SUPPLEMENTARY
Householder Sketch for Accurate and Accelerated
Least-Mean-Squares Solvers

Jyotikrishna Dass and Rabi Mahapatra
Texas A&M University
June 11, 2021

APPENDIX

A. Theorems and Proofs

Theorem 2.2 (Householder Sketch). Let $X \in \mathbb{R}^{n \times d}$ be the original data matrix, $y \in \mathbb{R}^n$ be the corresponding output label or response vector, and $n \gg d$. Let $X = QR$ be Householder QR decomposition. Then, $(R, Q^T y)$ is a memory-efficient and theoretically accurate sketch of original data $(X, y)$ such that $X^T X = R^T R$, and has memory footprint of $(\frac{d(d+3)}{2})$ elements, computed in time $O(nd^2)$.

Proof. From Equations 1 and 2, and $X = QR$, where $QQ^T = Q^T Q = I$

$$\|Xw - y\|_2 = \|QRw - y\|_2 = \|QRw - QQ^T y\|_2 = \|Q\|_2 \|Rw - Q^T y\|_2 = \|Rw - Q^T y\|_2$$

(Accurate sketch) So, it is possible to replace the original data $(X, y)$ used in existing LMS solvers with $(R, Q^T y)$ which preserves the covariance $X^T X = R^T Q^T QR = R^T R$ and solves the optimization problem accurately. For example, Ridge regression with ridge parameter $\lambda$ solves $(X^T X + \lambda I)w = X^T y$ in primal form which can be reformulated to $(R^T R + \lambda I)w = R^T (Q^T y)$.

(Memory savings) $R$ is a $d \times d$ upper triangular matrix with $(\frac{d(d-1)}{2})$ elements above the diagonal and $d$ on the diagonal resulting in $(\frac{d(d+1)}{2})$ elements compared to original data matrix $X$ that has $nd$ elements. $Q^T y$ is a reflected response vector. It is to be noted that only top $d$ rows of $Q^T$ will be sufficient to compute $Q^T y$ since $n \gg d$. Hence, reflected response vector $(Q^T y)$ is of size $d$ compared to the original LMS formulation with response vector $y$ of size $n$. Hence, the total memory footprint of $(R, Q^T y)$ is $O(\frac{d(d+3)}{2})$ elements which makes it memory-efficient than the original $(X, y)$ occupying $n(d + 1)$ space.

(Time complexity) The above sketch $(R, Q^T y)$ is computed via Householder QR decomposition (HOUSEHOLDER-QR in Step 1 of Algorithm 1) of $X$ which generates upper triangular matrix $R$, and orthonormal matrix $Q$ that is internally stored as Householder reflectors. The time complexity of the above decomposition is $O(nd^2 - d^3/3)$ (Golub & Van Loan [2012]). Calculation of $Q^T y$ is done implicitly by applying Householder reflectors to the response vector $y$ (MULTIPLY-QC in Step 2 of Algorithm 1) in time $O(nd)$ (Golub & Van Loan [2012]). Hence, it can be seen that the total computation time for the sketch $(R, Q^T y)$ is $O(nd^2 + nd - d^3/3)$ which results in $O(nd^2)$ for $n \gg d$. □
Theorem 2.3. Let \( X \in \mathbb{R}^{n \times d}, y \in \mathbb{R}^n, (R, Q^Ty) := \text{HOUSEHOLDER-SKETCH}(X, y) \) that accelerates primal ridge solver via \text{RIDGE-QR}. Then, \((R, Q^Ty)\) is also the Householder sketch for the corresponding Kernel Ridge Regression problem that accelerates the dual problem via \text{KERNELRIDGE-QR}, and solves it with the same memory and time complexity, independent of data size \( n \), as that of primal \text{RIDGE-QR}.

