カーネル適応アルゴリズムの低演算量化のための辞書データ作成法

A Low Complexity Method for Dictionary Construction for Kernel-based Adaptive Algorithms

牧崎幸司 † 西川清史 †
†首都大学東京大学院システムデザイン研究科情報通信システム学域

Koji MAKIZAKI† Kiyoshi NISHIKAWA†
†Department of Information and Communications Systems Engineering,
Tokyo Metropolitan Univesity

アブストラクト 線形適応フィルタを用いて非線形システムの学習を行うための手法として，近年，カーネル適応アルゴリズムが注目されている．カーネルトリックと呼ばれる手法を用いて，高次の特徴空間に写像された入力ベクトルに対して，線形適応アルゴリズムを適用することで，非線形システムの学習が可能となる．しかし，入力ベクトルを特徴空間へ写像する際に，過去の入力ベクトルとの内積が必要となり，演算量が多いという問題がある．本稿では，カーネル適応アルゴリズムにおいて必要となる辞書データ作成のための演算量を削減する手法を提案する．カーネル適応アルゴリズムで使用される辞書データの更新に必要となる演算を変形し，過去に演算された値を保持することで，辞書データの作成に必要となる演算量をほぼ半分に削減可能とする手法を提案する．

1 はじめに

適応フィルタは，エコーキャンセラやノイズ除去，音声圧縮などに広く利用されている [1]．従来，適応フィルタは線形システムの学習を中心に研究されており，非線形なシステムの学習には，Volterra フィルタ等が利用されている [2]-[8]．しかし，Volterra フィルタは，高次な系の学習に多大な演算が必要となる等の問題がある．

近年，線形適応フィルタを用いて非線形な系を学習するアルゴリズムとして，カーネル適応アルゴリズムが提案されている [8]-[11]．カーネル適応アルゴリズムは，カーネル法を線形適応アルゴリズムに適用し，導出される．カーネル法は，高次の特徴空間内におけるベクトルの内積をカーネル関数を用いて求める．この手法をカーネルトリックと呼ぶ [12]，[13]．また，学習に用いられるアルゴリズムとしては，NLMS アルゴリズムや RLS アルゴリズムに基づく Kernel NLMS アルゴリズムや Kernel RLS アルゴリズム等が提案されている [8]，[9]．

カーネル適応アルゴリズムでは，過去の入力ベクトルをトレーニングベクトルとして格納した辞書を作成し，リアルタイムに更新する．辞書の更新には，カーネルトリックを用いて入力ベクトルと辞書内の全てのトレーニングベクトルとの内積を求める必要がある．入力ベクトルは，辞書内のベクトルが張る特徴空間内に写像され，LMS アルゴリズムや RLS アルゴリズムなどの線形適応アルゴリズムが適用される．辞書内のトレーニングベクトルとの内積を求める操作は，用いられる適応アルゴリズムによらず必要であり，辞書の大きさが大きくなるにつれ演算量が増加する．特に NLMS アルゴリズムのような演算量の少ないアルゴリズムとの組み合わせの際に問題となる．

本報告では，辞書内のトレーニングベクトルとの内積を求める際に必要となる演算量を削減するための手法を提案する．過去に，辞書を更新する際に求めた演算結果を辞書に保持することで，カーネル法における演算量をほぼ半分に削減する．計算機シミュレーションにより，従来法と提案法の処理時間の比較結果を示す．結果より，提案法は従来法と比較して処理が高速となり，提案法の有効性を確認できることを示す．

2 準備

ここでは，カーネル法とカーネル適応アルゴリズムについて説明する．

2.1 カーネル法

線形適応アルゴリズムを用いて非線形な系を学習をするには，入力空間 X のデータ $\mathbf{x}$ に対して，線形適応アルゴリズムで学習できる特徴空間 F への非線形写像 $\Phi : X \to F$ を行う必要がある．

非線形適応フィルタは，次の 2 つのステップから構成される．まず，固定された非線形写像 Φ により，入力ベクトル $\mathbf{x}$ を高次な特徴空間 F 内のベクトルに変換する．次に，特徴空間 F 内において線形適応アルゴリズムを用いて学習を行う．特徴空間における線形適応フィルタの係数ベクトルを $\mathbf{w}$ とおくと，入力ベクトル $\mathbf{x}$ に対する非線

