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

L. Minah Yang, Alyson Fox, and Geoffrey Sanders

October 2018

Contents

1 Introduction

- Higham [???] [1]
- TSQR [???] [2], TSQR Analysis [???]
- GPU Refs [???] TPUs [???]
- spectral clustering [???]

Development of new GPUs that perform half precision arithmetic up to 16 times faster than double precision arithmetic motivates use of low precision computations whenever possible. Although IEEE 754 32-bit and 64-bit (single and double) precision floating point numbers have dominated scientific computing since 1985, there were many other number representation systems that preceded the current standards. Some of these systems used in the past include 24-bit binary floating point representation (Z3/Zuse), 32-bit decimal floating(Z4/Zuse), decimal arithmetic with conversions from binary storage (Mark V/ Bell Labs), fixed point arithmetic (Pilot ACE/NPL), etc... Following difficulties the original Pilot ACE faced with going out of range when using fixed point arithmetic, James H. Wilkinson studied rounding error analysis of floating point numbers in detail and published [3].

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 round-off errors. Although round-off error is only one of many sources of error in numerical algorithms, it naturally has a larger influence when using low precision. Therefore, 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 would either be too optimistic or pessimistic since they only allow uniform floating point precision type for all arithmetic operations and storage. 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. Since communication-avoiding, parallelizable QR algorithms already exist for tall-and-skinny matrices, we study how those algorithms behave in half-precision. We simulate half-precision arithmetic in our experiments in various ways that include conversions into single precision for computation and half precision for storage. While the standard Householder QR factorization algorithms are highly unstable in half-precision, our numerical simulations show that simulated mixed-precision implementation outperforms the pessimistic error bound and the Tall-and-Skinny QR (TSQR) algorithm often reduces the backward error of QR factorization. These results motivate detailed numerical

Why did GPU's become necessary? why would they be faster in low precision than CPU's why should we care?

This isn't the point. Different representations are cool but the real point is that we need to useless bits but we want the same accuracy or ~~close~~ close to the same solution as when we use ~~double~~ precision.

Why!! Explain.

What are people doing to battle this? What do the different mixed precision applications

Why QR? what may be lose or gain by doing QR in half? what applications may want to sacrifice orthogonality for faster computation and why studying the error is so important?

Talk about the history more why do we need a more flexible error analysis?

Belong is numerical section

Block was never discussed before hand

Doesn't belong here, move later in intro

* most of this paragraph belong in conclusion. we want to answer the questions of why we are studying this & what the impact would be if we show what we want to show.

analysis of half precision block QR factorization both for the purposes of replacing higher-precision QR (in applications less sensitive to error) and using the half precision versions to produce warm starts that initialize higher precision QR factorization.

We incorporated mixed-precision QR factorization into two applications: spectral clustering and sparse regression in the context of discovery of equations. When using subspace iteration for graph clustering applications, half precision accuracy in forming the eigenspace is sufficient for clustering with high precision and recall for some small-scale benchmark problems. Similarly, single precision accuracy in data-driven discovery of a simple system of ODEs is comparable to results from using double precision, and may even be more robust in noisy systems.

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, we consider performing the so-called *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, $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}_{m\times m}$, 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 Householder QR 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

Table 1 summarizes notation that appears in this paper. While important definitions are stated explicitly in the text, and this table serves to establish

2 Floating Point Numbers and Error Analysis Tools

2.1 Modern GPU Hardware

We will be using floating point operation error analysis tools developed in [1], as it allows a relatively simple framework for formulating error bounds for complex linear algebra operations. Let $\mathbb{F} \subset \mathbb{R}$ denote the space of some floating point number system with base b, precision t, significand/mantissa μ, and exponent range $[\eta_{\min}, \eta_{\max}]$. Then every element y in $\mathbb{F}$ can be written as

