Esparsificação de dicionário para métodos *kernel* baseada na ortogonalização de Gram-Schmidt

André Amaro Bueno e Magno T. M. Silva

***Resumo*— Neste artigo, propõe-se uma técnica de esparsificação de dicionário para métodos baseados em núcleo (*kernel*). Ela se baseia no processo de ortogonalização de Gram-Schmidt, permitindo que os vetores do espaço de dimensão infinita induzido pelo *kernel* gaussiano sejam representados por vetores de dimensão finita. A técnica proposta é aplicada a um algoritmo do tipo LMS *(least-mean-square)*. O algoritmo resultante apresenta vantagens em termos de custo computacional quando comparado com o algoritmo *kernel* LMS que utiliza o critério da novidade como técnica de esparsificação.**

***Palavras-Chave*— métodos baseados em *kernel*, filtragem adaptativa, processamento não linear de sinais**

***Abstract*— In this paper, we propose a technique for dictionary sparsification in kernel methods. It is based on the Gram-Schmidt orthogonalization process, which allows finite-dimensional vectors to represent vectors contained in the infinite-dimensional space induced by the Gaussian kernel. The proposed technique is applied to an LMS (least-mean-square)-type algorithm. The resulting algorithm presents a lower computational cost when compared to the kernel LMS implemented with the novelty criterion.**

***Keywords*— kernel-based-methods, adaptive filtering, nonlinear signal processing**

I. Introdução

Métodos baseados em núcleo (*kernel*) são capazes de resolver problemas não lineares, projetando o vetor de entrada em um espaço de dimensão mais alta, onde um método linear é usado [1]. Esses métodos têm sido amplamente utilizados em conjunto com diferentes técnicas de processamento de sinais, como máquinas de vetores de suporte (*support vector machines* – SVM), análise de componentes principais (*principal componente analysis* – PCA), filtragem adaptativa, entre outras [1], [2]. Apesar dessas técnicas terem objetivos diferentes, todas elas mapeiam um vetor coluna $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^M$, sendo M a dimensão do espaço de entrada, em um espaço de alta dimensão $\mathcal{H}$ como $\boldsymbol{\varphi}(\mathbf{u})$, usando um *kernel* de Mercer [1]–[6].

Kernel de Mercer é uma função contínua, simétrica e positiva-definida $\kappa : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$, tal que [1]

$$\kappa(\mathbf{u}, \mathbf{v}) = \langle \boldsymbol{\varphi}(\mathbf{u}), \boldsymbol{\varphi}(\mathbf{v}) \rangle_{\mathcal{H}} \triangleq \boldsymbol{\varphi}(\mathbf{u})^T \boldsymbol{\varphi}(\mathbf{v}), \quad (1)$$

para todo $\mathbf{u}, \mathbf{v} \in \mathcal{U}$, em que o superescrito T representa transposição. Essa equação é conhecida como truque do *kernel* já que o produto interno dos vetores mapeados $\boldsymbol{\varphi}(\mathbf{u})$ e $\boldsymbol{\varphi}(\mathbf{v})$ pode ser calculado de maneira eficiente no espaço $\mathcal{H}$, sem se conhecer explicitamente a função $\boldsymbol{\varphi}(\cdot)$ [1]. O *kernel* gaussiano é o mais utilizado na literatura e é definido como

$$\kappa(\mathbf{u},\mathbf{v}) = e^{-\zeta \|\mathbf{u}-\mathbf{v}\|^2}, \quad (2)$$

André A. Bueno e Magno T. M. Silva. Depto. Eng. de Sistemas Eletrônicos, Escola Politécnica da USP. E-mails: {aabueno,magno}@lps.usp.br. Este trabalho foi financiado pelo CNPq (304715/2017-4), FAPESP (2017/20378-9) e CAPES (88882.333337/2019-01 e código de financiamento 001).

em que $\|\cdot\|$ denota a norma euclidiana, $\zeta = 1/(2\sigma^2)$ é o parâmetro do *kernel* e σ sua largura. Pelo teorema de Mercer, o *kernel* gaussiano induz um espaço de Hilbert $\mathcal{H}$ de dimensão infinita [1].

Graças ao truque da Equação (1), técnicas que utilizam produtos internos, como é o caso de filtros adaptativos, podem ter versões baseadas em *kernel*. Dentre os diferentes filtros adaptativos *kernel* da literatura, o algoritmo *kernel least-mean-squares* (KLMS) [3] é o mais popular graças à sua simplicidade. No espaço de alta dimensão $\mathcal{H}$, o algoritmo LMS é usado para atualizar o vetor coluna dos coeficientes do filtro KLMS, denotado por $\boldsymbol{\omega}(n-1)$, a fim de se estimar o sinal desejado $d(n)$. As equações do KLMS são dadas por

$$y(n) = \boldsymbol{\varphi}(\mathbf{u}(n))^T \boldsymbol{\omega}(n-1), \quad (3)$$

$$e(n) = d(n) - y(n), \quad (4)$$

$$\boldsymbol{\omega}(n) = \boldsymbol{\omega}(n-1) + \mu e(n) \boldsymbol{\varphi}(\mathbf{u}(n)), \quad (5)$$

em que $\boldsymbol{\omega}(0) = \mathbf{0}$, $\mathbf{u}(n)$ é o vetor de entrada do filtro, $y(n)$ é a sua saída, $e(n)$ representa o erro de estimação, $d(n)$ o sinal desejado e μ um passo de adaptação. Pelo Teorema da Representação, $\boldsymbol{\omega}(n-1)$ pode ser expresso como uma combinação linear das amostras do passado de modo que a saída do filtro pode ser obtida como [3]

$$y(n) = \varphi(\mathbf{u}(n))^T \boldsymbol{\omega}(n-1) = \mu \sum_{i=1}^{n-1} e(i) \kappa(\mathbf{u}(n), \mathbf{u}(i)). \quad (6)$$

O cálculo de $y(n)$ utiliza os vetores de entrada do filtro desde o instante inicial até o instante $n-1$, ou seja, os vetores do conjunto $\mathcal{D}(n) = \{\mathbf{u}(1), \mathbf{u}(2), \cdots, \mathbf{u}(n-1)\}$. Esse conjunto é denominado dicionário e sua cardinalidade é $N(n) = n-1$.

