**PCA**

**Class Architicture:**

We have totally **7** function:

* **The constructor:**

We specify the number of the component (new features) that we want our data to have, and it specifies two variables, one to store the **component**, and another one to store the **mean**

* **Normalization:**

To standardize the data using the mean

* **Covariance:**

To calculate the covariance matrix

* **Calc\_eigen:**

To calculate the eigen values and vectors

* **Fit:**

It is the main function of our class

* **Transform:**

It takes the data matrix then transform it to the new **reduced data** using the same object after performing its **Fit function**

* **Reverse:**

Trying to de-transform the new (reduced) data to its original values

**Eigen:**

To calculate the Eigen Values and Vectors we made custom function using **sympy** library, the function has the following sub-functions:

* **calc\_determinant:**

its custom **recursive** function using **divide and conquer** method, to calculate the determinant of a given matrix by:

1. checking if the shape of a function is (2,2) if its true then the determinant is the:

(value [0][0] \* value [1][1]) – (value [0][1] \* value [1][0])

1. if the shape of the matrix is not a (2,2) then I will divide the matrix into **M** **sub-matrices**, which **M** is the number of **columns** or **rows** in the parent matrix “note the matrix is squared matrix so number of rows = number of columns”.

these new matrices will have the shape of (M-1, M-1) because we are removing the first row and **T’th column**, which **T** is the number of the **sub-matrix**.

We calculate the determinant to the sub-matrices using the same function, the final determinant to the original matrix will be:

the summation of all the sub-determinant each sub-determinant multiplied by (-1)^T and its **T** value “value [0][T] from the parent matrix” from the first row

see the following formula

A computer screen with text

Description automatically generated

* calc\_eigenvalues:

its custom function takes the result of:

A black symbols of a mathematical equation

Description automatically generated with medium confidence

Let’s make the above formula equals to matrix called Result, our calc\_eigen function takes the Result function then forming from it the equation of the determinant

then finding the values of Lambda that makes the determinant equals to zero by using Eq, and solve functions from sympy library

* calc\_eigenvectors:

in this function we are calculating the eigen vectors using the eigen values and Result matrix.

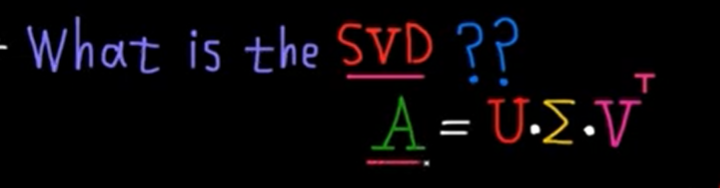
but we faced a problem:

**Problem Definition**

that the determinant of The Result matrix after substituting the Lambda with an Eigen value not always Equal to zero some-times the determinant is a very small value it almost equal to zero, that’s due to the Floating-point precision , and this make the solve functions from sympy library consider the matrix is not singular and give us trivial solution witch is all the vectors are equals to zeros, when in the same time its theoretically should equal to zero for the eigenvalues

**Solution**

So we used **SVD** (Singular Value Decomposition) matrix analyzation its robust to numerical issues because it works with floating-point approximations and doesn't rely directly on symbolic solvers, because it’s can give us a singular matrix and its determinant is exactly equal to **zero**



We can choose U or V^T use V^T because there are both squared matrices and their vectors are perpendicular on each other

**The SVD approach is robust to numerical issues because it works with floating-point approximations and doesn't rely directly on symbolic solvers. This makes it more suitable for cases where numerical precision is a concern**

**Extra Details about the problems we had**

1. we find out that the resulted eigen values from our custom function **calc\_eigenvalues** is find numbers that makes the determinant of the matrix is almost zero but not equal it

A screenshot of a computer

Description automatically generated

After comparing my custom function’s eigen values and

The built-in Numpy’s eigen values we found out this:

A screenshot of a computer

Description automatically generated

Our eigen values are different from the Numpy eigen values with Only one digit “our custom eigen is missing number 5 from the right”

But still, we have the same problem

1. after that we checked if the custom calc\_determinant function is the problem by comparing its results with the built in Numpy function

But still, we have the same problem

A screen shot of a computer code

Description automatically generated

But still, we have the same problem