summation (8.40) implements the (numerical) integral action of the PID controller.

The incremental PID control function is particularly useful if the actuator is controlled by an incremental signal. A step-motor is such an actuator. The motor itself implements the numerical integration (8.40). It is (only) $\Delta u(t_k)$ that is sent to the motor.

8.4.3 Some practical features of the PID controller

A practical PID controller must have certain features to be functional:

- **Integrator anti windup:** Large excitations of the control system, typically large disturbances or large setpoint changes, may cause the control signal to reach its maximum or minimum limits with the control error being different from zero. The summation in (8.40), which is actually a numerical integration, will then cause u to increase (or decrease) steadily – this is denoted *integral windup* – so that u may get a very high (or low) value. When the excitations are back to normal values, it may take a very long time before the large value of u is integrated back to a normal value (i.e. within 0 – 100%), causing the process output to deviate largely from the setpoint.

Preventing the windup is (not surprisingly) denoted *anti windup*, and it can be realized as follows:

1. Calculate an intermediate value of the control variable $u(t_k)$ according to (8.40), but do not send this value to the DA (Digital-to-Analog) converter.
2. Check if this intermediate value is greater than the maximum value u_{\max} (typically 100%) or less than the minimum value u_{\min} (typically 0%). If it exceeds one of these limits, set $\Delta u(t_k)$ in (8.40) to zero.
3. Write $u(t_k)$ to the DA converter.

- **Bumpless transfer:** Suppose the controller is switched from automatic to manual mode, or from manual to automatic mode (this will happen during maintenance, for example). The transfer between modes should be bumpless, ideally. Bumpless transfer can be realized as follows:

- *Bumpless transfer from automatic to manual mode:* In manual mode it is only the manual (or nominal) control signal u_0 – adjusted by the operator – that controls the process. Manual mode is equivalent to setting the controller gain to zero (or multiplying the gain by zero). We assume here that the control signal $u(t_k)$ in (8.40) has a proper value, say u_{good} , so that the control error is small, immediately before the switch to manual mode. To implement bumpless transfer, set $u(t_{k-1})$ in (8.40) equal to u_{good} immediately after the switching moment.
- *Bumpless transfer from manual to automatic mode:* Nothing special has to be done during the switching except activating all term in (8.36) – (8.39).

8.4.4 Selecting the sampling time of the control system

The DA converter (digital to analog) which is always between the discrete-time control function and the continuous-time process to be controlled, implements holding of the calculated control signal during the time-step (sampling interval). This holding implies that the control signal is *time delayed* approximately $T_s/2$, see Figure 8.3. The delay influences

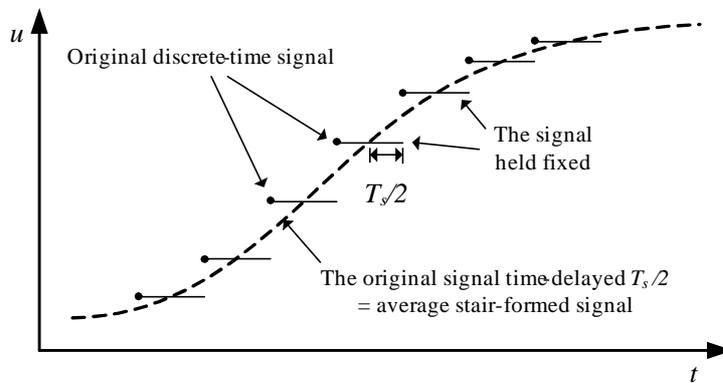


Figure 8.3: The DA-converter holds the calculated control signal throughout the sampling interval, thereby introducing an approximate time-delay of $T_s/2$.

the stability of the control loop. Suppose we have tuned a continuous-time PID controller, and apply these PID parameters on a discrete-time PID controller. Then the control loop will get reduced stability because of the approximate delay of $T_s/2$. As a rule of thumb (this can be confirmed from a frequency response based analysis), the stability reduction is small and tolerable if the time delay is less than *one tenth of the response-time* of the

control system as it would have been with a continuous-time controller or a controller having very small sampling time:

$$\frac{T_s}{2} \leq \frac{T_r}{10} \quad (8.41)$$

which gives

$$T_s \leq \frac{T_r}{5} \quad (8.42)$$

The response time is here the 63% rise time which can be read off from the setpoint step response. For a system the having dominating time constant T , the response-time is approximately equal to this time constant.

Chapter 9

Discrete-time state space models

9.1 General form of discrete-time state space models

The general form of a discrete-time state space model is

$$x(k+1) = f[x(k), u(k)] \quad (9.1)$$

$$y(k) = g[x(k), u(k)] \quad (9.2)$$

where x is the state variable, u is the input variable which may consist of control variables and disturbances (in this model definition there is no difference between these two kinds of input variables). y is the output variable. f and g are functions – linear or nonlinear. $x(k+1)$ in (9.1) means the state one time-step ahead (relative to the present state $x(k)$). Thus, the state space model expresses how the systems' state (variables) and output variables evolves along the discrete time axis.

The variables in (9.1) – (9.2) may actually be vectors, e.g.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (9.3)$$

where x_i is a (scalar) state variable, and if so, f and/or g are vector evaluated functions.

9.2 Linear discrete-time state space models

A special case of the general state space model presented above is the *linear* state space model:

$$x(k+1) = Ax(k) + Bu(k) \quad (9.4)$$

$$y(k) = Cx(k) + Du(k) \quad (9.5)$$

where A is the transition matrix, B is the input gain matrix, C is the output gain matrix or measurement gain matrix and D is the direct output gain matrix (in most cases, $D = 0$).

9.3 Discretization of continuous-time state space models

9.3.1 Discretization of non-linear continuous-time state-space models

Here are some situations where you need to discretize a continuous-time *non-linear* state-space model:

- Creating a simulation algorithm from a process model. This was actually described in Section 8.2.
- Defining the process model to be used as the basis of a state estimator in form of an observer, cf. Chapter 17, or a Kalman Filter, cf. Chapter 18.

The Forward Discretization method, cf. Section 8.1, is the simplest, most commonly used, and the most flexible method. Only this method will be described here.

Given this continuous-time state space model

$$\dot{x}(t) = f_c[x(t), u(t)] \quad (9.6)$$

Approximating the time derivative with Forward differentiation gives

$$\frac{x(t_{k+1}) - x(t_k)}{T_s} = f_c[x(t_k), u(t_k)] \quad (9.7)$$

Solving for $x(t_{k+1})$ gives

$$x(t_{k+1}) = x(t_k) + T_s f_c [x(t_k), u(t_k)] \quad (9.8)$$

which is a discrete-time state-space model, which is non-linear if f_c is a non-linear function of x and u .

9.3.2 Discretization of linear continuous-time state-space models

One situation where you need to discretize a continuous-time *linear* state-space model is in calculation of the Kalman gain of a Kalman Filter state estimator, cf. Chapter 18.

The most appropriate discretization methods are the *Forward difference method*, cf. 8.1, and the *Zero-order hold method*. These are described in the following.

Forward difference method

If you are going to implement the discretization “manually”, you may use the Forward difference method, cf. 8.1, because it is simple to use, and gives a discrete-time state-space model with no direct term in the output equation, i.e. the matrix D in $y(k) = Cx(k) + Du(k)$ is zero, which may be beneficial (simplifying) in controller and estimator designs. However, the dynamic properties – e.g. the stability – of the resulting model may differ noticeably from the properties of the original model. For example, if the time-step is too large compared to the time-constants of the system (you can calculate time-constants from the transfer function of the state-space model), the discrete-time system may be unstable even if the original continuous-time system is stable. However, with a reasonable value of the time-step – say less than one fifth of the smallest time-constant – the dynamic properties are almost the same.

Assume given the linear continuous-time state-space model

$$\dot{x}(t) = A_c x(t) + B_c u(t) \quad (9.9)$$

$$y(t) = C_c x(t) + D_c u(t) \quad (9.10)$$

Approximating the time derivative with Forward differentiation gives

$$\frac{x(t_{k+1}) - x(t_k)}{h} = A_c x(t) + B_c u(t) \quad (9.11)$$

Solving for $x(t_{k+1})$ gives

$$x(t_{k+1}) = \overbrace{(I + hA_c)}^A x(t_k) + \overbrace{hB_c}^B u(t_k) \quad (9.12)$$

$$= Ax(t_k) + Bu(t_k) \quad (9.13)$$

The output is given by

$$y(t_k) = \overbrace{C_c}^C x(t_k) + \overbrace{D_c}^D u(t_k) \quad (9.14)$$

$$= Cx(t_k) + Du(t_k) \quad (9.15)$$

(9.13) and (9.15) constitute a linear discrete-time state-space model.

Zero-order hold discretization

Often the process model to be discretized is the model of a physical having a sample-and-hold element on its input, as when the process is controlled by a computer via a DA converter (digital to analog). In such situations the input signal is held constant during the sampling interval, see Figure 9.1. This is denoted *Zero order hold* or ZOH.

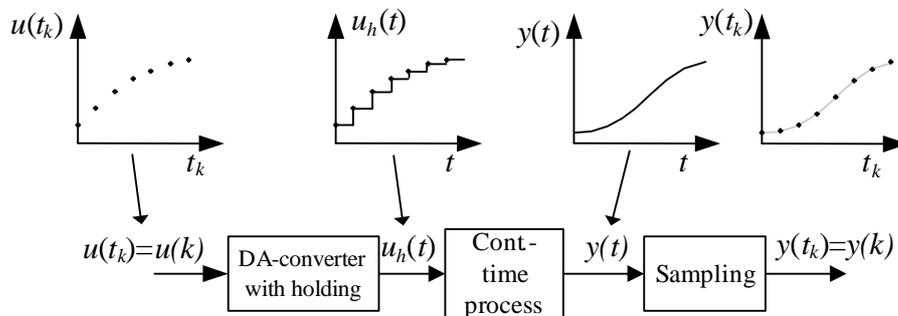


Figure 9.1: Block diagram of a process having a zero order hold element on its input. u_h is the piecewise constant (held) input signal.

Discretization with the *ZOH discretization method* gives a perfect discrete-time model. This means that the discrete-time model will produce exactly the same response as produced by the continuous-time model at the discrete points of time. (If we discretize using e.g. the Forward discretization method the responses will differ a little.) If you in a given application need to discretize a linear continuous-time system which has a

ZOH element at its input, you should use the ZOH discretization method, unless you have practical reasons for not doing it.

The ZOH discretization method is complicated to implement manually. In practice you will use functions in e.g. MATLAB or LabVIEW. Therefore, I will not describe the analytical methods for ZOH discretization.

Example 9.1 *Discretization of state-space model in MATLAB*

Here is an example of using the **c2d** function (continuous-to-discrete) in MATLAB:

```
A = [0,1;0,0];
B = [0;1];
C = [1,0];
D = [0];
Ts=0.1;%Sampling time
model_cont=ss(A,B,C,D);%ss creates a state-space model
model_disc=c2d(model_cont,Ts,'zoh')
A=model_disc.a
B=model_disc.b
C=model_disc.c
D=model_disc.d
```

The result as shown in MATLAB is:

```
Sampling time: 0.1
Discrete-time model.
A =
1.0000 0.1000
0 1.0000
B =
0.0050
0.1000
C =
1 0
D =
0
```

[End of Example 9.1]

Chapter 10

The z -transform

10.1 Definition of the z -transform

The z -transform of discrete-time signals plays much the same role as the Laplace transform for continuous-time systems.

The z -transform of the discrete-time signal $\{y(k)\}$, or just $y(k)$, is defined as follows:

$$\mathcal{Z}\{y(k)\} = \sum_{k=0}^{\infty} y(k)z^{-k} \quad (10.1)$$

For simplicity, I will use the symbol $y(z)$ for $\mathcal{Z}\{y(k)\}$ when it can not be misunderstood. Strictly, a different variable name should be used, for example $Y(z)$.

Example 10.1 z -transform of a constant

Assume that the signal $y(k)$ has constant value A . This signal can be regarded a step of amplitude A at time-step 0. z -transforming $y(k)$ gives

$$y(z) = \sum_{k=0}^{\infty} y(k)z^{-k} = \sum_{k=0}^{\infty} Az^{-k} = \frac{A}{1 - z^{-1}} = \frac{Az}{z - 1} \quad (10.2)$$

[End of example 10.1]

10.2 Properties of the z -transform

Below are the most important properties of the z -transform. These properties can be used when calculating the z -transform of composite signals.

- **Linearity:**

$$k_1y_1(z) + k_2y_2(z) \iff k_1y_1(k) + k_2y_2(k) \quad (10.3)$$

- **Time delay:** Multiplication by z^{-n} means time delay of n time-steps:

$$z^{-n}y(z) \iff y(k - n) \quad (10.4)$$

- **Time advancing:** Multiplication by z^n means time advancing by n time-steps:

$$z^n y(z) \iff y(k + n) \quad (10.5)$$

10.3 z -transform pairs

Below are several important z -transform pairs showing discrete-time functions and their corresponding z -transforms. The time functions are defined for $k \geq 0$.

$$\text{Unity impulse at time-step } k: \delta(k) \iff z^k \quad (10.6)$$

$$\text{Unity impulse at time-step } k = 0: \delta(0) \iff 1 \quad (10.7)$$

$$\text{Unity step at time-step } k = 0: 1 \iff \frac{z}{z - 1} \quad (10.8)$$

$$\text{Time exponential: } a^k \iff \frac{z}{z - a} \quad (10.9)$$

Example 10.2 z -transformation of a composite signal

Given the following discrete-time function:

$$y(k) = Ba^{k-n} \quad (10.10)$$

(which is a time delayed time exponential). The inverse z -transform of $y(k)$ can be calculated using (10.9) together with (10.3) and (10.4). The result becomes

$$y(z) = Bz^{-n} \frac{z}{z - a} = B \frac{z^{1-n}}{z - a} \quad (10.11)$$

[End of example 10.2]

10.4 Inverse z -transform

Inverse z -transformation of a given z evaluated function, say $Y(z)$, is calculating the corresponding time function, say $y(k)$. The inverse transform may be calculated using a complex integral¹, but this method is not very practical. Another method is to find a proper combination of precalculated z -transformation pairs, possibly in combination with some of the z -transform properties defined above.

In most cases where you need to calculate a time signal $y(k)$, its z -transform $Y(z)$ stems from a transfer function excited by some discrete-time input signal. You may then calculate $y(k)$ by first transferring the transfer function to a corresponding difference equation, and then calculating $y(k)$ iteratively from this difference equation as explained in Section 7.2.

¹ $y(k) = \frac{1}{2\pi j} \oint Y(z) z^k \frac{dz}{z}$, where the integration path must be in the area of convergence of $Y(z)$. [2]

Chapter 11

Discrete-time (or z -) transfer functions

11.1 Introduction

Models in the form of difference equations can be z -transformed to z -transfer *functions*, which plays the same role in discrete-time systems theory as s transfer functions do in continuous-time systems theory. More specific:

- The combined model of systems in a serial connection can be found by simply multiplying the individual z -transfer functions.
- The frequency response can be calculated from the transfer function.
- The transfer function can be used to represent the system in a simulator or in computer tools for analysis and design (as SIMULINK, MATLAB or LabVIEW)

11.2 From difference equation to transfer function

As an example we will derive the discrete-time or z -transfer function from input u to output y from the difference equation (7.4), which is repeated here:

$$y(k) = -a_1y(k-1) - a_0y(k-2) + b_0u(k-2) \quad (11.1)$$

First, we take the z -transform of both sides of the difference equation:

$$\mathcal{Z}\{y(k)\} = \mathcal{Z}\{-a_1y(k-1) - a_0y(k-2) + b_0u(k-2)\} \quad (11.2)$$

Using the linearity property (10.3) and the time delay property (10.4) (11.2) can be written as

$$\mathcal{Z}\{y(k)\} = -\mathcal{Z}\{a_1y(k-1)\} - \mathcal{Z}\{a_0y(k-2)\} + \mathcal{Z}\{b_0u(k-2)\} \quad (11.3)$$

and

$$y(z) = -a_1z^{-1}y(z) - a_0z^{-2}y(z) + b_0z^{-2}u(z) \quad (11.4)$$

which can be written as

$$y(z) + a_1z^{-1}y(z) + a_0z^{-2}y(z) = b_0z^{-2}u(z) \quad (11.5)$$

or

$$[1 + a_1z^{-1} + a_0z^{-2}]y(z) = b_0z^{-2}u(z) \quad (11.6)$$

$$y(z) = \underbrace{\frac{b_0z^{-2}}{1 + a_1z^{-1} + a_0z^{-2}}}_{H(z)}u(z) \quad (11.7)$$

$$= \underbrace{\frac{b_0}{z^2 + a_1z^1 + a_0}}_{H(z)}u(z) \quad (11.8)$$

where $H(z)$ is the z -transfer function from u to y . Hence, z -transfer functions can be written both with positive and negative exponents of z .¹

11.3 From transfer function to difference equation

In the above Section we derived a z -transfer function from a difference equation. We may go the opposite way – to derive a difference equation from a given z -transfer function. Some applications of this are

- Deriving a filtering algorithm from a filtering transfer function
- Deriving a control function from a given controller transfer function

¹In signal processing theory transfer functions are usually written with negative exponents of z , while in control theory they are usually written with positive exponents.

- Deriving a simulation algorithm from the transfer function of the system to be simulated

The procedure will be illustrated via a concrete example. Assume given the following transfer function:

$$H(z) = \frac{b_0}{z^2 + a_1z + a_0} = \frac{y(z)}{u(z)} \quad (11.9)$$

We start by cross multiplying (11.9):

$$(z^2 + a_1z + a_0)y(z) = b_0u(z) \quad (11.10)$$

which can be written as

$$z^2y(z) + a_1zy(z) + a_0y(z) = b_0u(z) \quad (11.11)$$

Taking the inverse transform of the above expression gives

$$\underbrace{z^2y(z)}_{y(k+2)} + \underbrace{a_1zy(z)}_{a_1y(k+1)} + \underbrace{a_0y(z)}_{a_0y(k)} = \underbrace{b_0u(z)}_{b_0u(k)} \quad (11.12)$$

Reducing each of the time indexes by 2 yields

$$y(k) + a_1y(k-1) + a_0y(k-2) = b_0u(k-2) \quad (11.13)$$

Usually it is practical to have the output variable alone on the left side:

$$y(k) = -a_1y(k-1) - a_0y(k-2) + b_0u(k-2) \quad (11.14)$$

11.4 Calculating time responses for discrete-time transfer functions

Assume given a transfer function, say $H(z)$, with input variable u and output variable y . Then,

$$y(z) = H(z)u(z) \quad (11.15)$$

If $u(z)$ is given, the corresponding time response in y can be calculated in several ways:

1. By finding a proper transformation pair in Section 10.3, possibly combined with some of the z -transform properties in Section 10.2.
2. By deriving a differential equation corresponding to the transfer function and then calculating $y(k)$ iteratively according to the difference equation. The procedure of deriving a differential equation corresponding to a given transfer function is explained in Section 11.3, and the calculation of time responses for a difference equation is described in Section 7.2.

11.5 Static transfer function and static response

The static version H_s of a given transfer function $H(z)$ will now be derived. Using the static transfer function the static response can easily be calculated. Assume that the input variable u is a step of amplitude U . The stationary response can be found using the final value theorem:

$$\lim_{k \rightarrow \infty} y(k) = y_s = \lim_{z \rightarrow 1} (z-1)y(z) \quad (11.16)$$

$$= \lim_{z \rightarrow 1} (z-1)H(z)u(z) \quad (11.17)$$

$$= \lim_{z \rightarrow 1} (z-1)H(z) \frac{zU}{z-1} \quad (11.18)$$

$$= H(1)U \quad (11.19)$$

Thus, we have the following static transfer function:

$$H_s = \frac{y_s}{u_s} = \lim_{z \rightarrow 1} H(z) = H(1) \quad (11.20)$$

Using the static transfer function the static response can be calculated by

$$y_s = H_s U \quad (11.21)$$

Example 11.1 Static transfer function

Let us consider the following transfer function:

$$H(z) = \frac{y(z)}{u(z)} = \frac{az}{z - (1-a)} \quad (11.22)$$

which is the transfer function of the lowpass filter (8.16) which is repeated here:

$$y(k) = (1-a)y(k-1) + au(k) \quad (11.23)$$

The corresponding static transfer function is

$$H_s = \frac{y_s}{u_s} = \lim_{z \rightarrow 1} H(z) = \lim_{z \rightarrow 1} \frac{a}{1 - (1-a)z^{-1}} = \frac{a}{1 - (1-a) \cdot 1} = 1 \quad (11.24)$$

Thus,

$$y_s = H_s u_s = u_s \quad (11.25)$$

Can we find the same correspondence between u_s and y_s from the difference equation (11.23)? Setting $y(k) = y(k-1) = y_s$ and $u(k) = u_s$ gives

$$y_s = (1-a)y_s + au_s \quad (11.26)$$

giving

$$\frac{y_s}{u_s} = \frac{a}{1 - (1 - a)} = 1 \quad (11.27)$$

which is the same as (11.24).

[End of Example 11.1]

11.6 Poles and zeros

Poles and zeros of z -transfer functions are defined in the same way as for s transfer functions: The zeros of the transfer function are the z -roots of numerator polynomial, and the poles are the z -roots of the denominator polynomial.

One important application of poles is stability analysis, cf. Section 13.

Example 11.2 *Poles and zeros*

Given the following z -transfer function:

$$H(z) = \frac{(z - b)}{(z - a_1)(z - a_2)} \quad (11.28)$$

The poles are a_1 and a_2 , and the zero is b .

[End of Example 11.2]

11.7 From s -transfer functions to z -transfer functions

In some cases you need to find a discrete-time z -transfer function from a given continuous-time s transfer function:

- In accurate model based design of a discrete controller for a process originally in the form of a continuous-time s transfer function, $H_p(s)$. The latter should be discretized to get a discrete-time process model before the design is started.

- Implementation of continuous-time control and filtering functions in a computer program.

There are several methods for discretization of an s transfer function. The methods can be categorized as follows, and they are described in the following sections:

1. **Discretization based on having a zero order hold (ZOH) element on the input of the system.** This method should be used in controller design of a process which has a sample and hold element on its input, as when a physical process is controlled by a computer via a DA converter (digital to analog). Zero order hold means that the input signal is held constant during the time-step or sampling interval. Figure 11.1 shows a block diagram of a continuous-time process with transfer function model $H(s)$ having a zero order hold element on its input.

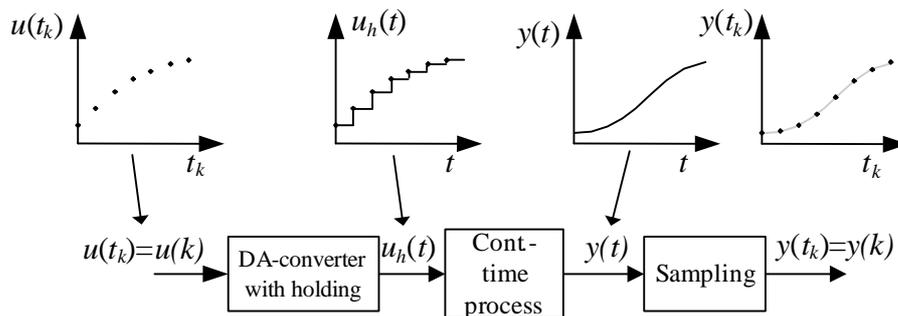


Figure 11.1: Block diagram of a process having a zero order hold element on its input. u_h is the piecewise constant (held) input signal.

ZOH discretization gives a perfect z -transfer function in the sense that it produces exactly the same response as produced by the s -transfer function at the discrete points of time. (If we discretize using e.g. the Forward differentiation the responses will differ a little.) The ZOH discretization method is actually complicated to implement manually, but tools as Matlab and LabVIEW have functions that perform the discretization easily, and in most practical applications, you will be using such tools.

2. **Using an appropriate approximation to time-derivatives**, as Forward Difference method, or Backward Difference method, or Tustin's method, cf. Section 8.1. In such cases the input signal is a

discrete-time signal with no holding (no ZOH element is assumed). The procedure has the following steps:

- From the given the continuous-time s -transfer function $H_c(s)$, derive the corresponding differential equation.
- Apply some approximation to the time-derivatives of the differential equation. If you do not have any other preferences, use the Backward difference method. The result is a difference equation.
- Calculate the z -transfer function from the difference equation, cf. Section 11.2.

The first two steps of this procedure are the same as used to discretize continuous-time lowpass filter in Section 8.3.

Here is an example of discretization using the `c2d` function in MATLAB:

Example 11.3 *Discretization using the `c2d` function in MATLAB*

The MATLAB code shown below discretizes the s -transfer function

$$H_{cont}(s) = \frac{2}{3 + 4s} \quad (11.29)$$

with sampling time $T_s = 0.1$.

```
Hcont = tf([2],[3,4]);
Ts=0.1;
Hdisc=c2d(Hcont,Ts,'zoh')
```

The result as shown in MATLAB is

Transfer function:

```
0.06241
-----
z - 0.8752
```

Sampling time: 0.1

[End of Example 11.3]

Here is an example of discretizing a s -transfer function manually using the Backward differentiation approximation:

Example 11.4 *Discretizing a first order transfer function*

We will discretize the following continuous-time transfer function:

$$H_c(s) = \frac{K}{Ts + 1} = \frac{y(s)}{u(s)} \quad (11.30)$$

1. Deriving the corresponding differential equation: Cross-multiplying gives

$$(Ts + 1)y(s) = Ku(s) \quad (11.31)$$

Resolving the parenthesis gives

$$Tsy(s) + y(s) = Ku(s) \quad (11.32)$$

Taking the inverse Laplace transform of both sides of this equation gives the following differential equation (because multiplying by s means time-differentiation in the time-domain):

$$T\dot{y}(t) + y(t) = Ku(t) \quad (11.33)$$

Let us use t_k to represent the present point of time – or discrete time:

$$T\dot{y}(t_k) + y(t_k) = Ku(t_k) \quad (11.34)$$

2. Applying the Backward differentiation approximation:

$$T\frac{y(t_k) - y(t_{k-1})}{T_s} + y(t_k) = Ku(t_k) \quad (11.35)$$

Solving for $y(t_k)$ gives the following difference equation:

$$y(t_k) = \frac{T}{T + T_s}y(t_{k-1}) + \frac{T_s K}{T + T_s}u(t_k) \quad (11.36)$$

3. Taking the z -transform of the difference equation:

$$y(z) = \frac{T}{T + T_s}z^{-1}y(z) + \frac{T_s K}{T + T_s}u(z) \quad (11.37)$$

from which we obtain the following z -transfer function:

$$H(z) = \frac{y(z)}{u(z)} = \left(z - \frac{T}{T + T_s} \right)^{-1} \frac{zT_s K}{T + T_s} \quad (11.38)$$

[End of Example 11.4]

Chapter 12

Frequency response of discrete-time systems

As for continuous-time systems, the frequency response of a discrete-time system can be calculated from the transfer function: Given a system with z -transfer function $H(z)$. Assume that input signal exciting the system is the sinusoid

$$u(t_k) = U \sin(\omega t_k) = U \sin(\omega k T_s) \quad (12.1)$$

where ω is the signal frequency in rad/s. The time-step is T_s .

It can be shown that the stationary response on the output of the system is

$$y(t_k) = Y \sin(\omega k T_s + \phi) \quad (12.2)$$

$$= U A \sin(\omega k T_s + \phi) \quad (12.3)$$

$$= \underbrace{U |H(e^{j\omega T_s})|}_Y \sin \left[\omega t_k + \underbrace{\arg H(e^{j\omega T_s})}_\phi \right] \quad (12.4)$$

where $H(e^{j\omega T_s})$ is the *frequency response* which is calculated with the following substitution:

$$H(e^{j\omega T_s}) = H(z)|_{z=e^{j\omega T_s}} \quad (12.5)$$

The *amplitude gain function* is

$$A(\omega) = |H(e^{j\omega T_s})| \quad (12.6)$$

The *phase lag function* is

$$\phi(\omega) = \arg H(e^{j\omega T_s}) \quad (12.7)$$

$A(\omega)$ and $\phi(\omega)$ can be plotted in a Bode diagram.

Figure 12.1 shows as an example the Bode plot of the frequency response of the following transfer function (time-step is 0.1s):

$$H(z) = \frac{b}{z-a} = \frac{0.0952}{z-0.9048} \quad (12.8)$$

Note that the plots in Figure 12.1 are drawn only up to the Nyquist frequency which in this case is

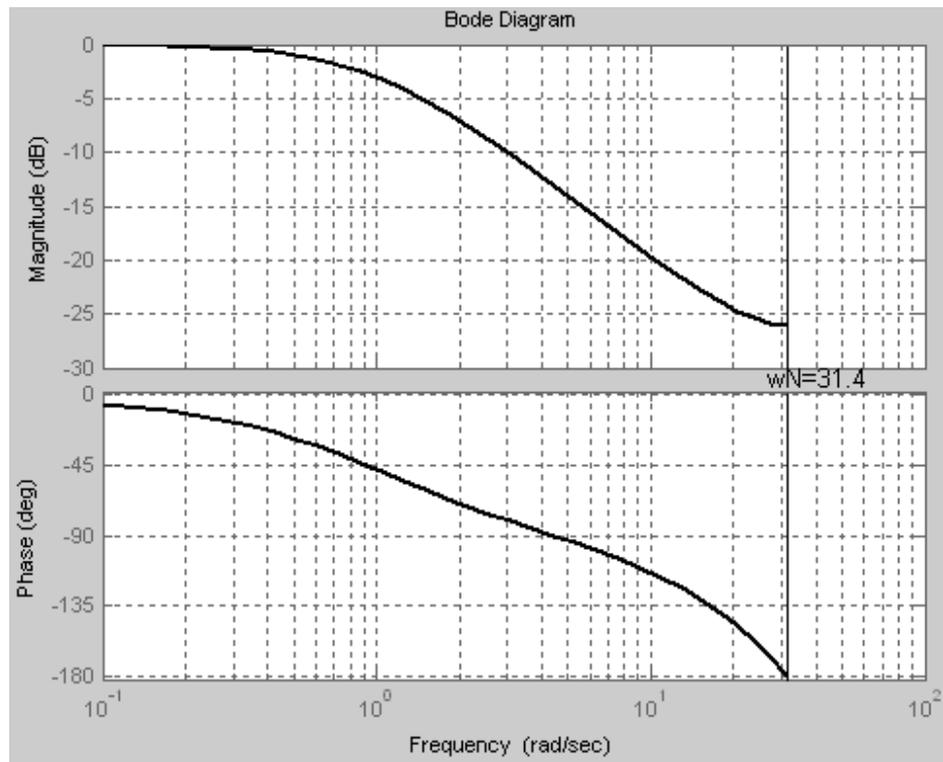


Figure 12.1: Bode plot of the transfer function (12.8). $\omega_N = 31.4$ rad/s is the Nyquist frequency (sampling time h is 0.1s).

$$\omega_N = \frac{\omega_s}{2} = \frac{2\pi/T_s}{2} = \frac{\pi}{T_s} = \frac{\pi}{0.1} = 10\pi \approx 31.4 \text{ rad/s} \quad (12.9)$$

The plots are not drawn (but they exist!) above the Nyquist frequency because of symmetry of the frequency response, as explained in the following section.

Example 12.1 *Calculating the frequency response manually from the z -transfer function*

Given the z -transfer function

$$H(z) = \frac{b}{z - a} \quad (12.10)$$

The frequency response becomes

$$H(e^{j\omega T_s}) = \frac{b}{e^{j\omega T_s} - a} \quad (12.11)$$

$$= \frac{b}{\cos \omega T_s + j \sin \omega T_s - a} \quad (12.12)$$

$$= \frac{b}{\underbrace{(\cos \omega T_s - a)}_{\text{Re}} + j \underbrace{\sin \omega T_s}_{\text{Im}}} \quad (12.13)$$

$$= \frac{b}{\sqrt{(\cos \omega T_s - a)^2 + (\sin \omega T_s)^2}} e^{j \arctan[(\sin \omega T_s)/(\cos \omega T_s - a)]} \quad (12.14)$$

$$= \frac{b}{\sqrt{(\cos \omega T_s - a)^2 + (\sin \omega T_s)^2}} e^{j \left[-\arctan\left(\frac{\sin \omega T_s}{\cos \omega T_s - a}\right) \right]} \quad (12.15)$$

The amplitude gain function is

$$A(\omega) = |H(e^{j\omega T_s})| = \frac{b}{\sqrt{(\cos \omega T_s - a)^2 + (\sin \omega T_s)^2}} \quad (12.16)$$

and the phase lag function is

$$\phi(\omega) = \arg H(e^{j\omega T_s}) = -\arctan\left(\frac{\sin \omega T_s}{\cos \omega T_s - a}\right) \text{ [rad]} \quad (12.17)$$

[End of Example 12.1]

Even for the simple example above, the calculations are cumbersome, and prone to errors. Therefore you should use some computer tool for calculating the frequency response, as MATLAB's Control System Toolbox or LabVIEW's Control Design Toolkit.

Symmetry of frequency response

It can be shown that the frequency response is symmetric as follows: $|H(e^{j\omega T_s})|$ and $\arg H(e^{j\omega T_s})$ are unique functions in the frequency interval $[0, \omega_N]$ where ω_N is the Nyquist frequency. In the following intervals $[m\omega_s, (m+1)\omega_s]$ (m is an integer) the functions are mirrored as indicated in Figure 12.2 which has a logarithmic frequency axis. (The Bode plots in this section are for the transfer function (12.8).) The symmetry appears

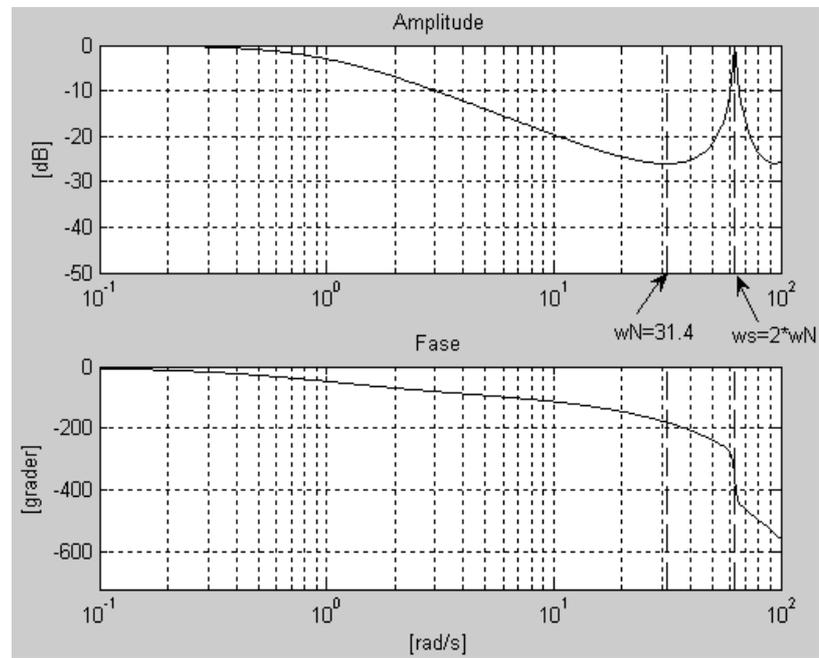


Figure 12.2: Bode plots of frequency response of (12.16). The frequency axis is logarithmic.

clearer in the Bode plots in Figure 12.3 where the frequency axis is linear.

Due to the symmetry of the frequency response, it is strictly not necessary to draw more of frequency response plots than of the frequency interval $[0, \omega_N]$.

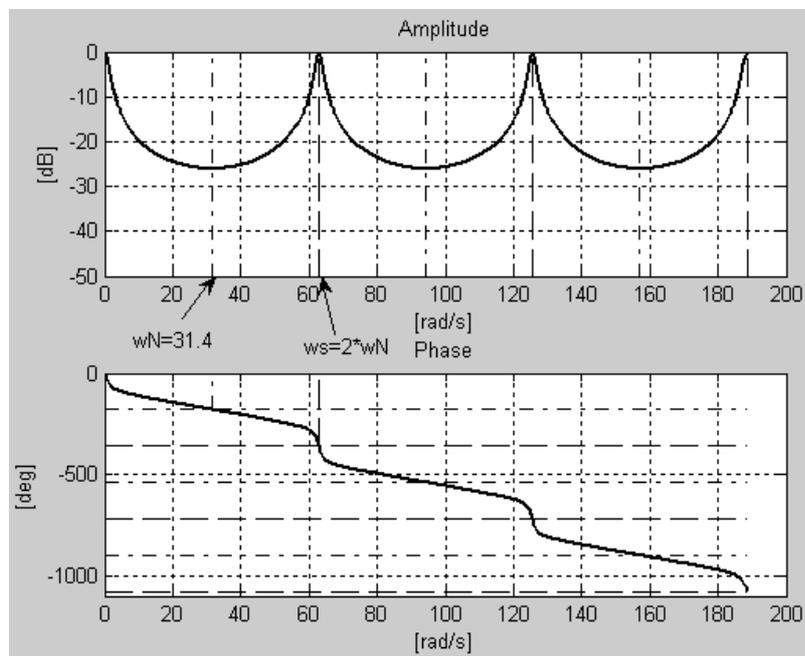


Figure 12.3: Bode plots of frequency response of (12.17). The frequency axis is linear to make the symmetries of the frequency responses clearer.

Chapter 13

Stability analysis of discrete-time dynamic systems

13.1 Definition of stability properties

Assume given a dynamic system with input u and output y . The stability property of a dynamic system can be defined from the *impulse response*¹ of a system as follows:

- **Asymptotic stable system:** The steady state impulse response is zero:

$$\lim_{k \rightarrow \infty} y_{\delta}(k) = 0 \quad (13.1)$$

- **Marginally stable system:** The steady state impulse response is different from zero, but limited:

$$0 < \lim_{k \rightarrow \infty} y_{\delta}(k) < \infty \quad (13.2)$$

- **Unstable system:** The steady state impulse response is unlimited:

$$\lim_{k \rightarrow \infty} y_{\delta}(k) = \infty \quad (13.3)$$

The impulse response for the different stability properties are illustrated in Figure 13.1. (The simulated system is defined in Example 13.1.)

¹An impulse $\delta(0)$ is applied at the input.

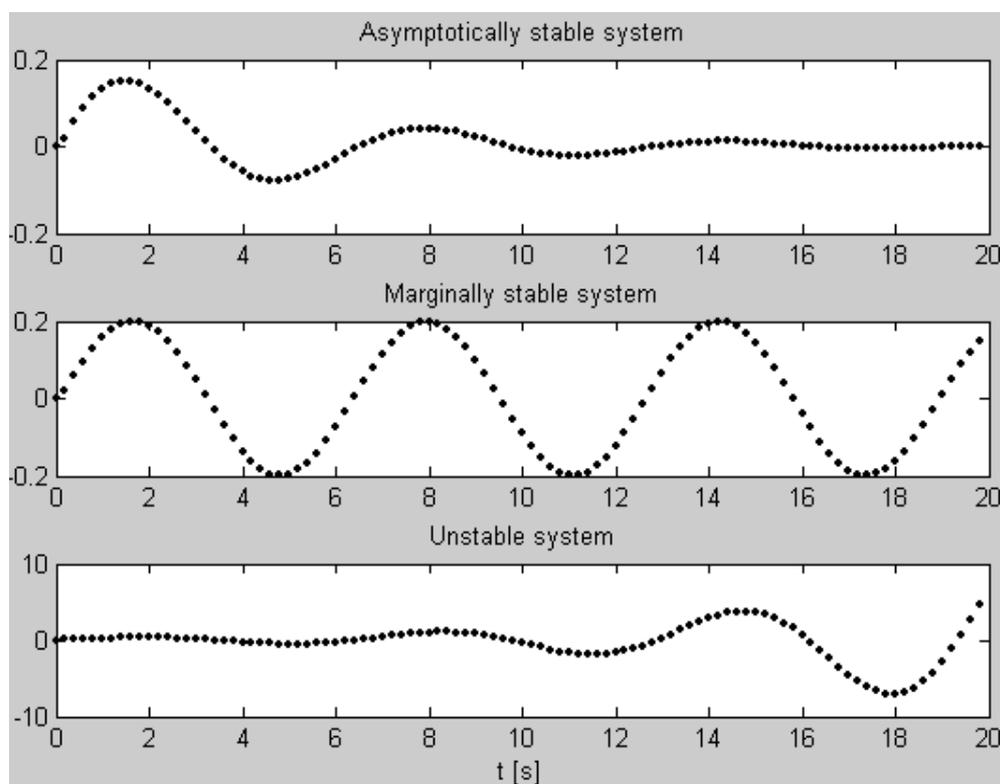


Figure 13.1: Impulse response and stability properties

13.2 Stability analysis of transfer function models

In the following we will base the analysis on the following fact: *The transfer function is the z -transformed impulse response.* Here is the proof of this fact: Given a system with transfer function $H(z)$. Assume that the input u is an impulse, which is a signal having value 1 at time index $k = 0$ and value zero at other points of time. According to (10.7) $u(z) = 1$. Then the z -transformed impulse response is

$$y(z) = H(z)u(z) = H(z) \cdot 1 = H(z) \quad (13.4)$$

(as stated).

Now, we proceed with the stability analysis of transfer functions. The impulse response $y_\delta(k)$, which defines the stability property of the system, is determined by the poles of the system's poles and zeros since the impulse responses is the inverse z -transform of the transfer function:

$$y_\delta(k) = \mathcal{Z}^{-1}\{H(z)\} \quad (13.5)$$

Consequently, the stability property is determined by the poles and zeros of $H(z)$. However, we will soon see that only the poles determine the stability.

We will now derive the relation between the stability and the poles by studying the impulse response of the following system:

$$H(z) = \frac{y(z)}{u(z)} = \frac{bz}{z-p} \quad (13.6)$$

The pole is p . Do you think that this system is too simple as a basis for deriving general conditions for stability analysis? Actually, it is sufficient because we can always think that a given z -transfer function can be partial fractionated in a sum of partial transfer functions or terms each having one pole. Using the superposition principle we can conclude about the stability of the original transfer function.

In the following, cases having of multiple (coinciding) poles will be discussed, but the results regarding stability analysis will be given.

The system given by (13.6) has the following impulse response calculated below. It is assumed that the pole in general is a complex number which may be written on polar form as

$$p = me^{j\theta} \quad (13.7)$$

where m is the magnitude and θ the phase. The impulse response is

$$y_\delta(k) = \mathcal{Z}^{-1} \left\{ \frac{bz}{z-p} \right\} \quad (13.8)$$

$$= \mathcal{Z}^{-1} \left\{ \frac{p}{1-pz^{-1}} \right\} \quad (13.9)$$

$$= \mathcal{Z}^{-1} \left\{ b \sum_{k=0}^{\infty} p^k z^{-k} \right\} \quad (13.10)$$

$$= bp^k \quad (13.11)$$

$$= b|m|^k e^{jk\theta} \quad (13.12)$$

From (13.12) we see that it is the *magnitude* m which determines if the steady state impulse response converges towards zero or not. From (13.12) we can now state the following relations between stability and pole placement (the statements about multiple poles have however not been derived here):

- **Asymptotic stable system:** All poles lie inside (none is on) the unit circle, or what is the same: all poles have magnitude less than 1.

- **Marginally stable system:** One or more poles – but no multiple poles – are on the unit circle.
- **Unstable system:** At least one pole is outside the unit circle. Or: There are multiple poles on the unit circle.

The “stability areas” in the complex plane are shown in Figure 13.2.

