KYBERNETIKA — VOLUME 32 (1996), NUMBER 3, PAGES 209 - 234

INVERSE UPDATED SYSTOLIC RLS ALGORITHM WITH REGULARIZED EXPONENTIAL FORGETTING[1]

JAN SCHIER

A systolic algorithm for the Recursive Least Squares identification with covariance update, using the block-accumulated regularization mechanism to increase numerical stability of the algorithm with respect to weakly informative data, is presented. The advantages over standard sequential implementation are that the sampling period of estimator is significantly reduced even with the robustifying modification of algorithm and that it is made independent of order of the identified system.

1. INTRODUCTION

Adaptive identification finds its use in many applications of both signal processing (channel equalisation, adaptive antenna array beamforming etc.) and adaptive control.

This paper focuses on the problem of increasing stability of the Recursive Least Squares (RLS) identification algorithm with update of the inverse factor of data-covariance-matrix (also referred to as the Inverse Updated RLS algorithm) for weakly informative data samples. It combines the inverse updated systolic RLS algorithm with covariance update, designed in [6, 9] (parallel version of the LDFIL algorithm [11]) with the block-accumulated regularized exponential forgetting, proposed in [3]. It should be mentioned that the inverse updated algorithm directly provides weighting coefficients of the regression model on its output. This property makes it attractive for use in adaptive control.

The reasoning for use of the parallel systolic version of the algorithm is that some applications have computational requirements (high sampling rate and/or large order of model required for proper description) which cannot be satisfied by usual sequential implementation of the estimator. These requirements yet increase if the regularization is used, because it increases the computational complexity of algorithm in an order of magnitude. As will be explained in the text, systolic version of an algorithm reduces the sampling period partly by introducing parallel processing, partly by using data pipelining.

[1] The research was partly supported by the Grant Agency of the Czech Republic under Grants No. 102/95/1614 and 102/95/0926.

NOTATIONAL CONVENTIONS

$\boldsymbol{x}, \boldsymbol{y}$	...	scalars
$\boldsymbol{a}, \boldsymbol{b}$	...	column vectors
$\boldsymbol{A}, \boldsymbol{B}$	...	matrices
$\boldsymbol{I}$	...	unit matrix
$\boldsymbol{a}', \boldsymbol{A}'$	...	a, A transpose
$\partial \boldsymbol{x} = n,\ \partial \boldsymbol{A} = n \times n$	...	dimension of a vector or matrix
$\boldsymbol{k}$	...	discrete time, time index of a sampling period; the observation of a process starts at $k = 1$
p.d.f.	...	probability density function
c.p.d.f.	...	conditional p.d.f.
PE	...	processing element
$d(1, k)$	...	set of input-output data since the beginning of observation till the time k
$\tilde{\boldsymbol{x}}$	...	x *before* the data update
$\bar{\tilde{\boldsymbol{x}}}$	...	x *after* the data update and *before* the time update
$\bar{\boldsymbol{x}}$	...	x *after* the time update
$\breve{\boldsymbol{x}}$	...	x *after* the data update and exponential forgetting, but *before* the addition of regularizing value
$\mathcal{N}(0, \sigma)$	...	normal (Gaussian) distribution with a zero mean value and a constant dispersion σ
$p(a\|b)$	...	c.p.d.f. of argument a, conditioned on b
$p(y(k)\|k-1; u(k))$	...	abbreviation for $p(y(k)\|y(1, k-1), u(1, k-1), u(k))$ (conditioning including the data history)
$\mathsf{E}[a]$	...	expected (mean) value of a random variable a
$\hat{y}(k\|k-1; u(k))$	...	abbreviation for the expected value $\mathsf{E}[y(k)\|y(1, k-1), u(1, k-1), u(k)]$

2. IDENTIFICATION OF THE SYSTEM MODEL

Linear regression model

Let us suppose the system to be described by a linear regression model

$$y(k) = \boldsymbol{\Theta}'(k)\boldsymbol{\varphi}(k) + e(k) \tag{1}$$

where

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

$\boldsymbol{\varphi}(k)$ is the data vector,

$\boldsymbol{\Theta}(k)$ is the vector of regression parameters, $\partial\boldsymbol{\Theta} = n$, and

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

Under these assumptions, the p.d.f. of the system output is described by a *normal distribution.*

Gauss–Wishart distribution of the system parameters

Let the *apriori c.p.d.f. of the system parameters* have the self-reproducing form of the *Gauss–inverse–Wishart* distribution

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

$$J_\Theta(k|k-1) = \begin{bmatrix} -\boldsymbol{\Theta}(k) \\ 1 \end{bmatrix}' \boldsymbol{V}(k|k-1) \begin{bmatrix} -\boldsymbol{\Theta}(k) \\ 1 \end{bmatrix} \tag{3}$$

where

$p(\boldsymbol{\Theta}(k), \sigma(k)|k-1)$ is the p.d.f. of parameters $\boldsymbol{\Theta}(k)$ 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 $\boldsymbol{\Theta}(k)$, conditioned on the data $d(0, k-1)$ [3]. For the *GiW*-distribution, it determines the shape of the distribution.

$\boldsymbol{V}(k|k-1)$ is a positive-definite symmetric *extended information matrix.* This matrix accumulates the information contained in the data measurements. For later use, it is useful to introduce the following sub-matrices

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{V}_\varphi & \boldsymbol{v}_{\varphi y} \\ \boldsymbol{v}'_{\varphi y} & v_y \end{bmatrix}, \quad \partial\boldsymbol{V} = (n+1) \times (n+1), \tag{4}$$

where

$\boldsymbol{V}_\varphi$ is an *information matrix*, $\partial\boldsymbol{V}_\varphi = n \times n$,
$\boldsymbol{v}_{\varphi y}$ is a vector, $\partial\boldsymbol{v}_{\varphi y} = n$, and
v_y is scalar.

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

$$\nu(k|k-1) > 0,$$

$\rho(\boldsymbol{V}, \nu)$ is a normalization constant [10].

Sufficient statistics of the GiW-distribution

The parameters $\boldsymbol{V}$ and ν fully determine *sufficient statistics* of the *GiW*-distribution [10].

Parameter estimate is given by the relation

$$\hat{\boldsymbol{\Theta}}(k+1|k) = \boldsymbol{V}_{\varphi}^{-1}(k+1|k)\,\boldsymbol{v}_{\varphi y}(k+1|k). \tag{5}$$

The sufficient statistics may be expressed also in an inverse form, which is used in the *Least Squares* (LS) methods [5, 10]:

$$\boldsymbol{P}(k|k-1) = \boldsymbol{V}_{\varphi}^{-1}(k|k-1) \tag{6}$$

$$\hat{\Lambda}(k|k-1) = v_y(k|k-1) - \boldsymbol{v}'_{\varphi y}(k|k-1)\,\boldsymbol{P}(k|k-1)\,\boldsymbol{v}_{\varphi y}(k|k-1) \tag{7}$$

$$\hat{\sigma}(k|k-1) = \frac{\hat{\Lambda}(k|k-1)}{\nu(k|k-1)-2}. \tag{8}$$

Recursive adaptive identification

Recursive identification consists of two steps: of the *data update*, when the parameters of the sufficient statistics are updated using new information gathered from the system, and of the *time update*, when the time evolution of the system is modeled using some forgetting method. Both steps will be described now.

Remark on notation. The 'hat' symbol ($\hat{\bullet}$) denotes the estimates of parameters. Because only the estimates are used in the following text, the 'hat' symbol will not be used, to simplify the data and time update relations.

Where appropriate to simplify the text, the following symbols will be used to refer to the particular phases of update: the *tilde* symbol ($\tilde{\bullet}$) will be used to refer to the value of a variable *before* the data update, stacked *bar* and *tilde* symbol ($\bar{\tilde{\bullet}}$) to refer to the value *after* the data update, and the *bar* symbol ($\bar{\bullet}$) for the value *after* the forgetting step (will be formulated in the next section).

