Low-Precision QR Factorization: Analysis, Algorithms, and Applications

L. Minah Yang, Alyson Fox, and Geoffrey Sanders

August 26, 2019

1 Introduction

Development of new graphics processing units (GPUs) that perform half precision arithmetic up to 16 times faster than double precision arithmetic motivates use of low precision computations whenever possible. GPUs were initially designed to support computer graphics and evolved to continue supporting that sole utility prior to the emergence of general purpose GPUs (GPGPUs). In recent years, GPGPUs were largely fueled by growing interest in deep learning, which requires dealing with large amounts of data that can be processed in parallel and can benefit from efficiently harnessing the highly parallelizable and wide memory bandwidth structure of GPUs. Some drawbacks of GPUs are that they are limited to simple tasks, and have a narrower range of capabilities in comparison to central processing units (CPUs). However, there are plenty of applications and algorithms in scientific computing that can potentially take advantage of GPUs.

Training deep artificial neural networks with various tasks such as multiplication [1] and storage [2], assigned to lower precision arithmetic can easily be implemented on GPUs, and these works feature methods that are already in common practice in data science applications.

The accuracy of a numerical algorithm depends on several factors including numerical stability and well-conditionedness of the problem, both of which may be sensitive to rounding errors. Low precision floats use fewer bits than high precision floats to represent the real numbers, and naturally incur larger rounding errors [illegible]. Therefore, error attributed to round-off may have a larger influence over the total error when using low precision, and some standard algorithms that are in wide use may no longer be numerically stable when using half precision floating arithmetic and storage.

The low precision computing environments that we consider are designed to imitate those of new GPUs, which employ multiple precision types. For example, Tesla V100's Tensor Cores perform matrix-multiply-and-accumulate of half precision input data with full-precision products and single-precision summation accumulate \cite. The existing rounding error analyses only allow a uniform floating point precision for storage and operations, and cannot accurately represent mixed-precision settings such as TensorCores. Therefore, using existing error analyses by rounding up to the higher precision within the mixed-precision setting would be too optimistic, and rounding down to the lowest precision within the mixed-precision setting would be too pessimistic. One form of battling the pessimistic nature of deterministic error bounds is probabilistic error bounds, but these also suffer from being restricted to uniform precision procedures. In this work, we develop a framework for mixed-precision rounding error analysis, and explore half-precision QR factorization for data and graph analysis applications.

Our findings are that the new mixed-precision error analysis produces tighter error bounds, and block QR algorithms are able to operate in low precision more robustly than non-block techniques.

Given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \geq n$, we consider performing the *QR factorization*, where

$$\mathbf{A} = \mathbf{QR}, \qquad \mathbf{Q} \in \mathbb{R}^{m \times m}, \qquad \mathbf{R} \in \mathbb{R}^{m \times n},$$

$\mathbf{Q}$ is orthogonal, $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}_{m \times m}$, and $\mathbf{R}$ is upper-trapezoidal, $\mathbf{R}_{ij} = 0$ for $i > j$.

The above formulation is a *full* QR factorization whereas a more efficient *thin* QR factorization results in $\mathbf{Q} \in \mathbb{R}^{m \times n}$ and $\mathbf{R} \in \mathbb{R}^{n \times n}$.

$$\mathbf{A} = \mathbf{QR} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0}_{m-n \times n} \end{bmatrix} = \mathbf{Q}_1\mathbf{R}_1$$

Here, $\mathbf{Q}_1\mathbf{R}_1$ is the *thin* QR factorization, where the columns of $\mathbf{Q}_1$ are orthonormal, and $\mathbf{R}_1$ is upper-triangular. In many applications, computing the *thin* decomposition requires less computation and is sufficient in performance.

In section 2, we will give an overview of the modern developments in hardware that motivates rounding error analysis that supports multiple precision types and present a set of error analysis tools. The HQR factorization algorithm and a mixed-precision rounding error analysis of its implementation is in section 3. In section 4, we present the TSQR algorithm as well as numerical experiments that show that TSQR can be useful in low precision environments. Section 5 explores the use of low and mixed precision QR algorithms as subroutines for two applications, spectral clustering and sparse regression in the context of discovery of equations.

1.1 Notation

While important definitions are stated explicitly in the text, Table 1 serves to establish basic notation.

2 Floating Point Numbers and Error Analysis Tools

2.1 Modern GPU Hardware

2.2 Representation of Real Numbers

Consider floating point number systems that are defined by

$$\text{significand} \times \text{base}^{\text{exponent}}. \tag{1}$$

This is the generic form of floating point representations, including the IEEE 754 Standard which was established in 1985 and has been been accepted and followed by most modern machines since. Let $\mathbb{F} \subset \mathbb{R}$ denote the space of some floating point number system with base $b \in \mathbb{N}$, precision $t \in \mathbb{N}$, significand $\mu \in \mathbb{N}$, and exponent range $[\eta_{\min}, \eta_{\max}] \subset \mathbb{Z}$. Then every element y in $\mathbb{F}$ can be written as

$$y = \pm\mu \times b^{\eta - t} \tag{2}$$

Symbol(s)	Definition(s)	Section(s)
$\mathbf{Q}$	Orthogonal factor of matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$: m-by-m (full) or m-by-n (thin)	1
$\mathbf{R}$	Upper triangular or trapezoidal factor of matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$: m-by-n (full) or n-by-n (thin)	1
$\mathbf{A}^{(k)}$	Matrix $\mathbf{A}$ after k Householder transformations.	3.1
$\mathrm{fl}(\mathbf{x})$, $\mathbf{x}$	Quantity $\mathbf{x}$ calculated from floating point operations	2
b, t, μ, η	Base/precision/mantissa/exponent bits	2
Inf	Values outside the range of representable numbers	3.1.2
k	Number of FLOPs	2
u_q	Unit round-off for precision t and base b: $\frac{1}{2}b^{1-t}$	2
δ_q	Quantity bounded by: $\lvert\delta_q\rvert < u_q$	2
$\gamma_q^{(k)}$, $\theta_q^{(k)}$	$\frac{ku_q}{1-ku_q}$. Quantity bounded by: $\lvert\theta_q^{(k)}\rvert \le \gamma_q^{(k)}$	2
$\mathbf{x}$, $\mathbf{A}$	Vector, matrix	2
m, n	Number of rows, columns of matrix , or length of vector	1
i, j	Row, column index of matrix or vector	3.1
$\lVert\mathbf{x}\rVert_2$, $\lVert\mathbf{A}\rVert_2$	Vector,operator 2-norm	3
$\lvert c\rvert$, $\lvert\mathbf{x}\rvert$,$\lvert\mathbf{A}\rvert$	Absolute value of constant, all elements of vector,matrix	3
$\mathbf{x}_i$	i^{th} element of vector $\mathbf{x}$	3.1
$\mathbf{A}[a:b,:]$, $\mathbf{A}[:,c:d]$	Rows a to b, columns c to d of matrix $\mathbf{A}$	3.1
$\hat{c}_i$	Cardinal vector	3
$\mathbf{0}_{m\times n}$, $\mathbf{I}_n$	m-by-n zero matrix, n-by-n identity matrix	1
$\mathbf{I}_{m\times n}$	$\begin{bmatrix}\mathbf{I}_n & \mathbf{0}_{n\times(m-n)}\end{bmatrix}^\top$	4
$\mathbf{P}_\mathbf{v}$, $\mathbf{P}_i$	Householder transformation define by $\mathbf{v}$. i^{th} Householder transformation in HQR	3.1
u_+, u_p, u_w	Unit round-off for sum, product, and storage (write)	2.3

Table 1: Basic definitions

where p is any integer in $[0, b^t - 1]$, and q is an integer in $[\eta_{\min}, \eta_{\max}]$. While base, precision, and exponent range are fixed and define a floating point number system, the sign, significand, and exponent uniquely identifies a single number within that system. As the significand is always an integer in $[0, b^{t-1}]$, it can be combined with the precision and represent a fraction between 0 and 1, $\frac{p}{b^t}$. Table 2 shows IEEE 754 floating point number types described by the same parameters as in Equation 2, and the remainder of this paper will discuss floating point arithmetic as is defined by the IEEE 754 Standard.

Name	b	t	# of exponent bits	$\eta_{\min}$	$\eta_{\max}$	u
IEEE754 half	2	11	5	-15	16	4.883e-04
IEEE754 single	2	24	8	-127	128	5.960e-08
IEEE754 double	2	53	11	-1023	1024	1.110e-16

Table 2: IEEE754 formats and their primary attributes.

Although operations we use on $\mathbb{R}$ cannot be replicated exactly due to the finite cardinality of $\mathbb{F}$, we can still approximate the accuracy of analogous floating point operations. We adopt

the rounding error analysis tools developed in [3], which allow a relatively simple framework for formulating error bounds for complex linear algebra operations. A short analysis of floating point operations (cf. Theorem 2.2 [3]) shows that the relative error is controlled by the unit round-off, $u = \frac{1}{2}b^{1-t}$.

Let op be any basic operation between 2 floating point numbers from the set $\mathrm{OP} = \{+, -, \times, \div\}$. The true value $(x \mathrm{\ op\ } y)$ lies in $\mathbb{R}$ and it is rounded using some conversion to a floating point number, $\mathrm{fl}(x \mathrm{\ op\ } y)$, admitting a rounding error. The IEEE 754 Standard requires *correct rounding*, which rounds the exact solution $(x \mathrm{\ op\ } y)$ to the closest floating point number, and in case of a tie, to the floating point number that has a mantissa ending in an even number. *Correct rounding* gives us an assumption for the error model where a single basic floating point operation yields a relative error, δ, bounded in the following sense,

$$\mathrm{fl}(x \mathrm{\ op\ } y) = (1+\delta)(x \mathrm{\ op\ } y), \quad |\delta| \le u, \quad \mathrm{op} \in \{+, -, \times, \div\}. \tag{3}$$

We use Equation 3 as a building block in accumulating errors from successive floating point operations (FLOPs). For example, consider computing $x+y+z$ where $x, y, z \in \mathbb{R}$. Let's assume that the machine can only compute one operation at a time, and we take the convention of computing the left-most operation first. Then there is a rounding error in computing $s_1 := \mathrm{fl}(x+y) = (1+\delta)(x+y)$, and another rounding error in computing $s_2 := \mathrm{fl}(s_1 + z) = (1+\hat{\delta})(s_1 + z)$, where $|\delta|, |\hat{\delta}| < u$. Let's expand the final result:

$$\mathrm{fl}(x+y+z) = \mathrm{fl}(\mathrm{fl}(x+y)+z) = \mathrm{fl}((1+\delta)(x+y)+z) \tag{4}$$
$$= (1+\hat{\delta})\left((1+\delta)(x+y)+z\right) \tag{5}$$
$$= (1+\hat{\delta})(1+\delta)(x+y) + (1+\hat{\delta})z \tag{6}$$

We can see that keeping track of rounding errors from each operation can quickly grow to be challenging, even with just two successive operations. A way of simplifying complicated expressions like Equation 6 is crucial in developing error analyses for complex linear algebra operations. Lemma 2.1 introduces one such convenient and elegant bound that simplifies accumulated rounding errors.

Lemma 2.1 (Lemma 3.1 [3]). *Let $|\delta_i| < u$ and $\rho_i \in \{-1, +1\}$ for $i = 1, \cdots, k$, and $ku < 1$. Then,*

$$\prod_{i=1}^{k}(1+\delta_i)^{\rho_i} = 1 + \theta^{(k)} \tag{7}$$

where

$$|\theta^{(k)}| \le \frac{ku}{1-ku} =: \gamma^{(k)}. \tag{8}$$

In other words, $\theta^{(k)}$ represents the accumulation of rounding errors from k successive operations, and it is bounded by $\gamma^{(k)}$. Allowing $\theta^{(k)}$'s to be any arbitrary value within the corresponding $\gamma^{(k)}$ bounds further aids in keeping a clear, simple error analysis. Applying this lemma to our example of adding three numbers results in,

$$\mathrm{fl}(x+y+z) = (1+\hat{\delta})(1+\delta)(x+y) + (1+\hat{\delta})z = (1+\theta^{(2)})(x+y) + (1+\theta^{(1)})z. \tag{9}$$

Since $|\theta^{(1)}| \le \gamma^{(1)} < \gamma^{(2)}$, we can further simplify Equation 9 to

$$\mathrm{fl}(x+y+z) = (1+\hat{\theta}^{(2)})(x+y+z), \quad \textit{where} \quad \hat{\theta}^{(2)} \le \gamma^{(2)}. \tag{10}$$

Typically, error bounds formed in the fashion of Equation 10 converted to relative errors in order to put the error magnitudes in perspective. In our example, for nonzero $(x+y+z)$, we have:

$$\frac{|(x+y+z)-\mathrm{fl}(x+y+z)|}{|x+y+z|} \leq \gamma^{(2)}. \tag{11}$$

Although Lemma 2.1 only requires $ku < 1$, we actually need $ku < \frac{1}{2}$, which implies $\gamma^{(k)} < 1$, in order to maintain a meaningful relative error bound. While this assumption, $\gamma^{(k)} < 1$, is easily satisfied by fairly large k in higher precision floating point numbers, it is a problem even for small k in lower precision floating point numbers. Table 3 shows the maximum value of k that still guarantees a relative error below 100% ($\gamma^{(k)} < 1$). Thus, accumulated rounding errors in lower precision types

precision	u	$k = \mathrm{argmax}^{(k)}(\gamma^{(k)} \leq 1)$
half	4.883e-04	512
single	5.960e-08	$\approx 4.194e06$
double	1.110e-16	$\approx 2.252e15$

Table 3: Upper limits of meaningful relative error bounds in the $\gamma^{(k)}$ notation.

grow unstable very quickly and with fewer operations in comparison to higher precision types. As k represents the number of FLOPs, this restricts low-precision floating point operations to smaller problem sizes and lower complexity algorithms.

To clearly illustrate how this restricts rounding error analysis in half precision, we now consider performing the dot product of two vectors. A forward error bound for dot products is:

$$\frac{|\mathbf{x}^\mathsf{T}\mathbf{y} - \mathrm{fl}(\mathbf{x}^\mathsf{T}\mathbf{y})|}{|\mathbf{x}|^\mathsf{T}|\mathbf{y}|} \leq \gamma^{(m)}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^m. \tag{12}$$

where details and proof for this statement can be found in Section 3.4 of [3]. While this result does not guarantee a high relative accuracy when $|\mathbf{x}^\mathsf{T}\mathbf{y}| \ll |\mathbf{x}|^\mathsf{T}|\mathbf{y}|$, high relative accuracy is expected in some special cases. For example, let $\mathbf{x} = \mathbf{y}$. Then we have exactly $|\mathbf{x}^\mathsf{T}\mathbf{x}| = |\mathbf{x}|^\mathsf{T}|\mathbf{x}| = \|\mathbf{x}\|_2^2$. This leads to

$$\left|\frac{\|\mathbf{x}\|_2^2 - \mathrm{fl}(\|\mathbf{x}\|_2^2)}{\|\mathbf{x}\|_2^2}\right| \leq \gamma_p^{(d+2)}. \tag{13}$$

Since vectors of length m accumulate rounding errors that are bounded by $\gamma^{(m)}$, the worst-case relative error bound for a dot product of vectors of length 512 is already at 100% ($\gamma_{\text{half}}^{(512)} = 1$).

