AN IMPROVED ALGORITHM FOR GENERALIZED LEAST SQUARES ESTIMATION

XIAO-WEN CHANG
School of Computer Science, McGill University
Montreal, Quebec, Canada H3A 0E9

DAVID TITLEY-PELOQUIN
Department of Bioresource Engineering, McGill University
Ste-Anne-de-Bellevue, Quebec, Canada, H9X 3V9

Abstract. The textbook direct method for generalized least squares estimation was developed by Christopher C. Paige about 40 years ago. He proposed two algorithms. Suppose that the noise covariance matrix, rather than its factor, is available. Both of the Paige’s algorithms involve three matrix factorizations. The first does not exploit the matrix structure of the problem, but it can be implemented by blocking techniques to reduce data communication time on modern computer processors. The second takes advantage of the matrix structure, but its main part cannot be implemented by blocking techniques. In this paper, we propose an improved algorithm. The new algorithm involves only two matrix factorizations, instead of three, and can be implemented by blocking techniques. We show that, in terms of flop counts, the improved algorithm costs less than Paige’s first algorithm in any case and less than his second algorithm in some cases. Numerical tests show that in terms of CPU running time, our improved algorithm is faster than both of the existing algorithms when blocking techniques are used.

1. Introduction. Suppose we have the linear model:

\[ y = Ax + v, \quad v \sim (0, \Sigma), \tag{1.1} \]

where \( y \) is an \( m \)-dimensional observation vector, \( A \in \mathbb{R}^{m \times n} \) is the model matrix with full column rank, \( x \in \mathbb{R}^n \) is an unknown parameter vector, and \( v \) is an \( m \)-dimensional random noise vector with zero mean and covariance matrix \( \Sigma \in \mathbb{R}^{m \times m} \).

Our goal is to estimate the parameter vector \( x \) based on given \( y, A, \) and \( \Sigma \). This problem arises in many applications in science, engineering, and economics.

When \( \Sigma \in \mathbb{R}^{m \times m} \) is symmetric positive definite, to estimate the parameter vector \( x \) in (1.1), typically one solves the generalized least squares (GLS) problem

\[ \min_x (y - Ax)^\top \Sigma^{-1} (y - Ax). \tag{1.2} \]

(Note that in this optimization problem \( x \) is a vector-valued variable, not the true unknown parameter vector in (1.1). For the sake of simplicity, we use the same

2020 Mathematics Subject Classification. Primary: 62J05, 65F20; Secondary: 15A23; 93E24.

Key words and phrases. Generalized least squares estimation, numerical algorithms, matrix factorizations, blocking, efficiency.

* Corresponding author.

This work was supported in part by NSERC of Canada Grant RGPIN-2017-0513.
symbol for both. This abuse of notation is common in the literature on this subject.) The solution of (1.2) is the best linear unbiased estimator (BLUE) of $x$ in (1.1). If the noise $v$ follows the normal distribution $N(0, \Sigma)$, the solution is also the maximum likelihood estimator of $x$ in (1.1).

Suppose that $\Sigma$ has the factorization $\Sigma = BB^T$. In some applications $B$ rather than $\Sigma$ is part of the original data. But if $\Sigma$ rather than $B$ is part of the original data, we can compute the Cholesky factorization of $\Sigma$ to find $B$, which is lower triangular. The textbook approach to solving the estimation problem is Paige's method [7], see for example [3, §4.3.3], [4, §2.7.4], and [5, §6.1.2]. The algorithm given in [7] is numerically reliable. More importantly, it can deal with a singular covariance matrix $\Sigma$ — in this case (1.2) is not defined, but the optimization problem introduced in [7] is well defined and its solution is still the BLUE of $x$ [6].

In [8], Paige proposed a second algorithm to solve the optimization problem introduced in [7]. It assumes that $B$ is lower triangular and takes advantage of this structure. This second algorithm is more efficient than the first algorithm given in [7] in terms of flops. However, one can use blocking techniques for modern computer processors to implement the first algorithm, while it is difficult to do so fully for the second algorithm. Thus, the first algorithm actually runs faster than the second one on modern computers, unless the problem size is small.

In this paper, we consider the case that $\Sigma$ rather than $B$ is part of the original data. We propose a more efficient algorithm for the GLS estimation problem. In terms of flop count, our algorithm is faster than Paige’s first algorithm in any case and Paige’s second algorithm in some cases. It can take advantage of blocking techniques and is faster than Paige’s two algorithms in terms of CPU time. This performance improvement is achieved by reducing the number of matrix factorizations from three required by Paige’s two algorithms to two required by our algorithm.

