



# Calculo de autovectores y procesamiento de imágenes

4 de junio de 2023

Métodos Numericos

Integrante	LU	Correo electrónico
Collasius, Federico	164/20	fede.collasius@gmail.com
Memoli Buffa, Pedro	376/20	demnitth@gmail.com
Schwartzmann, Alejandro Ezequiel	390/20	a.schwartzmann@hotmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		



**Facultad de Ciencias Exactas y Naturales**  
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

<https://exactas.uba.ar>

---

PALABRAS CLAVE: PCA, 2DPCA, reducción de dimensionalidad, similitud, compresión y procesamiento de imágenes

RESUMEN: En este trabajo implementamos una forma robusta del método de potencia con deflación en base a experimentación y lo comparamos con el método mas sofisticado de Eigen, obteniendo un rendimiento en términos de error comparable. También implementamos y experimentamos en base al método anterior los algoritmos de PCA y 2DPCA para el procesamiento y compresión de imágenes.

# Índice

<b>1. Introducción</b>	<b>2</b>
1.1. Presentación del problema	2
1.2. Métodos	2
1.2.1. Método de la Potencia con Deflación	2
1.2.2. PCA	2
1.2.3. 2DPCA	3
<b>2. Calculo autovectores</b>	<b>4</b>
2.1. Método de la potencia	4
2.1.1. Tiempo computó método de la potencia	4
2.1.2. Error en función de las iteraciones	5
2.1.3. Limitaciones del método naive	6
2.1.4. Método de la potencia con threshold	7
2.2. Método de la potencia con deflación	8
2.2.1. Fiabilidad del algoritmo	8
<b>3. Procesamiento de imágenes</b>	<b>10</b>
3.1. PCA	10
3.1.1. Implementacion	10
3.1.2. Eigenfaces	11
3.1.3. Autovalores y Varianza de los componentes principales	11
3.1.4. Reconstrucción de imágenes	12
3.1.5. Similaridad	13
3.1.6. Error de compresión	13
3.2. 2DPCA	15
3.2.1. Implementación	15
3.2.2. Eigenfaces	16
3.2.3. Autovalores y Varianza de los componentes principales	16
3.2.4. Reconstrucción de imágenes	17
3.2.5. Similaridad	17
3.2.6. Error de compresión	18
<b>4. Conclusiones</b>	<b>20</b>

# 1. Introducción

## 1.1. Presentación del problema

El cálculo de autovectores y autovalores de matrices es un problema central al álgebra lineal. Más allá de su interés teórico, presenta una infinidad de aplicaciones en todas las ramas de la ciencia e ingeniería, por lo que encontrar algoritmos que los puedan calcular numéricamente es extremadamente importante. Lamentablemente, el cálculo analítico de los mismos requiere calcular las raíces de un polinomio, los cuales no tienen fórmula cerrada para grados mayores a 4. Por esto es necesario depender de algoritmos iterativos que *converjan* a una solución. En este trabajo implementamos y experimentamos sobre uno de estos métodos, el método de la potencia con deflación. Además, presentamos una mejora a su implementación elemental y la comparamos con algoritmos más sofisticados.

Una aplicación clásica del cálculo de autovectores de una matriz es el procesamiento y compresión de imágenes. En general, la alta dimensión del espacio de imágenes presenta complicaciones espaciales para su procesamiento o almacenamiento. Por lo tanto, es de sumo interés lograr reducir una imagen a un conjunto de números menor a su dimensión, pero que aún contenga toda su información relevante. Para lograrlo implementamos y experimentamos sobre dos métodos de reducción de dimensiones: PCA y una variante específica para tratar imágenes, 2DPCA.

## 1.2. Métodos

### 1.2.1. Método de la Potencia con Deflación

El objetivo de esta sección del trabajo es implementar, experimentar y entender las limitaciones sobre el método de la potencia con deflación para calcular autovectores.

Sea  $M$  una matriz de  $n \times n$  diagonalizable y sin autovalores repetidos. Al ser diagonalizable existe  $B$  una base ortonormal  $\{v_1, \dots, v_n\}$  con autovalores respectivos que siguen un orden de  $|\lambda_1| > \dots > |\lambda_n|$ . Partiendo de  $S_0$  una condición inicial cuya proyección en  $B$  no tenga la primer coordenada nula, el método de la potencia se basa en la convergencia de la siguiente sucesión [1]:

$$S_k + 1 = M^k S_k / \|M^k S_k\|, \quad (1)$$

que converge en estas hipótesis al autovector  $v_1$  a diferencia de un signo. El autovalor puede obtenerse con el coeficiente de Rayleigh:

$$\lambda_n = S_k^t M S_k, \quad (2)$$

que converge al autovalor dominante  $\lambda_1$ .

Para calcular el resto de autovectores se hace uso de la propiedad de deflación. Esta consiste en que la matriz  $M - \lambda_1 \cdot v_1 \cdot v_1^t$  tiene los mismos autovectores y valores que  $M$  con excepción de los generados por  $v_1$ . Pueden obtenerse así el resto de autovectores ejecutando deflación luego de calcular un valor dominante y repitiendo el proceso  $n$  veces.

### 1.2.2. PCA

Como se mencionó previamente, el método PCA [2], o Principal Component Analysis, consiste en expresar un vector aleatorio de  $n$  componentes como suma de otros  $n$  vectores aleatorios independientes. Estos vectores se denominan *componentes principales*. El método consiste en obtener una base ortonormal de autovectores de la matriz de covarianza  $\Sigma$  del vector aleatorio  $V$ . Luego cualquier muestra del vector puede expresarse como combinación lineal de estos autovectores. La característica que los hace útiles es que la varianza de cada componente es su autovalor, y como son



Primera cara del dataset

independientes, el aporte de la varianza de cada componente a la varianza total puede simplemente obtenerse mirando el autovalor. Esto hace que pueda reducirse una imagen a algunas coordenadas de la base ortonormal que abarquen casi toda la varianza.

Para el procesamiento de imágenes, consideremos  $\mathbf{x}_i \in \mathbf{R}^n$  a la i-ésima imagen de nuestro dataset. Esta representa una muestra de la variable aleatoria de imágenes de caras de frente. Se almacena como un vector de largo  $n$  la cantidad total de píxeles. Para estimar la matriz de covarianza, se apilan todas las imágenes como filas en una matriz  $\mathbf{X} \in \mathbf{R}^{m \times n}$ . Luego se calcula  $\boldsymbol{\mu}_j = \frac{1}{m} \sum_{i=1}^m \mathbf{X}_{ij}$  la media del pixel j-esimo de las imágenes y se define la matriz centrada  $\mathbf{X}_c$  que contiene en la i-ésima fila al vector  $(\mathbf{x}_i - \boldsymbol{\mu})^t$ . La matriz de covarianza se estima como  $\mathbf{C} = \frac{\mathbf{X}_c^t \mathbf{X}_c}{n-1}$ .

Definimos a la imagen  $\mathbf{x}_i$  proyectada en el espacio de menor dimensión como el vector  $\mathbf{z}_i = (\mathbf{v}_1^t \mathbf{x}_i, \mathbf{v}_2^t \mathbf{x}_i, \dots, \mathbf{v}_k^t \mathbf{x}_i) \in \mathbf{R}^k$  donde  $\mathbf{v}_j$  es el autovector de  $\mathbf{C}$  asociado al j-ésimo autovalor al ser ordenados de mayor a menor y  $k$  define cuantas componentes se utilizan. Este proceso corresponde a proyectar la imagen sobre las  $k$  primeras componentes principales. La intención es que  $\mathbf{z}_i$  resuma la información más relevante de la imagen, descartando las dimensiones de poca varianza que resultan en detalles o las regiones de la imagen que no aportan rasgos distintivos.

### 1.2.3. 2DPCA

El método 2DPCA [3] surge debido a que cuando se aplica PCA a las imágenes, estas se transforman en vectores unidimensionales que forman la matriz de datos  $X$ , a la cual se le calcula la matriz de covarianza y sus autovectores. Esto obliga a calcular los autovectores de una matriz de dimensionalidad considerablemente alta.

Para superar esta limitación, se introduce 2DPCA (PCA bidimensional), que mantiene las imágenes en su espacio original como una matriz. En este contexto, consideramos una imagen representada como una matriz  $A$  de  $a \times b$ . Se le asocia un *feature vector* mediante la relación  $Y = AX$ , donde  $X$  es un vector que maximiza la dispersión de los vectores de características. Se puede demostrar que estos son los autovectores de la matriz

$$G = \frac{1}{n} \sum_{j=1}^n ((A_j - \bar{A}))^T (A_j - \bar{A})), \quad (3)$$

donde  $A_j$  es la imagen j-esima y  $\bar{A}$  la imagen media. Como desventaja de PCA, estos autovectores no tienen la interpretación de que su autovalor es el aporte a la varianza total.

Puede reconstruirse  $A$  como suma de productos externos de feature vectors  $Y_i$  y sus respectivos autovectores  $X_i$

$$A = \sum_{j=1}^b Y_j X_j^t \quad (4)$$

Esto permite almacenar  $k$  componentes del método en  $(A_1 X_1, \dots, A_n X_k)$ .