Data Update

Using the information matrix (4), data update is expressed by

$$\boldsymbol{V}(k|k) = \boldsymbol{V}(k|k-1) + \begin{bmatrix} \boldsymbol{\varphi}(k) \\ y(k) \end{bmatrix} \begin{bmatrix} \boldsymbol{\varphi}(k) \\ y(k) \end{bmatrix}'. \tag{9}$$

Using the covariance matrix $\boldsymbol{P}$ (6), it is expressed by the formulae of the *recursive least squares (RLS) identification*

$$\zeta = \boldsymbol{\varphi}'\tilde{\boldsymbol{P}}\boldsymbol{\varphi}, \tag{10}$$

$$\boldsymbol{\kappa} = (1+\zeta)^{-1}\tilde{\boldsymbol{P}}\boldsymbol{\varphi}, \tag{11}$$

$$\varepsilon = y - \tilde{\boldsymbol{\Theta}}\boldsymbol{\varphi}, \tag{12}$$

$$\bar{\tilde{\boldsymbol{\Theta}}} = \tilde{\boldsymbol{\Theta}} + \boldsymbol{\kappa}\varepsilon, \tag{13}$$

$$\bar{\tilde{\boldsymbol{P}}} = \tilde{\boldsymbol{P}} - (1+\zeta)\,\boldsymbol{\kappa}\boldsymbol{\kappa}', \tag{14}$$

where κ denotes the *Kalman gain* vector, ε the *prediction error* and ζ is an auxiliary variable.

Since the estimates are independent of the remainder after the RLS estimation Λ, we do not actually have to evaluate it.

Time Update — Regularized Exponential Forgetting

To approximate the time update of the cost function J_Θ, *regularized exponential forgetting* can be used, described by the formula

$$\bar{J}_\Theta = \lambda \bar{\bar{J}}_\Theta + (1-\lambda)\, J^*_\Theta, \tag{15}$$

where $0 < \lambda \leq 1$ (which may be time variable), is the *exponential forgetting factor*, which is used to weight the cost function J_Θ with an alternative value J^*_Θ.

The "alternative cost function" J^*_Θ is a value to which the cost function converges in the case of non-informative input data. Introducing "alternative parameters" $\boldsymbol{V}^* = (\boldsymbol{P}^*)^{-1}$ and $\boldsymbol{\Theta}^*$, specified by user, the additional term J^*_Θ may be expressed in the same way as the cost function J_Θ (3)

$$J^*_\Theta \hat{=} [\boldsymbol{\Theta} - \boldsymbol{\Theta}^*]'(\boldsymbol{P}^*)^{-1}[\boldsymbol{\Theta} - \boldsymbol{\Theta}^*] + \Lambda^* = \begin{bmatrix} -\boldsymbol{\Theta} \\ 1 \end{bmatrix} \boldsymbol{V}^* \begin{bmatrix} -\boldsymbol{\Theta} \\ 1 \end{bmatrix}, \tag{16}$$

Since the estimate $\boldsymbol{\Theta}$ is independent of evolution of Λ, also the alternative Λ^* is a 'don't care' term.

Using the alternative parameters, the time update formulae for $\boldsymbol{V} = \boldsymbol{P}^{-1}$ and $\boldsymbol{\Theta}$ are given by

$$\bar{\boldsymbol{P}} = [\lambda \bar{\bar{\boldsymbol{P}}}^{-1} + (1-\lambda)(\boldsymbol{P}^*)^{-1}]^{-1}, \tag{17}$$

$$\boldsymbol{A} \hat{=} \left[\boldsymbol{I} + (\frac{1}{\lambda} - 1)\, \bar{\bar{\boldsymbol{P}}}(\boldsymbol{P}^*)^{-1}\right]^{-1}, \tag{18}$$

$$\bar{\boldsymbol{\Theta}} = \boldsymbol{A}\bar{\bar{\boldsymbol{\Theta}}} + (\boldsymbol{I} - \boldsymbol{A})\, \boldsymbol{\Theta}^*, \tag{19}$$

where $\boldsymbol{A}$ is an auxiliary matrix introduced to simplify the update formula for $\boldsymbol{\Theta}$, or by

$$\bar{\boldsymbol{V}}_\varphi = \lambda \bar{\bar{\boldsymbol{V}}}_\varphi + (1-\lambda)\, \boldsymbol{V}^*_\varphi = \breve{\boldsymbol{V}}_\varphi + (1-\lambda)\boldsymbol{V}^*_\varphi, \tag{20}$$

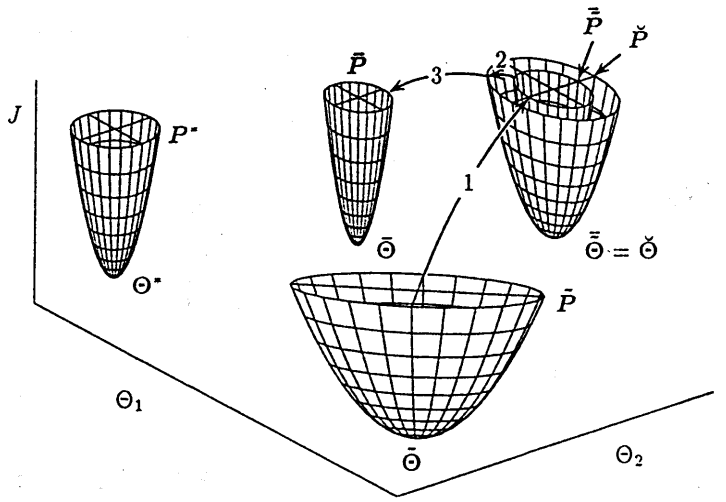
$$\begin{aligned} \bar{\boldsymbol{v}}_{\varphi y} &= \lambda \bar{\bar{\boldsymbol{v}}}_{\varphi y} + (1-\lambda)\, \boldsymbol{v}^*_{\varphi y} = \lambda \bar{\bar{\boldsymbol{v}}}_{\varphi y} + (1-\lambda)\boldsymbol{V}^*_\varphi \boldsymbol{\Theta}^* \\ &= \breve{\boldsymbol{v}}_{\varphi y} + (1-\lambda)\boldsymbol{V}^*_\varphi \boldsymbol{\Theta}^*, \end{aligned} \tag{21}$$

$$\bar{\boldsymbol{\Theta}} = \bar{\boldsymbol{V}}_\varphi^{-1} \bar{\boldsymbol{v}}_{\varphi y}. \tag{22}$$

Remark. In the limit case of $\boldsymbol{V}^*_\varphi \to 0$ (non-informative probability distribution), regularization forgetting degrades to 'standard' exponential forgetting.

To illustrate the process, evolution of the cost function J_Θ (3) during one step of the regularized identification is plotted in Figure 1. A two dimensional parameter

vector $\boldsymbol{\Theta} = [\Theta_1 \ \Theta_2]$ is considered, the vertical axis of the graph represents the cost function J_Θ. The placement of $\boldsymbol{\Theta}^*$ does not represent the situation when it follows the regularized estimate, it is rather chosen so that the evolution of $\boldsymbol{\Theta}$ during data update and regularization is better shown. Also the value of $\boldsymbol{P}^*$ is chosen so that the changes of $\boldsymbol{P}$ during update are shown.



1: Data update 2: Exponential forgetting 3: Regularization

Fig. 1. One period of regularized identification.

The elliptical paraboloids represent the cost function in the successive step of the identification period. Their cross-section reflects the shape of the covariance matrix P (6, 14, 17), their placement is determined by the value of parameter estimate $\boldsymbol{\Theta}$ (5, 13, 19) in the respective steps of update.

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

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

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

Selection of regularizing parameters

The alternative information matrix $\boldsymbol{V}_\varphi^* = (\boldsymbol{P}^*)^{-1}$ is a symmetric positive definite matrix. Its role is to prevent the covariance matrix $\boldsymbol{P}$ from exponential growth (i.e.

$\boldsymbol{V}$ from losing rank) if the estimator is not sufficiently excited. It should be small enough to allow the algorithm to adapt to parameter changes and to come into effect only when the data does not have sufficient information.

A suitable general-purpose choice for $\boldsymbol{V}_\varphi^*$ is $\mu\boldsymbol{I}$, $0 < \mu < 1$, $\boldsymbol{I}$ is a unit matrix [12, 13]. If matrix $\boldsymbol{P}$ is available from the preliminary analysis of the system, it may be used with advantage.

$\boldsymbol{\Theta}^*$ is a value, to which the parameter estimates converge in the case of non-informative data. It is possible either to require that $\bar{\boldsymbol{\Theta}}$ preserves the value identified from informative data — in that case $\boldsymbol{\Theta}^*$ should follow the development of estimate, or to require that $\bar{\boldsymbol{\Theta}}$ takes the value obtained in preliminary system analysis. A suitable choice in the first case is to assign $\boldsymbol{\Theta}^*$ the last regularized estimate

$$\boldsymbol{\Theta}^*(k) := \bar{\boldsymbol{\Theta}}(k-1). \tag{23}$$

Block-Accumulated Regularization

Regularized exponential forgetting in the standard form, as described in (17–19) or (20–22), is not suitable for systolic implementation (the principles of which will be described later). Instead, the *block-accumulated regularization* is used, proposed in [3, 4], which preserves data pipelining in the systolic array.

