Ciencia de la Computación

Estructuras de Datos Avanzada

Orientación de puntos - PCA

Docente Carlos Eduardo Atencio Torres

Jun 2023

Autor:
Iris Rocio Curo Quispe

Semestre VI
2023-1

Orientación de puntos - PCA

Iris Rocio Curo Quispe
iris.curo@ucsp.edu.pe

June 18, 2023

Abstract

En el presente trabajo se presenta la implementación del calculo de PCA y el vector director de un conjunto de puntos. Para graficar lo puntos se usó Python de forma que se pueda visualizar la nube de puntos junto con el vector director hallado para estos puntos, el cual será el primer eigenvector.

Keywords— PCA, vector director, números aleatorios.

1 Introducción

En el presente informe se describe la implementación de los códigos proporcionados en el repositorio de Github de *Diego Mazala* para realizar el Análisis de Componentes Principales (PCA, por sus siglas en inglés) utilizando el lenguaje de programación C++ y la librería Eigen3. El objetivo de esta implementación es calcular los eigenvectores de una nube de puntos tridimensionales y visualizarlos en un gráfico 3D.

El PCA es una técnica fundamental en el campo del procesamiento de datos y la reducción de dimensionalidad. Permite encontrar las direcciones principales en un conjunto de datos y explicar la selección de los mismos en términos de estas direcciones. En este informe, se presenta un análisis detallado del código `pca_test.cpp`, el cual implementa el PCA utilizando la biblioteca Eigen en C++. El objetivo principal de este informe es comprender cómo se realiza el cálculo del PCA, cómo se generan los resultados y qué información se puede obtener a partir de ellos.

El código `pca_test.cpp` comienza solicitando al usuario que especifica el tamaño de la matriz de datos. A continuación, genera una matriz de datos aleatorios utilizando una distribución uniforme. Los valores de esta matriz se guardan en un archivo de texto para su posterior análisis. Luego, el código utiliza la biblioteca Eigen para realizar el cálculo del PCA. Se emplea la clase `pca_t` proporcionada, que encapsula las operaciones necesarias para obtener los resultados del PCA. Una vez realizado el cálculo, los resultados se guardan en el mismo archivo de texto previamente creado.

En el informe, se tomarán y analizarán los resultados obtenidos a través del cálculo del PCA. Se muestran matrices como la matriz de entrada, la matriz centrada, la matriz de covarianza y la matriz de proyección. Además, se examinarán los vectores propios y los valores propios asociados al PCA, los cuales son fundamentales para comprender las direcciones principales en los datos y su contribución a la necesidad del conjunto. Además, se realizará una reproyección de la matriz de datos y se calculará el error entre la matriz original y la reproyección obtenida. Este error nos brinda información sobre la calidad de la representación de los datos utilizando el PCA y nos permite evaluar la efectividad de la técnica.

Para lograr esto, se realizaron los siguientes pasos:

- Generación de puntos aleatorios y vector de dirección: Se agregó una funcionalidad adicional para generar puntos aleatorios y un vector de dirección usando el primer *eigenvector* en C++. Se implementó un código que genera estos datos de manera aleatoria junto con el vector director calculado y los guarda en un archivo para su posterior utilización.
- Modificación del código para leer los datos generados: Por último, se modificó el código original en Python para leer los datos generados por el código en C++. Esto implicó realizar cambios en la forma en que se cargan y procesan los datos en el código Python, de manera que se puedan utilizar los datos generados en C++ para realizar la gráfica que los presenta.

2 Generación de puntos aleatorios y vector director

A continuación se presenta una explicación del código `pca_test.cpp` en cuestión. El código es un ejemplo de cómo realizar el PCA utilizando la biblioteca Eigen en C++:

