Fixed Order Implementation of Kernel RLS-DCD Adaptive Filters

Kiyoshi Nishikawa*, Yoshiki Ogawa*, Felix Albu†
*Department of Information and Communication Systems, Tokyo Metropolitan University
E-mail: knishikawa@m.ieice.org Tel: +81-42-585-8423
† Valahia University of Targoviste, Targoviste, Romania
E-mail: felix.albu@gmail.com

***Abstract*—In this paper, we propose an efficient structure of the kernel recursive least squares (KRLS) adaptive filters for implementing with low and fixed amount of computational complexity. The concept of kernel adaptive filters is derived by applying the kernel method to the linear adaptive filters for achieving the autonomous learning of non-linear environments. It is expected to provide a better noise reduction performance in non-linear environments than the conventional linear adaptive filters. One of the problems of the kernel adaptive filters is the required amount of calculation. Besides, they increase as the adaptation time advances as opposed to the linear case. In this paper, we propose an efficient implementation method of the KRLS dichotomous coordinate descent (DCD) adaptive algorithm. The proposed method enables us to implement at a constant amount of computation by fixing the order of the filter and the dictionary maintaining the fast rate of convergence of the KRLS-DCD algorithm. The effectiveness of the proposed method is confirmed by computer simulations.**

I. Introduction

In this paper, we propose an kernel recursive least squares (KRLS) dichotomous coordinate descent (DCD) adaptive algorithm and its efficient implementation method.

So far, the linear adaptive filters are widely researched and utilized in a lot of applications, such as acoustic echo canceler, channel equalization, system identification[1], [2]. The least mean square (LMS) and the recursive least squares (RLS) algorithms are well known learning algorithms for linear adaptive filters. It is known that they have complementary characteristics, i.e., the LMS algorithm can be implemented with lower computational cost, but the rate of convergence is relatively slow, and on the other hand, the RLS algorithm can achieve a faster rate of convergence, but at the expense of computational cost. Many modified algorithms of these algorithms are proposed so far. One of them is the RLS-DCD algorithm[3]. The algorithm achieves almost same rate of convergence as the RLS with comparative amount of calculation with the LMS and its variant algorithms.

Recently, the concept of the kernel adaptive filters is proposed by applying the kernel method to the linear adaptive filters[4]. The kernel adaptive filters enable us the autonomous learning of the non-linear systems. Several algorithms for kernel adaptive filters are proposed so far, e.g., the kernel LMS[5], kernel normalized LMS[6], kernel proportionate NLMS[7], kernel RLS[8], etc. These algorithms have the similar characteristics to the counterparts of the linear ones. The kernel RLS, for example, provides a faster rate of convergence but requires a larger amount of calculation compared with the kernel LMS algorithm. The kernel algorithms store the past input signal vectors for adaptation, and the stored vectors constitute a dictionary for estimating the unknown environments.

A special feature of the kernel algorithms that does not exists in the linear ones is that the order of the filter, and the dictionary will increase as the adaptation time advances, so that the required amount of calculation increases. The feature makes it impractical to implement the kernel RLS algorithm for on-line applications. To reduce the amount of calculation, several sparsification methods of the input signal are proposed[4], [6], [9]. Although we can decrease the rate of increment of the dictionary order, there are no limit on the maximum order. This may pose a problem in implementation environments with restriction on memory sizes, for example. Therefore, in [10], the method for fixing the order of dictionary is proposed. By predefining the maximum allowable order, it enables us to limit the amount of calculation required.

In this paper, we proposed an implementation method of the kernel RLS-DCD algorithm with the fixed order dictionary. In [11], we proposed the kernel RLS-DCD algorithm by applying the kernel method to the RLS-DCD algorithm. We showed that it can be implemented with $O(M)$ calculations where M shows the filter order, and at the same time, it maintains a comparative rate of convergence as that of the kernel RLS.

For the kenerl RLS-DCD, it is not effective to directly apply the method of [10] due to its criterion to select the element of dictionary to delete to maintain the dictionary order. Therefore, in this paper, we propose a new criterion suited for the kernel RLS-DCD to select an element for deletion. We show that, by using the proposed method, we can limit the maximum amount of computation with slightly sacrificing the rate of convergence. Through the computer simulations, we confirm the effectiveness of the proposed method.

II. Preparation

In this section, we briefly review the linear RLS-DCD algorithm[3], and kernel adaptive filters[4].

