

Práctica 1: Imágenes en MATLAB		Grupo	
		Puesto	
Apellidos, nombre	Sánchez Garzón, Laura	Fecha	
Apellidos, nombre	Remuñán Cid, Sara		

El objetivo de esta práctica es presentar al alumno las herramientas que ofrece MATLAB para la representación y manejo de imágenes.

Desarrolle cada ejercicio en un fichero de comandos ‘ejercicio_X.m’ separado y anote las observaciones que se le pidan en un documento aparte. Para conocer el funcionamiento preciso de los comandos que se introducen en este guión, utilice la ayuda de MATLAB. Para evitar posibles interferencias con otras variables o ventanas recuerde incluir siempre las instrucciones `clear all` y `close all` al principio de cada fichero de comandos.

Al finalizar la práctica, comprima el documento con las observaciones y los ficheros ‘.m’ generados en un único fichero con el nombre ‘FTDI_P1_ApellidosNombre1_ApellidosNombre2.zip’, conéctese al sistema de entrega de prácticas de Moodle y entréguelo.

1 Obtención y grabación de imágenes

1.1 Ejercicio 1: generación y representación de imágenes a partir de expresiones analíticas.

Para definir en MatLab una función escalar discreta, $\psi[n, m]$, en una región rectangular de un espacio bidimensional (2D), primero es necesario definir **dos** matrices con los valores que toma cada una de las dos variables (n, m) en cada punto de dicha región. Para ello:

- Se definen dos vectores, **m** y **n**, con los rangos de variación de las correspondientes variables que delimitan la región en que se va a definir la función escalar.
- Se utiliza la expresión `[N,M]=meshgrid(n,m)`, que genera en **N** y **M** las dos matrices indicadas.
- Se escribe la expresión de la función deseada, utilizando **N** en el lugar de **n** y **M** en el lugar de **m**.

Ejemplo:

Generar una versión discreta $\psi[n, m]$ de la función continua $\psi(x, y) = \cos(2\pi x) + \sin(6\pi y)$ en la región $0 \leq x \leq 4$, $0 \leq y \leq 4$, muestreándola con un retículo ortogonal de vectores $\vec{v}_1 = (0.05, 0)$, $\vec{v}_2 = (0, 0.05)$.

```
>> n=[0:0.05:4]; m=[0:0.05:4]; % Definición de los rangos de variación de las variables
>> [N,M]=meshgrid(n,m); % Generación de las matrices N y M
>> f=cos(2*pi*N)+sin(6*pi*M); % Definición de la función: x->N, y->M
```

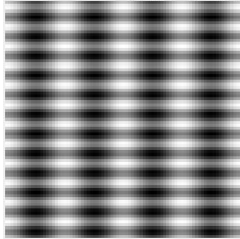
Visualice en el *workspace* de MatLab el contenido de las matrices **M** y **N** y el de la función **f**, para entender el modo en que MatLab genera funciones 2D.

Un modo de representar una función 2D es crear una imagen con tonos grises, en la que la posición de cada elemento o píxel representa una coordenada del espacio 2D, y el valor o nivel de gris de cada píxel representa el valor de la función en dicha coordenada. Una imagen es, desde este punto de vista, una matriz de valores.

Para representar una matriz de valores en forma de imagen se utiliza la función `imshow`, indicando la función a representar y los valores de la función a que corresponden el negro (habitualmente el mínimo de la función) y el blanco (habitualmente el máximo de la función).

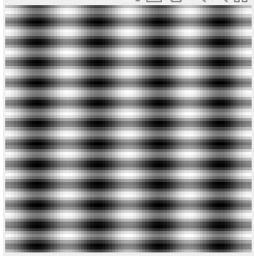
Ejemplo:

```
>> imshow(f, [-2 2], 'InitialMagnification', 100); % Muestra la función f como una imagen.
```



Observe que la función f toma como mínimo el valor -2 (que se muestra negro en la imagen) y como máximo el valor 2 (que se muestra blanco). Una redacción alternativa y genérica de este comando podría haber sido:

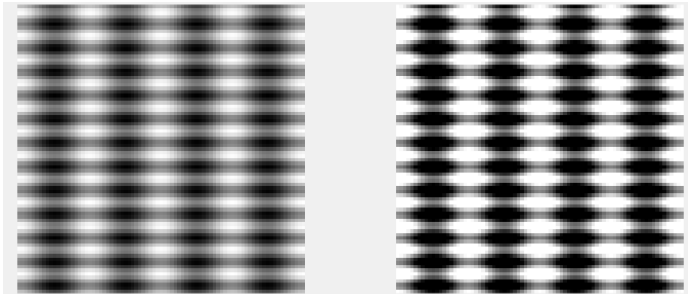
```
>> imshow(f, [min(min(f)) max(max(f))], 'InitialMagnification', 100);
```



El parámetro `'InitialMagnification'` fijado al valor 100 fuerza a que la imagen se presente a tamaño real (cada píxel de la imagen corresponde con un punto del monitor), siempre que el tamaño del monitor lo permita. Este es el único modo de garantizar que la imagen se presenta con la máxima precisión.

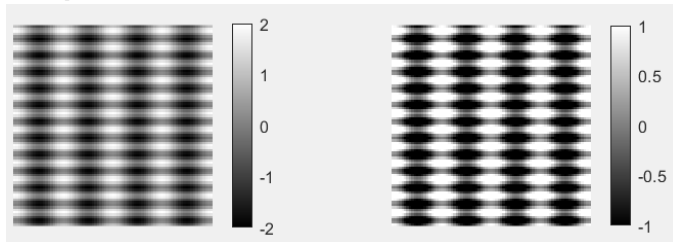
Para representar varias imágenes en una misma ventana utilice el comando `subplot` en combinación con el comando `imshow`. Ejemplo:

```
>> subplot(1,2,1), imshow(f, [-2 2]);  
>> subplot(1,2,2), imshow(f, [-1 1]);
```



Observe que en este caso no es posible controlar que la imagen se presente a tamaño real (por lo que el parámetro 'InitialMagnification' no es de utilidad). Explique por qué motivo las dos imágenes se ven distintas, para ello quizás le resulte de ayuda el incluir el comando colorbar tras la representación de cada imagen

```
>> subplot(1,2,1), imshow(f, [-2 2]); colorbar
>> subplot(1,2,2), imshow(f, [-1 1]); colorbar
```



Observaciones 1.1.1:

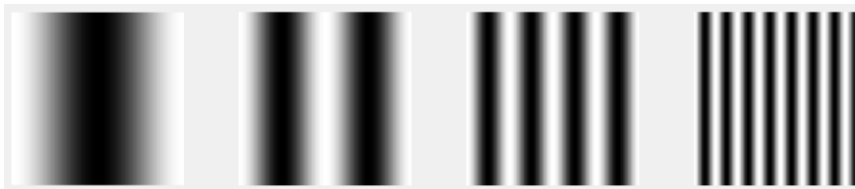
A mayor rango, menor salto en la escala de grises, por lo que se distinguen más colores. Si el rango es menor como en la segunda imagen, el salto es más pronunciado, y por tanto, la escala se ve más nítida.

Siguiendo esta aproximación, genere imágenes de las versiones discretas de las siguientes funciones; para ello defina las funciones en la región $0 \leq x < 1, 0 \leq y < 1$ (observe que los intervalos están abiertos en su extremo superior), muestreando con un retículo ortogonal de vectores $\vec{v}_1 = (1/256, 0), \vec{v}_2 = (0, 1/256)$ (de ahora en adelante, utilice este mismo retículo siempre que se le pida generar imágenes en el resto de la práctica.). Dibuje cada grupo de funciones en una figura distinta.