---

## 2. Calculo autovectores

Se quiere construir y estudiar el comportamiento de un algoritmo que combine el método de la potencia con deflación para calcular todos los autovectores y autovalores de una matriz que cumpla las hipótesis (num). Para esto primero se presenta en la sección 2.1 una implementación y experimentación del método de la potencia, a partir de la cual en la sección 2.2 se combina con el método de deflación para dar un algoritmo robusto que cumpla el propósito de la sección. Finalmente en 2.3 se compara el método con el cálculo más sofisticado de Eigen [4].

### 2.1. Método de la potencia

Una implementación rudimentaria del método de la potencia consiste en obtener hasta el valor  $k$ -ésimo de la sucesión (num) a partir de una condición inicial generada arbitrariamente. Presentamos una implementación naive de este algoritmo:

---

**Algorithm 1** Calcula el autovalor y autovector dominante de una matriz  $M$

---

```
1: procedure MÉTODO DELA POTENCIA( $M, k$ )
2:    $v_i \leftarrow Random(|Filas(M)|)$   $\triangleright$  Inicia con un vector aleatorio de tamaño igual a las filas de  $M$ 
3:    $v_i \leftarrow v_i / \|v_i\|$ 
4:    $s \leftarrow M \cdot ci$   $\triangleright$  Calcula el producto matricial de  $M$  por  $ci$ 
5:   for  $i \leftarrow 1$  to  $l$  do
6:      $s \leftarrow M \cdot s$   $\triangleright$  Obtiene el siguiente valor de la sucesion
7:      $s \leftarrow s / \|s\|$ 
8:   end for
9:    $\lambda \leftarrow s^T \cdot M \cdot s$   $\triangleright$  Calcula el autovalor
10:  return  $\lambda, s$   $\triangleright$  Retorna el autovalor y el autovector correspondiente
11: end procedure
```

---

Este consiste en partir de una condición inicial arbitraria con cada coordenada sampleada de una uniforme de -1 a 1 que se normaliza. Luego se computan recursivamente  $k$  valores de la sucesión (num) utilizando la identidad (num). Finalmente, se retorna el autovalor dominante dada la relación (num) junto con el autovector correspondiente.

En las siguientes secciones estudiamos el tiempo de cómputo del método en función de la cantidad de iteraciones y del tamaño de la matriz. Luego estudiamos el comportamiento del error en función de la cantidad de iteraciones para diversas matrices. Proseguimos haciendo un análisis de las limitaciones del método implementado de esa forma y basándonos en eso aplicamos una mejora al algoritmo. Finalmente, comparamos los autovalores calculados por la implementación mejorada con el de un algoritmo más sofisticado como es el de Eigen [referencia].

#### 2.1.1. Tiempo computó método de la potencia

Queremos ver como varía el tiempo de cómputo del algoritmo en función de la dimensión de la matriz. Esperamos ver un orden de tiempo cuadrático al tener este una complejidad de  $O(n^2)$ . Corremos el método de la potencia 100 iteraciones para 100 condiciones iniciales. Graficamos la mediana del tiempo de cómputo en función de la dimensión de la matriz en escala logarítmica. También superponemos una recta de pendiente 2 en el mismo gráfico para ver si resulta paralela al gráfico anterior, ya que esto implicaría que el tiempo va como  $O(n^2)$ .

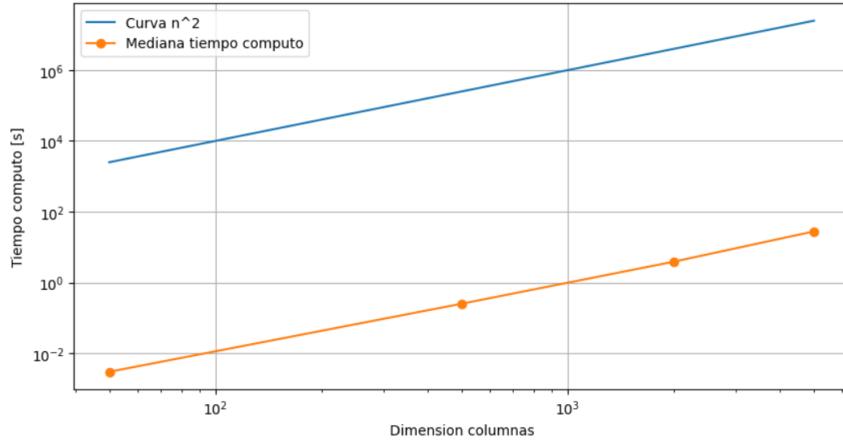


Figura 1: Tiempo de computo en función de la dimensión de las columnas en escala logarítmica

Consistente con que el algoritmo tiene una complejidad  $O(n^2)$  observamos que el tiempo de cómputo en escala logarítmica es una recta paralela a otra de pendiente 2. Vemos ahora que ocurre variando la cantidad de iteraciones y dejando fija la dimensión.

Fijando ahora la dimensión de la matriz como 100x100, corremos el método para 100 condiciones iniciales. Graficamos la mediana del tiempo de cómputo en función de la cantidad de iteraciones del método en escala logarítmica. También superponemos una recta de pendiente 1 en el mismo gráfico para ver si resulta paralela al gráfico anterior, ya que esto implicaría que el tiempo va como  $O(k)$ .

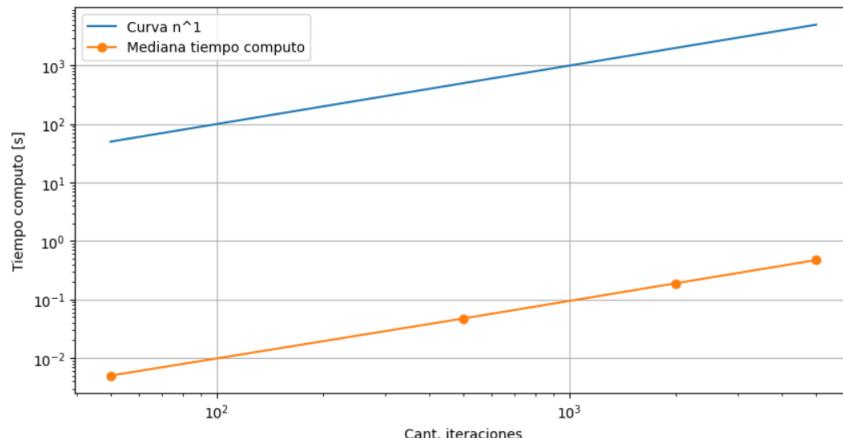


Figura 2: Tiempo de computo en función de la dimensión de las columnas en escala logarítmica

Como esperábamos, el tiempo de cómputo en función de la cantidad de iteraciones en escala logarítmica es una recta paralela a otra de pendiente 1. Esto indica que efectivamente el tiempo de cómputo va como  $O(k)$ .

Basándonos en los experimentos previos podemos concluir que el tiempo de cómputo del algoritmo va como  $O(kn^2)$ . De cualquier forma, para una matriz arbitraria no conocemos la cantidad de iteraciones necesarias para obtener un autovalor dominante suficientemente cercano al real. Esto motiva la siguiente sección, en la cual estudiamos el error en función de la cantidad de iteraciones para ver si hay una iteración óptima, o si varía mucho dependiendo la matriz.

### 2.1.2. Error en función de las iteraciones

Para matrices diagonalizables cuyo autovalor dominante es conocido y sabemos que la sucesión num (converge), queremos estudiar el error del método en la iteración i-esima del método. El error se define como la diferencia absoluta entre el valor real del autovalor y la aproximación por la sucesión. Para eso elegimos 3 matrices de 10x10 representativas de casos donde esperamos que el método tarde más o menos iteraciones en tener un error no-significativo.

Como en cada corrida del algoritmo se toma una condición inicial arbitraria, el gráfico de error vs. iteraciones va a naturalmente variar para cada condición. Con el fin de ver su comportamiento general tomamos 100 mediciones en cada caso, cada una correspondiente a una condición inicial distinta y graficamos los boxplots.

La primer matriz  $M^1$  que elegimos para estudiar es una diagonal con 5 en  $M_{11}^1$  y 1s en el resto de la diagonal. Esperamos que el error se aproxime a 0 en relativamente pocas iteraciones; ya que la diferencia entre el autovalor dominante de 5 y el resto de autovalores es considerablemente más grande que para las otras matrices que proponemos.

La segunda  $M^2$  es otra diagonal que tiene 2.01 en  $M_{11}^2$  y 2 en  $M_{22}^2$ , el resto de la diagonal está compuesta de 1s. Como la distancia entre el autovalor dominante 2.01 y el siguiente es de tan solo 0.01, es razonable esperar que la sucesión tarde más iteraciones en distinguir los primeros dos autovectores. Esperamos que en este caso tome en promedio más iteraciones en aproximarse al 0 que para  $M^1$ .

La tercera  $M^3$  también es diagonal y tiene 1.01 en  $M_{11}^3$  y el resto de la diagonal es 1s. Este es un caso extremo en el que todos los autovalores difieren por 0.01 del dominante 1.01.