We present a simple numerical experiment that shows that the standard deterministic error bound is too pessimistic, and cannot be practically used to approximate rounding error for half-precision arithmetic. In this experiment, we generated 2 million random half-precision vectors of length 512 from two random distributions: the standard normal distribution, $N(0,1)$, and the uniform distribution over $(0,1)$. Half precision arithmetic was simulated by calling Algorithm 1 for every multiplication and summation step required in calculating the dot product, $\mathrm{fl}(\mathbf{x}^\mathsf{T}\mathbf{y})$.

Algorithm 1: $\mathbf{z}_{\text{half}} = \texttt{simHalf}(f, \mathbf{x}_{\text{half}}, \mathbf{y}_{\text{half}})$ Simulate function $f \in$ OP\{`dot_product`\} in half precision arithmetic given input variables `x`, `y`. Function `castup` converts half precision floats to single precision floats, and `castdown` converts single precision floats to half precision floats by rounding to the nearest half precision float.

Input: $\mathbf{x}_{\text{half}}, \mathbf{y}_{\text{half}} \in \mathbb{F}_{\text{half}}^m$, $f : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^n$
Output: $\mathrm{fl}(f(\mathbf{x}_{\text{half}}, \mathbf{y}_{\text{half}})) \in \mathbb{F}_{\text{half}}^n$

```
x_single, y_single ← castup([x_half, y_half])
z_single ← fl(f(x_single, y_single))
z_half ← castdown(z_single)
return z_half
```

The casting up step is exact since all half precision numbers can be exactly represented in single precision, $\mathbb{F}_{\text{half}} \subset \mathbb{F}_{\text{single}}$, the second step incurs a rounding error from a single precision arithmetic operation, and the casting down step incurs a rounding error from casting down to half precision. Note that using Algorithm 1 for any operation in OP results in simulating half precision arithmetic, whereas using it with the dot product results in simulating mixed precision arithmetic instead. The relative error in this experiment is formulated as the left hand side of the inequality in Equation 12, where all operations outside of calculating $\mathrm{fl}(\mathbf{x}^\mathsf{T}\mathbf{y})$ is executed by casting up to double precision format and using double precision arithmetic. Table 4 shows statistics from computing the relative error for simulated half precision dot products of 512-length random vectors.

Random Distribution	Average	Standard deviation	Maximum
Standard normal	1.627e-04	1.640e-04	2.838e-03
Uniform $(0,1)$	2.599e-03	1.854e-03	1.399e-02

Table 4: Statistics from dot product backward relative error in for 512-length vectors stored in half-precision and computed in simulated half-precision.

We see that the inner products of vectors sampled from the standard normal distribution have backward relative errors that don't deviate much from the unit round-off (4.883e-4), whereas the vectors sampled from the uniform distribution tend to accumulate larger errors. Even so, the theoretical upper error bound of 100% is too pessimistic, and it is difficult to predict the kind of results this experiment shows. Recent works in developing probabilistic bounds on rounding errors of floating point operations have shown that the inner product relative backward error for the conditions used for this experiment is bounded by 5.466e-2 with probability 0.99.

Most importantly, no rounding error bounds (deterministic and probabilistic) allow flexibility in the precision types used for different operations. This restriction is the biggest obstacle in gaining an understanding of rounding errors to expect from computations done on emerging hardware that support mixed-precision such as GPUs that employ mixed-precision arithmetic. In this paper, we develop a mixed-precision error analysis that allows multiple precision types and is based on the rounding error analysis framework established in [3].

Lemma 2.2 shows rules from Lemma 3.3 in [3] that summarize how to accumulate errors represented by θ's and γ's.

Lemma 2.2. *For any positive integer k, let $\theta^{(k)}$ denote a quantity bounded according to $|\theta^{(k)}| \leq \frac{ku}{1-ku} = \gamma^{(k)}$. The following relations hold for positive integers i, j, and nonnegative integer k.*

Arithmetic operations between $\theta^{(k)}$'s.

$$(1+\theta^{(k)})(1+\theta^{(j)}) = (1+\theta^{(k+j)})$$

$$\frac{1+\theta^{(k)}}{1+\theta^{(j)}} = \begin{cases} 1+\theta^{(k+j)}, & j \le k \\ 1+\theta^{(k+2j)}, & j > k \end{cases}$$

Operations on γ's:

$$\gamma^{(k)}\gamma^{(j)} \le \gamma_{\max(k,j)}, \quad \text{for } \max_{(j,k)} u \le \frac{1}{2}$$

$$n\gamma^{(k)} \le \gamma^{(nk)}, \quad \text{for} \quad n \le \frac{1}{uk}$$

$$\gamma^{(k)} + u \le \gamma^{(k+1)}$$

$$\gamma^{(k)} + \gamma^{(j)} + \gamma^{(k)}\gamma^{(j)} \le \gamma^{(k+j)}$$

In Lemma 2.3, we present modified versions of the rules in Lemma 2.2. This mixed-precision error analysis relies on the framework given by Lemma 2.1, which best allows us to keep a simple analysis. These relations allow us to easily accumulate errors in terms of θ's and γ's, and aid in writing clear and simpler error analyses. The modifications support multiple precision types, whereas Lemma 2.2 assumes that the same precision is used in all operations.

We distinguish between the different precision types using subscripts — these types include products (p), sums (s), and storage formats (w).

Lemma 2.3. *For any nonnegative integer k and some precision q, let $\theta_q^{(k)}$ denote a quantity bounded according to $|\theta_q^{(k)}| \le \frac{ku_q}{1-ku_q} =: \gamma_q^{(k)}$. The following relations hold for two precisions s and p, positive integers, j_s, j_p, non-negative integers k_s and k_p, and $c > 0$.*

$$(1+\theta_p^{(k_p)})(1+\theta_p^{(j_p)})(1+\theta_s^{(k_s)})(1+\theta_s^{(j_s)}) = (1+\theta_p^{(k_p+j_p)})(1+\theta_s^{(k_s+j_s)}) \tag{14}$$

$$\frac{(1+\theta_p^{(k_p)})(1+\theta_s^{(k_s)})}{(1+\theta_p^{(j_p)})(1+\theta_s^{(j_s)})} = \begin{cases} (1+\theta_s^{(k_s+j_s)})(1+\theta_p^{(k_p+j_p)}), & j_s \le k_s, j_p \le k_p \\ (1+\theta_s^{(k_s+2j_s)})(1+\theta_p^{(k_p+j_p)}), & j_s \le k_s, j_p > k_p \\ (1+\theta_s^{(k_s+j_s)})(1+\theta_p^{(k_p+2j_p)}), & j_s > k_s, j_p \le k_p \\ (1+\theta_s^{(k_s+2j_s)})(1+\theta_p^{(k_p+2j_p)}), & j_s > k_s, j_p > k_p \end{cases} \tag{15}$$

Without loss of generality, let $1 \gg u_p \gg u_s > 0$. Let d, a nonnegative integer, and $r \in [0, \lfloor \frac{u_p}{u_s} \rfloor]$ be numbers that satisfy $k_s u_s = du_p + ru_s$. Alternately, d can be defined by $d := \lfloor \frac{k_s u_s}{u_p} \rfloor$.

$$\gamma_s^{(k_s)}\gamma_p^{(k_p)} \le \gamma_p^{(k_p)}, \quad \text{for } k_p u_p \le \frac{1}{2} \tag{16}$$

$$\gamma_s^{(k_s)} + u_p \le \gamma_p^{(d+2)} \tag{17}$$

$$\gamma_p^{(k_p)} + u_s \le \gamma_p^{(k_p+1)} \tag{18}$$

$$\gamma_p^{(k_p)} + \gamma_s^{(k_s)} + \gamma_p^{(k_p)}\gamma_s^{(k_s)} < \gamma_p^{(k_p+d+1)} \tag{19}$$

A proof for Equation 19 is shown in Appendix A.3.1. We use these principles to establish a mixed-precision rounding error analysis for computing the dot product, which is crucial in many linear algebra routines such as the QR factorization.

2.3 Inner product Mixed-Precision error

We will see in Section 3 that the inner product is a building block of the HQR factorization (HQR) algorithm, which was introduced in [4]. More generally, it is used widely in most linear algebra tools such as matrix-vector multiply and projections. Thus, we will generalize classic round-off error analysis of inner products to algorithms that may employ different precision types to different operations. Specifically, we consider performing an inner product with the storage precision, u_w, being lower than the summation precision, u_s. This is designed to provide a more accurate rounding error analysis of mixed precision floating point operations present in recent GPU technologies such as NVIDIA's TensorCore. Currently, TensorCore computes the inner product of vectors stored in half-precision by employing full precision multiplications and a single-precision accumulator. As the majority of rounding errors from computing inner products occur during summation (c.f. Section 3.1, [3]), the single precision accumulator immensely reduces the error in comparison to using only half-precision operations. This increase in accuracy combined with its speedy performance motivates us to: 1) study how to best utilize mixed-precision arithmetic in algorithms, and 2) to develop more accurate error analyses appropriate for mixed-precision algorithms.

Lemma 2.4 and Corollary 2.5 present two mixed-precision forward error bounds for inner products, which show a tighter bound than the existing error bounds. In both cases, we assume storage in the lowest precision with round-off value, u_w, and summation performed with a higher precision with round-off value, u_s, and let $d \approx mu_s/u_w$, where m is the length of the vectors. Although there are additional differing assumptions in these two lemmas, results from both show a strong dependence on d.

Lemma 2.4. *Let w, p, and s each represent floating point precisions for storage, product, and summation, where the varying precisions are defined by their unit round-off values denoted by u_w, u_p, and u_s. Let $\mathbf{x}, \mathbf{y} \in \mathbb{F}_w^m$ be two arbitrary vectors stored in w precision. If an inner product performs multiplications in precision p, and addition of the products using precision s, then,*

$$\mathrm{fl}(\mathbf{x}^T\mathbf{y}) = (\mathbf{x}+\Delta\mathbf{x})\mathbf{y} = \mathbf{x}(\mathbf{y}+\Delta\mathbf{y}), \tag{20}$$

where $|\Delta\mathbf{x}| \le \gamma_{p,s}^{(1,m-1)}|\mathbf{x}|$, $|\Delta\mathbf{y}| \le \gamma_{p,s}^{(1,m-1)}|\mathbf{y}|$ componentwise, and

$$\gamma_{p,s}^{(1,m-1)} := (1+u_p)(1+\gamma_s^{(m-1)}) - 1$$

This result is then stored in precision w and if we further assume that $u_w = u_p > u_s$, then $|\Delta\mathbf{x}| \le \gamma_w^{(d+2)}|\mathbf{x}|$ and $|\Delta\mathbf{y}| \le \gamma_w^{(d+2)}|\mathbf{y}|$ where $d := \lfloor \frac{(m-1)u_s}{u_w} \rfloor$.

Corollary 2.5 presents another mixed-precision forward error bound for mixed-precision inner products with additional constraints. Here, we assume that the vectors are being stored in a lower precision than the precision types being used for multiplications and additions. This scenario is similar to how TensorCore technology works in GPUs.

Corollary 2.5. *In addition to the assumptions in Lemma 2.4, assume $1 \gg u_w \gg u_s > 0$, and for any two numbers x, y in $\mathbb{F}_w$, their product xy is in $\mathbb{F}_s$. Let $\mathbf{x}, \mathbf{y} \in \mathbb{F}_w^m$ be two arbitrary vectors stored in w precision. If an inner product performs multiplications in full precision, and addition of the products using precision s, then,*

$$\mathrm{fl}(\mathbf{x}^T\mathbf{y}) = (\mathbf{x}+\Delta\mathbf{x})\mathbf{y} = \mathbf{x}(\mathbf{y}+\Delta\mathbf{y}), \tag{21}$$

where $|\Delta x| \le \gamma_w^{(d+1)}|x|$, $|\Delta y| \le \gamma_w^{(d+1)}|y|$ componentwise, and $d := \lfloor \frac{(m-1)u_s}{u_w} \rfloor$.

Proofs for Lemma 2.4 and Corollary 2.5 are shown in Appendix A.3.1. The analyses for these two differ only in the type of mixed-precision arithmetic performed within the inner product subroutine, and the difference is revealed to result in either $\gamma_w^{(d,1)}$ or $\gamma_w^{(d,2)}$. For the rest of this paper, we will refer to the forward error bound for the inner product as $\gamma_w^{(d,z)}$ for $z = 1, 2$ to generalize the analysis for varying assumptions. This simplification allows us to use the same analysis for the remaining steps of the HQR algorithm presented in the following sections.

3 Mixed-Precision HQR Factorization

The HQR algorithm uses Householder transformations to zero out elements below the diagonal of a matrix. We present this as zeroing out all but the first element of some vector, $\mathbf{x} \in \mathbb{R}^m$.

Lemma 3.1. *Given vector $\mathbf{x} \in \mathbb{R}^m$, there exist Householder vector $\mathbf{v}$ and Householder transformation matrix $\mathbf{P_v}$ such that $\mathbf{P_v}$ zeroes out $\mathbf{x}$ below the first element.*

$$\begin{aligned} \sigma &= -\operatorname{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2, \quad \mathbf{v} = \mathbf{x} - \sigma\mathbf{e}_1 \\ \beta &= \frac{2}{\mathbf{v}^\mathsf{T}\mathbf{v}} = -\frac{1}{\sigma\mathbf{v}_1}, \qquad \mathbf{P_v} = \mathbf{I}_m - \beta\mathbf{v}\mathbf{v}^\mathsf{T} \end{aligned} \tag{22}$$

The transformed vector, $\mathbf{P_v x}$, has the same 2-norm as $\mathbf{x}$ since Householder transformations are orthogonal,

$$\mathbf{P_v x} = \sigma\mathbf{e}_1 \tag{23}$$

In addition, $\mathbf{P_v}$ is symmetric and orthogonal, $\mathbf{P_v} = \mathbf{P_v}^\mathsf{T} = \mathbf{P_v}^{-1}$, and therefore involutary, $\mathbf{P_v}^2 = \mathbf{I}$.

