Efficient and Stable Algorithms to Extend Greville’s Method to Partitioned Matrices Based on Inverse Cholesky Factorization

Hufei Zhu

Abstract—Greville’s method has been utilized in (Broad Learning System) BLS to propose an effective and efficient incremental learning system without retraining the whole network from the beginning. For a column-partitioned matrix where the second part consists of \( p \) columns, Greville’s method requires \( p \) iterations to compute the pseudoinverse of the whole matrix from the pseudoinverse of the first part. The incremental algorithms in BLS extend Greville’s method to compute the pseudoinverse of the whole matrix from the pseudoinverse of the first part by just 1 iteration, which have neglected some possible cases, and need further improvements in efficiency and numerical stability. In this paper, we propose an efficient and numerical stable algorithm from Greville’s method, to compute the pseudoinverse of the whole matrix without retraining the whole network from the pseudoinverse of the first part by just 1 iteration, where all possible cases are considered, and the recently proposed inverse Cholesky factorization can be applied to further reduce the computational complexity.

In subsection A, we introduce Greville’s method [2] and its application to column-partitioned matrices in BLS. In subsection B, we deduce three theorems that will be utilized. Then in subsection C, we propose the modified Greville’s method for BLS, which considers all possible cases, and is improved in efficiency and numerical stability. Finally in subsection D, we apply the efficient inverse Cholesky factorization in [3] to further reduce the computational complexity.

II. THE PROPOSED IMPROVEMENTS FOR GREVILLE’S METHOD UTILIZED IN BLS

In subsection A, we introduce Greville’s method [2] and its application to column-partitioned matrices in BLS.

As in [1], write the \( m \times (n + p) \) column-partitioned matrix as

\[
A^{m+1} = \begin{bmatrix} A^m & H_{m+1} \end{bmatrix},
\]

where \( A^m \) is \( m \times n \) and \( H_{m+1} \) is \( m \times p \). Let

\[
A_k^{m+1} = \begin{bmatrix} A^m & H_{m+1}^{1:k} \end{bmatrix},
\]

where \( H_{m+1}^{1:k} \) denotes the first \( k \) columns of \( H_{m+1} \). Then

\[
A_k^{m+1} = \begin{bmatrix} A_{k-1}^{m+1} & h_{m+1}^k \end{bmatrix},
\]

where \( h_{m+1}^k \) denotes the \( k \)-th column of \( H_{m+1} \). Notice that when \( k = 0 \), \( H_{m+1}^{1:k} \) becomes empty and then (2) becomes

\[
A_0^{m+1} = A^m.
\]

Greville’s method [2] computes \( (A_p^{m+1})^+ = (A^{m+1})^+ \) from \( (A_0^{m+1})^+ = (A^m)^+ \) by \( p \) iterations. In the \( k \)-th iteration \((k = 1, 2, \ldots, p)\), \( (A_k^{m+1})^+ \) is [2] Theorem 5.7

\[
\begin{bmatrix} A_{k-1}^{m+1} & h_{m+1}^k \end{bmatrix}^+ = \left( \begin{bmatrix} A_{k-1}^{m+1} & h_{m+1}^k \end{bmatrix} - \tilde{d}_k \tilde{b}_k^T \right)^+,
\]

where

\[
\tilde{d}_k = (A_{k-1}^{m+1})^+ h_{k+1}^m,
\]

and \( \tilde{b}_k^T \) is computed from

\[
\tilde{c}_k = b_{k+1}^m - A_{k-1}^{m+1} \tilde{d}_k
\]

by

\[
\tilde{b}_k^T = \begin{cases} \tilde{c}_k^T & \text{if } \tilde{c}_k \neq 0 \\ (1 + \tilde{d}_k^T \tilde{d}_k)^{-1} \tilde{d}_k (A_{k-1}^{m+1})^+ & \text{if } \tilde{c}_k = 0 \end{cases}
\]
Here $A^+$ is the unique Moore-Penrose generalized inverse (i.e., the pseudoinverse) that satisfies \[ (9a) \]

\[
\begin{align*}
AA^+A &= A \\
A^+AA^+ &= A^+ \\
(AA^+)^T &= A^+ \\
(A^+A)^T &= A^+A.
\end{align*}
\]

\[ (9b) \]

\[ (9c) \]

\[ (9d) \]

In (11), the column vector $h_{m+1}^{m+1}$ in (5) is extended to the matrix $H_{m+1}$ with $p$ columns, and correspondingly (5), (6), (7) and (8) become

\[ (10) \]

\[
(A^{m+1})^+ = [A^mH_{m+1}]^+ = \begin{bmatrix} (A^m)^+ - DB^T \\
B^T \end{bmatrix},
\]

\[ (11) \]

\[ D = (A^m)^+H_{m+1}, \]

\[ (12) \]

\[
C = H_{m+1} - A^mD,
\]

respectively.

$C \neq 0$ is required in (13a), while $\tilde{c}_k \neq 0$ is required in (5a). Denote the $k$-th column of $C$ as $c_k$. In the next paragraph we will show that usually $c_k$ is different from $\tilde{c}_k$. Then it can be easily seen that $A_k$ cannot be extended to (13a), since $c_k \neq 0$ for each $k$ ($1 \leq k \leq p$) in (13) cannot ensure $\tilde{c}_k \neq 0$ in (5a), and actually $C \neq 0$ only means at least one $c_k \neq 0$.