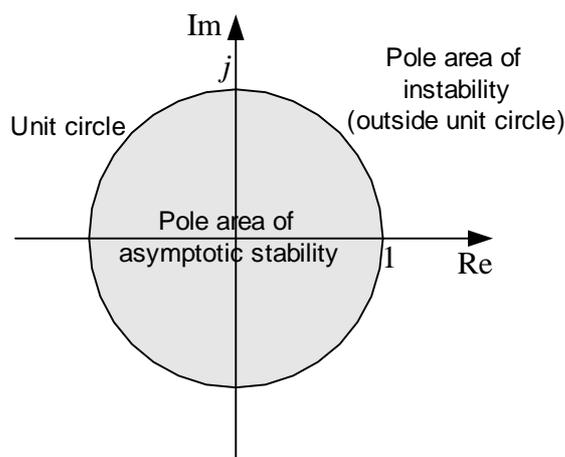


Figure 13.2: The different stability property areas of the complex plane

Let us return to the question about the relation between the *zeros* and the stability. We consider the following system:

$$H_1(z) = \frac{y(z)}{u(z)} = \frac{b(z-c)}{z-p} = (z-c)H(z) \quad (13.13)$$

where $H(z)$ is the “original” system (without zero) which were analyzed above. The zero is c . $H_1(z)$ can be written as

$$H_1(z) = \frac{bz}{z-p} + \frac{-bc}{z-p} \quad (13.14)$$

$$= H(z) - cz^{-1}H(z) \quad (13.15)$$

The impulse response of $H_1(z)$ becomes

$$y_{\delta_1}(k) = y_{\delta}(k) - cy_{\delta}(k-1) \quad (13.16)$$

where $y_{\delta}(k)$ is the impulse response of $H(z)$. We see that the zero does not influence whether the steady state impulse response converges towards zero or not. We draw the conclusion that the zeros of the transfer function do not influence the stability of the system.

Example 13.1 Stability analysis of discrete-time system

The three responses shown in Figure 13.1 are actually the impulse responses in three systems each having a transfer function on the form

$$\frac{y(z)}{u(z)} = H(z) = \frac{b_1 z + b_0}{z^2 + a_1 z + a_0} \quad (13.17)$$

The parameters of the systems are given below:

1. Asymptotically stable system: $b_1 = 0.019$, $b_0 = 0.0190$, $a_1 = -1.885$ and $a_0 = 0.923$. The poles are

$$z_{1,2} = 0.94 \pm j0.19 \quad (13.18)$$

They are shown in Figure 13.3 (the zero is indicated by a circle). The poles are inside the unity circle.

2. Marginally stable system: $b_1 = 0.020$, $b_0 = 0.020$, $a_1 = -1.96$ and $a_0 = 1.00$. The poles are

$$z_{1,2} = 0.98 \pm j0.20 \quad (13.19)$$

They are shown in Figure 13.3. The poles are on the unity circle.

3. Unstable system: $b_1 = 0.021$, $b_0 = 0.021$, $a_1 = -2.04$ and $a_0 = 1.08$. The poles are

$$z_{1,2} = 1.21 \pm j0.20 \quad (13.20)$$

They are shown in Figure 13.3. The poles are outside the unity circle.

[End of Example 13.1]

13.3 Stability analysis of state space models

Assume that the system has the following state space model:

$$x(k+1) = Ax(k) + Bu(k) \quad (13.21)$$

$$y(k) = Cx(k) + Du(k) \quad (13.22)$$

We can determine the stability by finding the corresponding transfer function from u to y , and then calculating the poles from the transfer function, as we did in the previous section. Let's derive the transfer

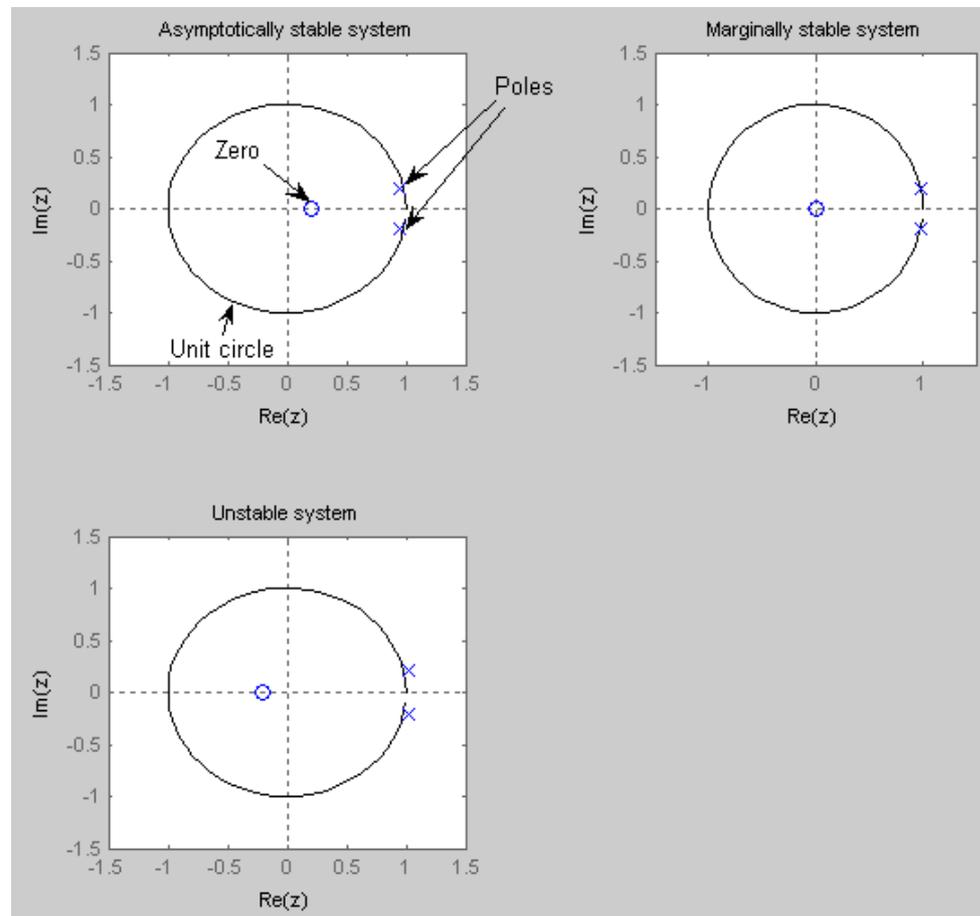


Figure 13.3: Example 13.1: Poles (and zeros) for the three systems each having different stability property

function: Take the \mathcal{Z} -transform of (13.21) – (13.22) to get (I is the identity matrix of equal dimension as of A)

$$zIx(z) = Ax(z) + Bu(z) \quad (13.23)$$

$$y(z) = Cx(z) + Du(z) \quad (13.24)$$

Solving (13.23) for $x(z)$ gives

$$x(z) = (zI - A)^{-1}Bu(z) \quad (13.25)$$

Inserting this $x(z)$ into (13.24) gives

$$y(z) = [C(zI - A)^{-1}B + D] u(z) \quad (13.26)$$

So, the transfer function is

$$H(z) = \frac{y(z)}{u(z)} = C(zI - A)^{-1}B + D \equiv C \frac{\text{adj}(zI - A)}{\det(zI - A)}B + D \quad (13.27)$$

The stability property can now be determined from the poles of this transfer function. The poles are the roots of the characteristic equation:

$$\det(zI - A) = 0 \quad (13.28)$$

But (13.28) actually defines the *eigenvalues* of A , $\text{eig}(A)$! The eigenvalues are the z -solutions to 13.28. Therefore, *the poles are equal to the eigenvalues*, and the relation between stability properties and eigenvalues are the same relation as between stability properties and poles, cf. Section 13.2. To make it clear:

- **Asymptotic stable system:** All eigenvalues (poles) lie inside (none is on) the unit circle, or what is the same: All eigenvalues have magnitude less than 1.
- **Marginally stable system:** One or more eigenvalues – but no multiple eigenvalues – are on the unit circle.
- **Unstable system:** At least one eigenvalue is outside the unit circle. Or: There are multiple eigenvalues on the unit circle.

The “stability areas” in the complex plane are as shown in Figure 13.2.

Example 13.2 *Stability analysis of a state-space model*

Given the following state-space model:

$$x(k+1) = \underbrace{\begin{bmatrix} 0.7 & 0.2 \\ 0 & 0.8 \end{bmatrix}}_A x(k) + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u(k) \quad (13.29)$$

$$y(k) = [1 \ 0] x(k) + [0] u(k) \quad (13.30)$$

It can be shown that the eigenvalues of A are 0.7 and 0.8. Both lies inside the unit circle, and hence the system is asymptotically stable.

[End of Example 13.2]

Chapter 14

Analysis of discrete-time feedback systems

You can analyze the dynamics (frequency response) and the stability of discrete-time feedback systems in the same way as you can analyze the dynamics and stability of continuous-time feedback systems, cf. Chapters 3 and 4, respectively. I assume that you already have knowledge about these topics.

Here is a summary of the relevant differences between analysis of continuous-time and discrete-time feedback systems:

- In the block diagrams etc. every s -transfer function is replaced by an equivalent z -transfer function, using a proper discretization method, e.g. the ZOH method.
- The stability property of any discrete-time system is given by the placement of the z -poles (or eigenvalues) in the complex plane. A discrete-time feedback system is asymptotically stable if all the poles of the closed-loop system lie inside the unit circle. These closed-loop poles are the poles of the tracking transfer function, $T(z)$.
- The stability property of a discrete-time feedback system can be analyzed in a Nyquist diagram or a Bode diagram stability based on the frequency response of the loop transfer function $L(z)$, which is the product of all the individual transfer functions of the feedback loop. Definitions of crossover frequencies and stability margins are as for continuous-time systems.

Here are some examples:

Example 14.1 Pole based stability analysis of feedback system

Assume given a control system where the P controller

$$H_c(z) = K_p \quad (14.1)$$

controls the process (which is actually an integrating process)

$$H_p(z) = \frac{K_i T_s}{z - 1} \quad (14.2)$$

We assume that $K_i = 1$ and $T_s = 1$. The loop transfer function becomes

$$L(z) = H_c(z)H_p(z) = \frac{K_p}{z - 1} = \frac{n_L(z)}{d_L(z)} \quad (14.3)$$

We will calculate the range of values of K_p that ensures asymptotic stability of the control system.

The characteristic polynomial is, cf. (5.4),

$$c(z) = d_L(z) + n_L(z) = z - 1 + K_p \quad (14.4)$$

The pole is

$$p = 1 - K_p \quad (14.5)$$

The feedback system is asymptotically stable if p is inside the unity circle or has magnitude less than one:

$$|p| = |1 - K_p| < 1 \quad (14.6)$$

which is satisfied with

$$0 < K_p < 2 \quad (14.7)$$

Assume as an example that $K_p = 1.5$. Figure 14.1 shows the step response in y_m for this value of K_p .

[End of Example 14.1]

Example 14.2 Stability analysis in Nyquist diagram

Given the following continuous-time process transfer function:

$$H_p(s) = \frac{y_m(z)}{u(z)} = \frac{K}{\left(\frac{s}{\omega_0}\right)^2 + 2\zeta\frac{s}{\omega_0} + 1} e^{-\tau s} \quad (14.8)$$

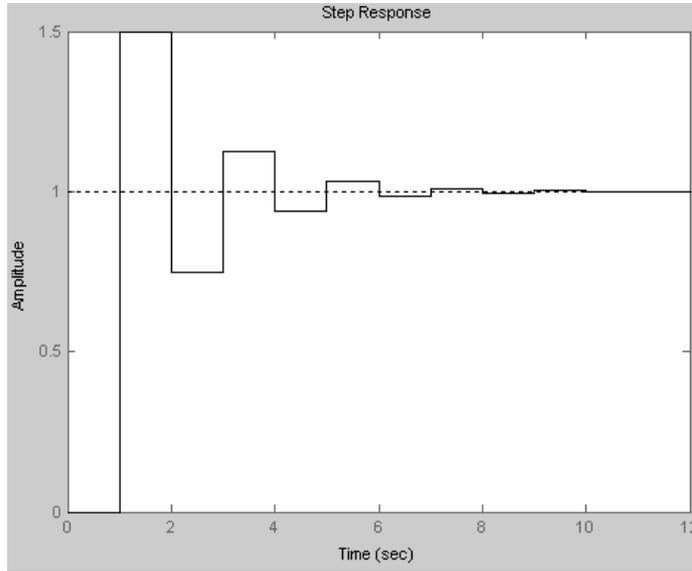


Figure 14.1: Example 14.1: Step response in y_m . There is a step in y_{mSP} .

with parameter values

$$K = 1; \zeta = 1; \omega_0 = 0.5 \text{ rad/s}; \tau = 1 \text{ s} \quad (14.9)$$

The process is controlled by a discrete-time PI-controller having the following z -transfer function, which can be derived by taking the z -transform of the PI control function (8.32),

$$H_c(z) = \frac{K_p \left(1 + \frac{T_s}{T_i}\right) z - K_p}{z - 1} \quad (14.10)$$

where the time-step (or sampling interval) is

$$T_s = 0.2 \text{ s} \quad (14.11)$$

Tuning the controller with the Ziegler-Nichols' closed-loop method [5] in a simulator gave the following controller parameter settings:

$$K_p = 2.0; T_i = 5.6 \text{ s} \quad (14.12)$$

To perform the stability analysis of the discrete-time control system $H_p(s)$ is discretized assuming zero order hold (using MATLAB or LabVIEW).

The result is

$$H_{pd}(z) = \frac{0.001209z + 0.001169}{z^2 - 1.902z + 0.9048} z^{-10} \quad (14.13)$$

The loop transfer function is

$$L(z) = H_c(z)H_{pd}(z) \quad (14.14)$$

Figure 14.2 shows the Nyquist plot of $L(z)$. From the Nyquist diagram we

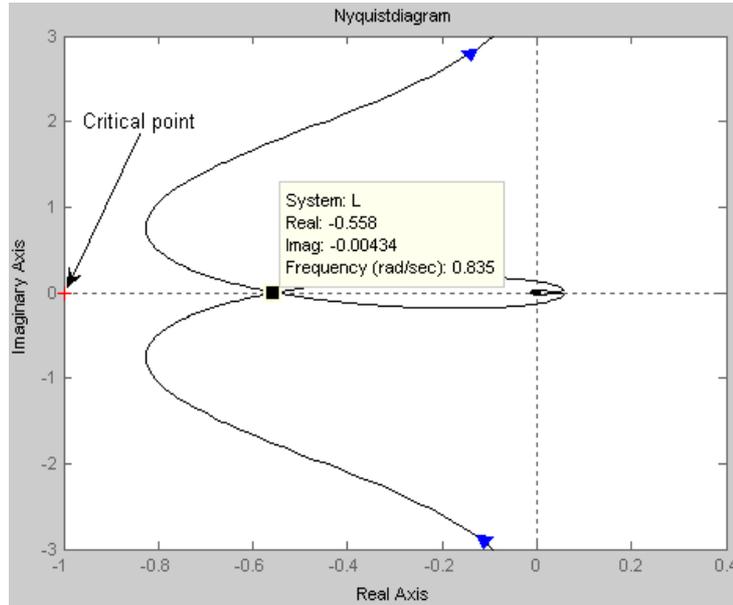


Figure 14.2: Example 14.2: Nyquist diagram of $L(z)$

read off

$$\omega_{180} = 0.835 \text{ rad/s} \quad (14.15)$$

and

$$\text{Re } L(e^{j\omega_{180}T_s}) = -0.558 \quad (14.16)$$

which gives the following gain margin, cf. (5.29),

$$GM = \frac{1}{|\text{Re } L(e^{j\omega_{180}T_s})|} = \frac{1}{|-0.558|} = 1.79 = 5.1 \text{ dB} \quad (14.17)$$

The phase margin can be found to be

$$PM = 35^\circ \quad (14.18)$$

Figure 14.3 shows the step response in y_m (unity step in setpoint y_{mSP}).

[End of Example 14.2]

Example 14.3 *Stability analysis in Bode diagram*

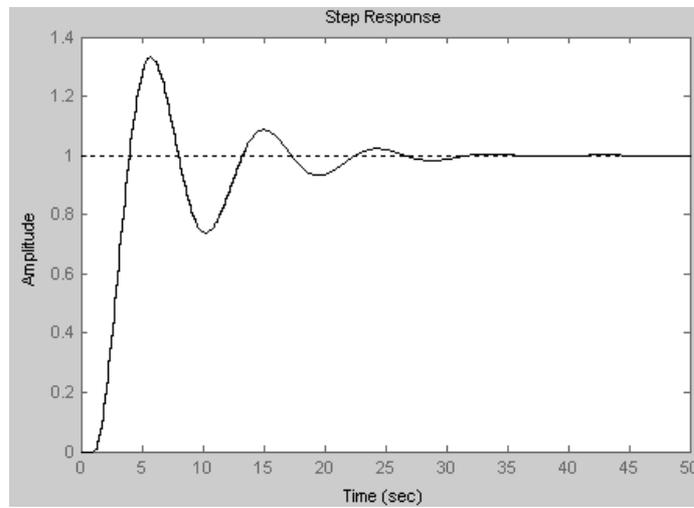


Figure 14.3: Example 14.2: Step response in y_m (unity step in setpoint $y_{m_{SP}}$)

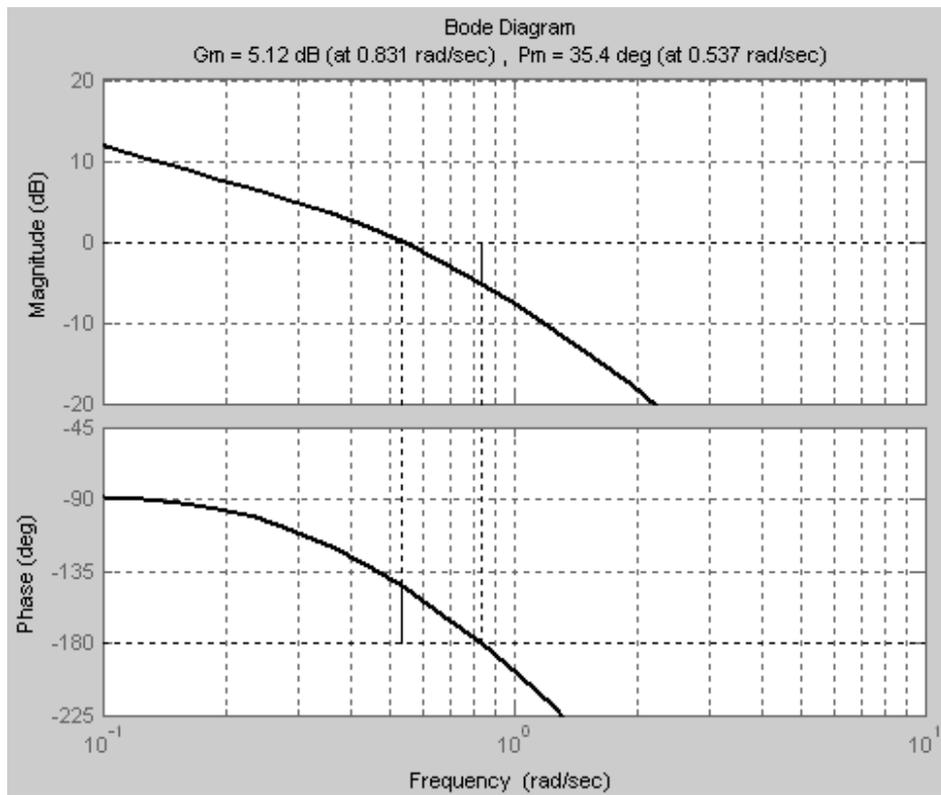
See Example 14.2. Figure 14.4 shows a Bode plot of $L(e^{j\omega T_s})$. The stability margins are shown in the figure. They are

$$GM = 5.12\text{dB} = 1.80 \quad (14.19)$$

$$PM = 35.3^\circ \quad (14.20)$$

which is in accordance with Example 14.2.

[End of Example 14.3]

Figure 14.4: Example 14.3: Bode plot of L

Part III

STOCHASTIC SIGNALS

Chapter 15

Stochastic signals

15.1 Introduction

In practical systems there are signals that vary more or less randomly. For example, measurement signals contain random noise, and process disturbances have some random component. Consequently, control signals and controlled variables, i.e. process output variables, have some random behaviour. The future value of a random signal can not be predicted precisely, i.e. such signals are non-deterministic, while steps, ramps and sinusoids are deterministic. In stead, random signals can be described with statistical measures, typically *expectation (or mean) value* and *standard deviation or variance (standard deviation is square root of variance)*.

Random signals may be denoted *stochastic signals*. Characteristics of assumed random process disturbances and random measurement noise are used in design of state estimators with Kalman Filters in Chapter 18.

15.2 How to characterize stochastic signals

15.2.1 Realizations of stochastic processes

A *stochastic process* may be characterized by its mean and standard deviation or variance. The stochastic process can be observed via one or more *realizations* of the process in the form of a *sequence* or time-series of samples, say $\{x(0), x(1), x(2) \dots\}$. Another realization of the same stochastic process will certainly show different sample values, but the mean

value and the variance will be almost the same (the longer the realization sequence is, the more equal the mean values and the variances will be). Figure 15.1 shows as an example *two different* realizations (sequences) of the same stochastic process, which in this case is Gaussian (normally) distributed with expectation (mean) value 0 and standard deviation 1. We see that the sequences are not equal.

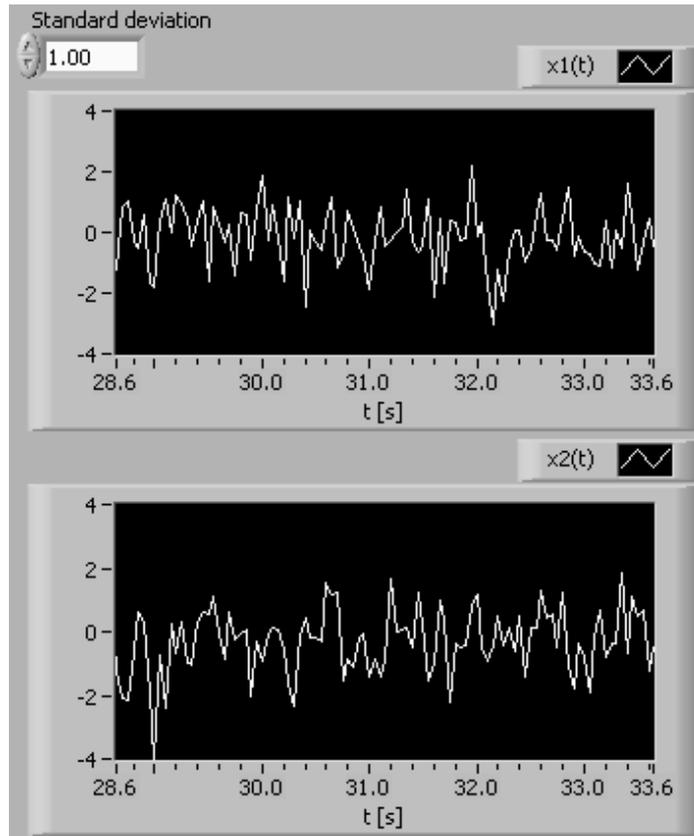


Figure 15.1: Two different realizations of the same stochastic process, which in this case is Gaussian (normally) distributed with expectation value 0 and standard deviation 1 . (Created with the Gaussian White Noise function in LabVIEW.)

15.2.2 Probability distribution of a stochastic variable

As known from statistics a stochastic variable can be described by its *probability distribution function*, PDF. Figure 15.2 shows two commonly used PDFs, the *Normal* (Gaussian) PDF and the *Uniform* PDF. With the

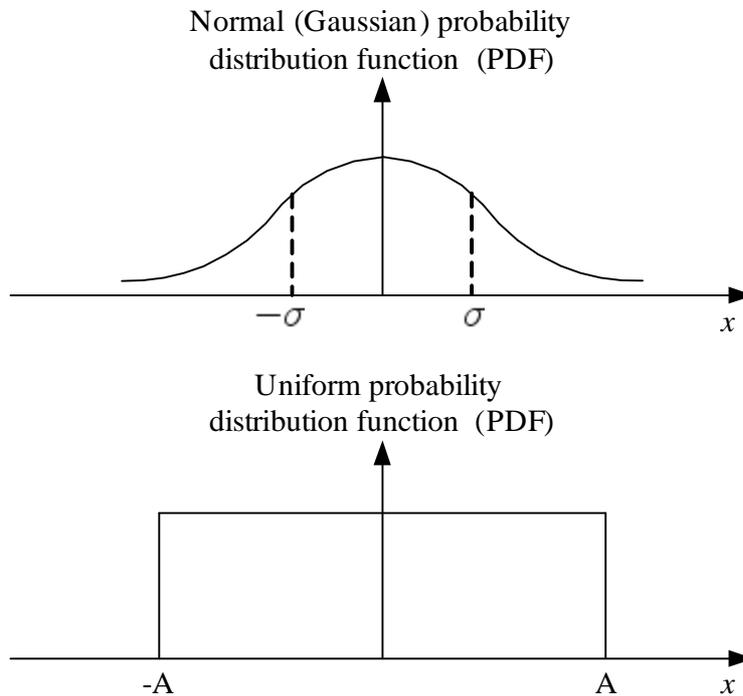


Figure 15.2: The Normal (Gaussian) PDF and the Uniform PDF

Normal PDF the probability that the variable has value in the range $\{-\sigma, +\sigma\}$ where σ is the standard deviation is approximately 68% (the standard deviation is defined below). With the Uniform PDF the probability that the variable has a value the $\{-A, +A\}$ range is uniform (constant) and the variable can not take any value outside this range.

A stochastic process is *stationary* if the PDF is time independent (constant), or in other words, if the statistical properties are time independent.

15.2.3 The expectation value and the mean value

The *expectation value*, $E(x)$, of the stochastic variable x is the mean (or average) value of x calculated from an infinite number of samples of x . For

a limited number N of samples the mean value can be calculated from

Mean value:

$$m_x = \frac{1}{N} \sum_{k=0}^{N-1} x(k) \quad (15.1)$$

Often these two terms (expectation value and mean value) are used interchangeably.

If x is a vector, say

$$x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \quad (15.2)$$

then the mean value of x has the form

$$m_x = \begin{bmatrix} m_{x_1} \\ m_{x_2} \end{bmatrix} = \begin{bmatrix} \frac{1}{N} \sum_{k=0}^{N-1} x_1(k) \\ \frac{1}{N} \sum_{k=0}^{N-1} x_2(k) \end{bmatrix} \quad (15.3)$$

15.2.4 Variance. Standard deviation

The *variance* of a stochastic variable is the mean or expected value of the squared difference between the value and its mean value:

$$\text{Var}(x) = E \left\{ [x(k) - m_x]^2 \right\} \quad (15.4)$$

The variance can be calculated from a sequence of samples as follows:

$$\text{Var}(x) = \frac{1}{N-1} \sum_{k=0}^{N-1} [x(k) - m_x]^2 \quad (15.5)$$

(Statistically it is better to divide by $N-1$ and not by N since the estimate of the statistical variance becomes unbiased using $N-1$.) The variance is some times denoted the *power* of the signal.

The *standard deviation* may give a more meaningful value of the variation of a signal. The standard deviation is the square root of the variance:

$$\sigma = \sqrt{\text{Var}(x)} \quad (15.6)$$

In many situations σ^2 is used as a symbol for the variance.

15.2.5 Auto-covariance. Cross-covariance

Sometimes it is useful to express how a stochastic variable, say $x(k)$, the time-index axis (k). This type of variance can be expressed by the *auto-covariance* :

Auto-covariance:

$$R_x(L) = E\{[x(k+L) - m_x][x(k) - m_x]\} \quad (15.7)$$

where L is the *lag*. Note that the argument of the auto-covariance function is the lag L . Figure 15.3 shows $R_x(L)$ for a signal x where the covariance decreased as the lag increases (this is typical). As indicated in Figure 15.3 the auto-covariance usually has a peak value at lag $L = 0$.

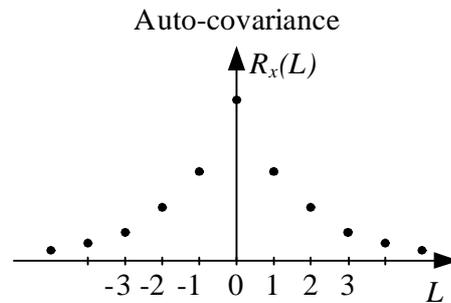


Figure 15.3: The auto-covariance for a signal x where the covariance decreased as the lag increases (this is typical)

If $L = 0$ the auto-covariance becomes the variance:

$$R_x(0) = E\{[x(k+0) - m_x][x(k) - m_x]\} \quad (15.8)$$

$$= E\{[x(k) - m_x]^2\} = \text{Var}(x) = \sigma^2 \quad (15.9)$$

In some applications x is a *vector*, say

$$x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \quad (15.10)$$

What does the auto-covariance look like in this case? For simplicity, assume that each of the four variables above have zero mean. The

auto-covariance then becomes

$$R_x(L) = E\{[x(k+L)][x(k)]^T\} \quad (15.11)$$

$$= E\left\{ \begin{bmatrix} x_1(k+L) \\ x_2(k+L) \end{bmatrix} \begin{bmatrix} x_1(k) & x_2(k) \end{bmatrix} \right\} \quad (15.12)$$

$$= \begin{bmatrix} E[x_1(k+L)x_1(k)] & E[x_1(k+L)x_2(k)] \\ E[x_2(k+L)x_1(k)] & E[x_2(k+L)x_2(k)] \end{bmatrix} \quad (15.13)$$

If $L = 0$, the auto-covariance becomes

$$R_x(0) = \begin{bmatrix} \underbrace{E\{[x_1(k)]^2\}}_{=\text{Var}(x_1)} & E[x_1(k)x_2(k)] \\ E[x_2(k)x_1(k)] & \underbrace{E\{[x_2(k)]^2\}}_{=\text{Var}(x_2)} \end{bmatrix} \quad (15.14)$$

Hence, the *variances are on the diagonal*.

The *cross-covariance* between two *different* scalar signals, say x and y , is

Cross-covariance:

$$R_{xy}(L) = E\{[x(k+L) - m_x][y(k) - m_y]\} \quad (15.15)$$

The cross-covariance can be *estimated* from sequences of sample values of x and y of length N with

$$\begin{aligned} R_{xy}(L) &= S \sum_{k=0}^{N-1-|L|} [x(k+L) - m_x][y(k) - m_y], \quad L = 0, 1, 2, \dots \end{aligned} \quad (15.16)$$

$$= S \sum_{k=1}^{N-1-|L|} [y(k-L) - m_y][x(k) - m_x] = R_{yx}(-L), \quad L = -1, -2, \dots \quad (15.17)$$

where S is a scaling factor which is defined below:

• **Raw estimate:**

$$S = 1 \quad (15.18)$$

• **Normalized estimate:**

$$S = \frac{1}{R_{xy}(0) \text{ in the raw estimate}} \quad (15.19)$$

This gives $R_{xy}(0) = 1$.

- **Unbiased estimate:**

$$S = \frac{1}{N - L} \quad (15.20)$$

This calculates $R_{xy}(L)$ as an average value of the product $[x(k + L) - m_x][y(k) - m_y]$. However, $R_{xy}(L)$ may be very “noisy” if L is large since then the summation is calculated with only a few additive terms (it is assumed that there are noise or random components in x and/or y).

- **Biased estimate:**

$$S = \frac{1}{N} \quad (15.21)$$

With this option $R_{xy}(L)$ is not an average value of the product $[x(k + L) - m_x][y(k) - m_y]$ since the sum of terms is divided by N no matter how many additive terms there are in the the summation. Although this makes $R_{xy}(L)$ become “biased” it reduces the “noise” in $R_{xy}(L)$ because the “noisy” terms are weighed by $1/N$ in stead of $1/(N - L)$. Unless you have reasons for some other selection, you may use biased estimate as the default option.

Correlation (auto/cross) is the same as covariance (auto/cross) except that the mean value, as m_x , is removed from the formulas. Hence, the cross-correlation is, cf. (15.15),

$$r_{xy}(L) = E\{x(k + L)y(k)\} \quad (15.22)$$

15.3 White and coloured noise

15.3.1 White noise

An important type of stochastic signals are the so-called *white noise signals (or processes)*. “White” is because in some sense white noise contains equally much of all frequency components, analogously to white light which contains all colours. White noise has zero mean value:

Mean value of white noise:

$$m_x = 0 \quad (15.23)$$

There is no co-variance or relation between sample values at different time-indexes, and hence the auto-covariance is zero for all lags L except for $L = 0$. Thus, the auto-covariance is the pulse function shown in Figure 15.4. Mathematically the auto-covariance function of white noise is

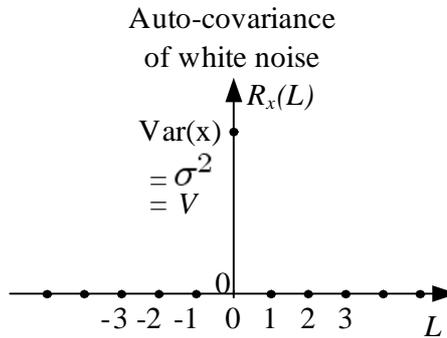


Figure 15.4: White noise has auto-correlation function like a pulse function.

$$R_x(L) = \text{Var}(x)\delta(L) = \sigma^2\delta(L) = V\delta(L) \quad (15.24)$$

Here, the short-hand symbol V has been introduced for the variance. $\delta(L)$ is the *unit pulse* defined as follows:

Unit pulse:

$$\delta(L) = \begin{cases} 1 & \text{when } L = 0 \\ 0 & \text{when } L \neq 0 \end{cases} \quad (15.25)$$

White noise is an important signal in estimation theory because the random noise which is always present in measurements, can be represented by white noise. For example, the variance of the assumed white measurement noise is used as an input parameter in the Kalman Filter design, cf. Chapter 18.

If you calculate the auto-covariance of a white noise sequence of finite length, the auto-covariance function will not be exactly as the ideal function shown in Figure 15.4, but the main characteristic showing a relatively large value at lag $L = 0$ is there.

Example 15.1 *White noise*

Figure 15.5 shows a simulated white noise signal x and its auto-covariance $R_x(L)$ (normalized) calculated from the most recent $N = 50$ samples of x .¹ The white noise characteristic of the signal is clearly indicated by $R_x(L)$.

[End of Example 15.1]

¹Implemented in LabVIEW.

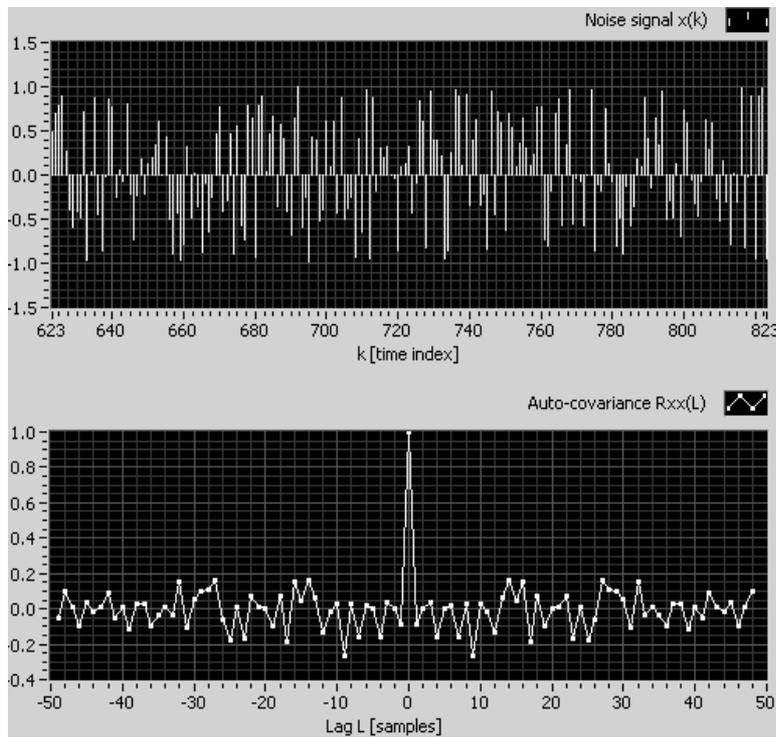


Figure 15.5: Example 15.1: Simulated white noise signal x and its auto-covariance $R_x(L)$ (normalized) calculated from the most recent $N = 50$ samples of x .

15.3.2 Coloured noise

As opposite to white noise, *coloured noise* does not vary completely randomly. In other words, there is a co-variance between the sample values at different time-indexes. As a consequence, the auto-covariance $R_x(L)$ is non-zero for lags $L \neq 0$. $R_x(L)$ will have a maximum value at $L = 0$, and $R_x(L)$ will decrease for increasing L .

You may generate coloured noise from white noise by sending the white noise through a dynamic system, typically a lowpass filter. Such a system is denoted *shaping filter*. The output signal of the shaping filter will be coloured noise. You can tune the colour of the coloured noise by adjusting the parameters of the shaping filter.

Example 15.2 Coloured noise

Figure 15.6 shows a simulated coloured noise signal x and its

auto-covariance $R_x(L)$ (normalized) calculated from the most recent $N = 50$ samples of x .² The coloured noise is the output of this shaping filter:

$$x(k) = ax(k-1) + (1-a)v(k) \quad (15.26)$$

which is a discrete-time first order lowpass filter. The filter input $v(k)$ is white noise. The filter parameter is $a = 0.98$. (If the filter parameter is 0 the filter performs no filtering, and the output is just white noise.) The

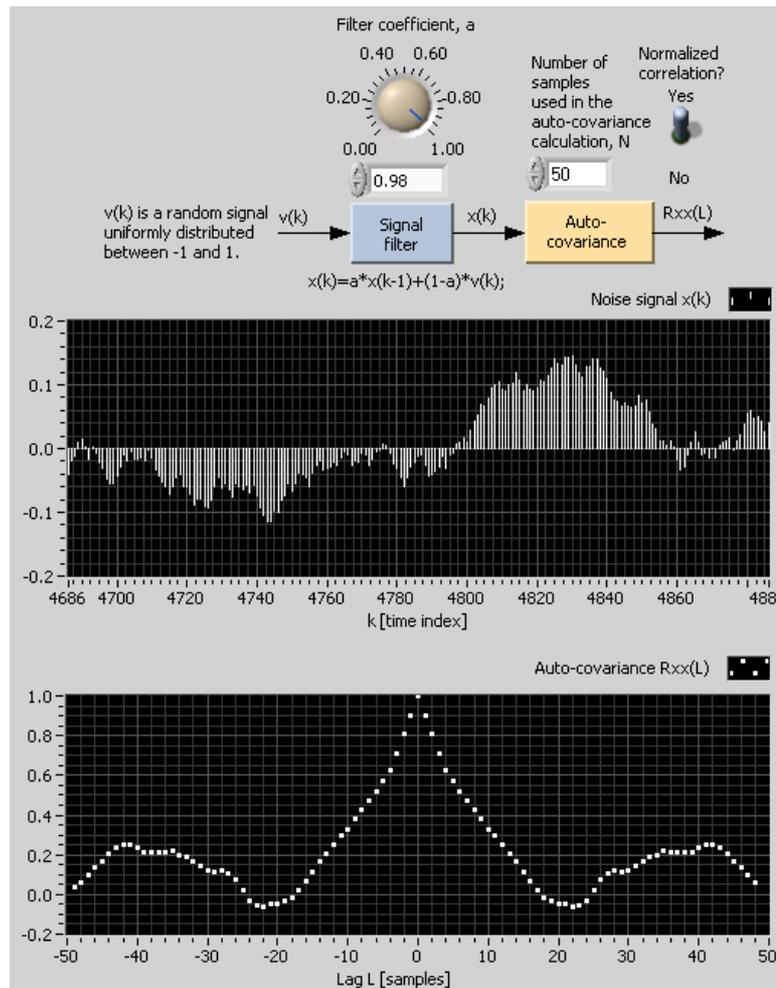


Figure 15.6: Example 15.1: Simulated coloured noise signal x and its auto-covariance $R_x(L)$ (normalized) calculated from the most recent $N = 50$ samples of x .

coloured noise characteristic of the signal is shown both in the plot of the

²Implemented in LabVIEW.

signal $x(k)$ in the upper diagram of Figure 15.6 and in the auto-covariance $R_x(L)$ shown in the lower diagram of Figure 15.6.

[End of Example 15.2]

15.4 Propagation of mean value and co-variance through static systems

If a stochastic (“random”) signals excites a static or dynamic system, the mean value and the co-variance of the output signal is different from those of the input. In this section we will concentrate on *static* systems. The results are useful e.g. in calculating the system gain needed to obtain a random signal of a specified variance when the source signal is a random signal of fixed variance.

The theory of the propagation of mean value and co-variance through *dynamic* systems is certainly important if you are going to analyze and design signal filters, controllers and state estimators assuming they are excited by random signals. However, it is my experience that this theory is not needed to be able to use the tools that exist for such applications (e.g. the Kalman Filter for state estimation). Therefore, I have omitted the topic of propagation of mean value and co-variance through dynamic systems in this book.

Assume given the following static linear system:

$$y(k) = Gv(k) + C \quad (15.27)$$

where v is a stationary stochastic input signal with mean value m_v and co-variance $R_v(L)$. y is the output of the system. G is the gain of the system, and C is a constant. In a multivariable system G is a matrix and C is a vector, but in the following we will assume that G and C are scalars, which is the most usual case.

Let us calculate the mean value and the auto-covariance of the output y . The mean value becomes

$$m_y = E[y(k)] = E[Gv(k) + C] = GE[v(k)] + C \quad (15.28)$$

$$= Gm_v + C \quad (15.29)$$

The auto-covariance of the output becomes

$$R_y(L) = E\{[y(k+L) - m_y][y(k) - m_y]\} \quad (15.30)$$

$$= E\{([Gv(k+L) + C] - [Gm_v + C])([Gv(k) + C] - [Gm_v + C])\} \quad (15.31)$$

$$= E\{(Gv(k+L) - Gm_v)(Gv(k) - Gm_v)\} \quad (15.32)$$

$$= E\{(G[v(k+L) - m_v])(G[v(k) - m_v])\} \quad (15.33)$$

$$= \underbrace{G^2 E\{([v(k+L) - m_v])([v(k) - m_v])\}}_{=R_v(L)} \quad (15.34)$$

$$= G^2 R_v(L) \quad (15.35)$$

If the system (15.27) is multivariable, that is, if v and y are vectors, G is a matrix and C is a vector. In this case we will get

$$R_y(L) = GR_v(L)G^T \quad (15.36)$$

Let us sum it up: For a scalar system (15.27):

Mean of output of stochastic static system:

$$m_y = Gm_v + C \quad (15.37)$$

and

Co-variance of output of stochastic static system:

$$R_y(L) = G^2 R_v(L) \quad (15.38)$$

The variance, which is equal to $R_y(0)$, becomes

Variance of output of stochastic static system:

$$\sigma_y^2 = G^2 \sigma_v^2 \quad (15.39)$$

And the standard deviation becomes

Standard deviation of output of stochastic static system:

$$\sigma_y = G\sigma_v \quad (15.40)$$

Example 15.3 *Mathematical operations to achieve specified output mean and variance*

Assume that you have signal generator available in a computer tool that can generate a white noise signal v having mean $m_v = 0$ and variance $\sigma_v^2 = 1$, and that you want to generate a signal y of mean $m_y = M$ and

variance $\sigma_y^2 = V$. Find proper mathematical operations on v that create this y .

The gain G can be calculated from (15.39):

$$G = \sqrt{\frac{\sigma_y^2}{\sigma_v^2}} = \sqrt{\frac{V}{1}} = \sqrt{V} = \sigma_y \quad (15.41)$$

The constant C can be calculated from (15.37):

$$C = m_y - Gm_v = M - G \cdot 0 = M \quad (15.42)$$

So, the mathematical operation is

$$y(k) = Gv(k) + C = \sqrt{V} \cdot v(k) + M = \sigma_y v(k) + M \quad (15.43)$$

In words: Multiply the input by the specified standard deviation and add the specified mean value to this result.

