Sliding Window Evaluation of the Wiener-Hopf Equation

Alfred KRAKER, Barna CSUKA, Zsolt KOLLAR

Dept. of Measurement and Information Systems, Budapest University of Technology and Economics
Magyar Tudósok körútja 2. (Building I) H-1117 Budapest, Hungary

{csuka, kollarzs}@mit.bme.hu

Submitted January 30, 2020 / Accepted April 14, 2020

Abstract. *This paper presents an efficient method for solving the Wiener-Hopf equation in a sliding window by calculating the correlation matrices recursively. Furthermore, a novel algorithm is introduced for evaluating the inverse of the auto-correlation matrix – the Recursion with Splitting the Correlation matrix into 4 Blocks for Inversion (RSC4BI) – which can significantly reduce the computational requirements. The presented procedure is optimized for special cases to achieve an efficient implementation which allows faster real-time signal processing or to reduce the response time – e.g. the latency – by distributing the computations over the time. The proposed method is also validated through numerical simulations and hardware implementation.*

Keywords

Adaptive filtering, sliding window Wiener filter, Wiener-Hopf equation, recursive matrix inversion, RSC4BI

1. Introduction

In this article, it is assumed that the system is a Finite Impulse Response (FIR) filter. This FIR filter is modelled as a Wiener Filter (WF) [1] which is able to track a system by linear time-invariant filtering. To perform the estimation of the filter coefficients, two statistical functions have to be calculated in matrix form: the auto-correlation of the input signal and the cross-correlation of the input and output signals. Using these matrices, the Wiener-Hopf (WH) equation can be formulated and its solution provides the optimum for the filter coefficients in a Minimum Mean Square Error (MMSE) sense [2]. These MMSE-filters – and their modified versions – are widely used in signal processing to identify a given dynamic system [3–6].

During the solution of the WH equation the auto-correlation matrix has to be inverted. Several algorithms can be applied to solve the inversion of matrices, depending on the properties of the matrix [7], [8]. The computational load of the inversion becomes critical, if a low complexity hardware with limited performance is applied. Furthermore, this operation is also critical, if real-time adaptive filtering has to be performed by the WF: during this adaptive method the filter coefficients adapt over time to track the internal changes of the dynamic system.

The challenges, that require this adaptive and real-time filtering are very diverse, such as compensation of current transformers [9], noise cancellation in microphones [10], or to calculate charge density on a dielectric surface [11]. Studying these examples, it can be stated that the matrix inversion is crucial step of these procedures due the $O\left(n^3\right)$ complexity. The applicability of this configuration is limited because of the real-time environment, as the applied matrices have to be small enough to fulfill the real-time conditions. Considering these circumstances, such algorithms have to be applied which can evaluate the WH equation in real-time and with low computational complexity. A common method for performing matrix inversion in a recursive manner is the split Levinson algorithm [12], [13]. Although the poor stability limits the usage of the method [14], but its behaviour can be predicted and compensated by computing the condition number [15].

This article presents a novel method for the efficient solution of the WH equation in a sliding window, the Sliding Wiener Filter (SWF). The direct calculation of the inverse of the auto-correlation matrix can be avoided by applying the Recursion with Splitting the Correlation matrix into 4 Blocks for Inversion (RSC4BI). Furthermore, not only the inverse, but the auto-correlation matrix and cross-correlation vector are also recursively evaluated using the results of the previous calculations.

The paper is organized as follows. Section 2 gives a short overview of adaptive filtering and the WH equation. The applied notations for the investigated signals are introduced as well. Section 3 describes the method of SWF, and the RSC4BI is presented as well. In Sec. 4 the complexity analysis of SWF is investigated and compared with the conventional WH solution. An optimized version of the RSC4BI algorithm is also presented for low complexity implementation. Simulation results for the proposed method in terms of runtime and quantization error are presented in Sec. 5. Finally, the conclusions are drawn.

Throughout the paper the following notations are used. The matrices are denoted by capital bold letters, vectors by small bold letters and scalars by plain letters. The superscript $()^{\mathrm{T}}$ denotes matrix or vector transposition; $()^{-1}$ denotes the multiplicative inverse of a matrix. The applied normal distributions are denoted by the common notation: $\mathcal{N}\left(\mu, \sigma^2\right)$, where μ is the mean and σ^2 is the variance of the distribution, while the $\mathrm{E}\{\cdot\}$ operator means the expected value of the parameter, and the average of the parameter is denoted by an overline: $\overline{(\cdot)}$.

2. Adaptive Filtering

The adaptive filter is modeled as a linear time-invariant filter having a finite impulse response with K coefficients. As a result, the output signal of the adaptive filter can be expressed by convolution using the coefficients $h(n)$ and the input signal $x(n)$ as

$$y(n) = x(n) * h(n) = \sum_{k=0}^{K-1} x(n-k)h(k). \tag{1}$$

The previous equation can be expressed also as matrix multiplication in an observation window containing N samples:

$$\mathbf{y}(n) = \mathbf{X}(n)\mathbf{h}(n) \tag{2}$$

where vectors $\mathbf{y}(n)$ and $\mathbf{h}(n)$ are expressed using the signals $y(n)$ and $h(n)$, respectively as:

$$\mathbf{y}^{\mathrm{T}}(n) = \begin{bmatrix} y(n) & y(n+1) & \dots & y(n+N-1) \end{bmatrix}, \tag{3}$$

$$\mathbf{h}^{\mathrm{T}}(n) = \begin{bmatrix} h(0) & h(1) & \dots & h(K-1) \end{bmatrix}, \tag{4}$$

and the data matrix $\mathbf{X}(n)$ with size $N \times K$ can be expressed using $x(n)$ as

$$\mathbf{X}(n) = \begin{bmatrix} x(n) & x(n-1) & \dots & x(n-K+1) \\ x(n+1) & x(n) & \dots & x(n-K+2) \\ \vdots & \vdots & \ddots & \vdots \\ x(n+N-1) & x(n+N-2) & \dots & x(n+N-K) \end{bmatrix}. \tag{5}$$

The error signal vector $\boldsymbol{\varepsilon}$ is the expected value of the difference of the desired signal vector $\mathbf{d}(n)$ and the output signal vector $\mathbf{y}(n)$:

$$\boldsymbol{\varepsilon}(n) = \mathrm{E}\{\mathbf{d}(n) - \mathbf{y}(n)\} \tag{6}$$

where $\mathbf{d}^{\mathrm{T}}(n) = \begin{bmatrix} d(n) & d(n+1) & \dots & d(n+N-1) \end{bmatrix}$.

The adaptive algorithm should minimize the cost function (C) as a function of the filter coefficients, considering (2) and (6) it can be expressed in MMSE sense as

$$C(\mathbf{h}(n)) = \mathrm{E}\left\{\boldsymbol{\varepsilon}^2(n)\right\} = \mathrm{E}\left\{\mathbf{d}(n)^{\mathrm{T}}\mathbf{d}(n)\right\} + \\ +\mathbf{h}^{\mathrm{T}}(n)\mathbf{R}(n)\mathbf{h}(n) - 2\mathbf{h}^{\mathrm{T}}(n)\mathbf{p}(n) \tag{7}$$

where

$$\mathbf{R}(n) = \mathrm{E}\left\{\mathbf{X}^{\mathrm{T}}(n)\mathbf{X}(n)\right\} \tag{8}$$

is the auto-correlation matrix with size $K \times K$ of the input signal, while

$$\mathbf{p}(n) = \mathrm{E}\left\{\mathbf{X}^{\mathrm{T}}(n)\mathbf{d}(n)\right\} \tag{9}$$

is the cross-correlation vector with size $K \times 1$ of the input and output signals.

