Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering

Parallel Algorithms for Robust Adaptive Identification and Square-Root LQG Control Synthesis

Jan Schier

Dissertation

Prague, October 1994

Preface

During the course of this research, which has been carried out in the Institute of Information Theory and Automation of the Academy of Sciences of the Czech Republic (ÚTIA AV ČR), several people have provided varying degrees of support and encouragement, and I would like to hereby record my thanks for their assistance.

First of all, I wish to express my sincere gratitude and appreciation to my advisor Dr. Miroslav Kárný, head of the Dept. of Adaptive Systems, for his enthusiastic and invaluable support and patient guidance. He has created an excellent environment for research; I consider myself fortunate to be a part of it.

I would like to thank Dr. Jiří Kadlec, my second and semi-official advisor, for introducing me into the world of systolic computing and for many enlightening discussions. He has been a vast reservoir of knowledge, and always available for assistance. I would also like to thank him for the friendship that has been established between us during my Ph. D. studies.

Thanks also to Dr. Alena Halousková, Dr. Josef Böhm and all colleagues from the Department of Adaptive Systems at ÚTIA AVČR for providing many ideas, for proofreading the thesis and giving many constructive comments, as well as for all their encouraging.

Special gratitude to Dr. Fiona M. F. Gaston from the Department of Electrical and Electronic Engineering at the Queen's University of Belfast for proofreading and commenting the whole thesis and for trying to translate everything into decent English. Thanks also to Miss Elmawati Sutanto from the Department of Cybernetics of the University of Reading for helping me to improve English in some parts of text.

Finally, I would like to especially thank my parents, Pavel and Ludmila Schier, for all their love, patience and support.

II

Contents

Chapter 1

Introduction

Adaptive control derives its name from the adaptation of the controller to a system where dynamics are apriori unknown or slowly varying. It has been developed in hundreds of variants and modifications [29]. The LQG adaptive controllers (Linear model, Quadratic criterion, Gaussian disturbances) represent one of the most advanced groups among them. Their theory has reached maturity, which has been confirmed by successes in controlling variety of difficult applications (cold-rolling mills [7], paper machines [14], etc.)

Some high-speed and/or large order applications, however, impose computational requirements that cannot be met in real-time conditions by usual sequential implementations of the self-tuning algorithms. The reason is that the computational complexity of the two basic parts of the adaptive controller, i.e. the estimator of the system model and the controller itself, is $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$, respectively, where n is number of parameters of the model used to describe the controlled system.

This is even more of a problem when using modern advanced adaptive identification algorithms [27, 28]. These algorithms increase the stability of the control loop since they are more numerically robust with respect to weakly exciting data which is often encountered in the feedback control, but this increase of robustness is paid for by an increase of the computational complexity of the algorithm.

These limitations of LQ control have motivated research into *fast computational schemes*. Successful attempts were made to transform the identification algorithms into fixed-point arithmetic [17, 31, 32]. This research was further stimulated by the development of *digital signal processor* (DSP) chips in the 80s. Thanks to the special architecture of the processor (integration of a hardware multiplier/accumulator and use of more than one memory bank, together with heavily exploited pipelining), the use of DSP significantly speeded up the algorithms. The fixed-point arithmetic used by the algorithms was advantageous since the early DSPs didn't have a floating point unit (in fact, the fixed-point DSPs are still often used for economical reasons).

The design of parallel algorithms has been also strongly influenced by the advent of *systolic/wavefront arrays*. The advances in VLSI technology, together with the notion of systolization, have made possible the design of special-purpose array processors [30]. This, in turn, resulted in the design of special algorithms with structural features that make them amenable to *VLSI systolic implementation*, like modularity, pipelineability, and local communications.

Finally, the design of highly parallel/pipelined algorithms has also benefited from the development and wide industrial availability of programmable processors specially designed for parallel implementations, such as *transputers*.

Typical representatives of the systolic algorithms, with good numerical properties, are the algorithms of the *QR family* [38]. These algorithms work with QR decomposition [13], which uses an orthogonal matrix (usually denoted by Q) to transform an arbitrary matrix to an upper triangular matrix (usually denoted by R). In identification, these algorithms are used for a recursive update of a square-root information filter, with implementation on a triangular array of processing elements. The decomposition is usually performed by the Givens rotations [12]. An alternative solution is to use the Recursive Modified Gram–Schmidt (RMGS) procedure [33].

Recently, the systolic implementation of the Recursive Least Squares (RLS) algorithm with inverse updates has been derived [35, 39, 43], using the methods of algorithmic engineering [37]. This systolic algorithm is attractive in adaptive control applications, since it provides the parameter estimates, necessary for the control design, directly. This fact has motivated the author to use this algorithm in the thesis.

The method that is most often used for adaptive tracking of parameters in systolic algorithms is simple exponential forgetting. However, under the condition of feedback control the data to be processed in the estimator are often nearly linearly dependent. Then, the algorithms using the exponential forgetting may run into numerical difficulties just because of its principle, regardless of the numerical properties of a particular parallel implementation.

Unfortunately, the alternative methods of adaptive identification that are robust with respect to non-informative data (like the regularization described in [27, 28, 34]), are not suitable for systolic implementation, because the regularization is in conflict with the pipelined implementation.

A suitable solution of efficient implementation of the regularized identification on a systolic array, was recently published in [19]: with certain reasonable assumptions, the data is aggregated into blocks that are processed using exponential forgetting. After the whole data block is processed, so called *block regularization* is performed, and the whole process is repeated. The result *at the end* of each data block is exactly the same as if the regularization was performed after processing each data sample. This modification is suitable for systolic implementation on $\mathcal{O}(n^2)$ processors, with the sampling rate $\mathcal{O}(1)$ and the output period of parameter estimates $\mathcal{O}(n)$. The resulting throughput is comparable with the systolic estimator using exponential forgetting. So far, the implementation of this method was solved for information filter only [19], where so called weight flushing [38] must be used to extract the parameter estimates.

In this thesis, the problem of finding an efficient parallel systolic implementation of the block regularization principle in the square-root RLS algorithm with inverse updates (covariance filter) will be solved. The advantage of this solution is that the parameter estimates are provided directly at the output of the regularized systolic estimator and hence the weight flushing mechanism [38] is no more needed. In result, the coupling of the estimator with the controller is easier.

Also the controller may be accelerated by a systolic implementation. The first parallel implementations of LQ control used the fact that the LQ optimization is related, by duality, to the Kalman filter (KF) [10, 16, 41]; hence, systolic arrays for Kalman filtering could also provide the LQ optimal control law. The advantage of this approach is its generality: the array works with a generic state-space description and noise covariance matrix [9]. Considering speed, however, this appears to be also a disadvantage: since the Kalman filter works with a generic state-space description of the system, it requires $\mathcal{O}(n^3)$ operations per iteration, where n is the state dimension. Using a triangular planar systolic array of $\mathcal{O}(n^2)$ processors, we can achieve speedup from $\mathcal{O}(n^3)$ to $\mathcal{O}(n)$. Another disadvantage of this approach is the rather limited use of the resulting array, considering various control strategies: the array is suitable for implementation of the iterations-spread-in-time control strategy [23]; the receding horizon strategy requires however quite complicated outer feedbacks.

The computational complexity of the *LQ controller* may be reduced to $\mathcal{O}(n^2)$ by using the nonminimal canonical representation of the system, corresponding to an input–output (i/o) description of the plant, with a shifting arrangement of the input and output history [3, 23]. Using a systolic implementation on a triangular array of $\mathcal{O}(n^2)$ processors, the iteration period that is independent of the system order may be achieved in this case.

This solution has also been chosen for the systolic controller used in the thesis. Compared with the implementations described in [4, 5], we will introduce some improvements to the array – for example, the regulation problem with constant input reference will be treated as a new issue, important for controlling plants with a non-zero static offset.

1.1 The goal of the thesis

The goal of this thesis is to design a parallel systolic architecture for the multistep linear quadratic Gaussian adaptive controller. The reason for this research is to make the adaptive controller applicable in fast and/or large order applications.

Emphasis is given to the following points:

- Improvement of the numerical robustness of the systolic adaptive estimator with inverse updates (i.e. working with the factorized covariance matrix) [43] by implementation of the block regularization [19], while preserving the compactness and speed of the systolic architecture.

- Design of a compact systolic architecture for LQ single input single output (SISO) controller, with an implementation of the following types of control inputs [23]:

    - program control
    - regulation problem with a constant input reference

- Design of a mutual connection of systolic arrays for identification, LQ control design and for system input generation and of their connection to the controlled system.

- Outlines of implementation of the cautious control strategy (this strategy treats the uncertainty of the parameter estimates in the control law synthesis [23]).

The final goal will be to provide complete and detailed *cell level function description* of the systolic LQG controller, suitable for a direct implementation in parallel systems.

Chapter 2

Layout of the Thesis

The next chapters treat the following topics:

Chapter 3. This chapter introduces the *notation conventions and shortcuts* to be used in the thesis.

Chapter 4. This is an *introductory* chapter. A scheme of the adaptive control loop is presented here and the basic terms and quantities for a mathematical description of the control loop are introduced, following [23].

This chapter also introduces the control goal as a minimization of the loss function and outlines the principle of Bellman dynamic programming as a mathematical background for LQ optimization. Finally, it explains the need of the sufficient statistics of the controlled system to avoid the dimensionality growth in the control synthesis.

Chapter 5. This chapter gives an overview of *system identification*, to the extent necessary for the thesis.

Using a Bayesian probabilistic concept, it gives a summary of the Recursive Least Squares (RLS) identification algorithm [17, 35]. It explains the principle of the regularized time update and derives the mechanism of the block regularization [19]. Finally, it describes the square-root implementation of the RLS algorithm and of the regularization.

As an important conclusion it shows that the same regularization mechanism that is used for the information filter [19] may be also used for the square root RLS algorithm with inverse updates.

Chapter 6. This chapter presents **new results** of the thesis. It deals with the *systolic implementation of the block-regularized RLS algorithm.*

To give the reader the necessary background, an *introduction* to the principles of *systolic arrays* is given first. The reader, who is already familiar with this topic, may skip directly to explanation of function of the systolic array for RLS identification.

Most important part of this chapter is the detailed explanation and description of the *implementation of the block regularization in the RLS algorithm with the covariance (inverse) update*, accompanied by several illustrative examples.

Chapter 7. This is another *preparatory* chapter. The *principle* of the *LQ control algorithm* is explained here and the *square-root implementation* of this algorithm is derived, to give the necessary background knowledge for the next chapter.

Unlike most derivations published so far [3, 9], the practically important case of *non-zero* and possibly *time-varying setpoints* is considered. The resulting algorithm may also be used to treat a *non-zero static offset* of the controlled system.

Chapter 8. In this chapter, the *systolic implementation of the LQ control algorithm* is described and all the systolic mechanisms, that are used in this implementation, are explained, as a **new result** of the thesis.

The problems of *connecting* the systolic *selftuner* with the *controlled system* are solved.

Illustrative examples, documenting the behaviour of the systolic selftuner and the influence of the regularization on the control quality, are given.

Finally, the principle of implementation of the cautious control strategy is outlined in brief.

Chapter 9. Here, **concluding remarks** are given, summarizing the contribution of the thesis.

The most important novel contribution of the thesis is contained in Chapters 6 and 8.

Chapter 3

Notation

In this chapter, we shall set the notation conventions and introduce some shortcuts to be used in the paper.

Main symbols

[3]	...	citation
(5)	...	reference to equation
a, b	...	column vectors
A, B	...	matrices
I	...	unit matrix, $I = \text{diag}[1 \ldots 1]$,
a', A'	...	a, A transpose
∂x	...	dimension of the vector x
(n, n)	...	dimensions of a matrix
$S > 0 \;\; (S \geq 0)$	...	symmetric positive (semi) definite matrix
k	...	discrete time, time index of a sampling period; the observation of the process starts at $k = 1$
$\equiv$	...	symbol of equivalence
$:=$	...	symbol of assignment

Some abbreviations

p.d.f.	...	probability density function
c.p.d.f.	...	conditional p.d.f.
PE	...	processing element
w.r.t.	...	with respect to
i/o	...	inputs and outputs of a PE
$x(1,k)$	...	set of values of x from the beginning of observation till the time k (discrete time $j = 1, \ldots, k$)
$d(0,k)$	...	set of input-output data from the beginning of observation till the time k, including the initial data $d(0)$
$x(k \mid k-1)$	...	variable, computed in the k-th step using information, contained in data $d(0, k-1)$
w	...	abbrev. for both w_y *and* w_u (i.e. both input and output setpoint)
$\tilde{x}$	...	in equation: variable *before* the data update in PE description: *input* value of variable modified in PE
$\bar{\tilde{x}}$	...	variable *after* the data update and *before* the time update
$\breve{x}$	...	variable *after* the data update and exponential forgetting, but *before* addition of the regularizing parameter
$\bar{x}$	...	in equation: variable *after* the time update in PE description: *output* value of variable modified in PE

Probability distributions

$\mathcal{N}(0,\sigma)$	...	normal (Gaussian) distribution with zero mean value and constant dispersion σ
$p(a\|b)$	...	c.p.d.f. of a random variable a, conditioned by a random variable b
$p(y(k)\|k-1;u(k))$	...	abbreviation for $p(y(k)\|d(1,k-1),d(0),u(k))$
$\mathsf{E}\,[a]$	...	expected (mean) value of a random variable a
$\hat{y}(k\|k-1;u(k))$	...	abbreviation for the expected value $\mathsf{E}\,[y(k)\|d(1,k-1),d(0),u(k)]$
$\mathrm{Cov}\,[a,b]$	...	covariance $\mathsf{E}\,[(a-\mathsf{E}\,[a])(b-\mathsf{E}\,[b])']$

Note on time indexing

Where suitable for simplicity and better readability of the relations, the time index will be omitted. In recursive formulae, the ‘tilde’ – ‘bar’ notation will be used where possible.

Chapter 4

The adaptive control — basic notions

In this chapter, the *basic notions*, related to the adaptive control synthesis, to be used throughout the thesis are introduced. The chapter deals with the following topics:

- A scheme of a closed control loop is presented,
- The related quantities describing and quantifying its behaviour are mentioned,
- Operating modes of the controller are explained,
- The principle of the dynamic programming is outlined,
- Sufficient statistics of data and of input and output (i/o) reference values are introduced to the dynamic programming to avoid the dimensionality growth.

4.1 Adaptive control loop

The adaptive control loop, considered in the following text, is presented on Figure 4.1. It consists of four parts [23, 47] — the *controlled system* or *plant* (S), the *adaptive recursive identifier* or *estimator*(I), the *control law designer* (C) and the *generator of the system input* ($u(k)$).

The adaptive *identifier* (*estimator*) collects the inputs u and outputs y of the system and estimates parameters Θ of the system model, chosen by the

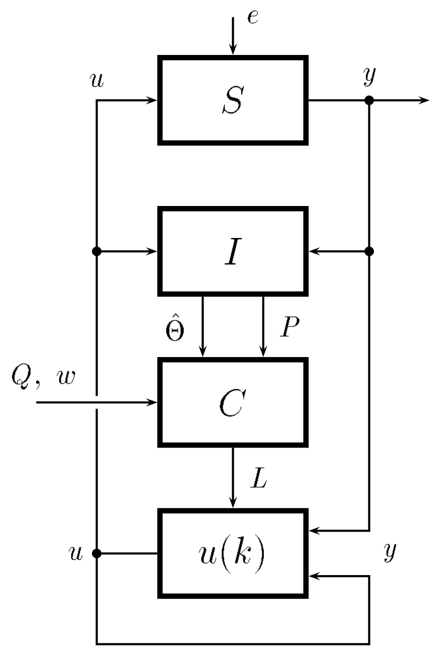


Figure 4.1: A scheme of an adaptive control loop

designer, so that it fits to the real system as well as possible. The estimate $\hat{\Theta}$ and the covariance matrix P of the parameters are passed to the controller.

The *controller* uses these estimates, together with the input and output penalties and setpoints, chosen by the user, to compute the *control law* L.

The control law is passed to the *system input generator*. In this block, using the past inputs and outputs, the current system input u is computed.

4.1.1 Signals and variables in the control loop

In further derivations, the following signals will be used (Fig 4.1):

System input u — quantity directly manipulated by the controller and fed into the system to influence it.

System output y — quantity influenced by the system input u and by the past values of the system input and output; $y(k) = f(u(k), d(0, k-1))$.

Reference values (setpoints) w_u and w_y — quantities, set by the user. They represent the desired value of the system input and output. The task of the controller is to generate the system input so that the system output follows the reference value as close as possible.

Estimate of system parameters $\hat{\Theta}$ and its *covariance* P — these variables are passed from the system estimator to the controller and are necessary for the control synthesis.

Unmeasurable disturbance e — characterizes the influence of the unmeasurable random quantities (industrial noise, temperature fluctuations, mismodelling, etc.) on the system output. We suppose this signal to be white and to have a normal distribution $\mathcal{N}(0, \sigma)$.

4.1.2 Control horizon

The controller plans the control actions for a certain period, called the *control horizon*. We will assume the control horizon to be finite. Throughout the thesis, we will denote it by H.

4.1.3 Quantification of the closed loop behaviour

To be able to quantify the closed-loop system behaviour, we introduce the following notions:

Control error is the output difference

$$e_y = y - w_y, \tag{4.1}$$

Control effort is the input difference

$$e_u = u - w_u. \tag{4.2}$$

Global error is a vector of the following arrangement:

$$e_g = \left[\begin{array}{cccccc} e_y(H), & e_u(H), & \ldots, & e_y(1), & e_u(1) \end{array} \right]', \tag{4.3}$$

Control law L is a mapping assigning the current value of the system input to the past input-output data and reference values:

$$\begin{gathered} L : (w(1,k), d(0,k-1)) \rightarrow u(k) \in \mathcal{U}(k|k-1), \\ k = 1, 2, \ldots, H, \end{gathered} \tag{4.4}$$

where

$\mathcal{U}(k|k-1)$ denotes the *set of admissible inputs.* (The admissible range of the inputs is limited by the technological restrictions, economical reasons, etc.

Remark: In the following text, unless it is necessary to differentiate them explicitly, we shall refer both w_y and w_u simply as w.

4.1.4 Operating modes of the controller

The task of the controller is to compensate the disturbances in the control loop and to keep the output signal as close as possible to the reference value.

The controller may work in several operating modes, each of which has a corresponding pattern of the reference values:

- *Regulation problem.* In this case the reference value is equal to some constant setpoint:

  $$w_y(k) = w_y = const.;\ k = 1 \ldots H$$

  This requirement is met either with

  $$w_u(k) = w_u = const.$$

  or with

  $$w_u(k) = u(k-1).$$

- *Program control.* The time development of the reference values is known in advance:

  $$w_y(k) = w_y(k|0),\ \ w_u(k) = w_u(k|0);\ k = 1 \ldots H$$

  is known, and this prior information is available to the controller.

- *Positional servo problem.* The future development of the setpoint is uncertain. The setpoint is expected to remain unchanged, but it cannot be guaranteed. In this case we choose

  $$\hat{w}_y(k|k-1) = w_y(k-1)$$

For the systolic implementation of the controller that will be described in the thesis, we make the following assumptions about the setpoints :

- The setpoints are conditioned only by the past values of themselves:

$$p(w(k)|k-1; w(1,k-1)) = p(w(k)|w(1,k-1)). \tag{4.5}$$

- There is no direct connection between the setpoints w and the controlled process output y, therefore we can assume that

$$p(y(k)|k-1; u(k), w(1,k)) = p(y(k)|k-1, u(k)), \tag{4.6}$$

i.e. the setpoint can influence the output only through the data already present in the condition.

4.2 Description of the control goal

Our aim in the closed loop control is to make, by affecting the system input, the system output following the required time evolution. To quantify the system behaviour, we define the *loss function* [23] as a quadratic form

$$J_0(1,H) = J_0(w(1,H), d(0,H)) = \left[\sum_{k=1}^{H}(q_y(k) + q_u(k))\right]. \tag{4.7}$$

The terms q_y and q_u are abbreviations for the weighted squares of deviations of the system output and input from the required values:

$$q_y = Q_y e_y^2, \tag{4.8}$$
$$q_u = Q_u e_u^2, \tag{4.9}$$

where Q_y and Q_u are the output and input penalties, respectively.

Since the future data entering the loss function are not available when the system input has to be decided, it is appropriate to design the control strategy using the *expected value* of the *loss function* as a criterion:

$$\begin{aligned} K_0(1,H) &= \mathsf{E}[J_0(1,H)] = \mathsf{E}[J_0(w(1,H), d(1,H))|d(0)] = \\ &= \int J_0(w(1,H), d(1,H))p(d(1,H)|d(0))p(w(1,H))\mathrm{d}d(1,H)\mathrm{d}w(1,H), \end{aligned} \tag{4.10}$$

where the symbol of integration represents multiple integration over the definition range of $d(0,H)$ and $w(1,H)$.

Using the expected value of the loss function, we can describe the control goal in the following way: we design a control strategy L, that minimizes the mean value of a quadratic loss function:

$$L^*(\cdot) = \arg\min_L \int J_0(1,H)p(d(1,H)|d(0))p(w(1,H))\mathrm{d}d(1,H)\mathrm{d}w(1,H) \tag{4.11}$$

4.3 Dynamic programming

To fulfill the information restriction (4.4), the inputs minimizing the loss function must depend only on the available information, i.e. on the past data ($u = u(k|k-1)$) [48]. To ensure this, we use the *dynamic programming*, which transforms the minimization of the loss function to a sequence of one-step minimizations.

Let us have the c.p.d.f.'s

$$p(y(k)|k-1; u(k)), \quad k = 1, 2, \ldots, H, \tag{4.12}$$

$$p(w_y(k)|w_y(1, k-1)), \quad k = 1, 2, \ldots, H. \tag{4.13}$$

and

$$p(w_u(k)|w_u(1, k-1)), \quad k = 1, 2, \ldots, H. \tag{4.14}$$

We can rewrite the control criterion (4.10) in a recursive form:

$$\begin{aligned} K(k|k-1) &= \mathsf{E}\left[J(k) + K(k+1|k)|k-1; u\right], \\ k &= H, \ldots, 1. \end{aligned} \tag{4.15}$$

starting from

$$K(H+1|H) = 0, \tag{4.16}$$

where

$$J(k) = q_y(k) + q_u(k). \tag{4.17}$$

It holds for J_0 and J

$$J_0 = \sum_{i=1}^{H} J(i)$$

The above formulation of the control criterion is related to (4.10) by the formula:

$$K_0(1, H) = \frac{1}{H} K(1|0)$$

Using the recursive definition of the control criterion, the control synthesis problem defined by (4.11) may be reformulated as:

> determine the c.p.d.f.'s
>
> $$p(u(k)|k-1; w(1,k)), \quad k = 1, 2, \ldots, H,$$
>
> minimizing in each step the criterion (4.15).

To solve this recursion, let us assume first, that we have determined the *future* inputs $u(k)$, $k = 1, \ldots, H-1$. Then only the last input $u(H)$ has to be computed. To choose an optimal value for this input, we have to determine the c.p.d.f.

$$p(u(H)|H-1; w(1,H)) \tag{4.18}$$

as a function of $d(0, H-1)$ and $w(1, H-1)$, so that the criterion

$$\begin{aligned} K(H|H-1) = &\int J(H)p(y(H)|H-1; u(H))\cdot \\ &\cdot p(u(H)|H-1; w(1,H))p(w(H)|w(1,H-1))\cdot \\ &\cdot p(w(1,H-1))\mathrm{d}y(H)\mathrm{d}u(H)\mathrm{d}w(H)\mathrm{d}w(1,H-1). \end{aligned} \tag{4.19}$$

is minimized for all possible values of $w(1, H)$ and $d(0, H-1)$.

From the definition, the loss function $J(H)$ is nonnegative. The c.p.d.f. $p(y(H)|H-1; u(H))$ is also nonnegative. The integral

$$\int J(H)p(y(H)|H-1; u(H))\mathrm{d}y(H)$$

must be nonnegative with some global minimum. It follows that $K(H|H-1)$ will be minimized, if all probability of $u(H)$ is placed in the minimum of the integrated function $J(H)$ (assuming that the set of admissible inputs (4.4) is chosen so that the minimizing u^* exists). The optimal choice is then:

$$p(u(H)|H-1; w(1,H)) = \delta(u(H) - u^*_H(w(1,H), d(0,H-1))), \tag{4.20}$$

where $\delta(\cdot)$ is the Dirac δ-function and

$$u^*(H|w(1,H), d(0,H-1)) \in \operatorname{Arg} \min_{u(H) \in \mathcal{U}(H|H-1)} K(H), \tag{4.21}$$

$$K^*(H|H-1) = \min_{u(H) \in \mathcal{U}(H|H-1)} K(H|H-1).$$

For the next step holds:

$$
\begin{gathered}
K(H-1|H-2) = \int [K^*(H|H-1)+ \\
+J(H-1)p(y(H-1)|H-2;u(H-1)] \cdot \\
\cdot p(u(H-1)|H-2, w(1,H-1))p(w(H-1)|w(1,H-2)) \cdot \\
\cdot p(w(1,H-2))\mathrm{d}y(H-1)\mathrm{d}u(H-1)\mathrm{d}w(H-1)\mathrm{d}(w(1),w(H-2)).
\end{gathered}
$$

We can proceed in the optimization in the above described way, getting the following result:

- The optimal control strategy, minimizing the criterion (4.10), is deterministic:
$$u^*(k) = u^*(k|w(1,k), d(0,k-1)), \tag{4.22}$$
- and is generated by the functional recursion
$$J(k) = Q_y(k)e_y^2(k) + Q_u(k)e_u^2(k) \tag{4.23}$$
$$
\begin{gathered}
K(k|k-1) = \int [K^*(k+1|k) + J(k)] \cdot \\
\cdot p(y(k)|k-1;u(k))p(u(k)|k-1;w(1,k)) \cdot \\
\cdot p(w(k)|w(1,k-1))p(w(1,k-1)) \cdot \\
\mathrm{d}y(k)\mathrm{d}u(k)\mathrm{d}w(k)\mathrm{d}w(1,k-1)
\end{gathered} \tag{4.24}
$$
$$u^*(k) \in \operatorname*{Arg\ min}_{u(k)\in\mathcal{U}(k|k-1)} K(k|k-1) \tag{4.25}$$
$$K^*(k|k-1) = \min_{u(k)\in\mathcal{U}(k|k-1)} K(k|k-1) \tag{4.26}$$
$$k = H, H-1, \ldots, 1$$

  starting with
$$K(H+1|H) = 0 \tag{4.27}$$

The minimum of the criterion is

$$K_0^*(1,H) = \frac{1}{H}K^*(1|0).$$

4.4 Need of a sufficient statistics

As was already said, the optimal control law (4.4) for each k is to be calculated as a function of the data $w(1,k-1), d(1,k-1)$, which are, however, not known at the beginning of control design. Hence, the domain of the function $u^*(k)$ (4.22, 4.25) is in general of a growing dimension. It may be reduced to finite and fixed dimension under the following conditions [23, 48]:

- There exist sufficient statistics $\mathcal{S}_y(k|k-1), \mathcal{S}_w(k|k-1)$, that contain information equivalent to data $d(0,k-1), w(1,k-1)$. These statistics are deterministic, multivariate, and they are functions of the observed data:

$$\begin{aligned} p(y(k)|k-1;u(k)) &= p(y(k)|\mathcal{S}_y(k|k-1),u(k)), & (4.28)\\ p(w(k)|w(1,k-1)) &= p(w(k)|\mathcal{S}_w(k|k-1)). & (4.29)\end{aligned}$$

- The evolution of these statistics is known:

$$\begin{aligned} \mathcal{S}_y(k+1|k) &= f_y(y(k),u(k),\mathcal{S}_y(k|k-1),k), & (4.30)\\ \mathcal{S}_w(k+1|k) &= f_w(w(k),\mathcal{S}_w(k|k-1),k). & (4.31)\end{aligned}$$

  The evolution of the statistics $\mathcal{S}_w(k+1|k)$ is, in fact, described by the relations, given in subsection (4.1.4).

- It is possible to express the loss function as follows

$$J_0(1,H) = \sum_{k=1}^{H} J(y(k),u(k),w(k),\mathcal{S}_J(k|k-1)) \qquad (4.32)$$

  where $\mathcal{S}_J(k|k-1)$ is a fixed-dimensional function of data such that

$$\mathcal{S}_J(k+1|k) = f_J(y(k),w(k),u(k),\mathcal{S}_J(k|k-1)). \qquad (4.33)$$

  With the quadratic form (4.17) used for the loss function, the formula (4.32) simplifies to

$$J_0(1,H) = \sum_{k=1}^{H} J(y(k),u(k),w(k)), \qquad (4.34)$$

  (the particular terms in the quadratic form depend only on the input-output value from the corresponding step, but they are independent of the past value of the loss function).

Let us denote

$$\mathcal{S}(k|k-1) = (\mathcal{S}_J(k|k-1), \mathcal{S}_y(k|k-1), \mathcal{S}_w(k|k-1)) \tag{4.35}$$

Under the above given conditions, the optimal control law, giving $u^*(k)$, operates on $w(k)$ and on the statistics $\mathcal{S}(k|k-1)$. To simplify the following relations, we shall use the abbreviations

$$J = J(y(k), u(k), w(k)) \tag{4.36}$$

and

$$K^* = K^*(k+1|\mathcal{S}(k|k-1)). \tag{4.37}$$

Using the quadratic form of the loss criterion the sufficient statistics simplify to

$$\mathcal{S} = (\mathcal{S}_y(k|k-1), \mathcal{S}_w(k|k-1)). \tag{4.38}$$

The optimal control law then reads ('tilde' symbol denotes value entering the step of recursion, 'bar' symbol the resulting value — see Chapter 3):

$$u^*(w, \mathcal{S}) \in \operatorname{Arg}\min_u \mathsf{E}\left[\tilde{K}^* + J|u, w, \mathcal{S}\right], \tag{4.39}$$

$$\bar{K}^* = \min_u \mathsf{E}\left[\tilde{K}^* + J|u, w, \mathcal{S}\right], \tag{4.40}$$

$$k = H, H-1, \ldots, 1, \tag{4.41}$$

$$u(k) \in \mathcal{U}(k, \mathcal{S}(k|k-1))$$

$$K^*(H+1|H) = 0. \tag{4.42}$$

and the functional recursion (4.24, 4.25, 4.26) is modified in a corresponding way.

4.5 Summary

In this chapter, we have introduced the basic notions related to the problem, solved in the thesis: to the adaptive control.

We described the scheme of the adaptive control loop and mentioned the related quantities and the possible operating modes of the controller. After description of the control goal, we gave the formulae of the dynamic programming. Finally, we noted the necessity of using the sufficient statistics of the system output and of the i/o reference values to avoid the dimensionality growth in the dynamic programming.

In the *next chapter*, devoted to the *system identification*, we shall solve the problems of determination of the sufficient statistics $\mathcal{S}_y$. We shall parameterize the probability distribution $p(y|k-1;u) = p(y|\mathcal{S}_y(k|k-1),u)$ by a linear regression model, leading to self-reproducing statistics. We shall also introduce formulae for a recursive update of parameters of this statistics. Finally, we shall describe the square-root implementation of the identification algorithm.

Chapter 5

Adaptive RLS identification

In the preceding chapter, we have shown the need of the system model $p(y(k)|k-1; u(k))$ (4.12) for computation of the dynamic programming recursion. We have also learned that to avoid the dimensionality growth in the dynamic programming, it is necessary to use sufficient statistics $S_y(k+1|k) = f_y(y(k), u(k), S_y(k|k-1), k)$ (4.30).

In this chapter we shall focus on the following topics:

- Bayesian description of the system model and its relation to the Recursive Least Squares (RLS) identification of the ARX model, data and time update of the sufficient statistics [48]

- Regularized time update, concept of block-regularized update [19, 21, 27, 28, 34]

- Square-root implementation of the RLS algorithm [36, 43]

- Implementation of regularization in the square-root RLS algorithm as an input of alternative data [19].

The reader, who is familiar with the Bayesian probabilistic concept, may skip directly to the description of the regularized time update (Sec. 5.3).

On the other hand, the reader should not skip the explanation of the block-regularization (Sec. 5.4), which represents rather uncommon concept and will be essential for the systolic implementation of the regularized estimator.

5.1 Recursive identification of the system model

5.1.1 Probabilistic description of the system

The c.p.d.f. $p(y(k)|k-1; u(k))$, introduced in the previous chapter, is dependent on the system structure.

To describe the uncertainty of our knowledge of the system, we introduce a set of (slowly varying) system models, formally parameterized by $\{\mu\}$, conditioning the system output [23]:

$$p(y(k)|k-1; u(k), \mu(k)). \tag{5.1}$$

We suppose the uncertainty of the model determination to be described by a c.p.d.f.

$$p(\mu(k)|k-1; u(k)). \tag{5.2}$$

Let us further suppose that the control strategy $p(u(k)|k-1, \mu(k))$ does not use more information about the unknown parameters than is contained in the past input-output data. The conditioning by the system model μ is redundant then, and it holds

$$p(u(k)|k-1, \mu(k)) = p(u(k)|k-1). \tag{5.3}$$

This relation is definition of conditions that are called *natural conditions of control* [23, 48]. From the natural conditions of control it also follows that

$$p(\mu(k)|k-1; u(k)) = p(\mu(k)|k-1). \tag{5.4}$$

Using the above given conditional probabilities and the natural conditions of control, we can integrate out the conditioning by the set of system models:

$$p(y(k)|k-1; u(k)) = \int p(y(k)|k-1; u(k), \mu(k)) p(\mu(k)|k-1) \mathrm{d}\mu(k). \tag{5.5}$$

Data and time update of the system model

To describe the evolution of the set of models, we introduce the *data* and the *time update* of the set [23, 19].

The *data update* describes the influence of the information, gained from a data measurement, on the system model.

Applying the Bayes rule, we can write in terms of conditional probabilities:

$$p(\mu(k)|k) = \frac{p(y(k)|k-1; u(k), \mu(k))p(\mu(k)|k-1)}{\int p(y(k)|k-1; u(k), \mu(k))p(\mu(k)|k-1)\mathrm{d}\mu(k)}. \tag{5.6}$$

The *time update* of the set of models is generally described by the formula:

$$p(\mu(k+1)|k) = \int p(\mu(k+1)|k; \mu(k))p(\mu(k)|k)\mathrm{d}\mu(k). \tag{5.7}$$

where

$$p(\mu(k+1)|k; \mu(k)) \tag{5.8}$$

formally describes the time development of the set of models.

Usually, however, we are not able to give exact description of this development and we have to approximate this step by various methods of "forgetting".

In [23, 26], the forgetting step was treated as a compromise between two ways of modification of the a-posteriori c.p.d.f. of parameters:

$$p_0(\mu(k+1)|k) = p(\mu(k+1) = \mu(k)|k), \tag{5.9}$$

which represents the situation when the parameters are constant and their a-posteriori c.p.d.f. is not influenced by the forgetting step, and

$$p_1(\mu(k+1)|k) = p^*(\mu(k+1)|k), \tag{5.10}$$

where $p^*(\mu(k+1)|k)$ is an alternative c.p.d.f., chosen by user.

In the forgetting step, we combine these two c.p.d.f.'s:

$$p(\mu(k+1)|k) = p_0(\mu(k+1)|k)^{\lambda(k)} p_1(\mu(k+1)|k)^{(1-\lambda(k))} \tag{5.11}$$

$$0 < \lambda(k) \le 1. \tag{5.12}$$

The *weighting* or *forgetting factor* $\lambda(k)$ expresses our believe in modification of the a-posteriori c.p.d.f. of parameters. In [19], this formulation of forgetting is called *regularized forgetting.*

We should note that from the Bayesian point of view, the well-known *exponential forgetting* is a *special case* of the regularized forgetting for p_1 set to a limit, *non-informative* Gaussian distribution.

