

---

# **TOOLBOX DE APLICACIONES DE LA NORMA DE FROBENIUS**

## **En procesamiento de imágenes**

---

**Pablo Soto-Quirós**  
Instituto Tecnológico de Costa Rica

**Jeffry Chavarría-Molina**  
Instituto Tecnológico de Costa Rica

**Juan José Fallas-Monge**  
Instituto Tecnológico de Costa Rica



# CONTENIDOS

---

Lista de símbolos	v
Introducción	vii
<b>1 Funciones globales del toolbox</b>	<b>1</b>
1.1 ReadImage DataBase	1
1.1.1 Sintaxis	2
1.2 Matrix2Image	2
1.2.1 Sintaxis	2
1.3 double2uint8	2
1.3.1 Sintaxis	2
1.4 Video2Matrix	2
1.4.1 Sintaxis	3
1.5 Matrix2Video	3
1.5.1 Sintaxis	3
1.6 img_miss_pix	3
1.6.1 Sintaxis	4
<b>2 Filtro para remover ruido</b>	<b>5</b>
2.1 RankConstrainedFilterX	6
2.1.1 Sintaxis	6
2.2 NoiseFunction	6

iii

2.2.1	Sintaxis	7
2.2.2	Sintaxis alternativa	7
2.3	FullRankConstrainedFilterX	8
2.3.1	Sintaxis	8
2.3.2	Sintaxis alternativa	8
2.4	SaveBlurredImages	9
2.4.1	Sintaxis	9
2.4.2	Sintaxis alternativa	9
2.5	SaveFilteredImages	10
2.5.1	Sintaxis	10
<b>3</b>	<b>Factorización no negativa</b>	<b>11</b>
3.1	nnfLS	13
3.1.1	Sintaxis	14
3.2	ConstructionNNFM	15
3.2.1	Sintaxis	15
3.3	ReconstructionExternalFace	15
3.3.1	Sintaxis	16
3.4	ReconstructionExternalFacesDirectory	16
3.4.1	Sintaxis	16
<b>4</b>	<b>Proyecciones aleatorias bilaterales</b>	<b>17</b>
4.1	LowRankMatrixBRP	20
4.1.1	Sintaxis	20
4.2	ImageCompression	20
4.2.1	Sintaxis	20
<b>5</b>	<b>GoDec</b>	<b>21</b>
5.1	GoDec	24
5.1.1	Sintaxis	25
5.2	GoDecVideoFull	26
5.2.1	Sintaxis	26
5.3	GoDecImageFull	26
5.3.1	Sintaxis	27
<b>6</b>	<b>K-SVD</b>	<b>29</b>
6.1	ksvd	32
6.1.1	Sintaxis	32
6.2	clean_ksvd	33
6.2.1	Sintaxis	33
Referencias		35

# SIMBOLOS

---

$\mathbb{C}^m$	Conjunto de vectores de entradas complejas con $m$ entradas
$\mathbb{C}^{m \times n}$	Conjunto de matrices de entradas complejas con $m$ filas y $n$ columnas
$\ X\ _{fr}$	Norma de Frobenius de la matriz $X$
$\mathbb{C}_k^{m \times n}$	Conjunto de las matrices de $m$ filas, $n$ columnas y de rango a lo sumo $k$
$A^\dagger$	Pseudoinversa de Moore-Penrose de la matriz $A$
$A^*$	Transpuesta conjugada de la matriz $A$
$I_n$	Matriz identidad de orden $n$
$A_{m \times n}$	Matriz de $m$ filas y $n$ columnas
SVD	Descomposición en valores singulares
$\text{rank}(A)$	Rango de la matriz $A$
$\text{card}(A)$	Cardinalidad de la matriz $A$ . Cantidad de entradas distintas de cero.



# INTRODUCCIÓN

---

El presente manual ofrece una descripción detallada del Toolbox de aplicaciones de la norma de Frobenius en procesamiento de imágenes. De manera sintetizada, el toolbox aborda las siguientes aplicaciones de la norma de Frobenius, las cuales están organizadas a partir del Capítulo 2 de la siguiente manera:

1. **Capítulo 2:** Construcción de filtros para remover ruido de una imagen.
2. **Capítulo 3:** Factorización no negativa de matrices aplicada a la reconstrucción de imágenes.
3. **Capítulo 4:** Aproximación de la matriz de rango bajo mediante el algoritmo BRP y su aplicación a la compresión de imágenes
4. **Capítulo 5:** Algoritmo GoDec aplicado a la detección de variaciones en un conjunto de imágenes.
5. **Capítulo 6:** Algoritmo K-SVD aplicado a la reconstrucción de imágenes con pixeles perdidos.

Como todas las aplicaciones anteriores se relacionan con procesamiento de imágenes, es fundamental tener clara la forma de cómo se representa una imagen computacionalmente. Además de cómo proceder con la representación en caso que se requiera trabajar con una base de imágenes. Sobre esto se detallará a continuación con el objetivo de que el lector se familiarice con la representación matricial que se utilizará en las funciones del toolbox. En concreto, en la Figura I.1 se muestra la representación matricial de un bloque de una imagen en escala de grises. La imagen I.2 muestra el proceso que se debe seguir para

vectorizar, como vector columna, la matriz anterior. Si se realiza el proceso mostrado en las figuras I.1 y I.2, pero a la imagen completa, se termina con una representación vectorizada de toda la imagen (ver [1]).

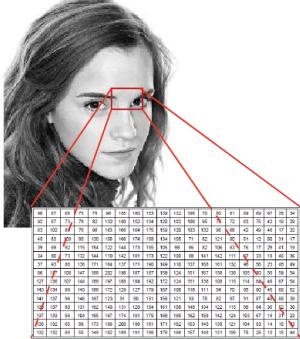


Figura I.1: Representación matricial de un bloque de una imagen.

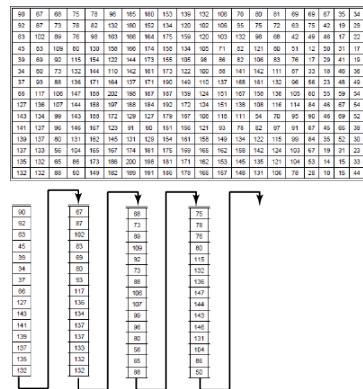
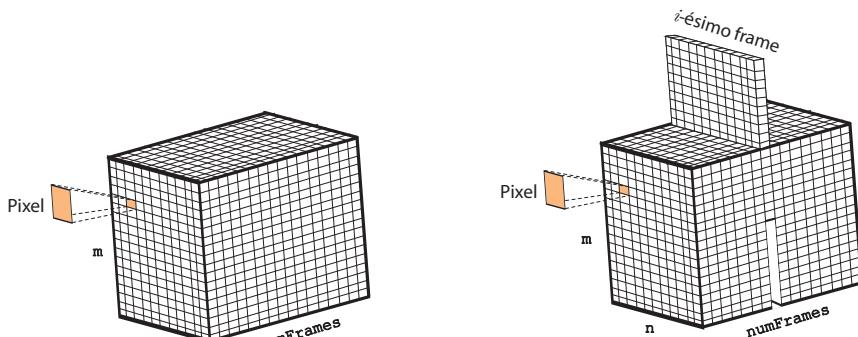


Figura I.2: Vectorización (como columna) de la representación matricial.

Si se cuenta con una base de datos de imágenes, en escala de grises, todas del mismo tamaño, podemos hacer este proceso de vectorización a cada una de las imágenes. A partir de la vectorización de todas la imágenes de la base se diseña una matriz  $A$  de tal manera que la información asociada a la  $i$ -ésima imagen quedará almacenada en la  $i$ -ésima columna de  $A$ . Por lo tanto, si cada imagen de la base tiene una representación matricial de dimensión  $m \times n$ , su vectorización tendrá tamaño  $(mn) \times 1$ . Si en la base existen  $z$  imágenes, entonces la matriz  $A$  es de dimensión  $(mn) \times z$ .

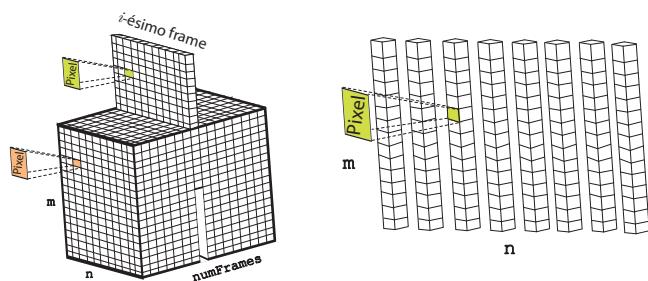
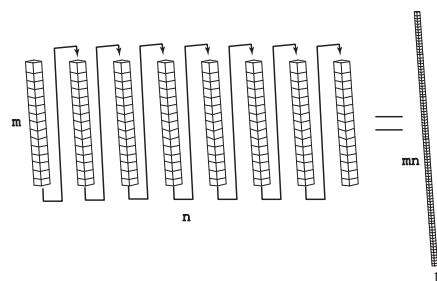
En el caso que se esté procesando un video  $V$ , entonces se sigue una codificación matricial similar. En efecto, suponga que  $V$  es un video con  $\text{numFrames}$  frames, cada uno de tamaño  $m \times n$  pixeles. Matricialmente  $V$  puede representarse mediante un arreglo tridimensional de tamaño  $m \times n \times \text{numFrames}$ , tal como se muestra en la Figura I.3. Por su parte, las Figuras I.4 y I.5 muestran la vectorización que se realiza para el  $i$ -ésimo frame como matriz columna de tamaño  $mn \times 1$ . Si este proceso se repite para todos los frames, el video puede codificarse mediante una matriz  $A$  de tamaño  $mn \times \text{numFrames}$ , la cual se muestra en la Figura I.6.



(a) Representación matricial de un video.

(b) Representación del  $i$ -ésimo frame.

Figura I.3: Representación tridimensional de un video.

Figura I.4: Procesamiento del  $i$ -ésimo frame.Figura I.5: Vectorización del  $i$ -ésimo frame.

$$\begin{aligned} A = & \quad \text{Matriz resultante} \\ mn & \quad \text{Número de elementos} \\ \text{numFrames} & \quad \text{Número de frames} \end{aligned}$$

Figura I.6: Construcción de la matriz  $A$ .