[End of Example 15.3]

Part IV

**ESTIMATION OF
PARAMETERS AND
STATES**

Chapter 16

Estimation of model parameters

16.1 Introduction

Mathematical models are the basis of simulators and theoretical analysis of dynamic systems, as control systems. The model can be in form of differential equations developed from physical principles or from transfer function models, which can be regarded as “black-box”-models which expresses the input-output property of the system. Some of the parameters of the model can have unknown or uncertain values, for example a heat transfer coefficient in a thermal process or the time-constant in a transfer function model. We can try to estimate such parameters from measurements taken during experiments on the system, see Figure 16.1.

Many methods for parameter estimation are available[6]. In this chapter the popular *least squares method* – or the LS-method – is described. You will see how the LS-method can be used for parameter estimation of both static models and dynamic models of a given structure.

In some cases, for example in analysis and design of feedback control systems and in signal modeling, you may want a *black-box model* which is a dynamic model in the form of a state-space model or a transfer function with non-physical parameters. Developing such black-box models is briefly described in the final section of this chapter.

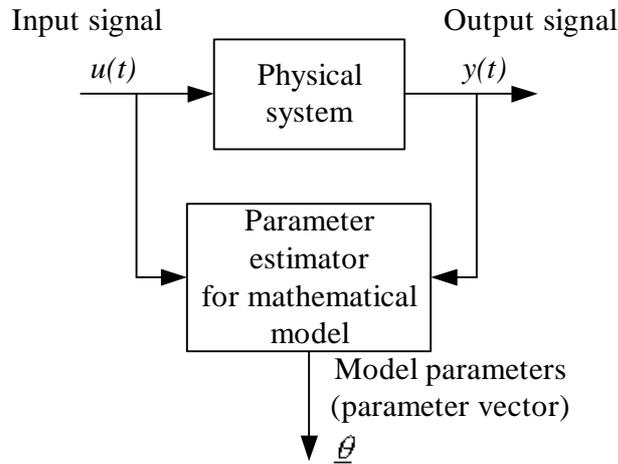


Figure 16.1: Estimation of parameters of a mathematical model from time-series of the input variable and the output variable.

16.2 Parameter estimation of static models with the Least squares (LS) method

In the following, the Least squares (LS-) method for calculating a solution of a set of linear equations, is briefly described, using terminology suitable for parameter estimation. After the LS-method is presented, a number of examples of parameter estimation follows.

16.2.1 The standard regression model

Assume given the following model:

$$y = \varphi_1\theta_1 + \dots + \varphi_n\theta_n \quad (16.1)$$

$$= \underbrace{[\varphi_1 \ \dots \ \varphi_n]}_{\varphi} \underbrace{\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}}_{\theta} \quad (16.2)$$

$$= \varphi\theta \quad (16.3)$$

which is called the *regression model*. φ_i is the *regression variable* (with known value). φ is the *regression vector* (with known value). y is the

observed variable (with known value). θ is an unknown parameter vector, and we will use the LS-method to estimate a value of θ . Note that the regression model is *linear in the parameter vector* θ .

Assume that we have m corresponding values of y and φ . Then we can write the following m equations according to the model:

$$y_1 = \varphi_{11}\theta_1 + \cdots + \varphi_{1n}\theta_n = \varphi_1\theta \quad (16.4)$$

$$\vdots$$

$$y_m = \varphi_{m1}\theta_1 + \cdots + \varphi_{mn}\theta_n = \varphi_m\theta \quad (16.5)$$

These m “stacked” equations can more compactly be written as

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}}_Y = \underbrace{\begin{bmatrix} \varphi_{11} & \cdots & \varphi_{1n} \\ \vdots & \ddots & \vdots \\ \varphi_{m1} & \cdots & \varphi_{mn} \end{bmatrix}}_\Phi \underbrace{\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}}_\theta \quad (16.6)$$

$$= \underbrace{\begin{bmatrix} \varphi_1 \\ \vdots \\ \varphi_m \end{bmatrix}}_\Phi \underbrace{\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}}_\theta \quad (16.7)$$

or, even more compact, as

$$Y = \Phi\theta \quad (16.8)$$

which is a set of equations from which we will calculate or estimate a value of the unknown θ using the LS-method.¹

¹In mathematical literature (16.8) is more often written on the form $b = Ax$. I have used symbols which are common in the field of system identification.

16.2.2 The LS problem

We define the *equation-error vector* or *prediction-error vector*², E , as the difference between the left side and the right side of (16.8):

$$E = \begin{bmatrix} e_1 \\ \vdots \\ e_m \end{bmatrix} \quad (16.9)$$

$$= \begin{bmatrix} y_1 - \varphi_1 \theta \\ \vdots \\ y_m - \varphi_m \theta \end{bmatrix} \quad (16.10)$$

$$= Y - \Phi \theta \quad (16.11)$$

Figure 16.2 illustrates the equation-errors or prediction errors for the case of the model

$$y = \varphi \theta \quad (16.12)$$

to be fitted to two data points.

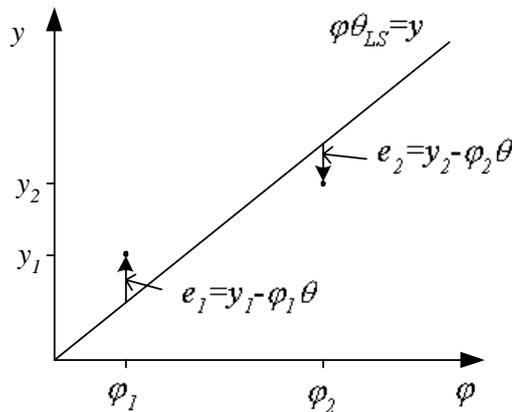


Figure 16.2: Equation-errors or prediction errors e_1 and e_2 for a simple case

The problem is to calculate a value – an estimate – of the unknown parameter-vector θ so that the following quadratic criterion function, $V(\theta)$,

²The name prediction-error vector is because the term $\Phi \theta$ can be regarded as a prediction of the observed (known) “output” y .

is minimized:

$$V(\theta) = e_1^2 + e_2^2 + \cdots + e_m^2 \quad (16.13)$$

$$= E^T E \quad (16.14)$$

$$= (Y - \Phi\theta)^T (Y - \Phi\theta) \quad (16.15)$$

$$= (Y^T - \theta^T \Phi^T) (Y - \Phi\theta) \quad (16.16)$$

$$= Y^T Y - Y^T \Phi\theta - \theta^T \Phi^T Y + \theta^T \Phi^T \Phi\theta \quad (16.17)$$

In other words, the problem is to estimate θ so that the sum of quadratic prediction errors is minimized.

16.2.3 The LS solution

Since $V(\theta)$ is a quadratic function of the unknown parameters θ , the minimum value of $V(\theta)$ can be calculated by setting the derivative of V with respect to θ equal to zero. This is illustrated in Figure 16.3. Using the differentiation rules for expressions containing vectors and then setting the derivative equal to zero, yields

$$\frac{dV(\theta)}{d\theta} = 2\Phi^T \Phi\theta - 2\Phi^T Y \stackrel{!}{=} 0 \text{ (vector)} \quad (16.18)$$

or

$$\Phi^T \Phi\theta = \Phi^T Y \quad (16.19)$$

(16.19) is called the *normal equation*. The θ -solution of (16.19) can be found by pre-multiplying (16.19) with $(\Phi^T \Phi)^{-1}$. The result is

$$\theta_{\text{LS}} = (\Phi^T \Phi)^{-1} \Phi^T Y \quad (16.20)$$

which is the *LS-solution* of (16.8)³. All right side terms in (16.8) are known.

Note: To apply the LS-method, the model must be written on the regression model form (16.8), which consists of m (16.3) “stacked”.

Example 16.1 *LS-estimation of two parameters*

Given the model

$$z = cx + d \quad (16.21)$$

³ $(\Phi^T \Phi)^{-1} \Phi^T$ is the so-called pseudo-inverse of Φ .

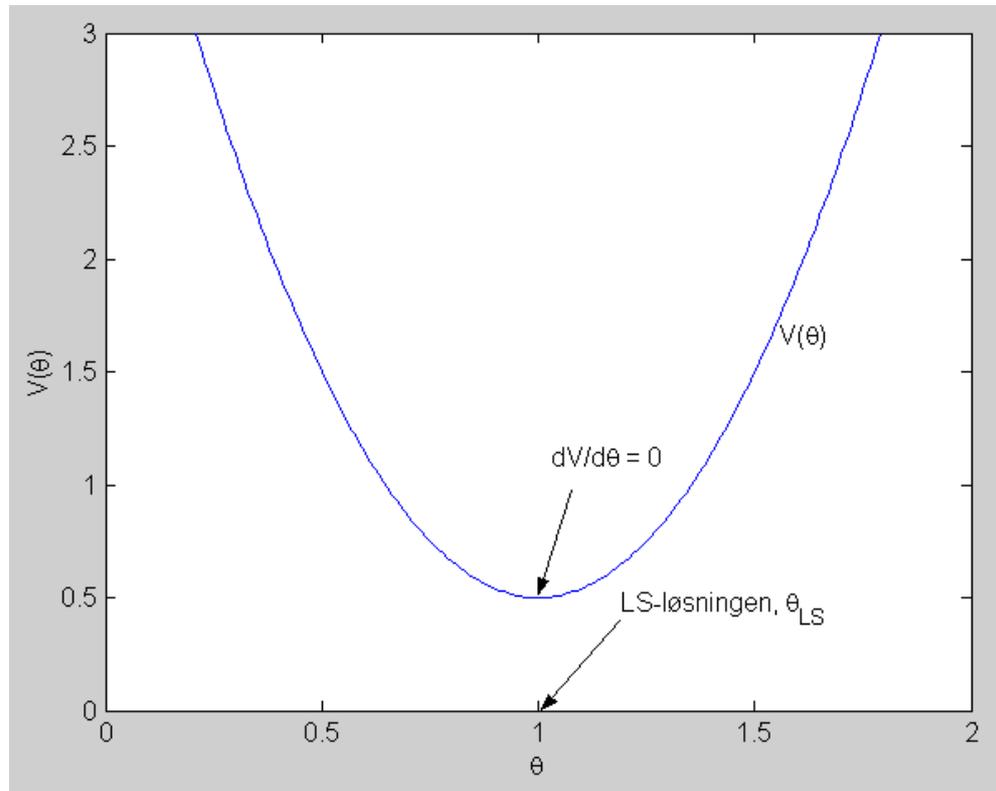


Figure 16.3: The LS solution θ_{LS} corresponds to the minimum value of the quadratic function $V(\theta)$, and can be calculated by setting the derivative $V(\theta)/d\theta$ to zero.

where the parameters c and d will be estimated from three given sets of corresponding values of z and x , as shown here:

$$0.8 = c \cdot 1 + d \quad (16.22)$$

$$3.0 = c \cdot 2 + d \quad (16.23)$$

$$4.0 = c \cdot 3 + d \quad (16.24)$$

which can be written on the form

$$\underbrace{\begin{bmatrix} 0.8 \\ 3.0 \\ 4.0 \end{bmatrix}}_Y = \underbrace{\begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix}}_\Phi \underbrace{\begin{bmatrix} c \\ d \end{bmatrix}}_\theta \quad (16.25)$$

$$= \underbrace{\begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \end{bmatrix}}_\Phi \underbrace{\begin{bmatrix} c \\ d \end{bmatrix}}_\theta \quad (16.26)$$

which is on the required regression model form (16.8). We can now calculate the LS-estimate of $\underline{\theta}$ using (16.20):

$$\theta_{\text{LS}} = (\Phi^T \Phi)^{-1} \Phi^T Y \quad (16.27)$$

$$= \left(\begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0.8 \\ 3.0 \\ 4.0 \end{bmatrix} \quad (16.28)$$

$$= \begin{bmatrix} 1.6 \\ -0.6 \end{bmatrix} = \begin{bmatrix} c_{\text{LS}} \\ d_{\text{LS}} \end{bmatrix} \quad (16.29)$$

[End of Example 16.1]

Example 16.2 *LS-estimation of valve parameter*

Figure 16.4 shows a valve. We assume that the flow q is given by the model

$$q = K_v u \sqrt{p} \quad (16.30)$$

where u is the control signal and p is the pressure drop. The valve

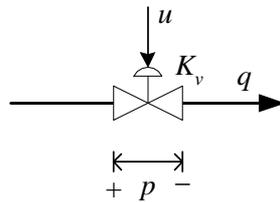


Figure 16.4: Valve where the valve parameter K_v shall be estimated

parameter K_v will be estimated from m corresponding values of q , u and p . We start by writing the model on the regression model form $y = \varphi \theta$:

$$\underbrace{q}_y = \underbrace{u \sqrt{p}}_{\varphi} \underbrace{K_v}_{\theta} \quad (16.31)$$

The “stacked” regression model (16.8) becomes

$$\underbrace{\begin{bmatrix} q_1 \\ \vdots \\ q_m \end{bmatrix}}_Y = \underbrace{\begin{bmatrix} u_1 \sqrt{p_1} \\ \vdots \\ u_m \sqrt{p_m} \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} K_v \end{bmatrix}}_{\theta} \quad (16.32)$$

LS-estimate $K_{v_{LS}}$ can now be calculated from (16.20):

$$\theta_{LS} = K_{v_{LS}} = (\Phi^T \Phi)^{-1} \Phi^T Y \quad (16.33)$$

$$= \left(\begin{bmatrix} u_1 \sqrt{p_1} & \cdots & u_m \sqrt{p_m} \end{bmatrix} \begin{bmatrix} u_1 \sqrt{p_1} \\ \vdots \\ u_m \sqrt{p_m} \end{bmatrix} \right)^{-1} \quad (16.34)$$

$$\cdot \begin{bmatrix} u_1 \sqrt{p_1} & \cdots & u_m \sqrt{p_m} \end{bmatrix} \begin{bmatrix} q_1 \\ \vdots \\ q_m \end{bmatrix} \quad (16.35)$$

$$= \frac{q_1 u_1 \sqrt{p_1} + \cdots + q_m u_m \sqrt{p_m}}{(u_1 \sqrt{p_1})^2 + \cdots + (u_m \sqrt{p_m})^2} \quad (16.36)$$

[End of Example 16.2]

16.2.4 Criterion for convergence of estimate towards the true value

The prediction-error vector is

$$E = Y - \Phi \theta_{LS} \quad (16.37)$$

It can be shown [6] that the LS-estimate θ_{LS} converges towards the true value θ_0 of the parameter vector as the number m of sets of observations goes to infinity, only if E is “white noise”. White noise means that the elements of E are random numbers with zero mean value. The opposite of white noise is coloured noise. A constant having value different from zero is an (extreme) example of coloured noise. E becomes coloured if there are systematic equation errors. Such systematic equation errors can be reduced or eliminated by choosing a more accurate model structure, for example by assuming the model $y = ax + b$ in stead of $y = ax$ if the observed values of y contains a constant term b .

16.2.5 How to compare and select among several candidates?

If you are to compare and select the best model among a number of estimated models, you may use some mathematical criterion as the basis of your selection. The criterion function V given by (16.13) can be used. The procedure is to calculate the value of V with the estimated parameter vector θ_{LS} inserted for θ . In general, *the less V , the better is the estimate.*

V given by (16.13) depends on the number m of observations or equations in the regression model. To compare values of V from different estimates, a normalized criterion should be used, for example

$$V_{\text{norm}} = \frac{1}{m}V(\theta_{\text{LS}}) = \frac{1}{m}E^T E \quad (16.38)$$

where E is calculated using (16.37).

You can expect an estimate to give a smaller value of the criterion V if the number of parameters n is increased. This is because increasing n introduces more degrees of freedom in the model. If we increased n to be equal to m (giving equal numbers of unknown parameters and equations) we could achieve perfect modeling and hence $V = 0$! The drawback of choosing a large n is that the random noise in the experimental data will be modeled, causing the model to lose its generality since the noise in one experiment is not equal to the noise in another experiment. A good strategy is to use approximately one half of the logged data (time-series) for model development (estimation) and the rest of the data for validation (calculating V). In this way you can avoid ending up with a model which includes a noise model.

If you must choose among models of a different number of parameters, you can apply the *Parsimony Principle*: *Among a number of proper models, choose the simplest one, i.e., the one having smallest number of parameters, n .* Proper models are those models which have normalized criterion functions V of roughly equal value. The parsimony principle expresses that a high number of parameters, n , should be punished. This can be stated mathematically as follows. The FPE index (Final Prediction Error) is defined as [6]

$$\text{FPE} = \frac{1 + \frac{n}{m}}{1 - \frac{n}{m}} V \quad (16.39)$$

The fraction in (16.39) increases as n increases, and consequently, large values of n are punished. So, the proper model is chosen as that model having the smallest value of FPE.

16.3 Parameter estimation of dynamic models

16.3.1 Introduction

In this section you will see how unknown parameters in *dynamic models*, as differential equations or transfer functions, can be estimated. The

LS-method (least squares) can be used to estimate a model that has a predefined model form or model order, e.g. a specific first order nonlinear differential equation with one or more unknown parameters which we want to estimate. A so-called *subspace method* can be used to estimate black-box models which are models that does not have any specific model order, but we seek a model that reflects the dynamic properties of the process. Both the LS-method and a subspace method are described in sections that follow.

16.3.2 Good excitation is necessary!

Assume, as an example, that you want to estimate the time-constant T of a first order transfer function. You can not estimate T , which is related to the dynamics of the system, if y and u have constant values all the time. Thus, it is necessary that the excitation signal, $u(t)$, has sufficiently rich variation to give the LS-method enough information about the dynamics of the system to produce an accurate estimate of T . Several good excitation signals are described in the following.

PRB-signal (Pseudo-Random Binary signal)

Figure 16.5 shows a realization of a PRB-signal. This signal can have only

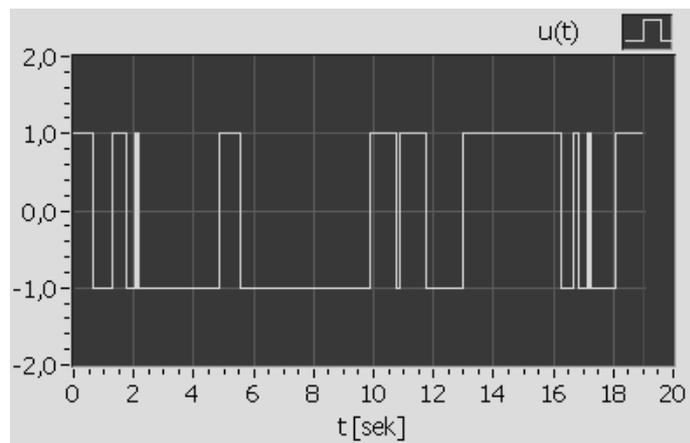


Figure 16.5: PRB-signal with probability of shift equal to $p = 0.1$. The time-step of the signal is $h = 0.1$ sec.

one of two possible values, say U and $-U$, with the probability p for the

signal to *change* value from one time-step to the next. You (the user) must choose p (and the signal amplitudes, of course). The less p , the more seldom the signal changes value. p should be chosen small enough for the system to have a chance to react with a significant response before the input value changes.

Below is a pseudo-program which realizes a PRB-signal with 1 and -1 as possible values. In the program it is assumed that the `rand`-function returns a uniformly distributed random signal between 0 and 1.

⁴

```
p=0.1; %Probability is set to 0.1.
u=1;
while not stop
{
if rand<p
{
u=-u;
}
}
}
```

Chirp signal

See Figure 16.6. This is a sine wave where the frequency increases linearly

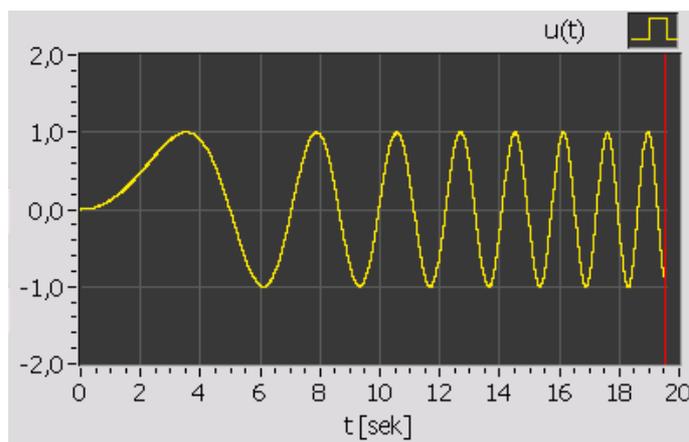


Figure 16.6: The chirp signal is a sine wave with constantly increasing frequency.

⁴In MATLAB the function `rand` returns a random number uniformly distributed between 0 and 1. In LabVIEW the function `Random Number` does the same.

with time:

$$u(t) = U \sin(2\pi ft) \quad (16.40)$$

where the frequency f [Hz] is

$$f = K_f t \quad (16.41)$$

where the frequency coefficient K_f is chosen so that a proper frequency range is covered so that the dynamics of the system is excited sufficiently.

Up-down-up signal

See Figure 16.7. This signal is simple to generate manually during the experiment. This signal gives in many cases enough excitation for the estimator to calculate accurate parameter estimates, but the period of the signal shift must be large enough to give the system output a chance to approximately stabilize between the steps.

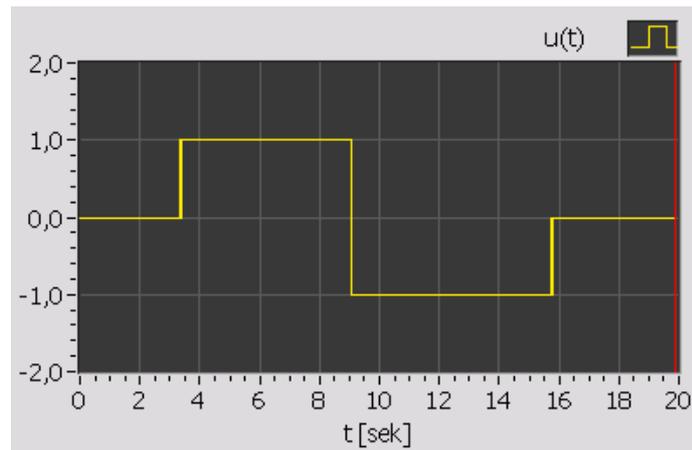


Figure 16.7: Up-down-up signal

16.3.3 How to check that a model is good?

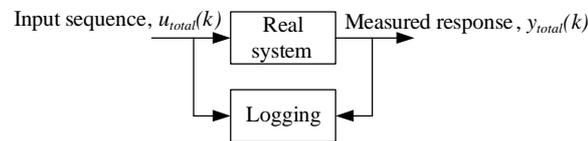
To check that an estimated model is good (accurate), you can select one or more of the methods below:

- **Compare simulated signal with measured signal:** You can *simulate* using the estimated model and compare the simulated

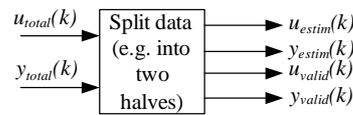
output signal with the logged or measured output signal. Of course, the simulator model and the real system must be excited by the same input signal. This input signal should not be the same as that used for model estimation. As mentioned on page 168, one strategy is to use approximately one half of the logged data (time-series) for model development and the rest of the data for validation (here, simulation). If the error between the simulated and the measured output signals is small, you may conclude that the model gives a good representation of the true system. A large error may come from poor (weak) excitation or a wrong model structure.

Figure 16.8 illustrates the procedure described above.

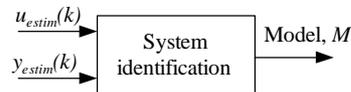
1. Excite the real system, and log input and output:



2. Split data, for estimation and for validation :



3. Estimate model:



4. Check (validate) model using e.g. simulation:

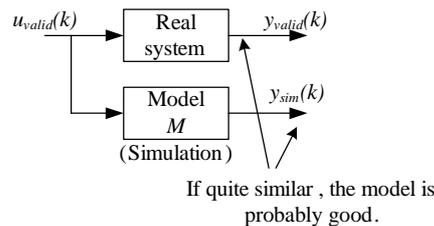


Figure 16.8: Procedure for system identification, and checking that the model is good.

- **Frequency response; Poles; Time constants etc.:** These quantities can be calculated from an estimated model, and they must be in accordance with assumed or observed properties of the system.

For example, estimated poles should match with the dynamic properties of the system to be modeled.

- **Check prediction error:** The prediction error E given by (16.37) should be “white”, that is, it should be similar to a random signal. The less coloured prediction error, the better estimate. The “colour” of the prediction error can be checked by calculating (and plotting) the auto-correlation of E .

Probably, the best method for checking if the model is good is the first among these (comparing simulated with measured response), and it may be sufficient to use only this method.

16.3.4 Estimation of differential equation models using the LS-method

Assume as an example the following non-linear a differential equation:

$$A\dot{h}(t) = -K_1\sqrt{h(t)} + K_2u(t) \quad (16.42)$$

(which can be the model describing the level h in a liquid tank with outflow through a valve with fixed opening and inflow via a pump controlled by the signal u). Suppose that K_1 and K_2 will be estimated, while A is known. (16.42) written on regression model form (16.3) becomes

$$\underbrace{A\dot{h}(t)}_y = \underbrace{[-\sqrt{h(t)} \quad u(t)]}_{\varphi} \underbrace{\begin{bmatrix} K_1 \\ K_2 \end{bmatrix}}_{\theta} \quad (16.43)$$

Eventual *time-delayed* signals causes no problems for regression model form. Here is an example of a model with time-delay:

$$\dot{x}(t) = ax(t) + bu(t - \tau) \quad (16.44)$$

which written on regression model form (16.3) is

$$\underbrace{\dot{x}(t)}_y = \underbrace{[x(t) \quad u(t - \tau)]}_{\varphi} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{\theta} \quad (16.45)$$

Time-derivatives, as $\dot{h}(t)$ in (16.43), must be calculated numerically when used in a regression model. Two simple numerical approximations to the time-derivatives is the *Euler's forward method*::

$$\dot{x}(t_k) \equiv \dot{x}_k \approx \frac{x_{k+1} - x_k}{T_s} \quad (16.46)$$

where T_s is the time-step between the discrete points of time, t_k and t_{k-1} , and the *Euler's backward method*:

$$\dot{x}_k \approx \frac{x_k - x_{k-1}}{T_s} \quad (16.47)$$

A more accurate approximation to time-derivatives is the *center difference method*, which is the average of the two Euler methods described above:

$$\dot{x}_k \approx \frac{\frac{x_{k+1} - x_k}{T_s} + \frac{x_k - x_{k-1}}{T_s}}{2} = \frac{x_{k+1} - x_{k-1}}{2T_s} \quad (16.48)$$

The center difference method is illustrated in Figure 16.9.

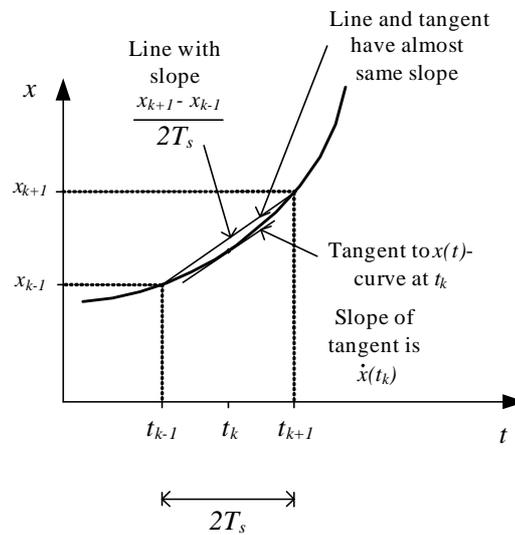


Figure 16.9: The center difference method as an approximation to a time-derivative: $\dot{x}(t_k) \approx \frac{x_{k+1} - x_{k-1}}{2T_s}$

Higher order derivatives can be calculated numerically by successive application of the center difference method in cascade, as follows:

$$\ddot{x}(t_k) = \ddot{x}_k \approx \frac{\dot{x}_{k+1} - \dot{x}_{k-1}}{2T_s} = \frac{\frac{x_{k+2} - x_k}{2T_s} - \frac{x_k - x_{k-2}}{2T_s}}{2T_s} = \frac{x_{k+2} - 2x_k + x_{k-2}}{4T_s^2} \quad (16.49)$$

16.3.5 Estimation of black-box models using subspace methods

A black-box model is a dynamic model in the form of a state-space model or a transfer function with *non-physical* parameters. A black-box model

represents the dynamic properties of the system and does not contain explicit information about the physics of the system. Such models are useful in many situations, for example in model-based analysis and design (as controller tuning) of control systems and in signal modeling.

It is common to have black-box models in the form of *discrete-time models* – not continuous-time models. This is because discrete-time models are more directly related to the discrete-time nature of the sampled data (time-series) from experiments from which the black-box model is developed.

Once you have a discrete-time model – a transfer function model or a state-space model, you can analyze it in many ways:

- Run simulations
- Plot frequency response
- Plot poles or eigenvalues
- Transform to equivalent continuous-time model to find e.g. time-constants or other dynamic characteristics.

How can you estimate a black-box model in practice? Several estimation methods are available[6], as the least squares method described earlier in this chapter, the prediction error method, and a subspace method. I suggest you using a *subspace method*. These are effective, and easy to use in practice. Subspace methods estimate a discrete-time black-box state-space model on the form of a linear – possibly multivariable – state-space model

$$x(t_{k+1}) = Ax(t_k) + Bu(t_k) + Ke(t_k) \quad (16.50)$$

$$y(t_k) = Cx(t_k) + Du(t_k) + e(t_k) \quad (16.51)$$

with initial state $x(0)$. $x(t_k)$ is the state vector, which is actually calculated during the estimation. $u(t_k)$ is the known input vector, $y(t_k)$ is the known output vector, $e(t_k)$ is the (white or random) noise-vector, and A , B , C , D , and K are coefficient matrices. t_{k+1} means the time at timestep no. k which is an integer time-index. $t_k = kT_s$ where T_s [sec] is the timestep or sampling time. The order n of this model is the number of states, which is the number of elements in the state-vector.

From this state-space model a z -transfer function from u til y can be calculated with the following formula:

$$H_{y,u}(z) = C(zI - A)^{-1}B + D \quad (16.52)$$

This formula can be derived by taking the \mathcal{Z} -transform of (16.50) – (16.51) and then solving for output $y(z)$, cf. Section 13.3. However, in practice you will probably use a proper function in e.g. MATLAB or LabVIEW, cf. the examples below.

All the five matrices in (16.50) – (16.51) are estimated. These matrices are assumed having some special canonical forms. Even the initial state is estimated, from known time-series of the input u and the corresponding output y . Once a state-space model is estimated, a transfer function may be calculated using (16.52). You must select the model order n so that you are content with the accuracy of the model. This can be done by running simulations with the model for different orders n , see Figure 16.8.

Note that if you assume that the system contains a time-delay of T_d [s], you need at least the following model order to include the time-delay in the model properly:

$$n_{\min} = \frac{T_d}{T_s} \quad (16.53)$$

where T_s is the sampling time.

In the following are two examples about estimating a black-box model of an electrical motor using subspace estimation, with MATLAB (System Identification Toolbox) and LabVIEW (System Identification Toolkit), respectively.

Example 16.3 *Subspace model estimation in MATLAB*

Figure 16.10 shows an electrical DC motor. It is manipulated with an input voltage signal, u , and the rotational speed is measured with a tachometer which produces a output voltage signal, y , which is proportional to the speed. During one experiment lasting for about 17 seconds, u was adjusted manually, and both the sequences (time series) of $u(t_k)$ and $y(t_k)$ were saved to a file. The sampling time was 0.02 s. Figure 16.11 shows a small extract of the log file as displayed with Notepad.⁵ The first column contains time stamps, but they are not used in this application. The second column contains the input sequence $u(t_k)$, and the third column contains the output sequence, $y(t_k)$.

Actually, the input and output sequences were divided into two parts:

⁵The whole logfile, named logfile1.lvm, is available from the home page of this book at <http://techteach.no>.

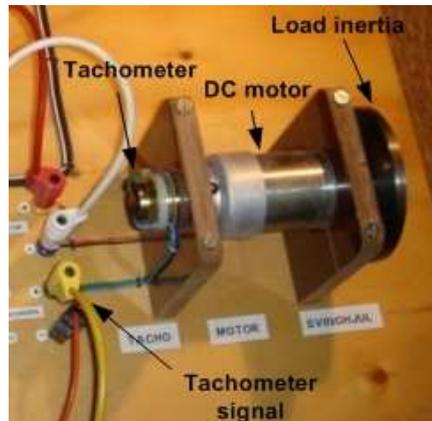


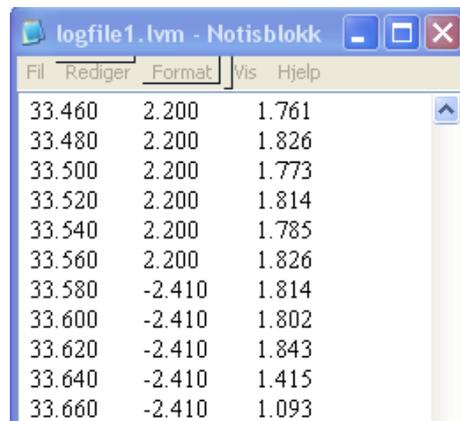
Figure 16.10: Example 16.3: Electrical (DC) motor for which an s -transfer function is estimated

- The first half, named u_{estim} and y_{estim} , were used to estimate a transfer function.
- The second half, named u_{valid} and y_{valid} , were used to check that the model is a good model. This is done by simulating the model with u_{valid} as input signal and comparing the simulated response, y_{sim} , with y_{valid} . If y_{valid} is quite similar to y_{sim} we can conclude that the model is good.

The estimation is made with the **n4sid**-function in MATLAB's System Identification Toolbox.

Below is the MATLAB-code which accomplishes the task.

```
%Loads data from file into workspace.
load logfile1.lvm;
Ts=0.02; %Sampling interval
L=length(logfile1);%(Matrix name becomes same as logfile name.)
N=round(L/2);
%Generates proper time signal, and extracts data from logfile:
t_estim=Ts*[1:N]';
u_estim=logfile1(1:N,2);
y_estim=logfile1(1:N,3);
t_valid=Ts*[N+1:L]';
u_valid=logfile1(N+1:L,2);
y_valid=logfile1(N+1:L,3);
modelorder=1;%Defines order of estimated model.
```



Time	Input	Output
33.460	2.200	1.761
33.480	2.200	1.826
33.500	2.200	1.773
33.520	2.200	1.814
33.540	2.200	1.785
33.560	2.200	1.826
33.580	-2.410	1.814
33.600	-2.410	1.802
33.620	-2.410	1.843
33.640	-2.410	1.415
33.660	-2.410	1.093

Figure 16.11: Example 16.3: An extract of the log file.

```

%Estimation of model. Model is on internal theta-format:
model_est=n4sid([y_estim u_estim],modelorder);
%th2tf-function calculates numerator and denominator coeff. arrays
%in z-transfer function:
[num,den]=th2tf(model_est);
H_disc=tf(num,den,Ts); %Generates an LTI-model from z-transf func.
y_sim=lsim(H_disc,u_valid,t_valid);%Simulates with u_valid as input.
figure(1)
%Plots y_estim and u_estim:
plot(t_estim,[y_estim,u_estim]);
ylabel('V');xlabel('t [s]')
figure(2)
%Plots y_sim, y_valid, and u_valid.
plot(t_valid,[y_sim,y_valid,u_valid]);
ylabel('V');xlabel('t [s]')
H_cont=d2c(H_disc,'zoh') %Converts to s-transfer function.

```

Figure 16.12 shows the input u_{estim} and output y_{estim} used for the estimation.

I selected order $n = 1$ for the model because it seemed not to be any large improvement in using a higher order, and according to the parsimony principle of system identification, you should select the simplest model among proper models. Figure 16.13 shows the input u_{valid} and output y_{valid} used for validating the model, together with the simulated output y_{sim} . Since y_{valid} is quite similar to y_{sim} we can conclude that the model is good.

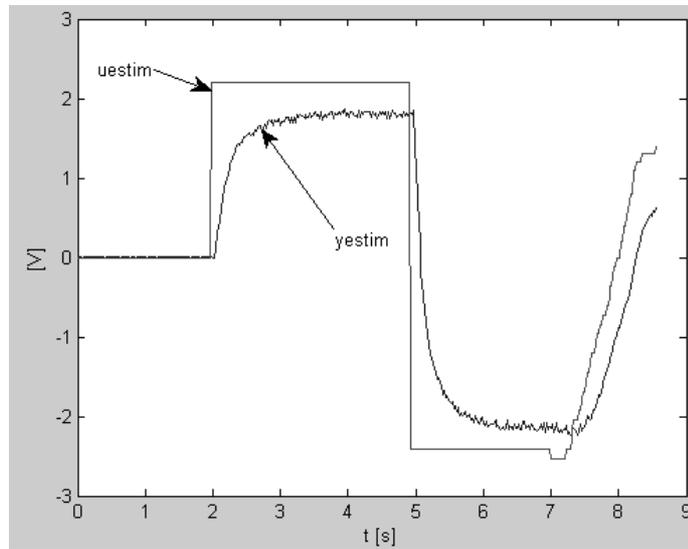


Figure 16.12: Example 16.3: The input u_{estim} and output y_{estim} used for the estimation.

The resulting discrete-time transfer function model was

$$H_{disc}(z) = \frac{0.05788}{z - 0.9344} \quad (16.54)$$

This model was converted to the following continuous-time transfer function using the **d2c** function:

$$H_{cont}(s) = \frac{2.993}{s + 3.393} \quad (16.55)$$

$$= \frac{2.993/3.393}{(1/3.393)s + 1} = \frac{0.88}{0.29s + 1} = \frac{K}{Ts + 1} \quad (16.56)$$

Thus, the gain is 0.88, and the time-constant is 0.29 sec. I know from experience with the motor that these are good values!

[End of Example 16.3]

The next example shows how I used LabVIEW to accomplish the same system identification task as in Example 16.3.

Example 16.4 *Subspace model estimation in LabVIEW*

The introduction to the example is the same as in is Example 16.3.

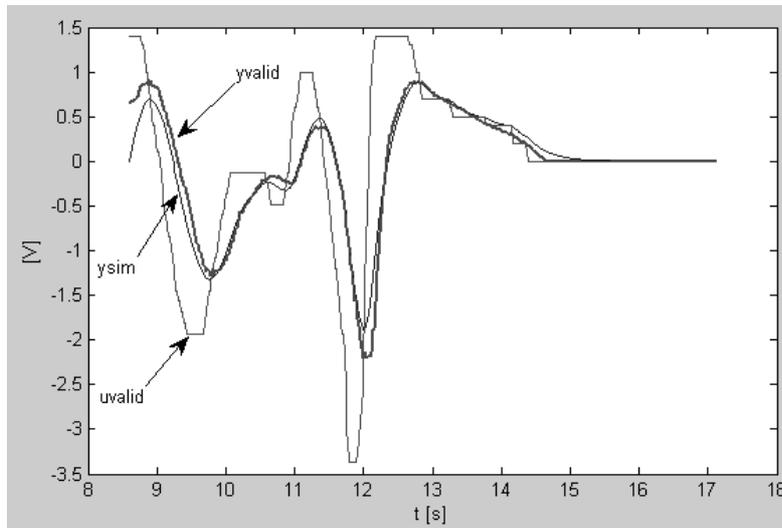


Figure 16.13: Example 16.3: The input u_{valid} and output y_{valid} used for validating the model, together with the simulated output y_{sim} .

Figure 16.14 shows the front panel (the user interface) of the LabVIEW program (sysid.vi), Figure 16.15 shows the block diagram (programming code) of the program.

The result of the estimation is the continuous-time transfer function

$$H_{cont}(s) = \frac{y(s)}{u(s)} = \frac{0.0031s + 3.08}{s + 3.54} \approx \frac{3.08}{s + 3.54} \quad (16.57)$$

$$= \frac{3.08/3.54}{(1/3.54)s + 1} = \frac{0.87}{0.28s + 1} = \frac{K}{Ts + 1} \quad (16.58)$$

with no time-delay. Thus, the gain is 0.87, and the time-constant is 0.28 sec, which are very similar to the values found with MATLAB in Example 16.3 (and these parameter values are good).

Below are a number of comments to the front panel and to the block diagram of the LabVIEW program:

Comments to the *front panel*, see Figure 16.14:

1. The program loop time of 1 sec is the loop or cycle time of the program. This means that the code implementing the estimation etc. is executed each second. This continuous program execution makes it possible for the user to see the consequences of adjusted settings “immediately”.

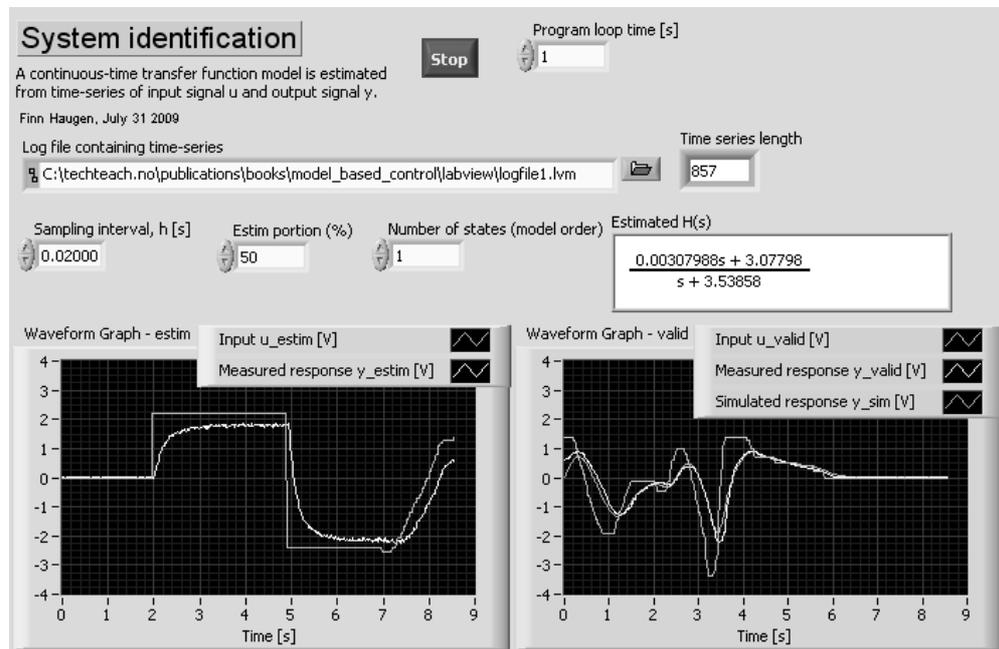


Figure 16.14: Example 16.4: The front panel of the LabVIEW program sysid.vi.

2. The user can select the model order.
3. The user can select the percentage portion of the original time-series to be used for estimation. In this example I have selected to use the first 50% for estimation, and the second 50% for validation via simulation.
4. The front panel shows in respective charts the data used for estimation (left) and the data used for validation (right). From the chart to the right we see that the model is good, as y_{sim} and y_{valid} are quite similar.