- El programa comienza verificando si se ha proporcionado un argumento en la línea de comandos, que representa el tamaño de la matriz de datos a generar. Si no se proporciona el argumento adecuado, se muestra un mensaje de uso y se finaliza el programa.
- A continuación, se genera una matriz de datos aleatorios llamada `pca_data_matrix` utilizando la biblioteca Eigen. La matriz tiene `"m"` filas y 3 columnas (3D), donde `"m"` es el tamaño especificado por el usuario. Los valores de la matriz se generan utilizando una distribución uniforme en el rango de -5.0 a 5.0, este rango se puede modificar.
- Después de generar la matriz de datos, se guarda en un archivo de texto llamado `pca_data.txt`. El archivo contiene la matriz de datos y se estructura de la siguiente manera: los valores de la matriz se escriben fila por fila, separados por espacios. A continuación, se realiza el cálculo del PCA utilizando la clase `pca_t` la cual esta definida en `pca.h`.
- El resultado del PCA se guarda en el archivo `pca_data.txt`. Primero, se obtienen los vectores propios (*eigenvectors*) de la matriz de covarianza y se guarda el primer vector propio en el archivo. Luego, se cierra el archivo y se muestra un mensaje indicando que la matriz y el vector director han sido guardados exitosamente en el archivo `pca_data.txt`.
- A continuación, se realiza el cálculo del PCA nuevamente utilizando la clase `pca_t` y se muestran los resultados en la consola. Se imprimen las siguientes matrices y vectores: la matriz de entrada, la matriz centrada, la matriz de covarianza, la matriz de proyección, la matriz de medias, los valores propios (*eigenvectors*) y los vectores propios (*eigenvectors*). Estos resultados proporcionan información detallada sobre los diferentes pasos y resultados del PCA.
- Además, se realiza una reproyección de la matriz de datos utilizando la función `reprojection()` y se calcula el error entre la matriz original y la reproyección utilizando la función `norm()`. El resultado de la reproyección y el error se imprimen en la consola.

Listing 1: Código `pca_test.cpp` que genera una matriz de datos aleatorios, realiza el cálculo del PCA utilizando la biblioteca Eigen en C++.

```
#include <iostream>
#include <fstream>
#include <random>
#include "pca.h"

#include <experimental/filesystem>

using namespace std;
namespace fs = std::experimental::filesystem;

int main(int argc, char* argv[])
{

    if (argc != 2)
    {
        std::cout << "Usage: " << argv[0] << "(matrix size) m x 3 e.g. 10 x 3"
            << std::endl;
        return EXIT_FAILURE;
    }

    int m = std::stoi(argv[1]);

    // Generar puntos aleatorios en pca_data_matrix
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<float> dis(-5.0, 5.0);

    Eigen::Matrix<float, Eigen::Dynamic, 3> pca_data_matrix(m, 3);
```

```
28      for (int i = 0; i < pca_data_matrix.rows(); ++i)
29      {
30          for (int j = 0; j < pca_data_matrix.cols(); ++j)
31          {
32              pca_data_matrix(i, j) = dis(gen);
33          }
34      }
35
36      // Guardar la matriz en un archivo de texto
37      std::ofstream outfile("pca_data.txt");
38      if (outfile.is_open())
39      {
40          outfile << "Puntos:\n";
41          for (int i = 0; i < pca_data_matrix.rows(); ++i)
42          {
43              for (int j = 0; j < pca_data_matrix.cols(); ++j)
44              {
45                  outfile << pca_data_matrix(i, j) << " ";
46              }
47              outfile << std::endl;
48          }
49
50          pca_t<float> pca;
51          pca.set_input(pca_data_matrix);
52          pca.compute();
53
54          Eigen::Matrix<float, Eigen::Dynamic, Eigen::Dynamic> eigen_vectors = pca
                .get_eigen_vectors();
55          Eigen::Matrix<float, Eigen::Dynamic, 1> first_eigenvector =
                eigen_vectors.col(0);
56
57          // Escribir el primer eigenvector en el archivo
58          outfile << "Vector Director:\n";
59          for (int i = 0; i < first_eigenvector.rows(); ++i)
60          {
61              outfile << first_eigenvector(i) << " ";
62          }
63          outfile << std::endl;
64
65          outfile.close();
66          std::cout << "Matriz y Vector Director guardados en pca_data.txt" << std
                ::endl;
67      }
68      else
69      {
70          std::cerr << "No se pudo abrir el archivo pca_data.txt" << std::endl;
71          return EXIT_FAILURE;
72      }
73
74      // Realizar el c lculo de PCA
75      pca_t<float> pca;
76      pca.set_input(pca_data_matrix);
77      pca.compute();
78
79      std::cout
80          << "Input Matrix:      \n" << pca.get_input_matrix() << std::endl << std
                ::endl
81          << "Centered Matrix:   \n" << pca.get_centered_matrix() << std::endl <<
                std::endl
82          << "Covariance Matrix: \n" << pca.get_covariance_matrix() << std::endl
                << std::endl
83          << "Projection Matrix: \n" << pca.get_projection_matrix() << std::endl
                << std::endl
84          << "Mean Matrix:       \n" << pca.get_mean() << std::endl << std::endl
85          << "Eigen Values:      \n" << pca.get_eigen_values() << std::endl << std
                ::endl
```

```
86          << "Eigen Vectors:     \n" << pca.get_eigen_vectors() << std::endl <<
                std::endl;
87
88      const auto& reprojection = pca.reprojection();
89      auto error = (pca_data_matrix - reprojection).norm();
90
91
92      std::cout
93          << "Reprojected Matrix:\n" << reprojection << std::endl << std::endl
94          << std::fixed
95          << "Error:             \n" << error << std::endl << std::endl;
96
97      return EXIT_SUCCESS;
98  }
```

```
1  Eigen::MatrixXf generarFigura(int n)
2  {
3      std::random_device rd;
4      std::mt19937 gen(rd());
5      std::uniform_real_distribution<float> dis(0, 2 * M_PI);
6
7      Eigen::MatrixXf figura(n, 3);
8      for (int i = 0; i < n; ++i)
9      {
10         float x = static_cast<float>(i - n / 2);
11         float theta1 = dis(gen);
12         float theta2 = dis(gen);
13         float y = x * std::sin(theta1);
14         float z = x * std::sin(theta2);
15
16         figura(i, 0) = x;
17         figura(i, 1) = y;
18         figura(i, 2) = z;
19     }
20
21     return figura;
22 }
```

