CHEME 5820: Machine Learning for Engineers **Spring 2025**

Lecture 2a: Eigendecomposition of Data and Systems

Lecturer: Jeffrey Varner

Topics

- **Eigendecomposition** is a fundamental concept in linear algebra that decomposes a matrix into its constituent parts, the eigenvectors and eigenvalues. Eigendecomposition is widely used in many areas of mathematics, engineering, and physics to analyze data and systems.
- **QR decomposition** is a factorization of a matrix into an orthogonal matrix and an upper triangular matrix. The QR decomposition is useful for solving linear systems of equations, and for computing the eigenvalues and eigenvectors of a matrix (using the QR iteration algorithm), among other applications.
- **Gram-Schmidt orthogonalization** is a procedure that generates a set of mutually orthogonal vectors starting from a set of linearly independent vectors. The Gram-Schmidt orthogonalization procedure is used to compute the QR decomposition of a matrix.

1 Introduction

The eigendecomposition of a matrix is a fundamental concept in linear algebra. In this lecture, we will discuss the eigendecomposition of a matrix, and how it can be used to analyze data and systems in unsupervised machine learning. Eigendecomposition allows us to decompose a matrix into its constituent parts, the eigenvectors and eigenvalues, which can be used to understand the structure of the data or the system represented by the matrix.

2 Eigendecomposition

Suppose we have a real square matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$ which could be a measurement dataset, e.g., the columns of $\mathbf{A}$ represent feature vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$ or an adjacency array in a graph with m nodes, etc. Eigenvalue-eigenvector problems involve finding a set of scalar values $\{\lambda_1, \ldots, \lambda_m\}$ called eigenvalues and a set of linearly independent vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ called eigenvectors such that:

$$\mathbf{A} \cdot \mathbf{v}_j = \lambda_j \mathbf{v}_j \qquad j = 1, 2, \ldots, m \tag{1}$$

where $\mathbf{v} \in \mathbb{C}^m$ and $\lambda \in \mathbb{C}$. We can put the eigenvalues and eigenvectors together in matrix-vector form, which gives us an interesting matrix decomposition:

$$\mathbf{A} = \mathbf{V} \cdot \text{diag}(\lambda) \cdot \mathbf{V}^{-1} \tag{2}$$

where $\mathbf{V}$ denotes the matrix of eigenvectors, where the eigenvectors form the columns of the matrix $\mathbf{V}$, $\text{diag}(\lambda)$ denotes a diagonal matrix with the eigenvalues along the main diagonal, and $\mathbf{V}^{-1}$ denotes the

inverse of the eigenvalue matrix. The eigendecomposition of a matrix is a powerful tool that can be used to analyze data and systems in many areas of mathematics, engineering, and physics. Let's discuss some of the interpretations of the eigendecomposition, data reduction, which is an interesting application of the eigendecomposition, and then how we compute the eigendecomposition of a matrix.

2.1 Interpretation of Eigendecomposition

Eigenvectors represent fundamental directions of the matrix $\mathbf{A}$. For the linear transformation defined by a matrix $\mathbf{A}$, eigenvectors are the only vectors that do not change direction during the transformation. Thus, if we think about the matrix $\mathbf{A}$ as a machine, we put the eigenvector $\mathbf{v}_\star$ into the machine, and we get back the same eigenvector $\mathbf{v}_\star$ multiplied by a scalar, the eigenvalue $\lambda_\star$. On the other hand, eigenvalues are scale factors for their eigenvector. An eigenvalue is a scalar that indicates how much a corresponding eigenvector is stretched or compressed during a linear transformation represented by the matrix $\mathbf{A}$. We can use the eigendecomposition to diagonalize the matrix $\mathbf{A}$, i.e., transform the matrix into a diagonal form where the eigenvalues lie along the main diagonal. To see this, solve the eigendecomposition for the $\text{diag}(\lambda) = \mathbf{V}^{-1} \cdot \mathbf{A} \cdot \mathbf{V}$. We can also use the eigenvalues to classify a matrix $\mathbf{A}$ as positive (semi)definite or negative (semi)definite (which will be handy later). Further, if the matrix $\mathbf{A}$ is symmetric, and all entries are positive, then all the eigenvalues will be real-valued, and the eigenvectors will be orthogonal (handy properties, as we shall soon see). Finally, another interpretation is that eigenvectors represent the most critical directions in the data or system, and eigenvalues (or functions of them) represent their importance. Let's now consider the special case where the matrix $\mathbf{A}$ is both symmetric and real-valued.