The idea of this method is the following: Let us keep the alternative parameters $\boldsymbol{V}_\varphi^*$ and $\boldsymbol{\Theta}^*$ constant over $N \geq n$ periods of identification (i.e. over processing of N data samples), where $n = \partial\boldsymbol{\Theta}$

$$\begin{aligned} \boldsymbol{V}_\varphi^* &= \boldsymbol{V}_\varphi^*(k) = \boldsymbol{V}_\varphi^*(k+1) = \ldots = \boldsymbol{V}_\varphi^*(k+N-1), \\ \boldsymbol{\Theta}^* &= \boldsymbol{\Theta}^*(k) = \boldsymbol{\Theta}^*(k+1) = \ldots = \boldsymbol{\Theta}^*(k+N-1). \end{aligned} \tag{24}$$

The regularized forgetting (20) may be seen also as consequent steps of exponential forgetting

$$\check{\boldsymbol{V}} = \lambda\bar{\bar{\boldsymbol{V}}}, \tag{25}$$

(where $\check{V}$ denotes the value after the data update and exponential forgetting) and of addition of regularizing parameters

$$\bar{\boldsymbol{V}}_\varphi = \check{\boldsymbol{V}}_\varphi + (1-\lambda)\,\boldsymbol{V}_\varphi^*, \tag{26}$$

$$\bar{\boldsymbol{v}}_{\varphi y} = \check{\boldsymbol{v}}_{\varphi y} + (1-\lambda)\,\boldsymbol{V}_\varphi^*\boldsymbol{\Theta}^*. \tag{27}$$

Hence, it is possible to perform only the exponential forgetting over the N steps, then to interrupt temporarily data processing and to perform all additions of the regularizing parameters at once in this interruption, weighted by an *accumulated forgetting factor*. As will be shown later, this addition takes n periods. In this way, the data pipelining is preserved.

This mechanism is described by the following algorithm (because time relations over several periods of identification are shown, it is necessary to use time indexing again. To simplify the indexing, time $k-1$ is 'subtracted' in all indices, e.g., instead of k, 1 is written)

1. Data update with standard exponential forgetting over N periods

$$
\begin{aligned}
&\check{\boldsymbol{V}} := \boldsymbol{V}(1|0), \ \lambda_N := 1 && \text{a)}\\
&\text{for } i := 1 \text{ to } N\\
&\quad \check{\boldsymbol{V}} := \lambda(i)\left\{\check{\boldsymbol{V}} + \begin{bmatrix} \varphi(i) \\ y(i) \end{bmatrix}\begin{bmatrix} \varphi(i) \\ y(i) \end{bmatrix}'\right\} && \text{b)}\\
&\quad \lambda_N := \lambda_N \lambda(i) && \text{c)}\\
&\text{end}\\
&\lambda_N := 1 - \lambda_N && \text{d)}
\end{aligned}
\tag{28}
$$

2. Accumulated regularization in the Nth period

$$
\begin{aligned}
&\boldsymbol{v}^*_{\varphi y} := \boldsymbol{V}^*_{\varphi}\boldsymbol{\Theta}^* && \text{a)}\\
&\boldsymbol{V}_{\varphi}(N+1|N) := \check{\boldsymbol{V}}_{\varphi} + \lambda_N \boldsymbol{V}^*_{\varphi} && \text{b)}\\
&\boldsymbol{v}_{\varphi y}(N+1|N) := \check{\boldsymbol{v}}_{\varphi y} + \lambda_N \boldsymbol{v}^*_{\varphi y} && \text{c)}\\
&\boldsymbol{\Theta}(N+1|N) := \boldsymbol{V}^{-1}_{\varphi}(N+1|N)\, \boldsymbol{v}_{\varphi y}(N+1|N) && \text{d)}
\end{aligned}
\tag{29}
$$

$0 < \lambda_N \leq 1$ is an *accumulated forgetting factor.*

Using the block-accumulated regularization, the last regularized estimate is N periods old. Hence, (23) is replaced by

$$\boldsymbol{\Theta}^*(k) := \bar{\boldsymbol{\Theta}}(k - N). \tag{30}$$

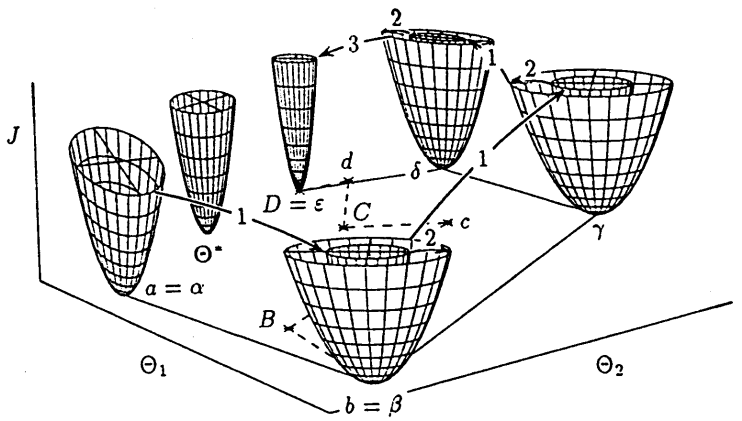
To give better idea of the mechanism, a comparison of evolution of parameter estimates with both the standard and the block-accumulated regularization is given in the graph in Figure 2. For the block-regularized identification, it shows also the evolution of the cost function J_Θ (16).

The graph is drawn for regularization applied after 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 to the position of parameter estimates after the data update (9), the upper-case letters are used to refer to that after the regularization (20–22).

The Greek letters denote the tracking trajectory resulting from the block-regularized identification; the arrows show the development of the covariance matrix $\boldsymbol{P}$ through one block of the block-regularized identification — number 1 denotes the data update (9), number 2 the exponential forgetting (1 + 2 = (28e)), number 3 denotes the addition of the alternative parameters (29b–c).

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



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

Fig. 2. Comparison of standard and block-accumulated regularization.

3. SQUARE–ROOT IMPLEMENTATION OF THE RLS ALGORITHM

In practice, the square-root version of the estimator is used, because it guarantees symmetry and positive definiteness of the covariance/information matrix and can be implemented on a systolic array.

Square-root Decomposition of the RLS Algorithm

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

$$\boldsymbol{P} = \boldsymbol{R}\boldsymbol{R}', \tag{31}$$

where $\boldsymbol{R}$ is an upper triangular matrix.

Applying the decomposition in the formulae (10, 11, 14), we get:

$$\zeta = \boldsymbol{\varphi}'\tilde{\boldsymbol{R}}\tilde{\boldsymbol{R}}'\boldsymbol{\varphi} \tag{32}$$

$$\bar{\boldsymbol{R}}\bar{\boldsymbol{R}}' = \tilde{\boldsymbol{R}}\tilde{\boldsymbol{R}}' - (1+\zeta)\boldsymbol{\kappa}\boldsymbol{\kappa}' \tag{33}$$

$$\boldsymbol{\kappa} = (1+\zeta)^{-1}\tilde{\boldsymbol{R}}\tilde{\boldsymbol{R}}'\boldsymbol{\varphi} \tag{34}$$

It is possible to update directly the square-root factor $\boldsymbol{R}$. The formulae of RLS identification (10)–(14) then transform into two steps:

1. matrix-vector multiplication

$$\begin{bmatrix} \boldsymbol{a} \\ \varepsilon \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{R}}' & 0 \\ \tilde{\boldsymbol{\Theta}}' & 1 \end{bmatrix} \begin{bmatrix} -\boldsymbol{\varphi} \\ y \end{bmatrix}, \tag{35}$$

and

2. inverse update

$$\begin{bmatrix} s^{\frac{1}{2}} & s^{\frac{1}{2}}\boldsymbol{\kappa}' \\ 0 & \bar{\boldsymbol{R}}' \\ 0 & \breve{\boldsymbol{\Theta}}' \end{bmatrix} = \boldsymbol{G}\boldsymbol{\Omega}\boldsymbol{Q} \begin{bmatrix} 1 & \mathbf{o} \\ a & \tilde{\boldsymbol{R}}' \\ \varepsilon & \tilde{\boldsymbol{\Theta}}' \end{bmatrix}, \tag{36}$$

$$\boldsymbol{Q} \equiv \boldsymbol{\Phi}_n \cdots \boldsymbol{\Phi}_2 \boldsymbol{\Phi}_1, \tag{37}$$

$$\boldsymbol{\Phi}_i = \begin{bmatrix} \cos\phi_i & & \sin\phi_i & \\ & \boldsymbol{I} & & \\ -\sin\phi_i & & \cos\phi_i & \\ & & & \boldsymbol{I} \end{bmatrix}, \tag{38}$$

$$\boldsymbol{\Omega} = \begin{bmatrix} 1 & & \\ & (1/\sqrt{\lambda})\boldsymbol{I}_{n\times n} & \\ & & 1 \end{bmatrix}, \tag{39}$$

$$\boldsymbol{G} = \begin{bmatrix} 1 & & \\ & \boldsymbol{I} & \\ -\varepsilon/\sqrt{s} & & 1 \end{bmatrix}, \tag{40}$$

where $\boldsymbol{Q}$ is an orthogonal matrix given as a product of elementary rotations $\boldsymbol{\Phi}_n \cdots \boldsymbol{\Phi}_1$ with rotation matrix $\boldsymbol{\Phi}_i$ zeroing the ith element of vector $\boldsymbol{a}$ with respect to the first element of the same column in the composed matrix (the rotation matrix has sines and cosines of the rotation angle in intersections of the first and the ith row and column); $\boldsymbol{\Omega}$ is a weighting matrix and matrix $\boldsymbol{G}$ is used to update the parameters.

