第27回信号処理シンポジウム
ANAインターコンチネンタル石垣リゾート
2012年11月28 日～11月30日

Kernel RLS-DCD 適応フィルタの収束特性に関する検討

A Study on Convergence Properties of Kernel RLS-DCD Adaptive Filters

小川芳樹　　西川清史
首都大学東京 システムデザイン研究科 情報通信システム学域

Yoshiki OGAWA　　Kiyoshi NISHIKAWA
Department of Information and Communications Systems, Tokyo Metropolitan University

アブストラクト　本報告では，RLS-DCD に基づくカーネル適応アルゴリズムを提案する．カーネル適応アルゴリズムは，カーネル法を応用することで非線形システムの学習を可能とするが，学習の進行に伴い増加する辞書次数に比例する演算量の多さが課題となっている．特に，Kernel-RLS アルゴリズムにおいては，良好な収束特性に引きかえ，辞書次数の 2 乗に比例して演算量が増加する．本稿では，RLS アルゴリズムにおける補助正規方程式の解法として DCD アルゴリズムを用い，カーネル適応フィルタへ拡張することで，低演算量で実現が可能な Kernel-RLS-DCD アルゴリズムを提案する．

1　はじめに

本報告では，RLS-DCD に基づくカーネル適応アルゴリズムを提案する．提案法は，従来の RLS 型カーネル適応アルゴリズムと比べて同等の収束特性を，演算量が LMS 型と同等程度で実現可能という特徴を有する．

適応フィルタは，未知系の入力信号および出力信号から，未知系のパラメータの逐次学習を可能とし，ディジタル通信における通信路等化など，様々な応用で実用化されている [1]．適応フィルタには，少ない残留誤差で高速に推定する収束特性，システムの変化に対する追従特性などが要求される．また，適応フィルタの学習に用いられるアルゴリズムは適応アルゴリズムと呼ばれ，アルゴリズム実現に要する演算量も，フィルタを構成する際の重要な指針である．従来の適応フィルタは，線形の未知システムを想定している．システムを学習するための線形適応アルゴリズムとして，LMS(Least Mean Squares) アルゴリズムや RLS(Recursive Least Squares) アルゴリズムが広く知られている [1]．LMS アルゴリズムは実装の簡便さや良好な追従特性が特徴である．RLS アルゴリズムは多くの演算を必要とするが，高速な収束特性を示す．

線形適応アルゴリズムを利用した場合，無線通信路などのシステムの非線形性が強い環境では特性が悪化することが報告されている [2]．これに対して，非線形適応フィルタを用いることで等化性能の改善が可能である．非線形適応フィルタの構成法のひとつとして，カーネル法を応用したカーネル適応フィルタが研究されている [2][3][4]．中でも，RLS アルゴリズムを拡張した Kernel-RLS アルゴリズムは，線形の場合と同様に Kernel-LMS アルゴリズムに比べ，高速な収束性能を示すことが知られている．カーネル適応フィルタでは，過去の入力信号を辞書として蓄積し学習に用いる．辞書の次数がフィルタ次数と等しくなり，時間とともに増加する．Kernel-RLS アルゴリズムでは演算量が次数の 2 乗に比例するため，学習が進行し次数が増加するに伴い，演算量が爆発的に増加する問題がある．

本報告では，RLS アルゴリズムにおける補助正規方程式の解法として，DCD(Dichotomous Coordinate Descent) を用いる手法に着目し，カーネル適応フィルタへの拡張である Kernel RLS-DCD アルゴリズムを提案する [5]．Kernel RLS-DCD アルゴリズムを用いることで，従来の RLS 型カーネル適応アルゴリズムと比較し，収束特性を大きく損なわずに演算量を LMS アルゴリズムと同等程度まで削減できることを示す．シミュレーションにより，提案法の有用性を示す．

2　準備

ここでは準備として，適応フィルタ，RLS-DCD アルゴリズム，およびカーネル適応アルゴリズムについて説明する．以下では，行列やベクトルを太字 (e.g. $\mathbf{R}$，$\mathbf{r}$)，行列やベクトルの要素インデックスを下付き文字 (e.g. $R_{i,j}$，r_i)，そして行列における列ベクトルを上付き文字 (e.g. $\mathbf{R}^{(p)}$) でそれぞれ表す．また，$\mathbf{R}^{\mathrm{T}}$ は $\mathbf{R}$ の転置であり，$\mathbf{R}(n)$ のような括弧書きは時刻 n における変数を表している．