To show that usually $c_k$ is different from $\tilde{c}_k$, substitute (6) into (7) to obtain

\[ (14) \]

\[
\tilde{c}_k = h_k^{m+1} - A_k^{m+1}(A_k)^+h_k^{m+1},
\]

and substitute (11) into (12) to obtain

\[ (15) \]

\[
C = H_{m+1} - A^m(A^m)^+H_{m+1},
\]

from which we can deduce

\[ (16) \]

\[
c_k = h_k^{m+1} - A^m(A^m)^+h_k^{m+1}.
\]

From (16) and (14), it can be seen that only \[ (17) \]

\[
\tilde{c}_1 = c_1,
\]

and for $k = 2, 3, \cdots, p$, usually \[ (18) \]

\[
\tilde{c}_k \neq c_k.
\]

since usually

\[ (19) \]

\[
A^m(A^m)^+ \neq A_k^{m+1}(A_k^{m+1})^+.
\]

### B. Three Theorems about $\tilde{c}_k$

In this subsection, we deduce three theorems relevant to $\tilde{c}_k$.

Firstly about (19), we have

**Theorem 1.** If $\tilde{c}_k = 0$, then

\[ (20) \]

\[
A_k^{m+1}(A_k^{m+1})^+ = A_k^{m+1}(A_k^{m+1})^+ + .
\]

**Proof.** Applying (5) to obtain $A_k^{m+1}(A_k^{m+1})^+ = [A_k^{m+1}h_k^{m+1}] + (A_k^{m+1})^+ - \tilde{c}_k\tilde{b}_k^T$, i.e.,

\[ (21) \]

\[
A_k^{m+1}(A_k^{m+1})^+ = A_k^{m+1}(A_k^{m+1})^+ + - A_k^{m+1}\tilde{c}_k\tilde{b}_k^T + h_k^{m+1}\tilde{b}_k^T.
\]

Now we only need to verify that the last two entries in the right side of (21) satisfy

\[ (22) \]

\[
-A_k^{m+1}\tilde{c}_k\tilde{b}_k^T + h_k^{m+1}\tilde{b}_k^T = 0.
\]

Since $\tilde{c}_k = 0$, from (14) we can deduce

\[ (23) \]

\[
A_k^{m+1}(A_k^{m+1})^+h_k^{m+1} = h_k^{m+1},
\]

into which we substitute (6) to obtain

\[ (24) \]

\[
A_k^{m+1}\tilde{c}_k = h_k^{m+1}.
\]

Then we can substitute (24) into (22) to verify (22).

On the other hand, notice that $\tilde{c}_k = 0$ is equivalent to the last 3rd and 4th rows in page 166] to

\[ (25) \]

\[
\Re(A^{m+1}) = \Re(A_k^{m+1}),
\]

where

\[ (26) \]

\[
\Re(A) = \{ y \in \mathbb{R}^n : y = Ax \text{ for some } x \in \mathbb{R}^k \}
\]

is the range of any $A \in \mathbb{R}^{n \times k}$.

From Theorem 1, we derive

**Theorem 2.** If

\[ (27) \]

\[
\tilde{c}_k = \tilde{c}_{k-1} = \cdots = \tilde{c}_1 = 0,
\]

then

\[ (28) \]

\[
\tilde{c}_{k+1} = c_{k+1}.
\]

**Proof.** Apply (20) iteratively to obtain $A_k^{m+1}(A_k^{m+1})^+ = A_k^{m+1}(A_k^{m+1})^+ = A_k^{m+1}(A_k^{m+1})^+ = \cdots = A_0^{m+1}(A_0^{m+1})^+$, which can be substituted into (14) to deduce

\[ (29) \]

\[
\tilde{c}_{k+1} = h_k^{m+1} - A_0^{m+1}(A_0^{m+1})^+h_k^{m+1}.
\]

Finally we can substitute (29) into (16) to deduce (28).

Since the condition $C = 0$ in (13b) is equivalent to

\[ (30) \]

\[
c_1 = c_2 = \cdots = c_p = 0,
\]

we can deduce $\tilde{c}_1 = c_1 = 0$ from (17), and then we can apply Theorem 2 and (30) iteratively to deduce $\tilde{c}_2 = c_2 = 0$, $\tilde{c}_3 = c_3 = 0$, $\cdots$, and $c_p = c_p = 0$. Correspondingly we have

**Theorem 3.** If $C = 0$, i.e., (30) satisfies, then

\[ (31) \]

\[
\tilde{c}_1 = \tilde{c}_2 = \cdots = \tilde{c}_p = 0.
\]

Since $\tilde{c}_k = 0$ is equivalent to (25), Theorem 3 is also equivalent to: if $C = 0$, then

\[ (32) \]

\[
\Re(A_k^{m+1}) = \Re(A_k^{m+1}) = \cdots = \Re(A_0^{m+1}).
\]
C. Modified Greville’s method for BLS Considering All Possible Cases and Improved in Efficiency and Numerical Stability

From Theorem 3, it can be seen that when the condition in (13a) satisfies, the condition in (8a) also satisfies for \( k = 1, 2, \cdots, p \). Then (33) can be applied to compute \( \tilde{b}_i^T, \tilde{b}_j^T, \cdots, \) and \( \tilde{b}_p^T \). Correspondingly, (33) can be extended to (13b).

