1er Coloquio del Departamento de Matemáticas

Problemas Inversos en Procesamiento de Imágenes

Joaquín Delgado
Mario Medina

Problemas Inversos en Procesamiento de Imágenes

Joaquín Delgado
Mario Medina

Casa abierta al tiempo
Universidad Autónoma Metropolitana

Departamento de Matemáticas, Universidad Autónoma
Metropolitana-Iztapalapa.

Contenido

1. Procesamiento de imágenes con Matlab

En esta sección veremos cómo se trabaja con imágenes en Matlab, la forma en que podemos guardar en un archivo una imagen que está dada en uno de los múltiples formatos que soporta este entorno de programación. Usaremos distintos comandos del Toolbox de *Image processing* de Matlab.

Una imagen digital, se compone de pixeles, a los cuales podemos pensar como los puntos de una pantalla. Una imagen digital nos dice como se le da color a cada pixel. Una imagen puede constar de 512×512 pixeles. Por lo que la imagen consta de 262144 y requiere de una buena cantidad de memoria en la computadora. Así, si queremos tener un manejo eficiente de la cantidad de memoria, debemos lograr o un buen sistema de compresión de imágenes o buscar almacenar la información más importante de la imagen.

Una imagen es una señal que puede representarse, en el caso de imágenes en blanco y negro, es decir, en escala de grises, por una matriz `M` de tamaño $m \times n$, donde cada entrada `M(i,j)` de la matriz representa el valor en la escala de grises para el pixel correspondiente. Para una imagen en color, su representación matemática está dada por un arreglo tridimensional de tamaño $m \times n \times 3$, donde los subarreglos bidimensionales `M(:,:,1)`, `M(:,:,2)` y `M(:,:,3)` representan respectivamente, la cantidad de rojo, de verde y de azul de la imagen original.

Como siempre que trabajamos con Matlab, es necesario establecer primero el directorio donde tenemos guardadas las imágenes con las que se trabajará y donde también se guardarán las imágenes que obtendremos a partir de las originales, así como los m-archivos que generemos.

Uno de los formatos más usuales al trabajar con imágenes en el formato `*.jpg`; para poder transformarla a otro formato existen distintas manera de hacerlo.

La primera de ellas es transformar la imagen a escala de grises, lo cual es una de las formas más habituales. Una imagen la representamos como una matriz donde a cada entrada le asignemos un valor el cual nos dice que tan brillante u oscuro es el pixel correspondiente a esta entrada. Una forma de realizar esto es asignar un dato de tipo double a esta información mediante un número de punto flotante entre 0 y 1. El valor 0 corresponde al negro y el valor 1 al blanco. La segunda manera de asignar el gado de brillantez u oscuridad del pixel es darle un tipo de dato `uint8`, el cual le asigna un entero entre 0 y 255 para representar el brillo a un pixel dado. El valor 0 corresponde al negro y el valor 255 al blanco. Observemos que La clase uint8 solo necesita 1/8 del almacenamiento que requiere la clase double. Pero, por otra parte, muchas funciones matemáticas solo pueden aplicarse a números en la clase double.

En el formato de imagen binaria se almacena una imagen como una matriz, pero los únicos valores que le asignamos son 0 o 1, el primer valor corresponde a darle el color negro al pixel; mientras que con el segundo valor le asignamos el color blanco. Es decir, un pixel es blanco o negro.

Otro de los formatos donde podemos guardar una imagen es como una imagen indexada. Para ello usamos dos matrices: la primera contiene la información acerca del tamaño de imagen y un número para cada pixel, mientras que la segunda matriz, llamada mapa de color, es tal que su tamaño puede ser diferente al tamaño de la imagen. Los valores contenidos en la primera matriz son las instrucciones de qué color usar de acuerdo al mapa de color dado por la segunda matriz.

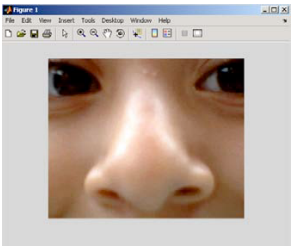


FIGURA 1.1

El último formato que mencionaremos es el de una imagen RGB, usado para imágenes en color y que se representa por un arreglo tridimensional, compuesta por tres matrices del mismo tamaño, que la imagen. La primera matriz corresponde al color rojo, mientras que la segunda corresponde al color verde y finalmente, la tercera da la información correspondiente al color azul de cada uno de los pixeles de la imagen.

El comando que permite leer una imagen es `imread( )`. Como lo muestra el siguiente ejemplo.

```
>> autoMM=imread('autorretratoM.jpg','jpg');
```

Este comando establece que los datos de la imagen se leen del archivo llamado `autorretratoM.jpg` y quedará almacenado en el archivo autoMM. De la misma manera que Matlab leyó un archivo de tipo *.jpg también puede leer archivos en formatos `*.bmp`,`*.tiff`, `*.pcx`, `*.gif`,`*.png`,`*xwd`.

A continuación obtenemos las principales características del archivo AutoMM

```
>> whos
  Name          Size                        Bytes  Class

  autoMM      288x352x3                   304128  uint8 array

Grand total is 304128 elements using 304128 bytes
```

Con lo que confirmamos que nuestra imagen está representada como un arreglo tridimensional de tamaño $288 \times 352 \times 3$, del tipo `uint8`.

Para ver la imagen usamos el comando imshow (Figura 1.1)

```
>>imshow(I)
```

Aparecerá en la ventana de imágenes la correspondiente al archivo que hemos llamado `I`.

FIGURA 1.2

Para transformar una imagen en colores RGB a una escala de grises (Figura 1.2) se usa el comando

```
>> I1=rgb2gray(I)
>> imshow(I1)
```

Para obtener la información de las dos imágenes escribimos el siguiente comando:

```
>> whos
  Name      Size                Bytes  Class
  I       288x352x3             304128  uint8 array
  I1      288x352               101376  uint8 array

Grand total is 405504 elements using 405504 bytes
```

Observamos que la imagen está representada como una matriz (bidimensional) de tamaño 288×352; es decir, el archivo tiene la forma de una función $I1 = I1(x, y)$, donde las coordenadas (x, y) son las entradas correspondientes a la componente (x, y) de la matriz, x corresponde a las renglones y y a las columnas de la misma. Observemos que las coordenadas se leen de la *esquina superior izquierda hacia abajo y hacia la derecha.*

También podemos considerar la submatriz de tamaño 30×30 de la imagen `I1`, submatriz dada por `I1(15:45,1:30)`, la cual corresponde al ojo derecho de la imagen en grises (Figura 1.3)

Por otra parte, si queremos ver nuestra imagen debemos seleccionar la paleta de colores mediante el comando colormap que define la función de color que se usará. El comando

```
>> help colormap
```

provee de más información sobre esta función.

La sucesión de comandos

```
>> colormap('jet');
>> imagesc(autoMM)
>> axis equal
>> axis off
```

FIGURA 1.3

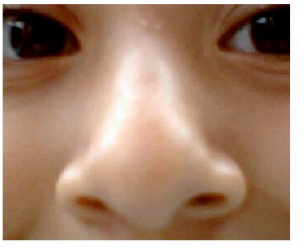


FIGURA 1.4

logra que la imagen sea una imagen donde no aparezcan los ejes (Figura 1.4)

Como hemos mencionado, en Matlab una imagen a color está representada por un arreglo tridimensional y podemos realizar operaciones en nuestro arreglo tridimensional, por ejemplo,

```
>> NegautoMM=255-autoMM;
>> imagesc(NegautoMM);
>> axis equal
>> axis off
```

y obtenemos la siguiente imagen (Figura 1.5).

```
>> autoMM=imread('autorretratoM.jpg','jpg');
>> colormap('jet');
>> imagesc(autoMM)
>> axis equal
```

FIGURA 1.5

```
>> axis off
>> imagesc(autoMM(1:144,1:176));axis equal; axis off;
```

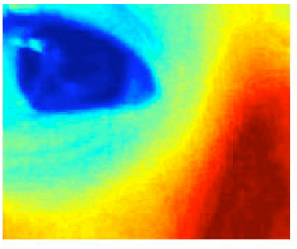


FIGURA 1.6

```
>> imagesc(autoMM(1:144,1:176,1:3));axis equal; axis off;
```

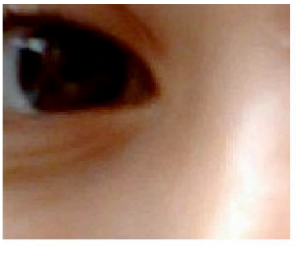


FIGURA 1.7

```
>> autoMM1=autoMM(:,:,1);
>> imagesc(autoMM1)
```

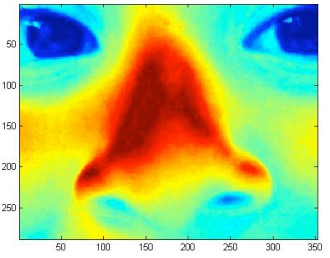


FIGURA 1.8

Escriba el lector los siguientes comandos en la ventana de comandos, vea las gráficas y de una explicación de las gráficas que ha obtenido.

```
>> autoMM1=autoMM(:,:,1);
>> imagesc(autoMM1)
>> autoMM2=autoMM(:,:,2);
>> imagesc(autoMM2)
>> autoMM3=autoMM(:,:,3);
>> imagesc(autoMM3)
>> auto1MM=autoMM(1,:,:);
>> imagesc(auto1MM)
>> autoM1M=autoMM(:,1,:);
>> imagesc(autoM1M)
```

Para grabar en un archivo el contenido de una imagen usamos el comando `imwrite`, por ejemplo,

```
>>imwrite(variable, 'nombre_archivo.jpg')
```

A continuación presentamos una serie de comandos de Matlab que nos permiten cambiar el formato de una imagen.

FORMATO ORIGINAL	FORMATO NUEVO	COMANDO
Escala de grises,indexada,RGB	Binario	`dither()`
Escala de grises	indexado	`gray2ind()`
Indexado	Escala de grises	`ind2gray()`
Indexado	RGB	ind2rgb()
RGB	Escala de grises	`rgb2gray()`
RGB	Indexado	`rgb2ind()`

Es de observar que cuando almacenemos una imagen debemos usar el tipo `uint8`, pues como hemos mencionado se requiere menos memoria que con el tipo double. Más adelante al procesar una imagen, se realizan operaciones matemáticas sobre la o las matrices que la representan, por lo que es necesario convertir la imagen a valores tipo double, lo que podemos logar con el comando

```
>>I=im2double(I);
```

el cual convierte una imagen `I` de tipo `uint8` a tipo double.

Por otra parte, el comando

```
>>I=im2uint8(I);
```

transforma una imagen llamada `I` de double a `uint8`.

Uno de los problemas que surgen al trabajar con imágenes es el siguiente. Se tiene una imagen con una muy alta resolución y esta ocupa un cierto espacio en el disco duro de la PC, ?‘cómo lograr tener una imagen que a simple vista sea indistinguible de la imagen original, pero la cual ocupe un menor espacio en el disco duro?

Consideremos la imagen estándar de Lena:

FIGURA 1.9

Una forma simple de obtener una imagen que ocupe una menor cantidad de bytes, es la siguiente. Primero se leer la imagen con el comando `imread`,

```
>> lena=imread(lena.gif)
```

Con el comando whos obtenemos las características de la imagen, en este caso la imagen tiene un tamaño de 512×512 y esta ocupa 262144 bytes en formato `uint8`. A continuación se considera la submatriz de renglones y columnas impares

```
>> lena1=lena(1:2:end,1:2:end);
```

Procedemos a guardar la imagen. Al leerla con `imread` obtenemos la imagen de la Figura 1.10

FIGURA 1.10

Esta imagen es casi la misma que la original, tiene un tamaño de 256×256 y ocupa 65536 bytes de memoria también en formato `uint8`. Ahora observemos la Figura 1.11 Aquí usamos sólo uno de cada 10 renglones y una de cada 10 columnas, obteniendo una matriz de tamaño 52×52, la cual ocupa un espacio de 2704 bytes. Como vemos, la imagen obtenida dista mucho de tener una buena resolución.

1.1. **Añadir ruido a una imagen.** Otro de los temas interesante al tratar con imágenes corresponde a la manera de filtrar el ruido a una imagen que lo posea. Para ello, Matlab posee diversas formas de ruido que se pueden agregar a una imagen, con el fin de probar métodos de filtrado. Veamos primero la manera de añadir ruido a una imagen.