Listing 2: Función para generar una distribución tipo mariposa de puntos aleatorios

3 Generación de gráfico 3D

El código presentado en `pca_example.py` tiene como objetivo cargar los datos de un archivo llamado tpca_data.txt para graficar los puntos y un vector director en un espacio tridimensional, como ejemplo, se puede ver la imagen 2. A continuación, se presenta una descripción paso a paso del funcionamiento del código:

- En primer lugar, se cargan los datos de un archivo llamado `pca_data.txt`, que contiene la información de los puntos y el vector director. Los puntos se almacenan en un arreglo bidimensional llamado *"points"*, mientras que el vector director se guarda en un arreglo unidimensional llamado *"direction"*.
- A continuación, se calculan los puntos extremos de la nube de puntos utilizando las funciones `np.min()` y `np.max()`. Estos puntos extremos se utilizarán para determinar la longitud y posición del vector director en el gráfico.
- La longitud del vector director se calcula utilizando la función `np.linalg.norm()`, que devuelve la norma euclidiana del arreglo. Luego, se determina un factor de escala dividiendo la norma de la distancia entre los puntos extremos de la nube de puntos por la longitud del vector director. Este factor de escala se reafirma para ajustar la longitud del vector director proporcionalmente a la distancia entre los puntos extremos.