3.1 HQR Factorization Algorithm

Given $\mathbf{A} \in \mathbb{R}^{m\times n}$ and Lemma 3.1, HQR is done by repeating the following processes. For $i = 1, 2, \cdots, n$,

Step 1) Find and store the Householder vector ($\mathbf{v}$) and constant (β) that zeros out the i^{th} column of $\mathbf{A}$ beneath the i^{th} element.

Step 2) Apply the Householder transformation to the bottom right partition ($\mathbf{A}[i:m, i:n]$) of the matrix.

until only an upper triangular matrix remains.

Consider the following 4-by-3 matrix example adapted from [3]. Let $\mathbf{P}_i$ represent the i^{th} Householder transformation of this algorithm.

$$\mathbf{A} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{\text{apply } \mathbf{P}_1 \text{ to } \mathbf{A}^{(1)} = \mathbf{A}} \left[\begin{array}{c|cc} \times & \times & \times \\ \hline 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{array}\right] \xrightarrow{\text{apply } \mathbf{P}_2 \text{ to } (\mathbf{A}^{(1)} = \mathbf{P}_1\mathbf{A})}$$

$$\left[\begin{array}{cc|c} \times & \times & \times \\ 0 & \times & \times \\ \hline 0 & 0 & \times \\ 0 & 0 & \times \end{array}\right] \xrightarrow{\text{apply } \mathbf{P}_3 \text{ to } (\mathbf{A}^{(2)} = \mathbf{P}_2\mathbf{P}_1\mathbf{A})} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{bmatrix}$$

Since the final matrix $\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1\mathbf{A}$ is upper-triangular, this is the R factor of the QR decomposition. Set $\mathbf{Q}^\mathsf{T} = \mathbf{P}_3\mathbf{P}_2\mathbf{P}_1$. Then we can formulate $\mathbf{Q}$ via

$$\mathbf{Q} = (\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1)^\mathsf{T} = \mathbf{P}_1^\mathsf{T}\mathbf{P}_2^\mathsf{T}\mathbf{P}_3^\mathsf{T} = \mathbf{P}_1\mathbf{P}_2\mathbf{P}_3,$$

where the last equality results from the symmetric property of $\mathbf{P}_i$'s.

Returning to the general case, we have

$$\mathbf{Q}_{\text{full}} = \mathbf{P}_1 \cdots \mathbf{P}_n \quad \text{and} \quad \mathbf{R}_{\text{full}} = \mathbf{Q}^\mathsf{T}\mathbf{A} = \mathbf{P}_n \cdots \mathbf{P}_1\mathbf{A}, \tag{24}$$

for the orthogonal factor in a full QR factorization, and

$$\mathbf{Q}_{\text{thin}} = \mathbf{P}_1 \cdots \mathbf{P}_n\mathbf{I}_{m\times n} \quad \text{and} \quad \mathbf{R}_{\text{thin}} = \mathbf{I}_{m\times n}^\mathsf{T}\mathbf{Q}^\mathsf{T}\mathbf{A} = \mathbf{I}_{m\times n}^\mathsf{T}\mathbf{P}_n \cdots \mathbf{P}_1\mathbf{A}. \tag{25}$$

3.1.1 HQR Factorization Implementation

The Householder transformation is implemented by a series of inner and outer products, since Householder matrices are rank-1 updates of the identity. This is much less costly than forming $\mathbf{P_v}$, then performing matrix-vector or matrix-matrix multiplications. For some $\mathbf{P_v} = \mathbf{I} - \beta\mathbf{v}\mathbf{v}^\mathsf{T}$, we result in the following computation.

$$\mathbf{P_v x} = (\mathbf{I} - \beta\mathbf{v}\mathbf{v}^\mathsf{T})\mathbf{x} = \mathbf{x} - (\beta\mathbf{v}^\mathsf{T}\mathbf{x})\mathbf{v} \tag{26}$$

The routine shown in Equation 26 is used in forming R and Q. Given a vector $\mathbf{x} \in \mathbb{R}^m$, Algorithm 2 calculates the Householder constant β and Householder vector $\mathbf{v}$, that zero out $\mathbf{x}$ below the first element, and also returns σ. Algorithm 3 is the HQR algorithm where information necessary to build $\mathbf{Q}$ is returned instead of explicitly forming $\mathbf{Q}$ — the Householder vector and constant at the k^{th} step is stored as the k^{th} column of matrix $\mathbf{V} \in \mathbb{R}^{m\times n}$ and the k^{th} element of vector $\boldsymbol{\beta} \in \mathbb{R}^n$.

Finally, the $\mathbf{Q}$ factor can be built using Algorithm 4. While this algorithm shows how to left multiply $\mathbf{Q}$ to any input matrix $\mathbf{B}$ given $\mathbf{V}$ and $\boldsymbol{\beta}$, putting in $\mathbf{B} \equiv I_{m\times n}$ will yield $\mathbf{Q}_{\text{thin}}$.

Algorithm 2: β, $\mathbf{v}$, σ = **hh_vec**($\mathbf{x}$). Given a vector $\mathbf{x} \in \mathbb{R}^m$, return the Householder vector $\mathbf{v}$, a Householder constant β, and σ such that $(I - \beta\mathbf{v}\mathbf{v}^\mathsf{T})\mathbf{x} = \sigma(\mathbf{e}_1)$, and $\mathbf{v}_1 = 1$.

Input: $\mathbf{x} \in \mathbb{R}^m$
Output: $\mathbf{v} \in \mathbb{R}^m$, and $\sigma, \beta \in \mathbb{R}$ such that $(I - \beta\mathbf{v}\mathbf{v}^\mathsf{T})\mathbf{x} = \pm\|\mathbf{x}\|_2\mathbf{e}_1 = \sigma\mathbf{e}_1$
/* to ensure the sign of σ is opposite of $\mathbf{x}_1$, this is the standard as LAPACK. Helps prevent [illegible] */
$\mathbf{v} \leftarrow \mathbf{x}$
$\sigma \leftarrow -\operatorname{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2$
$\mathbf{v}_1 \leftarrow \mathbf{x}_1 - \sigma$ // this is referred to as $\mathbf{v}$ later on
$\beta \leftarrow -\frac{\mathbf{v}_1}{\sigma}$
$\mathbf{v} \leftarrow \frac{1}{\mathbf{v}_1}\mathbf{v}$
return β, $\mathbf{v}$, σ

3.1.2 Normalization of Householder Vectors

Equation 22 gives a single Householder transformation matrix $\mathbf{P_v}$ for all $\mathbf{v}'$ in Span($\mathbf{v}$), which allows for many different ways of normalizing the Householder vectors as well as the choice of not

Algorithm 3: $\mathbf{V}, \boldsymbol{\beta}, \mathbf{qr}(A)$ Given a matrix $A \in \mathbf{R}^{m\times n}$ where $m \geq n$, return matrix $\mathbf{V} \in \mathbf{R}^{m\times n}$, vector $\boldsymbol{\beta} \in \mathbf{R}^n$, and upper triangular matrix R. An orthogonal matrix Q can be generated from V and $\boldsymbol{\beta}$, and QR = A.

Input: $A \in \mathbf{R}^{m\times n}$ where $m \geq n$
Output: $\mathbf{V}, \boldsymbol{\beta}$, R
$\mathbf{V}, \boldsymbol{\beta} \leftarrow \mathbf{0}_{m\times n}, \mathbf{0}_m$
for $i = 1 : n$ **do**
  $\mathbf{v}, \beta, \sigma \leftarrow \mathtt{hh\_vec}(\mathbf{A}[i : \mathrm{end}, i])$
  $\mathbf{V}[i : \mathrm{end}, i], \boldsymbol{\beta}_i, \mathbf{A}[i, i] \leftarrow \mathbf{v}, \beta, \sigma$ /* Store the Householder vectors and constants */
  /* The next two steps update A */
  $\mathbf{A}[i + 1 : \mathrm{end}, i] \leftarrow \mathrm{zeros}(m - i)$
  $\mathbf{A}[i : \mathrm{end}, i + 1 : \mathrm{end}] \leftarrow \mathbf{A}[i : \mathrm{end}, i + 1 : \mathrm{end}] - \beta\mathbf{v}\mathbf{v}^\top \mathbf{A}[i : \mathrm{end}, i + 1 : \mathrm{end}]$
return $\mathbf{V}, \boldsymbol{\beta}, \mathbf{A}[1 : n, 1 : n]$

Algorithm 4: $\mathbf{QB} \leftarrow \mathtt{hh\_mult}(\mathbf{V}, \mathbf{B})$: Given a set of householder vectors $\{\mathbf{v}_i\}_{i=1}^n$ and their corresponding constants $\{\beta_i\}_{i=1}^n$, compute $\mathbf{P}_1 \cdots \mathbf{P}_n\mathbf{B}$, where $\mathbf{P}_i = \mathbf{I} - \beta_i\mathbf{v}_i\mathbf{v}_i^\top$

Input: $\mathbf{V} \in \mathbf{R}^{m\times n}$, $\boldsymbol{\beta} \in \mathbf{R}^n$ where $m \geq n$. $\mathbf{B} \in \mathbf{R}^{m\times d}$.
Output: QB
/* $\mathbf{v}_1, \ldots, \mathbf{v}_n$ ... and $\mathbf{P}_1 \cdots \mathbf{P}_n\mathbf{B}$ ... */
for $i = 1 : n$ **do**
  $\mathbf{B}_i \leftarrow \mathbf{B}_i - \boldsymbol{\beta}_i\mathbf{v}_i(\mathbf{v}_i^\top \mathbf{B}_i)$
return B

normalizing them. However, this equivalence ($\mathbf{P}_\mathbf{v} = \mathbf{P}_{\mathbf{v}'}$ for all $\mathbf{v}' \in \mathrm{Span}(\mathbf{v})$) is not guaranteed due to rounding errors when using floating point numbers and operations. When using high precision floating point numbers such as double-precision floats, rounding errors that accumulate from the normalization of Householder vectors rarely and barely contribute to the overall stability of the HQR algorithm performed. In contrast, lower precision floating point numbers with limited dynamic range may be more sensitive to the un/normalization choice. For example, if we leave the Householder vectors unnormalized while using half-precision, it is possible to accumulate `Inf`'s in inner products of "large" vectors. As a result, picking a normalization scheme for $\mathbf{v}$ is important in low-precision calculations. Some methods and reasons for the normalization of $\mathbf{v}$ are as follows:

- Set the first element of $\mathbf{v}$, $\mathbf{v}_1$, as 1 for efficient storage of many Householder vectors.
- Set the 2-norm of $\mathbf{v}$ to $\sqrt{2}$ to always have $\beta = 1$.
- Set the 2-norm of $\mathbf{v}$ to 1 to prevent extremely large values, and to always have $\beta = 2$.

The LINPACK implementation of the HQR factorization uses the first method of normalizing via setting $\mathbf{v}_1$ to 1, and is shown in Algorithm 2. The first normalizing method adds an extra rounding error to β and $\mathbf{v}$ each, whereas the remaining methods incur no rounding error in forming $\beta = 1$ and 2 can be represented exactly.

3.2 Rounding Error Analysis

We present an error analysis for the HQR factorization where all inner products are performed with mixed-precision, and all other calculations are done in the storage precision, w.

Assumption 3.2 lays out the generalized mixed-precision inner product we will be using over and over again in the remainder of this paper.

Assumption 3.2. *Let w, p, and s each represent floating point precisions for storage, product, and summation, where the varying precisions are defined by their unit round-off values denoted by u_w, u_p, and u_s, and we can assume $1 \gg u_w \gg u_s$, and $u_p \in \{0, u_w\}$. Within the inner product subroutine, products are done in precision p and summation is done in precision s, and the result stored in precision w. All operations other than inner products are done in the storage precision, w.*

3.2.1 Error analysis for forming Householder Vector and Constant

Calculating the Householder vector and constant is a major routine for the HQR factorization.

Error analysis for $\mathbf{v}$: In this section, we show how to bound the error when employing the mixed precision dot product procedure for Algorithm 2. We begin by extending the inner-product error shown in Lemmas 2.4 and 2.5 to the 2-norm error.

Lemma 3.3 (2-norm round-off error). *Consider a mixed-precision scheme as is outlined in Assumption 3.2. Let $\mathbf{x} \in \mathbf{F}_w^m$ be an arbitrary n-length vector stored in w precision. The forward error bound for computing the 2-norm of $\mathbf{x}$ is*

$$\mathrm{fl}(\|\mathbf{x}\|_2) = (1 + \theta_w^{(d)(z)})\|\mathbf{x}\|_2. \tag{27}$$

where $|\theta_w^{(d)(z)}| \leq \gamma_w^{(d)(z)}|\mathbf{x}|$ for $z \in \{1, 2\}$ and $d := \lfloor \frac{(m-1)u_s}{u_w} \rfloor$.

There is no error incurred in evaluating the sign of a number or flipping the sign. Therefore, the error bound for computing $\sigma = -\text{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2$ is exactly the same as that for the 2-norm.

$$fl(\sigma) = \hat{\sigma} = fl(-\text{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2) = \sigma + \Delta\sigma, \quad |\Delta\sigma| \leq \gamma_w^{(d+1)}|\sigma| \tag{28}$$

Let $\tilde{\mathbf{v}}_1$ be the penultimate value $\mathbf{v}_1$ held ($\tilde{\mathbf{v}}_1 = \mathbf{x}_1 - \sigma$). We can now show the round-off error for $\tilde{\mathbf{v}}_1$ and $\mathbf{v}_i$ where $i = 2, \ldots, n$. Then the round-off errors for $\tilde{\mathbf{v}}_1$ and $\mathbf{v}_i$'s are

$$\begin{aligned} fl(\mathbf{v}_1) &= \mathbf{v}_1 = \tilde{\mathbf{v}}_1 + \Delta\tilde{\mathbf{v}}_1 \\ &= fl(\mathbf{x}_1 - \sigma) = (1+\delta_w)(\sigma + \Delta\sigma) = (1 + \theta_w^{(d+2)})\tilde{\mathbf{v}}_1 \end{aligned}$$

and

$$fl(\mathbf{v}_i) = \hat{\mathbf{v}}_i = fl\left(\frac{\mathbf{x}_i}{\mathbf{v}_1}\right) = (1+\delta_w)\frac{\mathbf{x}_i}{\tilde{\mathbf{v}}_1 + \Delta\tilde{\mathbf{v}}_1} = (1 + \theta_w^{(1+2(d+2))})\tilde{\mathbf{v}}_i$$

The above equalities (as opposed to inequalities) are permitted since θ values are allowed to be flexible within the corresponding γ bounds.

