A ROUNDOFF ERROR ANALYSIS OF THE OJA'S SUBSPACE RULE[1]

Tamás Szabó and Gábor Horváth

Technical University of Budapest
Department of Measurement and Instrument Engineering
Email: szabo@mmt.bme.hu, horvath@mmt.bme.hu

ABSTRACT

This paper deals with the effects of finite precision data representation and arithmetics in principal component analysis (PCA) networks. PCA or Karhunen Loéve Transform (KLT) is a statistical method that determines an optimal linear transformation of input vectors of a stationary stochastic process. The PCA networks are single layer linear neural networks that use some versions of Oja's learning rule. The paper concentrates on the errors which will arise during learning if fixed point data representation and arithmetics are used. It gives analytical results based on the additive noise model of quantization. In the analysis all three components of the finite precision effects are considered: (i) the error due to the input data quantization, (ii) the error caused be finite precision representation of the weights of the network, and (iii) the effects of the finite precision arithmetics. The results can be used directly to determine the required word-lengths for special hardware implementation of the neural net.

1. INTRODUCTION

In measurement systems there is a frequent need to represent signals or multidimensional measurement data in an efficient way. In these cases the goal is to transform the original multidimensional data onto a much fewer dimensional space without any error or with as few information losses as possible. Principal Component Analysis or Karhunen Loéve Transformation (KLT) is an optimal linear way of this transformation. KLT projects the N-dimensional input data onto an M-dimensional subspace, where usually $M<<N$, in such a way that the mean square error of the approximation is minimal. The dimension reduction is reached by transforming the correlated input data into statistically independent or at least uncorrelated components. The difficulty of this transformation is that the transformation matrix depends on the data to be transformed. The transformation matrix is formed from the eigenvectors of the autocorrelation matrix of the multidimensional data, so the classical way of KLT is a rather computation-intensive process. This high computational burden is the main obstacle of using KLT in real-time embedded measuring systems. Recently some linear neural architectures - the PCA networks - and some unsupervised learning rules were proposed for solving the KLT problem without the previous computation of the autocorrelation matrix [1]-[3]. These learning rules are based on Oja's subspace rule [1]. The PCA neural networks can determine both the transformation vectors - the eigenvectors - and the KLT of the signal/data to be analyzed in real-time if the networks are implemented in parallel hardware.

Neural nets can be implemented on analog or digital hardware. In analog realizations it is quite easy to implement multipliers and adders, but the modification of the weight values, the on-line learning is rather hard to implement. Further, there may be problems with the accuracy and long-term stability of the weights. In digital realizations not the stability and the realization of the on-line learning, but the efficient implementation of the high-speed multipliers makes most of the difficulties. Because of technological limitations for fast and efficient parallel realizations fixed point data representations and computation have to be used. Unfortunately, in this case the questions must be answered: how much precision is required to represent data and weights, and how precise arithmetics must be used? We address these questions in this paper.

The first neural network to solve the principal component problem was proposed by Oja in 1982. Since that time a whole family of PCA networks was developed (e. g. [3] and [4]), which use very similar learning algorithms. All of them are single-layer linear neural nets with fixed number of neurons using unsupervised learning, and all of them can be regarded as some extensions of the Oja's single-neuron network which determines only the first principal component. In this paper we will deal only with the single-output Oja's network. However, because of the common root of all these networks the results of this analysis can be easily applied to the other networks of the whole family.

The Oja's network is a simple linear combiner where the output is the dot product of the input data vector ($\boldsymbol{x}$), and a proper weight vector ($\boldsymbol{w}$): $y = \boldsymbol{w}^T \boldsymbol{x}$. The required transform can be achieved if the first eigenvector of the autocorrelation matrix of the input process is used as the weight vector of the network. This vector is formed as a result of a convergent learning process, where the learning algorithm is a normalized version of the Hebbian learning rule:

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \mu(k) y(k) [\boldsymbol{x}(k) - \boldsymbol{w}(k) y(k)] \qquad (1)$$

Here $\boldsymbol{x}(k)$ is the k-th sample of the N-dimensional input data sequence that has been centered to zero mean, $\boldsymbol{w}(k)$ is the weight vector at the k-th learning step, $y(k)$ is the corresponding output and $\mu(k)$ is the learning rate.