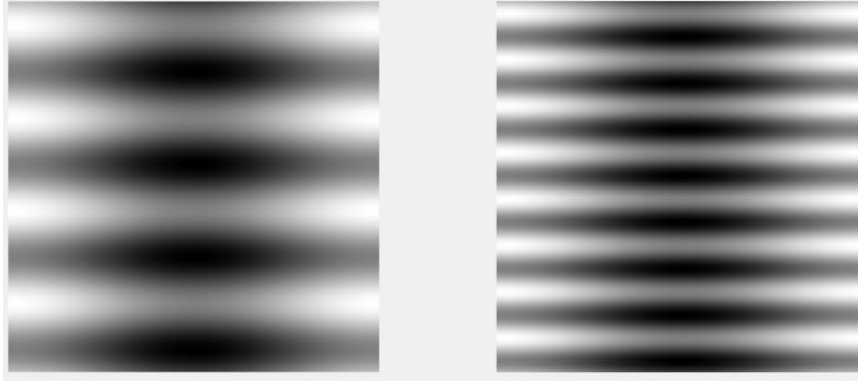
$$1. \quad \psi_1(x, y) = \frac{4x}{5}, \quad \psi_2(x, y) = \frac{y}{2}, \quad \psi_3(x, y) = \frac{2x}{5} + \frac{y}{4}$$



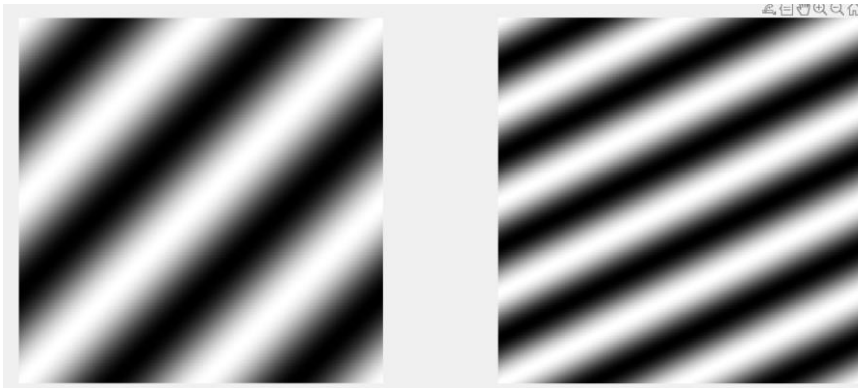
$$2. \quad \psi_1(x, y) = \cos(2\pi x), \quad \psi_2(x, y) = \cos(4\pi x), \quad \psi_3(x, y) = \cos(8\pi x), \quad \psi_4(x, y) = \cos(16\pi x)$$



3. $\psi_1(x, y) = \cos(2\pi x) + \sin(8\pi y)$, $\psi_2(x, y) = \cos(2\pi x) + \sin(16\pi y)$



4. $\psi_1(x, y) = \cos(4\pi x + 4\pi y)$, $\psi_2(x, y) = \cos(4\pi x + 8\pi y)$



En cada uno de los cuatro grupos de funciones, relacione el patrón o imagen que observa con la expresión de la función a que corresponda, indicando por qué cada imagen tiene el aspecto que observa.

Para representar las imágenes utilice el comando `imshow` especificando el valor mínimo y máximo de cada función. Ejemplo:

```
>> imshow(f, [min(min(f)) max(max(f))]);
```

Observaciones 1.1.2:

En el primer grupo, aquellas funciones que sólo depende de x se desplazan en el eje horizontal (cuando x toma valores más bajos, la imagen representada toma valores en la escala de grises más bajos) y viceversa. Se aprecia en la última al mezclar x e y que hay desplazamiento diagonal.

En el segundo grupo, al aplicar senos o cosenos, se obtienen funciones armónicas, valores que suben y bajan periódicamente en la escala de grises. El número de ciclos depende del número por el que estén multiplicados estos senos y cosenos.

En el tercer grupo, se cruzan un seno y un coseno, en los dos ejes, por lo que sale un patrón de cruzamiento entre armónicos.

