Antes de nada, como te dije, cambiamos el punto 6 del temario planeado a uno relacionado. El título podría ser "Algunos métodos de restauración de imágenes" u otra variante que te guste. El punto 7 lo omitimos, por lo que esta hoja es la última.

Necesitarás usar `matlab`. Supongo que tienes acceso *online* porque en principio todos los alumnos de la UAM lo tenéis. No te asustes si no recuerdas demasiado de `matlab` porque los programas principales te los daré yo y tus tareas serán fundamentalmente experimentar con ellos ajustando los parámetros. También puedes usar `octave`, si lo prefieres, en ese caso deberás cargar los paquetes `image` y `signal`, incluyendo al principio de los programas de esta hoja `pkg load image` y `pkg load signal`.

Para comenzar debes aprender las bases de lo que se llama análisis de Fourier discreto

1) Lee §2.2.1 [Cha20] hasta el final de la página 63. Conviene que primero des un vistazo a la parte de §1.2.1 en la página 10 donde se explica la notación $e(x)$ y otras cosas.

En `matlab` la DCT se calcula con `dct` y su inversa con `idct`. Los análogos bidimensionales que aparecerán en los programas son `dct2` e `idct2`.

Tomemos una función 1-periódica que no sea una combinación lineal de senos y cosenos, por ejemplo

$$f(x) = \operatorname{sen}\Big(\frac{1}{2}\operatorname{sen}(2\pi x) - \frac{1}{4}\cos(6\pi x)\Big).$$

Como f es suave, las frecuencias altas deberían tener poca importancia al analizar f. El siguiente programa, discretiza $f(x)$ en la variable `y` usando `N` nodos, omite los últimos `N-M` valores de su DCT (los de las frecuencias altas) e invierte el resultado para obtener `yrec`. Si dibujas las gráficas de `y` e `yrec`, por ejemplo con `plot(x,y,x,yrec)`, verás que se parecen mucho a pesar de que nos hemos olvidado de casi el 90 % de las frecuencias (esta es la clave para algunos métodos de compresión, aunque se suelen usar *wavelets* en vez de cosenos [Wal08]).

```
N = 128
M = 15
x = linspace(0,1,N);
y = sin(sin(2*pi*x)/2-cos(6*pi*x)/4);

mmask = zeros(1,N);
mmask(1:M) = 1;
yrec = idct( dct(y).*mmask );
```

2) Haz una tabla o una gráfica que muestre el tamaño de $\|\mathtt{y} - \mathtt{yrec}\|_\infty$ cuando `M` vale 10, 20,..., 100.

3) Utiliza a lo más una página y media para: 1) Describir las fórmulas para la DFT, la DCT y sus inversas; 2) dar una prueba de la fórmula de inversión para la DFT (con matrices unitarias o como tú prefieras); 3) incluir el ejercicio anterior para ilustrar que las funciones suaves tienen DCT (y DFT) que decae rápidamente.

Al igual que la transformada de Fourier habitual, la DFT y la DCT se extienden a más variables. Simplemente se aplican las fórmulas a cada variable por separado. Una imagen en blanco y negro (tonos de gris) es una función que al pixel (i, j) le asigna un número que indica su tono de gris. Este número normalmente está entre 0 y 255 aunque en los programas `matlab` posteriores se escala entre 0 y 1 con la instrucción `im2double`, donde 0 es negro y 1 es blanco[1]. Entonces tiene sentido considerar la DCT de una foto en blanco y negro. Habitualmente habrá discontinuidades correspondientes a aristas o sombras y por tanto no es de esperar que haya un decaimiento drástico como en el ejemplo anterior. Como un comentario al margen, en el formato JPEG [Cha20, §2.2.2] [Sal02] se hace una división en trozos de 8×8 píxeles para que en la mayoría de ellos sea fácil que la porción de foto sea suave.

En el resto de la hoja experimentarás con tres métodos para restaurar ciertos defectos de imágenes y dos de ellos tienen que ver con el análisis de Fourier discreto.

1. Filtrado homomorfo. Vamos a ver cómo usar la DCT para homogeneizar la iluminación de una imagen. A veces una iluminación descompensada responde a fines artísticos, como en los cuadros de Caravaggio, pero otras veces no es intencionada y queda antiestético que, por ejemplo, según vamos de derecha a izquierda de una foto esté más oscurecida por la posición del Sol u otra fuente puntual. Para eliminar gradientes de iluminación, se parte del *modelo de iluminación-reflectividad* y se usa lo que tiene el pomposo nombre de *filtrado homomorfo*. Puedes encontrar la descripción al final de §2.2.4 en [Cha20], desde "The second example..." en la página 83. Mira también en mi web la página:

`http://matematicas.uam.es/~fernando.chamizo/dark/d_homom.html`

donde hay imágenes de tamaño mayor y un programa del que usarás una parte.

4) Escriba unas pocas líneas resumiendo el modelo y el método de filtrado. Si te es posible, da un vistazo a alguna de las referencias.