2.1　適応フィルタ

図 1 に，標準的な適応フィルタの構造を示す [1]．ここで，$x(n)$ は入力信号，$d(n)$ は所望信号，$v(n)$ は加法性雑音信号，$y(n)$ は出力信号，$e(n)$ は誤差信号を表す．係数更新に用いる誤差信号 $e(n)$ は，以下の式によって計算さ

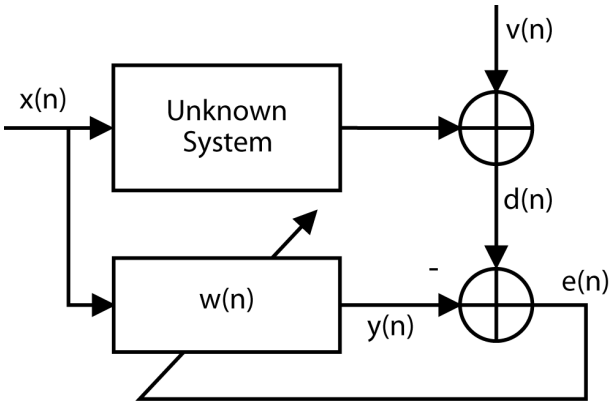

図 1: 適応フィルタによるシステム同定の構成図

れる．

$$e(n) = d(n) - y(n) = d(n) - \mathbf{x}(n)^{\mathrm{T}}\mathbf{w}(n) \tag{1}$$

$$\mathbf{x}(n) = [x(n)\ \ x(n-1)\ \ \cdots\ \ x(n-M-1)]^{\mathrm{T}} \tag{2}$$

$$\mathbf{w}(n) = [w_0(n)\ \ w_1(n)\ \ \cdots\ \ w_{M-1}(n)]^{\mathrm{T}} \tag{3}$$

ここで，$\mathbf{w}$(n) は適応フィルタの係数ベクトル，M はフィルタ次数である．適応アルゴリズムは，評価関数 $E[e^2(n)]$ もしくは $\sum_n e^2(n)$ を最小とするフィルタ係数 $\mathbf{w}$(n) を逐次的に推定し，更新する．

2.2 RLS アルゴリズム

最小二乗法に基づく適応アルゴリズムとして，RLS アルゴリズムが知られている．式 (1) より，二乗誤差和 $\sum_n e^2(n)$ を最小化する目的において，以下の正規方程式と呼ばれる方程式が導かれる．

$$\mathbf{R}(n)\mathbf{w}(n) = \boldsymbol{\beta}(n) \tag{4}$$

ここで $\mathbf{R}$ は $(M+1)\times(M+1)$ の $x(n)$ の時間平均自己相関行列で，$\boldsymbol{\beta}$ は $M+1$ の $x(n)$ と $d(n)$ の時間平均相互相関ベクトルである．RLS アルゴリズムは再帰的に式 (4) の解を求めることを可能とし，以下で表される評価関数を最小化するよう係数を更新する．

$$\varphi(n) = \sum_{n=1}^{N} \lambda^{N-n} e^2(n) \tag{5}$$

ここで，λ はシステム変化に対応するための忘却係数と呼ばるパラメータであり，$0 < \lambda \leq 1$ の範囲の実数である．$\mathbf{w}(n)$ を求めるためには，$\mathbf{R}(n)$ の逆行列を求める必要がある．逆行列の補題 [6] を用いた，一般的な RLS アルゴリズムの係数更新式を Alg.1 に示す．RLS アルゴリズムの実現には，$\mathrm{O}(M^2)$ の演算量がかかることが知られている [6].

Algorithm 1 RLS アルゴリズム

$$\mathbf{w}(n) = \mathbf{w}(n-1) + e(n)\mathbf{k}(n) \tag{6}$$

$$\mathbf{k}(n) = \frac{\mathbf{P}(n-1)\mathbf{x}(n)}{\lambda + \mathbf{x}^{\mathrm{T}}(n)\mathbf{P}(n-1)\mathbf{x}(n)} \tag{7}$$

$$\mathbf{P}(n) = \lambda^{-1}\left[\mathbf{P}(n-1) - \mathbf{k}(n)\mathbf{x}^{\mathrm{T}}(n)\mathbf{P}(n-1)\right] \tag{8}$$

2.3 RLS-DCD アルゴリズム

演算量の緩和を目的に，$\mathbf{w}$ の近似値を求める手法として，RLS-DCD アルゴリズムが提案されている [7]．以下，$\mathbf{w}(n)$ の近似値を $\hat{\mathbf{w}}(n)$ と表すこととする．正規方程式より，残差ベクトル $\mathbf{r}(n)$ と差分方程式を以下のように定義する．

$$\mathbf{r}(n-1) = \boldsymbol{\beta}(n-1) - \mathbf{R}(n-1)\hat{\mathbf{w}}(n-1) \tag{9}$$

$$\begin{cases} \Delta\mathbf{R}(n) = \mathbf{R}(n) - \mathbf{R}(n-1) \\ \Delta\boldsymbol{\beta}(n) = \boldsymbol{\beta}(n) - \boldsymbol{\beta}(n-1) \\ \Delta\mathbf{w}(n) = \mathbf{w}(n) - \hat{\mathbf{w}}(n-1) \end{cases} \tag{10}$$