Regularization as Input of Alternative Data

It is important for the systolic implementation of regularization that the addition of alternative parameters may be considered an input of some alternative data, which is processed in the estimator in the same way as the data samples from the identified system.

Using the partitioning (4), it is possible to introduce triangular square root decomposition of matrix $\boldsymbol{V}^*$ and to express $\boldsymbol{V}^*$ as a sum of data dyads ($\bullet$ represents a scalar don't care term)

$$\begin{bmatrix} \boldsymbol{V}_{\varphi}^* & \boldsymbol{v}_{\varphi y}^* \\ (\boldsymbol{v}_{\varphi y}^*)' & v_y^* \end{bmatrix} = \begin{bmatrix} \boldsymbol{U}^* & \boldsymbol{u}^* \\ 0 & \bullet^* \end{bmatrix}' \begin{bmatrix} \boldsymbol{U}^* & \boldsymbol{u}^* \\ 0 & \bullet^* \end{bmatrix} \doteq \sum_{i=1}^{n} \begin{bmatrix} \boldsymbol{U}_i^* & \boldsymbol{u}_i^* \end{bmatrix}' \begin{bmatrix} \boldsymbol{U}_i^* & \boldsymbol{u}_i^* \end{bmatrix}, \tag{41}$$

where

$$\left[\begin{array}{cc} \boldsymbol{U}_i^* & u_i^* \end{array}\right] \hat{=} \text{ith row of the matrix } \left[\begin{array}{cc} \boldsymbol{U}^* & \boldsymbol{u}^* \\ 0 & \bullet^* \end{array}\right]. \tag{42}$$

In a similar way, a square-root decomposition of $\boldsymbol{V}$ may be defined.

Using the above decomposition, the addition of the regularizing matrix $\boldsymbol{V}^*$ to the information matrix $\breve{\boldsymbol{V}}$ (29) may be expressed in a recursive form

$$\begin{array}{ll} \boldsymbol{u}^* := \boldsymbol{U}^* \boldsymbol{\Theta}^* & \text{a)} \\ \text{for } i = 1 \text{ to } n & \\ \quad \breve{\boldsymbol{V}} := \breve{\boldsymbol{V}} + \sqrt{\lambda_N} \left[\begin{array}{cc} \boldsymbol{U}_i^* & u_i^* \end{array}\right]' \sqrt{\lambda_N} \left[\begin{array}{cc} \boldsymbol{U}_i^* & u_i^* \end{array}\right] & \text{b)} \\ \text{end} & \\ \boldsymbol{V}(k+1|k) := \breve{\boldsymbol{V}} & \text{c)} \end{array} \tag{43}$$

where the first formula (43 a) results from (41) and (5).

The relation for addition of the regularizing "data" (43 b) has the same form as the formula of exponential update (28 b), with the only difference that instead of the information matrix, the input data is weighted by the forgetting factor.

Result:

1. *Regularization may be implemented in the same way as data update* – by entering recursively a vector of the "fictive data" (42). Each such vector contains one row of a triangular factor of the regularizing information matrix (42).

   The element of $\boldsymbol{u}^*$, which is used in this vector, is given as product $\boldsymbol{u}^* = \boldsymbol{U}^* \boldsymbol{\Theta}^*$. Since $\boldsymbol{\Theta}^*$ is assigned the last regularized estimate, it is possible to prepare this product while computing the exponential updates (i.e. before processing the regularizing input).

   Then, efficient pipelining of computation of $\boldsymbol{u}^*$ with processing of the input data is possible.

2. Entering the same data, the evolution of both the information matrix (if using the information filter [3]) and of the covariance matrix (if using the algorithm with inverse updates [9, 9]) is exactly the same (up to inversion):

$$\begin{aligned} \breve{\boldsymbol{V}}(1|0) &= \breve{\boldsymbol{C}}^{-1}(1|0) \\ &\vdots \\ \breve{\boldsymbol{V}}(N+1|N) &= \breve{\boldsymbol{C}}^{-1}(N+1|N). \end{aligned}$$

   After processing the *same* "regularizing data", it must hold:

$$\boldsymbol{V}(N+1|N) = \boldsymbol{C}^{-1}(N+1|N)$$

4. A SYSTOLIC IDENTIFICATION ALGORITHM

In this section, the basic ideas of systolic array will be explained and the systolic implementation of the covariance update RLS algorithm [6, 9] will be described.

Systolic array

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

Algorithm decomposition. All PEs in the array work synchronously. For implementation in a systolic array, the algorithm must be decomposed to elementary operations (e. g. multiplication of individual elements in vector-matrix multiplication). Each this operation is performed in one PE, its result is sent to the input of the neighbouring processor and there used in the next period.

Data pipelining. The data flow through the array in a *pipelined* fashion. They pass through the array in consecutive "waves", each of them being composed of one data vector. In each step, there are several "waves" passing through the array.

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

A simple example of a systolic array. Let us describe now the systolic matrix-vector multiplication, $\boldsymbol{a} = -\boldsymbol{R}'\boldsymbol{\varphi}$ (35) as a trivial example of a systolic array (actually, $\tilde{R}$ is used, but it is unimportant for this purpose).

The elements of $\boldsymbol{a}$ are scalar products of the rows of $\tilde{\boldsymbol{R}}'$ with the vector $\boldsymbol{\varphi}$. The ith element, a_i, is given by:

$$a_i = -\sum_{j=1}^{i} R'_{ij}\varphi_j = a_{i\underline{k}} - \sum_{j=1}^{k-1} \tilde{R}'_{ij}\varphi_j,\ n = \partial\varphi, i = 1 \ldots n \tag{44}$$

$$a_{i\underline{k}} = -\sum_{j=k}^{i} \tilde{R}'_{ij}\varphi'_j, \tag{45}$$

where the *underlined* index $\underline{k}$ denotes number of elements $\tilde{R}'_{ij}\varphi_j$, accumulated in a partial result.

For systolic implementation of this computation, a triangular array may be used. Each PE has an internal register to store R'_{ij}, two inputs and two outputs, the first pair to pass $a_{i\underline{k}}$, the second pair to pass φ_j. Its function is described in Figure 3.

Let an infinite sequence of vectors

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

(k is a discrete time index) be multiplied, one vector after the other.

Using systolic array, $\varphi(k+1)$ may be entered as soon as $\varphi(k)$ has been processed in the diagonal PEs and passed to the first sub-diagonal. The intermediate results then propagate through the array in subsequent waves.

The process is shown in Figure 4, which traces motion of vectors through the systolic array over several steps of the pipelined multiplication. The boxes in the figure represent individual processors of the array, each storing *one element* of matrix $\boldsymbol{R}$.

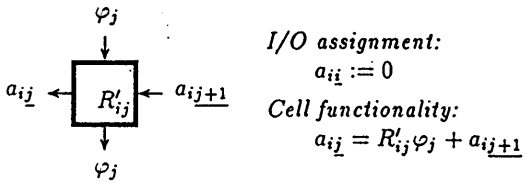


Fig. 3. Function scheme of PE for multiplication.

Schematic chart of the array, time skew of input/output vectors. In practice, usually only a simple schematic chart combined with the cell descriptions is used to describe the array.

This chart shows the contour of the array, the directions of input and output and (eventually) time skew of the entering/exiting vectors (since the elements of input (output) vector often enter (emerge from) the array not all at once, but one after another, each element also entering (emerging from) a different PE).