Proof. Kernel Ridge Regression with original data \((X, y)\) solves \((K + \lambda I)\beta = y\), where, \( K \in \mathbb{R}^{n \times n} \) is the Kernel matrix, and \( \beta \in \mathbb{R}^n \) is the vector of dual variables. For any pair of row vectors in input data, \( x_i, x_j \in \mathbb{R}^{1 \times d} \), each element of Kernel matrix \( K(i, j) = \kappa(x_i, x_j) \), where \( \kappa() \) is a Reproducing Kernel Hilbert Space (RKHS) kernel function such that \( \kappa(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \) and \( \phi() \) is transformation from input space to RKHS feature space \cite{Burges1998}.

For a linear kernel function, \( \kappa(x_i, x_j) = x_i x_j^T \), \( K = XX^T \), and the objective is to solve the equation \((XX^T + \lambda I)\beta = y\) such that the model coefficient in input space, \( w = X^T \beta \). By applying \( X = QR \) via \text{HOUSEHOLDER-QR}(X), the above dual problem reformulates to

\[
(XX^T + \lambda I)\beta = y \\
\Rightarrow (QRR^TQ^T + \lambda I)\beta = y \\
\Rightarrow (QRR^TQ^T + \lambda Q^T\beta = Q^Ty \\
\Rightarrow (RR^T + \lambda I)\tilde{\beta} = \tilde{y}
\]

where, \( \tilde{y} = Q^Ty \) and \( \tilde{\beta} = Q^T\beta \). The model coefficients in the input space,

\[
w = X^T\beta = R^TQ^T\beta = R^T\tilde{\beta}
\]

Hence, solving \((XX^T + \lambda I)\beta = y\), system of \( n \) equations in \( n \) unknowns in \text{KERNELRIDGE} with original \((X, y)\) is equivalent to solving a much smaller system of \( d \) equations in \( d \) unknowns \((n \gg d)\), accurately and faster, with \((RR^T + \lambda I)\tilde{\beta} = \tilde{y}\) in \text{KERNELRIDGE-QR} with memory-efficient \((R, Q^Ty)\) sketch. It is worth noting here that once \((R, Q^Ty)\) sketch is available, the memory and time complexity for solving dual in \text{KERNELRIDGE-QR} is independent of data size \( n \), and is same to that of solving the same problem in primal form via \text{RIDGE-QR}. Figure 4(a) demonstrates the above similarity in solving \text{RIDGE-QR}, and \text{KERNELRIDGE-QR} (with linear kernel) based on computation time. Moreover, \text{KERNELRIDGE-QR} calculates the model coefficient \( w \) using a triangular matrix in \( w = R^T\tilde{\beta} \) in \( d^2 \) flops compared to \( (2n - 1)d \) flops for \( w = XX^T\beta \) in the original \text{KERNELRIDGE} with \((X, y)\).

For any non-linear kernel function such as Radial Basis Function, it is possible to represent \( K \approx AA^T \) with some low-rank matrix, \( A \in \mathbb{R}^{n \times k} \) via any kernel approximation techniques \cite{Williams2001, Si2017}. This can be followed by constructing memory-efficient \((R, Q^Ty) := \text{HOUSEHOLDER-SKETCH}(A, y)\) from Algorithm 1. Now, solving the approximated dual problem formulation for non-linear kernels via \text{KERNELRIDGE-QR} is equivalent in space and time complexity to solving the approximated problem in primal form via \text{RIDGE-QR} on \((R, Q^Ty)\). Moreover, any of the above \text{RIDGE-QR} or \text{KERNELRIDGE-QR} is faster than solving the primal form via \text{RIDGE} with \((A, y)\).