式 (10) から，正規方程式を以下の通り書き直すことができる．

$$\mathbf{R}(n)\left[\hat{\mathbf{w}}(n-1) + \Delta\mathbf{w}(n)\right] = \boldsymbol{\beta}(n) \tag{11}$$

さらに変形することで，

$$\mathbf{R}(n)\Delta\mathbf{w}(n)$$

$$= \boldsymbol{\beta}(n) - \mathbf{R}(n)\hat{\mathbf{w}}(n-1) \tag{12}$$

$$= \boldsymbol{\beta}(n) - \mathbf{R}(n-1)\hat{\mathbf{w}}(n-1) - \Delta\mathbf{R}(n)\hat{\mathbf{w}}(n-1) \tag{13}$$

$$= \mathbf{r}(n-1) + \Delta\boldsymbol{\beta}(n) - \Delta\mathbf{R}(n)\hat{\mathbf{w}}(n-1) \tag{14}$$

を得る．これより，

$$\boldsymbol{\beta}_0(n) = \mathbf{r}(n-1) + \Delta\boldsymbol{\beta}(n) - \Delta\mathbf{R}(n)\hat{\mathbf{w}}(n-1) \tag{15}$$

で置き換えれば，正規方程式を解く代わりに

$$\mathbf{R}(n)\Delta\mathbf{w}(n) = \boldsymbol{\beta}_0(n) \tag{16}$$

を，$\Delta\mathbf{w}(n)$ について解く問題とすることができる (これを補助正規方程式と呼ぶ)．$\hat{\mathbf{w}}(n)$ は，

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \Delta\hat{\mathbf{w}}(n) \tag{17}$$

によって求められる．正規方程式の再帰的解法アルゴリズムを Alg.2 に示す．ここで $\boldsymbol{\Pi}$ は $\boldsymbol{\Pi} = \eta\mathbf{I}_N$，$\eta$ は微小値，そして $\mathbf{I}_N$ は $N \times N$ の単位行列を表す．

Alg.2 中の式 (21) における補助正規方程式の解法として，CG(Conjugate Gradient) や，CD(Coordinate Descent) といったアルゴリズムが提案されているが，本稿では特に演算量の面で優れている DCD(Dichotomous Coordinate

Algorithm 2 正規方程式の再帰的解法

Initialize: $\hat{\mathbf{w}}(-1) = 0$, $\mathbf{r}(-1) = 0$, $\mathbf{R}(-1) = \mathbf{\Pi}$
for $n = 0, 1, 2, \cdots$ **do**

$$\mathbf{R}(n) = \lambda \mathbf{R}(n-1) + \mathbf{x}(n)\mathbf{x}(n)^{\mathrm{T}} \tag{18}$$

$$e(n) = d(n) - \mathbf{x}(n)^{\mathrm{T}}\hat{\mathbf{w}}(n-1) \tag{19}$$

$$\boldsymbol{\beta}_0(n) = \lambda \mathbf{r}(n-1) + e(n)\mathbf{x}(n) \tag{20}$$

$$\mathbf{R}(n)\Delta\mathbf{w}(n) = \boldsymbol{\beta}_0(n) \Rightarrow \Delta\hat{\mathbf{w}}(n), \mathbf{r}(n) \tag{21}$$

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \Delta\hat{\mathbf{w}}(n) \tag{22}$$

end for

Descent) アルゴリズム [8][9] を用いる．DCD アルゴリズムは，解のベクトル要素の二進表現を用いた近似に基づいている．はじめに最上位ビットから再帰近似を行い，ビット毎に更新を行ってゆく．DCD アルゴリズムのパラメータとして，解の表現ビット数 M_b，変動範囲 H，成功反復数の上限値 N_u が用いられる．解の精度は，$2^{-M_b}H$ で表される．また，N_u の値が大きいと，解の誤差は少なくなるが，反復数が増える．DCD アルゴリズムを Alg.3 に示す．また，演算量の比較を表 1 に示す．表 1 より，DCD アルゴリズムを用いることで，演算量を O(M) に削減可能なことがわかる．