The rest of the paper is organized as follows. In Section 2 we briefly introduce Paige’s two algorithms. In Section 3 we present our improved algorithm. We compare the flop counts of the three algorithms in Section 4. In Section 5 we discuss the efficient implementation of our algorithm using block computations. In Section 6 we give numerical test results to show that the performance of our algorithm is better than that of the two existing algorithms. Finally, we give a brief summary in Section 7.

2. Paige’s approach. In this section, we review Paige’s approach to the estimation problem when $\Sigma$ rather than its factor $B$ is directly available. Specifically, we present the main steps of Paige’s two algorithms. For simplicity, we assume that $A$ has full column rank and $\Sigma$ is nonsingular. For the more general case, see [7].

Paige’s approach first computes the Cholesky factorization of $\Sigma$:

$$\Sigma = BB^T, \quad (2.3)$$

where $B \in \mathbb{R}^{m \times m}$ is lower triangular with positive diagonal elements. For the random vector $v$ in (1.1) we can find a random vector $u$ such that

$$v = Bu, \quad u \sim (0, I).$$

Thus the linear model (1.1) can be replaced by

$$y = Ax + Bu, \quad u \sim (0, I).$$
It is easy to see that the optimization problem (1.2) is equivalent to
\[
\min_{\mathbf{x}, u} \mathbf{u}^T \mathbf{u}
\]
\[\text{s.t. } \mathbf{y} = \mathbf{A} \mathbf{x} + \mathbf{Bu}. \tag{2.4}\]

Notice that this new formulation does not involve any inverse and it is well defined even if \(\mathbf{B}\) is not invertible.

2.1. **Algorithm 1.** Paige’s first algorithm for solving (2.4) was proposed in [7]. After the Cholesky factorization (2.3) is computed, the algorithm computes the following factorizations by Householder transformations:
\[
\mathbf{Q}^T \mathbf{A} = \begin{bmatrix} \mathbf{0} \\ \mathbf{L}_\mathbf{A} \end{bmatrix}_{m-n}^n, \quad \mathbf{Q}^T \mathbf{B} \mathbf{P} = \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{L}_2 \end{bmatrix}_{m-n}^m, \tag{2.5}\]

where \(\mathbf{Q} = [\mathbf{Q}_1, \mathbf{Q}_2] \in \mathbb{R}^{m \times m}\) and \(\mathbf{P} = [\mathbf{P}_1, \mathbf{P}_2] \in \mathbb{R}^{m \times m}\) are orthogonal, and \(\mathbf{L}_\mathbf{A}\) and \(\mathbf{L}\) are nonsingular lower triangular. In (2.5), the first factorization is the QL factorization of \(\mathbf{A}\) and the second one is the LQ factorization of \(\mathbf{Q}^T \mathbf{B}\). Note that \(\mathbf{Q}\), as a product of Householder transformations, is not formed explicitly in computations for efficiency. Here we make a remark. Actually in the original algorithm given in [7], the two lower triangular factors in the above factorizations are upper triangular, but for the sake of consistency with the second algorithm to be introduced later, we use lower triangular matrices in (2.5) instead, and there is no essential difference. Then with
\[
\mathbf{z} := \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{P}_1^T \mathbf{u} \\ \mathbf{P}_2^T \mathbf{u} \end{bmatrix}, \tag{2.6}\]

we transform the optimization problem (2.4) to
\[
\min_{\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2} ||\mathbf{z}_1||^2_2 + ||\mathbf{z}_2||^2_2
\]
\[\text{s.t. } \begin{bmatrix} \mathbf{Q}^T_1 \mathbf{y} \\ \mathbf{Q}^T_2 \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{L}_\mathbf{A} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{L}_2 \end{bmatrix}_{21} \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix}. \tag{2.7}\]

Obviously the optimal solution \((\hat{\mathbf{x}}, \hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2)\) should satisfy
\[
\mathbf{L}_1 \hat{\mathbf{z}}_1 = \mathbf{Q}^T_1 \mathbf{y}, \quad \mathbf{L}_\mathbf{A} \hat{\mathbf{x}} = \mathbf{Q}^T_2 \mathbf{y} - \mathbf{L}_{21} \hat{\mathbf{z}}_1, \quad \hat{\mathbf{z}}_2 = \mathbf{0}. \tag{2.8}\]

The two lower triangular systems can easily be solved to give \(\hat{\mathbf{x}}\), the estimator of the true parameter vector \(\mathbf{x}\) in (1.1). It is easy to show that the error covariance matrix of the estimator \(\hat{\mathbf{x}}\) satisfies
\[
\text{cov}\{\hat{\mathbf{x}} - \mathbf{x}\} = \mathbf{L}_{\mathbf{A}}^{-1} \mathbf{L}_2 \mathbf{L}^T_{\mathbf{A}}. \tag{2.9}\]

