Speed-up of the Matrix Computation on the Ridge Regression

Woochan Lee[1], Moonseong Kim[2,*] and Jaeyoung Park[3,*]
[1] Department of Electrical Engineering, Incheon National University
Incheon 22012, Republic of Korea
[e-mail: wlee@inu.ac.kr]
[2] Department of IT Convergence Software, Seoul Theological University
Bucheon 14754, Republic of Korea
[e-mail: moonseong@stu.ac.kr]
[3] Department of Computer Engineering, Hongik University
Seoul 04066, Republic of Korea
[e-mail: jypdeca@hongik.ac.kr]
[*]Corresponding authors: Moonseong Kim, Jaeyoung Park

Received January 8, 2021; accepted March 4, 2021;
published October 31, 2021

Abstract

Artificial intelligence has emerged as the core of the 4th industrial revolution, and large amounts of data processing, such as big data technology and rapid data analysis, are inevitable. The most fundamental and universal data interpretation technique is an analysis of information through regression, which is also the basis of machine learning. Ridge regression is a technique of regression that decreases sensitivity to unique or outlier information. The time-consuming calculation portion of the matrix computation, however, basically includes the introduction of an inverse matrix. As the size of the matrix expands, the matrix solution method becomes a major challenge. In this paper, a new algorithm is introduced to enhance the speed of ridge regression estimator calculation through series expansion and computation recycle without adopting an inverse matrix in the calculation process or other factorization methods. In addition, the performances of the proposed algorithm and the existing algorithm were compared according to the matrix size. Overall, excellent speed-up of the proposed algorithm with good accuracy was demonstrated.

Keywords: Machine Learning, Matrix Computation, Ridge Regression, Series Expansion, Simulation Acceleration

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-00098, Advanced and Integrated Software Development for Electromagnetic Analysis). This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science, ICT & Future Planning (No. NRF -2019R1G1A1007832).

http://doi.org/10.3837/tiis.2021.10.001 **ISSN : 1976-7277**

1. Introduction

The rise of the fourth industrial revolution attracted many researchers' attention to artificial intelligence and big data techniques. In particular, interest in machine learning that is a branch of artificial intelligence is explosive. Regression analysis (which is a kind of supervised learning) that estimates trends by learning the already known data set is the basic technique of machine learning [1], and the utilization of the technique has never been reduced. Machine learning techniques, including regression, require massive data processing under recent sensor data environments. Due to computational limitation, it is hard to adopt these novel machine learning techniques especially in IoT (Internet of Things) or mobile devices [2]. As the data size grows large, it is a significant concern of how fast the data is processed as well as accurate forecasting.

For radar applications, linear regression can be used to improve target detection performance by estimating the clutter ridge [3]. Ridge regression with regularization also helps to enhance position accuracy under a strong jamming environment [4]. Thus, ridge regression techniques can play an important role in the tremendous data input condition.

However, ridge regression contains a matrix inverse-related part. Thus the treatment of this part takes a large portion of the total time for getting a ridge regression estimator. Therefore, implementing a fast regression model can be achieved by accelerating this calculation part. The proposed algorithm gets through series expansion and computation recycle without adopting an inverse matrix in the calculation process or other matrix factorization techniques. The proposed algorithm shows superior performance compared to previously proposed algorithms, and it shows an acceptable margin of error.

This paper's idea is based initially on the idea proposed by Lee [5]. However, this idea is limited to speeding up the acquisition of the regression estimator through Cholesky factorization in ordinary least squares methods. This paper expanded its scope to the case of ridge regression, developed a new algorithm without any matrix inverse or factorization, and verified its performance.

The rest of this paper is organized as follows. Section 2 reviews prior research on speeding up by reducing ridge regression calculation. Section 3 proposes a new algorithm and Section 4 shows simulation results based on the new algorithm. Section 5 presents conclusions.

2. Regression Models and Previous Work

First, system of linear regression function is defined by:

$$\begin{aligned} a_1x_{11} + a_2x_{12} + \cdots + a_nx_{1n} &= y_1 \\ a_1x_{21} + a_2x_{22} + \cdots + a_nx_{2n} &= y_2 \\ &\vdots \\ a_1x_{n1} + a_2x_{n2} + \cdots + a_nx_{nn} &= y_n \end{aligned} \tag{1}$$

where x is observed value (known data set), and a is coefficient. Based on Equation (1), for instance, a system of linear equations can be set up to illustrate the relationship of employment rate (corresponding y) on scores of n grade points (corresponding x). This linear equation is represented by a matrix equation of $Xa = y$. Vector y can be projected onto the column space

of the matrix X, and the least squares method finds the projected vector $y = \hat{y}$, which minimizes the sum of squared residuals, as Equation (2). Furthermore, the regression estimator $\hat{a}$ is represented as follows [6]:

$$\hat{y} = X(X^T X)^{-1} X^T y = X\hat{a}, \tag{2}$$

$$\hat{a} = (X^T X)^{-1} X^T y \tag{3}$$

Here, X^+ can be defined as:

$$X^+ = (X^T X)^{-1} X^T$$

which is the Moore-Penrose generalized inverse [7].

The regression coefficient $\hat{a}$ is linearly combined with the new observed values, and it produces the new estimated value $\hat{y}$. In other words, once $\hat{a}$ is found, for example, a new student's employment rate (y) can be estimated with a new student's grade points. Therefore, to obtain the coefficient $\hat{a}$ is the core of the regression analysis and machine learning.