Algorithm 3 DCD アルゴリズム

Initialize: $\Delta\hat{\mathbf{w}} = 0$, $\mathbf{r} = \beta_0$, $\alpha = H/2$, $m = 1$
for $k = 1, \cdots, N_u$ **do**

$$p = \arg\max_{n=1,\cdots,N} |r_n| \tag{23}$$

while $|r_p| \leq (\alpha/2)R_{p,p}$ **do**

$$m = m + 1, \quad \alpha = \alpha/2 \tag{24}$$

if $m > M_b$ **then** the algorithm stops
end while

$$\Delta\hat{w}_p = \Delta\hat{w}_p + \mathrm{sign}(r_p)\alpha \tag{25}$$

$$\mathbf{r} = \mathbf{r} - \mathrm{sign}(r_p)\alpha\mathbf{R}^{(p)} \tag{26}$$

end for

表 1: 各アルゴリズムの演算量の比較

アルゴリズム	×	+	÷
NLMS	$2M + 3$	$2M + 3$	1
RLS	$M^2 + 5M + 1$	$M^2 + 3M$	1
RLS-DCD	$3M$	$2MN_u + 6M$	N_u

2.4 カーネル法

近年，対象とする信号を高次元の特徴空間に写像することで解析を行う，カーネル法と呼ばれる手法の，信号処理の分野への応用が検討されている [2]．入力信号 $\mathbf{u}$，$\mathbf{u}'$ が，非線形変換 ϕ によって高次元特徴空間に写像されるとき，それぞれ $\phi(\mathbf{u})$，$\phi(\mathbf{u}')$ と表すこととする．特徴空間が再生核ヒルベルト空間であると仮定すると，それらの内積を，

$$\phi(\mathbf{u})^{\mathrm{T}} \cdot \phi(\mathbf{u}') = \kappa(\mathbf{u}, \mathbf{u}') \tag{27}$$

と表されるような，任意のカーネル関数 κ を定義することができる [2]．カーネル関数は対称性，正定値性を満たす必要があり，線形カーネル，多項式カーネル，ガウスカーネルなどがよく用いられる．特にカーネル適応フィルタには，以下の式で与えられるガウスカーネルが広く用いられている．

$$\kappa(\mathbf{u}, \mathbf{u}') = \exp(-a\|\mathbf{u} - \mathbf{u}'\|^2) \tag{28}$$

ここで，a はカーネルパラメータと呼ばれ，推定対象のシステムに応じて設定する．以下では提案法にガウスカーネルを用いるものとする．カーネル関数を利用することで，変換 ϕ の詳細が不明でも，高次元空間上の内積の値を得ることができる．これをカーネルトリックと呼ぶ．カーネル法を用いて非線形システムの学習を行うアルゴリズムとして，Kernel-LMS アルゴリズム，Kernel-NLMS アルゴリズム，Kernel-RLS アルゴリズムなどが提案されている [2][3][4].

3 提案法

ここでは，提案する RLS-DCD アルゴリズムに基づくカーネル適応アルゴリズムに関して述べる．

3.1 カーネル適応アルゴリズム [2]

n 次元の特徴空間に写像された入力信号 $\phi(\mathbf{u})$ に対するフィルタ係数 $\mathbf{w}'$ を，係数ベクトル $\boldsymbol{\alpha}$ を用いて，以下のように表現する．

$$\mathbf{w}'_n = \alpha_1\phi(\mathbf{u}_0) + \cdots + \alpha_n\phi(\mathbf{u}_{n-1}) \tag{29}$$

特徴空間に写像された入力信号列 $\phi(\mathbf{u}_n)$ を，トレーニングベクトルと呼ぶ．これを用いて，出力信号 y は以下のように変形できる．

$$y_n = \phi(\mathbf{x}_n)^{\mathrm{T}}\mathbf{w}'_n \tag{30}$$

$$= \phi(\mathbf{x}_n)^{\mathrm{T}}(\alpha_1\phi(\mathbf{x}_1) + \cdots + \alpha_n\phi(\mathbf{x}_{n-1})) \tag{31}$$

$$= \alpha_1\phi(\mathbf{x}_n)^{\mathrm{T}}\phi(\mathbf{x}_1) + \cdots + \alpha_n\phi(\mathbf{x}_n)^{\mathrm{T}}\phi(\mathbf{x}_{n-1}) \tag{32}$$

$$= \alpha_1\kappa(\mathbf{x}_n, \mathbf{x}_1) + \cdots + \alpha_n\kappa(\mathbf{x}_n, \mathbf{x}_{n-1}) \tag{33}$$