The lower triangular \(\mathbf{L}_{\mathbf{A}}^{-1} \mathbf{L}_2\) is the Cholesky factor of \(\text{cov}\{\hat{\mathbf{x}} - \mathbf{x}\}\). This structure is useful when \(\text{cov}\{\hat{\mathbf{x}} - \mathbf{x}\}\) is used for some other computations. Note that \(\mathbf{L}_2\) is not involved in finding \(\hat{\mathbf{x}}\). Thus, if one wants to find only \(\hat{\mathbf{x}}\), not its error covariance matrix, \(\mathbf{L}_2\) in (2.5) does not need to be lower triangular. Using this fact, we can reduce the computational cost. For convenience, the whole algorithm described above will be referred to as Paige’s Algorithm 1.

Now we look at the computational cost of the algorithm. The whole computational cost for obtaining the estimator \(\hat{\mathbf{x}}\) is dominated by the cost of computing (2.3) and (2.5). Computing (2.3) costs about \(m^3/3\) flops; see, e.g., [5, p. 164]. Computing the factorization of \(\mathbf{A}\) in (2.5) by the Householder transformations costs about

\[
\begin{align*}
\min_{\mathbf{x}, u} & \quad \mathbf{u}^T \mathbf{u} \\
\text{s.t.} & \quad \mathbf{y} = \mathbf{A} \mathbf{x} + \mathbf{Bu}. 
\end{align*}
\]
2mn^2 - 2n^3/3 flops, which is identical to the cost of computing the QR factorization by the Householder transformations; see, e.g., [5, p. 249]. Computing \( Q^T B \) by using the Householder transformations costs about \( 4m^2 n - 2mn^2 \) flops and computing its factorization in (2.5) by Householder transformations costs \( 4m^3/3 \) flops. Thus, the total cost of the first algorithm is about

\[
c_1(m, n) := 5m^3/3 + 4m^2 n - 2n^3/3 \text{ flops.}
\] (2.9)

2.2. Algorithm 2. Paige’s second algorithm proposed in [8] takes advantage of the lower triangular structure of \( B \). After the Cholesky factorization of \( \Sigma \) in (2.3) is computed, the algorithm performs orthogonal transformations by using the lower triangular structure of \( B \):

\[
Q^T [y, A, B] = \begin{bmatrix}
1 & 0 & 0 \\
0 & I_\nu & 0 \\
0 & 0 & P
\end{bmatrix}
\begin{bmatrix}
0 & 0 & L_3 \\
\eta & 0 & g^T \\
y & L_A & L_{21} \ h & L_{2j}
\end{bmatrix}
\begin{bmatrix}
m-n-1 \\
1 & n & m-n-1 & 1 & n
\end{bmatrix},
\] (2.10)

where \( Q \in \mathbb{R}^{m \times m} \) and \( P \in \mathbb{R}^{m \times m} \) are orthogonal, and \( L_A, L_1, \) and \( L_2 \) are lower triangular. For example, for \( m = 5 \) and \( n = 3 \), the transformation (2.10) can be described by

\[
\begin{bmatrix}
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times
\end{bmatrix}
\begin{bmatrix}
\times \\
\times \\
\times \\
\times \\
\times
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\times \\
\times \\
\times \\
\times \\
\times
\end{bmatrix}
\begin{bmatrix}
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times
\end{bmatrix}.
\]

The transformation consists of \( m - 1 \) steps, starting from the top right corner of \([y, A]\). In each step, one complete superdiagonal of \([y, A]\) is annihilated while the lower triangular structure of \( B \) is preserved. When \( k \leq n \), in step \( k \), \( k \) rotations are performed from the left to annihilate the \( k \) elements in the superdiagonal of \([y, A]\). When \( k > n \), in step \( k \), \( n+1 \) rotations are performed from the left to annihilate the \( n+1 \) elements in the superdiagonal of \([y, A]\). After each rotation from the left is performed, a rotation from the right is performed on \( B \) to annihilate the element produced by the rotation from the left. In the above example, in step 2, we apply two rotations to \([y, A, B]\) from the left to annihilate \( a_{23} \) and \( a_{12} \). After each of the two rotations is applied, we apply a rotation to \( B \) from the right to annihilate the element of \( B \) brought by the rotation from the left, namely \( b_{23} \) and \( b_{12} \). This step can be described schematically as follows:

\[
\begin{array}{c}
\times \times \ \overset{2}{\bullet} \\
\times \times \ \overset{1}{\bullet}
\end{array}
\rightarrow
\begin{array}{c}
\times \times \ \overset{23}{\bullet} \\
\times \times \ \overset{21}{\bullet}
\end{array}
\]