In the following, a matrix or a vector is indicated by bold letters, e.g., $\mathbf{R}$, or $\mathbf{r}$; the index of a matrix or a vector by lower letters as $R_{i,j}$ or r_i; and a column vector of a matrix by

superscript as $\mathbf{R}^{(p)}$. The transpose of a matrix $\mathbf{R}$ is indicated as $\mathbf{R}^T$, and variables at time n is expressed as $\mathbf{R}(n)$.

A. RLS-DCD algorithm

The recursive least squares (RLS) algorithm is one of the standard algorithms for linear adaptive filters. The RLS algorithm enables us to obtain the optimum filter which minimizes the sum of least squares $\sum_n e^2(n)$ where $e(n)$ is the error signal. The optimum filter is obtained by solving the normal equations which is defined as

$$\mathbf{R}(n)\mathbf{w}(n) = \boldsymbol{\beta}(n) \tag{1}$$

where $\mathbf{w}(n)$, $\mathbf{R}(n)$, and $\boldsymbol{\beta}(n)$ are the filter coefficients, the autocorrelation matrix of the input signal of the filter, and the cross-correlation vector between the input and the output of the filter respectively. To directly obtain the optimum solution $\mathbf{w}_o$, we must calculate the inverse of the matrix $\mathbf{R}$. Instead, the RLS algorithm uses the matrix inversion lemma to solve the equation (1). Although the RLS enables us to obtain the optimum solution without calculating the matrix inversion, it requires $O(M^2)$ multiplications, where M show the filter order, and which is a magnitude larger that that of LMS-type algorithms. For reducing the computational load, several methods have been proposed, and the RLS-DCD algorithm is one of those methods.

We describe briefly the RLS-DCD algorithm. Let us express the approximate solution at $n-1$ as $\hat{\mathbf{w}}(n-1)$. Then, we define the residual vector at time $n-1$ as

$$\mathbf{r}(n-1) = \boldsymbol{\beta}(n-1) - \mathbf{R}(n-1)\hat{\mathbf{w}}(n-1) \tag{2}$$

and the following variables

$$\begin{cases} \Delta\mathbf{R}(n) = \mathbf{R}(n) - \mathbf{R}(n-1) \\ \Delta\boldsymbol{\beta}(n) = \boldsymbol{\beta}(n) - \boldsymbol{\beta}(n-1) \\ \Delta\mathbf{w}(n) = \mathbf{w}(n) - \hat{\mathbf{w}}(n-1) \end{cases} \tag{3}$$

Using the equation (3), we rewrite the equation (1) as

$$\mathbf{R}(n)\left[\hat{\mathbf{w}}(n-1) + \Delta\mathbf{w}(n)\right] = \beta(n) \tag{4}$$

Then, we obtain the equation

$$\begin{aligned} \mathbf{R}(n)\Delta\mathbf{w}(n) &= \boldsymbol{\beta}(n) - \mathbf{R}(n)\hat{\mathbf{w}}(n-1) \\ &= \boldsymbol{\beta}(n) - \mathbf{R}(n-1)\hat{\mathbf{w}}(n-1) - \Delta\mathbf{R}(n)\hat{\mathbf{w}}(n-1) \\ &= \mathbf{r}(n-1) + \Delta\boldsymbol{\beta}(n) - \Delta\mathbf{R}(n)\hat{\mathbf{w}}(n-1). \end{aligned} \tag{5}$$

By introducing a new variable $\boldsymbol{\beta}_0(n)$ defined as

$$\boldsymbol{\beta}_0(n) = \mathbf{r}(n-1) + \Delta\boldsymbol{\beta}(n) - \Delta\mathbf{R}(n)\hat{\mathbf{w}}(n-1) \tag{6}$$

and substituting into (5), the auxiliary system of equations are obtained as

$$\mathbf{R}(n)\Delta\mathbf{w}(n) = \boldsymbol{\beta}_0(n) \tag{7}$$

By solving these equations, $\Delta\mathbf{w}(n)$ could be obtained and $\hat{\mathbf{w}}(n)$, an approximate solution of (1), is calculated by the equation

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \Delta\hat{\mathbf{w}}(n). \tag{8}$$

In Alg.1, we show the algorithm for solving the normal equations recursively. In that, $\boldsymbol{\Pi}$ is given by $\boldsymbol{\Pi} = \eta\mathbf{I}_N$ where η is a small value, and $\mathbf{I}$ is an identity matrix of size $N \times N$.