To improve the numerical stability and reduce the computational complexity, substitute (11) into (13b) to obtain

\[
B^T = (I + D^T (A^m)^T H_{m+1})^{-1} D^T (A^m)^T,
\]

which can be written as

\[
B^T = (I + \tilde{D} H_{m+1})^{-1} \tilde{D}
\]

where

\[
\tilde{D} = D^T (A^m)^T.
\]

Then we can utilize equation (20) in [41], i.e.,

\[
(I + PQ)^{-1}P = P(I + QP)^{-1},
\]

to deduce

\[
B^T = \tilde{D}(I + H_{m+1} \tilde{D})^{-1}
\]

from (34).

To compute \( B^T \) by (13a), obviously the condition in (8a) (i.e., \( \tilde{c}_k \neq 0 \)) should be satisfied for all \( k = 1, 2, \cdots, p \). This condition is much stronger than the above-described condition of at least one \( \tilde{c}_k \neq 0 \). Thus in (13a), “if \( C \neq 0 \)” should be modified into “if each \( \tilde{c}_k \neq 0 \) \( (1 \leq k \leq p) \)”, and it is required to consider the condition of only \( (1 \leq i < p - 1) \) \( \tilde{c}_{i:k} \) satisfying \( \tilde{c}_k = 0 \), i.e., \( C \neq 0 \) but several \( \tilde{c}_k = 0 \).

Now we can modify (13) into

\[
B^T = \begin{cases} 
(I + D^T D)^{-1} \tilde{D} & \text{if } C = 0, m \geq \max(n, p) \\
(I + \tilde{D} H_{m+1})^{-1} \tilde{D} & \text{if } C = 0, n \geq m \geq p \\
\tilde{D}(I + H_{m+1} \tilde{D})^{-1} & \text{if } C = 0, m \leq p \\
C & \text{if each } \tilde{c}_k \neq 0 (1 \leq k \leq p) \\
\cdots & \text{if } C \neq 0 \text{ but several } \tilde{c}_k = 0,
\end{cases}
\]

where the original Greville’s method [2] can be utilized to compute \( (A^m)^+ \) from \( (A^m_+)^+ = (A^m)^+ \) by \( p \) iterations, if the condition for (38c) is satisfied.

D. To Apply the Recently Proposed Inverse Cholesky Factorization to Compute All \( \tilde{c}_{i:k} \)s Efficiently

In (38d) and (38e), all \( p \tilde{c}_{i:k} \) \( (k = 1, 2, \cdots, p) \) are required. If they are computed by (14) in \( p \) iterations, \( p - 1 \) \( (A^m_+)^+ \)s \( (k = 2, 3, \cdots, p) \) also need to be computed by (5) in \( p - 1 \) iterations, and then actually it is no longer required to apply (10), (11), (12) and (13) once to compute \( (A^m_+)^+ \) from \( (A^m_+)^+ \) directly. Thus in what follows, we will propose an efficient algorithm to compute all \( p \tilde{c}_{i:k} \)s efficiently, which is based on the recently proposed efficient inverse Cholesky factorization [3], and does not require the above-mentioned \( p - 1 \) \( (A^m_+)^+ \)s \( (k = 2, 3, \cdots, p) \).

To apply the efficient inverse Cholesky factorization [3], firstly let us derive

**Theorem 4.** If each \( \tilde{c}_{i:k} \neq 0 \) \( (1 \leq k \leq p) \), then \( C^T C \) is positive definite.

Proof. If each \( \tilde{c}_{i:k} \neq 0(1 \leq k \leq p) \), the condition in (38e) is satisfied, and then we can utilize the computation in (38d), which can be written as [1]

\[
B^T = C^+ = (C^T C)^{-1} C^T.
\]

Since \( (C^T C)^{-1} \) exists, \( C \) must be full column rank, and then \( C^T C \) must be positive definite [5].

From Theorem 4, it can be seen that if each \( \tilde{c}_{i:k} \neq 0(1 \leq k \leq p) \), there exists [5] the Cholesky factor of the positive definite \( C^T C \), i.e., the lower-triangular \( \Omega \) that satisfies

\[
\Omega^T = C^T C,
\]

from which we can deduce

\[
\Omega^{-1} = (C^T C)^{-1}.
\]

From (41) it can be seen that the upper-triangular \( \Omega^{-T} \) is the inverse Cholesky factor [3] of \( C^T C \), which can be denoted as

\[
G = \Omega^{-T}.
\]

To obtain \( G \), we can utilize the efficient Cholesky factorization proposed in [3] to compute \( G_k \) from \( G_{k-1} \) iteratively for \( k = 2, 3, \cdots, p \), by equation (11) in [3], i.e.,

\[
G_k = \begin{bmatrix} G_{k-1} & \tilde{g}_k \\ 0_{k-1}^T & g_{kk} \end{bmatrix},
\]

where

\[
g_{kk} = 1/\sqrt{c_{i:k}^T c_{i:k} - c_{i:k}^T C_{k-1} G_{k-1}^T C_{k-1}^T c_{i:k}},
\]

and

\[
\tilde{g}_k = -g_{kk} G_{k-1} G_{k-1}^T C_{k-1}^T c_{i:k}.
\]

Notice that equations (44) and (45) are derived from equations (3) and (17) in [3].