La manera más simple es buscar información con respecto al comando `imnoise` en la ventana de ayuda. A continuación consideraremos

FIGURA 1.11

algunos ejemplos de ruido agregados a la imagen de Lena. El comando para añadir ruido es

```
>>imnoise(imagen,'tipo de ruido','par\'ametros')
```

Donde imagen representa la imagen original a la cual deseamos añadir ruido, `'tipo de ruido'` representa el tipo de ruido que deseamos añadir, el cual puede ser ruido gaussiano blanco con varianza y media constante (por default, los valores considerados son media nula y varianza igual a 0.01), sal y pimienta (`salt & pepper`), ruido tipo Poisson, ruido multiplicativo y ruido gaussiano con media nula y varianza dependiente de la intensidad. Para mayor información, el lector puede consultar la ventana de ayuda.

Añadiremos ruido gaussiano, de tipo salt & pepper y ruido tipo Poisson. La imagen con ruido gaussiano será guardada como `lena_gauss`,la que tiene ruido tipo sal y pimienta en `lena_salt` y la imagen con ruido de Poisson en `lena_poi`. Para ello ejecutamos siguientes comandos

```
lena=imread('lena.gif','gif');
lena_gauss=imnoise(lena,'gaussian');
lena_salt=imnoise(lena,'salt & pepper');
lena_poi=imnoise(lena,'poisson');

subplot(2,2,1),subimage(lena),
title('imagen original Lena') axis off
subplot(2,2,2),subimage(lena_gauss),
title('ruido gaussiano') axis off
subplot(2,2,1),subimage(lena_salt),
title('ruido sal & pepper') axis off
subplot(2,2,2),subimage(lena_poi),
title('ruido poisson') axis off
```

La imagen original y las imágenes con ruido se muestran en la Figura 1.12

1.2. **Histograma de una imagen y su ecualización.** Para una imagen es posible obtener un histograma, donde se muestra la frecuencia de la intensidad de cierto tono en la escala de grises. A partir del histograma podemos darnos cuenta de si hay ciertos tonos preponderantes en la imagen; de ser así, podemos realizar una ecualización de la imagen con el fin de tener una imagen más plana y poder distinguir ciertos detalles que en la imagen original no pueden ser vistos con claridad.

Primero leemos una imagen en color, la convertimos a escala de grises, y calculamos su histograma. Posteriormente se equaliza y esta ecualización se transforma en una imagen. Finalmente obtenemos el histograma de la imagen ecualizada.

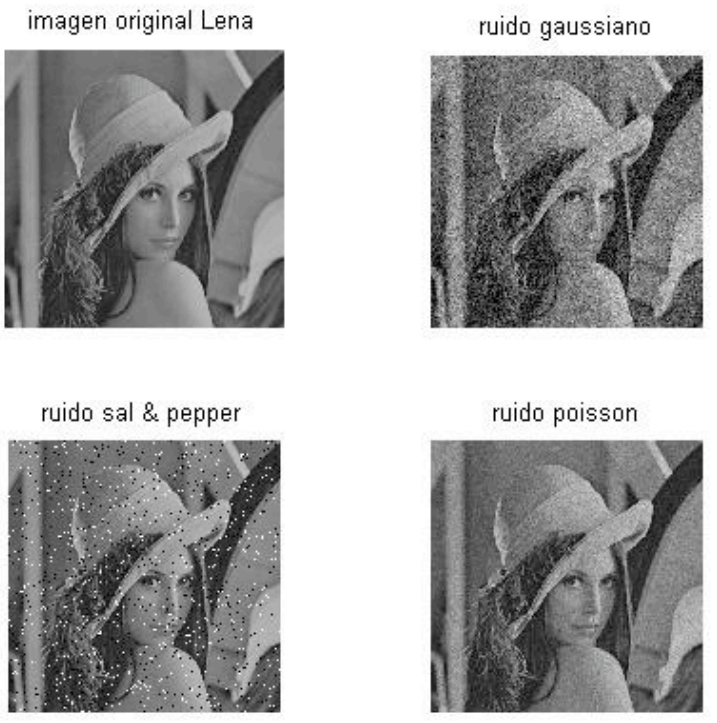


FIGURA 1.12

```
I=imread('mmrauto.jpg','jpg');
I1=rgb2gray(I); %convertir a escala de grises
I2=histeq(I1); %equalizador
subplot(2,2,1), imshow(I1) subplot(2,2,2), imhist(I1,80)
%grafico en escala de grises e histograma
subplot(2,2,3), imshow(I2) subplot(2,2,4), imhist(I2)
%grafico de imagen equalizada e histograma
```

A continuación mostramos las imágenes con sus histogramas correspondientes.

El histograma de la primera imagen en escala de grises muestra que esta no presenta demasiado contraste, mientras que el histograma de la imagen ecualizada muestra una mejor distribución.

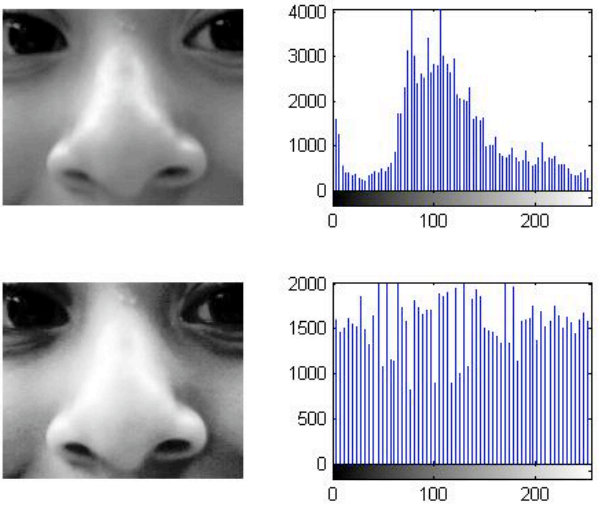


FIGURA 1.13

2. FILTRADO

Los filtros son usados para disminuir efectos no deseados en una imagen, como pueden ser ruido debido a la obtención de la imagen, o efectos de su digitalización o transmisión por algún medio.

Como estamos interesados en imágenes que por definición es un arreglo plano y finito de pixeles, comenzaremos por definir un sistema discreto general en dos variables discretas (indices discretos), como sigue: sea $x_{kl},\ k,l \in \mathbb{Z}$ la señal de entrada[1]; la señal de salida será $y_{ij} = T(x_{kl}),\ i,j \in \mathbb{Z}$ (véase la Figura 2.1). En principio la señal de entrada y salida son números reales que se refieren a escala de grises o de color, aunque también se pueden considerar complejos, si se quiere incluir alguna fase, por ejemplo. Ocasionalmente tendremos oportunidad de considerar imágenes 1-dimensionales, como cuando se considera una fila o columna del total de la imagen (ver Figura 2.2) en cuyo caso las señales de entrada y salida serán denotadas por $x = (x_k)$ y $y = (y_i)$.

[1]Por simplicidad supondremos que las señales de entrada y salida están definidas en toda la retícula de índices enteros, positivos y negativos. Más adelante veremos que hay diversos criterios para extender la señal de entrada fuera de la retícula rectángular en la que está orginalmente definida la imagen: $k = 1, 2, \ldots, M-1$, $l = 1, 2, \ldots, N-1$

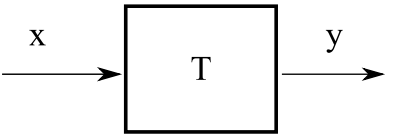


FIGURA 2.1. Sistema discreto

FIGURA 2.2. Imagen del "cameraman" y la imagen 1-dimensional de la fila 100

Para un sistema lineal discreto podemos escribir

$$y_{ij} = \sum_{k,l \in \mathbb{Z}} h_{ij,kl} x_{kl}, \tag{2.1}$$

para ciertos coeficientes reales o complejos $h_{ij,kl}$. Si el sistema lineal (2.1) es invariante bajo traslaciones entonces $h_{ij,kl} = h_{i-k,j-l}$ y el sistema lineal tiene la forma de una convolución discreta:

$$y_{ij} = \sum_{k,l} h_{i-k,j-l} x_{kl} \tag{2.2}$$

por ello $h_{m,n}$ se conoce como el *kernel de convolución.* También se conoce como *la respuesta al impulso unitario* debido a que si la señal de entrada es $x = \delta^{(mn)}$ donde

$$\delta_{kl}^{(mn)} = \begin{cases} 1 & \text{si } k = m \text{ y } l = n \\ 0 & \text{en otro caso} \end{cases}$$

y hacemos variar (m,n) entonces la respuesta es

$$\begin{aligned} y_{ij} &= \sum_{k,l} h_{i-k,j-l}\delta_{kl}^{(mn)} \\ &= h_{i-m,j-n} \end{aligned}$$

que como i,j,m,n son arbitrarios, entonces la respuesta al impulso unitario es precisamente h.

De acuerdo a (2.2), la respuesta $y=(y_{ij})$ a una entrada arbitraria $x=(x_{kl})$ es la convolución de h con x

$$y_{ij} = \sum_{k,l} h_{i-k,j-l}x_{kl} = \sum_{k,l} h_{k,l}x_{i-k,j-l}; \tag{2.3}$$

esta última igualdad es cierta[2], pues hemos supuesto que la señal está definida en toda la retícula $(k,l)\in\mathbb{Z}\times\mathbb{Z}$.

Para referencia posterior escribimos la convolución discreta en dimensión uno

$$y_i = \sum_k h_{i-k}x_k = \sum_k h_k x_{i-k}. \tag{2.4}$$

2.1. **La convolución discreta en una dimensión.** Para entender mejor el efecto de la convolución consideremos el caso unidimensional (2.4). En la Figura 2.3 se muestra la operación de convolución con un kernel h_k de soporte $k\geq 0$ (es decir, es distinto de cero para $k\geq 0$). En este caso la convolución se reduce a

$$y_n = \sum_{k\in\mathbb{Z}} h_{n-k}x_k = \sum_{0\leq k\leq n} h_{n-k}x_k = \sum_{\substack{i+j=n\\ i,j\geq 0}} h_i x_j$$

Los primeros términos son:

$$\begin{aligned} y_0 &= h_0x_0 \\ y_1 &= h_0x_1 + h_1x_0 \\ y_2 &= h_0x_2 + h_1x_1 + h_2x_0 \end{aligned}$$

se muestra como se genera paso a paso la convolución. En el primer paso, se invierte la gráfica del kernel respecto de la vertical y se desplaza sucesivamente de izquiera a derecha. Los valores que quedan encerrados en el cuadro son los que se multiplican para obtener la convolución. El algoritmo completo puede resumirse así:

(1) Invertir el kernel h_{-k}
(2) Desplazar el kernel por el índice n, h_{n-k}
(3) Multiplicar y sumar. Los términos que quedan alineados.

[2]Si hacemos el cambio de variable $\bar{k}=i-k$, $\bar{l}=j-l$, entonces $\sum_{k,l\in\mathbb{Z}} h_{i-k,j-l}x_{kl} = \sum_{\bar{k},\bar{l}\in\mathbb{Z}} h_{\bar{k},\bar{l}}x_{i-\bar{k},j-\bar{l}} = \sum_{k,l\in\mathbb{Z}} h_{k,l}x_{i-k,j-l}$

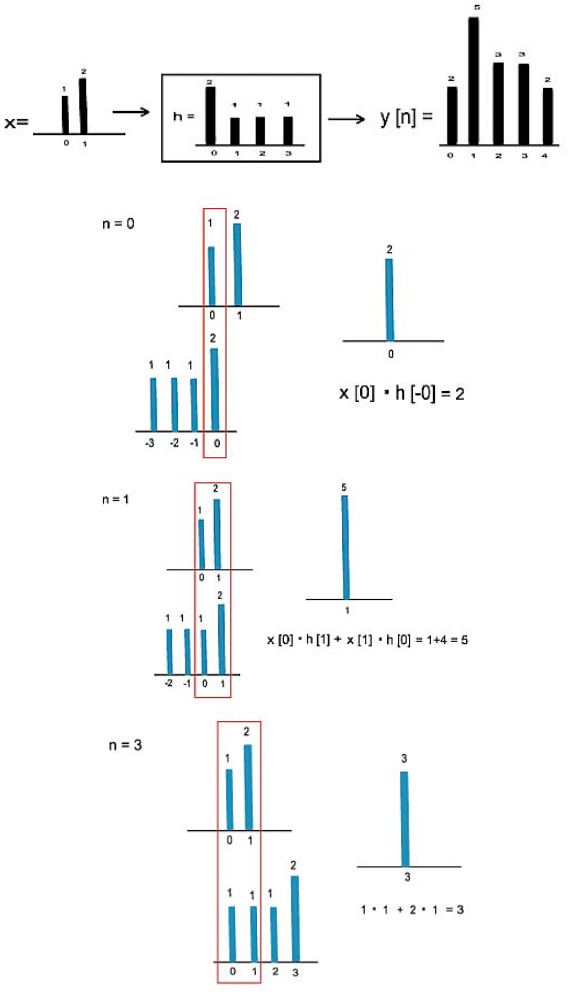


FIGURA 2.3. Resultado de la convolución (arriba) y cálculo paso a paso.

