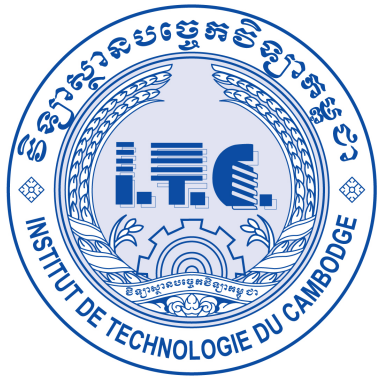

CHAPTER V
Eigendecomposition

PHOK Ponna and NEANG Pheak

Institute of Technology of Cambodia
Department of Applied Mathematics and Statistics (AMS)

2022–2023

Contents

Contents

Introduction

Matrix decompositions are a useful tool for reducing a matrix to their constituent parts in order to simplify a range of more complex operations. Perhaps the most used type of matrix decomposition is the eigendecomposition that decomposes a matrix into eigenvectors and eigenvalues. This decomposition also plays a role in methods used in machine learning, suchas in the Principal Component Analysis method or PCA. After completing this tutorial, you will know:

- What an eigendecomposition is and the role of eigenvectors and eigenvalues.
- How to calculate an eigendecomposition in Python with NumPy.
- How to confirm a vector is an eigenvector and how to reconstruct a matrix from eigenvectors and eigenvalues.

Contents

Eigendecomposition of a matrix is a type of decomposition that involves decomposing a square matrix into a set of eigenvectors and eigenvalues.
One of the most widely used kinds of matrix decomposition is called **eigendecomposition**, in which we decompose a matrix into a set of eigenvectors and eigenvalues.
A vector v is an eigenvector of a matrix A if it satisfies the following equation

$$A.v = \lambda.v$$

This is called the eigenvalue equation, where A is the parent square matrix that we are decomposing, v is the **eigenvector** of the matrix, and λ is the lowercase Greek letter lambda and represents the **eigenvalue** scalar.

A matrix could have one eigenvector and eigenvalue for each dimension of the parent matrix. Not all square matrices can be decomposed into eigenvectors and eigenvalues, and some can only be decomposed in a way that requires complex numbers. The parent matrix can be shown to be a product of the eigenvectors and eigenvalues.

$$A = Q.\Lambda.Q^T$$

Where Q is a matrix comprised of the eigenvectors, Λ is the uppercase Greek letter lambdaand is the diagonal matrix comprised of the eigenvalues, and Q^T is the transpose of the matrix comprised of the eigenvectors.
Eigendecomposition can also be used to calculate the principal components of a matrix in the Principal Component Analysis method or PCA that can be used to reduce the dimensionality of data in machine learning.

Contents

An eigendecomposition is calculated on a square matrix using an efficient iterative algorithm, of which we will not go into the details. Often an eigenvalue is found first, then an eigenvector is found to solve the equation as a set of coefficients. The eigendecomposition can be calculated in NumPy using the eig() function. The example below first defines a 3×3 square matrix. The eigendecomposition is calculated on the matrix returning the eigenvalues and eigenvectors.

```
# eigendecomposition
from numpy import array
from numpy.linalg import eig
# define matrix
A = array([
  [1, 2, 3],
  [4, 5, 6],
   [7, 8, 9]])
print(A)
# factorize
values, vectors = eig(A)
```

```
print(values)
print(vectors)
```

We can confirm that a vector is indeed an eigenvector of a matrix. We do this by multiplying the candidate eigenvector by the value vector and comparing the result with the eigenvalue. First, we will define a matrix, then calculate the eigenvalues and eigenvectors. We will then test whether the first vector and value are in fact an eigenvalue and eigenvector for the matrix.
The eigenvectors are returned as a matrix with the same dimensions as the parent matrix, where each column is an eigenvector, e.g. the first eigenvector is vectors[:, 0]. Eigenvalues are returned as a list, where value indices in the returned array are paired with eigenvectors by column index, e.g. the first eigenvalue at values[0] is paired with the first eigenvector at vectors[:, 0].

```
# confirm eigenvector
from numpy import array
from numpy.linalg import eig
# define matrix
A = array([
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]])
# factorize
values, vectors = eig(A)
# confirm first eigenvector
B = A.dot(vectors[:, 0])
print(B)
C = vectors[:, 0] * values[0]
print(C)
```

The example multiplies the original matrix with the first eigenvector and compares it to the first eigenvector multiplied by the first eigenvalue. Running the example prints the results of these two multiplications that show the same resulting vector, as we would expect.

Contents

We can reverse the process and reconstruct the original matrix given only the eigenvectors and eigenvalues. First, the list of eigenvectors must be taken together as a matrix, where each vector becomes a row. The eigenvalues need to be arranged into a diagonal matrix. The NumPy diag() function can be used for this. Next, we need to calculate the inverse of the eigenvector matrix, which we can achieve with the inv() NumPy function. Finally, these elements need to be multiplied together with the dot() function.

```
# reconstruct matrix
from numpy import diag
from numpy.linalg import inv
from numpy import array
from numpy.linalg import eig
# define matrix
A = array([
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]])
print(A)
# factorize
values, vectors = eig(A)
# create matrix from eigenvectors
Q = vectors
# create inverse of eigenvectors matrix
R = inv(Q)
# create diagonal matrix from eigenvalues
L = diag(values)
# reconstruct the original matrix
B = Q.dot(L).dot(R)
print(B)
```

The above complication with the Σ diagonal only exists with the case where m and n are not equal. The diagonal matrix can be used directly when reconstructing a square matrix, asfollows.

```
# reconstruct square matrix from svd
from numpy import array
from numpy import diag
from scipy.linalg import svd
# define matrix
A = array([
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]])
print(A)
# factorize
U, s, V = svd(A)
# create n x n Sigma matrix
```

```
Sigma = diag(s)
# reconstruct matrix
B = U.dot(Sigma.dot(V))
print(B)
```