2.2 Symmetric Real Matrices

The eigendecomposition of a symmetric real matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$ has some special properties. First, the eigenvalues of a symmetric real matrix are real-valued. Next, the eigenvectors of a symmetric real matrix are orthogonal, i.e., $< v_i, v_j >= 0$. This means that the eigenvectors of a real-symmetric matrix form an orthogonal basis for the space spanned by the matrix $\mathbf{A}$. Further, we can make that basis orthonormal by normalizing the eigenvectors, i.e., $\hat{v}_i = v_i / ||v_i||$ where $||v_i||$ is the norm (length) of the eigenvector v_i. However, where would we encounter a real-value symmetric matrix $\mathbf{A}$? One example is the covariance matrix of a dataset.

The covariance matrix of a dataset is an example of a symmetric real matrix that we will encounter in a variety of applications. The covariance matrix is a square matrix that summarizes the variance and covariance of the features in the dataset. Suppose we have a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ where each $\mathbf{x}_i \in \mathbb{R}^m$ is a feature vector. The covariance of feature vectors i and j, denoted as $\text{cov}(\mathbf{x}_i, \mathbf{x}_j)$, is an $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ real-valued symmetric matrix $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ with elements:

$$\Sigma_{ij} = \text{cov}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_i \, \sigma_j \, \rho_{ij} \qquad \text{for} \quad i, j \in \mathcal{D} \tag{3}$$

where σ_i denote the standard deviation of the feature vector $\mathbf{x}_i$, σ_j denote the standard deviation of the feature vector $\mathbf{x}_j$, and ρ_{ij} denotes the correlation between features i and j in the dataset $\mathcal{D}$. The correlation is given by:

$$\rho_{ij} = \frac{\mathbb{E}(\mathbf{x}_i - \mu_i) \cdot \mathbb{E}(\mathbf{x}_j - \mu_j)}{\sigma_i \sigma_j} \qquad \text{for} \quad i, j \in \mathcal{D} \tag{4}$$

The diagonal elements of the covariance matrix $\Sigma_{ii} \in \mathbf{\Sigma}$ are the variances of features i, while the off-diagonal elements $\Sigma_{ij} \in \mathbf{\Sigma}$ for $i \neq j$ measure the relationship between features i and j in the dataset $\mathcal{D}$. The interesting bit of this, is that we can use the eigendecomposition for data reduction, i.e., factor the dataset $\mathcal{D}$ into its most important features. For example, if $n \gg 2$, we can use the eigendecomposition to reduce

the dimensionality of the dataset $\mathcal{D}$ to 2 or 3 dimensions, which can be visualized. We can also use the eigendecomposition to find the most important features in the dataset, which can be used for clustering, classification, or other machine learning tasks.

Data reduction

Data reduction is a common application of the eigendecomposition of a matrix. Suppose we have a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ where each $\mathbf{x}_i \in \mathbb{R}^m$ is a feature vector. We can use the eigendecomposition of the covariance matrix of the dataset to reduce the dimensionality of the dataset. Let $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ be the covariance matrix of the dataset $\mathcal{D}$. Then the equation $\mathbf{\Sigma} \cdot \mathbf{v}_j = \lambda_j \mathbf{v}_j$ gives us the eigenvectors and eigenvalues of the covariance matrix. Becuase the covariance matrix is symmetric, the eigenvectors are orthogonal, and the eigenvalues are real-valued. Further, we can normalize the eigenvectors to make them an orthonormal, i.e., $\hat{\mathbf{v}}_i = \mathbf{v}_i / ||\mathbf{v}_i||_2$ where $||\mathbf{v}_i||_2$ is the $l2$-norm (Euclidean length) of the eigenvector $\mathbf{v}_i$.