Although this learning rule is a little-bit more complicated than the LMS rule, we can find some similarities between the two algorithms. This similarity can be found not only in the

[1]This work was supported by the Hungarian Fund for Scientific Research (OTKA) under contract T 021003.

algorithms but in the error analysis. So in our analysis we can follow the basis steps used in [7]. Unfortunately the consequence of the higher complexity of the Oja's rule is that the analytical expressions for the PCA networks will be more complex, which can be compensated only with the neglecting of the less important terms as you will see later.

2. THE ROUNDOFF ERROR MODEL

In this section we take into account the sources of the errors caused by finite precision computation, their effects to the algorithm and show the approach which is used in the error analysis.

Three different sources of errors caused by fixed point computation can be distinguished . These are the followings:

- the error due to quantization of the input signals,
- the effects due to the quantization of the weights of the neural network,
- the error at the output of the network due to the application of finite precision arithmetics (this means re-quantizations after additions, multiplications, etc.)

In analyzing the effects of finite word-length data representation and arithmetics the most promising way is to use the additive noise model of quantization. It is a statistical approach where the main points are as follows [5]:

The effect of quantization is modeled by an additive noise, so a quantized signal can be represented as the sum of the original signal and a quantization error (noise), $e(k)$, where $e(k)$

- is a stationary uniformly distributed white noise process;
- is independent of the signal.

If q is the quantization step, then $e(k)$ is a zero mean process with $\sigma^2 = q^2/12$ variance. Although this simple model is true exactly only if rather strong conditions are met [5], in many practical cases - especially when the data to be quantized are Gaussian - this statistical model is true with high accuracy.

In the sequel we assume that:

A1. the input signal of the PCA network is a multi-dimensional (quasi-)stationary Gaussian process with zero mean, where the components of the multidimensional signal are strongly correlated,

A2. the input sequences have been properly scaled, so that their values lie in [-1;1],

A3. at a given iteration step the weight vector $\boldsymbol{w}(k)$ and the input vector $\boldsymbol{x}(k)$ are uncorrelated; some correlation can be found only between the weight vector, $\boldsymbol{w}(k)$ and the previous input data vectors $\boldsymbol{x}(k\text{-}1)$, $\boldsymbol{x}(k\text{-}2)$, etc.

A4. we are in a point of the iterative process where the algorithm has almost already converged, i. e. we are close enough to the final steady state. In this case during the further training steps the weight vector does not change significantly, so it can be considered as a constant rather than a probability variable. Moreover, $E\{\boldsymbol{w}\boldsymbol{w}^T\}$ is one of the diads of the input covariance matrix $\boldsymbol{R}$.

We use B_d+1-bit numbers for the input and output data representation, B_c+1-bit numbers for the weight values and the same representation is used for all the partial results of the learning algorithm. Further we assume that:

A5. rounding is used in the quantization, thus the quantization error associated with a B_d+1-bit number has variance $\sigma_d^2 = 2^{-2B_d}/12$, and the quantization error associated with a B_c+1-bit number has variance $\sigma_c^2 = 2^{-2B_c}/12$.

A6. that during additions no overflow occurs, hence, additions do not introduce any errors,

A7. the only errors due to the finite precision arithmetics are associated with the multiplications. At the calculation of the inner products errors will arise after the product is quantized. Furthermore, the inner product can be computed in two different ways: if all partial multiplications of an $\boldsymbol{a}^T\boldsymbol{b}$ are computed without quantization and only the final result after summation of the partial products are quantized in B_d bits, then the variance of the additive noise is σ_d^2, but if all the N scalar products in $\boldsymbol{a}^T\boldsymbol{b}$ are quantized individually before summation, the variance of the quantization error is $N\sigma_d^2$. (N is the dimension number of the vectors).

Applying the statistical model of quantization we can rewrite the basic equations of the Oja's network:

The output of the network will be:

$$\hat{y}(k) = [\boldsymbol{w}_q^T(k)\boldsymbol{x}_q(k)]_q \tag{2}$$

where

$$\boldsymbol{x}_q(k) = \boldsymbol{x}(k) + \boldsymbol{e}_x(k) \tag{3}$$

is the quantized input vector and $\boldsymbol{e}_x(k)$ is the corresponding zero mean quantization noise with variance σ_d^2. Similarly the actual weight vector at time k can be written as

$$\hat{\boldsymbol{w}}(k) = \boldsymbol{w}(k) + \boldsymbol{\rho}(k), \tag{4}$$

where $\boldsymbol{\rho}(k)$ is the noise caused by the finite precision arithmetics and the finite word-length representation of the weight vector.