Hence, \((R, Q^Ty)\) is also the Householder sketch for Kernel Ridge Regression, where, \( R \) is defined based on linear or non-linear kernel, for accelerating the dual problem via \text{KERNELRIDGE-QR}. \(\square\)
**Theorem 3.1** (Distributed Householder-QR [Dass et al. 2018]). Let \( X = (X_1^T \ldots X_p^T)^T \), where, \( X_i \in \mathbb{R}^{n \times d} \) be local data matrix of parallel worker, \( i = 1, \ldots, p \), where \( n \gg d \), and, \( n = \hat{n}p \). Let, \( X_i = Q_iR_i \) be constructed via local Householder-QR (see Algorithm 1) for each \( i = 1, \ldots, p \), in parallel. Then, \( X = QR \) for the complete data matrix can be constructed exactly, such that \( Q = \text{diag}(Q_1, \ldots, Q_p)Q_M \), and \( R = R_M \), where \( R_{\text{stack}} = Q_M R_M \) via another Householder-QR on \( R_{\text{stack}} = (R_1^T \ldots R_p^T)^T \) gathered from all workers. The above Distributed Householder-QR has a computational time complexity of \( O(\hat{n}pd^2) \), with a communicated data volume of \( \left(\frac{d(d+1)}{2}\right) \) elements by each worker.

**Proof.**

\[
X = \begin{pmatrix}
X_1 \\
X_2 \\
\vdots \\
X_p
\end{pmatrix} = \begin{pmatrix}
Q_1R_1 \\
Q_2R_2 \\
\vdots \\
Q_pR_p
\end{pmatrix} = \text{diag}(Q_1, \ldots, Q_p) \begin{pmatrix}
R_1 \\
R_2 \\
\vdots \\
R_p
\end{pmatrix}
\]

Let us define, \( R_{\text{stack}} = \begin{pmatrix}
R_1 \\
R_2 \\
\vdots \\
R_p
\end{pmatrix} = Q_M R_M \), via Householder-QR in Algorithm 1 (or Theorem 2.1). Then,

\[
X = \text{diag}(Q_1, \ldots, Q_p)R_{\text{stack}} = \text{diag}(Q_1, \ldots, Q_p)Q_M R_M
\]

Also, it is given that \( X = QR \) via Householder-QR on complete matrix \( X \). Hence, \( Q = \text{diag}(Q_1, \ldots, Q_p)Q_M \), is the orthogonal matrix, and, \( R = R_M \) is the upper triangular matrix.

**Time complexity and Communication volume.** For a given local data \( X_i \in \mathbb{R}^{\hat{n} \times d} \), where, \( n = \hat{n}p \), each \( X_i = Q_iR_i \) at \( i \)-th parallel worker is computed via local Householder-QR (as per Algorithm 1, and Theorem 2.1). From Theorem 2.2, each local Householder-QR takes \( O(\hat{n}d^2 - d^3/3) \), in parallel for all the workers. Subsequently, \( R_{\text{stack}} = Q_M R_M \) is performed via master Householder-QR in time \( O(\times pd \times d^2 - d^3/3) \), where, \( R_{\text{stack}} \in \mathbb{R}^{pd \times d} \) is obtained by gathering (communicating) local upper-triangular matrices \( R_i \in \mathbb{R}^{d \times d} \), i.e., \( \left(\frac{d(d+1)}{2}\right) \) elements from each parallel worker \( i = 1, \ldots, p \) to the master (\( i = 1 \)). Hence, total computation time for Distributed Householder-QR is \( O(\hat{n}d^2 + pd^3 - 2d^3/3) \) or \( O(\frac{\hat{n}}{p}d^2) \) for \( \hat{n} \gg d \) (i.e. \( n \gg pd \)). It is worth noticing that the above computational time is dominant by local Householder-QR as observed in Figure 2(a).
Corollary 3.1.1 (Distributed Multiply-Qc). Let \( c = (c_1^T | \ldots | c_p^T)^T \in \mathbb{R}^n \), where, \( c_i \in \mathbb{R}^\hat{n} \) be some local vector at parallel worker with local data matrix \( X_i, \ i = 1, \ldots, p \), where \( \hat{n} \gg d \), and, \( n = \hat{n}p \). Let, orthogonal matrices \( Q_M \), and \( Q_i, \ i = 1, \ldots, p \) be constructed via Distributed Householder-QR as per Theorem 3.1 such that \( Q = \text{diag}(Q_1, \ldots, Q_p)Q_M \). Then, the reflected vector, \( Q^T c \) (or \( Qc \)) can be constructed exactly by making \((p+1)\) calls to Multiply-Qc (see Step 2 in Algorithm 1) such that \( Q^T c = Q_M^T ((Q_1^T c_1)^T | \ldots | (Q_p^T c_p)^T)^T \) or, \( Qc = \text{diag}(Q_1, \ldots, Q_p)Q_M(c_1^T | \ldots | c_p^T)^T \). The above Distributed Multiply-Qc has a computational time complexity of \( O(\frac{n}{p}d + pd^2) \), with a communicated data volume of \( (d) \) elements by each worker.