Now let us consider the case that \( \tilde{c}_{i:k} \neq 0 \) is satisfied for only \( i = 1, 2, \cdots, k \), where \( k < p \). According to Theorem 4, we can conclude that \( C_{k:k}^T C_k \) is positive definite, where \( C_k \) denotes the first \( k \) columns of \( C \). With the positive definite \( C_k^T C_k \), the upper-triangular \( G_k \) in (43) can be computed by [3]

\[
G_k G_k^T = (C_k^T C_k)^{-1},
\]
and $B_k^T$ can be computed by $B_k^T = C_k^+ = (C_k^T C_k)^{-1} C_k^T$ (i.e., (49)), into which substitute (46) to obtain

$$B_k^T = C_k^+ = G_k C_k^T C_k^T.$$  
(47)

In (47), $C_k$ can be computed by (12), i.e.,

$$C_k = H_m^{1:k} A^m D_k,$$  
(48)

where $D_k$ is computed by (11), i.e.,

$$D_k = (A^m)^+ H_m^{1:k}.$$  
(49)

$B_k$ and $D_k$ can be applied to compute $(A_k^{m+1})^+$ by (10), i.e.,

$$(A_k^{m+1})^+ = [A^m | H_m^{1:m+1}]^+ = [(A^m)^T - D_k B_k^T]^T.$$  
(50)

In the above case of $\tilde{c}_i \neq 0$ for $i = 1, 2, \ldots, k-1$, the corresponding $C_{k-1}$ and $G_{k-1}$ can be utilized to compute $\tilde{c}_k$ and its squared length efficiently, as shown in the following Theorem 5. The proof of Theorem 5 is given in Appendix A.

**Theorem 5.** When $\tilde{c}_i \neq 0$ for all $1 \leq i \leq k-1$, $\tilde{c}_k (k = 2, 3, \ldots, p)$ defined by (14) is equal to

$$\tilde{c}_k = c_k - C_{k-1} G_{k-1} C_{k-1}^T c_{k-1},$$  
(51)

and the squared length of $\tilde{c}_k$ can be computed by

$$|\tilde{c}_k|^2 = \tilde{c}_k C_{k-1} G_{k-1}^T C_k c_k.$$  
(52)

It can be seen that the pseudoinverse $(A_{k-1}^{m+1})^+$ utilized in (14) to compute $\tilde{c}_k$ is no longer required in (51) and (52). Moreover, we can substitute (52) into (44) to compute $g_{kk}$ by

$$g_{kk} = 1/\sqrt{|\tilde{c}_k|^2} = 1/\sqrt{\tilde{c}_k^T \tilde{c}_k}.$$  
(53)

With Theorem 5, we can prove the Inverse Proposition of Theorem 4, i.e.,

**Theorem 6.** If $C^T C$ is positive definite, then each $\tilde{c}_k (1 \leq k \leq p)$ defined by (14) satisfies $\tilde{c}_k \neq 0$.

Proof. If $C^T C$ is positive definite, there exists $G_k$ the Cholesky factor of $C^T C$, i.e., $\Omega$ satisfying (40), and then there also exists the inverse Cholesky factor of $C^T C$, i.e., $G$ satisfying (42). Accordingly, we can compute $G_k$ from $G_{k-1}$ iteratively for $k = 2, 3, \ldots, p$ by (53), (54) and (43), and the initial $G_1$ can be computed by

$$G_1 = 1/\sqrt{c_1^T c_1}.$$  
(54)

that is deduced from (46). From (54) we obtain $c_1 \neq 0$, from which and (17) we deduce $c_1 \neq 0$. Moreover, from (55) we deduce $\tilde{c}_k \neq 0$ ($2 \leq k \leq p$) for $\tilde{c}_k$ computed by (51). Since $\tilde{c}_1 \neq 0$, from Theorem 5 we can deduce that $\tilde{c}_2$ defined by (14) is equal to $\tilde{c}_2$ computed by (51), and then is not zero. Similarly, we can apply Theorem 5 iteratively to deduce that $\tilde{c}_k$ defined by (14) is equal to $\tilde{c}_k$ computed by (51), and then is not zero, for $k = 3, 4, \ldots, p$.

Now from Theorem 4 and Theorem 6, it can be seen that the condition of $C^T C$ being positive definite is equivalent to the condition of each $\tilde{c}_k \neq 0$ ($1 \leq k \leq p$), where $\tilde{c}_k$ is defined by (14). Then we can write (38) as

$$(I + D_k^T D_k)^{-1} \tilde{D} if C = 0, m \geq max(n, p)$$

(55a)

$$(I + \tilde{D} H_{m+1}^{-1})^{-1} \tilde{D} i f C = 0, n \geq m \geq p$$

(55b)

$$(I + \tilde{D} H_{m+1}^{-1})^{-1} \tilde{D} i f C = 0, m \leq p$$

(55c)

$C^+$ if $C$ is positive definite

(55d)

$\ldots$ if $C \neq 0$ and $C^T C$ is not positive definite .