If we neglect the higher order error terms, (2) can be rewritten as:

$$\hat{y}(k) = \boldsymbol{w}^T(k)\boldsymbol{x}(k) + \boldsymbol{\rho}^T(k)\boldsymbol{x}(k) + \boldsymbol{w}^T(k)\boldsymbol{e}_x(k) + \eta(k). \tag{5}$$

Here $\eta(k)$ is the error of the fixed point arithmetics. It is white noise and independent of the signals and the rest of the error sequences. It has zero mean and σ_d^2, or $N\sigma_d^2$ variance according to the computation of the inner product (***A7***).

We are interested in the total mean square output error that is the mean square value of last three terms at the right side of (5):

$$\xi_{error} = E\left\{\boldsymbol{\rho}^T(k)\boldsymbol{x}(k)\right\}^2 + E\left\{\boldsymbol{w}^T(k)\boldsymbol{e}_x(k)\right\}^2 + E\{\eta(k)\}^2 \tag{6}$$

Since $\boldsymbol{e}_x(k)$ and $\eta(k)$ are independent of each other and are uncorrelated to the input signal $\boldsymbol{x}(k)$ and since they are white noises with zero means, further using (***A3***), the error term in (6) has zero mean for all k, which implies that the presence of the quantization error does not affect the learning characteristic of the algorithm.

The second term of (6) can be easily derived as:

$$E\left\{\boldsymbol{w}^T(k)\boldsymbol{e}_x(k)\right\}^2 = E\left\{\boldsymbol{w}^T(k)\boldsymbol{w}(k)\right\}\sigma_d^2 = \sigma_d^2 \tag{7}$$

To derive the first term of (6) in steady state, we follow the approach of [8] and [7]. This term can be written as:

$$E\left\{\boldsymbol{\rho}^T(k)\,\boldsymbol{x}(k)\right\}^2 = tr\left(E\left\{\boldsymbol{\rho}^T(k)\,\boldsymbol{\rho}^T(k)\right\}\boldsymbol{R}\right) \tag{8}$$

where $\boldsymbol{R} = E\{\boldsymbol{x}(k)\boldsymbol{x}^T(k)\}$. Thus, we have to determine $E\{\rho(k)\rho^T(k)\}$.

Similarly we can rewrite (1) as:

$$\hat{\boldsymbol{w}}(k+1) = \hat{\boldsymbol{w}}(k) + \left[\mu\hat{y}(k)\{\boldsymbol{x}_q(k) - [\hat{y}(k)\hat{\boldsymbol{w}}(k)]_q\}\right]_q \quad (9)$$

This requantization signal-flow can be followed in Fig. 1.

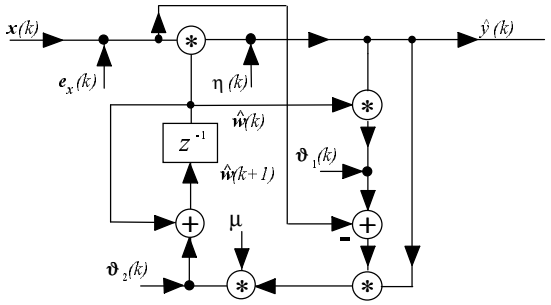

Fig. 1 The multiquantizer chain model of the PCA network

3. ANALYSIS OF THE ERROR MODEL

In the sequel we will not show the time indexes unless it disturbs understanding. Evolving the right side of (9) we get:

$$\begin{aligned} &\boldsymbol{w}(k+1) + \rho(k+1) = \boldsymbol{w} + \rho + \mu\{y + \boldsymbol{w}^T\boldsymbol{e}_x + \rho^T\boldsymbol{x} + \eta\} * \\ &\{\boldsymbol{x} + \boldsymbol{e}_x - (y + \boldsymbol{w}^T\boldsymbol{e}_x + \rho^T\boldsymbol{x} + \eta)(\boldsymbol{w} + \rho) - \vartheta_1\} + \vartheta_2 \end{aligned} \quad (10)$$