The optimal solution for the filter coefficients – know as the WH equation – can be given in MMSE sense by the derivative of $C(\mathbf{h}(n))$ with respect to $\mathbf{h}(n)$. The optimal coefficients $\hat{\mathbf{h}}_{\mathrm{opt}}(n)$ with size $K \times 1$ are given in form [2]:

$$\hat{\mathbf{h}}_{\mathrm{opt}}(n) = \mathbf{R}^{-1}(n)\mathbf{p}(n). \tag{10}$$

3. Evaluation of the WH Equation

In this section the calculation steps for evaluating the WH equation in a sliding window of length N – by a sample-by-sample manner – are described. First, the matrix $\mathbf{X}(n+1)$ is expressed based on the previous matrix $\mathbf{X}(n)$ with the aid of a permutation matrix and an identity vector. Then, the correlation matrices $\mathbf{R}(n+1)$ and $\mathbf{p}(n+1)$ are expressed from the matrices elaborated in the previous step based on two different methods: using a permutation matrix and using dyadic products. Finally, the RSC4BI algorithm is introduced for the calculation of the inverse of the auto-correlation matrix $\mathbf{R}^{-1}(n+1)$. The algorithm calculates $\mathbf{R}^{-1}(n+1)$ using the previously calculated matrices $\mathbf{R}(n)$, $\mathbf{R}(n+1)$ and $\mathbf{R}^{-1}(n)$. The algorithm exploits higher efficiency by splitting the matrix into four submatrices.

For the representation of the column and row elements of matrix $\mathbf{X}(n)$ from (5), the following notation will be applied:

$$\mathbf{X}(n) = \begin{bmatrix} \mathbf{x}_1(n) & \mathbf{x}_2(n) & \dots & \mathbf{x}_K(n) \end{bmatrix} = \begin{bmatrix} \mathbf{x}^1(n) \\ \mathbf{x}^2(n) \\ \vdots \\ \mathbf{x}^N(n) \end{bmatrix} \tag{11}$$

where $\mathbf{x}_1, \mathbf{x}_2, \dots$ are the column vectors, and $\mathbf{x}^1, \mathbf{x}^2, \dots$ are the row vectors of $\mathbf{X}$.

3.1 Calculation of $\mathbf{X}(n+1)$

For the calculation of the matrix $\mathbf{X}(n+1)$ derived from the matrix $\mathbf{X}(n)$ the permutation matrix and the identity vector are defined. The $K \times K$ permutation matrix is given as

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \end{bmatrix}. \tag{12}$$

The $N \times 1$ identity vector – where only the ith element is 1 – is expressed as

$$\mathbf{e}_i^{\mathrm{T}} = \begin{bmatrix} 0 & \dots & 0 & e_i{=}1 & 0 & \dots & 0 \end{bmatrix}. \qquad (13)$$

Finally, the matrix $\mathbf{X}(n{+}1)$ can be expressed as a permuted version of $\mathbf{X}(n)$ and compensated by the dyadic product of the difference of the time domain vectors – $\mathbf{x}_1(n{+}1)$ and $\mathbf{x}_K(n)$ – and the identity vector:

$$\mathbf{X}(n{+}1) = \mathbf{X}(n)\mathbf{P} + (\mathbf{x}_1(n{+}1) - \mathbf{x}_K(n))\,\mathbf{e}_1^{\mathrm{T}}. \qquad (14)$$

3.2 Calculation of $\mathbf{R}(n{+}1)$

The auto-correlation matrix $\mathbf{R}(n{+}1)$ can be expressed using $\mathbf{R}(n)$ in the following 2 different methods.

3.2.1 Using Permutation Matrix

Applying (14), the transposed version of $\mathbf{X}(n{+}1)$ can be expressed as:

$$\mathbf{X}^{\mathrm{T}}(n{+}1) = \mathbf{P}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}(n) + \mathbf{e}_1\,(\mathbf{x}_1(n{+}1) - \mathbf{x}_K(n))^{\mathrm{T}}. \qquad (15)$$

As a further step, the matrix $\mathbf{R}(n{+}1)$ can be expressed through (8), (14) and (15) as:

$$\begin{aligned}\mathbf{R}(n{+}1) =& \mathbf{X}^{\mathrm{T}}(n{+}1)\mathbf{X}(n{+}1) = \\ =& \mathbf{P}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}(n)\mathbf{X}(n)\mathbf{P} + \\ &+ \mathbf{P}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}(n)\,(\mathbf{x}_1(n{+}1) - \mathbf{x}_K(n))\,\mathbf{e}_1^{\mathrm{T}} + \\ &+ \mathbf{e}_1\,(\mathbf{x}_1(n{+}1) - \mathbf{x}_K(n))^{\mathrm{T}}\,\mathbf{X}(n)\mathbf{P} + \\ &+ \mathbf{e}_1\,(\mathbf{x}_1(n{+}1) - \mathbf{x}_K(n))^{\mathrm{T}}\,(\mathbf{x}_1(n{+}1) - \mathbf{x}_K(n))\,\mathbf{e}_1^{\mathrm{T}}\end{aligned} \qquad (16)$$

where the first term in (16) can be expressed with the of (8) as

$$\mathbf{P}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}(n)\mathbf{X}(n)\mathbf{P} = \mathbf{P}^{\mathrm{T}}\mathbf{R}(n)\mathbf{P}. \qquad (17)$$

As a result, the updated auto-correlation matrix is expressed from the previous results using the permutation matrix and the identity vector.

3.2.2 Using Dyadic Product

The auto-correlation matrix can also be expressed also through dyadic product as [16], [17]:

$$\mathbf{R}(n) = \sum_{i=1}^{N} (\mathbf{x}^i(n))^{\mathrm{T}}\mathbf{x}^i(n). \qquad (18)$$

The updated version of the auto-correlation matrix can be derived in a similar manner:

$$\mathbf{R}(n{+}1) = \sum_{i=1}^{N} (\mathbf{x}^i(n{+}1))^{\mathrm{T}}\mathbf{x}^i(n{+}1). \qquad (19)$$

Due to the calculations in a sliding window, it can be noted that

$$\mathbf{x}^i(n{+}1) = \mathbf{x}^{i+1}(n), \quad \text{for } i = 1,\dots,N{-}1. \qquad (20)$$

As a result, the updated auto-correlation matrix can be also expressed by subtracting the dyadic product of the oldest vector and adding the dyadic product of the newest vector to the previous matrix [16], [17]:

$$\begin{aligned}\mathbf{R}(n{+}1) =& \mathbf{R}(n) - \\ &- \left(\mathbf{x}^1(n)\right)^{\mathrm{T}}\mathbf{x}^1(n) + \\ &+ \left(\mathbf{x}^N(n{+}1)\right)^{\mathrm{T}}\mathbf{x}^N(n{+}1).\end{aligned} \qquad (21)$$

3.3 Calculation of $\mathbf{p}(n{+}1)$

The cross-correlation vector $\mathbf{p}(n{+}1)$ can be similarly expressed using $\mathbf{p}(n)$ in the following 2 different ways.

3.3.1 Using Permutation Matrix

Similarly to the method presented in Sec. 3.2.1, $\mathbf{d}(n{+}1)$ can expressed using the permutation matrix and the identity vector as

$$\mathbf{d}(n{+}1) = \mathbf{P}\mathbf{d}(n) - \mathbf{e}_N\,(d(n){-}d(n{+}N)). \qquad (22)$$

The updated cross-correlation vector can be derived based on (9), (15) and (22) as

$$\begin{aligned}\mathbf{p}(n{+}1) =& \mathbf{X}^{\mathrm{T}}(n{+}1)\mathbf{d}(n{+}1) = \\ =& \mathbf{P}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}(n)\mathbf{P}\mathbf{d}(n) - \\ &- \mathbf{P}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}(n)\mathbf{e}_N(d(n){-}d(n{+}N)) + \\ &+ \mathbf{e}_1\,(\mathbf{x}_1(n{+}1){-}\mathbf{x}_K(n))^{\mathrm{T}}\,\mathbf{P}\mathbf{d}(n) - \\ &- \mathbf{e}_1\,(\mathbf{x}_1(n{+}1){-}\mathbf{x}_K(n))^{\mathrm{T}}\,\mathbf{e}_N(d(n){-}d(n{+}N)).\end{aligned} \qquad (23)$$

As a result the updated cross-correlation vector is expressed with the aid of the permutation matrix and the identity vector.