However, when implementing various regression analyses including least squares regression, there are differences in performance depending on how the regression estimator $\hat{a}$ is derived. The difference in performance is noticeable as the size of the desired matrix grows.

There are well-known regression models that have already been proposed, such as lasso regression, ridge regression, quantile regression, and progress trees [8]. Of these models, the ridge regression has an additional constraint on linear regression coefficients to the ordinary least squares model. This additional constraint (λ) is to minimize the sum of the square of weights [9].

Under these constraints, ridge regression model gives the effect of regularization and more stable future predictions. Taking vector-matrix form of ridge regression model, the ridge regression estimator (β) can be obtained as:

$$\beta(\lambda) = (X^T X + \lambda I)^{-1} X^T y \tag{4}$$

Since the calculation of $(X^T X + \lambda I)^{-1}$ takes a large portion of the total time for getting ridge regression estimator, it is required to have accelerated method to obtain ridge regression estimator.

One way to speed up the inverse matrix calculation of ridge regression is factorizing the given matrix $A = (X^T X + \lambda I)^{-1}$ into matrices such as upper triangular matrix and diagonal matrix. Among the methods to factor the given matrix, Singular Value Decomposition (SVD) method has the advantage of being able to apply any matrix even if the given matrix A is not a square matrix. The SVD method factors the matrix A as follows [10]:

$$A = U\Sigma V^T \tag{5}$$

where U and V are orthogonal matrices and Σ is a diagonal matrix. However, it needs huge calculations to factor an arbitrary matrix with SVD [11]. Another approach is to use Cholesky decomposition, which factors the given matrix A as follows:

$$A = LL^T = R^T R \tag{6}$$

where L is a lower triangular matrix and R is an upper triangular matrix. Although Cholesky decomposition needs an additional process to factor the matrix, it reduces calculation time and has lower computational complexity [12]. It is also known that it is faster than SVD. However, it is somewhat unstable in terms of accuracy [13].

Another approach based on QR decomposition, which Q represents an orthogonal matrix, and R represents an upper triangular matrix, was previously proposed. There are different methods to compute QR decomposition such as Gram-Schmidt Orthonormalization, Householder Reflections and Givens rotation [14]. The method factorizes given matrix A into:

$$A = QR \tag{7}$$

with Q and R matrices. QR decomposition reduces computations through factorization like the above prior researches.

However, a numerical roundoff error to make Q prone to lose orthogonality, which is defined as follows:

$$fl(x, y) = (x, y) + \delta \tag{8}$$

where x and y are two vectors and δ is the error from rounding operation, is likely to occur in a classical Gram-Schmidt Orthonormalization process while constructing Q matrix [15]. Also, Householder Reflections and Givens rotation to perform QR factorization are known to be slower than Cholesky decomposition [16].

3. The Proposed Algorithm

Total computing time heavily depends on the calculation of an inverse related part, and the several existing methods are using matrix factorization. It takes much time to compute an inverse matrix directly or to factor matrix directly into the desired form. The proposed algorithm approximates an inverse matrix through series expansion and exploits computation recycle instead of computing an inverse matrix directly or adopting any matrix factorization techniques.

Now, the given matrix $A = (X^T X + \lambda I)^{-1}$ is approximated through a series expansion of the inverse matrix as below:

$$(I + K)^{-1} = I - K + K^2 - K^3 + K^4 \ldots \tag{9}$$

where $\|K\| \ll 1$ [17].

For the series expansion to converge, the conditions of $\|K\| \ll 1$ should be absolutely satisfied. If the norm of the matrix is greater than 1, the series expansion must diverge and the error increases, resulting in inaccurate approximation.

With norm condition satisfied, the norm of the matrix K is much less than 1 during the series expansion process, and the norms of the other matrix terms such as K^2 and K^3 are gradually smaller as the order of terms increases in the process of expansion. Therefore, the first few terms are sufficient to provide a good approximation of the inverse. For example, Equation (9) can be approximated with 3 expansion terms (3 terms expansion case):

$$(I + K)^{-1} \approx I - K + K^2 \tag{10}$$

where $\|K\| \ll 1$. It is also possible to add more terms to Equation (10) to improve approximation accuracy. With this knowledge, to obtain ridge regression estimator with series expansion, the part $(X^TX + \lambda I)^{-1}$ in the original Equation (4) can be changed into the expression starting with an identity matrix as Equation (10). From Equation (4), the expression for ridge regression estimator can be transformed as:

$$\begin{aligned}\hat{\beta} &= \left(\lambda\left(I + \frac{1}{\lambda}X^TX\right)\right)^{-1} X^Ty \\ &= \frac{1}{\lambda}\left(I + \frac{1}{\lambda}X^TX\right)^{-1} X^Ty\end{aligned} \tag{11}$$

Recalling Equation (10), Equation (11) takes the form of (10) when $\frac{1}{\lambda}X^TX$ is substituted into K. To expand Equation (11) in series, $\left\|\frac{1}{\lambda}X^TX\right\|$ should be much less than 1 to make overall system converge. To satisfy this strict condition, just the value of λ can be adjusted since the matrix X is already given. In this paper, the initial setting is $\lambda = 10n^2$ (where n is the column size of the matrix X and X^TX is a square matrix of order n), and definitely, other settings are also possible. Therefore, the value of λ depends on the size of the matrix X. This assumption about regularization factor λ is reasonable because the emphasis on the diagonal element takes effect when the diagonal element is greater than off-diagonal elements.