De (6), observa-se que um dos maiores problemas dos métodos baseados em *kernel* é o elevado custo computacional já que o dicionário cresce linearmente com o tempo. Diante disso, algumas técnicas de esparsificação foram propostas na literatura (ver, e.g., [1], [7], [8] e suas referências). Dentre essas técnicas, o critério da novidade é um dos mais utilizados em filtragem adaptativa baseada em *kernel*. Para decidir se o vetor $\mathbf{u}(n)$ deve ou não fazer parte no dicionário, essa técnica calcula a distância entre $\mathbf{u}(n)$ e todos os vetores do dicionário $\mathbf{u}_i$, $i = 1,2,\cdots,N(n)$. Se a distância mínima for maior que um limiar positivo L_N, ou seja, se $\min_i \|\mathbf{u}(n) - \mathbf{u}_i\| > L_N$, o vetor $\mathbf{u}(n)$ é incluído no dicionário como o elemento $\mathbf{u}_{N(n+1)}$ com $N(n+1) = N(n)+1$. Caso contrário, o dicionário e sua cardinalidade permanecem como antes, ou seja, $\mathcal{D}(n+1) \leftarrow \mathcal{D}(n)$ e $N(n+1) = N(n)$.

O critério da novidade proporciona uma redução do custo computacional de métodos baseados em *kernel*. Mesmo assim, a utilização desses métodos em problemas que exigem

soluções em tempo real, ainda está longe da realidade [5], [6]. O estudo de técnicas de esparsificação que proporcionem uma redução ainda maior no custo computacional sem causar uma degradação exagerada no desempenho desses métodos é algo de interesse. Neste trabalho, é proposta uma técnica de esparsificação que utiliza a ortogonalização de Gram-Schmidt para criar um subespaço vetorial do espaço induzido pelo *kernel* gaussiano e trabalha com as projeções dos vetores mapeados neste subespaço. A cada novo vetor de entrada $\mathbf{u}(n)$, a técnica verifica se a projeção do vetor $\boldsymbol{\varphi}(\mathbf{u}(n))$ nesse subespaço é menor que um limiar pré-definido. Se isso ocorrer, o vetor $\mathbf{u}(n)$ é inserido no dicionário e um novo vetor da base desse subespaço é calculado. Caso contrário, o dicionário e a base permanecem inalterados.

O trabalho é organizado da seguinte forma. Na Seção II, são definidos os conjuntos envolvidos e detalhadas propriedades do *kernel* gaussiano. Na Seção III, descreve-se em detalhe o procedimento para se chegar a uma base ortonormal para um subespaço do espaço induzido pelo *kernel* gaussiano a partir dos vetores de um dicionário. Na Seção IV, descreve-se como é possível criar um dicionário esparso a partir do procedimento de geração da base. Na Seção V, a técnica proposta é aplicada a um algoritmo adaptativo do tipo LMS. Na Seção VI, o algoritmo em conjunto com a técnica de esparsificação proposta é comparado com o KLMS que utiliza o critério da novidade por meio de simulações. Por fim, as conclusões são mostradas na Seção VII.

II. *Kernel* GAUSSIANO E CONJUNTOS ENVOLVIDOS

Considerando o *kernel* gaussiano de (2), o vetor mapeado $\boldsymbol{\varphi}(\mathbf{u}) \in \mathcal{H}$ pertence a uma hiperesfera de raio unitário já que $\|\boldsymbol{\varphi}(\mathbf{u})\| = \sqrt{\langle \boldsymbol{\varphi}(\mathbf{u}), \boldsymbol{\varphi}(\mathbf{u})\rangle_{\mathcal{H}}} = \sqrt{\kappa(\mathbf{u},\mathbf{u})} = 1$, $\forall\ \mathbf{u} \in \mathcal{U}$. Além disso, os vetores mapeados $\boldsymbol{\varphi}(\mathbf{u}), \boldsymbol{\varphi}(\mathbf{v}) \in \mathcal{H}$ não são ortogonais entre si, pois de (1) e (2) verifica-se que $\langle \boldsymbol{\varphi}(\mathbf{u}), \boldsymbol{\varphi}(\mathbf{v})\rangle_{\mathcal{H}} > 0$, $\forall\ \mathbf{u}, \mathbf{v} \in \mathcal{U}$. Dessa forma, o conjunto imagem $\mathcal{I}$ da função $\boldsymbol{\varphi}(\cdot)$ é um setor de uma hiperesfera de raio unitário onde não existem vetores ortogonais. Em geral, o espaço de entrada $\mathcal{U}$ não coincide com o $\mathbb{R}^M$ e portanto, há um conjunto de saída $\mathcal{S} \subset \mathcal{I}$ para o qual os vetores de $\mathcal{U}$ são mapeados. Uma das condições necessárias para que os vetores do conjunto $\mathcal{S}$ formem um espaço vetorial é que para qualquer escalar $\alpha \in \mathbb{R}$, o vetor $\alpha\boldsymbol{\varphi}(\mathbf{u})$ pertença à hiperesfera de raio unitário. Como a norma desse vetor é dada por $\|\alpha\boldsymbol{\varphi}(\mathbf{u})\| = |\alpha|\|\boldsymbol{\varphi}(\mathbf{u})\| = |\alpha|$ e vale um apenas se $|\alpha| = 1$, conclui-se que os vetores de $\mathcal{S}$ não formam um espaço vetorial. Consequentemente, não é possível encontrar uma base para $\mathcal{S}$.

Em contrapartida, é possível utilizar alguns vetores do conjunto $\mathcal{S}$ para criar uma base ortonormal de um espaço $\mathcal{B} \subset \mathcal{H}$, que contém as projeções dos vetores do conjunto $\mathcal{S}$. Na Figura 1, é mostrado o diagrama de Venn dos conjuntos envolvidos. Cabe observar que diferentemente de $\mathcal{I}$ e $\mathcal{S}$ que são conjuntos imagem, $\mathcal{B}$ é um subespaço vetorial de $\mathcal{H}$. Na figura, os espaços vetoriais são representados por linhas grossas, enquanto os conjuntos imagem por linhas finas.