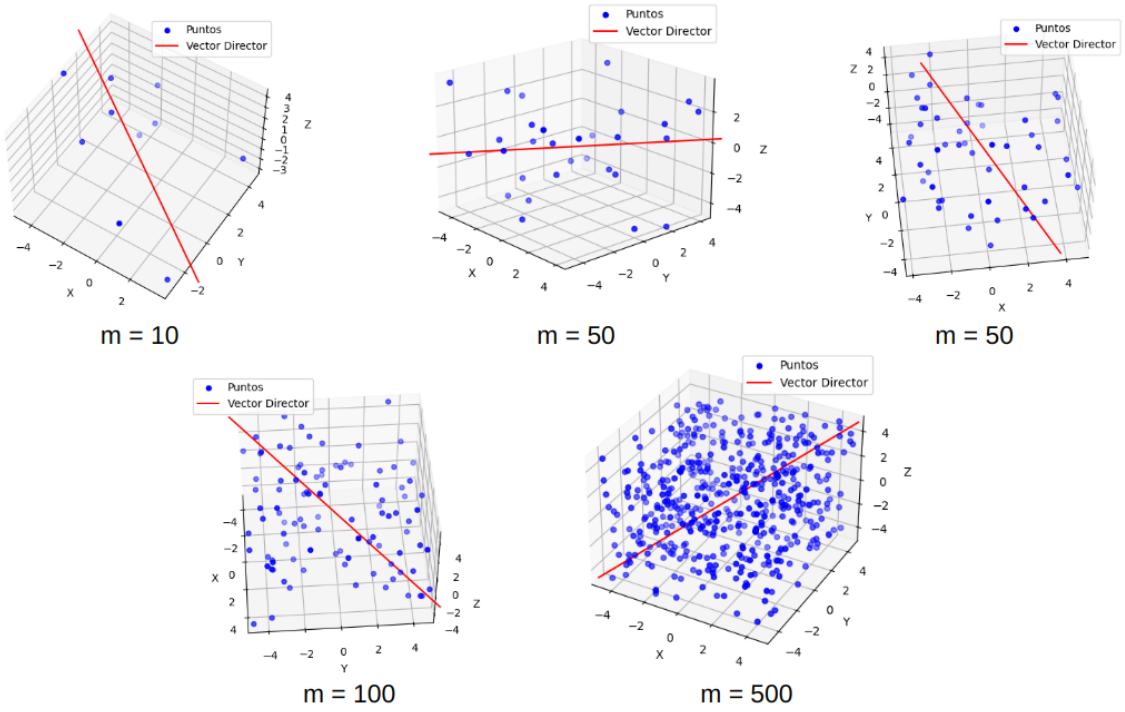

Figure 1: Gráfico que muestra `m` puntos aleatorios generados junto con el vector director calculado usando el primer *eigenvector* al aplicar PCA.

- A continuación, se calculan los puntos de inicio y fin del vector director. El punto de inicio se determina como el punto medio entre los puntos extremos de la nube de puntos, mientras que el punto de fin se obtiene sumando el vector director escalado al punto de inicio.
- Luego, se crea una figura y unos ejes 3D utilizando la biblioteca matplotlib. Los puntos se grafican como una dispersión tridimensional utilizando la función `scatter()`, y se les asigna el color azul.
- Finalmente, el vector director se dibuja en el gráfico usando la función `plot3D()`. Los puntos de inicio y fin del vector se sustentan como argumentos para definir las coordenadas `X, Y` y `Z` de la línea que representa el vector director. Esta línea se dibuja en color rojo.
- Se configuran los límites de los ejes del gráfico utilizando los puntos extremos de la nube de puntos, lo que garantiza que todos los puntos y el vector director sean visibles en el gráfico.
- Se agregan etiquetas a los ejes `X, Y` y `Z`, y se agrega una leyenda que indica la representación de los puntos y el vector director en el gráfico.

  Finalmente, se muestra el gráfico resultante utilizando la función `show()`.

Listing 3: Implementación de `pca_example.py` para graficar la nube de puntos con su vector director.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Cargar los datos desde el archivo pca_data.txt
with open('pca_data.txt', 'r') as file:
    lines = file.readlines()

# Extraer los puntos y el vector director
points = np.loadtxt(lines[1:-2])
direction = np.fromstring(lines[-1], dtype=float, sep=' ')