Graficamos en tres gráficos consecutivos el error en función de la cantidad de iteraciones del algoritmo:

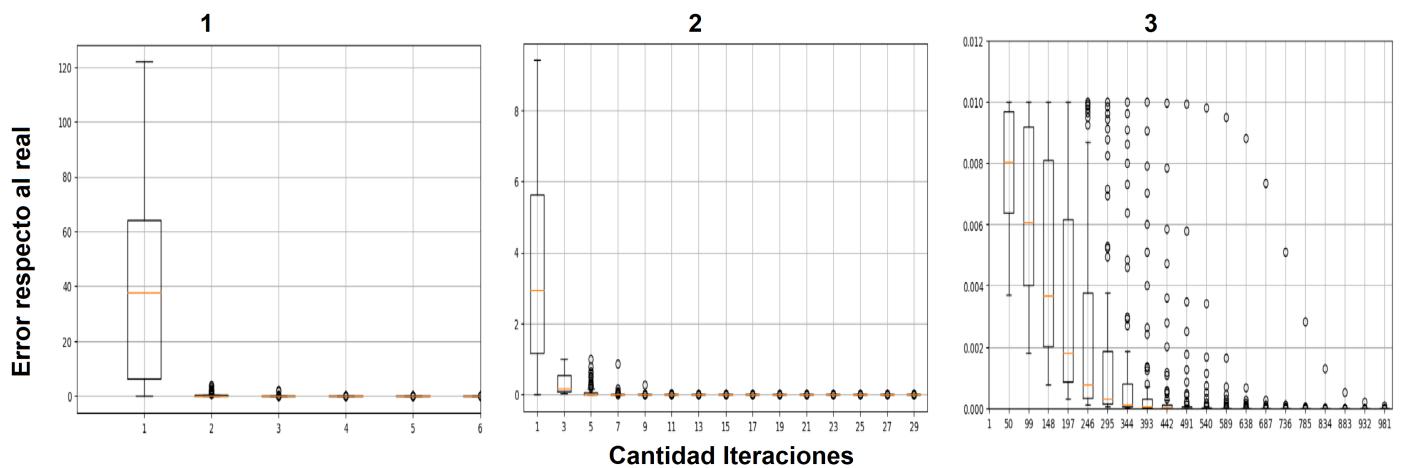


Figura 3: Tres gráficos de errores respecto al autovalor real en función de la cantidad de iteraciones ejecutadas por el algoritmo para las matrices  $M_1$ ,  $M_2$  y  $M_3$ .

Observamos bastante variación en la cantidad de iteraciones para llegar a un punto en el cual se pueda considerar que la sucesión *convergió*. Para los primeros dos casos, antes de la iteración 15 se obtiene en todas las 100 condiciones iniciales una estimación del autovalor que no difiere de forma significativa del real. Pero en el caso más extremo puede llegar a tardar casi dos órdenes más de iteraciones.

Como esperábamos por el carácter aleatorio de la condición inicial, vemos mucha varianza para las primeras iteraciones. Si bien en los primeros dos casos esta dispersión pasa a ser despreciable luego de pocas iteraciones, para la matriz más extrema esta varianza se mantiene por cientos de iteraciones. Peor aún se observan muchos outliers hasta alrededor de la iteración 700. Esto es indicativo que no es buena práctica fijar la cantidad de iteraciones antes de correr el algoritmo, ya que la convergencia respecto a un  $\epsilon$  chico se puede dar en la iteración 400, 800 o posiblemente muchísimo más. Esto lleva a razonar sobre las limitaciones del método de la potencia implementada de forma naive, y da lugar para proponer mejoras.

### 2.1.3. Limitaciones del método naive

Consistente con los experimentos y el desarrollo en la introducción, la cantidad de iteraciones que toma el algoritmo en producir un autovalor dominante está muy fuertemente condicionado por la *condición inicial* (varianza del error en una iteración dada) y por la *matriz propia* (mediana del

error en una iteración dada). Esto hace que a priori es muy difícil dar una cantidad de iteraciones al algoritmo que balancee error y tiempo de cómputo. Otro problema es que no se sabe si la matriz que se le pasa al método es tal que converge. Puede no ser diagonalizable o tener dos autovalores dominantes.

Una forma de solucionar este problema es no fijar la cantidad de iteraciones del método. Se puede hacer que el método haga todas las iteraciones necesarias hasta detectar que los autovalores aproximados difieren menos de un threshold  $\epsilon$ . Para hacer este método más robusto, se puede considerar que se convergió cuando un número  $n$  de autovalores aproximados consecutivos difieren por menos de  $\epsilon$ . Para protegerse de los casos de que la condición inicial sea ortogonal al autovector dominante, o de que la sucesión no converja, se puede fijar una cantidad de iteraciones máximas, tras la cual se toma otra condición inicial y se repite una cantidad prefijada de veces. Todas esas modificaciones se implementan sobre el algoritmo naive en la siguiente sección.

#### 2.1.4. Método de la potencia con threshold

Teniendo en cuenta todas las limitaciones desarrolladas previamente, presentamos una implementación más robusta del método de la potencia. Este también avisa si todo indica a que el algoritmo no converge:

---

**Algorithm 2** Calcula el autovalor dominante junto con su respectivo autovector

---

```

1: procedure MÉTODODELAPOENCIA( $M, \epsilon, maxIter, maxRepeticiones, k$ )
2:    $intentos \leftarrow 0$ 
3:   while  $intentos < maxRepeticiones$  do
4:      $ci \leftarrow Random(|Filas(M)|)$                                  $\triangleright$  Parte de una condicion inicial arbitraria
5:      $ci \leftarrow ci / \|ci\|$ 
6:      $s \leftarrow M \cdot ci$                                           $\triangleright S_0$ 
7:      $iteraciones \leftarrow 0$ 
8:      $cantElemsEnThreshold \leftarrow 0$ 
9:      $\lambda \leftarrow s^t \cdot M \cdot s$                                       $\triangleright \lambda_0$ 
10:    while  $cantElemsEnThreshold < k$  and  $iteraciones < maxIter$  do
11:       $s \leftarrow M \cdot s$                                           $\triangleright$  Calcula  $S_i$ 
12:       $s \leftarrow s / \|s\|$ 
13:       $\lambda_1 \leftarrow s^t \cdot M \cdot s$                                 $\triangleright$  Calcula  $\lambda_i$ 
14:      if  $|\lambda_1 - \lambda| < \epsilon$  then
15:         $cantElemsEnThreshold \leftarrow cantElemsEnThreshold + 1$ 
16:      else
17:         $cantElemsEnThreshold \leftarrow 0$ 
18:      end if
19:       $iteraciones \leftarrow iteraciones + 1$ 
20:       $\lambda \leftarrow \lambda_1$                                           $\triangleright$  Acutaliza  $\lambda_i$ 
21:    end while
22:    if  $iteraciones == maxIter$  then
23:       $intentos \leftarrow intentos + 1$ 
24:    else
25:      break
26:    end if
27:  end while
28:  if  $intentos > maxRepeticiones$  then return método fallo
29:  end if
30:  return  $\lambda, s$ 
31: end procedure
```

---

El algoritmo toma una matriz cuadrada  $m$ , un threshold  $\epsilon$  que actúa como un nivel de precisión

del algoritmo, una máxima cantidad de iteraciones permitida, una máxima cantidad de repeticiones a ejecutar el método si el algoritmo no convergió, y finalmente un  $k$  que cuenta la cantidad de elementos que difieren menos del threshold para considerar que la sucesión está suficientemente cercana del autovector real.

En la práctica estos son demasiados parámetros para tener en cuenta siempre, por lo que toman como parámetros default un threshold de  $10^{-6}$ , un maxIter de  $n^3$  con  $n$  las columnas de la matriz, una máxima cantidad de repeticiones de 5 y un  $k$  de 30.

## 2.2. Método de la potencia con deflación

Se combina el método de la potencia con threshold y la deflación de las matrices para producir un algoritmo que calcule todos los autovectores y autovalores de la matriz.

---

```

1: function AUTOVECTORES( $M$ )
2:    $Eigens \leftarrow []$                                       $\triangleright$  Array que contiene el par [autovalor, autovector]
3:    $\epsilon \leftarrow 10^{-6}$ 
4:    $maxIter \leftarrow n^3$ 
5:    $maxRepeticiones \leftarrow 5$ 
6:    $k \leftarrow 30$ 
7:   for  $i \leftarrow 0$  to  $n$  do
8:      $Eigens[i] \leftarrow$  métodoDeLaPotencia( $M, \epsilon, maxIter, maxRepeticiones, k$ )
9:      $M \leftarrow M - Eigen[i][0], Eigen[i][1], Eigen[i][1]^T$             $\triangleright$  Aplica deflacion
10:  end for
11:  return  $Eigens$ 
12: end function

```

---

Este algoritmo itera  $n$  veces sobre la cantidad de columnas o filas de  $M$ . En cada iteración corre el método de la potencia con threshold implementado previamente usando los parámetros *default* mencionados en la sección anterior. Luego aplica deflación sobre la matriz. La siguiente iteración aplica el algoritmo para obtener el autovector y valor dominante sobre la matriz con deflación de la iteración previa.