Proof. From Theorem 3.1 for \( X = (X_1^T | \ldots | X_p^T)^T \), its corresponding orthogonal matrix \( Q = \text{diag}(Q_1, \ldots, Q_p)Q_M \). Hence, \( Q^T = Q_M^T \text{diag}(Q_1^T, \ldots, Q_p^T) \)

For a given vector \( c \in \mathbb{R}^n \), \( Q^T c \) via MULTIPLY-Qc (Step 2 in Algorithm 1) can be equivalently computed from \( c = (c_1^T | \ldots | c_p^T)^T \) comprising local vector \( c_i \in \mathbb{R}^\hat{n} \), where, \( i = 1, \ldots, p \), as follows:

\[
Q^T c = Q_M^T \text{diag}(Q_1^T, \ldots, Q_p^T)c = Q_M^T \begin{pmatrix}
Q_1^T c_1 \\
Q_2^T c_2 \\
\vdots \\
Q_p^T c_p
\end{pmatrix}
\]

In Distributed MULTIPLY-Qc algorithm, the above is implemented as follows. Each worker, \( i = 1, \ldots, p \), computes its local reflected vectors \( Q_i^T c_i \in \mathbb{R}^d \) via MULTIPLY-Qc (refer Step 2 in Algorithm 1) in parallel with time \( O(2\hat{n}d) \) as shown in Theorem 2.2. Once these local reflected vectors, each of size \( d \) elements are gathered (communicated) from each worker to the master, a stacked vector \((Q_1^T c_1)^T | \ldots | (Q_p^T c_p)^T)^T \in \mathbb{R}^{pd \times d}\) is constructed. Then, a master MULTIPLY-Qc is applied on this stacked vector using \( Q_M^T \) in time \( O(2pd^2) \), i.e., \( O(2pd^2) \). Hence, total computation time of Distributed MULTIPLY-Qc is \( O(2\hat{n}d + 2pd^2) \), i.e., \( O(\frac{n}{p}d + pd^2) \), since \( n = \hat{n}p \).
B. Figures

For more clarity, we provide enlarged figures from Section 4 (Experiments and Results) of the main paper. Following is the organization of figures in the supplementary document.

**Figure 1:** (a)(b) Page 6 , (c)(j) Page 7 , (d)(e)(f) Page 8 , (g)(h)(i) Page 9, (k)(l) Page 10

**Figure 2:** (a)(b)(c) Page 11

**Figure 3:** (a)(b)(c) Page 12 , (d)(e)(f) Page 13

**Figure 4:** (a)(b)(c) Page 14
Figure 1: Sequential training time (a) Ridge (b) LASSO
Figure 1: Sequential training time on data $n \times d$

**Elastic-net**

- ELASTICCV ($d=3$)
- ELASTIC-BOOST ($d=3$)
- ELASTIC-QR ($d=3$)
- ELASTICCV ($d=5$)
- ELASTIC-BOOST ($d=5$)
- ELASTIC-QR ($d=5$)
- ELASTICCV ($d=7$)
- ELASTIC-BOOST ($d=7$)
- ELASTIC-QR ($d=7$)

