Linear Algebra for Image Compression

Ethan Davis

March 2025

1 Introduction

Linear algebra is a pillar of high performance computing (HPC) relied on by numerical systems with examples such as parallelization, graphics, and machine learning. Take principal component analysis (PCA) as an example, it is used to reduce the dimensions of a matrix to train a machine learning model so that the model is more efficient and has better performance. Image compression is another example and it relies on the fundamental theorem of linear algebra that is described by Gilbert Strang as the four subspaces of a matrix, their orthogonality, the orthonormal bases that diagonalize a matrix, and the singular value decomposition [4]. Companies like Netflix use image compression to stream video, and organizations like NASA use it to send images from Mars 140 million miles away. In this paper we consider how linear algebra is used for image compression from first principles of theory to implementation of operations that are highly parallelizable.

2 Method

We use the Olivetti Faces dataset for image compression. Each image is grayscale and size 64×64. Since the images are grayscale, each pixel is an integer between $0 - 255$. Therefore, each image is a 64×64 matrix with elements between $0 - 255$. The Olivetti Faces dataset is available through scikit-learn as an archive from AT&T Laboratories Cambridge.

We do image compression by representing images by only their top k principal components (PC). To find the principal components of images we use eigen value decomposition (EVD) and singular value decomposition (SVD). To do EVD and SVD we implement the power method (PM) and the QR method (QRM).

2.1 Proof of Correctness

2.1.1 Power Method

The power method is used to find the dominant eigenpair [9]. Taking powers of a matrix $\mathbf{A}$ amplifies the dominant eigenpair such that other eigenpairs become effectively zero. Therefore, the product $\mathbf{A}^k\mathbf{x}$ where $\mathbf{x}$ is a random initial vector has the same direction as the dominant eigenvector.

We consider the diagonal factorization $\mathbf{A} = \mathbf{V}\mathbf{W}\mathbf{V}^{-1}$ where $\mathbf{V}$ and $\mathbf{W}$ are the eigenvectors and eigenvalues of $\mathbf{A}$, respectively [5]. Let $\mathbf{x}$ be a random initial vector and $\mathbf{c} = \mathbf{V}^{-1}\mathbf{x}$ be a combination of the eigenvectors. Then, the product $\mathbf{A}^k\mathbf{x}$ can be written as a linear combination of the eigenvectors [5].

$$\mathbf{A}^k\mathbf{x} = \mathbf{V}\mathbf{W}^k\mathbf{V}^{-1}\mathbf{x} = \mathbf{V}\mathbf{W}^k\mathbf{c} = \mathbf{c}_1\lambda_1^k\mathbf{v}_1 + \mathbf{c}_2\lambda_2^k\mathbf{v}_2 + \cdots + \mathbf{c}_n\lambda_n^k\mathbf{v}_n.$$

The power method can also be used to find the dominant eigenvalue. The Rayleigh quotient can be used to find the corresponding eigenvalue given an eigenvector [9]. The Rayleigh quotient assumes $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ is an

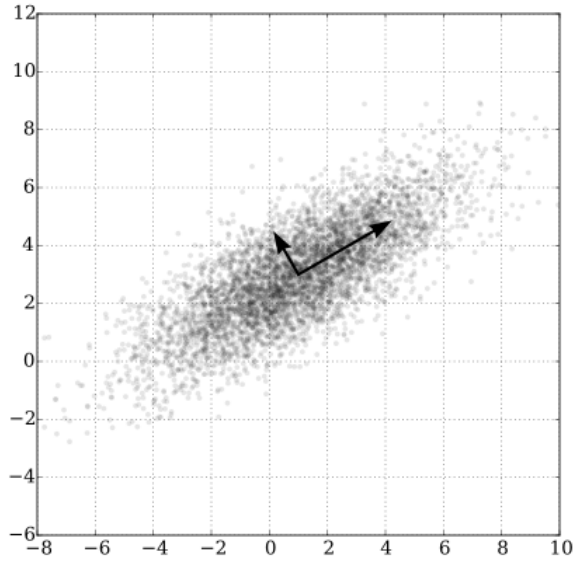

Figure 1: Example of principal components from Wikipedia. The eigenvectors determine the direction of these vectors. The eigenvalues determine the magnitude of these vectors. When choosing the top k principal components we choose the eigenvectors with larger eigenvalues first.

eigenpair and then solves for λ.