5.1.2 Autoregression model with exogenous input (ARX model)

Let us parameterize the c.p.d.f.'s by an *AutoRegression model with eXogenous input* [48].

Using this model, we describe the dependence of $y(k)$ on the past input-output data by the following formula:

$$\begin{aligned} y(k) &= \Theta'(k)\varphi(k) + e(k) = \\ &= \sum_{i=1}^{l_y} a_i(k)y(k-i) + \sum_{j=0}^{l_u} b_j(k)u(k-j) + e(k) \end{aligned} \tag{5.13}$$

where

$y(k)$ denotes the current system output,

$\varphi(k)$ is a data vector composed of the recent system input and past outputs and inputs, consisting of

$y(k-i)$ — the past system outputs and

$u(k-j)$ — the current and the past system inputs,

$\Theta(k)$ is the vector of regression parameters, consisting of

$a_i(k)$ — autoregression parameters, related to the past values of the system output and

$b_j(k)$ — regression parameters, related to the current and the past values of the system input,

$e(k)$ is a scalar $\mathcal{N}$ white noise with zero mean value and dispersion σ,

$$\sigma = \mathrm{Cov}\left[e(k), e(k)\right].$$

Actual arrangement of the regression parameters in the vector Θ is generally not important and depends on the demands of a particular algorithmization. Throughout the following text, we shall suppose, because of needs of the control algorithm, that the vector has an arrangement with paired a_i and b_i coefficients:

$$\Theta = [b_0, a_1, b_1, \ldots, a_l, b_l], \tag{5.14}$$

where

$$
\begin{aligned}
l &= \max(l_y, l_u), \\
a_i &= 0, \;\; l_y < i \le l, \\
b_i &= 0, \;\; l_u < i \le l.
\end{aligned}
\tag{5.15}
$$

If $b_l = 0$, it is not included in Θ.

For later use we define the *number of regression parameters* n:

$$n = \partial\Theta. \tag{5.16}$$

The ARX model specifies two initial moments only. For the Bayesian estimation, however, full probabilistic description of the system is needed. This question will be discussed in the following paragraph.

5.1.3 Normal distribution of the system output

Using the above introduced ARX model (5.13) for the system description and assuming the disturbance $e(k)$ to be white and to have normal character $\mathcal{N}(0, \varsigma(k))$, the *predictive c.p.d.f. of the system output* can be described by a *normal distribution*:

$$
\begin{aligned}
& p(y(k)|k-1; u(k), \Theta(k), \sigma(k)) = \\
&= \frac{1}{\sqrt{2\pi\sigma(k)}} \exp\left\{-\frac{1}{2\sigma(k)}[y(k) - \mathsf{E}[y(k)]]^2\right\} = \\
&= \frac{1}{\sqrt{2\pi\sigma(k)}} \exp\left\{-\frac{1}{2\sigma(k)}[y(k) - \Theta'(k)\varphi(k)]^2\right\}.
\end{aligned}
\tag{5.17}
$$

The system model μ (5.1) is defined by a set of $n+1$ system parameters, parameterizing the distribution:

$$\mu = \begin{bmatrix} \Theta & \sigma \end{bmatrix}.$$

Let us remind the reader of the formula for the Bayesian recursion (5.6). In this paragraph we discussed the choice of sufficient statistics for the c.p.d.f. $p(y(k)|k-1; u(k), \mu(k))$, but the Bayesian recursion still requires the a-priori p.d.f. $p(\mu(k)|k-1)$ to be specified. In the next paragraph, we shall specify the statistics for this probability function.

5.1.4 Gauss-inverse-Wishart distribution of the system parameters

The prior c.p.d.f. of the system parameters has a self-reproducing form of the *Gauss-inverse-Wishart (GiW) distribution*:

$$\begin{aligned} p(\Theta(k), \sigma(k)|k-1) = \rho(V,\nu)\sigma(k)^{-\frac{\nu(k|k-1)+n-2}{2}} \cdot \\ \cdot \exp\left\{-\frac{1}{2\sigma(k)} J(\Theta(k)|k-1)\right\}, \end{aligned} \tag{5.18}$$

$$J(\Theta(k)|k-1) = \begin{bmatrix} -\Theta(k) \\ 1 \end{bmatrix}' V(k|k-1) \begin{bmatrix} -\Theta(k) \\ 1 \end{bmatrix}. \tag{5.19}$$

where

$p(\Theta(k), \sigma(k)|k-1)$ is the p.d.f. of the parameters Θ in the k-step of identification, conditioned on the data $d(0, k-1)$.

$J(\Theta(k)|k-1)$ is a cost function of parameters $\Theta(k)$, conditioned on the data $d(0, k-1)$ [19]. For the GiW-distribution, it determines the shape of the distribution.

$V(k|k-1)$ is a positive-definite symmetric *extended information matrix*. This matrix accumulates the information contained in the data measurements. For later use, we shall divide this matrix to the following submatrices:

$$V = \begin{bmatrix} V_\varphi & v_{\varphi y} \\ v'_{\varphi y} & v_y \end{bmatrix}, \tag{5.20}$$

where

V_φ is an (n, n) *information matrix*,

$v_{\varphi y}$ is an $(n, 1)$ vector, and

v_y is scalar.

The meaning of these parts will be explained in the subsection 5.3.

$\nu(k|k-1)$ is a *number of degrees of freedom* of the GiW-distribution. It characterizes the number of data effectively accumulated in the information matrix; $\nu(k|k-1) > 0$.

$\rho(V, \nu)$ is a normalization constant [48].

The extended information matrix V and the number of degrees of freedom ν fully determine the sufficient statistics of the GiW-distribution [48].

Note on arrangement of vector $[-\Theta'(k)\ 1]$ and of the extended information matrix. The vector $[-\Theta'(k)\ 1]$ and the information matrix $V(k|k-1)$ in (5.18) are usually used in the arrangement with 1 on the top (and v_y in the upper left corner) [23, 48]. The arrangement that is used here follows the needs of the systolic estimator, as will be shown latter.

5.1.5 Relating Bayesian estimation to the Recursive Least Squares identificiation.

To relate the Bayesian estimation to the Recursive Least Squares (RLS) identification, it is advantageous to write down the GiW-distribution of the system parameters in another form [17, 23, 48], with the sufficient statistics using the inverse of the information matrix:

$$\begin{aligned} p(\Theta(k), \sigma(k)|k-1) &= \rho(V, \nu)\sigma(k)^{-\frac{\nu(k|k-1)+n-2}{2}} \cdot \\ &\cdot \exp\left\{-\frac{1}{2\sigma(k)} J(\Theta(k)|k-1)\right\}, \end{aligned} \tag{5.21}$$

$$\begin{aligned} J(\Theta(k)|k-1) &= \Lambda(k|k-1) + \\ &+ \left[\Theta(k) - \hat{\Theta}(k|k-1)\right]' P^{-1}(k|k-1) \left[\Theta(k) - \hat{\Theta}(k|k-1)\right], \end{aligned} \tag{5.22}$$

where the parameters

$$P = V_{\varphi}^{-1} \tag{5.23}$$

$$\Lambda = v_y - v_{\varphi y}' P v_{\varphi y} \tag{5.24}$$

$$\hat{\Theta} = P v_{\varphi y} \tag{5.25}$$

$$\sigma = \frac{\Lambda}{\nu - 2} \tag{5.26}$$

have the following meaning:

P is the inverse of the information matrix,

Λ is a non-negative *remainder after the least squares (LS) estimation*,

$\hat{\Theta}$ is the *LS estimate* of the system parameters and

σ is an *estimate* (mean value) of covariance of noise on the system output.

5.2 Remarks on notation

1. Since we shall work only with estimate in the following text, there is no need to differ explicitly between the estimate and the real value of parameters and we can drop the 'hat' sign out to simplify the notation.

2. Following the conventions, introduced in Chapter 3, we shall often drop the time index out and use the following symbols to represent the position of the variables in updates:

   - 'tilde' ($\tilde{\bullet}$) for a variable before the update, e.g. $\tilde{P} = P(k|k-1)$,
   - 'bar-tilde' ($\bar{\tilde{\bullet}}$) for a variable after the data update, but before the time update, e.g. $\bar{\tilde{P}} = P(k|k)$, and
   - 'bar' ($\bar{\bullet}$) for a variable after the time update, e.g. $\bar{P} = P(k+1|k)$.

5.3 Regularized update of sufficient statistics

The probabilistic functions, entering the Bayes recursion, are updated in each step of identification by the data gathered from the system. So far we have introduced the self-reproducing probability distributions to describe these functions. A logical continuation of this introduction is to give the recursive formulae for the data and time update of the statistics determining these distributions.

5.3.1 Data update

The data update of the information matrix is performed using the following formula:

$$\bar{\tilde{V}} = \tilde{V} + \begin{bmatrix} \varphi \\ y \end{bmatrix} \begin{bmatrix} \varphi \\ y \end{bmatrix}'. \tag{5.27}$$

The parameter estimates may be computed using the formulae (5.23) and (5.25):

$$\Theta = V_{\varphi}^{-1} v_{\varphi y}.$$

This relation holds for variables both before and after the update.

The following relations hold for the data update of the parameters of the GiW-distribution expressed in the form (5.21):

$$\zeta \quad = \quad \varphi' \tilde{P} \varphi \tag{5.28}$$

$$\begin{aligned}
\varepsilon &= y - \tilde{\Theta}'\varphi && (5.29)\\
\kappa &= -(1+\zeta)^{-1}\tilde{P}\varphi && (5.30)\\
\bar{\tilde{\Theta}} &= \tilde{\Theta} - \kappa\varepsilon && (5.31)\\
\bar{\tilde{P}} &= \tilde{P} - (1+\zeta)\kappa\kappa' && (5.32)\\
\bar{\tilde{\Lambda}} &= \tilde{\Lambda} + \frac{\varepsilon^2}{1+\zeta} && (5.33)\\
\bar{\tilde{\nu}} &= \tilde{\nu} + 1 && (5.34)
\end{aligned}$$

where

ε is an *a-priori prediction error* – a difference between the output value, predicted from the *a-priori estimate* of parameters and the real system output, and

κ is the *Kalman gain* vector, weighting ε in the update of Θ.

Usually, κ is introduced without the minus sign (with a corresponding change in the update formula of Θ). We use the minus sign purely for algorithmic reasons of the square-root implementation (Sec. 5.5).

5.3.2 Initial values of the distribution parameters

A usual choice of the initial values is

$$\begin{aligned}
V(1|0) &= \eta I, \;\; \eta > 0,\\
\nu(1|0) &= 2 + \mu, \;\; \mu > 0,
\end{aligned}$$

where η and μ are small positive numbers.

The initial value of the information matrix may be any symmetric positive definite matrix.

For the statistics (5.28–5.33), the usual initial values are:

$$\begin{aligned}
P(1|0) &= (1/\eta)I, \;\; \eta > 0,\\
\Theta(1|0) &= 0,\\
\Lambda(1|0) &= \omega, \;\; \omega > 0,\\
\nu(1|0) &= 2 + \mu, \;\; \mu > 0,
\end{aligned}$$

where η, ω and μ are small positive numbers again.

The initial value of the covariance matrix may be any symmetric positive definite matrix. The initial value of parameter estimate Θ may be generally arbitrary and that of Λ may be any small positive number [23].

The above given choice represents in both cases almost uniform prior distribution (i.e. total uncertainty about the parameter estimate).

A suitable choice may also be the information from the preliminary analysis of the system [24], if available.

5.3.3 Regularized time update

Let us recall that in Section 5.1.1, we used formula (5.11) for the time update of the system model, with an alternative c.p.d.f. (5.10) chosen by user.

Using the GiW-distribution, the alternative distribution p_1 (5.10) is determined by parameters

$$V_\varphi^* \text{ and } \Theta^*. \tag{5.35}$$

Let us denote the function $J(\Theta(k)|k-1)$ in the exponent of the alternative GiW-distribution (5.22), using the star-denoted parameters, as J^*:

$$\begin{aligned} J^*(\Theta(k)) &= \begin{bmatrix} -\Theta(k) \\ 1 \end{bmatrix}' V^* \begin{bmatrix} -\Theta(k) \\ 1 \end{bmatrix} \\ &= \Lambda^* + [\Theta(k) - \Theta^*]' (P^*)^{-1} [\Theta(k) - \Theta^*] . \end{aligned} \tag{5.36}$$

Expressed using the cost functions J and J^*, the weighting of the probability distribution functions (5.11) reads:

$$J(\Theta(k+1)|k) = \lambda(k) J(\Theta(k)|k) + (1 - \lambda(k)) J^*(\Theta(k)). \tag{5.37}$$

With

$$v_{\varphi y}^* = V_\varphi^* \Theta^*, \tag{5.38}$$

derived from (5.25), we can write the time-update formulae for V_φ, $v_{\varphi y}$ and ν:

$$\begin{aligned} \bar{V}_\varphi &= \lambda \bar{\tilde{V}}_\varphi + (1 - \lambda) V_\varphi^*, \\ \bar{v}_{\varphi y} &= \lambda \bar{\tilde{v}}_{\varphi y} + (1 - \lambda) v_{\varphi y}^* \\ &= \lambda \bar{\tilde{v}}_{\varphi y} + (1 - \lambda) V_\varphi^* \Theta^*, \\ \bar{v}_y &= \lambda \bar{\tilde{v}}_y + (1 - \lambda) v_y^*, \\ \bar{\nu} &= \lambda \bar{\tilde{\nu}} + (1 - \lambda) \nu^*, \end{aligned} \tag{5.39}$$

where λ stands for $\lambda(k)$.

The updated vector of regression parameters is computed from

$$\bar{\Theta} = \bar{V}_{\varphi}^{-1}\bar{v}_{\varphi y}. \tag{5.40}$$

Using the parameters of the alternative form of the GiW-distribution (5.23, 5.25), the time update formulae have the following form:

$$\begin{aligned} \bar{P}^{-1} &= \lambda\bar{\tilde{P}}^{-1} + (1-\lambda)(P^*)^{-1} \\ \bar{\Lambda} &= \lambda\bar{\tilde{\Lambda}} + (1-\lambda)\Lambda^* \\ A &= \left\{I + \left(\frac{1}{\lambda} - 1\right)\bar{\tilde{P}}(P^*)^{-1}\right\}^{-1} \\ \bar{\Theta} &= A\bar{\tilde{\Theta}} + (I-A)\Theta^* \end{aligned} \tag{5.41}$$

Remarks.

- As was stated in section 5.1.1, setting the alternative c.p.d.f. $p^*(\mu(k+1)|k)$ to a non-informative one, we get exponential forgetting as a special case of the regularized forgetting. This corresponds to setting $V^* \to 0$ in (5.36), with a corresponding effect in (5.37), (5.39) and (5.41).

- Now, the effect of regularization on numerical properties of the identification algorithm can be clearly seen: using the exponential forgetting and with a poor or no information contained in the data samples, the information matrix V would lose the accumulated information very quickly ($V \to 0$ or, equally, $P \to \infty$) during the time update (5.39). This would result in numerical problems with inversion of V_{φ} when computing the parameter estimates (5.40).

  Using regularization, however, the resulting information matrix is prevented from the rank deficiency by adding the full rank matrix V^* in the time update.

- The Λ parameter is usually, as unimportant for Θ estimation, not computed. If required, the time update of this variable is computed from (5.41). In that case, a possible choice of the alternative parameter is $\Lambda \to 0+$. Choosing also $\nu^* \to 2+$ in (5.39), the value of estimate of covariance of noise on the system output (5.26) is preserved during the time update.

  In this thesis, it will only be needed to compute Λ in implementation of the cautious control strategy (Sec. 8.9).

5.3.4 An alternative view of the regularized time update

We can differentiate two phases in the time-update equations for V_φ and $v_{\varphi y}$: in the first phase, exponential forgetting is applied to the parameters after the data update:

$$\begin{aligned} \breve{V}_\varphi &= \lambda \bar{\tilde{V}}_\varphi, \\ \breve{v}_{\varphi y} &= \lambda \bar{\tilde{v}}_{\varphi y}, \end{aligned} \tag{5.42}$$

or, equally,

$$\breve{P} = \frac{1}{\lambda} \bar{\tilde{P}}, \tag{5.43}$$

where the 'breve' sign ($\breve{V}$) denotes the intermediate result.

In the second phase, the regularizing parameter, weighted by $(1-\lambda)$, is added to the intermediate result:

$$\begin{aligned} \bar{V}_\varphi &= \breve{V}_\varphi + (1-\lambda) V_\varphi^*, \\ \bar{v}_{\varphi y} &= \breve{v}_{\varphi y} + (1-\lambda) v_{\varphi y}^*, \end{aligned} \tag{5.44}$$

or

$$\begin{aligned} \bar{P} &= (\breve{P}^{-1} + (1-\lambda)(P^*)^{-1})^{-1}, \\ \breve{A} &= [I + (1-\lambda)\breve{P}(P^*)^{-1}]^{-1}, \\ \bar{\Theta} &= \breve{A}\breve{\Theta} + (1-\breve{A})\Theta^*. \end{aligned} \tag{5.45}$$

The only difference between the auxiliary matrix A, introduced by (5.41), and $\breve{A}$, used in the above formulae is that $\breve{A}$ uses the parameters after the phase of exponential forgetting, while A those immediately after data update.

This view of regularization as addition of alternative parameters to the result after the exponential forgetting will be extensively used in the description of the block-regularized identification (Section 5.4).

5.3.5 Rules for selection of Θ^*, V_φ^* and λ

While the identification algorithm itself does not have any special requirements on the selection of the alternative parameters (except that the V_φ^* matrix has to be a symmetric positive definite matrix), we should consider their influence on the control application (especially on the stability of control).

From the Bayesian viewpoint, the *parameters of the alternative distribution determine the shape of the a-posteriori distribution in the limit case when the processed data does not bear any information about the identified system.* It follows that if the input data contains information, the effect of the alternative distribution should be insignificant. The parameters of this distribution should also reflect the available a-priori information about the system [24].

If we do not have any a-priori information, the estimate of the regression parameters Θ should keep its last value identified from an informative data.

The weighting coefficient λ determines, together with the matrix V_φ^*, the speed of tracking parameters on one hand and the noise sensitivity on the other hand. An appropriate choice is a value equal to that used in the exponential forgetting, or slightly smaller.

We can summarize the possible choices into the following items:

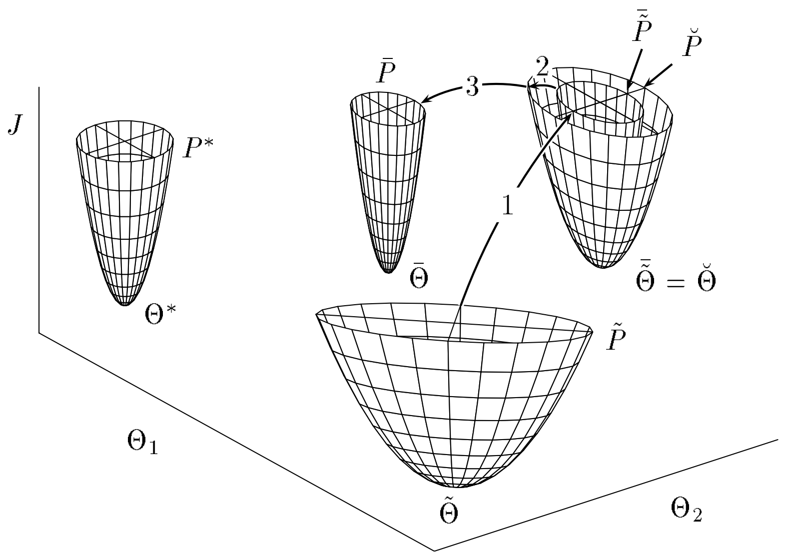
- If we do not have any information from the a-priori analysis of the identified system, and we want to track the variations of the system parameters, a suitable choice may be to use the last regularized estimate $\bar{\Theta}$ for Θ^* and a diagonal (or nearly-diagonal) matrix with the diagonal elements lesser in an order of magnitude than those of V_φ for V^*.

  With these settings, $\bar{\Theta}$ follows the changes of parameters as long as the estimator gets new information from the identified system. If the input data contains only poor information or no information at all, $\bar{\Theta}$ keeps the value computed from the last informative data, until new information is received. Of course, it is influenced by the noise on the system output, since, due to the less informative character of the alternative distribution, the noise sensitivity of the estimator increases, but with a suitable choice of the forgetting factor and of the parameters of the alternative distribution, the estimate is still reasonably stable (Sec. 6.5).

- If we have prior information about the identified system, and we want to use this prior information in the case of non-informative data (we do not expect the system to evolve, or, even if the system evolves, we consider for some reason the parameters using the a-priori information to be the best choice for the case of non-informative data), we should use this prior information for the alternative distribution.

5.3.6 Graphical illustration of regularization

To give a better idea of the regularization process, the development of the parameter estimates Θ and of the covariance matrix P (5.22–5.25) during one step of the regularized identification is drawn in Figure 5.1.



1: Data update 2: Exponential forgetting 3: Regularization

Figure 5.1: One step of regularized identification

The figure is plotted for a two dimensional parameter vector $\Theta = [\Theta_1 \ \ \Theta_2]$, the vertical axis represents the loss function J (5.22):

$$J = \Lambda + \left[\Theta - \hat{\Theta}\right]' (P)^{-1} \left[\Theta - \hat{\Theta}\right],$$

where Θ is some sample of parameters (either $\Theta(k)$ or $\Theta(k+1)$, dependent on whether before of after the time update) and $\hat{\Theta}$ stands for the parameter estimate in various phases of the identification step ($\tilde{\Theta}$, $\bar{\tilde{\Theta}}$, $\breve{\Theta}$ or $\bar{\Theta}$, dependent on where we are in the step).

The elliptical paraboloids represent the cost function in the successive phases of the identification step. Their cross-section reflects the shape of the covariance matrix, their placement is determined by the value of parameter estimate in the respective phases.

The arrows in the figure denote the successive phases of the identification step:

1. data update: $\tilde{P} \rightarrow \bar{\tilde{P}}$, $\tilde{\Theta} \rightarrow \bar{\tilde{\Theta}}$,
2. exponential forgetting: $\breve{P} = \frac{1}{\lambda}\bar{\tilde{P}}$, $\breve{\Theta} = \bar{\tilde{\Theta}}$,
3. regularization — addition of the alternative parameters:

$$\begin{aligned} \bar{P} &= (\breve{P}^{-1} + (1-\lambda)(P^*)^{-1})^{-1}, \\ \breve{A} &= [I + (1-\lambda)\breve{P}(P^*)^{-1}]^{-1}, \\ \bar{\Theta} &= \breve{A}\breve{\Theta} + (1-\breve{A})\Theta^*. \end{aligned}$$

For simplicity, we do not display the remainder after the least squares Λ in the graph.

5.4 Block-regularized forgetting

In this section, we shall explain the block-regularization mechanism [19], which will be used later in the systolic implementation of the identification algorithm (Chapter 6).

Using the block-regularization, the two phases of the time update (Section 5.3.4) are separated and the first phase (exponential forgetting) is performed together with the data updates. The second phase (addition of the alternative parameters) is, under some reasonable assumptions, accumulated and performed all at once after several data updates. The reason for using the block-regularization is that the properties of the algorithm, required by the systolic implementation (Section 6.3) (pipelineability and communications locality) [22], could not be achieved with the standard way of regularization [19, 21]. The requirement of pipelining determines also that the length of accumulation be at least n data samples ($n = \partial\Theta$).

Performing the regularization only after we have processed at least n data samples, we can achieve decrease of the sampling rate of the resulting systolic array to 1/2 of the value without regularization in the worst case, independent of the system order.

5.4.1 Derivation of the block regularized forgetting

Reference parameters constant over N periods

To be able to accumulate the time update steps, we shall suppose that Θ^* and V_φ^* are constant over N steps of identification, $N \geq n$, $n = \partial\Theta$:

$$\begin{aligned} V_\varphi^* &= V_\varphi^*(k) = V_\varphi^*(k+1) = \ldots = V_\varphi^*(k+N-1), \\ \Theta^* &= \Theta^*(k) = \Theta^*(k+1) = \ldots = \Theta^*(k+N-1), \end{aligned} \tag{5.46}$$

where N is the length of the data block (5.47).

Splitting the two phases of data update apart

To explain the mechanism of the block-regularization, let us consider first *one complete step* of data and time update (5.27, 5.39) of matrix V_φ with regularization using *constant matrix* V_φ^*:

$$V_\varphi(k+1|k) = \lambda(k)\left[V_\varphi(k|k-1) + \varphi(k)\varphi'(k)\right] + (1-\lambda(k))V_\varphi^*$$

and *two steps*:

$$\begin{aligned} & V_\varphi(k+2|k+1) = \\ &= \lambda(k+1)\left[V_\varphi(k+1|k) + \varphi(k+1)\varphi'(k+1)\right] + (1-\lambda(k+1))V_\varphi^* = \\ &= \lambda(k+1)\{\lambda(k)[V_\varphi(k|k-1) + \varphi(k)\varphi'(k)] + \\ &\quad +\varphi(k+1)\varphi'(k+1)\} + (1-\lambda(k+1))(\lambda(k)+1)V_\varphi^* \end{aligned}$$

The formulae may be split into two parts: the first, corresponding to the standard exponential forgetting, and the second, corresponding to the addition of alternative parameters. It means that (with constant alternative parameters) using the standard exponential forgetting over N periods of the identification and then adding the alternative parameters with an appropriate weight, we get exactly the same result, as if using the regularized time update in each step of identification.

Let us write the whole procedure in an algorithmic form. For brevity, we "subtract" $(k-1|k-1)$ in the time index of information matrix, writing $(1|0)$ instead of $(k|k-1)$ and $(N+1|N)$ instead of $(k+N|k+N-1)$, and $(k-1)$ in the time index of λ. We also use the sign ($\check{\bullet}$) to denote the information matrix during the data updates and exponential forgetting, but before the accumulated regularization:

1. Standard exponential update over N periods:

$$
\begin{array}{lll}
\breve{V} := V(1|0) & a) & \\
\breve{\Theta} := \breve{V}_{\varphi}^{-1}\breve{v}_{\varphi y} & b) & \\
\lambda(1,N) := 0 & c) & \\
\underline{for}\ \ i := 1\ \underline{to}\ N & & \\
\quad \underline{if}\ i = N - n + 1 & & \\
\qquad \Theta^{*} := \breve{\Theta} & d) & (5.47) \\
\quad \underline{end} & & \\
\quad \breve{V} := \lambda(i)\left\{\breve{V} + [\varphi'(i)\ y(i)]'[\varphi(i)\ y(i)]\right\} & e) & \\
\breve{\Theta} := \breve{V}_{\varphi}^{-1}\breve{v}_{\varphi y} & f) & \\
\lambda(1,N) := \lambda(i)\lambda(1,N) + (1 - \lambda(i)) & g) & \\
\underline{end} & &
\end{array}
$$

2. Accumulated regularization in the N-th period:

$$
\begin{array}{ll}
v_{\varphi y}^{*} := V_{\varphi}^{*}\Theta^{*} & a) \\
V_{\varphi}(N+1|N) := \breve{V}_{\varphi} + \lambda(1,N)V_{\varphi}^{*} & b) \\
v_{\varphi y}(N+1|N) := \breve{v}_{\varphi y} + \lambda(1,N)v_{\varphi y}^{*} & c) \\
\bar{\Theta}(N+1|N) := V_{\varphi}^{-1}(N+1|N)v_{\varphi y}(N+1|N) & d)
\end{array}
\qquad (5.48)
$$

where $\lambda(1,N) > 0$ is an *accumulated forgetting factor*.

Remarks

- The *accumulated forgetting factor* may be also computed from a one shot formula

$$\lambda(1,N) = \sum_{i=1}^{N}\prod_{j=i+1}^{N}\lambda(j)(1-\lambda(i)), \qquad (5.49)$$

$$\prod_{j=N+1}^{N}\lambda(j) = 1.$$

- For constant weighting factor $\lambda(i) = \lambda,\ i = 1, \ldots, N$, the above formula simplifies to

$$\lambda_N = \sum_{i=0}^{N-1} [\lambda^i(1-\lambda)] = 1 - \lambda^N, \tag{5.50}$$

where λ_N denotes the accumulated forgetting factor computed from constant λ.

5.4.2 Choice of Θ^* for the block-regularized forgetting

In our systolic implementation (5.47 d), we use the parameter estimate computed n steps before the end of the data update block for the regularizing parameters Θ^* to simplify necessary control mechanisms (Subsec. 6.3.3).

With this choice, Θ is identified $N - n$ steps at the beginning of a data block using standard exponential forgetting and the initial conditions from the beginning of the data block. Experience shows that for small $N - n$, this does not have a significant influence on the quality of identification.

5.4.3 Comparison of standard and block-accumulated regularization

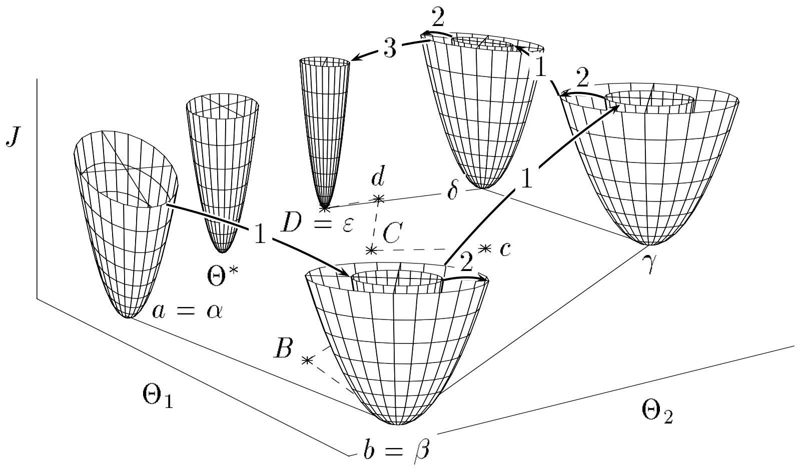
To give better idea of the mechanism, a comparison of the trajectories of parameter estimates, obtained using both the standard and the block-accumulated regularization, is presented in the graph in Fig. 5.2. For the block-regularized identification, it shows also the development of the cost function J (5.22).

The graph is drawn for a block of three data samples.

The letters of the Latin alphabet denote the trajectory resulting from the standard regularization; the lower-case letters are used to refer the position of the parameter estimates after the data update (5.27), the upper-case letters are used to refer that after the regularization (5.39).

The Greek letters denote the tracking trajectory resulting from the block-regularized identification; the arrows show the development of the covariance matrix P through one block of the block-regularized identification — number 1 denotes the data update (5.27), number 2 the exponential forgetting (5.42) ($1 + 2 =$ (5.47 e)), number 3 denotes the addition of the alternative parameters (5.48 b–c).

Both trajectories start at the point $a = \alpha$ and finish at the point $D = \varepsilon$.



1: Data update 2: Exponential forgetting 3: Accumulated regularization

Figure 5.2: Comparison of standard and block-accumulated regularization

Note: The referenced equations work with the information matrix V. The shape of the elliptical horizontal cross-section of the paraboloids in this graph and in Fig. 5.1 is, however, determined by the covariance matrix $P = V_\varphi^{-1}$.

5.5 A square-root RLS algorithm

Having explained the principle of the regularized RLS identification (5.28–5.32, 5.41) and of the mechanism of the block-regularization, we shall continue by description of the square-root implementation of the block-regularized algorithm [19].

The reasons for the square-root implementation, which works with a square-root of the information or of the covariance matrix, are that it prevents this matrix from a loss of positive definiteness due to the roundoff errors in the updates, that it guarantees its symmetry and finally that it is suitable for efficient pipelined parallel implementation.

In this section we shall discuss two topics:

- The square-root implementation of the RLS algorithm [43] (5.28–5.32) (which is often called *inverse update*) with the first phase of the regularized time update (exponential forgetting) (5.43), and

- Square-root implementation of the second phase of the regularized time update — of addition of the alternative information matrix $V^* = (P^*)^{-1}$.

The solution of the second topic for the inverse updated RLS algorithm is an original contribution of this thesis.

5.5.1 Square-root decomposition of the covariance matrix

Let us introduce the triangular square-root decomposition of the covariance matrix by a formula:

$$P = RR', \tag{5.51}$$

where R is an upper triangular matrix.

The decomposition (5.51) guarantees that the symmetry and positive definiteness of P will always be preserved, provided that matrix R is of full rank.

Let us further introduce an auxiliary vector a as a product

$$a = -\tilde{R}'\varphi. \tag{5.52}$$

Applying the decomposition in the formulae (5.28), (5.30), (5.32) and (5.43), we get the following relations for the data update:

$$\begin{aligned} \zeta &= \varphi'\tilde{P}\varphi = a'a, & (5.53)\\ \kappa &= -(1+\zeta)^{-1}\tilde{P}\varphi = (1+\zeta)^{-1}\tilde{R}a, & (5.54)\\ \bar{\tilde{R}}\bar{\tilde{R}}' &= \tilde{R}\tilde{R}' - (1+\zeta)\kappa\kappa', & (5.55) \end{aligned}$$

and for the time update:

$$\bar{R}\bar{R}' = \frac{1}{\lambda}\bar{\tilde{R}}\bar{\tilde{R}}'. \tag{5.56}$$

Introducing

$$s = 1 + aa' = 1 + \zeta, \tag{5.57}$$

we can rearrange the formulae (5.53–5.54) into form of single matrix equation:

$$\begin{bmatrix} s^{\frac{1}{2}} & s^{\frac{1}{2}}\kappa' \\ 0 & \bar{\tilde{R}}' \end{bmatrix}' \begin{bmatrix} s^{\frac{1}{2}} & s^{\frac{1}{2}}\kappa' \\ 0 & \bar{\tilde{R}}' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ a & \tilde{R}' \end{bmatrix}' \begin{bmatrix} 1 & 0 \\ a & \tilde{R}' \end{bmatrix} \tag{5.58}$$

and rewrite the expression for the remainder after the least squares estimation (5.33) in a square-root form:

$$(\bar{\tilde{\Lambda}}^{\frac{1}{2}})^2 = (\tilde{\Lambda}^{\frac{1}{2}})^2 + \frac{(\varepsilon)^2}{(s^{\frac{1}{2}})^2} \tag{5.59}$$

5.5.2 Data update of the covariance matrix decomposition

Using the QR update (Chapter 1), [13, 38], the data update may be performed directly on the above introduced decomposition R of the covariance matrix. Moreover, the QR transformation is suitable for a pipelined systolic implementation [38]. In this subsection, we shall explain the principle of this transformation.