Estudiamos su comportamiento respecto al del método de Eigen para matrices simétricas arbitrarias:

### 2.2.1. Fiabilidad del algoritmo

Queremos comparar el comportamiento del algoritmo para calcular autovectores y valores con un algoritmo más sofisticado como es el de Eigen. Generamos 200 matrices simétricas de 100x100 con cada coordenada sampleada de una uniforme  $U([-1, 1])$  y tomamos como *error* a la norma 2 de la diferencia de autovalores de nuestro método con el de Eigen ordenados de mayor a menor. Usamos un threshold de  $10^{-6}$  y una cantidad de valores de tolerancia de 30. Corremos el método hasta 5 veces. Lo graficamos en un histograma:

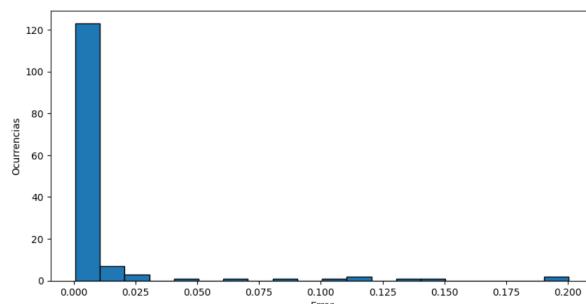


Figura 4: Histograma de error respecto al autovalor returned por Eigen

---

Vemos cómo para las 200 matrices, la mayor parte de la diferencia entre autovalores está por debajo de 0.03. Se observan unos cuantos outliers con un error que llega hasta un orden de magnitud mayor al resto. Estudiando los autovalores de la matriz simétrica generada con Eigen nos encontramos cómo para todos estos casos (error mayor a 0.03) hay autovalores que en valor absoluto difieren recién en el tercer o cuarto decimal! Como vimos originalmente estas matrices requieren un número muy elevado de iteraciones para converger siempre, por lo que tiene sentido que a pesar de correr el algoritmo hasta 5 veces 10000 iteraciones, no llega a un resultado tan preciso como el resto.

De cualquier forma vemos que el comportamiento es muy similar al de Eigen, y que incluso en las matrices poco óptimas, el método da resultados con un error muy bajo. Consideramos que el algoritmo es fiable y robusto para obtener una muy buena aproximación de autovalores y vectores.

### 3. Procesamiento de imágenes

Estudiamos una aplicación del método de la potencia con deflación al procesamiento y compresión de imágenes con los algoritmos de PCA y 2DPCA. Se observa para cada caso distintos resultados y experimentaciones sobre un espacio muestral de imágenes de 112x92 de caras frontales.

#### 3.1. PCA

Se representa cada una de las imágenes como vectores de 10304 coordenadas. Presentamos una implementación de PCA y observamos como son las componentes principales de nuestro vector aleatorio imagen de caras. Luego vemos como se reconstruyen las imágenes a partir de las componentes principales y experimentamos con los autovalores de los componentes. Finalmente hacemos un estudio de la similaridad entre imágenes y el error de compresión del método.

##### 3.1.1. Implementacion

Con el fin de minimizar el tiempo de cómputo para comprimir imágenes, la implementación que presentamos de PCA sobre un conjunto de vectores se divide en dos rutinas. La primera calcula los principales parámetros estadísticos y componentes principales de una muestra, y la segunda proyecta un conjunto de vectores sobre esos componentes.

La rutina 1 toma un conjunto de muestras de vectores representados como una matriz  $X$  donde cada fila es una muestra. Retorna estimaciones del vector medio  $\mu$  y la desviación estandar  $\sigma$  para cada coordenada puntual, la matriz de covarianza  $\Sigma$  y los autovalores  $\Lambda$  y autovectores  $V$  de esa matriz. Para calcular los autovectores se usa el algoritmo presentado en la sección anterior con los parámetros *default*.

---

##### Algorithm 3 Rutina 1: Calcula los componentes principales

---

```
1: procedure MÉTODODELAPOTENCIASOBREMATRIZDECORVARIANZA( $X$ )
2:    $\mu \leftarrow mean(X)$                                  $\triangleright$  Calcula el promedio muestral para cada coordenada
3:    $\sigma \leftarrow sd(X)$                                  $\triangleright$  Calcula la desviación muestral para cada coordenada
4:    $X_C \leftarrow center(X, \mu)$                        $\triangleright$  Le resta a cada fila de  $X$  el promedio muestral
5:    $\Sigma \leftarrow \frac{X_C^t X_C}{n-1}$                    $\triangleright$  Estima la matriz de covarianza
6:    $\Lambda, V \leftarrow Autovectores(\Sigma)$ 
7:   return  $\mu, \sigma, \Lambda, V$ 
8: end procedure
```

---

Se asume que la Rutina 2 ya corrió la 1 para un conjunto de muestras independientes e idénticamente distribuidas, y que todo lo que retorna son variables globales. Esta toma un conjunto  $X$  de vectores, un número  $k$  y retorna las primeras  $k$  coordenadas de  $X$  en la base ortonormal de autovectores que retorna el algoritmo anterior.

---

##### Algorithm 4 Rutina 2: Proyección de las imágenes en la base ortonormal de autovectores

---

```
1: procedure PCA( $X, k$ )
2:    $m \leftarrow$  número de imágenes en  $X$ 
3:   for  $i \leftarrow 1$  to  $m$  do
4:      $X_i \leftarrow (X_i - \mu)/\sigma$                        $\triangleright$  Normalizar la imagen
5:      $X_{\text{proj}} \leftarrow X_i \cdot_k V$                  $\triangleright$  Proyecta  $X_i$  sobre  $k$  componentes de la base  $V$ 
6:   end for
7:   return  $X$                                        $\triangleright$  Retornar las imágenes transformadas
8: end procedure
```

---

La complejidad del primer método es de  $O(n^6)$  siendo  $n$  la cantidad de coordenadas del vector aleatorio. Esto se debe a que el método de la potencia tiene una complejidad de  $O(n^2k)$  siendo  $k$  la cantidad de iteraciones, y en el caso de threshold se limitan estas iteraciones hasta  $O(n^3)$ . Luego se ejecuta el método de la potencia hasta  $n$  veces por lo que la complejidad termina siendo  $O(n^6)$ , que es en el peor caso extremadamente alto.

La gran utilidad de separar el método en dos rutinas es que la primera puede considerarse como un *estudio* estadístico del vector aleatorio a partir de muchas muestra. A pesar de que puede potencialmente tardar horas en obtenerse los autovectores, una vez se tienen los componentes principales, la segunda rutina calcula todo en una complejidad mucho menor de  $O(mn^2k)$ , siendo  $k$  la cantidad de componentes y  $m$  la cantidad de imágenes. Esto se debe a que obtener la proyección de un vector en una base es  $O(n^2)$  al hacer  $n$  productos internos donde cada uno tiene un tiempo de cómputo  $O(n)$ .

En la siguiente sección vemos como son los componentes principales, denominadas eigenfaces, de nuestro espacio muestral de caras.

### 3.1.2. Eigenfaces

En el contexto del análisis de componentes principales (PCA), las eigenfaces son los autovectores de la matriz de covarianza de las imágenes de rostros. En otras palabras, son las direcciones en las cuales las imágenes de rostros tienen la mayor varianza.

Cada eigenface representa una característica en la variación de las imágenes de rostros. Por ejemplo, una eigenface puede representar la variación en la iluminación, otra puede representar la variación en la orientación de la cara, etc.

Graficamos en la figura 5 las primeras 5 eigenfaces desaplanadas a matrices de 112x96

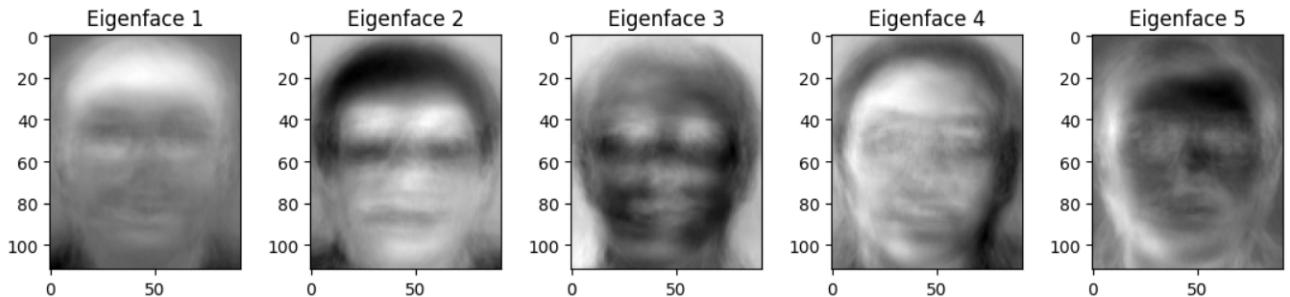


Figura 5: Primeras 5 eigenfaces usando PCA

Consistente con la noción de *componente principal* del espacio muestral, todos los autovectores de la matriz de covarianza preservan la forma de cara. Observamos ahora cuál es el peso de cada eigenface en la varianza total de las caras.