$$y = \pm\mu \times b^{\eta-t}, \tag{1}$$

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
$\mathrm{fl}(\mathbf{x})$;$\hat{\mathbf{x}}$	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 q: $\frac{1}{2}b^{1-t}$	2
δ_q	Quantity bounded by: $\lvert\delta_q\rvert < u_q$	2
$\gamma_q^{(k)}$	$\frac{ku_q}{1-ku_q}$	2
$\theta_q^{(k)}$	Quantity bounded by: $\lvert\theta_q^{(k)}\rvert \leq \gamma_q^{(k)}$	2
$\mathbf{x}$	Vectors	2
$\mathbf{A}$	Matrix	1
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
$\mathbf{A}[a:b, c:d]$	Rows a to b and columns c to d of matrix $\mathbf{A}$	3.1
$\mathbf{A}[a:b,:]$, $\mathbf{A}[:, c:d]$	Rows a to b, columns c to d of matrix $\mathbf{A}$	3.1
$\hat{e}_i$	Cardinal vector	3
$\mathbf{0}_{m\times n}$	m-by-n zero matrix	1
$\mathbf{I}_n$	n-by-n identity matrix	3.1
$\mathbf{I}_{m\times n}$	$\begin{bmatrix}\mathbf{I}_n \\ \mathbf{0}_{m-n\times n}\end{bmatrix}$	
$\mathbf{P}_v$	Householder transformation: $\mathbf{I} - \beta\mathbf{v}\mathbf{v}^\top$	3.1
$\mathbf{P}_i$	i^{th} Householder transformation in the Householder QR algorithm	3.1
u_s, u_p, u_w	Unit round-off for sum, product, and storage (write)	2.2
$\gamma_{p,q}^{(k_p,k_q)}$	$(1+\gamma_p^{(k_p)})(1+\gamma_q^{(k_q)}) - 1$	2.2
$\theta_{p,q}^{(k_p,k_q)}$	Quantity bounded by: $\lvert\theta_{p,q}^{(k_p,k_q)}\rvert < \gamma_{p,q}^{(k_p,k_q)}$	2.2

Table 1: Basic definitions

? I thought it was traditionally a fraction.

where μ is any integer in $[0, b^t - 1]$, and η is an integer in $[\eta_{min}, \eta_{max}]$. While 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 using these error analysis tools in [1]. developed

A short analysis of floating point operations (cf. Theorem 2.2 [1]) shows that the relative error is controlled by the unit round-off, $u := \frac{1}{2}b^{1-t}$. Table 2 shows IEEE precision types described by the same parameters as in Equation 1. Let op be any basic operation between 2 floating point numbers from the set $\mathrm{OP} = \{+, -, \times \div\}$. The true value $(x \text{ op } y)$ lies in $\mathbb{R}$ and it is rounded to the

$x, y \in \mathbb{R}$

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.

"is this the case for all representation or just IEEE?

nearest floating point number, $\mathrm{fl}(x \text{ op } y)$, admitting a rounding error. Then, a single basic floating point operation yields a relative error, δ, bounded in the following sense,

$$\mathrm{fl}(x \text{ op } y) = (1+\delta)(x \text{ op } y), \quad |\delta| \le u, \quad \text{op} \in \{+, -, \times, \div\} \tag{2}$$

This is an assumption for the error model.

We use Equation 2 as a building block in accumulating errors from successive floating point operations in product form. Lemma 2.1 introduces convenient bounds that simplify round-off error analyses.

dot product? confusing sentance.

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

There's not real lead up to this lemma. why is it useful? where did it come from?

$$\prod_{i=1}^{k} (1+\delta_i)^{\rho_i} = 1 + \theta^{(k)} \tag{3}$$

where

$$|\theta^{(k)}| \le \frac{ku}{1-ku} =: \gamma^{(k)}. \tag{4}$$

In other words, $\theta^{(k)}$ represents the accumulation of k successive round-off errors, and it is bounded by $\gamma^{(k)}$. As this notation is often used to formulate upper bounds for relative error, requiring $\gamma^{(k)} < 1$ ensures that the error bound is meaningful. While the assumption $\gamma^{(k)} < 1$ is easily satisfied by fairly large k in higher precision floating point numbers, it is a problem 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% ($\gamma^{(k)} < 1$).

in what sense?

lost the reader. would help to give an example.

typical

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

Table 3: Upper limits of validity in the $\gamma^{(k)}$ notation.

it doesn't invalidate γ it just >1.