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \quad \text{becomes} \quad \mathbf{x}^\top\mathbf{A}\mathbf{x} = \lambda\mathbf{x}^\top\mathbf{x} \quad \text{becomes} \quad \frac{\mathbf{x}^\top\mathbf{A}\mathbf{x}}{\mathbf{x}^\top\mathbf{x}} = \lambda.$$

See Figure 2 for the pseudocode to find the dominant eigenpair of a matrix $\mathbf{A}$. It iterates a constant number of steps. Alternatively, stabilization of the Rayleigh quotient can be used to check for sufficient convergence. The pseudocode is also MATLAB.

```
function [v,w] = PowerMethod(A)
    n = size(A,1);
    v = ones(n,1);
    for k = 1:MAX
        v = A * v;
        v = v / norm(v);
    end
    w = (v' * A * v) / (v' * v);
end
```

Figure 2: Power method MATLAB pseudocode.

2.1.2 Deflation

The power method is used to find the dominant eigenpair. It can also be used to find the non-dominant eigenpairs [8]. We can deflate the effect of the dominant eigenpair such that iterations of the power method amplify the next dominant eigenpair.

Consider $\mathbf{Av} = \lambda\mathbf{v}$ is the dominant eigenpair of a matrix $\mathbf{A}$. Then $\mathbf{vv}^\top$ is a rank-1 linear combination of $\mathbf{v}$ that is essentially a projection onto $\mathbf{v}$. The product $\lambda\mathbf{vv}^\top$ scales the column space by the magnitude of the eigenvalue. The difference $\mathbf{A} - \lambda\mathbf{vv}^\top$ removes the effect of the dominant eigenpair $\lambda\mathbf{vv}^\top$ from $\mathbf{A}$. Therefore, iteration of the power method will converge to the next dominant eigenpair, which is orthogonal to $\mathbf{v}$.

In other words, each iteration of the power method deflates the matrix $\mathbf{A}$ to the updated matrix $\mathbf{B} = \mathbf{A} - \lambda\mathbf{vv}^\top$. The next iteration of the power method takes the matrix $\mathbf{B}$. This method of deflation can be used to find all eigenpairs by the power method.

See Figure 3 for the pseudocode of deflation. It takes the matrix $\mathbf{A}$, the eigenvector $\mathbf{v}$, and the eigenvalue λ and gives the updated matrix $\mathbf{B}$. Then, see Figure 4 for pseudocode that uses the power method and deflation to find all eigenpairs. For a matrix with size n, then n iterations of the power method and deflation can be used to find all eigenpairs.

```
function B = PowerMethodDeflation(A,v,w)
    B = A - w * (v * v');
end
```

Figure 3: Power method deflation MATLAB pseudocode.

```
function [V,W] = PowerMethodN(A)
    n = size(A,1);
    V = zeros(n,n);
    W = zeros(n,n);
    for k = 1:n
        [v,w] = PowerMethod(A);
        V(:,k) = v;
        W(k,k) = w;
        A = PowerMethodDeflation(A,v,w);
    end
end
```

Figure 4: Power method MATLAB pseudocode to find all eigenpairs.

2.1.3 Householder Reflection

To understand Householder reflections, we need to understand the Householder matrix [2]. This matrix is at the heart of QR factorization [6]. The Householder matrix can be understood from the projection matrix [6].

We consider projecting a vector $\mathbf{b}$ onto a vector $\mathbf{a}$. See Figure 5 for an example. The projection $\mathbf{p}$ is proportional to $\mathbf{a}$ such that $\mathbf{p} = k\mathbf{a}$ for some scalar k. We let $\mathbf{e} = \mathbf{b} - \mathbf{p} = \mathbf{b} - k\mathbf{a}$ be the orthogonal error between $\mathbf{a}$ and $\mathbf{b}$. Therefore, the dot product $\mathbf{ae} = 0$ and we can solve for k.

$$\mathbf{ae} = \mathbf{a}(\mathbf{b} - k\mathbf{a}) = \mathbf{ab} - k\mathbf{aa} = 0 \quad \text{becomes} \quad k = \mathbf{ab}/\mathbf{aa}.$$