形適応フィルタの出力信号は次のように表される．

$$f(\mathbf{x}) = \Phi^T(\mathbf{x})\,\mathbf{w} \tag{1}$$

ここで，$\mathbf{w}$ は，m 個のトレーニングベクトル $\mathbf{y}_j$ $(j = 1,\ldots,m)$ の写像 $\Phi\left(\mathbf{y}_j\right)$ の線形結合

$$\mathbf{w} = \sum_{j=1}^{m} \alpha(j)\,\Phi\left(\mathbf{y}_j\right) \tag{2}$$

で与えられるとする．このとき，

$$f(\mathbf{x}) = \sum_{j=1}^{m} \left(\Phi^T(\mathbf{x})\,\Phi\left(\mathbf{y}_j\right)\right)\alpha(j) \tag{3}$$

と表すことができる [12], [13]．いま，$\boldsymbol{\alpha} = [\alpha(1),\ldots,\alpha(m)]^T$ と表すものとする．$\boldsymbol{\alpha}$ は m 個のトレーニングベクトルに対する重みであり，$\boldsymbol{\alpha}$ を未知系の係数ベクトルと見なし，適応アルゴリズムを構築する．式 (3) より，特徴空間内の内積 $\Phi(\mathbf{x})^T\,\Phi\left(\mathbf{y}_j\right)$ を求めることが可能であれば，非線形適応アルゴリズムが構築できる [9]．特徴空間 F 内での内積を計算するための一手法が，カーネル法である [12], [13]．

カーネル法では，データは高次の特徴空間に写像される．写像されたベクトル同士の内積を求めるために使用される関数をカーネル関数と呼ぶ．カーネル関数は，入力空間 X から特徴空間 F への写像 $\Phi(\cdot)$ を用いて以下のように与えられる．

$$\forall \mathbf{a},\mathbf{b} \in X \qquad k(\mathbf{a},\mathbf{b}) = \Phi^T(\mathbf{a})\,\Phi(\mathbf{b}) \tag{4}$$

ここで，$k(\cdot,\cdot)$ がカーネル関数であり，特徴空間における内積を与える関数として捉えることができる．式 (4) において，右辺の高次な内積 $\Phi^T(\mathbf{a})\,\Phi(\mathbf{b})$ を求める代わりにカーネル関数を用いて，内積を直接計算する．この手法のことをカーネルトリック (kernel trick) と呼ぶ [12], [13]．カーネル適応アルゴリズムに用いられるカーネル関数が満足すべき条件として，式 (5) がある [8]．

$$k(\mathbf{a},\mathbf{a}) = 1 \tag{5}$$

カーネル適応フィルタを実現するためのカーネル関数として，この条件を満たす以下のガウシアンカーネルが広く用いられている [8]-[10]．

$$k(\mathbf{a},\mathbf{b}) = \exp\left(-\|\mathbf{a}-\mathbf{b}\|^2\right) \tag{6}$$

ここで，$\|\cdot\|$ はユークリッドノルムを示し，

$$\|\mathbf{a}\| = \sqrt{\mathbf{a}^T\mathbf{a}} \tag{7}$$

である．本報告も以下ではガウシアンカーネルを用いることとする．

2.2 カーネル適応アルゴリズム

カーネル法を適応アルゴリズムに適用するにあたり，式 (3) を次のように書き換える．

$$f(\mathbf{x}) = \begin{bmatrix} \Phi(\mathbf{x})^T\,\Phi(\mathbf{y}_1) \\ \Phi(\mathbf{x})^T\,\Phi(\mathbf{y}_2) \\ \vdots \\ \Phi(\mathbf{x})^T\,\Phi(\mathbf{y}_m) \end{bmatrix}^T \boldsymbol{\alpha} = \begin{bmatrix} k(\mathbf{x},\mathbf{y}_1) \\ k(\mathbf{x},\mathbf{y}_2) \\ \vdots \\ k(\mathbf{x},\mathbf{y}_m) \end{bmatrix}^T \boldsymbol{\alpha} \tag{8}$$