(55e)

On the other hand, the condition of $C$ being full column rank is equivalent to the condition of $C^T C$ being positive definite [5], and then is also equivalent to the condition of each $\tilde{c}_k \neq 0$ ($1 \leq k \leq p$). Then we can also write (38) as

$$B^T = \begin{cases} (I + D_k^T D_k)^{-1} \tilde{D} if C = 0, m \geq max(n, p) \\
(I + \tilde{D} H_{m+1}^{-1})^{-1} \tilde{D} if C = 0, n \geq m \geq p \\
(I + \tilde{D} H_{m+1}^{-1})^{-1} \tilde{D} if C = 0, m \leq p \\
C^+ if C is full column rank \\
\ldots if C \neq 0 is not full column rank. \end{cases}$$

(56a)

(56b)

(56c)

(56d)

(56e)

III. THE PROPOSED ALGORITHM FOR COLUMN-PARTITIONED MATRICES IN BLS

The algorithm for the pseudoinverse of a column-partitioned matrix is described in Algorithm 1. In Algorithm 1, there is a while loop including all 26 rows except row 1, of which the first iteration will be introduced in what follows.

The index $i$ for the while loop denotes that the pseudoinverse $A_{k+1}^{m+1} = [A^m | H_m^{1:m+1}]$ (defined by (2)) is available, and the initial $i$ is set to 0 in row 1. In row 3, $D$ and $C$ are computed by (11) and (12), respectively. Then in row 4, the function $[k, G_k] = \text{InvChol}(C, \lambda)$ defined in Algorithm 2 is applied to find the minimum $k \geq 0$ satisfying $\tilde{c}_{k+1} = 0$ (i.e., $|\tilde{c}_{k+1}|^2 < \varepsilon$ where $\varepsilon \rightarrow 0$ is a positive number near zero, e.g., $\varepsilon = 10^{-10}$) and the corresponding $k \times k$ inverse Cholesky factor $G_k$ satisfying (46), or find the $k$ equal to the column number of $C$ and the corresponding $G_k$. In Algorithm 2, $|\tilde{c}_k|^2$ is computed by (52) for $k = 1, 2, \ldots, p$, until the first $|\tilde{c}_k|^2 = 0$ (i.e., $|\tilde{c}_k|^2 < \varepsilon$) or $k$ reaches the column number of $C$. When $|\tilde{c}_k|^2 \neq 0$, $G_k$ is computed iteratively from $C_k$ by (53), (54) and (43). Notice that the above function $[k, G_k] = \text{InvChol}(C, \lambda)$ can also be implemented with Algorithm 3 instead of Algorithm 2, when the Matlab built-in function “chol” is preferred. In Algorithm 2 and Algorithm 3, the positive real number $\lambda$ is the ridge parameter satisfying $\lambda \rightarrow 0$, which is utilized to approximate the generalized inverse with the ridge inverse [6], as in the original BLS [1].

If $k = 0$, i.e., $\tilde{c}_{k+1} = c_1 = 0$, $\delta$ is decided in rows 7–13, which means that the first $\delta$ columns of $C$ are zeros; Otherwise in rows 15–17, the $k \times k$ matrix $G_k$ is applied to compute $B_k^T$ by (47), and then $B_k^T$ is applied to to update $(A_i^{m+1})^+$ into $(A_{i+k}^{m+1})^+$ by (50). Moreover, if $i < p$ after the above operations, $\tilde{c}_{k+1} = 0$ must have been found in row 4, and the first $\delta$ columns of $C$ are zeros when $k = 0$. Thus the the pseudoinverse $(A_i^{m+1})^+$ is updated into $(A_{i+k}^{m+1})^+$ by (50).

1 We can also use (53) instead of (52), at the cost of higher complexity.
Algorithm 1: Pseudoinverse of Col.-Partitioned Matrix

Input: $H_{m+1}$ with $p$ columns, $A_0^{m+1} = A^m$, $(A_n^{m+1})^+$, the ridge parameter $\lambda \rightarrow 0$