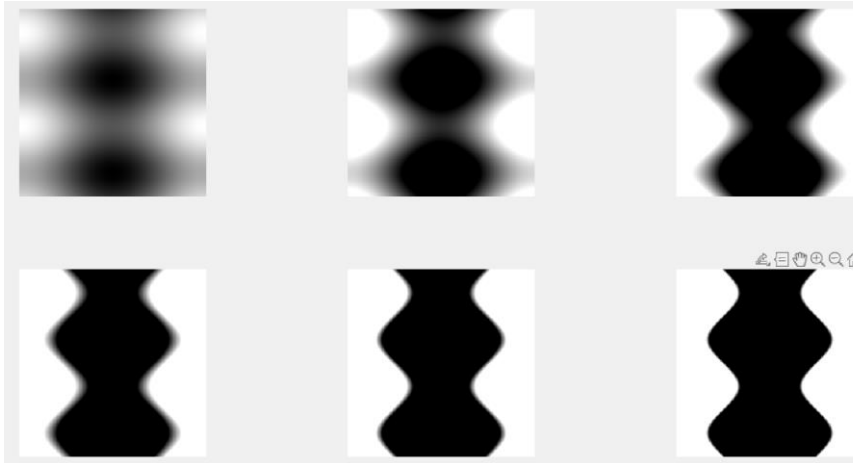
En el cuarto grupo, se aprecia un armónico en diagonal porque combina vectores tanto en x como en y.

1.2 Ejercicio 2: tipos de imágenes en MatLab.

Las imágenes generadas por el procedimiento del ejercicio anterior son de tipo `double`, es decir, sus píxeles pueden tomar cualquier valor real y puede operarse directamente con ellas (sumas, productos, operaciones lógicas, etc.). Para representarlas es necesario indicar a la función `imshow` el rango de valores que toman los píxeles, indicando el valor que corresponde al negro (r_0) y al blanco (r_L). Si no se indica, MatLab asume por defecto que el rango es $r_0 = 0.0$, $r_L = 1.0$.

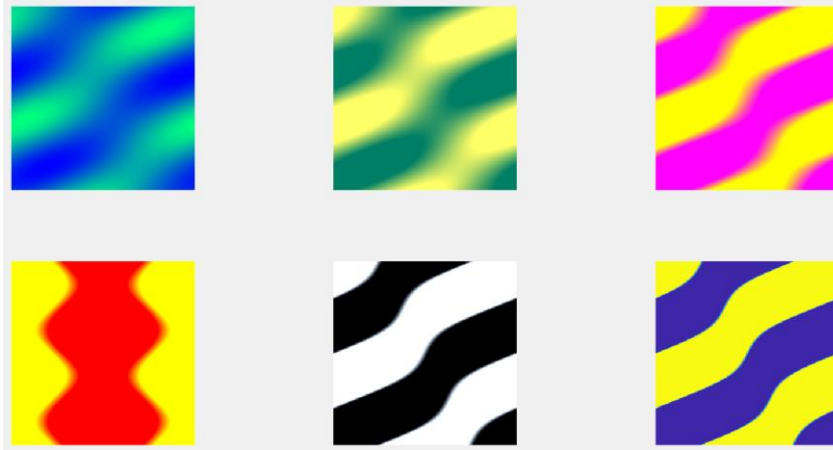
Si se utilizan respectivamente el mínimo valor de la imagen (`min(min(f))`) y el máximo (`max(max(f))`), se garantiza que se aprovecha todo el rango de L niveles representables (rango que suele ser de $L = 256$ niveles distintos), por lo que la imagen se presentará con la máxima definición de niveles. Si no se utiliza este *rango ajustado a la imagen*, los valores de la imagen inferiores a r_0 se presentarán como negros y los superiores a r_L se presentarán como blancos, por lo que se perderá resolución en la representación de los tonos de gris.

Genere la función discreta correspondiente a la función $\psi(x, y) = 32 \cdot \cos(2\pi x) + 16 \cdot \sin(4\pi y)$. Obtenga su mínimo y su máximo. En una misma ventana represente seis imágenes, la primera con el rango óptimo, las cuatro siguientes con rangos sucesivamente inferiores (mitad del anterior para el blanco y mitad del anterior para el negro), y la última sin indicar el rango. Comente la evolución que se observa en las seis imágenes representadas y el motivo de dicha evolución.



Observaciones 1.2:

Cuanto menos rango hay de escala de grises, más nítida se ve la imagen porque mayor es el salto de colores.