Because it has not known the method that gives an optimal value of λ [18], the value of λ can be assumed to be determined freely in this paper. Therefore, instead of choosing adaptive values according to the data characteristics of the given matrix, it is also possible to choose any value that meets the convergence condition, for example, $\lambda = 10^9$. It should be noted that the choice of λ is chosen to compensate for the collinearity of the data set in a real-world application.

Expanding Equation (11), the final form is obtained as:

$$\hat{\beta} = \frac{1}{\lambda}\left(I - \frac{1}{\lambda}X^TX + \frac{1}{\lambda^2}(X^TX)^2\right)(X^Ty) \tag{12}$$

Next, a new approach to calculate the series expansion effectively is proposed. The new approach reuses the previously calculated matrix-vector products in the next matrix calculation. In the series expansion of $(X^TX + \lambda I)^{-1}$, X^Ty and X^TX multiplied by X^Ty are repeated as shown in Equation (12). Based on this observation, we obtain a much faster result by multiplying $-\frac{1}{\lambda}X^TX$ by $(X^TX)(X^Ty)$ that is calculated in the previous step than calculating $\frac{1}{\lambda^k}(X^TX)^k(X^Ty)$ through multiplying X^TX by k times in kth step. This speed-up is since it we can change forward matrix-matrix operation into backward matrix-vector operation. **Fig. 1** presents the comparison between the forward matrix-matrix operation and the backward matrix-vector operation. The computational load difference between two methods considering the computational complexity of the product of the $m \times n$ matrix and $n \times 1$ column vector is significant.

$$(A^TA)y = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix}\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

$$mn^2 + n^2 \text{ (flops)}$$

(a) Forward direction calculation

$$A^T(Ay) = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix}\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

$$nm + nm = 2nm \text{ (flops)}$$

(b) Backward direction calculation

Fig. 1. Flops comparison of matrix-matrix-vector operations

Specifically, in the worst-case scenario, the computational complexity is $n^2 + n^3 + n^2$ flops when the Equation (11) is computed with the forward calculation by multiplying (X^TX) from the beginning (**Fig. 1.** (a)) like Equation (13):

$$\hat{\beta} = \frac{1}{\lambda}(X^Ty) - \frac{1}{\lambda^2}(X^TX)(X^Ty) + \frac{1}{\lambda^3}(X^TX)(X^TX)(X^Ty) \tag{13}$$

However, the backward calculation by the form of Equation (13) takes only $2n^2 + 2n^2$ flops:

$$\hat{\beta} = \frac{1}{\lambda}(X^Ty) - \frac{1}{\lambda}(X^TX)(\frac{1}{\lambda}(X^Ty)) + \frac{1}{\lambda}(X^TX)(\frac{1}{\lambda^2}(X^TX)(X^Ty)) \tag{14}$$

This is because the term of $\frac{1}{\lambda}(X^Ty)$ in the first step of Equation (14) is recycled to obtain $\frac{1}{\lambda^2}(X^TX)(X^Ty)$ in the second step, and $\frac{1}{\lambda^2}(X^TX)(X^Ty)$ is also reused to calculate $\frac{1}{\lambda^2}(X^TX)^2(X^Ty)$ in the third step. Since each matrix-vector product takes only $2n^2$ flops, the computational complexity of the backward matrix-vector operation is $O(n^2)$ compared to that of the forward matrix-matrix operation, of which the computational complexity is $O(n^3)$. Therefore, the proposed recycling algorithm shows much better performance in terms of speed than previous algorithms. **Table 1** shows the pseudo-code of the proposed recycling algorithm.

Table 1. Expansion term reuse algorithm for the acceleration of matrix computation

Calculation of $\boldsymbol{\beta}$

```
X_mul ← XᵀX
b1 ← Xᵀy
b2 ← -(1/lambda)·(X_mul·b1)
b3 ← -(1/lambda)·(X_mul·b2)
b4 ← -(1/lambda)·(X_mul·b3)
…
beta ← -(1/lambda)·(b1+b2+b3+b4)
```

Actually, Equation (4) can be converted Equation (15) by the well-known matrix inversion lemma [19-21]. By this lemma, the size of matrix inversion part can be further reduced when n is greater than m. Thus, the inverse part ($(X^TX+\lambda I)^{-1}$) is now ($(XX^T+\lambda I)^{-1}$), then the size of the inverse part becomes m^2, which is much smaller than n^2 when $n \gg m$. The derivation of modified form of (4) is as follows:

$$\begin{aligned}\beta &= (X^TX+\lambda I)^{-1}X^T\mathrm{y}\\ &= \frac{1}{\lambda}(I - I + \lambda(\lambda I + X^TX)^{-1})X^Ty\\ &= \frac{1}{\lambda}(X^T - (I - \lambda(\lambda I + X^TX)^{-1})X^Ty\\ &= \frac{1}{\lambda}(X^T - (\lambda I + X^TX - \lambda I)(\lambda I + X^TX)^{-1}X^T)y\\ &= \frac{1}{\lambda}(X^T - X^TX(\lambda I + X^TX)^{-1}X^T)y\\ &= \frac{1}{\lambda}X^T(I - X(\lambda I + X^TX)^{-1}X^T)y\\ &= X^T\left(\frac{1}{\lambda}I - \frac{1}{\lambda}\left(I + X^TX\frac{1}{\lambda}\right)^{-1}X^T\frac{1}{\lambda}\right)y\\ &= X^T(XX^T+\lambda I)^{-1}y\end{aligned} \tag{15}$$

The pseudo-code of the new size-reduced algorithm based on the transformation of Equation (15) is shown in **Table 2**. The significant differences compared to the original algorithm described in **Table 1** are 1) the value of variable `X_mul`, 2) the stating value of `b1`, and 3) the final form of `beta`.