Several algorithms can be used to solve the auxiliary equations (12). One of those algorithms is the dichotomous coordinate descent (DCD) algorithm that enables us to implement the RLS-type algorithm with the computation complexity of $O(M)$. The DCD algorithm is based on the binary expression of the elements of the solution vector. The DCD iteratively updates from the most significant bit of its elements and proceeds to less significant bits. The parameters of the DCD algorithm includes the number of bits M_b, the amplitude range $[-H, H]$, and the maximum number of *successful* iterations N_u. We can control the amount of calculation by the value of N_u, namely, by assigning a larger value to N_u, a smaller error can be achieved, at the same time, the number of iteration, and hence, the amount of calculation increases. In Alg. 2, we show the DCD algorithm. In the following, we restrict ourselves to use the DCD algorithm to solve the auxiliary normal equations because of its lower computational requirement.

B. Kernel adaptive filters

Next, we briefly summarize the kernel adaptive filters. The concept of kernel adaptive filter is derived by applying the kernel method to the conventional linear adaptive filters.

To apply the kernel method, the input signal $\mathbf{u}$ is mapped into a higher order characteristics space. We express this mapping as $\phi(\mathbf{u})$. Then, we try to expand and approximate the coefficient vector $\mathbf{w}(n)$ in terms of the $\phi(\mathbf{u})$ as $\mathbf{w}'(n)$:

$$\mathbf{w}'(n) = \alpha_1\phi(\mathbf{u}_0) + \cdots + \alpha_n\phi(\mathbf{u}_{n-1}) \tag{18}$$

where $\alpha_i, i = 0, \ldots, n$ are the weights to be determined. Using

Algorithm 1 Procedure for Solving Recursively the Normal Equations

Initialize: $\hat{\mathbf{w}}(-1) = 0, \quad \mathbf{r}(-1) = 0, \quad \mathbf{R}(-1) = \boldsymbol{\Pi}$
for $n = 0, 1, 2, \cdots$ **do**
  $\mathbf{R}^{(1)}(n) = \lambda\mathbf{R}^{(1)}(n-1) + \mathbf{x}(n)\mathbf{x}(n)^T$ (9)
  $e(n) = d(n) - \mathbf{x}(n)^{\mathrm{T}}\hat{\mathbf{w}}(n-1)$ (10)
  $\boldsymbol{\beta}_0(n) = \lambda\mathbf{r}(n-1) + e(n)\mathbf{x}(n)$ (11)
  $\mathbf{R}(n)\Delta\mathbf{w}(n) = \boldsymbol{\beta}_0(n) \Rightarrow \Delta\hat{\mathbf{w}}(n), \mathbf{r}(n)$ (12)
  $\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \Delta\hat{\mathbf{w}}(n)$ (13)
end for

Algorithm 2 DCD algorithm

Initialize: $\Delta\hat{\mathbf{w}} = 0, \quad \mathbf{r} = \beta_0, \quad \alpha = H/2, \quad m = 1$
for $k = 1, \cdots, N_u$ **do**
  $p = \arg\max_{n=1,\cdots,N} |r_n|$ (14)
  while $|r_p| \le (\alpha/2)R_{p,p}$ **do**
    $m = m + 1, \quad \alpha = \alpha/2$ (15)
    if $m > M_b$ **then** the algorithm stops
  end while
  $\Delta\hat{w}_p = \Delta\hat{w}_p + \mathrm{sign}(r_p)\alpha$ (16)
  $\mathbf{r} = \mathbf{r} - \mathrm{sign}(r_p)\alpha\mathbf{R}^{(p)}$ (17)
end for

the kernel trick, the output signal $y(n)$ is expressed as

$$\begin{aligned} y(n) &= \phi(\mathbf{x}_n)^{\mathrm{T}}\mathbf{w'}_n \\ &= \phi(\mathbf{x}_n)^{\mathrm{T}}(\alpha_1\phi(\mathbf{x}_1) + \alpha_2\phi(\mathbf{x}_2) + \cdots + \alpha_n\phi(\mathbf{x}_{n-1})) \\ &= \alpha_1\phi(\mathbf{x}_n)^{\mathrm{T}}\phi(\mathbf{x}_1) + \cdots + \alpha_n\phi(\mathbf{x}_n)^{\mathrm{T}}\phi(\mathbf{x}_{n-1}) \\ &= \alpha_1\kappa(\mathbf{x}_n, \mathbf{x}_1) + \alpha_2\kappa(\mathbf{x}_n, \mathbf{x}_2) + \cdots + \alpha_n\kappa(\mathbf{x}_n, \mathbf{x}_{n-1}) \end{aligned} \quad (19)$$

where $\kappa(\cdot,\cdot)$ shows the kernel function. By defining the vector $\mathbf{h}$ as

$$\mathbf{h} = [\kappa(\mathbf{x}_n, \mathbf{x}_1), \kappa(\mathbf{x}_n, \mathbf{x}_2), \cdots, \kappa(\mathbf{x}_n, \mathbf{x}_{n-1})]^{\mathrm{T}} \quad (20)$$

and substituting into (19), we have a simpler expression of $y(n)$

$$y(n) = \mathbf{h}^{\mathrm{T}}\boldsymbol{\alpha}. \quad (21)$$

This equation can be interpreted as the input-output relation of a filter $\boldsymbol{\alpha}$. Hence, in the kernel adaptive filtering, $\boldsymbol{\alpha}$ contains the filter coefficients to be adjusted and, for updating $\boldsymbol{\alpha}$, the linear adaptive algorithms can be used with slight modifications[4].