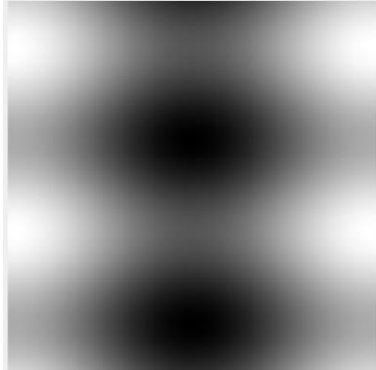
En términos de eficiencia en la presentación de imágenes, una alternativa en general más eficiente consiste en que los píxeles de una imagen no representen directamente niveles de intensidad, sino índices de una tabla o mapa de colores (VLT de *Video Lookup Table*) que contiene la intensidad o color a representar, motivo por el que se denominan *imágenes indexadas*. En este caso los valores de los píxeles han de ser enteros positivos (ya que son índices de una tabla), y la imagen será de tipo `uint8` (un *byte* por píxel, es decir, un rango de $[0, 255]$ posibles niveles o colores) o bien de tipo `uint16` (dos *bytes* por píxel, es decir, un rango de $[0, 65535]$ posibles colores) por lo que requerirá menos recursos para almacenarse y representarse que una imagen de tipo `double`. La desventaja es que con imágenes (en realidad matrices) de tipos enteros, MatLab no permite aplicar directamente casi ninguna operación (es necesario efectuar conversiones de tipos).

MatLab incluye la función `im2uint8` para convertir una imagen de tipo `double` en una de tipo `uint8`, y la `im2uint16` para convertir a imágenes de tipo `uint16`. Estas funciones se encargan de desplazar (para que no haya valores negativos), escalar (para abarcar el rango $[0, 255]$ o $[0, 65535]$) y redondear (para que los valores de los píxeles sean enteros) los rangos de variación de la imagen original; sin embargo, las funciones asumen que la imagen de tipo `double` se encuentra dentro del rango $0 \leq \nu[m, n] \leq 1$. Si esto no se verifica, como en el caso que nos ocupa, es necesario realizar las conversiones mediante comandos. Así, para convertir una imagen `f` de tipo `double` a una imagen `ima` de tipo `uint8`, basta con desplazar, escalar y redondear los valores de tipo `double` de la imagen original (es decir, cuantificarlos en el rango del nuevo tipo):

```
>> min_f=min(min(f)); max_f=max(max(f)); % Obtengo los extremos
>> step_f=(max_f-min_f)/256;           % Intervalo que corresponde a cada nivel
>> ima=uint8(round((f-min_f)/step_f)); % Desplazo y escalo
```

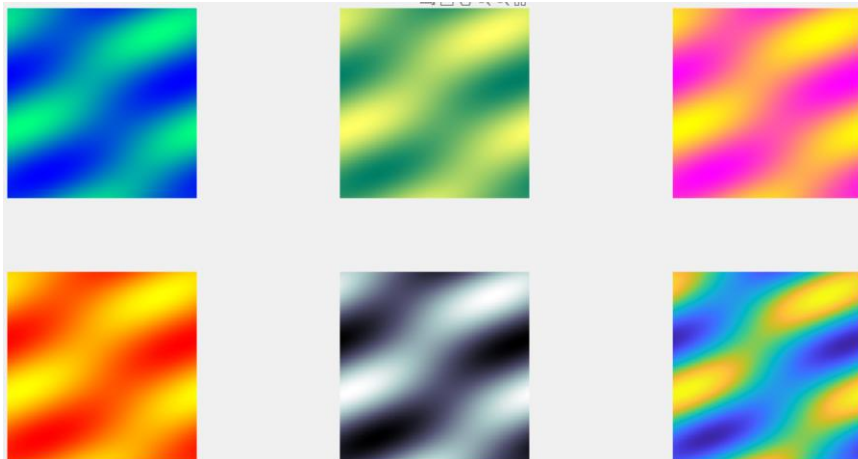
Para representar una imagen de tipo `uint8` o `uint16` es necesario indicar cuál es la VLT (en MatLab se denomina *colormap*) o mapa de colores. Para generar un mapa de colores en escala de grises y utilizarlo para representar una imagen `ima`, debe escribir:

```
>> figure; imshow(ima, gray(256)); % VLT de 256 niveles de gris
```