Table 2. Size-reduced expansion term reuse algorithm by the matrix inversion lemma

Calculation of $\boldsymbol{\beta}$

```
X_mul ← XXᵀ
b1 ← y
b2 ← -(1/lambda)·(X_mul·b1)
b3 ← -(1/lambda)·(X_mul·b2)
b4 ← -(1/lambda)·(X_mul·b3)
…
beta ← -Xᵀ·(1/lambda)·(b1+b2+b3+b4)
```

4. Simulation Results and Analysis

We measured the time of acquisition of the ridge regression estimator (β) per each algorithm and compared the proposed algorithm with algorithms based on the built-in functions from commercial MATLAB (an abbreviation of MATrix LABoratory) program by MathWorks. First, the matrix $M = X^TX + \lambda I$ in Equation (4) ($\beta(\lambda) = (X^TX + \lambda I)^{-1}X^Ty$) is defined, then the following four algorithms are executed:

1) After deriving inverse matrix by MATLAB's `inv`, then multiply this with $b = X^T y$,

2) By MATLAB's `backslash`, derive $\beta(\lambda) = M \backslash b$,

3) M is factorized by Cholesky decomposition, then apply backward/forward substitution (self-coded),

4) Apply proposed series expansion and computation recycle algorithm (3 terms expansion and 4 terms expansion cases).

The dimension of the matrix X is $m \times n$, where $m = 1{,}000$ and n is in range of $1{,}000$ to $15{,}000$ in units of $1{,}000$. According to the value of n, the size of the matrix $X^T X$ lies between $1{,}000$ by $1{,}000$ and $15{,}000$ by $15{,}000$. The elements of the matrix X are generated by the MATLAB command `rand`, which creates a matrix whose elements uniformly distributed in the interval [0, 1]. In this fictional data set, the collinearity of the columns is not a significant issue as the size of the matrix grows. We executed the simulation using Intel i7-7700 processor @ 3.6GHz, RAM 16GB and MATLAB, 2018b. We measured the total time in terms of the CPU times with MATLAB command `cputime` for each experiment.

It should be noted that the MATLAB command backslash does not necessarily use Gaussian elimination. It selects an optimized method according to the shape of the matrix, which can be reorganized as shown in **Fig. 2** [22].

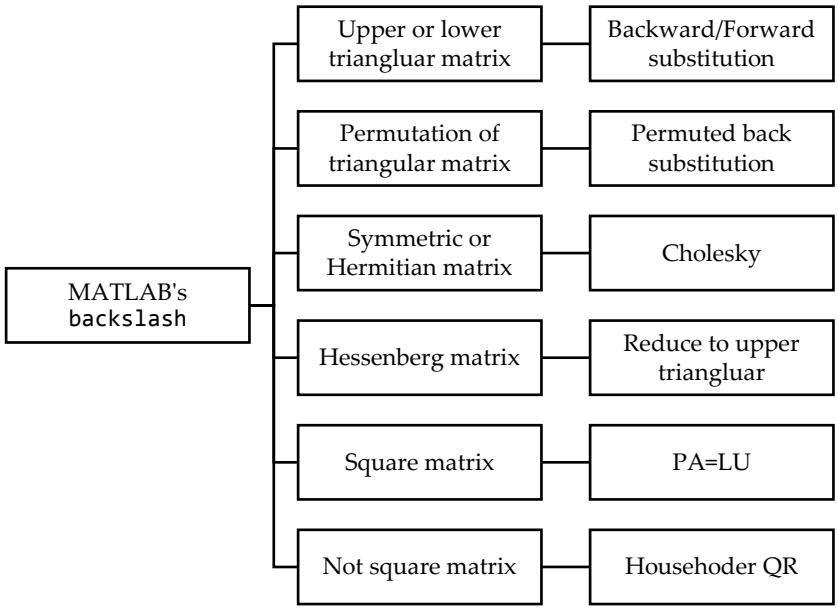

Fig. 1. Algorithms for MATLAB built-in function backslash (figure reconstructed from [22])

Table 3 shows the comparison of computation cost between the proposed algorithm and Cholesky factorization-based algorithm. To be specific, it takes mn^2 flops to calculate $X^T X$, $\frac{1}{3}n^3$ flops to perform initial Cholesky factorization, and $2n^2$ flops to do backward/forward substitution. The MATLAB command `backslash` also might be executed by selecting Cholesky factorization.

Table 3. Computation cost comparison of proposed algorithm *vs.* Cholesky factorization (common X^TX matrix building: mn^2 (flops))

Class	Operations Count	Notes
Proposed algorithm	1) $mn^2 + 6n^2$ 2) $mn^2 + 8n^2$	1) 3 terms expansion $(I - K + K^2)$ 2) 4 terms expansion $(I - K + K^2 - K^3)$

Fig. 3 presents the comparison of computation time between the proposed algorithm with a 3 terms expansion $(I - K + K^2)$ and other algorithms. The MATLAB's inv command has the worst performance. Assuming implemented by Gauss-Jordan elimination, its computational cost is known as approximately n^3 flops [7]. Our proposed algorithm has the best performance compared to MATLAB's `inv` command and Cholesky factorization-based algorithms. It also should be noted that the MATLAB built-in command-based algorithms can exaggerate their performance. For example, even though the time complexity of the algorithm of the MATLAB's built-in function is $O(n^3)$, its actual performance (time complexity) can be $O(n^2)$ due to MATLAB's optimization.