Output: $(A_p^{m+1})^+ = (A^m)^+ = [A_m^m H_{m+1}]^+$

1. $i = 0$
2. while $i < p$
3. $D = (A_i^{m+1})^+ H_{m+1}^{i+1:p}$, $C = H_{m+1}^{i+1:p} - A_i^{m+1} D$;
4. $[k, G_k] = \text{InvChol}(C, \lambda)$;
5. $\delta = 1$;
6. if $k = 0$ (i.e., $|\tilde{c}_1|^2 \leq \varepsilon$) then
7.   $\delta = 1$; //break
8.   //Terminates the for loop
9. end
10. for $j = 2 : p - i$ do
11.   if $|c_{ij}|^2 \leq \varepsilon$ then
12.     $\delta = \delta + 1$;
13.   else
14.     break; //Terminates the for loop
15. end
16. $B_i^T = G_k G_k^T C_k^T$;
17. $(A_{i+i}^{m+1})^+ = \left[ (A_{i+i}^{m+1})^+ - D_k B_i^T \right]$;
18. $i = i + k$;
19. end
20. if $i < p$ then
21. $H_\delta = H_{m+1}^{i+1:i+1+\delta}$;
22. $D_\delta = \left( \begin{array}{c} D_{i+1,i} \delta \text{ if } k = 0 \\ (A_i^{m+1})^+ H_\delta \text{ if } k \neq 0 \end{array} \right)$;
23. $B_\delta^T = \left( \begin{array}{c} (I + D_\delta^T D_\delta)^{-1} D_\delta \text{ if } m \geq n + i + \delta \\ (I + D_\delta^T H_\delta)^{-1} D_\delta \text{ if } n + i + m \geq \delta \\ D_\delta^T (I + H_\delta^T H_\delta)^{-1} \text{ if } m \leq \delta, \end{array} \right)$ where $H_\delta$ is $m \times \delta$, and $D_\delta$ is computed by $D_\delta = D_\delta^T (A_i^{m+1})^+$ if required;
24. $(A_{i+\delta}^{m+1})^+ = \left[ (A_{i+\delta}^{m+1})^+ - D_\delta B_\delta^T \right]$;
25. $i = i + \delta$;
26. end
27. end

in rows 20–24, where $B_i^T$ is computed by (55a)/(55b)/(55c), and $D_\delta = D_{i+1,i}^{1:1}$ is the first $\delta$ columns of $D$ if $k = 0$, or computed by (49) if $k \neq 0$.

In the first iteration, (55d) is implemented in row 15 if $k = p$, (55a)/(55b)/(55c) is implemented in row 22 if $\delta = p$, and (55e) is corresponding to all other cases. Moreover, if $i < p$ after the above-described first iteration of the while loop, the next iteration of the while loop will start with $H_{m+1}^{i+1:p}$ (including only the last $p - i$ columns of $H_{m+1}$), $A_{m+i}^{m+1} = [A_m^m H_{m+1}^{i+1:p}]$ and $(A_{m+i}^{m+1})^+$.

Algorithm 2: The InvChol function implemented with the inverse Cholesky factorization

Function $[k, G_k] = \text{InvChol}(C, \lambda)$

for $k = 1 : \text{size}(C, 2)$ do
1. $\tilde{c}_k = c_k^T c_k - c_k^T C_{k-1} G_{k-1} G_{k-1}^T C_{k-1}^T c_k$;
2. if $|\tilde{c}_k|^2 < \varepsilon$ then
3.   $k = k - 1$;
4.   break; //Terminates the for loop
5. else
6. $g_{kk} = 1 / |\tilde{c}_k|^2$;
7. $\tilde{g}_k = -g_{kk} G_{k-1} G_{k-1}^T C_{k-1} C_{k-1}^T c_k$;
8. $G_k = \tilde{g}_k G_{k-1} + g_{kk} G_{k-1}$;
9. $G_1 = \tilde{G}_1$;
10. end
11. return $k, G_k$;
12. end

Algorithm 3: The InvChol function implemented with the Matlab built-in function “chol”

Function $[\tilde{G}, FLAG] = \text{chol}(C^T C + \lambda I)$

if isempty($\tilde{G}$) then
1. $\tilde{g} = \text{diag}(\tilde{G})$, $d = \text{find}(\tilde{g} < \varepsilon)$;
2. if isempty($d$) then
3.   $\tilde{G} = \tilde{G}(1:d(1) - 1, 1:d(1) - 1)$;
4. end
5. $G_k = \tilde{G}^{-1}$, $k = \text{size}(G_k, 1)$;
6. return $k, G_k$;
7. end

IV. THE PROPOSED ALGORITHM FOR ROW-PARTITIONED MATRICES IN BLIS

The incremental learning for the increment of input data in [1] utilizes the pseudoinverse of the row-partitioned matrix

$x A_m^m = \begin{bmatrix} A_m^m \\ A_x^m \end{bmatrix}$, (57)

where $A_m^m$ is $m \times n$, and $A_x^m$ can be assumed to be $q \times n$. Equation (c) in [2 Ex. 1.16] can be written as

$(A^T)^T = (A^T)^+$, (58)

into which we can substitute (57) to obtain

$(x A_m^m)^T = [(A_m^m)^T A_x^m]^+$. (59)

Then substitute (59) into (10) to obtain

$(x A_m^m)^T = \left[ [(A_m^m)^T + DB^T]^T \right]$, (60)

i.e.,

$(x A_m^m)^+ = \left[ (A_m^m)^+ - BD^T B \right]$. (61)
Algorithm 4: Pseudoinverse of Row-Partitioned Matrix

Input: $A_x$ with $q$ rows, $\delta^* A_n^m = A_n^m$, $(\delta^* A_n^m)^\top$, the ridge parameter $\lambda \to 0$