En la Figura 2.4 se muestra la convolución de la señal continua $f(t)$ con el kernel continuo

$$[f * g](t) = \int_{-\infty}^{\infty} f(t)g(t-\tau)\,d\tau$$

en completa analogía con el caso discreto.

En la Figura se muestran dos ejemplos de convolución de unaseñal de entrada x superposición de una de alta y otra de baja frecuencia. El

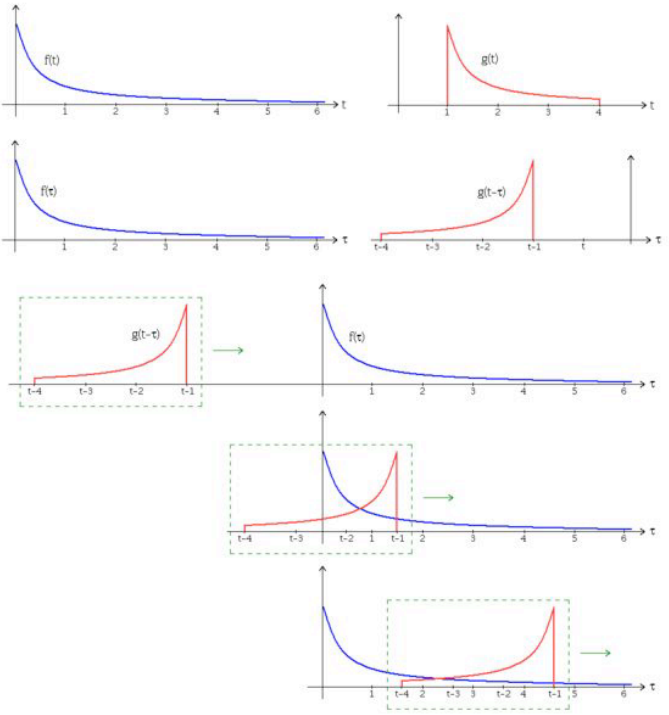


FIGURA 2.4. Convolución de la señal continua $f(t)$ con el kernel contino $g(t)$. El soporte del kernel determina la región que que se multiplica e integra.

primero caso se puede considerar un filro “pasa bajo" y el segundo un filtro “pasa-alto".

2.2. **Convolución y transformada discreta de Fourier.** Considere una señal $(x_k)_{k=-\infty}^{\infty}$. Su transformada de Fourier en tiempo discreto (DTFT) es

$$X(\omega) = \sum_k x_k e^{-j\omega k},$$

con inversa

$$x_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega) e^{j\omega k}\, d\omega.$$

Si x_k es el voltaje a través de una resistencia de 1 Ohm. La potencia instantánea es

$$\frac{|x_k|^2}{R} = |x_k|^2$$

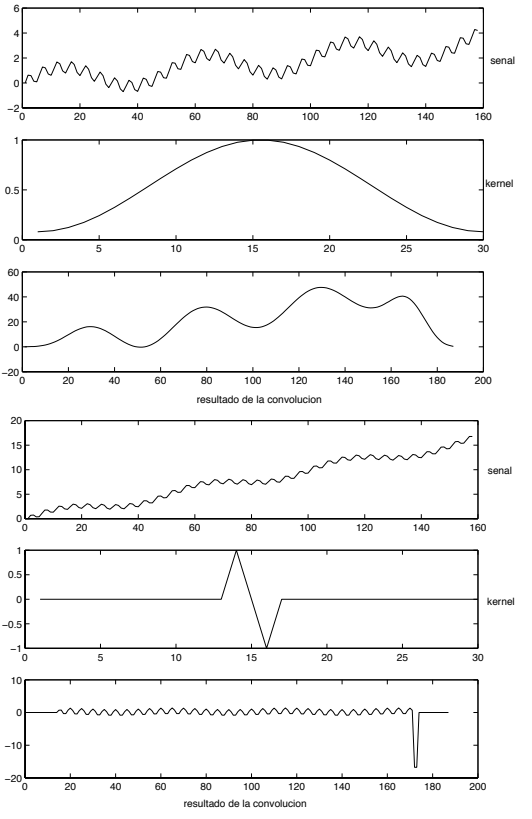


FIGURA 2.5. Filtro pasa-bajo y pasa-alto resultado de convolucionar con el kernel indicado.

y por lo tanto la energía total es

$$E = \sum_{k=-\infty}^{\infty} |x_k|^2.$$

La condición $E < \infty$ es suficiente para la existencia de la transformada de Fourier en tiempo discreto. Sin embargo, las señales de comunicación, determinísticas o estocásticas, tienen una duración infinita y su energía es infinita, de modo que su transformada de Fourier no está definida.

La DTFT $X(\omega)$ toma valores complejos, es una función continua y 2π-periódica. Al evaluar numéricamente la DTFT se presentan dos problemas:

- La secuencia x_k puede tener energía infinita.

- $X(\omega)$ es una función continua de la frecuencia ω y debe ser discretizada para trabajar en un procesador digital.

Para resolver el primer problema consideraremos que la secuencia ventaneada igual a cero, para $k = L, L+1, \ldots$.

Para el segundo, consideraremos que $X(\omega)$ se evalúa en un numero N finito de frecuencias equidistantes en el intervalo $[-\pi, \pi]$ con incrementos de $2\pi/N$, es decir se consideran el conjunto discreto de frecuencias $\omega_k = 2\pi k/N$ con $k = 0, 1, \ldots N-1$. Si se elige N lo suficientemente grande los valores $X(2\pi k/N)$ se aproximan a la función $X(\omega)$ continua origen del muestreo. Para evitar problemas de muestreo insuficiente se debe elegir N tal que $N > L$.

Al muestrear la DTFT de esta manera se obtiene la expresion correspondiente a la trasformada discreta de Fourier (DFT)

$$X_n = \sum_{k=0}^{N-1} x_k e^{-j2\pi nk/N}, \quad n = 0, 1, \ldots, N-1.$$

En *Matlab* se implementa mediante el algoritmo conocido como la tansformada rápida de Fourier (FFT: Fast Fourier Transform) descubierto por Cooley y Tukey en 1965. La idea es partir el cálculo de la DFT de un número compuesto de muestras $N = N_1 N_2$ en DFTs de tamaño menor N_1 y N_2. Por ello es conveniente usar, en tanto sea posible, muestras que sean de la forma $N = 2^p$.

En el caso bidimensional se definen de manera completamente análoga la DTFT y la DFT. Si consideremoas la señal bidimensional $(x_{kl})_{k,l\in\mathbb{Z}}$ la DTFT se define como la función de dos variables complejas

$$X(\omega_1, \omega_2) = \sum_{k,l\in\mathbb{Z}} x_{kl} e^{-j2\pi(\omega_1 k + \omega_2 l)}.$$

Si la señal es muestreada en una retícula de tamaño $M \times N$, entonces tenemos la DTF

$$X(m, n) = \sum x_{kl} e^{-j2\pi\left(\frac{mk}{N} + \frac{nl}{M}\right)}.$$

La razón principal de introducir la DTFT y la DFT, es la propiedad de convolución; Si X, Y son las transformadas de Fourier discretas de x, y respectivamente, la convolución (2.2) se puede expresar en el espacio de frecuencias como la multiplicación

$$Y = HX.$$

donde H se conoce como la *función de transferencia* y es la transformada de Fourier discreta de la respuesta al impulso unitario.

2.3. **Convolución discreta.** En esta sección vamos a considerar la convolución discreta de una señal discreta bidimensional que correspondería a una imagen plana en escala de grises. En la práctica se usa una *máscara* (el soporte de la función de respuesta al impulso unitario) que se desliza por toda la imagen combinando linealmente sus valores con los de la máscara.

Denotamos la convolución por $g = h * f$, explícitamente

$$g_{mn} = \sum_{k,l\in\mathbb{Z}} h_{mn} f_{k-m,l-n} = \sum_{k,l\in\mathbb{Z}} h_{m-k,n-l} f_{k,l}, \tag{2.5}$$

donde la última igualdad se da porque la señal y la función de respuesta están definidas en toda la retícula $\mathbb{Z}\times\mathbb{Z}$.

Debido a que la imagen es finita, en vez de (2.5) tendrem que considerar

$$g_{mn} = \sum_{\substack{k=0\\l=0}}^{M,N} h_{mn} f_{k-m,l-n},$$

donde la imagen tiene tamaño $M\times N$. Observe que la primera identidad en (2.5) requiere que la imagen esté definida fuera de la retícula finita, en tanto que la segunda identidad requiere lo mismo para la función de respuesta. Mostraremos mediante ejemplos sencillos cómo extender la imagen o el kernel fuera de la retícula finita de la imagen.

Consideremos el siguiente ejemplo. La máscara es

8	1	6
3	5	7
4	9	2

La imagen original tiene tamaño 5×5

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

A continuación damos se efectúan los siguientes pasos:

(1) Rote el kernel de convolución 180 grados alrededor de la celda central.

2	9	4
7	5	2
6	9	8

(2) Deslice la celda central de la máscara hasta la posición del elemento. Por ejemplo para calcular el valor del pixel de

salida $(2,4)$ se coloca la máscara (con números pequeños en la siguiente tabla) con su centro sobre en el pixel con valor 14

17	24	1^{2}	8^{9}	15^{4}
23	5	7^{7}	14^{5}	16^{3}
4	6	13^{6}	20^{1}	22^{8}
10	12	19	21	3
11	18	25	2	9

(3) Multiplique cada valor de la máscara por el pixel de la imagen debajo de la máscara y sume

$$1 \cdot 2 + 8 \cdot 9 + 15 \cdot 4 + 7 \cdot 7 + 14 \cdot 5 + 16 \cdot 3 + 13 \cdot 6 + 20 \cdot 1 + 22 \cdot 8 = 575$$

2.4. **Correlación discreta.** Este es otro tipo de operación similar a la convolución que prrsentamos por completez. La correlación del kernel h y la señal x es

$$y_{mn} = \sum_{k,l \in \mathbb{Z}} h_{km} x_{m+k,n+l} = \sum_{k,l \in \mathbb{Z}} h_{k-m,l-n} x_{kl}, \tag{2.6}$$

A diferencia de la convolución la máscara no se rota. Consideremos a manera de ejemplo, la misma máscara e imagen que en el caso que antes.

8	1	6
3	5	7
4	9	2

El algoritmo de correlación es el siguiente:

(1) Deslice la celda central de la máscara hasta la posición del elemento. Por ejemplo para calcular el valor del pixel de salida $(2,4)$ se coloca la máscara (con números pequeños en la siguiente tabla) con su centro sobre en el pixel con valor 14

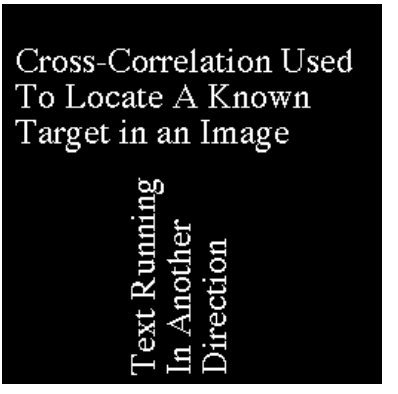
17	24	1^{8}	8^{1}	15^{6}
23	5	7^{3}	14^{5}	16^{7}
4	6	13^{4}	20^{9}	22^{2}
10	12	19	21	3
11	18	25	2	9

(2) Multiplique cada valor de la máscara por el pixel de la imagen debajo de la máscara y sume

$$1 \cdot 8 + 8 \cdot 1 + 15 \cdot 6 + 7 \cdot 3 + 14 \cdot 5 + 16 \cdot 7 + 13 \cdot 4 + 20 \cdot 9 + 22 \cdot 2 = 585$$

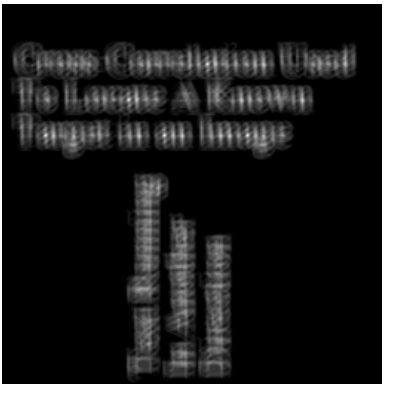
El siguiente ejemplo muestra el uso de la correlación para detectar un patrón conocido. Primero cargamos la imagen y seleccionamos el texto cuyas ocurrencias queremos detectar en la imagen; en este caso la letra ‘‘a".