Givens rotations

Let us have a set of n $(n = \partial\Theta)$ orthogonal rotation matrices (Givens rotations [12]) Φ_i:

$$\Phi_i = \begin{array}{c c} \left[\begin{array}{ccc} \cos\phi_i \cdots\cdots & \sin\phi_i \cdots\cdots & \\ \vdots \quad I & \vdots & \\ -\sin\phi_i \cdots\cdots & \cos\phi_i \cdots\cdots & \\ \vdots & \vdots & I \end{array}\right] & \begin{array}{c} \cdots 1 \\ \\ \cdots i \\ \\ \end{array} \\ \begin{array}{ccc} 1 & \quad\quad i & \quad \end{array} & \end{array} \tag{5.60}$$

$$(\Phi_i)'\Phi_i = I, \; i = 1, \ldots, n.$$

where the rotation angle ϕ_i is computed so that the matrix zeros i-th element of vector a with respect to the first element of the same column in the following equation:

$$\begin{aligned} \begin{bmatrix} s^{\frac{1}{2}} & s^{\frac{1}{2}}\kappa' \\ 0 & \bar{\tilde{R}}' \end{bmatrix} &= \Phi_n \cdots \Phi_2 \Phi_1 \begin{bmatrix} 1 & 0 \\ a & \tilde{R}' \end{bmatrix} \\ &= Q \begin{bmatrix} 1 & 0 \\ a & \tilde{R}' \end{bmatrix}, \end{aligned} \tag{5.61}$$

$$Q \equiv \Phi_n \cdots \Phi_2 \Phi_1, \quad \partial R = (n, n),$$

Premultiplying both sides of this equation by their transpositions we get, due to the orthogonality of matrices Φ_i (and following orthogonality of their product Q), the formula (5.58). It means that using the Givens rotations, it is possible to update the covariance matrix in a square root form.

Computation of rotation angle ϕ

Denoting the element $s^{\frac{1}{2}}$ (5.61) before the i-th elementary rotation as ${}_is^{\frac{1}{2}}$, with ${}_1s^{\frac{1}{2}} = 1$, the rotation angle ϕ_i is given as

$$\phi_i = \arctan \frac{a_i}{{}_is^{\frac{1}{2}}}. \tag{5.62}$$

In practice, however, we usually do not compute the rotation angle itself, but the sine and cosine of this angle:

$$\cos\phi_i = \frac{a_i}{\sqrt{({}_is^{\frac{1}{2}})^2 + (a_i)^2}} \tag{5.63}$$

$$\sin\phi_i = \frac{{}_is^{\frac{1}{2}}}{\sqrt{({}_is^{\frac{1}{2}})^2 + (a_i)^2}} \tag{5.64}$$

5.5.3 Exponential forgetting

To perform the exponential forgetting, we left-multiply (5.61) by matrix Ω:

$$\begin{bmatrix} s^{\frac{1}{2}} & s^{\frac{1}{2}}\kappa' \\ 0 & \bar{R}' \end{bmatrix} = \Omega \begin{bmatrix} s^{\frac{1}{2}} & s^{\frac{1}{2}}\kappa' \\ 0 & \tilde{\bar{R}}' \end{bmatrix}, \tag{5.65}$$

where

Ω is a diagonal matrix:

$$\Omega = \mathrm{diag}\{1\ (1/\lambda^{\frac{1}{2}})I_n\},$$

I_n is a unit matrix, $\partial I_n = (n, n)$

5.5.4 Update of remainder after the least squares estimation

For completeness, we give the formula for the square-root update of the remainder after the least squares estimation Λ (5.59):

$$\begin{bmatrix} \breve{\Lambda}^{\frac{1}{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} \lambda^{\frac{1}{2}} & \\ & 1 \end{bmatrix} \begin{bmatrix} \cos\phi_\Lambda & \sin\phi_\Lambda \\ -\sin\phi_\Lambda & \cos\phi_\Lambda \end{bmatrix} \begin{bmatrix} \tilde{\Lambda}^{\frac{1}{2}} \\ \varepsilon/s^{\frac{1}{2}} \end{bmatrix}, \tag{5.66}$$

where $\sin\phi_\Lambda$ and $\cos\phi_\Lambda$ are computed from

$$\cos\phi_\Lambda = (\varepsilon/s^{\frac{1}{2}})/\sqrt{(\tilde{\Lambda}^{\frac{1}{2}})^2 + (\varepsilon/s^{\frac{1}{2}})^2},$$
$$\sin\phi_\Lambda = \tilde{\Lambda}^{\frac{1}{2}}/\sqrt{(\tilde{\Lambda}^{\frac{1}{2}})^2 + (\varepsilon/s^{\frac{1}{2}})^2}.$$

Since Λ is not needed if we do not count with the uncertainty of parameter estimates (cautious control strategy - Sec. 8.9), it will not be treated in the following text, until it is really used.

5.5.5 Update of parameters

Also the data update of parameters (5.31) may be expressed as a matrix transformation.

Let us join the vector of updated parameters Θ as an additional bottom row to the composed matrices in (5.61). The parameters may be updated by multiplication by a non-orthogonal transformation matrix G:

$$\begin{bmatrix} s^{\frac{1}{2}} & s^{\frac{1}{2}}\kappa \\ 0 & \bar{\tilde{R}}' \\ 0 & \bar{\tilde{\Theta}}' \end{bmatrix} = G \begin{bmatrix} s^{\frac{1}{2}} & s^{\frac{1}{2}}\kappa \\ 0 & \tilde{R}' \\ \varepsilon & \tilde{\Theta}' \end{bmatrix}, \tag{5.67}$$

where:

$$G = \begin{array}{cl} \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & I & \vdots \\ -\frac{\varepsilon}{\sqrt{s}} & \cdots & 1 \end{bmatrix} & \begin{array}{l} \cdots 1 \\ \\ \cdots n+2 \end{array} \\ \begin{array}{ccc} 1 & \quad & n+2 \end{array} & \end{array} \tag{5.68}$$

Since the transformation matrix G modifies only the bottom row of (5.67) and does not influence the entries of the matrices in (5.61), it may be combined with the orthogonal transformation matrix Q without affecting the relation between the equations (5.61) and (5.58).

5.5.6 Summary of the inverse updated RLS

Let us summarize now the steps of the square-root data update, including the exponential forgetting the update of parameters and:

Step 1: computation of a (5.52) and ε (5.29) (matrix-vector multiplication):

$$\begin{bmatrix} a \\ \varepsilon \end{bmatrix} = \begin{bmatrix} \tilde{R}' & 0 \\ \tilde{\Theta}' & 1 \end{bmatrix} \begin{bmatrix} -\varphi \\ y \end{bmatrix} \tag{5.69}$$

Step 2: data update of R (5.61), exponential forgetting (5.65), update of parameters (5.67):

$$\begin{bmatrix} s^{\frac{1}{2}} & s^{\frac{1}{2}}\kappa \\ 0 & \bar{R}' \\ 0 & \bar{\Theta}' \end{bmatrix} = G\Omega_1 Q_1 \begin{bmatrix} 1 & 0 \\ a & \tilde{R}' \\ \varepsilon & \tilde{\Theta}' \end{bmatrix} \tag{5.70}$$

where Ω_1 and Q_1 are matrices Ω and Q extended to be compatible with the matrix G:

$$\Omega_1 = \mathrm{diag}\{\Omega \;\; 1\}, \quad Q_1 = \mathrm{diag}\{Q \;\; 1\}. \tag{5.71}$$

5.5.7 Regularization

As the last topic in this section, we shall show that the addition of the alternative parameters (5.44, 5.45, 5.48) may be, in a square-root form, implemented as an input of "alternative" regularizing data [19]. We shall show that the same data may be used both for the identification algorithm working with the information matrix V and for that working with covariance P.

Factorization of the alternative information matrix V^*

Let us recall the formula of the time update of information matrix $\breve{V}$ (5.44):

$$\bar{V} = \breve{V} + (1-\lambda)V^*.$$

We can uniquely factorize the matrices $\breve{V}_\varphi$ and V^*_φ as a product of an upper triangular matrix and its transposition (in the same way as we factorized the covariance matrix P (5.51)):

$$V_\varphi = U'U \tag{5.72}$$
$$V^*_\varphi = (U^*)'U^* \tag{5.73}$$

where U and U^* are upper triangular matrices of dimensions (n, n).

Let us further introduce the following relations:

$$\begin{bmatrix} V_\varphi & v_{\varphi y} \\ v'_{\varphi y} & v_y \end{bmatrix} = \begin{bmatrix} U & u \\ 0 & \bullet \end{bmatrix}' \begin{bmatrix} U & u \\ 0 & \bullet \end{bmatrix} \tag{5.74}$$

$$\begin{bmatrix} V^*_\varphi & v^*_{\varphi y} \\ (v^*_{\varphi y})' & v^*_y \end{bmatrix} = \begin{bmatrix} U^* & u^* \\ 0 & \bullet^* \end{bmatrix}' \begin{bmatrix} U^* & u^* \\ 0 & \bullet^* \end{bmatrix}, \tag{5.75}$$

where $\bullet$ and $\bullet^*$ represent scalar terms, unimportant for the regularization process.

Using the above relations and (5.25), we can find the following formulae:

$$u = U\Theta = U^{-T} v_{\varphi y}, \tag{5.76}$$

$$u^* = U^*\Theta^* = (U^*)^{-T} v^*_{\varphi y}. \tag{5.77}$$

The decomposition of the alternative information matrix can be expressed as a sum of data dyads (both sides of this formula are equal except for v^*_y):

$$\begin{bmatrix} V^*_\varphi & v^*_{\varphi y} \\ (v^*_{\varphi y})' & v^*_y \end{bmatrix} \doteq \sum_{i=1}^{n} \begin{bmatrix} U^*_i & u^*_i \end{bmatrix}' \begin{bmatrix} U^*_i & u^*_i \end{bmatrix}, \tag{5.78}$$

where

$$\begin{bmatrix} U^*_i & u^*_i \end{bmatrix} \hat{=} \; i\text{-th row of the matrix } \begin{bmatrix} U^* & u^* \\ 0 & \bullet^* \end{bmatrix}. \tag{5.79}$$

Using this decomposition, we can express the addition of the regularizing matrix V^* to the information matrix $\check{V}$ in a recursive form:

$\underline{for}$ $i = 1$ $\underline{to}$ n

$$\begin{aligned} & u^*_i := U^*_i\Theta^* && a) \\ & \check{V} := \check{V} + \lambda(1,N) \begin{bmatrix} U^*_i & u^*_i \end{bmatrix}' \begin{bmatrix} U^*_i & u^*_i \end{bmatrix} = && \\ & \quad = \check{V} + \sqrt{\lambda(1,N)} \begin{bmatrix} U^*_i & u^*_i \end{bmatrix}' \sqrt{\lambda(1,N)} \begin{bmatrix} U^*_i & u^*_i \end{bmatrix} && b) \end{aligned} \tag{5.80}$$

$\underline{end}$

$$V(k+1|k) := \check{V} \qquad c)$$

Comparing the formula for the data update with exponential forgetting (5.47 e) with the relation for the recursive regularization (5.80 b), we can see, that they both have the same form — in both cases we add data dyads to the information matrix. The only difference is that in the case of data update, we apply forgetting on the updated information matrix, while in the case of regularization we apply the accumulated forgetting factor $\lambda(1, N)$ (5.49) on the added dyad. We can conclude:

Result

1. *The regularization may be performed in the same way as the data update* – when regularizing, we enter recursively the vector of regularizing "alternative" data (5.79). This vector contains one line of a triangular factor of the alternative information matrix U^*, extended by one element of the vector u^*, which is placed in the same position where y is placed in the standard data vector. This element is prepared in advance as a product $u^* = U^* \Theta^*$. In the systolic implementation, we shall prepare it while computing the data updates.

   The whole vector will be weighted by a square root of the accumulated forgetting factor $\lambda(1, N)$ (5.49).

   Since it is necessary to process n vectors of regularizing data, the regularization takes n steps.

2. Entering the same data, the time development of both the information matrix and the covariance matrix (using appropriate update algorithms) is the same (except for inversion).

   Conclusion: We can use *the same* regularizing data (decomposition of the regularizing *information matrix* V_φ^*) to update the factor of covariance matrix in the square-root RLS algorithm, as we did in this section to update the information matrix.

5.6 Summary

In this chapter, system identification was discussed.

The most important topics, presented in this chapter, were the square-root implementation of the RLS algorithm with covariance update, the mechanism of the block regularization, necessary for the systolic implementation of the regularization, and finally the square-root implementation of the regularization as an input of alternative regularizing data. The important and original conclusion is that it is possible to use the same alternative data as input for regularization, regardless of whether we work with the information matrix or with the matrix.

In the next chapter, an introduction to the technology of systolic arrays will be given and the systolic implementation of the block-regularized RLS algorithm with covariance update will be described.

Chapter 6

Systolic estimator

This chapter is devoted to the parallel implementation of the regularized RLS algorithm on a systolic array.

We shall discuss here the following topics:

- Introduction to the function principles of the systolic arrays,
- Systolic implementation of the square-root covariance filter with exponential forgetting and related problems [43], and
- Modification of the RLS systolic array to implement the regularization mechanism (the most important part of this chapter).

The main result of the chapter is the detailed description of the processing elements (PEs) of the regularized RLS array.

The reader, familiar with the function principles of the systolic arrays, may skip the first section and go directly to Sec. 6.2.

6.1 Systolic system

6.1.1 Introduction

Before we describe the systolic implementation of the inverse updated RLS identification, let us explain in brief what we understand by a systolic system.

Systolic array - a regular network of interconnected processors

By a *systolic array* we understand a regular network of mutually interconnected processing elements (PE), connected to the outside world. The network has some regular shape (e.g. row, triangle, square or trapezoid). Unlike the processors in a standard computer, the PEs in the systolic array do not use any global bus, but instead the neighbouring PEs are interconnected by point-to-point *links* in some regular fashion.

Decomposition of implemented algorithm

The algorithm, realized on a systolic array, is decomposed to the elementary operations (e.g. multiplication of elements in a vector-matrix multiplication), which are realized either in all PEs or in a group of PEs in the network (e.g. in a subarray of a triangular shape in an array of a trapezoidal shape). In each step, a PE applies the elementary operation to the partial result(s), received from the preceding (in the sense of the data flow) PE(s) and sends the result(s) to the succeeding PE(s), which will process this result in the next step (considering the above mentioned example of the vector-matrix multiplication, the result of multiplication of two elements is passed to the neighbouring processor, which adds to it the product of the subsequent elements in the next period).

The PEs in the array can implement several tasks, that are performed sequentially. For example, the array can implement a multiplication of a vector by a matrix and addition of another vector to the result — in this case the whole array performs the vector-matrix multiplication and one row of processors performs yet the vector addition.

Data pipelining

Data pipelining is a characteristic property of the systolic arrays. The data passes through the array in consecutive "waves". The data which have entered the array in the same time pass through in one "wave".

The array does not process only one data at a time. On contrary, consecutive data waves pass through the array in the same time and the PEs that are not passed by one wave in the given instant, process data that have entered the array in the different time steps.

The term "systolic" comes from some similarity between the rhythmic work of the processors and the contractions of the heart, pushing the blood to the arteries.

Synchronization of the array

Though various technologies may be used to implement the synchronization of the PEs in the array, we may generally use the following rules for the array design:

- if one PE sends the data to the another, the receiving PE must receive this data in the next step and, on the other hand,
- if the PE is waiting to receive some data from the link to process them in the current step, the PE on the other end of this link must have sent the data to the link in the previous step so that the waiting processor can continue the processing,
- if the data, that the PE sent to the link, were not received, or if the PE was waiting for data, while no data was sent to the link, the PE cannot continue its function and the array crashes.

A *formal definition* of a systolic array is given in [1].

An exact description of the synchronization issues is given in the theory of *communicating sequential processes* (CSP) [15] or in the *calculus of communicating systems* [42].

Basic performance characteristics of the systolic arrays

The performance of a parallel implementation for a recursive algorithm is in general characterized by the following figures:

Latency - number of time units (steps) which elapse from input of data until the output of the corresponding results. It is proportional to the data path from the input to the output of the array. If the latency is not the same for all elements of the input and output vector, usually the longest latency is of interest.

Sampling period is the time between two successive data inputs.

Output period is the time between two successive outputs of result. It is equal to the sampling period, if one output corresponds to one input, or to a multiple of the sampling period, if one output is computed from several inputs (e.g. in the case or regularized identification).

Speed up is the ratio of the sampling period of a sequential implementation to the sampling period of the parallel implementation.

Efficiency is defined as the ratio of the speed up to the number of PEs used to achieve this speed up.

Description of the array

To describe the systolic arrays, we use the following types of charts:

- Description of the cell (PEs), with a scheme of the cell, containing the i/o denotation and the internal registers, and with an i/o and function description,

- Scheme of the array, with the contour of the array and alignment of matrices/vectors to this contour and with a scheme of direction and time skew of the vector inputs and outputs of the array (the term of time skew will be explained in the example in the next subsection)

- Time stepping of the array function (only if suitable for better clarity of the function explanation).

6.1.2 A simple example of a systolic array — matrix-vector multiplication

In this subsection, which is intended as an introduction to the principles of systolic arrays, we shall give a trivial example to help the reader understand the above terms. Those, who are familiar with this topic, may skip this part.

We shall describe a systolic implementation of matrix-vector multiplication

$$a = -\tilde{R}'\varphi,$$

which was used in the previous chapter (5.52).

Let us recall that φ is the entering column vector, a is the resulting column vector and R' is a lower triangular matrix.

The elements of a are scalar products of the rows of $\tilde{R}'$ with the vector $-\varphi$. The scalar product a_i of vectors $\tilde{R}'_i$ (i-th row of $\tilde{R}'$) and φ is given by:

$$a_i = \sum_{j=1}^{i} \tilde{R}'_{ij}(-\varphi_j),\ i = 1 \dots n,\ n = \partial \tilde{R}'_i = \partial \varphi.$$

Systolic implementation of a vector-vector multiplication

Let us focus first on the vector-by-vector multiplication.

Beginning the multiplication from the i-th (=last) element, we can express a_i as a sum of partial result $a_{i\underline{k}}$ and of the rest of the vector multiplication:

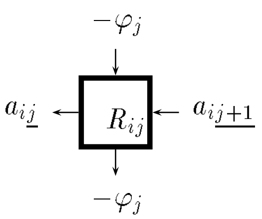
$$a_i = a_{i\underline{k}} + \sum_{j=1}^{k-1} R_{ij}(-\varphi_j), \tag{6.1}$$

$$a_{i\underline{k}} = \sum_{j=k}^{i} R_{ij}(-\varphi_j), \tag{6.2}$$

where the underlined index $\underline{k}$ is the index of elements in the last product added to the partial result.

The above formula can be implemented on a row of PEs. Each PE in this row has two inputs and two outputs and stores one element of R_i in an internal register. One i/o pair is used to pass φ_j, the second i/o pair to pass a_i. Each element of φ is processed in one time step. The PE computes the product of R_{ij} and φ_j for one j, adds the partial sum $a_{i\underline{j+1}}$ and then sends $a_{i\underline{j}}$ to the next PE. The next element will process $a_{i\underline{j}}$ in the next step. This process is repeated until the whole product is completed.

The chart bellow contains the function description of one PE in this row:



I/O assignment:
$a_{i\underline{i}} = 0$

Cell functionality:
$a_{i\underline{j}} = R_{ij}(-\varphi_j) + a_{i\underline{j+1}}$

Function of the multiplication element

Figure 6.1 shows position of the data on the entries of first two PEs in the row at the beginning of each step.

Data pipelining

The efficiency of the array in Fig. 6.1 is very low — since only one PE is actvive in each step, it is the reciprocal of the length of the multiplied vectors.

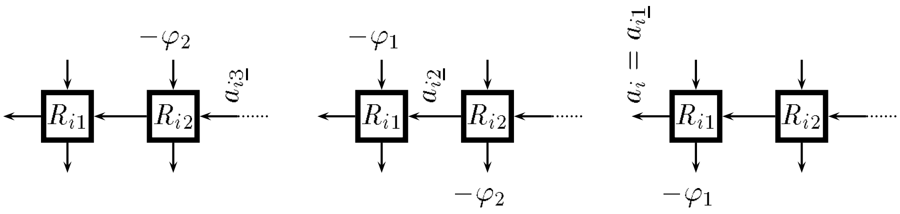


Figure 6.1: Tracing the vector-vector multiplication

Let us suppose now, that a fixed vector r (generally, it may change in time, but the changes must be synchronized with the movement of data through the array) is multiplied by an infinite sequence of vectors

$$\{\varphi(1),\ \varphi(2),\ \ldots\}$$

(this corresponds to a typical situation in real-time control, where we have to process a new data sample in each step).

$\varphi_j(k+1)$ can be entered as soon as $\varphi_j(k)$ has been processed in the j-th PE — the *data pipelining* can be used. Since each element of φ must enter the cell in the same time as the corresponding partial result, φ must enter the array with a *time skew* — the rightmost element first (provided that the product is computed from right to left), next element at the next time step, etc.

The function of the data pipelining is illustrated in Figure 6.2 by tracing the multiplication of two vectors (the index in parentheses is used to denote quantities related to the same input vector).

Here, using the data pipelining, full utilization of all PEs has been achieved.

Matrix-vector multiplication

To implement the matrix-vector multiplication, we simply stack the rows of PEs one on another, so that the outputs and inputs of φ in the neighbouring rows correspond, and connect these outputs and inputs together. As a result, we get a lower triangular array of PEs. The vector φ passes downward through this array.

Since this is just an extension of the previous example of vector-vector multiplication, the function of the PEs in the array remains exactly the

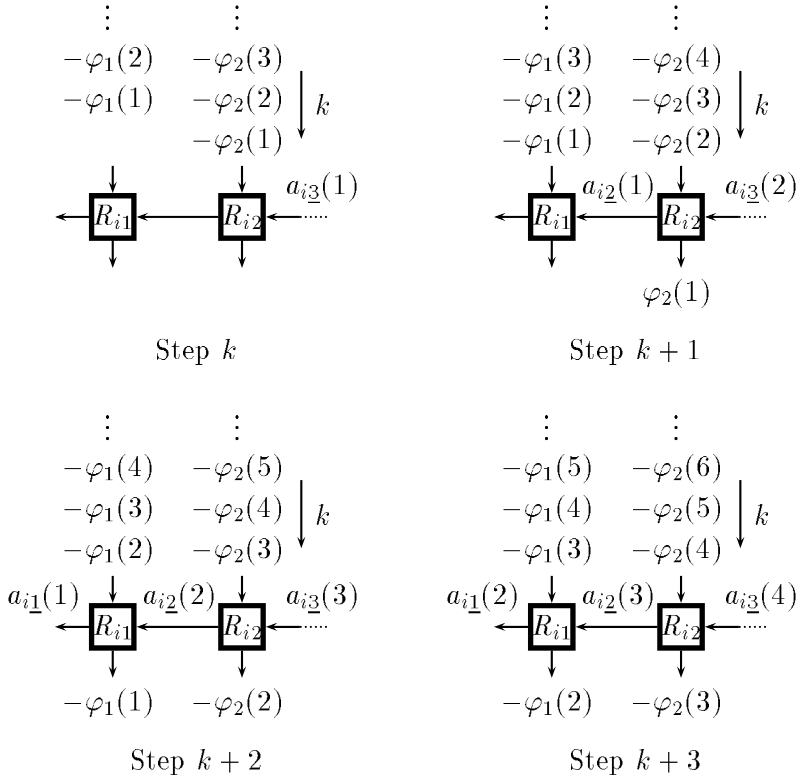


Figure 6.2: Data pipelining

same. The input vector φ enters the array through the diagonal PEs, the *whole vector* at the *same time* (it passes the rows of the array still with a time skew) and is passed downwards.

The output vector a emerges from the array on the left side, time-skewed; the upper element leaves the array first, the bottom element last.

For better idea of the data movements, we trace the function of this array in Fig. 6.3.

Schematic chart of the array

As was noticed in the introduction, we use schematic charts to describe the array, instead of a detailed scheme as given in Fig. 6.1, for example.

The chart shows the shape of the array, the direction of the input and output and the time skew of the entering/exiting vectors.

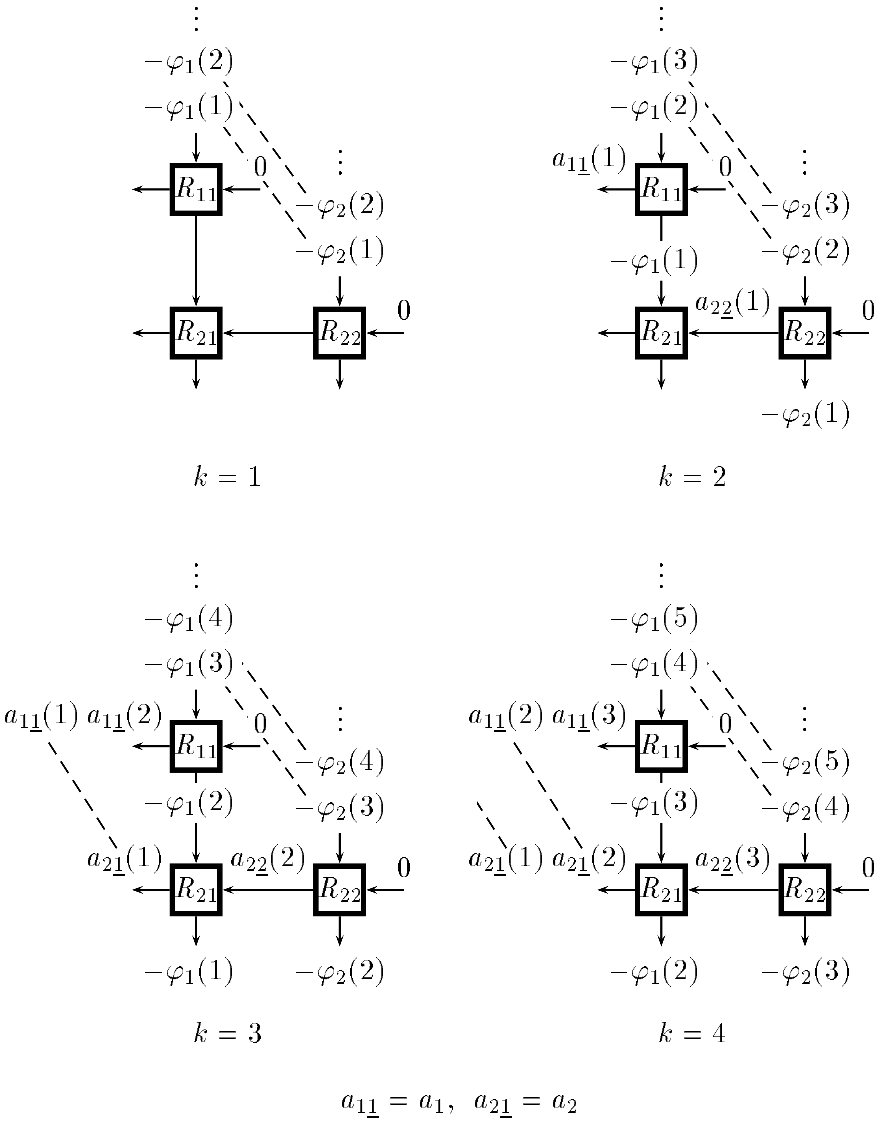


Figure 6.3: Tracing of matrix-vector multiplication

A chart for the example of scalar multiplication of two vectors on a systolic row of PEs is given in Fig 6.4.

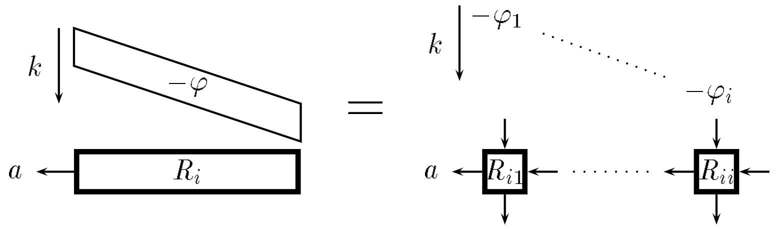


Figure 6.4: Systolic array and corresponding schematic chart

The rectangle, drawn in a thick line, shows the shape of the array. The letter R_i inside the rectangle represents the vector R_i, stored in the elements of this array.

The skewed shape above the rectangle, which is drawn in a thin line, is used to represent the input vector φ. The skew of this shape reflects the time skew of the vector.

The arrow, drawn on the left side of the entering vector, shows the direction, in which the vector enters the array.

For completeness, a chart of the array for vector-matrix multiplication is given in Fig. 6.5.

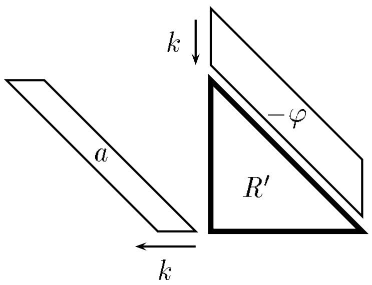


Figure 6.5: Scheme of an array for multiplication $a = R'(-\varphi)$

Latency, sampling and output period of the multiplication array

In Section 6.1.1, we have introduced latency and sampling and output period as the basic performance characteristics of a systolic array.

For the array for matrix-vector multiplication, both the *sampling* and the *output period* are independent of the length of the multiplied vector and are equal to one step of the array. The *latency* is different for different elements of the output vector a: for a_i, it is equal to i steps. Since all PEs are utilized all the time, the *efficiency* of the array is 100%.

6.1.3 Summary

In this section, we have described function principles and introduced some basic performance characteristics of systolic arrays.

6.2 Systolic implementation of the RLS algorithm

Now, we are ready to describe systolic implementation of the square root RLS algorithm (Sec. 5.5), following [39, 43]. In this section, we shall only consider implementation of the data updates and exponential forgetting; systolic implementation of regularization will be discussed in the next section.

To describe the array for the RLS algorithm, we shall first extend the above given example of computation of the vector a with computation of the prediction error ε (5.69). Then, we shall describe the systolic implementation of updates of the covariance matrix factor R and of the parameter estimates Θ by the Givens rotations (5.70). Finally, the modifications necessary to merge properly both operations in one array will be discussed.

6.2.1 First step of the square-root RLS algorithm

To implement the formula (5.69), we shall add one more row, containing transposed vector of the parameter estimates Θ, to the multiplication array from the above example (Fig. 6.5). The scheme of the resulting array is in Fig. 6.6.

The function description of the PEs is exactly the same as in the above example of matrix-vector multiplication.

Remark. In the following text, we shall refer the part of the array, where the R' matrix is stored, as the upper part of the array, and the part, where the parameter estimates Θ' are stored, as the bottom row of the array.

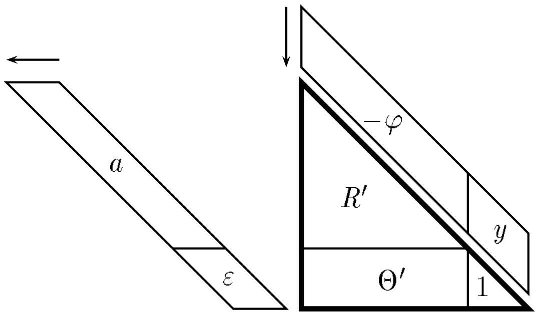


Figure 6.6: Mapping of R' and Θ' to a systolic array

6.2.2 Second step of the square-root RLS algorithm

Let us consider now the systolic implementation of the update equation (5.70). For simplicity, we shall suppose that the forgetting factor λ is constant and that it is stored in the array.

The shape of the array is similar to the first step, with the exceptions that one column is added to the left side of the array, for computation of the rotations ϕ_i (5.62) and that the rightmost PE in the bottom row (denoted by "1" in Fig. 6.6) is not used now. The positions of R' and Θ in the array remain the same as in the first step.

The vector $[a\ \varepsilon]$ (5.69, 5.70) enters this column from the left side, the top element first. To compute $s^{\frac{1}{2}}$ (5.57, 5.58) and the product $s^{\frac{1}{2}}\kappa$ (5.54, 5.58), number 1 is entered to the top of the column, where the rotations are computed and a zero vector to the top of the array and they are passed downward and modified by rotations. The result is used in the lower cells — to compute $-\varepsilon/\sqrt{s}$ (5.68) in the left column and to update the parameter estimates in the array itself.

The scheme of the array and the arrangement of the inputs and output are drawn in Fig. 6.7. The upper and lower parts of the array are drawn as separate blocks. This is carried out in order to illustrate *a)* that for the computation of $\sqrt{s}$ and $\sqrt{s}\kappa$, only the upper part of the array is used, *b)* that these variables (after having been computed) are used in the lower part of the array. Finally, this arrangement of the drawing is used to emphasize the different characters of rotations applied in the upper and lower parts of the array.

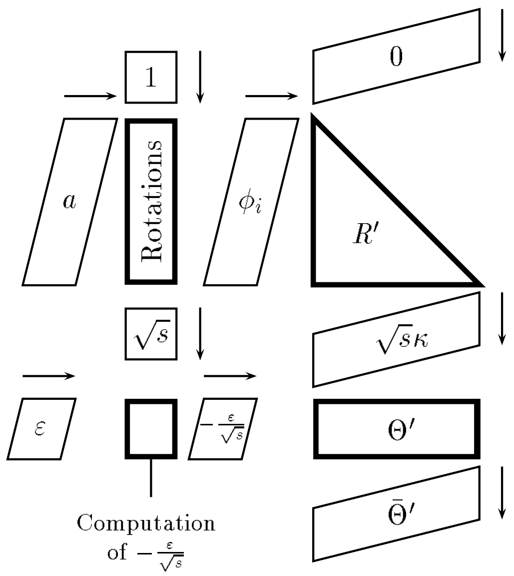


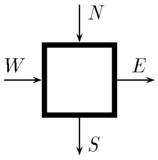
Figure 6.7: Systolic implementation of the RLS update

Update array — description of processors

Notation of input and output variables. In the example of systolic matrix-vector multiplication, given in the previous section, we were using the underlined index to refer to the exact position of the partial result in the array. We used this information to help the reader understand how the data is propagated and pipelined, but to give a complete and unambiguous description of the array, this is not generally necessary. Instead, we shall use the notation introduced in Chapter 3 in the rest of thesis: for a given variable, which is updated in the cell, we shall use the 'tilde' symbol to refer the value before the update and the 'bar' symbol to refer the value after the update. The variable which is not updated will be left without any symbol.

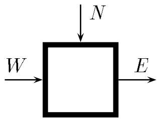
Description of processors. The function of the PEs in the array for the RLS update is described in Fig. 6.8 (left column), and Fig. 6.9 (right part of array, where R' and Θ are stored).

Upper part



I/O assignment:
N: top element: 1
other elements: $\sqrt{\tilde{s}}$
S: $\sqrt{s}$
W: a_i
E: $\cos\phi_i$, $\sin\phi_i$
Cell functionality:
$\cos\phi_i := \sqrt{\tilde{s}}/\sqrt{\tilde{s}+a_i^2}$
$\sin\phi_i := a_i/\sqrt{\tilde{s}+a_i^2}$
$\sqrt{s} := \sqrt{\tilde{s}}\cos\phi + a_i\sin\phi$

Lower PE



I/O assignment:
N: $\sqrt{s}$
W: ε
E: $\omega := -\varepsilon/\sqrt{s}$

Figure 6.8: Description of processors in the left column of the update array

6.2.3 Merging both steps together

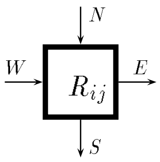
So far, the array for the matrix-vector multiplication and the array for the RLS update were considered separately. The goal is to perform both the multiplication (5.69) and the update (5.70) pipelined in one array. Full derivation of the array, which performs both steps of the RLS update simultaneously, may be found in [35]. Here, key problems and their solutions are selected.

Opposite directions of horizontal data flows

Let us first simply merge both of the above described arrays together (mapping the first column of the array in Fig. 6.7 to the triangular part of the array and connecting the output of the multiplication array with the horizontal input of the update array, as schematically shown in Fig. 6.10).