Furthermore, we can solve for the projection matrix $\mathbf{P}$ that projects any vector onto the vector $\mathbf{a}$ for any projection $\mathbf{p}$.

$$\mathbf{p} = \mathbf{a}k = \mathbf{a}(\mathbf{a}^\top\mathbf{b}/\mathbf{a}^\top\mathbf{a}) = (\mathbf{a}\mathbf{a}^\top)\mathbf{b}/(\mathbf{a}^\top\mathbf{a}) = \mathbf{P}\mathbf{b} \quad \text{becomes} \quad \mathbf{P} = (\mathbf{a}\mathbf{a}^\top)/\mathbf{a}^\top\mathbf{a}.$$

We can use the projection matrix $\mathbf{P}$ to find the Householder matrix or elementary reflector. Given some vector $\mathbf{b}$ and its projection $\mathbf{p}$, we can subtract away the projection twice to get the reflection of $\mathbf{b}$ over its projection [6]. Reflection occurs across the vector orthogonal to $\mathbf{p}$. If we consider the matrix $\mathbf{H}$ to reflect $\mathbf{b}$ over $\mathbf{p}$, then we can solve for the Householder matrix.

$$\mathbf{Hb} = \mathbf{b} - 2\mathbf{p} = \mathbf{b} - 2\mathbf{Pb} = (\mathbf{I} - 2\mathbf{P})\mathbf{b} \quad \text{becomes} \quad \mathbf{H} = \mathbf{I} - 2\mathbf{P}.$$

We notice that the reflection $\mathbf{Hb}$ of some vector $\mathbf{b}$ has the same norm $\|\mathbf{b}\|$.

$$\|\mathbf{Hb}\|^2 = \|-\mathbf{p}\|^2 + \|\mathbf{e}\|^2 = \|\mathbf{p}\|^2 + \|\mathbf{e}\|^2 = \|\mathbf{b}\|^2 \quad \text{becomes} \quad \|\mathbf{Hb}\| = \|\mathbf{b}\|.$$

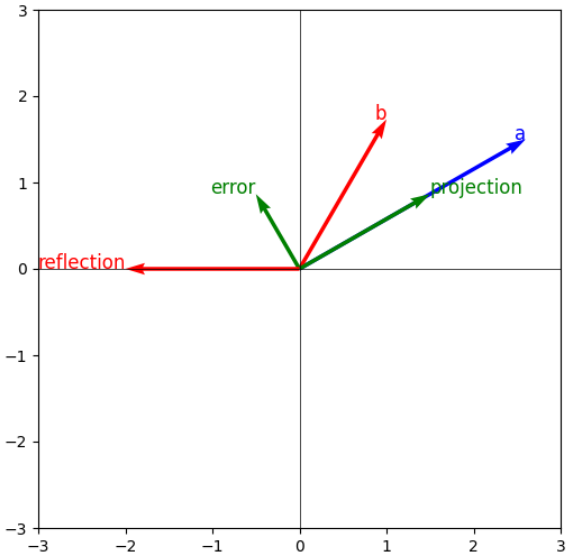

Figure 5: Projection of $\mathbf{b}$ onto $\mathbf{a}$ and reflection of $\mathbf{b}$ over $\mathbf{a}$.

2.1.4 QR Factorization

We can use Householder reflections to get the factorization $\mathbf{A} = \mathbf{QR}$ where $\mathbf{A}$ is a matrix, $\mathbf{Q}$ is an orthogonal matrix, and $\mathbf{R}$ is an upper triangular matrix [2]. By choosing vectors $\mathbf{v} = \mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1$ and $\mathbf{u} = \mathbf{v}/\|\mathbf{v}\|$, we can reflect any vector $\mathbf{x}$ onto the first vector of the standard basis $\mathbf{e}_1$. In other words, $\mathbf{Hx} = \|\mathbf{x}\|\mathbf{e}_1$.