C. Sparsification of the dictionary

For implementing the kernel adaptive filters, the past input signal vectors $\mathbf{x}(n)$ are stored as a dictionary. As the time advances, the number of stored vectors and the order of the filter increase. Hence, the number of summation in (19) increases as n. This is the feature that is not exists in the linear adaptive filtering.

The increase of the filter order and the dictionary size results in the increase of the required amount of calculation at each time. Therefore, when we apply the kernel adaptive filter to applications, we need a sparsification of the input signal in order to maintain the affordable amount of computation. For sparsifying the input signal, several methods are proposed so far[4], [6], [9]. In this paper, we consider to use the one proposed in [6] due to its simple structure.

The input signal vectors are stored in a matrix, or the dictionary, of size $L \times M$ where M and L respectively show the length of the adaptive filter $\mathbf{w}$, and the number of signals stored. In this paper, we call L as the order of the dictionary.

In the spasification method of [6], the input signal at time n is compared with each element of $\mathbf{h}$ by the equation

$$\max|h_j(n)|_{j=1,\cdots,m} < \mu_0 \quad (22)$$

where μ_0 is the predefined threshold value. Only when this relation holds, $\mathbf{x}(n)$ will be added to the dictionary.

D. Fixed order implementation

Even if the sparsification method of the input signal is used, we cannot precisely estimate the required order of the dictionary in advance. Besides, if the environments are changed during the adaptation, a higher order shall be required. This might be a problem when implementing the kernel adaptive filters in the environments with a limited amount of memories such as embedded hardware.

To avoid this problem, a method is proposed for fixing the order of the filter and the dictionary for the kernel RLS[10]. In this method, the allowable maximum size of the dictionary $L_{\max}$ is set advance. When the order of the dictionary become larger than $L_{\max}$, the criterion for deletion $dis(\mathbf{D}_i)$ will be calculated. Then, the vector, an element of the dictionary, that gives the minimum $dis(\mathbf{D}_i)$ will be deleted. In Fig. 1, an example of the deletion of a vector is shown.

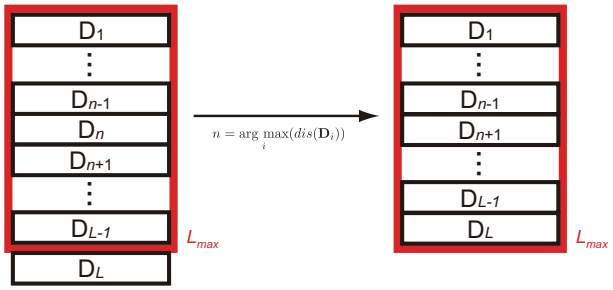

Fig. 1. Method for fixing the order of the dictionary. In this example, D_n is deleted and inserting D_L without changing the order of the dictionary.

In this method, the convergence characteristics of the adaptive filter will be affected by the selection of the maximum order of the dictionary $L_{\max}$ and the criterion for determining the element for delete. Therefore, we must carefully select them.

III. The Proposed Method

Here, we propose the kernel adaptive filter based on the RLS-DCD algorithm.

A. Order variation of the filter and the dictionary

As mentioned before, the order of the filter $\boldsymbol{\alpha}$, and also, that of characteristic space will increase as the time advances. Namely, the number of terms in the summation of (19) increases as n. Therefore, we cannot directly apply the linear RLS-DCD to the kernel adaptive filter.