Error analysis for β: Now we show the derivation of round-off error for the Householder constant, β.

$$\begin{aligned} \hat{\beta} = fl\left(-\frac{\mathbf{v}_1}{\hat{\sigma}}\right) &= -(1+\delta_w)\frac{\tilde{\mathbf{v}}_1 + \Delta\tilde{\mathbf{v}}_1}{(\sigma + \Delta\sigma)} \\ &= -(1+\theta_w^{(1)})\frac{(1+\theta_w^{(d+z+2)})\mathbf{v}_1}{(1+\theta_w^{(d+z+1)})\sigma} \\ &= (1+\theta_w^{(d+z+3+2(d+z+1))})\beta \\ &= (1+\theta_w^{(3d+3z+5)})\beta, \end{aligned}$$

where $z = 1$ or $z = 2$ depending on which mixed-precision inner product procedure was used. These two results are formalized in Lemma 3.4 below.

Lemma 3.4. *Given $\mathbf{x} \in \mathbb{R}^m$, consider the constructions of $\beta \in \mathbb{R}$ and $\mathbf{v} \in \mathbb{R}^m$ such that $\mathbf{P_v x} = \sigma\mathbf{e}_1$ (c.f. Lemma 3.1) by using Algorithm 2. Then the forward error of forming $\mathbf{v}$ and β with the floating point arithmetic with the mixed-precision scheme outlined in Assumption 3.2 are*

$$\begin{aligned} \|\hat{\mathbf{v}}\|_2 &= (1+\theta_w^{(1+2(d+z+2))})\|\mathbf{v}\|_2, \text{ and} \\ \hat{\beta} &= (1+\theta_w^{(3d+3z+5)})\beta, \end{aligned}$$

where $z \in \{1, 2\}$ and $d = \lfloor \frac{(m-1)u_r}{u_w} \rfloor$.

3.2.2 Applying a Single Householder Transformation

Applying a Householder transformation is implemented by a series of inner and outer products, since Householder matrices are rank-1 updates of the identity. This is much less costly than forming $\mathbf{P_v}$, then performing matrix-vector or matrix-matrix multiplications. For some $\mathbf{P_v} = I - \beta\mathbf{v}\mathbf{v}^\top$, we result in the following computation.

$$\mathbf{P_v x} = (I - \beta\mathbf{v}\mathbf{v}^\top)\mathbf{x} = \mathbf{x} - (\beta\mathbf{v}^\top\mathbf{x})\mathbf{v} \tag{29}$$

Applying $\mathbf{P_v}$ to zero out the target column of a matrix. Let $\mathbf{x} \in \mathbb{R}^m$ be the target column we wish to zero out beneath the first element. Recall that we chose a specific $\mathbf{v}$ such that $\mathbf{P_v x} = \sigma\mathbf{e}_1$. As a result, the only error lies in the first element, σ, and that is shown in Equation 28. Note that the normalization choice of $\mathbf{v}$ does not impact the Householder transformation matrix ($\mathbf{P_v}$) nor its action on $\mathbf{x}$, $\mathbf{P_v x}$.

Applying $\mathbf{P_v}$ to the remaining columns of the matrix. Now, let $\mathbf{x}$ and $\mathbf{v}$ have no special relationship, as $\mathbf{v}$ was constructed given some preceding column.

Set $\mathbf{w} = \beta\mathbf{v}^\top\mathbf{x}\mathbf{v}$. Note that $\mathbf{x}$ is exact, whereas $\mathbf{v}$ and β were still computed with floating point operations. The errors incurred from computing $\mathbf{v}$ and β need to be included as well as the new rounding errors accumulating from the action of applying $\mathbf{P_v}$ to a column.

We show the error for forming $fl(\hat{\mathbf{v}}^\top\mathbf{x})$ first.

$$\begin{aligned} fl(\hat{\mathbf{v}}^\top\mathbf{x}) &= (1+\theta_w^{(d+z)})(\mathbf{v} + \Delta\mathbf{v})^\top\mathbf{x}, \\ &\quad \text{where } \theta_w^{(d+z)} \text{ is incurred from the action of a dot product,} \\ &= (1+\theta_w^{(d+z)})(1+\theta_w^{(1+2(d+z+2))})\mathbf{v}^\top\mathbf{x} \\ &= (1+\theta_w^{(3d+3z+5)})\mathbf{v}^\top\mathbf{x} \end{aligned}$$

Now we can form $fl(\mathbf{w})$.

$$\begin{aligned} \hat{\mathbf{w}} &= (1+\theta_w^{(2)})(\beta + \Delta\beta)(1+\theta_w^{(3d+3z+5)})\mathbf{v}^\top\mathbf{x}\mathbf{w} \\ &\quad \text{where } \theta_w^{(2)} \text{ results from multiplying } \beta \text{ and } \mathbf{v}^\top\mathbf{x} \text{ to } \mathbf{w}, \\ &= (1+\theta_w^{(2)})(1+\theta_w^{(3d+3z+5)})\beta(1+\theta_w^{(3d+3z+5)})\mathbf{v}^\top\mathbf{x}\mathbf{w} \\ &= (1+\theta_w^{(6d+6z+12)})\mathbf{w} \end{aligned}$$

Finally, we can add in the vector subtraction operation and complete the rounding error analysis of applying a Householder transformation to any vector.

$$fl(\mathbf{P_v x}) = fl(\mathbf{x} - \hat{\mathbf{w}}) = (1+\delta_w)(1+\theta_w^{(6d+6z+12)})\mathbf{w} \tag{30}$$

$$= (1+\theta_w^{(6d+6z+13)})\mathbf{P_v x} \tag{31}$$

$$= (\mathbf{P_v} + \Delta\mathbf{P_v})\mathbf{x}, \quad \|\Delta\mathbf{P_v}\|_F \leq \gamma_w^{(6d+6z+13)} \tag{32}$$

Details behind the matrix norm error bound in Equation 32 is shown in the appendix. Constructing both $\mathbf{Q}$ and $\mathbf{R}$ relies on applying Householder transformations in the above two ways: 1) to zero out below the diagonal of a target column, and 2) to update the bottom right submatrix. We now have the tools to formulate the forward error bound on $\mathbf{Q}$ and $\mathbf{R}$ calculated from the HQR factorization.

3.2.3 HQR Factorization Forward Error Analysis

Consider a thin QR factorization where $\mathbf{A} \in \mathbb{R}^{m\times n}$ for $m \geq n$, we have $\mathbf{Q} \in \mathbb{R}^{m\times n}$ and $\mathbf{R} \in \mathbb{R}^{n\times n}$. The pseudo-algorithm in Section 3 shows that each successive Householder transformation is applied to a smaller lower right submatrix each time.

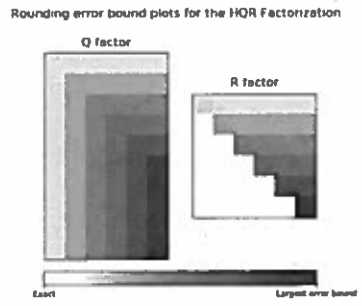

Figure 1: Grayscale representation of distribution of rounding errors bounds for the HQR algorithm.

Consequently, rounding error bounds for each element of R and Q can be specifically computed by its location within the matrices, as is displayed in Figure 1 for a 10-by-6 example. Instead of continuing with a componentwise analysis of how accumulated rounding errors are distributed by HQR, we transition into normwise error analyses. To do this, we use the analysis in shown in the preceding section (summarized in Equation 32 to implicitly form the matrix norm error of the Householder transformation matrix, $\mathbf{P_v}$. Then, we use the result of Lemma 3.7 in [3] to get a normwise bound on the perturbation effect of multiple matrix multiplications. This result is shown in Theorem 3.5, and the proof is shown extensively in A.3.

Theorem 3.5. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ have full rank, n. Let $\hat{\mathbf{Q}} \in \mathbb{R}^{m\times n}$ and $\hat{\mathbf{R}} \in \mathbb{R}^{n\times n}$ be the thin QR factors of $\mathbf{A}$ obtained via the HQR algorithm with a mixed-precision scheme as is outlined in Assumption 3.2. Let $d = \lfloor \frac{(m-1)u_s}{u_w} \rfloor$, and $z = 1$ or $z = 2$. Then we have normwise forward error bounds*

$$\hat{\mathbf{R}} = \mathbf{R} + \Delta\mathbf{R} = \mathbf{P}_n \cdots \mathbf{P}_1 \mathbf{A}, \tag{33}$$

$$\hat{\mathbf{Q}} = \mathbf{Q} + \Delta\mathbf{Q} = \hat{\mathbf{P}}_1 \cdots \hat{\mathbf{P}}_n \mathbf{I}, \tag{34}$$

$$\tag{35}$$

where

$$\|\Delta\mathbf{Q}\|_F \leq n^{3/2} z \gamma_w^{(6d+6z+13)}, \tag{36}$$

and for column j in $\{1, \ldots, n\}$,

$$\|\Delta\mathbf{R}[:,j]\|_2 \leq j z \gamma_w^{(6d+6z+13)} \|\mathbf{A}[:,j]\|_2. \tag{37}$$

We can also form a backward error. Let $\mathbf{A} + \Delta\mathbf{A} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$, where $\mathbf{Q}$ and $\mathbf{R}$ are obtained via Algorithm 3. Then,

$$\|\Delta\mathbf{A}\|_F \leq n^{3/2} z \gamma_w^{(6d+6z+13)} \|\mathbf{A}\|_F \tag{38}$$

3.2.4 HQR Comparison to Uniform Precision Analysis

The mixed-precision segments of the analysis behind Theorem 3.5 derive from the mixed-precision inner product scheme outlined in Assumption 3.2, and is propagated to form the error bounds for a single Householder transformation as is shown in Equation 32. All steps to form the error bounds in Theorem 3.5 from the error bound for a single Householder transformation (Equation 32) directly follow the analyses in Section 19.3 of [3]. In these steps, we generalize the single Householder transformation error bound,

$$\text{fl}(\mathbf{P_v}\mathbf{x}) = (\mathbf{P_v} + \Delta\mathbf{P_v})\mathbf{x}, \qquad \|\Delta\mathbf{P_v}\|_F \leq \epsilon, \tag{39}$$

for some small quantity $0 < \epsilon \ll 1$, and propagate it through the for-loop in Algorithm 3. This then results in forward error bound coefficients $m\epsilon$ or $n^{3/2}\epsilon$. Since this ϵ value remains constant, the rounding error analysis for both mixed-precision and uniform-precision schemes are essentially the same with different values for ϵ. The uniform precision equivalent of Equation 32 is shown in Equation 40.

$$\text{fl}(\mathbf{P_v}\mathbf{x}) = (\mathbf{P_v} + \Delta\mathbf{P_v})\mathbf{x}, \qquad \|\Delta\mathbf{P_v}\|_F \leq \tilde{\gamma}^{(m)}, \tag{40}$$

which is derived in detail in [3]. Therefore, we only need to compare $\gamma^{(6d+6z+13)}$ against $\tilde{\gamma}^{(cm)}$, where c is a small integer. Although d relies on both m and the precisions w and s, we can generally assume that $cm \gg (6d + 6z + 13)$ in most mixed-precision settings. Therefore, the new bounds in Theorem 3.5 are much tighter than the existing ones, and more accurately describe the kind of rounding error accumulated in mixed-precision computational settings.

4 Tall-and-Skinny QR

Some important problems that require QR factorizations of overdetermined systems include least squares problems, eigenvalue problems, low rank approximations, as well as other matrix decompositions. Although Tall-and-Skinny QR (TSQR) broadly refers to row-block QR factorization methods, we will discuss a specific variant of TSQR which is also known as the AllReduce algorithm [5]. In this paper, the TSQR/AllReduce algorithm refers to the most parallel variant of all row-block QR factorization algorithms discussed in [6]. A detailed description and rounding error analysis of this algorithm can be found in [5], and we present a pseudocode for the algorithm in Algorithm 5. Our initial interest in this algorithm came from its parallelizable nature, which is particularly suitable to implementation on GPUs and could lead to speed-ups. Additionally, our numerical simulations (discussed in Section 4.3) show that TSQR can not only increase the speed but also outperform the traditional HQR factorization in low precisions.

4.1 TSQR/AllReduce Algorithm

Algorithm 5 takes a tall-and-skinny matrix, **A** and organizes it into row-blocks. HQR factorization is performed on each of those blocks, and pairs of **R** factors are combined to form the next set of **A** matrices to be QR factorized. This is repeated until only a single **R** factor remains, and the **Q** factor is built from all of the Householder constants and vectors stored at each level. The most gains from parallelizing can be made in the initial level where the maximum number of independent HQR factorizations occur. For any given tall-and-skinny matrix, more than one configuration of this algorithm may be given. (e.g. a 1600-by-100 matrix can be partitioned into 2, 4, 8, or 16 initial row-blocks, but a 1600-by-700 matrix can only be partitioned into 2 initial blocks. The choice in the initial partition may be given by the number of nodes available, the shape of the matrix, and determine the recursion depth which we call level. Our numerical experiments show that this choice has an impact in the accuracy of the QR factorization.

A level refers to number of recursions in a particular TSQR implementation. An L-level TSQR algorithm partitions the original matrix into 2^L submatrices in the initial or 0^{th} level of the algorithm, and 2^{L-i} QR factorizations are performed in level i for $i = 1, \ldots, L$. The set of matrices

that are QR factorized at each level i are called $\mathbf{A}_j^{(i)}$ for $j = 1, \ldots, 2^{L-i}$, where superscript (i) corresponds to the level, and the subscript j indexes the row-blocks within level i. In the following sections, Algorithm 5 (tsqr) will find a TSQR factorization of a matrix $A \in \mathbf{R}^{m \times n}$ where $m \gg n$. The inline function qr refers to Algorithm 3, hh_mult is Algorithm 4, and we use Algorithm 2 as a subroutine of qr.

4.1.1 TSQR Notation

Due to the multi-level nature of the TSQR algorithm, some new notation are needed to avoid confusion. In the final task of constructing $\mathbf{Q}$, we need to aggregate the $\mathbf{Q}$ factors from each block at each level. We partition each $\mathbf{Q}_j^{(i)}$ factor from level i and apply $\mathbf{Q}^{(i-1)}$ factors from level $i-1$ to them. We write $\mathbf{Q}_j^{(i)} = [\hat{\mathbf{Q}}_{j,1}^{(i)\mathsf{T}}\ \hat{\mathbf{Q}}_{j,2}^{(i)\mathsf{T}}]^{\mathsf{T}}$ where the partition (approximately) halves $\mathbf{Q}_j^{(i)}$. The functions $o(j)$ and $\phi(j)$ are defined such that $\mathbf{Q}_j^{(i)}$ is applied to $\hat{\mathbf{Q}}_{o(j),\phi(j)}^{(i+1)}$. For $j = 1, \ldots, 2^{L-i}$ at level i, we need $j = 2(o(j)-1)+\phi(j)$ where $o(j) = \lceil \frac{j}{2} \rceil$ and $\phi(j) = 2+j-2o(j)$. Section 4.1.2 shows full linear algebra details for a single-level ($L = 1$, 2 initial blocks) example. The reconstruction of $\mathbf{Q}$ can be implemented more efficiently (c.f. [7]), but the reconstruction method in Algorithm 5 is presented for a clear, straightforward explanation.