$$
\begin{aligned}
\mathbf{Hx} &= (\mathbf{I} - 2\mathbf{P})\mathbf{x} && (1)\\
&= (\mathbf{I} - 2\mathbf{uu}^\top / \mathbf{u}^\top \mathbf{u})\mathbf{x} && (2)\\
&= (\mathbf{I} - 2\mathbf{uu}^\top)\mathbf{x} && (3)\\
&= \mathbf{x} - 2\mathbf{uu}^\top \mathbf{x} && (4)\\
&= \mathbf{x} - 2\mathbf{u}(\mathbf{u}^\top \mathbf{x}) && (5)\\
&= \mathbf{x} - 2\mathbf{u}\Big(\frac{\mathbf{v}^\top \mathbf{x}}{\|\mathbf{v}\|}\Big) && (6)\\
&= \mathbf{x} - 2\mathbf{u}\Big(\frac{(\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1)^\top \mathbf{x}}{\|\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1\|}\Big) && (7)\\
&= \mathbf{x} - 2\mathbf{u}\Big(\frac{(\mathbf{x}^\top - \|\mathbf{x}\|\mathbf{e}_1^\top)\mathbf{x}}{\|\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1\|}\Big) && (8)\\
&= \mathbf{x} - 2\mathbf{u}\Big(\frac{\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1}{\|\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1\|}\Big) && (9)\\
&= \mathbf{x} - 2\Big(\frac{\mathbf{v}}{\|\mathbf{v}\|}\Big)\Big(\frac{\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1}{\|\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1\|}\Big) && (10)\\
&= \mathbf{x} - 2\Big(\frac{\mathbf{x} - \|\mathbf{x}\|\mathbf{e_1}}{\|\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1\|}\Big)\Big(\frac{\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1}{\|\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1\|}\Big) && (11)\\
&= \mathbf{x} - 2\Big(\frac{(\mathbf{x} - \|\mathbf{x}\|\mathbf{e_1})(\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1)}{\|\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1\|^2}\Big) && (12)\\
&= \mathbf{x} - 2\Big(\frac{(\mathbf{x} - \|\mathbf{x}\|\mathbf{e_1})(\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1)}{(\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1)^\top(\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1)}\Big) && (13)\\
&= \mathbf{x} - 2\Big(\frac{(\mathbf{x} - \|\mathbf{x}\|\mathbf{e_1})(\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1)}{(\mathbf{x}^\top - \|\mathbf{x}\|\mathbf{e}_1^\top)(\mathbf{x} - \|\mathbf{x}\|\mathbf{e}_1)}\Big) && (14)\\
&= \mathbf{x} - 2\Big(\frac{(\mathbf{x} - \|\mathbf{x}\|\mathbf{e_1})(\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1)}{\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1 - \|\mathbf{x}\|\mathbf{x}_1 + \|\mathbf{x}\|^2}\Big) && (15)\\
&= \mathbf{x} - 2\Big(\frac{(\mathbf{x} - \|\mathbf{x}\|\mathbf{e_1})(\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1)}{2\|\mathbf{x}\|^2 - 2\|\mathbf{x}\|\mathbf{x}_1}\Big) && (16)\\
&= \mathbf{x} - 2\Big(\frac{(\mathbf{x} - \|\mathbf{x}\|\mathbf{e_1})(\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1)}{2(\|\mathbf{x}\|^2 - \|\mathbf{x}\|\mathbf{x}_1)}\Big) && (17)\\
&= \mathbf{x} - (\mathbf{x} - \|\mathbf{x}\|\mathbf{e_1}) && (18)\\
&= \|\mathbf{x}\|\mathbf{e}_1. && (19)
\end{aligned}
$$

Reflecting onto the first vector of the standard basis gives us a vector with zeros below the first element. With multiple Householder reflections, we can transform a matrix $\mathbf{A}$ into an upper triangular matrix $\mathbf{R}$ with zeros below the diagonal. Then, the product of all Householder matrices is the orthogonal matrix $\mathbf{Q}$. This gives us the factorization $\mathbf{A} = \mathbf{QR}$.

$$
\mathbf{Q}_n \cdots \mathbf{Q}_2\mathbf{Q}_1\mathbf{A} = \mathbf{R} \quad \text{becomes} \quad \mathbf{A} = (\mathbf{Q}_n \cdots \mathbf{Q}_2\mathbf{Q}_1)^{-1}\mathbf{R} = \mathbf{Q}_1^\top \mathbf{Q}_2^\top \cdots \mathbf{Q}_n^\top \mathbf{R} = \mathbf{QR}.
$$

We know that for an orthogonal matrix $\mathbf{Q}$, its transpose equals its inverse $\mathbf{Q}^\top = \mathbf{Q}^{-1}$ since $\mathbf{Q}^\top\mathbf{Q} = \mathbf{Q}^{-1}\mathbf{Q} = \mathbf{I}$. This comes from the definition of an orthogonal matrix where each vector is orthonormal [6]. The dot product of each vector with itself is one, and the dot product with all other vectors is zero because they are orthonormal.