where ϑ_1 and ϑ_2 are the errors of arithmetics, similar to $\eta(k)$. They are independent of the data and the other errors and each other further they have zero mean. For simplicity, we take into account the effects of the two successive multiplication (by μ and y) in one step by ϑ_2. Of course, its value depends on the way the scalar products are computed (see *A7*). μ is often a power of 2, which implies that the elements of the previous scalar product are shifted to the right ($\mu < 1$) and requantized to B_c+1 bits. In this case, ϑ_2 is a white noise vector whose components have zero means and σ_c^2 variances. By using (10) we will look for $\rho(k+1)$ in form of

$$\rho(k+1) = \boldsymbol{F}(k)\rho(k) + \boldsymbol{b}(k) \quad (11)$$

where

$$\boldsymbol{F} = \boldsymbol{I} + \mu\left(\boldsymbol{x}\boldsymbol{x}^T - 2\boldsymbol{x}^T\boldsymbol{w}\,\boldsymbol{w}\,\boldsymbol{x}^T - \boldsymbol{w}^T\boldsymbol{x}\,\boldsymbol{x}^T\boldsymbol{w}\,\boldsymbol{I}\right) \quad (12)$$

and

$$\boldsymbol{b} = \mu\left(y\boldsymbol{e}_x + \boldsymbol{e}_x^T\boldsymbol{w}\boldsymbol{x} - 2y\eta\,\boldsymbol{w} - y\vartheta_1 - 2y\boldsymbol{e}_x^T\boldsymbol{w}\boldsymbol{w} + \eta\,\boldsymbol{x}\right) + \vartheta_2 \quad (13)$$

At this point for the simplicity, we assume that the algorithm starts with $\boldsymbol{w}(0) = \boldsymbol{0}$ and, of course, $\rho(0) = \boldsymbol{0}$. By denoting $E\{\rho\,\rho^T\}$ by $\boldsymbol{P}$ and $E\{\boldsymbol{b}\boldsymbol{b}^T\}$ by $\boldsymbol{Q}$, we get

$$\boldsymbol{P}(k+1) = E\{\boldsymbol{F}^T(k)\boldsymbol{P}(k)\boldsymbol{F}(k)\} + \boldsymbol{Q}(k) \quad (14)$$

with initial condition $\boldsymbol{P}(0) = \boldsymbol{0}$.

Next we need to determine the steady state value of Eq. (14). We assume that it converges because the algorithm also converges. To do this we have to compute the two expectation values $E\{\boldsymbol{F}^T\boldsymbol{P}\boldsymbol{F}\}$ and $\boldsymbol{Q}$. Both expressions are pretty complex. They have terms multiplied by different power of μ. In practical cases the learning factor (μ) is small enough ($\mu << 1$). Moreover, this factor have to be set small enough to enable to converge the algorithm well, because the amplitude of the fluctuation of the weight vector around the ideal one in steady state is proportional to the value of the learning factor. Now we deal with only the case of constant learning factor. This is why the terms proportional by the 2nd power of μ can be neglected.

Using the assumption that we are close to the steady state (*A4*) $\boldsymbol{R}\boldsymbol{w} = \lambda\boldsymbol{w}$, where λ is the corresponding (maximal) eigenvalue. Denoting $E\{\boldsymbol{w}\boldsymbol{w}^T\} = \boldsymbol{S}$ and using that the vector $\rho(k+1)$ and $\boldsymbol{w}(k+1)$ depends only on data up to time k and it does not depend on $\boldsymbol{x}(k+1)$, (*A3*) we can get:

$$\begin{aligned} &\boldsymbol{P}(k+1) = (1 - 2\lambda\mu)\boldsymbol{P} + \mu\{\boldsymbol{P}(\boldsymbol{R} - 2\boldsymbol{R}\boldsymbol{S}) + (\boldsymbol{R} - 2\boldsymbol{S}\boldsymbol{R})\boldsymbol{P}\} + \\ &+ E\{\vartheta_2\vartheta_2^T\} \end{aligned} \quad (15)$$

In this case $\boldsymbol{Q}$ has a very simple form: the only quantization effect is the last one in the multiquantizer chain.