### 3.1.3. Autovalores y Varianza de los componentes principales

A medida que aumentamos la cantidad de componentes usadas, captamos más varianza. Esto se debe a que los componentes principales como vectores aleatorios son independientes, y la varianza que aporta cada componente es su autovalor. Graficamos en la figura 6 la varianza captada por los primeros 800 componentes. Observamos como a partir del autovalor del componente 409 estos decaen de un orden de  $10^{-2}$  a  $10^{-7}$ , por lo que el algoritmo (que funciona a una precisión de  $10^{-6}$ ) no puede asegurar que ese sea una buena aproximación al autovalor. Comparando con numpy, este retorna un autovalor de  $2,39 \times 10^{-15}$  en el índice 410.

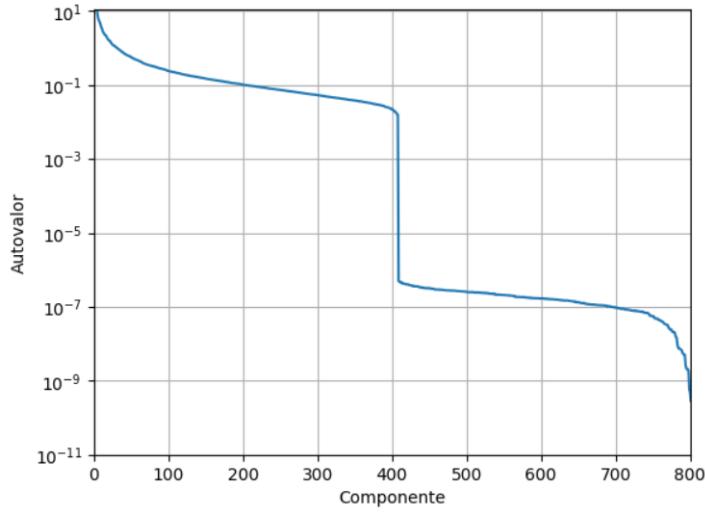


Figura 6: Autovalor en el eje y correspondiente al componente x-esima

El % de varianza total que acumulas los primeros se calcula como  $100 \times S_k/T$  donde  $S_k$  es la suma de autovalores hasta el componente k-esimo y  $T$  es la suma total. Si bien no podemos asegurar que los autovectores a partir de 409 sean correctos, si podemos asegurar que los primeros 409 autovectores abarcan virtualmente el 100 % de la varianza de las imágenes, ya que  $T$  está acotado por  $S_{409} + (2,39 \times 10^{-15}) \times 9634$ . Esto determina que basta estudiar la proyección de las imágenes hasta el componente 409 para obtener una representación casi perfecta. Comprobamos esto en la siguiente sección.

### 3.1.4. Reconstrucción de imágenes

En la Figura 7 se puede ver la reconstrucción de 4 rostros a partir de una cantidad creciente de componentes principales. Estas se reconstruyen sumando la combinación lineal de las eigenfaces multiplicadas por su respectivo componente.

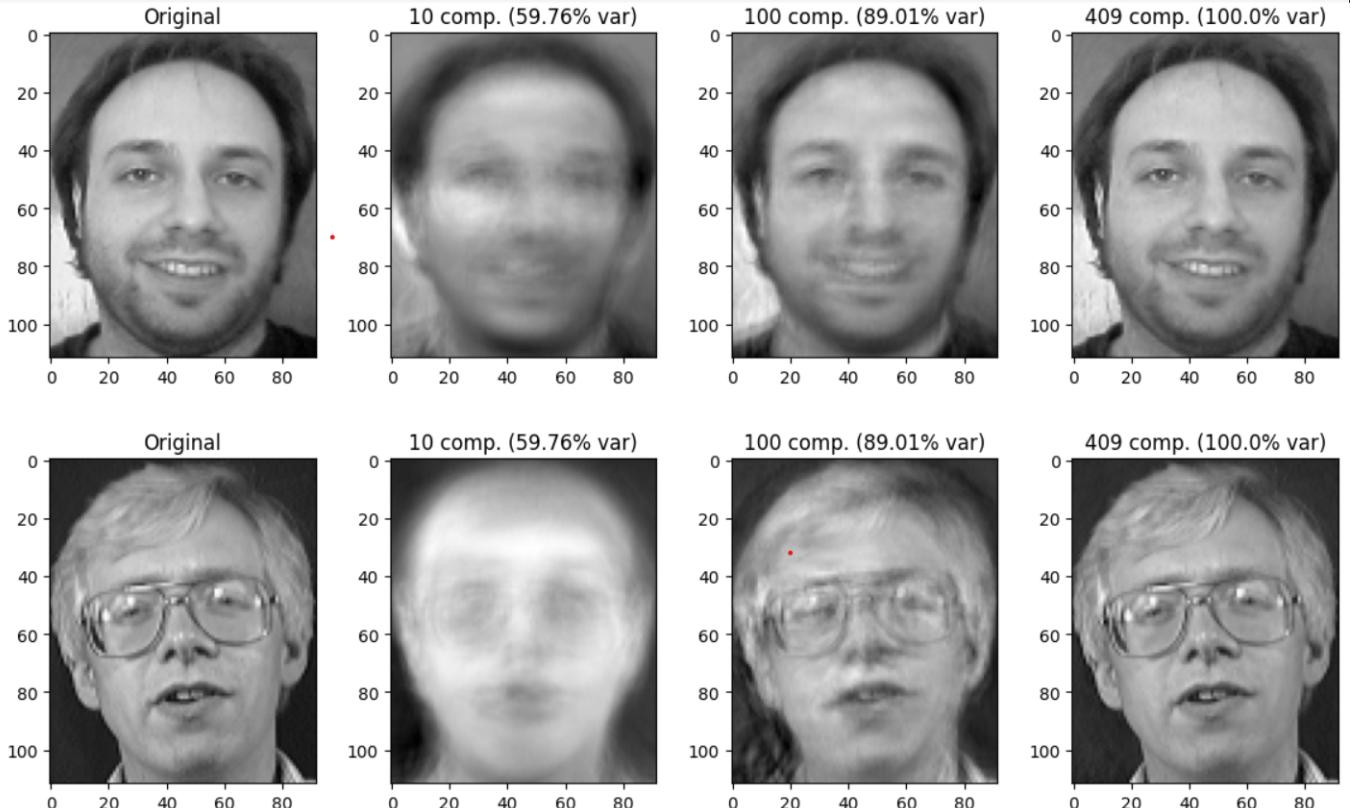


Figura 7: Reconstrucción de 2 caras usando cada vez más componentes

Consistente con el resultado previo, para la componente 409 la imagen es indistinguible de la original. Esto representa un 4 % de la cantidad total de componentes principales. Es decir, se puede reconstruir cualquier cara del dataset con un 4 % los autovectores de la matriz de covarianza. Este es un resultado muy interesante y útil, ya que permite comprimir imágenes compuestas por 10343 números a 409, una compresión del 96 %.

### 3.1.5. Similaridad

Una métrica razonable de similaridad para un vector aleatorio es la matriz de correlación estimada a partir de ese grupo. Vemos en esta sección como se compara la similaridad del dataset global con la similaridad de las caras de una persona. Esto lo hacemos para imágenes de dimensionalidad reducida utilizando PCA para distintos componentes. Graficamos para distintos componentes la matriz de correlación del dataset global y el *promedio* de las matrices de correlación estimadas para las caras de cada persona individualmente.

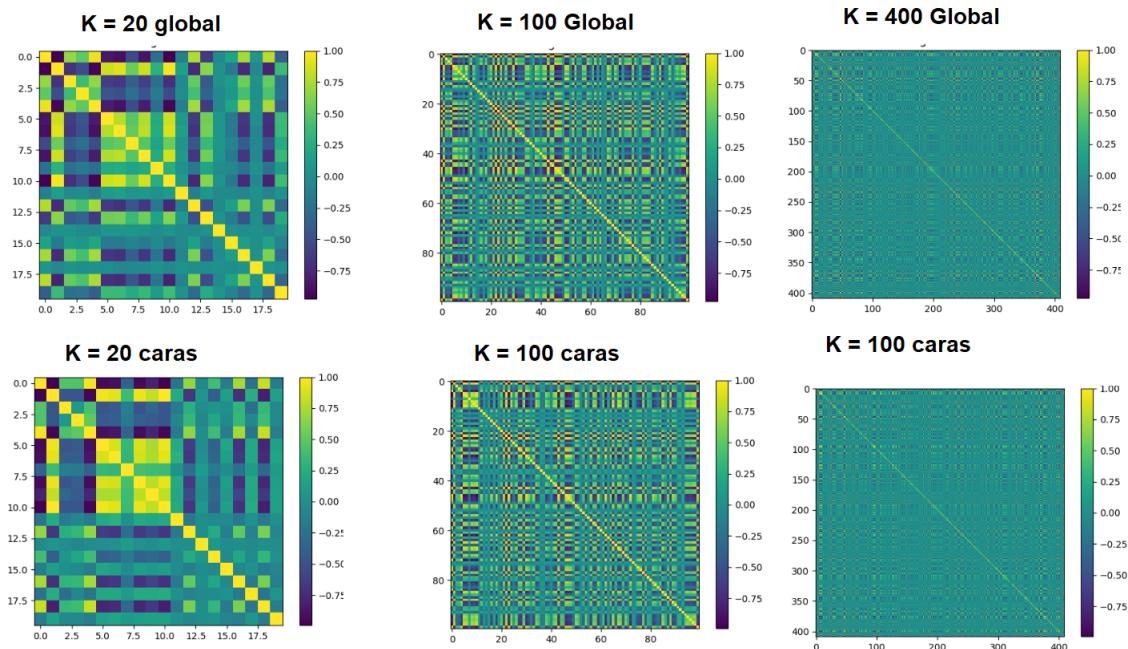


Figura 8: Matriz de Correlación de Imágenes Centradas para distintos k

Como es razonable, las matrices de correlación estimadas a partir de un conjunto global de personas y sobre una única cara en distintas posiciones es casi indistinguible.