## Capítulo 1

---

# FUNCIONES GLOBALES DEL TOOLBOX

---

En este capítulo se describen las funciones generales que se utilizan, de manera común, en las diferentes aplicaciones mostradas en este manual. Por su carácter global, estas funciones pueden ser de gran utilidad para cualquier usuario que esté realizando implementaciones en Matlab relacionadas con el procesamiento de imágenes. Procederemos con la descripción de las funciones y su respectiva sintaxis.

### 1.1 ReadImage DataBase

Esta función recibe la dirección de una carpeta con imágenes (la ruta se especifica en el parámetro `Textpath`), todas del mismo tamaño, en el formato especificado y en escala de grises (en caso que no lo esté, la función las convierte a escala de grises). Los formatos admitidos son `.jpg`, `.pgm`, `.png`, `.tif`, `.bpm`. La función asume como formato por defecto `.jpg`. Retorna los siguientes argumentos:

- $m, n$ : Son valores enteros tales que  $m \times n$  representa la dimensión de cada una de las imágenes en la carpeta.
- $A$ : Es una matriz que contiene en sus columnas la información numérica de las imágenes en la carpeta. La columna  $i$  de dicha matriz, corresponde a la  $i$ -ésima imagen.  
La matriz  $A$  tiene dimensión  $mn \times \text{numImg}$ .

**Nota:** No hay restricción en que en la carpeta existan archivos de otra naturaleza u otras carpetas. La función valida y lee únicamente los archivos en el formato especificado.

### 1.1.1 Sintaxis

```
1 [A,m,n] = ReadImage DataBase('Textpath','Extension','ext')
```

Donde el parámetro Extension es opcional.

## 1.2 Matrix2Image

Esta función toma una matriz  $X$  cuyas columnas corresponden a la vectorización de un grupo de imágenes, una columna por cada imagen. La función reconstruye dichas imágenes y las guarda en un directorio. La función recibe:

- m y n: Valores enteros tales que  $m \times n$  corresponde a la dimensión de cada una de las imágenes.
- X: Es una matriz de tamaño  $mn \times \text{numImage}$ , tal que cada columna representa una imagen de tamaño  $m \times n$  y numImage es el número de imágenes.
- Textpath: Ruta de un directorio donde se guardarán las imágenes.
- Name: Raíz común de los nombres de las imágenes que se guardarán en Textpath. La función guarda en el directorio Textpath cada imagen con el nombre NameYYY.jpg, donde YYY representa una secuencia numérica generada automáticamente por la función.

### 1.2.1 Sintaxis

```
1 Matrix2Image(X,m,n,'Textpath','Name');
```

## 1.3 double2uint8

Esta función convierte los valores de una matriz  $X$  en uint8. Para lo cual cada valor es rescalado en el rango  $[0,255]$  de forma lineal. Es decir, el min( $X$ ) se asigna a cero (0), mientras que el max( $X$ ) se asigna a 255. Los valores intermedios son asignados de forma proporcional.

### 1.3.1 Sintaxis

```
1 X = double2uint8(X)
```

## 1.4 Video2Matrix

Esta función recibe un video y lo codifica como una matriz, lo cual será de gran utilidad en el Capítulo 5. La función recibe los siguientes parámetros:

- PathVideo: Es la ruta de un archivo de video con extensión .mp4.

- **Scale:** Este valor se utiliza para escalar cada uno de los frames en una fracción del tamaño original, debe cumplir que:  $0 < \text{Scale} \leq 1$ . Su valor por defecto es 1, en cuyo caso se utilizará el tamaño de los frames del video original.
- **numFrames:** Número de frames a utilizar en el video. Este valor es un entero positivo que debe ser menor o igual que el número total de frames en el video original. De omitirse, se utilizarán todos los frames del video original.

Esta función retorna:

- **X0:** Una matriz de tamaño  $mn \times \text{numFrames}$ , donde cada columna de A corresponde a un frame del video, redimensionado como vector columna y en escala de grises.
- **m1 y n1:** Tamaño de cada uno de los frames del video, luego del proceso de escalado.

#### 1.4.1 Sintaxis

```
1 [X0,m1,n1]=Video2Matrix('PathTextVideo','Scale',Scale,'numFrames',numFrames);
```

Los parámetros **Scale** y **numFrames** son opcionales.

### 1.5 Matrix2Video

Esta función se encarga de construir un video con extensión **.mp4** a partir de una matriz **X** de tamaño  $mn \times \text{numFrames}$ , donde cada una de las columnas de **X** representa un frame de tamaño  $m \times n$ . La función recibe los siguientes argumentos:

- **X:** Matriz de tamaño  $mn \times \text{numFrames}$ .
- **m y n:** Valores enteros tal que el número de filas de X corresponde a  $mn$ . Estos valores se utilizan para redimensionar las columnas de X a una matriz de tamaño  $m \times n$ . De manera que el  $i$ -ésimo frame del video corresponde a la  $i$ -ésima columna de X, redimensionada a tamaño  $m \times n$ .
- **Path\Name:** Representa la dirección del directorio donde será guardado el archivo de video con el nombre **Name.mp4**. En caso que **Path** se omita, entonces la función guardará el archivo de video en el directorio de trabajo actual de Matlab.

#### 1.5.1 Sintaxis

```
1 Matrix2Video(X,m,n,'Path\Name')
```

### 1.6 img\_miss\_pix

Esta función recibe la dirección de una imagen y un escalar **porc**, tal que  $0 < \text{porc} < 1$ , y genera una imagen, en el mismo directorio, con la cantidad de píxeles perdidos indicados por **porc**. Por ejemplo, si **porc** = 0.25 significa que se va a generar una imagen con 25% de píxeles perdidos.

- **Textpath:** Es la ruta de la imagen.
- **porc:** Escalar que denota el porcentaje de pixeles perdidos que tendrá la nueva imagen.

### 1.6.1 Sintaxis

```
| img_miss_pix('Textpath',porc);
```

## Capítulo 2

---

### FILTRO PARA REMOVER RUIDO

---

Esta primera aplicación supone que se cuenta con una base de datos de imágenes, la cual se codifica matricialmente en una matriz  $A$ , mediante la aplicación de la función `ReadImage DataBase` (ver la función 1.1). A partir de  $A$  se genera otra matriz  $C$  con la aplicación de algún tipo de ruido (sobre esto se detallará en la sección 2.2). Así que  $C$  se puede conceptualizar como la matriz de imágenes asociadas a la base de datos, pero con ruido. El objetivo de la aplicación es determinar una matriz  $X$  de rango reducido, que se denomina un *filtro*, que permitirá limpiar el ruido a una nueva imagen que no es parte de la base de datos, pero que tiene cierta similitud con las imágenes en ella. La calidad del resultado dependerá de factores como qué tan similar o no es la imagen que se está tratando de limpiar, en comparación con las imágenes en la base, y la similitud del tipo de ruido que tiene la imagen, en comparación con el tipo de ruido que se utilizó para construir a la matriz  $C$ . Si dicha imagen con ruido se vectoriza y se representa con  $\tilde{w}$ , entonces la imagen filtrada quedará determinada por la vectorización:

$$w = X \cdot \tilde{w}$$

Como problema de optimización, la aplicación se formula de la siguiente manera. Dadas  $A \in \mathbb{C}^{m \times n}$ ,  $B \in \mathbb{C}^{m \times p}$  y  $C \in \mathbb{C}^{q \times n}$ , se busca encontrar la matriz  $X \in \mathbb{C}^{p \times q}$  que de solución al problema de minimización:

$$\min_{X \in \mathbb{C}_k^{m \times n}} \|A - BXC\|_{fr},$$

## 6 FILTRO PARA REMOVER RUIDO

tal que  $\|X\|_{fr}$  sea mínima también. Si se toma  $m = p$  y  $B = I_m$ , el problema se convierte en:

$$\min_{X \in \mathbb{C}_k^{m \times n}} \|A - XC\|_{fr} \quad (2.1)$$

La solución del problema (ver [2]) 2.1 está dada por:

$$X = (A \cdot C^\dagger \cdot C)_k \cdot C^\dagger,$$

y

$$(A \cdot C^\dagger \cdot C)_k = \sum_{i=1}^k (\sigma_i u_i v_i^*) \quad \text{si } 1 \leq k \leq \text{rank}(A \cdot C^\dagger \cdot C)$$

En este último caso,  $\sigma_i$  es el i-ésimo valor singular de la matriz  $A \cdot C^\dagger \cdot C$ . Además,  $u_i$  y  $v_i$  son los vectores singulares por izquierda y por derecha, respectivamente, asociados a la SVD de la matriz  $A \cdot C^\dagger \cdot C$ .

A continuación se procede con la descripción de las funciones asociadas al problema, junto con su respectiva sintaxis.

### 2.1 RankConstrainedFilterX

Esta función calcula la matriz  $X$  de rango reducido que es la solución del problema 2.1.

#### 2.1.1 Sintaxis

```
1 X=RankConstrainedFilterX(A,C,k)
```

Donde  $A \in \mathbb{C}^{m \times n}$ ,  $C \in \mathbb{C}^{q \times n}$  y  $X \in \mathbb{C}^{m \times q}$ . Además, el parámetro  $k$  es la restricción para el rango.

#### Ejemplo

```
1 clc; clear; close all
2 A = [-1 1 2; 3 0 -1; 2 -2 3; 4 0 3];
3 C=[-2 1 6; -2 4 5];
4 k=2;
5 X=RankConstrainedFilterX(A,C,k)
```

Lo anterior genera:

```
X =
0.1671    0.2145
-0.3042    0.0125
1.0623   -0.8304
0.4589   -0.2319
```

### 2.2 NoiseFunction

Esta función recibe la dirección de una carpeta con imágenes (la ruta se especifica en el parámetro `Textpath`), todas del mismo tamaño y en escala de grises (en caso que no lo

esté, la función las convierte a escala de grises). Utiliza la función ReadImageDataBase (ver sección 1.1) y en ella puede consultar los formatos admitidos para las imágenes. Retorna los siguientes argumentos:

- A: Es una matriz que contiene en sus columnas la información numérica de las imágenes en la carpeta. La columna  $i$  de dicha matriz, corresponde a la  $i$ -ésima imagen. La matriz A tiene dimensión  $mn \times \text{numImg}$ . La descripción de los parámetros  $m$ ,  $n$  y  $\text{numImg}$  debe ser consultada en la sección 1.1.
- C: Es una matriz que se genera a partir de A, mediante la aplicación de un ruido. Las opciones y parámetros asociados al ruido, que se utilizan para construir la matriz C, son:
  - gaussian, meangauss y sigmagauss que representan, respectivamente, que se utilizará ruido blanco gaussiano con media meangauss y varianza sigmagauss. Los valores por defecto para la media y la varianza son meangauss = 0 y sigmagauss = 0.01, respectivamente.
  - s&p y d, que representan, respectivamente, que se utilizará el ruido denominado *Salt and pepper*, y d es la densidad de dicho ruido. El valor por defecto para d es d = 0.05.
  - speckle y sigmaspeckle que representan, respectivamente, que se utilizará el ruido denominado *speckle*, con una varianza sigmaspeckle. El valor por defecto para sigmaspeckle es sigmaspeckle = 0.05.
- m, n: Son valores enteros tales que  $m \times n$  representa la dimensión de cada una de las imágenes en la carpeta.