3.3.2 Using Dyadic Product

Following the derivations presented in Sec. 3.2.2 the cross-correlation vector can be also expressed by a dyadic product as

$$\mathbf{p}(n) = \sum_{i=1}^{N} (\mathbf{x}^i(n))^{\mathrm{T}} d(n + i - 1). \qquad (24)$$

The consecutive cross-correlation vector in the sliding window can be calculated using a dyadic product as

$$\mathbf{p}(n{+}1) = \sum_{i=1}^{N} (\mathbf{x}^i(n{+}1))^{\mathrm{T}} d(n{+}i). \qquad (25)$$

Based on (20), (24) and (25), the updated cross-correlation vector can be expressed from the previous cross-correlation vector as

$$\begin{aligned}\mathbf{p}(n{+}1) =& \mathbf{p}(n) - \\ &- \left(\mathbf{x}^1(n)\right)^{\mathrm{T}} d(n) + \\ &+ \left(\mathbf{x}^N(n{+}1)\right)^{\mathrm{T}} d(n{+}N).\end{aligned} \qquad (26)$$

3.4 Calculation of $\mathbf{R}^{-1}(n+1)$ – the RSC4BI Algorithm

In this section an efficient algorithm for the calculation of the inverse of the updated auto-correlation matrix ($\mathbf{R}^{-1}(n+1)$) is presented. The following supplementary matrix notations and their splitting into four submatrices are used for the auto-correlation matrices and their inverse:

$$\mathbf{R}(n) = \mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{a}_{12} \\ \mathbf{a}_{21}^{\mathrm{T}} & a_{22} \end{bmatrix}, \quad (27)$$

$$\mathbf{R}(n+1) = \mathbf{B} = \begin{bmatrix} b_{11} & \mathbf{b}_{12}^{\mathrm{T}} \\ \mathbf{b}_{21} & \mathbf{B}_{22} \end{bmatrix}, \quad (28)$$

$$\mathbf{R}^{-1}(n) = \mathbf{U} = \begin{bmatrix} \mathbf{U}_{11} & \mathbf{u}_{12} \\ \mathbf{u}_{21}^{\mathrm{T}} & u_{22} \end{bmatrix}, \quad (29)$$

$$\mathbf{R}^{-1}(n+1) = \mathbf{V} = \begin{bmatrix} v_{11} & \mathbf{v}_{12}^{\mathrm{T}} \\ \mathbf{v}_{21} & \mathbf{V}_{22} \end{bmatrix}. \quad (30)$$

These notations can be seen in Fig. 1.

The derivation of inverse of such hypermatrices with 4 submatrices is presented in the Appendix. Based on (16) and (17) it can be shown that $\mathbf{A}_{11} = \mathbf{B}_{22}$. Furthermore, the submatrix $\mathbf{B}_{22}^{-1}$ can also be expressed using (A-3) and (A-7) as

$$\mathbf{B}_{22}^{-1} = \mathbf{A}_{11}^{-1} = \mathbf{U}_{11} - \mathbf{u}_{12}u_{22}^{-1}\mathbf{u}_{21}^{\mathrm{T}}. \quad (31)$$

With the result of the previous equation, the reduced submatrix of $\mathbf{B}$ can be also expressed using (A-7) as

$$b_{11,r} = b_{11} - \mathbf{b}_{12}^{\mathrm{T}}\mathbf{B}_{22}^{-1}\mathbf{b}_{21}. \quad (32)$$

As $\mathbf{V} = \mathbf{B}^{-1}$, the upper left corner element of $\mathbf{V}$ can also be easily expressed using (31), (32), and (A-3) as

$$v_{11} = b_{11,r}^{-1} = (b_{11} - \mathbf{b}_{12}^{\mathrm{T}}\mathbf{B}_{22}^{-1}\mathbf{b}_{21})^{-1}. \quad (33)$$

Similarly, the lower left submatrix of $\mathbf{V}$ can be also calculated using (A-4) as

$$\mathbf{v}_{21} = -\mathbf{B}_{22}^{-1}\mathbf{b}_{21}b_{11,r}^{-1}. \quad (34)$$

The upper right submatrix of $\mathbf{V}$ can be calculated using (A-5) or based on (34) as

$$\mathbf{v}_{12}^{\mathrm{T}} = \mathbf{v}_{21}^{\mathrm{T}} = -b_{11,r}^{-1}\mathbf{b}_{21}^{\mathrm{T}}\mathbf{B}_{22}^{-1}. \quad (35)$$

As a final step, the submatrix $\mathbf{V}_{22}$ can be expressed with the aid of (A-6) as

$$\mathbf{V}_{22} = \mathbf{B}_{22}^{-1} + \mathbf{B}_{22}^{-1}\mathbf{b}_{21}b_{11,r}^{-1}\mathbf{b}_{12}^{\mathrm{T}}\mathbf{B}_{22}^{-1}. \quad (36)$$

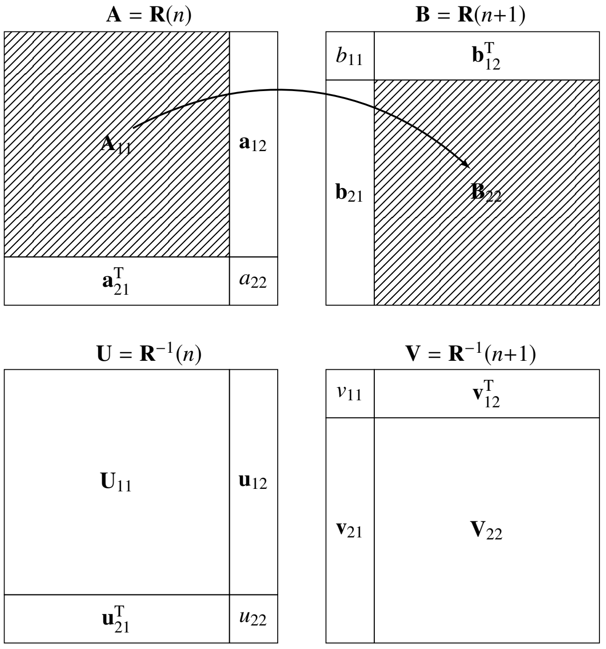


Fig. 1. Structure of the correlation matrix ($\mathbf{R}$) and its inverse ($\mathbf{R}^{-1}$).

Finally, the updated inverse auto-correlation matrix can be summed and expressed based on the matrices $\mathbf{B}$,$\mathbf{U}$ and the calculated submatrices of $\mathbf{V}$ as

$$\mathbf{R}^{-1}(n+1) = \mathbf{V} = \begin{bmatrix} b_{11,r}^{-1} & -b_{11,r}^{-1}\mathbf{b}_{21}^{\mathrm{T}}\mathbf{B}_{22}^{-1} \\ -\mathbf{B}_{22}^{-1}\mathbf{b}_{21}b_{11,r}^{-1} & \mathbf{B}_{22}^{-1} + \mathbf{B}_{22}^{-1}\mathbf{b}_{21}b_{11,r}^{-1}\mathbf{b}_{12}^{\mathrm{T}}\mathbf{B}_{22}^{-1} \end{bmatrix}. \quad (37)$$

The steps for calculating the inverse of the auto-correlation matrix – the RSC4BI algorithm – can be seen in Alg. 1.

Algorithm 1. The RSC4BI algorithm.

```
INPUT: $\mathbf{B} \equiv \mathbf{R}(n+1)$, $\mathbf{U} \equiv \mathbf{R}^{-1}(n)$
OUTPUT: $\mathbf{V} \equiv \mathbf{R}^{-1}(n+1)$
 function RSC4BI($\mathbf{B}$,$\mathbf{U}$)
   calculate $\mathbf{B}_{22}^{-1}$ using the submatrices of $\mathbf{U}$        (31)
   calculate $b_{11,r}$ using $\mathbf{B}_{22}^{-1}$ and $\mathbf{B}$              (32)
   using $\mathbf{B}_{22}^{-1}$ and $b_{11,r}$ express the submatrices of $\mathbf{V}$:
     calculate $v_{11}$                                      (33)
     calculate $\mathbf{v}_{21}$ and $\mathbf{v}_{12}^{\mathrm{T}}$                        (34)-(35)
     calculate $\mathbf{V}_{22}$                                      (36)
   form $\mathbf{V}$ using $v_{11}, \mathbf{v}_{21}, \mathbf{v}_{12}^{\mathrm{T}}, \mathbf{v}_{22}$                  (37)
 end function
```

3.5 Remarks on the Proposed Calculations

3.5.1 Initialization

As it is presented in this section, $\mathbf{R}^{-1}(n+1)$ can be evaluated based on $\mathbf{R}^{-1}(n)$, therefore the initialization value of the recursion has to be defined prior to starting the algorithm. In this section two different methods are recommended to perform the initialization.