In the proposed method, we extend the equation (9) of Alg.1 which is the update equation of $\mathbf{R}(n)$ as (23). Hence, the required amount of calculation for (9) is increased from $\mathrm{O}(M)$ to $\mathrm{O}(L^2)$.

$$\mathbf{R}(n) = \lambda\begin{bmatrix}\mathbf{R}(n-1) & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 1\end{bmatrix} + \mathbf{h}(n)\mathbf{h}(n)^{\mathrm{T}} \quad (23)$$

Similarly, we propose to extend the equations (10), (11), and (13) as

$$e(n) = d(n) - \mathbf{h}(n)^{\mathrm{T}}\begin{bmatrix}\boldsymbol{\alpha}(n-1) \\ 0\end{bmatrix} \quad (24)$$

$$\boldsymbol{\beta}_0 = \lambda\begin{bmatrix}\mathbf{r}(n-1) \\ 0\end{bmatrix} + e(n)\mathbf{h}(n) \quad (25)$$

$$\boldsymbol{\alpha}(n) = \begin{bmatrix}\boldsymbol{\alpha}(n-1) \\ 0\end{bmatrix} + \Delta\boldsymbol{\alpha}(n) \quad (26)$$

Note that the DCD algorithm shown in Alg.1 is independent of other calculations. This implies that it is not affected by the increment of the filter order, and hence, the algorithm can be used without modification in the propose method.

Algorithm 3 Proposed kernel-RLS-DCD algorithm

Initialize: $\mathbf{D}(0) = \{\mathbf{x}(n)\}, \boldsymbol{\alpha}(0) = \mathbf{r}(0) = 0, \mathbf{R}(0) = \boldsymbol{\Pi}$
for $n = 1, 2, 3, \cdots$ **do**
  $\mathbf{h}(n) = \kappa(\mathbf{x}(n), \mathbf{D}(n-1))$ (27)
  if $\mathbf{x}(n)$ will be added to the dictionary **then**
    $\mathbf{D}(n) = \mathbf{D}(n-1) \cup \{\mathbf{x}(n)\}$ (28)
    $\mathbf{h}(n) = \begin{bmatrix}\mathbf{h}(n)\\ 1\end{bmatrix}$ (29)
    $\mathbf{R}(n) = \lambda \begin{bmatrix}\mathbf{R}(n-1) & 0\\ 0 & 1\end{bmatrix} + \mathbf{h}(n)\mathbf{h}(n)^{\mathrm{T}}$ (30)
    $e(n) = d(n) - \mathbf{h}(n)^{\mathrm{T}} \begin{bmatrix}\boldsymbol{\alpha}(n-1)\\ 0\end{bmatrix}$ (31)
    $\boldsymbol{\beta}_0(n) = \lambda \begin{bmatrix}\mathbf{r}(n-1)\\ 0\end{bmatrix} + e(n)\mathbf{h}(n)$ (32)
    $(\Delta\boldsymbol{\alpha}(n), \mathbf{r}(n)) = \mathrm{DCD}(\mathbf{R}(n), \boldsymbol{\beta}_0(n))$ (33)
    $\boldsymbol{\alpha}(n) = \begin{bmatrix}\boldsymbol{\alpha}(n-1)\\ 0\end{bmatrix} + \Delta\boldsymbol{\alpha}(n)$ (34)
  else
    $\mathbf{D}(n) = \mathbf{D}(n-1)$ (35)
    $\mathbf{R}(n) = \lambda\mathbf{R}(n-1) + \mathbf{h}(n)\mathbf{h}(n)^{\mathrm{T}}$ (36)
    $e(n) = d(n) - \mathbf{h}(n)^{\mathrm{T}}\boldsymbol{\alpha}(n-1)$ (37)
    $\boldsymbol{\beta}_0(n) = \lambda\mathbf{r}(n-1) + e(n)\mathbf{h}(n)$ (38)
    $(\Delta\boldsymbol{\alpha}(n), \mathbf{r}(n)) = \mathrm{DCD}(\mathbf{R}(n), \boldsymbol{\beta}_0(n))$ (39)
    $\boldsymbol{\alpha}(n) = \boldsymbol{\alpha}(n-1) + \Delta\boldsymbol{\alpha}(n)$ (40)
  end if
end for

B. Kernel-RLS-DCD algorithm[11][12]

We show the proposed algorithm for updating the adaptive filter in Alg. 3. In this algorithm, $\mathbf{x}(n)$, $d(n)$, $e(n)$, $\mathbf{h}(n)$ and $\mathbf{D}(n)$ are respectively the input, the desired, the error signals, the mapped set of the input signal, and the dictionary. Also, $\boldsymbol{\alpha}(n)$ is the filter coefficient vector, and $\kappa(\cdot,\cdot)$ the kernel function. λ, and μ_0 are the forgetting factor and the threshold value respectively. Note that *DCD()* indicates that the DCD algorithm is used to solve the auxiliary normal equations.