ここで，ベクトル $\mathbf{h}$ を

$$\mathbf{h} = [\kappa(\mathbf{x}_n, \mathbf{x}_1) \ \cdots \ \kappa(\mathbf{x}_n, \mathbf{x}_{n-1})]^{\mathrm{T}} \tag{34}$$

と定義し，(33) 式を置き換えることで，$y(n) = \mathbf{h}^{\mathrm{T}}\boldsymbol{\alpha}$ と変形できる．ここでベクトル $\boldsymbol{\alpha}$ を適応フィルタの係数とみなして，線形適応アルゴリズムを用いることが可能となる [2]．提案法では，$\boldsymbol{\alpha}$ の更新に RLS-DCD アルゴリズムを利用することを検討する．

3.2 次数変動への対応

学習が進行するに従い，トレーニングベクトルが蓄積され，フィルタ係数ベクトル $\boldsymbol{\alpha}$ の次数および特徴空間の次元が増加する．すなわち式 (33) の和の項数は時刻 n とともに増加する．このため，従来の RLS-DCD を直接用いることはできず，辞書次数が増加を考慮し，行列やベクトルの次数が増加する状態で学習が可能となる更新式を導出する必要が生じる．

提案法では，Alg.2 の，式 (18) の $\mathbf{R}(n)$ の更新式を，以下のように次数変化に対応したものに拡張する．

$$\mathbf{R}(n) = \lambda \begin{bmatrix} \mathbf{R}(n-1) & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} + \mathbf{h}(n)\mathbf{h}(n)^{\mathrm{T}} \tag{35}$$

ここで $\mathbf{0}$ は $\mathbf{R}(n-1)$ と同じ行数，または同じ列数の零ベクトルである．同様に，式 (19)(20)(22) についても，以下で示す更新式を提案する．

$$e(n) = d(n) - \mathbf{h}(n)^{\mathrm{T}} \begin{bmatrix} \boldsymbol{\alpha}(n-1) \\ 0 \end{bmatrix} \tag{36}$$

$$\boldsymbol{\beta}_0 = \lambda \begin{bmatrix} \mathbf{r}(n-1) \\ 0 \end{bmatrix} + e(n)\mathbf{h}(n) \tag{37}$$

$$\boldsymbol{\alpha}(n) = \begin{bmatrix} \boldsymbol{\alpha}(n-1) \\ 0 \end{bmatrix} + \Delta\boldsymbol{\alpha}(n) \tag{38}$$

なお，DCD アルゴリズム部は独立しているため，次数変動の影響を受けず，そのまま利用することができる．提案する正規方程式の再帰的解法を，Alg.4 に示す．

Algorithm 4 次数変動に対応した正規方程式の再帰的解法

1: **Initialize:** $\hat{\mathbf{w}}(-1) = 0$, $\mathbf{r}(-1) = 0$, $\mathbf{R}(-1) = \mathbf{\Pi}$
2: **for** $n = 0, 1, 2, \cdots$ **do**
3: $$\mathbf{R}(n) = \lambda \begin{bmatrix} \mathbf{R}(n-1) & 0 \\ 0 & 1 \end{bmatrix} + \mathbf{h}(n)\mathbf{h}(n)^{\mathrm{T}} \tag{39}$$
4: $$e(n) = d(n) - \mathbf{h}(n)^{\mathrm{T}} \begin{bmatrix} \boldsymbol{\alpha}(n-1) \\ 0 \end{bmatrix} \tag{40}$$
5: $$\boldsymbol{\beta}_0 = \lambda \begin{bmatrix} \mathbf{r}(n-1) \\ 0 \end{bmatrix} + e(n)\mathbf{h}(n) \tag{41}$$
6: $$\mathbf{R}(n)\Delta\boldsymbol{\alpha}(n) = \boldsymbol{\beta}_0(n) \Rightarrow \Delta\boldsymbol{\alpha}(n), \mathbf{r}(n) \tag{42}$$
7: $$\boldsymbol{\alpha}(n) = \begin{bmatrix} \boldsymbol{\alpha}(n-1) \\ 0 \end{bmatrix} + \Delta\boldsymbol{\alpha}(n) \tag{43}$$
8: **end for**

3.3 入力信号のスパース化

過去の入力信号をすべて利用すると，膨大な演算が必要となる．そこで，入力信号をスパース化し，有限の次数でカーネル適応フィルタを構成する手法が知られている [10]．過去の信号は，辞書と呼ばれる行列を別途構成して用いられる．具体的には，カーネル関数によって定められる閾値 μ_0 を定義し，入力信号の写像集合 $\mathbf{h}$ と比較し，

$$\max|h_j(n)|_{j=1,\cdots,m} < \mu_0 \tag{44}$$

を満たす場合，内積の定義から類似したベクトルが辞書に存在しないとみなし，辞書にそのときの入力信号を追加する．一方，閾値以上であるときは，既に類似のベクトルが辞書に存在しているとみなし，辞書およびその他の変数には変化を加えず，学習を続行させる．なお，信号のスパース化の手法には，様々な手法が提案されている [2] が，ここでは実装の容易さから [10] の手法を用いるものとする．