III. UMA BASE ORTONORMAL PARA O ESPAÇO $\mathcal{B}$

Em um determinado instante de tempo, considerado fixo ao longo desta seção, o dicionário $\mathcal{D}$ é formado pelos vetores

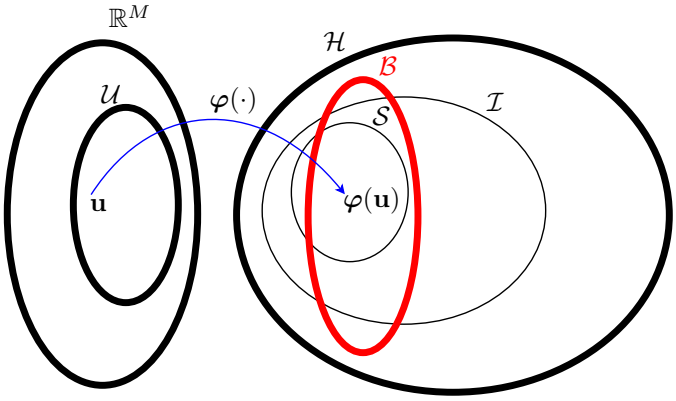

Fig. 1. Diagrama de Venn dos espaços vetoriais e conjuntos imagem envolvidos.

$\mathbf{u}_i$, $i = 1, 2, \cdots, N$. Os vetores mapeados $\boldsymbol{\varphi}(\mathbf{u}_i) \in \mathcal{S}$ podem ser utilizados para gerar uma base ortonormal $\mathbf{B}_N = \{\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \cdots, \boldsymbol{\beta}_N\}$ do espaço $\mathcal{B}$. Assim, a projeção do vetor $\boldsymbol{\varphi}(\mathbf{u}_\ell)$ em $\mathcal{B}$, denotada por $\mathbf{p}_\ell \triangleq \text{proj}_{\mathcal{B}}\boldsymbol{\varphi}(\mathbf{u}_\ell) \in \mathcal{B}$, pode ser representada por

$$\mathbf{p}_\ell = p_{\ell,1}\boldsymbol{\beta}_1 + p_{\ell,2}\boldsymbol{\beta}_2 + \cdots + p_{\ell,N}\boldsymbol{\beta}_N, \tag{7}$$

em que $p_{\ell,1}, p_{\ell,2}, \cdots, p_{\ell,N}$ são as componentes de $\mathbf{p}_\ell$ nas direções $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \cdots, \boldsymbol{\beta}_N$, respectivamente. Essas componentes podem ser agrupadas no vetor

$$\widetilde{\mathbf{p}}_\ell = [\, p_{\ell,1} \quad p_{\ell,2} \quad \cdots \quad p_{\ell,N}\,]^T. \tag{8}$$

Vale ressaltar que o vetor $\widetilde{\mathbf{p}}_\ell$ tem N elementos e é uma representação na base $\mathbf{B}_N$ do vetor $\boldsymbol{\varphi}(\mathbf{u}_\ell)$ de dimensão infinita. Com essa representação, é possível chegar a versões de métodos baseados em *kernel* diferentes das existentes na literatura. Para isso, é necessário encontrar o vetor $\widetilde{\mathbf{p}}_\ell$ para um dado vetor $\mathbf{u}_\ell$. Como mostrado a seguir, esse processo independe de se conhecer explicitamente o vetor $\boldsymbol{\varphi}(\mathbf{u}_\ell)$, o que torna essa proposta viável.

A operação de produto interno será utilizada extensivamente ao longo desta seção. Por isso, cabe lembrar as seguintes propriedades [9]:

P1: $\langle \mathbf{u}, \alpha\mathbf{v}\rangle_{\mathcal{V}} = \alpha\langle \mathbf{u}, \mathbf{v}\rangle_{\mathcal{V}}$,

P2: $\langle \mathbf{u}, \mathbf{v} + \mathbf{w}\rangle_{\mathcal{V}} = \langle \mathbf{u}, \mathbf{v}\rangle_{\mathcal{V}} + \langle \mathbf{u}, \mathbf{w}\rangle_{\mathcal{V}}$,

$\forall\ \mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathcal{V}$, $\alpha \in \mathbb{R}$, e $\mathcal{V}$ representa um espaço vetorial. Além disso, para uma base ortonormal $\mathbf{B}_N = \{\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \cdots, \boldsymbol{\beta}_N\}$ vale [9]

$$\langle \boldsymbol{\beta}_i, \boldsymbol{\beta}_j\rangle_{\mathcal{B}} = \begin{cases} 1 & se \quad i = j \\ 0 & se \quad i \neq j, \end{cases} \tag{9}$$

$i,j = 1, 2, \cdots, N$. Assim, usando (9) e as propriedades **P1** e **P2**, o produto interno dos vetores $\mathbf{p}_k$, $\mathbf{p}_\ell \in \mathcal{B}$ pode ser calculado como $\langle \mathbf{p}_k, \mathbf{p}_\ell\rangle_{\mathcal{B}} = \widetilde{\mathbf{p}}_k^T\, \widetilde{\mathbf{p}}_\ell$.