See Figure 6 for pseudocode that takes a vector $\mathbf{x}$ and gives the vector $\mathbf{u}$ such that $\mathbf{Hx} = \|\mathbf{x}\|\mathbf{e}_1$ where $\mathbf{e}_1$ is the first vector of the standard basis [2]. Then, see Figure 7 for pseudocode that takes a matrix $\mathbf{A}$ and uses Householder reflections to get the orthogonal matrix $\mathbf{Q}$ and upper triangular matrix $\mathbf{R}$ that together give the QR factorization [2]. This pseudocode iterates Householder reflections of $\mathbf{A}$ such that each iteration cancels the elements below the diagonal of a column vector and creates $\mathbf{R}$.

```
function u = HouseholderReflection(x)
    n = size(x,1);
    e1 = eye(n,1);
    v = x - norm(x) * e1;
    u = v / norm(v);
end
```

Figure 6: Householder reflection MATLAB pseudocode.

```
function [Q,R] = QrDecomposition(A)
    n = size(A,1);
    Q = eye(n);
    R = A;
    for k = 1:n
        u = zeros(n,1);
        u(k:n) = HouseholderReflection(R(k:n,k));
        Q = Q - 2 * Q * (u * u');
        R = R - 2 * (u * u') * R;
    end
end
```

Figure 7: QR decomposition MATLAB pseudocode.

2.1.5 QR Method

Iterations of QR factorization can be used for EVD and SVD. For EVD the eigenpairs are found. Alternatively, for SVD, the right singular vectors and singular values are found, and then finding the left singular vectors is a function of the right singular vectors and the singular values.

To understand the QR method, we have to understand similar matrices [5]. Similar matrices have the same eigenvalues and different eigenvectors. Therefore, similar matrices describe a family of matrices with the same characteristics but are transformed, such as by rotations or reflections in space.

All matrices $\mathbf{A}$ and $\mathbf{C}$ where $\mathbf{A} = \mathbf{B}\mathbf{C}\mathbf{B}^{-1}$ are similar [5]. They have the same eigenvalues. For eigenpair $\mathbf{C}\mathbf{x} = \lambda\mathbf{x}$, then $\mathbf{B}\mathbf{C}\mathbf{B}^{-1}$ has the same eigenvalue with the new eigenvector $\mathbf{B}\mathbf{x}$.

$$(\mathbf{B}\mathbf{C}\mathbf{B}^{-1})(\mathbf{B}\mathbf{x}) = \mathbf{B}\mathbf{C}\mathbf{x} = \mathbf{B}\lambda\mathbf{x} = \lambda(\mathbf{B}\mathbf{x}).$$

The QR method iterates k transformations of the matrix $\mathbf{A}$ until $\mathbf{A}_k = \mathbf{W}$ is the diagonal eigenvalue matrix of $\mathbf{A}$ that is similar to $\mathbf{A}$ [3]. In other words, for some symmetric and non-singular matrix $\mathbf{A}$ the QR method converges to the eigenvalue matrix $\mathbf{W}$. It also gives us the eigenvector matrix $\mathbf{V}$.

Let us consider $\mathbf{A}$ is a symmetric and non-singular matrix. By the QR factorization, we know that $\mathbf{A} = \mathbf{Q}\mathbf{R}$. Then $\mathbf{Q}^\top\mathbf{A} = \mathbf{Q}^{-1}\mathbf{A} = \mathbf{R}$ and $\mathbf{Q}^\top\mathbf{A}\mathbf{Q} = \mathbf{R}\mathbf{Q}$. We notice that $\mathbf{R}\mathbf{Q}$ is symmetric since $(\mathbf{Q}^\top\mathbf{A}\mathbf{Q})^\top = \mathbf{Q}^\top\mathbf{A}^\top\mathbf{Q} = \mathbf{Q}^\top\mathbf{A}\mathbf{Q} = \mathbf{R}\mathbf{Q}$. Furthermore, for similar and symmetric matrices $\mathbf{A}$ and $\mathbf{C}$ we can say that $\mathbf{A} = \mathbf{Q}^{-1}\mathbf{C}\mathbf{Q}$.