The first method is a simple solution through inversion of the auto-correlation matrix using the conventional procedures. However, this method requires all the samples to arrive in the observation window, thus the filter starts to follow the signal only after collecting the requested number of input samples.

The second method is the expansion of the inverse matrix according to the number of input samples. This solution is performed by applying the Levinson algorithm spreading the computational load across the incoming samples [12].

3.5.2 Zero Valued Input

During signal acquisition, it may occur that the analog-to-digital converter records zero values if the signal level is too small. In such a case, $\mathbf{X}(n)$ contains zero value which may lead to the rows of $\mathbf{X}$ not being linearly independent anymore. Since $\mathbf{R} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$, the rank of the auto-correlation matrix becomes lower than the number of columns. As a result, $\mathbf{R}$ becomes singular, i.e. $\det(\mathbf{R}) = 0$, therefore the inversion of the matrix is not possible. To avoid this issue, the incoming signal should be perturbed by an additive white noise – e.g. dither – whose level has to be low enough not to notably alter the input signal, but still large enough to enable the inversion of the auto-correlation matrix. Without this perturbation, if the signal falls to zero then the auto-correlation matrix is also zero, so it is not invertible and the algorithm is divergent. After the signal is present again, the diverged algorithm can not find a proper solution, thus it can not be convergent furthermore.

4. Complexity Analysis of the SWF

4.1 Updating the Matrices X, R and p

Since the SWF requires initial values, it is assumed in the following subsections that $\mathbf{X}(n)$, $\mathbf{R}(n)$ and $\mathbf{p}(n)$ are already available at the beginning of the calculations.

4.1.1 Calculate $\mathbf{X}(n+1)$

In order to update the matrix $\mathbf{X}$, Equation (14) has to be evaluated. After detailed investigation of the equation it can be concluded that only memory operations are required to calculate the updated $\mathbf{X}(n+1)$ matrix.

4.1.2 Calculate $\mathbf{R}(n+1)$ by Permutation

To create the new auto-correlation matrix, $\mathbf{R}(n+1)$, Equation (16) has to be evaluated. The complexity of each term for addition is calculated separately.

The first term can be calculated according to (17) by simple memory operations: shifting columns and rows.

The first factor in the second term, $\mathbf{P}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}(n)$ only contains memory operations. The remaining factor is a matrix generation, where only the first column differs from zero which is equal to $\mathbf{x}_1(n+1) - \mathbf{x}_K(n)$. The multiplication of these factors is not a complete matrix multiplication; because of the zero columns, only the first column has to be evaluated. Considering the simple matrix identity for transposing the product of matrices, the third term is already given by transposing the second one.

The subtraction in the forth term is already given, so only the multiplication is taken into account. Due to the special form of matrices, the zero columns and rows, this operation can be reduced to a scalar multiplication of two vectors of length N. As a result, this term is a zero matrix except the element in the upper left corner.

After these calculations, the terms have to be added. As the last three terms contain only one row or column vector or a single element, and an auto-correlation matrix is symmetric, only a partial matrix addition is required. The summarized computational requirements of updating $\mathbf{R}$ by permutation are presented in Tab. 1.

4.1.3 Calculate $\mathbf{p}(n+1)$ by Permutation

To create the new cross-correlation matrix, Equation (23) has to be evaluated. After the memory operations of the permutation, a $K \times N$ matrix is multiplied by an N element long column vector.

The first factor of the second term, $\mathbf{P}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}(n)$ is already discussed in Sec. 4.1.2. The remaining factor is a column vector generation, where only the last element differs from zero, which is $d(n) - d(n+N)$. The multiplication by this vector takes the last column from the matrix to the result.

The first factor of the third term is a matrix generation, where only the first row ($\mathbf{x}_1(n+1) - \mathbf{x}_K(n)$) differs from zero, which is already calculated when updating $\mathbf{R}$ (Sec. 4.1.2). This matrix is multiplied by a permuted column vector, which operation can be reduced to a scalar multiplication due to the zero rows.

The subtractions in the forth term are already calculated, so only the multiplication is taken into account. The factors of the product are almost empty, only the first row of the matrix and last element of the column vector are not zero. Due to the structure of the factors, only a single multiplication is required.

Summarizing these computations, the terms have to be added considering that the last two terms contain a single element. The summarized computational requirements for updating $\mathbf{p}$ by permutation are presented in Tab. 2.

4.1.4 Calculate $\mathbf{R}(n+1)$ by Dyadic Product

According to (21), two matrices of size $K \times K$ are generated from $\mathbf{x}^1$ and $\mathbf{x}^N$. As the multiplication of two vectors $\mathbf{z}^{\mathrm{T}}\mathbf{z}$ results in a symmetric matrix, only the upper triangular parts have to be calculated, thus only $\left(K^2 + K\right)/2$ multiplications are required instead of K^2. These considerations can be applied for the additions as well.

Operations	$\times$	$+$
$\mathbf{P}^T\mathbf{X}^T(n)\mathbf{X}(n)\mathbf{P}$	0	0
$\mathbf{P}^T\mathbf{X}^T(n)\cdot$ $\cdot(\mathbf{x}_1(n+1)-\mathbf{x}_K(n))\,\mathbf{e}_1^T$	NK	$(N-1)K+N$
$\mathbf{e}_1(\mathbf{x}_1(n+1)-\mathbf{x}_K(n))^T\cdot$ $\cdot\,\mathbf{X}(n)\mathbf{P}$	0	0
$\mathbf{e}_1(\mathbf{x}_1(n+1)-\mathbf{x}_K(n))^T\cdot$ $\cdot(\mathbf{x}_1(n+1)-\mathbf{x}_K(n))\,\mathbf{e}_1^T$	N	$N-1$
Summing terms	0	$K+1$
In total:	$NK+N$	$NK+2N$

Tab. 1. Computational requirements to update **R** by permutation (16).

Operations	$\times$	$+$
$\mathbf{P}^T\mathbf{X}^T(n)\mathbf{P}\mathbf{d}(n)$	NK	$(N-1)K$
$\mathbf{P}^T\mathbf{X}^T(n)\cdot$ $\cdot\,\mathbf{e}_N(d(n)-d(n+N))$	K	1
$\mathbf{e}_1(\mathbf{x}_1(n+1)-\mathbf{x}_K(n))^T\cdot$ $\cdot\,\mathbf{P}\mathbf{d}(n)$	N	$N-1$
$\mathbf{e}_1(\mathbf{x}_1(n+1)-\mathbf{x}_K(n))^T\cdot$ $\cdot\,\mathbf{e}_N(d(n)-d(n+N))$	1	0
Summing terms	0	$N+2$
In total:	$NK+N+$ $+K+1$	$NK+2N-$ $-K+2$

Tab. 2. Computational requirements to update **p** by permutation (23).

	Permutation	Dyadic product
$\mathbf{R}(n+1)$	$NK+2N$	K^2+K
$\mathbf{p}(n+1)$	$NK+2N-K+2$	$2K$

Tab. 3. Number of additions to calculate **p** and **R**.

	Permutation	Dyadic product
$\mathbf{R}(n+1)$	$NK+N$	K^2+K
$\mathbf{p}(n+1)$	$NK+N+K+1$	$2K$

Tab. 4. Number of multiplications to calculate **p** and **R**.

4.1.5 Calculate $\mathbf{p}(n+1)$ by Dyadic Product

According to (26), two column vectors of length K are generated from $\mathbf{x}^1$ and $\mathbf{x}^N$. These vectors multiplied by $d(n)$ and $d(N)$ and finally, these three vectors are summed.

4.1.6 Comparison of Permutation- and Dyadic Product-Based Methods

The computational complexities of $\mathbf{R}(n+1)$ and $\mathbf{p}(n+1)$ are summarized and presented in Tab. 3 and Tab. 4. To choose the appropriate method, the computational requirements – in terms of additions and multiplications – have to be compared.