### 3.1.6. Error de compresión

Queremos ver el error de compresión de una imagen al correr PCA en el caso donde se obtuvieron los componentes *utilizando* a esa imagen, y en el caso cuando no se la utilizo. Para medir el error de compresión usamos una métrica en la cual restamos la imagen reconstruida a la original y le tomamos valor absoluto. Graficamos el resultado para dos  $k$  (cantidad de componentes) distintas usando imágenes de la mitad de la dimensión original:

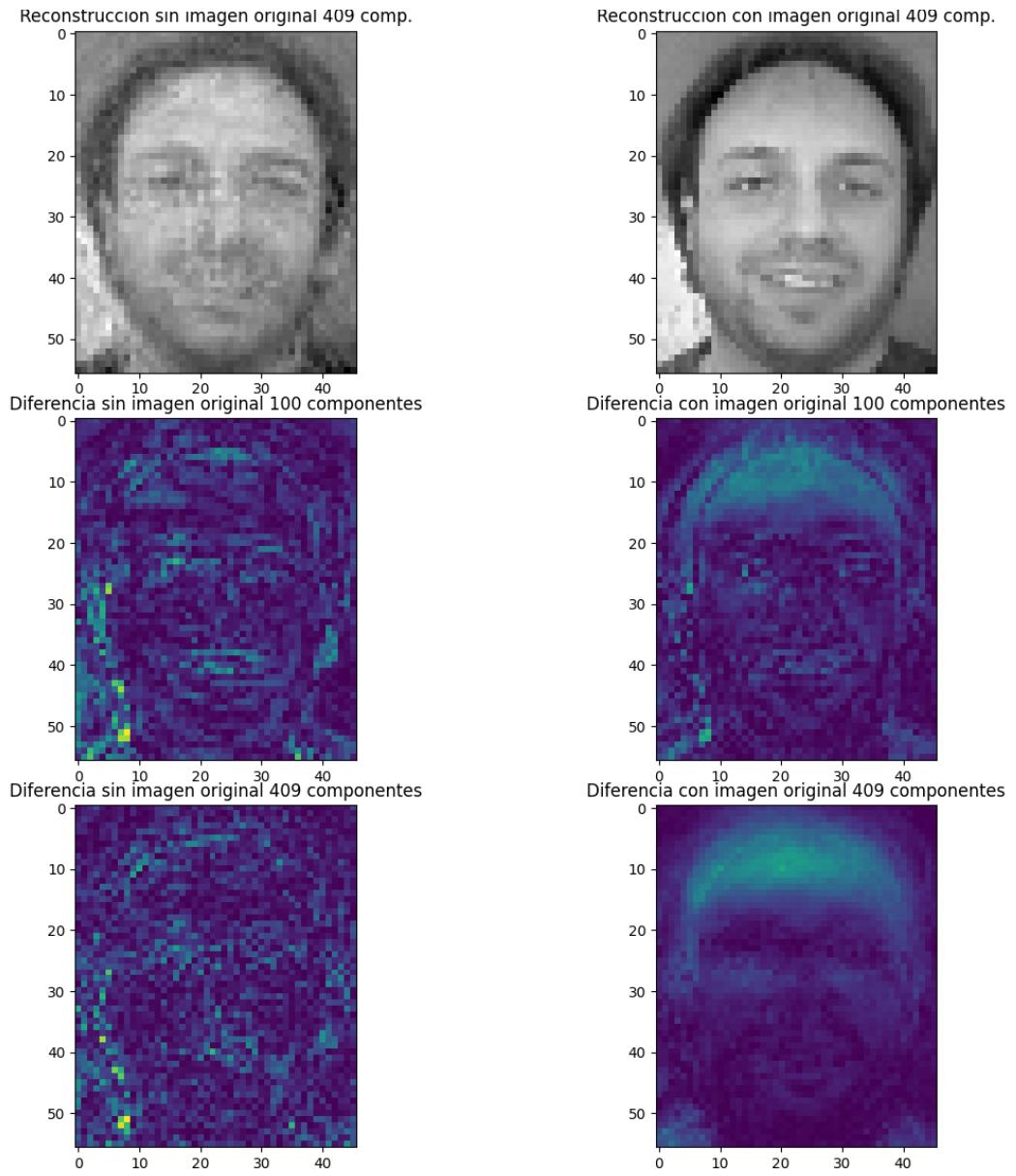


Figura 9: Error de compresión de una cara que fue comprimida y descomprimida usando una matriz de covarianza que se estimo con sin la cara vs el error con una matriz que se estimo con la cara.

Observamos como el efecto de no tener la imagen en el conjunto de muestras afecta considerablemente la calidad. Concluimos que PCA funciona mejor cuando la imagen a comprimir fue parte del estudio estadístico con el que se calculo la matriz de covarianza.

## 3.2. 2DPCA

Se representa cada una de las imágenes como matrices de 112 x 96. Presentamos una implementación de 2dPCA y observamos como son las componentes principales de nuestra matriz aleatoria imagen de caras. Luego vemos como se reconstruyen las imágenes a partir de las componentes principales y experimentamos con los autovalores de los componentes. Finalmente hacemos un estudio de la similaridad entre imágenes y el error de compresión del método.

### 3.2.1. Implementación

En 2DPCA buscamos disminuir las dimensiones de la matriz con la que trabajamos. Inicialmente, cargamos las imágenes en un arreglo de imágenes  $X$ . Luego construimos la matriz de covarianza de la imagen  $G$  (3) y le calculamos los autovectores. Al igual que en PCA dividimos el algoritmo en 2 rutinas. La primera calcula los *parámetros estadísticos* de la muestra -matriz de covarianza, sus autovectores, media, etc.- y la segunda obtiene los primeros  $k$  feature vectors.

La primera rutina toma un conjunto  $X$  de imágenes en forma matricial

---

#### Algorithm 5 Rutina 1: Calcula los componentes principales

---

```

1: procedure MÉTODO DELA POTENCIA SOBRE MATRIZ DE COVARIANZA( $X$ )
2:    $\mu \leftarrow \text{mean}(X)$                                  $\triangleright$  Calcula el promedio muestral para cada coordenada
3:    $\sigma \leftarrow \text{sd}(X)$                                  $\triangleright$  Calcula la desviación muestral para cada coordenada
4:    $G \leftarrow (X_1 - \mu)^T(X_1 - \mu)$ 
5:   for  $i \leftarrow 2$  to  $|X|$  do
6:      $G \leftarrow G + (X_i - \mu)^T(X_i - \mu)$                  $\triangleright$  Calcula la matriz de correlación de imágenes
7:   end for
8:    $\Lambda, V \leftarrow \text{Autovectores}(G)$ 
9:   return  $\mu, \sigma, \Lambda, V$ 
10: end procedure
```

---

Se asume que la Rutina 2 ya corrió la 1 para un conjunto de muestras independientes e idénticamente distribuidas, y que todo lo que retorna son variables globales. Esta toma un conjunto  $X$  de vectores, un número  $k$  y retorna los primeros  $k$  feature vectors de cada imagen.

---

#### Algorithm 6 Rutina 2: Feature vectors de un conjunto $X$ de imágenes

---

```

1: procedure 2DPCA( $X, k$ )
2:    $m \leftarrow$  número de imágenes en  $X$ 
3:    $F = []$ 
4:   for  $i \leftarrow 1$  to  $m$  do
5:      $X_i \leftarrow (X_i - \mu)/\sigma$                                  $\triangleright$  Normalizar la imagen
6:      $X_{\text{proy}} \leftarrow X_i \cdot_k V$                              $\triangleright$  Retorna  $X_i V_j$  en un vector  $(X_i V_j, \dots, X_k V_j)$ 
7:      $F[i] \leftarrow X_{\text{proy}}$ 
8:   end for
9:   return  $F$                                                $\triangleright$  Retorna las imágenes transformadas
10: end procedure
```

---

La complejidad del primer método es de  $O(m^6)$  siendo  $m$  la cantidad de columnas de la imagen. Esto se debe a que el método de la potencia tiene una complejidad de  $O(m^2k)$  siendo  $k$  la cantidad de iteraciones, y en el caso de threshold se limitan estas iteraciones hasta  $O(m^3)$ . Luego se ejecuta el método de la potencia hasta  $m$  veces por lo que la complejidad termina siendo  $O(m^6)$ . Si bien es alto, pca tiene una complejidad  $O(mn^2)$  que es incomparablemente mayor. Esta es la gran ventaja de 2dPCA, el estudio estadístico del espacio de imágenes toma mucho menos tiempo.