### 2.2.1 Sintaxis

```
1 [C,A,m,n]=NoiseFunction('Textpath')
```

Al utilizar la sintaxis anterior se creará la matriz C, a partir de A, utilizando por defecto un ruido gaussiano, con meangauss = 0 y sigmagauss = 0.01. La función NoiseFunction tiene argumentos opcionales que pueden ser personalizados por el usuario, en cuanto a los tres tipos de ruido disponibles (gaussian, s&p y speckle), así como los parámetros inherentes a cada uno de ellos. Esto se detallará a continuación.

### 2.2.2 Sintaxis alternativa

A continuación se le muestran los tres tipos adicionales de sintaxis que puede utilizar para esta función. En todos los casos, si solo se especifica el tipo de ruido y no se incluyen los parámetros adicionales, entonces la función utiliza los valores definidos por defecto.

```
1 [C,A,m,n]=NoiseFunction('Textpath','NoiseOption','gaussian','sigmagauss',n_0,'meangauss',n_1);
```

En este caso se aplica, para la construcción de la matriz C, ruido blanco gaussiano, con la desviación y media que especifique en los valores n\_0 y n\_1, respectivamente.

```
1 [C,A,m,n]=NoiseFunction('Textpath','NoiseOption','s&p','d',n_0);
```

Por su parte, con la sintaxis anterior, para construir a C, se aplica el tipo de ruido *Salt and pepper*, con la densidad que se especifique en n\_0.

## 8 FILTRO PARA REMOVER RUIDO

```
1 [C,A,m,n]=NoiseFunction('Textpath','NoiseOption','speckle','sigmaspeckle',n_0);
```

Finalmente, con la sintaxis anterior, para construir a C, se aplica el tipo de ruido *speckle*, con la desviación que se especifique en n\_0.

### 2.3 FullRankConstrainedFilterX

Esta función es una versión más completa de RankConstrainedFilterX (ver sección 2.1), aplicada específicamente para procesamiento de imágenes. Esta función recibe una dirección de una carpeta con imágenes (la ruta se especifica en el parámetro Textpath), todas del mismo tamaño y en escala de grises (en caso que no lo esté, la función las convierte a escala de grises). Esta función utiliza las funciones ReadImageDataBase (ver sección 1.1) y NoiseFunction (ver sección 2.2). En la función ReadImageDataBase puede consultar los formatos admitidos para las imágenes. Retorna los siguientes argumentos:

- m, n: Son valores enteros tales que  $m \times n$  representa la dimensión de cada una de las imágenes en la carpeta.
- A: Es una matriz que contiene en sus columnas la información numérica de las imágenes en la carpeta. La columna  $i$  de dicha matriz, corresponde a la  $i$ -ésima imagen. La matriz A tiene dimensión  $(mn) \times \text{numImg}$ , donde numImg es la cantidad de imágenes en formato .jpg que hay en la carpeta.
- X: Es la matriz de rango reducido que es la solución del problema 2.1. Corresponde al filtro construido a partir de la función NoiseFunction (ver sección 2.2).

#### 2.3.1 Sintaxis

```
1 [X,m,n,A]=FullRankConstrainedFilterX('Textpath')
```

#### 2.3.2 Sintaxis alternativa

La función FullRankConstrainedFilterX cuenta con parámetros adicionales que le permiten al usuario personalizar los siguientes valores:

- k: Es el valor que restringe el rango de la matriz X. Se debe cumplir que

$$0 < k \leq \min\{m \cdot n, \text{numImg}\}$$

Si este valor no se especifica, se toma el valor por defecto  $k = \min\{m \cdot n, \text{numImg}\}$ .

- NoiseOption: Permite personalizar parámetros asociados al ruido. Dado que FullRankConstrainedFilterX utiliza a la función NoiseFunction, entonces se pueden personalizar todos los parámetros incluidos en esa función. Para más detalles, ver la sección 2.2, específicamente en la descripción de la matriz C.

A continuación se le muestran algunas alternativas de sintaxis para esta función:

```
1 [X,m,n,A]=FullRankConstrainedFilterX('Textpath','k',k_0,'NoiseOption','gaussian','sigmgauss',n_0,'meangauss',n_1);
```

En este caso se aplica, para la construcción de la matriz C, ruido blanco gaussiano, con la desviación y media que especifique en los valores n\_0 y n\_1, respectivamente.

```
1 [X,m,n,A]=FullRankConstrainedFilterX('Textpath','k',k_0,'NoiseOption','s&p','d',n_0);
```

Por su parte, con la sintaxis anterior, para construir a C, se aplica el tipo de ruido *Salt and pepper*, con la densidad que se especifique en n\_0.

```
1 [X,m,n,A]=FullRankConstrainedFilterX('Textpath','k',k_0,'NoiseOption','speckle','sigmaspeckle',n_0);
```

Finalmente, con la sintaxis anterior se aplica el tipo de ruido *speckle*, con la desviación que se especifique en n\_0.

## 2.4 SaveBlurredImages

Esta función toma una base de datos de imágenes en escala de grises (en caso que no lo esté, la función las convierte a escala de grises), todas del mismo tamaño y permite crear otra base de imágenes, a partir de la anteriores, pero con ruido. La función recibe la dirección de una carpeta con imágenes (parámetro pathOriginal) y otra dirección de la carpeta en la que escribirá las imágenes con ruido (parámetro pathWriteNoise). Esta función utiliza la función NoiseFunction, entonces se pueden personalizar todos los parámetros adicionales incluidos en esa función. Para más detalles, ver la sección 2.2, específicamente en la descripción de la matriz C y los formatos admitidos para las imágenes.

### 2.4.1 Sintaxis

```
1 SaveBlurredImages('pathOriginal','pathWriteNoise');
```

Al utilizar la sintaxis anterior se crearán las imágenes con ruido, utilizando los parámetros por defecto. En este caso, utilizando por defecto un ruido gaussiano, con meangauss = 0 y sigmagauss = 0.01. En el caso que se quiera personalizar el tipo de ruido y sus parámetros, se debe utilizar alguna de las sintaxis que se muestran a continuación.

### 2.4.2 Sintaxis alternativa

```
1 SaveBlurredImages('pathOriginal','pathWriteNoise','NoiseOption','gaussian','sigmagauss',n_0,'meangauss',n_1);
```

En este caso, para la construcción de las nuevas imágenes, se aplica ruido blanco gaussiano, con la desviación y media que especifique en los valores n\_0 y n\_1, respectivamente.

```
1 SaveBlurredImages('pathOriginal','pathWriteNoise','NoiseOption','s&p','d',n_0);
```

Por su parte, en este caso se aplica el tipo de ruido *Salt and pepper*, con la densidad que se especifique en n\_0.

```
1 SaveBlurredImages('pathOriginal','pathWriteNoise','NoiseOption','speckle','sigmaspeckle',n_0);
```

Finalmente, con la sintaxis anterior se aplica el tipo de ruido *speckle*, con la desviación que se especifique en n\_0.

## 2.5 SaveFilteredImages

Esta función recibe una base de datos de imágenes en escala de grises (en caso que no lo esté, la función las convierte a escala de grises), todas del mismo tamaño y con ruido. Luego, la función permite crear otra base de imágenes, con las imágenes filtradas. La función recibe la dirección de una carpeta con imágenes con ruido (pathNoise) y otra dirección de la carpeta en la que escribirá las imágenes filtradas (pathWriteFiltered). Utiliza a ReadImageDataBase (ver sección 1.1), y en ella puede consultar los formatos admitidos para las imágenes.

Es recomendable utilizarla posterior a la función FullRankConstrainedFilterX, dado que utiliza la matriz X de rango reducido que es la solución del problema 2.1, y que corresponde al filtro construido para limpiar a las imágenes con ruido.

### 2.5.1 Sintaxis

```
1 SaveFilteredImages('pathNoise','pathWriteFiltered',X);
```

## Capítulo 3

---

# FACTORIZACIÓN NO NEGATIVA APLICADA A LA RECONSTRUCCIÓN DE IMÁGENES

---

Esta segunda aplicación también supone que se cuenta con una base de datos de imágenes, la cual se codifica matricialmente en una matriz  $A$ , mediante la aplicación de la función `ReadImage DataBase` (ver la función 1.1). La aplicación consiste en determinar matrices  $W$  y  $H$ , de entradas no negativas, tales que  $A \approx WH$ . En el contexto del procesamiento de imágenes, las columnas de la matriz  $W$  pueden verse como una base para el espacio de las caras. Y la matriz  $H$  son los coeficientes de las combinaciones lineales que permiten reconstruir cualquier cara del conjunto original de imágenes, utilizando a la matriz  $W$ . Esto es, al multiplicar  $W$  por la columna  $i$  de  $H$  se obtiene la columna  $i$  de  $A$ , que es la vectorización de la  $i$ -ésima imagen. Adicionalmente, si se tiene una imagen que no pertenece al conjunto de imágenes almacenadas en  $A$ , pero que guardan cierta similitud con ellas, entonces es posible utilizar la matriz  $W$  para realizar una reconstrucción de la imagen. En efecto, si  $\omega$  es la vectorización de la imagen que se va a reconstruir, entonces el producto:

$$W^\dagger \cdot \omega \tag{3.1}$$

genera una vectorización de la imagen reconstruida.