Considering the update procedure of the auto-correlation matrix, the following inequalities should be investigated based on the aforementioned tables:

$$NK+2N \le K^2+K, \tag{38}$$

$$NK+N \le K^2+K. \tag{39}$$

After rearranging (39), the following criterion can be formed: $N \le K$. Also, Equation (38) can be expressed as

$$\frac{N}{K} \le \frac{K+1}{K+2}. \tag{40}$$

The right side of (40) converges to 1 from below, therefore N has to be lower than K to fulfil the inequality. This is in correspondence with the result obtained from (39), thus the permutation method has lower computational requirements if $N \le K$; otherwise the dyadic calculation is recommended.

Similar inequalities can be considered during the upgrade of the cross-correlation vector based on the aforementioned tables:

$$NK+2N-K+2 \le 2K, \tag{41}$$

$$NK+N+K+1 \le 2K. \tag{42}$$

If these equations are rearranged, the following conditions are generated:

$$N \le 3-8/(K+2), \tag{43}$$

$$N \le (K-1)/(K+1). \tag{44}$$

However Equation (43) could be fulfilled in some degenerate cases (e.g. $N<3$), but the right side of (44) converges to 1 which leads to the following non-satisfiable criterion: $N<1$. This requirement can never be satisfied, thus $\mathbf{p}(n+1)$ should be calculated by dyadic products in all cases.

4.2 Complexity of the RSC4BI Algorithm

Since the presented four blocks inversion can be applied for general cases as long as $\mathbf{A}_{11} = \mathbf{B}_{22}$ is valid, therefore this case is investigated first. After that, the auto-correlation matrices are discussed which are symmetric, thus the computational requirements can be further reduced.

4.2.1 General Case

For the expression of the submatrices of $\mathbf{V}$, two temporary matrices need to be calculated: $\mathbf{B}_{11}^{-1}$ according to (31) and $b_{11,r}$ according to (32).

The second part of the subtraction in (31) is generated by multiplying two vectors. Before evaluating this multiplication, one of these vectors has to be modified by $u_{22}^{-1} = 1/u_{22}$ to minimize the number of operations performed.

According to the evaluation of $\mathbf{B}_{22}^{-1}$, $b_{11,r}$ is generated similarly. However, it is a single value, therefore only this scalar is generated from the components of $\mathbf{B}$. The number of required operations are independent of the calculation order of the vector multiplications. If $b_{11,r}$ is already given, then v_{11} can be expressed through a division.

Operations	$\times$	$+$	$/$
$\mathbf{B}_{11}^{-1}$ (31)	$K(K-1)$	$(K-1)^2$	1
$b_{11,r}$ (32)	$K(K-1)$	$K(K-2)+1$	0
v_{11} (33)	0	0	1
$\mathbf{v}_{21}$ (34)	$K(K-1)$	$(K-1)(K-2)$	0
$\mathbf{v}_{12}^{\mathrm{T}}$ (35)	$K(K-1)$	$(K-1)(K-2)$	0
$\mathbf{V}_{22}$ (36)	$(K-1)^2$	$(K-1)^2$	0
In total:	$5K^2-6K+1$	$5K^2-12K+7$	2

Tab. 5. Computational requirements to update $\mathbf{R}^{-1}$ (Sec. 4.2.1).

Operations	$\times$	$+$	$/$
$\mathbf{B}_{11}^{-1}$ (31)	$(K+2)(K-1)/2$	$K(K-1)/2$	1
$b_{11,r}$ (32)	$K(K-1)$	$K(K-2)+1$	0
v_{11} (33)	0	0	1
$\mathbf{v}_{21}$ (34)	$K-1$	0	0
$\mathbf{v}_{12}^{\mathrm{T}}$	0	0	0
$\mathbf{V}_{22}$ (36)	$K(K-1)/2$	$K(K-1)/2$	0
In total:	$2K^2-2$	$2K^2-3K+1$	2

Tab. 6. Computational requirements to update $\mathbf{R}^{-1}$ (Sec. 4.2.2).

Algorithm 2. The reduced RSC4BI algorithm.

```
INPUT: $\mathbf{B} \equiv \mathbf{R}(n+1)$, $\mathbf{U} \equiv \mathbf{R}^{-1}(n)$
OUTPUT: $\mathbf{V} \equiv \mathbf{R}^{-1}(n+1)$
1:  function RSC4BI_REDUCED($\mathbf{B}$,$\mathbf{U}$)
2:    create temporary variable ($\mathbf{w}$):
3:      calculate $\mathbf{B}_{22}^{-1} = \mathbf{U}_{11} - \mathbf{u}_{12}u_{22}^{-1}\mathbf{u}_{21}^{\mathrm{T}}$
4:      calculate $\mathbf{w} = \mathbf{B}_{22}^{-1}\mathbf{b}_{21}$
5:      calculate $b_{11,r} = b_{11} - \mathbf{b}_{12}^{\mathrm{T}}\mathbf{w}$
6:    express the submatrices of $\mathbf{V}$:
7:      calculate $v_{11} = 1/b_{11,r}$
8:      calculate $\mathbf{v}_{21} = -\mathbf{w}v_{11}$
9:      calculate $\mathbf{v}_{12}^{\mathrm{T}} = \mathbf{v}_{21}^{\mathrm{T}}$
10:     calculate $\mathbf{V}_{22} = \mathbf{B}_{22}^{-1} - \mathbf{v}_{21}\mathbf{w}^{\mathrm{T}}$
11:   form $\mathbf{V}$ using $v_{11}, \mathbf{v}_{21}, \mathbf{v}_{12}^{\mathrm{T}}, \mathbf{V}_{22}$
12: end function
```

After having these temporary matrices, the remaining components of $\mathbf{V}$ are calculated. The vector component, $\mathbf{v}_{21}$ is derived as Equation (34) shows: a column vector is generated by a multiplication, which result is normalized by $b_{11,r}^{-1} = v_{11}$. Now $\mathbf{v}_{21} \neq \mathbf{v}_{12}$, therefore $\mathbf{v}_{12}^{\mathrm{T}}$ has to be computed similarly to $\mathbf{v}_{21}$.

During the calculation of the last submatrix, $\mathbf{V}_{22}$, the order of the multiplications should be carefully selected. If the left to right order is applied, then the complexity increases up to K^3 because of the full matrix multiplication. Avoiding this issue – considering that the matrix multiplication is associative –, an outside to the inside approach should be applied as shown in (45) to decrease the complexity to K^2 (The proposed execution order of the operations are shown with circled numbers):

$$\mathbf{B}_{22}^{-1} \overset{\textcircled{1}}{\cdot} \mathbf{b}_{21} \overset{\textcircled{2}}{\cdot} b_{11,r}^{-1} \overset{\textcircled{4}}{\cdot} \mathbf{b}_{12}^{\mathrm{T}} \overset{\textcircled{3}}{\cdot} \mathbf{B}_{22}^{-1}. \tag{45}$$

Taking these considerations into account, the number of required operations can be decreased if the temporary results of $\mathbf{v}_{12}$ and $\mathbf{v}_{21}$ are applied.

In total, this RSC4BI-inversion of $\mathbf{R}(n+1)$ achieves an $O\left(K^2\right)$ complexity by utilizing the special structures of matrices (see in Fig. 1). The summarized computational requirements are presented in Tab. 5.

4.2.2 Reduced Case

Using the fact that $\mathbf{R}$ is an auto-correlation matrix, therefore $\mathbf{R} = \mathbf{R}^{\mathrm{T}}$. Considering these symmetry properties, it can be satisfactory in certain cases to evaluate only the upper triangular part instead of the whole matrix. Also, in (31), it can be assumed that $\mathbf{u}_{12} = \mathbf{u}_{21}$, thus calculating only the upper triangular part is satisfactory.