Para se obter uma base ortonormal, pode-se usar o processo de ortogonalização de Gram-Schmidt, que calcula os vetores da base a partir dos vetores de um conjunto gerador não ortogonal. Esse processo é descrito pelas equações [9]

$$\boldsymbol{\beta}_1 = \frac{\boldsymbol{\varphi}(\mathbf{u_1})}{\|\boldsymbol{\varphi}(\mathbf{u_1})\|} = \boldsymbol{\varphi}(\mathbf{u_1}), \tag{10}$$

$$\boldsymbol{\beta}_m \triangleq \frac{\boldsymbol{\gamma}_m}{\|\boldsymbol{\gamma}_m\|}, \tag{11}$$

$$\boldsymbol{\gamma}_m \triangleq \boldsymbol{\varphi}(\mathbf{u}_m) - \sum_{i=1}^{m-1} \langle \boldsymbol{\varphi}(\mathbf{u}_m), \boldsymbol{\beta}_i\rangle_{\mathcal{H}}\boldsymbol{\beta}_i, \tag{12}$$

$m = 2, 3, \cdots, N$. Os vetores da base podem ser escritos como uma combinação linear dos vetores do dicionário mapeados [9], ou seja,

$$\boldsymbol{\beta}_i = \sum_{j=1}^{N} h_{ij} \boldsymbol{\varphi}(\mathbf{u}_j), \quad i = 1, 2, \cdots, N. \tag{13}$$

De (11) e (12), observa-se que o vetor $\boldsymbol{\beta}_m$, só depende dos vetores $\boldsymbol{\varphi}(\mathbf{u}_j)$, $j = 1, 2, \cdots, m$. Assim, $h_{ij} = 0$ para $j > i$ em (13). Uma vez calculada a base, pode-se calcular a projeção do vetor $\boldsymbol{\varphi}(\mathbf{u}_\ell) \in \mathcal{S}$ no espaço $\mathcal{B}$ como

$$\mathbf{p}_\ell = \text{proj}_{\mathcal{B}} \boldsymbol{\varphi}(\mathbf{u}_\ell) = \sum_{i=1}^{N} \langle \boldsymbol{\varphi}(\mathbf{u}_\ell), \boldsymbol{\beta}_i \rangle_{\mathcal{H}} \boldsymbol{\beta}_i. \tag{14}$$

Substituindo (13) em (14) e usando as propriedades **P1** e **P2**, chega-se a

$$\mathbf{p}_\ell = \sum_{i=1}^{N} \sum_{j=1}^{N} h_{ij} \langle \boldsymbol{\varphi}(\mathbf{u}_\ell), \boldsymbol{\varphi}(\mathbf{u}_j) \rangle_{\mathcal{H}} \boldsymbol{\beta}_i. \tag{15}$$

Usando o truque do kernel, dado por (1), obtém-se

$$\mathbf{p}_\ell = \sum_{i=1}^{N} \sum_{j=1}^{N} h_{ij} \kappa(\mathbf{u}_\ell, \mathbf{u}_j) \boldsymbol{\beta}_i. \tag{16}$$

Comparando (16) com (7), conclui-se que

$$p_{\ell,i} = \sum_{j=1}^{N} h_{ij} \kappa(\mathbf{u}_\ell, \mathbf{u}_j). \tag{17}$$

Por conveniência, definem-se as matrizes triangulares inferiores

$$\mathbf{H}_i = \begin{bmatrix} \mathbf{h}_{1i} \\ \mathbf{h}_{2i} \\ \vdots \\ \mathbf{h}_{ii} \end{bmatrix} \triangleq \begin{bmatrix} h_{11} & 0 & \dots & 0 \\ h_{21} & h_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_{i1} & h_{i2} & \dots & h_{ii} \end{bmatrix} \tag{18}$$

$i = 1, 2, \cdots, N$. As linhas dessas matrizes são representadas pelos vetores $\mathbf{h}_{ki}$ de dimensão $1 \times i$ para $k \leq i$. Cabe notar que a matriz $\mathbf{H}_{i+1}$ de dimensão $(i+1) \times (i+1)$ pode ser obtida a partir da matriz $\mathbf{H}_i$, acrescentando-se um vetor coluna de zeros de dimensão i e o vetor $\mathbf{h}_{(i+1)(i+1)}$ da forma

$$\mathbf{H}_{i+1} = \begin{bmatrix} \mathbf{H}_i & \mathbf{0}_{i\times 1} \\ \mathbf{h}_{(i+1)(i+1)} \end{bmatrix}. \tag{19}$$

Consequentemente, também vale

$$\mathbf{h}_{k(i+1)} = [\mathbf{h}_{ki} \quad 0], \quad k = 1, 2, \cdots i. \tag{20}$$

Como $\boldsymbol{\beta}_1 = \boldsymbol{\varphi}(\mathbf{u}_1)$, utilizando-se a matriz $\mathbf{H}_i$, tem-se que $\mathbf{H}_1 = h_{11} = 1$. Definem-se também os vetores

$$\boldsymbol{\kappa}_i(\mathbf{u}_\ell) \triangleq [\kappa(\mathbf{u}_\ell,\mathbf{u}_1) \quad \kappa(\mathbf{u}_\ell,\mathbf{u}_2) \quad \cdots \quad \kappa(\mathbf{u}_\ell,\mathbf{u}_i)]^T, \tag{21}$$

$i = 1,2, \cdots, N$. Usando (17) em conjunto com essas definições, o vetor $\widetilde{\mathbf{p}}_\ell$ de (8) pode ser escrito como

$$\widetilde{\mathbf{p}}_\ell = \mathbf{H}_N \, \boldsymbol{\kappa}_N(\mathbf{u}_\ell). \tag{22}$$

Para completar o cálculo de $\widetilde{\mathbf{p}}_\ell$, ainda é necessário obter a matriz $\mathbf{H}_N$. Substituindo (13) em (12), usando as propriedades **P1** e **P2** e o truque do *kernel* de (1), chega-se a

$$\boldsymbol{\gamma}_m = \boldsymbol{\varphi}(\mathbf{u}_m) - \sum_{i=1}^{m-1} \sum_{j=1}^{N} h_{ij} \kappa(\mathbf{u}_m,\mathbf{u}_j) \sum_{k=1}^{N} h_{ik} \boldsymbol{\varphi}(\mathbf{u}_k). \tag{23}$$

Definindo

$$\overline{h}_{mk} \triangleq \begin{cases} 1, & m = k \\ -\displaystyle\sum_{i=1}^{m-1} \sum_{j=1}^{N} h_{ij} \, h_{ik} \, \kappa(\mathbf{u}_m,\mathbf{u}_j), & m \neq k \end{cases} \tag{24}$$