Para contextualizar mejor esta aplicación y antes de proceder con detalles más técnicos, se mostrará un ejemplo del uso de las funciones utilizando para ello un grupo de imágenes extraídas de **the extended Yale face database B** ([3]), las cuales son de tamaño  $192 \times 168$ . Las 570 imágenes utilizadas para construir la matriz  $A$ , que es de tamaño  $32256 \times 570$ , pueden ser descargadas del siguiente [vínculo](#). Al ejecutar el siguiente código (luego de la introducción se muestra el detalle de las funciones utilizadas):

```
1 [W,H,m,n]=ConstructionNNFM('BaseParaTrabajar570Caras','IteraMax',1000);
```

se obtiene la matriz  $W$  de tamaño  $32256 \times 285$  que corresponde a la base de caras (285 caras en total, que corresponde al valor de  $r$  por defecto calculado para esta matriz). Una parte de dicha base de caras se muestra en la Figura 3.1.

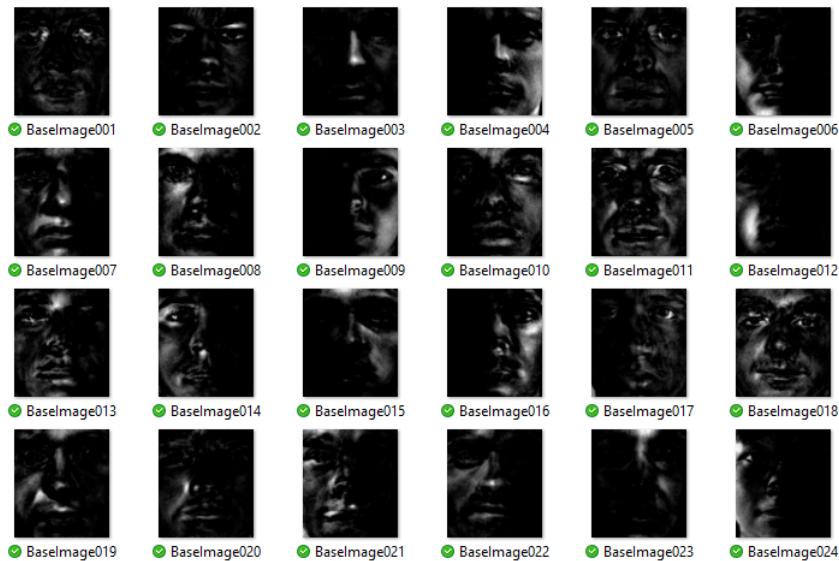


Figura 3.1: Parte de la base de caras.

Por otra parte, en la carpeta disponible [aquí](#) se incluyeron 39 archivos que no forman parte de las 570 imágenes usadas para construir la matriz  $A$ , pero tienen cierta similitud con dichas imágenes, y por ello pueden ser utilizadas para probar el proceso de reconstrucción. En nuestro caso usaremos la imagen *yaleB39(1).jpg*, que se muestra en la Figura 3.2.



Figura 3.2: Imagen externa *yaleB39(1).jpg* que será reconstruida.

Al ejecutar el código siguiente:

```
1 ReconstructionExternalFace('BaseCaras_Yale_JJ\1caraDeCadaPersona\yaleB39(1).jpg',W,'Imagen');
```

se utiliza la fórmula mostrada en 3.1 y se obtiene como resultado la reconstrucción que se muestra en la Figura 3.3:



Figura 3.3: Reconstrucción de la imagen *yaleB39(1).jpg* utilizando la matriz  $W$ .

Como problema de optimización, la factorización no negativa parte de una matriz  $A_{m \times n}$  y busca determinar matrices  $W_{m \times r}$  y  $H_{r \times n}$  ( $W, H \geq 0$ ) que sean solución de:

$$\min_{W, H \geq 0} \|A - WH\|_{fr}^2, \quad (3.2)$$

donde  $0 < r \leq \min\{m, n\}$ . La solución del problema anterior se puede realizar mediante un método iterativo aplicando las *multiplicative update rules* (ver [4]), las cuales aplican las siguientes reglas de actualización para las matrices  $H$  y  $W$ :

$$H_{au} = H_{au} \cdot \frac{(W^T A)_{au}}{(W^T W H)_{au}}, \quad W_{ia} = W_{ia} \cdot \frac{(A H^T)_{ia}}{(W H H^T)_{ia}}$$

En [4] se demuestra que el método iterativo propuesto converge a un punto estacionario de la distancia:  $\|A - WH\|_{fr}$ . A continuación se muestra el pseudocódigo.

---

**Algoritmo 1:** Pseudocódigo general.

---

**Result:** matrices  $W$  y  $H$

- 1 Inicialice  $W$  y  $H$ ;
- 2 **for**  $z=1$ : *iteraciones* **do**
- 3     **for**  $a=1:r$  **do**
- 4         **for**  $u=1:n$  **do**
- 5              $H_{au} \leftarrow H_{au} \cdot \frac{(W^T A)_{au}}{(W^T W H)_{au}}$
- 6         **end**
- 7         **for**  $i=1:m$  **do**
- 8              $W_{ia} \leftarrow W_{ia} \cdot \frac{(A H^T)_{ia}}{(W H H^T)_{ia}}$
- 9         **end**
- 10     **end**
- 11 **end**

---

Ahora procederemos con la descripción de las funciones que son parte de esta aplicación.

### 3.1 nnfLS

Esta función usa las *multiplicative update rules* para realizar una factorización no negativa de la matriz  $A$ , de tal manera que  $A \approx WH$ .

Esta función recibe:

- A: Es la matriz que se desea factorizar, de tamaño  $mA \times nA$ .
- r: Es la restricción para el rango. El valor por defecto es  $\lfloor \frac{\min\{mA, nA\}}{2} \rfloor$ , donde  $\lfloor \cdot \rfloor$  denota la función piso.
- IteraMax: Número máximo de iteraciones. Su valor por defecto es  $2 \cdot nA$ .
- Tol: Es la tolerancia para el error. Su valor por defecto es  $10^{-6}$ .

### 3.1.1 Sintaxis

```
1 [W,H,VectorError] = nnfLS(A,'r',r,'IteraMax',IteraMax,'Tol',Tol)
```

Donde los parámetros r, IteraMax y Tol son opcionales.

#### Ejemplo

Considere la matriz:

```
A=[1 2 5; 2 3 5; 6 7 1; 6 1 0];
```

Al ejecutar la función con

```
1 rng(1); %Se establece la semilla en 1 para poder replicar el experimento.
2 [W,H,VectorError]=nnfLS(A,'r',3,'IteraMax',1000)
```

Se obtiene:

$W =$

1.0041	0.0007	0.3556
0.9744	0.0690	0.5137
0.0000	0.3967	1.1398
0.0003	0.4070	0.0000

$H =$

0.9378	0.1176	4.6689
14.7401	2.4567	0.0000
0.1341	5.2866	0.8774

Y luego, al realizar el producto  $WH$  se obtiene:

$ans =$

1.0000	2.0000	5.0000
2.0000	3.0000	5.0000
6.0000	7.0000	1.0000
6.0000	1.0000	0.0012

### 3.2 ConstructionNNFM

Utiliza la función `ReadImageDataBase` (ver la sección 1.1) para leer una carpeta de imágenes y construir la matriz  $A$ , que contiene la vectorización de las imágenes en sus columnas. Posteriormente, usa la función `nnfLS` (ver la sección 3.1) para construir las matrices  $W$  y  $H$ , las cuales se almacenan en la carpeta “Results” (si la carpeta no existe, la función la creará automáticamente) como archivos independientes en formato `.mat`. Finalmente, la función genera una carpeta dentro de “Results” de nombre “ImagesBase”, en la cual se guardarán las imágenes generadas a partir de la matriz  $W$  y que corresponden a las imágenes de la base del espacio de caras.

Esta función recibe:

- `Extension`: Es la extensión del conjunto de imágenes que se van leer. Los formatos admitidos son `.jpg`, `.pgm`, `.png`, `.tif`, `.bpmp`. La función asume como formato por defecto `.jpg`.
- `r`: Es la restricción para el rango. El valor por defecto es  $\lfloor \frac{\min\{mA, nA\}}{2} \rfloor$ , donde  $\lfloor \cdot \rfloor$  denota la función piso.
- `IteraMax`: Número máximo de iteraciones. Su valor por defecto es  $2 \cdot nA$ .
- `Tol`: Es la tolerancia para el error. Su valor por defecto es  $10^{-6}$ .

Además de las matrices  $W$  y  $H$  la función devuelve los valores  $m$  y  $n$ , tales que  $m \times n$  representa la dimensión de cada una de las imágenes.

#### 3.2.1 Sintaxis

```
1 [W,H,m,n] = ConstructionNNFM('Textpath','Extension','ext',r,'IteraMax',itera,'Tol',tol)
```

Donde los parámetros `Extension`, `r`, `IteraMax` y `Tol` son opcionales.

### 3.3 ReconstructionExternalFace

Para utilizar esta función ya debe tenerse calculada la matriz  $W$ , ya sea directamente con la función `nnfLS` (ver la sección 3.1) o con la función `ConstructionNNFM` (ver la sección 3.2). La función `ReconstructionExternalFace` recibe un path de una imagen que no forma parte del grupo original de imágenes, pero guarda cierta similitud con ellas. Luego, usa la matriz  $W$  para reconstruir dicha imagen. Este proceso permite evaluar visualmente la calidad de la matriz  $W$ , que corresponde a la base del espacio de caras. La función `ReconstructionExternalFace` guardará la nueva imagen en formato `.jpg`, en el directorio “Results” dentro de la carpeta de trabajo. Si dicha carpeta no existe, la crea automáticamente. La función recibe:

- `Textpath`: Ruta de la imagen que se quiere reconstruir.
- `W`: Es la matriz que corresponde a la base del espacio de caras, obtenida a partir de la función `nnfLS` (ver la sección 3.1) o con la función `ConstructionNNFM` (ver la sección 3.2).
- `NameOutput`: Es el nombre que se utilizará para guardar la imagen reconstruida.