Comments to the *block diagram*, see Figure 16.15:

1. The **Read From Measurement File** function reads the saved data (sequences) from the log file and makes them available to the LabVIEW program. The 2-dimensional arrays of data in the file are extracted to proper 1-dimensional arrays using **Index Array** functions.
2. The function **SI Split Signal** splits the signals to be used for

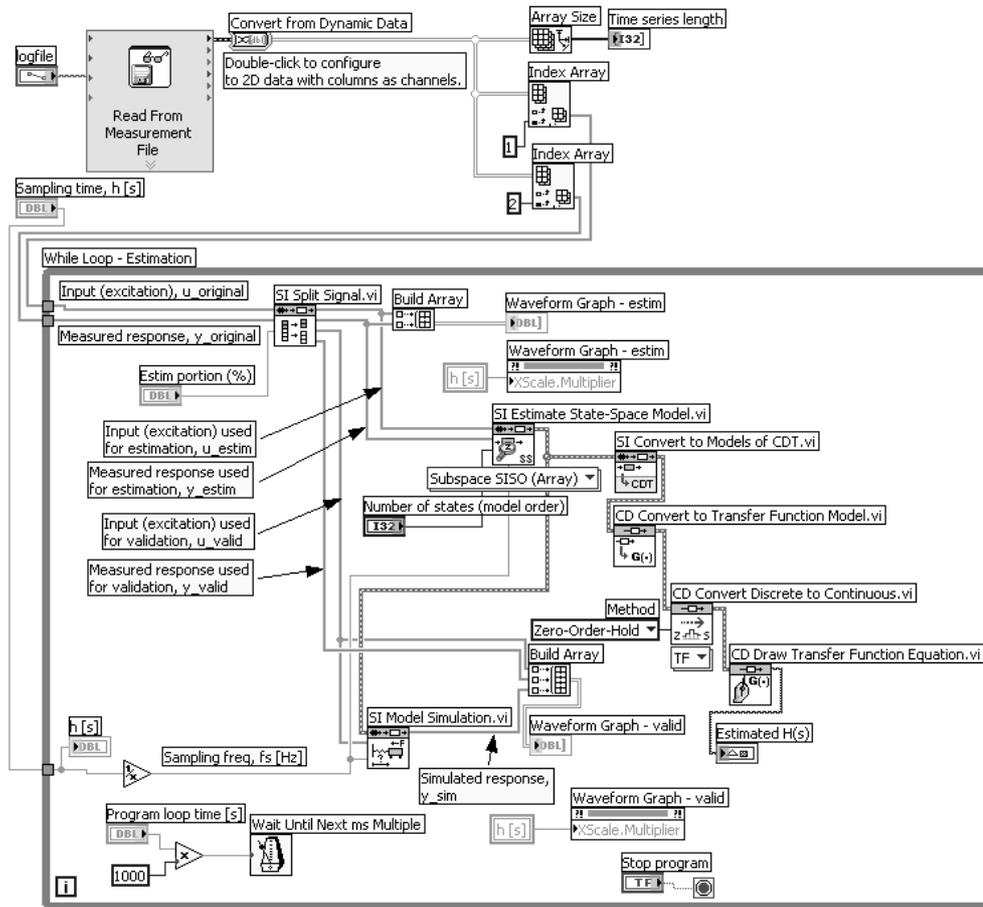


Figure 16.15: Example 16.4: The block diagram of the LabVIEW program `sysid.vi`.

estimation and validation (simulation) according to the **Estim Portion (%)** value.

3. The function **SI Estimate State-Space Model**⁶ estimates a discrete-time state-space model using a subspace estimation method.
4. The estimated model is eventually converted to a continuous-time transfer function, $H_{cont}(s)$, using functions in the Control and Simulation toolkit (at the right part of the block diagram). The result is (16.57).
5. The function **SI Model Simulation** simulates the estimated model using u_{valid} as input signal. The simulated response, y_{sim} , is shown

⁶SI = System Identification

together with *yvalid* in the chart at the right side of the front panel.

[End of Example 16.4]

Chapter 17

State estimation with observers

17.1 Introduction

An *observer* is an algorithm for *estimating the values of state variables* of a dynamic system. Why can such state estimates be useful?

- **Supervision:** State estimates can provide valuable information about important variables in a physical process, for example feed composition to a reactor, environmental forces acting on a ship, load torques acting on a motor, etc.
- **Control:** In general, the more information the controller has about the process it controls, the better (more accurate) it can control it. In particular, some control methods assumes that the state of the process to be controlled are known. If the state variables are not measured, they may be estimated, and the estimates can be used by the controller as if they were measurement.

Note that even process disturbances and process parameters can be estimated. The clue is to model the disturbances or parameters as ordinary state variables.

Relevant control methods that may benefit from state estimators are:

- *Feedforward control* [5], where the feedforward can be based on estimated disturbances.
- *Cascade control* [5], where the inner loops can be based on estimated states.

- *Feedback linearization*, see Section 20, where the feedbacks can be based on estimated states.
- *LQ (linear quadratic) optimal control*, see Section 21, where the feedbacks can be based on estimated states.
- *Model-based predictive control (MPC)*, see Section 22, where the prediction of future behaviour and optimization can be based on an estimated present state.

Observers are calculated from specified estimator *error dynamics*, or in other words: how fast and stable you want the estimates to converge to the real values (assuming you could measure them). An alternative to observers is the *Kalman Filter* which is an estimation algorithm based on stochastic theory. The Kalman Filter produces state estimates that contain a minimum amount of noise in the assumed presence of random process disturbances and random measurement noise. In observers however, such stochastic signals acting on the system is not in focus. The theory and implementation of observers are simpler than with Kalman Filters, and this is beneficial. One particular drawback about observers is that they are not straightforward to design for systems having more than one measurement, while this is straightforward for Kalman Filters. The Kalman Filter is described in Chapter 18, for discrete-time systems (the discrete-time Kalman Filter is more commonly used than the continuous-time Kalman Filter).

I have chosen to describe *continuous-time* – not discrete-time – observers. This makes the mathematical operations involved in design of the observer simpler. In a practical implementation you will use a computer, which operates in discrete time. Consequently, to obtain an observer ready for computer implementation, you will need to discretize the observer algorithm, but that is straightforward using Forward discretization. There is a potential danger about just discretizing a continuous-time algorithm: The resulting algorithm may become unstable if the sampling time is too large. However, with the computer power of today, there is probably no problem selecting sufficiently small sampling time in the implementation for any given application.

As with every model-based algorithm you should test your observer with a *simulated process* before applying it to the real system. You can implement a simulator in e.g. LabVIEW or MATLAB/Simulink since you already have a model (the observer is model-based). In the testing, you start with testing the observer with the *nominal model* in the simulator, including process and measurement noise. This is the model on which you are basing the observer. Secondly, you should *introduce some reasonable model errors*

by making the simulator model somewhat different from the observer model, and check if the observer still produces usable estimates.

17.2 How the observer works

The purpose of the observer is to estimate assumed unknown states in the process. Figure 17.1 shows a block diagram of a real system (process) with observer. The operating principle of the observer is that the

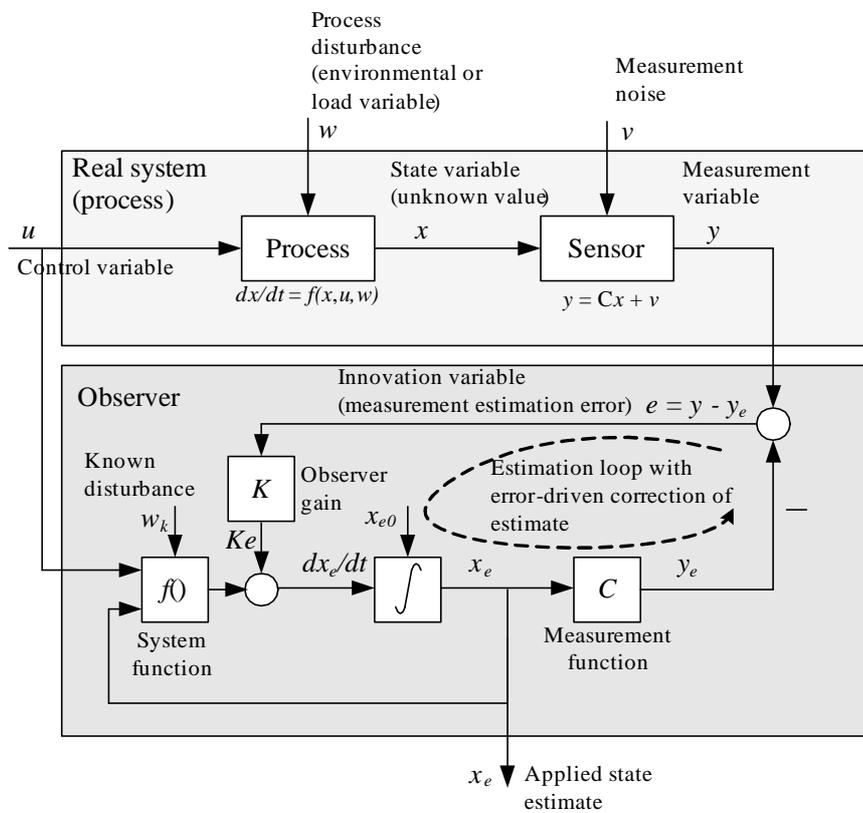


Figure 17.1: Real system (process) with observer

mathematical model of the process is running – or being simulated – in parallel with the process. If the model is perfect, x_e will be equal to the real states, x . But in practice there are model errors, so there will be a difference between x and x_e . It is always assumed that at least one of the states are measured. If there is a difference between x and x_e , there will also be a difference between the real measurement y and the estimated measurement y_e . This difference is the error e of the measurement estimate

y_e (it is also denoted the innovation variable):

$$e = y - y_e \quad (17.1)$$

This error is used to update the estimate via the observer gain K . Thus, the correction of the estimates is error-driven – which is the same principle as of a error-driven control loop.

The numerical value of the observer gain determines the strength of the correction. In the following section we will calculate a proper value of K .

17.3 How to design observers

17.3.1 Deriving the estimation error model

We assume that the process model is (the time t is omitted for simplicity)

$$\dot{x} = f(x, u, w) \quad (17.2)$$

where x is the state variable (vector), u is the control variable (vector), and w is the disturbance (vector). f is a possibly nonlinear function (vector).

Furthermore we assume that the process measurement y is given by

$$y = Cx + v \quad (17.3)$$

C is a matrix, and v is measurement noise, which is assumed to be random, and therefore not predictable. This noise influence the state estimates, and we will later see how we can reduce or minimize the influence of noise. If the sensor (including scaling) is producing a value in a proper engineering unit, e.g. m/s, °C or Pa, the element(s) of C have numerical value 1 or 0. For example, if the system has the two states $x_1 =$ position and $x_2 =$ speed, and only the position is measured with a sensor which gives the position in unit of meter, then

$$C = [1 \quad 0] \quad (17.4)$$

The state estimates, x_e , are calculated from the model together with a correction term being proportional to the measurement estimate error:

$$\dot{x}_e = f(x_e, u, w_k) + Ke \quad (17.5)$$

$$= f(x_e, u, w_k) + K(y - y_e) \quad (17.6)$$

$$= f(x_e, u, w_k) + K(Cx + v - Cx_e) \quad (17.7)$$

$$= f(x_e, u, w_k) + KC(x - x_e) + Kv \quad (17.8)$$

where w_k are process disturbances that are assumed to have known values by measurement etc.

The measurement estimate is given by

$$y_e = Cx_e \quad (17.9)$$

The measurement noise v is not included in (17.9) because v is assumed not to be predictable.

It is of course of crucial importance that the error of the state estimate is small. So, let us derive a model of the state estimate error. We define this error as

$$e_x = x - x_e \quad (17.10)$$

These variables are actually vectors. In detail (17.10) looks like this:

$$\begin{bmatrix} e_{x_1} \\ e_{x_2} \\ \vdots \\ e_{x_n} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} - \begin{bmatrix} x_{1_e} \\ x_{2_e} \\ \vdots \\ x_{n_e} \end{bmatrix} \quad (17.11)$$

Now, we subtract (17.8) from (17.2):

$$\dot{x} - \dot{x}_e = f(x, u, w) - [f(x_e, u, w_k) + KC(x - x_e) + Kv] \quad (17.12)$$

$$= [f(x, u, w) - f(x_e, u, w_k)] - KC(x - x_e) - Kv \quad (17.13)$$

Let us assume that the difference between the values of the two functions in the square bracket in (17.13) is caused by a small difference between x and x_e . Then we have

$$[f(x, u, w) - f(x_e, u, w_k)] \approx \left. \frac{\partial f(\cdot)}{\partial x} \right|_{x_e(t), u(t), w_k(t)} \cdot (x - x_e) \quad (17.14)$$

Here we define

$$A_c \equiv \left. \frac{\partial f(\cdot)}{\partial x} \right|_{x_e(t), u(t), w_k(t)} \quad (17.15)$$

A_c is the Jacobian (partial derivative) of the system function f (subindex c in A_c is for “continuous-time”), and it is the same as the resulting transition matrix after linearization of the non-linear state-space model, cf. Section 1.4. Now, (17.13) can be written

$$\dot{x} - \dot{x}_e = A_c(x - x_e) - KC(x - x_e) - Kv \quad (17.16)$$

or, using (17.10):

$$\dot{e}_x = A_c e_x - KC e_x - Kv \quad (17.17)$$

$$= (A_c - KC) e_x - Kv \quad (17.18)$$

which defines the *error dynamics* of the observer. (17.18) is the *estimation error model* of the observer. Now, assume that we disregard the impact that the measurement noise v has on e_x . Then the estimation error model is

$$\dot{e}_x = (A_c - KC) e_x \quad (17.19)$$

which is an autonomous system (i.e. not driven by external inputs). If that system is asymptotically stable, each of the error variables, e_{x_i} , will converge towards zero from any non-zero initial value. Of course this is what we want – namely that the estimation errors become zero. More specifically, the dynamics (with respect to speed and stability) of e_x is given by the eigenvalues of the system matrix,

$$A_e \equiv A_c - KC \quad (17.20)$$

And the observer gain K is a part of A_e ! (The next section explains how we can calculate K .)

Note that the matrices A_c and C in (17.20) are matrices of a linearized process model, assumed to be on the form

$$\Delta\dot{x} = A_c\Delta x + B_c\Delta u \quad (17.21)$$

$$\Delta y = C\Delta x + D\Delta u \quad (17.22)$$

As pointed out above, A_c can be calculated by linearization of the system function at the operating point:

$$A_c \equiv \left. \frac{\partial f(\cdot)}{\partial x} \right|_{x_e(t), u(t), w_k(t)} \quad (17.23)$$

To calculate the observer gain the B_c matrix is actually not needed, but when you use e.g. the LabVIEW function named **CD Ackerman.vi** to calculate K , you still need B_c , as demonstrated in Example 17.2. B_c is found by linearization:

$$B_c \equiv \left. \frac{\partial f(\cdot)}{\partial u} \right|_{x_e(t), u(t), w_k(t)} \quad (17.24)$$

In (17.22) the C and D matrices comes “automatically”. For example, if the system has the two states $x_1 =$ position and $x_2 =$ speed, and only the position is measured with a sensor which gives the position in unit of meter, then

$$C = [1 \quad 0] \quad (17.25)$$

And D is a matrix of proper dimension containing just zeros.

17.3.2 Calculation of the observer gain

Here is a procedure for calculating K :

1. Specify proper error dynamics in the term of the eigenvalues of (17.20). As explained below, these eigenvalues can be calculated from the specified *response time* of the observer.
2. Calculate K from the specified eigenvalues.

These two steps are explained in detail in the following.

Regarding step 1: What are “proper eigenvalues” of the error dynamics? There are many options. A good option is Butterworth eigenvalues, and we will concentrate on this option here. The characteristic equation from which the eigenvalues are calculated, is then a Butterworth polynomial. They are a common way to specify the denominator of a lowpass filter in the area of signal processing. The step response of such filters have a slight overshoot, with good damping. (Such step responses will also exist in an observer if a real state variable changes value abruptly.) Below are Butterworth polynomials of order 2, 3, and 4, which are the most relevant orders.¹

$$B_2(s) = (Ts)^2 + 1.4142(Ts) + 1 \quad (17.26)$$

$$B_3(s) = (Ts)^3 + 2(Ts)^2 + 2(Ts) + 1 \quad (17.27)$$

$$B_4(s) = (Ts)^4 + 2.6131(Ts)^3 + 3.4142(Ts)^2 + 2.6131(Ts) + 1 \quad (17.28)$$

The parameter T is used to define the speed of the response. (In normalized Butterworth polynomials $T = 1$.) The speed is inversely proportional to T , so the smaller T the faster response. (We will specify T more closely below.) To give an impression of Butterworth dynamics, Figure 17.2 shows the step response of Butterworth filters of order 2, 3, and 4, all with $T = 1$:

$$H_2(s) = \frac{1}{B_2(s)} \quad (17.29)$$

$$H_3(s) = \frac{1}{B_3(s)} \quad (17.30)$$

$$H_4(s) = \frac{1}{B_4(s)} \quad (17.31)$$

Let us define the response time T_r as the *observer response time* as the time that the step response needs to reach 63% of the steady state value of

¹Other orders can be found from the butter function in MATLAB and LabVIEW.

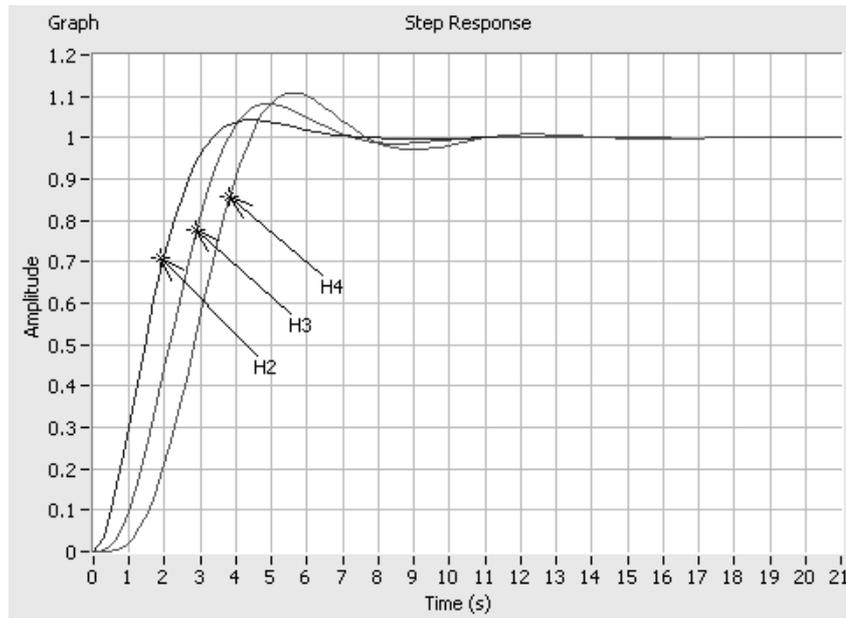


Figure 17.2: Step response of normalized Butterworth filters (with $T = 1$) of order 2, 3 and 4.

the response². From Figure 17.2 we can see that a *rough and simple* – still useful – estimate of T_r is

$$T_r \approx nT \quad (17.32)$$

where n is the order of the transfer function, which is the number of the poles and eigenvalues. T_r will be the *only tuning parameter of the observer!* Once T_r is specified, the T parameter to be used in the appropriate Butterworth polynomial among (17.26) – (17.28) is

$$T \approx \frac{T_r}{n} \quad (17.33)$$

And once the Butterworth polynomial among (17.26) – (17.28) is determined, you must calculate the eigenvalues $\{s_1, s_2, \dots, s_n\}$ as the roots of the polynomial:

$$\{s_1, s_2, \dots, s_n\} = \text{root}(B_n) \quad (17.34)$$

Figure 17.3 sums up the procedure of calculating the observer gain K .

²Similar to the time-constant of first order dynamic system.

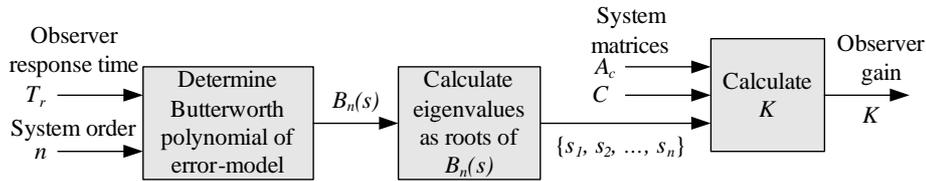


Figure 17.3: The procedure of calculating the observer gain K .

Calculation of observer gain in MATLAB and LabVIEW

Both MATLAB and LabVIEW has functions to calculate the roots of a polynomial (e.g. in MATLAB the function is **roots**).

Step 2 in the procedure list above is calculate the observer gain K from the specified eigenvalues $\{s_1, s_2, \dots, s_n\}$ of $A - KC$. We have

$$\text{eig}(A - KC) = \{s_1, s_2, \dots, s_n\} \quad (17.35)$$

As is known from mathematics, the eigenvalues are the s -roots of the characteristic equation:

$$\det[sI - (A - KC)] = (s - s_1)(s - s_2) \cdots (s - s_n) = 0 \quad (17.36)$$

By equating the polynomials on the left and the right side of (17.36) you can calculate the elements of K . Here are some options for doing this:

- **Manual calculations** (cf. Example 17.1).
- **Use functions in e.g. MATLAB or LabVIEW** (cf. Example 17.2):
 - In LabVIEW you can also use the function (block) **CD Ackerman.vi**, which is straightforward.
 - In MathScript you can use the function **acker** which is straightforward.
 - In MATLAB you can use the function **acker**. However, using **acker** is a little tricky: **acker** (MATLAB) calculates the gain K_1 so that the eigenvalues of the matrix $(A_1 - B_1 K_1)$ are as specified. **acker** is used as follows:
 $K_1 = \text{acker}(A_1, B_1, \text{eigenvalues})$
 But we need to calculate K so that the eigenvalues of $(A - KC)$

is as specified. Now, the eigenvalues of $A - KC$ are the same as the eigenvalues of

$$(A - KC)^T = A^T - C^T K^T \quad (17.37)$$

Therefore we use **acker** as follows:

```
K1=acker(A',C',eigenvalues);
K=K1'
```

In Example the observer gains are calculated with manual calculations, and in Example with MATLAB and LabVIEW.

Example 17.1 *Calculating the observer gain K with manual calculations*

Given a second order continuous-time model with the following system matrices:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, C = [1 \ 0], D = [0] \quad (17.38)$$

State-variable x_2 shall be estimated with an observer. $x_1 = y$ is measured. We specify that the response time of the estimator is 0.2 s, which implies that the parameter T in (17.33) is

$$T = \frac{T_r}{n} = \frac{0.2}{2} = 0.1 \text{ s} \quad (17.39)$$

The eigenvalues of the observer error dynamics are the roots of the characteristic equation:

$$0 = \det [sI - (A - KC)] \quad (17.40)$$

$$= \det \left\{ \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} [1 \ 0] \right) \right\} \quad (17.41)$$

$$= \det \left\{ \begin{bmatrix} s + K_1 & -1 \\ K_2 & s \end{bmatrix} \right\} \quad (17.42)$$

$$= s^2 + K_1 s + K_2 \quad (17.43)$$

The Butterworth polynomial, which is of second order, becomes

$$B_2(s) = (Ts)^2 + 1.4142(Ts) + 1 = T^2 s^2 + 1.4142Ts + 1 \quad (17.44)$$

Before comparing polynomials we divide (17.44) by T^2 so that it gets the same form as (17.43):

$$B_2^*(s) = s^2 + \frac{1.4142}{T}s + \frac{1}{T^2} \quad (17.45)$$

$$s^2 + \frac{1.4142}{T}s + \frac{1}{T^2} \equiv s^2 + K_1s + K_2 \quad (17.46)$$

Comparing coefficients between (17.43) and (17.45) gives the following observer gains:

$$K_1 = \frac{1.4142}{T} = \frac{1.4142}{0.1} = 14.1 \quad (17.47)$$

$$K_2 = \frac{1}{T^2} = \frac{1}{0.1^2} = 100 \quad (17.48)$$

[End of Example 17.1]

Example 17.2 *Calculating the observer gain K in MATLAB and LabVIEW*

Given a second order continuous-time model with the following system matrices:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, C = [1 \ 0], D = [0] \quad (17.49)$$

State-variable x_2 shall be estimated with an observer. $x_1 = y$ is measured. We specify that the response time of the estimator is 0.2 s, which implies that the parameter T in (17.33) is

$$T = \frac{T_r}{n} = \frac{0.2}{2} = 0.1 \text{ s} \quad (17.50)$$

The Butterworth polynomial becomes

$$B_2(s) = (Ts)^2 + 1.4142(Ts) + 1 = T^2s^2 + 1.4142Ts + 1 \quad (17.51)$$

from which we can calculate the roots, which are the specified eigenvalues of the observer.

MATLAB:

The following MATLAB-script calculates the estimator gain K :

```
A = [0,1;0,0];
B = [0;1];
```

196

```
C = [1,0];
D = [0];
n=2;
Tr=0.2;
T=Tr/n;
B2=[T*T,1.4142*T,1];
eigenvalues=roots(B2);
K1=acker(A',C',eigenvalues);
K=K1'
```

The result is

```
K =
14.142
100
```

which are the same as found with manual calculations in Example 17.1.

LabVIEW/MathScript:

The following MathScript-script calculates the estimator gain K :

```
A = [0,1;0,0];
B = [0;1];
C = [1,0];
D = [0];
n=2;
Tr=0.2;
T=Tr/n;
B2=[T*T,1.4142*T,1];
eigenvalues=roots(B2);
K=acker(A,C,eigenvalues,'o') %o for observer
```

The result is as before

```
K =
14.142
100
```

LabVIEW/Block diagram:

The function **CD Ackerman.vi** on the Control Design and Simulation Toolkit palette in LabVIEW can also be used to calculate the observer gain K . Figure 17.4 shows the front panel, and Figure 17.5 shows the block diagram of a LabVIEW program. The same K as above is obtained.

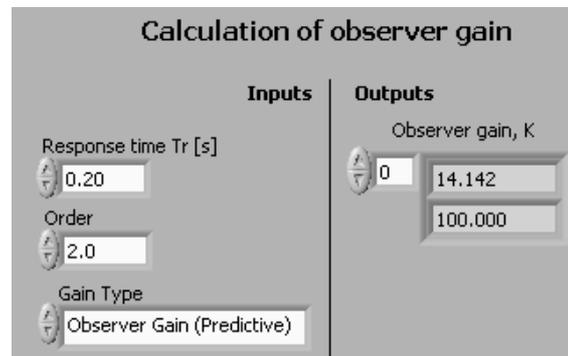


Figure 17.4: Example 17.2: Front panel of the LabVIEW program to calculate the observer gain K

[End of Example 17.2]

What if the estimates are too noisy?

Real measurement contains random noise. This noise will propagate through the observer via the term $Ke = K(y - y_e)$ where y is the more-or-less noisy measurement. This implies that the state estimate x_e will contain noise. What can you do if you regard the amount of noise to be too large? You can reduce the measurement-based updating of the estimate by ensuring that the observer gain K is be smaller. You can obtain this by increasing the specified response time T_r defined earlier in this section.³

17.4 Observability test of continuous-time systems

It can be shown that a necessary condition for placing the eigenvalues of the observer for a system at an arbitrary location in the complex plane is that the system is *observable*. A consequence of non-observability is that the **acker** functions in MATLAB and in LabVIEW used to calculate K (cf. Example 17.2) gives an error message.

³Don't just reduce the values of K directly. The consequence can be an unstable observer loop!

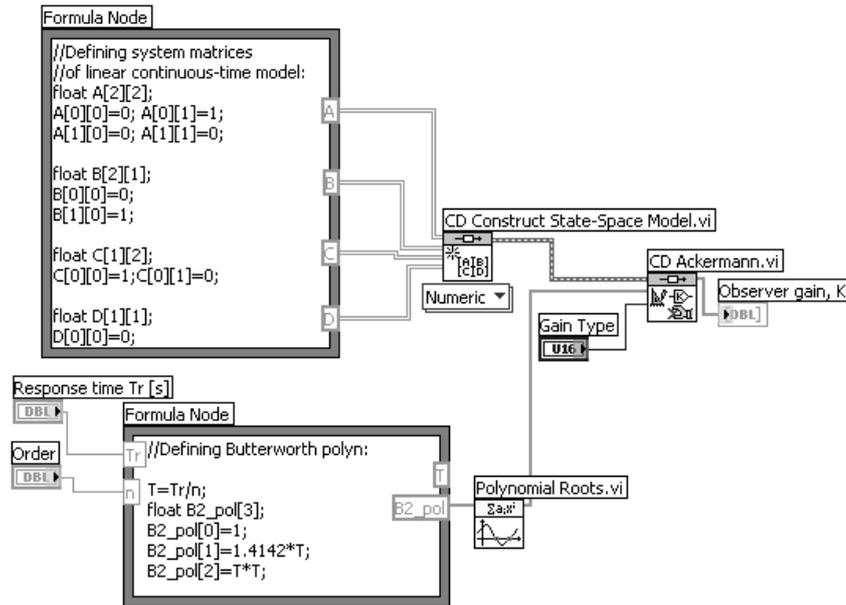


Figure 17.5: Example 17.2: Block diagram of the LabVIEW program to calculate the observer gain K

How is observability defined? A dynamic system given by

$$\dot{x} = Ax + Bu \tag{17.52}$$

$$y = Cx + Du \tag{17.53}$$

is said to be *observable* if every state $x(t_0)$ can be determined from the observation of $y(t)$ over a finite time interval, $[t_0, t_1]$.

How can you check if a system is non-observable? Let us make a definition:

Observability matrix:

$$M_{\text{obs}} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{17.54}$$

The following can be shown:

Observability Criterion:

The system (17.52) – (17.53) is observable if and only if the observability matrix M_{obs} has rank equal to n where n is the order of the system model (the number state variables).

The rank can be checked by calculating the determinant of M_{obs} . If the determinant is non-zero, the rank is full, and hence, the system is observable. If the determinant is zero, system is non-observable.

Non-observability has several consequences:

- The transfer function from the input variable u to the output variable y has an order that is less than the number of state variables (n).
- There are state variables or linear combinations of state variables that do not show any response.
- The eigenvalues of an observer for the system can not be placed freely in the complex plane, and the **acker** functions in MATLAB and in LabVIEW and also the **CD Ackerman.vi** in LabVIEW used to calculate K (cf. Example 17.2) gives an error message.

Example 17.3 Observability

Given the following state space model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & a \\ 0 & 0 \end{bmatrix}}_A \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_B u \quad (17.55)$$

$$y = \underbrace{\begin{bmatrix} c_1 & 0 \end{bmatrix}}_C \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \end{bmatrix}}_D u \quad (17.56)$$

The observability matrix is ($n = 2$)

$$M_{\text{obs}} = \begin{bmatrix} C \\ CA^{2-1} = CA \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} c_1 & 0 \end{bmatrix} \\ \begin{bmatrix} c_1 & 0 \end{bmatrix} \begin{bmatrix} 0 & a \\ 0 & 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} c_1 & 0 \\ 0 & ac_1 \end{bmatrix} \quad (17.57)$$

The determinant of M_{obs} is

$$\det(M_{\text{obs}}) = c_1 \cdot ac_1 - 0 \cdot 0 = a c_1^2 \quad (17.58)$$

The system is observable only if $a c_1^2 \neq 0$.

- Assume that $a \neq 0$ which means that the first state variable, x_1 , contains some non-zero information about the second state variable, x_2 . Then the system is observable if $c_1 \neq 0$, i.e. if x_1 is measured.

- Assume that $a = 0$ which means that x_1 contains no information about x_2 . In this case the system is non-observable despite that x_1 is measured.⁴

[End of Example 17.3]

17.5 Discrete-time implementation of the observer

The model defining the state estimate is given by (17.5), which is repeated here:

$$\dot{x}_e = f(x_e, u, w_k) + Ke \quad (17.59)$$

Of course, it is $x_e(t)$ that you want. It can be found easily by solving the above differential equation numerically. The easiest numerical solver, which probably is accurate enough – given that the sampling or discretization time, h , is small enough – is the Forward integration method. This method can be applied by substituting the time-derivative by the forward difference:

$$\dot{x}_e \approx \frac{x_e(t_{k+1}) - x_e(t_k)}{T_s} = \underbrace{f[x_e(t_k), u(t_k), w_k(t_k)]}_{f(\cdot, t_k)} + Ke(t_k) \quad (17.60)$$

Solving for $x_e(t_{k+1})$ gives the observer algorithm, which is ready for being programmed:

Observer:

$$x_e(t_{k+1}) = x_e(t_k) + T_s [f(\cdot, t_k) + Ke(t_k)] \quad (17.61)$$

An example of discrete-time implementation is given in Example 17.4.

It is important to prevent the state estimates from getting unrealistic values. For example, the estimate of a liquid level should not be negative. And it may be useful to give the user the option of resetting an estimate to a predefined value by clicking a button etc. The following code implements such limitation and reset of the estimate x_{1e} :

```
...
x1e_k1=x1e_k+Ts*(f_k1+K1*e); //Normal update of estimate.
```

⁴When I tried to calculate an observer gain for this system with $a = 0$, I got the following error message from LabVIEW/MathScript: “Error in function acker at line 9. Control Design Toolset: The system model is not observable.”

```

if(x1e_k1>x1_max) {x1e_k1=x1_max;} //Limit to max.
if(x1e_k1>x1_min) {x1e_k1=x1_min;} //Limit to min.
if(reset==1) {x1e_k1=x1_reset;} //Reset
...

```

17.6 Estimating parameters and disturbances with observers

In some applications it may be useful to estimate parameters and/or disturbances in addition to the “ordinary” state variables. One example is dynamic positioning systems for ship position control where the Kalman Filter is used to estimate environmental forces acting on the ship (these estimates are used in the controller as a feedforward control signal).

These parameters and/or disturbances *must be represented as state variables*. They represent additional state variables. The original state vector is *augmented* with these new state variables which we may denote the *augmentative states*. The observer is used to estimate the augmented state vector which consists of both the original state variables and the augmentative state variables. But how can you model these augmentative state variables? The augmentative model must be in the form of differential equations because that is the model used when designing observers. To set up an augmentative model you must make an assumption about the behaviour of the augmentative state. Let us look at some augmentative models.

- **Augmentative state is (almost) constant, or we do not know how it varies:** Both these assumptions are expressed with the following differential equations describing the augmentative state variable x_a :

$$\dot{x}_a(t) = 0 \quad (17.62)$$

This is the most common way to model the augmentative state.

- **Augmentative state has (almost) constant rate:** The corresponding differential equation is

$$\ddot{x}_a = 0 \quad (17.63)$$

or, in state space form, with $x_{a1} \equiv x_a$,

$$\dot{x}_{a1} = x_{a2} \quad (17.64)$$

$$\dot{x}_{a_2} = 0 \quad (17.65)$$

where x_{a_2} is another augmentative state variable.

Once you have defined the augmented model, you can design and implement the observer in the usual way. The observer estimates both the original states and the augmentative states.

The following example shows how the state augmentation can be done in a practical (simulated) application.

Example 17.4 *Observer for estimating level and flow*

Figure 17.6 shows a liquid tank with a level control system (PI controller) and an observer. (This system is also be used in Example 18.2 where a Kalman Filter is used in stead of an observer.) We will design an observer to estimate the outflow F_{out} . The level h is measured.

Mass balance of the liquid in the tank is (mass is ρAh)

$$\rho A_{tank} \dot{h}(t) = \rho K_p u - \rho F_{out}(t) \quad (17.66)$$

$$= \rho K_p u - \rho F_{out}(t) \quad (17.67)$$

After cancelling the density ρ the model is

$$\dot{h}(t) = \frac{1}{A_{tank}} [K_p u - F_{out}(t)] \quad (17.68)$$

We assume that we do not know how the outflow is actually varying, so we use the following augmentative model describing its behaviour:

$$\dot{F}_{out}(t) = 0 \quad (17.69)$$

The model of the system is given by (17.68) – (17.69). The parameter values of the tank are displayed (and can be adjusted) at the front panel, see Figure 17.6. The sampling time is

$$T_s = 0.1 \text{ s} \quad (17.70)$$

Although it is not strictly necessary, it is convenient to rename the state variables using standard names. So we define

$$x_1 = h \quad (17.71)$$

$$x_2 = F_{out} \quad (17.72)$$

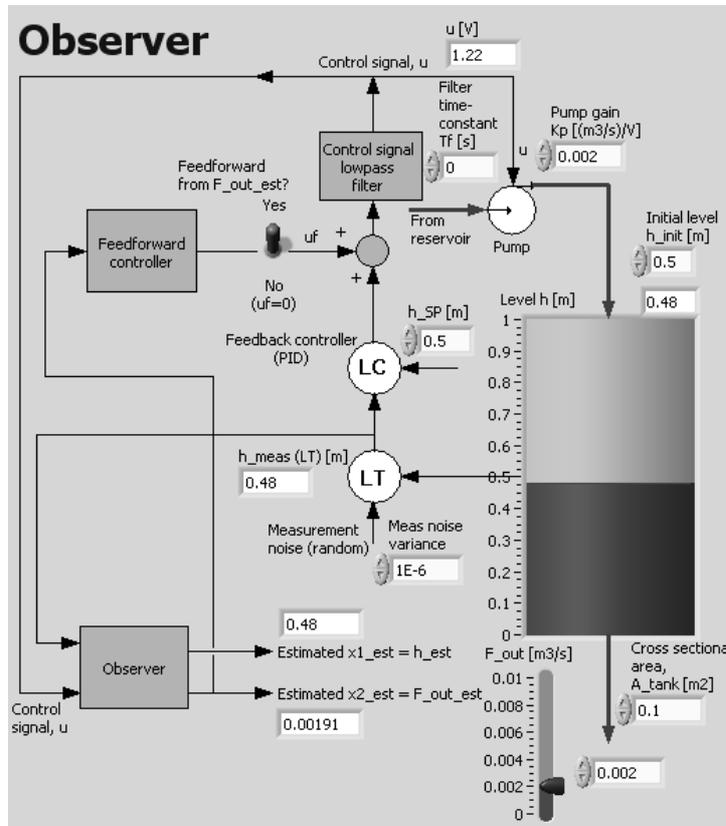


Figure 17.6: Example 17.4: Liquid tank with level control system and observer for estimation of outflow

The model (17.68) – (17.69) is now

$$\dot{x}_1(t) = \frac{1}{A_{tank}} [K_p u(t) - x_2(t)] \equiv f_1(\cdot) \quad (17.73)$$

$$\dot{x}_2(t) = 0 \equiv f_2(\cdot) \quad (17.74)$$

The measurement equation is

$$y = x_1 \quad (17.75)$$

The initial estimates are as follows:

$$x_{1_p}(0) = x_1(0) = y(0) \text{ (from the sensor)} \quad (17.76)$$

$$x_{2_p}(0) = 0 \text{ (assuming no information about initial value)} \quad (17.77)$$

The observer algorithm is, according to (17.61),

$$x_{1e}(t_{k+1}) = x_{1e}(t_k) + T_s [f_1(\cdot, t_k) + K_1 e] \quad (17.78)$$

$$= x_{1e}(t_k) + T_s \left\{ \frac{1}{A_{\text{tank}}} [K_p u(t_k) - x_{2e}(t_k)] + K_1 e \right\} \quad (17.79)$$

$$x_{2e}(t_{k+1}) = x_{2e}(t_k) + T_s [f_2(\cdot, t_k) + K_2 e] \quad (17.80)$$

$$= x_{2e}(t_k) + T_s K_2 e \quad (17.81)$$

To calculate observer gain K we need a linearized process model on the form

$$\Delta \dot{x} = A_c \Delta x + B_c \Delta u \quad (17.82)$$

$$\Delta y = C \Delta x + D \Delta u \quad (17.83)$$

Here:

$$A_c = \left[\begin{array}{cc} \frac{\partial f_1}{\partial x_1} = 0 & \frac{\partial f_1}{\partial x_2} = -\frac{1}{A_{\text{tank}}} \\ \frac{\partial f_2}{\partial x_1} = 0 & \frac{\partial f_2}{\partial x_2} = 0 \end{array} \right] \bigg|_{x_e(t_k), u(t_k)} \quad (17.84)$$

$$= \left[\begin{array}{cc} 0 & -\frac{1}{A_{\text{tank}}} \\ 0 & 0 \end{array} \right] \quad (17.85)$$

$$B_c = \left[\begin{array}{c} \frac{\partial f_1}{\partial u} = \frac{K_p}{A_{\text{tank}}} \\ \frac{\partial f_2}{\partial u} = 0 \end{array} \right] \bigg|_{x_e(t_k), u(t_k)} \quad (17.86)$$

$$= \left[\begin{array}{c} -\frac{1}{A_{\text{tank}}} \\ 0 \end{array} \right] \quad (17.87)$$

$$C = [1 \quad 0] \quad (17.88)$$

$$D = [0] \quad (17.89)$$

The Butterworth polynomial is (17.26) which is repeated here:

$$B_2(s) = (Ts)^2 + 1.4142(Ts) + 1 \quad (17.90)$$

where T is given by (17.33) which is repeated here:

$$T \approx \frac{T_r}{n} \quad (17.91)$$

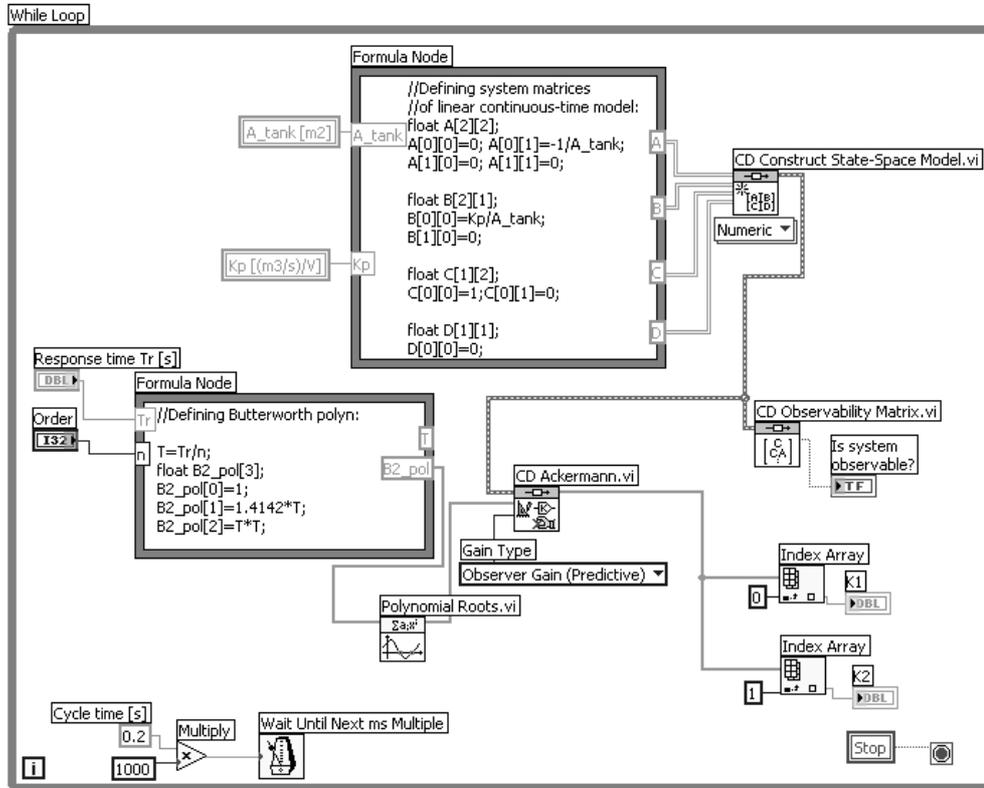


Figure 17.7: Example 17.4: While-loop for calculating the observer gain K

where $n = 2$ (the number of states). I specify the observer response time T_r to be

$$T_r = 2 \text{ s} \quad (17.92)$$

The observer gain K is calculated using function blocks in LabVIEW, see Figure 17.5. The result is

$$K = \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} = \begin{bmatrix} 1.414 \\ -0.1 \end{bmatrix} \quad (17.93)$$

Figure 17.8 shows the responses after a stepwise change of the outflow. (The level is controlled with a PI controller with settings $K_c = 10$ and $T_i = 10$ s.) The figure shows the “real” (simulated) and estimated level and outflow. We see from the lower chart in the figure that the Kalman Filter seems to estimate the outflow well, with response time approximately 2 sec, as specified, and with zero error in steady state.

Figure 17.9 shows the implementation of the observer with C-code in a Formula Node. (The Formula Node is just one part of the block diagram.)