(23) pode ser reescrita como

$$\boldsymbol{\gamma}_m = \sum_{k=1}^{m} \overline{h}_{mk} \, \boldsymbol{\varphi}(\mathbf{u}_k). \tag{25}$$

Com a definição de $\overline{h}_{mk}$ em (24), o primeiro termo que aparece do lado direito de (23) foi inserido no somatório de (25). Substituindo (25) no numerador de (11) e usando (13), obtém-se

$$\boldsymbol{\beta}_m = \frac{1}{\|\boldsymbol{\gamma}_m\|} \sum_{k=1}^{m} \overline{h}_{mk} \, \boldsymbol{\varphi}(\mathbf{u}_k) = \sum_{k=1}^{m} h_{mk} \boldsymbol{\varphi}(\mathbf{u}_k). \tag{26}$$

Definindo o vetor linha

$$\overline{\mathbf{h}}_{mi} = [\, \overline{h}_{m1} \;\; \overline{h}_{m2} \;\; \cdots \;\; \overline{h}_{m(i-1)} \;\; \overline{h}_{mi}] \tag{27}$$

em que $\overline{h}_{m\ell} = 0$ para $\ell > m$, e usando (26), chega-se à relação

$$\mathbf{h}_{mi} = \overline{\mathbf{h}}_{mi} / \|\boldsymbol{\gamma}_m\|. \tag{28}$$

Lembrando que $\mathbf{H}_1 = h_{11} = 1$ e usando a relação (19), a matriz $\mathbf{H}_N$ pode ser calculada de forma iterativa. Assim, conhecendo-se $\mathbf{H}_i$, das definições (24), (21) e (18), calcula-se o vetor $\overline{\mathbf{h}}_{(i+1)(i+1)}$ como

$$\overline{\mathbf{h}}_{(i+1)(i+1)} = [\, -\boldsymbol{\kappa}_i^T(\mathbf{u}_{i+1}) \mathbf{H}_i^T \mathbf{H}_i \quad 1\,]. \tag{29}$$

Para se calcular o vetor $\mathbf{h}_{(i+1)(i+1)}$ ainda é preciso calcular $\|\boldsymbol{\gamma}_{i+1}\|$. Usando (25), as propriedades **P1** e **P2** e o truque do kernel de (1), chega-se a

$$\|\boldsymbol{\gamma}_{i+1}\| = \sqrt{\sum_{k=1}^{i+1} \sum_{\ell=1}^{i+1} \overline{h}_{(i+1)k} \, \overline{h}_{(i+1)\ell} \, \kappa(\mathbf{u}_k, \mathbf{u}_\ell)}. \tag{30}$$

Os valores de $\kappa(\mathbf{u}_k, \mathbf{u}_\ell)$, $k,\ell = 1, 2, \cdots, i+1$ podem ser armazenados na matriz quadrada $\mathbf{G}_{i+1}$ de dimensão $(i+1) \times (i+1)$, chamada gramiano. Essa matriz é simétrica, pois $g_{k\ell} = g_{\ell k} = \kappa(\mathbf{u}_\ell,\mathbf{u}_k)$, em que $g_{k\ell}$ denotam seus elementos. Além disso, os elementos de sua diagonal principal são iguais a $g_{kk} = \kappa(\mathbf{u}_k,\mathbf{u}_k) = 1$. Pode-se notar que

$$\mathbf{G}_{i+1} = \begin{bmatrix} \mathbf{G}_i & \boldsymbol{\kappa}_i(\mathbf{u}_{i+1}) \\ \boldsymbol{\kappa}_i^T(\mathbf{u}_{i+1}) & 1 \end{bmatrix}. \tag{31}$$

Trocando $\kappa(\mathbf{u}_\ell,\mathbf{u}_k)$ por $g_{\ell k}$ em (30), usando (27) e (31), obtém-se

$$\|\boldsymbol{\gamma}_{i+1}\| = \sqrt{\overline{\mathbf{h}}_{(i+1)(i+1)} \cdot \mathbf{G}_{i+1} \cdot \overline{\mathbf{h}}_{(i+1)(i+1)}^T}. \tag{32}$$

Assim, usando (28), calcula-se

$$\mathbf{h}_{(i+1)(i+1)} = \overline{\mathbf{h}}_{(i+1)(i+1)} / \|\boldsymbol{\gamma}_{i+1}\| \tag{33}$$

e de (19) obtém-se $\mathbf{H}_{i+1}$. Esse procedimento deve ser repetido até se obter $\mathbf{H}_N$.

É importante observar que não é necessário conhecer explicitamente os vetores da base $\mathbf{B}_N$ nem os vetores mapeados $\boldsymbol{\varphi}(\mathbf{u}_i)$, $i = 1, 2, \cdots, N$ para se calcular $\widetilde{\mathbf{p}}_\ell$. A seguir, esse vetor será usado em um método de esparsificação de dicionário e também para representar $\boldsymbol{\varphi}(\mathbf{u}_\ell)$ em um algoritmo adaptativo baseado em *kernel*.

IV. Esparsificação do dicionário

O procedimento descrito na Seção III pode ser usado para decidir se um vetor $\mathbf{u}(n)$ deve ou não fazer parte do dicionário $\mathcal{D}(n) = \{\mathbf{u}_1, \; \mathbf{u}_2, \; \cdots, \; \mathbf{u}_{N(n)}\}$. Para isso, utilizando (22), deve-se calcular a representação do vetor $\boldsymbol{\varphi}(\mathbf{u}(n))$ na base $\mathbf{B}_{N(n)}$, denotada por $\widetilde{\mathbf{p}}(n)$, como será visto a seguir.

