Eigenvalues and Eigenvectors

Abstract

Eigenvalues and eigenvectors are fundamental concepts in linear algebra with implications across mathematics, data science, and engineering. The following strategies are discussed: i) Introduction to eigenvalues and eigenvectors, their definitions, properties and practical examples ii) Diagonolization iii) The Jordan form iv) Matrix powers v) Markov chains vi) PCA (principal component analysis) vii) SVD (singular value decomposition)

---

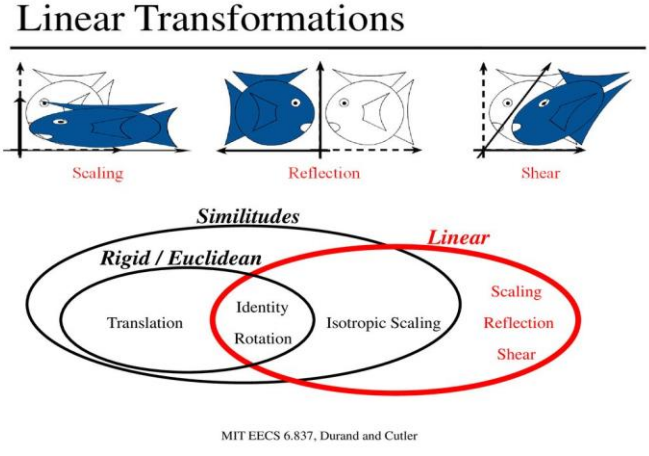
1. Introduction

First of all, the greek letter *λ (lambda)* is used for eigenvalues in most literature. Etymologically, "eigenvalue" and "eigenvector" are derived from the German word *"eigenwert*", which means *"proper value".* Moreover, the concept of eigenvalues was initially developed by Jean d'Alambert.

Pic1. *Jean-Baptiste le Rond d'Alembert*

He was was a french mathematician, mechanician, physicist, philosopher, and music theorist. His mother left him on the steps of a local church and he was consequently sent to a home for orphans. Moreover, his father died when he was nine years old. For most of his life he worked for the Paris Academy of Science and the French Academy.

Basically, eigenvalues and eigenvectors provide insight into how a matrix behaves as a *linear transformation*, revealing its essential geometric and scaling properties.



Pic2. *linear transformations*

Let's suppose $\underline{A}$ is *a square matrix,* where $\underline{A}$ has dimensions *n×n* (the same number of rows and columns). In addition, a non-zero vector (it is not the zero vector) $\underline{v}$ is called **eigenvector** if there is a scalar $\underline{\lambda}$ that is called **eigenvalue** of $\underline{A}$.

- Main formula: $Av = \lambda v$
- Formula by moving terms to the same side of equation: $Av - \lambda v = 0$ -> $(A - \lambda I)v = 0$

$$Square\ Matrix\ M = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$$

Pic3. *Square matrix*

Properties of eigenvalues and eigenvectors

1. if λ is an eigenvalue of a square matrix A with an eigenvector v then each non-zero scalar multiplication of v is also eigenvector belonging to λ. For instance, 2v, 200v, 100v, etc.

2. if A is an n×n matrix with an eigenvalue (*λ*), then the set S of all eigenvectors of A belonging to λ together with the zero vector, O, is a subspace R^n.

3. if A is invertible, which means it has an inverse, λ = 0 is not an eigenvalue of it. Thus, if λ = 0 is an eigenvalue of A then A has no inverse.

4. If k is natural number then λ^k is an eigenvalue of the matrix A^k with the same eigenvector v. On the other hand, if A is invertible then the eigenvalue of the inverse matrix A^{-1} is λ^{-1} with the same eigenvector v.