Fig. 4 presents the comparison between our algorithm and other algorithms again with the time axis in log-scale to recognize clearly the performance improvement of the proposed algorithm. When the size of the matrix is small, the performance improvement is not clear compared to MATLALB's `inv`, `backslash`, or self-implemented Cholesky factorization. However, the performance improvement of the proposed algorithm is obvious as the size of the matrix increases.

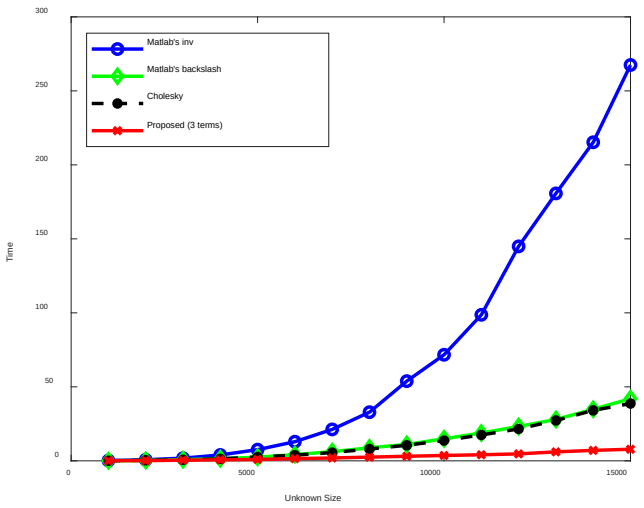

Fig. 3. Computation time comparison (3 terms expansion)

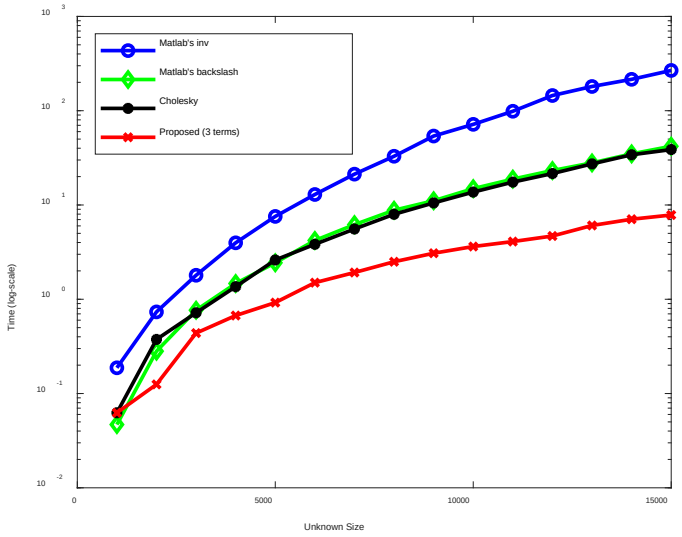

Fig. 4. Computation time comparison (log-scale, 3 terms expansion)

Fig. 5 demonstrates the error performance of the proposed algorithm. Generally, the error performance of MATLAB `backslash` command has been verified, we evaluated the error performance of our algorithm with reference to MATLAB's `backslash` command. As we expected, approximation by series expansion obviously has the error. However, the relative error range is only about 10^{-4} to 10^{-6}, which is acceptable. In this paper, we use $\lambda = 10n^2$, but the error will be reduced further if we use a larger λ value.

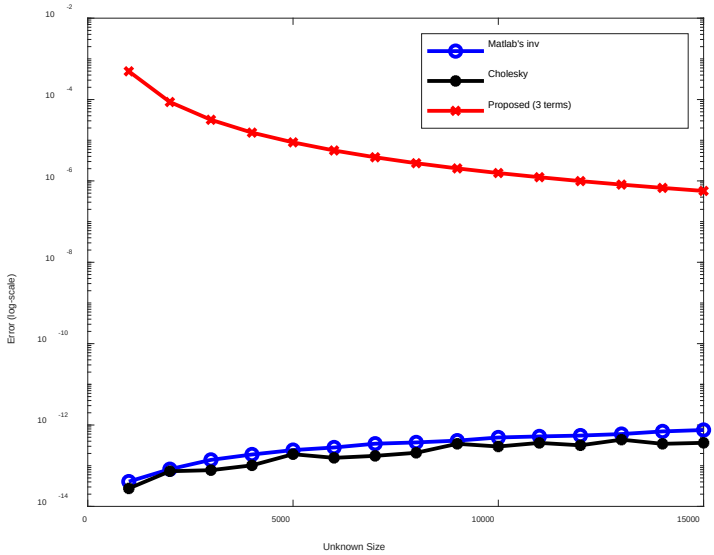

Fig. 5. Error Comparison (3 terms expansion)

Fig. 6 shows the comparison between the proposed algorithm with 4 terms expansion ($I - K + K^2 - K^3$) and other algorithms. **Fig. 7** presents **Fig. 6** again with the time axis in log-scale. When the size of the matrix is small, the performance improvement of our proposed algorithm with 4 terms expansion is not definite compared to other algorithms. However, the performance improvement can be confirmed as the size of the matrix increases. Definitely, 4 terms expansion case is slower than 3 terms expansion case due to the inclusion of more terms, but its error performance is better than 3 terms expansion case.