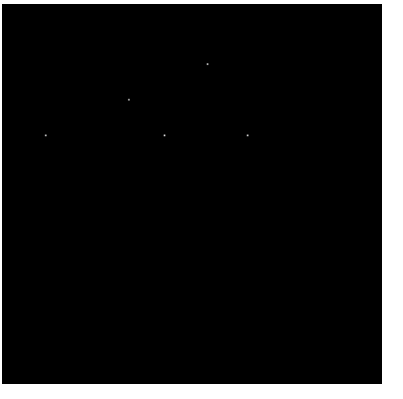
```
bw=imread('text.tif');
a=bw(59:71,81:91);
imshow(bw);
figure,imshow(a);
```



Para usar la correlación usamos la función `imfilter` que ya hemos usado antes, esta vez con la opción `'corr'`; previamente debemos convertir la imagen a double, ya que el formato `tif` que consta de valores lógicos de ceros y unos

```
h=double(a);
x=double(bw);
y=imfilter(x,h,'corr');
figure, imshow(y,[ ]); tresh= max(y(:))-10;
figure, imshow(y>tresh);
```





Los puntos que aparecen en el gráfico de umbral, son los que exceden un valor umbral ligeramente menor que el del valor máximo.

2.5. **Efecto de borde.** Debido a que toda imagen es finita, en cualquiera de los dos métodos de convolución o correlación, hay problema cuando la máscara sobrepasa los bordes la del imagen. Existen dos métodos de contornar el problema: (a) Rellenar con ceros y (b) replicación de bordes. En el primer método simplemente los valores de la imagen necesarios para efectuar la operación se extienden como cero; en el segundo caso, se repiten.

(a) Rellenar con ceros:

		?'Qué valor asignar?		
		$?^8$	$?^1$	$?^6$
17	24	1^2	8^9	15^4
23	5	7^7	14^5	16^3
4	6	13^6	20^1	22^8
10	12	19	21	3
11	18	25	2	9

		cero		
		0^8	0^1	0^6
17	24	1^2	8^9	15^4
23	5	7^7	14^5	16^3
4	6	13^6	20^1	22^8
10	12	19	21	3
11	18	25	2	9

(b) Replicación de bordes

		¿Qué valor asignar?		
		$?^8$	$?^1$	$?^6$
17	24	1^2	8^9	15^4
23	5	7^7	14^5	16^3
4	6	13^6	20^1	22^8
10	12	19	21	3
11	18	25	2	9

		valores replicados		
		1^8	8^1	15^6
17	24	1^2	8^9	15^4
23	5	7^7	14^5	16^3
4	6	13^6	20^1	22^8
10	12	19	21	3
11	18	25	2	9

En la Figura 2.6 se muestra la imagen orignal y el efecto de rellenar con ceros como un borde obscuro. En la Figura

2.6. **Densidad espectral de potencia.** Queremos encontrar la distribución de la energía y la potencia como función de la frecuencia ω. por el teorema de Parseval

$$E = \sum_{k=-\infty}^{\infty} |x_k|^2 = \int_{-\pi}^{\pi} |X(e^{j2\pi f})|^2 \, df$$

Por lo tanto la densidad espectral de energía es $\Phi_X(\omega) = \frac{1}{2\pi}|X(\omega)|^2$ o bien $\Phi_X(f) = |X(e^{j2\pi f})|^2$.

Como se ha mencionado, las señales de comunicación tiene duración infinita, por lo que típicamente su transformada de Fourier no está

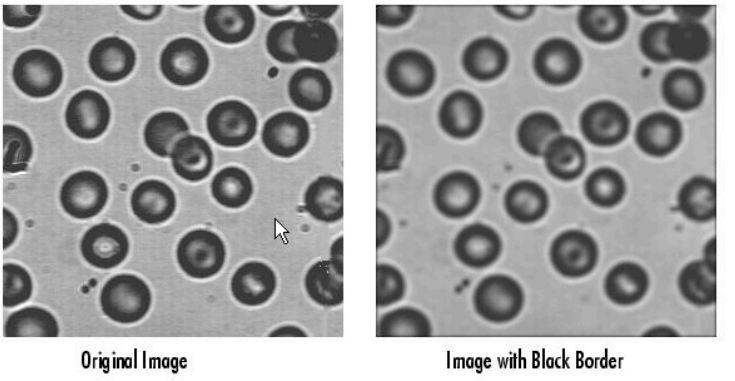


FIGURA 2.6. Imagen original y efecto de rellenar con ceros

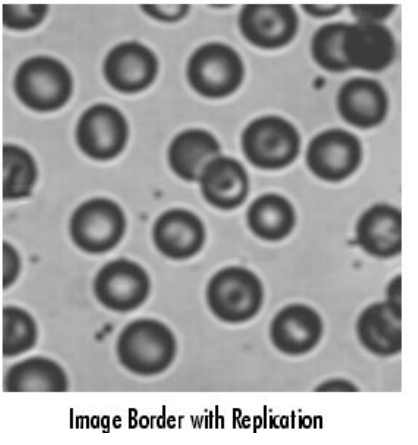


FIGURA 2.7. Efecto de replicación de bordes

definida. Otra alternativa es considerar *la potencia* en vez de la energía. La potencia promedio en el tiempo es

$$\frac{1}{2T}\sum_{k=-T}^{T}|x_k|^2$$

Nuevamente por la indentida de Parseval

$$\begin{aligned}\text{Potencia promedio} &= \lim_{T\to\infty}\frac{1}{2T}\sum_{k=-T}^{T}|x_k|^2\\ &= \lim_{T\to\infty}\int_{-\pi}^{\pi}|X(e^{j2\pi f})|^2\,df\\ &= \int_{-\pi}^{\pi}\lim_{T\to\infty}|X(e^{j2\pi f})|^2\,df\\ &= \int_{-\pi}^{\pi}S_X(2\pi f)\,df\end{aligned}$$

donde $S_X(2\pi f)$ es la *densidad espectral de potencia (PSD).* El teorema de *Wienner-Kintchine* asegura que la PSD es igual a la transformada de Fourier de la fución de autocorrelación; en el caso continuo entonces

$$S_X(\omega) = \int_{-\infty}^{\infty} r_{xx}(\tau)e^{-j\omega\tau}\,d\tau$$

donde

$$r_{xx}(\tau) = \lim_{T\to\infty}\int_{-T}^{T} x(t)x(t-\tau)\,dt; \tag{2.7}$$

para el caso discreto

$$S_X(\omega) = \sum_{k=-\infty}^{\infty} r_{xx}(k)e^{-j\omega k}$$

donde

$$r_{xx}(k) = \sum_{n=-\infty}^{\infty} x_n x_{n-k} \tag{2.8}$$

Si las señales son variables aleatorias, las relaciones (2.7) y (2.8) deben sustituirse por las esperanzas.

3. Deconvolución mediante un filtro de Wiener

Consideremos el sistema continuo

$$y(t) = h(t) * x(t) + v(t) \tag{3.1}$$

Donde * denota al operador convolución, y los otros elementos involucrados son:

- Una señal de entrada $x(t)$ (que no conocemos) al tiempo t.
- Un sistema lineal temporal-invariante $h(t)$, también conocida como respuesta al impulso unitario.
- Ruido aditivo desconocido $v(t)$, el cual es independiente de $x(t)$.
- La señal observada $y(t)$.

El objetivo es hallar alguna función $g(t)$ de manera que sea posible dar una estimación de $x(t)$ en forma de convolución:

$$\hat{x} = g(t) * y(t) \tag{3.2}$$

y de tal manera que el error cuadrático medio sea minimizado. Este tipo de filtros se llaman *óptimos*.

El filtrado de Wiener nos provee de esta función $g(t)$. Una manera de describir este filtro es usando el dominio de frecuencia:

$$G(f) = \frac{H(f)^* S(f)}{|H(f)|^2 S(f) + N(f)} \tag{3.3}$$

donde:

- $G(f)$ y $H(f)$ son, respectivamente las transformadas de Fourier de $g(t)$ y $h(t)$, en frecuencia f; $H(f)^*$ denota su conjugado.
- $S(f)$ es la densidad espectral de potencia media de la señal de entrada $x(t)$.
- $N(f)$ es la densidad espectral de potencia media del ruido $v(t)$.

La operación de filtrado puede realizarse tanto en el dominio temporal o en el dominio de frecuencia:

$$\hat{X}(f) = G(f)Y(f) \tag{3.4}$$

y posteriormente realizar la transformada inversa de Fourier de $\hat{X}(f)$ para obtener $\hat{x}(t)$. Es de observar que para el caso de imáagenes, los argumentos y considerados anteriormente son argumentos bidimensionales; pero aún con esto, el resultado el es mismo.

3.0.1. *Interpretación del filtro de Wienner.* La operación de filtrado de Wiener se hace más clara al reescribir la ecuación anterior como:

$$\begin{aligned} G(f) &= \frac{1}{H(f)} \left[\frac{|H(f)|^2}{|H(f)|^2 + \frac{N(f)}{S(f)}} \right] \\ &= \frac{1}{H(f)} \left[\frac{|H(f)|^2}{|H(f)|^2 + \frac{1}{SNR(f)}} \right] \end{aligned} \tag{3.5}$$

En esta expresión, $1/H(f)$ es la inversa del sistema original, $SNR(f)$ y es la razón entre la señal y el ruido. En el caso de que el ruido se anule (i.e. la razón entre señal y ruido sea infinita), entonces el término dentro de los paréntesis cuadrados es igual a 1, lo que significa que el filtro de Wiener es precisamente la inversa del sistema, como era de esperarse. Sin embargo, cuando el ruido a cierta frecuenta aumenta entonces la razón entre la señal y el ruido tiende a disminuir, de manera que el término dentro del paréntesis cuadrado también se hace más pequeño. Esto quiere decir que el filtro de Wiener atenúa las frecuencias que

dependen de la razón entre la señal y el ruido. La ecuación anterior para el filtro de Wiener necesita que conozcamos el contenido espectral de una imagen típica, asícomo del ruido. Usualmente, no se tiene acceso a estas cantidades exactas, pero quizá sípodemos tener una buena estimación de los mismos. Por ejemplo, en el caso de fotografías la señal (imagen original) posee de manera típica frecuencias bajas fuertes y frecuencias altas débiles y en muchos casos el ruido tendrá frecuencias relativamente planas.

3.0.2. *Deducción del filtro de Wienner.* Como se ha dicho anteriormente, deseamos obtener una estimación de la señal original que minimice el error cuadrático medio, el cual podemos expresar como:

$$\epsilon(f) = \mathbb{E}\,|X(f) - \hat{X}(f)|^2$$

donde $\mathbb{E}$ denota la esperanza. Si sustituimos el término $\hat{X}(f)$, y rearreglamos se tendrá:

$$\begin{aligned}\epsilon(f) &= \mathbb{E}[|X(f) - G(f)Y(f)|^2 \\ &= \mathbb{E}\,|X(f) - G(f)\,(H(f)X(f) + V(f))|^2 \\ &= \mathbb{E}\,|(1 - G(f)H(f))\,X(f) - G(f)V(f)|^2\end{aligned}$$

Al expandir el término cuadrado obtenemos (por simplicidad omitimos lo sucesivo el argumento f)

$$\begin{aligned}\epsilon = [1 - GH][1 - GH]^*\mathbb{E}|X|^2 + [1 - GH]G^*\mathbb{E}[XV^*] \\ + G\,[1 - GH]^*\,\mathbb{E}\left[XV^*\right] + GG^*\mathbb{E}|V|^2\end{aligned}$$

Pero, estamos suponiendo que el ruido es independiente de la señal, por lo que

$$\mathbb{E}\left[XV^*\right] = \mathbb{E}\left[XV^*\right]$$

Asimismo, de la definición de la densidad de potencia espectral,

$$\begin{aligned}S &= \mathbb{E}|X|^2, \\ N &= \mathbb{E}|V|^2.\end{aligned}$$

En consecuencia, se tiene

$$\epsilon = [1 - GH]\,[1 - GH]^*\,S + GG^*N.$$

Con el fin de hallar el valor mínimo del error, derivamos con respecto a G (es de notar que este en una variable compleja), e igualamos a cero:

$$\frac{d\epsilon}{dG} = G^*N - H\,[1 - GH]^*\,S = 0,$$

despejando G se obtiene la expresión (3.3) para el filtro de Wiener:

$$G = \frac{H^*S^*}{|H|^2S^* + N^*} = \frac{H^*S}{|H|^2S + N},$$

ya que las densidades espectrales de potencia S, N son reales.