A schematic chart of our array for multiplication is given in Figure 6. The reader should notice, how the time skew of the output vector $\boldsymbol{a}$ (cf. Figure 4) is represented.

5. PARALLEL IMPLEMENTATION OF THE ESTIMATOR

Having explained the function principles of the systolic array, it is possible to describe the systolic implementation of the RLS algorithm with update of the covariance matrix. As described above, this update consists of two steps, in the first of which is a vector-matrix multiplication performed and in the second the covariance and parameter estimates are updated. Here, both steps will be first described separately and then, their merging into one array will be treated.

Systolic implementation of the first step of update

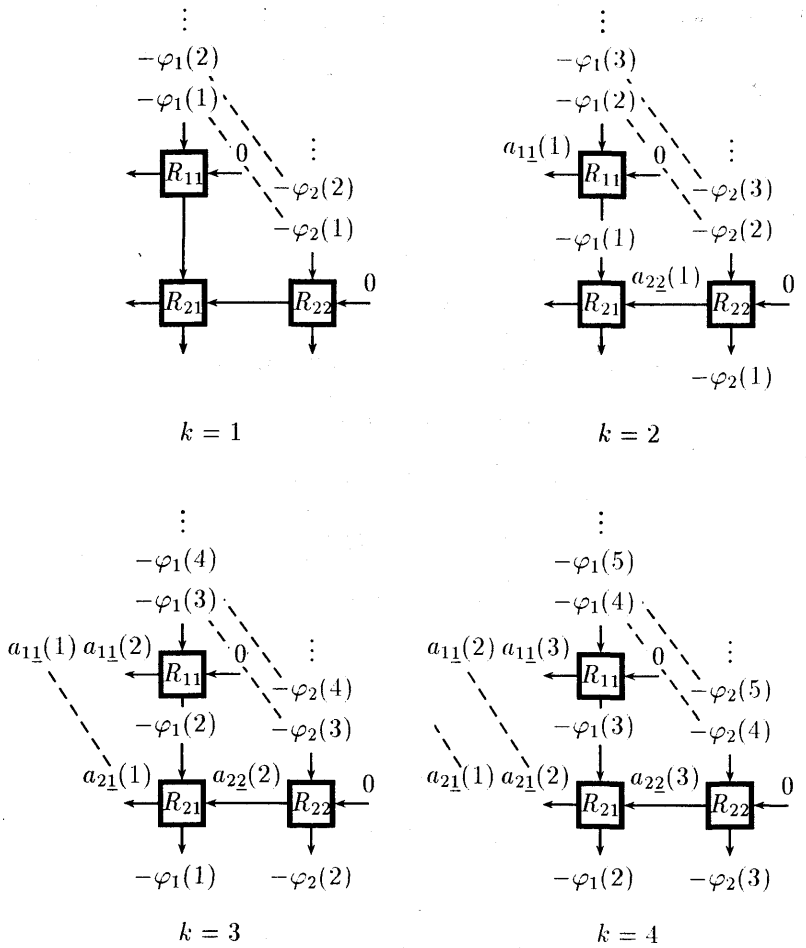
The transposed factor of the covariance matrix $\boldsymbol{R}'$ (31) and the vector of parameters $\boldsymbol{\Theta}$ (13) are stored in an $(n+1 \times n+1)$ dimension lower triangular systolic array. $\boldsymbol{R}'$ is stored in the upper part of the array, $\boldsymbol{\Theta}$ in the bottom row. The scheme of the array is in Figure 6 — cf. (35), function of the cells was described in Figure 3.

Systolic implementation of the second step of update

The scheme of the array implementing the second step of inverse update (36) is drawn in Figure 7. The auxiliary vector $\boldsymbol{a}$ enters the array from left; it is used

to compute the rotations in the left column. The rotations are computed from top down, beginning with $\cos\phi_1$, $\sin\phi_1$, using scalar equal to 1 when entered and propagating through the array from top downwards — cf. (36). They are propagated rightwards through the array.

The zero vector from the top of composed matrix on the right-hand side of (36), which is used to accumulate the Kalman gain κ, is entered through the diagonal of the array and propagated in the vertical direction.



$$a_{11} = a_1, \quad a_{21} = a_2$$

Fig. 4. Tracing of matrix-vector multiplication.

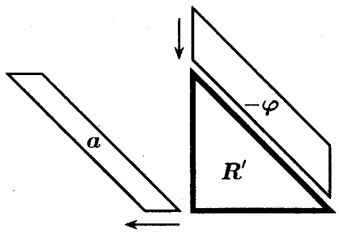


Fig. 5. Schematic chart of an array for multiplication of a triangular matrix with data vector.

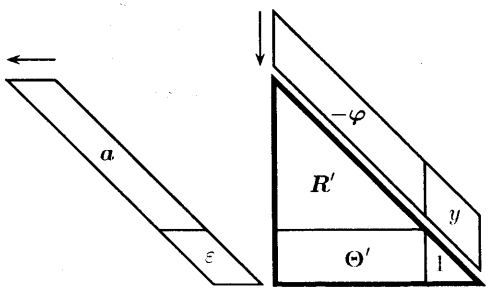


Fig. 6. Implementation of the first step of data update.

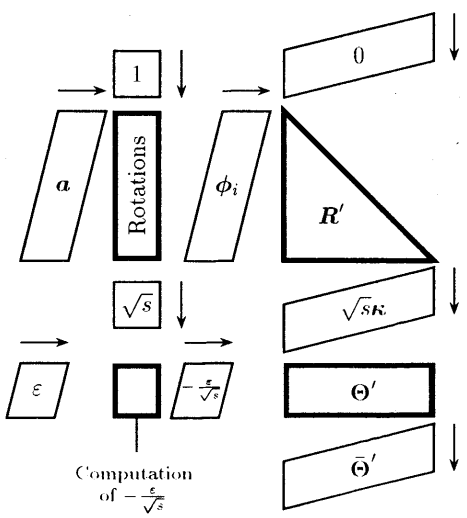
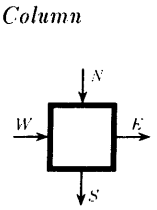


Fig. 7. Systolic implementation of the RLS update.

For completeness, PEs used in upper part of the array are described in Figure 8. Description of PEs in the bottom row is not given, but may easily be derived. Here,

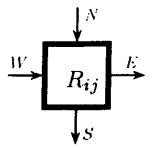
'tilde' and 'bar' symbols are not used here in the sense of the variable before data update and after regularization; 'tilde' symbol is rather used to refer to the value of variable on the input of PE and 'bar' symbol to refer to the value on the output of PE.



Left Column

I/O assignment:
N: top element: 1
other elements: $\sqrt{\tilde{s}}$
S: $\sqrt{\tilde{s}}$
W: a_i
E: $\cos\phi_i$, $\sin\phi_i$

Cell functionality:
$\cos\phi_i := \sqrt{\tilde{s}}/\sqrt{\tilde{s}+a_i^2}$
$\sin\phi_i := a_i/\sqrt{\tilde{s}+a_i^2}$
$\sqrt{\bar{s}} := \sqrt{\tilde{s}}\cos\phi + a_i \sin\phi$



Triangular 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:
$C_\phi := \cos\phi_i$, $S_\phi := \sin\phi_i$
$\sqrt{\bar{s}}\bar{k}_j := C_\phi\sqrt{\tilde{s}}\tilde{k}_j + S_\phi \tilde{R}_{ij}$
$\bar{R}_{ij} := (1/\sqrt{\lambda})(-S_\phi\sqrt{\tilde{s}}\tilde{k}_j + C_\phi \tilde{R}_{ij})$

Fig. 8. Description of the processors in the left column of the update array.

Implementation of both steps in one array

Our aim to perform both steps of data update — the multiplication (35) and the inverse update (36) — simultaneously in one composed array, as in Figure 9. However, merging simply both the above described arrays together, the following problem would arise: since the partially computed products $a_{i\underline{k}}$ and the rotations ϕ_i pass through the array in opposite directions, for the multiplication the elements of $\boldsymbol{R}$ would be used, that have not yet been updated in the previous recursion.

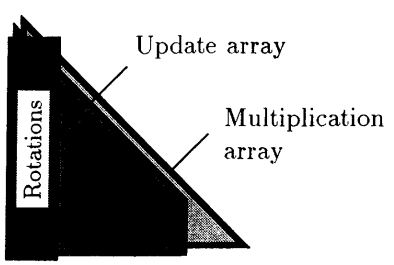


Fig. 9. Merging both steps of update in one array.