Output: $(\frac{\pi}{q} A_n^m)^+ = (\frac{\pi}{q} A_n^m)^+$

1. $i = 0$
2. while $i < q$
3. \[ D^T = A_x^{i+1:q} : (\frac{\pi}{q} A_n^m)^+; \]
4. \[ C = (A_x^{i+1:q} : T) - (\frac{\pi}{q} A_n^m)^T D; \]
5. \[ [k, G_k] = \text{InvChol}(C, \lambda); \]
6. \[ \delta = 1; \]
7. if $k = 0$ (i.e., $\tilde{c}_1 < \varepsilon$) then
8. \[ for j = 2; q - i do \]
9. \[ if |c_{ij}|^2 < \varepsilon \text{ then} \]
10. \[ \delta = \delta + 1; \]
11. \[ \text{else} \]
12. \[ break; // Terminate the for loop \]
13. \[ end \]
14. \[ else \]
15. \[ B_k = C_k G_k G_k^T; \]
16. \[ (\frac{\pi}{q} A_n^m)^+ = (\frac{\pi}{q} A_n^m)^+ - B_k D_k^T B_k; \]
17. \[ i = i + k; \]
18. \[ end \]
19. if $k \leq p - i - 1$ then
20. \[ A_x^\delta = A_x^{i+1:i+k}; \]
21. \[ D^\delta = \begin{cases} D^T, & \text{if } k = 0 \\ A_x^\delta (\frac{\pi}{q} A_n^m)^+ & \text{if } k \neq 0 \end{cases}; \]
22. \[ B^\delta = \begin{cases} D^T (I + D^T D)^{-1}, & \text{if } n \geq \max(m + i, \delta) \\ D^T (I + A_x^\delta D^T D)^{-1}, & \text{if } m + i \geq n \geq \delta \\ (I + D^T D)^{-1} D^T, & \text{if } n \leq \delta \end{cases}; \]
23. \[ \text{where } A_x^\delta \text{ is } \delta \times n; \]
24. \[ (\frac{\pi}{q} A_n^m)^+ = (\frac{\pi}{q} A_n^m)^+ - B_k D_k^T B_k; \]
25. \[ i = i + \delta; \]
26. \[ end \]

Obviously $[A^m | H_{m+1}]$ in (10) is replaced with $[(A^m)^T | A^T_x]$ in (59). Accordingly in (11), (12), (35) and (38), $A^m$ and $H_{m+1}$ should be replaced by $(A_n^m)^T$ and $A_x^T$, respectively, to obtain $D = ((A_n^m)^T)^+ \ A_x^T$ that can be written as

$$D^T = A_x (A_n^m)^+,$$

$$C = A_x^T - (A_n^m)^T D,$$

$$\hat{D} = D^T \left((A_n^m)^+\right)^T$$

that can be written as

$$\hat{D}^T = (A_n^m)^+ D,$$

and

$$B^T = \begin{cases} (I + D^T D)^{-1} \hat{D}, & \text{if } C = 0, n \geq \max(m, q) \\ (I + \hat{D} A_x^T D)^{-1} \hat{D}, & \text{if } C = 0, m \geq \max(n, q) \\ \hat{D} (I + A_x^T \hat{D})^{-1}, & \text{if } C = 0, n \geq q \\ C^+, & \text{if each } \tilde{c}_k \neq 0 (1 \leq k \leq q) \\ \ldots & \text{if } C \neq 0 \text{ but several } \tilde{c}_k = 0 \end{cases};$$

that can be written as

$$\hat{D}^T (I + D^T D)^{-1}, \text{ if } C = 0, n \geq \max(m, q);$$

$$\hat{D}^T (I + A_x^T D)^{-1}, \text{ if } C = 0, m \geq \max(n, q);$$

$$(C^+)^T, \text{ if each } \tilde{c}_k \neq 0 (1 \leq k \leq q);$$

$$\ldots, \text{ if } C \neq 0 \text{ but several } \tilde{c}_k = 0;$$

where $A_x$ is $n \times n$. In (65), $\tilde{c}_1$ can be obtained by (17), and according to Theorem 5, when $\tilde{c}_1 \neq 0$ for all $1 \leq i \leq k - 1$, $\tilde{c}_k (k = 2, 3, \ldots, p)$ can be computed by (51), where $G_{k-1}$ is the inverse Cholesky factor of $C_{k-1}^T C_{k-1}$. $G_{k-1}$ can be computed by (44), (45) and (43) when $k \geq 3$, or by (54) when $k = 2$.

Moreover, since (38) can be written as (55), (65) can be written as

$$\hat{D}^T (I + D^T D)^{-1}, \text{ if } C = 0, n \geq \max(m, q);$$

$$\hat{D}^T (I + A_x^T D)^{-1}, \text{ if } C = 0, m \geq \max(n, q);$$

$$(C^+)^T, \text{ if } C^T C \text{ is positive definite};$$

$$\ldots, \text{ if } C \neq 0 \text{ and } C^T C \text{ is not positive definite}.$$

Obviously, (65) can also be written as the form that is similar to (56), which is omitted for simplicity.

Let

$$\frac{\pi}{q} A_n^m = \begin{bmatrix} A_n^m \\ A_x^{1:k} \end{bmatrix};$$

where $A_x^{1:k}$ denotes the first $k$ rows of $A_x$. When $k = 0$, $A_x^{1:k}$ becomes empty and then (65) becomes

$$\frac{\pi}{q} A_n^m = A_n^m.$$

Then the algorithm for the pseudoinverse of a row-partitioned matrix is shown in Algorithm 4, where $C_k$ and $D_k$ denote the first $k$ columns of $C$ and $D$, respectively, and the function $[k, G_k] = \text{InvChol}(C, \lambda)$ in row 5 can be implemented with Algorithm 2 or Algorithm 3.