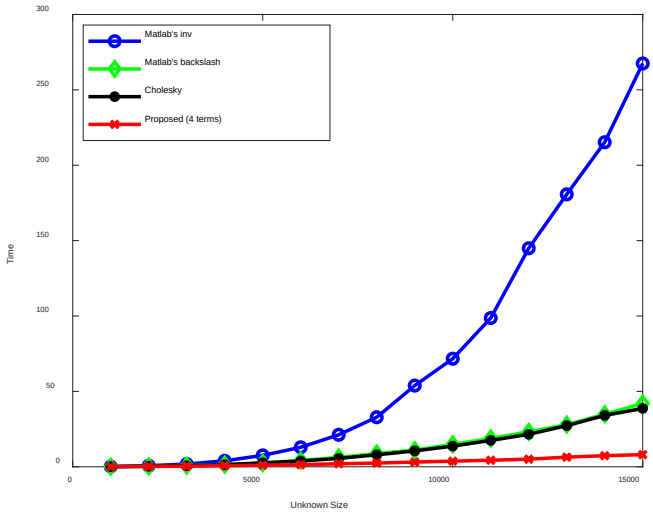

Fig. 6. Computation time comparison (4 terms expansion)

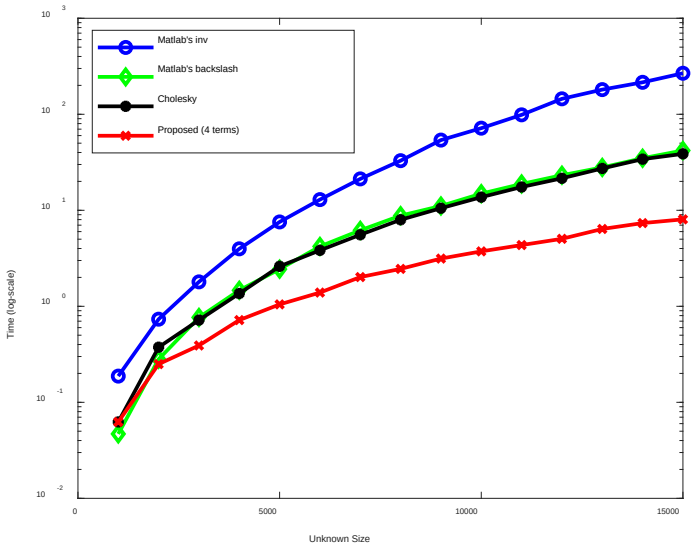

Fig. 7. Computation time comparison (log-scale, 4 terms expansion)

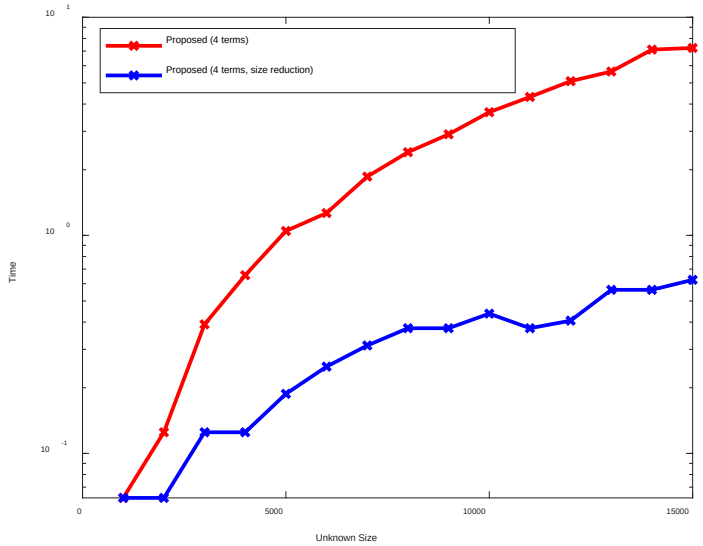

Fig. 8. Performance comparison: the original version *vs.* size reduction version

Fig. 8 represents the performance comparison between the original version based on **Table 1** (colored red) and the size-reduced version based on **Table 2** (colored blue). The simulation results are actually focusing on the original algorithm, which is basically better in $m \gg n$ case. But if $n \gg m$ case, the size-reduced version performs better as shown in **Fig. 8**.

Fig. 9 presents the error analysis of overall algorithms with $\lambda = 10n^2$. The error performance of the 3 terms expansion is also included for the comparison. Since the higher number of terms are included, the closer the real series expansion of the inverse matrix. The error of the 4 terms expansion case is about $\frac{1}{100}$ of that of the 3 terms expansion case.