ここで，$\mathbf{h} = [k(\mathbf{x},\mathbf{y}_1),\ldots,k(\mathbf{x},\mathbf{y}_m)]^T$ とおく．このとき，$\mathbf{h}$ を未知系 $\boldsymbol{\alpha}$ への入力ベクトルと見なし，適応アルゴリズムによりフィルタ $\boldsymbol{\alpha}$ の係数を更新する [8]．

トレーニングベクトルが格納された行列 $[\mathbf{y}_1,\ldots,\mathbf{y}_m]$ は，辞書 $\mathbf{D}$ と呼ばれる．

$$\mathbf{D} = \begin{bmatrix} \mathbf{y}_1 & \cdots & \mathbf{y}_m \end{bmatrix} \tag{9}$$

辞書 $\mathbf{D}$ 内のベクトルは，次に見る様に，過去に用いられた m 個の入力ベクトル $\mathbf{x}$ により構成される．

辞書 $\mathbf{D}_n$，および，$\mathbf{h}_n$ は以下のアルゴリズムにより更新される．

Initialization

$\mathbf{D}_1 = \mathbf{y}_1 = \mathbf{x}_1$

$\mathbf{h}_1 = k(\mathbf{x}_1,\mathbf{y}_1)$

$\boldsymbol{\alpha}_1 = 0 \quad , \quad m = 1$

for $n = 2,3,\cdots$

if $\max_{j=1,\ldots,m} |k\left(\mathbf{x}_n,\mathbf{y}_j\right)| > \mu_0$ (10)

$\mathbf{D}_n = \mathbf{D}_{n-1}$

$\mathbf{h}_n = \begin{bmatrix} k(\mathbf{x}_n,\mathbf{y}_1) & \cdots & k(\mathbf{x}_n,\mathbf{y}_m) \end{bmatrix}^T$ (11)

else

$m = m + 1$

$\mathbf{D}_n = \mathbf{D}_{n-1} \cup \{\mathbf{x}_n\}$

$\mathbf{h}_n = \begin{bmatrix} k(\mathbf{x}_n,\mathbf{y}_1) & \cdots & k(\mathbf{x}_n,\mathbf{y}_m) \end{bmatrix}^T$ (12)

end if

end for

ここで，$\mathbf{x}_n$ は入力ベクトル，d_n は所望信号，m はフィルタ次数，$k(\cdot,\cdot)$ はカーネル関数，μ_0 は閾値である．閾値 μ_0 はモデルのスパース性の度合いにより決定され，$0 < \mu_0 < 1$ の値をとる．

カーネル適応アルゴリズムでは，閾値 μ_0 により決定される条件に基づいて，M 次の入力ベクトル $\mathbf{x}_n$ を $M \times m$ 次の辞書 $\mathbf{D}_n$ に保持する．ガウシアンカーネルは，2 つのベクトルのユークリッド距離より値が定められ，ベクトル

同士の類似度を 0 から 1 までの実数で表す．条件文 (10) は，入力ベクトル $\mathbf{x}_n$ との類似度が閾値 μ_0 より大きいトレーニングベクトル $\mathbf{y}_j$ が存在するか検索し，条件に合わなければ，$\mathbf{x}_n$ を新しいトレーニングベクトルとして辞書 $\mathbf{D}_n$ に格納する．

各時刻において，$\mathbf{x}_n$ と $\mathbf{D}_n$ から，カーネル演算によって m 次の $\mathbf{h}_n$ が得られ，$\mathbf{h}_n$ を入力ベクトルと見なし，線形適応アルゴリズムによるフィルタ係数の更新がなされる．

フィルタの更新には，NLMS(Normalized Least Mean Square) アルゴリズム [1] や RLS(Recursive Least Squares) アルゴリズム [1] などの線形適応アルゴリズムを用いる手法が提案されている．以下にその概略を示す．