### 3.3.1 Sintaxis

```
1 ReconstructionExternalFace('Textpath',W,'NameOutput');
```

## 3.4 ReconstructionExternalFacesDirectory

Esta función extiende a `ReconstructionExternalFace` (ver la sección 3.3) a un directorio de imágenes por reconstruir, las cuales no forman parte del grupo inicial de imágenes, pero guardan cierta similitud con ellas. Los parámetros de esta función son:

- `Textpath`: Es la dirección de la carpeta de las imágenes que desea resconstruir.
- `W`: Es la matriz que corresponde a la base del espacio de caras, obtenida a partir de la función `nnfLS` (ver la sección 3.1) o con la función `ConstructionNNFM` (ver la sección 3.2).
- `NameOutput`: Es la raíz común que se utilizará para guardar las imágenes reconstruidas, al cual se le agrega un consecutivo numérico. El valor por defecto es `Reconstructed`.
- `Extension`: Es la extensión que se debe utilizar para guardar las imágenes reconstruidas. Los formatos admitidos son `.jpg`, `.pgm`, `.png`, `.tif`, `.bpm`. La función asume como formato por defecto `.jpg`.

### 3.4.1 Sintaxis

```
1 ReconstructionExternalFacesDirectory('Textpath',W,'Extension','ext','NameOutput','name')
```

Donde los parámetros `Extension` y `NameOutput` son opcionales.

## Capítulo 4

---

# APROXIMACIÓN DE LA MATRIZ DE RANGO BAJO MEDIANTE EL ALGORITMO BRP Y SU APLICACIÓN A LA COMPRESIÓN DE IMÁGENES

---

La aproximación de rango bajo de una matriz  $L \in \mathbb{R}^{m \times n}$  es un problema de optimización que busca aproximar a  $L$  con una matriz  $\hat{L}_r \in \mathbb{R}^{m \times n}$  tal que

$$\|L - \hat{L}_r\|_{fr}^2 = \min_{L_r \in \mathbb{R}_r^{m \times n}} \|L - L_r\|_{fr}^2$$

Si  $L = U_L \Sigma_L V_L^T$  es la descomposición en valores singulares (SVD) de  $L$ , entonces  $\hat{L}_r$  queda determinada por la SVD  $r$ -truncada (ver [5]), i.e.:

$$\hat{L}_r = \lfloor L \rfloor_r = U_{L,r} \Sigma_{L,r} V_{L,r}^T \quad (4.1)$$

La SVD es altamente precisa, pero tiene una alta complejidad computacional para matrices grandes [6]. Por ese motivo, se utilizará un método alternativo para aproximar la matriz  $\hat{L}_r$ , denominado *proyecciones aleatorias bilaterales* (bilateral random projection method, cuyas siglas en inglés son BRP), el cual fue desarrollado por [7] y mejorado mediante un esquema de potencias, bajo ciertas condiciones de invertibilidad, por Zhou and Tao (ver [8]). En [9] se extiende el esquema de potencias introducido por Zhou and Tao, relajando las condiciones de invertibilidad. Con respecto a lo anterior, si  $A_1 \in \mathbb{R}^{n \times r}$  es una matriz cualquiera de rango completo, entonces se define la matriz  $Y_2 = (L^T L)^c A_1$  de tamaño  $n \times r$ , donde  $c$  es un parámetro entero y positivo que se utiliza para mejorar la generación de las matrices de proyección bilateral, y puede ser ajustado por el usuario. En nuestro caso fijamos  $c = 3$ , pero este será un parámetro opcional de las funciones del Toolbox, de manera que el usuario puede explorar otras opciones. Luego, se calcula la descomposición

QR de  $Y_2$ , de tal manera que  $Y_2 = Q_{n \times r} R_{r \times r}$ . Finalmente, el algoritmo BRP aproxima a  $\hat{L}_r$  por:

$$\hat{L}_r \approx LQ_r Q_r^T,$$

donde  $Q_r = Q(:, 1 : r)$ , esto es, la matriz construida con las primeras  $r$  columnas de  $Q$ . Si se define  $A_{m \times r} = LQ_r$  y  $B_{r \times n} = Q_r^T$ , entonces podemos almacenar las matrices  $A$  y  $B$ , a partir de las cuales se puede reconstruir una aproximación para la matriz de rango bajo, quien a su vez corresponde a una aproximación de la matriz  $L$ .

### Ejemplo aplicado a la compresión de imágenes

Para contextualizar la presente aplicación de la matriz de rango bajo en la compresión de imágenes y su aproximación mediante el algoritmo BRP, desarrollaremos un ejemplo tomando como base la siguiente imagen de un **soldado**, que es de tamaño  $3376 \times 6000$ , y que fue tomada con fines ilustrativos del sitio **Wallpapers.com**. Si se representa dicha imagen mediante la matriz  $L$ , entonces  $L \in \mathbb{R}^{3376 \times 6000}$  y al ejecutar la instrucción:

```
1 ImageCompression('images/soldado.jpg', 0.01, 'BRP');
```

se obtienen las matrices  $L_{3376 \times 6000}$  (que representa la imagen original),  $A_{3376 \times 33}$  y  $B_{33 \times 6000}$ , las cuales son almacenadas en archivos de texto. El archivo asociado a  $L$  tiene un tamaño de  $149MB$ , mientras que los tamaños de los archivos de texto asociados a  $A$  y  $B$  son  $868kb$  y  $1.91MB$ , respectivamente. Con ello se observa que la suma de los tamaños de estos últimos archivos es significativamente menor que el tamaño del archivo asociado a  $L$ . En la Figura 4.1 se muestra la imagen original del soldado construida con la matriz  $L$ , mientras que en la Figura 4.2 se muestra la imagen reconstruida utilizando las matrices  $A$  y  $B$ , usando un radio de compresión del 1%, esto es, apenas se está utilizando un 1% del número de valores singulares no nulos, que para este caso son 33. Obviamente, si el radio de compresión se aumenta, el tamaño de los archivos aumentará, pero la calidad de la reconstrucción mejorará. Por ejemplo, en la Figura 4.3 se muestra la reconstrucción de la imagen usando un radio de compresión del 5% (en este caso el archivo de texto de  $A$  tiene un tamaño de  $4.60MB$ , mientras que el de  $B$  es  $9.66MB$ ).



Figura 4.1: Imagen original del soldado.

Con respecto a los tiempos de ejecución, la construcción de las matrices  $A$  y  $B$  para la imagen del **soldado** tomó un tiempo promedio de 1.22 segundos. Para observar la ganancia en términos de tiempos de ejecución del algoritmo BRP, con respecto a la fórmula 4.1, se repitió el experimento utilizando dicha fórmula y tomando  $A = U_{L,r}\Sigma_{L,r}$  y  $B = V_{L,r}^T$ . En la Figura 4.4 se muestra el resultado de la compresión usando la SVD truncada y un radio de compresión del 5%. Como se puede ver, las diferencias entre las Figuras 4.3 y



Figura 4.2: Imagen del soldado luego de la compresión, usando un radio de compresión del 1%.



Figura 4.3: Imagen del soldado luego de la compresión, usando un radio de compresión del 5%.

4.4 son casi imperceptibles. Sin embargo, la diferencia fundamental está en el tiempo de ejecución. Para este caso, la construcción de las matrices  $A$  y  $B$  tomó un tiempo promedio de 35.93 segundos. Con ello se refuerza el hecho que para la aproximación de la matriz de rango bajo es mucho más rápido si se realiza con el algoritmo BRP, en lugar de la SVD truncada.



Figura 4.4: Imagen del soldado luego de la compresión con la SVD truncada, usando un radio de compresión del 5%.

Ahora se procederá con la descripción de las funciones del toolbox.

## 4.1 LowRankMatrixBRP

Esta función recibe una matriz  $L$  y calcula una aproximación de rango bajo usando el método BRP. Los parámetros de esta función son:

- $L$ : Es la matriz que se quiere aproximar con la matriz de rango bajo usando el método BRP. Su dimensión es  $m \times n$ .
- $r$ : Condición para el rango tal que  $0 < r \leq \text{rank}(L)$ . El valor por defecto se toma como  $\text{floor}(\min(m, n) / 2)$ .
- $c$ : Parámetro del esquema de potencias utilizado por el método, se utiliza para mejorar la generación de las matrices de proyección bilateral (ver [10]). Debe ser entero, positivo y podría ser ajustado por el usuario. Su valor por defecto es 3.

El algoritmo retorna las matrices  $A$  y  $B$ , de tal manera que luego el usuario puede reconstruir la aproximación de rango bajo de  $L$  mediante el producto  $A \cdot B$ .

### 4.1.1 Sintaxis

```
1 [A,B]=LowRankMatrixBRP(L,'r',r,'c',c);
```

Donde los parámetros  $r$  y  $c$  son opcionales. En caso de ser omitidos, se tomarán sus valores por defecto.

## 4.2 ImageCompression

Esta función recibe una imagen y la comprime utilizando un radio de compresión indicado por el usuario, el cual corresponde al porcentaje de valores singulares no nulos que se deben utilizar. El radio de compresión es un decimal  $d$  tal que  $\frac{1}{\min\{m,n\}} \leq d \leq 1$ . Los parámetros de esta función son:

- **Textpath**: Es la dirección de la imagen que desea comprimir.
- **radio**: Corresponde al porcentaje de valores singulares no nulos que se deben utilizar. Es un decimal  $d$  tal que  $\frac{1}{\min\{m,n\}} \leq d \leq 1$ .
- **Opcion**: Etiqueta con la que se indica si se desea aplicar el método BRP o la SVD truncada. Las etiquetas válidas con 'BRP' y 'SVD'.

Al utilizar esta función en la carpeta "Results", dentro del directorio de trabajo, se guardará la imagen original, la imagen reconstruida y la información de las matrices  $A$  y  $B$  en dos archivos de texto, que son las que representan la compresión. Si la carpeta "Results" no existe, la crea automáticamente.

### 4.2.1 Sintaxis

```
1 ImageCompression('Textpath',radio,'Opcion');
```

## Capítulo 5

---

# ALGORITMO GODEC APLICADO A LA DETECCIÓN DE VARIACIONES EN UN CONJUNTO DE IMÁGENES

---

En este capítulo se mostrarán dos aplicaciones del algoritmo GoDec relacionadas con la detección de variantes en un conjunto de imágenes. En la primera de ellas (que denominaremos GoDec–Video) se aplica GoDec para la detección de movimientos en un video grabado sobre un fondo estático, para obtener dos videos nuevos, uno con el fondo libre de los objetos en movimiento y otro en donde únicamente se muestren los objetos en movimiento, sin el fondo. En la segunda aplicación que se mostrará de GoDec (que denominaremos GoDec–Imágenes) se cuenta con una lista de imágenes de un rostro, de tal manera que todas ellas tienen algún tipo de sombra o reflejo que impide ver el rostro completo. La aplicación consiste en la reconstrucción del rostro pero libre de sombras y reflejos.