**Linear Regression**

- LINREG ($d=3$)
- LINREG-BOOST ($d=3$)
- LINREG-QR ($d=3$)
- LINREG ($d=5$)
- LINREG-BOOST ($d=5$)
- LINREG-QR ($d=5$)
- LINREG ($d=7$)
- LINREG-BOOST ($d=7$)
- LINREG-QR ($d=7$)
Figure 1: Sequential training time on $n = 24M$, and various feature dimension $d = \{3, 5, 7, 10, 25, 50\}$ (d) Ridge, (e) LASSO, (f) Elastic-net
Figure 1: Sequential training time for various hyper-parameter set size $|\mathcal{A}|$ (g) Ridge, (h) LASSO, (i) Elastic-net
Figure 1: Sequential training time for various hyper-parameter set size $|\mathbf{A}|$ 
Figure 1 (k) 3D Road Network dataset (l) Household Power Consumption dataset
Figure 2: Training time analysis for Distributed RIDGE-QR with zoomed insets depicting communication time (a): Stage 1: Distributed Householder-QR timings, (b): Stage 2: Distributed Multiply-Qc and RIDGE solver timings, (c): Combined timing percentage spent on each stage for computation and communication.
Figure 3: (Scalability) Parallel speedup for DISTRIBUTED RIDGR-QR on synthetic datasets of size (a) $500K \times 100$ (b) $1M \times 100$ (c) $2M \times 100$
Figure 3: (Scalability) Parallel speedup for **DISTRIBUTED RIDGR-QR** on synthetic datasets for various feature dimension size $d = \{5, 10, 25, 50, 100\}$ (d) $500K \times d$ (e) $1M \times d$ (f) $2M \times d$.
Figure 4: (a): Comparing distributed implementations of RIDGE-QR, KERNEL-RIDGE-QR (linear kernel), and RIDGE-ADMM for $10M \times 10$ synthetic data based on (a) Computation time (b) Accuracy ($\times 10^{-6}$), $w^*$ comparison of RIDGE-QR and RIDGE-Boost, $w^*$ is solution from scikit-learn Ridge. (c) Accuracy ($\times 10^{-11}$) comparison of LINREG-QR and LINREG-Boost on Household Power Consumption dataset ($\approx 2M \times 8$), $w^*$ is solution from scikit-learn LINEAR_REGRESSION.
C. Algorithms

Algorithm 1: HOUSEHOLDER-SKETCH(X, y); see Theorem 2.2

Input: A matrix X ∈ ℜ^{n×d}, a vector y ∈ ℜ^n
Output: A matrix R ∈ ℜ^{d×d} is upper triangular such that X^TX = R^TR, and a vector y ∈ ℜ^d is top d elements of the reflected vector QT y

1. (V, R) := HOUSEHOLDER-QR(X) // see Theorem 2.1, Algorithm 4
2. y := MULTIPLY-Qc(V, y, 'T') // implicit QT y, see (Golub & Van Loan, 2012), see Algorithm 5
3. R := R[0 : d, :] // d×d triangular block
4. y := y[0 : d] // top d elements
5. return (R, y)

Algorithm 2: LMS-QR(X, y, params)

Input: A matrix X ∈ ℜ^{n×d}, a vector y ∈ ℜ^n, and a list of LMS parameters, params
Output: A vector of model coefficients, w ∈ ℜ^d

1. (R, y) := HOUSEHOLDER-SKETCH(X, y) // see Algorithm 1
2. w := LMS(R, y, params) // LINREG, RIDEcsv, LASSOCV, ELASTICCV in scikit-learn
3. return w

Algorithm 3: DISTRIBUTED LMS-QR(p, X, y, params)

Input: A scalar p > 0 parallel workers (cores or users), a matrix X = (X_1^T | ... | X_p^T) ∈ ℜ^{n×d}, a vector y = (y_1^T | ... | y_p^T), ∈ ℜ^p, a list of LMS parameters, params
Output: A vector of model coefficients, w ∈ ℜ^d