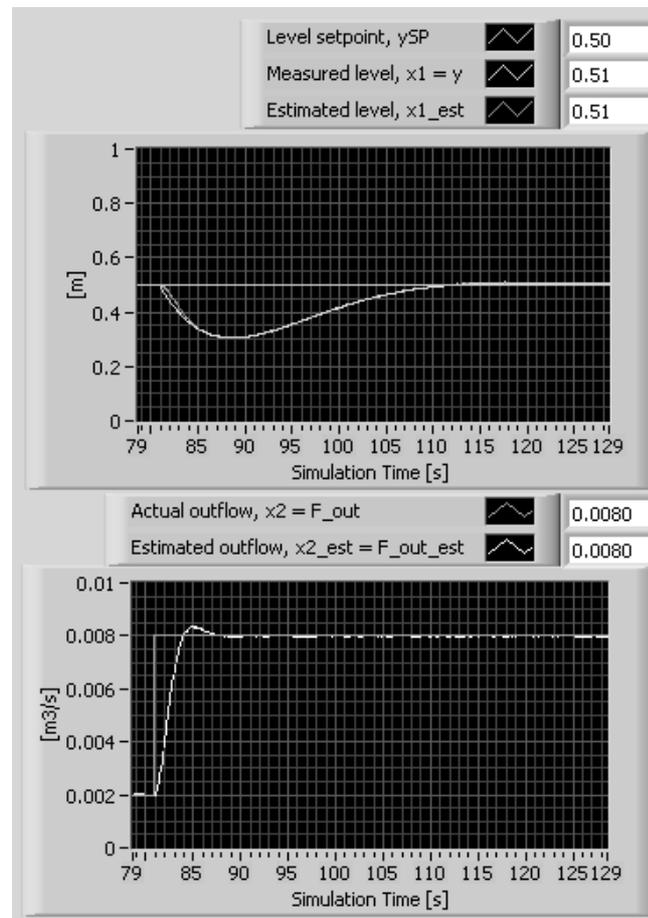


Figure 17.8: Example 17.4: The responses after a stepwise change of the outflow.

The total block diagram consists of one While loop where the observer gains are calculated, and one Simulation loop containing the Formula Node, PID controller, and the tank simulator.) Limitation of the estimated states to maximum and minimum values is included in the code. The input a is used to force the observer to run just as a simulator which is very useful at sensor failure, cf. Section 17.8.

[End of Example 17.4]

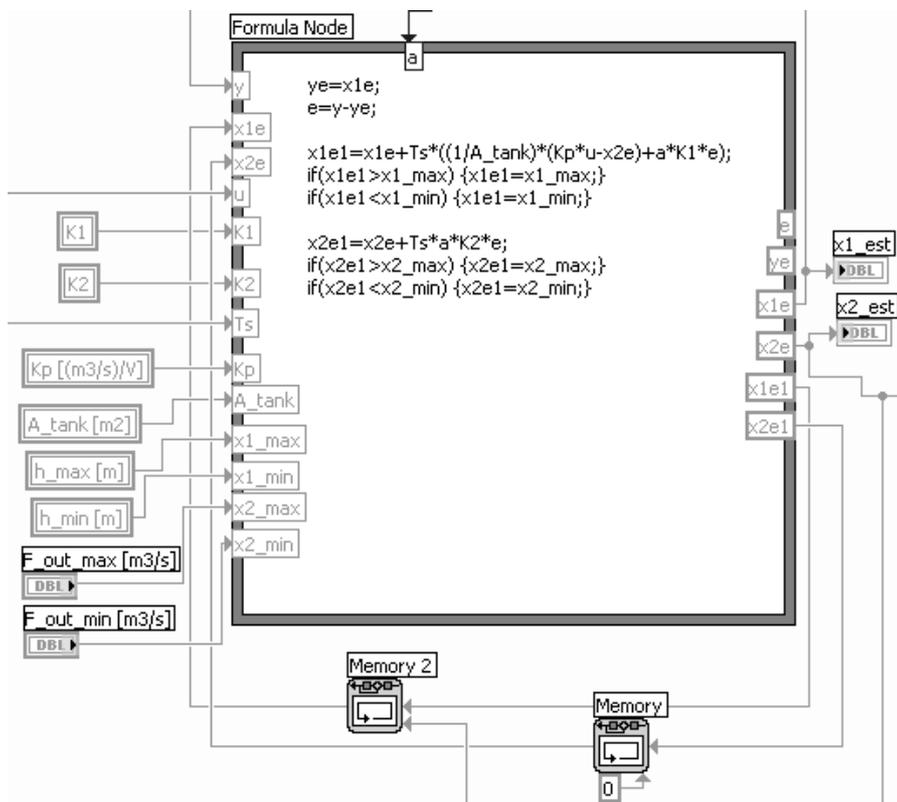


Figure 17.9: Example 17.4: Implementation of the observer in a Formula Node. (The observer gain K is fetched from the While loop in the Block diagram, see Figure 17.7, using local variables.)

17.7 Using observer estimates in controllers

In the introduction to this chapter are listed several control functions which basically assumes that measurements of states and/or disturbances (loads) are available. If measurements from “hard-sensors” for some reason are not available, you can try using an estimate as provided by a soft-sensor as an observer (or Kalman Filter) in stead. One such control function is feedforward control. Figure 17.10 shows feedforward from estimated disturbance.

Example 17.5 *Level control with feedforward from estimated disturbance (load)*

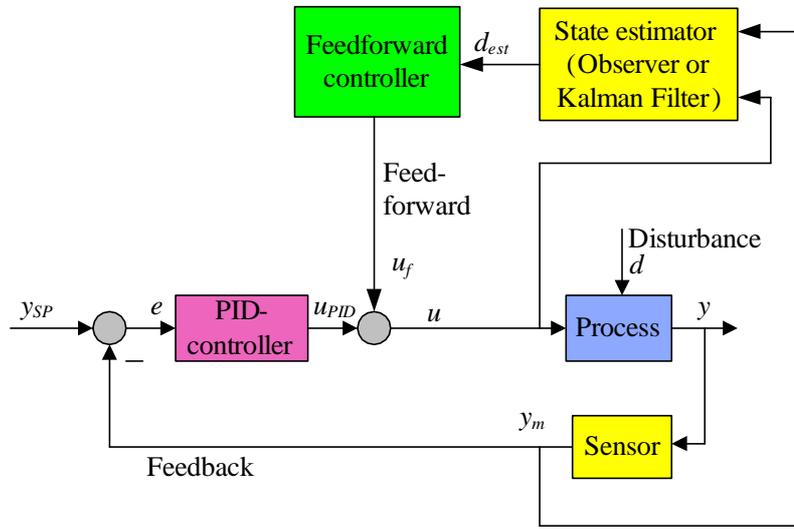


Figure 17.10: Control system including feedforward control from estimated disturbance (with observer or Kalman Filter)

Figure 17.6 in Example 17.4 shows the front panel of a LabVIEW program of a simulated level control system. On the front panel is a switch which can be used to activate feedforward from estimated outflow, $F_{out_{est}}$. The estimator for $F_{out_{est}}$ based on observer was derived in that example. Let us now derive the feedforward controller, and then look at simulations of the control system.

The feedforward controller is derived from a mathematical model of the process. The model is given by (17.68), which is repeated here:

$$\dot{h}(t) = \frac{1}{A_{tank}} [K_p u - F_{out}(t)] \quad (17.94)$$

Solving for the control variable u , and substituting process output variable h by its setpoint h_{SP} gives the feedforward controller:

$$u_f(t) = \underbrace{\frac{A_{tank} \dot{h}_{SP}(t)}{K_p}}_{u_{f_{SP}}} + \underbrace{\frac{F_{out}(t)}{K_p}}_{u_{f_d}} \quad (17.95)$$

Let us assume that the level setpoint h_{SP} is constant. Then, $\dot{h}_{SP}(t) = 0$, and the feedforward controller becomes

$$u_f(t) = \frac{F_{out}(t)}{K_p} \quad (17.96)$$

Assuming that the estimate $F_{out_{est}}(t)$ is used in stead of F_{out} , the feedforward controller becomes

$$u_f(t) = \frac{F_{out_{est}}(t)}{K_p} \quad (17.97)$$

Let us look at a simulation where the outflow has been changed as a step from 0.002 to 0.008 m³/s. Figure 17.11 shows the level response with feedforward. Compare with Figure 17.8 which shows the response *without*

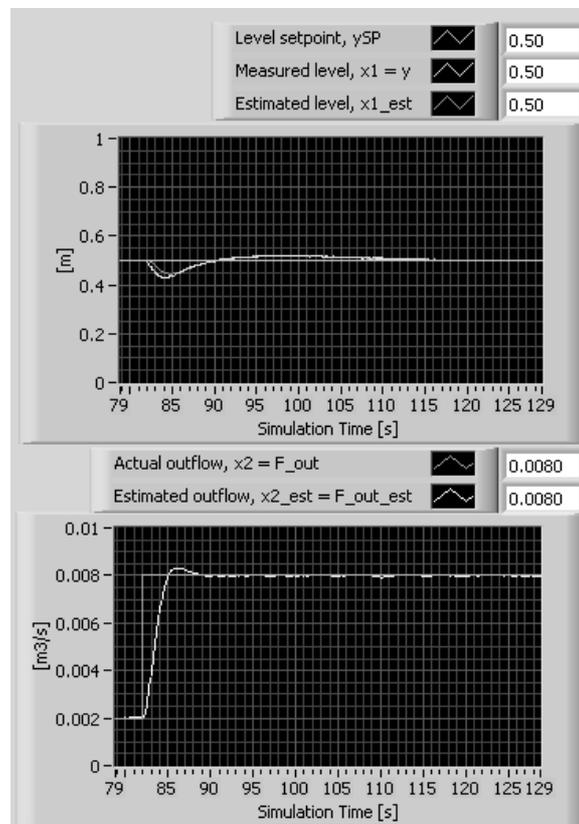


Figure 17.11: Example 17.5: Level response with feedforward from estimated outflow

feedforward. There is a substantial improvement by using feedforward from outflow– even if the outflow was not measured (only estimated)!

[End of Example 17.5]

17.8 Using observer for increased robustness of feedback control at sensor failure

If in a feedback control system the sensor fails so that the controller (e.g. a PID controller) receives an erroneous measurement signal, then the controller will adjust the control signal to a too large or a too low value. For example, assume that the level sensor fails and sends zero level measurement signal to the level controller. Then the level controller adjusts the control signal to maximum value, causing the tank to become full.

This problem can be solved as follows:

- Base the feedback on the estimated measurement, y_e , as calculated by an observer (or a Kalman Filter).
- While the sensor is failing (assuming some kind of measurement error detection has been implemented, of course): Prohibit the estimate from being updated by the (erroneous) measurement. This can be done by simply multiplying the term Ke by a factor, say a , so that the resulting estimator formula is

$$x_e(t_{k+1}) = x_e(t_k) + T_s [f(\cdot, t_k) + aKe(t_k)] \quad (17.98)$$

a is set to 1 is the default value, to be used when x_e is to be updated by the measurement (via the measurement estimate error e). a is set to 0 when x_e shall not be updated, implying that x_e effectively is

$$x_e(t_{k+1}) = x_e(t_k) + T_s f(\cdot, t_k) \quad (17.99)$$

which is just a *simulator* of the process. So, the controller uses a more-or-less correct simulated measurement in stead of an erroneous real measurement. *This will continue the “normal” operation of the control system, delaying or preventing dangerous situations.*

Example 17.6 Increased robustness of level control system with observer at sensor failure

This example is based on the level control system studied earlier in this chapter.

The following two scenarios are simulated. In both, the level sensor fails by suddenly producing zero voltage (indicating zero level) at some point of time.

- Scenario 1 (not using observer):** Nothing special has been done to handle the sensor failure. The level controller continues to control using the erroneous level measurement. (Actually, the observer is not in use.) The measurement value of zero causes the controller to act as the tank actually is empty, thereby increasing the control signal to the inlet pump to maximum, causing the tank to become full (which could be a dangerous situation in certain cases or with other processes). Figure 17.12 shows the simulated responses.

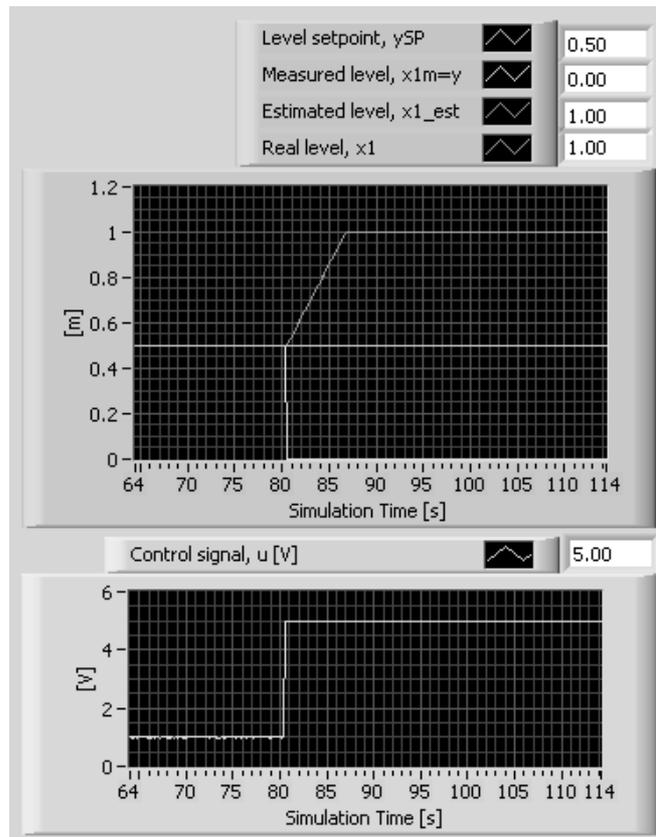


Figure 17.12: Example 17.6: Scenario 1: Simulation of level control system with sensor failure.

- Scenario 2 (using observer):** The level controller uses continuously the estimated level as calculated by the observer for feedback control. When the sensor fails (as detected by some assumed error-detection algorithm or device), the state estimates are prevented from being updated by the measurement. This is done by setting the parameter a in (17.98) equal to zero, and consequently the observer just runs as a simulator. To illustrate that the control

system continues to work well after the has sensor failed, the level setpoint is changed from 0.5 to 0.7 m. The outflow is changed towards the end of the simulation.

The simulations show that the control system continues to work despite the sensor failure: The level follows the setpoint. However, when the outflow is increased, the level is decreasing. This is because the estimator is not able to estimate the outflow correctly since the observer has no measurement-based update of the estimates. So, the control system may work well for some time, but not for ever because unmodeled disturbances can cause the states to diverge from the true states. *Still the robustness against sensor failure has been largely improved!*

[End of Example 17.6]

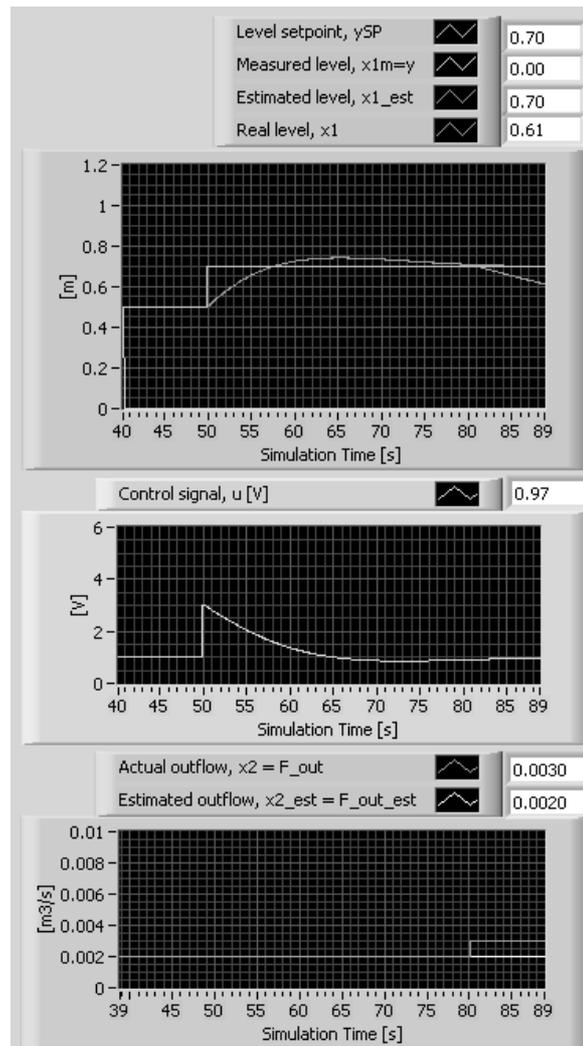


Figure 17.13: Example 17.6: Scenario 2: Simulation of level control system with sensor failure. Robustness is increased thanks to the observer!

Chapter 18

State estimation with Kalman Filter

18.1 Introduction

This chapter describes the *Kalman Filter* which is a commonly used method to *estimate the values of state variables* of a dynamic system that is excited by stochastic (random) disturbances and stochastic (random) measurement noise. Such state estimates may be useful for the purpose of *supervision and control*, cf. the introductory section to the chapter about observers, see page 17.

The Kalman Filter algorithm was developed by Rudolf E. Kalman around 1960 [7]. There is a continuous-time version of the Kalman Filter and several discrete-time versions. (The discrete-time versions are immediately ready for implementation in a computer program.) Here the *predictor-corrector* version of the discrete-time Kalman Filter will be described. This version seems to be the most commonly used version.

An alternative to Kalman Filters are *observers*. Observers have the same structure as the Kalman Filter, but they are calculated from specified estimator error dynamics, or in other words: how fast and stable you want the estimates to converge to the real values (assuming you could measure them). The theory and implementation of observers are simpler than with Kalman Filters, and this is beneficial. One particular drawback about observers is that they are not straightforward to design for systems having more than one measurement, while this is straightforward for Kalman Filters. Observers are described in Chapter 17.

As with every model-based algorithm you should test your Kalman Filter with a *simulated process* before applying it to the real system. You can implement a simulator in e.g. LabVIEW or MATLAB/Simulink since you already have a model (the Kalman Filter is model-based). In the testing, you start with testing the Kalman Filter with the *nominal model* in the simulator, including process and measurement noise. This is the model on which you are basing the Kalman Filter. Secondly, you should *introduce some reasonable model errors* by making the simulator model somewhat different from the Kalman Filter model, and check if the Kalman Filter still produces usable estimates.

18.2 Observability of discrete-time systems

A necessary condition for the Kalman Filter to work correctly is that the system for which the states are to be estimated, is *observable*. Therefore, you should check for observability before applying the Kalman Filter. (There may still be other problems that prevent the Kalman Filter from producing accurate state estimates, as a faulty or inaccurate mathematical model.)

Observability of discrete-time systems can be defined as follows [18]: The discrete-time system

$$x(k+1) = Ax(k) + Bu(k) \quad (18.1)$$

$$y(k) = Cx(k) + Du(k) \quad (18.2)$$

is observable if there is a finite number of time steps k so that knowledge about the input sequence $u(0), \dots, u(k-1)$ and the output sequence $y(0), \dots, y(k-1)$ is sufficient to determine the initial state of the system, $x(0)$.

Let us derive a criterion for the system to be observable. Since the influence of input u on state x is known from the model, let us for simplicity assume that $u(k) = 0$. From the model (18.1) – (18.2) we get

$$y(0) = Cx(0) \quad (18.3)$$

$$y(1) = Cx(1) = CAx(0) \quad (18.4)$$

$$\vdots$$

$$y(n-1) = CA^{n-1}x(0) \quad (18.5)$$

which can be expressed compactly as

$$\underbrace{\begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}}_{M_{\text{obs}}} x(0) = \underbrace{\begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(n-1) \end{bmatrix}}_Y \quad (18.6)$$

Let us make a definition:

Observability matrix:

$$M_{\text{obs}} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad (18.7)$$

(18.6) has a unique solution only if the rank of M_{obs} is n . Therefore:

Observability Criterion:

The system (18.1) – (18.2) is observable if and only if the observability matrix has rank equal to n where n is the order of the system model (the number state variables).

The rank can be checked by calculating the determinant of M_{obs} . If the determinant is non-zero, the rank is full, and hence, the system is observable. If the determinant is zero, system is non-observable.

Non-observability has several consequences:

- The transfer function from the input variable u to the output variable y has an order that is less than the number of state variables (n).
- There are state variables or linear combinations of state variables which do not show any response.
- The steady-state value of the Kalman Filter gain can not be computed. This gain is used to update the state estimates from measurements of the (real) system.

Example 18.1 *Observability*

Given the following state space model:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}}_A \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_B u(k) \quad (18.8)$$

$$y(k) = \underbrace{\begin{bmatrix} c_1 & 0 \end{bmatrix}}_C \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \underbrace{[0]}_D u(k) \quad (18.9)$$

The observability matrix is ($n = 2$)

$$M_{\text{obs}} = \begin{bmatrix} C \\ CA^{2-1} = CA \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} c_1 & 0 \end{bmatrix} \\ \begin{bmatrix} c_1 & 0 \end{bmatrix} \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} c_1 & 0 \\ c_1 & ac_1 \end{bmatrix} \quad (18.10)$$

The determinant of M_{obs} is

$$\det(M_{\text{obs}}) = c_1 \cdot ac_1 - c_1 \cdot 0 = a c_1^2 \quad (18.11)$$

The system is observable only if $a c_1^2 \neq 0$.

- Assume that $a \neq 0$ which means that the first state variable, x_1 , contains some non-zero information about the second state variable, x_2 . Then the system is observable if $c_1 \neq 0$, i.e. if x_1 is measured.
- Assume that $a = 0$ which means that x_1 contains no information about x_2 . In this case the system is non-observable despite that x_1 is measured.

[End of Example 18.1]

18.3 The Kalman Filter algorithm

18.3.1 The basic Kalman Filter algorithm

The *Kalman Filter* is a state estimator which produces an optimal estimate in the sense that *the mean value of the sum (actually of any linear combination) of the estimation errors* gets a minimal value. In other words, The Kalman Filter gives the following sum of squared errors:

$$E[e_x^T(k)e_x(k)] = E[e_{x_1}^2(k) + \dots + e_{x_n}^2(k)] \quad (18.12)$$

a minimal value. Here,

$$e_x(k) = x_{est}(k) - x(k) \quad (18.13)$$

is the estimation error vector. (The Kaman Filter estimate is sometimes denoted the “least mean-square estimate”.) This assumes actually that the model is linear, so it is not fully correct for nonlinear models. It is assumed that the system for which the states are to be estimated is excited by *random (“white”) disturbances (or process noise)* and that the measurements (there must be at least one real measurement in a Kalman Filter) contain *random (“white”) measurement noise*.

The Kalman Filter has many applications, e.g. in dynamic positioning of ships where the Kalman Filter estimates the position and the speed of the vessel and also environmental forces. These estimates are used in the positional control system of the ship. The Kalman Filter is also used in soft-sensor systems used for supervision, in fault-detection systems, and in Model-based Predictive Controllers (MPCs) which is an important type of model-based controllers.

The Kalman Filter algorithm was originally developed for systems assumed to be represented with a *linear* state-space model. However, in many applications the system model is *nonlinear*. Furthermore the linear model is just a special case of a nonlinear model. Therefore, I have decided to present the Kalman Filter for nonlinear models, but comments are given about the linear case. The Kalman Filter for nonlinear models is denoted the *Extended Kalman Filter* because it is an extended use of the original Kalman Filter. However, for simplicity we can just denote it the Kalman Filter, dropping “extended” in the name. The Kalman Filter will be presented without derivation.

The Kalman Filter presented below assumes that the system model consists of this discrete-time (possibly nonlinear) state space model:

$$x(k+1) = f[x(k), u(k)] + Gw(k) \quad (18.14)$$

and this (possibly nonlinear) measurement model:

$$y(k) = g[x(k), u(k)] + Hw(k) + v(k) \quad (18.15)$$

A *linear* model is just a special case:

$$x(k+1) = \underbrace{Ax(k) + Bu(k)}_{=f} + Gw(k) \quad (18.16)$$

and

$$y(k) = \underbrace{Cx(k) + Du(k)}_{=g} + Hw(k) + v(k) \quad (18.17)$$

The models above contains the following variables and functions:

- x is the state vector of n state variables:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (18.18)$$

- u is the input vector of m input variables:

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} \quad (18.19)$$

It is assumed that the value of u is known. u includes control variables and known disturbances.

- f is the system vector function:

$$f = \begin{bmatrix} f_1() \\ f_2() \\ \vdots \\ f_n() \end{bmatrix} \quad (18.20)$$

where $f_i()$ is any nonlinear or linear function.

- w is random (white) disturbance (or process noise) vector:

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_q \end{bmatrix} \quad (18.21)$$

with auto-covariance

$$R_w(L) = Q\delta(L) \quad (18.22)$$

where Q (a $q \times q$ matrix of constants) is the auto-covariance of w at lag $L = 0$. $\delta(L)$ is the unit pulse function, cf. (15.25). A standard

assumption is that

$$Q = \begin{bmatrix} Q_{11} & 0 & 0 & 0 \\ 0 & Q_{22} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & Q_{nn} \end{bmatrix} = \text{diag}(Q_{11}, Q_{22}, \dots, Q_{nn}) \quad (18.23)$$

Hence, the number q of process disturbances is assumed to be equal to the number n of state variables. Q_{ii} is the variance of w_i .

- G is the process noise gain matrix relating the process noise to the state variables. It is common to assume that $q = n$, making G square:

$$G = \begin{bmatrix} G_{11} & 0 & 0 & 0 \\ 0 & G_{22} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & G_{nn} \end{bmatrix} \quad (18.24)$$

In addition it is common to set the elements of G equal to one:

$$G_{ii} = 1 \quad (18.25)$$

making G an identity matrix:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = I_n \quad (18.26)$$

- y is the measurement vector of r measurement variables:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix} \quad (18.27)$$

- g is the measurement vector function:

$$g = \begin{bmatrix} g_1() \\ g_2() \\ \vdots \\ g_r() \end{bmatrix} \quad (18.28)$$

where $g_i()$ is any nonlinear or linear function. Typically, g is a linear function on the form

$$g(x) = Cx \quad (18.29)$$

where C is the measurement gain matrix.

- H is a gain matrix relating the disturbances *directly* to the measurements (there will in addition be an indirect relation because the disturbances acts on the states, and some of the states are measured). It is however common to assume that H is a zero matrix of dimension $(r \times q)$:

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \cdots & \vdots \\ 0 & 0 & \cdots & H_{rq} \end{bmatrix} \quad (18.30)$$

- v is a random (white) measurement noise vector:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_r \end{bmatrix} \quad (18.31)$$

with auto-covariance

$$R_v(L) = R\delta(L) \quad (18.32)$$

where R (a $r \times r$ matrix of constants) is the auto-covariance of v at lag $L = 0$. A standard assumption is that

$$R = \begin{bmatrix} R_{11} & 0 & 0 & 0 \\ 0 & R_{22} & 0 & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & R_{rr} \end{bmatrix} = \text{diag}(R_{11}, R_{22}, \dots, R_{rr}) \quad (18.33)$$

Hence, R_{ii} is the variance of v_i .

Note: If you need to adjust the “strength” or power of the process noise w or the measurement noise v , you can do it by increasing the variances, Q and R , respectively.

Here you have the *Kalman Filter*: (The formulas (18.35) – (18.37) below are represented by the block diagram shown in Figure 18.1.)

Calculation of Kalman Filter state estimate:

1. This step is the *initial step*, and the operations here are executed only once. Assume that the initial guess of the state is x_{init} . The initial value $x_p(0)$ of the predicted state estimate x_p (which is calculated continuously as described below) is set equal to this initial value:

Initial state estimate

$$x_p(0) = x_{init} \quad (18.34)$$

2. Calculate the *predicted measurement estimate* from the predicted state estimate:

Predicted measurement estimate:

$$y_p(k) = g[x_p(k)] \quad (18.35)$$

(It is assumed that the noise terms $Hv(k)$ and $w(k)$ are not known or are unpredictable (since they are white noise), so they can not be used in the calculation of the predicted measurement estimate.)

3. Calculate the so-called *innovation process or variable* — it is actually the measurement estimate error — as the difference between the measurement $y(k)$ and the predicted measurement $y_p(k)$:

Innovation variable:

$$e(k) = y(k) - y_p(k) \quad (18.36)$$

4. Calculate the *corrected state estimate* $x_c(k)$ by adding the corrective term $Ke(k)$ to the predicted state estimate $x_p(k)$:

Corrected state estimate:

$$\underline{x_c(k) = x_p(k) + Ke(k)} \quad (18.37)$$

Here, K is the *Kalman Filter gain*. The calculation of K is described below.

Note: It is $x_c(k)$ that is used as the state estimate in applications¹.

About terminology: The corrected estimate is also denoted the *posteriori* estimate because it is calculated *after* the present measurement is taken. It is also denoted the *measurement-updated* estimate.

Due to the measurement-based correction term of the Kalman Filter, you can expect the errors of the state estimates to be smaller than if there were no such correction term. This correction can be regarded as a *feedback correction of the estimates*, and it is well known from dynamic system theory, and in particular control systems theory, that feedback from measurements reduces errors. This feedback is indicated in Figure 18.1.

5. Calculate the *predicted state estimate* for the next time step, $x_p(k+1)$, using the present state estimate $x_c(k)$ and the known input $u(k)$ in process model:

Predicted state estimate:

$$x_p(k+1) = f[x_c(k), u(k)] \quad (18.38)$$

¹Therefore, I have underlined the formula.

(It is assumed that the noise term $Gv(k)$ is not known or is unpredictable, since it is random, so it can not be used in the calculation of the state estimate.)

About terminology: The predicted estimate is also denoted the *priori* estimate because it is calculated *before* the present measurement is taken. It is also denoted the *time-updated* estimate.

(18.35) – (18.38) can be represented by the block diagram shown in Figure 18.1.

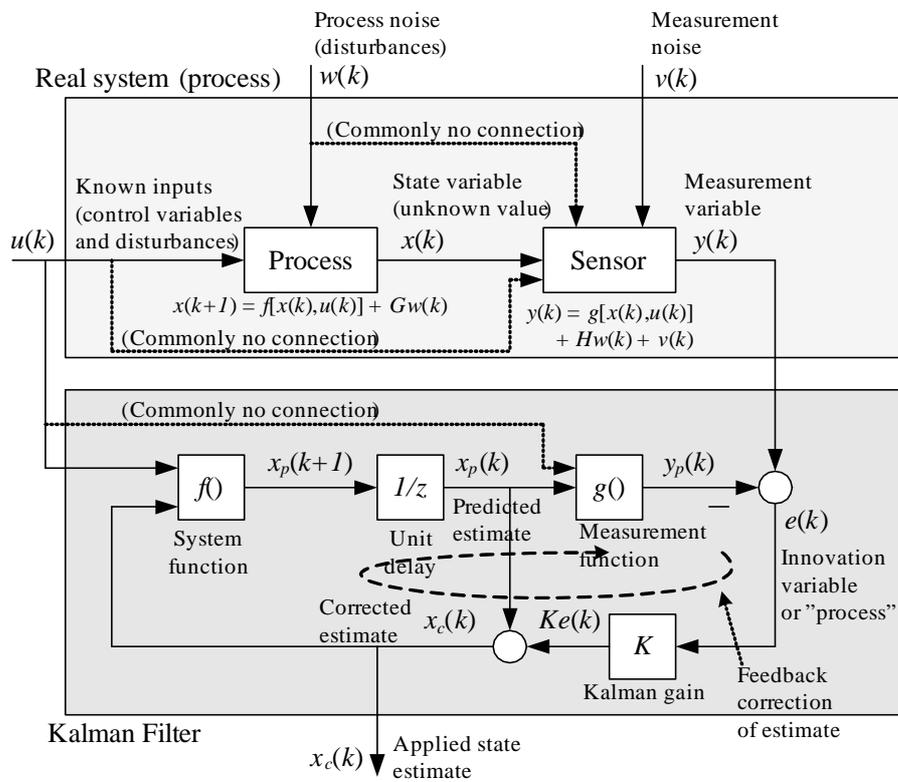


Figure 18.1: The Kalman Filter algorithm (18.35) – (18.38) represented by a block diagram

The *Kalman Filter gain* is a time-varying gain matrix. It is given by the algorithm presented below. In the expressions below the following matrices are used:

- Auto-covariance matrix (for lag zero) of the estimation error of the

corrected estimate:

$$P_c = R_{e_{x_c}}(0) = E \left\{ (x - m_{x_c})(x - m_{x_c})^T \right\} \quad (18.39)$$

- Auto-covariance matrix (for lag zero) of the estimation error of the predicted estimate:

$$P_d = R_{e_{x_d}}(0) = E \left\{ (x - m_{x_d})(x - m_{x_d})^T \right\} \quad (18.40)$$

- The transition matrix A of a linearized model of the original nonlinear model (18.14) calculated with the most recent state estimate, which is assumed to be the corrected estimate $x_c(k)$:

$$A = A_{disc} = \left. \frac{\partial f(\cdot)}{\partial x} \right|_{x_c(k), u(k)} \quad (18.41)$$

There are two alternative orders in calculating $A = A_{disc}$:

Number one: First linearize continuous-time model, then discretize:

1. Given the continuous-time nonlinear process model. Linearize it at the operating point to obtain

$$A_{cont} = \left. \frac{\partial f_{cont}(\cdot)}{\partial x} \right|_{x_c(k), u(k)} \quad (18.42)$$

2. Then calculate $A = A_{disc}$ as the discretized version of A_{cont} . You may use the Forward method of discretization in manual calculations:

$$A = A_{disc} = I + T_s A_{cont} = I + T_s \cdot \left. \frac{\partial f_{cont}(\cdot)}{\partial x} \right|_{x_c(k), u(k)} \quad (18.43)$$

However, it may be more convenient to use a proper function in the mathematical tool at hand, e.g. MATLAB or LabVIEW, to accomplish the discretization.

Number two: First discretize continuous-time model, then linearize:

1. Given the continuous-time nonlinear process model. Discretize it to obtain $x(k+1) = f[x(k), u(k)]$.
2. Then calculate $A = A_{disc}$ as the linearized version of $f[x(k), u(k)]$:

$$A_{disc} = \left. \frac{\partial f_{disc}(\cdot)}{\partial x} \right|_{x_c(k), u(k)} \quad (18.44)$$

It is my experience that order number one is the most convenient of the two: *First linearize continuous-time model, then discretize*. This is because it is somewhat easier to linearize the continuous-time model, and in addition you can exploit the computer for discretization.

- The measurement gain matrix C of a linearized model of the original nonlinear model (18.15) calculated with the most recent state estimate:

$$C = \left. \frac{\partial g(\cdot)}{\partial x} \right|_{x_c(k), u(k)} \quad (18.45)$$

However, it is very common that $C = g()$, and in these cases no linearization is necessary.

The *Kalman Filter gain* is calculated as follows (these calculations are repeated each program cycle):

Calculation of Kalman Filter gain:

1. This step is the initial step, and the operations here are executed only once. The initial value $P_p(0)$ can be set to some guessed value (matrix), e.g. to the identity matrix (of proper dimension).
2. Calculation of the *Kalman Gain*:

Kalman Filter gain:

$$K(k) = P_p(k)C^T [CP_p(k)C^T + R]^{-1} \quad (18.46)$$

3. Calculation of *auto-covariance of corrected state estimate error*:

Auto-covariance of corrected state estimate error:

$$P_c(k) = [I - K(k)C] P_p(k) \quad (18.47)$$

4. Calculation of *auto-covariance of the next time step of predicted state estimate error*:

Auto-covariance of predicted state estimate error:

$$P_p(k+1) = AP_c(k)A^T + GQG^T \quad (18.48)$$

18.3.2 Practical issues

1. **Order of formulas in the program cycle:** The Kalman Filter formulas can be executed in the following order:

- (18.46), Kalman Filter gain
- (18.36), innovation process (variable)
- (18.37), *corrected state estimate*, which is the state estimate to be used in applications
- (18.38), predicted state estimate of next time step
- (18.47), auto-covariance of error of corrected estimate
- (18.41), transition matrix in linear model
- (18.45), measurement matrix in linear model
- (18.48), auto-covariance of error of predicted estimate of next time step

2. **Limiting and resetting the state estimates.** It is important to prevent the state estimates from getting unrealistic values. For example, the estimate of a liquid level should not be negative. And it may be useful to give the user the option of resetting an estimate to a predefined value by clicking a button etc. The following code implements such limitation and reset of the corrected estimate x_{1c} :

```
...
x1c=x1p+K11*e;
if(x1c>x1_max) {x1c=x1_max;}
if(x1c<x1_min) {x1c=x1_min;}
if(reset==1) {x1c=x1_reset;}
...
```

And the following code implements such a limitation and reset of the predicted estimate x_{1p} :

```
...
x1p1=x1c+Ts*f1;
if(x1p1>x1_max) {x1p1=x1_max;}
if(x1p1<x1_min) {x1p1=x1_min;}
if(reset==1) {x1p1=x1_reset;}
...
```

3. **Steady-state Kalman Filter gain.** If the model is linear and time invariant (i.e. system matrices are not varying with time) the auto-covariances P_c and P_p will converge towards steady-state values. Consequently, the Kalman Filter gain will converge towards a

steady-state Kalman Filter gain value, K_s^2 , which can be pre-calculated. It is quite common to use only the steady-state gain in applications.

For a nonlinear system K_s may vary with the operating point (if the system matrix A of the linearized model varies with the operating point). In practical applications K_s may be re-calculated as the operating point changes.

Figure 18.2 illustrates the information needed to compute the steady-state Kalman Filter gain, K_s .

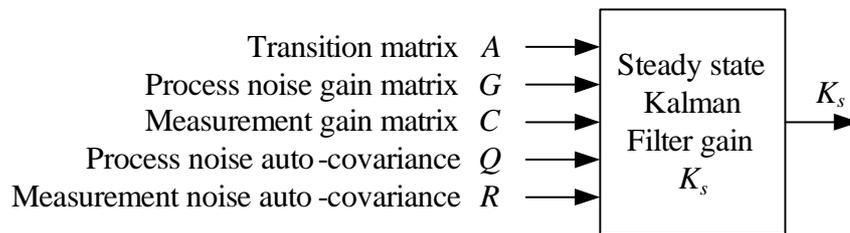


Figure 18.2: Illustration of what information is needed to compute the steady-state Kalman Filter gain, K_s .

4. **Dual-rate Kalman Filter.** Assume that the measurement rate is relatively small compared to the rate at which we want the Kalman Filter to produce a state estimate. Here are some cases where this apply:

- You want to have a larger rate of updating the predicted (simulated) state estimate than is used for reading the measurement from the sensor, perhaps because the process model needs a high-rate
- This is the case if the sensor for some reason is sampled slowly compared to the rate of calculating the predicted (or simulated) estimat. – or in the extreme case is deactivated due to e.g. failure. In such situations you can actually let the Kalman Filter run as normal, that is, with the same rate of both the corrector part and the predictor part – *but* you force the effective Kalman gain K to become zero. This can be done by simply multiplying K by a factor, say a , so that the corrected estimate is

$$x_c(k) = x_p(k) + aKe(k) \quad (18.49)$$

²MATLAB and LabVIEW have functions for calculating the steady-state Kalman Gain.

$a = 1$ is the default value, to be used when x_c is to be updated by the measurement (via the innovation process e), and $a = 0$ is used when x_c shall not be updated, implying that x_c is equal to x_p :

$$x_c(k) = x_p(k) + 0 \cdot Ke(k) = x_p(k) \quad (18.50)$$

18.3.3 Features of the Kalman Filter

1. **Model errors.** There are always model errors since no model can give a perfect description of a real (practical system). It is possible to analyze the implications of the model errors on the state estimates calculated by the Kalman Filter, but this will not be done in this book. However, in general, the estimation errors are smaller with the Kalman Filter than with a so-called *ballistic state estimator*, which is the state estimator that you get if the Kalman Filter gain K is set to zero. In the latter case there is no correction of the estimates. It is only the predictions (18.38) that make up the estimates. The Kalman Filter then just runs as a simulator.

Note that you can try to estimate model errors by *augmenting* the states with states representing the model errors. Augmented Kalman Filter is described in Section 18.5.

2. **The error-model:** Assuming that the system model is linear and that the model is correct (giving a correct representation of the real system), it can be shown that the behaviour of the error of the corrected state estimation, $e_{x_c}(k)$, cf. (18.13), is given by the following *error-model*:³

Error-model of Kalman Filter:

$$e_{x_c}(k+1) = (I - KC) Ae_{x_c}(k) + (I - KC) Gv(k) - Kw(k+1) \quad (18.51)$$

This model can be used to analyze the Kalman Filter.

Note: (18.13) is not identical to the auto-covariance of the estimation error which is

$$P_c(k) = E\{[e_{x_c}(k) - m_{x_c}(k)][e_{x_c}(k) - m_{x_c}(k)]^T\} \quad (18.52)$$

But (18.13) is the *trace* of P_c (the trace is the sum of the diagonal elements):

$$e_{x_c} = \text{trace}[P_c(k)] \quad (18.53)$$

³You can derive this model by subtracting the model describing the corrected state estimate from the model that describes the real state (the latter is simply the process model).

3. **The dynamics of the Kalman Filter.** The error-model (18.51) of the Kalman Filter represents a dynamic system. The dynamics of the Kalman Filter can be analyzed by calculating the eigenvalues of the system matrix of (18.51). These eigenvalues are

$$\{\lambda_1, \lambda_2, \dots, \lambda_n\} = \text{eig}[(I - KC)A] \quad (18.54)$$

4. **The stability of the Kalman Filter.** It can be shown that *the Kalman Filter always is an asymptotically stable dynamic system* (otherwise it could not give an optimal estimate). In other words, the eigenvalues defined by (18.54) are always inside the unity circle.
5. **Predictor type Kalman Filter.** In the *predictor type of the Kalman Filter* there is only one formula for the calculation of the state estimate:

$$x_{est}(k+1) = Ax_{est}(k) + K[y(k) - Cx_{est}(k)] \quad (18.55)$$

Thus, there is no distinction between the predicted state estimate and the corrected estimate (it the same variable). (Actually, this is the original version of the Kalman Filter[7].) The predictor type Kalman Filter has the drawback that there is a time delay of one time step between the measurement $y(k)$ and the calculated state estimate $x_{est}(k+1)$.

18.4 Tuning the Kalman Filter

Usually it is necessary to fine-tune the Kalman Filter when it is connected to the real system. The process disturbance (noise) auto-covariance Q and/or the measurement noise auto-covariance R are commonly used for the tuning. However, since R is relatively easy to calculate from a time series of measurements (using some variance function in for example LabVIEW or MATLAB), we only consider adjusting Q here.

What is good behaviour of the Kalman Filter? How can good behaviour be observed? It is when the estimates seems to have reasonable values as you judge from your physical knowledge about the physical process. In addition the estimates must not be too noisy! What is the cause of the noise? In real systems it is mainly the measurement noise that introduces noise into the estimates. How do you tune Q to avoid too noisy estimates? The larger Q the stronger measurement-based updating of the state estimates because a large Q tells the Kalman Filter that the variations in the real state variables are assumed to be large (remember that the process

noise influences on the state variables, cf. (18.15)). Hence, the larger Q the larger Kalman Gain K and the stronger updating of the estimates. But this causes more measurement noise to be added to the estimates because the measurement noise is a term in the innovation process e which is calculated by K :

$$x_c(k) = x_p(k) + Ke(k) \quad (18.56)$$

$$= x_p(k) + K \{g[x(k)] + v(k) - g[x_p(k)]\} \quad (18.57)$$

where v is real measurement noise. So, the main tuning rule is as follows: *Select as large Q as possible without the state estimates becoming too noisy.*

But Q is a matrix! How to select it “large” or “small”? Since each of the process disturbances typically are assumed to act on their respective state independently, Q can be set as a diagonal matrix:

$$Q = \begin{bmatrix} Q_{11} & 0 & 0 & 0 \\ 0 & Q_{22} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & Q_{nn} \end{bmatrix} = \text{diag}(Q_{11}, Q_{22}, \dots, Q_{nn}) \quad (18.58)$$

where each of the diagonal elements can be adjusted independently. If you do not have any idea about numerical values, you can start by setting all the diagonal elements to one, and hence Q is

$$Q = Q_0 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (18.59)$$

where Q_0 is the only tuning parameter. If you do not have any idea about a proper value of Q_0 you may initially try

$$Q_0 = 0.01 \quad (18.60)$$

Then you may adjust Q_0 or try to fine tune each of the diagonal elements individually.