Due to the symmetry, $(\mathbf{b}_{12}^{\mathrm{T}}\mathbf{B}_{22}^{-1})^{\mathrm{T}}$ is equal to $\mathbf{B}_{22}^{-1}\mathbf{b}_{21}$:

$$\left(\mathbf{b}_{12}^{\mathrm{T}}\mathbf{B}_{22}^{-1}\right)^{\mathrm{T}} = \left(\mathbf{B}_{22}^{-1}\right)^{\mathrm{T}}\left(\mathbf{b}_{12}^{\mathrm{T}}\right)^{\mathrm{T}} = \mathbf{B}_{22}^{-1}\mathbf{b}_{21}. \tag{46}$$

However Equation (32) could be optimized because the scalar product of the vectors contains repetitive terms in the sum, but considering (37), the calculation of the temporary variable $\mathbf{w} = \mathbf{B}_{22}^{-1}\mathbf{b}_{21}$ is useful, thus it is required for the calculations of $\mathbf{v}_{12}^{\mathrm{T}}$, $\mathbf{v}_{21}$ and $\mathbf{V}_{22}$. Therefore the scalar value $\mathbf{b}_{12}^{\mathrm{T}}\mathbf{B}_{22}^{-1}\mathbf{b}_{21}$ is calculated in two steps, keeping the order of the operations. This result is subtracted from b_{11} to obtain $b_{11,r} = v_{11}^{-1}$.

Using these temporary results, the remaining parts of $\mathbf{V}$ are easily evaluated: $\mathbf{v}_{12}$ (requiring $(K-1)$ multiplications) is equal to $\mathbf{v}_{21}$. Furthermore, the calculation of $\mathbf{B}_{22}^{-1}\mathbf{b}_{21}b_{11,r}^{-1}\mathbf{b}_{12}^{\mathrm{T}}\mathbf{B}_{22}^{-1}$ is reduced to the multiplication of the two already calculated vectors where only the upper triangular part has to be computed.

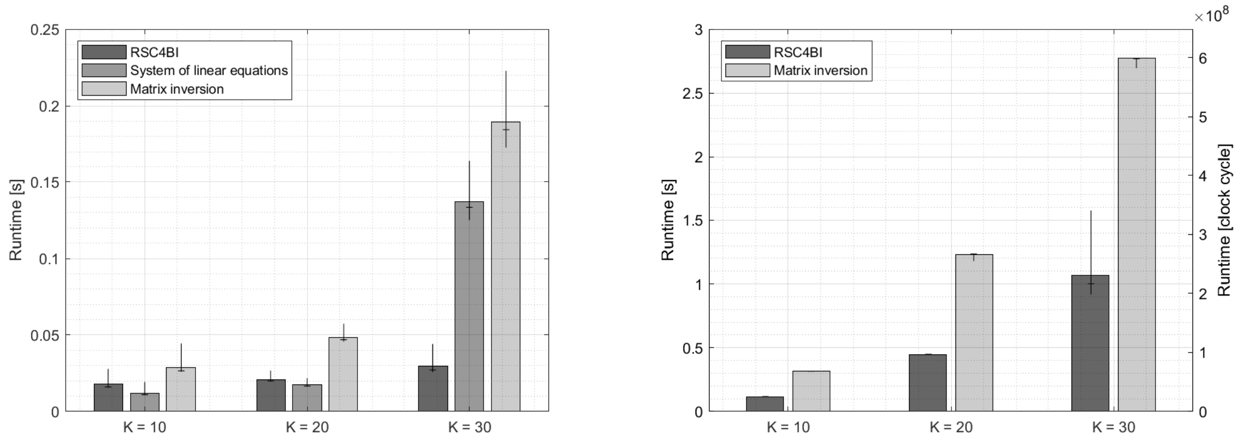
This reduced algorithm is presented in Alg. 2, while the computational requirements are summarized in Tab. 6. Comparing the results to Tab. 5, it can be stated that the complexity of the RSC4BI-algorithm can be reduced to less than half by taking into account the symmetry properties of the matrices.

5. Simulation Results

5.1 Parameters and the Applied Signals

The simulations are performed using MATLAB2019A. The observed filter is a highpass FIR-filter with 4 taps: $[0.2, -0.5, 0.7, 0.1]$. The applied sampling frequency is 10 kHz.

To investigate the SWF method which is presented in Sec. 3, these 0.1 s long probe signals are generated:



(a) Runtime on computer (applied clock frequency: 3.7 GHz). (b) Runtime on microcontroller (applied clock frequency: 216 MHz).

Fig. 2. Averaged runtime of different methods depending on the filter length.

- Gaussian white noise with $\mathcal{N}(0.1, 1)$,
- 200 Hz noisy sine wave with random phase between $[-\pi; \pi]$, and
- noisy multisine signal from 10 up to 500 Hz (frequency step: 10 Hz) with random phase between $[-\pi; \pi]$.

In case of a noisy signal the previously described Gaussian white noise is added ($\mathcal{N}(0.1, 1)$). The output signal of the observed filter is generated by convolution (1) to obtain a reference signal for the investigations.

To compare the RSC4BI to the conventional inversion methods, two further signals are also created by using different algorithms for matrix inversion. The first signal is generated by classic matrix inversion which is performed by Lower-Upper (LU) decomposition in Matlab. This computation may suffer from round-off errors, therefore to avoid this defect and returning to that the initial problem is the solution of a system of linear equations – according to (2) – Matlab recommends to use the so called "*backslash division*" (denoted by \) operator. This procedure generates the second signal solving the equations by a proper solver depending on the matrix properties. This algorithm is optimized to the computational time and it reduces the round-off errors as well.

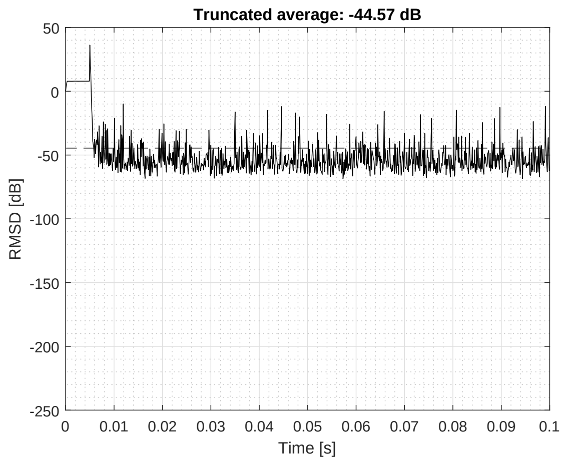
The observed system is estimated and followed by a WF. Its taps are computed from the WH equation – according to (10) – which is evaluated by using the reduced RSC4BI-algorithm (Alg. 2), the classic matrix inversion, and the system of linear equations method. During the simulations, it is assumed that there is no *a priori* knowledge regarding the observed filter, it is considered as black box with a given output signal. Therefore the length of the filter (the observed one and the WF) may differ, so their taps can not be compared directly, only their output signals.

5.2 Runtime of the Procedures

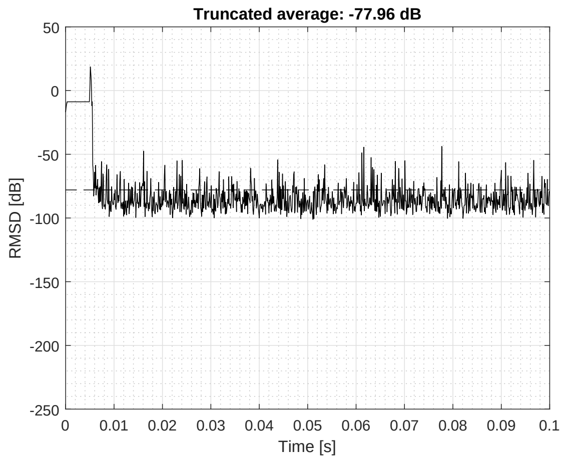
The simulations are performed on a desktop computer containing an AMD Ryzen 7 2700X processor with 32 GB RAM and on a STM32F722 microcontroller. Each method is evaluated 10,000 times as a single thread application and the runtime of every calculation is separately measured. The observation window includes N=20 samples, while the filter length is changed between the values K=(10, 20, 30), because the calculation complexity of $\mathbf{R}(n+1)$ is dependent only on K. The averages of the measured runtimes are presented in Fig. 2. The bar chart is completed by additional crosses to illustrate the medians and the 5 – 95% percentiles as well. The system of linear equations is a specialized algorithm in Matlab therefore only two methods were investigated on the microcontroller (see Fig. 2(b)), while Fig. 2(a) contains the results of three algorithms. As Section 4 states, the RSC4BI algorithm has lower computational requirements, if the number of the filter taps increase, then the runtime increases slower than in the case of classic matrix inversion or solving the system of linear equations. However, it should be noted that the optimized built-in solution of Matlab, using system of linear equations is the fastest if K is low, but it grows faster with increasing values of K than the proposed RSC4BI method.