Genere la función discreta correspondiente a $\psi(x, y) = 32 \cdot \cos(2\pi x + 3\pi y) + 16 \cdot \sin(4\pi y)$, conviértala a una imagen de tipo `uint8` y las seis en una misma ventana utilizando seis mapas de colores distintos a `gray` (escójalos de entre los descritos en la ayuda de la función `colormap()`, utilizando la ayuda de MATLAB con el comando `help`), y siempre con 256 colores distintos.

```
>> subplot(2,3,1), imshow(ima, gray(256));
```



1.3 Ejercicio 3: lectura de imágenes a partir de un archivo.

Para leer una imagen de un archivo utilice la función `imread()`:

```
>> [ima,map]=imread('MRI_pseudo_colored.jpg');
```

Si la imagen almacenada es una imagen indexada, `ima` será una matriz o *array* de **dos** dimensiones de tipo `uint8` o `uint16`, y `map` será su tabla de *colores*, una matriz cuyas filas indican las tres componentes R, G, B del color que representan (si las tres componentes fueran iguales, representarían un nivel de gris).

Si la imagen grabada no es indexada, se trata de una imagen *true-color*, es decir, una imagen en la que el valor de cada píxel indica directamente su color; no es un índice de una VLT por lo que en estas imágenes `map` está vacío.. Una imagen *true-color* puede estar formada por dos o tres bandas o matrices de tipo `uint8` (es decir, un *array* de **dos** o **tres** dimensiones). Si se trata de un *array* de dos dimensiones (es decir, que a cada elemento de la imagen le corresponde un valor), el valor de los píxeles es directamente

un nivel de luminancia o nivel de gris que va de 0 (negro) a 255 (blanco); por lo tanto, no incluye información de color. Si se trata de un *array* de tres dimensiones, a cada elemento de la imagen le corresponden tres valores enteros que indican directamente las componentes roja, verde y azul de cada píxel, cada una de ellas variable entre 0 (componente inexistente) y 255 (máxima saturación en esa componente). Dentro de la categoría *true-color*, las imágenes binarias son un tipo especial que solamente presenta dos valores; estas imágenes están formadas por una sola banda (es decir, un *array* de **una** dimensión) de tipo `uint8` (con valores 0 o 255), o `logical` (con valores 0 o 1).









Lea y represente las imágenes¹ `MRI_pseudo_colored.jpg`, `CT_abdomen.jpg`, `Skin.tif` y `Xray_th.tif`. Indique en la tabla adjunta en qué formato están:

Comentado [MEV1]: Ojo que aquí en el código usas `subimage` que no se lo has explicado...

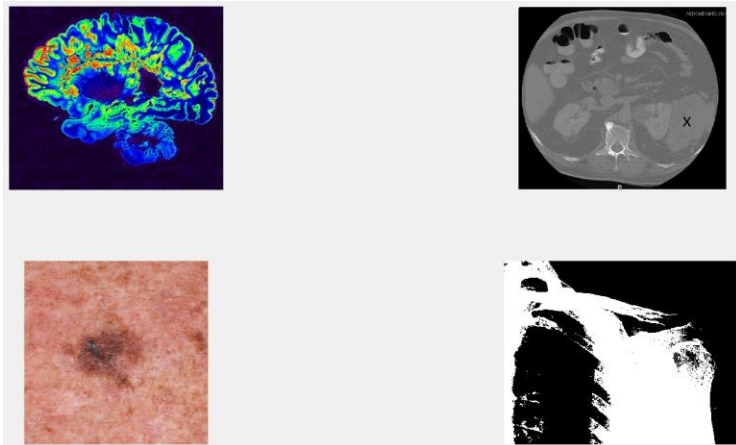
Pero vamos, no hay mayor problema porque con `imshow` sale igual. Lo único que en la solución habría que quitar los números de los `axis`, meto las imágenes que salen, quitado las otras...