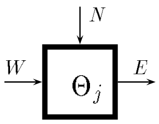
The following problem arises: the matrix-vector product is accumulated from right to left (Fig. 6.6), while the rotations propagate from left to right (Fig. 6.7). Because of these opposite data flows, it is necessary to wait until

Upper part



I/O assignment:
N: diagonal elements: 0
other elements: $\sqrt{\tilde{s}}\tilde{k}_j$
S: $\sqrt{\bar{s}}\bar{k}_j$
W: $\cos\varphi_i$, $\sin\varphi_i$
E: $\cos\varphi_i$, $\sin\varphi_i$
Cell functionality:
$C := \cos\phi_i$, $S := \sin\phi_i$
$\sqrt{\bar{s}}\bar{k}_j := C\sqrt{\tilde{s}}\tilde{k}_j + S\tilde{R}_{ij}$
$\bar{R}_{ij} := (1/\sqrt{\lambda})(-S\sqrt{\tilde{s}}\tilde{k}_j + C\tilde{R}_{ij})$

Lower part



I/O assignment:
N: $\sqrt{\bar{s}}k_j$
S: $\bar{\Theta}_j$
W: ω
$E := W$
Cell functionality:
$\bar{\Theta}_j := \tilde{\Theta}_j + \omega\sqrt{\bar{s}}k_j$

Figure 6.9: Description of processors in the right part of the update array

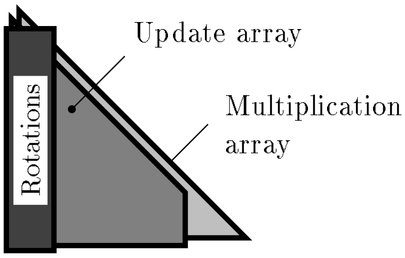


Figure 6.10: Merging of multiplication and update array

the update is completely finished (i.e. until the rotation wave has completely passed through the array) before entering the next data vector φ, otherwise the elements of matrix R, that have not yet been updated, would be used for the vector-matrix multiplication inside the array.

Restoration of data pipelining

The data pipelining of the resulting array may restored using methods of algorithmic engineering [37]. The solution is described in [39] and [43]. It works on the following principle: the respective elements of the data vector φ are multiplied with the old, non-updated elements of the covariance matrix factor R, but before the product is passed to the next PE, it is rotated with an added correction. This correction is added in each cell so that the final vector-matrix product is correct. For proper time-space synchronization of the values propagating through the array, delay has to be added into the horizontal links. Applying the '2-slowing lemma', used in the systolic algorithm design [40], the delay may be introduced by running only half of PEs in each step (Fig. 6.11).

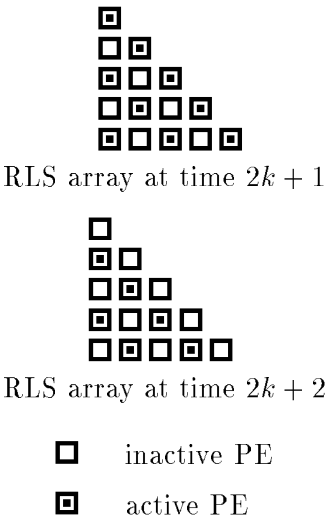


Figure 6.11: Switching of active and inactive cells in the resulting array

The initial value of correction is entered in the diagonal elements, and the correction is passed trough the array together with the data sample φ and the vector $\sqrt{s}\kappa$. Together with number "1", entered to "rotations" column (Fig. 6.7), the initial value of the correction forms vector α,

$$\alpha = [1\ 0\ \ldots 0]. \tag{6.3}$$

Loading of the array with the data vector φ and with the initial values of the vector $\sqrt{s}\kappa$ and of the correction α is sketched in Fig. 6.12.

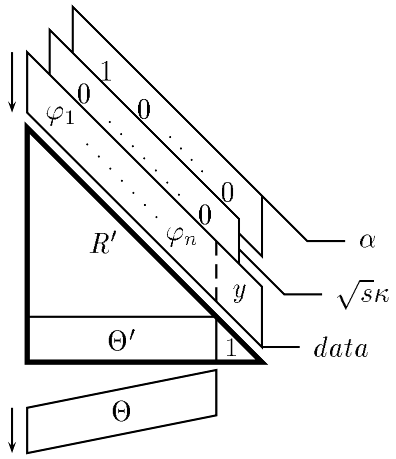


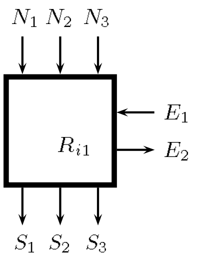
Figure 6.12: Loading of array with the data φ and with the initial values of κ and α

6.2.4 Cell-level description of the resulting array

Function description of the processing elements in the resulting pipelined implementation of RLS algorithm with exponential forgetting is given in Figure 6.13 and 6.14 (upper part of the array), Figure 6.15 (bottom row except the rightmost cell) and Figure 6.16 (rightmost cell of the bottom row).

6.2.5 Summary

In this subsection, we have described systolic implementation of the first (5.69) and second (5.70) step of the RLS identification and how to merge both steps in one systolic array.



Cell functionality:

$\gamma := \tilde{a}_i - \tilde{R}_{i1}\varphi_1$

$C = \cos\phi_i := \sqrt{\tilde{\alpha}_1} / \sqrt{\tilde{\alpha}_1 + \gamma^2}$

$S = \sin\phi_i := \gamma / \sqrt{\tilde{\alpha}_1 + \gamma^2}$

$\bar{\alpha}_1 := C\,\tilde{\alpha}_1 + S\,(\tilde{a}_i - \tilde{R}_{i1}\varphi_1)$

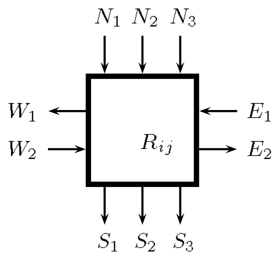
$\sqrt{\bar{s}}\bar{k}_1 := C\,\sqrt{\tilde{s}}\tilde{k}_1 + S\,\tilde{R}_{i1}$

$\bar{R}_{i1} := (1/\sqrt{\lambda})\left(-S\,\sqrt{\tilde{s}}\tilde{k}_1 + C\,\tilde{R}_{i1}\right)$

I/O assignment:

N_1: φ_1
N_2: top cell: $\tilde{\alpha}_1 := 1$
other cells: $\tilde{\alpha}_1$
N_3: top cell: $\sqrt{\tilde{s}}\tilde{k}_1 := 0$
other cells: $\sqrt{\tilde{s}}\tilde{k}_1$
S_1: φ_1
S_2: $\bar{\alpha}_1$
S_3: $\sqrt{\bar{s}}\bar{k}_1$
E_1: top cell: 0
other cells: $\tilde{a}_i$
E_2: top cell: not used
other cells: ϕ^i

Figure 6.13: Description of the left column of the upper part of a systolic array for RLS identification with exponential forgetting



Cell functionality:

$C := \cos\phi_i$

$S := \sin\phi_i$

$\bar{\alpha}_j := C\,\tilde{\alpha}_j + S\,(\tilde{a}_i - \tilde{R}_{ij}\varphi_j)$

$\sqrt{\bar{s}}\bar{k}_j := C\,\sqrt{\tilde{s}}\tilde{k}_j + S\,\tilde{R}_{ij}$

$\bar{R}_{ij} := (1/\lambda)\left(-S\,\sqrt{\tilde{s}}\tilde{k}_j + C\,\tilde{R}_{ij}\right)$

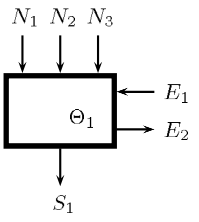
$\bar{a}_i := (1/\lambda)(-S\,\tilde{\alpha}_j + C\,(\tilde{a}_i - \tilde{R}_{ij}\varphi_j))$

I/O assignment:

N_1: φ_j
N_2: diagonal cells: $\tilde{\alpha}_j := 0$
other cells: $\tilde{\alpha}_j$
N_3: diagonal cells: $\sqrt{\tilde{s}}\tilde{k}_j := 0$
other cells: $\sqrt{\tilde{s}}\tilde{k}_j$
S_1: φ_j
S_2: $\bar{\alpha}_j$
S_3: $\sqrt{\bar{s}}\bar{k}_j$
W_1: $\bar{a}_i$
W_2: $\cos\phi_i,\ \sin\phi_i$
E_1: diagonal cells: 0
other cells: $\tilde{a}_i$
E_2: diagonal cells: not used
other cells: $\cos\phi_i,\ \sin\phi_i$

Figure 6.14: Description of the internal columns of the upper part of a systolic array for the RLS identification with exponential forgetting

First cell



I/O assignment:

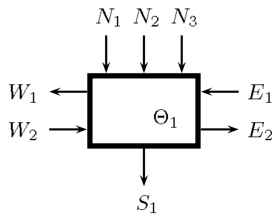
N_1: φ_1
N_2: $\sqrt{s} := \alpha_1$
N_3: $\sqrt{s}k_1$
S_1: $\bar{\Theta}_1$
E_1: $\tilde{\varepsilon}$
E_2: ω

Cell functionality:

$\omega := -\frac{\tilde{\varepsilon}-\tilde{\Theta}_1\varphi_1}{\sqrt{s}}$

$\bar{\Theta}_1 := \tilde{\Theta}_1 + \omega\sqrt{s}k_1$

Internal cells



I/O assignment:

N_1: φ_j
N_2: α_j
N_3: $\sqrt{s}k_j$
S_1: $\bar{\Theta}_j$
W_1: $\bar{\varepsilon}_j$
W_2: ω
E_1: $\tilde{\varepsilon}_j$
E_2: cells $\Theta_2 \ldots \Theta_{n-1}$: ω
cell Θ_n: not used

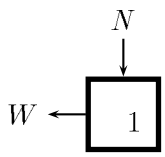
Cell functionality:

$\bar{\Theta}_j := \omega\sqrt{s}k_j + \tilde{\Theta}_j$

$\bar{\varepsilon}_j := \omega\alpha_j + \tilde{\varepsilon}_j - \tilde{\Theta}_1\varphi_j$

Figure 6.15: Description of the lower part of a systolic array for the RLS identification with exponential forgetting

Last column



I/O assignment:

N: y
W: y

Cell functionality:

$W := N$

Figure 6.16: Description of the last cell in the lower row of a systolic array for the RLS identification with exponential forgetting

6.3 Systolic implementation of regularization

Having explained implementation of the RLS algorithm with exponential forgetting, we can finally describe systolic implementation of the block-regularized exponential forgetting. Application of this mechanism to the systolic RLS algorithm with inverse updates is one of the original results of this thesis.

To implement the block-regularized forgetting, as described in Sec. 5.5, it is necessary to implement the following mechanisms:

- Selection of parameter estimates that will be used as regularizing parameters Θ^* (5.47 d).
- Computation of vector u^* (5.80 a).
- Multiplication of the alternative data with the accumulated forgetting factor $\lambda(1, N)$ (5.49) or λ_N (5.50).
- Switching between the real data samples and the alternative data — cf. (5.47 e) and (5.80 b).
- Switching off of the exponential forgetting during processing of the the alternative data — cf. (5.47 e) and (5.80 b).

6.3.1 Computation of vector u^*

Let us suppose, for simplicity, constant forgetting factor $\lambda(k) = \lambda = const.$ in the following text.

Let us further suppose that the accumulated forgetting factor λ_N has been computed before the start of identification and also that the multiplication of the alternative data U^* with this factor has been performed before the start of identification.

Then, we can denote, to simplify the formulae given in this section and the resulting processor descriptions, the factor of the information matrix U^*, multiplied by $\sqrt{\lambda_N}$, as $U^{\lambda *}$:

$$U^{\lambda *} = \sqrt{\lambda_N} U^*, \tag{6.4}$$

Let us further denote, also to simplify the formulae, the vector, resulting from multiplication of $\sqrt{\lambda_N} U^*$ with Θ^* (5.80) as $u^{\lambda *}$:

$$u^{\lambda *} = \sqrt{\lambda_N} u^* = \sqrt{\lambda_N} U^* \Theta^*, \tag{6.5}$$

the rows of $U^{\lambda *}$ as $U_i^{\lambda *}$, and the elements of $u^{\lambda *}$ as

$$u_i^{\lambda *} = U_i^{\lambda *}\Theta^*. \tag{6.6}$$

Loading of the alternative information matrix factor to the array

Matrix $U^{\lambda *}$, before being used as the alternative data input, has to be multiplied with the regularizing estimates Θ^* to compute the vector $u^{\lambda *}$.

It is required when implementing the multiplication that $u_1^{\lambda *}$ is entered to the array the second step after entering the last data sample of the block (second step, because all PEs in array are active only every second step - Fig. 6.11), so that no unnecessary latency is introduced by the regularization. Also, $u_1^{\lambda *}$ must be entered into the array together (in the same step) with the first regularizing vector $U_1^{\lambda *}$.

Let us recall that the estimate computed n steps before the end of the data block is used for Θ^*. Then, a natural solution is to store Θ^* in the same PEs, where Θ is computed, i.e. in the bottom row of the array, and to compute there also the vector $u^{\lambda *}$. $U^{\lambda *}$ will be entered into this row skewed in time, so that the leftmost element of each its row enters the array first. The partially completed product is shifted right until it enters, after being completed, the rightmost cell in the row (Fig. 6.17). In this figure, the first

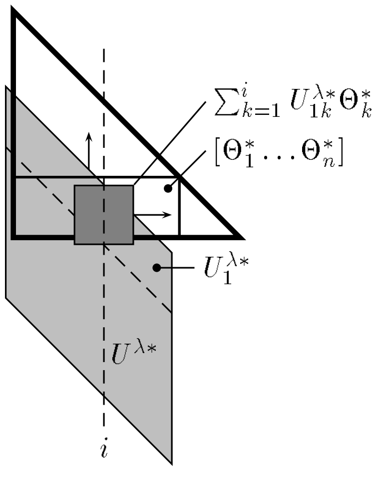


Figure 6.17: Loading of matrix $U^{\lambda *}$ into the array

row of the matrix $U^{\lambda*}$ is schematically denoted by a dashed line and the partially completed product is denoted by a hatched frame.

Remark:

- The rows of $U^{\lambda*}$ are to be entered to the diagonal of the array every second period. Hence, they must enter the bottom row of the array only every second period. This corresponds with the switching of active PEs in the array.

6.3.2 Storing of the regularizing parameters Θ^*

In the preceding paragraph, it was explained that the computation (6.6) of vector $u^{\lambda*}_{\varphi y}$ (6.5) is mapped to the lower row of the array, where also the parameter estimates are updated (because the estimates are being assigned to Θ^* (5.47 d). However, the estimate is updated every second step, while the regularizing parameters Θ^* must remain constant over $2n$ steps (over the time necessary to multiply n rows of matrix $U^{\lambda*}$ (6.4, 5.80 a).

For this reason, it is necessary to store the estimate that is used for Θ^* in a register in each cell of the bottom row and to multiply $U^{\lambda*}$ with the value stored in this register.

6.3.3 Control and synchronization of the array

It follows from the description given in this section that all mechanisms, necessary to implement the regularized forgetting (storing of Θ^*, computation of $u^{\lambda*}$ (5.80 a, 6.6), switching between the real data samples and the regularizing data, switching of the forgetting (cf. (5.47 e) and (5.80 b)) have to be properly timed and synchronized.

To ensure this synchronization, a matrix with binary elements is propagated through the array together with the alternative data $U^{\lambda*}$. Two values of the matrix elements represent two states of the array elements: the state when the identification is performed, and the state when the regularization is performed. This matrix will be referred as a *control matrix* and its elements as a *control signal* in the following text, since they are used to control switching between data update and regularization in the array.

The control matrix is entered to the array from the bottom, in the same way as the regularizing matrix $U^{\lambda*}$ (6.4, 5.73, 5.50) (Fig. 6.18). While the matrix $U^{\lambda*}$ is entered only to the PEs containing the parameter estimates (i.e. NOT to the rightmost cell of the bottom row of the array), the control matrix is entered to ALL cells in the bottom row (because it switches also the input of the cell marked by “1”).

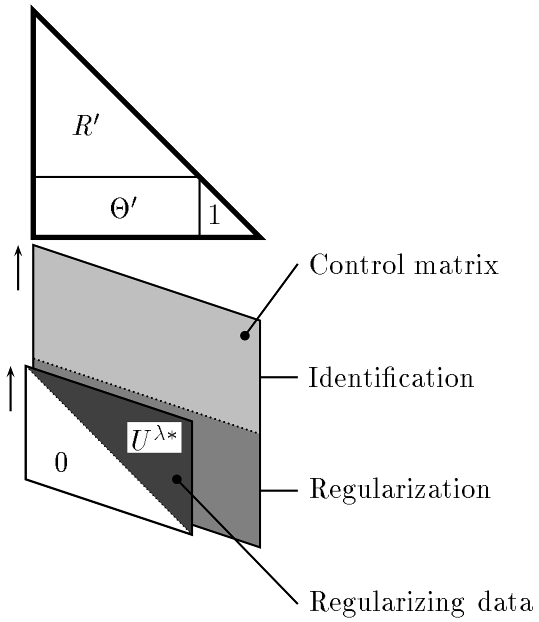


Figure 6.18: Alignment of control matrix with regularizing data and its position at the entry of the array

Alignment of the control and regularizing matrices with the data samples

The time alignment of the control matrix with the regularizing data and with the blocks of data samples is shown in Fig. 6.19.

Control of writing Θ to the storage

The reason for using Θ computed n steps before the end of the data block for Θ^* (Subsec. 5.4.2) is that it is possible then to control writing Θ to the store with the same control matrix that is used for all other events in the array.

The control mechanism of the Θ^* store is following: the updated parameter Θ is written to store, if the element of the control matrix (the control

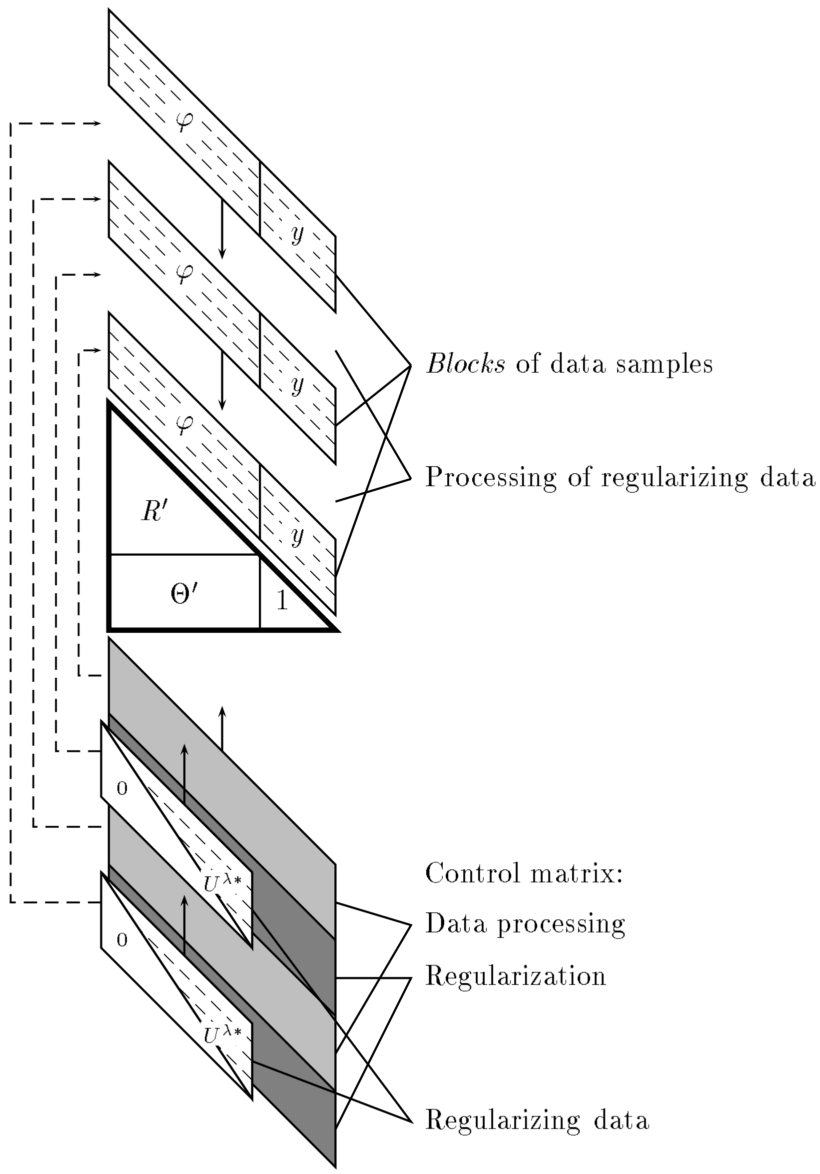


Figure 6.19: Alignment of control matrix with data samples

signal) (Fig. 6.18), which is in that moment in the given PE, is in the "identification" level. If the control signal is in the "regularization level", the value stored in the Θ^* storage is not modified and is used for multiplication with the regularizing matrix $U^{\lambda*}$ (6.6) (Fig. 6.20).

Switching between the real and alternative data

The rows of the matrix $U^{\lambda*}$ (6.4), after having been used for computation of $u^{\lambda*}$ (6.5), are to be entered as the alternative data (Subsec. 5.5.7). The i-th row of the matrix ($U_i^{\lambda*}$) must be entered into the diagonal elements of the upper part of the array in the same time when the i-th element of vector $u^{\lambda*}$ is entered to the rightmost PE of the bottom row of the array.

Considering how is the computation of $u_i^{\lambda*}$ implemented (Fig. 6.17), the simplest solution is to shift $U^{\lambda*}$, after it has been entered into the bottom row, upward through the array, as shown in Fig. 6.17.

The control matrix is shifted through the array together with $U^{\lambda*}$ and is used in diagonal cells to control switching of the array entries between the real and alternative data. This mechanism is (for one column of the systolic array) schematically illustrated in Fig. 6.20.

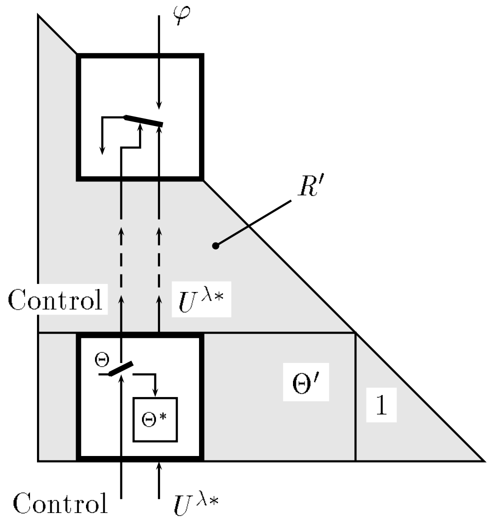


Figure 6.20: Control of the Θ^* store and of the array inputs

Switching the forgetting on and off

Let us recall that during processing of the regularizing data $[U^{\lambda *} \; u^{\lambda *}]$ (6.4, 6.5), the exponential forgetting must be switched off — cf. (5.47 e) and (5.80 b).

The forgetting is applied in the data update (5.70). The switching of forgetting must be synchronized with propagation of rotations for the given input vector through the array (Fig. 6.7). To ensure this, the control signal in the left column of the array, after it enters the diagonal of the array, is propagated in the way sketched in Fig. 6.21 — cf. Fig. 6.23 and 6.24.

The notation used in this figure is following: the squares represent PEs. The arrows, drawn in a full line, represent propagation of the control signal in one step of the array.

The arrows, drawn in a dashed line, are used to denote several steps of propagation of the signal.

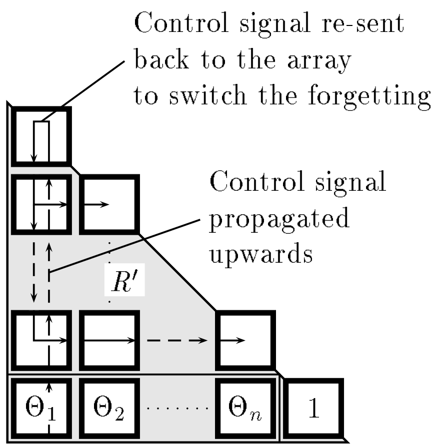


Figure 6.21: Control of forgetting

6.3.4 Buffering of the input data

Using the block regularized identification algorithm, the input data, that are sampled in regular intervals, are processed in blocks (interlaced by the blocks of "alternative" regularizing data).

Hence, the system must be sampled in a slower rate than the systolic estimator runs and a data buffer must be used between the identified system and the estimator.

The time relation between the moments when the data is sampled and when it enters the array is illustrated in Fig. 6.22. This figure contains two time axes, drawn in the same scale, one showing the instants of data sampling, the second showing the instants of entering the data into the array. The arrows between the two axes show the correspondence between these instants. The figure is plotted for $N = n = 2$, $n = \partial\varphi$.

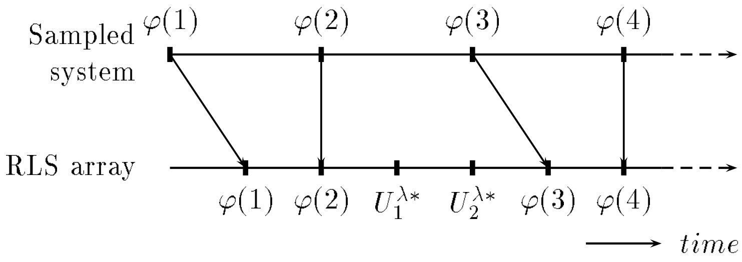


Figure 6.22: Time relation between sampling of the system and entering the data into the regularized systolic estimator, $N = n = 2$

6.3.5 Timing of events in the array

In Table 6.1, relations determining timing of some events in the array are given. The first data vector $\varphi(1)$ is considered to enter the array at $k = 1$. Some rows of the table contain, in fact, entries, that are in more general form contained in other rows. The aim of these redundant entries is to explicitly refer the respective time values.

6.3.6 Summary

In this section, the mechanisms necessary for implementation of the block-regularized forgetting in the systolic RLS algorithm with inverse updates were discussed.

Event	Remark	Time k
$\varphi(1)$	first data sample entering the array	1
$\varphi(i),\ i \leq N$	i-th data sample entering the array	$2(i-1)+1$
$U_{1j}^{\lambda *},\ j = 1 \ldots n$	first alternative data entering the bottom row	$2N - n + j - 1$
$U_{ij}^{\lambda *},\ i,j = 1 \ldots n$	i-th alternative data entering the bottom row	$2(N+i-1) - n + j$
$U_1^{\lambda *}$	first alternative data on the diag. after shifts	$2N+1$
$U_i^{\lambda *},\ i = 1 \ldots n$	i-th alternative data on the diag. after shifts	$2(N+i-1)+1$
$U_n^{\lambda *}$	n-th alternative data on the diag. after shifts	$2(N+n-1)+1$
$\hat{\Theta}_j,\ j = 1 \ldots n$	output of regularized parameters	$2N + 3n + j - 1$
$\varphi(N+1)$	first entry of the next block entering the diagonal	$2(N+n)$
	Period of the blocks	$2(N+n)$

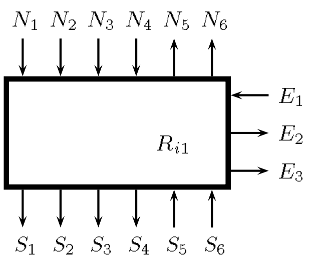
Table 6.1: Timing of the estimator - $N \geq n$

6.4 Cell-level description of regularized array

Having explained all necessary mechanisms, it is possible to give description of the processing elements used in the block-regularized systolic RLS estimator.

For better clarity, the control signal which is shifted upwards through the array is denoted by Ctrl_1 and the signal shifted together with the forgetting factor by Ctrl_2, though it is the same signal, only in different places of the array. The assignment $\mathrm{Ctrl}_2 = \mathrm{Ctrl}_1$ is made in the upper left cell.

The description is given in Fig. 6.23 (left column of the upper part of the array), Fig. 6.24 (internal columns of the upper part of the array), Fig. 6.25 (bottom row of the array) and Fig. 6.26 (last cell in the bottom row of the array).



I/O assignment:

N_1: top cell: φ_1
other cells: d_1
N_2: top cell: $\tilde{\alpha}_1 := 1$
other cells: $\tilde{\alpha}_1$
N_3: top cell: $\sqrt{\tilde{s}}\tilde{\kappa}_1 := 0$
other cells: $\sqrt{\tilde{s}}\tilde{\kappa}_1$
N_4: top cell: not used
other cells: $\mathrm{Ctrl}_{2\,in}$
N_5: top cell: not used
other cells: $\mathrm{Ctrl}_{1\,out}$
N_6: top cell: not used
other cells: $U_{i1}^{\lambda *}{}_{out}$
S_1: d_1
S_2: $\bar{\alpha}_1$
S_3: $\sqrt{\bar{s}}\bar{\kappa}_1$
S_4: Ctrl^2_{out}
S_5: Ctrl^1_{in}
S_6: $U_{i1}^{\lambda *}{}_{in}$
E_1: top cell: 0
other cells: $\tilde{a}_i$
E_2: top cell: not used
other cells: $\cos\phi_i$, $\sin\phi_i$
E_3: top cell: not used
other cells: $\mathrm{Ctrl}_{2\,out}$

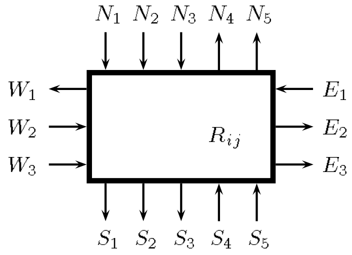
Cell functionality:

Top cell:
(switching of entries)
$\mathrm{Ctrl}_2 := \mathrm{Ctrl}_1$
if $\mathrm{Ctrl}_1 = 1$ *then* $d_1 := \varphi_1$
else $d_1 := U_{i1}^{\lambda *}$ (regularization)
end

Other cells:
$\mathrm{Ctrl}_{1\,out} := \mathrm{Ctrl}_{1\,in}$
$\mathrm{Ctrl}_{2\,out} := \mathrm{Ctrl}_{2\,in}$

All cells:
$\tilde{\beta} := \sqrt{\tilde{s}}\tilde{\kappa}_1$ (auxiliary variable)
$\gamma := \tilde{a}_i - \tilde{R}_{i1} d_1$ (auxiliary variable)
$C = \cos\phi_i := \tilde{\alpha}_1 / \sqrt{\tilde{\alpha}_1^2 + \gamma^2}$
$S = \sin\phi_i := \gamma / \sqrt{\tilde{\alpha}_1^2 + \gamma^2}$
$\bar{\alpha}_1 := C\,\tilde{\alpha}_1 + S\,\gamma$
$\bar{\beta}_1 := C\,\tilde{\beta}_1 + S\,\tilde{R}_{i1}$
$\bar{R}_{i1} := -S\,\tilde{\beta}_1 + C\,\tilde{R}_{i1}$
if $\mathrm{Ctrl}_2 = 1$ *then*
$\bar{R}_{i1} := (1/\sqrt{\lambda})\bar{R}_{i1}$
end
$\sqrt{\bar{s}}\bar{\kappa}_1 := \bar{\beta}$

Figure 6.23: Description of processors in the first column of the upper part of the block-regularized RLS array



Cell functionality:

Diagonal cells:

(switching of entries)

if $\mathrm{Ctrl}_1 = 1$ *then* $d_j := \varphi_j$

else $d_j := U_{ij}^{\lambda *}$ (regularization)

end

Other cells:

$\mathrm{Ctrl}_{out}^1 := \mathrm{Ctrl}_{in}^1$

$\mathrm{Ctrl}_{out}^2 := \mathrm{Ctrl}_{in}^2$

All cells:

$C := \cos\phi$

$S := \sin\phi$

$\tilde{\beta} := \sqrt{\tilde{s}}\tilde{\kappa}_i$ (auxiliary variable)

$\gamma := \tilde{a}_i - \tilde{R}_{ij} d_j$ (auxiliary variable)

$\bar{\alpha}_j := C\,\tilde{\alpha}_j + S\,\gamma$

$\bar{\beta}_j = C\,\tilde{\beta}_j + S\,\tilde{R}_{ij}$

$\bar{R}_{ij} := -S\,\tilde{\beta}_j + C\,\tilde{R}_{ij}$

$\bar{a}_i := -S\,\tilde{\alpha}_j + C\,\gamma$

if $\mathrm{Ctrl}_2 = 1$ *then*

$\bar{R}_{ij} := (1/\sqrt{\lambda})\bar{R}_{ij}$

$\bar{a}_i := (1/\sqrt{\lambda})\bar{a}_i$

end

$\sqrt{\bar{s}}\bar{\kappa}_i := \tilde{\beta}$

I/O assignment:

N_1: diagonal cells: φ_j
other cells: d_j

N_2: diagonal cells: $\tilde{\alpha}_j := 0$
other cells: $\tilde{\alpha}_j$

N_3: diagonal cells: $\sqrt{\tilde{s}}\tilde{\kappa}_j := 0$
other cells: $\sqrt{\tilde{s}}\tilde{\kappa}_j$

N_4: diagonal cells: not used
other cells: $\mathrm{Ctrl}_{1\,out}$

N_5: diagonal cells: not used
other cells: $U_{ij\ out}^{\lambda *}$

S_1: d_j

S_2: $\bar{\alpha}_j$

S_3: $\sqrt{\bar{s}}\bar{\kappa}_j$

S_4: Ctrl_{in}^1

S_5: $U_{ij\ in}^{\lambda *}$

W_1: $\bar{a}_i$

W_2: $\cos\phi_i$, $\sin\phi_i$

W_3: $\mathrm{Ctrl}_{2\,in}$

E_1: diagonal cells: 0
other cells: $\tilde{a}_i$

E_2: diagonal cells: not used
other cells: $\cos\phi_i$, $\sin\phi_i$

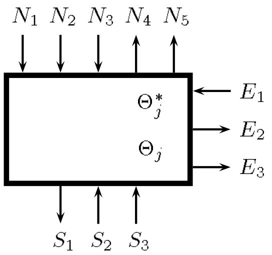
E_3: $\mathrm{Ctrl}_{2\,out}$

Figure 6.24: Description of processors in the internal columns of the upper part of the block-regularized RLS array

6.5 Illustrative example

The function of the algorithm is illustrated on identification of a simulated system. The identified system is described by an ARX model of the following

First column



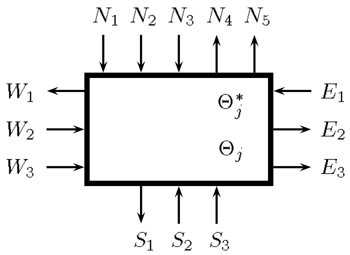
I/O assignment:

N_1: d_1
N_2: α_1
N_3: $\sqrt{s}\kappa_1$
N_4: $\mathrm{Ctrl}_{1\,out}$
N_5: $U^{\lambda *}_{i1\ out}$
S_1: Θ_1
S_2: $U^{\lambda *}_{i1\ in}$
S_3: $\mathrm{Ctrl}_{1\,in}$
E_1: $\tilde{\varepsilon}_1$
E_2: ω
E_3: $U^{\lambda *}_{i1}\Theta^*_1$

Cell functionality:

$\sqrt{s} := \alpha_1$
$\mathrm{Ctrl}_{1\,out} := \mathrm{Ctrl}_{1\,in}$
$U^{\lambda *}_{i1\ out} := U^{\lambda *}_{i1\ in}$
$\omega := \frac{\tilde{\varepsilon}_1 - \tilde{\Theta}_1 d_1}{\sqrt{s}}$
$\bar{\Theta}_1 := \tilde{\Theta}_1 - \omega\sqrt{s}\kappa_1$
if $\mathrm{Ctrl}_1 = 1$ *then* $\Theta^*_1 := \bar{\Theta}_1$
else $E_3 := U^{\lambda *}_{i1}\Theta^*_1$
end

Internal columns



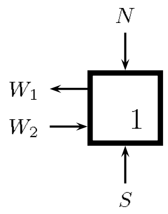
I/O assignment:

N_1: d_j
N_2: α_j
N_3: $\sqrt{s}\kappa_j$
N_4: $\mathrm{Ctrl}_{1\,out}$
N_5: $U^{\lambda *}_{ij\ out}$
S_1: Θ_j
S_2: $U^{\lambda *}_{ij\ in}$
S_3: $\mathrm{Ctrl}_{1\,in}$
W_1: $\bar{\varepsilon}_j$
W_2: ω
W_3: $\sum_{k=1}^{j-1} U^{\lambda *}_{ik}\Theta^*_k$
E_1: $\tilde{\varepsilon}_j$
E_2: cells $\Theta_2 \ldots \Theta_{n-1}$: ω
cell Θ_n: not used
E_3: $\sum_{k=1}^{j} U^{\lambda *}_{ik}\Theta^*_k$

Cell functionality:

$\mathrm{Ctrl}_{1\,out} := \mathrm{Ctrl}_{1\,in}$
$U^{\lambda *}_{ij\ out} := U^{\lambda *}_{ij\ in}$
$\bar{\Theta}_j := \tilde{\Theta}_j - \omega\sqrt{s}\kappa_j$
$\bar{\varepsilon}_j := \tilde{\varepsilon}_j - \tilde{\Theta}_j d_j - \omega\alpha_j$
if $\mathrm{Ctrl}_1 = 1$ *then* $\Theta^*_j := \bar{\Theta}_j$
else $E_3 := W_3 + U^{\lambda *}_{ij}\Theta^*_j$
end

Figure 6.25: Description of processors in the lower part of the block-regularized RLS array



I/O assignment:
N: y
W_1: ε_{n+1}
W_2: $u_i^{\lambda *} = U_i^{\lambda *}\Theta^*$
S: Ctrl_1
Cell functionality:
if $\mathrm{Ctrl}_1 = 1$ *then* $\varepsilon_{n+1} = y$
else $\varepsilon_{n+1} = u_i^{\lambda *}$
end

Figure 6.26: Function description of the rightmost processing cell in the bottom row of the block-regularized RLS array

structure:

$$y(k) = a_1 y(k-1) + a_2 y(k-2) + b_0(k)u(k) + b_1(k)u(k-1) + ce(k), \quad (6.7)$$

where a_1 and a_2 were constant coefficients with values

$$a_1 = -0.2, \quad a_2 = -0.05,$$

and $b_0(k)$ and $b_1(k)$ were time variable coefficients with initial values

$$b_0 = 0.2, \quad b_1 = 0.3$$

and time development described by the graph on Fig. 6.27,

The output disturbance $e(k)$ was discrete normal white noise $\mathcal{N}(0,1)$.