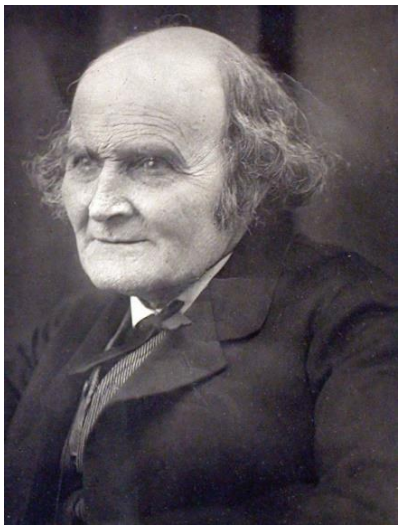
5. If A is a square matrix with eigenvalues $\lambda_1, \lambda_2, \lambda_3 ... \lambda_k$. Then;

a) The determinant (det(A)): $\lambda_1 * \lambda_2 * \lambda_3 * ... * \lambda_k$

b) The trace (tr(A)): $\lambda_1 + \lambda_2 + \lambda_3 + ... + \lambda_k$

6. If A is a square matrix (*n×n*) with distinct eigenvalues $\lambda_1, \lambda_2, \lambda_3 ... \lambda_k$ and corresponding eigenvectors $v_1, v_2, v_3 ... v_k$ (where $1 \leq k \leq n$). Hence, these eigenvectors are linearly independent. Furthermore, this proposition is vital for the proof of *Cayley-Hamilton* theorem.

Cayley-Hamilton: every square matrix A over a commutative ring (such as the real or complex numbers or the integers) is a root of the characteristic equation, that is p(A) = O where p represents the characteristic polynomial. For example, a 1 × 1 matrix A = (a), the characteristic polynomial is given by $p(\lambda) = \lambda - a$, *and so* $p(A) = (a) - a(1) = 0$ is trivial.



Pic4. *Arthur Cayley*

He was a British mathematician who worked mostly on algebra. He helped found the modern British school of pure mathematics, and was a professor at Trinity College, Cambridge. Published nearly 1000 papers, making him one of the most productive mathematicians of his time. As a result, his research laid the foundation for modern linear algebra used in machine learning, physics, and engineering today.

Pic5. *Sir William Rowan Hamilton*

He was an Irish mathematician and physicist who made major contributions to abstract algebra, classical mechanics, and optics. His career included the analysis of geometrical optics, Fourier analysis, and quaternions. Additionally, Hamilton was Andrews Professor of Astronomy at Trinity College Dublin, also the 3rd director of Dunsink Observatory. The Hamilton Institute at Maynooth University is named after him. Throughout his life he had problems with alcohol and love. He fell in love with Catherine but due to unfortunate circumstances his ended up marrying Helen which he regretted for the rest of his life.

Example 1 (Compute eigenvalues):

Suppose that A = $\begin{bmatrix} 6 & 2 \\ 3 & 5 \end{bmatrix}$, and we will compute all eigenvalues of A.

Solution:

Firstly, we have to find $det(A-\lambda I)$. Foremost, our matrix is a 2×2 matrix, and we can find determinant via *theorem a (determinant of 2×2 matrices)*.

***Theorem a**: $det(\begin{bmatrix} a & b \\ c & d \end{bmatrix})$ = ad – bc*

det(A - λI) => $\det(\begin{bmatrix} 6-\lambda & 2 \\ 3 & 5-\lambda \end{bmatrix})$ => (6-λ)(5 - λ) – 6 => λ^2 - 11λ + 24 = 0
Then, we are able to solve this problem with 2 ways,

1st:

λ2 - 11λ + 24 = 0⇔ (λ - 8)(λ - 3) = 0 ⇔ ***λ = {8, 3}***

2nd (using quadratic formula):

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In this case, λ = (11 $\pm$ 5) / 2 => ***λ = 8 and λ = 3***

Example 2 (Compute eigenvectors):
We already found eigenvalues. For this reason, we can find eigenvectors corresponding to these eigenvalues, using (A - 6I)v = 0 and (A + 2I)v = 0 linear systems.

Solution:
λ = 8:

A = $\begin{bmatrix} -2 & 2 \\ 5 & -3 \end{bmatrix}$

Linear system,

-2v1 + 2v2 = 0 (R1)
5v1 -3v2 = 0 (R2)
*operation: ((R1/2) * 3) + R2*
-3v1 + 3v2 = 0 => v2 = v1 => v = (v1, v1) ⇔ ***v = v1(1, 1)***
~~2v1 = 0~~

λ = 3:

A = $\begin{bmatrix} 3 & 2 \\ 5 & 2 \end{bmatrix}$

Linear system,

3v1 + 2v2 = 0 (R1)

5v1 + 2v2 = 0 (R2)

operation: R1 - R2

3v1 + 2v2 = 0 => v2 = -3/2v1 => v = (2v1, -3v1) ⇔ ***v = v1(2, -3)***

~~-2v1 = 0~~

Example 3 (Compute eigenvalues in python):

```
[6] import numpy as np

    A = np.array([
        [6, 2],
        [3, 5]])

    eigenvalues = np.linalg.eigvals(A)
    print(f"Eigenvalues (λ): {eigenvalues}")

    Eigenvalues (λ): [8. 3.]
```

Pic6. Eigenvalues in python

Results (output) are same, 8 and 3.