入力信号のスパース化を行う場合の，提案する Kernel-RLS-DCD アルゴリズムのフィルタ更新式を，Alg.5 に示す．ここで，$\mathbf{x}(n)$ は入力ベクトル，$d(n)$ は所望信号，$e(n)$ は誤差信号，$\mathbf{h}(n)$ は入力信号の写像集合，$\boldsymbol{\alpha}(n)$ は $\mathbf{h}(n)$ に対応するフィルタ係数，λ は忘却係数，μ_0 は閾値，$\kappa(\cdot,\cdot)$ はカーネル関数，$\mathbf{D}(n)$ は辞書，DCD() は補助正規方程式を DCD アルゴリズムで解くことを表す．

4 シミュレーション

提案法の効果を確認するため，シミュレーションによる従来のカーネル適応アルゴリズムとの比較を行う．

4.1 前方予測

まず，以下の式で与えられる前方予測のシミュレーションを行った [10].

$$\begin{aligned} d_n =& 0.1\sin(d_{n-1}\pi) + \left(0.8 - 0.5\exp\left(-d_{n-1}^2\right)\right) d_{n-1} \\ & - \left(0.3 + 0.9\exp\left(-d_{n-1}^2\right)\right) d_{n-2} \end{aligned} \tag{59}$$

初期値 (d_{-1}, d_{-2}) は，区間 $[0, 1]$ においてランダムに与えられる．また，生成する信号には $\sigma^2 = 0.01$ である加法性白色ガウスノイズを付加する．信号長は 3000，フィルタタップ数 $M = 8$ であり，$n = 1000$ のとき，式を以下のように変化させた．

$$\begin{aligned} d_n =& 0.1\sin(d_{n-1}\pi) + \left(0.2 - 0.7\exp\left(-d_{n-1}^2\right)\right) d_{n-1} \\ & - \left(0.8 + 0.8\exp\left(-d_{n-1}^2\right)\right) d_{n-2} \end{aligned} \tag{60}$$

Kernel-RLS-DCD アルゴリズムにおけるパラメータは $\lambda = 0.995$，$H = 1$，$N_u = 4$，$M_b = 16$，$\mu_0 = 0.8$ を選択した．Kernel-NLMS アルゴリズム，Kernel-RLS アルゴリズムを

Algorithm 5 Kernel-RLS-DCD アルゴリズム

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**
  $\mathbf{h}(n) = \kappa(\mathbf{x}(n), \mathbf{D}(n-1))$ (45)
  if $\max|h_j(n)|_{j=1,\cdots,m} > \mu_0$ **then**
    $\mathbf{D}(n) = \mathbf{D}(n-1)$ (46)
    $\mathbf{R}(n) = \lambda\mathbf{R}(n-1) + \mathbf{h}(n)^{\mathrm{T}}\mathbf{h}(n)$ (47)
    $e(n) = d(n) - \mathbf{h}(n)^{\mathrm{T}}\boldsymbol{\alpha}(n-1)$ (48)
    $\boldsymbol{\beta}_0(n) = \lambda\mathbf{r}(n-1) + e(n)\mathbf{h}(n)$ (49)
    $(\Delta\boldsymbol{\alpha}(n), \mathbf{r}(n)) = \mathrm{DCD}(\mathbf{R}(n), \boldsymbol{\beta}_0(n))$ (50)
    $\boldsymbol{\alpha}(n) = \boldsymbol{\alpha}(n-1) + \Delta\boldsymbol{\alpha}(n)$ (51)
  else
    $\mathbf{D}(n) = \mathbf{D}(n-1) \cup \{\mathbf{x}(n)\}$ (52)
    $\mathbf{h}(n) = \begin{bmatrix}\mathbf{h}(n)\\ 1\end{bmatrix}$ (53)
    $\mathbf{R}(n) = \lambda\begin{bmatrix}\mathbf{R}(n-1) & 0\\ 0 & 1\end{bmatrix} + \mathbf{h}(n)^{\mathrm{T}}\mathbf{h}(n)$ (54)
    $e(n) = d(n) - \mathbf{h}(n)^{\mathrm{T}}\begin{bmatrix}\boldsymbol{\alpha}(n-1)\\ 0\end{bmatrix}$ (55)
    $\boldsymbol{\beta}_0(n) = \lambda\begin{bmatrix}\mathbf{r}(n-1)\\ 0\end{bmatrix} + e(n)\mathbf{h}(n)$ (56)
    $(\Delta\boldsymbol{\alpha}(n), \mathbf{r}(n)) = \mathrm{DCD}(\mathbf{R}(n), \boldsymbol{\beta}_0(n))$ (57)
    $\boldsymbol{\alpha}(n) = \begin{bmatrix}\boldsymbol{\alpha}(n-1)\\ 0\end{bmatrix} + \Delta\boldsymbol{\alpha}(n)$ (58)
  end if
end for