$$\mathbf{A} = \mathbf{Q}\mathbf{C}\mathbf{Q}^{-1} = \mathbf{Q}\mathbf{C}\mathbf{Q}^\top = \mathbf{Q}\mathbf{C}^\top\mathbf{Q}^\top = (\mathbf{Q}^\top\mathbf{C}\mathbf{Q})^\top = \mathbf{Q}^\top\mathbf{C}^\top\mathbf{Q} = \mathbf{Q}^\top\mathbf{C}\mathbf{Q} = \mathbf{Q}^{-1}\mathbf{C}\mathbf{Q}.$$

The QR method iterates QR factorizations of similar matrices.

$$\mathbf{A} = \mathbf{Q}_1\mathbf{R}_1 = \mathbf{Q}_1\mathbf{R}_1\mathbf{Q}_1\mathbf{Q}_1^\top = \mathbf{Q}_1\mathbf{A}_1\mathbf{Q}_1^\top \quad \text{becomes} \quad \mathbf{A}_{k-1} = \mathbf{Q}_k\mathbf{A}_k\mathbf{Q}_k^\top.$$

We can write $\mathbf{A}_k$ as a function of $\mathbf{A}$ and all $\mathbf{Q}$.

$$\mathbf{A}_k = \mathbf{Q}_k^\top\mathbf{A}_{k-1}\mathbf{Q}_k = (\mathbf{Q}_1\mathbf{Q}_2\cdots\mathbf{Q}_k)^\top\mathbf{A}(\mathbf{Q}_1\mathbf{Q}_2\cdots\mathbf{Q}_k).$$

Repeated products $\mathbf{Q}_k^\top\mathbf{A}_{k-1}\mathbf{Q}_k$ of matrices similar to $\mathbf{A}$ amplifies the eigenvalues on the diagonal and makes the off-diagonal become effectively zero. We consider the factorization $\mathbf{A} = \mathbf{V}\mathbf{W}\mathbf{V}^{-1}$ where $\mathbf{V}$ and $\mathbf{W}$ are the eigenvectors and eigenvalues, respectively [5]. Then as $k \to \infty$ we have $\mathbf{Q}_1\mathbf{Q}_2\cdots\mathbf{Q}_k \to \mathbf{V}$ and $\mathbf{A}_k \to \mathbf{W}$ [3].

$$\mathbf{A}_k = (\mathbf{Q}_1\mathbf{Q}_2\cdots\mathbf{Q}_k)^\top\mathbf{A}(\mathbf{Q}_1\mathbf{Q}_2\cdots\mathbf{Q}_k) = (\mathbf{Q}_1\mathbf{Q}_2\cdots\mathbf{Q}_k)^\top\mathbf{V}\mathbf{W}\mathbf{V}^{-1}(\mathbf{Q}_1\mathbf{Q}_2\cdots\mathbf{Q}_k) = \mathbf{W}.$$

This is a generalization of diagonalizing the matrix $\mathbf{A}$. Rather than diagonalize $\mathbf{A}$ the QR method diagonalizes any matrix similar to $\mathbf{A}$. The result is the diagonal eigenvalues $\mathbf{W}$ and orthogonal eigenvectors $\mathbf{V}$.

See Figure 8 for the pseudocode of the QR method. Here, we assume a constant number of iterations. Alternatively, iteration can end when the sum of the off-diagonal elements is sufficiently small.

```
function [V,W] = QrMethod(A)
    n = size(A,1);
    V = eye(n);
    for k = 1:MAX
        [Q,R] = QrDecomposition(A);
        A = R * Q;
        V = V * Q;
    end
    W = A;
end
```

Figure 8: QR method MATLAB pseudocode.