This reflects on two sources of difficulty: 1) Successive operations in lower precision types grow unstable more quickly, and 2) the upper bound given by $\gamma^{(k)}$ becomes suboptimal faster in low precision.

doesn't Δ & $\delta 2$ say the same thing. However they different? Explain to the reader why we need to consider both.

For example, the forward error bound for dot products is stated in Equation 5, as presented in [1].

$$\frac{|\mathbf{x}^\top \mathbf{y} - \mathrm{fl}(\mathbf{x}^\top \mathbf{y})|}{|\mathbf{x}|^\top |\mathbf{y}|} \le \gamma^{(m)}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^m \tag{5}$$

need to define.

Since $\gamma_{\text{half}}^{(512)} = 1$, the worst-case relative error bound for a dot product of vectors of length 512 is already at 100%.

A simple numerical experiment 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 50,000 random half-precision vectors of length 512 from two random distributions: the standard normal distribution and the uniform distribution over $(0, 1)$. Half-precision arithmetic was simulated by: 1) casting up to single-precision, 2) computing single-precision addition or multiplication, and 3) casting back down to half-precision, at every single operation. The error calculated is formulated as the left hand side of the inequality in Equation 5, and double precision arithmetic is used for calculating the error as well as the true solution. Table 4 shows the results from computing the relative error for dot products of 512-length half-precision vectors using half-precision arithmetic.

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. Sample sizes were 2e6 for each type of random distribution, and recall that the unit round-off for half-precision is 4.883e-4.

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, whereas the vectors sampled from the uniform distribution 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. While the probabilistic error bound does get the correct order of magnitude for a maximal error (with probability 99%), it is not enough to describe the probability distribution of inner product errors.

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 the magnitudes of rounding errors to expect from computations done on emerging hardware such as GPUs that employ mixed-precision arithmetic. In this paper, we develop a mixed-precision error analysis that allows multiple precision types that is based on the rounding error analysis framework established in [1]. This mixed-precision error analysis relies on the framework given by Lemma 2.1, which best allows us to keep a simple analysis.

In Lemma 2.2, we present modified versions of relations in Lemma 3.3 from [1]. These relations allow us to easily deal with accumulated errors, and aid in writing clear and simpler error analyses. The modifications support multiple precision types, whereas Lemma 3.3 in [1] 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 2.2. *For any nonnegative integer k and some precision q, let $\theta_q^{(k)}$ denote a quantity bounded according to $|\theta_q^{(k)}| \leq \frac{ku_q}{1-ku_q} =: \gamma_q^{(k)}$. 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 $c > 0$.

$$(1+\theta_p^{(k_p)})(1+\theta_p^{(j_p)})(1+\theta_s^{(k_s)})(1+\theta_s^{(j_s)}) = (1+\theta_p^{(k_p+j_p)})(1+\theta_s^{(k_s+j_s)}) \tag{6}$$

$$\frac{(1+\theta_p^{(k_p)})(1+\theta_s^{(k_s)})}{(1+\theta_p^{(j_p)})(1+\theta_s^{(j_s)})} = \begin{cases} (1+\theta_s^{(k_s+j_s)})(1+\theta_p^{(k_p+j_p)}), & j_s \le k_s, j_p \le k_p \\ (1+\theta_s^{(k_s+2j_s)})(1+\theta_p^{(k_p+j_p)}), & j_s \le k_s, j_p > k_p \\ (1+\theta_s^{(k_s+j_s)})(1+\theta_p^{(k_p+2j_p)}), & j_s > k_s, j_p \le k_p \\ (1+\theta_s^{(k_s+2j_s)})(1+\theta_p^{(k_p+2j_p)}), & j_s > k_s, j_p > k_p \end{cases} \tag{7}$$

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 $k_s u_s = d u_p + r u_s$. Alternatively, d can be defined by $d := \lfloor \frac{k_s u_s}{u_p} \rfloor$.

$$\gamma_s^{(k_s)}\gamma_p^{(k_p)} \le \gamma_p^{(k_p)}, \quad \text{for } k_p u_p \le \frac{1}{2} \tag{8}$$

$$\gamma_s^{(k_s)} + u_p \le \gamma_p^{(d+2)} \tag{9}$$

$$\gamma_p^{(k_p)} + u_s \le \gamma_p^{(k_p+1)} \quad \textit{(A loose bound)} \tag{10}$$

$$\gamma_p^{(k_p)} + \gamma_s^{(k_s)} + \gamma_p^{(k_p)}\gamma_s^{(k_s)} < \gamma_p^{(k_p+d+1)} \tag{11}$$

A proof for Equation 11 is shown in Appendix A.