18.5 Estimating parameters and disturbances with Kalman Filter

In many applications the Kalman Filter is used to estimate parameters and/or disturbances in addition to the “ordinary” state variables. One

example is dynamic positioning systems for ship position control where the Kalman Filter is used to estimate environmental forces acting on the ship (these estimates are used in the controller as a feedforward control signal).

These parameters and/or disturbances *must be represented as state variables*. They represent additional state variables. The original state vector is *augmented* with these new state variables which we may denote the *augmentative states*. The Kalman Filter is used to estimate the augmented state vector which consists of both the original state variables and the augmentative state variables. But how can you model these augmentative state variables? The augmentative model must be in the form of a difference equation because that is the model form when designing a Kalman Filter. To set up an augmentative model you must make an assumption about the behaviour of the augmentative state. Let us look at some augmentative models.

- **Augmentative state is (almost) constant:** The most common augmentative model is based on the assumption that the augmentative state variable x_a is slowly varying, almost constant. The corresponding differential equation is

$$\dot{x}_a(t) = 0 \quad (18.61)$$

Discretizing this differential equation with the Euler Forward method gives

$$x_a(k+1) = x_a(k) \quad (18.62)$$

which is a difference equation ready for Kalman Filter algorithm. It is however common to assume that the state is driven by some noise, hence the augmentative model become:

$$x_a(k+1) = x_a(k) + w_a(k) \quad (18.63)$$

where w_a is white process noise with assumed auto-covariance on the form $R_{w_a}(L) = Q_a\delta(L)$. As pointed out in Section 18.3, the variance Q_a can be used as a tuning parameter of the Kalman Filter.

- **Augmentative state has (almost) constant rate:** The corresponding differential equation is

$$\ddot{x}_a = 0 \quad (18.64)$$

or, in state space form, with $x_{a1} \equiv \dot{x}_a$,

$$\dot{x}_{a1} = x_{a2} \quad (18.65)$$

$$\dot{x}_{a2} = 0 \quad (18.66)$$

where x_{a_2} is another augmentative state variable. Applying Euler Forward discretization with sampling interval h [sec] to (18.65) – (18.66) and including white process noise to the resulting difference equations gives

$$x_{a_1}(k+1) = x_{a_1}(k) + hx_{a_2}(k) + w_{a_1}(k) \quad (18.67)$$

$$x_{a_2}(k+1) = x_{a_2}(k) + w_{a_2}(k) \quad (18.68)$$

Once you have defined the augmented model, you can design and implement the Kalman Filter in the usual way. The Kalman Filter then estimates both the original states and the augmentative states.

The following example shows how the state augmentation can be done in a practical (simulated) application. The example also shows how to use functions in LabVIEW and in MATLAB to calculate the steady state Kalman Filter gain.

Example 18.2 *Kalman Filter for estimating level and flow*

This example is essentially the same as Example 17.4 where an *observer* was used.

Figure 17.6 shows the system as drawn on the front panel of a LabVIEW based simulator. In stead of the observer (see down left on the front panel) a Kalman Filter is now used. We will design a steady state Kalman Filter to estimate the outflow F_{out} . The level h is measured.

Mass balance of the liquid in the tank is (mass is ρAh)

$$\rho A_{tank} \dot{h}(t) = \rho K_p u - \rho F_{out}(t) \quad (18.69)$$

$$= \rho K_p u - \rho F_{out}(t) \quad (18.70)$$

After cancelling the density ρ the model is

$$\dot{h}(t) = \frac{1}{A_{tank}} [K_p u - F_{out}(t)] \quad (18.71)$$

We assume that the unknown outflow is slowly changing, almost constant. We define the following augmentative model:

$$\dot{F}_{out}(t) = 0 \quad (18.72)$$

The model of the system is given by (18.71) – (18.72). Although it is not necessary, it is convenient to rename the state variables using standard names. So we define

$$x_1 = h \quad (18.73)$$

$$x_2 = F_{out} \quad (18.74)$$

The model (18.71) – (18.72) is now

$$\dot{x}_1(t) = \frac{1}{A_{tank}} [K_p u(t) - x_2(t)] \equiv f_{cont_1} \quad (18.75)$$

$$\dot{x}_2(t) = 0 \equiv f_{cont_2} \quad (18.76)$$

Applying Euler Forward discretization with time step T_s and including white disturbance noise in the resulting difference equations yields

$$x_1(k+1) = x_1(k) + \underbrace{\frac{T_s}{A_{tank}} [K_p u(k) - x_2(k)] + w_1(k)}_{f_1(\cdot)} \quad (18.77)$$

$$x_2(k+1) = \underbrace{x_2(k)}_{f_2(\cdot)} + w_2(k) \quad (18.78)$$

or

$$x(k+1) = f[x(k), u(k)] + w(k) \quad (18.79)$$

w_1 and w_2 are independent (uncorrelated) white process noises with assumed variances $R_{w_1}(L) = Q_1\delta(L)$ and $R_{w_2}(L) = Q_2\delta(L)$ respectively. Here, Q_1 and Q_2 are variances. The multivariable noise model is then

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \quad (18.80)$$

with auto-covariance

$$R_w(L) = Q\delta(L) \quad (18.81)$$

where

$$Q = \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \quad (18.82)$$

Assuming that the level x_1 is measured, we add the following measurement equation to the state space model:

$$y(k) = g[x_p(k), u(k)] + v(k) = x_1(k) + v(k) \quad (18.83)$$

where v is white measurement noise with assumed variance

$$R_v(L) = R\delta(L) \quad (18.84)$$

where R is the measurement variance.

The following numerical values are used:

$$\text{Sampling time: } T_s = 0.1 \text{ s} \quad (18.85)$$

$$A_{tank} = 0.1 \text{ m}^2 \quad (18.86)$$

$$K_p = 0.002 \text{ (m}^3/\text{s)/V} \quad (18.87)$$

$$Q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.0001 \end{bmatrix} \quad (18.88)$$

$$R = 0.0001 \text{ m}^2 \text{ (Gaussian white noise)} \quad (18.89)$$

We will set the initial estimates as follows:

$$x_{1p}(0) = x_1(0) = y(0) \text{ (from the sensor)} \quad (18.90)$$

$$x_{2p}(0) = 0 \text{ (assuming no information about initial value)} \quad (18.91)$$

The Kalman Filter algorithm is as follows: The predicted level measurement is calculated according to (18.35):

$$y_p(k) = g[x_p(k), u(k)] = x_{1p}(k) \quad (18.92)$$

with initial value as given by (18.90). The innovation variable is calculated according to (18.36), where y is the level measurement:

$$e(k) = y(k) - y_p(k) \quad (18.93)$$

The corrected state estimate is calculated according to (18.37):

$$x_c(k) = x_p(k) + Ke(k) \quad (18.94)$$

or, in detail:

$$\begin{bmatrix} x_{1c}(k) \\ x_{2c}(k) \end{bmatrix} = \begin{bmatrix} x_{1p}(k) \\ x_{2p}(k) \end{bmatrix} + \underbrace{\begin{bmatrix} K_{11} \\ K_{21} \end{bmatrix}}_K e(k) \quad (18.95)$$

This is the applied estimate!

The predicted state estimate for the next time step, $x_p(k+1)$, is calculated according to (18.38):

$$x_p(k+1) = f[x_c(k), u(k)] \quad (18.96)$$

or, in detail:

$$\begin{bmatrix} x_{1p}(k+1) \\ x_{2p}(k+1) \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} x_{1c}(k) + \frac{T_s}{A_{tank}} [K_p u(k) - x_{2c}(k)] \\ x_{2c}(k) \end{bmatrix} \quad (18.97)$$

To calculate the steady state Kalman Filter gain K_s the following information is needed, cf. Figure 18.2:

$$A = I + T_s \left. \frac{\partial f_{cont}}{\partial x} \right|_{x_p(k), u(k)} \quad (18.98)$$

$$= I + T_s \left[\begin{array}{cc} \left. \frac{\partial f_{cont_1}}{\partial x_1} = 0 \right|_{x_p(k), u(k)} & \left. \frac{\partial f_{cont_1}}{\partial x_2} = -\frac{1}{A_{tank}} \right|_{x_p(k), u(k)} \\ \left. \frac{\partial f_{cont_2}}{\partial x_1} = 0 \right|_{x_p(k), u(k)} & \left. \frac{\partial f_{cont_2}}{\partial x_2} = 0 \right|_{x_p(k), u(k)} \end{array} \right] \quad (18.99)$$

$$= \begin{bmatrix} 1 & -\frac{T_s}{A_{tank}} \\ 0 & 1 \end{bmatrix} \quad (18.100)$$

$$G = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_2 \text{ (identity matrix)} \quad (18.101)$$

$$C = \left. \frac{\partial g(\cdot)}{\partial x} \right|_{x_p(k), u(k)} \quad (18.102)$$

$$= \begin{bmatrix} 1 & 0 \end{bmatrix} \quad (18.103)$$

$$Q = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.0001 \end{bmatrix} \text{ (initially, may be adjusted)} \quad (18.104)$$

$$R = 0.000001 \text{ m}^2 \quad (18.105)$$

Figure 18.3 shows the “real” (simulated) and estimated level and outflow. We see from the lower chart in the figure that the Kalman Filter estimates the outflow well, and with zero error in steady state.

Figure 18.4 shows how the steady state Kalman Filter gain K_s is calculated using the **Kalman Gain** function. The figure also shows how to check for observability with the **Observability Matrix** function. The linear continuous-time model is discretized using the **Convert Continuous to Discrete** function. Figure 18.5 shows the values of the Kalman gains.

Figure 18.6 shows the implementation of the Kalman Filter equations in a Formula Node. (The Formula Node is just one part of the block diagram. The total block diagram consists of one While loop where the Kalman gains are calculated, and one Simulation loop containing the Formula Node, PID controller, and the tank simulator.)

The steady state Kalman Gain K_s is calculated in the LabVIEW program. Alternatively, K_s can be calculated in MATLAB, as shown below. The **dlqe**⁴ function belongs to the Control System Toolbox.

⁴dlqe = Discrete-time linear quadratic estimator.

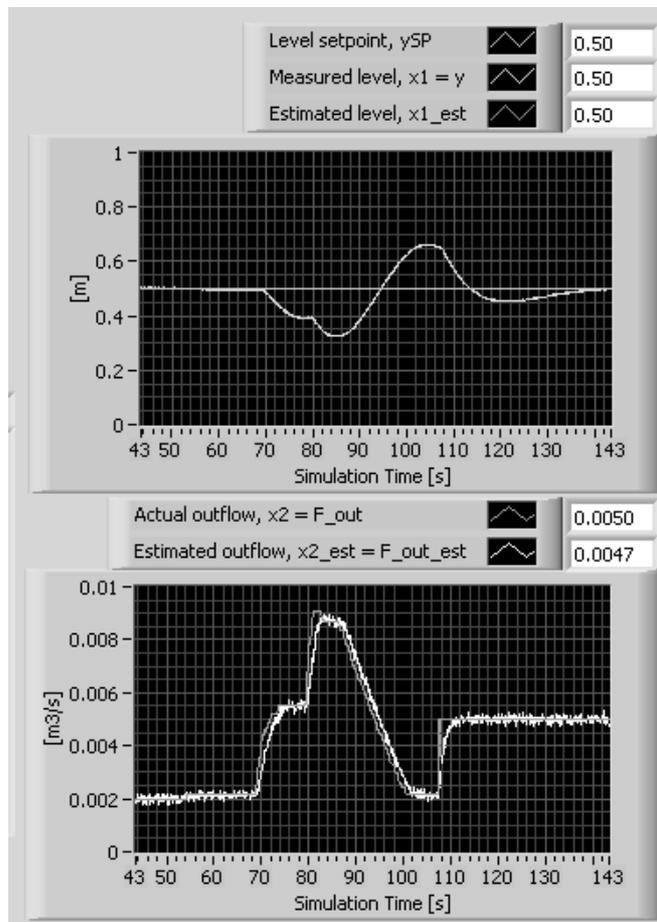


Figure 18.3: Example 18.2: Responses in the level and the real and estimated outflow

```

A_tank=0.1;
Kp=0.002;
A_cont=[0,-1/A_tank;0,0]
B_cont=[Kp/A_tank;0];
C_cont=[1,0];
D_cont=[0];
Ts=0.1;
%Generates a state-space model:
sys_cont=ss(A_cont,B_cont,C_cont,D_cont);
sys_disc=c2d(sys_cont,Ts); %Discretizing
A=sys_disc.a;
C=sys_disc.c;
G=[1,0;0,1]

```

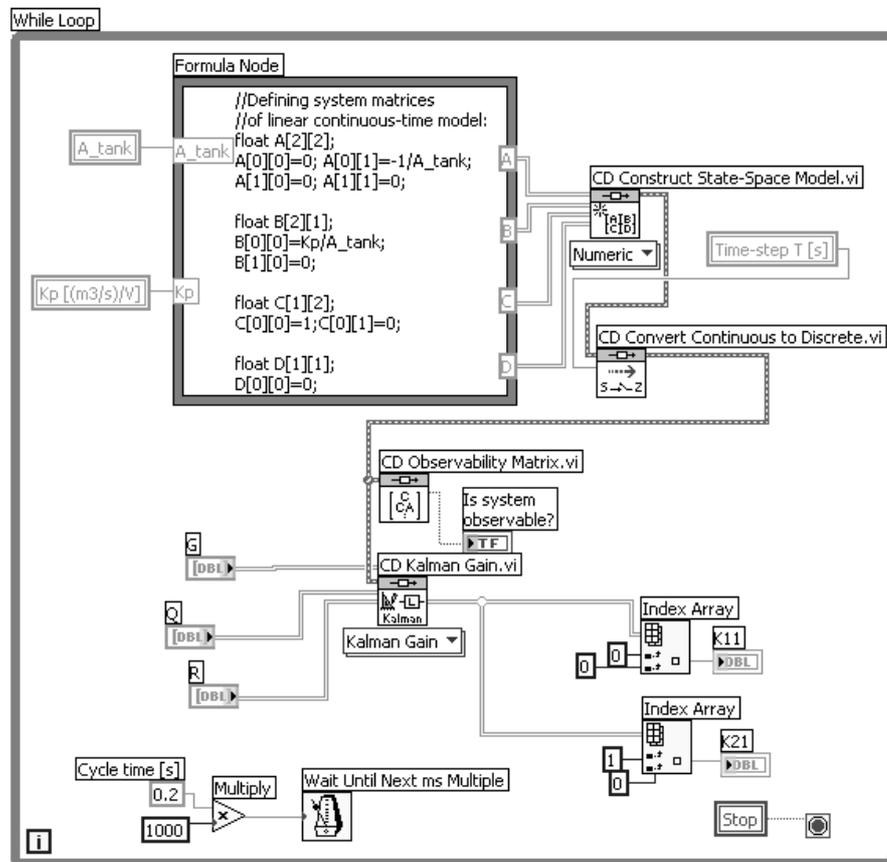


Figure 18.4: Example 18.2: Calculation of the steady state Kalman Filter gain with the Kalman Gain function, and checking for observability with the Observability Matrix function.

$$\begin{aligned}
 C &= [1, 0] \\
 Q &= [0.01, 0; 0, 1e-4] \\
 R &= [1e-6]; \\
 [Ks, Pp, Pc, E] &= dlqe(A, G, C, Q, R)
 \end{aligned}$$

Ks is the steady state Kalman Filter gain. (Pp and Pc are steady state estimation error auto-covariances, cf. (18.48) and (18.47). E is a vector containing the eigenvalues of the Kalman Filter, cf. (18.54).)

MATLAB answers

$$\begin{aligned}
 Ks &= \\
 &0.99991 \\
 &-0.09512
 \end{aligned}$$

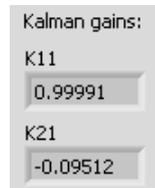


Figure 18.5: Example 18.2: The Kalman gains

which are the same values as calculated by the LabVIEW function **Kalman Gain**, cf. Figure 18.6.

[End of Example 18.2]

18.6 Using the Kalman Filter estimates in controllers

In the introduction to this chapter are listed several control functions which basically assumes that measurements of states and/or disturbances (loads) are available. If measurements from “hard-sensors” for some reason are not available, you can try using an estimate provided by a soft-sensor as Kalman Filter – or an observer – in stead. Figure 17.10 shows the structure of the system, with observer.

I will not show any example here. In stead I refer to the Example 17.5 where feedforward from estimated outflow (disturbance) in a level control system was implemented. The outflow was estimated with an observer, but the control structure will be exactly the same if a Kalman Filter was used in stead of an observer. The principle is the same as in Example 17.5 – and the simulated responses show the same improvement – so I refer to that example.

18.7 Using the Kalman Filter for increased robustness of feedback control at sensor failure

The Kalman Filter can be used to increase the robustness of a feedback control system against sensor failure. This is explained – and a concrete example including simulated responses is given – in Section 17.8 about

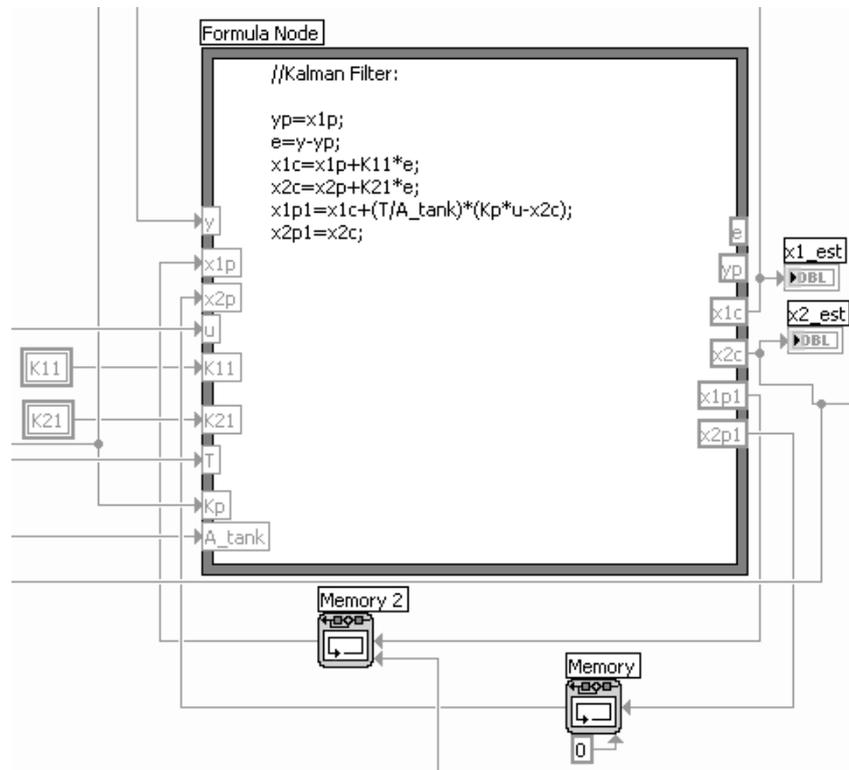


Figure 18.6: Example 18.2: Implementation of the Kalman Filter equations in a Formula Node. (The Kalman Filter gain is fetched from the While loop in the Block diagram, see Figure 18.4, using local variables.)

using observers for this purpose. Kalman Filters and observers are used in the same way, so I refer to that section for detailed information about this topic.

Part V

MODEL-BASED CONTROL

Chapter 19

Testing robustness of model-based control systems with simulators

This part of the compendium describes several *model-based* controllers. A model-based controller contains the mathematical model of the process to be controlled – either explicitly (as in Feedback Linearization, cf. Chapter 20) or implicitly (as in LQ Optimal Control, cf. Chapter 21). Of course, a process model can never give a perfect description of a physical process. Hence, there are *model errors*. The model errors are in the form of

- erroneous model structure, and/or
- erroneous parameter values.

So, the controller is based on more or less erroneous information about the process. If the model errors are large, the real control system may behave quite different from what is specified during the design. The control system may even become unstable.

A control system that is supposed to work in real life must be sufficiently *robust*. How can you check if the system is robust (before implementation)? You can *simulate the control system*. In the simulation you include reasonable model errors. How do you include model errors in a simulator? By using *different* models in the control function and in the process in the simulator. This is illustrated in Figure 19.1. You may use the initial model, M_0 , in the control function, while you use a changed

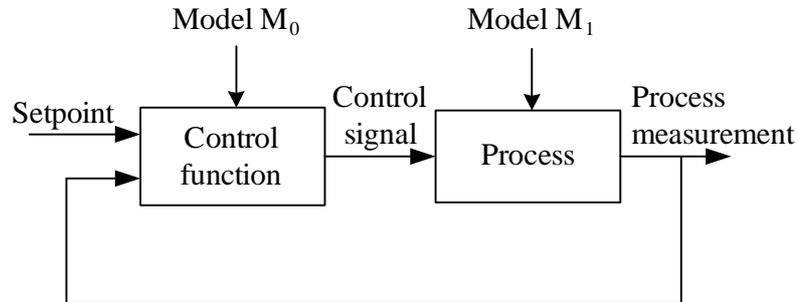


Figure 19.1: Testing the control system with model errors. Models M_0 and M_1 are made different by purpose.

model, M_1 , for the process. You must thoroughly plan which model errors (changes) that you will make, and whether the changes are *additive* or *multiplicative*. A parameter, say K , is changed additively if it is changed as

$$K_1 = K_0 + \Delta K \quad (19.1)$$

A multiplicative change is implemented with

$$K_1 = FK_0 \quad (19.2)$$

where the factor F may be set to e.g. 1.2 (a 20% increase) or 0.8 (a 20% decrease).

Even if the process model is accurate, the behaviour of the controller can be largely influenced by measurement noise. Therefore, you get a more real picture if you also include measurement noise in the simulator. Typically the measurement noise is a random signal¹.

Finally, I will mention that there are control system design methods which ensures robustness of the control system. In the design phase you specify assumed maximum model errors, together with performance specifications. The control function is typically a non-standard controller, which have to be simplified before implementation. It is however beyond the scope of this compendium to describe these design methods. More information can be found in e.g. [13].

¹Simulation tools as LabVIEW and Simulink contains signal generators for random signals.

Chapter 20

Feedback linearization

20.1 Introduction

Feedback linearization is a multivariable control method¹ that is based on a mathematical model of the process to be controlled. Hence it is a *model-based* control method. The model is a state-space model. It is assumed that the values of all of the state variables and process disturbances are available at any instance of time – either from measurements or from an estimator (an observer or a Kalman Filter). Hence, this control method demands much information about the process which may not be easy to get in practical applications. However, when this information is available the control system may give faster control than with PID controllers or other linear control functions. Since the control function is model-based, the performance may be ensured over a large operating range.

The control function consists of two parts:

- *A decoupler and linearizer* which is based on the process model and the instantaneous values of the states and the disturbances.
- *A multiloop PID controller* which is designed for the decoupled linear process.

¹The method can be applied to monovariable processes, too.

20.2 Deriving the control function

The following two sections cover these cases, respectively:

- *All* the state variables are controlled
- *Not all* the state variables are controlled

20.2.1 Case 1: All state variables are controlled

It is assumed that the process model is a state space model on the following form:

$$\dot{x} = f(x, v) + B(x, v) \cdot u \quad (20.1)$$

or, simpler,

$$\dot{x} = f + Bu \quad (20.2)$$

x is the state vector, v is the disturbance vector, and u is the control vector. f is a vector of scalar functions, and B is a matrix of scalar functions.

Note that the control vector u is assumed to appear *linearly* in the model.

Assume that the output vector is

$$y = x \quad (20.3)$$

By taking the derivative of (20.3) and using (20.2) we obtain the following differential equation describing the process output vector:

$$\dot{y} = f + Bu \quad (20.4)$$

Assume that r_y is the reference (or setpoint) of y .

With the above assumptions, we derive the control function as follows: We start by defining the *transformed control vector* as

$$z \stackrel{\text{def}}{=} f + Bu \quad (20.5)$$

Then (20.4) can be written as

$$\dot{y} = z \quad (20.6)$$

which are n decoupled or independent *integrators* (n is the number of state variables), because $y(t) = \int_0^t z d\tau$. The transfer function from z to y is

$$\frac{y(s)}{z(s)} = \frac{1}{s} \quad (20.7)$$

We can denote (20.6) as the *transformed process*.

We will now derive the control function for this integrator process, and thereafter derive the final control function. How can you control an integrator? With feedback and feedforward! A good choice for the *feedback controller* is a PI controller (proportional plus integral) because the controller should contain integral action to ensure zero steady-state control error in the presence of unmodelled disturbances (and there are such in a real system). The proportional action is necessary to get a stable control system (if a pure integral controller acts on an integration process the closed loop system becomes marginally stable, i.e. it is pure oscillatory). The multiloop feedback PI controller is

$$z_{fb} = K_p e + K_i \int_0^t e d\tau \quad (20.8)$$

where e is the control error:

$$e \stackrel{\text{def}}{=} r_y - y \quad (20.9)$$

In (20.8) K_p and K_i are diagonal matrices:

$$K_p = \begin{bmatrix} K_{p1} & 0 & \cdots & 0 \\ 0 & K_{p2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_{pn} \end{bmatrix} = \text{diag}(K_{p1}, K_{p2}, \cdots, K_{pn}) \quad (20.10)$$

$$K_i = \begin{bmatrix} K_{i1} & 0 & \cdots & 0 \\ 0 & K_{i2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & K_{in} \end{bmatrix} = \text{diag}(K_{i1}, K_{i2}, \cdots, K_{in}) \quad (20.11)$$

where the scalar values are

$$K_{ij} = \frac{K_{pj}}{T_{ij}} \quad (20.12)$$

where K_{pj} is the proportional gain and T_{ij} is the integral time of control loop no. j . K_{pj} and T_{ij} can be calculated in several ways. *Skogestad's method* is one option. Skogestad's method is reviewed in Appendix A. From Table A.1 (the second row) we get, since $\tau = 0$ and $K = 1$,

$$K_{pj} = \frac{1}{T_{Cj}} \quad (20.13)$$

and

$$T_{ij} = cT_{Cj} \quad (20.14)$$

where T_{C_j} is the specified time constant of feedback loop no. j , and c is a coefficient that can be set to e.g. 2, cf. Appendix A.

In addition to the PI feedback action the controller should contain *feedforward* from the reference r_y to get fast reference tracking when needed (assuming the reference is varying). The feedforward control function can be derived by substituting the process output y in the process model (20.6) by r_y and then solving for y , giving

$$z_{\text{ff}} = \dot{r}_{y_f} \quad (20.15)$$

where index f indicates lowpass filter which may be of first order. A pure time differentiation should not be implemented because of noise amplification by the differentiation. Therefore the reference should be lowpass filtered before its time derivative is calculated.

The control function for the process (20.6) based on the sum of the feedback control function and the feedforward control function is as follows:²

$$z = z_{\text{fb}} + z_{\text{ff}} \quad (20.16)$$

$$= \underbrace{K_p e + K_i \int_0^t e d\tau}_{z_{\text{fb}}} + \underbrace{\dot{r}_{y_f}}_{z_{\text{ff}}} \quad (20.17)$$

Now it is time to get the final control function, that is, the formula for the control vector u . From (20.5) we get

$$u = B^{-1}(z - f) \quad (20.18)$$

Here we use (20.17) to get the final control function:

$$\underline{u = B^{-1} \left(K_p e + K_i \int_0^t e d\tau + \dot{r}_{y_f} - f \right)} \quad (20.19)$$

If the reference is constant, as is the typical case in process control, the \dot{r}_{y_f} term has value zero, and it can therefore be left out in the control function.

Figure 20.1 shows a block diagram of the control system.

Here are some characteristics of the control system:

- The controller is *model based* since it contains f and B from the process model.

²It is the sum because the process (the integrator) has a linear mathematical model.

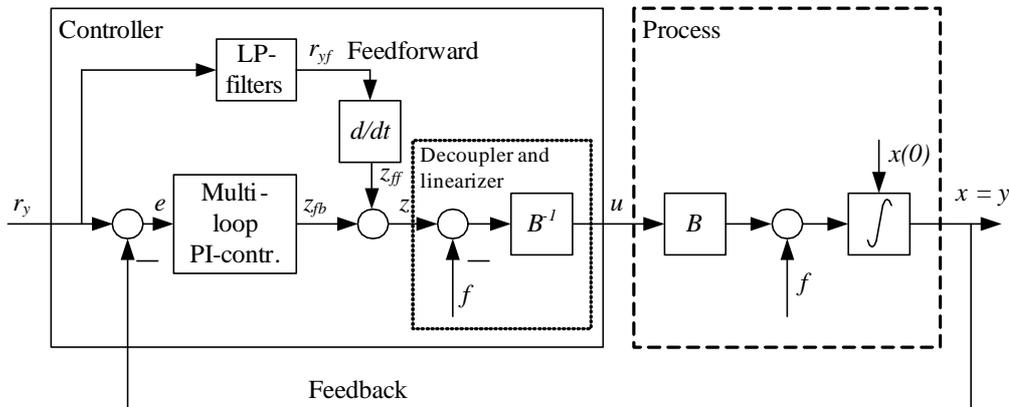


Figure 20.1: Block diagram of the control system based on feedback linearization

- Since the process disturbance is an argument of f and/or B the controller implements *feedforward* from the disturbance. (It also implements feedforward from the reference, due to the term $\dot{r}_y M$ in the controller.)
- The control system is *linear* even if the process is nonlinear.
- The control system consists of n *decoupled single-loop* control systems. This is illustrated in Figure 20.2.

Example 20.1 Feedback Linearization applied to level control

In this example Feedback Linearization will be applied to a level control system. Figure 20.3 shows the control system. It is assumed that the outflow is proportional to the control signal u and to the square root of the pressure drop along the control valve. The process model based on mass balance is (ρ is density)

$$\rho A \dot{h} = \rho q_{in} - \rho K_v u \sqrt{dP} \tag{20.20}$$

or

$$\dot{h} = \underbrace{\frac{q_{in}}{A}}_{=f} + \underbrace{\left(-\frac{K_v \sqrt{dP}}{A} \right)}_{=B} u \tag{20.21}$$

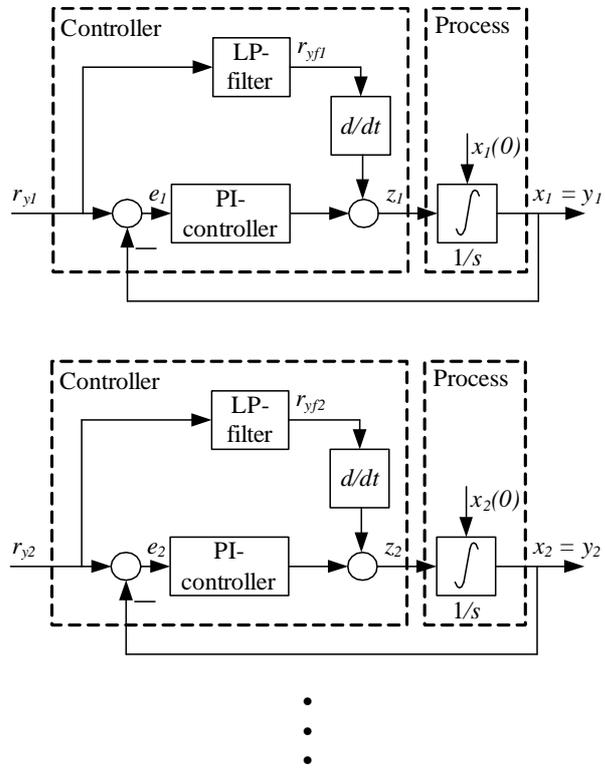


Figure 20.2: The control system consists of n decoupled single-loop control systems.

The control function becomes, cf. (20.19),

$$u = B^{-1} \left(K_p e + K_i \int_0^t e d\tau + \dot{r}_{y_f} - f \right) \tag{20.22}$$

$$= \left(-\frac{K_v \sqrt{dP}}{A} \right)^{-1} \left(K_p e + K_i \int_0^t e d\tau + \dot{r}_{y_f} - \frac{q_{in}}{A} \right) \tag{20.23}$$

$$= -\frac{A}{K_v \sqrt{dP}} \left(K_p e + K_i \int_0^t e d\tau + \dot{r}_{y_f} - \frac{q_{in}}{A} \right) \tag{20.24}$$

This control function requires that the differential pressure dP the inflow q_{in} are measured.

[End of Example 20.1]

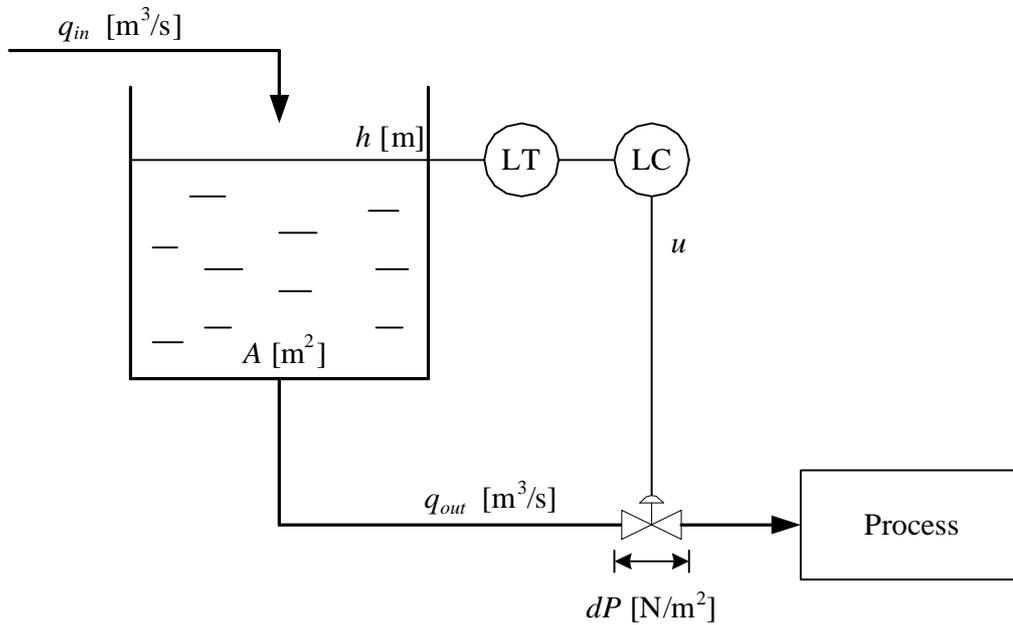


Figure 20.3: Example 20.1: Level control system

20.2.2 Case 2: Not all state variables are controlled

In Section 20.2.1 it is assumed that all the state variables are to be controlled, i.e. there is a reference for each of the state variables. Feedback linearization can be used also in cases where *not all* the state variables are controlled. The typical case is in positional control. Positions are only one set of the state variables. The other set is the velocities (rate of change of position). We will here focus on this typical case.

With position and velocity as state variables, the model of the process (e.g. motor or vessel) can be written on the following form where x is position, and u is the input (control variable), and y is the output:

$$\ddot{x} = f(x, \dot{x}, v) + B(x, \dot{x}, v) \cdot u \quad (20.25)$$

or simply

$$\ddot{x} = f + Bu \quad (20.26)$$

The output variable is the position:

$$y = x \quad (20.27)$$

By taking the second order time-derivative of (20.27) and using (20.26) we

obtain this differential equation describing the process output:

$$\ddot{y} = f + Bu \quad (20.28)$$

Assume that r_y is the reference of y (position). To derive the control function we define the *transformed control vector* as

$$z \stackrel{\text{def}}{=} f + Bu \quad (20.29)$$

(20.28) can then be written as

$$\ddot{y} = z \quad (20.30)$$

which are n decoupled (independent) *double integrators*. The transfer function from z to y is

$$\frac{y(s)}{z(s)} = \frac{1}{s^2} \quad (20.31)$$

These double integrators can be controlled with feedback with PID controllers plus feedforward:

$$z_{\text{fb}} = K_p e + K_i \int_0^t e d\tau + K_d \frac{de_f}{dt} \quad (20.32)$$

where e is the control error:

$$e \stackrel{\text{def}}{=} r_y - y \quad (20.33)$$

In (20.32) e_f is lowpass filtered control error (the derivative term should always contain a lowpass filter). K_p , K_i and K_d are diagonal matrices similar to (20.10). The scalar values on the diagonal of these matrices are

$$K_{p_j} \quad (20.34)$$

$$K_{i_j} = \frac{K_{p_j}}{T_{i_j}} \quad (20.35)$$

$$K_{d_j} = K_{p_j} T_{d_j} \quad (20.36)$$

K_{p_j} , T_{i_j} and T_{d_j} can be calculated with e.g. *Skogestad's method*, cf. Appendix A. According to Skogestad's formulas shown in Table A.1 (the bottom row with $\tau = 0$ and $K = 1$),

$$K_{p_j} = \frac{1}{4(T_{C_j})^2} \quad (20.37)$$

and

$$T_{i_j} = 4T_{C_j} \quad (20.38)$$

$$T_{d_j} = 4T_{C_j} \quad (20.39)$$

where T_{C_j} is the specified time constant of feedback loop no. j .

Note the following about using Skogestad's method for tuning double-integrators:

- Skogestad's formulas assumes a *serial* PID function. If your controller actually implements a *parallel* PID controller (as in the PID controllers in LabVIEW PID Control Toolkit and in the Matlab/Simulink PID controllers), you should transform from serial PID settings to parallel PID settings. If you do not implement these transformations, the control system may behave unnecessarily different from the specified response. The serial-to-parallel transformations are given by (A.8) – (A.10).
- For the double integrator I have seen in simulations that the actual response-time (63% rise-time) of the closed-loop system may be about twice the specified time-constant T_C . Consequently, you can set T_C to about half of the response-time you actually want to obtain.

In addition to the PID feedback action the controller should contain *feedforward* from the reference y_r to get fast reference tracking when needed (assuming the reference is varying). The feedforward control function can be derived by substituting the process output y in the process model (20.6) by y_r and then solving for y , giving

$$z_{\text{ff}} = \ddot{r}_{y_f} \quad (20.40)$$

where index f indicates lowpass filter, which should be of second order.

Now, we have the following control function for the process (20.30) consisting of the sum of the feedback control function and the feedforward control function:

$$z = z_{\text{fb}} + z_{\text{ff}} \quad (20.41)$$

$$= \underbrace{K_p e + K_i \int_0^t e d\tau + K_d \frac{de_f}{dt}}_{z_{\text{fb}}} + \underbrace{\ddot{r}_{y_f}}_{z_{\text{ff}}} \quad (20.42)$$

The final control function is found from (20.29):

$$u = B^{-1}(z - f) \quad (20.43)$$

Here we use and (20.42) to get

$$\underline{u = B^{-1} \left(K_p e + K_i \int_0^t e d\tau + K_d \frac{de_f}{dt} + \ddot{r}_{y_f} - f \right)} \quad (20.44)$$

If the reference is constant, as is the typical case in process control, the \ddot{r}_{y_f} term has value zero, and it can therefore be left out in the control function.

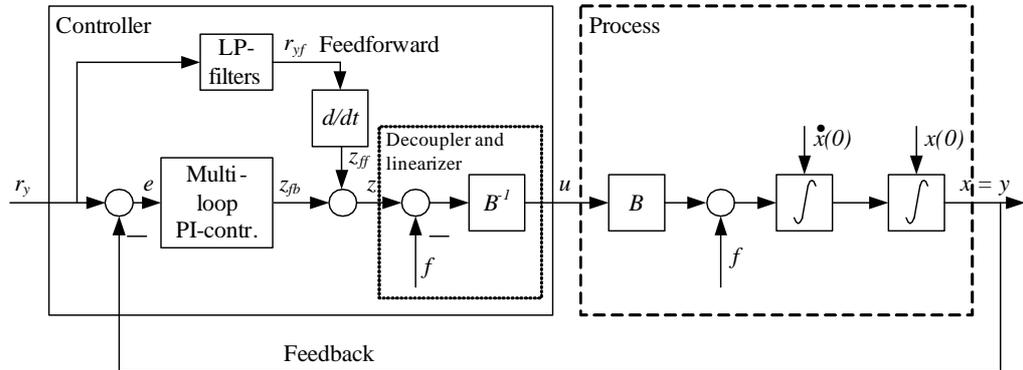


Figure 20.4: Block diagram of the control system based on feedback linearization

Figure 20.4 shows a block diagram of the control system.

The control system consists of n *decoupled single-loop* control systems. This is illustrated in Figure 20.5.

Example 20.2 *Feedback Linearization applied to motion control*

Given the following mathematical model of a body:

$$m\ddot{y} = F_c + F_d \quad (20.45)$$

where y [m] is position, $m = 10$ kg is mass, F_c [F] is force generated by the controller, and F_d [N] is net disturbance force (sum of damping, friction, gravitation, etc).

Assume that the position reference of y is r_y , and that the specified time-constant of the control system is 1 s. Also, assume that the PID controller is a parallel PID controller.

To derive the control function we first write the process model on the standard form (20.26):

$$\ddot{y} = \underbrace{\frac{1}{m} F_d}_f + \underbrace{\frac{1}{m} F_c}_B \underbrace{u}_u \quad (20.46)$$

The control function becomes

$$\underline{u} = B^{-1} \left(K_p e + K_i \int_0^t e d\tau + K_d \frac{de_f}{dt} + \ddot{r}_{y_f} - f \right) \quad (20.47)$$

$$= \underline{\underline{\left(\frac{1}{m} \right)^{-1} \left[K_p e + K_i \int_0^t e d\tau + K_d \frac{de_f}{dt} + \ddot{r}_{y_f} - \left(\frac{1}{m} F_d \right) \right]}} \quad (20.48)$$

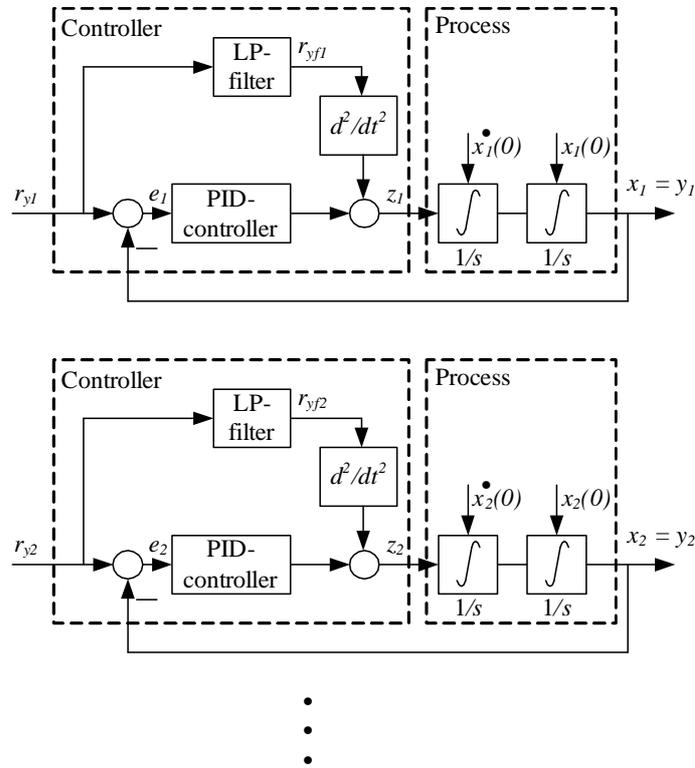


Figure 20.5: The control system consists of n decoupled single-loop control systems.

where e is the control error:

$$e = r_y - y \tag{20.49}$$

Now, we use Skogestad’s method to tune the PID controller. Since for a double integrator the actual response-time (63% rise-time) of the closed-loop system is about twice the specified time-constant T_C , we specify

$$T_C = 0.5 \text{ s} \tag{20.50}$$

The PID parameters of an assumed serial PID controller becomes

$$K_{p_s} = \frac{1}{4(T_C)^2} = \frac{1}{4(0.5)^2} = 1 \tag{20.51}$$

$$T_{i_s} = 4T_C = 4 \cdot 0.5 = 2 \text{ s} \tag{20.52}$$

$$T_{d_s} = 4T_C = 4 \cdot 0.5 = 2 \text{ s} \tag{20.53}$$

Finally, the PID parameters of an assumed parallel PID controller becomes, cf. (A.8) – (A.10),

$$\underline{K_{pp}} = K_{ps} \left(1 + \frac{T_{ds}}{T_{is}} \right) = 1 \cdot \left(1 + \frac{2 \text{ s}}{2 \text{ s}} \right) = \underline{2} \quad (20.54)$$

$$\underline{T_{ip}} = T_{is} \left(1 + \frac{T_{ds}}{T_{is}} \right) = 2 \text{ s} \cdot \left(1 + \frac{2 \text{ s}}{2 \text{ s}} \right) = \underline{4 \text{ s}} \quad (20.55)$$

$$\underline{T_{dp}} = T_{ds} \frac{1}{1 + \frac{T_{ds}}{T_{is}}} = 2 \text{ s} \cdot \frac{1}{1 + \frac{2 \text{ s}}{2 \text{ s}}} = \underline{1 \text{ s}} \quad (20.56)$$

[End of Example 20.2]

Chapter 21

LQ (Linear Quadratic) optimal control

21.1 Introduction

Optimal control of a process means that the control function is designed so that a given *optimization criterion* or *performance index* gets a minimal value. It is assumed that the process can be described by a *linear* model, and that the criterion contains a *quadratic* function of the state variables and the control variables.¹ This type of optimal control is therefore denoted *Linear Quadratic* control – or *LQ control*. The reference (setpoint) is assumed to be zero in the basic LQ control problem, and hence the term *LQ regulation* or LQR is also used. We will however consider a non-zero reference in this chapter, cf. Section 21.3.