2.2.1 Kernel NLMS アルゴリズム [8]

NLMS アルゴリズムに基づく Kernel NLMS(KNLMS) によるフィルタ更新の手順を次に示す．なお，辞書の更新のため，フィルタ次数 m は時変となる．時刻 n に対して m が変化しない場合は，通常の NLMS アルゴリズムと同一の更新式が用いられる．

$$e_n = d_n - \mathbf{h}_n^T \boldsymbol{\alpha}_{n-1} \tag{13}$$

$$\boldsymbol{\alpha}_n = \boldsymbol{\alpha}_{n-1} + \frac{\eta e_n}{\epsilon + \|\mathbf{h}_n\|^2}\mathbf{h}_n \tag{14}$$

一方，m が変化した場合のフィルタ更新式は，以下で与えられる．

$$e_n = d_n - \mathbf{h}_n^T \begin{bmatrix} \boldsymbol{\alpha}_{n-1} \\ 0 \end{bmatrix} \tag{15}$$

$$\boldsymbol{\alpha}_n = \begin{bmatrix} \boldsymbol{\alpha}_{n-1} \\ 0 \end{bmatrix} + \frac{\eta e_n}{\epsilon + \|\mathbf{h}_n\|^2}\mathbf{h}_n \tag{16}$$

ここで，η はステップサイズ，ϵ は安定化パラメータである．式 (15)，(16) では，時刻 $n-1$ に対して，フィルタ次数 m が増加していることに注意する．

2.2.2 Kernel RLS アルゴリズム [9]

RLS アルゴリズムに基づく Kernel RLS(KRLS) によるフィルタ更新の手順を次に示す．この場合も m の値の変化に応じて異なる更新式を用いる．時刻 n に対して m が変化しない場合は，通常の RLS アルゴリズムと同一の更新式が用いられる．

$$e_n = d_n - \mathbf{h}_n^T \boldsymbol{\alpha}_{n-1} \tag{17}$$

$$\mathbf{K}_n = \frac{\mathbf{P}_{n-1}\mathbf{h}_n}{\lambda + \mathbf{h}_n^T \mathbf{P}_{n-1}\mathbf{h}_n} \tag{18}$$

$$\boldsymbol{\alpha}_n = \boldsymbol{\alpha}_{n-1} + \mathbf{K}_n e_n \tag{19}$$

$$\mathbf{P}_n = \lambda^{-1}\left(\mathbf{P}_{n-1} - \mathbf{K}_n \mathbf{h}_n^T \mathbf{P}_{n-1}\right) \tag{20}$$

一方，m が変化した場合のフィルタ更新式は，以下で与えられる．

$$e_n = d_n - \mathbf{h}_n^T \begin{bmatrix} \boldsymbol{\alpha}_{n-1} \\ 0 \end{bmatrix} \tag{21}$$

$$\mathbf{K}_n = \frac{\begin{bmatrix} \mathbf{P}_{n-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}\mathbf{h}_n}{\lambda + \mathbf{h}_n^T \begin{bmatrix} \mathbf{P}_{n-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}\mathbf{h}_n} \tag{22}$$

$$\boldsymbol{\alpha}_n = \begin{bmatrix} \boldsymbol{\alpha}_{n-1} \\ 0 \end{bmatrix} + \mathbf{K}_n e_n \tag{23}$$

$$\mathbf{P}_n = \lambda^{-1}\left(\begin{bmatrix} \mathbf{P}_{n-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} - \mathbf{K}_n \mathbf{h}_n^T \begin{bmatrix} \mathbf{P}_{n-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}\right) \tag{24}$$

ここで，λ は忘却係数である．また，$\mathbf{0}$ は全ての要素が 0 である $m-1$ 次のベクトルを示す．式 (21)-(24) では，フィルタ次数 m の増加に対応した更新式となっている．

どちらのアルゴリズムを用いても，カーネル部分における計算は共通である．カーネル部分では，入力ベクトルと辞書内のトレーニングベクトルとの特徴空間 F における内積を求める必要がある．特に NLMS アルゴリズムを用いた場合，入力ベクトルの次数 $M = 4$ のとき，カーネル部分の演算回数はフィルタ更新の演算回数の 2 倍近くかかるなど，カーネル演算のコストが多く，問題となる．