2.2 Inner product Mixed-Precision error

As seen in Section 3, the inner product is a building block of the Householder QR method. 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 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, this 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 error analysis for mixed-precision algorithms to better understand them.

Lemmas 2.3 and 2.4 present two mixed-precision forward error bounds for inner products, which show a tighter bound than the existing error bounds. Both of them show a dependence of quantity d, which is defined to be proportional to the ratio between the precision types used in storage (u_w) and sums (u_s).

Lemma 2.3. *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,*

$$\mathrm{fl}(\mathbf{x}^\top \mathbf{y}) = (\mathbf{x} + \Delta\mathbf{x})\mathbf{y} = \mathbf{x}(\mathbf{y} + \Delta\mathbf{y}), \tag{12}$$

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

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

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

Lemma 2.4 presents another mixed-precision forward error bound for inner products with additional constraints to the types of different precisions being used for different operations. 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.

Lemma 2.4. *Let* w *and* s *each represent floating point precisions for storage and summation, where the unit round-off values for each precision are denoted by* u_w *and* u_s*. Furthermore, assume* $1 \gg u_w \gg u_s > 0$*, and that for any two arbitrary numbers* x *and* y *in* $\mathbb{F}_w$*, their product* xy *is in* $\mathbb{F}_s$*. Let* $\mathbf{x}, \mathbf{y} \in \mathbb{F}_w^n$ *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,*

$$\mathrm{fl}(\mathbf{x}^\top \mathbf{y}) = (\mathbf{x} + \Delta\mathbf{x})\mathbf{y} = \mathbf{x}(\mathbf{y} + \Delta\mathbf{y}), \tag{13}$$

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

Proofs for Lemmas 2.3 and 2.4 are shown in Appendix A. The analyses for these two lemmas differ only in the type of mixed-precision arithmetic performed within the inner product subroutine, and the difference is revealed to result in $\gamma_w^{(d+1)}$ vs $\gamma_w^{(d+2)}$. For the rest of this paper, we will refer to the forward error bound for the inner product as γ_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 Householder QR algorithm since inner products are the only computation that use mixed-precision arithmetic.

unclear what you mean, save it for next section. / presented in the following sections,

3 Householder QR Factorization

(HQR) never defined before

The Householder QR factorization uses Householder transformations to zero out elements below the diagonal of a matrix. First, we consider the simpler task of zeroing out all but the first element of a 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* $\mathbf{P}_\mathbf{v}$ *such that* $\mathbf{P}_\mathbf{v}$ *zeroes out* $\mathbf{x}$ *below the first element.*

$$\begin{aligned} \sigma &= -\operatorname{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2, \quad \mathbf{v} = \mathbf{x} - \sigma\hat{e_1}, \\ \beta &= \frac{2}{\mathbf{v}^\top\mathbf{v}} = -\frac{1}{\sigma\mathbf{v}_1}, \quad \mathbf{P}_\mathbf{v} = \mathbf{I}_m - \beta\mathbf{v}\mathbf{v}^\top \end{aligned} \tag{14}$$

The transformed vector, $\mathbf{P}_\mathbf{v}\mathbf{x}$*, has the same 2-norm as* $\mathbf{x}$ *since Householder transformations are orthogonal.*

$$\mathbf{P}_\mathbf{v}\mathbf{x} = \sigma\hat{e_1} \tag{15}$$

In addition, $\mathbf{P}_\mathbf{v}$ *is symmetric and orthogonal* $\mathbf{P}_\mathbf{v} = \mathbf{P}_\mathbf{v}^\top = \mathbf{P}_\mathbf{v}^{-1}$*, and therefore involutary* $\mathbf{P}_\mathbf{v}^2 = \mathbf{I}$.