El filtro bidimensional de Winner análog a (3.5) es

$$H(u,v) = \frac{H(u,v)}{|H(u,v)|^2 + N(u,v)/S(u,v)}$$

donde $H(u,v)$, $N(u,v)$ son las transformada de Fourier del núcleo (PSF) $h(t,s)$, y del ruido $v(t,s)$ respectivamente. El cociente N/S representa la razón ruido-señal.

4. Modelo de convolución de borrado de imágenes

El efecto de degradación de una imagen al pasar por un aparato óptico defectuoso, o los efectos aberrantes del medio óptico como la atmósfera, el agua, etc. pueden modelarse con gran generalidad por una sistema discreto general 2.1, que si el efecto de la degradación se puede suponer el mismo sobre toda la imagen de entrada, entonces es una convolución discreta. El ruido siempre está presente en mayor o menor grado, por lo que nuestro modelo de degradación de la imagen es el análogo de (3.1) en dimensión dos

$$y = h * x + v \tag{4.1}$$

donde x es la fuente, y la imagen y v es ruido que son que son matrices $\mathbb{R}^{M\times N}$, v una matriz estocástica, típicamente ruido blanco o Poisson. Llamaremos al modelo (4.1) *modelo de borrado de imágenes.*

Si se conocen los parámetros que definen a v y h hablamos del *problema directo*; en cambio, si se conoce y y se trata de recuperar x hablamos del *problema inverso*.

Un problema inverso $y = T(x)$—determinar x a partir del dato y—se dice *bien planteado a la Hadamard* si las siguientes condiciones son satisfechas

- Para toda y consistente con el problema, la solución existe.
- Para toda y consistente con el problema, la solución es única.
- La solución depende continuamente del dato.

En caso contrario hablamos de un problema inverso *mal planteado*.

Observe que en la definición del problema inverso T puede ser lineal o no y los espacios en los que está definida $T\colon X \to Y$, aunque no se especifican, deben ser al menos espacios topológicos. En la práctica se supone que son espacios de Banach o de Hilbert; pero es importante especificar las normas en X y Y, ya que el mal condicionamiento depende crucialmente de las normas utilizadas para definir continuidad, y de existencia y unicidad de soluciones.

La condición del item (3) es muy importante para las aplicaciones, pues en la práctica se conoce el dato $\hat{y}$ con cierto grado de incertidumbre experimental, de medición o por contaminación por ruido. Podemos escribir entonces el problema práctico como

$$y = T(x) + \eta$$

donde η representa una variable aleatoria que modela este grado de incertidubmbre. Si queremos usar métodos determinísticos para abordar este problema, entonces pensamos en una reliazción de η como una variación del valor exacto de y y por lo tanto nos preguntamos por la dependencia de la solución respecto del dato. En el caso concreto de procesamiento de imágenes, hemos desarrollado ya una herramienta no determinística para abordar este problema: el filtro de Wienner. Este método será abordado en una sección posterior de este trabajo.

En el caso lineal y dimensión finita un problema inverso (sin ruido) se puede escribir

$$y = Ax + \eta \tag{4.2}$$

donde $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$, $y \in \mathbb{R}^m$ y se puede estudiar si el problema es mal planteado o no, en términos del los valores singulares de la matriz A (véase el Apéndice 1). En nuestro caso, la ecuación de convolución (4.1) no está escrito en la forma estándar (4.2), pero se puede llevar a esa forma

5. Deconvolución con Matlab

A continuación veremos la manera como Matlab implementa la deconvolución mediante el filtro de Wiener usando la instrucción `deconvwnr`.

Como hemos visto, es posible desborrar una imagen usando el filtro de Wiener, el cual puede ser usado de manera efectiva en el caso de que la frecuencia característica de la imagen y del ruido son conocidas o estimadas. En ausencia de ruido, el filtro de Wiener es simplemente el filtro inverso ideal.

Para ilustrar el uso de la deconvolución usando el filtro de Wienner, trabajeremos con un ejemplo sintético, tomaremos una imagen estádar, simularemos el efecto de borrado debido a un defecto del dispositivo mediante diversas PSFs (Point Spread Function) que vienen incluidas en Matlab, y la función `imfilter`.

```
I = imread('mmrauto.jpg','jpg');
figure;imshow(I);title('imagen original');
```

A continuación se crea una PSF que simula el deffecto en la imagen debida al movimiento súbito de una cámara

```
LEN = 31;THETA = 11;PSF = fspecial('motion',LEN,THETA);
```

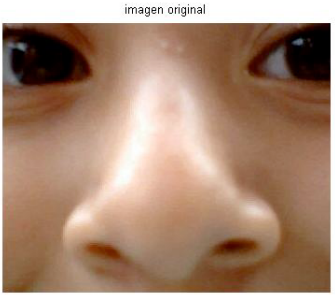


FIGURA 5.1. Imagen original para probar el filtro de Wienner

para especificar esta PSF es necesario dar la longitud del movimiento de la cámara en pixeles (LEN=31) y del ángulo del desplazamiento en grados (THETA=11).

Ya con esta PSF, se procede a usar el comando `imfilter` para calcular la convolución de la PSF con la imagen original

```
imagen_borrosa = imfilter(I,PSF,'circular','conv');
figure; imshow(imagen_borrosa);title('imagen borrosa');
```

Finalmente procedemos al desborrado de la imagen y su presentación, usando los siguientes comandos.

```
Imagen_desborrada = deconvwnr(imagen_borrosa,PSF);
figure;imshow(Imagen_desborrada);
title('imagen restaurada usando PSF')
```

FIGURA 5.2. Imagen borrosa debido a un desplazamiento súbito de la cámara

FIGURA 5.3. Recuperación de la imagen mediante el filtro de Wienner

Es posible mejorar los resultados al usar deconvolución por el método de Wiener, simplemente haciendo uso de los parámetros opcionales del comando `deconwnr` y dar valores específicos a la razón entre la señal y el ruido, que ayude a un refinamiento del desborrado.

6. VALORES SINGULARES Y COMPRESIÓN DE IMAGENES

Uno de los usos de la descomposición en valores singulares es la compresión de imágenes.

A continuación se muestra el m--archivo de Matlab que usamos para comprimir la imagen de Lena, por medio de los valores singulares de esta matriz, este archivo también se usa para obtener diversas imágenes intermedias-

```
close all
[A,map]=imread('lena.gif');
B=im2double(A,'indexed');
imshow(B,map)
[u,s,v]=svd(B);
C=zeros(size(B));
for j=1:5
    C=C+s(j,j)*u(:,j)*v(:,j).';
end
C=floor(C);
k=find(C<1); C(k)=1;
Error=abs(B-C);
D=255-255/max(max(Error))*Error; E=256-D;
subplot(2,2,1), imshow(B,map)
title('Imagen original')
```


```
subplot(2,2,2), imshow(C,map)
title('Imagen con 5 valores singulares')
subplot(2,2,3), imshow(D,map)
title('La diferencia entre las im\'agenes')
subplot(2,2,4), imshow(E,map)
title('Negativo de la diferencia')
set(gcf,'Unit','inches','Paperposition',[0,0,2,1])
```

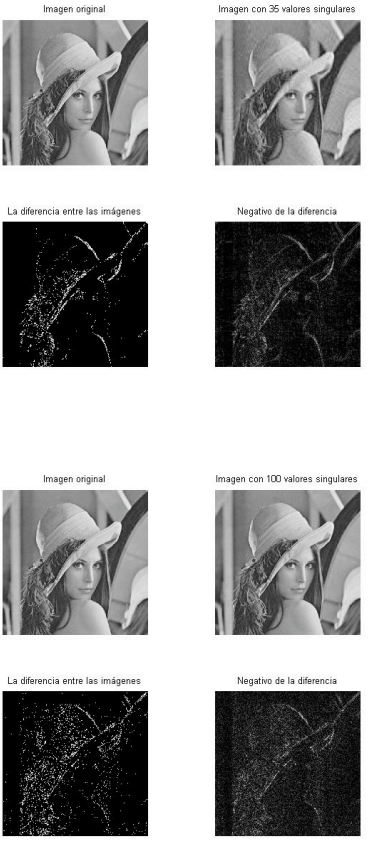
Las siguientes imágenes muestran como, al usar más valores singulares de la matriz asociada a la imagen de Lena, se obtienen mejores aproximaciones de la imagen original.

Primero mostramos las imágenes que se obtienen al usar los primeros cinco valores singulares de la matriz asociada a la imagen Lena. Posteriormente usaremos cada vez una mayor cantidad de valores singulares de la matriz asociada a la imagen de Lena, y veremos como van cambiando las imágenes al aumentar la cantidad de valores singulares tomados en cuenta.

Observemos que la imagen asociada a la aproximación usando los primeros cinco valores singulares es una mala aproximación; la aproximación usando los primeros 60 valores singulares ya es una buena aproximación, mientras que al usar los primeros 100 valores singulares, la aproximación es ya prácticamente, a simple vista, la imagen original.

7. Introducción a los problemas inversos

El problema de deconvolución de una imagen es un ejemplo de problema mal planteado. El filtro de Wienner nos da un método de invertir el operador de convolución (4.1) con un operador de



convolución cuya función se transferencia es precisamente el filtro de Wiener $G(f)$ (3.3). En esta sección vamos a considerar el problema general de "invertir" un operador lineal en dimensión finita que define el problema inverso $y = Ax$, alias, una matriz $A \in \mathbb{R}^{m\times n}$.

Como en general no se puede hablar de la inversa de una matriz rectangular, primeramente (a) es necesario primero generalizar la noción de inversa de una matriz; seguidamente (b)es necesario estudiar cómo varía la solución respecto del dato.

8. Problemas inversos mal planteados

Una matriz rectangular general $A \in \mathbb{R}^{m\times n}$ define un operador lineal $\mathbb{R}^n \to \mathbb{R}^m$. Recordemos que un problema inverso $y = Ax$ se dice mal

planteado, si no existe solución, si no es única o depende de manera discontinua del dato y.

La falta de unicidad se resuelve considerando el operador lineal inducido en espacio cociente $V = R^n/N(A)$, donde $N(A)$ es el núcleo de A. En concreto, si adoptamos el producto interior usual en los espacios $\mathbb{R}^n$, $\mathbb{R}^m$, se sabe que $N(A) \perp Im(A^*)$ y $N(A^*) \perp Im(A)$, donde A^* es el operador adjunto[3], vista como transformación lineal $\mathbb{R}^m \to \mathbb{R}^n$. Además se sabe que $\mathbb{R}^n = N(A) \oplus Im(A^*)$, $\mathbb{R}^m = N(A^*) \oplus Im(A)$ y podemos identificar $\mathbb{R}^n/N(A) = Im(A^*)$. Estaremos considerando entonces $A|_{Im(A^*)}: Im(A^*) \to \mathbb{R}^m$. Finalmente, la existencia se "resuelve" considerando este como un operador $A|_{Im(A^*)}: Im(A^*) \to Im(A)$.

En conclusión, podemos hablar de existencia y unicidad del problema inverso $y = Ax$ sólo para $y \in Im(A)$, y dos soluciones $y = Ax = Ax'$ diferen en una solución del problema homogéneo $x = x + x_0$, $Ax_0 = 0$.

8.1. **La pseudoniversa.**

Definición 8.1. *La pseudoinversa, inversa generalizada o inversa de Moore--Penrose del operador lineal $A: \mathbb{R}^n \to \mathbb{R}^m$ es la única extensión lineal del operador lineal*

$$\left(A|_{Im(A^*)}\right)^{-1} : Im(A) \to Im(A^*)$$

al operador lineal

$$D(A^\dagger) \equiv Im(A) \oplus N(A^*) \xrightarrow{A^\dagger} Im(A^*) \oplus N(A)$$

cuyo núcleo es precisamente $N(A^*)$.

Con ayuda del teorema del teorema de descomposición singular (véas Apéndice 1) podemos dar una expresión expl´ icita de la pseudoinversa.

Proposición 8.2. *Sean*

$$\begin{aligned} U &= [u_1|u_2|\cdots|u_m] \in \mathbb{R}^{m\times m}, \\ V &= [v_1|v_2|\cdots|v_n] \in \mathbb{R}^{n\times n}, \\ S &= diag(s_1, s_2, \ldots, s_p), \quad p = \min(m, n) \end{aligned}$$

la descomposición singular de $A \in \mathbb{R}^{m\times n}$ entonces, si r es el rango de A, la pseudoinversa es

$$A^\dagger = \sum_{i=1}^{r} s_i^{-1} v_i u_i^T \tag{8.1}$$

La expresión (8.1) muestra el origen de un problema mal planteado: *la presencia de valores singulares pequeños produce grandes variaciones*

[3]visto como matriz no es más que la transpuesta

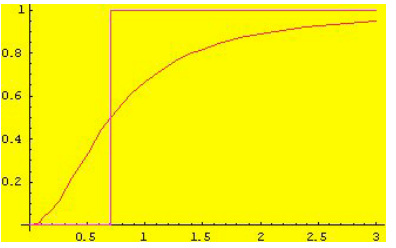


FIGURA 9.1. Funciones de filtro de regularización

en las componentes de solución del problema inverso que se multiplican por sus recíprocos.