// (V, R) := DISTRIBUTED HOUSEHOLDER-QR(X), see Theorem 3.1
1. for every worker i ∈ {1, 2, ..., p} do
2. (V_i, R_i) := HOUSEHOLDER-QR(X_i) // see Theorem 2.1
3. R_i := R_i[0 : d, :] // d×d triangular block
4. R_stack := GATHER(R_i, root = 0) // R_stack = vstack(R_1, ..., R_p) at Master
5. end
6. if i == 1 then // check for Master
7. (V, R) := HOUSEHOLDER-QR(R_stack) // see Theorem 2.1
8. R := R[0 : d, :] // d×d triangular block
9. end
10. // V := [V_1, ..., V_p, V_M] is never centralized or shared
11. // Q := diag(Q_1, ..., Q_p)Q_M, and, R = R_M, see Theorem 3.1
12. // y := DISTRIBUTED MULTIPLY-Qc(V, y, 'T'), see Corollary 3.1
13. for every worker i ∈ {1, 2, ..., p} do
14. y_i := MULTIPLY-Qc(V_i, y_i, 'T') // implicit QT y_i, see Algorithm 5
15. y_stack := GATHER(y_i, root = 0) // y_stack = vstack(y_1, ..., y_p) at Master
16. if i == 1 then // check for Master
17. y_M := MULTIPLY-Qc(V_M, y_stack, 'T') // implicit QT_M y_stack, see Algorithm 5
18. y_M := y_M[0 : d] // select top d elements
19. end
20. y := y_M // y = QT y = QT_M((Q_M^T y_1)^T | ... | (Q_M^T y_p)^T)^T

// Solving LMS
21. w := LMS(R, y, params) // run LMS solver at Master
22. BROADCAST(w, root = 0) // every worker receives the global model
23. end
24. return w
Algorithm 4: \((V, R) \leftarrow X\), via \textsc{Householder-QR}, refer Theorem 2.1

\begin{algorithm}
\begin{algorithmic}[1]
  \STATE \textbf{Input:} A matrix \(X \in \mathbb{R}^{n \times d}\)
  \STATE \textbf{Output:} Householder reflector set \(V\), Upper trapezoidal matrix \(R \in \mathbb{R}^{n \times d}\)
  \FOR{j \leftarrow 1 \text{ to } d}
    \STATE \(v_j \leftarrow X(j : n, j)\)
    \STATE \(v_j(1) \leftarrow v_j(1) + \text{sign}(v_j(1)) \times \|v_j\|_2 \) // scalar update
    \STATE \(v_j \leftarrow \frac{v_j}{\|v_j\|_2} \) // vector normalization
    \STATE \(X(j : n, j : d) \leftarrow X(j : n, j : d) - 2 \times v_j < v_j, X(j : n, j : d) >\)
    \STATE \(R = X(j : n, j : d)\)
  \ENDFOR
  \STATE \(V \leftarrow [v_1, v_2, \ldots, v_d] \) // set of d-reflectors
  \RETURN \((V, R)\)
\end{algorithmic}
\end{algorithm}

Algorithm 5: Computing implicit \(Q^T y\) via \textsc{Multiply-Qc}

\begin{algorithm}
\begin{algorithmic}[1]
  \STATE \textbf{Input:} Householder reflector set \(V\), a vector \(y \in \mathbb{R}^n\)
  \STATE \textbf{Output:} \(\bar{y} \leftarrow (Q^T y) \in \mathbb{R}^n\)
  \STATE \(c \leftarrow y\)
  \FOR{j \leftarrow 1 \text{ to } d}
    \STATE \(c(j : n) \leftarrow c(j : n) - 2 \times v_j(v_j^T c(j : n))\)
  \ENDFOR
  \STATE \(\bar{y} \leftarrow c\)
  \RETURN \(\bar{y}\)
\end{algorithmic}
\end{algorithm}

References