Suppose we wanted to reduce the dimensionality of the feature vectors $\mathbf{x} \in \mathcal{D}$ from m to k dimensions, where $k < m$, i.e., we want to transform $\mathbf{x}_i \rightarrow \mathbf{y}_i$ where $\mathbf{y}_i \in \mathbb{R}^k$. We may want to do this for a variety of reasons, for example, to visualize the data in 2 or 3 dimensions, or to reduce the computational complexity of a machine learning algorithm. To make this possible, we suppose there exists a projection matrix $\mathbf{P}$ such that:

$$\mathbf{y}_i = \mathbf{P}\mathbf{x}_i \quad i = 1, 2, \ldots, n \tag{5}$$

The projection matrix $\mathbf{P}$ will be a $k \times m$ matrix composed of some transform vectors. The open question is what are the best transform vectors to use? The answer is to use the eigenvectors of the covariance matrix $\mathbf{\Sigma}$. We are going to put off discussing why this is the case until we discuss the *Principal Component Analysis* (PCA) algorithm. However, of we build our transformation matrix using the eigenvectors gives the reduced feature vector corresponding to $x \in \mathcal{D}$ is given by:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix} = \begin{bmatrix} — \ \hat{\mathbf{v}}_1^\top \ — \\ — \ \hat{\mathbf{v}}_2^\top \ — \\ \vdots \\ — \ \hat{\mathbf{v}}_k^\top \ — \end{bmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \tag{6}$$

where $\hat{\mathbf{v}}_\star^\top$ denotes the transpose of the scaled eigenvector $\hat{\mathbf{v}}_\star$ of the covariance matrix $\mathbf{\Sigma}$. Thus, each component of the reduced vector $\mathbf{y}$ corresponding to $\mathbf{x} \in \mathcal{D}$ is given by:

$$y_j = \hat{\mathbf{v}}_j^\top \cdot \mathbf{x} \quad j = 1, 2, \ldots, k \tag{7}$$

3 Computing the Eigendecomposition of a Matrix

There are several techniques to compute the eigendecomposition of a matrix. The most common method is power iteration, an iterative algorithm that finds the eigenvector corresponding to the largest eigenvalue of a matrix. The power iteration method is simple and easy to implement. Still, it may not converge to the desired eigenvector if the matrix is ill-conditioned or has multiple eigenvalues of the same magnitude. Alternatively, we can use the QR algorithm, a more robust method to find all a matrix's eigenvalues and eigenvectors.

3.1 Power Iteration

The power iteration method is an iterative algorithm to compute the largest eigenvalue and its corresponding eigenvector of a square (real) matrix; we'll consider only real-valued matrices here, but this approach can also be used for matrices with complex entries.

The power iteration method consists of two phases:

- **Phase 1: Eigenvector**: Suppose we have a real-valued square diagonalizable matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$ whose eigenvalues have the property $|\lambda_1| \geq |\lambda_2| \cdots \geq |\lambda_m|$. Then, the eigenvector $\mathbf{v}_1 \in \mathbb{C}^m$ which corresponds to the largest eigenvalue $\lambda_1 \in \mathbb{C}$ can be (iteratively) estimated as:

$$\mathbf{v}_1^{(k+1)} = \frac{\mathbf{A}\mathbf{v}_1^{(k)}}{\|\mathbf{A}\mathbf{v}_1^{(k)}\|} \quad k = 0, 1, 2 \ldots \tag{8}$$

  where $\|\star\|$ denotes the L2 (Euclidean) vector norm. The power iteration method converges to a value for the eigenvector as $k \to \infty$ when a few properties are true, namely, $|\lambda_1|/|\lambda_2| < 1$ (which is unknown beforehand), and we pick an appropriate initial guess for $\mathbf{v}_1$ (in our case, a random vector will work)
- **Phase 2: Eigenvalue**: Once we have an estimate for the eigenvector $\hat{\mathbf{v}}_1$, we can estimate the corresponding eigenvalue $\hat{\lambda}_1$ using the Rayleigh quotient. This argument proceeds from the definition of the eigenvalues and eigenvectors. We know, from the definition of eigenvalue-eigenvector pairs, that:

$$\mathbf{A}\hat{\mathbf{v}}_1 - \hat{\lambda}_1\hat{\mathbf{v}}_1 \simeq 0 \tag{9}$$