El efecto es más notable ante la presencia de ruido, pues si en vez del dato y se considera el dato con ruido $\hat{y} = y + \eta$, $||\eta|| = \delta$, entonces el error al problema inverso es del orden $O\left(\delta s^{-r}\right)$.

9. INTRODUCCIÓN A LOS ESQUEMAS DE REGULARIZACIÓN

Un método de regularización consiste en aproximar la solución altamente inestable de la inversa genearalizada $x = A^{\dagger}y_{\delta}$, donde $y_{\delta} = y + \eta$, $||\eta|| = \delta$, por una familia de operadores R_{α} dependientes de un parámetro, tales que $R_{\alpha}y_{\delta}$ aproxime a $A^{\dagger}y_{\delta}$. Sea entonces

$$x_{\alpha,\delta} = R_{\alpha}y_{\delta}$$

la condición que se exije es que sea posible encontrar, para cada $\delta > 0$ un valor del parámetro $\alpha(\delta)$, tal que

$$||A^{\dagger}y - x_{\alpha(\delta),\delta}|| \to 0, \quad \text{cuando } \delta \to 0. \tag{9.1}$$

Uno de los métodos más utilizado para construir un esquema de regularización, se basa en la descomposición singular y consiste en atenuar los recíprocos de valores singulares pequeños; así en vez de (8.1) se usa el esquema

$$R_{\alpha} = \sum_{i=1}^{r} w_{\alpha}(s_i^2)s_i^{-1}v_i u_i^T \tag{9.2}$$

con la propiedad de la función de filtro

$$\begin{array}{c} \lim_{s\to 0} w_{\alpha}(s^2) = 0, \\ \lim_{s\to\infty} w_{\alpha}(s^2) = 1 \end{array} \tag{9.3}$$

En la Figura se muestran dos posiblidades La función de filtro escalonada es

$$w_{\alpha}(s^2) = \begin{cases} 1 & \text{si } s^2 > \alpha, \\ 0 & \text{si } s^2 \leq \alpha \end{cases};$$

y el esquema de regularización se llama *regularización por truncamiento de valores singulares* (TSVD). La función sigmoidea es

$$w_\alpha(s^2) = \frac{s^2}{s^2+\alpha}.$$

y el esquema de regularización se llama *regularización de Tikhonov.* Ambas fuciones de filtro, satisfacen las condiciones (9.3). La condición (9.1) se satisface tomando α suficientemente pequeña dependiendo de δ.

10. Modelo matemático de borrado

En esta sección presentamos la deducción del modelo de borrado (4.1) a partir de un modelo continuo de borrado[4].

Consideremos dos planos $x'-y'$ correspondiente a la *fuente*, y $x-y$ correspondiente a la imagen. La energía en el punto (x,y) se debe a la contribución de las intensidades de todos los fuentes del la fuente. El efecto de borrado debido al dispositivo se representa por la integral

$$I(x,y) = \int_{\mathbb{R}^2} h(x,x',y,y')f(x',y')\,dx'dy'$$

donde $f(x',y')$ es la intensidad de la fuente en el punto (x',y') e $I(x,y)$ la de la imagen en el punto (x,y). El kernel $h(x,x',y,y')$ se llama la *función de dispersión puntual* (PSF por sus siglas en inglés). Si consideramos la imagen particionada en regiones rectangulares (pixeles) $\Omega_{ij} = [x_i - \Delta x/2, x_i + \Delta x/2] \times [y_i - \Delta y/2, y_i + \Delta y/2]$, entonces la energía total que recibe la región Ω_{ij} será

$$I_{ij} = \int_{\Omega_{ij}} I(x,y)\,dx\,dy \tag{10.1}$$

En modelo estocástico de grabación de datos de un dispositivo CCD, como las cámaras fotográficas digitales que usamos comunmente, la señal grabada en el pixel Ω_{ij} es una variable aleatoria D_{ij} que se distribuye como

$$G_{ij} \sim Poisson(I_{ij}) + Normal(0,\sigma^2). \tag{10.2}$$

La componente de Poisson modela el número de fotones por unidad de tiempo que llegan al pixel, en tanto que el término gaussiano modela el ruido de fondo; ambas se suponen independientes. Denotemos una realización arbiraria del proceso G_{ij} por g_{ij}. El arreglo g de $m \times n$ cuyas componentes son g_{ij} se conoce como la imagen borrada discreta. Si en la realización del proceso (10.2) (podemos pensar que la

[4]Curtis R. vogel, "Computational Methods for Inverse Problems", SIAM Frontiers in Mathematics, 2002

realización corresponde a la captura de la imagen) aplicamos una regla de integración numérica en (10.1), por ejemplo cuadratura por punto medio, tendremos

$$g_{ij} = \sum_{\mu=0}^{m-1}\sum_{\mu=0}^{n-1} h_{i,\mu,j,\nu} f_{\mu,\nu} + \eta_{ij} \tag{10.3}$$

donde

$$f_{\mu\nu} = f(x_\mu, y_\nu)$$
$$h_{i,\mu,j,\nu} = h(x_\mu, x'_\mu, y_\nu, y'_\nu)\,\Delta x\,\Delta y$$

y los errores de cuadratura y ruido gaussiano se incluyen en η_{ij}. Si el defecto de borrado es espacialmente invariante, tendremos

$$g_{ij} = \sum_{\mu=0}^{m-1}\sum_{\mu=0}^{m-1} h_{i-\mu,j-\nu} f_{\mu,\nu} + \eta_{ij} \tag{10.4}$$

que es nuestro modelo discreto de borrado.

10.1. **Modelado de la falta de foco.** El defecto de borrado se modela por diversas PSF, para modelar la falta de foco, se puede usar un kernel gaussiano

$$h(s,t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{s^2+t^2}{2\sigma^2}\right)$$

el kernel discretizado en (10.4) es entonces (C denota la constante $\frac{1}{\sigma\sqrt{2\pi}}$)

$$h_{i-\mu,j-\nu} = C \exp\left(-\frac{(i-\mu)^2}{2\sigma^2}\right) \exp\left(-\frac{(j-\nu)^2}{2\sigma^2}\right) f_{\mu\nu} \tag{10.5}$$
$$= C^{1/2} \exp\left(-\frac{(i-\mu)^2}{2\sigma^2}\right) C^{1/2} \exp\left(-\frac{(j-\nu)^2}{2\sigma^2}\right) \tag{10.6}$$
$$\equiv A_{i,\mu} B j, \nu \tag{10.7}$$

Donde las matrices $A = (A_{i,\mu}) \in \mathbb{R}^{m\times m}$, $B = (B_{i,\mu}) \in \mathbb{R}^{N\times n}$ son simétricas[5], luego nuestro modelo de borrado discreto es

$$g_{ij} = A_{i,\mu} B_{j,\nu} f_{\mu\nu} + \eta_{ij}.$$
$$= A_{i,\mu} f_{\mu\nu} B_{\nu,j} + \eta_{ij},$$

relación que puede escribirse matricialmente como

$$g = AfB + \eta. \tag{10.8}$$

[5]La única diferencia de las matrices A y B aparte de la notación de índices, es su tamaño. Si denotamos por $V(n)$ al vector columna $V(n) = C^{1/2}[1, \exp(-1^2/2\sigma^2), \exp(-2^2/2\sigma^2), \ldots, \exp(-n^2/2\sigma^2)]^T$, entonces $A = V(m)V(m)^T$ y $B = V(n)V(n)^T$. Esta propiedad, llamada de separabilidad del kernel, se explota en el Apéndice 2.

FIGURA 10.1. Imagen original e imagen borrosa

Aunque la relación entre la fuente f, y la imagen g no está escrita en la forma estándar `vector=matrix x vector` para aplicar directamente el teorema de descomposició singular, la relación (10.8) sin ruido, define un operador lineal

$$T\colon \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}, \quad Tf = AfB \tag{10.9}$$

del que podemos estudiar su descomposición singular y la inversa generalizada.

Proposición 10.1. *Sean* $[U_A, S_A, V_A]$, $[U_B, S_B, V_B]$ *las descomposiciones singulares de A y B, respectivamente, entonces la inversa generalizada es*

$$T^{\dagger}g = U_A S_A^{-1} V_A^T \, g \, U_B S_B^{-1} V_B^T \tag{10.10}$$

donde $S_A = diag(s_1^A, s_2^A, \ldots, S_m^A)$, *es la matriz diagonal de valores singulares de A. Sea* $w_\alpha(s^2)$ *cualquiera de las funciones de filtrado por TSVD o Tikhonov, y* $w_\alpha(S_A^2)$ *la matriz diagonal que se obtiene al mapear*[6] w_α *a cada elemento de* $(S_A)^2$*; sean* $w_\alpha(S_B^2)$ *definido de manera análoga, entonces la solución regularizada es*

$$f_{\alpha,\delta} = R_\alpha g^\delta = U_A \, w_\alpha(S_A^2) S_A^{-1} V_A^T \, g \, U_B \, w_\alpha(S_B^2) S_B^{-1} V_B^T.$$

En el apéndice 3 se muestra el código en `Mathematica` del desborrado de imagen usando el teorema anterior.

En la Figura 10.1 se muestra la imagen original y la imagen borrosa, depués de aplicarle el operador de borrado (10.8). A la imagen borrosa se la ha agregado además ruido gaussiano. El dato es entonces la imagen borrosa con ruido de la Figura 10.2 y se intenta recupera la imagen original.

Si intentamos recuperar la imagen original mediante la inversa generalizada (10.10) se obtiene el resultado mostrado en la Figura 10.3.

[6] es decir, aplicar la función a cada elemento del arreglo

FIGURA 10.2. Imagen borrosa con ruido

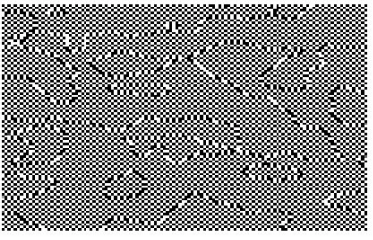


FIGURA 10.3. Recuperación de la imagen original mediante la inversa generalizada

En la secuencia de la Figura 10.4 se muestra la recuperación de la imagen con distintos valores del parámetro de regularización α por el método de Tikhonov Finalmente en la Figura 10.5 se muestra la regularizació óptima y la imagen orignal. Cabe mencionar que en un problema práctico, la solución orignal no se conoce, así que la elección del parámetro es un problema complicado. Existen diversas técnicas para la elección de éste, dependiendo de la información que se disponga de la solución ya sea *a priori* o *a posteriori*. Mencionamos sólo algunos: el método de la curva L y el principio de discrepancia de Morozov.

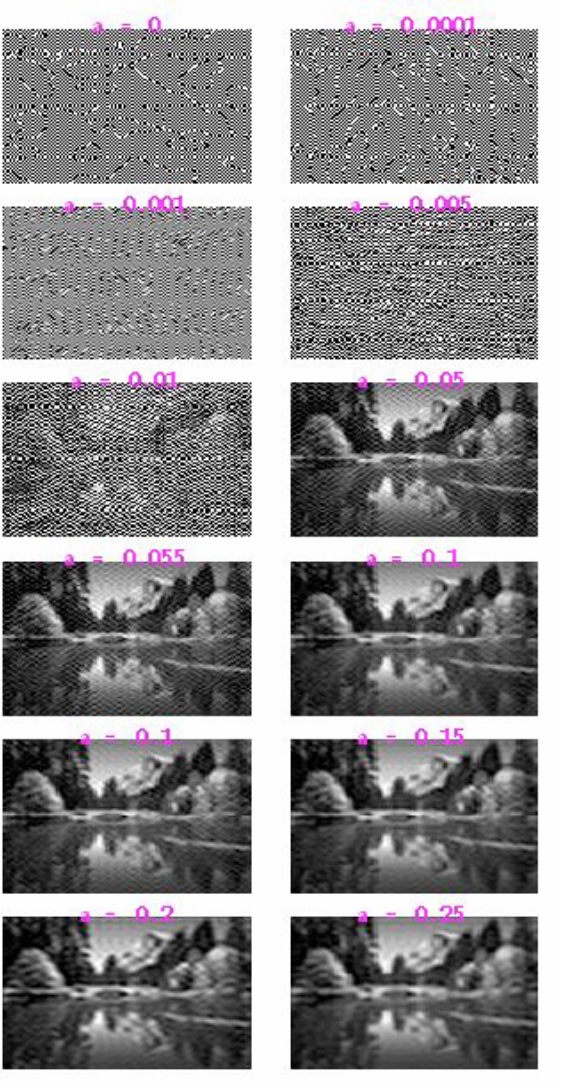


FIGURA 10.4. Reularización de Tikhonov, el parámetro de regularización aumenta de arriba a abajo y de izquierda a derecha

APÉNDICE 1. EL TEOREMA DE DESCOMPOSICIÓN SINGULAR

Teorema 10.2 (Descomposición Singular)**.** *Sea A una matriz real $m \times n$, entonces existen matrices ortogonales $U = [u_1|u_2|\cdots|u_m] \in \mathbb{R}^{m\times m}$,*

$V = [v_1|v_2|\cdots|v_n] \in \mathbb{R}^{n\times n}$ *tales que*

$$U^T AV = diag(s_1, s_2, \ldots, s_p) \equiv S, \quad p = \min(m, n) \tag{10.11}$$

donde $s_1 \geq s_2 \geq \cdots s_p \geq 0$.