比較対象とし，どちらも 3.3 で説明した入力信号のスパース化を適用しており，それぞれに最適なパラメータを設定した．すべてのアルゴリズムのカーネル関数としてガウスカーネルを用い，カーネルパラメータの値は [10] に従い，$a = 3.73$ を選択した．

MSE の比較を図 2 に示す．また，実行時間の比較を図 3 に示す．図 2 より，提案法は，Kernel-RLS アルゴリズムと比較して，初期収束および追従性能では若干の特性劣化が見られるが，残留誤差については同等の特性を有することがわかる．システム変化後の最終的な残留誤差は，Kernel-RLS アルゴリズムより少ないことがわかる．また，図 3 より，システム変化によって次数が増大したとき，Kernel-RLS アルゴリズムは実行時間が大きく増加しているのに対し，提案法は Kernel-NLMS アルゴリズムとほぼ同等の実行時間に抑えられていることがわかる．

4.2 レイリーチャネル等化

次に，レイリーチャネルの等化を行った [2]．信号長は 2000 で，$n = 1000$ のとき，パスの係数をランダムに変化させ，システム変化が発生したことを想定した．パス数

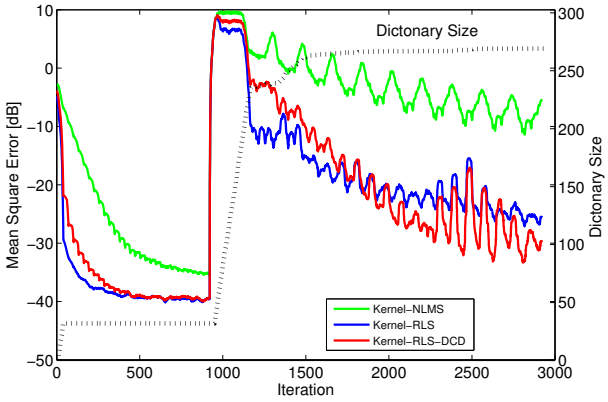

図 2: 前方予測の MSE による比較

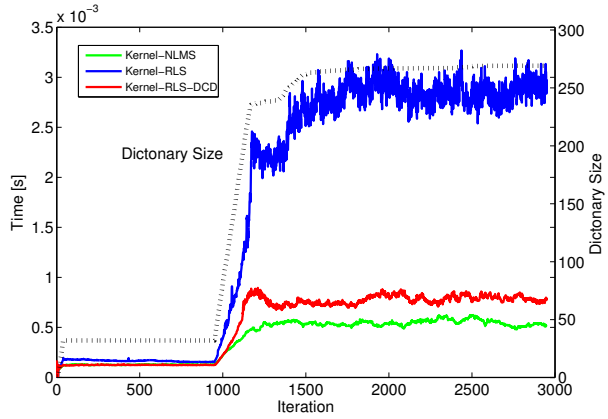

図 3: 前方予測の実行時間による比較

として $M = 5$，最大ドップラー周波数として $f_D = 100$Hz，サンプリングレートとして $T_s = 0.8\mu$s を選択した．Kernel-RLS-DCD アルゴリズムにおけるパラメータは $\lambda = 0.995$，$H = 1$，$N_u = 8$，$M_b = 16$，$\mu_0 = 0.8$ を選択した．Kernel-NLMS アルゴリズム，Kernel-RLS アルゴリズムを比較対象とし，どちらも 3.3 で説明した入力信号のスパース化を適用しており，それぞれに最適なパラメータを設定した．すべてのアルゴリズムのカーネル関数としてガウスカーネルを用い，$a = 0.05$ を選択した．

MSE の比較を図 4 に示す．また，実行時間の比較を図 5 に示す．図 4 より，システム変化前，提案法は Kernel-RLS アルゴリズムとほぼ同等の収束特性を示していることがわかる．一方，システム変化後に Kernel-RLS アルゴリズムの追従特性が悪化しているのに対して，提案法はシステム変化前と同等の水準の残留誤差を実現している．また，図 5 より，Kernel-RLS アルゴリズムの実行速度がフィルタ次数に伴い増加していくのに対して，提案法では Kernel-NLMS アルゴリズム同様に，安定した実行速度の推移が見られる．