A particular feature of the LQ controller is that it will stabilize any linear process! However, you do not have a guarantee that it will stabilize any nonlinear process, even if this process is linearizable. A simulation study should be made to check if the control system works well under varying conditions, including model errors.

LQ control can be applied to both monovariable and multivariable processes. It turns out that the control function is based on feedback from all the states of the process. If not all the states can be measured, an observer or a Kalman Filter can be used to estimate the states, and the controller then uses the estimated states as if they were measured. This

¹The main reason why a quadratic criterion is used is that the control function is relatively easy to derive and easy to implement :-)

principle is denoted the *certainty equivalence principle*. It turns out that the control function and the state estimator can be designed independently as long as the process is linear. The principle of separate design of the controller and the estimator is denoted the *separation principle*.

LQ controllers can be designed for continuous-time and for discrete-time systems, and for stochastic systems (systems excited by random disturbances) and deterministic systems (random noise is not taken into account in the controller design). I have chosen to describe LQ control for deterministic continuous-time systems. Of course, in a practical implementation you will (probably) need a discrete-time implementation of the controller, and this will be described in this chapter.

LQ control is quite similar to Model-based Predictive Control (MPC), which has become an important control function the last decades. Also MPC is based on a quadratic criterion. However, MPC takes into account limitations in the control variables and the state variables, hence making it somewhat more useful than LQ controller, but also much more computational demanding to implement. MPC is described in Chapter 22.

21.2 The basic LQ controller

In basic LQ control it is assumed that the process to be controlled is given by the following linear state-space model

$$\dot{x}(t) = Ax(t) + Bu(t) \quad (21.1)$$

The LQ controller brings the state x from any initial state $x(0)$ to zero in an optimal way. What is “optimal”? It is defined by the optimization criterion:

$$J = \int_{t=0}^{t=\infty} [x^T(t)Qx(t) + u^T(t)Ru(t) + 2x^T(t)Nu(t)] dt \quad (21.2)$$

It is very common that the weight matrix N is a zero matrix (of proper dimension), and in these cases the criterion is

$$J = \int_{t=0}^{t=\infty} [x^T(t)Qx(t) + u^T(t)Ru(t)] dt \quad (21.3)$$

N is assumed to be zero in the following.

Q and R are *weight matrices* of the states and the control signal, respectively. Q and R are selected by the user, and they are the tuning

parameters of the LQ controller. Q is a symmetric positive semidefinite matrix, and R is a symmetric positive definite matrix.

The criterion (21.2) gives you (the user) the possibility to punish large variations in the states (by selecting a large Q) or to punish large variations in the control variable u (by selecting a large R). It is fair to say that the LQ controller is a user-friendly controller because the tuning parameters (Q and R) are meaningful, at least in the qualitative sense.

As an example, assume that the system has two state variables, x_1 and x_2 , hence

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (21.4)$$

and one (scalar) control variable, u , and that the weight matrices are:

$$Q = \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \quad (21.5)$$

$$R = [R_{11}] \quad (21.6)$$

The criterion J becomes

$$J = \int_{t=0}^{t=\infty} [x^T Q x + u^T R u] dt \quad (21.7)$$

$$= \int_{t=0}^{t=\infty} \left\{ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} Q_{11} & 0 \\ 0 & Q_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + u R u \right\} dt \quad (21.8)$$

$$= \int_{t=0}^{t=\infty} \{ Q_{11} x_1^2 + Q_{22} x_2^2 + R_{11} u^2 \} dt \quad (21.9)$$

Thus, J is a sum of quadratic terms of the state variables and the control variable.

It can be shown that the control function that gives J a minimum value is as follows

Optimal (LQ) controller:

$$u(t) = -G(t)x(t) \quad (21.10)$$

In other words, *the control signal is based on feedback from a linear combination of the state variables*. The controller gain $G(t)$ (a matrix) is given by a so-called Riccati equation which will not be shown here. Figure 21.1 shows a block diagram of the control system.

In the above example the controller becomes

$$u(t) = -G(t)x(t) = - \begin{bmatrix} G_{11}(t) & G_{12}(t) \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \quad (21.11)$$

$$= - [G_{11}(t)x_1(t) + G_{12}(t)x_2(t)] \quad (21.12)$$

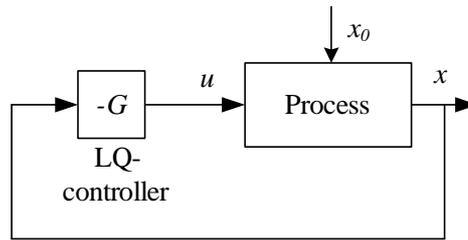


Figure 21.1: Control system with optimal LQ controller

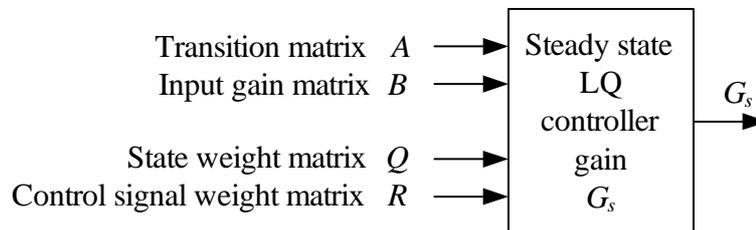
It is common to implement the *steady-state value* G_s of the controller gain:

Steady-state optimal (LQ) controller:

$$u(t) = -G_s x(t) \quad (21.13)$$

G_s can be calculated offline, and in advance (before the control system is started).

Figure 21.2 illustrates the information that the **Linear Quadratic Regulator** function needs to compute the steady-state LQ controller gain G_s .

Figure 21.2: Information needed to compute the steady-state LQ controller gain, G_s .

Matlab and LabVIEW have functions to calculate G_s :

- *Matlab*: The syntax of using the Matlab function **dlqr** (discrete linear quadratic regulator) is
 $[G_s, S, E] = \text{dlqr}(A, B, Q, R, N)$

where G_s is the steady-state controller gain, S is the solution of the Riccati-equation mentioned above (not shown here), and E is the vector containing the eigenvalues of the control system (see comment below). If N is actually a zero matrix, which is typically the case, it can be omitted from the argument list of **dlqr**.

- *MathScript* (in LabVIEW): The name and the syntax of the MathScript function is the same as for **dlqr** in Matlab.
- *LabVIEW Control Design Toolkit palette*: The function **CD Linear Quadratic Regulator** calculates the steady-state controller gain. To construct the state-space model you can use **CD Construct State-Space Model**. This is shown in Figure 21.3.

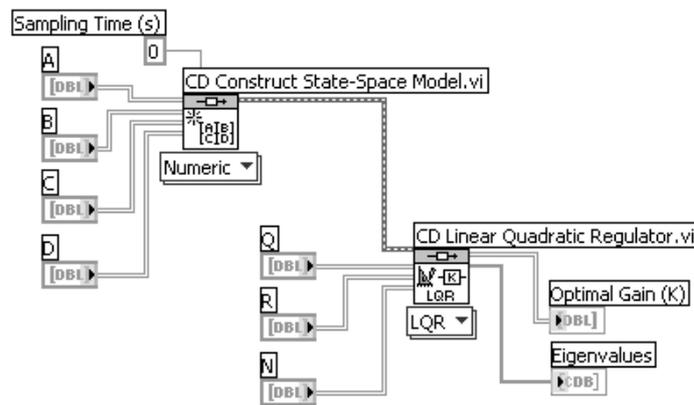


Figure 21.3: On the Control Design Toolkit palette in LabVIEW you can use the function **CD Linear Quadratic Regulator**. (If N is zero, you can keep this input unwired.) The state-space model can be constructed using **CD Construct State-Space Model**.

Here are comments about the LQ controller:

- **The reference is zero, and there are no process disturbances:** These assumptions seem somewhat unrealistic, because in real control systems the reference is typically non-zero, and the disturbances are non-zero. Therefore, for the controller to represent realistic control problems, the variables in (21.1) should actually be regarded as deviation variables about some operating point. Then, how do you bring the states to the operating point, so that the mean value control error is zero? By enforcing *integrators* into the controller. This is described in detail in Section 21.3.
- **Controllability:** The process to be controlled has to be *controllable*. If it is not controllable, there exists no finite steady-state value of the gain G . Controllability means that there exists a control signal $u(t)$ so that any state can be reached from any initial state in finite time.

It can be shown that a system is controllable if the rank of the controllability matrix

$$M_{\text{control}} = \begin{bmatrix} B : AB : A^2B : \dots : A^{n-1}B \end{bmatrix} \quad (21.14)$$

is n (the order of A , which is the number of state variables).

- **Non-measured states:** If not all the states are measured, you can use estimated states in stead of measured states in the controller:

$$u(t) = -G_s x_{\text{est}}(t) \quad (21.15)$$

where x_{est} is the estimated state vector from a state estimator (observer or Kalman Filter). Figure 21.4 shows a block diagram of the control system with state estimator.

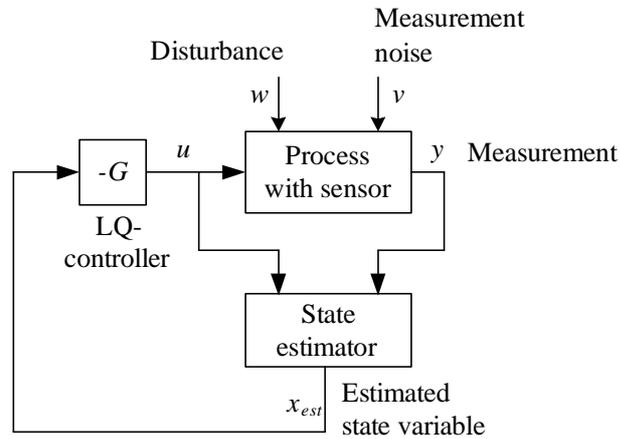


Figure 21.4: Control system with state estimator

- **The eigenvalues of the control system:** Assume that the controller is (21.15). By combining this controller with the process model (21.1) we get the following model of the *control system* (the closed-loop system):

$$\dot{x} = Ax + B(-G_s x) \quad (21.16)$$

$$= (A - BG_s)x \quad (21.17)$$

The eigenvalues $\{s_1, s_2, \dots, s_n\}$ of the control system are the eigenvalues of the transition matrix $(A - BG_s)$ in (21.17):

$$0 = \det[sI - (A - BG_s)] \quad (21.18)$$

$$= (s - s_1)(s - s_2) \cdots (s - s_n) \quad (21.19)$$

$$= s^n + a_{n-1}s^{n-1} + \cdots + a_1s + a_0 \quad (21.20)$$

- **Stability of the control system:** It can be shown that a LQ control system is asymptotically stable.² In other words, the eigenvalues of the control system are in the left half of the complex plane.
- **Tuning of the LQ controller:** From the criterion (21.2) we can conclude that *a larger value of the weight of one particular state variable* causes the time response of that state variable to become smaller, and hence *the control error (deviation from zero) is smaller*. But what should be the initial values of the weight matrices, before they are tuned? One possibility is

$$Q = \text{diag} \left\{ \frac{1}{|x_{i_{\max}}|^2} \right\} \quad (21.21)$$

where $x_{i_{\max}}$ is the assumed maximum value of state variable x_i , and

$$R = \text{diag} \left\{ \frac{1}{|u_{j_{\max}}|^2} \right\} \quad (21.22)$$

where $u_{j_{\max}}$ is the assumed maximum value of control variable u_j .

- **Pole placement design of the control system:** Above it was stated that the eigenvalues of the control system are the roots of the characteristic equation

$$0 = \det [sI - (A - BG)] \quad (21.23)$$

For most systems, the poles, $\{s_i\}$, of the system are the same as the eigenvalues. In the following the term poles are used in stead of eigenvalues. In pole placement design of control systems the poles are specified by the user, and, assuming the controller has the linear feedback structure as given by (21.13), it is usually possible to solve (21.23) for the controller gain matrix G . This is exactly the same problem as with observer design, cf. Chapter 17, but we do not address pole placement design of controllers any further here.

Example 21.1 *LQ control of pendulum*

This example is about stabilization of a pendulum on a cart using LQ optimal control. A reference for the system described here is the text-book *Nonlinear Systems* by H. K. Khalil (Pearson Education, 2000). The

²Not a big surprise, since the controller minimizes the criterion J .

original reference is Linear Optimal Control Systems by H. Kwakernaak and R. Sivan (Wiley, 1972).

Mathematical model

Figure 21.5 shows the cart with the pendulum.

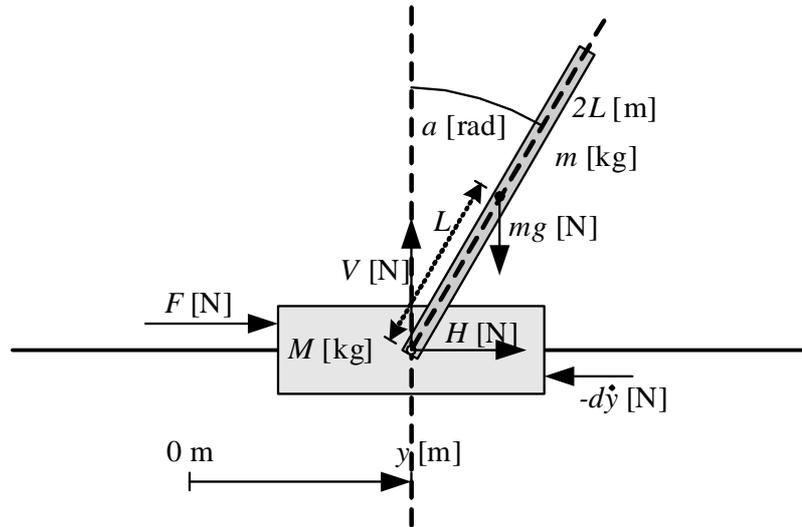


Figure 21.5: Cart with pendulum

A motor (in the cart) acts on the cart with a force F . This force is manipulated by the controller to stabilize the pendulum in an upright position or in a downright position at a specified position of the cart.

A mathematical model of the system is derived below. This model is used to design a stabilizing controller, namely an optimal controller. The mathematical model is based on the following principles:

1. Force balance (Newton's Second Law) applied to the horizontal movement of the center of gravity of the pendulum:

$$m \frac{d^2}{dt^2} (y + L \sin a) = H \quad (21.24)$$

The differentiation must be carried out, but the result of it not shown here.

2. Force balance applied to the vertical movement of the center of

gravity of the pendulum:

$$m \frac{d^2}{dt^2} (L \cos a) = V - mg \quad (21.25)$$

The differentiation must be carried out, but the result of it not shown here.

3. Torque balance (the rotational version of the Newton's Second Law applied to the center of gravity of the pendulum:

$$I\ddot{a} = VL \sin a - HL \cos a \quad (21.26)$$

4. Force balance applied to the cart:

$$M\ddot{y} = F - H - d\dot{y} \quad (21.27)$$

In the above equations,

- I is the moment of inertia of the pendulum about its center of gravity. For the pendulum shown in Figure 1,

$$I = \frac{mL^2}{12} \quad (21.28)$$

- V and H are vertical and horizontal forces, respectively, in the pivot.
- d is a damping coefficient.

From Eq. (21.24) – (21.27), the internal forces V and H can be eliminated, resulting in two differential equations (not containing V and H), which are not shown here. These two differential equations can be written as the following non-linear state-space model:

$$\dot{x}_1 = x_2 \quad (21.29)$$

$$\dot{x}_2 = \frac{-m^2 L^2 g \cos x_3 \sin x_3 + (u + mL x_4^2 \sin x_3 - dx_2) (I + mL^2)}{D_1} \quad (21.30)$$

$$\dot{x}_3 = x_4 \quad (21.31)$$

$$\dot{x}_4 = \frac{(m + M) (mgL \sin x_3) - (u + mL x_4^2 \sin x_3 - dx_2) mL \cos x_3}{D_1} \quad (21.32)$$

where

$$D_1 = (I + mL^2) (m + M) - m^2 L^2 \cos^2 x_3 \quad (21.33)$$

In the above model,

- $x_1 = y$ (cart horizontal position)
- $x_2 = \dot{y}$ (cart horizontal speed)
- $x_3 = a$ (pendulum angular position)
- $x_4 = \dot{a}$ (pendulum angular speed)

Controller

To stabilize the pendulum either vertically up or vertically down, a specified possibly non-zero position, r , of the cart, a steady-state LQ (Linear Quadratic Regulator) controller is used. The feedback control function is as follows:

$$u = -G_{11} [x_1 - r] - G_{12}x_2 - G_{13}x_3 - G_{14}x_4 \quad (21.34)$$

The controller output, u , is applied as the force F acting on the cart. Hence the force is calculated as a linear combination of the states of the system. The states are assumed to be available via measurements. (Thus, there is a feedback from the measured states to the process via the controller.)

The controller gains, G_{11} , G_{12} , G_{13} , G_{14} , are calculated by the LabVIEW MathScript function **lqr** (Linear Quadratic Regulator) which has the following syntax:

$$[G, X, E] = \text{lqr}(A, B, Q, R);$$

where G is the calculated gain. (X is the steady-state solution of the Riccati equation, and E is the eigenvalue vector of the closed loop system.) A and B are the matrices of the linear state-space process model corresponding to the nonlinear model (21.29) – (21.32). A and B are presented below. Q is the state variable weight matrix and R is the control variable weight matrix used in the LQ(R) optimization criterion given by (21.3).

Q has the following form:

$$Q = \begin{bmatrix} Q_{11} & 0 & 0 & 0 \\ 0 & Q_{22} & 0 & 0 \\ 0 & 0 & Q_{33} & 0 \\ 0 & 0 & 0 & Q_{44} \end{bmatrix} \quad (21.35)$$

while Q_{ii} can be used as controller tuning parameters. R is

$$R = [R_{11}] \quad (21.36)$$

and is also used as tuning parameter.

The control system can be used to stabilize the cart and the pendulum in two different operation points, namely vertically up and vertically down. The process model is nonlinear. Therefore, the linear matrices A and B are derived for each of these operating points. The linearization of (21.29) – (21.32) can be carried out using the following assumptions:

- When the angular position $x_3 = a$ is close to zero, $\cos x_3$ is approximately equal to 1.
- When the angular position $x_3 = a$ is close to $180^\circ = \pi$ rad, $\cos x_3$ is approximately equal to -1 .
- When the angular position $x_3 = a$ is close to zero or close to $180^\circ = \pi$ rad, $\sin x_3$ is very similar to x_3 , and hence is substituted by x_3 .
- The angular speed $x_4 = \dot{a}$ is small, approximately zero in both operating points.

Using these assumptions, it can be shown that the linearized model becomes

$$\begin{bmatrix} \Delta \dot{x}_1 \\ \Delta \dot{x}_2 \\ \Delta \dot{x}_3 \\ \Delta \dot{x}_4 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{(I+mL^2)d}{D_2} & -s\frac{m^2L^2g}{D_2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & s\frac{mLd}{D_2} & \frac{(m+M)mgL}{D_2} & 0 \end{bmatrix}}_A \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{I+mL^2}{D_2} \\ 0 \\ \frac{mL}{D_2} \end{bmatrix}}_B [\Delta u] \quad (21.37)$$

where

$$D_2 = (I + mL^2)(m + M) \quad (21.38)$$

and

$$S = \begin{cases} 1 & \text{for pendulum in vertical up position} \\ -1 & \text{for pendulum in vertical down position} \end{cases} \quad (21.39)$$

The total control signal is

$$u(t) = u_{\text{op}} + \Delta u(t) \quad (21.40)$$

where u_{op} is the control signal needed to keep the system at the operating point, nominally. However, $u_{\text{op}} = 0$ in both operating points.

Because the parameters of the linear model (parameters of A , only) are different in these two operating points, the controller gains will also be different.

Model for simulator and model for controller design

The model (21.24) – (21.27) is the basis of both the simulator (representing the real process) and the controller. However, to make it possible to check if the control system is robust, two sets of model parameters are available in the front panel of the simulator:

- Model parameters $M_{\text{real}}, m_{\text{real}}, L_{\text{real}}, d_{\text{real}}$ used in the simulator.
- Model parameters $M_{\text{model}}, m_{\text{model}}, L_{\text{model}}, d_{\text{model}}$ used in the design of the controller.

By default, these two parameter sets have equal numerical values, but if you want to check for robustness of the controller against variations or inaccurate knowledge about one certain parameter, you must set the values of the corresponding parameters to different values, e.g. set d_{model} to a different value from d_{real} .

Figure 21.6 shows simulated responses for the control system. The positional reference was changed. The cart converges to the reference value, and the pendulum is stabilized upright despite the changes of the the cart position.

[End of Example 21.1]

21.3 LQ controller with integral action

21.3.1 Introduction

The basic LQ controller described in Section 21.2 is based on assumptions that in many applications are unrealistic or idealized. The following is more realistic:

- The reference is non-zero.
- There is a reference for a number of the states variables (not for the whole state vector).
- The process disturbance has non-zero mean value.

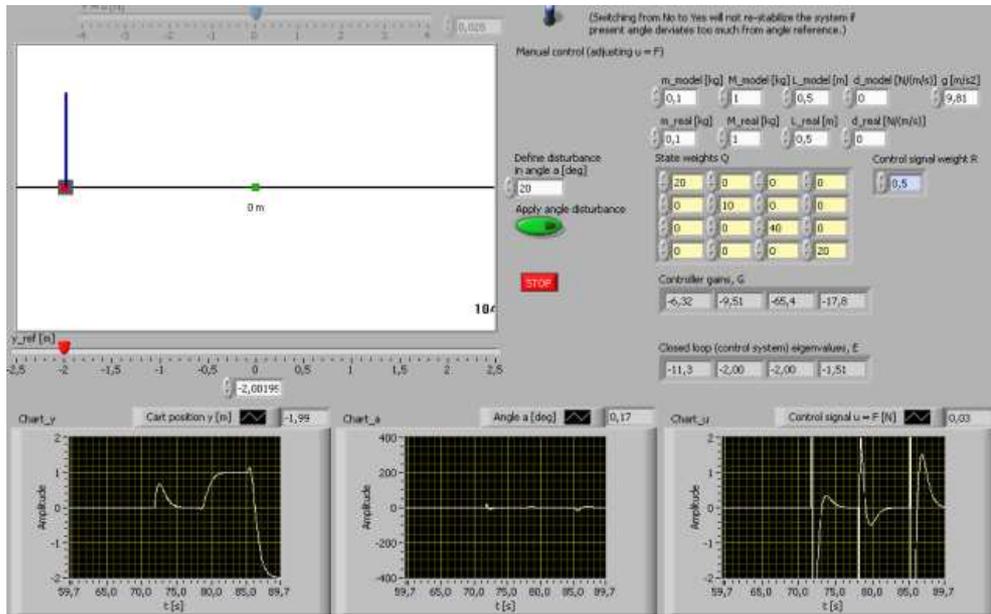


Figure 21.6: Simulated responses for the control system of the pendulum on the cart

The controller given by (21.13) does not ensure zero steady-state control error under these assumptions. What is needed is integral action in the controller!

21.3.2 Including integrators in the controller

Figure 21.7 shows a block diagram of the control system with integrators included in the controller. The integrator block actually represents a number of single integrators, as many as there are reference variables.

Example 21.2 *LQ controller with integrator*

Figure 21.8 shows the detailed structure of a process with two state variables being controlled by a LQ controller with integrator.

[End of Example 21.2]

The output of the integrators are regarded as augmentative state variables, x_{int} . These state variables are given by the following differential

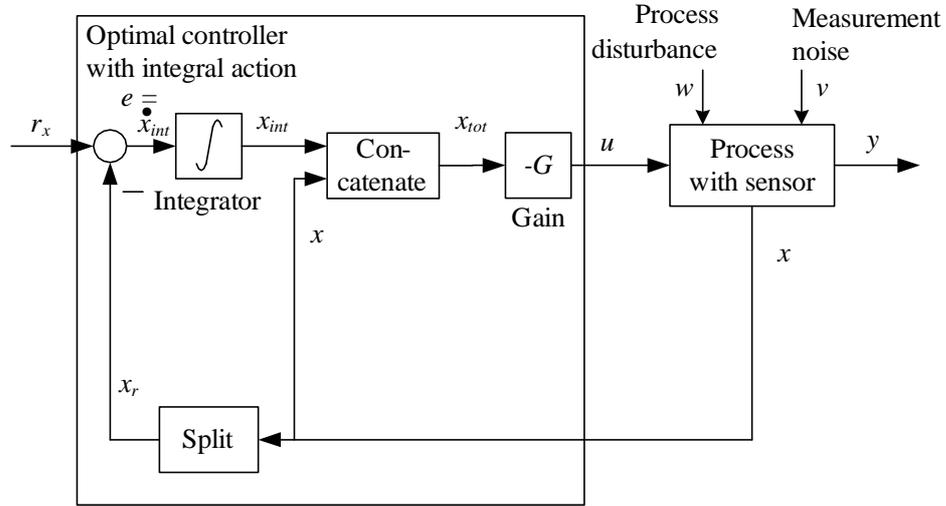


Figure 21.7: Optimal control system with integrators in the controller. e is the control error.

equation(s):

$$\dot{x}_{int} = r_x - x_r \quad (21.41)$$

which corresponds to this integral equation:

$$x_{int}(t) = \int_0^t (r_x - x_r) d\tau \quad (21.42)$$

Here, r_x is the reference vector for the state vector x_r which consists of those state variables among the process state vector x that are to track a reference. (In the above equations it is assumed that x_r is directly available from measurements. If x_r is taken from a state estimator, $x_{r,est}$ is used in stead of x_r , of course.)

The total state vector that is used to design the LQ controller is the state vector consisting of the original process state vector augmented with the integrator state vector:

$$x_{tot} = \begin{bmatrix} x \\ \cdots \\ x_{int} \end{bmatrix} \quad (21.43)$$

The control variable u is given by the control function

$$u = -Gx_{tot} = -G \begin{bmatrix} x \\ \cdots \\ x_{int} \end{bmatrix} \quad (21.44)$$

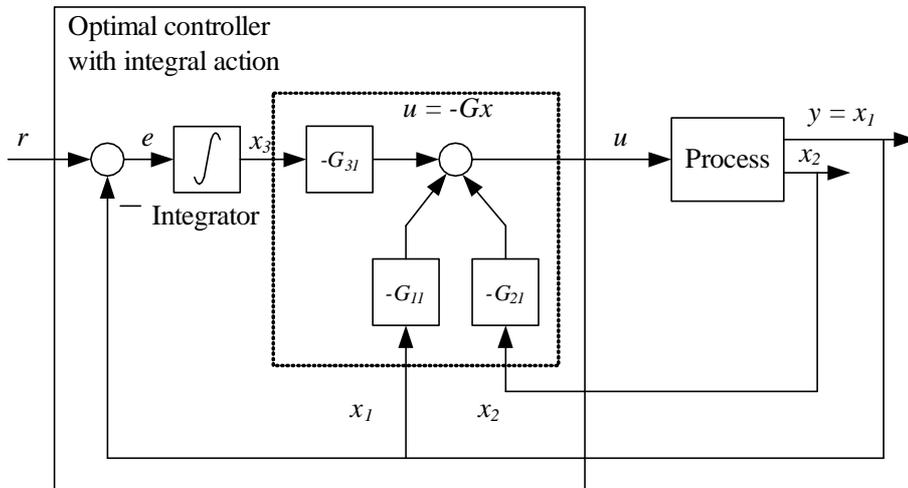


Figure 21.8: Example 21.2: A process with two state variables being controlled by a LQ controller with integrator

Note: When writing up the state-space model that is used for designing the LQ controller, you can disregard the reference r_x , i.e. you set it to zero because it is not taken into account when calculating the controller gain G . But of course it must be included in the implemented controller which is given by (21.44), with x_{int} given by (21.46).

21.3.3 Discrete-time implementation of the LQ controller

In a computer-based implementation of a LQ controller you will probably need to discretize the continuous-time integrator (21.41). This can be done using Backward or Forward discretization. The Backward method is the best with respect to numerical accuracy, and it can be applied to (21.41) without any problems because it is a linear differential equation. Applying Backward discretization on (21.41) gives

$$\dot{x}_{int}(t_k) \approx \frac{x_{int}(t_k) - x_{int}(t_{k-1})}{T_s} = r_x(t_k) - x_r(t_k) \quad (21.45)$$

Solving for $x_{int}(t_k)$ gives the final integrator algorithm ready for being programmed:

$$x_{int}(t_k) = x_{int}(t_{k-1}) + T_s [r_x(t_k) - x_r(t_k)] \quad (21.46)$$

A practical issue of any controller having integral action is *anti windup*, which is a feature to prevent the integrator to “wind up” – or increasing its

output continually – while the total control signal is at its saturation limit, either the maximum or the minimum limit. If anti windup is not implemented, the control error may become unnecessarily large for an unnecessarily long time. Anti windup can be implemented in a control program loop as follows:

```
x_int_k=x_int_k1+Ts*(rx-xr);
...//Calculate x_tot.
u_draft=-G*x_tot;

if (u_draft > u_max)
{
x_int_k=x_int_k1;//Integral not changed
...//Calculate x_tot.
u=-G*x_tot;
}

elseif (u_draft < u_min)
{
x_int_k=x_int_k1;//Integral not changed
...//Calculate x_tot.
u=-G*x_tot;
}

else
{
u=u_draft;
}

DAC(u);
```

Above, DAC(u) means send u to the digital-analog converter.

Chapter 22

Model-based predictive control (MPC)

Model-based predictive control or *MPC* has become an important control method, and it can be regarded as the next most important control method in the industry, next to basic PID control and PID control based methods. Commercial MPC products are available as separate products, or as modules included in automation products. Matlab has support for MPC with the MPC Toolbox. LabVIEW has support for MPC with the Control Design and Simulation Toolkit.

MPC has found many applications, for example

- Dynamic positioning (positional control) of sea vessels
- Paper machine control
- Batch process control
- Control of oil/water/gas separator trains

MPC can be applied to multivariable processes¹ and – depending on the MPC implementation – non-linear processes. The controller function is based on a continuous calculation of the optimal or “best” future sequence or time-series of the control variable, u . This calculation is based on predicting the future behaviour of the process to be controlled. Of course, a *mathematical process model* is used to predict this future behaviour.

¹which are processes with more than one control variable and process output variables.

The *optimization criterion* which is *minimized* by an optimization algorithm in the MPC tool has typically the following quadratic form²:

$$\text{Optimization criterion, } J \quad (22.1)$$

$$= \sum_{k=0}^{N_p} \left\{ Q_1 [e_1(t_k)]^2 + Q_2 [e_2(t_k)]^2 + \dots + Q_n [e_n(t_k)]^2 \right\} \quad (22.2)$$

$$+ \sum_{k=1}^{N_c} R_1 \left\{ [\Delta u_1(t_k)]^2 + R_2 [\Delta u_2(t_k)]^2 + \dots + R_r [\Delta u_r(t_k)]^2 \right\} \quad (22.3)$$

$$= \text{Sum of weighed future squared control errors} \\ + \text{Sum of weighed future squared increments of control variables}$$

Here:

- k is discrete time index.
- i is the time index along the prediction horizon.
- e_i is the difference between the reference (setpoint) r_i and the predicted process variable \hat{y}_i :

$$e_i = r_i - \hat{y}_i \quad (22.4)$$

- $\Delta u_j(t_k)$ is the predicted change in control action at time k .
- N_p (a scalar) is the prediction horizon, i.e. the number of samples in the future during which the MPC controller predicts the plant output.
- N_c (a scalar) is the control horizon, i.e. the number of samples within the prediction horizon during which the MPC controller can affect the control action.
- Q_i is the weight (or cost) of control error no. i . *This is typically a tuning parameter.* The larger Q_i , the smaller the control error e_i – typically causing the control signal to vary more.
- R_j is the weight (or cost) of the increment or change of control variable no. j . *This is a tuning parameter.* The larger R_j , the smaller is the variations of the control variable – typically causing the control error to vary more.

²The main reason why the criterion contains *quadratic (squared)* terms is that it makes solving the optimization problem easier, mathematically.

The kind of mathematical process model that the MPC algorithm uses to predict the future behaviour of the process varies between the different MPC implementation. The alternatives are:

- Impulse response model (which can be derived from simple experiments on the process)
- Step response model (same comment as above)
- Transfer function model (which is a linear dynamic model which may be derived from experiments on the process)
- Linear state-space model
- Non-linear state-space model (which is the most general model form since it may include nonlinearities and it may be valid over a wide operating range)

In addition to the process model the optimization takes into account *constraints*, i.e. specified maximum and minimum values, of y and Δu , and u (the total control signal):

$$y_{\min} \leq y \leq y_{\max} \quad (22.5)$$

$$\Delta u_{\min} \leq \Delta u \leq \Delta u_{\max} \quad (22.6)$$

$$u_{\min} \leq u \leq u_{\max} \quad (22.7)$$

The ability to take into account such constraints in a well-defined way is a strong feature of MPC.

Figure 22.1 illustrates how predictive control works. The MPC controller predicts the plant output for time $k + N_p$. At the next sample time, $k + 1$, the prediction and control horizons move forward in time, and the MPC controller predicts the plant output again. Figure 22.2 shows how the prediction horizon moves at each sample time k . The control horizon moves forward along with the prediction horizon. This is called the *receding horizon-principle*. Before moving forward, the controller sends the control action $u(t_k)$ to the plant. If there is a change of the reference or the disturbance within the prediction horizon and the controller knows about this change, the MPC will adjust the control variable *in advance*.

The MPC controller needs information about the present state of the process and disturbances (environmental variables) acting on the process when using the model to predict the future responses. These states and disturbances must be either measured or estimated.

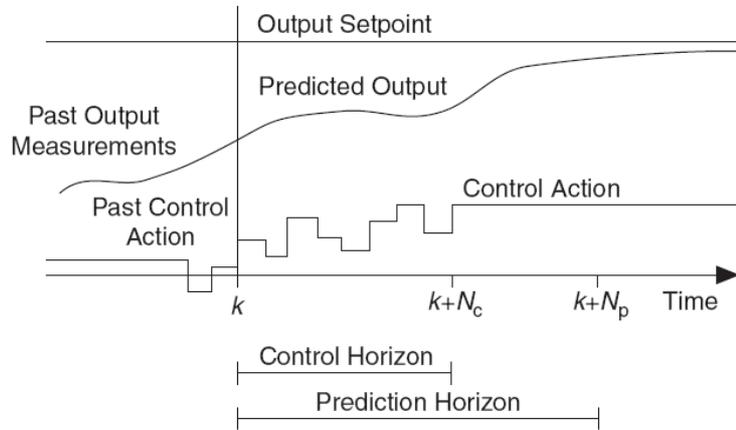
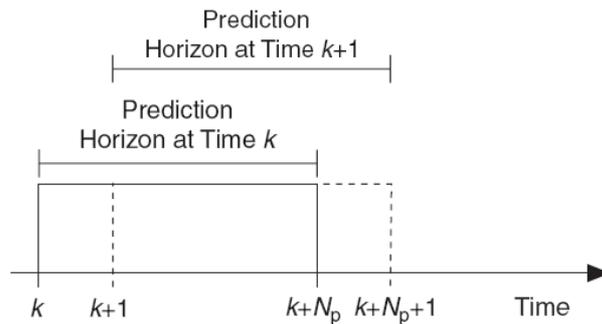


Figure 22.1: How MPC works

Figure 22.2: How the prediction horizon moves at each sample time k

Example 22.1 MPC control of a real air heater

Figure 22.3 shows a lab process consisting of an heated air tube where the air temperature (at temperature sensor 1) has been controlled with both MPC and – for comparison – a PID controller in a number of different cases described in the following. The control system is implemented on a PC with LabVIEW. The MPC controller and the Advanced PID controller in LabVIEW Control Design and Simulation Toolkit are used. The sampling time is 0.5 s.

Mathematical modeling

The MPC controller in LabVIEW requires a process model in the form of a discrete-time state-space model. Although advanced system identification

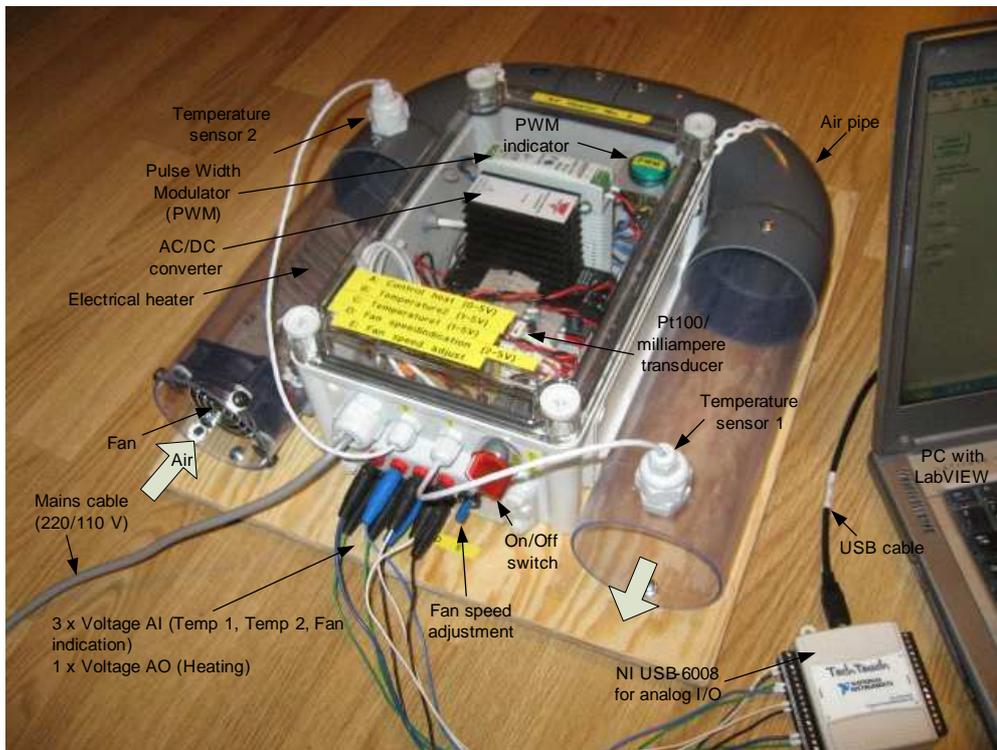


Figure 22.3: Example 22.1: A lab process consisting of a heated air tube where the air temperature will be controlled.

functions are available in LabVIEW System Identification Toolkit, a simple, manual model development is accomplished in this example: The model is estimated manually from the step response of the process.

Figure 22.4 shows to the left the process step response, i.e. the response in the temperature due to a step change of the control signal to the heater. This response indicates that a proper model is a “time-constant with time-delay” transfer function with the following parameters:

$$\text{Gain: } K = 3.5 \text{ } ^\circ\text{C/V} \quad (22.8)$$

$$\text{Time-constant: } T = 22 \text{ s} \quad (22.9)$$

$$\text{Time-delay: } \tau = 2 \text{ s} \quad (22.10)$$

To validate the model, and to possibly fine-tune model parameters, a simulator based on the estimated transfer function is run in real time and in parallel with the real process. The simulator and the real process is of course excited by the same control signal, which is an arbitrarily adjusted

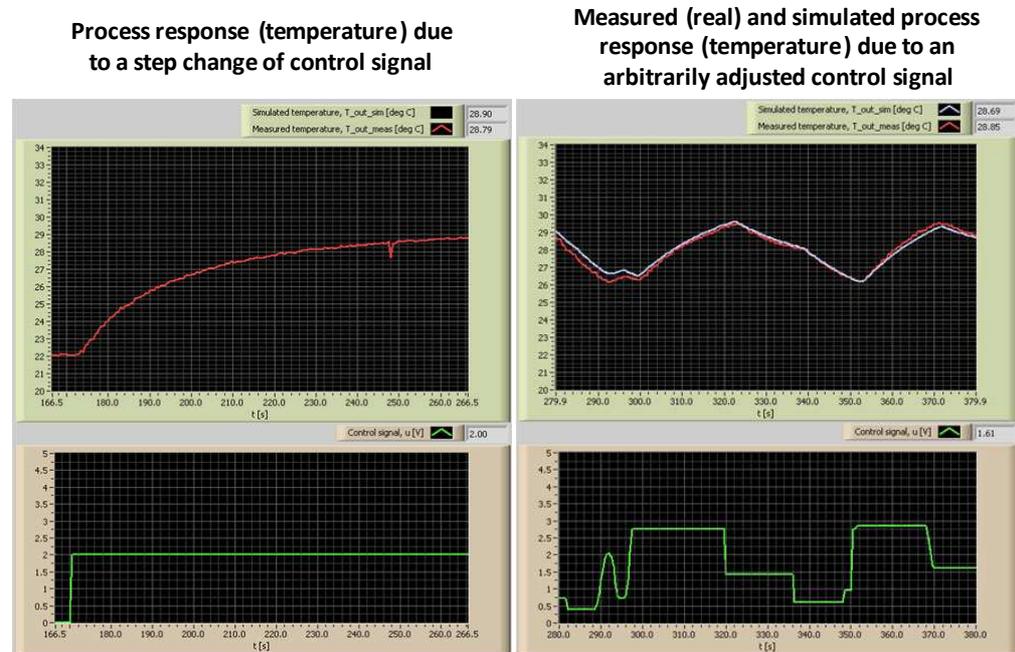


Figure 22.4: Example 22.1: Left: Process step response (step in control signal) used for model estimation. Right: Process response after arbitrarily adjusted control signal used for model validation

signal, see the plots to the right in Figure 22.4. It is clear that the model is quite good (accurate).

To obtain a discrete-time state-space model, model conversion functions in LabVIEW are used. (The state-space model is however not shown here.)