C. Fixed order dictionary in the proposed method

Let us consider to apply the method for fixing the order of the dictionary, which is described in Sec. II-D, to the kernel RLS-DCD algorithm. In [10], the discarding criterion is calculated from the relation between the diagonal terms of the inverse of the Gram matrix and the filter coefficients. However, the computational cost required to calculate the inverse of the Gram matrix is very high and, in the RLS-DCD, the Gram matrix is not used for updating the coefficients. Therefore, it is not effective to calculate the inverse of the Gram matrix for the single purpose of calculating the discarding criterion. Instead, in this paper, we propose a new discarding criterion suited for the kernel RLS-DCD algorithm.

First, we notice that from the equation (19), the output signal y_n of the filter is expressed as

$$y_n = \sum_{i=1}^{L} \alpha_i \kappa(\mathbf{x}_n, \mathbf{D}_i). \tag{41}$$

On the other hand, the output signal $\breve{y}_n$ under the condition that the k-th vector $\mathbf{D}_k$ is deleted from the dictiory is expressed as

$$\breve{y}_n = \sum_{i=1}^{k-1} \alpha_i \kappa(\mathbf{x}_n, \mathbf{D}_i) + \sum_{i=k+1}^{L} \alpha_i \kappa(\mathbf{x}_n, \mathbf{D}_i). \tag{42}$$

Hence, the difference of these signals show the error due to the deletion of $\mathbf{D}_k$ and is given as

$$E_{n,k} = |y_n - \breve{y}_n| = |\alpha_k| \, \kappa(\mathbf{x}_n, \mathbf{D}_k). \tag{43}$$

By averaging the errors for all the vectors stored in the dictionary, we obtain

$$E_k = \frac{1}{L} \left| E_{1,k}, E_{2,k}, \ldots, E_{L,k} \right|. \tag{44}$$

In this paper, we propose to select k which minimizes E_k.

As a result, the proposed procedure to fix the order of the dictionary is shown in Alg. 4.

Algorithm 4 Updated algorithm of fixed order dictionary in the proposed method

if $L > L_{max}$ **then**
  $k = \arg\min_{1 \le m \le L} \frac{1}{L} \sum_{i=1}^{L} |\alpha_m| \, \kappa(\mathbf{D}_i, \mathbf{D}_m)$ (45)
  $\mathbf{D} = \mathbf{D} \setminus \{\mathbf{D}_k\}$ (46)
  Remove the kth row or column from $\mathbf{h}, \boldsymbol{\alpha}, \mathbf{R}, \mathbf{r}$
end if

IV. Simulation Results

Here, we show some results of computer simulations using the proposed method.

A. Comparison of the proposed and the conventional methods

First, we show the results of channel equalization of a multi-path Rayleigh fading channel[4].

Conditions of the simulations are follows. The length of signal was 1000, the number of path M is $M = 5$, the maximum Doppler frequency was $f_D = 100$ Hz, and the sampling rate was set as $T_s = 0.8$ μs. The parameters for the proposed algorithm were $\lambda = 0.955$, $H = 1$, $N_u = 8$, $M_b = 16$, and $\mu_0 = 0.8$. We compared the proposed, the kernel NLMS, and the kernel RLS algorithms. The Gaussian kernel was used as the kernel function for all the algorithms, and the kernel parameter a was set as $a = -0.1$. The order of dictionaries was not fixed in this simulation.

We show the comparison of the mean squared errors (MSEs) in Fig. 2. On the other hand, Fig. 3 shows the comparison of execution time of simulation. From Fig. 2, we see that the proposed method provides almost same convergence characteristics and the excess errors with the kernel RLS algorithm, and better performance than the kernel NLMS algorithm. Besides, from Fig. 3, it is shown that the execution time of the kernel RLS algorithm increases as the order of the dictionaries increases, and on the other hand, the proposed method maintains almost flat time as that of the kernel NLMS algorithm.

B. Simulation with fixed order dictionary

Next, we consider the effect of fixed order implementation on the convergence characteristics and on the excess MSE of the kernel RLS DCD algorithm.

Here, we show the results of simulation of the forward prediction of Mackey-Glass sequence[4].

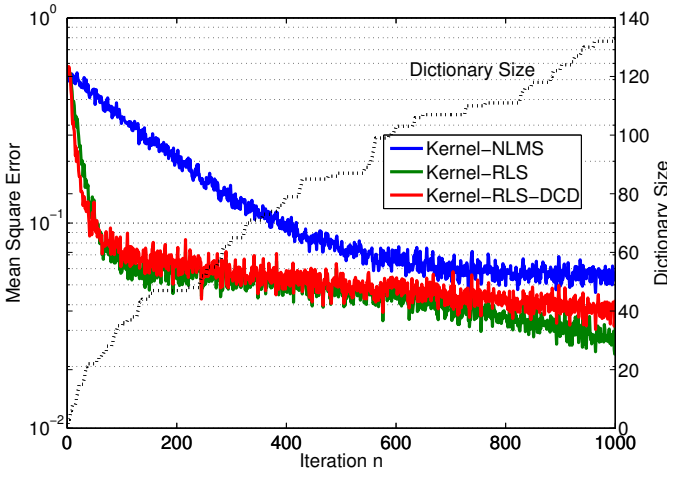

Fig. 2. Comparison of the algorithms in terms of MSE for multi-path Rayleigh channel equalization.