4.1.2 Single-level Example

In the single-level version of this algorithm, we first bisect $\mathbf{A}$ into $\mathbf{A}_1^{(0)}$ and $\mathbf{A}_2^{(0)}$ and compute the QR factorization of each of these submatrices. We combine the resulting upper-triangular matrices, i.e., $\mathbf{A}_1^{(1)} = \begin{bmatrix} \mathbf{R}_1^{(0)} \\ \mathbf{R}_2^{(0)} \end{bmatrix}$, which we QR factorize and we repeat etc.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1^{(0)} \\ \mathbf{A}_2^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)}\mathbf{R}_1^{(0)} \\ \mathbf{Q}_2^{(0)}\mathbf{R}_2^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2^{(0)} \end{bmatrix} \begin{bmatrix} \mathbf{R}_1^{(0)} \\ \mathbf{R}_2^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2^{(0)} \end{bmatrix} \mathbf{A}_1^{(1)} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2^{(0)} \end{bmatrix} \mathbf{Q}_1^{(1)}\mathbf{R}$$

Whereas the R factor of $\mathbf{A}_1^{(1)}$ is the final R factor of the QR factorization of the original matrix, $\mathbf{A}$, we still need to construct $\mathbf{Q}$. Bisecting $\mathbf{Q}_1^{(1)}$ into two submatrices ($\mathbf{Q}_{1,1}^{(1)}$ and $\mathbf{Q}_{1,2}^{(1)}$) allows us to write and compute the product more compactly, i.e

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2^{(0)} \end{bmatrix} \mathbf{Q}_1^{(1)} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2^{(0)} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{Q}}_{1,1}^{(1)} \\ \hat{\mathbf{Q}}_{1,2}^{(1)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)}\hat{\mathbf{Q}}_{1,1}^{(1)} \\ \mathbf{Q}_2^{(0)}\hat{\mathbf{Q}}_{1,2}^{(1)} \end{bmatrix}$$

Algorithm 5 takes a tall-and-skinny matrix $\mathbf{A}$ and level L and finds a QR factorization by initially partitioning $\mathbf{A}$ into 2^L row-blocks. This variant includes the building of the final $\mathbf{Q}$,

4.2 TSQR Rounding Error Analysis

The TSQR algorithm presented in Algorithm 5 is a divide-and-conquer strategy for the QR factorization that uses the HQR for the subproblems. Divide-and-conquer methods can naturally be implemented in parallel and accumulate less rounding errors. For example, the single-level TSQR decomposition of a tall-and-skinny matrix, $\mathbf{A}$ requires 2 total HQRs of matrices of sizes $\lfloor \log_2(\frac{m}{n}) \rfloor$-by-$n$, $\lfloor \log_2(\frac{m}{n}) \rfloor$-by-$n$, and $2n$-by-n. The single-level TSQR strictly uses more FLOPs, but the dot

Algorithm 5: Q, R = tsqr(A, L). Finds a QR factorization of a tall, skinny matrix, A.

Input: $\mathbf{A} \in \mathbf{R}^{m \times n}$ where $m \gg n$, $L \leq \lfloor \log_2(\frac{m}{n}) \rfloor$, and 2^L is the initial number of blocks.
Output: $\mathbf{Q} \in \mathbf{R}^{m \times n}$, $\mathbf{R} \in \mathbf{R}^{n \times n}$ such that $\mathbf{QR} = \mathbf{A}$

```
h ← ⌊m/2^L⌋                                   // Number of rows for all but the last block
r ← m − (2^L − 1)h                            // Number of rows for the last block
/* Split A into 2^L blocks. Data may become unbalanced. */
for j = 1 : 2^L − 1 do
    A_j^(0) ← A[(j − 1)h + 1 : jh, :]
A_{2^L}^(0) ← A[(2^L − 1)h : m, :]            // Last block may have more rows
/* Store Householder vectors in columns of array Y^(0), Householder constants
   in components of vector β^(0), and set up the next level. */
for i = 0 : L − 1 do
    /* The inner loop can be parallelized */
    for j = 1 : 2^{L−i} do
        Y_{2j−1}^(i), β_{2j−1}^(i), R_{2j−1}^(i) ← qr(A_{2j−1}^(i))
        Y_{2j}^(i), β_{2j}^(i), R_{2j}^(i) ← qr(A_{2j}^(i))
        A_j^(i+1) ← [ R_{2j−1}^(i) ; R_{2j}^(i) ]
/* At the bottom level, get the final R factor. */
Y_1^(L), β_1^(L), R ← qr(A_1^(L))
Q_1^(L) ← hh_mult(Y_1^(L), I_{2n×n})
/* Compute Q factor by applying stored Householder vectors, Y^(i), to Q
   factors. */
for i = L − 1 : −1 : 1 do
    for j = 1 : 2^{L−i} do
        Q_j^(i) ← hh_mult( Y_j^(i), [ Q̂_{o(j),φ(j)}^(i+1) ; 0_{n×n} ] )
/* At the top most level, compute the final Q factor from Q^(1) factors. */
Q ← [].
for j = 1 : 2^L do
    Q ← [ Q ; hh_mult( Y_j^(0), [ Q̂_{o(j),φ(j)}^(1) ; 0_{k×n} ] ) ]
return Q, R
```

product subroutines may accumulate smaller rounding errors and certainly have smaller upper bounds since they are performed on shorter vectors, and lead to a more accurate solution overall. These concepts are elucidated in [5], where the rounding error analysis of TSQR is shown in detail in [5]. We summarize the main results in Theorem 4.1.

Theorem 4.1. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ have full rank, n, and $\hat{\mathbf{Q}} \in \mathbb{R}^{m\times n}$ and $\hat{\mathbf{R}} \in \mathbb{R}^{n\times n}$ be the thin QR factors of $\mathbf{A}$ obtained via Algorithm 5. Then we have normwise forward error bounds*

$$\hat{\mathbf{A}} = \mathbf{A} + \Delta\mathbf{A} = \mathbf{Q}(\mathbf{R} + \Delta\mathbf{R}).$$
$$\hat{\mathbf{Q}} = \mathbf{Q} + \Delta\mathbf{Q}.$$

where

$$\|\Delta\mathbf{R}\|_F, \|\Delta\mathbf{A}\|_F \leq \left[n\tilde{\gamma}_{\frac{m}{2L}} + (1+n\tilde{\gamma}_{\frac{m}{2L}})\left\{(1+n\tilde{\gamma}_{2n})^L - 1\right\}\right]\|\mathbf{A}\|_F. \tag{41}$$

and

$$\|\Delta\mathbf{Q}\|_F \leq \sqrt{n}\left[(1+n\tilde{\gamma}_{\frac{m}{2L}})(1+n\tilde{\gamma}_{2n})^L - 1\right]. \tag{42}$$

Furthermore, if we assume $n\tilde{\gamma}_{\frac{m}{2L}}, n\tilde{\gamma}_{2n} \ll 1$, the coefficient for $\|\mathbf{A}\|_F$ in Equations 41 can be approximated as

$$\left[n\tilde{\gamma}_{\frac{m}{2L}} + (1+n\tilde{\gamma}_{\frac{m}{2L}})\left\{(1+n\tilde{\gamma}_{2n})^L - 1\right\}\right] \simeq n\tilde{\gamma}_{\frac{m}{2L}} + Ln\tilde{\gamma}_{2n}, \tag{43}$$

and the right hand side of Equation 42 can be approximated as

$$\sqrt{n}\left[(1+n\tilde{\gamma}_{\frac{m}{2L}})(1+n\tilde{\gamma}_{2n})^L - 1\right] \simeq \sqrt{n}\left(n\tilde{\gamma}_{\frac{m}{2L}} + Ln\tilde{\gamma}_{2n}\right) \tag{44}$$

We can also form a backward error, where $\mathbf{A} + \Delta\mathbf{A}_{TSQR} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$, and both $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$ are obtained via Algorithm 5. Then,

$$\|\Delta\mathbf{A}_{TSQR}\|_F = \|\mathbf{Q}\Delta\mathbf{R} + \Delta\mathbf{Q}\hat{\mathbf{R}}\|_F \simeq \sqrt{n}\left(n\tilde{\gamma}_{\frac{m}{2L}} + Ln\tilde{\gamma}_{2n}\right)\|\mathbf{A}\|_F \tag{45}$$

In Section 3.2.4, we discussed how the steps of the HQR algorithm result as some multiple of ϵ, where ϵ is the error bound for a single Householder transformation and is described in Equation 39. Similarly, the analysis behind Theorem 4.1 can be generalized via defining ϵ_1 to be the error bound for a single Householder transformation corresponding to the vector length at the initial level 0, $\frac{m}{2L}$, and ϵ_2 to be the error bound for a Householder transformation corresponding to vector length in all deeper levels, $2n$. This generalization leads to the error bound coefficients

$$n\epsilon_1 + Ln\epsilon_2 \quad \text{for} \quad \|\Delta\mathbf{Q}\|_F, \|\Delta\mathbf{A}_{TSQR}\|_F \tag{46}$$
$$\sqrt{n}(n\epsilon_1 + Ln\epsilon_2) \quad \text{for} \quad \|\Delta\mathbf{R}\|_F, \|\Delta\mathbf{A}\|_F. \tag{47}$$

In a uniform-precision setting, these correspond to

$$\epsilon_1 = \gamma_{\frac{m}{2L}} \quad \text{and} \quad \epsilon_2 = \gamma_{2n}, \tag{48}$$

and in the mixed-precision setting outlined in Assumption 3.2, they correspond to

$$\epsilon_1 = \gamma_{u_w}^{(d_1+6L+13)}, \; d_1 = \lfloor(\frac{m}{2L}+1)\frac{u}{u_w}\rfloor, \text{ and,} \tag{49}$$
$$\epsilon_2 = \gamma_{u_w}^{(d_2+6L+13)}, \; d_2 = \lfloor\frac{(2n+1)u}{u_w}\rfloor \tag{50}$$

In both settings, we see that increasing L may decrease ϵ_1 but still increase the overall bounds as the larger L still could have an adverse effect on the coefficients in Theorem 4.1. This trade-off is precisely the balance between the size of initial blocks and the number of levels in the TSQR algorithm, and an optimal TSQR scheme would ideally minimize ϵ_1 and ϵ_2 with the choice of L. These error bounds are studied in detail in Section 4.2.4 following Section.

4.2.1 HQR and TSQR error bound comparison

We compare the error bounds for HQR and TSQR algorithms. Let's consider the larger error bounds in the uniform precision equivalents of Theorems 3.5 and 4.1, which are the bounds of $\Delta\mathbf{Q}$ and $\Delta\mathbf{A}$. In order for the a meaningful TSQR error bound to outperform the bound for the HQR algorithm, we need integers $m, n > 0$, and $L \geq 0$ such that,

$$1 \gg n^{3/2}\gamma^{(m)} \gg n^{3/2}(\gamma^{(\frac{m}{2L})} + L\gamma^{(2n)}).$$

If we assume $\frac{m}{2L} = 2n$, the HQR bound is $\frac{4L}{3}$ larger than the bound for TSQR with L levels. For example, in single precision, a HQR of a 2^{17}-by-2^6 matrix results in an upper bound relative backward error ($\|\mathbf{A} - \mathbf{QR}\|_F / \|\mathbf{A}\|_F$) of ≈1.002, but a TSQR with $L = 8$ is bounded by ≈3.516e-02. This exemplifies a case in which stability is not guaranteed in HQR, but is stable when using TSQR even in the worst-case. Let's now consider some 2^{20}-by-2^{12} matrix that HQR factorizations performed with double precision. The error bound for HQR is 1.686e-7 whereas the error bound for TSQR with 12 levels, which is 5.351e-10. In general, we can conjecture that values of L that can make $m2^{-L}$ and $2Ln$ much smaller than m should produce a TSQR that outperforms HQR in worst-case scenarios, at least in uniform precision settings. However, the range of matrix sizes that TSQR can accommodate decreases as L grows larger. Figure 2 shows the matrix sizes HQR, 2-level TSQR and 4-level TSQR can accommodate as well as their respective error bounds.

Let's consider a mixed-precision setting such as in Assumption 3.2, and we assume $u_p = u_w$ so that $z = 2$. In order for the a meaningful TSQR error bound to outperform the bound for the HQR algorithm, we now need integers $m, n > 0$, and $L \geq 0$ such that,

$$1 \gg n^{3/2}\gamma_{u_w}^{(6d+25)} \gg n^{3/2}(\gamma_{u_w}^{(6d_1+25)} + L\gamma_{u_w}^{(6d_2+25)}),$$

where $d = \lfloor\frac{(m-1)u}{u_w}\rfloor$, $d_1 = \lfloor(\frac{m}{2L}-1)\frac{u}{u_w}\rfloor$, and $d_2 = \lfloor\frac{(2n-1)u}{u_w}\rfloor$.

In contrast to the analysis for uniform precision settings, large L values do not necessarily reduce the error bounds of TSQR. While large L can imply $m \gg m2^{-L} + 2Ln$, it doesn't always lead to $d \gg d_1 + Ld_2$. Although the theoretical error bounds do not give a clear indication of the worst-case performances of HQR and TSQR in mixed-precision settings, our numerical simulations show that TSQR can still outperform HQR in ill-conditioned matrices. These experiments are discussed in detail in Section 4.3 in the next section.

4.3 Numerical Experiment

In Section 4.2.4, we theorized that conditions exist where TSQR could outperform HQR, and that these conditions were hard to identify in mixed-precision settings. An empirical comparison of these two QR factorization algorithms in double precision can be found in [5], where they conclude that deeper TSQR tends to produce more accurate QR factorizations than HQR. However, using TSQR with deep levels (large L) can actually start to perform worse than TSQR with shallower

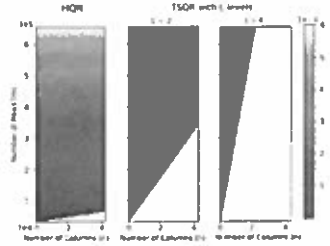

Figure 2: Non-white space indicates allowable matrix sizes for each scheme, and color map represents error bounds for $\|\Delta Q\|_F$ in double precision.

levels (smaller L). We instead focused on comparing HQR and TSQR performances in a mixed-precision setting. Our numerical simulations show that TSQR can still outperform HQR in low, mixed-precision settings in practice, even though the theoretical bounds don't guarantee stability. Our empirical results do not behave as the theoretical bounds suggest, and even show opposite trends at times. This discrepancy highlights the shortcomings of deterministic error bounds that are too pessimistic.