5.3 Root Mean Square Deviation of SWF

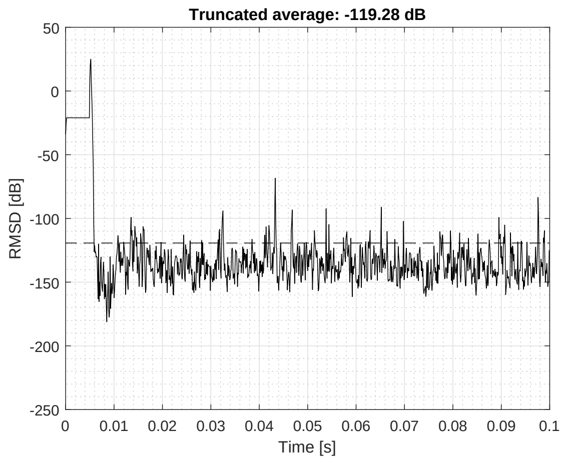
The first simulation is made by Monte Carlo method to investigate the performance of the algorithm by applying different probe signals. The SWF contains K=20 taps, while the observation window encloses N=20 samples. As a result, the Root Mean Square Deviation (RMSD) of the reference and the modeled output signals is calculated. 100,000 simulations are performed whose RMSD is evaluated as follows: $\mathrm{RMSD}(n) = \sqrt{\overline{\varepsilon_i^2(n)}}$. The delay elements and the observation window are empty at the beginning therefore there is a decaying transient, so the first 15 ms of the signal is cut off to obtain the average over the time (displayed by dashed line).



(a) Input: noisy multisine signal.



(b) Input: noisy sine wave.



(c) Input: white noise ($\mu = 0.1$).

Fig. 3. Root Mean Square Deviation of SWF algorithm over time with different probe signals.

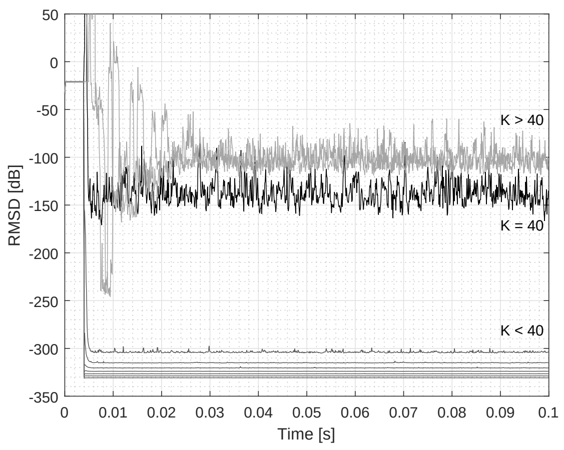


Fig. 4. Root Mean Square Deviation of SWF algorithm over time from $K = 5$ to $K = 50$ with $N = 40$.

These results are presented in Fig. 3, where the RMSD and its average are displayed in dB in case of the different probe signals. The deviation is the lowest if the input signal is only noise (Fig. 3(c)). However, it has to be noted that the signal level of noise is lower than the amplitude of the sine waves, which leads to lower RMSD if only noise is present.

5.4 Dependencies on N and K

As it is presented in Sec. 3, the SWF algorithm depends not only on K but on N as well, so the effect of these dependencies are investigated in Fig. 4 for the following cases: $N<K$, $N=K$ and $N>K$. These simulations are similar to the former ones (Sec. 5.3), but the input signal is only the noise ($\mathcal{N}(0.1, 1)$) and K is running from 5 up to 50 with a step size of 5. It can be seen that K should not be greater than N, because it leads to high RMSD. Using the same settings for K and N decreases the RMSD and gives a better result, but the $K<N$ case provides the best precision.

To understand these deviances in RMSD values, the initial problem has to be taken into account: the $\mathbf{y} = \mathbf{Xh}$ system of linear equations has to be solved (2). If $K<N$, then the system is overdetermined, therefore a best solution can be found in MMSE sense. The existence of the solution is not necessary to be investigated in this case, because the system ($\mathbf{h}(n)$) exists. In the case $K=N$, the exact solution of the system is created, whose RMSD is limited by the numeric precision of the applied procedures. If $K>N$, then there is an underdetermined system, where the solution is not a unique solution, but a whole subspace. A global optimal solution – in MMSE sense – may be found in this subspace, but in certain cases only a local optimum is found, especially if the numeric precision is worse, than in case of matrix inversion, when the conditional number gets higher.

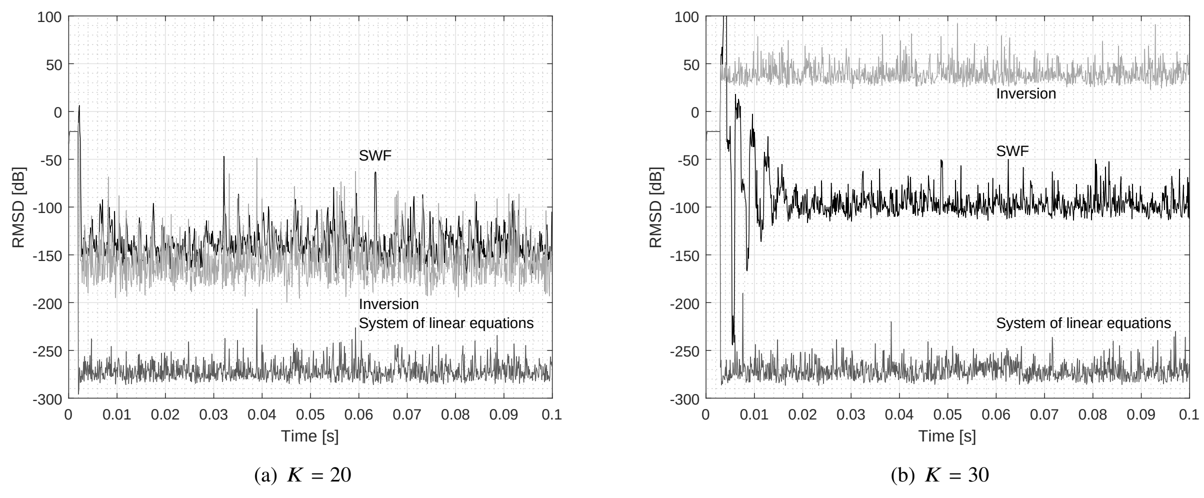


Fig. 5. Comparison of the different methods depending on filter length.

5.5 RMSD-Comparison of the Different Inversion Methods

The output of the RSC4BI algorithm is compared not only to the convolved reference signal, but to the two conventional methods as well. These results are presented in Fig. 5 for K= (20, 30) cases while N=20. If N=K is considered, the precision of the calculations depends on the algorithm, and as Fig. 5(a) shows, the RMSD of the system of linear equations is lower compared to the inversion and RSC4BI. Because of the underdeterminated system, if K=30 (Fig. 5(b)), the inversion can not find the proper solution (Sec. 5.4), while the system of linear equations gives unchanged precision and the RMSD of RSC4BI is higher, but still at an acceptable level.

6. Conclusion

In this paper an efficient step-by-step algorithm, the SWF is presented for the solution of the WH equation in a sliding window. Not only an algorithm for calculating the inverse of the auto-correlation matrix is given – the RSC4BI algorithm – but a recursive update procedure for the matrices **R** and **p** is presented as well. Additionally, two solutions for the initialization of the recursive algorithm were also suggested.

The inverse calculation is deduced to matrix multiplications and simple inversion of singular elements. This procedure is optimized for low complexity execution which makes the RSC4BI algorithm preferable for real-time applications and cost-effective processors. As the theoretical and simulated results show, the complexity of the algorithm is $O\left(K^2\right)$ because the inversion of the auto-correlation matrix is deduced to the inversion of a single element, meanwhile the matrix inversion generally requires $O\left(K^3\right)$ operations.