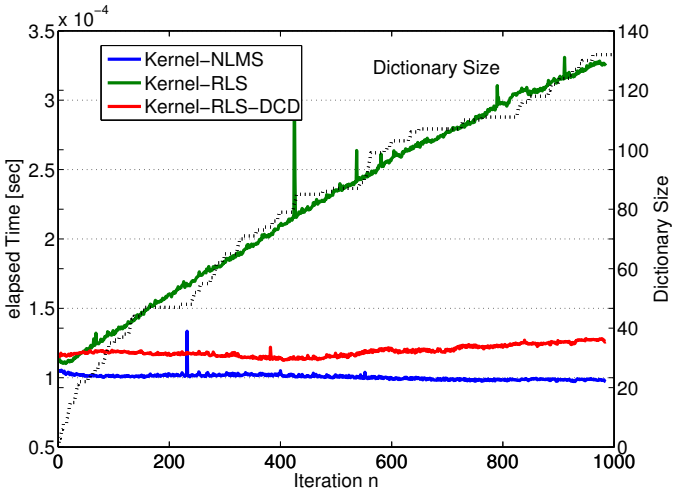

Fig. 3. Comparison of the algorithms in terms of execution time for multi-path Rayleigh channel equalization.

Mackey-Glass sequence is expressed by

$$\frac{dx(t)}{dt} = -a_0 x(t) + \frac{b_0 x(t-\tau)}{1 + x(t-\tau)^{10}} \tag{47}$$

where the constants a_0, b_0, and τ were set as $a_0 = 0.1$, $b_0 = 0.2$, $\tau = 30$. The length of the training signal was 1000 and the ensemble average of 100 independent simulations are shown. The number of taps for the adaptive filter was set as $M = 7$, and $\mathbf{x}_n = [x_n, \ldots, x_{n-6}]^T$, $y_n = x_{n+1}$.

The parameters for the proposed kernel RLS-DCD were set as $\lambda = 0.995$, $H = 1$, $N_u = 8$, $M_b = 16$, and $\mu_0 = 0.9$. We compared the kernel RLS-DCD algorithm with and without order fixing method. For order fixing method, the maximum order of the dictionary was set as $L_{\max} = 20, 40$, and 60. The Gaussian kernel with the kernel parameter $a = -0.05$ was used as the kernel function for all algorithms.

The comparison of the MSE characteristics is shown in Fig. 4. Also, in Fig. 5, we show the order of the dictionary and the indexes of the patterns stored in the dictionary. Note that, in these figures, fo-Kernel-RLS-DCD shows the kernel RLS-DCD with the proposed fixed order dictionary. From these figures, we can confirm that by decreasing the value of $L_{\max}$ the excess MSEs are slightly increasing. However, when $L_{\max} = 60$, the excess MSE of the fixed order structure is almost same as that of the non-fixed algorithm whose dictionary order reached about 120 as shown in Fig. 5.

When we did not fix the dictionary order, the average order at $n = 1000$ is about $L = 120$, so that the proposed method enables us to reduce the order by half. Also, in Fig. 5, we can confirm that the vectors in the dictionary will effectively changed to maintain the MSE characteristics.

V. Conclusions

In this paper, we proposed a fixed order implementation method of the kernel RLS-DCD algorithm. The proposed method enables us to implement with a fixed order dictionary to eliminate the variation of the required computation. Through the computer simulations implementing the proposed method, we showed that the method maintains the fast rate of convergence of the kernel RLS algorithm with less computational complexity.