4.3.1 Experiment Details

We used Julia v1.0.4 for all of the numerical simulations. This programming language allows half precision storage as well as `castup` and `castdown` operations to and from single and double precisions, but has no half precision arithmetic. Therefore, we relied on using Algorithm 1 for $f \in \mathrm{OP} \cup \{\texttt{dot\_product}\}$ to simulate half and mixed-precision arithmetic operations. For HQR, we created a mixed-precision version of the LAPACK routine xGEQRF where the dot product subroutine was approximated by $\mathrm{fl}(\mathbf{x}_{\text{half}}^T \mathbf{y}_{\text{half}})$ with `simHalf(dot_product, x_half, y_half)` to simulate the mixed-precision setting described in Assumption 3.2 with $u_p = 0$ (which implies $z = 1$), and used Algorithm 1 on all other basic operations in OP to simulate half/storage precision arithmetic. This HQR was then used as a subroutine of TSQR as well. All in all, our experiments nearly replicated the mixed-precision setting we assumed for the error analysis in Sections 3.2 and 4.2.

We generated random matrices of size 4000-by-100 and computed their HQR and TSQR for $L = 1, \ldots, 6$ in a mixed-precision setting that simulates Assumption 3.2 with $z = 1$. The relative backward error, $\|\mathbf{QR} - \mathbf{A}\|_F / \|\mathbf{A}\|_F$, was computed by casting up $\mathbf{Q}$, $\mathbf{R}$,and $\mathbf{A}$ to double precision to compute the Frobenius norms. The mixed-precision HQR error bounds $n^{\frac{(6d+6z+13)}{2}}$ and $n^{3/2}\gamma_{2^{(6d+6z+13)}}$ for $m = 4000$ and $n = 100$ are 0.936 and 9.364 respectively, and the mixed-precision TSQR bounds for $L = 1, \ldots, 5$ are even larger. This indicates that our error bounds do not guarantee stability.

Following example from [5], we used n-by-n random matrices, $\mathbf{A}_\alpha$, constructed via

$$\mathbf{A}_\alpha = \frac{\mathbf{Q}'(\alpha\mathbf{E} + \mathbf{I})}{\|\mathbf{Q}'(\alpha\mathbf{E} + \mathbf{I})\|_F}, \tag{51}$$

where $\mathbf{Q}' \in \mathbb{R}^{n\times n}$ is a random orthogonal matrix and $\mathbf{E} \in \mathbb{R}^{n\times n}$ is the matrix of 1's. The random orthogonal matrix $\mathbf{Q}'$ is generated by taking a QR factorization of an iid 4000-by-100 matrix sampled from $Unif(0,1)$, and we used the built-in QR factorization function in Julia. By construction, $\mathbf{A}_\alpha$ has 2-norm condition number $n\alpha + 1$. By varying α from 1e-4 to 1, we varied the condition number from 1.1 to 101, and we generated 10 samples for each value of α.

4.3.2 Results

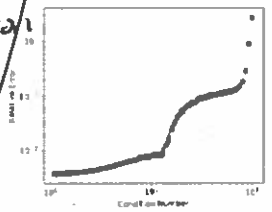

Figure 3: HQR errors for matrices with varying condition numbers.

We computed the relative backward error, $\frac{\|\mathbf{QR} - \mathbf{A}\|_F}{\|\mathbf{A}\|_F}$, of the traditional HQR algorithm (Figure 3) as well as the errors of TSQR with $L = 1, \ldots, 5$ (Figure 4). Figure 3 shows the errors for HQR plotted against the theoretical condition number, and Figure 4 shows the errors for HQR and TSQR for our test matrices where $L = 0$ is the HQR, and $L = 1, \ldots, 5$ correspond to TSQR with L levels. The black markers on the vertical axis of Figure 4 correspond to the HQR errors shown in Figure 3.

Figure 3 shows that the error of the QR factorization worsens as the condition number increases, and there is a monotonically increasing trend between errors and condition numbers. Figure 4 reveals two different trends for the errors as we compare HQR to TSQR with only one level, then to deeper levels of TSQR. One trend occurs for matrices with smaller condition numbers, where HQR and all levels of TSQR are stable, but deepening the levels of TSQR worsens the errors. Another trend occurs for matrices with higher condition numbers, where single-level and 2-level TSQR yield smaller errors than HQR. In these cases, TSQR with 3 or more levels have errors similar to or worse than 2-level TSQR, but these errors tend to not rise above the HQR errors.

These results suggests that TSQR can significantly outperform HQR even in mixed-precision settings, and particularly when HQR is unstable due to high condition numbers. Although this experiment focused on condition numbers, identifying other properties that point to better performances of TSQR than HQR can further broaden the potential use of mixed-precision TSQR in applications.

5 Applications

Many applications in scientific computing typically employ double precision when lower precision may actually be sufficient. However, identifying which applications can tolerate lower precision and still achieve sufficient results is a challenging task that wasn't particularly relevant prior to the emergence of new technology that supports low precision. Since low and mixed precision settings benefit from speed-ups and lighter storage, applications that process large amounts of data

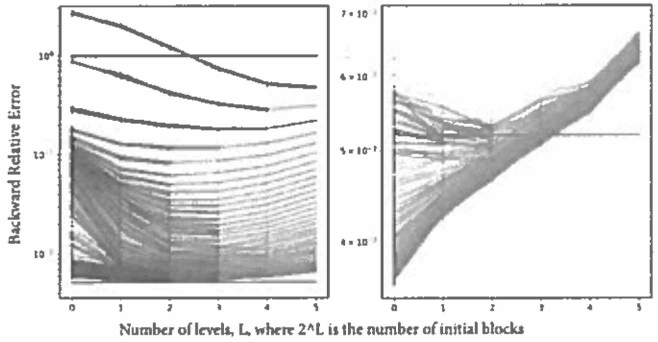

Figure 4: Left plot shows the relative error of QR factorization for matrices with condition numbers ranging from 5.3 to 101, and the right plot shows the errors for matrices with condition numbers ranging from 1.1 to 5.3.

are potential candidates for this research. In this paper, we discuss our results from applying our mixed-precision HQR as a subroutine of an iterative eigensolver in the context of spectral clustering.

~~Graph partitioning.~~ A graph is defined by a set of nodes and a set of edges between the nodes. Partitioning, or clustering, is a task that seeks communities within a graph such that nodes within a community are *more similar* to each other than to nodes outside of that community. In datasets where the true communities are known, we can use pairwise-precision and pairwise-recall (c.f.[?]) which are defined in Definition 5.1) to evaluate the accuracy of a clustering task.

Definition 5.1. Pairwise-precision and pairwise-recall are measured by checking for every pair of nodes if the pair is classified into the same cluster (positive), or else (negative).

$$\text{Precision} = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Positive}}, \tag{52}$$

$$\text{Recall} = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Negative}}. \tag{53}$$

5.1 Spectral Graph ~~Partitioning~~

Some spectral clustering methods utilize identifying k dominant eigenvectors of a similarity matrix of a graph, which then can be used to identify k clusters. We will use subspace iteration, a variant of the power method defined in Algorithm 6, which uses a QR factorization of a tall-and-skinny matrix at each iteration and converges to dominant eigenvectors quickly. Although we only experimented

with comparing mixed-precision HQR to uniform precision HQR, TSQR could also be used in this application. Another potential use of iterative eigensolvers for spectral clustering is in identifying the second smallest eigenvalue and its eigenvector pair, called Fiedler value and vector. In addition, many eigenproblems outside of spectral clustering only require finding a few eigen pairs. This family of problems tends to admit tall-and-skinny matrix structures and could utilize TSQR as well.

5.1.1 Subspace Iteration

Subspace iteration is a modification of the power method, which computes an invariant subspace with dimension $p > 1$ (c.f.[8]). A variant of this algorithm is shown below in Algorithm 6.

Algorithm 6: Q = `subIter(A,max_iter,`τ`,k)`. Find orthogonal basis (given by columns of output matrix Q) of an invariant subspace of the input adjacency matrix, A.

Input: Adjacency matrix $A \in \{0,1\}^{m\times m}$ where $m \geq n$, `max_iter`, the maximum number of iterations, τ the threshold for the eigenspace error, and k, the suspected number of clusters.

Output: Q

```
Initialize Y ∈ R^{m×k}, a random matrix.
Q, R ← qr(Y) for i = 1, ..., max_iter do
    Y ← AQ
    if ‖Y − QQ^T Y‖_2 / ‖Y‖_2 < τ then
        exit loop.
    Q, R ← qr(Y)
return Q
```

This algorithm is an iterative method with two possible stopping criteria.

1. The maximum number of iterations to complete before exiting the loop is declared as max_iter.
2. If the eigenspace error is smaller than τ, then exit the loop.

In practice, we added a third stopping criterion in the case that the declared τ value was too small, which would force an exit from the loop when the eigenspace error began to increase.

5.1.2 Density-based Spatial clustering of Applications with Noise (DBSCAN)

DBSCAN is a density-based spatial clustering algorithm introduced in [9] and is widely used in practice. This algorithm only requires input data, location of nodes, and two parameters, radius of neighborhoods, and minPts, minimum number of points required to form a dense region. The two parameters for the DBSCAN algorithm were tuned to provide the best result, given that we used the same set of parameters for the entire experiment.

5.2 Experiment Details and Results

Our main goal in this experiment was to test if the eigenspaces identified by lower precision HQR could produce sufficient graph partitioning. We used subspace iteration (Algorithm 6) to identify

eigenspaces, DBSCAN to partition the embedding of the nodes onto these eigenspaces, and precision and recall to evaluate clustering performance. We used a static graph of 5000 nodes with 19 known true partitions for the Graph Challenge [10]. The graphs we used were undirected and unweighted — the only elements in the adjacency matrices were 0's and 1's which can easily be represented in half, single, and double precision floats. For $i = 1, \ldots, 10$, let $\mathbf{Y}_{\text{half},i} \in \mathbb{F}_{\text{half}}^{5000\times 19}$ be the half precision storage of the i^{th} random matrix. Since any half precision float can be exactly represented in single and double precisions, $\mathbf{Y}_{\text{half},i}$'s can be easily cast up to single and double precisions, $\mathbf{Y}_{\text{single},i}$ and $\mathbf{Y}_{\text{double},i}$. We performed mixed-precision HQR within subspace iteration initialized by $\mathbf{Y}_{\text{half},i}$'s, and uniform-precision HQR for subspace iteration initialized by $\mathbf{Y}_{\text{single},i}$'s and $\mathbf{Y}_{\text{double},i}$'s. For trial $i = 1, \ldots, 10$, we repeated the following steps:

Step 1. Identify an orthogonal basis of dimension 19 with subspace iteration using the appropriate HQR routine for $\mathbf{Y}_{\text{half},i}$, $\mathbf{Y}_{\text{single},i}$, and $\mathbf{Y}_{\text{double},i}$.

Step 2. Apply DBSCAN to the output matrices of previous step to cluster most nodes into communities, and the remaining nodes as outliers.

Step 3. Then, we measure clustering performances of DBSCAN on the three different precision subspace iteration embeddings with precision and recall.

Subspace Iteration Results. Figure 5 shows the eigenspace error, $\|\mathbf{Y} - \mathbf{Q}\mathbf{Q}^{\mathsf{T}}\mathbf{Y}\|_2/\|\mathbf{Y}\|_2$, from the subspace iteration step of one trial. The τ values were set to $5u_{\text{single}}$ and $5u_{\text{double}}$ for the uniform precision HQRs and $5u_{\text{half}}$ for the mixed-precision HQR, and the solid lines are plotted to show the unit round-off values. The uniform precision implementations of subspace iterations reached their stopping criterion set by τ, and the mixed-precision implementation fluctuated close to τ but could not reach it. The convergence rate was approximately the same across the three different implementations, which suggests that the lower precision routines (mixed-precision HQR or uniform single precision HQR) can be used as speedy preconditioners to the high precision routine. In addition, if double precision eigenspace error is not necessary to achieve sufficient clustering results, we can simply use the lower precision HQR subspace iterations as full eigensolvers.

Clustering Results. Table 5 shows the worst-case precision and recall results from the 10 trials for each subspace iteration implementation. The DBSCAN algorithm and the calculation of precision and recall were computed in double precision, and the variance in precision and recall values for these 10 trials were in the range of 1e-6. The results show indiscernible difference in the precision and recall, which suggests that a lower precision HQR within subspace iteration can still lead to a sufficiently accurate clustering. Possible future works include trying different clustering methods for Step 2.

HQR scheme	Precision	Recall
mixed-precision	0.9822	0.9393
single-precision	0.9817	0.9407
double-precision	0.9822	0.9405

Table 5: Worst-case precision and recall values for 10 trials of DBSCAN on graph with 5000 nodes and 19 true clusters.

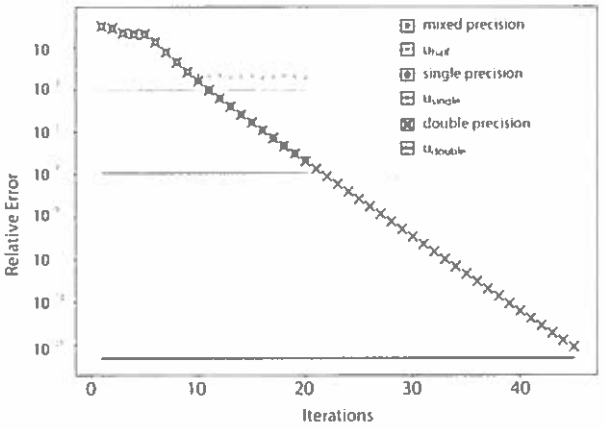

Figure 5: Eigenspace Error for subspace iteration with using doubles, singles, and half- precision traditional Householder QR factorizations.

6 Conclusion

Development of GPUs that support low precision floating point arithmetic has led to interest in half precision and mixed-precision algorithms that demonstrate speedier times, lower energy consumption, and lower memory usage. Loss in precision, stability, and representable range compensate for those advantages, although these shortcomings may have little to no impact in some applications. It may even be possible to navigate around those drawbacks with algorithmic design. In addition, the existing rounding error analysis cannot accurately bound the behavior of mixed-precision arithmetic.