It was also shown, that the reduced complexity degrades the calculation precision, but it does not influence significantly the stability and the convergence. Therefore, the proposed algorithms can be considered as a practical solution suitable for low complexity hardware for real-time signal processing or to reduce the response time by distributing the computational load over time.

Acknowledgments

The authors are thankful for Prof. Thomas Zwick, as this work was partially carried out during the stay of Zsolt Kollár as a visiting researcher at the Karlsruhe Institute of Technology at the Institute of Radio Frequency Engineering and Electronics (IHE).

References

[1] WIENER, N. The linear filter for a single time series. Chapter in *Extrapolation, Interpolation, and Smoothing of Stationary Time Series: With Engineering Applications*. M.I.T. Press, 1949, p. 81–103. ISBN: 978-0-2622-3002-5

[2] WIDROW, B., STEARNS, S. D. The adaptive linear combiner. Chapter in *Adaptive Signal Processing*. Prentice-Hall, 1985, p. 15–30. ISBN: 978-0-1300-4029-9

[3] HUDEC, R., MARCHEVSKY, S. Adaptive order-statistic LMS filters. *Radioengineering*, 2001, vol. 10, no. 1, p. 20–24. ISSN: 1210-2512

[4] KIZILKAYA, A., UKTE, A., ELBI, M. D. Statistical multirate high-resolution signal reconstruction using the EMD-IT based denoising approach. *Radioengineering*, 2015, vol. 24, no. 1, p. 226–232. DOI: 10.13164/re.2015.0226

[5] HU, Y., SONG, M., DANG, X. D., et al. Interference mitigation for the GPS receiver utilizing the cyclic spectral analysis and RR-MSWF algorithm. *Radioengineering*, 2017, vol. 26, no. 3, p. 798–807. DOI: 10.13164/re.2017.0798

[6] FAN, X., TAN, Z., SONG, P., et al. A variable step-size CLMS algorithm and its analysis. *Radioengineering*, 2020, vol. 29, no. 1. p. 182–188. DOI: 10.13164/re.2020.0182

[7] PHAM, D. T. Quick solution of least square equations and inversion of block matrices of low displacement rank. *IEEE Transactions on Signal Processing*, 1991, vol. 39, no. 9, p. 2122–2124. DOI: 10.1109/78.134452

[8] ASIF, A., MOURA, J. M. F. Block matrices with L-block-banded inverse: Inversion algorithms. *IEEE Transactions on Signal Processing*, 2005, vol. 53, no. 2, p. 630–642. DOI: 10.1109/TSP.2004.840709

[9] DADIĆ, M., MOSTARAC, P., MALARIĆ, R. Wiener filtering for real-time DSP compensation of current transformers over a wide frequency range. *IEEE Transactions on Instrumentation and Measurement*, 2017, vol. 66, no. 11, p. 3023–3031. DOI: 10.1109/TIM.2017.2717238

[10] CHEN, Y., RUAN, S., QI, T. An automotive application of real-time adaptive Wiener filter for non-stationary noise cancellation in a car environment. In *IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC 2012).* Hong Kong (China), 2012, p. 597–602. DOI: 10.1109/ICSPCC.2012.6335628

[11] ZHANG, B., GAO, W., QI, Z., et al. Inversion algorithm to calculate charge density on solid dielectric surface based on surface potential measurement. *IEEE Transactions on Instrumentation and Measurement*, 2017, vol. 66, no. 12, p. 3316–3326. DOI: 10.1109/TIM.2017.2730981

[12] MORETTIN, P. A. The Levinson algorithm and its applications in time series analysis. *International Statistical Review*, 1984, vol. 52, no. 1, p. 83–92. DOI: 10.2307/1403247

[13] DELSARTE, P., GENIN, Y. The split Levinson algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1986, vol. 34, no. 3, p. 470–478. DOI: 10.1109/TASSP.1986.1164830

[14] KRISHNA, H., WANG, Y. The split Levinson algorithm is weakly stable. *SIAM Journal on Numerical Analysis*, 1993, vol. 30, no. 5, p. 1498–1508. DOI: 10.1137/0730078

[15] BENESTY, J., GANSLER, T. Computation of the condition number of a nonsingular symmetric Toeplitz matrix with the Levinson-Durbin algorithm. *IEEE Transactions on Signal Processing*, 2006, vol. 54, no. 6, p. 2362–2364. DOI: 10.1109/TSP.2006.873494

[16] LI, J., ZAKHAROV, Y. V. Sliding window adaptive filter with diagonal loading for estimation of sparse UWA channels. In *OCEANS 2016 - Shanghai.* Shanghai (China), 2016, p. 1–5. DOI: 10.1109/OCEANSAP.2016.7485346

[17] ZAKHAROV, Y. V., NASCIMENTO, V. H. DCD-RLS adaptive filters with penalties for sparse identification. *IEEE Transactions on Signal Processing*, 2013, vol. 61, no. 12, p. 3198–3213. DOI: 10.1109/TSP.2013.2258340

[18] AYRES, F. The inverse of a matrix. Chapter in *Theory and Problems of Matrices*. McGraw-Hill, 1974, p. 55–63. ISBN: 9780070026568

About the Authors . . .

Alfred KRAKER was born in 1941. After receiving his M.Sc, he joined SIEMENS AG Österreich. Since 1970 he was responsible for leading the R&D-department in broadcast analog and digital audio (HW/SW), and later, he led the development of high frequency applications as well. Between 1980 and 1990, he was the Deputy Director of R&D of analogue/digital mixing consoles and sound switching used in radio and TV centers. From 1990, he was responsible for the department of Digital Signal Processing until his retirement in 2001. He published many papers in these areas and about 100 patents in analogue and digital signal processing. He was the Austrian chairman of “AUDIO engineering society”, and he is IEEE “Senior Member” and IEEE Signal Processing member.

Barna CSUKA was born in 1991. He received his M.Sc. from the Budapest University of Technology and Economics (BME) in 2016 where he is currently working on his Ph.D. He is an assistant lecturer in the Department of Measurement and Information Systems at BME. Since 2017, he is member of the MATLAB laboratory. His research interests include digital signal processing, measurement theory and digital data transmission.

Zsolt KOLLÁR was born in 1983. He received his diploma and Ph.D. degree in electric engineering from the Budapest University of Technology and Economics (BME) in 2008 and 2013, respectively. He is an associate professor in the Department of Measurement and Information Systems at BME. Since 2017, he is the head of the MATLAB laboratory. His research interests are digital signal processing, wireless communication and quantization issues.

Appendix A: Inverse of a Hypermatrix With 4 Submatrices

In this section the inverse of a matrix $\mathbf{M}$ with four submatrices is derived. The matrix $\mathbf{M}$ is given as

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22} \end{bmatrix}. \tag{A-1}$$

The inverse of the matrix $\mathbf{M}$ – denoted by matrix $\mathbf{K}$ – is given as:

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{bmatrix}. \tag{A-2}$$

Using the fact that $\mathbf{MK} = \mathbf{KM} = \mathbf{I}$, and assuming that $\mathbf{M}_{22}$ is nonsingular, the elements of $\mathbf{K}$ can be expressed by the elements of $\mathbf{M}$ as [18]:

$$\mathbf{K}_{11} = \mathbf{M}_{11,r}^{-1}, \tag{A-3}$$

$$\mathbf{K}_{12} = -\mathbf{M}_{11,r}^{-1}\mathbf{M}_{12}\mathbf{M}_{22}^{-1}, \tag{A-4}$$

$$\mathbf{K}_{21} = -\mathbf{M}_{22}^{-1}\mathbf{M}_{21}\mathbf{M}_{11,r}^{-1}, \tag{A-5}$$

$$\mathbf{K}_{22} = \mathbf{M}_{22}^{-1} + \mathbf{M}_{22}^{-1}\mathbf{M}_{21}\mathbf{M}_{11,r}^{-1}\mathbf{M}_{12}\mathbf{M}_{22}^{-1}. \tag{A-6}$$

In (A-3)–(A-6), the reduced submatrix $\mathbf{M}_{11,r}$ can be expressed as

$$\mathbf{M}_{11,r} = \mathbf{M}_{11} - \mathbf{M}_{12}\mathbf{M}_{22}^{-1}\mathbf{M}_{21}. \tag{A-7}$$