3 提案法

ここでは，提案するカーネル演算の手法を示す．

3.1 ガウシアンカーネルの変形

提案法では，カーネル関数としてガウシアンカーネルを用いると仮定する．ガウシアンカーネルを次のように展開する．

$$\begin{aligned} k(\mathbf{a}, \mathbf{b}) &= \exp\left(-\|\mathbf{a} - \mathbf{b}\|^2\right) \\ &= \exp\left(2\mathbf{a}^T\mathbf{b} - \mathbf{a}^T\mathbf{a} - \mathbf{b}^T\mathbf{b}\right) \end{aligned} \tag{25}$$

式 (25) を用いると，式 (11)，(12) の $\mathbf{h}_n$ を

$$\mathbf{h}_n = \exp\left(\begin{bmatrix} 2\mathbf{x}_n^T\mathbf{y}_1 - \mathbf{x}_n^T\mathbf{x}_n - \mathbf{y}_1^T\mathbf{y}_1 \\ 2\mathbf{x}_n^T\mathbf{y}_2 - \mathbf{x}_n^T\mathbf{x}_n - \mathbf{y}_2^T\mathbf{y}_2 \\ \vdots \\ 2\mathbf{x}_n^T\mathbf{y}_m - \mathbf{x}_n^T\mathbf{x}_n - \mathbf{y}_m^T\mathbf{y}_m \end{bmatrix}\right) \tag{26}$$

と変形することができる．式 (26) より，カーネルの $\exp(\cdot)$ 内の 3 つの項のうち，$\mathbf{x}_n$ を含む項は第 1 項 $\mathbf{x}_n^T\mathbf{y}_j$ $(j = 1, 2, \ldots, m)$ と第 2 項 $\mathbf{x}_n^T\mathbf{x}_n$ であり，この 2 項は毎時刻の計算を要する．第 1 項 $\mathbf{x}_n^T\mathbf{y}_j$ は m 回の内積をする必要があるが，第 2 項 $\mathbf{x}_n^T\mathbf{x}_n$ は全ての行に共通している項である

ため，m 回繰り返す必要がない．また，トレーニングベクトルは過去に用いられた入力ベクトルであり，第 3 項 $\mathbf{y}_j^T\mathbf{y}_j$ は，ある時刻 i ($i \le n$) における入力ベクトル $\mathbf{x}_i$ と自身との内積 $\mathbf{x}_i^T\mathbf{x}_i$ である．入力ベクトル $\mathbf{x}_n$ が辞書 $\mathbf{D}_n$ に格納される際，第 2 項 $\mathbf{x}_n^T\mathbf{x}_n$ の演算結果も辞書に格納する．$\mathbf{y}_j^T\mathbf{y}_j$ は辞書の参照から得られ，演算量の削減が期待できる．

3.2 提案する辞書の作成法

本報告では，入力ベクトル $\mathbf{x}_n$ が辞書 $\mathbf{D}_n$ に格納される際，$\mathbf{x}_n^T\mathbf{x}_n$ の演算結果を辞書 $\mathbf{D}_n$ に保持することを提案する．この結果，カーネル演算のうち式 (26) における $\exp(\cdot)$ 内の第 3 項 $\mathbf{y}_j^T\mathbf{y}_j$ の値を，毎時刻計算することなく参照のみで得ることが可能となる．

提案するアルゴリズムは以下のようになる．

Initialization

$$\chi_1 = \mathbf{x}_1^T\mathbf{x}_1 \tag{27}$$

$$\mathbf{D}_1 = \mathbf{y}_1 = \begin{bmatrix}\mathbf{x}_1 \\ \chi_1\end{bmatrix} \tag{28}$$

$$\mathbf{h}_1 = 1 \quad , \quad \boldsymbol{\alpha}_1 = 0 \quad , \quad m = 1$$

for $n = 2, 3, \cdots$

$$\chi_n = \mathbf{x}_n^T\mathbf{x}_n \tag{29}$$

$$\mathbf{h}_n = \exp\left(\mathbf{D}_{n-1}^T\begin{bmatrix}2\mathbf{x} \\ -1\end{bmatrix} - \chi_n\mathbf{1}\right) \tag{30}$$