We have developed a new framework for mixed-precision rounding error analysis and applied it to HQR, a widely used linear algebra routine, and implemented it in an iterative eigensolver in the context of spectral clustering. The mixed-precision error analysis builds from the inner product routine, which can be applied to many other linear algebra tools as well. The new error bounds more accurately describe how rounding errors are accumulated in mixed-precision settings. We also found that TSQR, a communication-avoiding, easily parallelizable QR factorization algorithm for tall-and-skinny matrices, can outperform HQR in mixed-precision settings for ill-conditioned, extremely overdetermined cases, which suggests that some algorithms are more robust against lower precision arithmetic. As QR factorizations of tall-and-skinny matrices are common in spectral clustering,

we experimented with introducing mixed-precision settings into graph partitioning problems. In particular, we applied DBSCAN to the spectral basis of a graph identified via subspace iteration that used our simulated mixed-precision HQR, which yielded clustering results tantamount to results from employing double-precision entirely.

Further works are needed to test larger, more ill-conditioned problems with different mixed-precision settings, and to explore other divide-and-conquer methods like TSQR that can harness parallel capabilities of GPUs while withstanding lower precisions.

Give a few more sentances of why your work changes the way we should think about stability.

A Proofs

A.1 Lemma 2.3 (Equation 19)

Proof. We wish to round up to the lower precision, p, since $1 \gg u_p \gg u_s$. Recall that $d = \lfloor k_s u_s / u_p \rfloor$ and $r \leq \lfloor u_p / u_s \rfloor$, and note

$$k_p u_p + k_s u_s = (k_p + d)u_p + r u_s \leq (k_p + d + 1)u_p$$

$$\begin{aligned}
\gamma_p^{(k_p)} + \gamma_s^{(k_s)} + \gamma_p^{(k_p)}\gamma_s^{(k_s)} &= \frac{k_p u_p}{1 - k_p u_p} + \frac{k_s u_s}{1 - k_s u_s} + \frac{k_p u_p}{1 - k_p u_p}\frac{k_s u_s}{1 - k_s u_s} \\
&= \frac{k_p u_p + k_s u_s - k_p k_s u_p u_s}{1 - (k_p u_p + k_s u_s) + k_p k_s u_p u_s} \\
&\leq \frac{(k_p + d + 1)u_p - k_p k_s u_p u_s}{1 - (k_p + d + 1)u_p + k_p k_s u_p u_s} \\
&< \frac{(k_p + d + 1)u_p}{1 - (k_p + d + 1)u_p} = \gamma_p^{(k_p + d + 1)}
\end{aligned}$$

□

A.2 Inner Products

A.2.1 Lemma 2.4

Let δ_p and δ_s be rounding error incurred from products and summations, and are bounded by: $|\delta_p| < u_p$ and $|\delta_s| < u$, following the notation in [3]. Let s_k denote the k^{th} partial sum, and let $\hat{s}_k$ denote the floating point representation of the calculated s_k

$$\begin{aligned}
\hat{s}_1 &= \text{fl}(\mathbf{x}_1\mathbf{y}_1) = \mathbf{x}_1\mathbf{y}_1(1 + \delta_{p,1}) \\
\hat{s}_2 &= \text{fl}(\hat{s}_1 + \mathbf{x}_2\mathbf{y}_2) \\
&= (\mathbf{x}_1\mathbf{y}_1(1 + \delta_{p,1}) + \mathbf{x}_2\mathbf{y}_2(1 + \delta_{p,2}))(1 + \delta_{s,1}) \\
\hat{s}_3 &= \text{fl}(\hat{s}_2 + \mathbf{x}_3\mathbf{y}_3) \\
&= ((\mathbf{x}_1\mathbf{y}_1(1 + \delta_{p,1}) + \mathbf{x}_2\mathbf{y}_2(1 + \delta_{p,2}))(1 + \delta_{s,1}) + \mathbf{x}_3\mathbf{y}_3(1 + \delta_{p,3}))(1 + \delta_{s,2})
\end{aligned}$$

We can see a pattern emerging. The error for a general length m vector dot product is then:

$$\hat{s}_m = (\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2)(1 + \delta_{p,1})\prod_{k=1}^{m-1}(1 + \delta_{s,k}) + \sum_{i=3}^{n}\mathbf{x}_i\mathbf{y}_i(1 + \delta_{p,i})\left(\prod_{k=i-1}^{m-1}(1 + \delta_{s,k})\right). \tag{54}$$

where each occurrence of δ_p and δ_s are distinct, but are still bound by u_p and u_s.

Using Lemma 2.1, we further simplify:

$$\begin{aligned}
\text{fl}(\mathbf{x}^\top\mathbf{y}) = \hat{s}_m &= (1 + \theta_p^{(1)})(1 + \theta_s^{(m-1)})\mathbf{x}^\top\mathbf{y} \\
&= (\mathbf{x} + \Delta\mathbf{x})^\top\mathbf{y} = \mathbf{x}^\top(\mathbf{y} + \Delta\mathbf{y})
\end{aligned}$$

Here $\Delta\mathbf{x}$ and $\Delta\mathbf{y}$ are vector perturbations.

By using Lemma 2.3 equation 19, we can bound the perturbations componentwise. Let $d = \lfloor \frac{(m-1)u_s}{u_p} \rfloor$ such that $(m-1)u_s = du_p + ru_s$.

$$\begin{aligned}
|\Delta\mathbf{x}| &\leq \gamma_p^{(d+2)}|\mathbf{x}| \\
|\Delta\mathbf{y}| &\leq \gamma_p^{(d+2)}|\mathbf{y}|
\end{aligned}$$

Furthermore, these bounds lead to a forward error result as shown in Equation 55 .

$$|\mathbf{x}^\top\mathbf{y} - \text{fl}(\mathbf{x}^\top\mathbf{y})| \leq \gamma_p^{(d+2)}|\mathbf{x}|^\top|\mathbf{y}| \tag{55}$$

A.2.2 Corollary 2.5

This proof follows similarly to the proof for Lemma 2.4. Since no error is incurred in the multiplication portion of the inner products, δ_i and $\delta_{s,i}$ are rounding error incurred from summations and storage. The error for a general m-length vector dot product is then,

$$\hat{s}_m = (\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2)\prod_{k=1}^{m-1}(1 + \delta_{s,k}) + \sum_{i=3}^{n}\mathbf{x}_i\mathbf{y}_i\left(\prod_{k=i-1}^{m-1}(1 + \delta_{s,k})\right). \tag{56}$$

Using Lemma 2.1, we further simplify:

$$\begin{aligned}
\text{fl}(\mathbf{x}^\top\mathbf{y}) = \hat{s}_m &= (1 + \theta_s^{(m-1)})\mathbf{x}^\top\mathbf{y} \\
&= (\mathbf{x} + \Delta\mathbf{x})^\top\mathbf{y} = \mathbf{x}^\top(\mathbf{y} + \Delta\mathbf{y})
\end{aligned}$$

Here $\Delta\mathbf{x}$ and $\Delta\mathbf{y}$ are vector perturbations.

By using Lemma 2.3 equation 19, we can bound the perturbations componentwise. Let $d = \lfloor \frac{(m-1)u_s}{u_p} \rfloor$ such that $(m-1)u_s = du_p + ru_s$.

$$\begin{aligned}
|\Delta\mathbf{x}| &\leq \gamma_p^{(d+1)}|\mathbf{x}| \\
|\Delta\mathbf{y}| &\leq \gamma_p^{(d+1)}|\mathbf{y}|
\end{aligned}$$

Furthermore, these bounds lead to a forward error result as shown in Equation 57 .

$$|\mathbf{x}^\top\mathbf{y} - \text{fl}(\mathbf{x}^\top\mathbf{y})| \leq \gamma_p^{(d+1)}|\mathbf{x}|^\top|\mathbf{y}| \tag{57}$$

A.3 Proof for Mixed-Precision HQR result

Here, we show a few results that are necessary for the proof for Theorem 3.5. Lemma A.1 shows normwise results for a single mixed-precision Householder transformation performed on a vector, and Lemma A.2 builds on Lemma A.1 to show normwise results for multiple mixed-precision Householder transformations on a vector. We build column-wise results for HQR based on these lemmas, then compute the matrix norms at the end.

Lemma A.1. *Let $\mathbf{x} \in \mathbb{R}^m$ and consider the computation of $\mathbf{y} = \mathbf{P}_c\mathbf{x} = \mathbf{x} - \beta\hat{\mathbf{v}}\hat{\mathbf{v}}^\top\mathbf{x}$, where $\hat{\mathbf{v}}$ has accumulated error shown in Lemma 3.4. Then, the computed $\hat{\mathbf{y}}$ satisfies*

$$\hat{\mathbf{y}} = (\mathbf{P} + \Delta\mathbf{P})\mathbf{x}, \quad \|\Delta\mathbf{P}\|_F \leq \gamma_w^{(6d+6z+13)}, \tag{58}$$

where $\mathbf{P} = \mathbf{I} - \beta\mathbf{v}\mathbf{v}^\top$ is a Householder transformation.

Proof. Recall that the computed $\hat{\mathbf{y}}$ accumulates componentwise error shown in Equation 32. Even though we don't explicitly form $\mathbf{P}$, forming the normwise error bound for this matrix makes the analysis simple. First, recall that any matrix $\mathbf{A}$ with rank r has the following relations between its 2-norm and Frobenius norm,

$$\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F \leq \sqrt{r}\|\mathbf{A}\|_2. \tag{59}$$

Then, we have

$$\|\mathbf{y}\|_2 = \|\mathbf{P}\mathbf{x}\|_2 \leq \|\mathbf{P}\|_2\|\mathbf{x}\|_2 = \|\mathbf{x}\|_2 \tag{60}$$

since $\mathbf{P}$ is orthogonal and $\|\mathbf{P}\|_2 = 1$. We now transition from the componentwise error to normwise error for $\Delta\mathbf{y}$, and write $\tilde{z} = 6d + 6z + 13$.

$$\|\Delta\mathbf{y}\|_2 = \left(\sum_{i=1}^m \Delta\mathbf{y}_i^2\right)^{1/2} \leq \gamma_w^{(\tilde{z})}\left(\sum_{i=1}^m \mathbf{y}_i^2\right)^{1/2} = \gamma_w^{(\tilde{z})}\|\mathbf{y}\|_2 \tag{61}$$

Combining Equations 60 and 61, we result in

$$\frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{x}\|_2} \leq \gamma_w^{(\tilde{z})}. \tag{62}$$

Now, notice that $\Delta\mathbf{P}$ is exactly $\frac{1}{\mathbf{x}^\top\mathbf{x}}\Delta\mathbf{y}\mathbf{x}^\top$.

$$\begin{aligned}(\mathbf{P} + \Delta\mathbf{P})\mathbf{x} &= (\mathbf{P} + \frac{1}{\mathbf{x}^\top\mathbf{x}}\Delta\mathbf{y}\mathbf{x}^\top)\mathbf{x} \\ &= \mathbf{P}\mathbf{x} + \frac{\mathbf{x}^\top\mathbf{x}}{\mathbf{x}^\top\mathbf{x}}\Delta\mathbf{y} = \mathbf{y} + \Delta\mathbf{y}\end{aligned}$$

We can compute the Frobenius norm of $\Delta\mathbf{P}$ by using $\Delta\mathbf{P}_{:j} = \frac{1}{\|\mathbf{x}\|_2^2}\Delta\mathbf{y}_i\mathbf{x}_j$.

$$\begin{aligned}\|\Delta\mathbf{P}\|_F &= \left(\sum_{i=1}^m\sum_{j=1}^m \Delta\mathbf{P}_{ij}^2\right)^{1/2} = \left(\sum_{i=1}^m\sum_{j=1}^m\left(\frac{1}{\|\mathbf{x}\|_2^2}\Delta\mathbf{y}_i\mathbf{x}_j\right)^2\right)^{1/2} \\ &= \left(\frac{1}{\|\mathbf{x}\|_2^4}\sum_{i=1}^m\sum_{j=1}^m \Delta\mathbf{y}_i^2\mathbf{x}_j^2\right)^{1/2} = \frac{1}{\|\mathbf{x}\|_2^2}\left(\sum_{i=1}^m \Delta\mathbf{y}_i^2\left(\sum_{j=1}^m \mathbf{x}_j^2\right)\right)^{1/2} \\ &= \frac{1}{\|\mathbf{x}\|_2^2}\left(\|\mathbf{x}\|_2^2\sum_{i=1}^m \Delta\mathbf{y}_i^2\right)^{1/2} = \frac{\|\mathbf{x}\|_2\|\Delta\mathbf{y}\|_2}{\|\mathbf{x}\|_2^2} = \frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{x}\|_2}\end{aligned}$$

Finally, using Equation 62, we result in

$$\|\Delta\mathbf{P}\|_F \leq \gamma_w^{(\tilde{z})}.$$

□

Lemma A.2. *Consider applying a sequence of transformations in the set $\{\mathbf{P}_i\}_{i=1}^r \subset \mathbb{R}^{m\times m}$ to $\mathbf{x} \in \mathbb{R}^m$, where $\mathbf{P}_i$'s are all Householder transformations, and we will assume that $r\gamma_w^{(\tilde{z})} < \frac{1}{2}$.*

$$\mathbf{y} = \mathbf{P}_r\mathbf{P}_{r-1}\cdots\mathbf{P}_1\mathbf{x} = \mathbf{Q}^\top\mathbf{x}$$

Then, $\hat{\mathbf{y}} = (\mathbf{Q} + \Delta\mathbf{Q})^\top\mathbf{x}$, where

$$\|\Delta\mathbf{Q}\|_F \leq r\gamma_w^{(\tilde{z})} \tag{63}$$

$$\|\Delta\mathbf{y}\|_2 \leq r\gamma_w^{(\tilde{z})}\|\mathbf{y}\|_2 \tag{64}$$

In addition, if we let $\hat{\mathbf{y}} = \mathbf{Q}^\top(\mathbf{x} + \Delta\mathbf{x})$, then

$$\|\Delta\mathbf{x}\|_2 \leq r\gamma_w^{(\tilde{z})}\|\mathbf{x}\|_2 \tag{65}$$

Proof. As was for the proof for Lemma A.1, we know $\Delta\mathbf{Q}^\top = \frac{1}{\|\mathbf{x}\|_2^2}\Delta\mathbf{y}\mathbf{x}^\top$. Recall that the HQR factorization applies a series of Householder transformations on $\mathbf{A}$ to form $\mathbf{R}$, and the same series of Householder transformations in reverse order to $\mathbf{I}$ to form $\mathbf{Q}$. Therefore, it is appropriate to assume that $\mathbf{x}$ is exact in this proof, and we form a forward bound on $\mathbf{y}$. However, we can still easily switch between forward and backward errors in the following way.

$$\begin{aligned}\hat{\mathbf{y}} = \mathbf{y} + \Delta\mathbf{y} &= \mathbf{Q}^\top(\mathbf{x} + \Delta\mathbf{x}) \\ &= (\mathbf{Q} + \Delta\mathbf{Q})^\top\mathbf{x} \\ \Delta\mathbf{y} = \Delta\mathbf{Q}^\top\mathbf{x} &= \mathbf{Q}^\top\Delta\mathbf{x}\end{aligned}$$

In addition, we can switch between $\Delta\mathbf{y}$ and $\Delta\mathbf{x}$ by using the fact that $\|\mathbf{Q}\|_2 = 1$.