### GoDec–Video

Luego de codificar matricialmente un video  $V$ , siguiendo los pasos expuestos en la introducción de este manual, se obtiene una matriz  $A$  que contiene la información de los frames del video. La aplicación del algoritmo GoDec a la matriz  $A$ , bajo ciertas condiciones, permite determinar una matriz  $L$  de rango reducido y una matriz  $S$  de baja cardinalidad o esparcida, de tal manera que  $A \approx L + S$ . Si se invierte el proceso a partir del cual se construyó la matriz  $A$ , pero aplicándolo a las matrices  $L$  y  $S$ , se pueden generar, respectivamente, los videos que muestran de manera separada el fondo fijo y los objetos en movimiento en  $V$ . A continuación se le muestra un ejemplo concreto generado con la instrucción:

```
1 GoDecVideoFull('GoDec\VideoWalking.mp4', 2, 2211840)
```

Donde los valores 2 y 2211840 utilizados en este ejemplo corresponden, respectivamente, a la restricción para el rango máximo de la matriz  $L$  y a la restricción máxima para la cardinalidad de  $S$ . La Figura 5.1 muestra el mismo frame para el video original y para los videos generados con las matrices  $L$  y  $S$ . La imagen generada con  $L$  corresponde a los objetos que no presentan movimiento en el frame correspondiente en el video original, mientras que la imagen generada con  $S$  presenta los objetos que sí estaban en movimiento. Es natural pensar que la visualización de los tres videos permitiría una comprensión más clara de la separación que se logra con el algoritmo GoDec. Por ese motivo en el siguiente **vínculo** se le incluye una animación que le permitirá ver los resultados del experimento. El video original que se utilizó para este ensayo fue tomado de [aquí](#).



Figura 5.1: Muestra del mismo frame, tanto del video original, como de los videos generados con las matrices  $S$  y  $L$ .

### GoDec-Imágenes

En este caso se cuenta con un conjunto de `numImage` imágenes del mismo objeto (que puede ser un rostro) y todas de tamaño  $m \times n$ . Las imágenes poseen sombras o reflejos que dificultan la visualización del objeto completo. El objetivo de la aplicación es utilizar el algoritmo GoDec para eliminar las sombras y los reflejos y así poder reconstruir una imagen limpia del objeto.

La codificación de las imágenes se realiza siguiendo el proceso descrito en la introducción de este manual y para así obtener la matriz  $A$ . Al aplicar el algoritmo GoDec nuevamente se obtienen matrices  $L$  y  $S$ , tales que  $A \approx L + S$ . Si la diferencia entre las imágenes originales corresponde a sombras o reflejos, entonces dichas distorsiones se pueden interpretar como pixeles de objetos en movimiento. De esta manera, de la matriz  $L$  se puede extraer la imagen reconstruida (cualquiera de las columnas de  $L$  representa una posible reconstrucción de la imagen). Por su parte, en la matriz  $S$  se almacenan las sombras y los reflejos. Procederemos con un ejemplo que permite mostrarle al lector el experimento realizado con la instrucción:

```
1 GoDecImageFull('sixFaces', 1, 50);
```

Donde los valores 1 y 50 utilizados en este ejemplo corresponden, respectivamente, a la restricción para el rango máximo de la matriz  $L$  y a la restricción máxima para la cardinalidad de  $S$ . Para este ejemplo, se supone que `sixFaces` es un directorio con 6 rostros con sombras. Los rostros utilizados se muestran en la Figura 5.2 y fueron tomados de **the extended Yale face database B** ([3]). Al ejecutar la instrucción anterior se crean dos nuevos directorios en la carpeta actual de trabajo, con los nombres `ImageForL` y `ImageForS`, donde se guardarán las imágenes filtradas. En la Figura 5.2 se puede observar el resultado obtenido con la aplicación de GoDec para el filtrado de sombras.

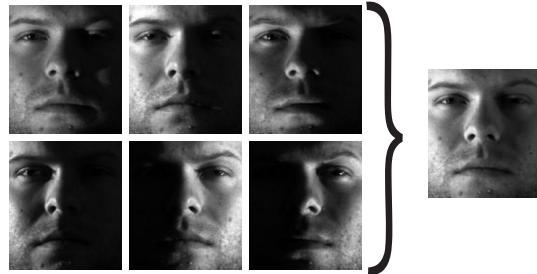


Figura 5.2: Filtrado de sombras mediante GoDec.

Teniendo claro las aplicaciones anteriores, se procederá con la descripción del problema de optimización intrínseco al algoritmo GoDec. Para ello se parte de una matriz densa  $A \in \mathbb{R}^{m \times n}$  y se busca determinar matrices  $L \in \mathbb{R}^{m \times n}$  y  $S \in \mathbb{R}^{m \times n}$ , con  $\text{rank}(L) \leq r$  y  $\text{card}(S) \leq k$ , tales que:

$$\min_{\text{rank}(L) \leq r; \text{card}(S) \leq k} \|A - L - S\|_{fr}^2 \quad (5.1)$$

La solución del problema 5.1 se hace de forma iterativa utilizando el algoritmo propuesto en [10] y denominado *Go Decomposition* (GoDec), el cual consiste en la actualización sucesiva de las matrices  $L_t$  y  $S_t$ , mediante las siguientes reglas:

$$L_t = \sum_{i=1}^r \lambda_i U_i V_i^T, \quad \text{SVD}(A - S_{t-1}) = U \Lambda V^T$$

$$S_t = \mathcal{P}_\Omega(A - L_t), \quad \Omega : |(A - L_t)_{i,j} \in \Omega| \neq 0, \quad |\Omega| \leq k$$

La función  $\mathcal{P}_\Omega$  toma una matriz densa  $Y$  y retorna una matriz esparcida de cardinalidad  $k$ , la cual contiene las  $k$  entradas de mayor valor absoluto en  $Y$ , las cuales se almacenan en su misma posición y las demás entradas se toman como cero. El algoritmo clásico de GoDec se puede ver en el Algoritmo 2.

---

**Algoritmo 2:** GoDec clásico.

---

**Input:**  $k, r, q, A_{m \times n}$   
**Result:**  $L, S$

1  $L_0 := A, S_0 := \mathbf{0}_{m \times n}, t := 0;$   
2 **while** Verdadero **do**  
3      $t := t + 1;$   
4      $\text{SVD} = \text{SVD}(A - S_t);$   
5      $L_t := S_r V_r D_r;$   
6      $S_t := \mathcal{P}_\Omega(A - L_t);$   
7      $E_t := \frac{\|A - L_t - S_t\|_{fr}^2}{\|A\|_{fr}^2};$   
8     **if**  $|E_t - E_{t-1}| < tol$  **then**  
9         **break**  
10     **end**  
11 **end**  
12 Retornar  $L, S$

---

En [10] se presenta una mejora (en términos de cantidad de cálculos y tiempo de ejecución) del método GoDec al sustituir la descomposición en valores singulares (SVD) por el método BRP (*bilateral random projection*). De esta forma, la actualización de  $L$  se puede realizar por medio de la fórmula 5.2, donde  $Y_1 = L_t A_1$  y  $Y_2 = L_t^T A_2$ , con  $A_1 \in \mathbb{R}^{n \times r}$  y  $A_2 \in \mathbb{R}^{m \times r}$  matrices aleatorias. El Algoritmo 3 presenta el método GoDec modificado.

$$L_{t+1} = Y_1 (A_2^T Y_1)^{-1} Y_2^T \quad (5.2)$$

---

**Algoritmo 3:** GoDec modificado con el método BRP.

---

```

Input:  $k, r, q, A_{m \times n}$ 
Result:  $L, S$ 
1  $L_0 := A, S_0 := \mathbf{0}_{m \times n}, t := 0;$ 
2 while Verdadero do
3    $t := t + 1;$ 
4    $L = A - S;$ 
5    $Y_2 = \text{randn}(n, r);$ 
6   for  $i=1:q+1$  do
7      $| Y_1 = L \cdot Y_2;$ 
8      $| Y_2 = L^T \cdot Y_1;$ 
9   end
10   $QR = qr(Y_2);$ 
11   $L_t := (L \cdot Q) \cdot Q^T;$ 
12   $S_t := \mathcal{P}_\Omega(A - L_t);$ 
13   $E_t := \frac{\|A - L_t - S_t\|_{fr}^2}{\|A\|_{fr}^2};$ 
14  if  $|E_t - E_{t-1}| < tol$  then
15    break
16  end
17 end
18 Retornar  $L, S$ 

```

---

Ahora se procederá con la descripción de las funciones del toolbox.

## 5.1 GoDec

Esta función calcula las matrices  $L$  y  $S$  correspondientes a la solución del problema 5.1. Recibe los siguientes argumentos:

- $X$ : Matriz de tamaño  $m \times n$ , para la que se resolverá el problema 5.1.
- $r$ : Es la cota máxima del rango de  $L$ . Este valor debe ser entero positivo y menor o igual que el rango de  $X$ .
- $k$ : Es la cardinalidad máxima que tendrá  $S$ . Este valor debe ser entero, positivo y menor que  $mn$ .
- $c$ : Parámetro del esquema de potencias utilizado por el método, se utiliza para mejorar la generación de las matrices de proyección bilateral (ver [10]). Debe ser entero, positivo y podría ser ajustado por el usuario. Su valor por defecto es 3.

- **Tol:** Tolerancia para la estabilización del error, tal que  $10^{-15} < \text{Tol} \leq 100$ . Su valor por defecto es  $10^{-8}$ .
- **IteraMax:** Número máximo de iteraciones realizadas por el algoritmo, en caso que no se alcance la tolerancia Tol. Este valor debe ser entero tal que  $1 < \text{IteraMax} < 10000$  y su valor por defecto corresponde a 100.
- **Opcion:** Etiqueta con la que se indica si se desea realizar GoDec con el método BRP o la SVD truncada. Las etiquetas válidas son 'BRP' y 'SVD'.

Esta función retorna:

- **L:** Matriz de tamaño  $m \times n$  y de rango menor o igual que r.
- **S:** Matriz de tamaño  $m \times n$  y de cardinalidad menor o igual a k.
- **Errors:** Vector en el que se almacena, para cada iteración, el valor del error calculado con la fórmula  $\|X - L - S\|_{fr}^2$ .