Settings of MPC and PID controller

MPC settings

Figure 22.5 shows the settings of various MPC parameters:

- **Horizons:** The prediction and control horizons are set not so different from process time-constant. They are set to 30, which corresponds to 15 sec since the the sampling time is 0.5 sec. (The process time-constant is 22 sec.)
- **Weightings:** The Output Error Weighting is set to 1. The Control Action Change Weighting is set by trial-and-error on real system to 40 (it can also be adjusted on the simulator, of course). Small

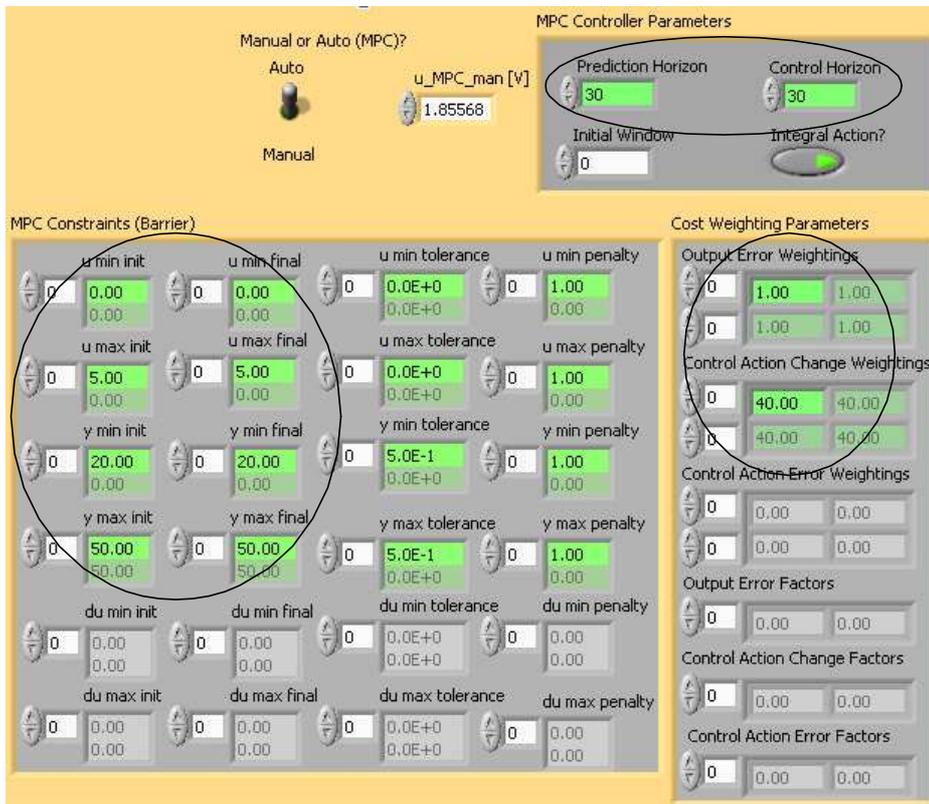


Figure 22.5: Example 22.1: MPC parameter settings

weightings gives fast, abrupt control. Large weighting gives sluggish control.

- **Constraints:** The constraints of the control signal u and the process output (measurement) y are set to the physical limits. The control signal range is 0 – 5 V, and the temperature measurement range is 20 – 50 °C.

PID settings

The PID controller is used as a PI controller with the following settings:

$$K_p = 0.42 \quad (22.11)$$

$$T_i = 18 \text{ s} \quad (22.12)$$

(The controller is tuned with Skogestad's method with the closed-loop time-constant set to – somewhat arbitrarily – 10 sec.)

Setpoint tracking with future setpoint step

Figure 22.6 shows the responses in the temperature and the control signal with PI control and MPC control after step changes of setpoint. MPC

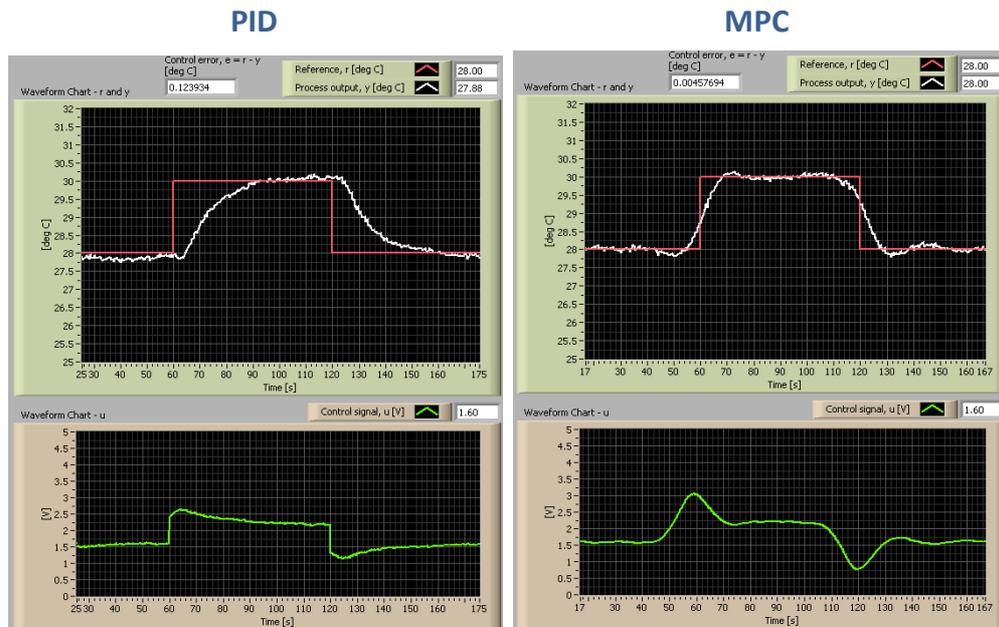


Figure 22.6: Example 22.1: Responses after step changes of the setpoint

control is much better than PID, because the MPC controller plans the control properly by taking into account the future setpoint changes. Observe that MPC starts changing control *ahead* of the setpoint change, while the PI controller changes the control action *after* setpoint is changed.

Setpoint tracking with future setpoint ramp

Figure 22.7 shows the responses with ramped changes of the setpoint. The MPC control gives excellent control, with almost zero control error, while the PI controller gives a clear non-zero control error. With a known future setpoint trajectory, the MPC is capable of giving superior control.

Propagation of measurement noise

Figure 22.8 shows how the measurement noise is propagated through the PID controller and MPC controller. The noise is more smoothed through the MPC (less propagation of noise), which is because more samples of the measurement signal are used in calculation of the control

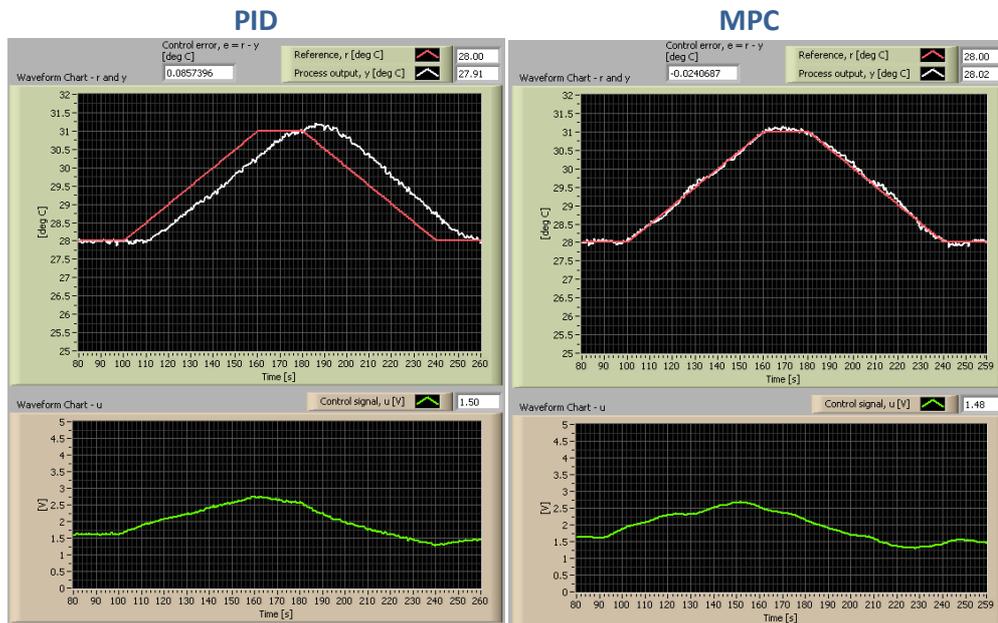


Figure 22.7: Example 22.1: : Responses after ramped changes of the setpoint.

signal with the MPC controller than with the PID controller. In other words, there is more “averaging” with the MPC controller.

Constrained control

Figure 22.9 shows the response in the temperature with MPC control when a maximum constraint of 30 °C is set for the process output variable (temperature). This constraint is one of the MPC parameters which the user can set. Actually, a tolerance of 0.5 K is set for this constraint. The Barrier method, which is an alternative to the Dual method, is used to define the constraints in the LabVIEW MPC used in this experiment. (The Dual method may give oscillations at the constraint limit, while the Barrier method does not.)

As seen from the response in Figure 22.9 the maximum limit of 30 °C is maintained, with a margin below of about 0.5 K.

Changing the weight of incremental control signal

Figure 22.10 shows responses with MPC control with increased and decreased weight of the control signal increment. As expected, with increased weight (more “expensive” control increments) the control signal will vary less, causing the process output variable (temperature) to respond

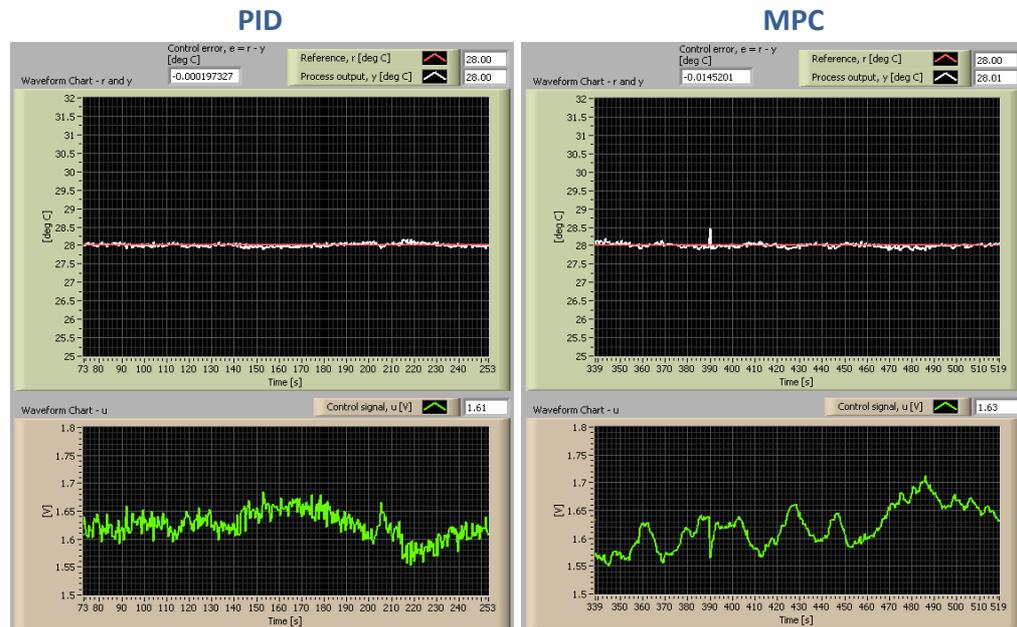


Figure 22.8: Example 22.1: Measurement noise propagation through the PI controller and the MPC controller

more slowly, cf. the left plots in Figure 22.10. And with decreased weight (“cheaper” control increments) the control signal will vary more, causing the process output variable (temperature) to respond more quickly, cf. the right plots in Figure 22.10. If the weight is set very close to zero, the MPC controller acts almost like an On/off controller, which is denoted “dead-beat control”.

[End of Example 22.1]

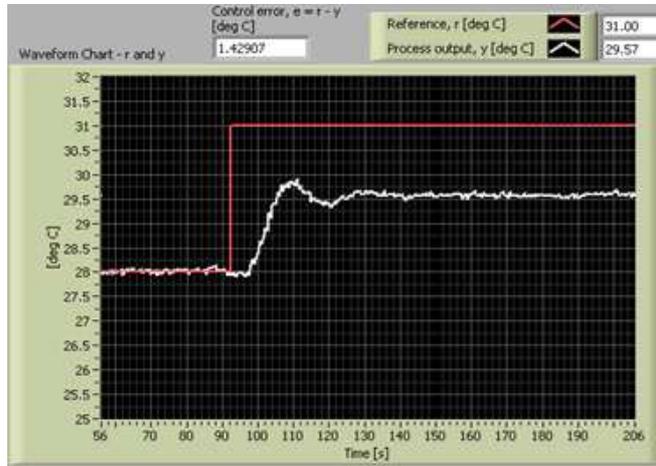


Figure 22.9: Example 22.1: Setting the process measurement constraint to 30 °C.

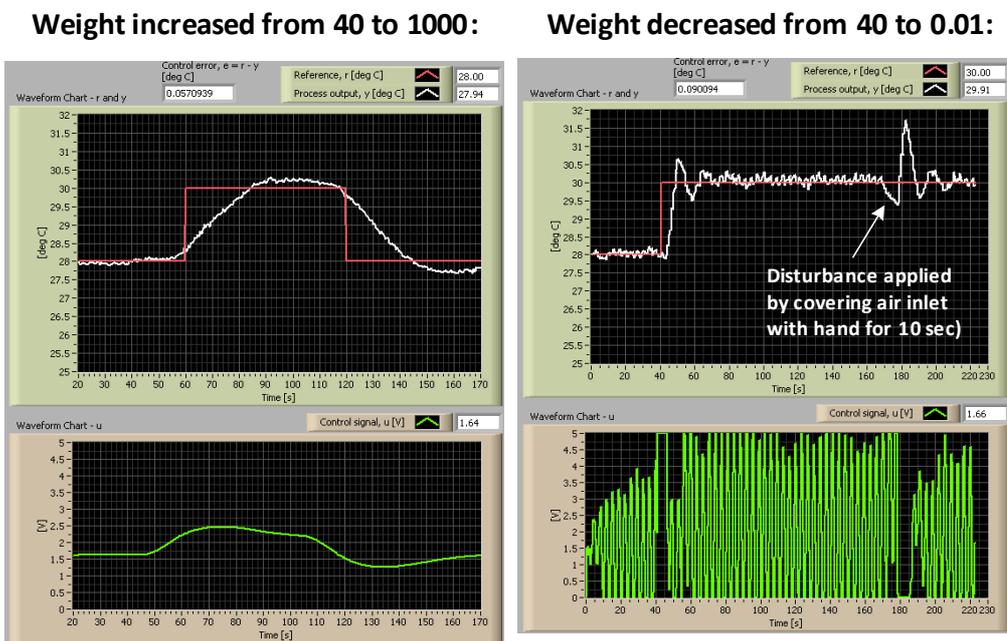


Figure 22.10: Example 22.1: Adjusting the weighting of incremental control signal to a relatively large value (left) and to a small value (right).

Chapter 23

Dead-time compensator (Smith predictor)

The dead-time compensator – also called the Smith predictor [15] – is a control method particularly designed for processes with dead-time (time delay). Compared to ordinary feedback control with a PI(D) controller, dead-time compensation gives improved setpoint tracking in all cases, and it may under certain conditions give improved disturbance compensation.

It is assumed that the process to be controlled has a mathematical model on the following transfer function form:

$$H_p(s) = H_u(s)e^{-\tau s} \quad (23.1)$$

where $H_u(s)$ is a partial transfer function without time delay, and $e^{-\tau s}$ is the transfer function of the time delay.

Simply stated, with dead-time compensation the bandwidth (quickness) of the control system is independent of the dead-time, and relatively high bandwidth can be achieved. However, the controller function is more complicated than the ordinary PID controller since it contains a transfer function model of the process. A dead-time compensator are implemented in some controller products.

Figure 23.1 shows the structure of the control system based on dead-time compensation. In the figure y_{mp} is a predicted value of y_m – therefore the name *Smith predictor*. y_{m1p} is a predicted value of the non time delayed internal process variable y_{m1} . There is a feedback from the predicted or calculated value y_{m1} . The PID controller is the controller for the non delayed process, $H_u(s)$, and *it is tuned for this process*. The controller

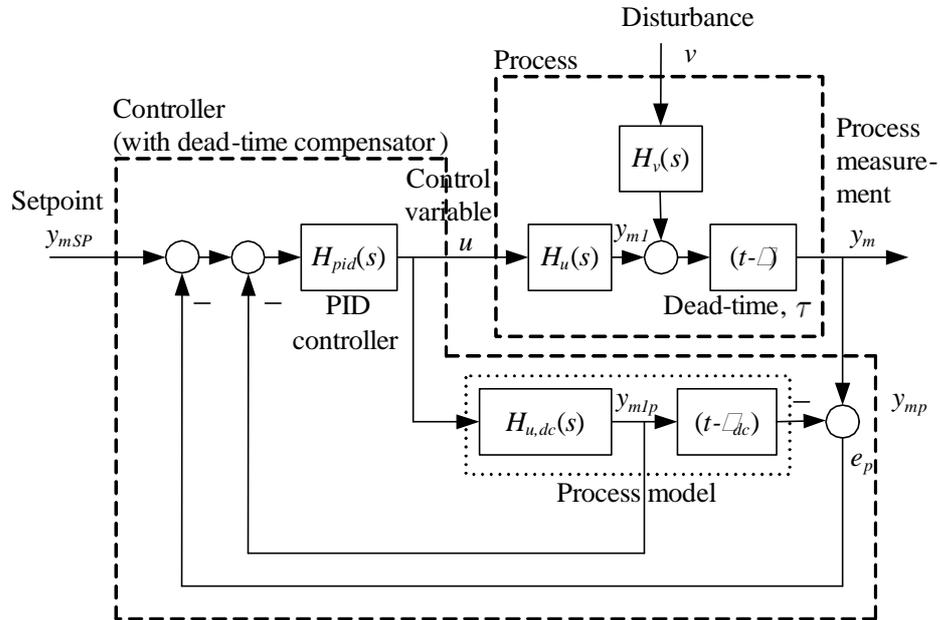


Figure 23.1: Structure of a control system based on dead-time compensation

tuning can be made using any standard method, e.g. the Skogestad's method which is reviewed in Appendix A. The bandwidth of this loop can be made (much) larger compared to the bandwidth if the time delay were included in the loop. The latter corresponds to an ordinary feedback control structure.

As long as the model predicts a correct value of y_m , the prediction error e_p is zero, and the signal in the outer feedback is zero. But if e_p is different from zero (due to modeling errors), there will be a compensation for this error via the outer feedback.

What is the tracking transfer function, $T(s)$, of the control system? To make it simple, we will assume that there are no modeling errors. From the block diagram in Figure 23.1 the following can be found:

$$T(s) = \frac{y_m(s)}{y_{mSP}(s)} = \frac{H_{pid}(s)H_u(s)}{1 + H_{pid}(s)H_u(s)} e^{-\tau s} = \frac{L(s)}{1 + L(s)} e^{-\tau s} \quad (23.2)$$

where

$$L(s) = H_{pid}(s)H_u(s) \quad (23.3)$$

is the loop transfer function of the loop consisting of the PID controller and the partial *non time-delayed* process $H_u(s)$.

How is the setpoint tracking and the disturbance compensation performance of the control system?

- **Setpoint tracking** is as if the feedback loop did not have time delay, and therefore *faster setpoint tracking* can be achieved with a dead-time compensator than with ordinary feedback control (with PID controller). However, the time delay of the response in the process measurement can not be avoided with a dead-time compensator.
- **Disturbance compensation:** In [11] the disturbance compensation with a dead-time compensating control system is investigated for a first order with time delay process. It was found that dead-time compensation gave better disturbance compensation (assuming a step in the disturbance) compared to ordinary feedback control only if the time delay (dead-time) τ is larger than the time constant T of the process.

Example 23.1 *Dead-time compensator*

Given a process with the following transfer functions (cf. Figure 23.1):

$$H_p(s) = \underbrace{\frac{K_u}{T_u s + 1}}_{H_u(s)} e^{-\tau s} \quad (23.4)$$

$$H_v(s) = \frac{K_v}{T_v s + 1} \quad (23.5)$$

where

$$K_u = 1; T_u = 0.5; K_v = 1; T_v = 0.5; \tau = 2 \quad (23.6)$$

The following two control systems have been simulated:

- *Dead-time compensator* for the process defined above. The internal controller, $H_{pid}(s)$, is a PI controller with the following parameter values:

$$K_p = 2.0; T_i = 0.36 \quad (23.7)$$

These PI parameters are calculated using Skogestad's method, cf. Table A.1, with $\tau = 0$, $T_C = 0.25$ ($= T/2$) and $k_1 = 1.44$.

- *Ordinary feedback control* with PI controller for the process defined above. The PI controller, $H_{pid}(s)$, is a PI controller with the following parameter values:

$$K_p = 0.12; T_i = 0.5 \quad (23.8)$$

These PI parameters are calculated using Skogestad's method, cf. Table A.1, with $T_C = 2$ ($= \tau$) and $k_1 = 1.44$.

Figure 23.2 shows the simulated responses for the two control systems due to a setpoint step and a disturbance step. The dead-time compensator gives better setpoint tracking and better disturbance compensation than ordinary feedback control does.

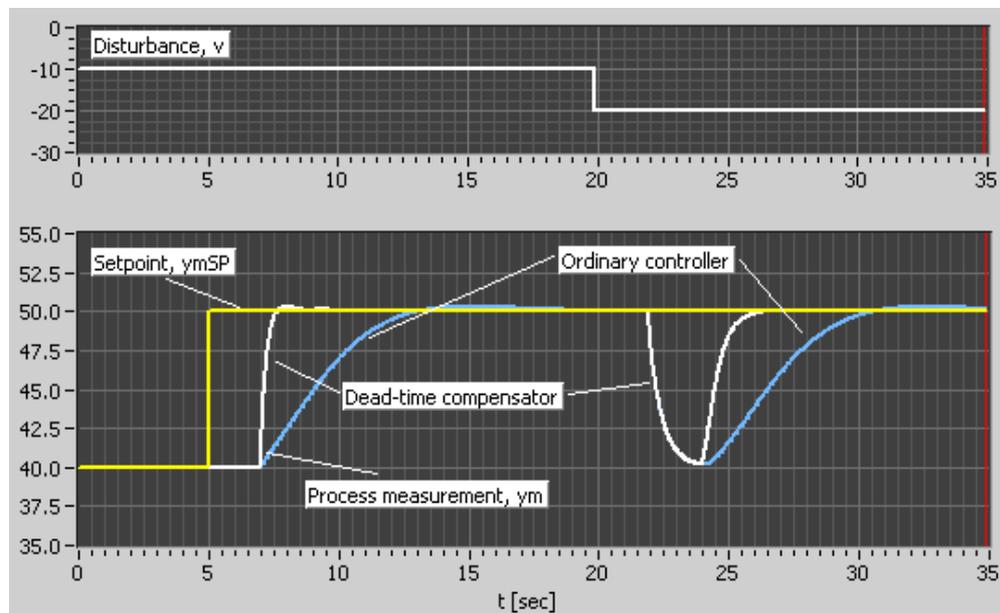


Figure 23.2: Example 23.1: Simulated responses for the two control systems due to a setpoint step and a disturbance step

[End of Example 23.1]

The dead-time compensator is model-based since the controller includes a model of the process. Consequently, the stability and performance robustness of the control system depend on the accuracy of the model. Running a sequence of simulations with a varied process model (changed model parameters) in each run is one way to investigate the robustness.

Appendix A

Model-based PID tuning with Skogestad's method

The information about Skogestad's method in this appendix is the same as given in [5].

A.1 The principle of Skogestad's method

Skogestad's PID tuning method [12]¹ is a model-based tuning method where *the controller parameters are expressed as functions of the process model parameters*. It is assumed that the control system has a transfer function block diagram as shown in Figure A.1.

Comments to this block diagram:

- The transfer function $H_{psf}(s)$ is a combined transfer function of the process, the sensor, and the measurement lowpass filter. Thus, $H_{psf}(s)$ represents all the dynamics that the controller “feels”. For simplicity we may denote this transfer function the “process transfer function”, although it is a combined transfer function.
- The process transfer function can stem from a simple step-response experiment with the process. This is explained in Sec. A.3.
- The block diagram shows a disturbance acting on the process. Information about this disturbance is not used in the tuning, but if

¹Named after the originator Prof. Sigurd Skogestad

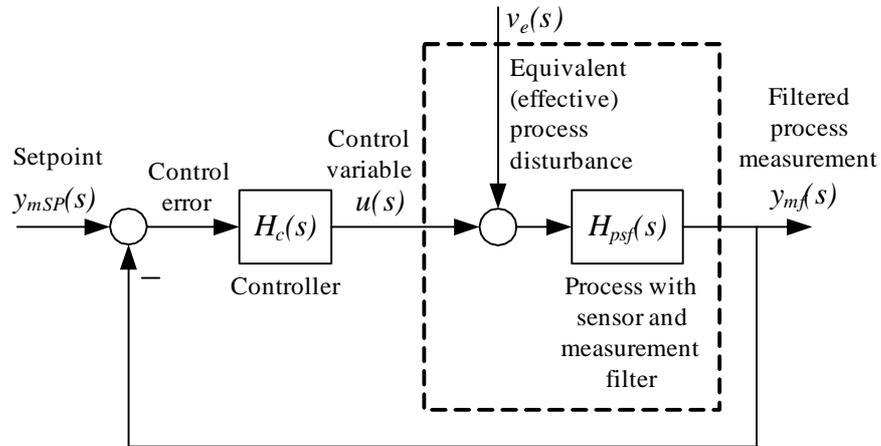


Figure A.1: Block diagram of the control system in PID tuning with Skogestad's method

you are going to test the tuning on a simulator to see how the control system compensates for a process disturbance, you should add a disturbance at the point indicated in the block diagram, which is at the process input. It turns out that in most processes the dominating disturbance influences the process dynamically at the “same” point as the control variable. Such a disturbance is called an *input disturbance*. Here are a few examples:

- Liquid tank: The control variable controls the inflow. The outflow is a disturbance.
- Motor: The control variable controls the motor torque. The load torque is a disturbance.
- Thermal process: The control variable controls the power supply via a heating element. The power loss via heat transfer through the walls and heat outflow through the outlet are disturbances.

The design principle of Skogestad's method is as follows. The control system *tracking transfer function* $T(s)$, which is the transfer function from the setpoint to the (filtered) process measurement, is *specified* as a first order transfer function with time delay:

$$T(s) = \frac{y_{mf}(s)}{y_{mSP}(s)} = \frac{1}{T_C s + 1} e^{-\tau s} \quad (\text{A.1})$$

where T_C is the time-constant of the control system which *the user must specify*, and τ is the process time delay which is *given* by the process model

(the method can however be used for processes without time delay, too). Figure A.2 shows as an illustration the response in y_{mf} after a step in the setpoint y_{mSP} for (A.1).

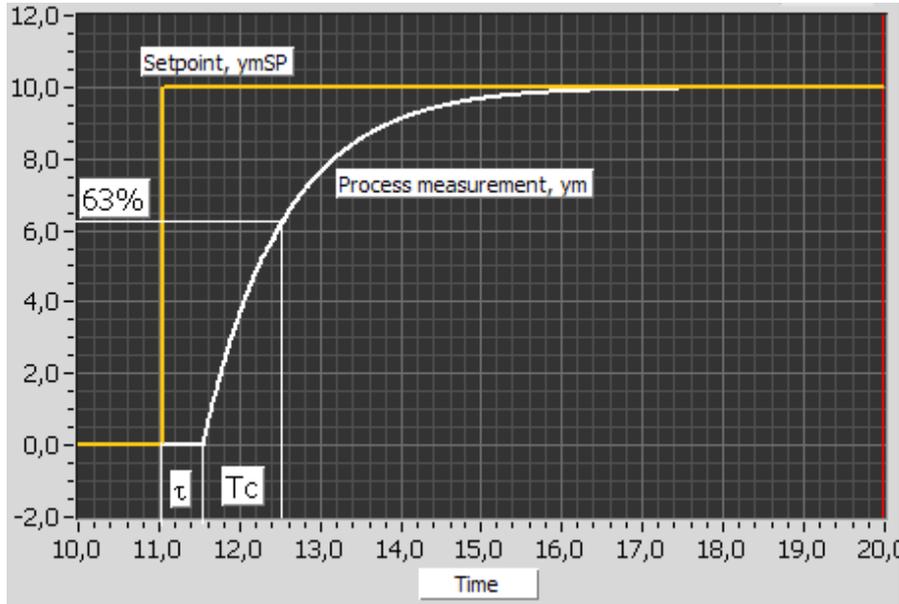


Figure A.2: Step response of the specified tracking transfer function (A.1) in Skogestad's PID tuning method

From the block diagram shown in Figure A.1 the tracking transfer function is

$$T(s) = \frac{H_c(s)H_{psf}(s)}{1 + H_c(s)H_{psf}(s)} \quad (\text{A.2})$$

Setting (A.2) equal to (A.1) gives

$$\frac{H_c(s)H_{psf}(s)}{1 + H_c(s)H_{psf}(s)} = \frac{1}{T_C s + 1} e^{-\tau s} \quad (\text{A.3})$$

Here, the only unknown is the controller transfer function, $H_c(s)$. By making some proper simplifying approximations to the time delay term, the controller becomes a PID controller or a PI controller for the process transfer function assumed.

A.2 The tuning formulas in Skogestad's method

Skogestad's tuning formulas for several processes are shown in Table A.1.²

Process type	$H_{psf}(s)$ (process)	K_p	T_i	T_d
Integrator + delay	$\frac{K}{s} e^{-\tau s}$	$\frac{1}{K(T_C + \tau)}$	$c(T_C + \tau)$	0
Time-constant + delay	$\frac{K}{Ts+1} e^{-\tau s}$	$\frac{T}{K(T_C + \tau)}$	$\min [T, c(T_C + \tau)]$	0
Integr + time-const + del.	$\frac{K}{(Ts+1)s} e^{-\tau s}$	$\frac{1}{K(T_C + \tau)}$	$c(T_C + \tau)$	T
Two time-const + delay	$\frac{K}{(T_1s+1)(T_2s+1)} e^{-\tau s}$	$\frac{T_1}{K(T_C + \tau)}$	$\min [T_1, c(T_C + \tau)]$	T_2
Double integrator + delay	$\frac{K}{s^2} e^{-\tau s}$	$\frac{1}{4K(T_C + \tau)^2}$	$4(T_C + \tau)$	$4(T_C + \tau)$

Table A.1: Skogestad's formulas for PI(D) tuning

For the “Two time-constant + delay” process in Table A.1 T_1 is the largest and T_2 is the smallest time-constant.³

Originally, Skogestad defined the factor c in Table A.1 as 4. This gives good setpoint tracking. But the disturbance compensation may become quite sluggish. To obtain faster disturbance compensation, you can use

$$c = 2 \quad (\text{A.4})$$

The drawback of such a reduction of c is that there will be more overshoot in the setpoint step respos, and the stability of the control loop will be reduced. Also, the robustness against changes of process parameters (e.g. increase of process gain and increase of process time-delay) will be somewhat reduced.

Skogestad suggests using

$$T_C = \tau \quad (\text{A.5})$$

for T_C in Table A.1 – unless you have reasons for a different specification of T_C .

A.3 How to find model parameters from experiments

The values of the parameters of the transfer functions in Table A.1 can be found from a mathematical model based on physical principles. The

²In the table, “min” means the minimum value (of the two alternative values).

³[12] also describes methods for model reduction so that more complicated models can be approximated with one of the models shown in Table A.1.

parameter values can also be found from a step-response experiment with the process. This is shown for the model *Integrator with time-delay* and *Time-constant with time-delay* in the following respective figures.

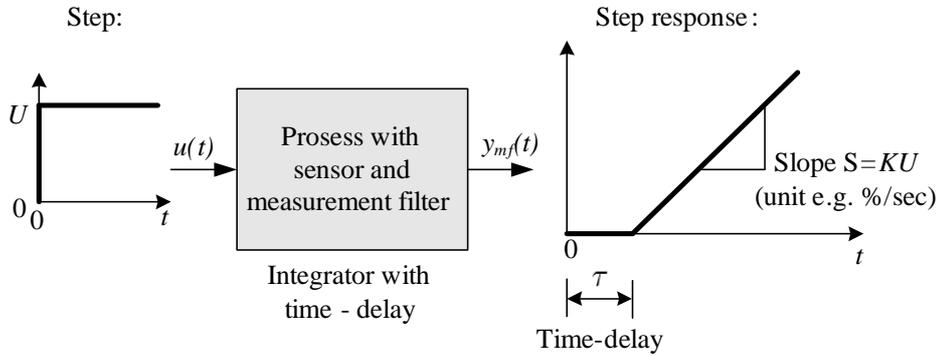


Figure A.3: How the transfer function parameters K and τ appear in the step response of an *Integrator with time-delay* process

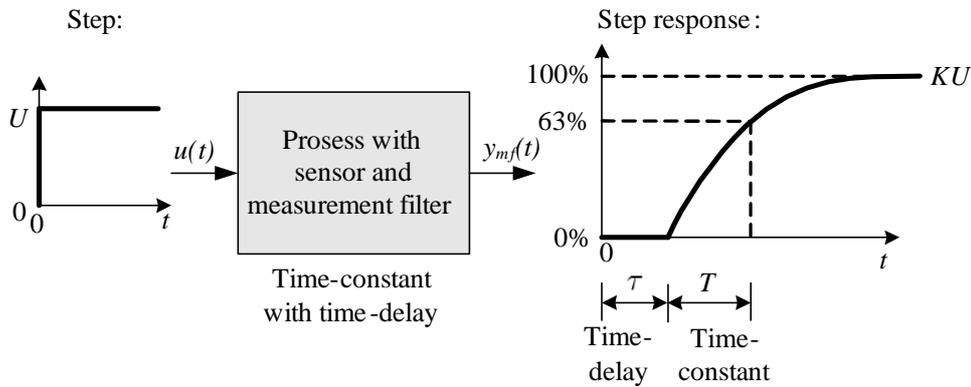


Figure A.4: How the transfer function parameters K , T , and τ appear in the step response of a *Time-constant with time-delay* process

A.4 Transformation from serial to parallel PID settings

Skogestad's formulas assumes a *serial* PID controller function (alternatively denoted cascade PID controller) which has the following

transfer function:

$$u(s) = K_{p_s} \frac{(T_{i_s}s + 1)(T_{d_s}s + 1)}{T_{i_s}s} e(s) \quad (\text{A.6})$$

where K_{p_s} , T_{i_s} , and T_{d_s} are the controller parameters. If your controller actually implements a *parallel* PID controller (as in the PID controllers in LabVIEW PID Control Toolkit and in the Matlab/Simulink PID controllers), which has the following transfer function:

$$u(s) = \left[K_{p_p} + \frac{K_{p_p}}{T_{i_p}s} + K_{p_p}T_{d_p}s \right] e(s) \quad (\text{A.7})$$

then you should transform from serial PID settings to parallel PID settings. If you do not implement these transformations, the control system may behave unnecessarily different from the specified response.⁴ The serial-to-parallel transformations are as follows:

$$K_{p_p} = K_{p_s} \left(1 + \frac{T_{d_s}}{T_{i_s}} \right) \quad (\text{A.8})$$

$$T_{i_p} = T_{i_s} \left(1 + \frac{T_{d_s}}{T_{i_s}} \right) \quad (\text{A.9})$$

$$T_{d_p} = T_{d_s} \frac{1}{1 + \frac{T_{d_s}}{T_{i_s}}} \quad (\text{A.10})$$

Note: The parallel and serial PI controllers are identical (since $T_d = 0$ in a PI controller). Therefore, the above transformations are not relevant for PI controller, only for PID controllers.

A.5 When the process has no time-delay

What if the process $H_p(s)$ is *without time-delay*? Then you can not specify T_C according to (A.5) since that would give $T_C = 0$ (zero response time of the control system). You must specify T_C to some reasonable value larger than zero. If you do not know what could be a reasonable value, you can simulate the control system for various values of T_C . If the control signal (controller output signal) is changing too quickly, or often reaches the maximum and minimum values for reasonable changes of the setpoint or the disturbance, the controller is too aggressive, and you can try increasing

⁴The transformations are important because the integral time and the derivative time are *equal*, cf. (20.38) and (20.39). If the integral time is substantially larger than the derivative time, e.g. four times larger, the transformations are not necessary.

T_C . If you don't want to simulate, then just try setting $T_C = T/2$ where T is the dominating (largest) time-constant of the process (assuming the process is a time-constant system, of course).

For the double integrator (without time-delay) I have seen in simulations that the actual response-time (or 63% rise-time) of the closed-loop system may be about twice the specified time-constant T_C . Consequently, you can set T_C to about half of the response-time you actually want to obtain.

Bibliography

- [1] Franklin G. F. , J. D. Powell, A. Emami-Naeini, *Feedback Control of Dynamic Systems*. Addison-Wesley, 1994.
- [2] Franklin G. F. and J. D. Powell, *Digital Control of Dynamic Systems*, Addison Wesley, 1980.
- [3] Goodwin G., Graebe S. F., Salgado M. E., *Control System Design*, Prentice Hall, 2001.
- [4] Grewal M. S. and A. P. Andrews, *Kalman Filtering, Theory and Practice*, Prentice Hall, 1993.
- [5] Haugen F, *Basic Dynamics and Control*, TechTeach (<http://techteach.no>), 2010.
- [6] Ljung L. , *System Identification, System Identification–Theory For The User*, Prentice-Hall, 1998.
- [7] Kalman R. E., *A New Approach to Linear Filtering and Prediction Problems*, Trans. ASME (Am. Soc. of Mech. Engineers), J. Basic Eng., mars 1960.
- [8] Kalman R. E. and R. C. Bucy, *New Results in Linear Filtering and Prediction Theory*, Trans. ASME (Am. Soc. of Mech. Engineers), J. Basic Eng., mars 1961.
- [9] Maciejowski J., *Predictive Control with Constraints*, Prentice Hall, 2002
- [10] Marlin Th. E. , *Process Control*, McGraw-Hill, 1995.
- [11] Meyer C., Seborg D. E., Wood R. K.: *A Comparison of the Smith Predictor and Conventional Feedback Control*, Chem. Eng. Sci., Vol . 31, 1976
- [12] S. Skogestad: *Simple Analytical Rules for Model Reduction and PID Controller Tuning*, J. Process Control, Vol. 13, 2003

- [13] Skogestad S. and Postlethwaite I., *Multivariable feedback control - Analysis and design*, 2nd Edition, Wiley, 2005.
- [14] Slotine J. J. E. and W. Li, *Applied Nonlinear Control*, Prentice-Hall, 1991.
- [15] Smith O. J.: *Closer Control of Loops with Dead Time*, Chem. Eng. Progr. 53.
- [16] Sorenson, H. W. (red.), *Kalman Filtering: Theory and Application*, IEEE Press, 1985.
- [17] Seborg D. , T. F. Edgar and D. A. Mellichamp, *Process Dynamics and Control*, John Wiley & Sons, 1989.
- [18] Åström K. J. and B. Wittenmark, *Computer Controlled Systems*, Prentice-Hall, 1990.

Index

- absolute form of PID controller, 101
- AD-converter, 99
- amplitude crossover frequency, 75
- amplitude gain, 26
- analysis of control systems, 39
- anti wind-up, 102
- anti windup, 271
- asymptotically stable, 57
- asymptotically stable system, 60
- augmented Kalman Filter, 231
- augmented observer, 201
- auto-covariance, 149

- Backward differentiation method, 94
- ballistic state estimator, 229
- bandpass filters, 33
- bandstop filters, 33
- bandwidth, 34, 46
- bandwidth
 - 1. ordens lavpassfilter, 35
- black-box model, 161, 175
- Bode diagram, 28
- Bode-Nyquist stability criterion, 78
- bumpless transfer, 102

- center difference method, 175
- certainty equivalence principle, 258
- characteristic equation, 59
- characteristic polynomial, 59, 68
- chirp signal, 171
- coloured noise, 153
- compensation properties, 41

- computer, 99
- controllability, 261
- convergence of LS-estimate, 168
- correlation, 151
- cross-covariance, 150
- crossover frequency, 46

- DA-converter, 99
- dB, 28
- dead-time compensator, 285
- decibel, 28
- decoupling, 245
- difference equation, 91
- discretize, 93

- equation-error vector, 164
- equilibrium point, 22
- error dynamics, 186, 190
- error model, 190
- error-model, 229
- estimation, 161
- estimation error model, 190
- Estimation of parameters in
 - dynamic models, 174
- Euler's backward method, 175
- Euler's forward method, 174
- excitation in parameter
 - estimation, 170
- Expectation value, 147
- extended Kalman Filter, 219

- feedback linearization, 245
- filter, 33
- Forward differentiation method, 94
- frequency components, 25
- frequency response, 25, 28, 29

- frequency response analysis, 45
- frequency response and transfer function, 28
- gain function, 28
- gain margin, 77
- highpass filters, 33
- impulse response, 56
- incremental form of PID controller, 101
- input disturbance, 290
- integral anti wind-up, 102
- Kalman Filter, 215, 218
- least squares method, 161
- linearization of non-linear models, 19
- local model, 20
- loop transfer function, 43
- lowpass filter
 - 2. ordens, 34
- lowpass filters, 33
- LQ control, 257
- LQR, 257
- LS-method, 161
- marginally stable, 57
- marginally stable system, 60
- mass-spring-damper, 16, 62
 - stabilitetsanalyse, 62
- mean value, 147
- measurement estimate error, 188
- measurement-updated estimate, 223
- model errors, 229, 243
- model-based predictive control, 273
- MPC, 273
- multiloop PID controller, 245
- normal equation, 165
- Nyquist's special stability criterion, 75
- Nyquist's stability criterion, 70
- Nyquist's stability criterion for open stable systems, 75
- observability, 197, 216
- observability matrix, 198, 217
- observed variable, 163
- observer, 185
- observer response time, 191
- operating point, 21
- optimal control, 257
- order
 - state-space model, 15
- parameter estimation, 161
- parameter vector, 163
- parsimony principle, 169
- passband, 33
- performance, 41
- phase crossover frequency, 75
- phase function, 28
- phase lag, 26
- phase margin, 77
- pole-funksjonen (MATLAB), 69
- poles and stability, 58
- positional form of PID controller, 101
- posteriori estimate, 223
- power of a signal, 148
- PRB-signal, 170
- prediction-error vector, 164
- predictor type Kalman Filter, 230
- priori estimate, 224
- Pseudo Random Binary Signal, 170
- pzmap-funksjonen (MATLAB), 69
- RC-circuit, 36
 - som lavpassfilter, 36
- regression model, 162
- regression variable, 162
- regression vector, 162
- reguleringssystem
 - stabilitetsanalyse, 69

- RHP (right half plane), 72
- robust control, 243
- robustness, 82, 210, 239, 288
- roots-funksjonen (MATLAB), 69
- Routh's stability criterion, 67

- self regulation, 51
- sensitivity bandwidth, 47
- sensitivity function, 43
- sensor failure, 210, 239
- separation principle, 258
- shaping filter, 153
- signal filter, 33
- Skogestad's method, 247, 252, 289
- Smith predictor, 285
- stabilitet
 - tilstandsrommodeller, 64
- stability, 55, 67
 - transferfunksjonsmodeller, 58
- stability margins, 77
- standard deviation, 148
- state-space model, 15
- static response, 92
- steady state Kalman Filter gain, 228
- stochastic signals, 145
- stopband, 33
- subspace methods, 176

- testing, 186, 215
- time-updated estimate, 224
- tracking properties, 41
- transfer function and frequency response, 28
- transformed control vector, 246, 252
- transformed process, 247

- unit pulse, 152
- unstable, 57
- unstable system, 61
- up-down-up signal, 172
- velocity form of PID controller, 101
- weight matrices, 258
- white noise, 151

- zero order hold (zoh), 120
- ZOH discretization, 120