where \( \overset{i}{\bullet} \) indicates the element is eliminated in the \( i \)-th rotation from the left, while \( \overset{i'}{\bullet} \) indicates this element is generated by the \( i \)-th rotation from the left and is eliminated in the \( i' \)-th rotation from the right.
Define
\[ z := P^T u = \begin{bmatrix} z_1 \\ \zeta \\ z_2 \end{bmatrix} \]
\[ = \begin{bmatrix} 1 \\ n \end{bmatrix} \]
\[ m - n - 1. \]
\[ \text{(2.11)} \]
Then, after the orthogonal transformation (2.10), the optimization problem (2.4) becomes
\[ \min \|z_1\|^2_2 + \zeta^2 + \|z_2\|^2_2 \]
\[ \text{s.t. } \eta = \begin{bmatrix} 0 \\ \eta \end{bmatrix} \]
\[ y = \begin{bmatrix} 0 \\ L_A \end{bmatrix} x + \begin{bmatrix} L_{11} & 0 \\ h & L_{22} \end{bmatrix} \begin{bmatrix} z_1 \\ \zeta \\ z_2 \end{bmatrix}. \]
Thus, the optimal solution \((\hat{x}, \hat{z}_1, \hat{\zeta}, \hat{z}_2)\) satisfies
\[ \hat{z}_1 = 0, \quad \hat{\zeta} = \eta / \rho, \quad \hat{z}_2 = 0, \quad L_A \hat{x} = \bar{y} - h \hat{\zeta}. \]
\[ \text{(2.12)} \]
We can also easily show that
\[ \text{cov}\{\hat{x} - x\} = L_A^{-1} L_2 L_2^T L_A^{-T}. \]
In a practical implementation, we can make the computation more efficient by avoiding unnecessary operations. In fact, after \(n + i\) steps of the orthogonal transformation (2.10), the first \(i\) elements of \(y\) will be zero, as will the first elements of \(z\) in (2.11) (cf. the first equality in (2.12)), and so the first \(i\) columns of \(B\) will no longer have any effect on the results, and can be ignored.

Now we consider the cost of the algorithm. According to [8, eq. (4.4)], the orthogonal transformation (2.10) can be implemented in \(m^2 n / 2 + mn^2 - 2n^3 / 3\) operations, where one operation means the work of the multiplication of a Givens rotation and a vector:
\[ \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \]
\[ \alpha \]
which involves 6 flops. Thus, the cost of the orthogonal transformation (2.10) is about \(3n^2 n + 6mn^2 - 4n^3\) flops. Adding the cost of computing (2.3) leads to the total cost of the second algorithm:
\[ c_2(m, n) := m^3 / 3 + 3m^2 n + 6mn^2 - 4n^3 \text{ flops.} \]
\[ \text{(2.13)} \]
Here we make a remark. If fast rotations are used instead of the standard ones, the cost of the whole algorithm will be less, see [8]. But on modern processors the gain in speed is modest. Thus, fast rotations are not used in modern software packages.

3. An improved algorithm. In this section we propose to modify Paige’s Algorithm 1 to make it more efficient. Algorithm 1 involves three matrix factorizations, while our modified algorithm involves only two matrix factorizations.

In the following we describe the main steps of our improved algorithm. We first compute the QL factorization of \(A\) using Householder transformations, as in (2.5):
\[ Q^T A = \begin{bmatrix} 0 \\ L_A \end{bmatrix}, \]
\[ \text{(3.14)} \]
where \(Q = H_n H_{n-1} \cdots H_1\). Each \(H_j\) is the Householder transformation used to annihilate \(A(1 : j + 1, j)\), and has the form:
\[ H_j = \begin{bmatrix} I_{m-n+j} & -\tau_j u_j u_j^T & 0 \\ 0 & I_{n-j} \end{bmatrix}, \quad j = n, n - 1, \ldots, 1. \]
\[ \text{(3.15)} \]
As in Paige’s Algorithm 1, \(Q\) is not formed explicitly in the computations.
With the orthogonal $Q \in \mathbb{R}^{m \times m}$, the linear model (1.1) can be transformed to

$$Q^T y = \begin{bmatrix} 0 \\ L_A \end{bmatrix} x + Q^T v, \quad Q^T v \sim (0, Q^T \Sigma Q).$$

(3.16)

Then we compute the covariance matrix of $Q^T v$:

$$\Sigma := Q^T \Sigma Q.$$  

(3.17)

Details on how to compute $\Sigma$ in an efficient way are given later.