### 5.1.1 Sintaxis

```
1 [L,S,Errors] = GoDec(X,r,k,'Opcion','c',c,'IteraMax',IteraMax,'Tol',Tol)
```

Los parámetros c, IteraMax y Tol son opcionales.

#### Ejemplo

Considere la matriz:

```
A=[-3 4 5 -4; 2 3 -5 1; 6 -5 -2 5; 3 -6 1 2];
```

Al ejecutar la función GoDec con

```
1 rng(1); %Se establece la semilla en 1 para poder replicar el experimento.
2 X=[-3 4 5 -4; 2 3 -5 1; 6 -5 -2 5; 3 -6 1 2];
3 [L,S,Errors]=GoDec(X,2,10,'BRP','IteraMax',50,'Tol',0.000001);
```

Genera:

**L =**

-4.4074	3.1177	4.1548	-4.0026
1.2590	2.9964	-5.0038	1.3115
5.1303	-5.7649	-2.7389	4.5667
2.9935	-6.0047	0.9952	2.5504

**S =**

1.4074	0.8823	0.8452	0
0.7410	0	0	-0.3115
0.8697	0.7649	0.7389	0.4333
0	0	0	-0.5504

## 5.2 GoDecVideoFull

Esta función aplica el proceso de GoDec a un video y genera, en el directorio de trabajo, los videos `L.mp4` y `S.mp4`, construidos con las matrices `L` y `S`, respectivamente. Esta función recibe los siguientes argumentos:

- `Textpath`: Es la dirección del archivo de video con extensión `.mp4`, el cual será codificado internamente, en escala de grises, como una matriz `X`.
- `r`: Este valor es la cota máxima para el rango de la matriz `L` que se determina al resolver el problema 5.1. Debe ser entero positivo y menor que el rango de `X`.
- `k`: Es la cardinalidad máxima que tendrá la matriz `S`. Debe ser entero, positivo y menor que `mn · numFrames` (número total de pixeles en el video completo).
- `Scale`: Este valor se utiliza para escalar cada uno de los frames en una fracción del tamaño original. Este escalamiento es útil para videos con alta resolución. Debe cumplir que  $0 < \text{Scale} \leq 1$ . Su valor por defecto es 1, en cuyo caso se estaría utilizando el tamaño de los frames del video original.
- `numFrames`: Es el número de frames que se utilizarán en el video. Este valor es un entero positivo que debe ser menor o igual que el número total de frames que tiene el video original. Si se omite, se utilizarán todos los frames.
- `c`: Parámetro del esquema de potencias utilizado por el método, se utiliza para mejorar la generación de las matrices de proyección bilateral (ver [10]). Debe ser entero, positivo y podría ser ajustado por el usuario. Su valor por defecto es 3.
- `Tol`: Tolerancia para la estabilización del error, tal que  $10^{-15} < \text{Tol} \leq 100$ . Su valor por defecto es  $10^{-8}$ .
- `IteraMax`: Número máximo de iteraciones realizadas por el algoritmo, en caso que no se alcance la tolerancia `Tol`. Este valor debe ser entero tal que  $1 < \text{IteraMax} < 10000$  y su valor por defecto corresponde a 100.
- `Opcion`: Etiqueta con la que se indica si se desea realizar GoDec con el método BRP o la SVD truncada. Las etiquetas válidas son '`BRP`' y '`SVD`'.

### 5.2.1 Sintaxis

```
1 GoDecVideoFull('Textpath',r,k,'Opcion','Scale',Scale,'numFrames',numFrames,'c',c,'Tol',Tol,'IteraMax
',IteraMax)
```

Los parámetros `Scale`, `numFrames`, `c`, `Tol` e `IteraMax` son opcionales.

## 5.3 GoDecImageFull

Esta función aplica el proceso de GoDec a un conjunto de imágenes en una base de datos. La función recibe:

- `Textpath`: Es la dirección del directorio de las imágenes. Para la lectura del directorio se utiliza la función `ReadImageDataBase` (ver la sección 1.1). El número de

imágenes se denota con numImage. Las imágenes se codifican en una matriz  $X$  de tamaño  $mn \times \text{numImage}$ , que es a la que se le aplica el algoritmo GoDec.

- $r$ : Este valor corresponde a la cota máxima del rango de la matriz  $L$ . Debe ser entero positivo y menor que el mínimo entre el número de imágenes en la base de datos y el número de pixeles en cada imagen.
- $k$ : Este valor corresponde a la cardinalidad máxima que tendrá la matriz  $S$ . Debe ser entero, positivo y menor que  $mn \cdot \text{numImage}$ .
- $c$ : Parámetro del esquema de potencias utilizado por el método, se utiliza para mejorar la generación de las matrices de proyección bilateral (ver [10]). Debe ser entero, positivo y podría ser ajustado por el usuario. Su valor por defecto es 3.
- $\text{Tol}$ : Tolerancia para la estabilización del error, tal que  $10^{-15} < \text{Tol} \leq 100$ . Su valor por defecto es  $10^{-8}$ .
- $\text{IteraMax}$ : Número máximo de iteraciones realizadas por el algoritmo, en caso que no se alcance la tolerancia  $\text{Tol}$ . Este valor debe ser entero tal que  $1 < \text{IteraMax} < 10000$  y su valor por defecto corresponde a 100.
- **Opcion**: Etiqueta con la que se indica si se desea realizar GoDec con el método BRP o la SVD truncada. Las etiquetas válidas son 'BRP' y 'SVD'.

Al ejecutar esta función se crean dos carpetas en el directorio actual de trabajo, denominadas `ImagesForS` y `ImagesForL`. En la primera de ellas se guardarán las imágenes generadas con la matriz  $S$ , mientras que en la segunda, las imágenes generadas con la matriz  $L$ .

### 5.3.1 Sintaxis

```
1 GoDecImageFull('Textpath',r,k,'Opcion','c',c,'Tol',Tol,'IteraMax',IteraMax);
```

Los parámetros  $c$ ,  $\text{Tol}$  e  $\text{IteraMax}$  son opcionales.



## Capítulo 6

---

# ALGORITMO K-SVD APLICADO A LA RECONSTRUCCIÓN DE IMÁGENES CON PIXELES PERDIDOS

---

En este capítulo se mostrará una aplicación de un método, denominado K-SVD, que permite la recuperación de píxeles perdidos en una imagen, la cual se basa en el artículo [11]. Matemáticamente, el método se explica de la siguiente manera: dada la matriz  $Y \in \mathbb{R}^{m \times n}$ , se busca aproximar matrices  $D \in \mathbb{R}^{m \times t}$  y  $X \in \mathbb{R}^{t \times n}$ ,  $1 \leq t \leq \min\{m, n\}$ , que sean solución de

$$\min_{D, X} \|Y - DX\|_{fr}^2 \quad (6.1)$$

sujeto a las condiciones  $\text{Card}(x_i) \leq c_0$ , para  $i = 1, \dots, n$ , donde  $x_i \in \mathbb{R}^n$  representa la  $i$ -ésima columna de  $X$  y  $\text{Card}(x_i)$  representa la cantidad de entradas de  $x_i$  diferentes de cero. En este caso, la constante  $c_0$  es un número entero positivo que se conoce como constante de esparcidad y corresponde a un parámetro de entrada para el algoritmo.

El problema (6.1) busca realizar una representación esparcida de los datos de entrada, determinando un cierto número de patrones elementales, denominados *átomos*, que combinados entre sí permitan realizar una reconstrucción. El algoritmo que da solución al problema (6.1) se llama K-SVD y corresponde a un método iterativo que alterna entre la escasa codificación de los datos almacenados en la matriz actual  $D$ , denominada *diccionario*, y un proceso de actualización de los átomos del diccionario para que haya un mejor ajuste. En el Algoritmo 4 se presenta el pseudocódigo.

**Algoritmo 4:** Algoritmo K-SVD.

---

**Input:**  $Y \in \mathbb{R}^{m \times n}$ ,  $D^{(0)} \in \mathbb{R}^{m \times t}$ ,  $c_0 \in \{1, 2, \dots, n\}$ ,  $iter \in \mathbb{N}^*$   
**Result:**  $D^{(iter)} \in \mathbb{R}^{m \times t}$  y  $X^{(iter)} \in \mathbb{R}^{t \times n}$

```

1  $D = D^{(0)}$ ;
2 for  $k=1:iter$  do
3    $\forall i = 1, \dots, n$  determine  $x_i = \underset{x}{\operatorname{argmín}} \|y_i - Dx\|_2^2$ , tal que  $\operatorname{Card}(x) \leq c_0$ ;
4   Defina  $X = (x_1 \dots x_n)$ ;
5   Calcule  $R = Y - DX$ ;
6   for  $j=1:t$  do
7     Calcule  $I$ , el vector de índices asociados a las entradas de  $x_j$  no nulas;
8     Calcule  $\tilde{R} = R_I + d_j x_j(I)$ ;
9     Determine  $\tilde{R} = U \Sigma V^T$  (SVD de  $\tilde{R}$ );
10    Sustituya  $d_j = u_1$ , donde  $u_1$  es la primera columna de  $U$ ;
11    Sustituya  $x_j(I) = \sigma_1^2 v_1^T$ , donde  $\sigma_1$  es el primer valor singular de  $\tilde{R}$  y  $v_1$  es
        la primera columna de  $V$ ;
12  end
13  Defina  $D^{(k)} = (d_1 \dots d_t)$ ;
14 end

```

---

El Algoritmo 4, en el paso 3, plantea aproximar cada columna de  $X$ , sujeto a una condición de esparsidad. En este paso se utilizó el algoritmo *orthogonal matching pursuit* (OMP) (tomado de [12]). Por otra parte, en el paso 8 del Algoritmo 4 tome en cuenta que:

- $R_I$  es la matriz definida a partir de  $R$  tomando las columnas asociadas a los índices en  $I$ . Por ende, el tamaño de  $R_I$  es  $m \times \operatorname{Card}(I)$ .
- $d_j$  es la  $j$ -ésima columna de  $D$  (de dimensión  $m \times 1$ ).
- $x_j(I)$  es el vector construido con las entradas no nulas de  $x_j$  (usando para ello los índices almacenados en  $I$ ). Por lo tanto, los vectores  $I$  y  $x_j(I)$  tienen tamaño  $1 \times \operatorname{Card}(I)$ .