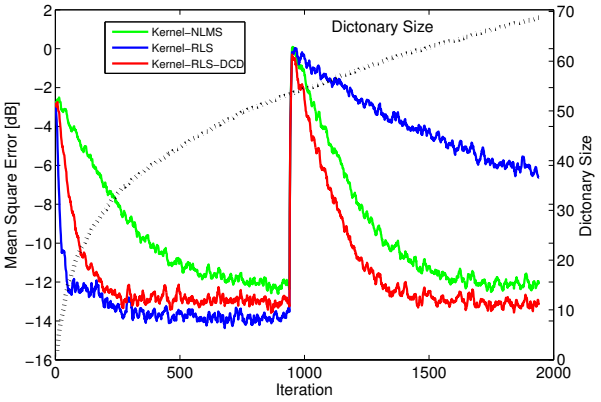

図 4: チャネル等化の MSE による比較

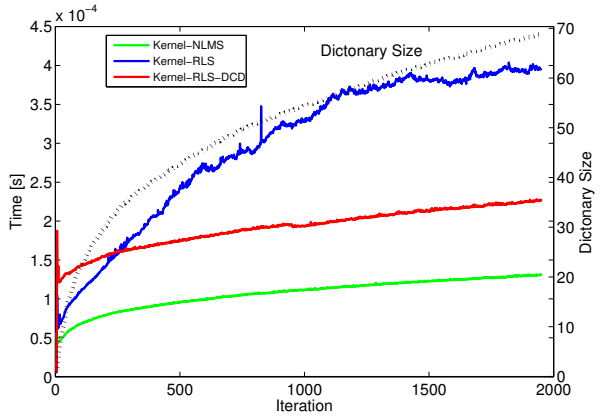

図 5: チャネル等化の実行時間による比較

5 まとめ

本稿では，RLS-DCD アルゴリズムを，非線形システムにおいても学習可能なカーネル適応アルゴリズムに拡張した，Kernel-RLS-DCD アルゴリズムを提案した．提案法は線形の RLS-DCD にカーネル法を適用し，非線形システムの学習を可能とする．カーネル適応フィルタのフィルタ次数の変化を考慮した更新式を導出し，入力信号のスパース化による演算量削減法に関して検討を行った．提案法を用いたコンピュータシミュレーションにより，良好な収束特性を持ち，かつ従来の RLS 型カーネル適応アルゴリズムよりも少ない演算量で学習が行えることを示した．

参考文献

[1] Ali H. Sayed, “Fundamentals of Adaptive Filtering,” WILEY, 2003.

[2] Weifeng Liu, José C. Príncipe, Simon Haykin “Kernel Adaptive Filtering,” WILEY, 2010.

[3] Yaakov Engel, Shie Mannor, and Ron Meir “The Kernel Recursive Least-Squares Algorithm,” IEEE TRANS. SIGNAL PROCESSING, Vol.52, No.8, Aug 2004.

[4] Weifeng Liu, Puskal P. Pokharel, and José C. Príncipe “The Kernel Least-Mean-Square Algorithm,” IEEE TRANS. SIGNAL PROCESSING, Vol.56, No.2, Feb 2008.

[5] Yoshiki OGAWA and Kiyoshi NISHIKAWA, “A Kernel Adaptive Filter Based on ERLS-DCD Algorithm,” Proc. International Technical Conference on Circuits/Systems, Computers and Communications, no.P4-13, pp.1228-1231, Gyeongju, Korea, 21st June, 2011.

[6] Simon Haykin, “Adaptive Filter Theory,” Prentice Hall, 2001.

[7] Yuriy V. Zakharov, George P. White, and Jie Liu, “Low-Complexity RLS Algorithms Using Dichotomous Coordinate Descent Iterations,” IEEE TRANS. SIGNAL PROCESSING, Vol.56, No.7, Jul.2008.

[8] Yuriy V. Zakharov and T.C.Tozer, “Multiplication-free iterative algorithm for LS problem,” Electron.Lett., vol.40, no.9, pp.567-569, Apr 2004.

[9] Yuriy Zakharov and Felix Albu, “Coordinate Descent Iterations in Fast Affine Projection Algorithm,” IEEE SIGNAL PROCESSING LETTERS, Vol.12, No.5, May 2005.

[10] Cédric Richard, José Carlos M. Bermudez, and Paul Honeine “Online Prediction of Time Series Data With Kernels,” IEEE TRANS. SIGNAL PROCESSING, Vol.57, No.3, Mar 2009.