# Calcular los puntos extremos de la nube de puntos
min_point = np.min(points, axis=0)
max_point = np.max(points, axis=0)

```

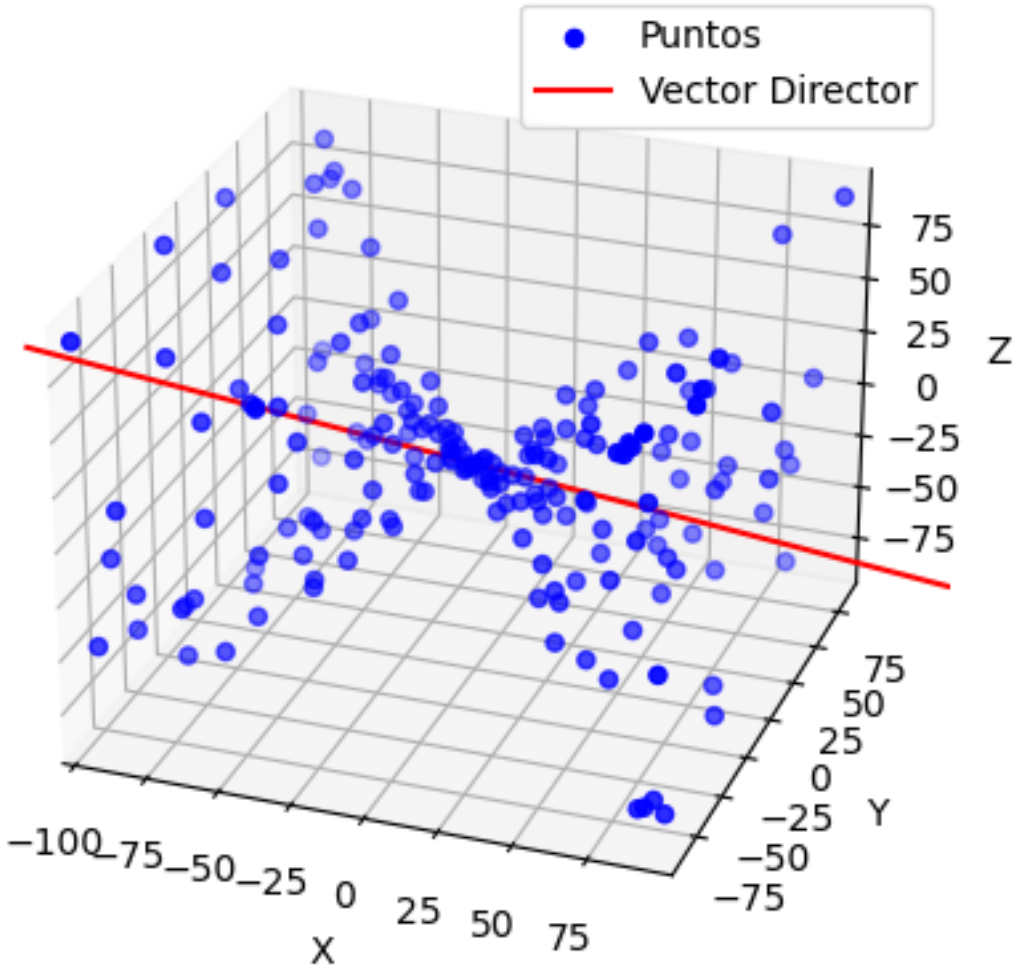

Figure 2: Gráfico que muestra `m = 200` puntos aleatorios generados con la distribución mariposa junto con el vector director calculado usando el primer *eigenvector* al aplicar PCA.

```
# Calcular la longitud del vector director
point_range = max_point - min_point
direction_length = np.linalg.norm(direction)
scaling_factor = np.linalg.norm(point_range) / direction_length
scaled_direction = direction * scaling_factor

# Calcular los puntos de inicio y fin del vector director
start_point = (min_point + max_point) / 2.0 - scaled_direction / 2.0
end_point = start_point + scaled_direction

# Crear la figura y los ejes 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Graficar los puntos
ax.scatter(points[:, 0], points[:, 1], points[:, 2], c='blue', label='Puntos')

# Graficar el vector director proporcionalmente
ax.plot3D([start_point[0], end_point[0]],
          [start_point[1], end_point[1]],
          [start_point[2], end_point[2]], 'red', label='Vector Director')

```

```
# Configurar los l mites de los ejes
ax.set_xlim(min_point[0], max_point[0])
ax.set_ylim(min_point[1], max_point[1])
ax.set_zlim(min_point[2], max_point[2])