2.1.6 Top K Principal Components

Given all n eigenpairs and singular triplets of some matrix $\mathbf{A}$, we can transform $\mathbf{A}$ to its rank-k representation where k is the top k eigenpairs and singular triplets [7]. We notice that the power and QR methods naturally order the eigenvalues and singular values along the diagonal by the effect of amplification. This ordering simplifies finding the top k elements.

For SVD, we find the rank-k transformation of $\mathbf{A}$ by $\mathbf{A}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top$ with size $m \times n$. Here, $\mathbf{U}_k$ is the rank-k input space with size $m \times k$, $\mathbf{\Sigma}_k$ is the rank-k singular values with size $k \times k$, and $\mathbf{V}_k^\top$ is the rank-k output space with size $k \times n$. See Figure 9 for pseudocode to find the rank-k transformation of $\mathbf{A}$ [7].

Then, for EVD, given the eigenvectors $\mathbf{V}$ of $\mathbf{A}^\top\mathbf{A}$, we can find the rank-$k$ transformation of $\mathbf{A}$ by principal component projection $\mathbf{A}_k = \mathbf{A}\mathbf{V}_k\mathbf{V}_k^\top$ where $\mathbf{V}_k$ has size $n \times k$, $\mathbf{V}_k\mathbf{V}_k^\top$ has size $n \times n$, $\mathbf{A}$ has size $m \times n$, and $\mathbf{A}_k$ has size $m \times n$. See Figure 10 for pseudocode to find the rank-k transformation of $\mathbf{A}$ [7].

```
function Ak = SvdTopK(U,S,V,k)
    Ak = U(:,1:k) * S(1:k,1:k) * V(:,1:k)';
end
```

Figure 9: SVD top k principal components MATLAB pseudocode.

```
function Ak = EvdTopK(A,V,k)
    Ak = A * V(:,1:k) * V(:,1:k)';
end
```

Figure 10: EVD top k principal components MATLAB pseudocode.

2.2 Implementation

For preprocessing the Olivetti Faces dataset we use Python, Scikit-learn, and Numpy to read the images from their AT&T Laboratories Cambridge archive and into memory on our local machine. By using Matplotlib we plot the images to preview them. The images are written to local disk as raw binary for processing image compression. For postprocessing after the Olivetti Faces dataset is compressed such that images are represented by their top k principal components we use Matplotlib to preview the results.

For processing the Olivetti Faces dataset we use C and its standard library to implement PM and QRM. For numerical stability we use BLAS and LAPACK for QR decomposition. Each image is processed in sequence and results are written to a binary file unique to its compression scheme.

The C implementation is available on our Github [1]. There is a README with instructions for execution on a MacBook Pro 14-inch 2021 with Apple M1 Pro. For installation of BLAS and LAPACK we use Homebrew for MacOS. In general, we use a MacBook Pro 14-inch 2021 with Apple M1 Pro for all file processing including to read the size of files.

3 Result

See Figure 11 an example from the Olivetti Faces dataset without image compression. All principal components are used to represent the image regardless of their magnitude even though some of PC could be redundant or insignificant. Since the images have size 64×64 then they have at most 64 principal components.

As a proof of concept we compress one image with all compression schemes using the top 4 principal components. See Figure 12 for the results. Each compressed image looks to be the same. We continue with further examples. However, given that all image compression schemes look to give the same results, we only show examples as they relate to the number of top k principal components used for image compression.

See Figures 13 through 18 for examples of image compression with the top 1, 2, 4, 8, 16, and 32 principal components. Figure 13 shows image compression with the top 1 PC and is interesting because we see all straight lines. This is interesting because it confirms what we expect to see since we are projecting onto one dimension.

For image compression with the top 2, 4, and 8 PC there is too much detail missing to reasonably make out the original image. With the top 16 PC we see what looks like the original images with minor details missing. Then with the top 32 PC we see what looks like the original images.

Figure 11: Example of images from Olivetti Faces dataset without compression.

Figure 12: Image compression with the top 4 principal components using all compression schemes.

Let us consider the file sizes on disk of the original and compressed images. Given what we see when we preview the compressed images, we would never want to share the images compressed with only the top 1, 2, or 4 principal components. Rarely would we want to share the images compressed with the top 8 PC. However, we could be willing to share the images compressed with the top 16 PC which are smaller than the original images. Furthermore, images compressed with the top 32 PC are difficult to distinguish from the original images and are smaller too. See Table 1 for the file sizes on disk.