The coefficient of the output disturbance was set either to $c = 0$ (no noise on the system output), or to $c = 0.02$.

Before the start of identification, the system was run 2 steps from zero initial conditions with white $\mathcal{N}(0,1)$ noise on input, to fill the data regressor with non-zero values.

The function of the systolic array was simulated using the X*lab* CADCS (CAD of Control Systems) system [44, 45, 46] (a MATLAB-compatible language with extensions for the system simulation) on an AT-compatible computer.

Settings of the identification algorithm in the simulation

For the simulation, the following settings of the identification algorithm were used:

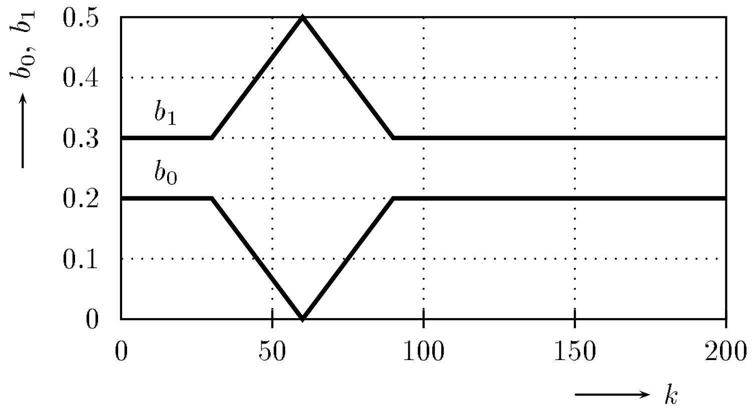


Figure 6.27: Time development of coefficients $b_0(k)$ and $b_1(k)$

- The length of the regressor for identification was set to $n = 4$.
- The length of the data block was set equal to the length of regressor, $N = n = 4$.
- The initial value of the covariance matrix factor $R(0)$ was set to a diagonal matrix
$$R(0) = 200I.$$
- The forgetting factor λ (from which the block coefficient λ_N was computed) was set to
$$\lambda = 0.8$$
- The matrix U^* (the triangular factor of the regularizing matrix) was set to a diagonal matrix
$$U^* = 0.07I.$$
(This is the value *before* weighting by $\sqrt{\lambda_N}$.)

The identification runs over 200 steps of the identified system, i.e. over 800 steps of the regularized systolic array (due to the regularization, with $N = n$, the array has to run four times as fast as the system is sampled — it has to process twice as much data as there are system samples and the data is entered only every second step). The time scale of graph of the system parameter development (Fig. 6.27) is related to the sampling period of the identified system. This length of the simulation run is rather short. The reason is that we want to show details of the time behaviour of elements of the covariance matrix factor.

Selection of input signal — influence of regularization

A discrete white noise $\mathcal{N}(0,1)$ was chosen for the input signal. The system was excited in the first 80 steps (in the first 320 steps of the systolic array). Then, the input was switched off to simulate the simplest case of poor excitation – zero input. From the 170 step (680 tick of the array) on, the input was switched on again.

To compare the influence of regularization, the identification was run both with the regularized forgetting and with the standard exponential forgetting. To get comparable results, 200 samples of the simulated system were mapped to 800 simulation steps when using the exponential forgetting (i.e. one data sample processed every four simulation steps), and the array was active only every second step of simulation.

6.5.1 Simulation results

The eight graphs, that follow, represent the results of these simulation. They show tracking of parameters and development of the first diagonal element (R_{11}) of the covariance matrix factor in the four simulation runs:

1. identification with block-regularized forgetting, no noise on the system output ($c = 0$) — Fig. 6.28,

2. identification with exponential forgetting, no noise on the system output ($c = 0$) — Fig. 6.29,

3. identification with block-regularized forgetting, white noise $\mathcal{N}(0,1)$ present on the system output ($c = 0.02$), — Fig. 6.30,

4. identification with exponential forgetting, white noise $\mathcal{N}(0,1)$ present on the system output ($c = 0.02$) — Fig. 6.31.

6.5.2 Comments on simulation results

- If there is no noise on the system output, the diagonal of the covariance matrix factor R grows exponentially to infinity if the system input is zero (i.e. if the input does not excite the identified system) (5.32, 5.41, 5.70), but because the prediction error ε (5.29) is zero, the parameter estimates remain unchanged. In this case, the only influence of the regularized forgetting is that it keeps the covariance matrix in finite limits (Fig. 6.28, 6.29). However, this case is mostly theoretical and as a simulation experiment it was intended to show the behaviour of the estimator in ideal conditions. In real use, the output noise is always present in data samples (sampling noise, induced noise in industrial environment, etc.). The growth of the covariance matrix due to non-informative data causes numerical instability of the Kalman gain vector (5.30, 5.70) [34]. If the growth is not prevented by regularization, it increases sensitivity of the estimator to the prediction error (5.31), which results in oscillations of the parameter estimate (cf. Fig. 6.30 and 6.31).

- The graphs in Fig. 6.28 and 6.30 show that R_{11} (diagonal element of the covariance matrix factor) oscillates if the system input is zero. These oscillations (concerning all elements of R) are caused by accumulation of data updates in blocks — during data updates, exponential forgetting is applied and the element grows. Then, with processing of the regularizing data, the value of the element is lessened at a jump. This process repeats with every new data block (until informative data is processed again).

  With standard regularization, the covariance matrix would, after switching the system input off, monotonously converge to the value of the regularizing matrix.

- The parameter estimates follow the evolution of the identified system with a longer period, if using the block-accumulated regularization. The reason is that the regularized estimate is received only after both the data block and the regularizing data have been processed. On the other hand, using the exponential forgetting, the parameter estimate is updated with every new data sample.

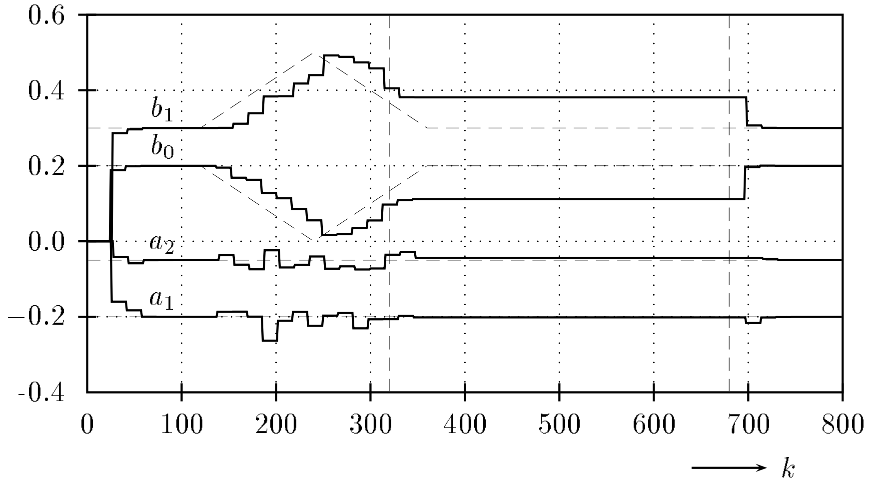
- In our simulation experiment, the input of the system was switched off before the variation of parameters was finished. The graphs show that with the selection of the regularizing parameters that is used in the thesis (Subsec. 5.4.2), the estimator preserves the last value of estimate identified from informative data until the system input is switched on again.

6.6 Summary

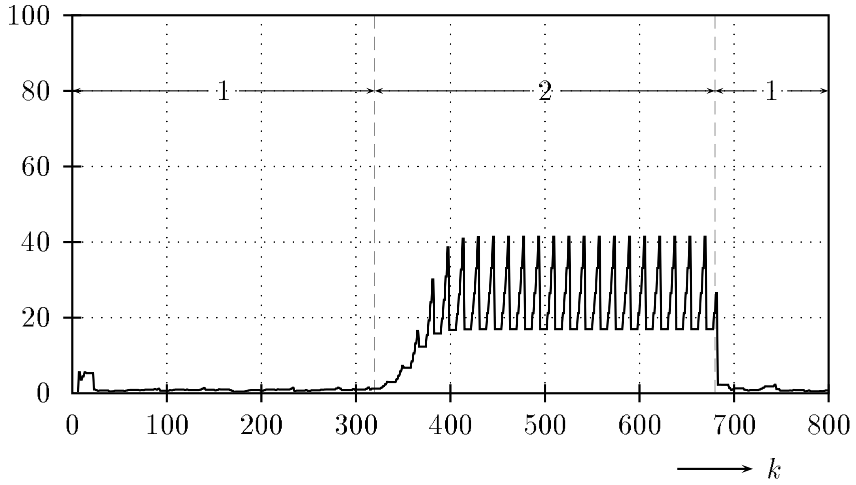
In this chapter, an introduction to the systolic systems was given and the systolic implementation of the regularized RLS estimator was described.

The original contribution of this chapter was a) detailed discussion of the problems that must be solved in the systolic implementation of the block-regularized exponential forgetting and b) the cell-level description of the resulting regularized systolic estimator.

In the next chapter, the problems of the LQG control synthesis will be discussed and a square-root version of the LQG algorithm will be derived.



Development of regularized parameters

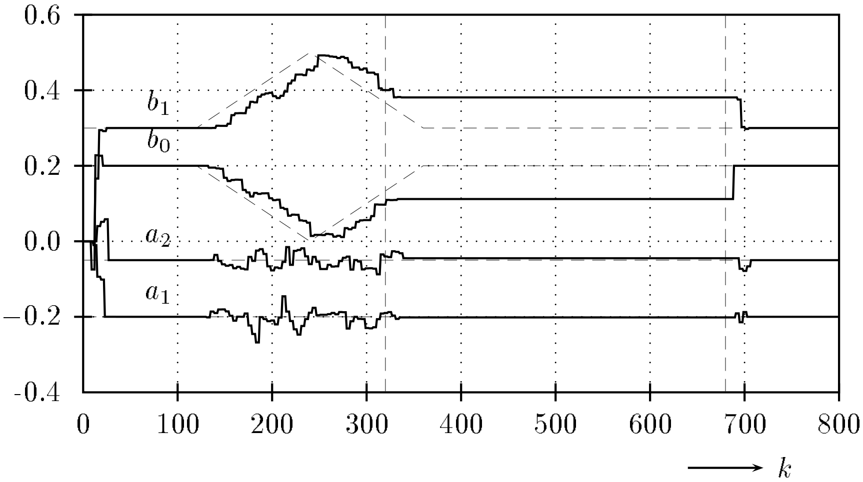


Development of R_{11}

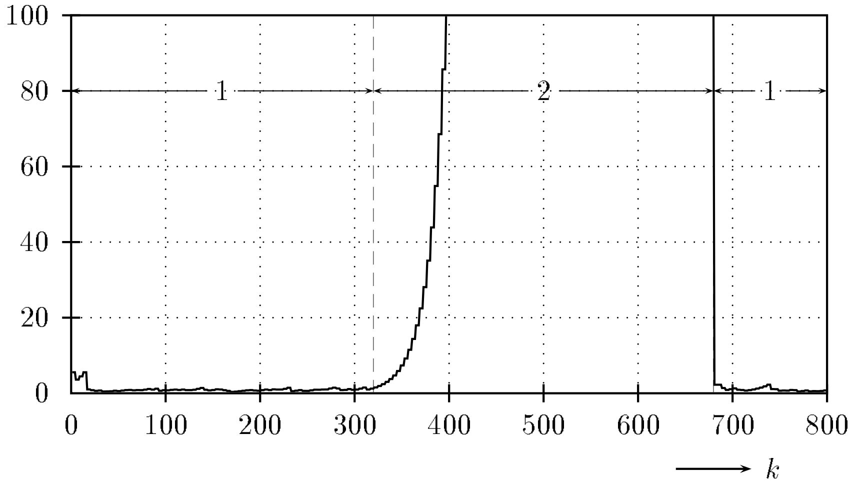
1 – non-zero input

2 – zero input

Figure 6.28: Regularized forgetting, no noise on system output



Development of parameters

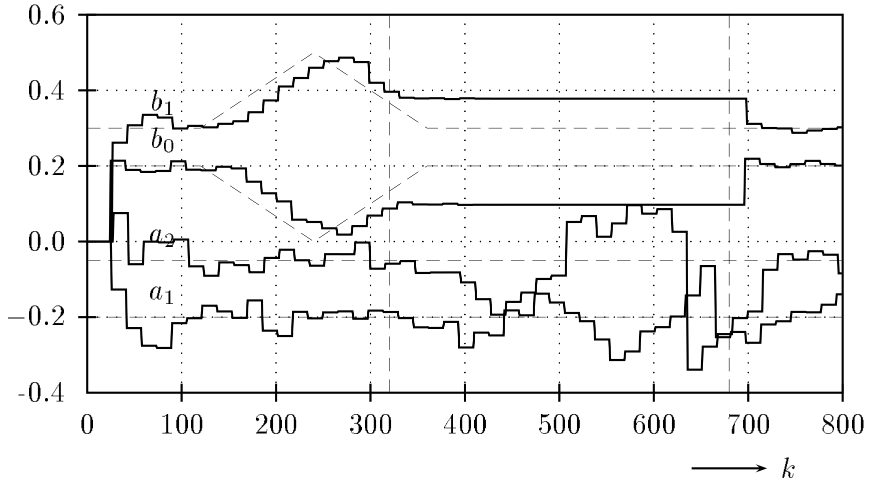


Development of R_{11}

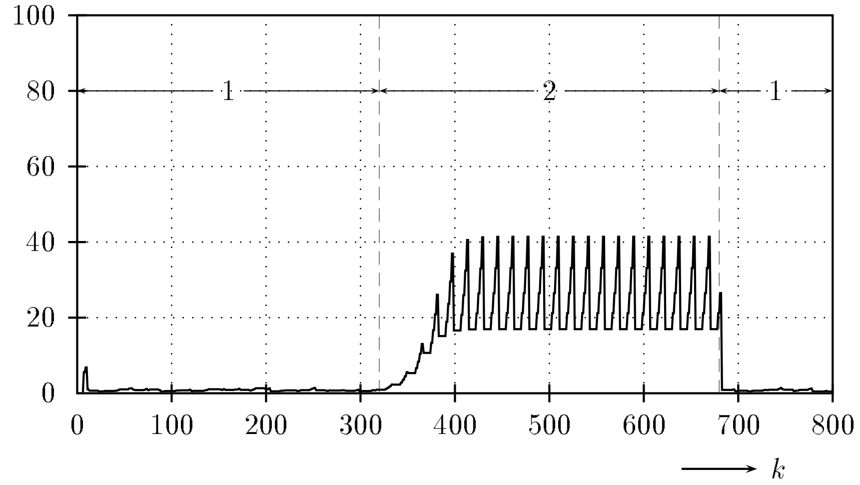
1 – non-zero input

2 – zero input

Figure 6.29: Exponential forgetting, no noise on system output



Development of regularized parameters

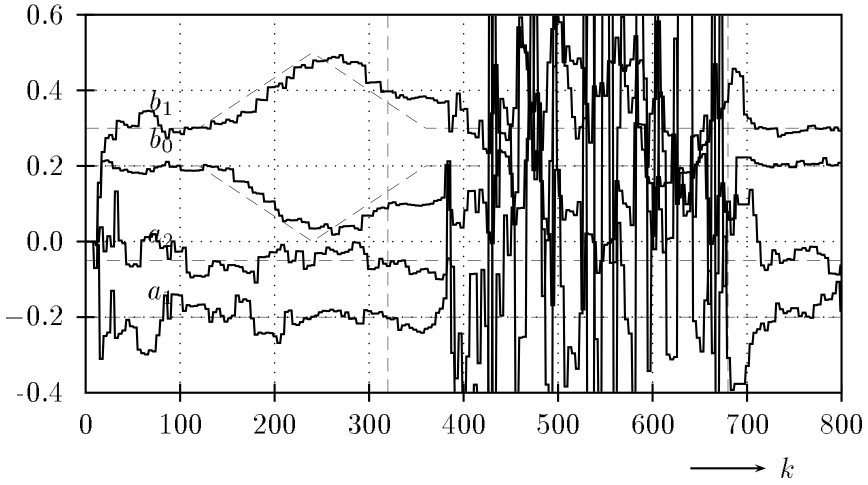


Development of R_{11}

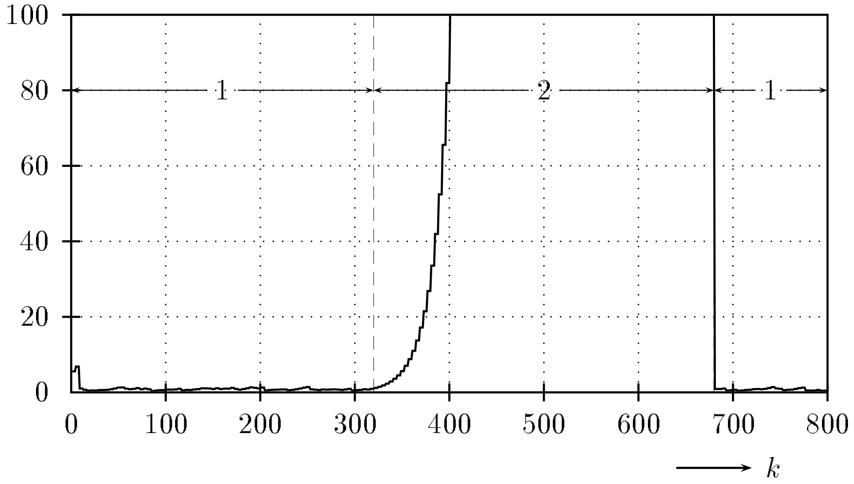
1 – non-zero input

2 – zero input

Figure 6.30: Regularized forgetting, noise on system output



Development of parameters



Development of R_{11}

1 – non-zero input

2 – zero input

Figure 6.31: Exponential forgetting, noise on system output

Chapter 7

Square-root LQG control synthesis

Having described the identification algorithm and its systolic implementation, in this chapter we shall return back to the problems of control synthesis. We shall introduce here the background knowledge necessary for description of the systolic implementation of the controller, which will be given in the next chapter.

First we shall explain the function of the receding horizon control strategy. Then we shall show, how the general formulae of dynamic programming are influenced by use of the ARX system model and of parameters of the GiW-distribution. Finally we shall derive the square–root algorithm for the LQG control synthesis.

7.1 Receding horizon control strategy

In adaptive control, we have to cope with two contradictory demands:

- The control horizon H should be as long as possible [23] (one important reason is to guarantee the stability of the control loop).

- We do not know the future development of the system parameters. Because of that, we have to treat them, when planning the control actions, as if they do not change in time, that is, we have to use the last available parameter estimate (the newest information about the development of parameters, that we have) for the whole control synthesis. However, making the control adaptive, we admit that the system parameters actually may vary in time. Then, the control plan prepared for a long time period may not correspond with the controlled system after some time.

To find some reasonable compromise, let us suppose that the changes of parameters be negligible between two successive parameter estimates. Every time we get a new estimate, we shall use it to compute the control for a horizon as long as possible, but short enough to finish the computation before we get the next estimate. We shall apply only the first step of the whole computed control plan, and then repeat the computation with the new parameter estimate.

Since we restart the control synthesis at the beginning of each control period, while the control horizon permanently recedes, we call this method *receding horizon control strategy.*

The time index k (4.41) in (4.39, 4.40) then takes values over the interval

$$k = k_0 + H, \ldots, k_0, \tag{7.1}$$

in each step of control, where k_0 is the beginning of control horizon.

From this point on, we shall work only with the receding horizon control strategy.

7.2 Linear Quadratic Gaussian control synthesis

In this section, we shall describe the LQG control algorithm.

Using the ARX model (5.13) and the parameters of the GiW-distribution (5.23 – 5.26), we shall derive an analytic formula to compute the optimal input of the controlled system. We shall show the necessity of a suitable approximation of evolution of the kernel of the cost-to-go function for the second and subsequent steps of the dynamic programming.

7.2.1 Dynamic programming formulae

Let us recall that using the sufficient statistics $\mathcal{S}$ with properties as defined in paragraph 4.4, the formulae for the dynamic programming (4.24 – 4.26) read

$$u^*(w, \mathcal{S}) \in \operatorname{Arg}\min_u \mathsf{E}\left[\tilde{K}^* + J | u, w, \mathcal{S}\right], \tag{7.2}$$

$$\bar{K}^* = \min_u \mathsf{E}\left[\tilde{K}^* + J | u, w, \mathcal{S}\right], \tag{7.3}$$

where

$$k = k_0 + H, k_0 + H - 1, \ldots, k_0,$$
$$u(k) \in \mathcal{U}(k, \mathcal{S}(k|k-1))$$
$$K^*_{H+1} = 0.$$

Using the parameters of the GiW-distribution (5.23 – 5.26), as defined in Section 5.1, the statistics $\mathcal{S}(k|k-1)$ (4.38) takes the form of a quadruplet

$$\mathcal{S} = (P,\ \sigma,\ \Theta,\ \nu). \tag{7.4}$$

7.2.2 Notational conventions

In analogy with the "tilde-bar" convention used in the part on identification (Chapter 3 and Section 4.4), the values of variables *entering* the step of minimization will be denoted by the *tilde* symbol, where appropriate to simplify the expressions. The values *resulting* from the minimization will be denoted by the *bar* symbol.

Time will be supposed to run over the period

$$k = k_0 + H, k_0 + H - 1, \ldots, k_0$$

and the initial value of K^* to be equal to zero:

$$K^*_{H+1} = 0.$$

7.2.3 State vector

With the arrangement of Θ as introduced in (5.14), it is possible to split the data vector into two parts:

$$\varphi = \begin{bmatrix} u & x' \end{bmatrix}', \tag{7.5}$$

where

u is the current system input and

x is the *state vector.* x is composed of the couples $y(k-i), u(k-i)$:

$$x(k-1) = [y(k-1), u(k-1), \ldots, y(k-l), u(k-l)]' \tag{7.6}$$

Using the state vector, we can write the model of the system (5.13) in the following form:

$$y = \Theta'\varphi + e = \begin{bmatrix} b_0 & \Theta_x' \end{bmatrix} \begin{bmatrix} u \\ x \end{bmatrix} + e, \tag{7.7}$$

where

b_0 is the first entry of Θ, which multiplies $u(k)$,

Θ_x is that part of Θ, which multiplies the state vector x (7.5), and

$e = \mathcal{N}(0, \sigma)$ is a normal white noise.

Hyperstate

Using the ARX model, the expectation in the dynamic programming equation (7.3) may be written as a function of the system state:

$$\bar{K}^* = \min_u \mathsf{E}\left[\tilde{K}^* + J|u, x, w, \mathcal{S}\right],$$

To express explicitly the dependence on x, we introduce so-called *hyperstate* $\mathcal{X}$, consisting of the state x and of the sufficient statistics $\mathcal{S}$ (7.4):

$$\mathcal{X} = (x;\ \mathcal{S}),$$

with the GiW-distribution:

$$\mathcal{X} = (x;\ P,\ \sigma,\ \Theta,\ \nu)$$

7.2.4 Derivation of an optimal input

Using the hyperstate $\mathcal{X}$ and the quadratic loss function (4.8, 4.9, 4.17) with the control error (4.1) and the control effort (4.2), we can rewrite the control criterion as follows:

$$\bar{K} = \mathsf{E}\left[\tilde{K}^* + q_y + q_u|\ \mathcal{X};\ u\right]. \tag{7.8}$$

The first step of dynamic programming (for $k = k_0 + H$) then reads:

$$\begin{aligned} \bar{K}_1^* &= \min_u \mathsf{E}\left[q_y + q_u|\ \mathcal{X};\ u\right] = \\ &= \min_u \left\{(\Theta'\varphi - w_y)^2 Q_y + (u - w_u)^2 Q_u + \sigma + \sigma\varphi' P\varphi\right\}. \end{aligned} \tag{7.9}$$

To evaluate the conditional expectation, the following formulae are used :

$$\mathsf{E}\,[y|\mathcal{S},u] = \hat{y}(\mathcal{S},u) = \Theta'\varphi \tag{7.10}$$

$$\mathrm{Cov}[y|\mathcal{S},u] = \sigma_y(\mathcal{S},u) = \sigma(1+\xi) \tag{7.11}$$

$$\xi = \varphi' P \varphi. \tag{7.12}$$

Let us split the covariance matrix P (5.32, 7.9) to the submatrices:

$$P = \begin{bmatrix} P_u & P'_{ux} \\ P_{ux} & P_x \end{bmatrix}, \tag{7.13}$$

where P_u is scalar, P_{ux} is vector and P_x is matrix of dimensions $\partial P_x = (2l, 2l)$, l is defined by (5.15).

To simplify the relation for the optimal input in the first step of the dynamic programming, let us suppose $Q_y = 1,\ w_y = w_u = 0$. Then, we may rewrite (7.9) in the following form:

$$\bar{K}_1^* = \min_u \left\{ (b_0^2 + \sigma P_u + Q_u)\left[u + \tfrac{1}{b_0^2+\sigma P_u+Q_u}(b_0\Theta_x + \sigma P_{ux})'x\right]^2 + \sigma + \right.$$
$$\left. + \, x'\left[\Theta_x\Theta_x' - \tfrac{(b_0\Theta_x+\sigma P_{ux})(b_0\Theta_x+\sigma P_{ux})'}{b_0^2+\sigma P_u+Q_u} + \sigma P_x\right]x\right\}.$$

The optimal input, minimizing this relation, is

$$u^* = -\frac{1}{b_0^2 + \sigma P_u + Q_u}(b_0\Theta_x + \sigma P_{ux})'x \tag{7.14}$$

and the optimal cost-to-go is a quadratic form in x:

$$\begin{aligned} \bar{K}_1^* &= \sigma + x'\left[\Theta_x\Theta_x' - \tfrac{(b_0\Theta_x+\sigma P_{ux})(b_0\Theta_x+\sigma P_{ux})'}{b_0^2+\sigma P_u+Q_u} + \sigma P_x\right]x = \\ &= \sigma + x'S(1|\mathcal{S})x. \end{aligned} \tag{7.15}$$

$\bar{K}^*$, as the smallest value of a nonnegative criterion, has to be nonnegative for any $\sigma \geq 0$ and x. Consequently, the kernel S has to be a nonnegative definite matrix.

7.2.5 Need for approximate evaluation

The second step of the dynamic programming

In the second step of the dynamic programming, the optimal cost-to-go would be

$$\bar{K}_2^* = \min_u \mathsf{E}[y^2 + u^2 Q_u + \sigma + x'S(1|\mathcal{S})x|\mathcal{X};u], \tag{7.16}$$

where

$$\begin{aligned}
\bar{K}_2^* &= K^*(k+H-1|\mathcal{X}(k+H-2)),\\
S(1|\mathcal{S}) &= S(1|\mathcal{S}(k+H|k+H-1)),\\
\mathcal{X} &= (x(k+H-2); \mathcal{S}(k+H-1|k+H-2)).
\end{aligned}$$

Evaluation of the first two terms of the above expression does not differ from the initial step. For the third term, with a slowly varying σ, it holds:

$$\begin{aligned}
&\mathsf{E}[\hat{\sigma}(k+H|k+H-1)|\mathcal{X}(k+H-2);\ u(k+H-1)] =\\
&= \mathsf{E}[\mathsf{E}[\sigma(k+H)|\mathcal{X}(k+H-1)]|\mathcal{X}(k+H+2)] =\\
&= \hat{\sigma}(k+H|k+H-2) = \hat{\sigma}(k+H-1|k+H-2)
\end{aligned}$$

The decisive difference between the first and the next steps of the dynamic programming is concentrated in the fourth term of (7.16).

Difficulty of evaluation

The nonlinear dependence of $S(1|\mathcal{S})$ on y and u prevents an analytical evaluation of the expectation and makes the minimization extremely difficult.

A reasonable solution is to replace $S(1|\mathcal{S})$ by an approximate kernel, which behaves as a constant with respect both to the expectation $\mathsf{E}[\,\cdot\,|\mathcal{X}; u]$ and to the minimization over u, in each step of optimization.

Approximation of the kernel

The approximation of the kernel is obtained by using an approximate statistics that guarantees the required invariance of the kernel. The simplest approximate statistics is constructed using the assumption that the statistics $\mathcal{S}(k+1|k)$ and $\mathcal{S}(k+1+i|k+i)$, $i = 1, 2, \ldots, H$ are near each other; in other words that the parameter estimates and their uncertainties are near some fixed values. This condition is fulfilled at least for large k and relatively short control horizon H (the slower the changes of parameters, the longer the horizon that is acceptable). The following form of the approximate statistics is convenient:

$$\mathcal{S}_a(k+1+i|k+i) = \mathcal{S}(k+1|k) \quad \text{for } i = 1, \ldots, H, \tag{7.17}$$

i.e. parameter estimates and their uncertainties are frozen at the values from the beginning of synthesis.

Using the approximate statistics, the dynamic programming equation reads

$$\bar{K}_a = \min_u \mathsf{E}[q_y + q_u + \tilde{K}_a^* | \, \mathcal{X}_a; u] \tag{7.18}$$

where

$$\begin{aligned}
\bar{K}_a^* &= K_a(i|x(i-1); \mathcal{S}_a(i|i-1)), \\
\tilde{K}_a^* &= K_a^*(i+1|x(i); \mathcal{S}_a(i+1|i)), \\
\mathcal{X}_a &= (x(i-1); \mathcal{S}_a(i|i-1)).
\end{aligned}$$

The recursion runs over $i = k_0 + H, \ldots, k_0$ with zero initial conditions.

Since the approximate optimal cost-to-go is a quadratic form, it reduces the equation (7.18) to an algebraic recursion on the kernel S.

7.2.6 Remark on notation

Since we shall work only with approximate statistics in the following text and to simplify notation, we shall not denote explicitly its use.

7.2.7 Control strategies

Cautious strategy

The above described strategy penalizes the data contained in regressor φ by a term $\sigma\varphi' P \varphi$. That means that the higher the uncertainty about the unknown parameters, the more cautious are the actions for this strategy. For this reason this strategy is called the *cautious strategy.* It has an advantage of smooth behaviour of the controller during the transient stages.

However, the assumption (7.17) does not decrease the uncertainty in time, while in reality the uncertainty decreases as we accumulate the data from the system. The pessimistic assumption about the time development of the uncertainty may result in an excessively cautious behaviour of the controller, which, in consequence, will excite the system insufficiently to decrease the uncertainty of the parameters estimates.

n-steps cautious strategy

For this reason, if the cautious behaviour of the controller is required, we use some approximation of decrease of the estimate uncertainty (that is, we approximate decrease of the covariance matrix P (7.4)).

The easiest approximation is to set the covariance matrix P to zero after some time (to 'switch off' the uncertainty), i.e. to use the following rule:

$$P_a(k+1+i|k+i) = \left\langle \begin{array}{ll} P(k+1|k) & \text{for } i = 1,2,\ldots,n \\ 0 & \text{for } i = n+1, n+2, \ldots, H \end{array} \right. \tag{7.19}$$

The other parameters of the approximate statistics remain unchanged:

$$\left. \begin{array}{rcl} \sigma_a(k+1+i|k+i) & = & \sigma(k+1|k), \\ \Theta_a(k+1+i|k+i) & = & \Theta(k+1|k) \end{array} \right. \quad \text{for } \; i = 1,2,\ldots,H. \tag{7.20}$$

This approximation expresses our confidence that the uncertainty of the estimate will be negligible after the next n steps of control design. Usually $n = 1$ is used, which is sufficient to prevent excessive action of the controller, while decreasing substantially the danger of passivity. We call this version the *n-steps cautious strategy.*

Certainty equivalence strategy

Often, the *certainty equivalence strategy* is used to simplify further the computations and to improve the excitation of the control loop. This strategy gives most confidence to the parameter estimate. The estimate is considered (for the control design) to be the true value of parameters, $\Theta = \hat{\Theta}$, $\sigma = \hat{\sigma}$ and the uncertainty of the estimate is neglected, $P \stackrel{!}{=} 0$.

The disadvantage of this strategy is the danger of excessive overshoots of the system input in the transient phase due to its great confidence.

Other way to limit excessive signals on the system input, is to limit the output of the controller [2].

7.2.8 Summary

In this section, we have introduced the parameters of the GiW-distribution to the dynamic programming formulae. We have shown that to make the computations of the dynamic programming recursion feasible, it is necessary to use an approximate statistics for modeling of the system evolution. The simplest approximation is fixing of parameters of the statistics at the beginning of recursion. It reduces the dynamic programming to a recursive minimization of a quadratic form.

At the end of the section, possible ways of handling the development of parameter estimate uncertainty were discussed.

7.3 A square-root LQG control design

In this section, we shall derive a square-root algorithm for the LQG control synthesis.

This algorithm is popular for its favourable numerical properties (robustness, guaranty of the positive semidefiniteness of the kernel S (7.15), etc.).