Los vectores v_i se llaman vectores singulares derechos y los u_i se llaman vectores propios izquierdos; esto satisfacen

$$\left.\begin{array}{rcl} Av_i &=& s_i u_i, \\ A^T v_i &=& s_i v_i \end{array}\right\} \quad i = 1, 2, \ldots, p.$$

La descomposición $A = USV^T$ es equivalente a la descomposición

$$A = \sum_{i=1}^{p} s_i u_i v_i^T$$

Muchos ambientes como Matlab o Mathematica incluyen rutinas para calcular la descomposición singular. Veamos algunos ejemplos en Matlab.

Una matriz de Hilbert $H = (H_{ij})$ tiene como elemntos las fracciones unitarias

$$H_{ij} = \frac{1}{i + j - 1},$$

por ejemplo, la matriz de 4×4

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}$$

es una matriz de Hilbert. En matlab `hilb(n)` crea la matriz de Hilbert de tamaño n

```
>> A=hilb(4)

A =
    1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
```

El cálculo de sus valores singulares muestra que la mariz de Hilbert es mal condiconada.

```
>> [U,S,V]=svd(A)

U =

   -0.7926    0.5821   -0.1792   -0.0292
   -0.4519   -0.3705    0.7419    0.3287
```

```
   -0.3224   -0.5096   -0.1002   -0.7914
   -0.2522   -0.5140   -0.6383    0.5146


S =

    1.5002         0         0         0
         0    0.1691         0         0
         0         0    0.0067         0
         0         0         0    0.0001


V =

   -0.7926    0.5821   -0.1792   -0.0292
   -0.4519   -0.3705    0.7419    0.3287
   -0.3224   -0.5096   -0.1002   -0.7914
   -0.2522   -0.5140   -0.6383    0.5146
```

Veamos un ejemplo de una matriz rectangular

```
>> A=[1 2 3 4; 5 6 7 8]

A =

     1     2     3     4
     5     6     7     8

>> [U,S,V]=svd(A)

U =

   -0.3762   -0.9266
   -0.9266    0.3762


S =

   14.2274         0         0         0
         0    1.2573         0         0


V =
```

```
   -0.3521    0.7590   -0.4001   -0.3741
   -0.4436    0.3212    0.2546    0.7970
   -0.5352   -0.1165    0.6910   -0.4717
   -0.6268   -0.5542   -0.5455    0.0488
```

Transponiendo no altera los valores singulares, pero intercambia los vectores singulares izquierdos y derechos.

```
>> A=A'

A =

     1     5
     2     6
     3     7
     4     8

>> [U,S,V]=svd(A)

U =

   -0.3521   -0.7590   -0.4001   -0.3741
   -0.4436   -0.3212    0.2546    0.7970
   -0.5352    0.1165    0.6910   -0.4717
   -0.6268    0.5542   -0.5455    0.0488


S =

   14.2274         0
         0    1.2573
         0         0
         0         0


V =

   -0.3762    0.9266
   -0.9266   -0.3762
```

Estamos interesados en estudiar las soluciones del sistema lineal $y = Ax$, con dato y, cuando A es muy "cerca" de ser singular. El siguiente resultado va en esta dirección

Teorema 10.3. *Considere la descomposición singular de A como en el teorema anterior. Si $k < r = rango(A)$ y*

$$A_k = \sum_{i=1}^{k} s_i u_i v_i^T$$

entonces

$$\min_{rango(B)=k} ||A - B|| = ||A - A_k|| = s_{k+1}$$

Este teorema nos dice que el valor singular más pequeño es la distancia (en la norma matricial subordinada a la norma euclideana) de A al conjunto de matrices de rango k.

Observe que si el primer valor singular positivo en orden ascendente ocupa el lugar r, entonces $r = \text{rango}(A)$. Como consecuencia de ello y la propiedad de los vectores singulares, los subespacios fundamentales de A están relacionados con los subespacios generados por los vectores singulares como sigue:

$$\begin{aligned} &\text{rango}(A) = r \\ &\ker(A) = \text{gen}\{v_{r+1}, \ldots, v_n\} \\ &\text{im}(A) = \text{gen}\{u_1, \ldots, u_r\} \end{aligned}$$

Más aún se tiene la descomposición de A:

$$A = \sum_{i=1}^{r} s_i u_i v_i^T.$$

En particular, si A es invertible, $r = n$ y la ecuación $y = Ax$ tiene por solución

$$x = A^{-1}y = \sum_{i=1}^{n} \frac{u_i^T y}{s_i} v_i.$$

Esta expansión muestra el efecto de valores singulares pequeños en la transformación lineal asociada al problema inverso $y = Ax$: pequeños cambios en el dato pueden producir grandes cambios en la solución del problema siendo las contribuciones más importantes en el error la correspondiente a s_n^{-1}, $s_{n-1}^{-1}, \ldots$ y así sucesivamente.

Apéndice 2. Convolución discreta y matrices de Toeplitz

En esta sección veremos algunas relaciones entre convolución bidimensional, unidimensional y matrices de Toeplitz.

Una matriz de $m \times n$ se puede arreglar por columnas en un gran vector columna de tamaño mn, como se muestra en el siguiente

diagrama

$$\begin{pmatrix} A = a_{11}^{\textcircled{1}} & a_{11}^{\textcircled{2}} & \cdots & a_{1n}^{\textcircled{n}} \\ a_{21}^{(n+1)} & a_{22}^{(n+2)} & \cdots & a_{2n}^{(2n)} \\ \cdots & & & \\ a_{m1}^{(n(m-1)+1)} & a_{m2}^{(n(m-1)+2)} & \cdots & a_{1n}^{(mn)} \end{pmatrix}$$

Denotemos por $\vec{A}$ el vector columna de mn correspondiente a la matriz A de $m \times n$ rearreglada por renglones como en el ejemplo anterior, entonces

$$\vec{A}_k = A_{ij} \Longleftrightarrow k = n(i-1) + j.$$

Probemos entonces que cualquier convolución bidimensional se convierte en una convolución unidimensional, cuando se rearregla la matriz de convolución.

Proposición 10.4. *Sean $f = (f_{\mu\nu}), g = (g_{ij}) \in \mathbb{R}^{m\times n}$, matrices relacionadas por convolución con kernel $h = (h_{s,t})_{s,t\in\mathbb{Z}}$. Entonces*

$$g_{ij} = \sum_{\mu,\nu=1}^{m,n} h_{i-\mu,j-\nu} f_{\mu\nu} \tag{10.12}$$

es equivalente a

$$\vec{g}_k = \sum_{k'=1}^{mn} \vec{h}_{k-k'} \vec{f}_{k'}$$

Demostración: Sean $k = n(i-1)+j$, $k' = n(\mu-1)+\nu$, entonces

$$k - k' = n(i-\mu) + j - \nu,$$

por lo tanto

$$\vec{h}_{k-k'} = h_{i-\mu,j-\nu}$$

y la suma (10.12) equivale a

$$\vec{g}_k = \sum_{\mu,\nu=1}^{m,n} \vec{h}_{k-k'} \vec{f}_{k'} = \sum_{k'=1}^{mn} \vec{h}_{k-k'} \vec{f}_{k'} \tag{10.13}$$

donde la última igualdad es debida a que la aplicación $(i,j) \mapsto k$ es una biyección cuando $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$ y $k = 1, 2 \ldots, mn$. □

La convolución 1--dimensional (10.13) se puede escribir matricialmente como

$$\begin{pmatrix} \vec{g}_1 \\ \vec{g}_2 \\ \vdots \\ \vec{g}_{mn} \end{pmatrix} = \begin{pmatrix} \vec{h}_0 & \vec{h}_{-1} & \vec{h}_{-2} & \cdots & \vec{h}_{1-mn} \\ \vec{h}_1 & \vec{h}_0 & \vec{h}_{-1} & \cdots & \vec{h}_{2-mn} \\ \vec{h}_2 & \vec{h}_1 & \vec{h}_0 & \cdots & \vec{h}_{2-mn} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ h_{mn-1} & \vec{h}_{mn-2} & \cdots & \cdots & h_{-1} \\ \vec{h}_{mn} & \vec{h}_{mn-1} & \vec{h}_{mn-2} & \cdots & \vec{h}_0 \end{pmatrix} \begin{pmatrix} \vec{f}_1 \\ \vec{f}_2 \\ \vdots \\ \vec{f}_{mn} \end{pmatrix}.$$

Se puede observar que la matriz es constante a lo largo de las diagonales descendentes de izquierda a derecha. Tales matrices se llaman de *Toeplitz*

Definición 10.5. *Una matrix $T = (T_{s,t})$ se dice de Toeplitz, si $T_{s,t} = t_{s-t}$*

Hemos probado entonces que

Proposición 10.6. *La convolución discreta bidimensional (10.12) se puede escribir en la forma estándar $\vec{g} = T\vec{f}$ donde T es una matriz de Toeplitz.*

Apéndice 3. El kernel gaussiano

El defecto de borrado se modela por diversas PSF, para modelar la falta de foco, se puede usar un kernel gaussiano

$$h(s,t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{s^2+t^2}{2\sigma^2}\right)$$

el kernel discretizado en (10.4) es entonces (C denota la constante $\frac{1}{\sigma\sqrt{2\pi}}$)

$$\begin{aligned} h_{i-\mu,j-\nu} &= C\exp\left(-\frac{(i-\mu)^2}{2\sigma^2}\right)\exp\left(-\frac{(j-\nu)^2}{2\sigma^2}\right) f_{\mu\nu} \\ &= C^{1/2}\exp\left(-\frac{(i-\mu)^2}{2\sigma^2}\right) C^{1/2}\exp\left(-\frac{(j-\nu)^2}{2\sigma^2}\right) \\ &= a_{i-\mu}a_{j-\nu} \end{aligned}$$

o en forma matricial,

$$h = a(m) \otimes a(n) \tag{10.14}$$

donde $a(n)$ denota el vector columna

$$a(n) = C^{1/2}\left[1, \exp(-1^2/2\sigma^2), \exp(-2^2/2\sigma^2), \ldots, \exp(-n^2/2\sigma^2)\right]^T,$$

y el producto directo general de dos vectores columna $a = [a_1, a_2, \ldots, a_m]^T$, $b = [b_1, b_2, \ldots, b_n]^T$ se define como $a \otimes b = ab^T$ o sea,

$$a \otimes b = \begin{pmatrix} a_1b_1 & a_1b_2 & \cdots & a_1b_n \\ a_2b_1 & a_2b_2 & \cdots & a_2b_n \\ \vdots & \vdots & \cdots & \vdots \\ a_mb_1 & a_mb_2 & \cdots & a_mb_n \end{pmatrix}.$$

Para nuestro caso (10.14) se escribe

$$h = \begin{pmatrix} a_1a_1 & a_1a_2 & \cdots & a_1a_n \\ a_2a_1 & a_2a_2 & \cdots & a_2a_n \\ \vdots & \vdots & \cdots & \vdots \\ a_ma_1 & a_ma_2 & \cdots & a_ma_n \end{pmatrix}$$

donde

$$a = C^{1/2}\left[1, \exp(-1^2/2\sigma^2), \exp(-2^2/2\sigma^2), \ldots, \right]^T,$$