# Configurar etiquetas y leyendas
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()

# Mostrar el gr fico
plt.show()
```

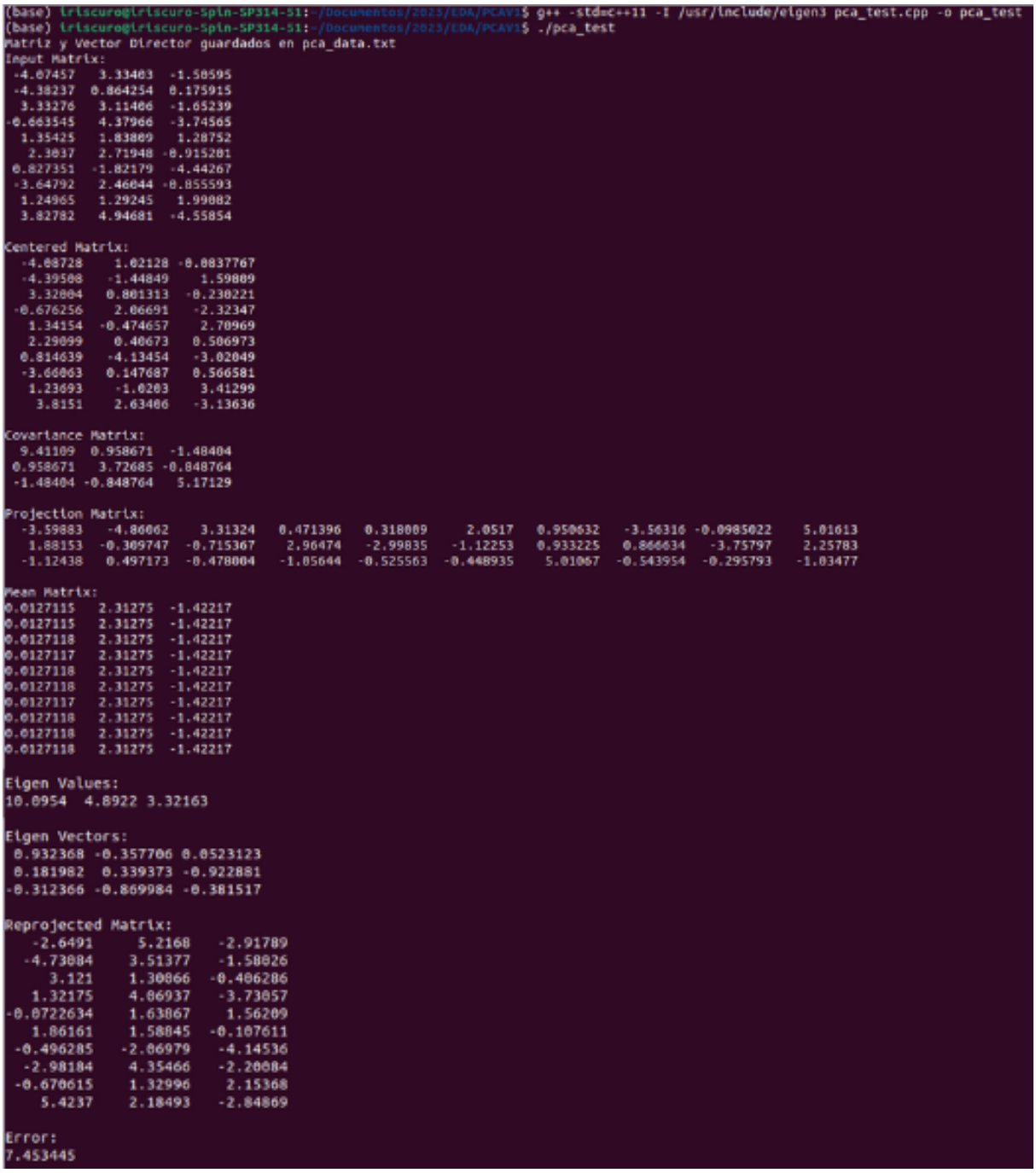

Figure 3: Captura de pantalla que muestra los resultados luego de ejecutar el código `pca_test.cpp` con 10 puntos.

4 Conclusiones

Gracias al análisis del código `pca_test.cpp` me permitió comprender cómo se implementa y se utiliza el PCA en el lenguaje de programación C++ utilizando la biblioteca Eigen, además de su visualización usando Python. Los resultados obtenidos brindaron información valiosa sobre las direcciones principales en los datos mediante la técnica del PCA. Durante el análisis de los resultados, se evaluaron diferentes matrices, como la matriz de entrada, la matriz centrada, la matriz de covarianza y la matriz de proyección. Estas matrices nos proporcionaron información sobre la estructura y la adquisición de los datos originales.

Los vectores propios y los valores propios asociados al PCA. Estos vectores propios representan las direcciones principales en los datos, y los valores propios indican la contribución relativa de cada dirección principal a la necesidad total del conjunto de datos. Estos resultados son fundamentales para comprender y visualizar las principales características de los datos. El cálculo del error entre la matriz original y la matriz reproyectada permitió evaluar la calidad de la aproximación obtenida mediante el PCA. Un error bajo indica una buena aproximación de los datos originales, mientras que un error alto puede sugerir que las direcciones principales seleccionadas no son suficientes para representar adecuadamente la alteración de los datos.