La segunda rutina calcula todo en una complejidad mucho menor de  $O(xm^2n^2k)$ , siendo  $k$  la cantidad de componentes,  $n$  la cantidad de filas y  $x$  la cantidad de imágenes. Esto se debe a que cada

multiplicación  $X_i V_j$  toma  $O(n^2 m^2)$ , y se ejecutan  $k$  de esas operaciones por imagen. Luego se repite  $x$  veces. Este tiempo es igual al de PCA.

En la siguiente sección vemos como son los componentes principales, denominadas eigenfaces, de nuestro espacio muestral de caras.

### 3.2.2. Eigenfaces

Se grafican las subimagenes asociadas a una cara, ya que cada feature vector existe en relacion a una cara.

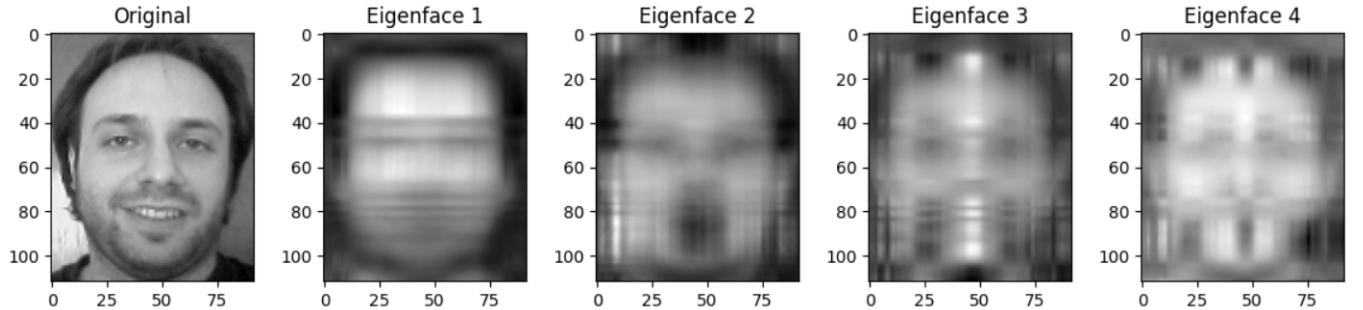


Figura 10: Primeras 4 eigenfaces asociada a una cara en 2dPCA

A diferencia de pca original, estas subimagenes no preservan de forma tan explicita la forma de cara.

### 3.2.3. Autovalores y Varianza de los componentes principales

Tomando los autovalores que calculamos de la matriz  $G$ , primero generamos un gráfico que muestra el valor de los autovalores en función de su índice, ordenados por tamaño.

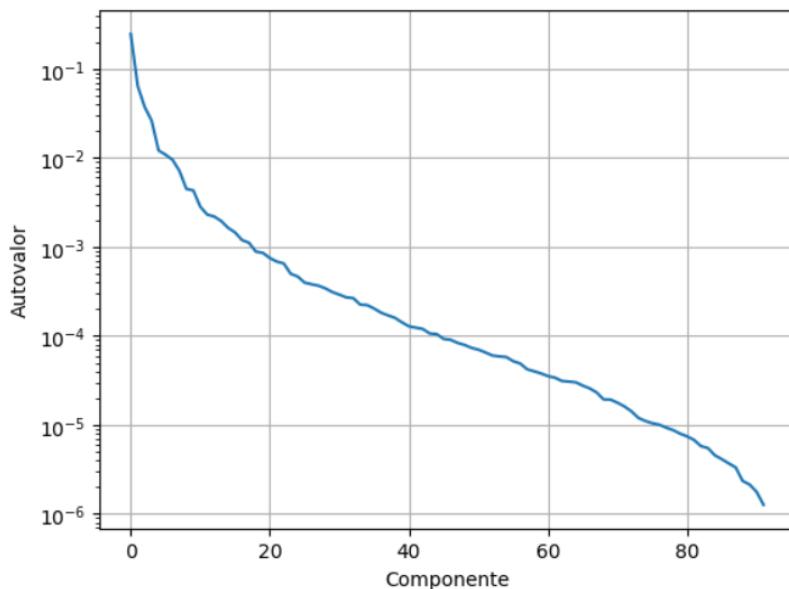


Figura 11: Autovalor en el eje y correspondiente al componente x-esima

Se observa que los primeros autovalores tienen valores significativamente más altos que los siguientes. A medida que aumenta el índice de los autovalores, su valor disminuye de manera pronunciada, indicando que contribuyen con menos información a la varianza total de los datos. A diferencia de PCA, no se observa un decaimiento tan pronunciado. Vemos como se traduce esto a la reconstrucción de imágenes.

### 3.2.4. Reconstrucción de imágenes

La siguiente figura muestra la reconstrucción de cuatro rostros a partir de una cantidad creciente de componentes principales. La reconstrucción de una imagen a partir de los componentes principales implica sumar las combinaciones lineales de las eigenfaces. Cada eigenface se pondera por el valor del componente principal correspondiente para esa imagen. Cabe destacar que hay hasta 92 componentes posibles.

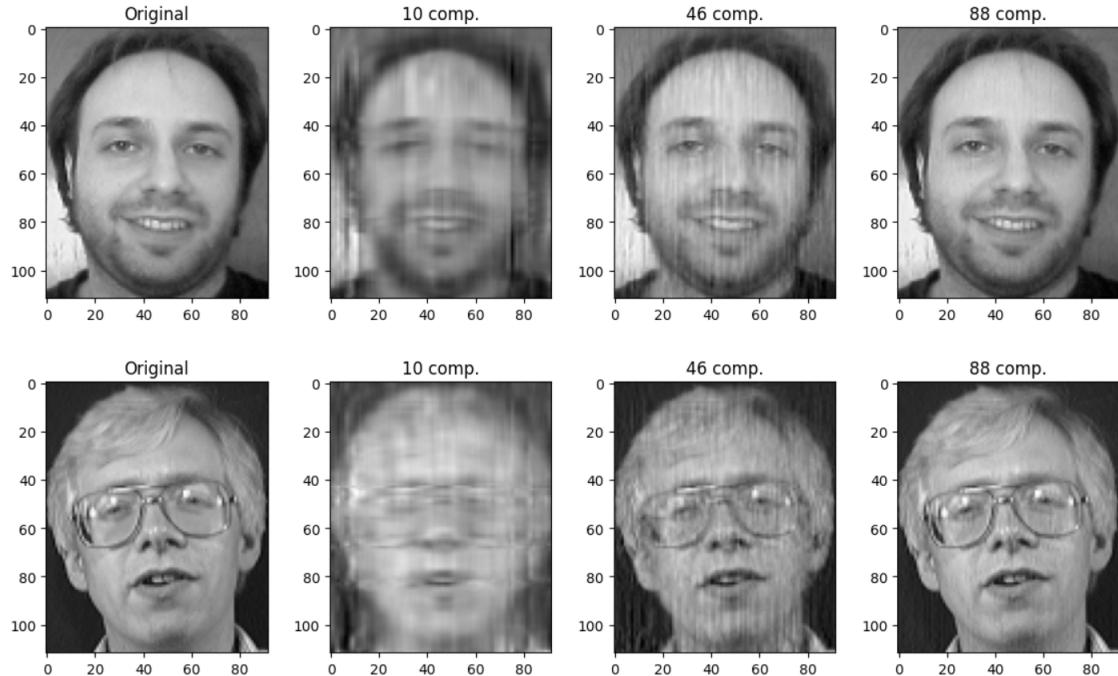


Figura 12: Reconstrucción de 2 caras usando cada vez más componentes

A diferencia del PCA, el 2DPCA necesita todas las componentes principales para reconstruir la imagen de manera que sea indistinguible de la original. A pesar de que las primeras 46 componentes abarcan el 99.3% de la suma de los autovalores, la imagen reconstruida sigue perdiendo bastante calidad.

Esto indica que, como esperábamos, no se puede dar una interpretación a los autovalores como determinantes del aporte a la varianza total. Consideramos que 2DPCA no es tan eficiente como PCA para la compresión de imágenes, ya que el almacenamiento de 92 componentes ocupa el mismo espacio que almacenar la imagen completa.

### 3.2.5. Similaridad

Una métrica razonable de similaridad para un grupo de vectores es la matriz de correlación estimada a partir de ese grupo. Vemos en esta sección cómo se compara la similaridad del dataset global con la similaridad de las caras de una persona. Esto lo hacemos para imágenes de dimensionalidad reducida utilizando 2DPCA para distintos componentes. Graficamos para distintos componentes la matriz de correlación del dataset global y el *promedio* de las matrices de correlación estimadas para las caras de cada persona individualmente.