Inicialmente, deve-se considerar as inicializações $\mathbf{H}_1 = 1$, $\mathbf{G}_1 = 1$ e $\widetilde{\mathbf{p}}(1) = 1$. O dicionário que será utilizado no instante

seguinte ($n = 2$) deve ser inicializado com o primeiro vetor de entrada, ou seja, $\mathcal{D}(2) = \{\mathbf{u}_1 = \mathbf{u}(1)\}$. Quando se tem o segundo vetor de entrada $\mathbf{u}(2)$, calcula-se a projeção de $\boldsymbol{\varphi}(\mathbf{u}(2))$ em $\mathbf{B}_1$ por meio da representação

$$\widetilde{\mathbf{p}}(2) = \mathbf{H}_1\, \boldsymbol{\kappa}_1(\mathbf{u}(2)) = \kappa(\mathbf{u}(2),\mathbf{u}_1).$$

Se a norma desse vetor for menor que um limiar $0 < L_{GS} < 1$ pré-definido, ou seja, se $\|\widetilde{\mathbf{p}}(2)\| < L_{GS}$, o vetor $\mathbf{u}(2)$ deve ser inserido no dicionário, fazendo $\mathcal{D}(3) = \mathcal{D}(2) \cup \{\mathbf{u}_2 = \mathbf{u}(2)\}$. Seguindo o procedimento, calcula-se $\mathbf{G}_2$, $\overline{\mathbf{h}}_{22}$, $\|\boldsymbol{\gamma}_2\|$, $\mathbf{h}_{22}$ e $\mathbf{H}_2$, utilizando as equações (31), (29), (32), (33) e (19). Com isso, tem-se implicitamente, um novo vetor $\boldsymbol{\beta}_2 \in \mathbf{B}_2$. Caso $\widetilde{\mathbf{p}}(2) \geq L_{GS}$, sua representação na base $\mathbf{B}_1$ é considerada adequada e e o dicionário permanece inalterado. Esse procedimento deve continuar para cada vetor de entrada $\mathbf{u}(n)$.

Essa técnica de esparsificação pode ser usada em conjunto com diferentes métodos baseados em *kernel*. Como exemplo, é proposto a seguir um algoritmo adaptativo do tipo LMS que considera essa forma de esparsificação.

V. O ALGORITMO GS-KLMS

Usando o procedimento descrito na Seção III, o vetor $\boldsymbol{\varphi}(\mathbf{u}(n))$ pode ser substituído por sua representação na base $\mathbf{B}_{N(n)}$, ou seja,

$$\widetilde{\mathbf{p}}(n) = \mathbf{H}_{N(n)}\, \boldsymbol{\kappa}_{N(n)}(\mathbf{u}(n)). \tag{34}$$

Usando essa representação como entrada do filtro LMS no espaço de alta dimensão, chega-se ao algoritmo

$$y(n) = \widetilde{\mathbf{p}}^T(n)\boldsymbol{\omega}(n-1), \tag{35}$$

$$e(n) = d(n) - y(n), \tag{36}$$

$$\boldsymbol{\omega}(n) = \boldsymbol{\omega}(n-1) + \mu e(n)\widetilde{\mathbf{p}}(n). \tag{37}$$

Cabe observar que no algoritmo KLMS descrito pelas equações (3)–(5), o vetor de coeficientes $\boldsymbol{\omega}$ tem dimensão infinita, considerando o *kernel* gaussiano. No entanto, ao se usar $\widetilde{\mathbf{p}}(n)$ como entrada, $\boldsymbol{\omega}$ passa a ter a dimensão do dicionário, ou seja, $N(n)$. Neste caso, como o vetor $\widetilde{\mathbf{p}}(n)$ foi calculado, é natural associar ao algoritmo (35)–(37) a técnica de esparsificação da Seção IV. O algoritmo resultante é denotado por GS-KLMS (Gram-Schmidt KLMS) e suas equações são mostradas na Tabela I. Nessa tabela, para simplificar a notação, os subscritos dos vetores e matrizes foram suprimidos. Além disso, a cardinalidade do dicionário foi denotada por N em vez de $N(n)$, mas cabe salientar que o valor de N aumenta de uma unidade quando um novo vetor de entrada é inserido em $\mathcal{D}$. Quando isso acontece, a dimensão do vetor de coeficientes do filtro também aumenta e um zero é inserido na $(N+1)$-ésima posição do vetor.

VI. RESULTADOS DE SIMULAÇÃO

Nesta seção, o algoritmo GS-KLMS da Tabela I é comparado com o algoritmo KLMS implementado de duas maneiras: (i) utilizando (6) sem nenhuma técnica de esparsificação, denotado simplesmente por KLMS, e (ii) em conjunto com o critério da novidade, denotado por NC-KLMS [1].

Considerou-se o problema de predição não linear, em que a sequência desejada é dada por [10], [11]

$$\begin{aligned} d(n) = {} & [0.8 - 0.5\exp(-d^2(n-1))]d(n-1) \\ & - [0.3 + 0.9\exp(-d^2(n-1))]d(n-2) \\ & + 0.1\mathrm{sen}(d(n-1)\pi) \end{aligned}$$

TABELA I
ALGORITMO GS-KLMS.

Entradas: $\{\mathbf{u}(n) \in \mathcal{U}, d(n)\},\ n = 1,2,\cdots$
Parâmetros: $\mu > 0$, $\zeta > 0$, $0 < L_{GS} < 1$.
Inicialização: $\mathcal{D}(2) = \{\mathbf{u}_1 = \mathbf{u}(1)\}$, $N = 1$, $y(1) = 0$, $e(1) = d(1)$, $\boldsymbol{\omega}(1) = \mu d(1)$, $\mathbf{H} = 1$, $\mathbf{G} = 1$, $\widetilde{\mathbf{p}}(1) = 1$.
Para $n = 2, 3, \cdots$, faça:

$$\boldsymbol{\kappa}(\mathbf{u}(n)) = \begin{bmatrix} \kappa(\mathbf{u}(n),\mathbf{u}_1) \\ \kappa(\mathbf{u}(n),\mathbf{u}_2) \\ \vdots \\ \kappa(\mathbf{u}(n),\mathbf{u}_N) \end{bmatrix}$$