We shall first rewrite the system model using the *the controllable canonical form* of the state-space description. Using the state-space description, we shall express the loss function $q_y + q_u$ in (7.18) as a quadratic form of the state and rewrite the kernel of the control criterion in a square-root form as a product of a triangular matrix and its transposition. We shall show, that using this matrix, it is possible to compute the iterations of the control design by orthogonal rotations.

This description will serve as a base to explain the function of the systolic implementation of the algorithm in the next chapter.

Remark: In the following derivations, we shall assume the beginning of the control horizon at $k_0 = 1$ to simplify the notation, if not stated otherwise.

7.3.1 Controllable canonical form of state-space description

The ARX model (5.13) may be rewritten using the state-space description:

$$\begin{aligned} \bar{x} &= A\tilde{x} + bu + ce, &(7.21)\\ y &= d'\bar{x}, &(7.22) \end{aligned}$$

where matrix A and vectors b, c, and d have the following form:

$$A = \begin{bmatrix} & \Theta_x' & & \\ 0 & \dots & & 0 \\ 1 & & 0 & 0 \\ & \ddots & \vdots & \vdots \\ & & 1\;\; 0 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} b_0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad c = d = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad (7.23)$$

and vector $\tilde{x} = x(k-1)$ is the state vector (7.5). In our derivations, we shall use the state-space model in a rearranged form:

$$\bar{x} = M \begin{bmatrix} u \\ \tilde{x} \end{bmatrix} + ce, \qquad (7.24)$$

where M is composed of A and b:

$$M = [b\ A] = \left[\begin{array}{ccc} & \Theta' & \\ I & 0 & 0 \end{array} \right]. \tag{7.25}$$

Remark: Notice that in the state-space model that is used here (7.21), (7.22), the output is computed from the *updated* state $\bar{x}$. This convention follows [2]. It is correct in the physical sense, since the delay, which must be contained in the system-controller loop, is contained in the controller.

7.3.2 The first step of dynamic programming

Let us recall that using the parameters of the GiW-distribution (7.9), the first step of the dynamic programming reads:

$$\begin{aligned} \bar{K}^* &= \min_u \left[(\Theta'\varphi - w_y)^2 Q_y + (u - w_u)^2 Q_u + \sigma + \sigma\varphi' P \varphi + \tilde{K}^* \right], \\ \tilde{K}^* &= 0. \end{aligned}$$

This relation may be rewritten in the following form:

$$\bar{K}^* = \min_u \left\{ [u\ \tilde{x}'\ 1] \tilde{\Phi}_1' \tilde{\Phi}_1 [u\ \tilde{x}'\ 1]' \right\}, \tag{7.26}$$

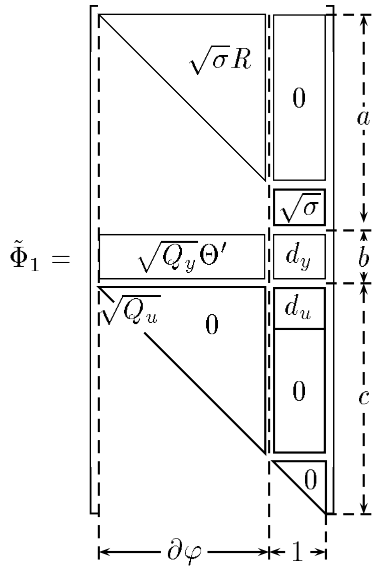
where $\tilde{x}$ denotes $x(H)$. Matrix $\tilde{\Phi}_1$—the square root form of the kernel of the quadratic form, is composed of several submatrices, as shown in Fig. 7.1. The contents of the zero submatrices, denoted explicitly in the scheme of matrix $\tilde{\Phi}_1$, is affected during the minimization procedure.

The submatrices that are not explicitly denoted, contain zeros.

Minimization of the quadratic form. To minimize the quadratic form, we shall first transform the matrix $\tilde{\Phi}_1$ to an upper triangular matrix $\bar{\Phi}_1$, using a suitable orthogonal transformation matrix T, zeroing parts a and b w.r.t. the part c (Fig. 7.1):

$$\bar{\Phi}_1 = T\tilde{\Phi}_1, \;\; TT' = I. \tag{7.27}$$

For this transformation, we shall use standard QR algorithm with Givens transformations [11, 22, 33]. Description of this algorithm is given in the next chapter.



$$P = R'R,$$
$$d_y = -\sqrt{Q_y}w_y,$$
$$d_u = -\sqrt{Q_u}w_u$$

Figure 7.1: Composition of matrix $\tilde{\Phi}_1$

With the triangularized matrix, the control criterion (7.26) reads:

$$\begin{aligned} \bar{K}^* &= \min_u \left\{[u\ \tilde{x}'\ 1]\bar{\Phi}_1' T' T \bar{\Phi}_1 [u\ \tilde{x}'\ 1]'\right\}, \\ &= \min_u \left\{[u\ \tilde{x}'\ 1]\bar{\Phi}_1' \bar{\Phi}_1 [u\ \tilde{x}'\ 1]'\right\} \end{aligned} \tag{7.28}$$

Let us partition the matrix $\bar{\Phi}_1$, as denoted in Fig. 7.2. The control input u affects the control criterion only through the row, containing the vector $[\gamma\ \Gamma\ \Gamma_c]$ and we can minimize its value by setting

$$u = -\frac{1}{\gamma}[\Gamma\ \Gamma_c][\tilde{x}\ 1]' = L[\tilde{x}\ 1]', \tag{7.29}$$

where L denotes the *control law*

$$L = -\frac{1}{\gamma}[\Gamma\ \Gamma_c] = [L\ L_c]. \tag{7.30}$$

The optimal (residual) value of the control criterion in the first step of the dynamic programming then reads:

$$\bar{K}^* = [\tilde{x}'\ 1][\bar{Z}\ \bar{z}]'[\bar{Z}\ \bar{z}][\tilde{x}'\ 1]' + \bar{o}^2. \tag{7.31}$$

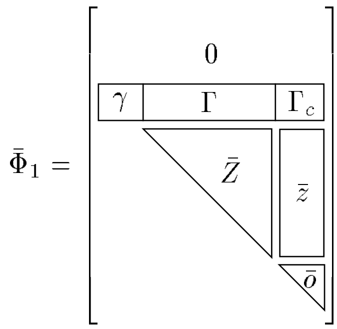


Figure 7.2: Composition of matrix $\bar{\Phi}_1$

Remark: Part a of matrix $\tilde{\Phi}_1$ (Fig. 7.1) is used only in the cautious control strategy. Using the certainty-equivalence strategy, this part is equal to zero and only parts b and c (Fig. 7.1) enter the triangularization procedure.

7.3.3 The subsequent steps of dynamic programming

In the second and subsequent steps of dynamic programming, the residual value of the control criterion from the last step is non-zero. The formula of the dynamic programming then reads:

$$\bar{K}^* = \min_u \{(\Theta'\varphi - w_y)^2 Q_y + (u - w_u)^2 Q_u + \sigma + \sigma\varphi' P\varphi + \tilde{K}^*\}. \tag{7.32}$$

Let us recall that

$\tilde{K}^*$ is the residual value of the criterion from the last step of programming,

$\bar{K}^*$ is the minimum value of the criterion in the given step.

Introducing (7.31) for $\tilde{K}^*$, we get:

$$\begin{aligned} \bar{K}^* \;=\; & \min_u \{(\Theta'\varphi - w_y)^2 Q_y + (u - w_u)^2 Q_u + \sigma + \sigma\varphi' P\varphi + \\ & + [\bar{x}'\ 1][\tilde{Z}\ \tilde{z}]'[\tilde{Z}\ \tilde{z}][\bar{x}'\ 1]' + \tilde{o}^2\}, \end{aligned} \tag{7.33}$$

where $\bar{x}$ is the value of x from the previous step of minimization.

Let us extend the matrix M (7.24, 7.25) so that we can write:

$$\begin{bmatrix} \bar{x} \\ 1 \end{bmatrix}' = \begin{bmatrix} M & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{x} \\ 1 \end{bmatrix}' + \begin{bmatrix} c \\ 0 \end{bmatrix} e = \underline{M} \begin{bmatrix} \tilde{x} \\ 1 \end{bmatrix}' + \underline{c}e, \tag{7.34}$$

where $\underline{M}$ is a composed matrix and $\underline{c}$ is a composed vector

$$\underline{M} = \begin{bmatrix} M & 0 \\ 0 & 1 \end{bmatrix}, \ \underline{c} = \begin{bmatrix} c \\ 0 \end{bmatrix}. \tag{7.35}$$

Performing expectation on (7.34) and using the result in (7.33), we may rewrite the formula as follows:

$$\begin{aligned} \bar{K}^* &= \min_u \{(\Theta'[u\ \tilde{x}']' - w_y)^2 Q_y + (u - w_u)^2 Q_u + \sigma + \sigma\varphi' P\varphi + \\ &\quad + [u\ \tilde{x}'\ 1]\underline{M}'[\tilde{Z}\ \tilde{z}]'[\tilde{Z}\ \tilde{z}]\underline{M}[u\ \tilde{x}'\ 1]' + o^2\} \\ &= \min_u \{[u\ \tilde{x}'\ 1]\tilde{\Phi}'\tilde{\Phi}[u\ \tilde{x}'\ 1]'\}. \end{aligned}$$

Matrix $\tilde{\Phi}$ is composed as shown on Fig. 7.3.

Simplification of computations on matrix $\tilde{\Phi}$. In the minimization step, we would transform the bottom part (d) of matrix $\tilde{\Phi}$ to a triangular form (let us denote the triangularized result by $\bar{\tilde{d}}$) and then zero parts a, b and c (Fig. 7.3) w.r.t. to the triangularized $\bar{\tilde{d}}$. Finally, we would compute the control law in the same way as in the first step of the control design.

The whole procedure, however, may be significantly simplified by exploiting the sparsity of matrix M, as introduced in (7.25).

Let us split first the matrix $\bar{\tilde{Z}}$ into the following submatrices:

$$\bar{\tilde{Z}} = \begin{bmatrix} \bar{\tilde{Z}}_1 & \bar{\tilde{Z}}_2 \\ & \bar{\tilde{Z}}_3 \end{bmatrix}, \tag{7.36}$$

where $\bar{\tilde{Z}}_1$ is scalar, $\bar{\tilde{Z}}_2$ is a row vector, and $\bar{\tilde{Z}}_3$ is an upper triangular matrix. (To preserve the consistency of notation with the original matrix, capital letters are used to denote scalar and vector).

Using the manipulations, given in Fig. 7.4, for reduction of part b w.r.t. the part d of matrix $\tilde{\Phi}$, and exploiting the structure of matrix M, the mul-

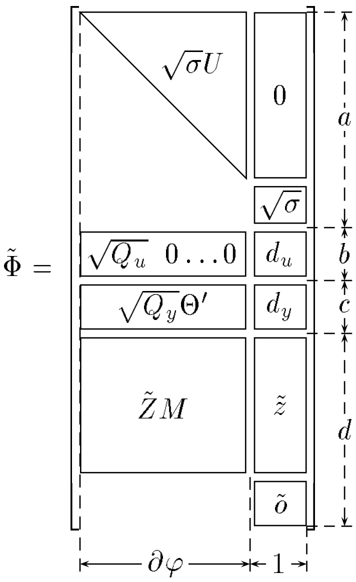


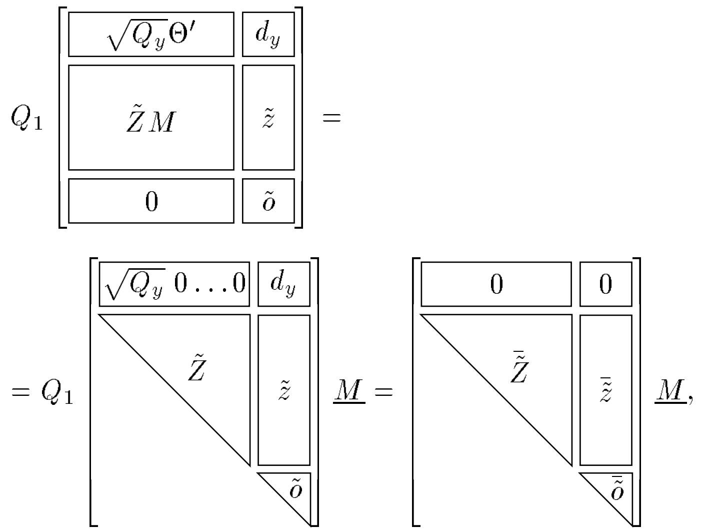
Figure 7.3: Composition of matrix $\tilde{\Phi}$

tiplication of matrix $\tilde{\bar{Z}}$ with matrix M (Fig. 7.4) significantly simplifies, as follows from this formula:

$$\tilde{\bar{Z}}M = \begin{bmatrix} \tilde{\bar{Z}}_1 & \tilde{\bar{Z}}_2 \\ & \tilde{\bar{Z}}_3 \end{bmatrix} \begin{bmatrix} & \Theta' & & \\ 1 & & 0 & 0 \\ & \ddots & \vdots & \vdots \\ & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \boxed{\tilde{\bar{Z}}_1\Theta' + [\tilde{\bar{Z}}_2\ 0\ 0]} & \\ \tilde{\bar{Z}}_3 & \begin{matrix} 0\ 0 \\ \vdots\ \vdots \\ 0\ 0 \end{matrix} \end{bmatrix} \qquad (7.37)$$

After these manipulations, matrix $\tilde{\Phi}$ (Fig.7.3) is transformed to matrix $\tilde{\bar{\Phi}}$ (Fig. 7.5). In $\tilde{\bar{\Phi}}$, the zero part denoted by e (two zero rows) has been added, so that there is space enough for the triangular matrix that will result from the minimization step.

Minimization. For minimization, we shall use exactly the same procedure as in the first step of programming (SubSec. 7.3.2)—by multiplication of $\tilde{\bar{\Phi}}$ by an orthogonal matrix T, we shall zero parts a, b and c (Fig. 7.5) w.r.t.



$$\partial\tilde{Z} = \partial\bar{\tilde{Z}} = (2l, 2l),\ \partial(\tilde{Z}M) = (2l, 2l+1),\ \partial\tilde{z} = \partial\bar{\tilde{z}} = (2l),$$
$$2l = \partial x(7.6).$$

Figure 7.4: Matrix manipulations to simplify the minimization step

the part d:

$$\bar{\Phi} = T\bar{\bar{\Phi}}, \tag{7.38}$$

with $\bar{\Phi}$ having the same structure as matrix $\bar{\Phi}_1$ in Fig. 7.2.

The *control law* minimizing the control criterion will be computed exactly as described for the first step of control design.

Remarks.

- As in the first step, part a is used only if implementing the cautious control strategy, otherwise it is set to zero.
- Since the residual o does not affect the control law, it is not necessary to compute it unless we are interested in its evolution.

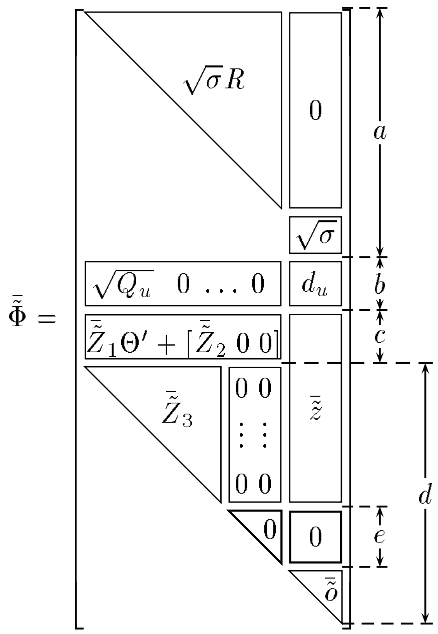


Figure 7.5: Composition of matrix $\bar{\tilde{\Phi}}$

7.4 Summary

The main result of this chapter is derivation of a square-root algorithm for LQ control design. The resulting algorithm may be summarized as follows:

$k = H$:

1. Compose matrix $\tilde{\Phi}_1$ (Fig. 7.1)
2. Perform the orthogonal rotation (7.27)

$k = H - 1, \ldots, 1$:

1. Perform the update $\tilde{Z} \rightarrow \bar{\tilde{Z}}$ (Second row in Fig. 7.4)
2. Compose matrix $\tilde{\Phi}$ (Fig. 7.3)
3. Perform the orthogonal rotation (7.38)

$k = 1$:

1. Compute the control law L (7.30) using the matrix $\bar{\Phi}$ (of the same structure as $\tilde{\Phi}_1$ (Fig. 7.2)), resulting from the minimizations

The control law L is actually computed only once — that is, at the beginning of the control horizon.

The algorithm exploits special structure of the system matrix in the controllable canonicle form to simplify the computations in the control synthesis. It is designed to cope with non-zero input and output setpoints. In the same way as the setpoints are treated, a non-zero static offset of the controlled system may be compensated.

In the next chapter, a systolic implementation of this algorithm will be described.

Chapter 8

Systolic controller

In this chapter, the systolic implementation of the LQ control algorithm, which was derived in the previous section, will be described. This implementation is similar to the arrays earlier described in [4, 5]. However, it differs from them in the way the penalties and setpoints are loaded. The array that is described here is expected to be suitable for later implementation of control with input restrictions.

The description of the systolic array consists of the following parts:

- implementation of the basic operations (vector addition, QR row vector reduction, etc.)
- synchronization of these operations in a single array
- detailed description of the array on the cell level

We suppose the use of the certainty–equivalence control strategy; at the end of the chapter we shall outline an extension for one-step approximation of the cautious strategy.

8.1 Systolic arrays for the basic operations

Going through the derivation of the square-root LQG algorithm, given in the previous chapter, we find that the generic step of the control design includes the following operations:

- Zeroing of a row vector w.r.t. a triangular matrix
- Multiplication of a vector by a scalar and addition of another vector

- Zeroing of a triangular matrix w.r.t. a triangular matrix (only with the cautious strategy) — this is, in fact, multiple application of the first item

8.1.1 QR array for row reduction

The update of an upper triangular (UT) matrix by a row vector by a series of orthogonal transformations, which was used in the previous chapter, is the standard QR algorithm for row reduction, well known from signal processing [11, 13, 22, 33].

QR algorithm for row reduction

Let us have an UT-matrix R (it has no connection with the upper triangular factor R of the covariance matrix, used in Chapter 5). The QR algorithm for row reduction performs such rotation of matrix R by an orthogonal matrix Q, $QQ' = I$, that the row vector x, added to matrix R, is zeroed and matrix R is updated:

$$\begin{bmatrix} 0 \\ \bar{R} \end{bmatrix} := Q \begin{bmatrix} x \\ \tilde{R} \end{bmatrix}, \tag{8.1}$$

where

x is the row vector,

$\tilde{R}$ is the UT matrix *before* update,

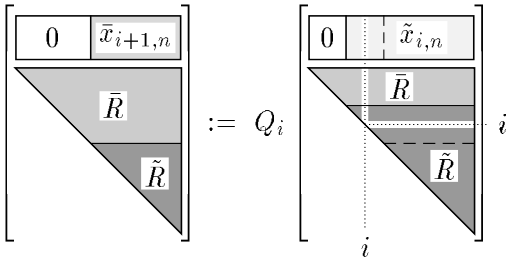
$\bar{R}$ is the UT matrix *after* update,

Q is the orthogonal transformation matrix.

Matrix Q is a product of n $(n+1 \times n+1)$ matrices of the Givens rotations, $n = \partial x$:

$$Q = Q_n Q_{n-1} \dots Q_1, \tag{8.2}$$

such that the rotation Q_1 is performed first and Q_n last. Each orthogonal rotation Q_i zeros the i-th element of the vector x and transforms appropriately the i-th line of the matrix $\tilde{R}$:



(8.3)

In this equation, $\tilde{x}_{i,n}$ denotes the non-zero part of vector x, *entering* the i-th step of transformation; $\bar{x}_{i+1,n}$ is the remaining non-zero part of the vector x *after* the i-th step of transformation. $\bar{R}$ is the *updated* part of matrix R, $\tilde{R}$ is the part of R that has not yet been updated.

The transformation matrix Q_i has the following structure:

$$Q_i = \begin{array}{cc} \left[\begin{array}{cccc} \cos\phi_i & \cdots & -\sin\phi_i & \cdots \\ \vdots & I & \vdots & \\ \sin\phi_i & \cdots & \cos\phi_i & \cdots \\ \vdots & & \vdots & I \end{array}\right] & \begin{array}{l} 1 \\ \\ i+1 \\ \\ \end{array} \\ \begin{array}{cccc} 1 & \qquad & i+1 & \qquad \end{array} & \end{array} \tag{8.4}$$

Sine and cosine of the rotation angle ϕ are computed from the formulae

$$\begin{aligned} \cos\phi_i &= \tilde{R}_{ii}/\sqrt{\tilde{R}_{ii}^2 + \tilde{x}_i^2} \\ \sin\phi_i &= \tilde{x}_i/\sqrt{\tilde{R}_{ii}^2 + \tilde{x}_i^2} \end{aligned}$$

The whole algorithm in the sequential form reads:

for $i = 1, \ldots, n$

  $C = \cos\phi_i = \tilde{R}_{ii}/\sqrt{\tilde{R}_{ii}^2 + \tilde{x}_i^2}$

  $S = \sin\phi_i = \tilde{x}_i/\sqrt{\tilde{R}_{ii}^2 + \tilde{x}_i^2}$

  $\bar{R}_{ii} = S\tilde{x}_i + C\tilde{R}_{ii}$

  for $j = i+1, \ldots, n$

    $\bar{x}_j = C\tilde{x}_j - S\tilde{R}_{ij}$

    $\bar{R}_{ij} = S\tilde{x} + C\tilde{R}_{ij}$

  end

end

Systolic implementation

The QR algorithm may be easily implemented on a systolic array. The array has a triangular shape where each PE stores one element of matrix R, as illustrated in Fig. 8.1. The vector x (to be reduced) enters the array from

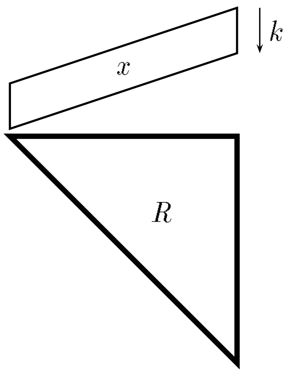
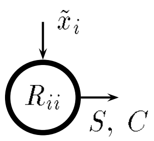


Figure 8.1: Scheme of the QR array for row reduction

the top, skewed in time, with the leftmost element entering the array first, and passes down through the array. In each row of the array, this vector is rotated with the row so that the element of the vector, which is in diagonal cell, is zeroed. The remaining elements are passed to the next row of the array.

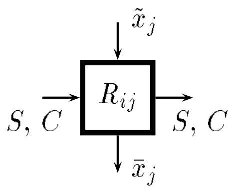
The sines and cosines of the rotation angles are calculated in the diagonal cells and through the horizontal links passed rightwards.

The scheme of the diagonal cell is in Fig. 8.2, of the off-diagonal cell in Fig. 8.3.



$$
\begin{aligned}
S = \sin\Phi &:= \tilde{x}_i/\sqrt{(\tilde{x}_i)^2 + (\tilde{R}_{ii})^2} \\
C = \cos\Phi &:= \tilde{R}_{ii}/\sqrt{(\tilde{x}_i)^2 + (\tilde{R}_{ii})^2} \\
\bar{R}_{ii} &:= S\tilde{x}_i + C\tilde{R}_{ii}
\end{aligned}
$$

Figure 8.2: Function scheme of the diagonal cell



$$
\begin{aligned}
\bar{x}_j &:= C\tilde{x}_j - S\tilde{R}_{ij} \\
\bar{R}_{ij} &:= S\tilde{x}_j + C\tilde{R}_{ij}
\end{aligned}
$$

Figure 8.3: Function description of the off-diagonal cell

Data pipelining

To achieve maximum efficiency of the use of the array, the data flow through the array is pipelined. The propagation of the pipelined data and of the rotation angles through the array is sketched in Fig. 8.4. The underlined part of index of an element of vector x denotes number of completed orthogonal rotations of this element (cf. Subsec. 6.1.2).

8.1.2 Systolic implementation of the scalar-vector multiplication and of the vector addition

The scalar-vector multiplication and the vector addition (Fig. 7.5, part c of the matrix $\bar{\bar{\Phi}}$) is implemented on a row of PEs: the configuration of inputs and outputs of the row is sketched in Figure 8.5 (the output vector is drawn by a dashed line). The elements of the multiplied vector are stored in the PEs and the multiplication coefficient passes through the row from left to right, synchronized with the vector to be added.

The function of the processing cells is described in Figure 8.6.

8.1.3 Summary

In this section, systolic implementation of two mechanisms used in the square-root algorithm for the LQ control design, that is of the QR algorithm for row reduction and of the scalar-vector multiplication and vector addition, was described.

8.2 Putting it all together ...

Having explained the principle of the algorithm as well as the systolic implementation of operations used in the algorithm, we are ready to describe

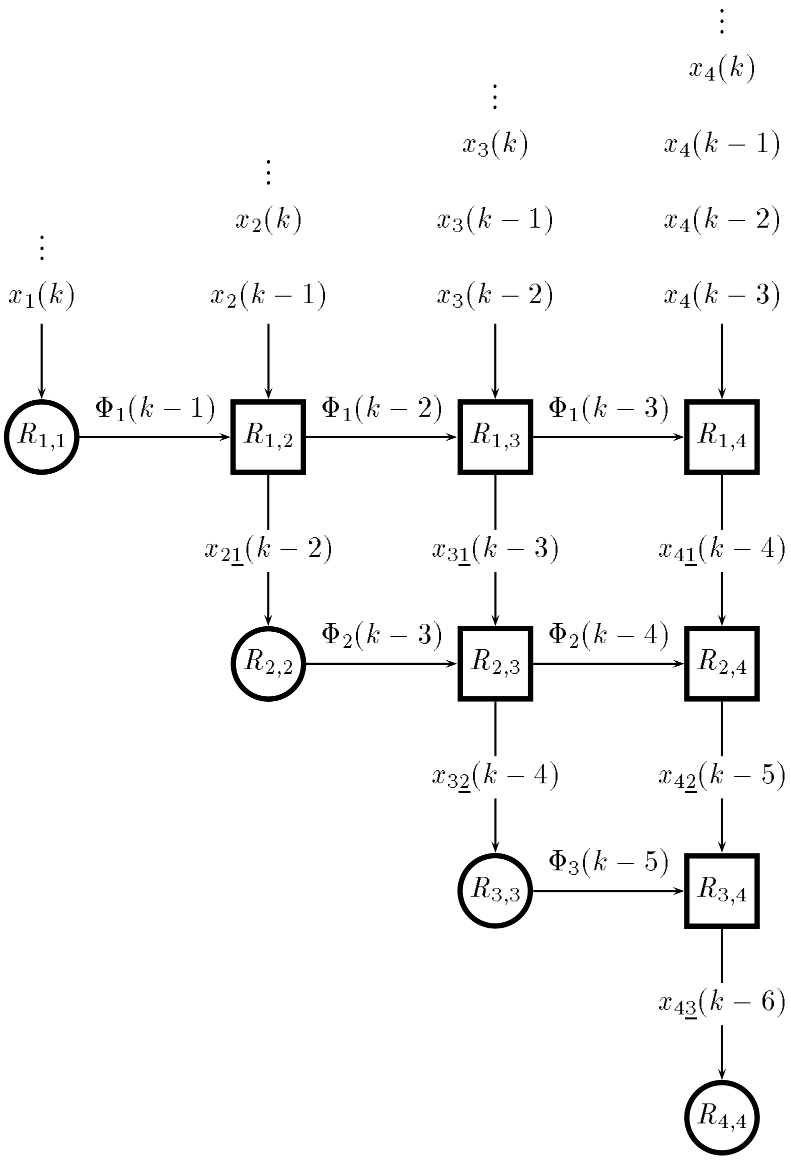


Figure 8.4: Data pipelining in the QR array for row reduction

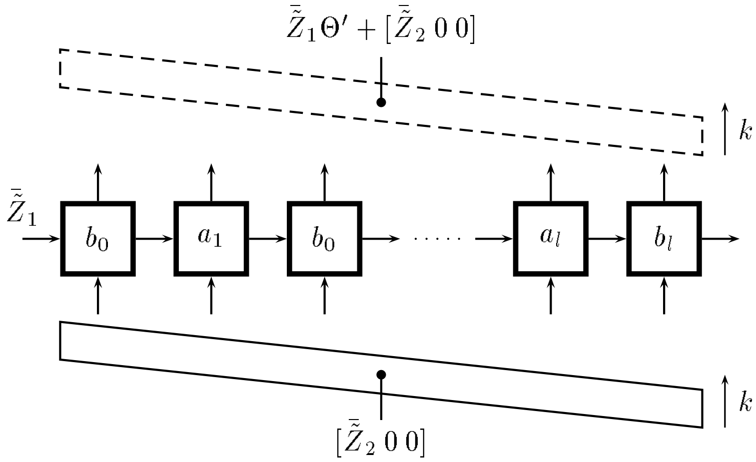
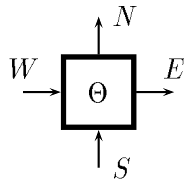


Figure 8.5: Systolic implementation of the scalar-vector multiplication and vector addition



I/O assignment:

N: $\bar{\tilde{Z}}_1\Theta + \bar{\tilde{Z}}_2$

S: $\bar{\tilde{Z}}_2$

W: $\bar{\tilde{Z}}_1$

E: $\bar{\tilde{Z}}_1$

Cell functionality:

$$E := W$$
$$N := W * \Theta + S$$

Figure 8.6: Function of processor for the scalar-vector multiplication and vector addition

function of the resulting systolic array for control design.

8.2.1 Mapping of variables

The whole LQ control algorithm is implemented on an upper triangular array of processors. Mapping of the variables contained in matrix $\Phi = \Phi_1$ (Fig. 7.2) (that is, after the minimization step) onto the array is shown in

Fig. 8.7. Vector z is mapped onto the diagonal of the array—the advantage of this mapping is that the vector is shifted not upward but along the diagonal, in the same way as the elements of matrix $\bar{\bar{Z}}_3$.

The figure also shows the time skew of the control law L (7.30) emerging from the array. The time skew of L on the output of the array is given by the mechanism of propagation of the rotations through the array (Fig. 8.11, 8.13 – 8.19). Scalar $\bar{o}$ is not contained in the figure, because it is not computed. The array has a dimension $(\partial\,\Theta \times \partial\,\Theta)$.

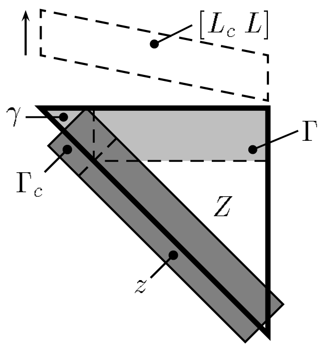


Figure 8.7: Position of variables in the LQ array after minimization, output of control law

8.2.2 Loading of parameters, penalties and setpoints

The parameter estimates Θ are loaded into the array always during the first iteration of the dynamic programming. They are mapped onto the upper row of the array.

The penalties and setpoints are entered in each iteration. They are loaded through the upper left corner of the array (see Fig. 8.8).

8.2.3 Matrix shifts between iterations

Fig. 8.9 shows implementation of shifts (7.37) of the components of matrix Z and of vector z in the array during the preparation of the next minimization step. Since also the reduction of penalty $\sqrt{Q_y}$ (transformation described in the second row of Fig. 7.4) is performed during these shifts, they actually involve transition from the matrix $\bar{\Phi} = \bar{\Phi}_1$ (Fig. 7.2) to the matrix $\bar{\bar{\Phi}}$

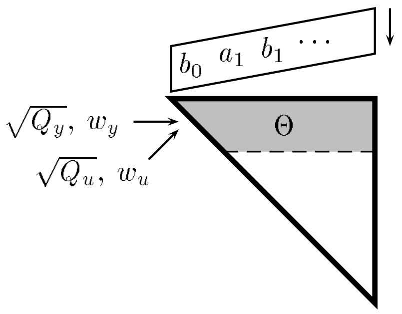


Figure 8.8: Loading of parameters, setpoints and penalties

(Fig. 7.5)). z_1 denotes the top element of z, $z_{2,n}$ denotes the rest of the vector.

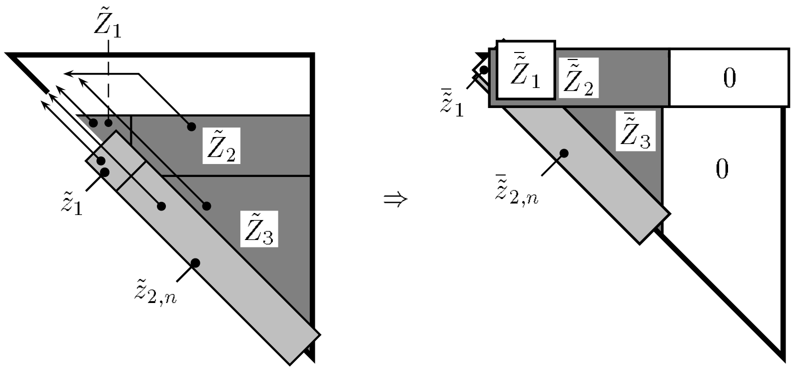


Figure 8.9: Shift of matrices between iterations

Row c of matrix $\bar{\tilde{\Phi}}$ (Fig. 7.5) is mapped onto the same row of the systolic array as the first row of submatrix $\bar{\tilde{Z}}_3$ (contained in part d of matrix $\bar{\tilde{\Phi}}$).

Hence, $\tilde{z}_1$ and $\tilde{Z}_1$ are shifted one position up along the main diagonal (and transformed to $\bar{\tilde{z}}_1$ and $\bar{\tilde{Z}}_1$) during the transition from $\bar{\Phi}$ to $\bar{\tilde{\Phi}}$; $\tilde{Z}_2$ is shifted one position along the diagonal and one position left (and also transformed) and finally $\tilde{z}_{2,n}$ and $\tilde{Z}_3$ are shifted two positions up along the diagonal (and transformed).

8.2.4 Interprocessor connections

The processing elements are connected into a hexagonal network, as sketched in Fig. 8.10. The arrows denote the direction of the signal flow. For proper synchronization of the data flow, the feedback connections in the upper row and the diagonal connections contain delays.