After obtaining $\Sigma$, we compute its Cholesky factorization:

$$\Sigma = LL^T.$$  

(3.18)

Here the triangular factor $L$ is exactly the same as the $L$ in (2.5) if we require that the diagonal elements of $L$ be positive. In fact, from (2.5), (2.3), (3.17) and (3.18), we have

$$LL^T = \left( Q^T \Sigma Q \right)^T = \left( Q^T \Sigma Q \right) = \Sigma = LL^T.$$  

(3.19)

Thus, due to the uniqueness of the Cholesky factorization, $L$ is just $L$ if the diagonal elements of $L$ are chosen to be positive.

For the random vector $Q^T v$ in (3.16), based on (3.17), (3.18) and (3.19), we can find a random vector $z$ such that

$$Q^T v = Lz := \begin{bmatrix} L_1 \\ L_{21} \\ L_2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}, \quad z \sim (0, I).$$

Then it is easy to verify that the optimization problem (1.2) is equivalent to

$$\min \|z_1\|_2^2 + \|z_2\|_2^2$$

s.t. $$\begin{bmatrix} Q^T y \\ Q^T z \end{bmatrix} = \begin{bmatrix} 0 \\ L_A \end{bmatrix} x + \begin{bmatrix} L_1 \\ L_{21} \\ L_2 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}.$$  

This has the same form as (2.7). The rest of the modified algorithm is exactly the same as that of Paige’s first algorithm introduced in Section 2.1.

Now we show how to compute $\Sigma$ in (3.17) in an efficient way. Let

$$\Sigma_k := \left( H_n \cdots H_{n-k+1} \right)^T \Sigma \left( H_n \cdots H_{n-k+1} \right) := \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{12}^T & \Sigma_{22} \end{bmatrix}.$$  

(3.20)

Thus, $\Sigma = \Sigma_n$. To simplify notation, we write $H_{n-k} = \begin{bmatrix} I - \tau uu^T & 0 \\ 0 & I \end{bmatrix}$ (cf. (3.15)).

Then with $\Sigma_0 := \Sigma$,

$$\Sigma_{k+1} = H_{n-k} \Sigma_k H_{n-k} = \begin{bmatrix} (I - \tau uu^T) \Sigma_{11} (I - \tau uu^T) \\ (I - \tau uu^T) \Sigma_{12}^T \\ (I - \tau uu^T) \Sigma_{22} \end{bmatrix}$$  

(3.21)

for $k = 0, 1, \ldots, n - 1$. For the (1,1) block of $\Sigma_{k+1}$,

$$\begin{aligned}
(I - \tau uu^T) \Sigma_{11} (I - \tau uu^T) &= \Sigma_{11} - (\tau u) \left\{ \Sigma_{11} u - [(\tau u)^T (\Sigma_{11} u) / 2] u \right\}^T \\
&\quad - \left\{ \Sigma_{11} u - [(\tau u)^T (\Sigma_{11} u) / 2] u \right\} (\tau u)^T \\
&\quad + \Sigma_{11} - (\tau u) w^T - w (\tau u)^T,
\end{aligned}$$

(3.22)

where $w := \Sigma_{11} u - [(\tau u)^T (\Sigma_{11} u) / 2] u$. For the (1,2) block of $\Sigma_{k+1}$,

$$\begin{aligned}
(I - \tau uu^T) \Sigma_{12} &= \Sigma_{12} - (\tau u) (u^T \Sigma_{12}).
\end{aligned}$$

(3.23)
The two equations (3.21) and (3.22) show how to compute $\Sigma_{k+1}$ from $\Sigma_k$. Due to the symmetry, we need only compute the lower (or upper) triangular part of $\Sigma_{k+1}$.

For clarity, we describe the modified algorithm as follows.

**Algorithm 3 Improved Algorithm**

1. Compute the QL factorization of $A$: $H_n \cdots H_1 A = \begin{bmatrix} 0 \\ L_A \end{bmatrix}$
2. Compute $\bar{\Sigma} = H_1 \cdots H_n \Sigma H_n \cdots H_1$
3. Compute $\tilde{y} = H_1 \cdots H_n y$
4. Compute the Cholesky factorization of $\bar{\Sigma}$: $\bar{\Sigma} = L L^T$
5. Solve $L(1:n,1:n) z_1 = \tilde{y}(1:n)$ for $z_1$
6. Solve $L_A \hat{x} = \tilde{y}(n+1:m) - L(n+1:m,1:n) z_1$ for $\hat{x}$