$\widetilde{\mathbf{p}}(n) = \mathbf{H}\boldsymbol{\kappa}(\mathbf{u}(n))$
$y(n) = \widetilde{\mathbf{p}}^T(n)\boldsymbol{\omega}(n-1)$
$e(n) = d(n) - y(n)$
Se $\|\widetilde{\mathbf{p}}(n)\| < L_{GS}$, **faça:**
$N \leftarrow N + 1$
$\mathcal{D}(n+1) = \mathcal{D}(n) \cup \{\mathbf{u}_N = \mathbf{u}(n)\}$

$$\mathbf{G} \leftarrow \begin{bmatrix} \mathbf{G} & \boldsymbol{\kappa}(\mathbf{u}(n)) \\ \boldsymbol{\kappa}^T(\mathbf{u}(n)) & 1 \end{bmatrix}$$

$$\overline{\mathbf{h}} = \begin{bmatrix} -\boldsymbol{\kappa}^T(\mathbf{u}(n))\,\mathbf{H}^T\mathbf{H} & 1 \end{bmatrix}$$

$$\|\boldsymbol{\gamma}\| = \sqrt{\overline{\mathbf{h}}\mathbf{G}\overline{\mathbf{h}}^T}$$

$$\mathbf{H} \leftarrow \begin{bmatrix} \mathbf{H} & \mathbf{0}_{N-1\times 1} \\ \overline{\mathbf{h}}/\|\boldsymbol{\gamma}\| \end{bmatrix}$$

$$\boldsymbol{\omega}(n) = \begin{bmatrix} \boldsymbol{\omega}(n-1) \\ 0 \end{bmatrix} + \mu e(n) \begin{bmatrix} \widetilde{\mathbf{p}}(n) \\ 0 \end{bmatrix}$$

Senão
$\mathcal{D}(n+1) = \mathcal{D}(n)$
$\boldsymbol{\omega}(n) = \boldsymbol{\omega}(n-1) + \mu e(n)\widetilde{\mathbf{p}}(n)$
fim
fim

para $0 < n \leq N_{\mathrm{it}}/2$ e

$$\begin{aligned} d(n) = {} & [0.2 - 0.7\exp(-d^2(n-1))]d(n-1) \\ & - [0.8 + 0.8\exp(-d^2(n-1))]d(n-2) \\ & + 0.2\mathrm{sen}(d(n-1)\pi) \end{aligned}$$

para $N_{\mathrm{it}}/2 < n \leq N_{\mathrm{it}}$ em que N_{it} é o número de iterações e $d(n) = 0.1$ para $n \leq 0$. Essa mudança abrupta de $d(n)$ em $n = N_{\mathrm{it}}$ é interessante para verificar uma nova convergência dos algoritmos. Adotaram-se $N_{\mathrm{it}} = 10^4$, $M = 2$, $\zeta = 1$ e $\mu = 0{,}5$ para todos os algoritmos. Além disso, considerou-se o horizonte de predição igual a 1, ou seja, o objetivo é predizer $d(n)$ dados os valores $d(1), d(2), \cdots, d(n-1)$. Os limiares L_N e L_{GS} foram ajustados para que os algoritmos NC-KLMS e GS-KLMS tivessem desempenhos similares em termos de erro quadrático médio em regime no primeiro trecho de predição, o que levou respectivamente a $L_N = 0.0075$ e $L_{GS} = 0.9$.

O erro quadrático ao longo das iterações para os três algoritmos é mostrado na Figura 2. O algoritmo KLMS apresenta o menor erro, pois inclui todos os vetores de entrada no dicionário. O NC-KLMS converge com a mesma velocidade do KLMS e é ligeiramente mais rápido que o GS-KLMS no primeiro trecho de predição. Os dois algoritmos que utilizam esparsificação do dicionário atingem aproximadamente o mesmo erro quadrático em regime, que é cerca de 4 dB mais alto que o do KLMS. Após a mudança abrupta em $d(n)$, os três algoritmos convergem com a mesma velocidade e a partir de $n = 8\times 10^3$, o GS-KLMS apresenta um erro quadrático menor que o do NC-KLMS e consequentemente, mais próximo do erro do KLMS.

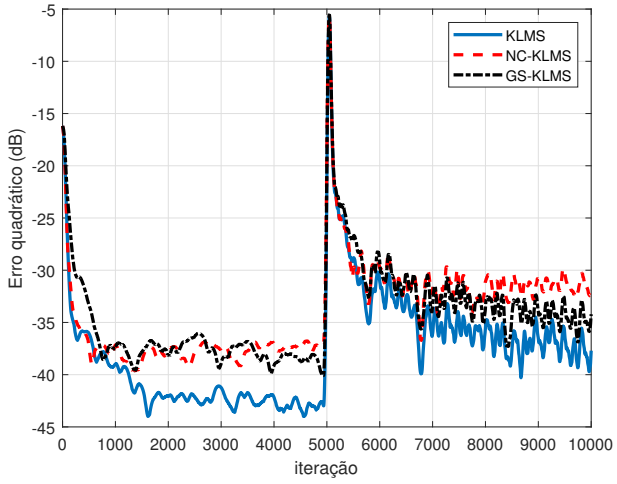

Fig. 2. Erro quadrático ao longo das iterações para KLMS, NC-KLM ($L_N = 0.0075$) e GS-KLMS ($L_{GS} = 0.9$); $M = 2$, $\zeta = 1$ e $\mu = 0{,}5$.

As cardinalidades dos dicionários dos algoritmos NC-KLMS e GS-KLMS são mostradas na Figura 3. No primeiro trecho de predição, a cardinalidade do dicionário do GS-KLMS chegou a 11, sendo que o 11º foi incluído na iteração $n = 48$. Em contrapartida, a cardinalidade do dicionário do NC-KLMS chegou a aproximadamente 700 vetores, incluídos ao longo das primeiras 10^3 iterações. No segundo trecho, a cardinalidade do dicionário do GS-KLMS chegou a 62, valor atingido 310 iterações depois da mudança abrupta em $d(n)$. Nesse trecho, a cardinalidade do dicionário do NC-KLMS chegou a aproximadamente 1700.