Error bound for $\|\Delta\mathbf{y}\|_2$: We will first find $\|\Delta\mathbf{Q}\|_2$, where this is NOT the forward error from forming $\mathbf{Q}$ with Householder transformations, but rather a backward error in accumulating Householder transformations. From Lemma A.1, we have $\|\Delta\mathbf{P}\|_F \leq \gamma_w^{(\tilde{z})} = \gamma_w^{(\tilde{z})}\|\mathbf{P}\|_2$ for any Householder transformation $\mathbf{P} \in \mathbb{R}^{m\times m}$, where $\tilde{z} = 6d + 6z + 13$ and $d = \lfloor\frac{(m-1)u}{w_u}\rfloor$, $z \in \{1, 2\}$. Therefore, this applies to the sequence of $\mathbf{P}_i$'s that form $\mathbf{Q}$ as well.

We will now use Lemma 3.7 from [3] to bound $\Delta\mathbf{Q}$.

$$\Delta\mathbf{Q}^\top = \left(\hat{\mathbf{Q}} - \mathbf{Q}\right)^\top = \prod_{i=r}^{1}(\mathbf{P}_i + \Delta\mathbf{P}_i) - \prod_{i=r}^{1}\mathbf{P}_i$$

$$\begin{aligned}\|\Delta\mathbf{Q}\|_F = \|\Delta\mathbf{Q}^\top\|_F &= \left\|\prod_{i=r}^{1}(\mathbf{P}_i + \Delta\mathbf{P}_i) - \prod_{i=r}^{1}\mathbf{P}_i\right\|_F \\ &\leq \left(\prod_{i=r}^{1}(1+\gamma_w^{(\tilde{z})}) - 1\right)\prod_{i=r}^{1}\|\mathbf{P}_i\|_2 = \prod_{i=r}^{1}(1+\gamma_w^{(\tilde{z})}) - 1\end{aligned}$$

The last equality results from the orthogonality of Householder matrices.

Let's take a look at the constant, $(1+\gamma_w^{(\tilde{z})})^r - 1$. From the very last rule in Lemma 2.2, we can generalize the following.

$$(1+\gamma_w^{(\tilde{z})})^r \leq (1+\gamma_w^{(\tilde{z})})^{r-2}(1+\gamma_w^{(\tilde{z})})(1+\gamma_w^{(\tilde{z})}) \tag{66}$$

$$\leq (1+\gamma_w^{(\tilde{z})})^{r-2}(1+\gamma_w^{(2\tilde{z})}) \leq \cdots \leq (1+\gamma_w^{(r\tilde{z})}) \tag{67}$$

So our quantity of interest can be bound by $(1+\tilde{\gamma}_{\omega}^{(\tilde{z})})^r - 1 \leq \tilde{\gamma}_{\omega}^{(r\tilde{z})}$.

Now we will use the following equivalent algebraic inequalities to get the final result

$$0 < a < b < 1 \Leftrightarrow 1-a > 1-b \Leftrightarrow \frac{1}{1-a} < \frac{1}{1-b} \Leftrightarrow \frac{a}{1-a} < \frac{b}{1-b} \tag{68}$$

In addition, we assume $r\tilde{\gamma}_{\omega}^{(\tilde{z})} < \frac{1}{2}$.

$$\begin{aligned}
(1+\tilde{\gamma}_{\omega}^{(\tilde{z})})^r - 1 \leq \tilde{\gamma}_{\omega}^{(r\tilde{z})} &= \frac{r\tilde{z}u_{\omega}}{1-r\tilde{z}u_{\omega}}, \text{ by definition.} \\
&\leq \frac{r\tilde{\gamma}_{\omega}^{(\tilde{z})}}{1-r\tilde{\gamma}_{\omega}^{(\tilde{z})}}, \text{ since } r\tilde{z}u_{\omega} < r\tilde{\gamma}_{\omega}^{(\tilde{z})}, \text{ by Equation 68} \\
&\leq 2r\tilde{\gamma}_{\omega}^{(\tilde{z})}, \text{ since } r\tilde{\gamma}_{\omega}^{(\tilde{z})} < \frac{1}{2} \text{ implies } \frac{1}{1-r\tilde{\gamma}_{\omega}^{(\tilde{z})}} < 2. \\
&= r\tilde{\gamma}_{\omega}^{(\tilde{z})}.
\end{aligned}$$

where $\gamma^{(cu)} := \frac{cu}{1-cu}$ for some small integer, c. If we had started with $(1+\tilde{\gamma}_{\omega}^{(\tilde{z})})^r - 1$, we can still result in $r\tilde{\gamma}_{\omega}^{(\tilde{z})}$ assuming that $2c$ is still a small integer. In conclusion, we have

$$(1+\tilde{\gamma}_{\omega}^{(\tilde{z})})^r - 1 \leq r\tilde{\gamma}_{\omega}^{(\tilde{z})},$$

which results in the bound for $\Delta\mathbf{Q}$ as shown in Equation 63.

$$\|\Delta\mathbf{Q}\|_2 \leq \|\Delta\mathbf{Q}\|_F \leq r\tilde{\gamma}_{\omega}^{(\tilde{z})}.$$

Now, we can get the norm bound for $\Delta\mathbf{y}$.

$$\|\Delta\mathbf{y}\|_2 = \|\Delta\mathbf{Q}\mathbf{x}\|_2 \leq \|\Delta\mathbf{Q}\|_2\|\mathbf{x}\|_2 \leq r\tilde{\gamma}_{\omega}^{(\tilde{z})}\|\mathbf{x}\|_2 \tag{69}$$

Bound for $\|\Delta\mathbf{x}\|_2$: We use the above result.

$$\|\Delta\mathbf{x}\|_2 = \|\mathbf{Q}\Delta\mathbf{y}\|_2\| \leq \|\mathbf{Q}\|_2\|\Delta\mathbf{y}\|_2 = \|\Delta\mathbf{y}\|_2 \leq r\tilde{\gamma}_{\omega}^{(\tilde{z})}\|\mathbf{x}\|_2 \tag{70}$$

While $r\gamma^{(k)} = r\frac{ku}{1-ku} < \frac{rku}{1-rku} = \gamma^{(rk)}$ holds true when $r > 0$ and $rku < 1$ are satisfied, the strict inequality implies that $r\gamma^{(k)}$ is a tighter bound than $\gamma^{(rk)}$. However, $\gamma^{(rk)}$ is easier to work with using the rules in Lemma 2.1. □

A.3.1 Proof for Theorem 3.5

First, we use Lemma A.2 directly on columns of $\mathbf{A}$ and $\mathbf{I}_{m\times n}$ to get a result for columns of $\hat{\mathbf{R}}$ and $\hat{\mathbf{Q}}$. Figure 1 shows that each elements of $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$ each go through different numbers of Householder transformations and Householder transformations for different lengths of vectors as well. We will use the maximum number of transformations and the length of the longest vector we perform a Householder transformation onto — that is n transformations of vectors of length m. For j in $\{1, \cdots, n\}$, the j^{th} column of $\hat{\mathbf{R}}$ and $\hat{\mathbf{Q}}$ are the results of j Householder transformations on $\mathbf{A}$ and $\mathbf{I}$.

$$\|\Delta\mathbf{Q}[:,j]\|_2 \leq j\tilde{\gamma}_{\omega}^{(\tilde{z})}\|\hat{c}_j\|_2 < \tilde{\gamma}_{\omega}^{(\tilde{z})} \tag{71}$$

$$\|\Delta\mathbf{R}[:,j]\|_2 \leq j\tilde{\gamma}_{\omega}^{(\tilde{z})}\|\mathbf{A}[:,j]\|_2 < \tilde{\gamma}_{\omega}^{(\tilde{z})}\|\mathbf{A}[:,j]\|_2 \tag{72}$$

Finally, we relate columnwise 2-norms to matrix Frobenius norms. It is straightforward to see the result for the $\mathbf{Q}$ factor.

$$\|\Delta\mathbf{Q}\|_F = \left(\sum_{j=1}^{n}\|\Delta\mathbf{Q}[:,j]\|_2^2\right)^{1/2} \leq \left(\sum_{j=1}^{n}(j\tilde{\gamma}_{\omega}^{(\tilde{z})})^2\|\hat{c}_j\|_2^2\right)^{1/2} \leq n^{1/2}\tilde{\gamma}_{\omega}^{(\tilde{z})} \tag{73}$$

Note that we bound $\sum_{j=1}^{n} j^2$ by n^3, but the summation is actually exactly $\frac{n(n+1)(2n+1)}{6}$. Therefore, a tighter bound would replace $n^{1/2}$ with $\left(\frac{n(n+1)(2n+1)}{6}\right)^{1/2}$.

We can bound the $\mathbf{R}$ factor in a similar way.

$$\|\Delta\mathbf{R}\|_F = \left(\sum_{j=1}^{n}\|\Delta\mathbf{R}[:,j]\|_2^2\right)^{1/2} \leq \left(\sum_{j=1}^{n}(j\tilde{\gamma}_{\omega}^{(\tilde{z})})^2\|\mathbf{A}[:,j]\|_2^2\right)^{1/2} \leq n\tilde{\gamma}_{\omega}^{(\tilde{z})}\|\mathbf{A}\|_F \tag{74}$$

Now let's get the backward error from the HQR factorization.

$$\begin{aligned}
\Delta\mathbf{A} = \mathbf{A} - \hat{\mathbf{Q}}\hat{\mathbf{R}} &= \mathbf{A} - \mathbf{Q}\hat{\mathbf{R}} + \mathbf{Q}\hat{\mathbf{R}} - \hat{\mathbf{Q}}\hat{\mathbf{R}} \\
&= \mathbf{Q}(\mathbf{R} - \hat{\mathbf{R}}) + (\mathbf{Q} - \hat{\mathbf{Q}})\mathbf{R} = \mathbf{Q}\Delta\mathbf{R} + \Delta\mathbf{Q}\mathbf{R}
\end{aligned}$$

A columnwise result for $\mathbf{A}$ is shown by:

$$\begin{aligned}
\|\Delta\mathbf{A}[:,j]\|_2 &= \|(\mathbf{Q}\Delta\mathbf{R} + \Delta\mathbf{Q}\hat{\mathbf{R}})[:,j]\|_2 \\
&\leq \|\mathbf{Q}\Delta\mathbf{R}[:,j]\|_2 + \|\Delta\mathbf{Q}\hat{\mathbf{R}}[:,j]\|_2 \\
&\leq \|\Delta\mathbf{R}[:,j]\|_2 + \|\Delta\mathbf{Q}\|_2\|\hat{\mathbf{R}}[:,j]\|_2 \\
&\leq \|\Delta\mathbf{R}[:,j]\|_2 + \|\Delta\mathbf{Q}\|_F\|(\mathbf{R}+\Delta\mathbf{R})[:,j]\|_2 \\
&\leq j\tilde{\gamma}_{\omega}^{(\tilde{z})}\|\mathbf{A}[:,j]\|_2 + n^{3/2}\tilde{\gamma}_{\omega}^{(\tilde{z})}\|(\mathbf{Q}^{\top}\mathbf{A} + \Delta\mathbf{R})[:,j]\|_2 \\
&\leq j\tilde{\gamma}_{\omega}^{(\tilde{z})}\|\mathbf{A}[:,j]\|_2 + n^{3/2}\tilde{\gamma}_{\omega}^{(\tilde{z})}(\|\mathbf{A}[:,j]\|_2 + \|\Delta\mathbf{R}[:,j]\|_2) \\
&\leq \left(j\tilde{\gamma}_{\omega}^{(\tilde{z})} + n^{3/2}\tilde{\gamma}_{\omega}^{(\tilde{z})}(1 + j\tilde{\gamma}_{\omega}^{(\tilde{z})})\right)\|\mathbf{A}[:,j]\|_2 \\
&= n^{3/2}\tilde{\gamma}_{\omega}^{(\tilde{z})}\|\mathbf{A}[:,j]\|_2,
\end{aligned}$$

where we assume $n\tilde{\gamma}_{\omega}^{(\tilde{z})} \ll 1$, and the last equality sweeps all non-leading order terms under the arbitrary constant c within the definition of $\tilde{\gamma}$.

$$\|\Delta\mathbf{A}\|_F = \left(\sum_{j=1}^{n}\|\Delta\mathbf{A}[:,j]\|_2^2\right)^{1/2} \leq \left(\sum_{j=1}^{n}(n^{3/2}\tilde{\gamma}_{\omega}^{(\tilde{z})})^2\|\mathbf{A}[:,j]\|_2^2\right)^{1/2} \leq n^{3/2}\tilde{\gamma}_{\omega}^{(\tilde{z})}\|\mathbf{A}\|_F \tag{75}$$

References

[1] M. Courbariaux, Y. Bengio, and J.-P. David. "Training deep neural networks with low precision multiplications." *arXiv preprint arXiv:1412.7024*, 2014.

[2] M. Courbariaux, J.-P. David, and Y. Bengio, "Low precision storage for deep learning," *arXiv preprint arXiv:1412.7024*, 2014.

[3] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2nd ed., 2002.

[4] A. S. Householder, "Unitary triangularization of a nonsymmetric matrix," *Journal of the ACM (JACM)*, vol. 5, no. 4, pp. 339–342, 1958.

[5] D. Mori, Y. Yamamoto, and S.-L. Zhang, "Backward error analysis of the allreduce algorithm for householder qr decomposition," *Japan Journal of Industrial and Applied Mathematics*, vol. 29, pp. 111–130, Feb 2012.

[6] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential qr and lu factorizations," *SIAM Journal on Scientific Computing*, vol. 34, no. 1, pp. A206–A239, 2012.

[7] G. Ballard, J. W. Demmel, L. Grigori, M. Jacquelin, H. Diep Nguyen, and E. Solomonik, "Reconstructing householder vectors from tall-skinny qr," vol. 85, pp. 1159–1170, 05 2014.

[8] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and van der Vorst H., *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1st ed., 2000.

[9] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

[10] E. Kao, V. Gadepally, M. Hurley, M. Jones, J. Kepner, S. Mohindra, P. Monticciolo, A. Reuther, S. Samsi, W. Song, D. Staheli, and S. Smith, "Streaming graph challenge: Stochastic block partition," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–12, Sep. 2017.