Example 4 (Compute eigenvectors in python):

```
import numpy as np
A = np.array([[6, 2], [3, 5]])
eigenvalues, eigenvectors = np.linalg.eig(A)
print("Eigenvectors:\n", eigenvectors)

Eigenvectors:
 [[ 0.70710678 -0.5547002 ]
 [ 0.70710678  0.83205029]]
```

Pic7. *Eigenvectors in python*

In this code, NumPy normalizes eigenvectors to have unit length. Consequently, results look different.

Notes:

1- *Numpy:* a Python library for numerical computing.
2- *Array:* a container for storing homogeneous data.
3- *Numpy.linalg:* provides functions for linear algebra operations, including;
    a. Computing eigenvalues & eigenvectors *(np.linalg.eig)*
    b. Finding determinants *(np.linalg.det)*

c. Computing eigenvalues*(np.linalg.eigvals)*
d. Computing matrix inverses *(np.linalg.inv)*

---

2. Diagonalization

Suppose the n by n matrix *A* has n linearly independent eigenvectors. If these eigenvectors are columns of a matrix *P*. Then $P^{-1}AP$ is a diagonal matrix *D*. Also, the eigenvalues of *A* are on the diagonal of *D*.

Diagonalization: $P^{-1}AP = D = \begin{bmatrix} \lambda 1 & \ddots \\ \ddots & \lambda n \end{bmatrix}$

Theorem b: *If an n by n matrix A is diagonalizable, it has n linearly independent eigenvectors. On the contrary, if the matrix A has no repeated eigenvalues which means they are distinct then its eigenvectors are linearly independent. For that reason, any matrix with distinct eigenvalues can be diagonalized.*

Example 5:

$$A = \begin{bmatrix} 6 & 2 \\ 3 & 5 \end{bmatrix} \quad \lambda = \{8, 0\}, \quad v1 = (1, 1), v2 = (2, -3)$$

$$D = \begin{bmatrix} 8 & 0 \\ 0 & 3 \end{bmatrix} \quad P = \begin{bmatrix} 1 & 2 \\ 1 & -3 \end{bmatrix} \quad P^{-1} = \begin{bmatrix} 3/5 & 2/5 \\ 1/5 & -1/5 \end{bmatrix}$$

1) $P * D = \begin{bmatrix} 1 & 2 \\ 1 & -3 \end{bmatrix} \begin{bmatrix} 8 & 0 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 8 & 6 \\ 8 & -9 \end{bmatrix}$
2) $P * D * P^{-1} = \begin{bmatrix} 8 & 6 \\ 8 & -9 \end{bmatrix} \begin{bmatrix} 3/5 & 2/5 \\ 1/5 & -1/5 \end{bmatrix} = \begin{bmatrix} 6 & 2 \\ 3 & 5 \end{bmatrix} \Leftrightarrow A$

As a result, A is diagonalizable, and the process of converting any square matrix into a diagonal matrix is called diagonalization.

Example 6 (in python):

```python
import numpy as np

A = np.array([[6, 2], [3, 5]])
P = np.array([[1, 2], [1, -3]])
D = np.array([[8, 0], [0, 3]])
P_inverse = np.linalg.inv(P)

# Verify A = PDP^(-1)
A_ = P @ D @ P_inverse
print(f"Original matrix: \n {A}")
print(f"Computed matrix: \n {A_}")
```

Pic8. Diagonalization in python

In there, **"@"** is used for matrix multiplication (row-by-column multiplication) due to for the fact that **"*"** is element-wise multiplication operator (it multiplies corresponding elements one by one).