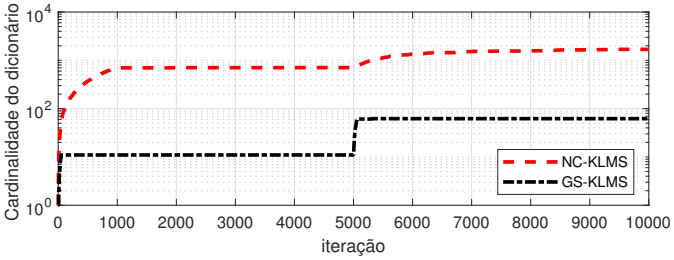

Fig. 3. Cardinalidade do dicionário ao longo das iterações para NC-KLMS ($L_N = 0.0075$) e GS-KLMS ($L_{GS} = 0.9$); $M = 2$, $\zeta = 1$ e $\mu = 0{,}5$.

Para comparar o custo computacional do GS-KLMS com o do NC-KLMS, foram computados os números de multiplicações e exponenciais acumuladas realizadas por esses algoritmos ao longo das iterações. Esses números são mostrados na Figura 4. Como se tratam de operações acumuladas, os números sempre aumentam ao longo das iterações. No GS-KLMS, considerou-se que a multiplicação de uma matriz triangular inferior de dimensão $N \times N$ por um vetor de dimensão N realiza $N(N+1)/2$ multiplicações. É possível observar que a economia em termos de curso computacional para o algoritmo GS-KLMS é expressiva. Comparando esses algoritmos na iteração $n = 10^4$, por exemplo, o GS-KLMS necessita cerca de 10^7 menos multiplicações acumuladas e de $9{,}6 \times 10^6$ menos exponenciais acumuladas que o NC-KLMS.

VII. Conclusões

Métodos baseados em *kernel* têm sido amplamente utilizados para resolver problemas não lineares em processamento de

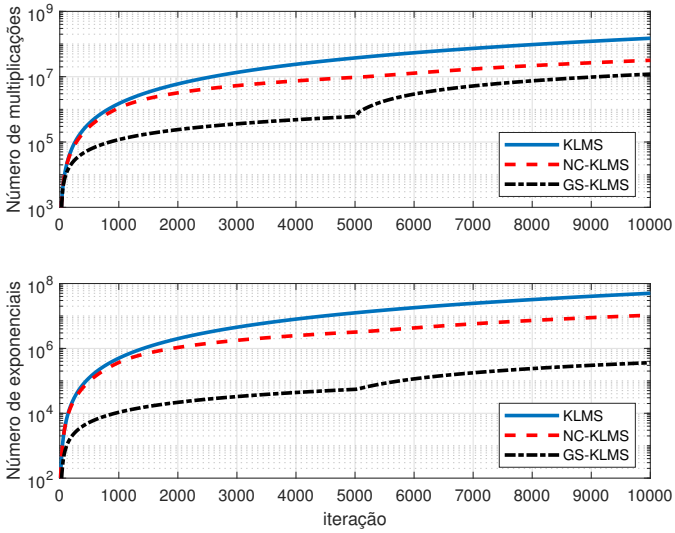

Fig. 4. Número de operações acumuladas para NC-KLMS ($L_N = 0.0075$) e GS-KLMS ($L_{GS} = 0.9$); $M = 2$, $\zeta = 1$ e $\mu = 0{,}5$.

sinais e aprendizado de máquina. Um dos maiores problemas relativos a esses métodos é a quantidades de vetores de entrada que são inseridos no dicionário. Neste artigo, um método baseado no processo de ortogonalização de Gram-Schmidt foi proposto e utilizado em um algoritmo de filtragem adaptativa do tipo LMS. Por meio de simulação de um problema de predição não linear, verificou-se que o algoritmo proposto consegue realizar menos cálculos e mantém menos vetores no dicionário quando comparado ao NC-KLMS para um mesmo desempenho em termos de erro quadrático em regime. Em um trabalho futuro, pretende-se considerar outras aplicações e buscar uma análise estocástica do algoritmo a fim de encontrar uma forma adequada de escolher o parâmetro L_{GS}.

Referências

[1] W. Liu, J. C. Príncipe, and S. Haykin, *Kernel Adaptive Filtering: A Comprehensive Introduction*, Wiley, 2010.

[2] B. Scholkopf and A. J. Smola, *Learning with Kernels: support vector machines, regularization, optimization, and beyond*, MIT Press, Cambridge, 2002.

[3] W. Liu, P. P. Pokharel, and J. C. Principe, "The kernel least-mean-square algorithm," *IEEE Trans. Signal Process.*, vol. 56, pp. 543–554, Feb. 2008.

[4] S. Garcia-Vega, X.-J. Zeng, and J. Keane, "Learning from data streams using kernel least-mean-square with multiple kernel-sizes and adaptive step-size," *Neurocomputing*, vol. 339, pp. 105–115, 2019.

[5] Y. Liu, C. Sun, and S. Jiang, "A reduced Gaussian kernel least-mean-square algorithm for nonlinear adaptive signal processing," *Circuits, Systems, and Signal Process.*, vol. 38, pp. 371–394, Jan. 2019.

[6] A. Flores and R. C. de Lamare, "Set-membership adaptive kernel NLMS algorithms: design and analysis," *Signal Process.*, vol. 154, pp. 1–14, 2019.

[7] C. Richard, J. C. M. Bermudez, and P. Honeine, "Online prediction of time series data with kernels," *IEEE IEEE Trans. Signal Process.*, vol. 57, pp. 1058–1067, Mar. 2009.

[8] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Trans. Signal Process.*, vol. 52, pp. 2275–2285, Aug. 2004.

[9] Carl D Meyer, *Matrix analysis and applied linear algebra*, vol. 2, Siam, 2000.

[10] M. Yukawa, "Multikernel adaptive filtering," *IEEE Trans. Signal Process.*, vol. 60, pp. 4672–4682, Sep. 2012.

[11] M. T. M. Silva, R. Candido, J. Arenas-García, and J. A. Azpicueta-Ruiz, "Improving multikernel adaptive filtering with selective bias," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Process.*, Apr. 2018, pp. 4529–4533.