  where we use the $\simeq$ symbol because we don't have the true eigenvector $\mathbf{v}_1$, only an estimate of it. To solve this expression for the (estimated) eigenvalue $\hat{\lambda}_1$, we multiply through by the transpose of the eigenvector and solve for the eigenvalue:

$$\hat{\lambda}_1 \simeq \frac{\hat{\mathbf{v}}_1^\top \mathbf{A}\hat{\mathbf{v}}_1}{\hat{\mathbf{v}}_1^\top \hat{\mathbf{v}}_1} = \frac{\langle \mathbf{A}\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_1 \rangle}{\langle \hat{\mathbf{v}}_1, \hat{\mathbf{v}}_1 \rangle} \tag{10}$$

  where $\langle \star, \star \rangle$ denotes an inner product While simple to implement, the power iteration method may exhibit slow convergence, mainly when the largest eigenvalue is close in magnitude to other eigenvalues, i.e., $|\lambda_1|/|\lambda_2| \sim 1$.

The power iteration method has several notable applications, particularly in the fields of machine learning and graph theory. Arguablbly the most famous application is the Google PageRank algorithm. Google's PageRank algorithm, which uses power iteration, utilizes the dominant eigenvalue and its corresponding eigenvector to assess the importance of web pages within a network. Specifically, the algorithm constructs a link matrix that represents the connections between pages, where the dominant eigenvector associated with the maximum eigenvalue reflects the relative importance of each page, with its entries normalized to sum to 1, thus providing a ranking of pages based on their likelihood of being visited over time. Pusedo code for the power iteration method is shown in Algorithm 1.

Algorithm 1 Power Iteration Method

Input: Matrix $A \in \mathbb{R}^{n \times n}$, initial vector $x_0 \in \mathbb{R}^n$, tolerance ϵ, maximum iterations N
Normalize the initial vector: $x_0 \leftarrow \frac{x_0}{\|x_0\|}$
for $k = 1, 2, \ldots, N$ **do**
    Compute the matrix-vector product: $y_k \leftarrow Ax_{k-1}$
    Normalize the resulting vector: $x_k \leftarrow \frac{y_k}{\|y_k\|}$
    Compute the Rayleigh quotient: $\lambda_k \leftarrow x_k^\top Ax_k / (x_k^\top x_k)$
    if $\|x_k - x_{k-1}\| < \epsilon$ **then**
        break
    end if
end for
Set $v \leftarrow x_k$ and $\lambda \leftarrow \lambda_k$
return λ, v

3.2 QR decomposition

The QR algorithm relies on the QR decomposition of a matrix, which is a factorization of a matrix into an orthogonal matrix and an upper triangular matrix. QR decomposition, originally developed by Francis in the early 1960s (1, 2), factors a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ into the product of an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ and an upper triangular matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$:

$$\mathbf{A} = \mathbf{QR} \tag{11}$$

where $\mathbf{Q}^\top = \mathbf{Q}^{-1}$. The QR decomposition is useful for solving linear systems of equations, computing the eigenvalues and eigenvectors of a matrix, and for finding the least squares solution of an overdetermined system of equations.

Gram-Schmidt Orthogonalization

To compute the QR decomposition of a matrix, we can use the Gram-Schmidt process. In principle, Gram-Schmidt orthogonalization generates a set of mutually orthogonal vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ starting from a set of linearly independent vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ using the procedure:

$$\mathbf{q}_k = \mathbf{x}_k - \sum_{i=1}^{k-1} \frac{\langle \mathbf{q}_i, \mathbf{x}_k \rangle}{\langle \mathbf{q}_i, \mathbf{q}_i \rangle} \mathbf{q}_i, \qquad k = 1, \ldots, n \tag{12}$$

where $\langle \cdot, \cdot \rangle$ denotes an inner product. To compute the columns of $\mathbf{Q}$ we set $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ equal the columns of $\mathbf{A}$ and follow the Gram-Schmidt procedure. The computation of $\mathbf{R}$ is made simple by exploiting the orthogonality of $\mathbf{Q}$, i.e., $\mathbf{Q}^{-1} = \mathbf{Q}^\top$ which yields:

$$\mathbf{R} = \mathbf{Q}^\top \mathbf{A} \tag{13}$$

The computation of the QR decomposition using the Gram-Schmidt orthogonalization procedure is shown in Algorithm 2.