Output:

```
Original matrix:
 [[6 2]
 [3 5]]
Computed matrix:
 [[6. 2.]
 [3. 5.]]
```

***Pic9**. Output of code*

The Jordan Form

The Jordan form, also known as a Jordan canonical form of a matrix, is a special form that simplifies matrix analysis, particularly for a defective, nondiagonalizable matrices. It consists of Jordan blocks, which are nearly diagonal matrices. For each missing eigenvector, there will be a 1 just above its main diagonal. The eigenvalues appear on the diagonal because J is a triangular, and distinct eigenvalues can always be decoupled.

Jordan form: A = PJP^{-1}: $\begin{bmatrix} \lambda 1 & 1 & 0 \\ 0 & \lambda 2 & 1 \\ 0 & 0 & \lambda 3 \end{bmatrix}$

Example 7 (Jordan form in python):

```python
import sympy as sp

A = sp.Matrix([[5, 4, 2], [0, 3, -2], [0, 1, 1]])
jordan, p = A.jordan_form()

print(f"Jordan Form:\n {jordan}")
print(f"Transformation Matrix:\n {p}")
```

***Pic 10**. The Jordan Form in python*

Notes:

1- *Sympy:* a Python library for symbolic mathematics.
2- *Sympy.Matrix*: a class is used to define matrices symbolically.
3- *Sympy.Matrix.jordan_form():* a method calculates the Jordan form and the transformation matrix to change the matrix to its Jordan form.

Matrix Powers

Shortly, matrix powers refer to the repeated multiplication of a matrix by itself. Hence;

$A^k = (PDP^{-1})*(PDP^{-1})*..{}_{(k\ times)}..*(PDP^{-1}) = PD^kP^{-1}$ *(for all* $k \geq 0$ *and* $PP^{-1} = 1$*).*

We can apply this technique to come up with explicit formulas for powers of diagonalizable matrices.

Example 8 (find an explicit formula for A^k):

$$\mathbf{A} = \begin{bmatrix} 6 & 2 \\ 3 & 5 \end{bmatrix} \ \lambda = \{8, 0\},\ \textbf{v1 = (1, 1), v2 = (2, -3)}$$

$$\mathbf{D} = \begin{bmatrix} 8 & 0 \\ 0 & 3 \end{bmatrix} \ \mathbf{P} = \begin{bmatrix} 1 & 2 \\ 1 & -3 \end{bmatrix} \ \mathbf{P^{-1}} = \frac{1}{5}\begin{bmatrix} 3 & 2 \\ 1 & -1 \end{bmatrix}$$

$A^k = PD^kP^{-1}$

$$= \frac{1}{5}\begin{bmatrix} 1 & 2 \\ 1 & -3 \end{bmatrix}\begin{bmatrix} (8)^k & 0 \\ 0 & (3)^k \end{bmatrix}\begin{bmatrix} 3 & 2 \\ 1 & -1 \end{bmatrix}$$

$$= \frac{1}{5}\begin{bmatrix} 1 & 2 \\ 1 & -3 \end{bmatrix}\begin{bmatrix} 3*(8)^k & 2*(8)^k \\ (3)^k & -(3)^k \end{bmatrix}$$

$$= \frac{1}{5}\begin{bmatrix} 3*((8)^k)+2*(3)^k & 2*((8)^k)-2*(3)^k \\ 3*((8)^k)-3*(3)^k & 2*((8)^k)+3*(3)^k \end{bmatrix}$$

⇔ formula for $\mathbf{A^k} = \frac{\mathbf{1}}{\mathbf{5}}\left[(\mathbf{8})^{\boldsymbol{k}}\begin{bmatrix}\mathbf{3} & \mathbf{2}\\ \mathbf{3} & \mathbf{2}\end{bmatrix} + (\mathbf{3})^{\boldsymbol{k}}\begin{bmatrix}\mathbf{2} & \mathbf{-2}\\ \mathbf{-3} & \mathbf{3}\end{bmatrix}\right]$

In conclusion, matrix powers are especially useful in Markov chains.

Markov chains

Markov chains, named after Andrey Markov, are stochastic models because they simulate real-life events that are random by nature (stochastic). In other words, It describes random processes where systems move between states, and a new state only depends on the current state, not the entire sequence of steps. This property is known as “Markov Property” or “Memorylessness”. “Memorylessness” makes it faster to run on computers, and powerful to study random processes and make prediction based on current conditions.