Now the matrix equation (15) can be rewritten as:

$$\boldsymbol{P}(k+1) = \boldsymbol{P} + \mu\{\boldsymbol{P}\boldsymbol{G} + \boldsymbol{G}\boldsymbol{P}\} + E\{\vartheta_2\vartheta_2^T\} \quad (16)$$

where $\boldsymbol{G} = \boldsymbol{R} - 2\lambda\boldsymbol{S} - \lambda\boldsymbol{I}$. In steady state from (16) we can get the matrix equation [6]:

$$\boldsymbol{P}\boldsymbol{G} + \boldsymbol{G}\boldsymbol{P} = -\frac{\boldsymbol{Q}}{\mu} \quad (17)$$

This equation can be solved if $\boldsymbol{G}$ has non zero eigenvalues only which is satisfied in this case, because of $\boldsymbol{R}$ is positive definite and $\boldsymbol{S}$ is its diad corresponding to its highest eigenvalue.

The solution of this matrix equation is:

$$(\tilde{P})_{ij} = \frac{-1}{\mu\,(H_i + H_j)} W^T Q W \quad (18)$$

where $(\tilde{P})_{ij}$ is the ijth element of $\boldsymbol{P}$ in steady state, $\boldsymbol{W}$ is a matrix composed of the orthonormal eigenvectors of $\boldsymbol{R}$ and

$$\boldsymbol{H} = \boldsymbol{W}\boldsymbol{G}\boldsymbol{W}^T = diag\langle -2\lambda,\ \lambda_2 - \lambda,\ \lambda_3 - \lambda, \ldots\ \lambda_N - \lambda, \rangle. \quad (19)$$

In this case $\boldsymbol{Q}$ is also diagonal $\boldsymbol{W}^T\boldsymbol{Q}\boldsymbol{W} = \boldsymbol{Q}$. The equations (8), (18) and (19) yield the final result:

$$E\{\rho^T(k)\boldsymbol{x}(k)\}^2 = tr(\tilde{\boldsymbol{P}}\boldsymbol{R}) = \sum_{l=1}^{N}\sum_{k=1}^{N}\frac{-1}{\mu\,(H_l + H_k)}Q_{lk}R_{kl} \quad (20)$$

Hence, the output mean square arithmetic error is equal to:

$$\xi_{arit} = (1+c)\sigma_d^2 + \sum_{l=1}^{N}\sum_{k=1}^{N}\frac{-1}{\mu\,(H_l + H_k)}\delta_{lk}\sigma_c^{*2}R_{kl} \quad (21)$$

where c is a constant (c=1 or c=N) depending on the way the vector product was computed (*A7*), δ_{lk} is the Kronecker-delta and σ_c^{*2} depends on the way the last two products were computed.

4. SIMULATION RESULTS

The main point of our analysis is the computation of the $E\{\rho^T(k)\boldsymbol{x}(k)\}^2 = tr(E\{\rho(k)\rho^T(k)\}\boldsymbol{R})$ component of (6). This component is directly determined by the steady state value of $\boldsymbol{P}$

that is $\tilde{\boldsymbol{P}}$. This is why we examine only this matrix in the following simulations.

In the simulation experiments four- and sixteen-dimensional uncorrelated Gaussian signal sequences were generated as test signals. From these signals we formed such multidimensional correlated signals, that the first three eigenvalues of their correlation matrix be significantly larger than all the other ones; their ratios of the eigenvalues ($\lambda_i / \lambda_{\min}$) were about 10, 5, 3, 1, ...1 for i=1, 2, 3 ...etc. The basis (the eigenvectors of the correlation matrix) was a randomly chosen orthogonal system. We made some trial runs for different μ and different word-length representations of the data and the weight vector. Of course, the speed of the convergence and the weight vector fluctuation around the optimal value in steady case depend on the value of μ. If μ is large then the convergence is fast, but the variance is also large. Depending on the value of μ there is an other effect of quantization: if μ is too small compared to the quantum size q_c, then the algorithm can stop before it converges completely; if some partial results in Eq. (1) are less in magnitude than q_c, then their quantized values are zero and the corresponding weight does not change. The error caused by this effect can be larger than the effect of the quantization noise, but we do not deal with this question here. Instead such word-lengths and μ were selected that this effect does not arise. The number of learning steps (m) was determined in such a way, that at a given μ it ensures to converge sufficiently if no quantization is applied (about 5-15000 steps according to μ). Computation without quantization means MATLAB-precision arithmetics. About ten times longer input sample sequence was pregenerated than it was required for the infinite precision algorithm to converge. We trained the fixed point network with m-length sequences randomly chosen from pregenerated samples. This learning procedure was repeated 100-times with the same initial conditions. After all trials the error of the weights in the last step was determined by $\boldsymbol{\rho}(m) = \hat{\boldsymbol{w}}(m) - \boldsymbol{w}(m)$. This error was the difference between the weights of the finite- and "infinite"-precision networks using the same input sequences. The steady state value of $\boldsymbol{P}$ was computed from these error vectors as:

$$\tilde{\boldsymbol{P}} = E\left\{ \boldsymbol{\rho}\ \boldsymbol{\rho}^T \right\} \tag{22}$$