Algorithm 2 QR decomposition using Classical Gram-Schmidt Orthogonalization

Input: Matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$
Output: Orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ and upper triangular matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$
$n, m \leftarrow$ dimensions of $\mathbf{A}$
$\mathbf{Q} \leftarrow$ zeros matrix of size $n \times m$
for $j = 1$ to m **do** ▷ Loop over the columns of $\mathbf{A}$
    $\mathbf{v}_j \leftarrow \mathbf{A}_j$ ▷ Get the jth column of $\mathbf{A}$
    for $k = 1$ to $j - 1$ **do**
        $\mathbf{q}_k \leftarrow \mathbf{Q}_k$ ▷ Get the kth column of $\mathbf{Q}$
        $\mathbf{v}_j \leftarrow \mathbf{v}_j - (\mathbf{q}_k^\top \mathbf{A}_j)\mathbf{q}_k$ ▷ Orthogonalize the jth column of $\mathbf{A}$
    end for
    $\mathbf{Q}_j \leftarrow \mathbf{v}_j / ||\mathbf{v}_j||_2$ ▷ Normalize the jth column of $\mathbf{Q}$
end for
$\mathbf{R} \leftarrow \mathbf{Q}^\top \mathbf{A}$ ▷ Compute the upper triangular matrix $\mathbf{R}$
return $\mathbf{Q}$, $\mathbf{R}$

The cost of the Gram-Schmidt orthogonalization procedure is $\mathcal{O}(nm^2)$ floating point operations (flops) for an $n \times m$ matrix $\mathbf{A}$ (3). However, in practice, the Gram-Schmidt orthogonalization procedure as shown *will often fail to generate mutually orthogonal vectors* because of rounding errors; we will address this issue subsequently when we discuss the *Modified Gram-Schmidt Orthogonalization* procedure which is shown in Algorithm 3.

Algorithm 3 QR decomposition using Modified Gram-Schmidt Orthogonalization

Input: Matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$
Output: Orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{n \times n}$ and upper triangular matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$
$n, m \leftarrow$ dimensions of $\mathbf{A}$
for $j = 1$ to m **do** ▷ Loop over the columns of $\mathbf{A}$
    $\mathbf{v}_j \leftarrow \mathbf{A}_j$ ▷ Initialize v_j to the jth column of $\mathbf{A}$
end for
for $j = 1$ to m **do** ▷ Loop over the columns of $\mathbf{A}$
    $\mathbf{q}_j \leftarrow \mathbf{v}_j / ||\mathbf{v}_j||$ ▷ Set the jth column of $\mathbf{Q}$ to the normalized jth column of $\mathbf{A}$
    for $k = j + 1$ to m **do**
        $\mathbf{v}_k \leftarrow \mathbf{v}_k - (\mathbf{q}_j^\top \mathbf{v}_k)\mathbf{q}_k$ ▷ Orthogonalize the kth column of $\mathbf{A}$
    end for
end for
$\mathbf{R} \leftarrow \mathbf{Q}^\top \mathbf{A}$ ▷ Compute the upper triangular matrix $\mathbf{R}$
return $\mathbf{Q}$, $\mathbf{R}$

Aside: Solution of Linear Systems of Equations

In addition to computing the eigendecomposition of a matrix, another handy applications of QR decomposition is to solve the linear systems of equations $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$. Once $\mathbf{A}$ has been factored into the prodict of $\mathbf{Q}$ and $\mathbf{R}$:

$$\mathbf{QR} \cdot \mathbf{x} = \mathbf{b} \tag{14}$$

we multiply both sides by $\mathbf{Q}^\top$:

$$\mathbf{R} \cdot \mathbf{x} = \mathbf{Q}^\top \cdot \mathbf{b} \tag{15}$$

where $\mathbf{Q}^\top \cdot \mathbf{Q} = \mathbf{I}$. Because $\mathbf{R}$ is an upper triangular matrix, we can solve the linear system of equations using back substitution. Back substitution is an algorithm for solving a system of linear equations whose matrix is upper triangular. Suppose we have an $n \times n$ system ($n \geq 2$) of equations which is upper triangular and non-singular of the form:

$$\mathbf{U}\mathbf{x} = \mathbf{b}$$

where $\mathbf{U}$ is an upper triangular matrix and $\mathbf{b}$ is a column vector. Then, the solution of to this system is given by:

$$\begin{aligned} x_n &= \frac{b_n}{u_{nn}} \\ x_i &= \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) \qquad i = n-1, \ldots, 1 \end{aligned}$$

where $u_{ii} \neq 0$. Back substitution requires $\mathcal{O}(n^2)$ floating point operations (flops).

3.3 The QR Iteration Algorithm

To compute the eigendecomposition of a matrix using QR decomposition, we first factorize the matrix $\mathbf{A}$ into the product of $\mathbf{Q}$ and $\mathbf{R}$, then we use the QR-iteration algorithm. The QR-iteration algorithm is an iterative method that computes the eigenvalues and eigenvectors of a matrix by repeatedly applying the QR decomposition to the matrix. The QR iteration algorithm estimates the eigenvalues and eigenvectors of a square matrix.

The QR-iteration algorithm consists of two phases:

- **Phase 1: Eigenvalues**. Let $\mathbf{A} \in \mathbb{R}^{n \times n}$. Then, starting with $k = 0$, we compute the QR decomposition of the matrix $\mathbf{A}_k$:

  $$\mathbf{A}_{k+1} \leftarrow \mathbf{R}_k \mathbf{Q}_k = \mathbf{Q}_k^\top \mathbf{A}_k \mathbf{Q}_k \qquad k = 0, 1, \ldots$$

  where $\mathbf{Q}_k$ is an orthogonal matrix, i.e., $\mathbf{Q}_k^\top = \mathbf{Q}_k^{-1}$ and $\mathbf{R}_k$ is an upper triangular matrix. As $k \to \infty$, the matrix $\mathbf{A}_k$ converges to a triangular matrix with the eigenvalues listed along the diagonal.
- **Phase 2: Eigenvectors**. We compute the eigenvectors associated with each eigenvalue by solving the homogenous system of equations:

  $$(\mathbf{A} - \lambda_j \mathbf{I}) \cdot \mathbf{v}_j = \mathbf{0}$$

  where $\mathbf{I}$ is the identity matrix.

4 Summary and Conclusions

In this lecture, we discussed the eigendecomposition of a matrix and how it can be used to analyze data and systems in the context of unsupervised machine learning. Eigendecomposition gives us a way to decompose a matrix into its constituent parts, the eigenvectors and eigenvalues, which can be used to understand the structure of the data or the system represented by the matrix. We introduced the QR decomposition, which is a factorization of a matrix into an orthogonal matrix and an upper triangular matrix. We introduced the Gram-Schmidt orthogonalization procedure, which is used to compute the QR decomposition of a matrix.

Gram-Schmidt orthogonalization generates a set of mutually orthogonal vectors starting from a set of linearly independent vectors. However, the Gram-Schmidt orthogonalization procedure may fail to generate mutually orthogonal vectors because of rounding errors. To fix this issue, we introduced the Modified Gram-Schmidt orthogonalization procedure, which is a more stable version of the Gram-Schmidt orthogonalization procedure. Finally, we introduced the QR iteration algorithm, which is an iterative method that computes the eigenvalues and eigenvectors of a matrix. The QR iteration algorithm estimates the eigenvalues and eigenvectors of a square matrix by repeatedly applying the QR decomposition to the matrix.

References

1. Francis JGF. The QR Transformation A Unitary Analogue to the LR Transformation—Part 1. The Computer Journal. 1961;4(3):265–271. doi:10.1093/comjnl/4.3.265.

2. Francis JGF. The QR Transformation—Part 2. The Computer Journal. 1962;4(4):332–345. doi:10.1093/comjnl/4.4.332.

3. Golub GH, van Loan CF. Matrix Computations. 4th ed. JHU Press; 2013. Available from: `http://www.cs.cornell.edu/cv/GVL4/golubandvanloan.htm`.