if $\max_{j=1,\ldots,m} |\mathbf{h}_n(j)| > \mu_0$

$$\mathbf{D}_n = \mathbf{D}_{n-1}$$

else

$$m = m + 1$$

$$\mathbf{h}_n = \begin{bmatrix}\mathbf{h}_n \\ 1\end{bmatrix} \tag{31}$$

$$\mathbf{D}_n = \mathbf{D}_{n-1} \cup \left\{\begin{bmatrix}\mathbf{x}_n \\ \chi_n\end{bmatrix}\right\} \tag{32}$$

end if

end for

ここで，$\mathbf{1}$ は要素が全て 1 である m 次のベクトルを示す．

また，提案法は式変形のみであるため，アルゴリズムの精度の劣化等は生じない．次に，演算回数を指標に提案法を評価する．図 1 に提案する辞書更新の手法を示す．

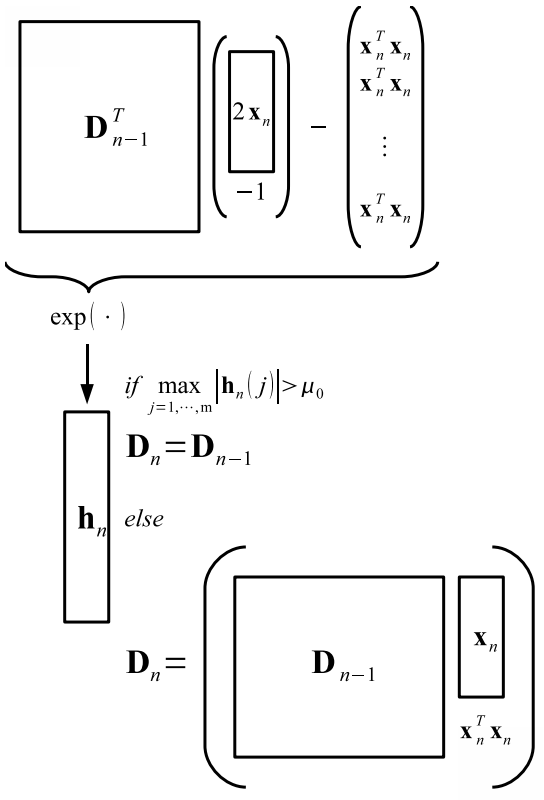

図 1: 提案する辞書更新手法

表 1: カーネル部分における提案法と従来法の演算回数

	×	+
提案法	$Mm + m + 2$	$M + 2m + 1$
従来法	Mm	$2Mm - m$

表 2: NLMS アルゴリズムと RLS アルゴリズムの演算回数

	×	+
NLMS	$3m + 2$	$3m$
RLS	$m^2 + 5m + 2$	$m^2 + 3m$

3.3 演算回数の比較

カーネル部分における，式 (11)，(12) 直接計算する手法 (以下，従来法と呼ぶ) と計算結果を保持する提案法の演算回数を比較する．従来法と提案法が毎時刻要する演算回数を表 1 に示す．

なお，表 1 には，カーネル部分の演算のみを示しており，適応アルゴリズムのための演算は含まれていない．NLMS アルゴリズムと RLS アルゴリズムが 1 時刻に要する演算回数を表 2 に示す．簡単のため，乗算と加算のウェイトは等しいものと仮定する．

いま，演算効率 $\mathcal{E}$ を

$$\text{演算効率 } \mathcal{E} = \frac{\text{提案法の演算回数}}{\text{従来法の演算回数}}$$

とする．KNLMS アルゴリズムにおける，従来法に対する提案法の演算効率を図 2 に示す．また，同様に KRLS アルゴリズムにおける演算効率を図 3 に示す．

図 2，3 において，M は入力ベクトル $\mathbf{x}_n$ の次数，m はフィルタ次数を示す．演算効率 $\mathcal{E}$ が 1 より大きい値を持つとき，演算効率が悪くなっていることを表す．一方，$\mathcal{E}$ が 1 より小さい値を持つとき，演算効率が良くなっていることを表す．

図 3 では，フィルタ次数 m が増加するほど演算効率 $\mathcal{E}$ の値が一定となることがわかる．これは，m が増加するにつれ，RLS アルゴリズムの演算量が指数関数的に増加するためである．