Since $\boldsymbol{Q}$ is diagonal in (18), $\tilde{\boldsymbol{P}}$ should also be diagonal. However, the results of the simulation experiments show that the although the off-diagonal elements are much smaller then the diagonal ones they are not equal to zeros. This deviation comes from our assumptions and from the fact that the terms proportional to μ^2 in (14) were neglected. A further reason of this difference is estimating $\tilde{\boldsymbol{P}}$ only the results of 100 runs are used. This was done because of the high computational requirements of the trial runs.

In Fig 2. we show a diagonal element of $\tilde{\boldsymbol{P}}$ versus word-length of weights with $B_d = 8$ and some given μ for 4-dimensional case. Similar results can be obtained for further simulation experiments with different word-lenghts and input dimensions.

One can see that our estimator is too optimistic when μ is large and the quantization of the data are significantly coarser than the quantization of the weights.

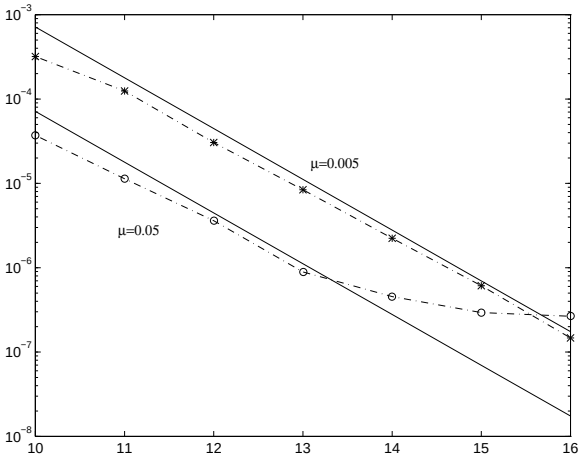

Fig. 2. A diagonal element of $\tilde{\boldsymbol{P}}$ versus the word-length of weights. Solid lines show the theoretical results.

This can be explained by the neglection of the terms proportional to μ^2 in (14) (σ_d^2 can be found in these terms). Nevertheless, the first term in (21) will be dominant, so even in this case we can estimate the output mean square error with quite high accuracy.

5. CONCLUSIONS

The roundoff error analysis of the Oja's subspace rule has been presented in this paper. We found that the accumulated error of the fixed-point algorithm is proportional to the reciprocal value of the learning rate parameter. Our model was verified by such simulation experiments where the word-lengths were selected according to practical considerations. The extension of the analysis and the analysis of the early termination of the algorithm are our further tasks.

REFERENCES

[1] Oja, E., Karhunen, J.: On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix, Helsinki University of Technology, Report TKK-F-A458, 1981

[2] Oja, E.: Neural networks, principal components and subspaces. *Int. J. Neural Syst.* 1, 61-68. 1989

[3] Sanger, T. D.: Optimal unsupervised learning in a single-layer linear feedforward neural network; *Neural Networks,* Vol. 2. 459-473, 1989

[4] Kung, S.Y., Diamantaras, C. I.: "A Neural Network Learning Algorithm for Adaptive Principal Component Extraction (APEX)" *Proc. of ICASSP* 1990. Vol. 2 pp. 861-864.

[5] Sripad, A. B., Snyder, D. L.: A necessary and sufficient condition for quantization error to be uniform and white *IEEE Trans on ASSP* Vol. 25. 422-448. 1977.

[6] Rózsa Pál, Linear algebra, *Tankönyvkiadó, Budapest 1991 (in Hungarian)*

[7] Caraicos, C., Liu; B.: A roundoff error analysis of the LMS adaptive algorithm; *IEEE Trans. on ASSP*, Vol. 32, 34-41. Febr. 1984

[8] J. E. Mazo: On the independence theory of equalizer convergence; *Bell Syst. Tech. J.*, Vol. 58, 963-993, May-June 1979