***Pic11**. Andrey Andreyevich Markov*

He was a Russian mathematician best known for his work on stochastic processes. He was also a strong chess player. Markov and his younger brother Vladimir Andreyevich Markov (1871–1897) proved *the Markov brothers' inequality*. His son, Andrey Andreyevich Markov (1903–1979), was also a notable mathematician, making contributions to constructive mathematics and recursive function theory.

Types of Markov Chains;

1) *Discrete-Time Markov Chains (DTMCs)*

   In DTMCs, the system changes state at specific time steps. They are called discrete considering that the state transitions occur at distinct, separate time intervals. They are used in *queuing theory* (study of the behavior of waiting lines), genetics, and economics.

2) *Continuous-Time Markov Chains (CTMCs)*

CTMCs differ from DTMCs in that state transitions can occur at any continuous time point, not at fixed intervals. This is important in chemical reactions and reliability engineering.

3) Reversible Markov Chains

Reversible Markov chains are special. The process of state change is the same whether the direction is forwards or backwards. They are widely used in statistical physics and economics.

4) Doubly Stochastic Markov Chains

Doubly stochastic Markov chains are defined by a transition probability matrix. In the matrix, the sum of the probabilities in each row and each column equals to 1. It means each row and each column represent a valid probability distribution. It is crucial in quantum computing and statistical mechanics.

Example 9 (Markov chains in python):

```
states = ["Room A", "Room B", "Room C"]
transition_matrix = np.array([
    [0.25, 0.5, 0.25],# From A -> 25% stay, 50% to B, 25% to C
    [0.35, 0.30, 0.35],  # From B -> 35% to A, 30% stay, 35% to C
    [0.0, 0.7, 0.3]   # From C -> 0% to A, 70% to B, 30% stay
])

num_steps = 10  # Number of steps
current_state = 0  # Start in first room (A)
state_sequence = [states[current_state]]

# Markov Chain
for i in range(num_steps):
    current_state = np.random.choice([0, 1, 2], p = transition_matrix[current_state])
    state_sequence.append(states[current_state])

# Print the room sequence
print("Room sequence: ", " => ".join(state_sequence))
```

Pic12. *Markov chains in python*

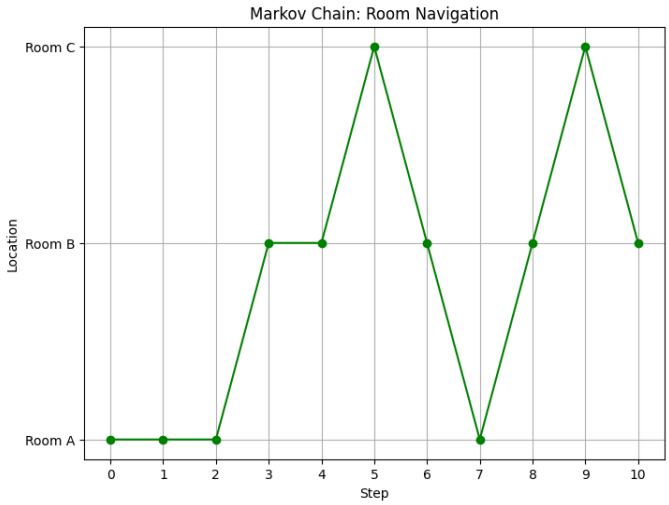
Output of ***Example 9**** *:*

Room sequence: Room A => Room A => Room A => Room B => Room B => Room C => Room B => Room A => Room B => Room C => Room B

```python
# Visualization
plt.figure(figsize=(8, 6))
plt.plot(range(len(state_sequence)), state_sequence, marker = 'o', c = 'g')
plt.xticks(range(len(state_sequence)), labels = range(len(state_sequence)))
plt.yticks([0, 1, 2], labels = states)
plt.title("Markov Chain: Room Navigation")
plt.grid()
plt.show()
```