Tabla 1.3:

	Clase (indexada / <i>true-color</i>)	Color / Grises / Binaria
<code>MRI_pseudo_colored.jpg</code>	True color	Color
<code>CT_abdomen.jpg</code>	True color	Grises
<code>Skin.tif</code>	Indexada	Color
<code>Xray_th.tif</code>	True color	Binaria

	<code>ima</code>	<code>879x1024x3 ...</code>
	<code>ima1</code>	<code>732x833 uint8</code>
	<code>ima2</code>	<code>560x560 uint8</code>
	<code>ima3</code>	<code>367x472 logi...</code>
	<code>map</code>	<code>[]</code>
	<code>map1</code>	<code>[]</code>
	<code>map2</code>	<code>256x3 double</code>
	<code>map3</code>	<code>[]</code>

¹ Descárguense estas imágenes de la página *moodle* de la asignatura.



A continuación, cambie la clase de imágenes que acaba de leer (respectivamente de VLT a *true-color* o de *true-color* a VLT) y represéntelas en una misma figura. Para ello, utilice las funciones `gray2ind()`, `rgb2ind()`, `ind2rgb()` e `ind2gray()`.

Convierta las tres imágenes a escala de grises (salvo las que ya lo estén) y represéntelas de nuevo. Para convertir una imagen en color en otra con niveles de gris se recomienda utilizar `rgb2gray()`; deduzca de la ayuda de esta función cómo convertir las imágenes dadas.

1.4 Ejercicio 4: grabación de imágenes a un archivo.

Según se ha visto en los ejercicios 1 y 2, la función `imshow` permite presentar imágenes de tipo `double`, con valores tanto positivos como negativos. Sin embargo, los formatos de almacenamiento de imágenes más extendidos exigen que éstas se encuentren indexadas (de tipo `uint8` o `uint16`) o que sean *true-color* con componentes de tipo `uint8`. De ahí que para almacenar o intercambiar imágenes sea recomendable que tengan este tipo antes de grabarlas, junto con su mapa de colores si son indexadas.

El objetivo de este ejercicio es grabar alguna de las imágenes obtenidas en los ejercicios 1 o 2. Cuando la imagen sea de tipo `double`, utilice primero la función `im2uint8()` para convertirla a una imagen *true-color* con componentes de tipo `uint8` (este paso no es estrictamente necesario, ya que la función `imwrite` lo realiza automáticamente para imágenes de tipo `double`). Recuerde que esta función (y, por lo tanto, la función `imwrite`) sólo arroja el resultado esperado si la imagen de tipo `double` toma valores en el rango $[0,1]$; si no, debe realizar la conversión usted mismo, siguiendo el procedimiento expuesto en el Ejercicio 2.

Una vez creada la imagen de tipo `uint8`, utilice la función `imwrite()` para almacenarla en un fichero. Luego intente visualizar la imagen grabada con un programa cualquiera (que no sea MatLab). Si no lo consigue, es porque el ajuste de rango de la imagen obtenida no es correcto.

Por último, utilice la función `imfinfo(ima)` si quiere verificar las propiedades de la imagen guardada. Utilice la ayuda de MATLAB para obtener más información acerca de esta función.

2 Operaciones con imágenes

2.1 Ejercicio 5: operaciones básicas con imágenes.

Este ejercicio propone realizar las operaciones necesarias para superponer (sumar) a una imagen que denominaremos *imagen original*, la imagen de la función discreta correspondiente a $\psi(x, y) = 0.5 + 0.5 \cdot \cos(2\pi x + 4\pi y)$.

Para poder operar en MatLab con una sola imagen (ponderarla, escalarla, etc.) sin perder rango de representación en el resultado esta siempre ha de ser de tipo `double` y siempre ha de ser *true-color*. Si la imagen es indexada no suele tener sentido operar con ella, ya que las variaciones de valor de sus píxeles suponen cambios de índices en una tabla, operación cuyo resultado es en general incierto. Si lo que se desea es operar con dos imágenes (sumarlas, restarlas, etc.), ambas deben ser de tipo `double` y de tipo *true-color*, y ambas deben tener las mismas dimensiones (filas, columnas y número de bandas o valores por píxel).