The computational cost of the algorithm is dominated by lines 1, 2, and 4. The cost of line 1 is about $2(m - n/3)n^2$ flops. Now we look at the cost of line 2. From (3.21) and (3.22), by simple calculations we see computing $\Sigma_{k+1}$ from $\Sigma_k$ costs about $4(m - k)^2 + 4(m - k)k$ flops. Thus the cost of line 2 is about

$$\sum_{k=0}^{n-1} (4(m - k)^2 + 4(m - k)k) \approx 8m^2n - 6mn^2 + 4n^3/3$$

flops. The cost of line 4 is $m^3/3$. Thus, the total cost of the modified algorithm is about

$$c_3(m, n) := m^3/3 + 8mn^2 - 4mn^2 + 2n^3/3 \text{ flops.}$$

(3.23)

Finally we make a remark about the numerical stability of the improved algorithm. If the given data is $(y, A, B)$, [7] showed that Algorithm 1 is numerically stable. If the given data is $(y, A, \Sigma)$, it is easy to extend the proof given in [7] to show the algorithm is still numerically stable, as long as a standard numerically stable algorithm is used for computing the Cholesky factorization (2.3). The conclusion of numerical stability also applies to Algorithm 2, which uses Givens rotations instead of Householder transformations used in Algorithm 1. Our improved algorithm can be regarded as a special implementation of Paige’s approach, and is also numerically stable. We could modify the proof given in [7] to show this, but we omit it for simplicity.

4. Comparison of flops. In this section we compare the costs of the three algorithms in terms of flops. To make reading easy, we list the three quantities in (2.9), (2.13) and (3.23) here:

| Algorithm   | $c_1(m, n)$ | $c_2(m, n)$ | $c_3(m, n)$ |
|-------------|-------------|-------------|-------------|
| Algorithm 1 | $5m^3/3 + 4m^2n - 2n^3/3$ | $m^3/3 + 3m^2n + 6mn^2 - 4n^3$ | $m^3/3 + 8mn^2 - 4mn^2 + 2n^3/3$ |
These costs have the following relationships:

\[ c_1(m, n) \leq c_2(m, n), \quad n \leq m \leq 5n/4 \]
\[ c_1(m, n) > c_2(m, n), \quad m > 5n/4 \]
\[ c_3(m, n) \leq c_1(m, n), \quad m \geq n \]
\[ c_3(m, n) < c_2(m, n), \quad n \leq m < (1 + 1/\sqrt{15})n \]
\[ c_2(m, n) < c_3(m, n) < 1.61c_2(m, n), \quad m > (1 + 1/\sqrt{15})n \]
\[ c_3(m, n) \approx \frac{1}{5}c_1(m, n) \approx c_2(m, n), \quad m \gg n. \] (4.25)

It is not difficult to verify these relationships, except the second inequality in (4.24), which was obtained as follows: set \( f(x) := c_3(x, 1)/c_1(x, 1) \) with \( x \geq 1 \) and find its maximizer \( \hat{x} = 7.6915 \), leading to the maximum value \( f(\hat{x}) = 1.6012 \).

5. **Block computation.** On modern computer processors, data communication time is more crucial than floating point operation time. Thus, we should not emphasize too much the importance of the flop count of an algorithm. Blocking techniques have been widely used to reduce data communication time, see, e.g., [5, §1.5.4].

For Paige’s Algorithm 1, there are block algorithms to compute the Cholesky factorization of \( \Sigma \) in (2.3), the QL factorization of \( A \), and the LQ factorization of \( Q^T B \) in (2.5). Specifically, we can use LAPACK [1] to compute them. The only thing we need to address is computing \( Q^T \). Notice that we do not form \( Q \) explicitly. When we apply block Householder transformations to \( A \) in computing its QL factorization, we also apply them to \( B \).

For Paige’s Algorithm 2, unfortunately, it is difficult to incorporate blocking techniques into the computation of (2.10), which involves rotations.

For our improved algorithm (see Algorithm 3), the two matrix factorization in lines 1 and 4 can be implemented using blocking techniques. We need only to show how to compute \( Q^T \Sigma Q \) in line 2 by blocking techniques. Recall from (3.14) and (3.15) that \( Q = H_1 \) comes from the QL factorization of \( A \). Suppose we use a block size \( d \), where for simplicity we assume \( n \) is divisible by \( d \). The block algorithm for computing (3.14) consists of \( n/d \) steps. At step \( i + 1 \), we accumulate \( \mathbf{H}^{(i+1)} := \mathbf{H}_{k_i} \mathbf{H}_{k_i-1} \ldots \mathbf{H}_{k_i-d+1} \), where \( k_i = n - id \). Thus,

\[ Q = \mathbf{H}^{(1)} \mathbf{H}^{(2)} \ldots \mathbf{H}^{(n/d)}. \]