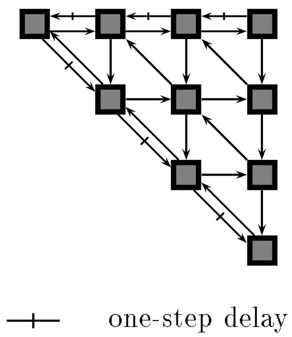
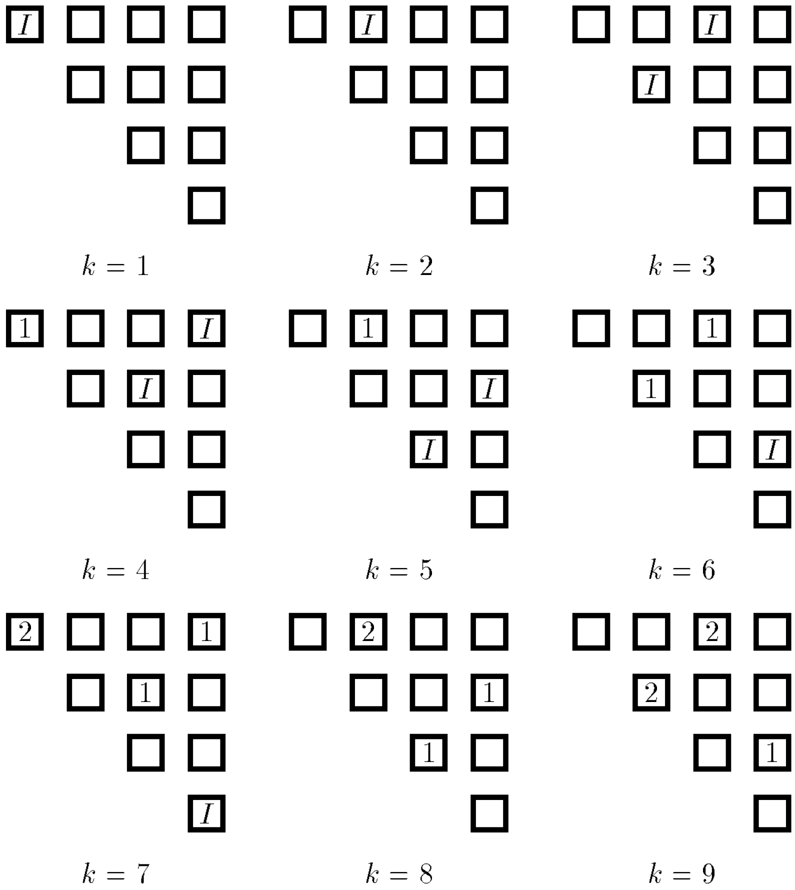


Figure 8.10: Hexagonally connected network of the LQ design array

8.2.5 Operating modes of processors

During the first step of the dynamic programming, the processors work in an *initialization mode*. During the subsequent iterations they work in two modes — in the *first mode*, they shift the matrices (Fig. 8.9), perform the reduction of $\sqrt{Q_y}$ and d_y (second row of the expression in Fig. 7.4) and the processors in the upper row perform the multiplication $\bar{\tilde{Z}}_1\Theta'$ (part c of matrix $\bar{\tilde{\Phi}}$ — Fig. 7.5). In the *second mode*, the processors in the upper row add the vector $[\bar{\tilde{Z}}_2\ 0\ 0]$ (that is, they complete the computation of the expression in the row c of matrix $\bar{\tilde{\Phi}}$ — Fig. 7.5) and the whole array performs the reduction of parts b and c of matrix $\bar{\tilde{\Phi}}$. In the last iteration, the upper row computes the control law L (7.30).

As a result of the dataflow in the array, the switching of the operating modes passes through the array in waves downwards along the main diagonal. To show this process, several steps of the control synthesis are traced in Fig. 8.11.



I – initialization mode

1 – 1st function mode

2 – 2nd function mode

Figure 8.11: Switching of function modes

8.2.6 Summary

In this section, the mechanisms used in the systolic implementation of the square-root LQ control design (loading of parameter estimates, setpoints and penalties, matrix shifts between iterations, interprocessor connections and operating modes of processors) were described.

The the control law computed by the array is the same as that computed by the sequential algorithm in [23].

8.3 Cell-level description of LQ array

Denotation of the processors

We use seven different types of PEs in the systolic array for the control design, denoted according to Fig. 8.12.

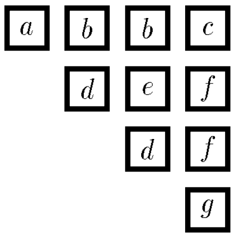
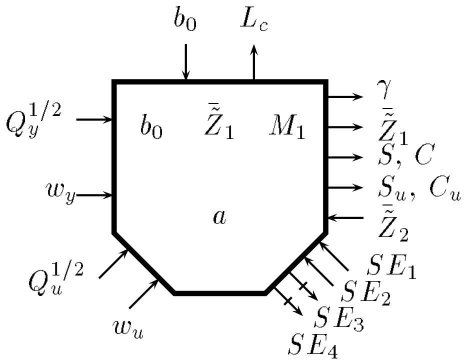


Figure 8.12: Notation of processors in the array for LQ control design

Description of processors

These processors are described in Figures 8.13 – 8.19.



Function mode 1

$\tilde{Z}_1 := SE_1$

$\tilde{z}_1 := SE_2$

$C := \tilde{Z}_1/\sqrt{Q_y + \tilde{Z}_1^2}$

$S := Q_y^{1/2}/\sqrt{Q_y + \tilde{Z}_1^2}$

$\bar{\tilde{Z}}_1 := S\,Q_y^{1/2} + C\,\tilde{Z}_1$

$M_1 := C\,d_y - S\,\tilde{z}_1$

$SE_3 := S\,d_y + C\,\tilde{z}_1$

Function mode 2

$d_u := -Q_u^{1/2} w_u$

$\tilde{x} := \bar{\tilde{Z}}_1 b_0 + \bar{\tilde{Z}}_2$

(auxiliary variable)

$\bar{\tilde{Z}}_3 := SE_1$

$\bar{\tilde{z}} := SE_2$

$C_u := \bar{\tilde{Z}}_3/\sqrt{Q_u + \bar{\tilde{Z}}_3^2}$

$S_u := Q_u^{1/2}/\sqrt{Q_u + \bar{\tilde{Z}}_3^2}$

$\bar{\tilde{Z}}_3 := S_u Q_u^{1/2} + C_u \bar{\tilde{Z}}_3$

$C := \bar{\tilde{Z}}_3/\sqrt{\bar{\tilde{Z}}_3^2 + x^2}$

$S := \tilde{x}/\sqrt{\bar{\tilde{Z}}_3^2 + \tilde{x}^2}$

$\gamma := S\tilde{x} + C\bar{\tilde{Z}}_3$

$\tilde{x}_w := S_u d_u + C_u M_1$

(auxiliary)

$SE_3 := C_u d_u - S_u M_1$

$L_c := (S\bar{\tilde{z}} + C\tilde{x}_w)/\gamma$

$SE_4 := \bar{x} = C\bar{\tilde{r}} - S\tilde{x}_w$

Initialization

$d_y := -Q_y^{1/2} w_y$

$d_u := -Q_u^{1/2} w_u$

$\tilde{x} := Q_y^{1/2} b_0$ (auxiliary)

$C := Q_u^{1/2}/\sqrt{\tilde{x}^2 + Q_u}$

$S := x/\sqrt{\tilde{x}^2 + Q_u}$

$\bar{\tilde{Z}}_1 := Q_y^{1/2}$

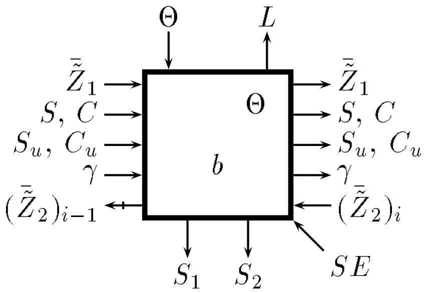
$\gamma := S\,\tilde{x} + C\,Q_u^{1/2}$

$L_c := (S\,d_y + C\,d_u)/\gamma$

$SE_3 := C\,d_y - S\,d_u$ $(\bar{z}_1)$

$\longrightarrow\!\!\!\!\!\!+$ one-step delay

Figure 8.13: Description of processor a



Initialization

$\Gamma := S\bar{\tilde{Z}}_1\Theta$

$S_2 := C\bar{\tilde{Z}}_1\Theta$

$L := \Gamma/\gamma$

Function mode 1

$\tilde{Z}_2 := SE$

$S_1 := -S\,\tilde{Z}_2$

$(\bar{\tilde{Z}}_2)_{i-1} := C\,\tilde{Z}_2$

Function mode 2

$\tilde{Z}_3 := SE$

$S_1 := -S_u\bar{\tilde{Z}}_3$

$\tilde{x} := \bar{\tilde{Z}}_1\Theta + (\bar{\tilde{Z}}_2)_i$

(auxiliary)

$\bar{\tilde{Z}}_3 := C_u\bar{\tilde{Z}}_3$

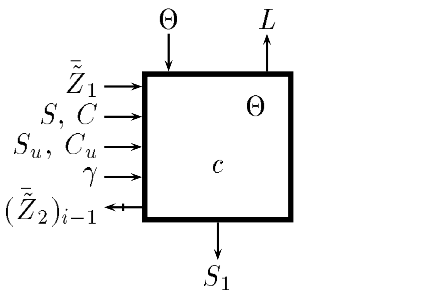
$\Gamma := S\,\tilde{x} + C\,\bar{\tilde{Z}}_3$

$L := \Gamma/\gamma$

$S_2 := \bar{x} = C\,\tilde{x} - S\,\bar{\tilde{Z}}_3$

—+— one-step delay

Figure 8.14: Description of processor b



Initialization

$\Gamma := S\,\bar{\tilde{Z}}_1\Theta$

$S_1 := C\,\bar{\tilde{Z}}_1\Theta$

$L := \Gamma/\gamma$

Function mode 1

$(\bar{\tilde{Z}}_2)_{i-1} := 0$

Function mode 2

$\tilde{x} := \bar{\tilde{Z}}_1\Theta$

(auxiliary)

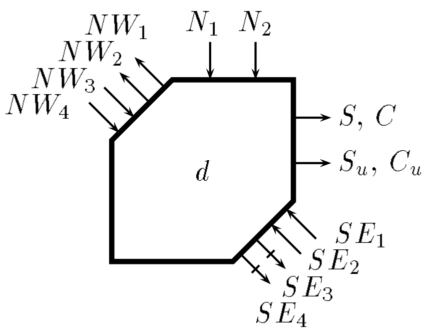
$\Gamma := S\,\tilde{x}$

$L := \Gamma/\gamma$

$S_1 := C\,\tilde{x}$

—+— one-step delay

Figure 8.15: Description of processor c



Function mode 1

$S_y := N_1/\sqrt{N_2^2 + SE_1^2}$

$C_y := SE_1^2/\sqrt{N_1^2 + SE_1^2}$

$NW_1 := S_y N_1 + C_y SE_1 \ (\bar{\tilde{Z}})$

$NW_2 := S_y NW_3 + C_y SE_2 \ (\bar{\tilde{z}})$

$SE_3 := C_y NW_3 - S_y SE_2$

Function mode 2

$S_u := N_1/\sqrt{N_1^2 + SE_1^2}$

$C_u := SE_1/\sqrt{N_1^2 + SE_1^2}$

$\Psi := S_u N_1 + C_u SE_1$

(auxiliary)

$\Psi_w := S_u NW_3 + C_u SE_2$

(auxiliary)

$SE_3 := C_u NW_3 - S_u SE_2$

$C := \Psi/\sqrt{\Psi^2 + N_2^2}$

$S := N_2/\sqrt{\Psi^2 + N_2^2}$

$NW_1 := SN_2 + C\Psi \ (\bar{Z})$

$NW_2 := SNW_4 + C\Psi_w \ (\bar{z})$

$SE_4 := CNW_4 - S\Psi_w$

Initialization

$\tilde{z} := NW_3$

$\tilde{x} := N_2$

(auxiliary)

$C := 0, \ S := 1$

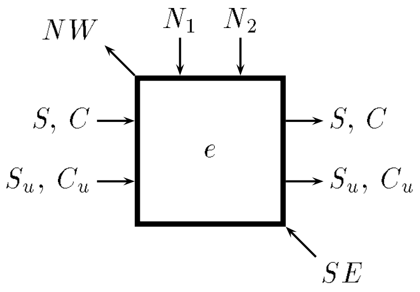
$NW_1 := \bar{Z}_1 = S\,\tilde{x} = \tilde{x}$

$NW_2 := \bar{z} = S\,NW_3 = NW_3$

$SE_3 := 0$

one-step delay

Figure 8.16: Description of processor d



Initialization

$\tilde{x} := N_2$
$\bar{x} := C\tilde{x}$
$S_1 := \bar{x}$
$NW := S\tilde{x}\ (\bar{Z}_3)$

x, Ψ: auxiliary variables

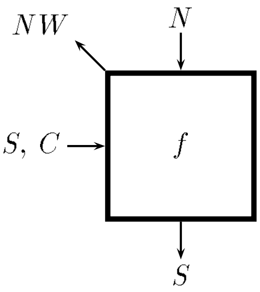
Function mode 1

$\tilde{x} := N_1$
$\bar{x} := C\,\tilde{x} - S\,SE$
$NW := S\,\tilde{x} + C\,SE\ (\bar{\bar{Z}}_3)$

Function mode 2

$S_1 := C_u\,N_1 - S_u\,SE$
$\Psi := S_u\,N_1 + C_u\,SE$
$\tilde{x} := N_2$
$\bar{x} := C\tilde{x} - S\Psi$
$NW := S\tilde{x} + C\Psi\ (\bar{Z}_3)$
$S_2 := \bar{x}$

Figure 8.17: Description of processor e



x: auxiliary variable

Initialization

$\tilde{x} := N$
$\bar{x} := C\tilde{x}$
$NW := S\tilde{x}\ (\bar{Z})$
$S := \bar{x}$

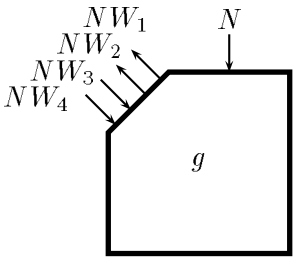
Function mode 1

$NW := 0\ (\bar{\bar{Z}})$

Function mode 2

$\tilde{x} := N$
$\bar{x} := C\tilde{x}$
$NW := S\tilde{x}\ (\bar{Z})$
$S := \bar{x}$

Figure 8.18: Description of processor f



Initialization

$\tilde{x} := N$

$C := 0$

$S := 1$

$NW_2 := NW_3$ $(\bar{z})$

$NW_1 := \bar{x} = S\tilde{x} = \tilde{x}$ $(\bar{Z}_3)$

x: auxiliary variable

Function mode 1

$S_y := 1$

$C_y := 0$

$NW_1 := 0$ $(\bar{\tilde{Z}}_3)$

$NW_2 := NW_3$ $(\bar{\tilde{z}})$

Function mode 2

$\tilde{r} = N$

$S_y := 1$

$C_y := 0$

$NW_1 := \bar{x} = S_y \tilde{x} = \tilde{x}$ $(\bar{Z}_3)$

$NW_2 := NW_4$ $(\bar{z})$

Figure 8.19: Description of processor g

8.4 Systolic computation of the system input

Having discussed the systolic implementation both of the regularized parameter estimator and of the control design, we have yet to describe the implementation of the last part of the adaptive controller (Fig. 4.1) — of generation of the system input u.

Let us recall that, having computed the control law L (7.30), we get the system input u from the relation (7.29):

$$u = L[\tilde{x}' \, 1]'.$$

In the systolic implementation, the system input may be computed on a row of processors, as drawn in Fig. 8.20, with the processor desriptions given in Fig. 8.21 and 8.22.

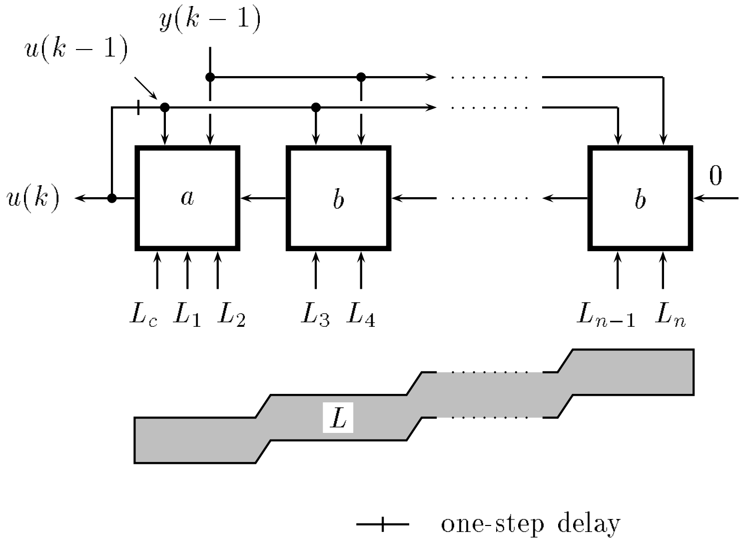
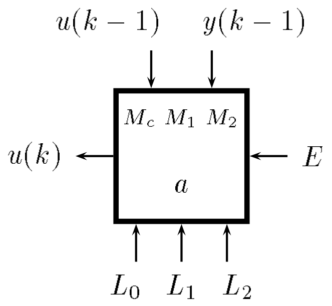


Figure 8.20: Systolic row for computation of the system input

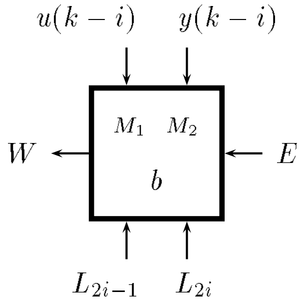
The control law enters the array from the bottom, skewed in time. Except for the first PE, where is the offset part L_c (7.30) of the control law entered, each PE is entered by two elements of the control law at the same time. This is denoted by the 'steps' on vector L in the scheme. We assume n even, otherwise we set $L_{n+1} \stackrel{!}{=} 0$.



Function:

if *new control law* then
  $M_0 := L_0$
  $M_1 := L_1$
  $M_2 := L_2$
end
  $u(k) := M_0 + E + M_1 y(k-1) +$
    $+ M_2 u(k-1)$

Figure 8.21: Processor a



Function:

if *new control law* then
  $M_1 := L_{2i-1}$
  $M_2 := L_{2i}$
end
if *rightmost cell* then
  $E := 0$
end
  $W := E + M_1 y(k-i) + M_2 u(k-i) + E$

Figure 8.22: Processor b

After the control law L enters the systolic array it is stored in memory in each PE (storage cells M_c, M_1, M_2 in processor a and storage cells M_1, M_2 in processor b) and held there until the new law is computed.

The computation of the new system input $u(k)$ is being completed in processor a, where the most recent system output $y(k-1)$ and input $u(k-1)$ enter. This is necessary so that the array does not introduce transport delay into the control loop.

8.5 Connecting the arrays together

To complete description of our systolic selftuner, it is necessary to consider the connection of the estimator to the controller and of both arrays to the above array for generation of the system input and to the controlled system itself. Also, the demands of proper synchronization of the estimator with the controller and of maximum utilization of both arrays set certain restrictions on selection of the control horizon length. All these questions will be solved in this section.

8.5.1 Connection of estimator to the controller

Comparing the time skew of parameter estimates on the output of the estimator (Fig. 6.7) with the required time skew of parameters on the entry of the controller (Fig. 8.8), we find that both arrays may be directly connected together, as is schematically sketched in Fig. 8.23.

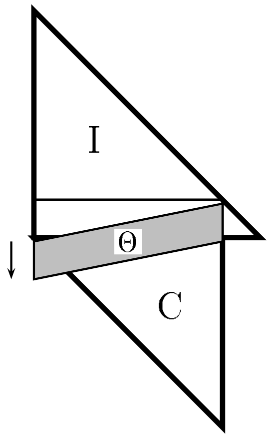


Figure 8.23: Connection of the estimator (I) with the controller (C)

To save one step of latency of the selftuner, which is spent by loading the estimates from the estimator to the controller, the processors in the top row of the controller may be mapped onto the processors in the bottom row of the estimator. However, the price paid for it is an increased computational load of the processors.

8.5.2 Synchronization of estimator with controller

For proper pipelining of the estimator with the controller and for maximum utilization of both arrays, the period of output of the control law (which equals to the period of loading of new parameter estimate into the controller) must be either the same or an integer multiple of the period of output of the regularized parameter estimates from the estimator.

The period of output of the control law from the controller (T_c) is equal to

$$T_c = 6H, \; H \geq 2, \tag{8.5}$$

where H is the length of the control horizon (decided by user).

The period of output of the regularized parameter estimates from the estimator (T_i) is equal to

$$T_i = 2(N + n), \tag{8.6}$$

where N is the number of samples in the data block (Tab. 6.1) and n is the number of regularizing data (i.e. the number of rows of matrix $U^{\lambda *}$ (6.4)).

Condition of synchronism. To meet the condition $T_c = cT_i$, c is a positive integer number, T_i is adjusted by setting the length of the data block N. The following relation holds for N:

$$N = \frac{3H}{c} - n, \quad N \geq n. \tag{8.7}$$

In the following text we shall suppose, for simplicity, $c = 1$.

8.5.3 Relative speed of the controlled system

Because the regularized estimator (Chapter 5, Section 6.3) processes not only the data samples from the controlled system, but also the regularizing data, the controlled system must be sampled at slower rate than the controller runs. The relative speed of the controller compared with the controlled system depends on the ratio $(N + n)/N$.

Let us denote the clock period of the controller (the period between the $(k-1)$-th and k-th step of the array) by t_c.

Let us further denote the sampling period of the controlled system by t_s. Then, the following relation holds for the ratio t_s/t_c (relative length of the system sampling period compared with the clock period of the controller):

$$t_s/t_c = 2(N + n)/N. \tag{8.8}$$

It follows that for $N = n$ (the length of data block is equal to the length of regularizing data), the ratio is:

$$t_s/t_c = 2(n+n)/n = 4,$$

i.e. the system is sampled four times as slow as the regularized estimator runs (this is the worst case).

For $N \to \infty$, the ratio tends to the value for exponential forgetting:

$$\lim_{N\to\infty} t_s/t_c = 2.$$

This means that the longer the data block compared with the block of regularizing data, the more the sampling period of the controlled system tends to the input period of the estimator, which is half its clock period. Were the exponential forgetting used in the estimator, the controlled system would be sampled at half the speed of the estimator and controller.

8.5.4 Buffer between the controller and the generator of the system input

The output of the array that generates the system input has to be synchronized with the controlled system.

The time skew of the control law must be reverted between the output of the controller (Fig. 8.7) and the input of the array computing the system input (Fig. 8.20), so that the selftuner does not introduce transport delay in the control loop.

To meet these requirements, there must be a buffer between the output of the controller and the input of the array generating $u(k)$, as drawn in Fig. 8.24. Writing to input of this buffer is controlled by the controller; reading from its output is controlled by the system input generator.

Delay in updating of the control law

Because the time skew of the control law is reverted between the controller and the generator of the system input, reading from the buffer to the generator may start only when the new control law has been completely written to the buffer. This means that it is necessary to wait until the whole control law was generated before it is possible to use it for computation of $u(k)$.

As was already stated, the clock period of the system input generator is equal to the sampling period of the controlled system, t_s (8.8), and is longer

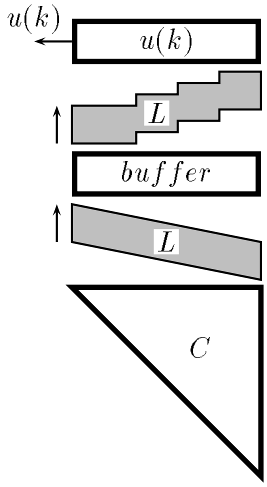


Figure 8.24: Buffer between the controller and the generator of the system input

than the clock period of the controller, t_c (Subsec. 8.5.3). Reading of the control law L from the buffer to the generator of the system input starts on the nearest sampling of the system after writing of the law to the buffer has been finished. That means that there is a delay d, $0 \leq d < t_s$ after L_n (last element of the control law) has been written from the controller to the buffer until it is read from the buffer to the input generator.

The delay after reading L_n from the buffer until the computation of the system input u with use of the new control law, equals to $t_s ceil(n/2)$ (function *ceil* rounds a number to the nearest greater integer).

The resulting delay from computation of L_c to application of the corresponding u to the controlled system is proportional to the order of the system and equals to

$$D = n + d + (t_s/t_c)ceil(n/2),\ 0 \leq d < t_s. \tag{8.9}$$

It is necessary to emphasize that this delay is in the path of the control law update (I – C – $u(k)$ in Fig. 4.1), not in the control loop itself (y – $u(k)$ – u – S in the same figure).

8.5.5 Minimum length of the control horizon

The function of the systolic arrays used in our selftuner set restrictions on the minimum length of the control horizon.

The first restriction is given by the minimum value of the output period of parameter estimates T_i (8.6), which, as follows from the function principles of the regularized estimator (Sec. 6.3), is equal to

$$\min T_i = 4n, \; n = \partial\Theta$$

Through the condition of synchronism (8.7), we get for the minimum length of the control horizon:

$$H \geq \frac{2}{3}n, \; H \in \mathcal{Z}. \tag{8.10}$$

The second restriction is given by the function of the control law buffer. The control law is written to this buffer every $6H$ steps (H is the length of the control horizon). It must not be changed until it is completely read by the system input generator. This is satisfied if the length of the control horizon meets the following condition:

$$6H \geq D, \tag{8.11}$$

where D is given by (8.9).

By a simple rearrangement, we get

$$H \geq \frac{1}{6}(n + d + (t_s/t_c)ceil(n/2)), \; H \in \mathcal{Z} \tag{8.12}$$

The resulting minimum length of the control horizon is determined by maximum of (8.10) and (8.12).

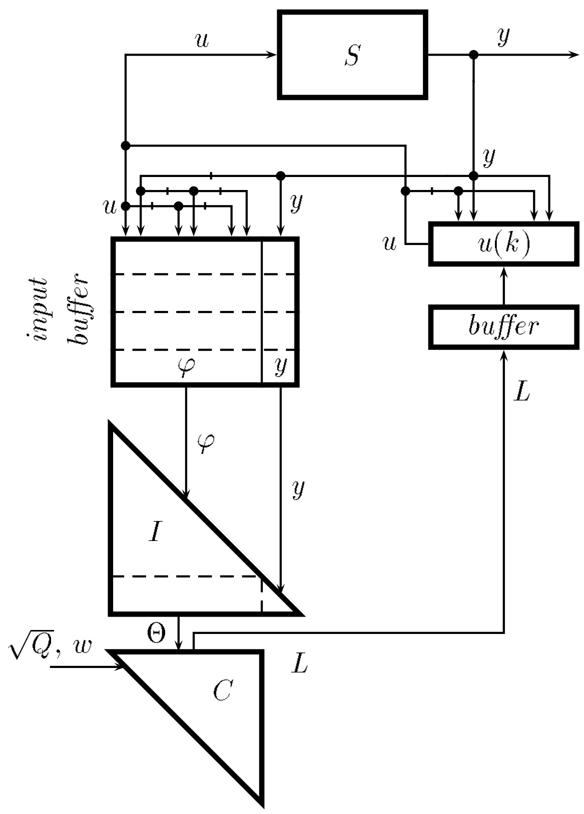
8.5.6 Summary

In this section, the questions of connecting the systolic LQ selftuner with the controlled system were considered. The necessity of buffer between the output of the controller and the input of the system input generator was explained. Finally, the limitations on the minimum length of the control horizon, resulting from the function of the block-regularized estimator and of the controller, as well as from the use of the control law buffer, were discussed.

8.6 Complete scheme of the adaptive controller

Having described all parts of the adaptive controller and their connections, we can give the complete scheme of the controller (Fig. 8.25).

It is necessary to emphasize that in spite of arrangement of the input buffer and estimator in the scheme, the data samples enter the estimator with no time skew.



Symbol —+— denotes one step delay element

Figure 8.25: Complete scheme of the adaptive LQ controller

For simplicity, the loading of the regularizing matrix into the estimator is not drawn in the figure, and neither are the buffer control links.

From the description of the estimator (Section 6.3.4) follows that the input buffer is of the FIFO (First-In-First-Out) type; the function of the output buffer was described in Section 8.5.4.

8.7 Timing of the controller

Let us finish the description of the adaptive controller by a summary of the most important relations concerning the timing of the controller and of the controlled system.

Length of the data block (8.7)

$$N = 3H - n$$

Sampling period of the system (8.8)

$$\frac{t_s}{t_c} = \frac{2(N+n)}{N}$$

Introducing the above relation for N:

$$\frac{t_s}{t_c} = \frac{6H}{3H-n}$$

Minimum length of the control horizon (8.12)

$$H \geq \max(\frac{2}{3}n,\ \frac{1}{6}(n + d + (t_s/t_c)ceil(\frac{n}{2}))),\ H \in \mathcal{Z}$$

8.8 Simulation example

In this section, we present the results of simulations of the systolic controller. For these simulations, an ARX model of the following structure was used:

$$y(k) = a_1 y(k-1) + a_2 y(k-2) + b_0 u(k) + b_1 u(k-1) + ce(k).$$

The coefficients of the model had the following values:

$$a_1 = 0.4,\ \ a_2 = -0.4,\ \ b_0 = 0.4,\ \ b_1 = 0.2,\ \ c = 0.02,$$

$e(k)$ was discrete normal white noise $\mathcal{N}(0,1)$.

8.8.1 Setting of parameters, common for all experiments

The setting of the following parameters was constant and common for all of the above described experiments:

Parameters of identification:

- The forgetting coefficient (5.47 e)

$$\lambda = 0.95$$

- The length of the regressor Θ (5.14, 5.70) in the identification algorithm

$$n = 4,$$

  i.e. the model used for estimation coincides with the simulated system.

- The matrix U^* (5.73) (the triangular factor of the regularizing matrix)

$$U^* = 0.3I$$

  Note: This is the value *before* multiplication by the accumulated weighting coefficient $\sqrt{\lambda_N}$ (5.49, 5.50, 5.80 b)

- The initial value of the covariance matrix factor R (5.23, 5.32, 5.51)

$$R(0) = 200I$$

Parameters of control design

- The length of the control horizon (Subsec. 4.1.2)

$$H = 6$$

- The input setpoint (4.1.1), (4.2)

$$w_u = 0$$

- The input and output penalty (4.8, 4.9)

$$\sqrt{Q_u} = 0.5, \quad \sqrt{Q_y} = 50$$

- The initial value of the control law L (7.30) (which is used before the first control law is computed by the systolic array)

$$L_c(0) = 0.5, \quad L(0) = [0.5\;0.5\;0.5]$$

The length of the simulation run was always set to

$$t = 400$$

measured in samples of the controlled system.

8.8.2 Description of the simulation experiments

The function of the systolic array was simulated using the X*lab* system [44, 46, 45], running on an AT-compatible computer.

Altogether, three simulation experiments are presented:

1. The first experiment was intended to show the influence of the regularization on the stability of control.

   This experiment involved two simulation runs, the first one with block-regularized identification, the second one with exponential forgetting.

   The parameters of the system, as well as the input and output setpoints, were kept constant in this experiment. The setpoints were set to the following values:

   $$w_y = 0.5, \quad w_u = 0.$$

2. The second experiment was intended to demonstrate the response of the systolic controller to a time variable output setpoint.

   In this experiment, only block-regularized identification was used. For the evolution of the setpoint, a periodic rectangular signal was used, described by the graph in Fig. 8.26.

3. In the third experiment, a time variable system was controlled. Also in this experiment a time variable output setpoint was used, with the same time development as in the previous experiment.

   The time variable parameter was a_1, the evolution of the parameter over the simulation run is shown in Fig. 8.27.

   In this experiment, both the block-regularized identification and the exponential forgetting were used.

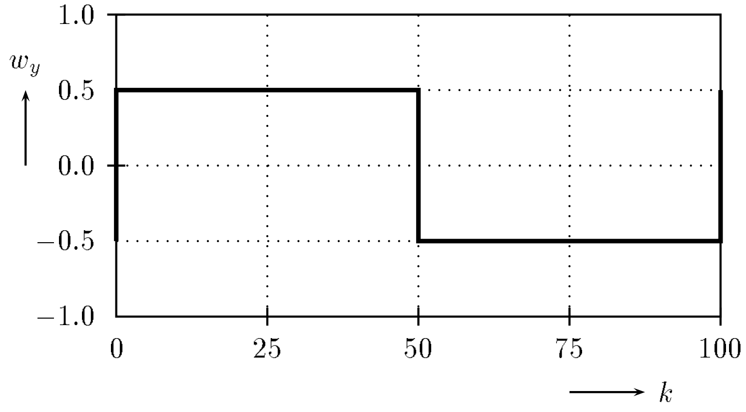


Figure 8.26: Time development of w_y

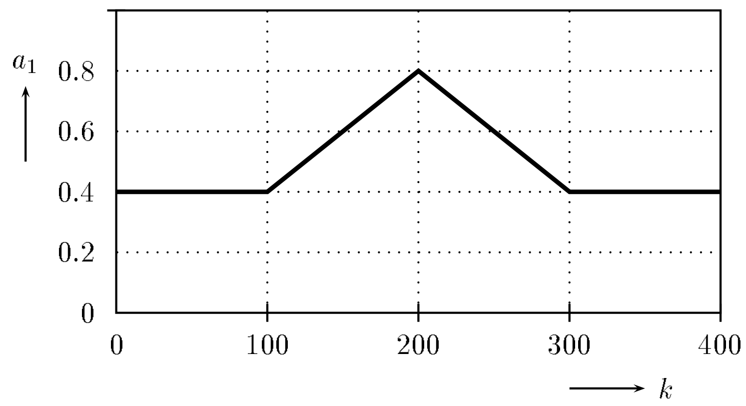


Figure 8.27: Time development of a_1

8.8.3 Timing of simulation

As explained in Section 8.5, for the block-regularized identification, the sampling period of the controlled system is longer than the clock period of the systolic controller due to processing of the regularizing data. Let us recall that the ratio of both periods is

$$t_s/t_c = 2(N + n)/N$$

and is, in general, non-integer.

Since we do not process any regularizing data when using the exponential forgetting, the sampling period of the system is always twice the clock period of the estimator, independent of the length of data regressor and of the control horizon.

In simulation, the sampling period of the controlled system, t_s, and the clock period of the controller, t_c, were implemented as integer multiples of an elementary simulation step (that is, the controlled system was sampled every t_s simulation steps and the clock period of the controller was t_c simulation steps). The following relations hold for them:

$$\begin{aligned} t_s &= 2(N+n)/gcd(N, 2(N+n)), \\ t_c &= N/gcd(N, 2(N+n)), \end{aligned}$$

where gcd denotes the greatest common divisor.

For exponential forgetting, $n = 0$, $t_s = 2$ and $t_c = 1$.

Expressed in elementary simulation steps, the length of the simulation is equal to

$$t_{es} = t\, t_s.$$

Using the regularized identification, we get the following parameters for our simulation example with the above given settings $H = 6$ and $n = 4$:

- the length of the data block N

  $$N = 14,$$

- the sampling and clock periods

  $$t_s = 18, \quad t_c = 7,$$

  expressed in elementary simulation steps.

For exponential forgetting, we get:

- number of data samples processed by the estimator between two outputs of control law from the controller

  $$N = 3H = 18,$$

- the sampling and clock periods

  $$t_s = 2, \quad t_c = 1.$$

Using regularized forgetting in the estimator, the length of the simulation run was equal to

$$t_{es} = 7200 \text{ steps.}$$

Using exponential forgetting, it was equal to

$$t_{es} = 800 \text{ steps.}$$

8.8.4 Remarks on simulation graphs

To show the results of the simulation experiments, two types of graphs are used for each experiment.

In the first type, the evolution of u and y is drawn, with the time axis scaled in the steps of the controlled system (i.e. the period of marks on the horizontal axis, denoted by k_s, is t_s).

In the second type, the evolution of parameter estimates is drawn. This type of graph is scaled in elementary simulation steps (i.e. the period of marks on the horizontal axis, denoted by k_{es}, is t_{es}). For easy comparison of both types of graphs, their horizontal dimensions are equal.

The marks on the horizontal axis denote the instants when the update of the control law in the system input generator begins (i.e. the instants, when the last—rightmost element of a new control law (L_n—Fig. 8.20) is read from the buffer to the generator—Fig. 8.24).

To make the graphs of evolution of parameter estimates better readable for experiments using exponential forgetting, only those values of estimates are plotted, that are used for the control design.