**Ejemplo de la aplicación en procesamiento de imágenes**

Como se mencionó anteriormente, una aplicación del algoritmo K-SVD consiste en la recuperación de pixeles perdidos en una imagen. Primero, debemos considerar una base de datos de  $s$  imágenes  $\mathcal{A} = \{A_1, A_2, \dots, A_s\}$ , donde  $A_j$  es una imagen de tamaño  $m \times n$ , para todo  $j = 1, \dots, s$ . Luego, se debe obtener la matriz  $Y$  a partir de bloques generados con cada imagen  $A_j$ , siguiendo los siguientes pasos:

- Cada imagen  $A_j$  se separa en bloques de tamaño  $p \times q$ , donde  $p$  y  $q$  son divisores de  $m$  y  $n$ , respectivamente.
- Se obtienen un total de  $\frac{mn}{pq}$  sub-bloques de tamaño  $p \times q$  de cada imagen  $A_j$  y  $r = \frac{smn}{pq}$  sub-bloques para toda la base de imágenes  $\mathcal{A}$ .
- Cada sub-bloque se transforma en un vector  $y_j$  de tamaño  $pq \times 1$ .
- Finalmente, se define una matriz  $Y = [y_1 \ y_2 \ \dots \ y_r]$  de tamaño  $pq \times r$ .

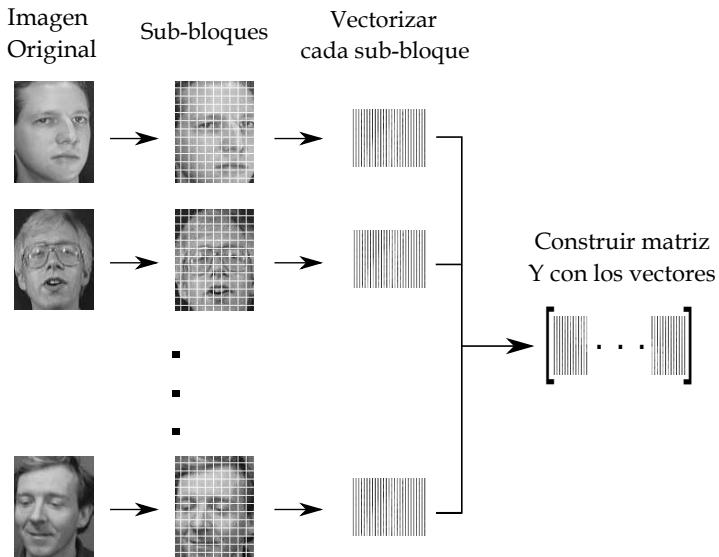


Figura 6.1: Representación matricial de las imágenes.

Para el ejemplo se utilizó la base de imágenes  $\mathcal{A}$  que puede descargar del siguiente vínculo. En la Figura 6.1 se visualiza una representación de los pasos a seguir para la construcción de la matriz  $Y$  a partir de  $\mathcal{A}$ . Posteriormente, se utiliza la matriz  $Y$  y se aplica el algoritmo K-SVD para obtener las matrices  $D^{(k)}$  y  $X^{(k)}$  que dan solución al problema (6.1). La matriz  $D^{(k)}$  corresponde al diccionario generado con el Algoritmo 4, que permite reconstruir una imagen con pixeles perdidos. La calidad de la reconstrucción depende de qué tan similar es la imagen por reconstruir, con respecto a las imágenes en  $\mathcal{A}$ . Para el ejemplo, se utilizó la siguiente imagen con pixeles perdidos.

Para reconstruir la imagen con pixeles perdidos, que también debe ser de tamaño  $m \times n$ , se divide la imagen en bloques de tamaño  $p \times q$  y se realiza la reconstrucción por bloques. Cada bloque se transforma en un vector  $z_j$  de tamaño  $pq \times 1$  y para cada bloque se realizan los siguientes pasos:

- El vector  $z_j$  puede tener algunas entradas iguales a 0, las cuales representan los pixeles perdidos. Utilizando esta información se define la *matriz diezmada* asociada a  $z_j$ , denotada  $D_{z_j}$ , de la siguiente manera:

$$D_{z_j}(i,:) = \begin{cases} D^{(k)}(i,:) & \text{si } z_j(i) \neq 0 \\ \mathbf{0} & \text{si } z_j(i) = 0 \end{cases},$$

donde  $D_{z_j}(i,:)$  y  $D^{(k)}(i,:)$  representan la fila  $i$  de las matrices  $D_{z_j}$  y  $D^{(k)}$ , respectivamente, y  $\mathbf{0}$  es el vector fila nulo. Esto es, la fila  $i$  de  $D_{z_j}$  se toma igual a la fila  $i$  de  $D^{(k)}$ , en caso que la  $i$ -ésima entrada del vector  $z_j$  no sea cero. Caso contrario, a la fila  $i$  de  $D_{z_j}$  se le asigna todas sus entradas iguales a cero.

- Utilizando un algoritmo de *compressed sensing* se determina  $x_{z_j}$ , tal que

$$x_{z_j} = \underset{x}{\operatorname{argmín}} \|z_j - D_{z_j}x\|_2^2,$$

sujeto a  $\text{Card}(x_{z_j}) \leq c_0$ . En este caso nuevamente se utilizó el algoritmo OMP.

- Finalmente, para reconstruir el  $j$ -ésimo bloque se realiza la multiplicación  $D^{(k)} \cdot x_{z_j}$ . El resultado de este producto se convierte en un bloque de tamaño  $p \times q$ , y corresponde a la reconstrucción del  $j$ -ésimo bloque de la fotografía.

Para efectos del ejemplo, la Figura 6.2 muestra los resultados obtenidos en la reconstrucción de la fotografía seleccionada.

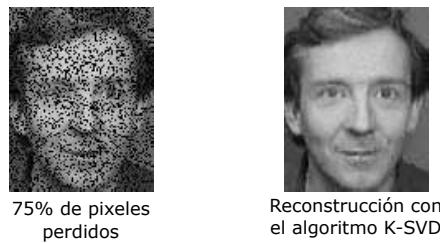


Figura 6.2: Reconstrucción de una imagen con 75% de pixeles perdidos mediante el algoritmo K-SVD.

Ahora se procederá con la descripción de las funciones del toolbox.

## 6.1 ksvd

Esta función calcula la matriz diccionario  $D$  y la matriz esparcida  $X$  del método K-SVD. Recibe los siguientes argumentos:

- **Textpath**: Ruta de la carpeta con las imágenes de entrenamiento.
- **r**: Número de columnas del diccionario (número de átomos).
- **c**: Constante de esparcidad.
- **Extension**: Las extensiones que se aceptan son jpg, pgm, png, tif, bpm. El valor por defecto es jpg
- **IteraMax**: Número máximo de iteraciones a realizar. El valor por defecto es 200.
- **Tol**: Tolerancia, para el método de paro, diferencia entre dos errores consecutivos menor que Tol. El valor por defecto es  $10^{-3}$ .

Esta función retorna:

- **D**: Matriz diccionario de tamaño  $m \times r$ .
- **X**: Matriz esparcida de tamaño  $r \times n$ .
- **Err**: Error relativo normalizado del método.

### 6.1.1 Sintaxis

```
1 [D,X,Err]=ksvd('Textpath',r,c,'Extension','.ext','IteraMax',IteraMax,'Tol',Tol);
```

Los parámetros **Extension**, **IteraMax** y **Tol** son opcionales.

## 6.2 clean\_ksvd

Esta función recibe la ruta de una imagen que presenta píxeles perdidos y la reconstruye utilizando la matriz de diccionario  $D$  generada por el método K-SVD. Esta función debe utilizarse posterior a la función `ksvd`, que es la que genera la matriz  $D$ .

Recibe los siguientes argumentos:

- `Textpath`: Ruta de la imagen que se desea reconstruir.
- `D`: Matriz diccionario generada por el método K-SVD.

Esta función retorna:

- `Y`: Matriz que representa la imagen reconstruida.

### 6.2.1 Sintaxis

```
1 Y=clean_ksvd('Textpath',D);
```



## REFERENCIAS

---

- [1] J. Chung and M. Chung, “Computing optimal low-rank matrix approximations for image processing,” in *2013 Asilomar Conference on Signals, Systems and Computers*, Nov 2013, pp. 670–674.
- [2] S. Friedland and A. Torokhti, “Generalized rank-constrained matrix approximations,” *SIAM Journal on Matrix Analysis and Applications*, vol. 29, no. 2, pp. 656–659, 2007.
- [3] A. Georghiades, P. Belhumeur, and D. Kriegman, “From few to many: Illumination cone models for face recognition under variable lighting and pose,” *IEEE Trans. Pattern Anal. Mach. Intelligence*, vol. 23, no. 6, pp. 643–660, 2001.
- [4] D. Lee and H. Seung, “Algorithms for non-negative matrix factorization,” *Adv. Neural Inform. Process. Syst.*, vol. 13, 02 2001.
- [5] C. Eckart and G. Young, “The approximation of one matrix by another of lower rank,” *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.
- [6] Z. Allen-Zhu and Y. Li, “LazySVD: Even faster SVD decomposition yet without agonizing pain,” in *Advances in Neural Information Processing Systems*, 2016, pp. 974–982.
- [7] M. Fazel, E. Candes, B. Recht, and P. Parrilo, “Compressed sensing and robust recovery of low rank matrices,” in *2008 42nd Asilomar Conference on Signals, Systems and Computers*. IEEE, 2008, pp. 1043–1047.
- [8] T. Zhou and D. Tao, “Bilateral random projections,” in *2012 IEEE International Symposium on Information Theory Proceedings*. IEEE, 2012, pp. 1286–1290.
- [9] J. Chavarría-Molina, J. J. Fallas-Monge, and P. Soto-Quiros, “Effective implementation to reduce execution time of a low-rank matrix approximation problem,” *Journal of Computational and Applied Mathematics*, vol. 401, p. 113763, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S037704272100385X>

- [10] T. Zhou and D. Tao, “Godec: Randomized low-rank & sparse matrix decomposition in noisy case,” in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, 2011.
- [11] M. Aharon, M. Elad, and A. Bruckstein, “K-svd: An algorithm for designing overcomplete dictionaries for sparse representation,” *IEEE Transactions on signal processing*, vol. 54, no. 11, pp. 4311–4322, 2006.
- [12] Y. Eldar and G. Kutyniok, *Compressed Sensing: Theory and Applications*, ser. Compressed Sensing: Theory and Applications. Cambridge University Press, 2012.