Each \( \mathbf{H}^{(i+1)} \) has the following form:

\[ \mathbf{H}^{(i+1)} = \begin{bmatrix} \mathbf{I}_{m-n+k_i} - \mathbf{U} \mathbf{T}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{n-k_i} \end{bmatrix}, \]

where \( \mathbf{U} \in \mathbb{R}^{(m-n+k_i) \times d} \) is formed by the Householder vectors \( \mathbf{u}_{k_i-d+1}, \mathbf{u}_{k_i-d+2}, \ldots, \mathbf{u}_{k_i} \) (see (3.15)) and \( \mathbf{T} \in \mathbb{R}^{d \times d} \) is lower triangular. The matrix \( \mathbf{I}_{m-n+k_i} - \mathbf{U} \mathbf{T}^T \) is called a Householder block reflector. For details about computing \( \mathbf{U} \) and \( \mathbf{T} \), see the subroutine \texttt{dgeqrf} of LAPACK [1]. For a variant of the Householder block reflectors used in computing the block QR factorization, see [9].

For computing \( Q^T \Sigma Q \), suppose that at the beginning of step \( i + 1 \), we have obtained

\[ \Sigma^{(i)} := (\mathbf{H}^{(i)})^T \ldots (\mathbf{H}^{(1)})^T \Sigma \mathbf{H}^{(1)} \ldots \mathbf{H}^{(i)} := \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{12}^T & \Sigma_{22} \end{bmatrix}, \]
where $\Sigma_{22}$ is $n - k_i$ by $n - k_i$. Now we would like to compute $\Sigma^{(i+1)}$, which is symmetric. Note that

$$
\Sigma^{(i+1)} = (H^{(i+1)})^T \Sigma^{(i)} H^{(i+1)}
$$

$$
= \begin{bmatrix}
(I - UT^T U^T) \Sigma_{11} (I - UT^T U^T) & (I - UT^T U^T) \Sigma_{12} \\
((I - UT^T U^T) \Sigma_{12})^T & \Sigma_{22}
\end{bmatrix}.
$$

The $(1,1)$ block of $\Sigma^{(i+1)}$ satisfies

$$
(I - UT^T U^T) \Sigma_{11} (I - UT^T U^T) = \Sigma_{11} - (UT^T) \left[ \Sigma_{11} U - (UT^T)(UT^T \Sigma_{11} U)/2 \right]^T - \left[ \Sigma_{11} U - (UT^T)(UT^T \Sigma_{11} U)/2 \right] (UT^T)^T.
$$

(5.26)

The right hand side of (5.26), where the second term is the transpose of the third term, shows how to compute the $(1,1)$ block in an efficient way. The $(1,2)$ block of $\Sigma^{(i+1)}$ satisfies

$$
(I - UT^T U^T) \Sigma_{12} = \Sigma_{12} - (UT^T)(UT^T \Sigma_{12}).
$$

(5.27)

The right hand side of (5.27) shows how to compute the $(1,2)$ block of $\Sigma^{(i+1)}$. More implementation details are given in the next section.

6. **Numerical tests.** We provide some numerical examples to compare the running times of the three algorithms. As all of three algorithms are numerically stable, we will not compare the accuracy of the computed solutions.

We implement Paige’s Algorithm 1, Paige’s Algorithm 2, and our improved algorithm in the Julia language [2]. We did not use more popular MATLAB because it is not easy to call LAPACK routines directly from it. All tests are performed on a 3.60 GHz Intel Core processor with 12 GB of RAM running the Windows 10 operating system.

For Paige’s Algorithm 1, we first compute the Cholesky factorization of $\Sigma$ using Julia’s built-in function `cholesky`. Then we compute the QL factorization of $A$ (see (2.5)) by calling the LAPACK routine `dgeqlf`. The factor $Q$ is not formed explicitly. We use the LAPACK routine `dormql` to form the products $Q^T y$ and $Q^T B$ using Householder block reflectors. Finally, we use the LAPACK routine `dgelqf` to compute $L$ in the LQ factorization of $Q^T B$, and Julia’s backslash function to solve the two lower-triangular systems in (2.8).

For Paige’s Algorithm 2, we once again compute the Cholesky factorization of $\Sigma$ using Julia’s built-in function `cholesky`. We then compute the factorizations in (2.10) using rotations implemented in Julia. Finally, we use Julia’s backslash function to solve the lower-triangular system in (2.12).