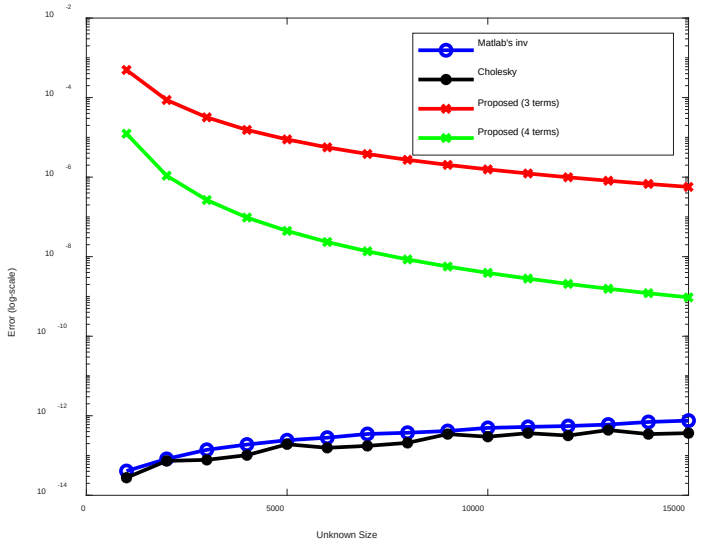

Fig. 9. Error Analysis ($\lambda = 10n^2$)

Fig. 10 illustrates an error analysis with $\lambda = n^2$, which is a smaller λ case than previous error analysis. The smaller λ means the matrix is less likely to be an identity matrix. Thus it may reflect the actual data characteristics in the real-world applications. However, as λ value decreases, the relative error increases because the norm of $\left\|\frac{1}{\lambda}X^TX\right\|$ becomes bigger than the previous case (**Fig. 9**). To be specific, the larger norm of $\left\|\frac{1}{\lambda}X^TX\right\|$ will cause that series expansion is less likely to be equal to the actual inverse part. Thus, λ should be reasonably chosen.

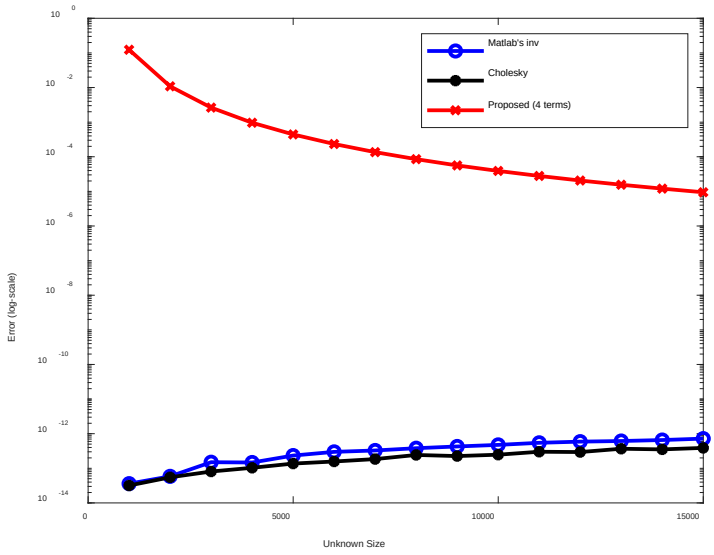

Fig. 10. Error Analysis ($\lambda = n^2$)

5. Conclusion

With the advent of big data applications like radar applications, the data size becomes huge, and the information is required to be processed quickly. In this paper, a new algorithm to improve the computational efficiency of ridge regression is proposed. The proposed algorithm uses series expansion and reuses matrix computation parts without inverse matrix and matrix decomposition. The performance is superior to existing algorithms in terms of speed, and the result shows good accuracy. For the 4 terms expansion example with 15,000 unknown case, the proposed algorithm shows approximately 5.2 times faster than MATLAB's `backslash`-based algorithm, with less than 10^{-8} relative error.

Currently, large regularization constraints λ so as to series expansion equal to the inverse part is using. A more generalized formula to adopt arbitrary regularization constraints λ will be pursued in future work. The improvement of regression performance will contribute to the development of artificial intelligence and machine learning, which are the key of the fourth industrial revolution.

Acknowledgement

Prof. Woochan Lee is the first author. Prof. Moonseong Kim and Prof. Jaeyoung Park are co-corresponding authors. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-00098, Advanced and Integrated Software Development for Electromagnetic Analysis). This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Ministry of Science, ICT & Future Planning (No. NRF -2019R1G1A1007832). The authors sincerely appreciate Mr. Minseong Kim for the administrative and technical support.

References

[1] Y. Ding, F. Liu, T. Rui, and Z. Tang, "Patch based Semi-supervised Linear Regression for Face Recognition," *KSII Transactions on Internet and Information Systems*, vol. 13, no. 8, pp. 3962-3980, 2019. Article (CrossRef Link)

[2] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in *Proc. of the 2015 international workshop on internet of things towards applications*, Seoul, Korea, pp. 7-12, November 1, 2015. Article (CrossRef Link)

[3] W. Zhang, Z. He, and H. Li, "Linear Regression Based Clutter Reconstruction for STAP," *IEEE Access*, vol. 6, pp. 56862-56869, October 2018. Article (CrossRef Link)

[4] P. Fang, W. Jun, X. Jian-jun, and S. Yi-chao, "An optimization method of airborne radar and IRST cooperative location based on ridge regression," in *Proc. of 2018 International Conference on Electronics Technology (ICET)*, Chendu, China, pp. 365-372, May 23-27, 2018. Article (CrossRef Link)

[5] Y. Na, M. Kim, D. Jun, and W. Lee, "Performance Comparison of Methods for Deriving Least Squares Estimator in Repeated Least Squares Method Application," in *Proc. of the 2018 KSII Spring Conference*, Jeju Island, Korea, pp. 25-26, May 27-28, 2018.