Pic13. *Visualization of Markov chains in python*

Output of ***Pic13****:*



Pic14*. Output of visualization*

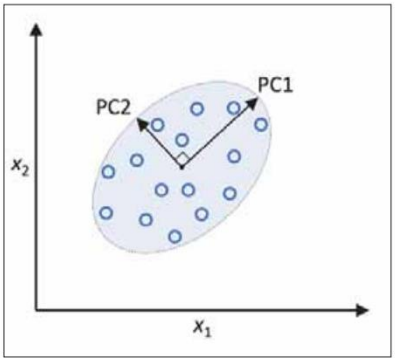
---

3. Dimensionality Reduction

High-dimensional data can lead to computational inefficiency, increased noise, and challenges in visualization. Dimensionality reduction techniques seek to preserve the most important information while simplifying the data's complexity.

a. PCA (Principal Component Analysis)

PCA is a popular dimensionality reduction algorithm that relies on eigenvalues and eigenvectors. It helps identify patterns in data based on the correlation between features. In short, PCA aims to find the directions of maximum variance in high-dimensional data and projects it onto a new subspace with the same or fewer dimensions than the original.



Pic15. *x1 and x2 are the original features, and PC1 and PC2 are the principal companents*

Example 10 (PCA using scikit-learn):

The code applies PCA to reduce the dimensionality of the dataset down to two dimensions.

Pic16*. PCA in python*

```
# Model
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
pca.fit(x_scaled)
x_pca = pca.fit_transform(x_scaled)

print("Original shape: {}".format(str(x_scaled.shape)))
print("Reduced shape: {}".format(str(x_pca.shape)))
```

```
Original shape: (150, 4)
Reduced shape: (150, 2)
```

Additionally, the explained variance ratio of each principal component indicates the proportion of the dataset's variance that is captured by each principal component.

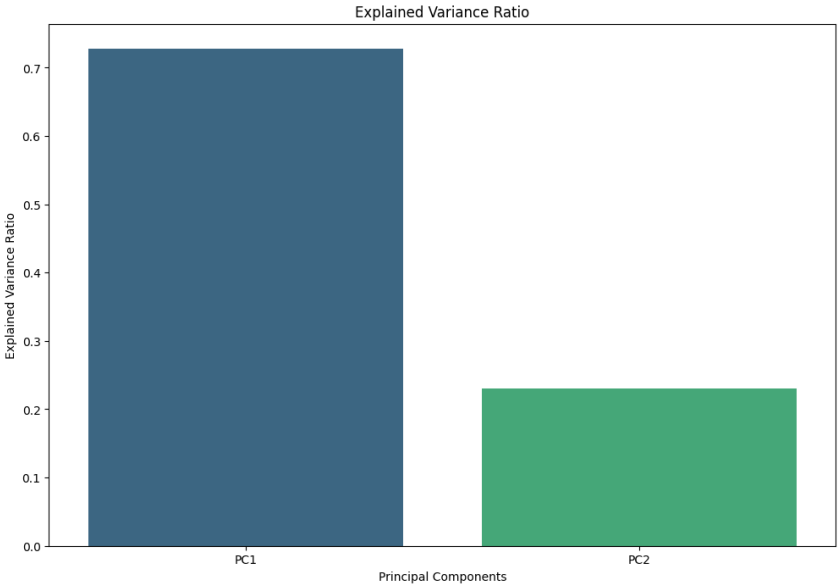
Example 11:

Pic17. *Explained variance ratio code and output ->* ***([0.72770452 0.23030523])***

```
print(f"Explained Variance Ratio {pca.explained_variance_ratio_}")

plt.figure(figsize = (12, 8))
sns.barplot(x = ["PC1", "PC2"], y = pca.explained_variance_ratio_, palette = "viridis")
plt.title("Explained Variance Ratio")
plt.xlabel("Principal Components")
plt.ylabel("Explained Variance Ratio")
plt.show()
```

```
Explained Variance Ratio [0.72770452 0.23030523]
```



Pic18. *Output of pic17*

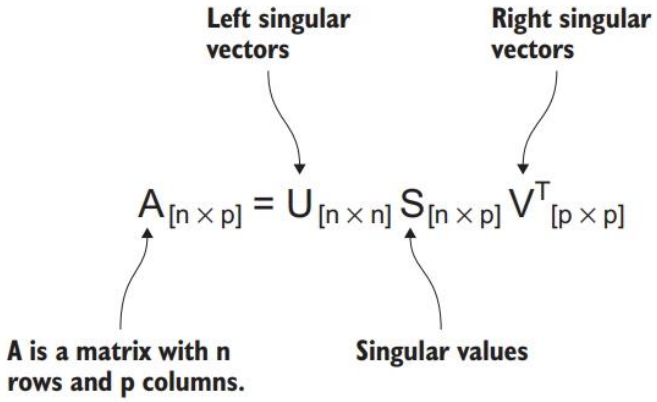
Simply process flow:

1) Normalize the data to have zero mean and unit variance.
2) Calculate covariance matrix the standardized data.
3) Compute eigenvectors and eigenvalues of the covariance matrix.
4) Sort eigenvalues in order and choose the top-k eigenvalues to form principal components.
5) Project the original data onto the principal components to create a lower-dimensional representation.

b. SVD (Singular Value Decomposition)

SVD is a standard matrix factorization technique that can decompose the training set matrix **A** into the matrix multiplication of three matrices $\mathbf{U\Sigma(S)V^T}$, where **V** contains the unit vectors that define all the principal components. Briefly;

- *U* contains orthogonal columns that represent the left singular vectors.
- *Σ(S)* is a diagonal matrix containing the singular values.
- V^T contains orthogonal rows representing the right singular vectors.



Pic19. *Matrix multiplication of three matrices*

$$\mathbf{V} = \begin{pmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{pmatrix}$$

Pic20. *Principal components matrix*

Example 12 (SVD in python):

```python
import numpy as np

A = np.array([[1, 2, 5],
              [4, 3, -2]])

U, S, Vt = np.linalg.svd(A)

print("U:\n", U)
print("S:\n", S)
print("V^T:\n", Vt)
```

Pic21. *SVD in python*

```
U:
 [[1. 0.]
 [0. 1.]]
S:
 [5.47722558 5.38516481]
V^T:
 [[ 0.18257419  0.36514837  0.91287093]
 [ 0.74278135  0.55708601 -0.37139068]
 [-0.64416033  0.74586985 -0.16951588]]
```

Pic22. *Output of SVD code in pic20*

Example 13 (SVD using scikit-learn):

```
from sklearn.decomposition import TruncatedSVD

x = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

svd = TruncatedSVD(n_components=2)
x_svd = svd.fit_transform(x)

print("Original shape: {}".format(str(x.shape))) # output: (3, 3)
print("Reduced shape: {}".format(str(x_svd.shape))) # output: (3, 2)
```

Pic23. *Truncated SVD*

Notes:

1- *Sklearn:* a Python library for machine learning.
2- *Sympy .decomposition*: a submodule in sklearn that provides techniques for dimensionality reduction.
3- *TruncatedSVD:* a dimensionality reduction technique that performs Singular Value Decomposition (SVD).

---

4. Summary

In a nutshell, eigenvalues and eigenvectors describe how matrices map space, the foundation of much mathematical and machine learning usage. Jordan form puts the matrix back together in simpler blocks when diagonalization is impossible. Matrix powers are vital in iterative models. Markov chains describe probabilistic transitions, forecasting system behavior based solely on the current state. PCA reduces high-dimensional data by mapping onto principal components that come from eigenvectors. SVD extends this by decomposing any matrix into orthogonal factors, allowing for uses such as image compression and recommendation systems.

---

5. References

Johnston, Nathaniel. Introduction to Linear and Matrix Algebra. Springer, 2021.

Singh, Kuldeep. Linear Algebra Step by Step. Oxford University Press, 2014.

Strang, Gilbert. Linear Algebra and Its Applications. 4th ed., Cengage Learning, 2006.

Müller, Andreas C., and Sarah Guido. Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly, 2017.

Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 3rd ed., O'Reilly, 2023.

Brink, Henrik, Joseph W. Richards, and Mark Fetherolf. Real-World Machine Learning. Manning Publications, 2017.

J. Ström, K. Åström, and T. Akenine-Möller, Immersive Linear Algebra

Markov chains- freecodecamp.

Jean d'Alambert - wikipedia.

Eigenvalues and eigenvectors - wikipedia.

Cayley – Hamilton theorem - wikipedia.

William Rowan Hamilton - wikipedia.

Arthur Cayley - wikipedia.

Andrey Markov - wikipedia.

PCA and SVD - medium