For our improved algorithm (see Algorithm 3), we first compute the QL factorization of $A$ using the LAPACK routine `dgeqlf`. As in Paige’s Algorithm 1, the factor $Q$ is not formed explicitly. For computing the $(1,1)$ block of $\Sigma$ as per (5.26), we first form $T$ using the LAPACK routine `dlarft`. Next, we form the products $M_1 = UT^T$ using `dtrmm`, $M_2 = \Sigma_{11} U$ using `dsymm`, and $M_3 = U^T M_2$ using `dgemm`. We then compute the update $M_3 \leftarrow M_2 - M_1 M_3$ using `dgemm` and the symmetric update $\Sigma_{11} \leftarrow \Sigma_{11} - M_1 M_2^T - M_2 M_1^T$ using `dsyr2k`. The computation of the $(1,2)$ block in (5.27) is done similarly. Finally, we use Julia’s `cholesky` and backslash built-in functions to compute the Cholesky factorization of $\Sigma$ and solve the resulting lower-triangular system.
In our tests, \( A \) is a random \( m \times n \) matrix, created using Julia’s built-in function \texttt{randn}, with different values of \( m \) and \( n = 80 \). The covariance matrix \( \Sigma \) has the form

\[
\Sigma = D + v_1 v_1^T + v_2 v_2^T,
\]

where \( v_1 \) and \( v_2 \) are random vectors created using \texttt{randn} and \( D \) is diagonal containing logarithmically equally-spaced values between \( 10^{-2} \) and \( 10^2 \). The vector \( y \) is formed as in (1.1) with \( x = [\sin(1), \ldots, \sin(n)]^T \). (Note that for a performance comparison of direct methods for full problems, the specific data instance is not really that relevant.)

![Figure 6.1. Average CPU running time per solve over 10 solves for \( A \in \mathbb{R}^{m \times n} \) with \( n = 80 \), \( m = 100 : 200 : 3500 \).](image)

In Figure 6.1 we show the CPU running time of the three algorithms when \( n = 80 \), for different values of \( m \). These are average running times per solve over 10 solves. In these tests, Paige’s Algorithm 2 is always slower than Paige’s Algorithm 1, despite requiring fewer flops. As mentioned earlier, this is because the dominant part of latter takes advantage of blocking, while the dominant part of the former does not. In these tests, the improved algorithm is always the fastest, increasingly so as the difference between \( m \) and \( n \) increases. For example, in Figure 6.1, when \( m = 3500 \), the improved algorithm is approximately 4.4 times faster than Paige’s first algorithm. This is close to the ratio predicted by the flop count of the two algorithms, \( c_1(m, n)/c_3(m, n) \approx 5 \) when \( m \gg n \), in (4.25).

7. \textbf{Summary and comments.} Based on Paige’s approach, we have proposed an improved algorithm for generalized least squares estimation. Our algorithm reduces the number of matrix factorizations to two, from three required by Paige’s two algorithms, when the noise covariance matrix \( \Sigma \), rather than its factor, is part of the original data. The improved algorithm is suitable for block computations on modern computer processors, like Paige’s Algorithm 1, but it is faster in terms of both flops and CPU running times. When the number of rows of \( A \) is much larger than its number of columns, the former can be almost five times as fast as the latter in terms of both flops and CPU running time. The improved algorithm is also numerically stable, like Paige’s two algorithms. Although it is presented here for full-column-rank \( A \) and nonsingular \( \Sigma \), our algorithm can easily be extended to deal with any more general case, as with Paige’s two algorithms.
Acknowledgements. The first author dedicates this in memory of his friend, collaborator and teacher Wenyu Sun. Both authors would like to thank Chris Paige for his encouragement and helpful comments.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, LAPACK Users’ Guide, SIAM, Philadelphia, PA, Third Edition, 1999.

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B Shah, Julia: A fresh approach to numerical computing, SIAM Review, 59 (2017), 65–98.

[3] Å. Björck, Numerical Methods for Least Squares Problems, SIAM, Philadelphia, PA, 1996.

[4] Å. Björck, Numerical Methods in Matrix Computations, Springer, Philadelphia, PA, 2015.

[5] G. H. Golub and C. F. Van Loan, Matrix Computations, The Johns Hopkins University Press, Baltimore, Maryland, 4th edition, 2013.

[6] S. Kourouklis and C. C. Paige, A constrained least squares approach to the general Gauss-Markov linear model, Journal of the American Statistical Association, 76 (1981), 620–625.

[7] C. C. Paige, Computer solution and perturbation analysis of generalized linear least squares problems, Math. Comput., 33 (1979), 171–183.

[8] C. C. Paige, Fast numerically stable computations for generalized linear least squares problems, SIAM J. Num. Anal., 16 (1979), 165–171.

[9] R. Schreiber and C. Van Loan, A storage efficient WY representation for products of Householder transformations, SIAM J. of Sci. Stat. Computing, 10 (1991), 53–57.