APÉNDICE 3. DESABORRADO DE IMAGENES EN MATHEMATICA

```
(* Desborrado de imagenes bidimensionales. Ejemplo
sintetico. El borrado de la imagen bidimensional descrito
por la funcion f[x,y] se modela por la discretizacion de la
integral de convolucion *)
```

$$g(x,y) = \int_0^1 \int_0^1 k(x-x', y-y') f(x', y') dx' dy'$$

```
(*con kernel Gaussiano bidimensional
k(x,y)=C exp(-(x^2+y^2)/2 g^2). La ecuacion se discretiza
usando cuadratura de punto medio.
Los nodos son i h, i=0,1,...,n donde h=1/n y los puntos
medios son ti=(i+1/2) h. La matriz de la discretizacion del
problema directo es Kij = h Ce Exp[-((i-j) h)^2/(2 g^2)] *)

In[1]:=
(*Nos situamos en el directorio donde tenemos la figura*)
SetDirectory["C:\Documents and Settings\Joaquin\
   Mis documentos\INVERSE"]

g=Import["monito.jpg"];

In[3]:=
Dimensions[g]

In[4]:=
{g[[2]]}
(*En Mathematica una imagen en formato jpf es una lista
de cuatro elementos,  cuyo primer elemento,g[[1]] es un
objeto RasterGraphics,  g[[1,1]] es la matriz de pixeles
en escalas de grises; el resto de los elementos de la
lista dan informacion adicional de la imagen*)

d=g[[1,1]];

In[6]:=
Dimensions[d]

In[7]:=
(*Convertimos a escala entre 0 y 1*)
```

```
d=N[d/Max[d]];
{m,n}=Dimensions[d]
 (*m es la altura,n es el ancho de imagen*)
Show[Graphics[Raster[d]]];
(*Kernel gaussiano*)
Plot3D[Exp[-x^2/(2 g^2)] Exp[-y^2/(2 g^2)]/(g^2 2
Pi)/.g->.0125,{x,-.1,.1},{y,-.1,.1},PlotPoints->50,
    PlotRange->All];

(*Verifiquemos que el kernel es una densidad de
    probabilidad*)
NIntegrate[Exp[-x^2/(2 g^2)]
   Exp[-y^2/(2 g^2)]/(g^2 2 Pi)/.g->.0125,
   {x,-1,1},{y,-1,1}]

In[12]:=
(*El operador de borrado;el parametro g controla
el desenfoque; ademas de la imagen borrada y,
regresa  las matrices A, B del operador de
  borrado*)

Blurr[g_,x_]:=
Block[{h=1/m,k=1/n,Ce=1/(g^2 2 Pi),A,B,y},
                {m,n}=Dimensions[x];
  For[i=1,i\[LessEqual]  m,i++,
    For[ii=1,ii\[LessEqual]  m,ii++,
       a[i_,ii_]:=h Ce^(1/2)
       Exp[-((i-ii) h)^2/(2 g^2)];];];
  For[j=1,j\[LessEqual]  n,j++,
   For[jj=1,jj\[LessEqual]  n,jj++,
    b[j_,jj_]:=
      k Ce^(1/2) Exp[-((j-jj) k)^2/(2 g^2)];
          ];
          ];
    A=Table[a[i,ii],{i,1,m},{ii,1,m}];
    B=Table[b[j,jj],{j,1,n},{jj,1,n}];
    y=A.x.B;
    Return[{A,B,x,y}];]

In[13]:=
(*Aplicamos el operador de borrado*)
 {A,B,x,y}=Blurr[.0125,d];
```

```
In[14]:=
Show[Graphics[Raster[x]]];
Show[Graphics[Raster[y]]];

In[16]:=
(*Version con ruido contaminada de y: *)
delta=.005;
noise=delta Table[Random[Real,{-1,1}],
    {i,1,m},{j,1,n}];
ynoisy=y+noise;
Show[Graphics[Raster[ynoisy]]];


In[19]:=
(*Descomposicion singular*)
svdA=SingularValues[A];
svdB=SingularValues[B];
{UA,SA,VA}=svdA;
{UB,SB,VB}=svdB;
(*Veamos las dimensiones*)
{Dimensions[A],Dimensions[UA],Dimensions[SA],
Dimensions[VA]}
{Dimensions[B],Dimensions[UB],Dimensions[SB],
Dimensions[VB]}

In[24]:=
(*Chequemos que la descomposicion es correcta hasta
la precision que da Chop*)
A-Transpose[UA].DiagonalMatrix[SA].VA//Chop
B-Transpose[UB].DiagonalMatrix[SB].VB//Chop

In[26]:=
ListPlot[Flatten[SA],PlotRange->All,
PlotLabel->"Valores singulares de A",AxesOrigin->{0,0}];
ListPlot[Flatten[SB],PlotRange->All,
PlotLabel->"Valores singulares de B",AxesOrigin->{0,0}];

In[28]:=

(*una prueba de lo que hace el comando Outer*)
list1={1,2,3};list2={a,b,c,h};
Outer[Times,list1,list2]//MatrixForm
```

```
In[30]:=
(*Los vectores singulares son matrices*)
 u[i_,j_]:=Outer[Times,Transpose[VA][[i]],UB[[j]]];

(*.. y una dimension tipica*)
Dimensions[u[1,2]]

In[32]:=
(*Graficamos algunos vectores singulares*)
ListPlot3D[u[3,4]];
ListPlot3D[u[10,4]];

In[34]:=
(*Inversa generalizada (Inv Gen)=inversa,en este caso,
  aunque el determinante es proximo de cero*)
Det[A]

In[35]:=
(*Invertimos las matrices diagonales directamente*)
SAinv=1/SA;ListPlot[SAinv];
SBinv=1/SB;ListPlot[SBinv];

In[37]:=
(*Solucion con la Inv Gen para el dato sin ruido*)
 xinv=Transpose[VA].DiagonalMatrix[SAinv].UA.y.
 Transpose[VB].DiagonalMatrix[SBinv].UB;

(*Imagen orignal y solucion con la Inv Gen para
 dato sin ruido*)
Show[Graphics[Raster[x]],PlotLabel-> "Imagen original"];
Show[Graphics[Raster[xinv],PlotLabel-> "Imagen recuperada
 con Inv Gen"]];

In[39]:=
(*Solucion con la Inv Gen para dato  con ruido*)
 xnoisy=Transpose[VA].DiagonalMatrix[SAinv].UA.ynoisy.
 Transpose[VB].DiagonalMatrix[SBinv].UB;
Show[Graphics[Raster[x],PlotLabel-> "Imagen original"]];
Show[Graphics[Raster[xnoisy],PlotLabel-> "Imagen
  recuperada con Inv Gen"]];

In[41]:=
```

```
(*Regularizaci\'on por truncamiento*)
alpha=.1;
R[s_]:=If[s>=alpha,s^(-1),0];
(*Mapeamos R sobre la matriz de valores singulares*)
SAinva=Map[R,SA,1];
SBinva=Map[R,SB,1];
ListPlot[Flatten[SAinva],PlotStyle->{RGBColor[1,0,0],
PointSize[.01]},PlotLabel-> "Truncamiento de
    reciprocos \n   de valores singulares de SA"];

ListPlot[Flatten[SBinva],PlotStyle->{RGBColor[1,0,0],
 PointSize[.01]},PlotLabel->"Truncamiento de
 reciprocos \n de valores singulares de SB"];

In[47]:=
(*Solucion regularizada por TSVD*)
  xnoisya=Transpose[VA].DiagonalMatrix[SAinva].UA.ynoisy.
  Transpose[VB].DiagonalMatrix[SBinva].UB;
Show[Graphics[Raster[xnoisya]],PlotLabel-> "Una solucion
  regularizada por TSDV"];


In[49]:=
(*Definamos la regularizacion por truncamiento como
funcion del parametro*)

Clear[a,s,R,SAinva,SBinva];

In[50]:=
Trunc[alpha_]:=
  Block[{R,SAinva,SBinva},R[s_]:=If[s>=alpha,s^(-1),0];
    SAinva:=Map[R,SA,1];
    SBinva:=Map[R,SB,1];
    xnoisya=
      Transpose[VA].DiagonalMatrix[SAinva].UA.ynoisy.
      Transpose[VB].DiagonalMatrix[SBinva].UB;
    Return[Graphics[{Raster[xnoisya],RGBColor[1,1,1],
     Text["alpha = "<>ToString[alpha],Scaled[{.5,.1}]]},
        TextStyle->{FontWeight->"Bold",FontSize->6}]];
    ]

In[51]:=
(*Despliegue*)
```

```
Show[GraphicsArray[{
        {Trunc[0],Trunc[.0001],Trunc[.001]},
        {Trunc[.005],Trunc[.01],Trunc[.05]},
        {Trunc[.055],Trunc[.1],Trunc[.1]},
        {Trunc[.15],Trunc[.2],Trunc[.25]}}]];

In[52]:=
(*Error en la norma L2*)
TSDVTruncErrorL2[alpha_]:=
  Block[{R,SAinva,SBinva,error,dx},R[s_]:=
       If[s>=alpha,s^(-1),0];
    SAinva:=Map[R,SA,1];
    SBinva:=Map[R,SB,1];
    xnoisya=
      Transpose[VA].DiagonalMatrix[SAinva].UA.ynoisy.
      Transpose[VB].DiagonalMatrix[SBinva].UB;
    dx=Flatten[x-xnoisya];
    error=Sqrt[Tr[dx^2]];
    Return[error];]



In[53]:=
TSDVerrorlistL2=Table[{da,TSDVTruncErrorL2[da]},
   {da,.02,.3,.005}];
ListPlot[TSDVerrorlistL2,PlotJoined->True,
   PlotRange->All,PlotLabel-> "Error L2 vs
   alpha en TSDV"];


In[55]:=
(*Calculemos el minimo*)
TSDVelistL2=Map[Part[#,2]&,TSDVerrorlistL2];
TSDVminerror=Min[TSDVelistL2]

In[57]:=
 Position[TSDVelistL2,TSDVminerror]

In[58]:=
 istar=Part[First[Position[TSDVelistL2,
                     TSDVminerror]],1]

In[59]:=
```

```
TSDVerrorlistL2[[istar]]

In[60]:=
alphastar=Part[TSDVerrorlistL2[[istar]],1];

In[61]:=
Show[GraphicsArray[{Graphics[Raster[x]],
      Trunc[alphastar]}]];


In[62]:=

(*Regularizacion de Tikhonov*)
Tikhonov[alpha_]:=Block[{R,SAinva,SBinva},
 w[s2_]:=s2/(s2+alpha);
 SAinva=Table[w[SA[[i]]^2]/SA[[i]],{i,1,Length[SA]}];
 SBinva=Table[w[SB[[i]]^2]/SB[[i]],{i,1,Length[SB]}];
 xnoisya=
  Transpose[VA].DiagonalMatrix[SAinva].UA.ynoisy.
  Transpose[VB].DiagonalMatrix[SBinva].UB;
  Return[Graphics[{Raster[xnoisya],RGBColor[1,1,1],
    Text["alpha = "<>ToString[alpha],Scaled[{.5,.1}]]},
    TextStyle->{FontWeight->"Bold",FontSize->6}]];
    ]

In[63]:=
 Show[GraphicsArray[{{Tikhonov[0],Tikhonov[.0001],
     Tikhonov[.001]},{Tikhonov[.005],Tikhonov[.01],
     Tikhonov[.05]},{Tikhonov[.055],Tikhonov[.1],
     Tikhonov[.1]},{Tikhonov[.15],Tikhonov[.2],
     Tikhonov[.25]}}]];

In[64]:=
(*Calculo del minimo error para la norma L2*)

Clear[a,s,R,SAinva,SBinva];

TikTruncErrorL2[a_]:=Block[{w,SAinva,SBinva,error},
 w[s2_]:=s2/(s2+a);
 SAinva=Table[w[SA[[i]]^2]/SA[[i]],{i,1,Length[SA]}];
 SBinva=Table[w[SB[[i]]^2]/SB[[i]],{i,1,Length[SB]}];
 xnoisya=
  Transpose[VA].DiagonalMatrix[SAinva].UA.ynoisy.
```

```
  Transpose[VB].DiagonalMatrix[SBinva].UB;
  dx=Flatten[x-xnoisya];
  error=Sqrt[Tr[dx^2]];
  Return[error];
    ]

In[66]:=
 TikerrorlistL2=
    Table[{dalpha,TikTruncErrorL2[dalpha]},
      {dalpha,.001,.03,.001}];

ListPlot[TikerrorlistL2,PlotJoined->True,
  PlotRange->All,PlotLabel-> "Error L2 vs.
    alpha en Tikhonov"];
TikelistL2=Map[Part[#,2]&,TikerrorlistL2];

In[69]:=
minerror=Min[TikelistL2]

In[70]:=
{istar}=Position[TikelistL2,minerror]//Flatten

In[71]:=
TikerrorlistL2[[istar]]

In[72]:=
alphastar=Part[TikerrorlistL2[[istar]] ,1]

In[73]:=
(* Solucion optima *)
Show[GraphicsArray[{Graphics[Raster[x]],
 Tikhonov[alphastar]}]];
```

FIGURA 10.5. Imagen original e imagen recuperada