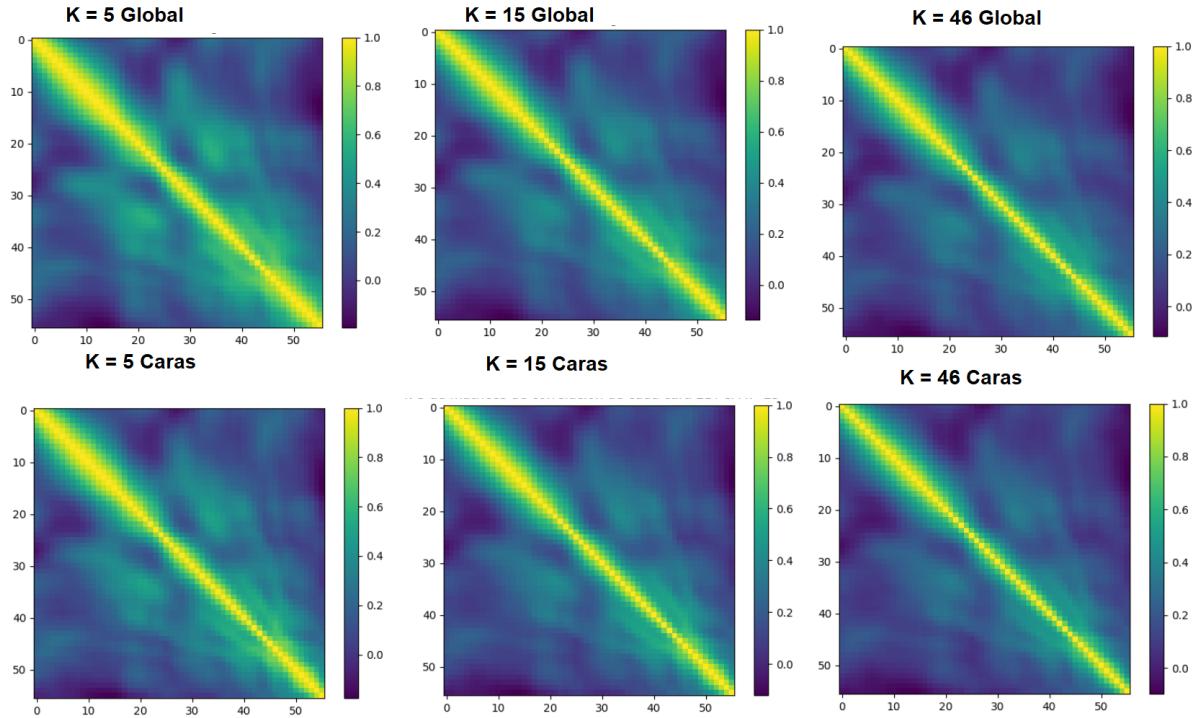


Figura 13: Matriz de Correlación de Imágenes Centradas para distintos  $k$

Como es razonable, las matrices de correlacion estimadas a partir de un conjunto global de personas y sobre una unica cara en distintas posiciones son casi indistinguibles.

### 3.2.6. Error de compresión

Queremos ver el error de compresión de una imagen al correr 2DPCA en el caso donde se obtuvieron los componentes *utilizando* a esa imagen, y en el caso cuando no se la utilizo. Para medir el error de compresión usamos una métrica en la cual restamos la imagen reconstruida a la original y le tomamos valor absoluto. Graficamos el resultado para dos  $k$  (cantidad de componentes) distintas usando imagenes de la mitad de la dimensión original:

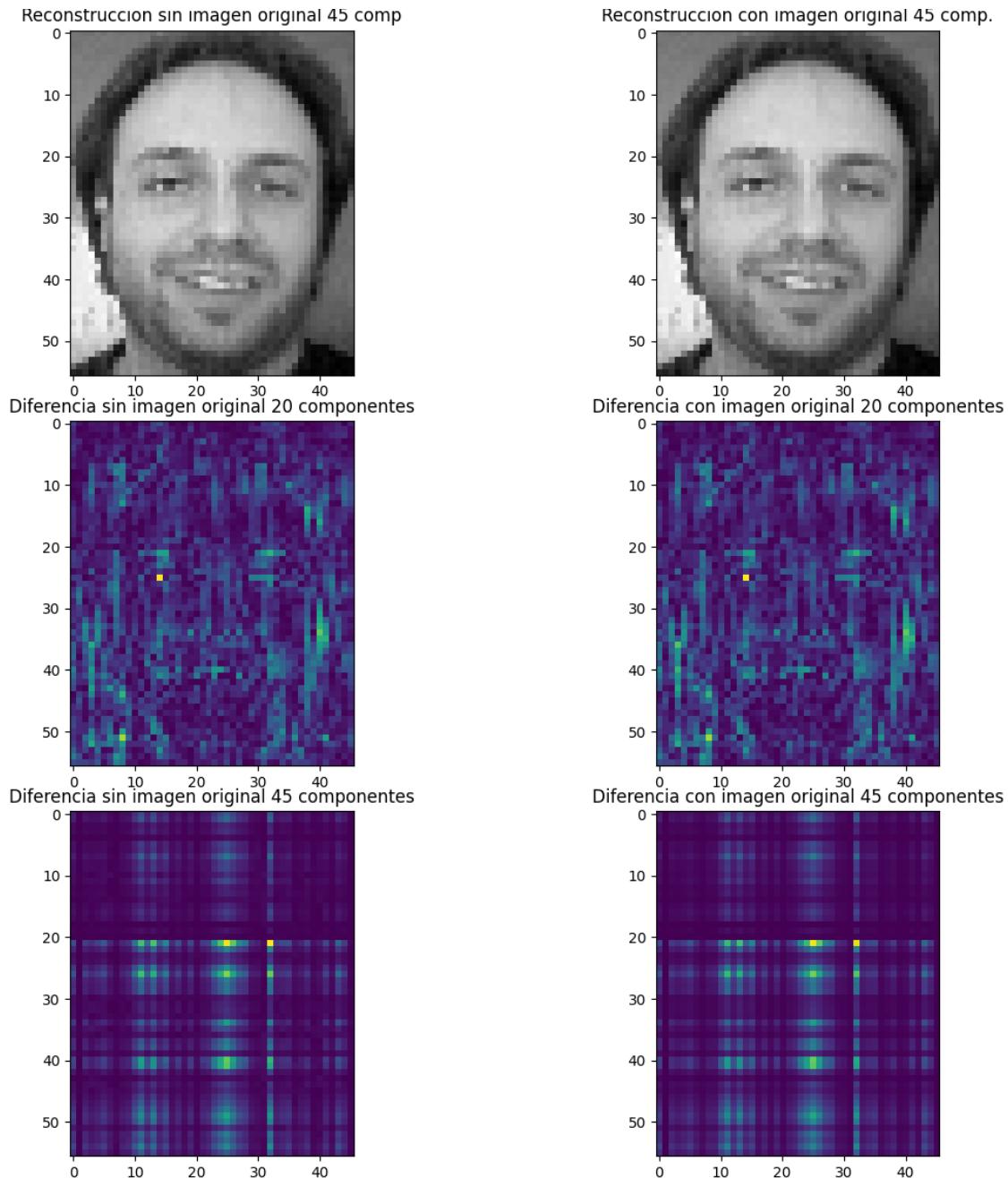


Figura 14: Error de compresión de una cara que fue comprimida y descomprimida usando una matriz de covarianza que se estimo con sin la cara vs el error con una matriz que se estimo con la cara.

Consistente con lo obtenido en secciones anteriores, 2dPCA tiene mayor error de compresión que PCA. De cualquier forma, el haber removido las imágenes de la persona no afectó a su error de compresión, lo que determina que 2DPCA es mucho más estable en términos de lo que aumenta el error de compresión al no tener una imagen en el dataset con el que se calculó los parámetros estadísticos.

---

## 4. Conclusiones

Experimentamos sobre una implementación naive del método de la potencia. Validamos su complejidad temporal de  $O(kn^2)$  siendo  $n$  la cantidad de filas de la matriz y  $k$  la cantidad de iteraciones y observamos como la fiabilidad del método depende excesivamente de un  $k$  que a priori no se conoce. Para solucionar esto presentamos una implementación mas robusta que itera todo lo necesario hasta llegar a un valor de precisión especificado de antemano. Utilizando esa implementación, armamos un algoritmo utilizando deflación para calcular todos los autovectores y valores de una matriz diagonalizable, obteniendo un rendimiento en términos de error comparable con el de Eigen.

Aplicamos ese algoritmo para el procesamiento y compresión de imágenes utilizando PCA y 2DPCA, ambos implementados en base a dos rutinas. La primer rutina hace un análisis estadístico de un conjunto de muestras, obteniendo así los componentes principales estimados. La segunda rutina proyecta un conjunto de imágenes a esas componentes principales. Para nuestras muestras de imágenes de frente de caras de 112x96, PCA logra una compresión del 96 %, mientras que 2DPCA no pudo reducir de forma significativa la dimensionalidad de la imagen.

## Referencias

- [1] [Metodo de la potencia con deflacion](#)
- [2] [PCA](#)
- [3] [2DPCA](#)
- [4] [Eigen eigenvalues](#)