The solution of this problem was described in [6, 7, 9] and is the following: the respective elements of the data vector φ are multiplied by the old, non-updated elements of the factor of covariance matrix $\boldsymbol{R}$, but before the product is passed to the next PE, it is rotated with an added correction. This correction is performed in each cell, resulting in correct final product. For proper time-space synchronization of the values propagating through the array, delay is added into the horizontal links. Applying the '2-slowing lemma', used in the systolic algorithm design [8], the delay may be introduced by running only half of PEs in each step (Figure 10).

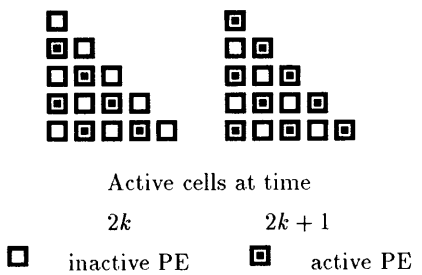


Fig. 10. Switching of active and inactive cells in RLS array.

The resulting array is sketched in Figure 11, $\boldsymbol{\alpha}$ is an auxiliary vector. Its first element is number 1 entering the computations of rotations, zeros on the subsequent positions are the corrections, rotated as the vector passes through the array. The zero vector β is used for computation of $s^{\frac{1}{2}}\boldsymbol{\kappa}$.

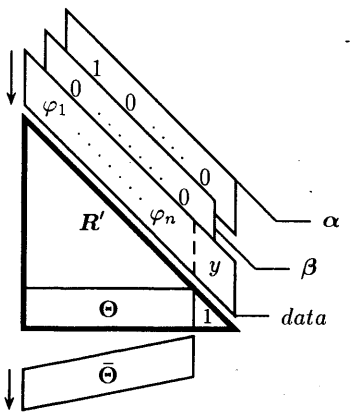


Fig. 11. Scheme of pipelined array for the RLS estimator.

The function of PEs in the array is described by the following equations ($\bar{\tilde{\boldsymbol{a}}}$ denotes $\boldsymbol{a}$ after the matrix-vector multiplication, but before the update itself):

Upper part ($\boldsymbol{R}'$):

First column cells:

$$\begin{aligned} \bar{\tilde{a}}_i &:= \tilde{a}_i - \tilde{R}'_{i1}\varphi_1 \\ S_\phi = \sin\phi_i &:= \bar{\tilde{a}}_i / \sqrt{\tilde{\alpha}_1^2 + \bar{\tilde{a}}_i^2} \\ C_\phi = \cos\phi_i &:= \tilde{\alpha}_1 / \sqrt{\tilde{\alpha}_1^2 + \bar{\tilde{a}}_i^2} \\ \begin{bmatrix} \bar{\beta}_1 & \bar{\alpha}_1 \\ \bar{R}_{i1} & 0 \end{bmatrix} &:= \begin{bmatrix} 1 & 0 \\ 0 & 1/\sqrt{\lambda} \end{bmatrix} \begin{bmatrix} C_\phi & S_\phi \\ -S_\phi & C_\phi \end{bmatrix} \begin{bmatrix} \tilde{\beta}_1 & \tilde{\alpha}_1 \\ \tilde{R}_{i1} & \bar{\tilde{a}}_i \end{bmatrix}. \end{aligned}$$

Internal cells:

$$\begin{aligned} \bar{\tilde{a}}_i &:= \tilde{a}_i - \tilde{R}_{ij}\varphi_j \\ \begin{bmatrix} \bar{\beta}_j & \bar{\alpha}_j \\ \bar{R}_{ij} & \bar{a}_i \end{bmatrix} &:= \begin{bmatrix} 1 & 0 \\ 0 & 1/\sqrt{\lambda} \end{bmatrix} \begin{bmatrix} C_\phi & S_\phi \\ -S_\phi & C_\phi \end{bmatrix} \begin{bmatrix} \tilde{\beta}_j & \tilde{\alpha}_j \\ \tilde{R}_{ij} & \bar{\tilde{a}}_i \end{bmatrix}. \end{aligned}$$

The cells in the bottom row are updated in a similar way.

6. SYSTOLIC IMPLEMENTATION OF THE BLOCK REGULARIZATION

This section contains the most important part of this paper — the implementation of the block-regularization in the systolic inverse-updated RLS algorithm will be described here.

Summarizing the above given description of the block-regularized forgetting, we find that the following mechanisms must be implemented:

- Multiplication of n rows of matrix $\boldsymbol{U}^*$ (41, 42) by $\boldsymbol{\Theta}^*$ (43 a)
- Switching between the data samples from the identified system and the regularizing "data"
- Computation of the accumulated forgetting factor λ_N (28 c, d)
- Switching on and off of the exponential forgetting (when processing the regularizing data, the exponential forgetting is not used — cf. (28 b) and (29 b, c)).
- Storing of $\boldsymbol{\Theta}^*$ during multiplication with $\boldsymbol{U}^*$ (n vectors $\boldsymbol{U}_i^*$ are multiplied with $\boldsymbol{\Theta}^*$ *consecutively*), writing of new $\boldsymbol{\Theta}^*$ to the storage.

It will be supposed for simplicity that the forgetting factor λ is constant. Then, λ_N and the product $\sqrt{\lambda_N}\boldsymbol{U}^*$ may be computed in advance.

Remark on notation

In the following text, $\boldsymbol{U}^{\lambda *}$ will be used to refer the product,

$$\boldsymbol{U}^{\lambda *} = \sqrt{\lambda_N}\boldsymbol{U}^{*}. \tag{46}$$

Similar notation will be used also for $\sqrt{\lambda_N}\boldsymbol{u}^{*}$

$$\boldsymbol{u}^{\lambda *} = \sqrt{\lambda_N}\boldsymbol{u}^{*} = \sqrt{\lambda_N}\boldsymbol{U}^{*}\boldsymbol{\Theta}^{*}. \tag{47}$$

The rows of $\boldsymbol{U}^{\lambda *}$ will be referred to as $\boldsymbol{U}_i^{\lambda *}$ and the elements of $\boldsymbol{u}^{\lambda *}$ as $u_i^{\lambda *}$.

Selection and storing of $\boldsymbol{\Theta}^{*}$

For implementation reasons (simplicity of mechanism to control writing $\boldsymbol{\Theta}^{*}$ to the storage), an assignment

$$\boldsymbol{\Theta}^{*}(k) := \breve{\boldsymbol{\Theta}}(k-n) \tag{48}$$

will be used for $\boldsymbol{\Theta}^{*}$, instead of (30), if $N > n$ — if the data block is longer than is the dimension of $\boldsymbol{\Theta}$ (this may be the case if using the regularized estimator for adaptive control [12]).

Hence, if $N > n$, only exponential forgetting is applied on the last regularized estimate $\bar{\boldsymbol{\Theta}}$ during the $N - n$ steps before it is stored in the $\boldsymbol{\Theta}^{*}$ storage. In result, N may be longer than n only to that degree that $\boldsymbol{\Theta}$ does not lose stability at the beginning of the data block for the given quality of input data.

For multiplication by the rows of matrix $\boldsymbol{U}^{*}$, $\boldsymbol{\Theta}^{*}$ is stored where it has been computed — in the bottom row of the array.

Loading of $\boldsymbol{U}^{\lambda *}$ into the array and its multiplication by $\boldsymbol{\Theta}^{*}$

The matrix $\boldsymbol{U}^{*}$ is entered into the array through the bottom row of the array, skewed in time. In the bottom row, it is also multiplied by $\boldsymbol{\Theta}^{*}$ (Figure 12).

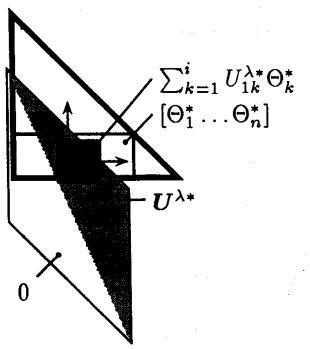


Fig. 12. Loading of the regularizing information matrix into the array and its multiplication with regularizing parameter estimates.

From the bottom row, the elements of $U^{\lambda *}$ are shifted up through the array to the diagonal and there processed as the alternative data.

For proper pipelining of $U^{\lambda *}$ with the data samples (so that $U_1^{\lambda *}$ arrives to the diagonal just after the last sample in the data block has been processed), U_{11}^{*} (the upper left element of the matrix) must clearly be entered to the bottom row of the array n steps before the end of the data block.

Because of switching of active PEs in the array (Figure 10), the rows of $U^{\lambda *}$ must be entered only every second tact period of the array.