[6] S. A. Van De Geer, "Least squares estimation," in *Encyclopedia of Statistics in Behavioral Science*, vol. 2, Chichester: John Wiley & Sons, 2005. Article (CrossRef Link)

[7] G. Williams, *Linear algebra with applications*, Burlington, MA, USA: Jones & Bartlett Learning, 2017.

[8] Y. Kokkinos and K. G. Margaritis, "Managing the computational cost of model selection and cross-validation in extreme learning machines via Cholesky, SVD, QR and eigen decompositions," *Neurocomputing*, vol. 295, pp. 29-45, 2018. Article (CrossRef Link)

[9] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55-67, February 1970. Article (CrossRef Link)

[10] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numerische Mathematik*, vol. 14, pp. 403-420, April 1970. Article (CrossRef Link)

[11] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, Philadelphia, PA, USA: Siam, 1997.

[12] A. Krishnamoorthy and D. Menon, "Matrix inversion using Cholesky decomposition," in *Proc. of 2013 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, Poznan, Poland, pp. 70-72, September 26-28, 2013. Article (CrossRef Link)

[13] J. H. Randall and A. A. Rayner, "The accuracy of least squares calculations with the Cholesky algorithm," *Linear Algebra and its Applications*, vol. 127, pp. 463-502, 1990. Article (CrossRef Link)

[14] C. H. Bischof and G. Quintana-Ortí, "Computing rank-revealing QR factorizations of dense matrices," *ACM Transactions on Mathematical Software (TOMS)*, vol. 24, no. 2, pp. 226-253, June 1998. Article (CrossRef Link)

[15] N. N. Abdelmalek, "Round off error analysis for Gram-Schmidt method and solution of linear least squares problems," *BIT Numerical Mathematics*, vol. 11, no. 4, pp. 345-367, 1971. Article (CrossRef Link)
[16] M. Lira, R. Iyer, A. A. Trindade, and V. Howle, "QR versus Cholesky: a probabilistic analysis," *International Journal of Numerical Analysis and Modeling*, vol. 13, no. 1, pp. 114-121, 2016. Article (CrossRef Link)
[17] W. Lee and D. Jiao, "Fast structure-aware direct time-domain finite-element solver for the analysis of large-scale on-chip circuits," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 5, no. 10, pp. 1477-1487, 2015. Article (CrossRef Link)
[18] V. R. Uslu, E. Egrioglu, and E. Bas, "Finding optimal value for the shrinkage parameter in ridge regression via particle swarm optimization," *American Journal of Intelligent Systems*, vol. 4, no.4, pp. 142-147, 2014. Article (CrossRef Link)
[19] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, Philadelphia, PA, USA: Siam, 2002.
[20] L. Szczecinski and D. Massicotte, "Low complexity adaptation of MIMO MMSE receivers, implementation aspects,"in *Proc. of GLOBECOM '05. IEEE Global Telecommunications Conference*, St. Louis, MO, USA, pp. 2327-2332, November 28-December 2, 2005. Article (CrossRef Link)
[21] C. Song, K. Lee, and I. Lee, "MMSE Based Transceiver Designs in Closed-Loop Non-Regenerative MIMO Relaying Systems," *IEEE Transactions on Wireless Communications*, vol. 9, no. 7, pp. 2310-2319, July 2010. Article (CrossRef Link)
[22] MIT OpenCoursceWare, "MATLAB backslash command to solve Ax = b," [Online] https://ocw.mit.edu/courses/mathematics/18-085-computational-science-and-engineering-i-fall-2008/study-materials/backslash.pdf, Accessed on: January 9, 2021

Woochan Lee received the B.S. and M.S. degrees in electrical engineering from Seoul National University, Seoul, Korea, in 2002 and 2005, respectively, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2016. He was commissioned as a Full-time Lecturer and a First Lieutenant with the Korea Military Academy, Seoul, Korea, from 2005 to 2008. He was a Deputy Director and a Patent Examiner with Korean Intellectual Property Office, Daejeon, Korea, from 2004 to 2017. In 2017, he joined the Department of Electrical Engineering, Incheon National University, Incheon, Korea, where he is currently working as an Associate Professor. His current research interests include computational electromagnetics, numerical analysis, and IoT applications with machine learning.

Moonseong Kim received the M.S. degree in Mathematics, August 2002 and the Ph.D. degree in Electrical and Computer Engineering, February 2007 both from Sungkyunkwan University, Korea. He was a Research Professor at Sungkyunkwan University in 2007. From December 2007 to October 2009, he was a Research Associate in ECE and CSE, Michigan State University, USA. He was a Deputy Director and a Patent Examiner with Korean Intellectual Property Office, Daejeon, Korea, from October 2009 to August 2018. In September 2018, he joined the Seoul Theological University, Bucheon, Korea, where he is currently working as an Assistant Professor and the Head of the Department of IT Convergence Software. His research interests include wired/wireless networking, sensor networking, mobile computing, network security protocols, and simulations/numerical analysis. Since March 2009, he has been an editor of KSII Transactions on Internet and Information Systems (TIIS).

Jaeyoung Park received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea, in 2003, and the M.S. and Ph.D. degrees in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2008 and 2013, respectively. He was a Research Scientist at Korea Institute of Science and Technology from 2013 to 2015 and was a Senior Research Scientist from 2013 to 2021. In 2021, he joined the Department of Computer Engineering at Hongik University, Seoul, Korea, where he is currently working as an Assistant Professor. His research interests include virtual reality, haptics, human-computer interaction (HCI).