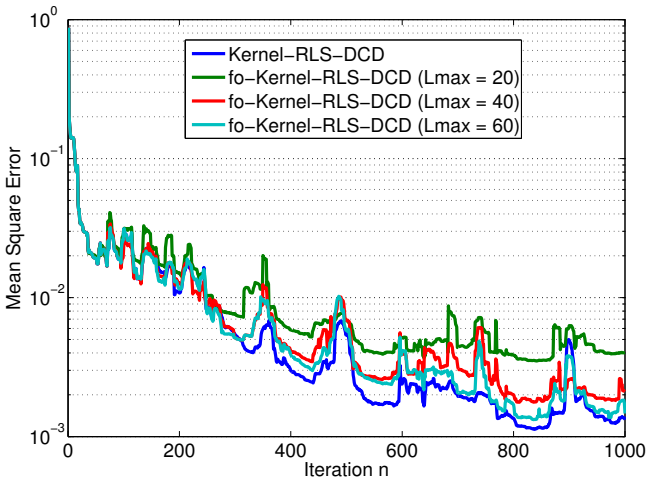

Fig. 4. Comparison of the algorithms in terms of MSE for Mackey-Glass time series. The order of the dictionary was fixed.

Acknowledgment

This work was partly supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0097

References

[1] A. H. Sayed, *Fundamentals of Adaptive Filtering*. John Wiley & Sons, 2003.
[2] A. H. Sayed, *Adaptive Filters*. John Wiley & Sons, 2008.
[3] Y. V. Zakharov, G. P. White, and J. Liu, "Low-Complexity RLS Algorithms Using Dichotomous Coordinate Descent Iterations," *IEEE Transactions on Signal Processing*, vol. 56, pp. 3150–3161, July 2008.
[4] W. Liu, J. C. Principe, and S. Haykin, *Kernel Adaptive Filtering*. Wiley, 2010.
[5] W. Liu, P. P. Pokharel, and J. C. Principe, "The Kernel Least-Mean-Square Algorithm," *IEEE Transactions on Signal Processing*, vol. 56, pp. 543–554, Feb. 2008.

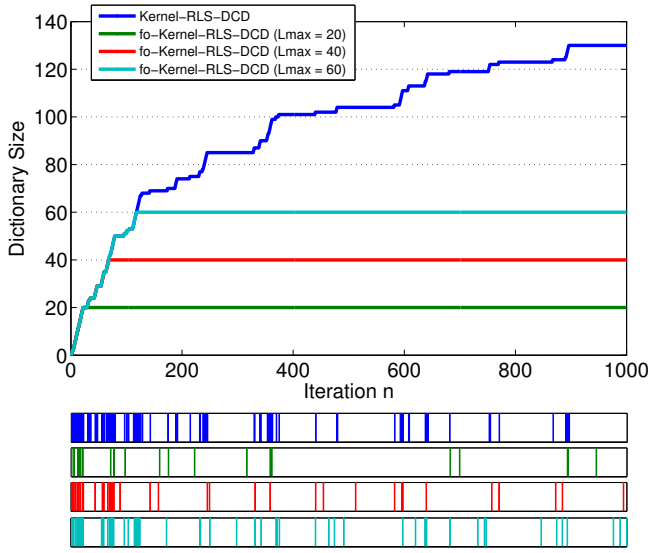

Fig. 5. Variation of the order of dictionary, and the input signal stored in the dictionary for Mackey-Glass time series simulation.

[6] C. Richard, J. C. M. Bermudez, and P. Honeine, "Online Prediction of Time Series Data With Kernels," *IEEE Transactions on Signal Processing*, vol. 57, pp. 1058–1067, Mar. 2009.

[7] F. Albu and K. Nishikawa, "The Kernel Proportionate NLMS Algorithm," in *Proc. EUSIPCO 2013*, (Marrakech, Morocco), Sept. 2013.

[8] Y. Engel, S. Mannor, and R. Meir, "The Kernel Recursive Least-Squares Algorithm," *IEEE Transactions on Signal Processing*, vol. 52, pp. 2275–2285, Aug. 2004.

[9] W. Liu, I. Park, and J. C. Principe, "An Information Theoretic Approach of Designing Sparse Kernel Adaptive Filters.," *IEEE transactions on neural networks*, vol. 20, pp. 1950–61, Dec. 2009.

[10] S. Van Vaerenbergh, I. Santamaria, W. Liu, and J. Principe, "Fixed-Budget Kernel Recursive Least-Squares," in *2012 International Conference on Acoustics, Speech and Signal Processing*, pp. 1882–1885, IEEE, Mar. 2012.

[11] Y. Ogawa and K. Nishikawa, "A Kernel Adaptive Filter based on ERLS-DCD Algorithm," in *Proc. of Intl Tech. Conf. Circuits Systems, Computer, Communications 2011*, (Gyeongju), pp. 1228–1231, June 2011.

[12] Y. Ogawa and K. Nishikawa, "A Study on Convergence Properties of Kernel RLS-DEC Adaptive Filters (In Japanese)," in *Proc. 27th Signal Processing Symposium*, (Okinawa), Nov. 2012.