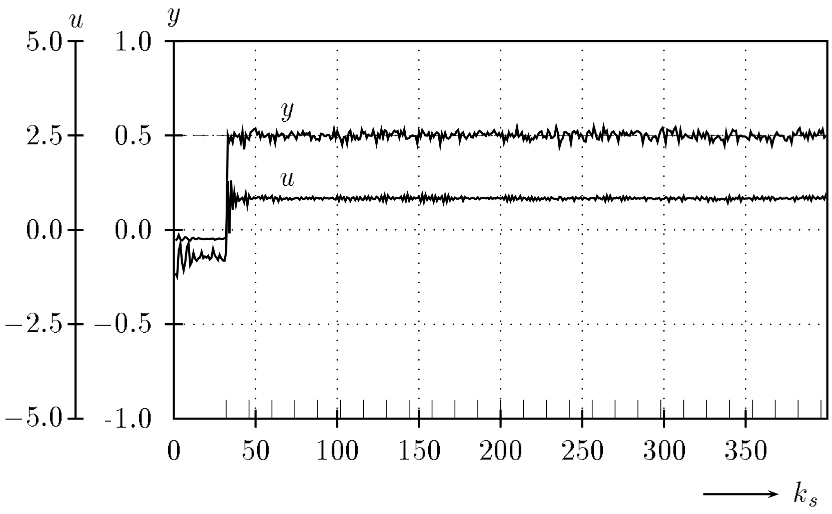
The evolution of setpoint and of actual values of parameters is plotted by a dashed line.

8.8.5 Comments on simulation results

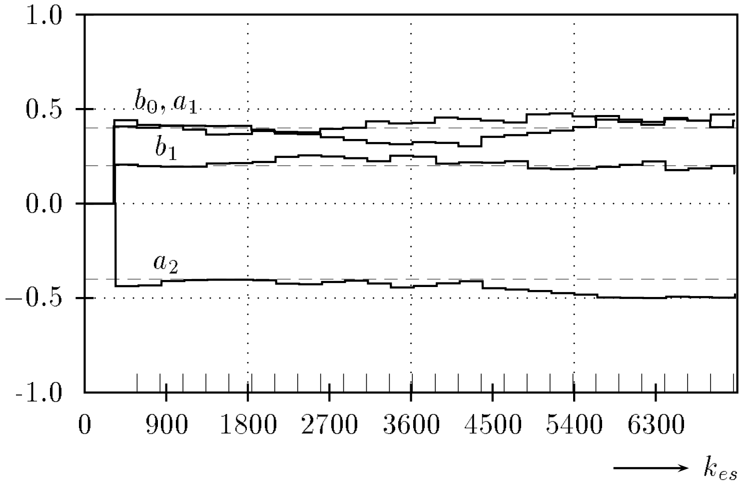
The results of the *first experiment* are shown in Fig. 8.28 (for block-regularized estimation) and in Fig. 8.29 (for estimation with exponential forgetting).

The results of the *second experiment* are shown in Fig. 8.30.

Finally, the results of the *third experiment* are shown in Fig. 8.31 (for block-regularized estimation) and in Fig. 8.32 (for estimation exponential forgetting).

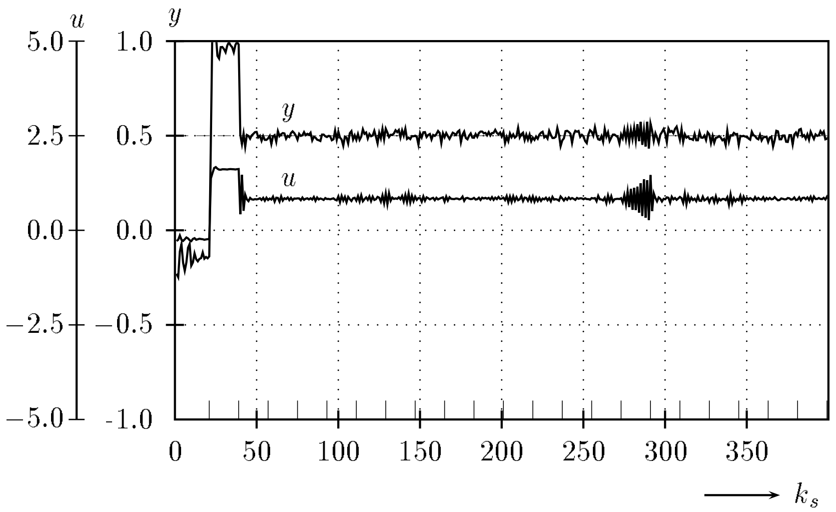


Development of u and y

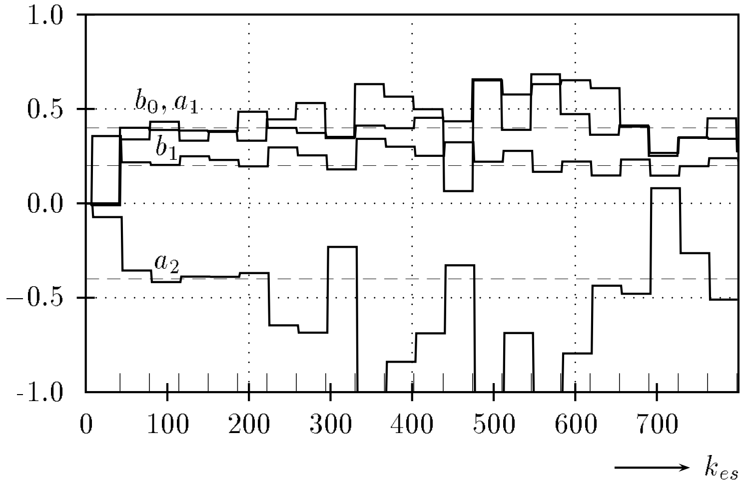


Development of parameter estimates

Figure 8.28: Fixed setpoint, block-regularized identification

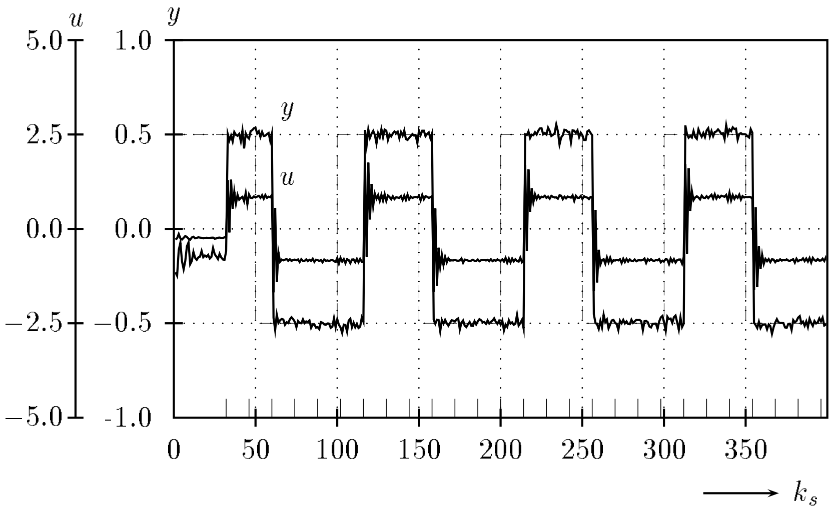


Development of u and y

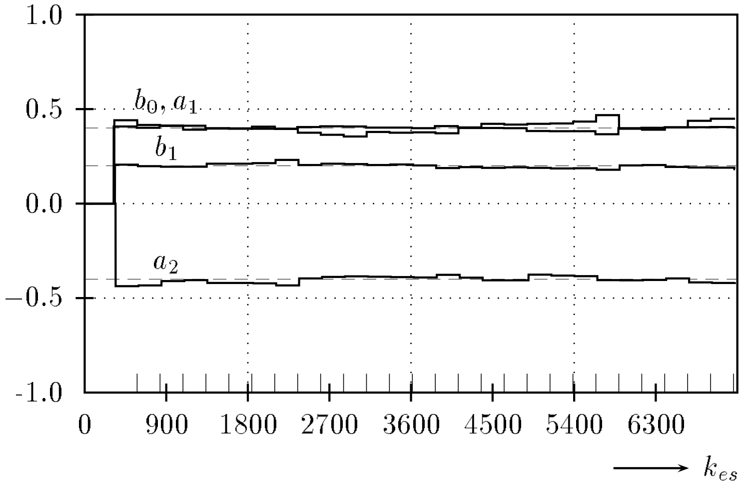


Development of parameter estimates

Figure 8.29: Fixed setpoint, exponential forgetting

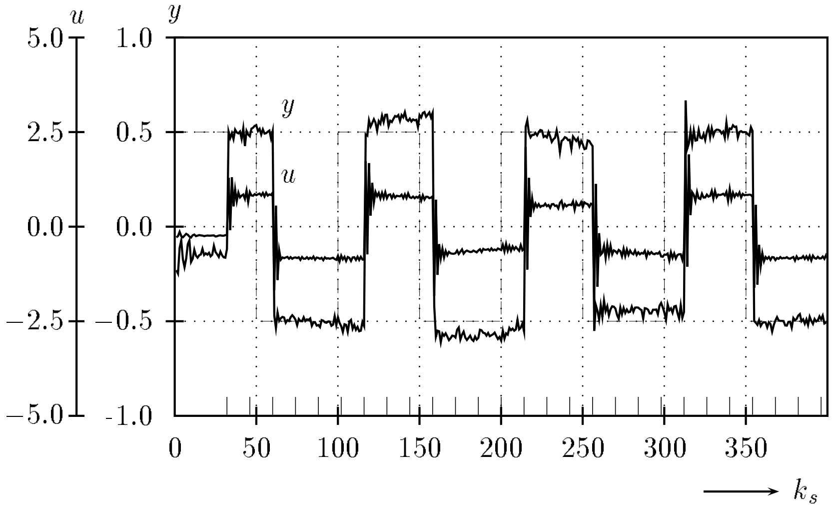


Development of u and y

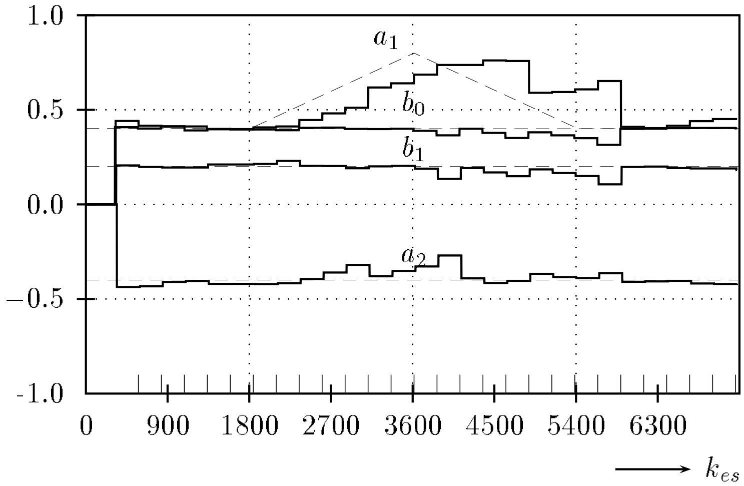


Development of parameter estimates

Figure 8.30: Rectangular evolution of setpoint, block-regularized identification

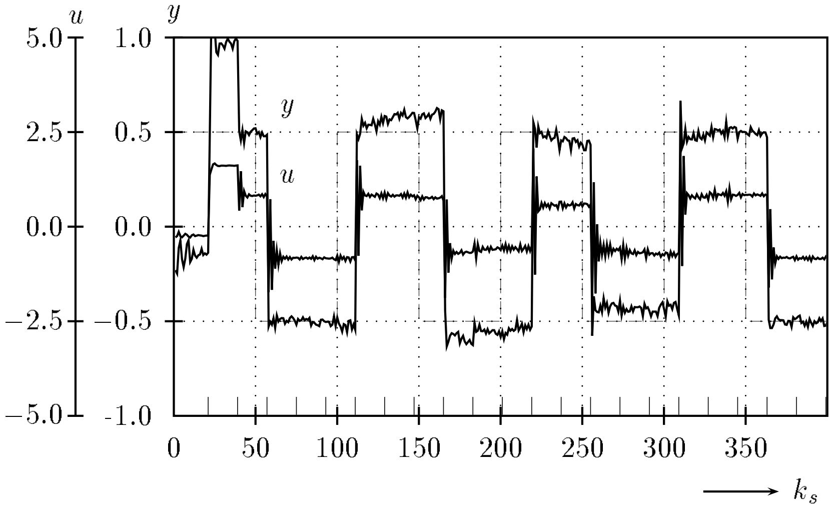


Development of u and y

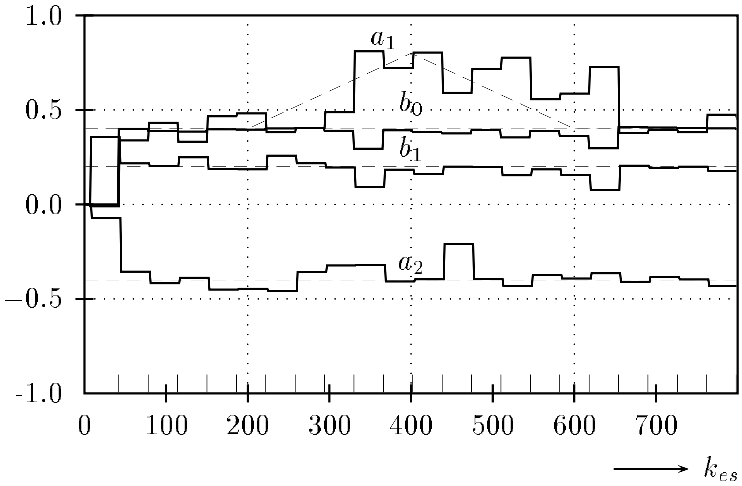


Development of parameter estimates

Figure 8.31: Rectangular evolution of setpoint, time-variable parameter, block-regularized identification



Development of u and y



Development of parameter estimates

Figure 8.32: Rectangular evolution of setpoint, time-variable parameter, exponential forgetting

Comments

- *Regularization* has *stabilizing influence on the control loop* (compare Fig. 8.28 and 8.29). In the first simulation run (with constant parameters and constant setpoints), it was also decreasing the dispersion of the parameter estimates.

  The experience from other simulation experiments shows that though the regularization does not guarantee its stability in all situations, it improves behaviour of the control loop, especially if the setpoints and/or system parameters are constant.

- The results of the second and of the third experiment show that the *systolic implementation introduces delay in the response* of the controller to the *change of setpoint.*

  The reason is following: the response to the change of setpoint is bounded to the period of computation of the control law. This period is equal to $6H$ clock periods of the systolic controller (where H is the length of the control horizon) or $3Hn/(n+N)$ sampling periods of the controlled system. During this time, the generator of the system input has to use constant control law and cannot respond to any change of setpoint.

  It can be easily seen from the graphs: the marks at the bottom part represent the instants, where the transfer of the new control law from buffer to input generator begins. The controller responds to the change of setpoint always after this mark (with $n/2$-steps delay, to be exact).

- The same is valid also for the *response to development of parameter estimates.* It cannot be see from the graphs of the third experiment very well, because the change of parameter between the control law readjustments was little, but it follows from the principle of the systolic controller.

8.9 Implementing cautious control strategy

In this section, main ideas of systolic implementation of the cautious control strategy (Subsec. 7.2.7) will be outlined, as one of the original contributions of the thesis. Due to the space limitations, no deeper details, like the cell descriptions, will be given. They may, however, be derived in a straightforward way. We hope that this way of description will be sufficient for reader having experience from study of the previous chapters.

Let us recall the structure of matrix $\bar{\tilde{\Phi}}$, minimized in the standard step of dynamic programming (Fig. 7.3). Using the cautious control strategy, it is necessary to perform reduction of matrix $\sqrt{\sigma}R$ (contained in the part a of matrix$\bar{\tilde{\Phi}}$) in one or more minimization steps.

It follows from comparison of position of matrix $\sqrt{\sigma}R$ in $\bar{\tilde{\Phi}}$ with position of matrix R in the formula for RLS data update (5.70) that R must be transposed before it is loaded from the estimator to the controller. To preserve the data pipelining in the controller, the reduction of matrix $\sqrt{\sigma}R$ may be performed only in the last step of dynamic programming. In that case, only the first row of this matrix influences the control law and it is not necessary to transpose and reduce the rest of matrix.

To implement this one step approximation (Subsec. 7.2.7) of the cautious control strategy, it is necessary to:

- compute the square root of estimate of covariance of the noise on system output $\sqrt{\sigma}$ (5.26),
- transpose the first column of the transposed factor of covariance matrix R', stored and updated in the systolic estimator (Fig. 6.6), to a row vector.
- multiply this vector (let us denote it by R_1) by $\sqrt{\sigma}$,
- load $\sqrt{\sigma}R_1$ to the array for the control design,
- perform reduction of $\sqrt{\sigma}R_1$. Using the receding horizon strategy, it is sufficient to perform only the rotation of the first row of the array, since the whole array is reinitialized in the next iteration.

This process may be implemented without adding any additional latency to the control law computation, only in the price of increase of computational burden of the processing elements.

8.9.1 Computation of $\sqrt{\sigma}$

To compute the estimate of $\sqrt{\sigma}$, the square root form of the formula (5.26) is used:

$$\sqrt{\sigma} = \sqrt{\Lambda}/\sqrt{\nu - 2},$$

where $\sqrt{\Lambda}$ is square root of (5.33).

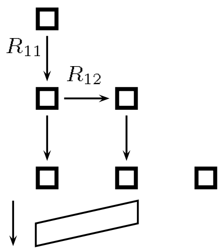
The update of $\sqrt{\Lambda}$ is performed using (5.66):

$$\begin{bmatrix} 0 \\ \sqrt{\bar{\Lambda}} \end{bmatrix} = \sqrt{\lambda} \begin{bmatrix} C & -S \\ S & C \end{bmatrix} \begin{bmatrix} \varepsilon / s^{\frac{1}{2}} \\ \tilde{\Lambda} \end{bmatrix}.$$

It is mapped, together with computation of ν (5.39) to the lower left corner of the systolic array (where also the computation of ε (5.34, 5.70) is being completed).

8.9.2 Transposition of R

For transposition of the first row of the covariance matrix factor, the following mechanism is used: the upper element (R_{11}) is sent down through the vertical links between the processing elements. The remaining elements are first sent rightwards to the diagonal, and from the diagonal they are also sent down, as sketched bellow:



The transposed vector emerges from the bottom row of the array, skewed in time, the leftmost element first.

8.9.3 Multiplication of R by $\sqrt{\sigma}$

The multiplication of R by $\sqrt{\sigma}$ is performed in the bottom row of the estimator: having been computed in the first PE of the bottom row, $\sqrt{\sigma}$ is sent rightwards to the remaining processors in the row and it passes through the i-th processing element in the same time as R_{1i}.

8.9.4 Loading R to the controller

The elements of R_1, emerging from the bottom edge of the estimator, are loaded to the controller together with the parameter estimates (Fig. 8.23).

8.9.5 Reduction of $\sqrt{\sigma}R$

The elements of the covariance matrix factor are reduced in the controller in the same way as row c of matrix $\bar{\tilde{\Phi}}$ (Fig. 7.3). This reduction adds, in the last step of the dynamic programming, one more rotation to the second operation mode of the processing elements in the first row of the array.

8.10 Summary

In this chapter, a systolic implementation of the square-root LQ controller was considered, up to the cell-level description. It should be emphasized that (though the penalties and setpoints are loaded in different way) the function of our systolic controller is exactly the same as of the sequential version described in [23].

An important contribution of this chapter is that it does not only give the description of the controller, but also solves the problems of connection of controller to the regularized estimator and of connection of the whole selftuner to the controlled system.

Chapter 9

Conclusions

This is the final chapter of the thesis. It will summarize the contribution of the thesis both to theory and to practice. Also, the areas of further research will be outlined as guidelines for problems which still remain to be solved.

9.1 What has been done ...

The thesis has been focused on a systolic implementation of the adaptive LQ controller with a robust parameter estimator. According to our knowledge, the solution, presented in the thesis, represents the first attempt to design a complete parallel systolic implementation of the LQ selftuner using novel, mathematically well-defined methods to improve numerical stability of the identification algorithm in adaptive control loop. The solution, presented in this work, results in an order reduction of the sampling period, compared with the sequential implementation of the selftuner.

The contribution of this thesis may be summarized into the following items:

Systolic block-regularized estimator

The block-accumulated regularized exponential forgetting [21, 18, 19] was implemented in the RLS array with inverse updates [39, 43] to improve robustness of this array to non-informative data (Sec. 6.3). The regularized forgetting, which is based on restricted exponential forgetting [27, 28], improves numerical stability of the adaptive estimator with respect to non-informative data.

- It was shown that the regularization can be used not only for time update of information filter [19], but also for update of the covariance matrix.

- The *block accumulated* regularization preserves the pipelining of the systolic algorithm and uses only the connections between the neighbouring processors. The resulting systolic estimator works on a triangular array of $\mathcal{O}((n+1)^2)$ processors and has a sampling period of $\mathcal{O}(1)$ and a period of parameter estimate generation of $\mathcal{O}(n)$, both compared to $\mathcal{O}(n^3)$ of the sequential implementation (restricted exponential forgetting [27]).
- The RLS array with inverse updates was used in the selftuner for two reasons: firstly, because it has the parameter estimates directly available at the output and can be directly connected with the array for control design (Sec. 8.5, 8.6); secondly, because it may be used for a systolic implementation of the cautious control strategy (Sec. 8.9).

The influence of regularization on the behaviour of the estimator in the conditions of weak excitation was demonstrated in simulation examples (Sec. 6.5).

Systolic LQ controller

An LQ controller was designed with *iteration period independent of the system order* (Chapter 8). The controller is implemented on a triangular array of $\mathcal{O}(n^2)$ processors. The increase of the system order is reflected in the increased number of processors used to construct the systolic array, however the data sampling period of the controller remains unchanged.

The presented solution is novel in the following respects:

- Unlike the previous designs [4, 5], the presented systolic controller treats the problem of program control and regulation with non-zero setpoints, which is important for practical applications. Both the setpoints and the penalties may be time-variable.
- The controller is designed to load the penalties and setpoints at the moment when they are used. This mechanism is expected to be advantageous if design of control with restrictions on system input is solved. The price paid for using this loading is that at every iteration, one more QR reduction must be performed.

- The problems associated with connection of the regularized selftuner to the controlled system are treated in detail (Sec. 8.5).

The behaviour of the resulting systolic LQ controller and the influence of regularization on its behaviour were demonstrated in simulation experiments (Sec. 8.8). The results of the experiments show that although it does not provide guarantee, the regularization has a positive influence on the stability of control loop.

Other contributions

- An introduction to the basic function principles of systolic arrays was given (Sec. 6.1).
- The principle of the block-regularized forgetting [19] was explained (Sec. 5.4).
- The principle of the square-root LQ control design was explained (Chapter 7).
- Main features of a systolic implementation of the cautious control strategy were outlined (Sec. 8.9).

9.2 Application results

Possible applications

Before the contribution of the thesis will be summarized from the practical viewpoint, let us consider possible applications of the systolic selftuner.

The use of the parallel controller is generally desirable for those applications, that, on one hand, require complex model for adequate description and, on the other hand, have high speed requirements.

Examples of such processes include cold rolling mills [7], flying drum shears, adaptive control of fast drives, etc. Another application area is control of chemical processes, again because of its high computational complexity.

The block-regularized systolic estimator may find use in many tasks of modern signal processing—let us name the adaptive beamfinding and beamforming as good examples [6, 49, 22].

Cell-level description of algorithms

Because of many problems that would necessarily need to be solved, and that would go far beyond the available extent of this work, the thesis does not deal with any particular application of the controller. Instead, the contribution to the practice is in the detailed cell-level description of the systolic algorithms, derived in this work:

- The cell-level description of the block regularized RLS estimator is given in Section 6.4.
- The cell-level description of the systolic controller is presented in Section 8.3

These descriptions may be directly used for implementation of algorithms on a parallel (pipelined) computing system.

Matlab—transputer bridge

Efficient design and development of control systems requires availability of a programming environment offering high-level support for matrix manipulations and simulations as well as an easy-to-use user interface including graphing capabilities.

Multiple tools satisfying these requirements were developed for PC applications, most of them taking inspiration in Matlab computing environment by MathWorks, Inc. (Matlab is a registered trademark).

No similar support, however, is available for the parallel computing systems. The only available programming language for development of parallel identification and control algorithms is C (or Occam for transputers, of a level between the assembler language and C). In result, the design of parallel control algorithms is often performed only on a theoretical level, because any simulation experiments represent quite cumbersome task.

During the preparation of this thesis, an environment for on-line connection of a parallel system to the Matlab program has been developed by author's colleagues [20] to solve this drawback. It uses Matlab not only as a computation engine, but also as an easy-to-use front-end interface for parallel function libraries.

In the present time, the systolic algorithms described in the thesis are being ported to this environment.

9.3 What remains to be done ...

There are still numerous problems to be solved, both at a theoretical level and in the field of practical applications. Let us name some of them:

- The systolic algorithms presented in the thesis could be extended to the multivariable case. In practical applications, multivariable control calls for fast implementation even more than the Single Input Single Output (SISO) case. So far, however, only very little has been done in this field.

- Fast implementation of control design with input constraints should be considered. The control of real systems always has to cope with limitations on the system input(s), given either by physical or by economic constraints. In the multivariable case, respecting these limitations leads to quadratic programming. A speed-up of computations by a suitable parallelization would be very desirable.

- The actual influence of the cautious control strategy on the control quality and the behaviour of the systolic implementation of this strategy have not been tested in this thesis, because of space limitations. These tests could be carried out in a future research.

- A mathematical analysis of the influence of regularization on the stability of the control loop could be performed. This was beyond the problems solved in this thesis.

- The simulation experiments with the systolic controller were using some "reasonable" initial control law, which was chosen so that the stability of the control loop is preserved. For practical applications, it would be better to use results of preliminary analysis of the controlled system [24]. In the same time, fast algorithms could be used in this analysis, since a need arises there to evaluate quickly numerous possible settings of controller.

Acknowledgement

The work on the thesis was supported by the research grant No. 102/93/0897 "Systolic Arrays of Processors for Identification of the Systems with Variable Structure" of the Grant Agency of the Czech Republic.

Bibliography

[1] D. D. Baer and J. Paradeans. A formal definition for systolic systems. In A. Albrecht, H. Jung, and K. Mehlhorn, editors, *Parallel Algorithms and Architectures*, Lecture Notes in Computer Science. Springer-Verlag, 1987.

[2] J. Böhm and M. Kárný. Self-tuning regulators with restricted inputs. *Kybernetika (Prague)*, 18(6):529–543, 1982.

[3] L. Chisci and G. Zappa. A systolic architecture for iterative LQ optimization. *Automatica*, 27(5):799–810, 1991.

[4] L. Chisci and G. Zappa. Systolic architecture for predictive control. In Fleming and Jones [8], pages 37–42. Proceedings of an IFAC Workshop held September 11–13, 1991, in Bangor, Wales, UK.

[5] L. Chisci and G. Zappa. Mapping LQ control design on fixed-size array processor. In Kárný and Warwick [25], pages 181–193. Proceedings of an IFAC Workshop, held September 1–2, 1992, in Prague, Czech Republic.

[6] L. D. J. Eggermont et al., editors. *VLSI Signal Processing VI*, New York, 1993. IEEE Signal Processing Society, IEEE Press. Proceedings of the IEEE Signal Processing Society Workshop, held October 20–22, 1993, in Veldhoven, The Netherlands.

[7] P. Ettler. Advanced algorithms contra lack of computing power. In Kárný and Warwick [25], pages 349–352. Proceedings of an IFAC Workshop, held September 1–2, 1992, in Prague, Czech Republic.

[8] P. J. Fleming and D. I. Jones, editors. *Algorithms and Architectures for Real-Time Control*, number 4 in IFAC Workshop Series, Oxford, New

York, Seoul, Tokyo, 1992. International Federation of Automatic Control, Pergamon Press. Proceedings of an IFAC Workshop held September 11–13, 1991, in Bangor, Wales, UK.

[9] F. M. F. Gaston and G. W. Irwin. Linear quadratic optimal controller realization on systolic arrays. In *IEE Proceedings UK IT Conference*, 1990.

[10] F. M. F. Gaston and G. W. Irwin. Systolic Kalman filtering: an overview. *Proc. IEE-D*, 137(4), 1990.

[11] W. M. Gentleman and H. T. Kung. Matrix triangularisation by systolic arrays. In *Real Time Signal Processing IV*, volume 298, pages 19–26, 1981. Proc. SPIE.

[12] W. Givens. Computing of plane unitary rotations transforming a general matrix to triangular form. *J. Soc. Ind. Appl. Math*, 6:26–50, 1958.

[13] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Series in the Mathematical Sciences. The Johns Hopkins University Press, 2 edition, 1989.

[14] A. Halousková, M. Kárný, and I. Nagy. Adaptive cross-direction control of paper basis weight. *Automatica*, 29(2):425–429, 1993.

[15] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Hemel Hempstead, 1985.

[16] J. M. Jover and T. Kailath. A parallel architecture for Kalman filter measurement update and parameter estimation. *Automatica—J. IFAC*, 22:43–57, 1986.

[17] J. Kadlec. *Probabilistic Identification of Regression Model in a Fixed-Point Arithmetic*. PhD thesis, Institute of Information Theory and Automation of the Academy of Sciences of the Czech Republic, Prague, 1986. in Czech.

[18] J. Kadlec. A joint criterion for exponential directional and mixed parameter tracking. In I. Landau, L. Dugard, and M. M'Saad, editors, *Preprints of the Fourth IFAC International Symposium on Adaptive Systems in Control and Signal Processing*, pages 687–692, Grenoble, France, 1992. IFAC.

[19] J. Kadlec. The cell-level description of systolic block regularised QR filter. In Eggermont et al. [6], pages 298–306. Proceedings of the IEEE Signal Processing Society Workshop, held October 20–22, 1993, in Veldhoven, The Netherlands.

[20] J. Kadlec. Direct software bridge matlab-transputer boards. In *Proceedings of the VII European Signal Processing Conference EUSIPCO-94*, Edinburgh, UK, 1994. Also available as a Res. Report 1809, Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Prague.

[21] J. Kadlec, F. M. F. Gaston, and G. W. Irwin. Systolic implementation of the regularised parameter estimator. In Yao et al. [49], pages 520–529. Proceedings of the IEEE Signal Processing Society Workshop, held October 28–30, 1992, in Napa, CA.

[22] N. Kalouptsidis and S. Theodoridis. *Adaptive System Identification and Signal Processing Algorithms.* Prentice Hall International Series in Acouctics, Speech and Signal Processing. Prentice Hall International Ltd., 1993.

[23] M. Kárný et al. Design of linear quadratic adaptive control: Theory and algorithms for practice. *Kybernetika (Prague)*, 21, 1985. Supplement.

[24] M. Kárný and A. Halousková. Pre-tuning of self-tuners. In D. Clarke, editor, *Advances in Model-Based Predictive Control.* Oxford University Press, Oxford, UK, 1994.

[25] M. Kárný and K. Warwick, editors. *Mutual Impact of Computing Power and Control Theory*, New York and London, 1993. Plenum Press. Proceedings of an IFAC Workshop, held September 1–2, 1992, in Prague, Czech Republic.

[26] R. Kulhavý. *Probabilistic Identification of a time variable systems with an unknown model of the parameter development.* PhD thesis, Institute of Information Theory and Automation of the Academy of Sciences of the Czech Republic, Prague, 1985. in Czech.

[27] R. Kulhavý. Restricted exponential forgetting in real-time identification. *Automatica*, 23:589–600, 1987.

[28] R. Kulhavý and M. B. Zarrop. On a general concept of forgetting. *Internat. J. Control*, 58(4):905–924, 1993.

[29] P. R. Kumar. A survey on some results in stochastic adaptive control. *SIAM J. Control and Applications*, 23:329–380, 1985.

[30] S. Y. Kung. *VLSI Array Processors.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[31] J. C. Le Roux and G. A. Guegen. Fixed point computation of partial correlation coefficients. *IEEE Trans. Acoust. Speech Signal Process.*, 25:257–259, 1981.

[32] D. Lee, M. Morf, and B. Friedlander. Recursive least-squares ladder estimation algorithm. *IEEE Trans. Acoust. Speech Signal Process.*, 29:627–641, June 1981.

[33] F. Ling, D. Manolakis, and J. G. Proakis. A recursive modified gram–schmidt algorithm for least squares estimation. *IEEE Trans. Acoust. Speech Signal Process.*, 34:829–835, 1986.

[34] L. Ljung and T. Söderström. *Theory and Practice of Recursive Identification.* MIT Press, Cambridge, MA, 1983.

[35] J. G. McWhirter. Systolic array for recursive least squares by inverse iterations. In Eggermont et al. [6], pages 435–443. Proceedings of the IEEE Signal Processing Society Workshop, held October 20–22, 1993, in Veldhoven, The Netherlands.

[36] J. G. McWhirter. A systolic array for recursive least squares estimation by inverse updates. In *International Conference on Control '94, University of Warwick*, London, March 21–24 1994. IEE.

[37] J. G. McWhirter. Algorithmic engineering in adaptive signal processing. *IEE Proc., Pt. F*, 139(3), June 1992.

[38] J. G. McWhirter and I. K. Proudler. *The QR Family*, chapter 7, pages 260–321. In *Prentice Hall International Series in Acouctics, Speech and Signal Processing* [22], 1993.

[39] J. G. McWhirter and I. K. Proudler. A systolic array for recursive least squares estimation by inverse updates. In *The International Conference on Control '94*, London, 1994. IEE. Proceedings.

[40] G. M. Megson. *An Introduction to Systolic Array Design.* Oxford University Press, U.K., 1992.

[41] P. C. Miller, F. M. F. Gaston, and G. W. Irwin. A novel broadcast architecture for the square root Kalman filter. In Fleming and Jones [8], pages 57–61. Proceedings of an IFAC Workshop held September 11–13, 1991, in Bangor, Wales, UK.

[42] R. Milner. Operational and algebraic semantics of concurrent processes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 19, pages 1201–1242. Elsevier Science Publishers B.V. (North-Holland), 1990.

[43] M. Moonen and J. G. McWhirter. A systolic array for recursive least squares by inverse updating. *Electronics Letters*, 29(13):1217–1218, 1993.

[44] P. Nedoma. Towards higher software effectiveness in the field of adaptive control. In K. Warwick, M. Kárný, and A. Halousková, editors, *Advanced Methods in Adaptive Control for Industrial Applications*, volume 158 of *Lecture Notes in Control and Information Sciences*, pages 149–153. Springer-Verlag, 1991.

[45] P. Nedoma and D. Hrudková. X*lab* – reference manual. Technical report, Institute of Information Theory and Automation of the Academy of Sciences of the Czech Republic, 1992.

[46] P. Nedoma and D. Hrudková. X*lab* – tutorial. Technical report, Institute of Information Theory and Automation of the Academy of Sciences of the Czech Republic, 1992.

[47] V. Peterka. *Digital Control of Processes with Stochastic Disturbances and Uncertain Characteristics.* Doctor of science thesis, Institute of Information Theory and Automation of the Academy of Sciences of the Czech Republic, Prague, 1975. in Czech.

[48] V. Peterka. Bayesian approach to system identification. In P. Eykhoff, editor, *Trends and Progress in System Identification*, IFAC Series for Graduates, Research Workers and Practising Engineers, chapter 8. Pergamon Press, Oxford, 1981.

[49] K. Yao et al., editors. *VLSI Signal Processing V*, New York, 1992. IEEE Signal Processing Society, IEEE Press. Proceedings of the IEEE Signal Processing Society Workshop, held October 28–30, 1992, in Napa, CA.