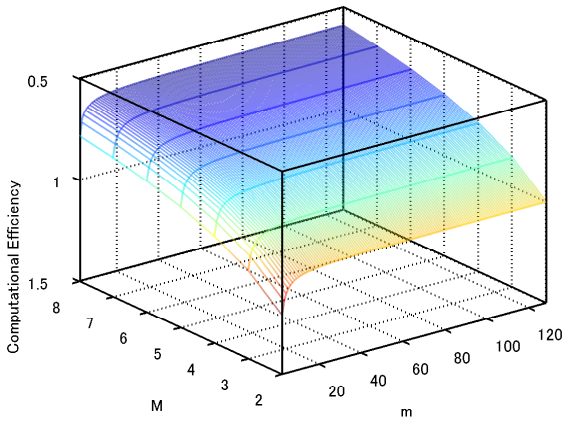

図 2: KNLMS における提案法の演算効率 $\mathcal{E}_{KNLMS}$

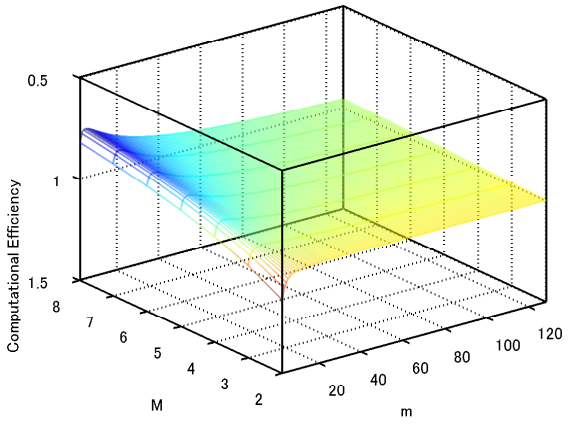

図 3: KRLS における提案法の演算効率 $\mathcal{E}_{KRLS}$

4 シミュレーション

適応予測によるシミュレーションを行った．シミュレーションを行った環境を表 3 に示す．入力信号は，初期値 x_{-1}，x_{-2} に区間 $(0, 1)$ 内で一様分布する乱数をそれぞれを与え，

$$\begin{aligned} x_n = &\left(0.8 - 0.5\exp\left(-x_{n-1}^{-2}\right)\right) x_{n-1} \\ &- \left(0.3 + 0.9\exp\left(-x_{n-1}^{2}\right)\right) x_{n-2} + 0.1\sin\left(x_{n-1}\pi\right) \end{aligned} \tag{33}$$

より生成される信号を，平均 0，分散 1 に正規化したものとする．入力ベクトルの次数 M は 4，雑音信号は SNR40dB の加法性白色ガウス雑音とした．

閾値 μ_0 は 0.8，信号長は 10000 とし，$n = 4000$ 時点で式 (33) の係数を変化させた．独立な 1000 回の試行を行い，1 回試行中の 1 時刻の演算に要した時間の平均により評価する．シミュレーションにおける 1 時刻に要した実時間を表 4 に示す．

結果より，KNLMS では約 0.73 倍，KRLS では約 0.91 倍に時間が短縮化された．なお，図 4 は収束特性を示している．このシミュレーションにおいて，フィルタ次数 m の平均は 33.9，分散は 24.5 であった．ここで，1 例として図 2，3 において，$M = 4$，$m = 34$ に該当する演算効率を見る．KNLMS アルゴリズムにおける演算効率は 0.78 倍，KRLS アルゴリズムにおける演算効率は 0.98 倍である．

表 3: シミュレーション環境

CPU	Intel Core 2 Quad 2.50 GHz
RAM	3.25 GB
シミュレーションツール	MATLAB 6.5.1 Release 13

表 4: 1 時刻に要した実時間

	KNLMS	KRLS
提案法 [μs]	149	238
従来法 [μs]	204	262
時間効率 (提案法/従来法)	0.73	0.91

5 まとめ

カーネル適応アルゴリズムにおいて，カーネル演算を変形し，過去に演算された値を辞書に保持することで，カーネル法に必要となる演算回数をほぼ半分に削減可能とする手法を提案した．またシミュレーションにより，収束特性などの精度を維持したまま，KNLMS アルゴリズムにおいては 0.73 倍，KRLS アルゴリズムにおいて 0.91 倍の時間短縮化が確認され，提案法の有効性が示された．