Las conversiones necesarias para conseguir esta situación se resumen en:

- Para convertir una imagen indexada en una imagen *true-color* utilice la función `ind2rgb(ima)`. Esta función devuelve una imagen *true-color* que además es de tipo `double` y definida en el rango $[0,1]$.
- Para convertir una imagen *true-color* de tipo `uint8` o `uint16` a una de tipo `double` utilice `im2double(ima)`. Esta función devuelve una imagen *true-color* definida en el rango $[0,1]$, y de la misma clase (indexada o *true-color*) que la original.
- Para convertir una imagen *true-color* de tres componentes (rojo, verde, azul) en una imagen *true-color* de una componente (nivel de gris), utilice `rgb2gray(ima)`. En este caso, se perderá la información de color.
- Para convertir una imagen *true-color* de una componente (nivel de gris) en una imagen *true-color* de tres componentes (rojo, verde, azul), MatLab no aporta una función específica: debe generar usted mismo un *array* con las tres componentes. En este caso, no se añadirá color (las tres componentes serán iguales), pero sí la posibilidad de que una operación genere color (por ejemplo, modificando de distinto modo cada componente). El código para convertir una imagen de una componente `ima_gray` en una de tres `ima_rgb` es el siguiente:

```
>> ima_rgb=ones(ima_h, ima_w, 3);  
>> ima_rgb(:, :, 1)=ima_gray;  
>> ima_rgb(:, :, 2)= ima_gray;  
>> ima_rgb(:, :, 3)= ima_gray;
```

Para realizar este ejercicio, en primer lugar, lea y represente la imagen original con la que desee trabajar (utilice `xray.jpg`). Obtenga, utilizando la función `size`, sus dimensiones (anchura y altura de la imagen, medidas en elementos o píxeles). Conviértala en una imagen de tipo `double`, para poder operar con ella.

A continuación, debe generar la imagen discreta en el mismo número de puntos que la imagen original. Para ello, defina la función discreta dada en el rango $0 \leq x < 1, 0 \leq y < 1$, pero varíe el tamaño de los vectores del retículo de modo que $\vec{v}_1 = (1/ima_w, 0)$, $\vec{v}_2 = (0, 1/ima_h)$, donde `ima_w` e `ima_h` son las dimensiones horizontal y vertical de la imagen original. De este modo habrá generado una imagen

true-color de una componente (es decir, con niveles de gris). Como la imagen `xray.jpg`² es *true-color* de tres componentes, deberá convertir esta imagen discreta en una de tres componentes, según se ha descrito más arriba.

Observe que las dos imágenes, la original y la generada, varían en un rango $[0,1]$. Súmelas y divida el resultado por dos para mantener el rango en $[0,1]$. Si el resultado no hubiera estado en este rango, debería desplazar y escalar la imagen resultante para adecuarla a él. Represente finalmente el resultado con `imshow` y comente el efecto observado:

Observaciones 2.1:

2.2 Ejercicio 6: Procesamiento a nivel de bloque

En el apartado anterior se ha descrito cómo realizar operaciones para modificar imágenes a nivel de píxel. En cambio, en algunos tipos de operaciones es conveniente dividir la imagen en bloques y aplicar una función a cada bloque de manera individual. Para ello podemos utilizar la función `blkproc()`

```
>> [ima_procesada] = blkproc(ima, [m n], fun)
```

Donde `m` y `n` indican las dimensiones del bloque a procesar y `fun` es un enlace a una función que acepta como entrada una matriz `x` (de dimensiones `m` y `n`) y devuelve una matriz, vector o escalar `y`, donde: `y = fun(x)`. Los enlaces a funciones se definen mediante el uso del comando '@'. Por ejemplo, para la función `mean()` sería:

```
>> func = @(x) mean(mean(x));
```

Para ilustrar el uso de esta función, lea la imagen `skin_gray.jpg` y utilice la función `blkproc()` para calcular el valor medio de bloques de tamaño 8x8 (se recomienda consultar la ayuda MATLAB). Represente los resultados obtenidos y comente su relación con una operación de escalado de una imagen.

Observaciones 2.2:

² Descárguese esta imagen de la página *moodle* de la asignatura.