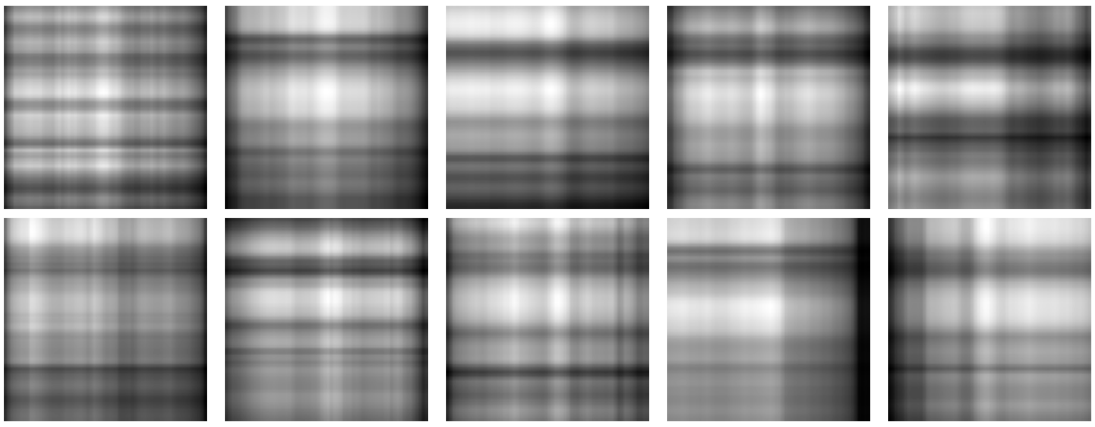

Figure 13: Olivetti Faces images compressed with the top 1 principal component.

Figure 14: Olivetti Faces images compressed with the top 2 principal components.

Figure 15: Olivetti Faces images compressed with the top 4 principal components.

Figure 16: Olivetti Faces images compressed with the top 8 principal components.

Figure 17: Olivetti Faces images compressed with the top 16 principal components.

Figure 18: Olivetti Faces images compressed with the top 32 principal components.

Top k PC	File Size (KB)
1	164
2	172
4	180
8	197
16	209
32	217
Original	266

Table 1: File sizes on disk.

4 Discussion

Using image compression as an example, our results show that principal component analysis (PCA) can be used to reduce the dimensions of a matrix. By removing a certain number of redundant or insignificant dimensions, our results show images that look to be the same as their original but are smaller on disk.

We used the fundamental theorem of linear algebra to find these results. For example, knowledge of the four subspaces and their orthogonality was used in our analysis of Householder reflection and QR decomposition. Furthermore, we find the orthonormal bases that diagonalize a matrix for EVD, and extend these results to find the SVD.

Our results are valid because they are backed by our proof of correctness. Through experiments we build the reliability of our results. These results show the impact of linear algebra in everyday life and use little burden to do so. Given the simplicity of our dataset and methods of image compression, there is much space for improvement of our results in future work.

References

[1] Ethan Davis. Eigenface. `https://github.com/davisethan/eigenface`, 2025.

[2] Cleve Moler. Compare gram-schmidt and householder orthogonalization algorithms. `https://blogs.mathworks.com/cleve/2016/07/25/compare-gram-schmidt-and-householder-orthogonalization-algorithms/`, 2016.

[3] University of Washington. The qr (eigenvalue) algorithm. `https://faculty.washington.edu/trogdon/105A/html/Lecture24.html`.

[4] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 5th edition, 2016.

[5] Gilbert Strang. *Introduction to Linear Algebra*, chapter 6. Wellesley-Cambridge Press, 5th edition, 2016.

[6] Gilbert Strang. *Introduction to Linear Algebra*, chapter 4. Wellesley-Cambridge Press, 5th edition, 2016.

[7] Gilbert Strang. *Introduction to Linear Algebra*, chapter 7. Wellesley-Cambridge Press, 5th edition, 2016.

[8] Stanford University. Deflation matrices. `https://web.stanford.edu/class/cs205/reviews/review_3.pdf`.

[9] Unknown. Power method for approximating eigenvalues. `https://ergodic.ugr.es/cphys/lecciones/fortran/power_method.pdf`.