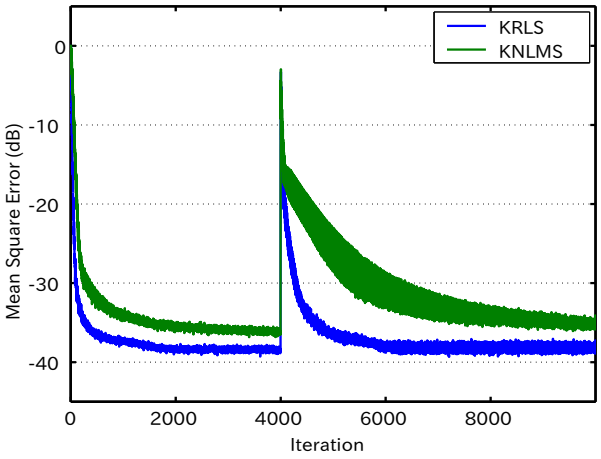

図 4: シミュレーションの収束特性

参考文献

[1] Simon Haykin, “Adaptive Filter Theory Fourth Edition,” Prentice Hall: 2001.

[2] Haiquan Zhao and Jiashu Zhang, “A Novel Adaptive Nonlinear Filter-Based Pipelined Feedforward Second-Order Volterra Architecture,” *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, VOL. 57, NO. 1, Jan., 2009

[3] Thomas M. Panicker, V. John Mathews and Giovanni L. Sicuranza, “Adaptive Parallel-Cascade Truncated Volterra Filters,” *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, VOL. 46, NO. 10, Oct., 1998

[4] Isao YAMADA, Takuya OKADA and Kohichi SAKANIWA, “A Note on Robust Adaptive Volterra Filtering Based on Parallel Subgradient Projection Techniques,” *IEICE TRANS. FUNDAMENTALS*, VOL. E86-A, NO. 8, Aug., 2003

[5] 高濱 優里, 梶川 嘉延, 野村 康雄, “NLMS 法を用いた 2 次適応 Volterra フィルタの収束特性の定式化,” **電子情報通信学会論文誌** *A*, Vol. J82-A, No. 7, pp. 944-953, Jul., 1999

[6] 木下 聡, 梶川 嘉延, 野村 康雄, “Volterra フィルタにおけるエリアジングの回避と演算量の削減,” **電子情報通信学会論文誌** *A*, Vol. J85-A, No. 1, pp. 10-16, Jan., 2002

[7] Hideyuki FURUHASHI, Yoshinobu KAJIKAWA, Yasuo NOMURA, “Linearization of Loudspeaker System Using a Subband Parallel Cascade Volterra Filter,” *IEICE TRANS. FUNDAMENTALS*, VOL. E90-A, NO. 8, Aug., 2007

[8] Cédric Richard, José Carlos M. Bermudez and Paul Honeine, “Online Prediction of Time Series Data With Kernels,” *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, VOL. 57, NO. 3, pp.1058-1067, Mar., 2009

[9] Yaakov Engel, Shie Mannor and Ron Meir, “The Kernel Recursive Least-Squares Alrorithm,” *IEEE TRANSACTIONS ON SIGNAL PROCESSING*, VOL. 52, NO. 8, pp.2275-2285, Aug., 2004

[10] Puskal P. Pokharel, Weifeng Liu and Jose C. Principe, “Kernel LMS,” *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2007

[11] 牧崎 幸司, 西川 清史, “繰り返し法によるカーネル NLMS アルゴリズムの収束特性改善,” *2010* **年電子情報通信学会総合大会**, A-4-6, Mar., 2010

[12] Nello Cristianini and John Shawe-Taylor 著, 大北 剛 訳, “サポートベクターマシン入門,” 共立出版: 2007

[13] Christopher M. Bishop 著, 元田 浩, 栗田 多喜夫, 樋口 知之, 松本 裕治, 村田 昇 監訳, “パターン認識と機械学習,” シュプリンガー・ジャパン株式会社: 2008