V. CONCLUSIONS

In BLS, Greville’s method [2] has been utilized to propose an effective and efficient incremental learning system without retraining the whole network from the beginning. For a column-partitioned matrix $A^{m+1} = [A^m | H_{m+1}]$, where $H_{m+1}$ includes $p$ columns, Greville’s method spends $p$ iterations to compute $(A^{m+1})^+$ from $(A^m)^+$, where $A^+$ denotes the pseudoinverse of the matrix $A$. However, the incremental algorithms in [11] extend Greville’s method to compute $(A^{m+1})^+$ from $(A^m)^+$ by just 1 iteration, which
have neglected some possible cases, and need further improvements in efficiency and numerical stability. In this paper, we propose an efficient and numerical stable algorithm from Greville’s method, to compute \((A^{m+1})^+\) from \((A^m)^+\) by just 1 iteration, where all possible cases are considered, and the efficient inverse Cholesky factorization in [3] can be applied to further reduce the computational complexity. Finally, we give the whole algorithm for column-partitioned matrices in BLS. On the other hand, we also give the proposed algorithm for row-partitioned matrices in BLS.

**APPENDIX A**

**Proof of Theorem 5**

Firstly, we verify (51). From (15) we deduce that the \(k\)-th column of \(C\) is \(c_{k, 1} = h_{k, 1} - A^m(A^m)^+ h_{k, 1}\), which is substituted into (51) to obtain \(\tilde{c}_{k, 1} = h_{k, 1} - A^m(A^m)^+ h_{k, 1} - C_{k-1} G_{k-1}^T C_{k-1}^T h_{k, 1} - A^m(A^m)^+ h_{k, 1}\). i.e.,

\[
\tilde{c}_{k, 1} = (I - A^m(A^m)^+ - C_{k-1} G_{k-1} G_{k-1}^T C_{k-1}^T + C_{k-1} G_{k-1} G_{k-1}^T C_{k-1}^T A^m(A^m)^+) h_{k, 1}.
\] (69)

Equation (69) can be written as

\[
\tilde{c}_{k, 1} = \left(I - A^m(A^m)^+ - C_{k-1} G_{k-1} G_{k-1}^T C_{k-1}^T \right) h_{k, 1},
\] (70)

since \(C_{k-1}^T A^m(A^m)^+\) in the 2nd row of (69) always satisfies

\[
C_{k-1}^T A^m(A^m)^+ = 0,
\] (71)

which will be verified in the next paragraph.

To verify (71), write (15) as

\[
C_{k-1} = H_{m+1}^{1:k-1} - A^m(A^m)^+ H_{m+1}^{1:k-1},
\]

which is substituted into (71) to obtain

\[
(H_{m+1}^{1:k-1})^T A^m(A^m)^+ - (H_{m+1}^{1:k-1})^T (H_{m+1}^{1:k-1})^T A^m(A^m)^+ = 0.
\] (72)

Substitute (9c) into (72) to obtain

\[
(H_{m+1}^{1:k-1})^T A^m(A^m)^+ - (H_{m+1}^{1:k-1})^T A^m(A^m)^+ A^m(A^m)^+ = 0,
\] (73)

into which substitute (48) to obtain

\[
I - A^m(A^m)^+ - C_{k-1} G_{k-1} G_{k-1}^T C_{k-1}^T h_{k, 1} - A^m(A^m)^+ h_{k, 1} = 0.
\] (74)

Obviously, (74) deduced from (71) always holds. Thus (71) has been verified.

After verifying (71), let us go back to (70) that has been deduced from (51). We focus on the entry

\[
I - A^m(A^m)^+ - C_{k-1} G_{k-1} G_{k-1}^T C_{k-1}^T h_{k, 1} - A^m(A^m)^+ h_{k, 1} = 0.
\] (75)

in (70). Substitute (47) into (75) to obtain

\[
I - A^m(A^m)^+ - C_{k-1} B_{k-1}^T h_{k, 1} - A^m(A^m)^+ h_{k, 1} = 0.
\] (76)

REFERENCES

[1] C. L. Philip Chen, and Z. Liu, “Broad Learning System: An Effective and Efficient Incremental Learning System Without the Need for Deep Architecture”, IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 1, Jan. 2018.

[2] A. Ben-Israel and T. Greville, *Generalized Inverses: Theory and Applications*, New York, USA, Wiley, 1974.

[3] H. Zhu, W. Chen, B. Li, and F. Gao, “An Improved Square Root Algorithm for V-BLAST Based on Efficient Inverse Cholesky Factorization”, IEEE Trans. Wireless Commun., vol. 10, no. 1, Jan. 2011.

[4] H. V. Henderson and S. R. Searle, “On Deriving the Inverse of a Sum of Matrices”, SIAM Review, vol. 23, no. 1, January 1981.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, third ed. Baltimore, MD: Johns Hopkins Univ. Press, 1996.

[6] Donald W. Marquardt, “Generalized Inverses, Ridge Regression, Biased Linear Estimation, and Nonlinear Estimation”, Technometrics, vol. 12, no. 3, Aug. 1970.