Control Signal

To synchronize all mechanisms in the array, a control signal is used. This signal, aligned with $U^{\lambda *}$, is propagated through the array, first upwards (Figure 13) – it is used in the bottom row to switch on and off writing of the Θ estimates to the storage of Θ^* and, when it reaches the diagonal of the array, to switch the array entries between the data samples and the regularizing vectors. In the upper left PE it is "bounced" and propagated through the array to switch on and off the forgetting. Complete path of the control signal through the array is sketched in Figure 14.

Finally, Figure 15 shows time alignment of the control matrix and regularizing data with the data samples rom the identified system.

In the cell descriptions (Figures 16 – 19), the control signal is referred to as Ctrl_1 when shifted upwards and as Ctrl_2 when propagated downwards.

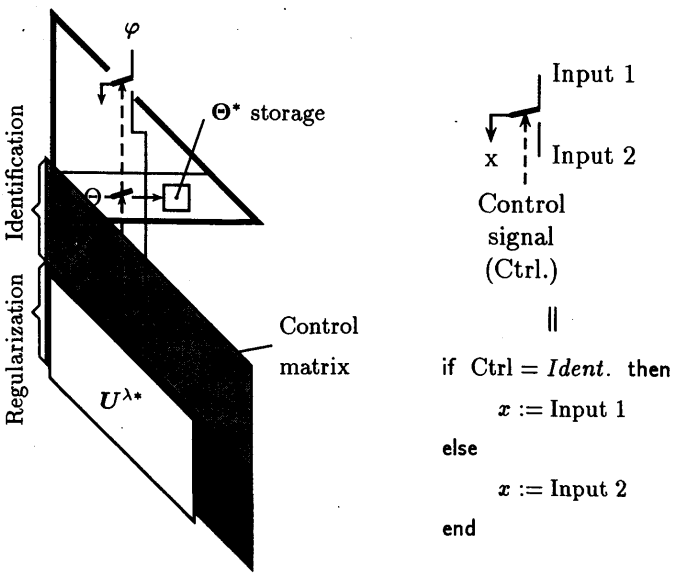


Fig. 13. Switching of the of regularizing parameters storage and of the data entries.

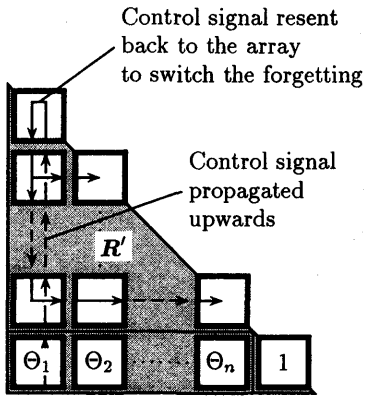


Fig. 14. Propagation of control signal through the array.

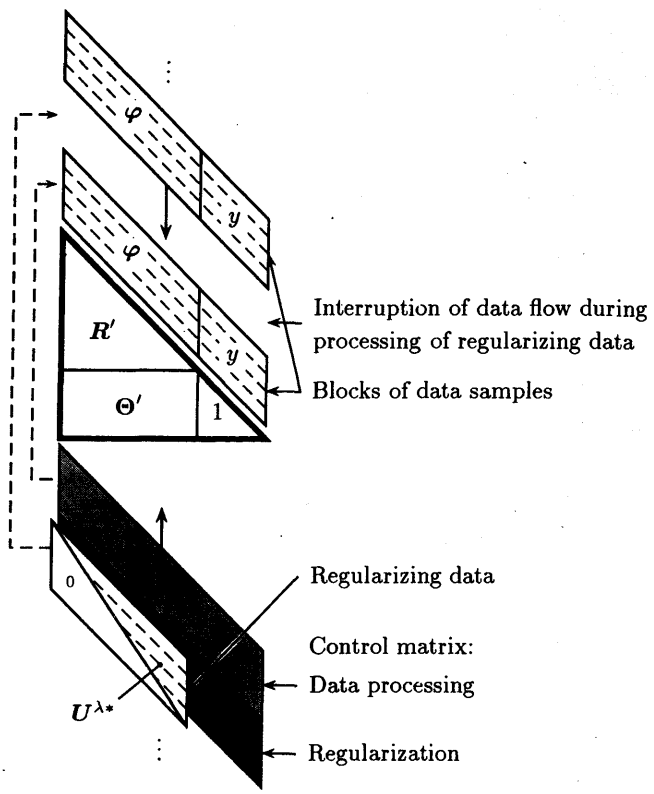
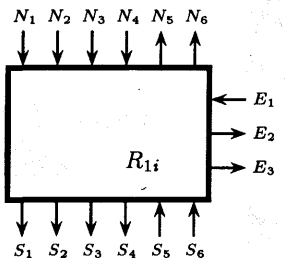


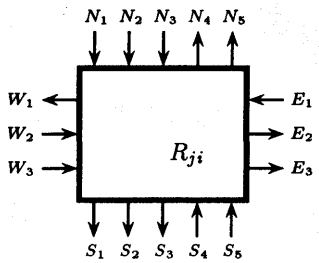
Fig. 15. Alignment of control matrix with the data samples.



Cell functionality:
Top cell:
$\mathrm{Ctrl}_2 := \mathrm{Ctrl}_1$
if $\mathrm{Ctrl}_1 = 1$ then
$x_1 := \varphi_1$
else
$x_1 := U_{i1}^{\lambda *}$
end
All cells:
$\mathrm{aux} := \tilde{a}_j + \tilde{R}_{1i} x_1$
$C_\phi := \cos\phi_i := \tilde{\alpha}_1 / (\tilde{\alpha}_1^2 + \mathrm{aux}^2)^{\frac{1}{2}}$
$S_\phi := \sin\phi := \mathrm{aux} / (\tilde{\alpha}_1^2 + \mathrm{aux}^2)^{\frac{1}{2}}$
$\bar{\beta}_1 := C_\phi \tilde{\beta}_1 + S_\phi \tilde{R}_{1i}$
$\bar{\alpha}_1 := C_\phi \tilde{\alpha}_1 + S_\phi \,\mathrm{aux}$
$\bar{R}_{1i} := -S_\phi \tilde{\beta}_1 + C_\phi \tilde{R}_{1i}$
if $\mathrm{Ctrl}_2 = 1$ then
$\bar{R}_{1i} := (1/\lambda) \bar{R}_{1i}$
end

I/O assignment:
N_1: top element: φ_1
other elements: x_1
N_2: top element: $\tilde{\alpha}_1 = 1$
other elements: $\tilde{\alpha}_1$
N_3: top element: $\tilde{\beta}_1 = 0$
other elements: $\tilde{\beta}_1$
N_4: top element: not used
other elements: Ctrl_2
S_4: Ctrl_2
S_5: Ctrl_1
S_6: $U_{i1}^{\lambda *}$
E_1: top element: 0
other elements: $\tilde{a}_i$
E_2: top element: not used
other elements: ϕ_i
E_3: top element: not used
other elements: Ctrl_2

Fig. 16. Processing cell in the left column of the upper part of array.

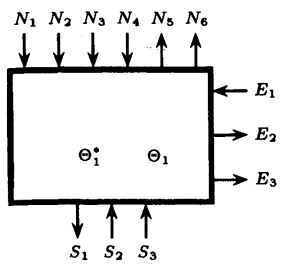


Cell functionality:
Diagonal cells:
if $\mathrm{Ctrl}_1 = 1$ then $x_j := \varphi_j$
else $x_j := U_{ij}^{\lambda *}$ end
All cells:
$C_\phi := \cos \phi_i$
$S_\phi := \sin \phi_i$
$\mathrm{aux} := \tilde{a}_i + \tilde{R}_{ji} x_j$
$\bar{\beta}_j := C_\phi \tilde{\beta}_j + S_\phi \tilde{R}_{ji}$
$\bar{\alpha}_j := C_\phi \tilde{\alpha}_j + S_\phi \,\mathrm{aux}$
$\bar{R}_{ji} := -S_\phi \tilde{\beta}_j + C_\phi \tilde{R}_{ji}$
$\bar{a}_i := -S_\phi \tilde{\alpha}_j + C_\phi \,\mathrm{aux}$
if $\mathrm{Ctrl}_2 = 1$ then
$\bar{R}_{ji} := (1/\lambda)\bar{R}_{ji}$
$\bar{a}_i := (1/\lambda)\bar{a}_i$
end

I/O assignment:
N_1: diagonal elements: φ_j
other elements: x_j
N_2: diagonal elements: $\tilde{\alpha}_j = 0$
other elements: $\tilde{\alpha}_j$
N_3: diagonal elements: $\tilde{\beta}_j = 0$
other elements: $\tilde{\beta}_j$
N_4: diagonal elements: not used
other elements: Ctrl_1
N_5: diagonal elements: not used
other elements: $U_{ij}^{\lambda *}$
S_1: x_j
S_2: $\bar{\alpha}_j$
S_3: $\bar{\beta}_j$
S_4: Ctrl_1
S_5: $U_{ij}^{\lambda *}$
W_1: $\bar{a}_i$
W_2: ϕ_i
W_3: Ctrl_2
E_1: diagonal elements: 0
other elements: $\tilde{a}_i$
E_2: diagonal elements: not used
other elements: ϕ_i
E_3: Ctrl_2

Fig. 17. Processing cell in an internal column of the upper part of array.

First cell



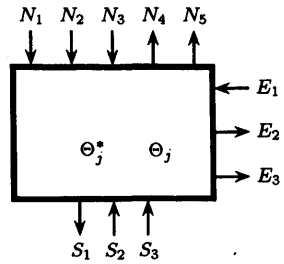
I/O assignment:

N_1: x_1
N_2: α_1
N_3: β_1
N_4: Ctrl_2
N_5: Ctrl_1
N_6: $U_{i1}^{\lambda *}$
S_1: Θ_1
S_2: $U_{i1}^{\lambda *}$
S_3: Ctrl_1
E_1: $\tilde{\varepsilon}_1$
E_2: ω
E_3: $U_{i1}^{\lambda *}\Theta_1^*$

Cell functionality:

$\mathrm{aux} := \tilde{\varepsilon}_1 + \tilde{\Theta}_1 x_1$
$\omega := -\mathrm{aux}/\alpha_1$
$\bar{\Theta}_1 := \omega\beta_1 + \tilde{\Theta}_1$
if $\mathrm{Ctrl}_1 = 1$ then
  $\Theta_1^* := \bar{\Theta}_1$
else
  $E_3 := U_{i1}^{\lambda *}\Theta_1^*$
end

Internal cells



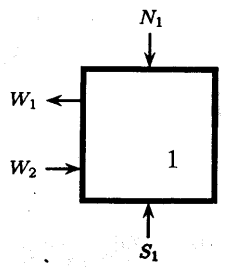
I/O assignment:

N_1: x_i
N_2: α_j
N_3: β_j
N_4: Ctrl_1
N_5: $U_{ij}^{\lambda *}$
S_1: Θ_j
S_2: $U_{ij}^{\lambda *}$
S_3: Ctrl_1
W_1: $\bar{\varepsilon}$
W_2: ω
W_3: $\sum_{k=1}^{j-1} U_{ik}^{\lambda *}\Theta_k^*$
E_1: $\tilde{\varepsilon}$
E_2: cells $\Theta_2 \ldots \Theta_{n-1}$: ω
  cell Θ_n: not used
E_3: $\sum_{k=1}^{j} U_{ik}^{\lambda *}\Theta^*(k)$

Cell functionality:

$\bar{\Theta}_j := \tilde{\Theta}_j + \omega\beta_j$
$\bar{\varepsilon} := \tilde{\varepsilon} + \tilde{\Theta}_j x_j + \omega\alpha_j$
if $\mathrm{Ctrl}_1 = 1$ then
  $\Theta_j^* := \bar{\Theta}_j$
else
  $E_3 := W_3 + U_{ij}^{\lambda *}\Theta_j^*$
end

Fig. 18. Processing cells in the bottom row of array.



I/O assignment:
N_1: y
W_2: $U_i^{\lambda *}\Theta^*$
S_1: Ctrl_1
Cell functionality:
if $\mathrm{Ctrl}_1 = 1$ then $W_1 := -y$
else $W_1 := -U_i^{\lambda *}\Theta^*$
end

Fig. 19. Last processing cell in the bottom row of the array.

Buffering of input data

Regularization interrupts processing of the input data for n periods every N periods (that is, during $N+n$ periods only N data samples are processed). Because of that, it is necessary to sample the identified system at a slower rate than the systolic estimator runs and to buffer the data samples. The ratio of the system sampling rate to the input rate of the estimator is equal to $N/N+n$, i.e. ranges from $1/2$ (worst case, for $N = n$) to 1 (limit case for exponential forgetting, i.e. $N \to \infty$).

Description of PEs in the array with the block-regularized forgetting

In this section, the resulting PEs implementing the block-regularized forgetting will be described.

Let us recall that, for better understandability, the control signal, when shifted upwards through the array, is referred to as Ctrl_1 and when 'bounced' and propagated down- and rightwards through the array, as Ctrl_2, though it is the same signal.

The description of the cells is given in Figure 16 – 19.

7. CONCLUSIONS

The paper describes implementation of the block-regularized exponential forgetting in the systolic algorithm for the Recursive Least Squares (RLS) identification with update of the covariance matrix (so called inverse-updated RLS algorithm). It is, to our knowledge, first attempt to increase robustness of the systolic implementation of the covariance filter to the non-informative data.

The paper introduces the reader into the principle of the regularization and of the block-accumulated regularization.

It treats the problems connected with systolic implementation of the block-regularized forgetting in the inverse-updated RLS estimator and finally gives a cell-level description of the resulting algorithm.

The regularized forgetting prevents the covariance matrix from unlimited growth in the conditions of poor excitation. The regularizing matrix may be an arbitrary

symmetric positive definite matrix, the choice depends on the user's needs. As an example, the use of the matrix obtained from the preliminary analysis of the estimated system may be given. The advantage of the method is that it is capable to preserve the value of parameter estimates, identified from informative data, in the conditions of poor excitation.

Using only the connections between the neighbouring processors, the proposed implementation of the block-regularized forgetting preserves compactness of the systolic estimator with exponential forgetting. The throughput of the resulting algorithm is reduced only to one half, compared with that of algorithm with exponential forgetting.

(Received December 12, 1993.)

REFERENCES

[1] D. D. Baer and J. Paradeans: A formal definition for systolic systems. In: Parallel Algorithms and Architectures (A. Albrecht, H. Jung and K. Mehlhorn, eds.), Lecture Notes in Computer Science, Springer–Verlag, Berlin 1987.

[2] L. D. J. Eggermont et al. (eds.): VLSI Signal Processing VI. In: Proceedings of the IEEE Signal Processing Society Workshop. IEEE Press, Veldhoven 1993.

[3] J. Kadlec: The cell-level description of systolic block regularised QR filter. In: Proceedings of the IEEE Signal Processing Society Workshop (Eggermont et al., eds.), Veldhoven 1993, pp. 298–306.

[4] J. Kadlec, F. M. F. Gaston and G. W. Irwin: Systolic implementation of the regularised parameter estimator. In: VLSI Signal Processing V (K. Yao et al., eds.), IEEE Press, New York 1992, pp. 520–529.

[5] M. Kárný et al.: Design of linear quadratic adaptive control: Theory and algorithms for practice. Kybernetika *21* (1985), Supplement.

[6] J. G. McWhirter: Systolic array for recursive least squares by inverse iterations. In: Proceedings of the IEEE Signal Processing Society Workshop (Eggermont et al., eds.), Veldhoven 1993, pp. 435–443.

[7] J. G. McWhirter: A systolic array for recursive least squares estimation by inverse updates. In: International Conference on Control '94, IEE, London 1994.

[8] G. M. Megson: An Introduction to Systolic Array Design. Oxford University Press, Oxford 1992.

[9] M. Moonen and J. G. McWhirter: A systolic array for recursive least squares by inverse updating. Electronics Letters *29* (1993), 13, 1217–1218.

[10] V. Peterka: Bayesian approach to system identification. In: Trends and Progress in System Identification (P. Eykhoff, ed.), IFAC Series for Graduates, Research Workers and Practising Engineers, Chapter 8. Pergamon Press, Oxford 1981.

[11] V. Peterka: Control of uncertain processes: Applied theory and algorithms. Kybernetika *22* (1986), Supplement.

[12] J. Schier: Parallel Algorithms for Robust Adaptive Identification and Square–root LQG Control. Ph.D. Thesis, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University, Prague 1994.

[13] J. Schier: A Systolic Algorithm for the Block-regularized RLS Identification. Research Report No. 1807, Institute of Information Theory and Automation, Prague 1994.

Ing. Jan Schier, CSc. Ústav teorie informace a automatizace AV ČR (Institute of Information Theory and Automation – Academy of Sciences of the Czech Republic), Pod vodárenskou věží 4, 182 08 Praha 8. Czech Republic.