Ahora vamos con la tarea que consiste en experimentar con el filtro usando un programa sencillo hecho por mí.

Copia la función `hom_ff` que está al final de la página web anterior, en un fichero llamado `hom_ff.m`. Como se explica en la línea de comentarios, los argumentos segundo tercero y cuarto son σ, γ_L y γ_H y el último es el nombre del fichero donde quieres que se guarde el resultado. Para utilizar esta función, considera este programa:

```
ima = imread('original.png');
ima = rgb2gray(ima);  % solo si original.png es en color
imwrite( ima ,'original_gris.png'); % solo si original.png es en color
ima = im2double(ima);
epsi = 1/512;
hom_ff( ima, 60, 0, 3, epsi, 'resultado.png')
```

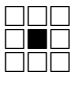
[1]El caso de imágenes en color no lo consideraremos pero es similar con tres números que corresponden a los *canales de color* R (*red*), G (*green*) y B (*blue*).

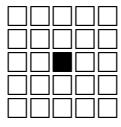
Esto aplica a la imagen de partida `original.png` el filtro con $\sigma = 60$, $\gamma_L = 0$ y $\gamma_H = 3$ para obtener `resultado.png`. Solo funciona con imágenes cuadradas y te recomiendo que las uses de tamaño 256×256 o 512×512 (en el primer caso quizá sea bueno duplicar `epsi`). El filtro está pensado para imágenes en blanco y negro (tonos de gris) pero lo mismo no sabes cómo convertir una imagen tomada con tu cámara a este formato con Photoshop o GIMP. Por eso he incluido las líneas 2 y 3 que pasan el original a blanco y negro y guardan una copia. Si ya lo estuviera, como en el ejemplo de mi web, esas dos líneas sobran. He puesto la extensión `png` pero puedes usar `jpg`, que es lo común para fotos, y otros formatos que te permita `matlab`. Prueba con una imagen cuadrada cualquiera que el anterior programa no te da errores, aunque el resultado sea absurdo. La imagen debe estar en el mismo directorio que el programa o en un *path* que reconozca `matlab`.

5) Haz una foto (preferible) o busca una imagen en la web que tenga un gradiente suave de iluminación, es decir, que esté más oscurecida a un lado que a otro sin cambios demasiado bruscos. Recórtala para que sea cuadrada (recuerda los tamaños recomendados) y empieza a practicar con los parámetros σ, γ_L y γ_H en el programa anterior hasta conseguir que `resultado.png` tenga una iluminación más nivelada que el original, o su versión en blanco y negro. Reproduce los dos o tres resultados que te parezcan más atractivos. En una imagen artística con un gradiente exagerado, los parámetros pueden ser muy diferentes de los del ejemplo con texto de mi web y los resultados no muy satisfactorios porque el modelo es solo aproximado.

Seguramente tengas que probar con varias fotos y parámetros antes de encontrar resultados que te gusten. En eso consiste la tarea. Cuanto mejor entiendas el filtro, menos al azar serán las pruebas.

2. El filtro de mediana. El *ruido sal y pimienta* [Wik20] en una imagen (siempre en esta hoja, en blanco y negro) es una contaminación que consiste en que aleatoriamente algunos píxeles pasan a ser blancos (sal) y otros negros (pimienta). Esto puede ocurrir cuando hay una saturación esporádica de la señal al transmitir la información. El *filtro de mediana* lo reduce de forma bastante espectacular sustituyendo el valor de cada píxel por la mediana de los valores en el cuadrado $(2r+1) \times (2r+1)$ que lo rodea con r píxeles en cada dirección. Habitualmente se toma $r = 1$.

$r = 1 \quad \rightarrow$ 

$r = 2 \quad \rightarrow$ 

El siguiente programa contamina una imagen con un ruido de sal y pimienta de densidad `d` y aplica el filtro con r igual a `r`. Tras el apartado anterior ya tendrás al menos una imagen en tonos de gris. Si prefieres partir de una en color, como antes, quita el `%` de comentario de las líneas 2 y 3.

```
ima = imread('original.png');
%ima = rgb2gray(ima);
%imwrite( ima ,'original_gris.png');
ima = im2double(ima);


% Imagen con ruido
d = 0.15; %densidad de ruido
imagn = imnoise(ima,'salt & pepper',d);
imwrite(imagn, 'noisy.png')

% Filtro mediano
r = 1;
imagm = medfilt2(imagn,[2*r+1,2*r+1]);
imwrite(imagm, 'resultado.png')
```

6) Practica con algunos ejemplos manteniendo `r=1`. Tienes que comparar la imagen contaminada `noisy.png` con la limpieza conseguida en `resultado.png`. Examina el resultado cuando `d` varía del 15 % indicado de píxeles dañados a otras proporciones. Escribe unas líneas explicando por qué el filtro de mediana funciona bien con el ruido de sal y pimienta cuando su densidad no es muy grande.

En principio, si la densidad del ruido es mayor, sería natural tomar `r` mayor que 1 pero eso da problemas.

7) Comprueba con ejemplos qué tipo de problemas aparecen y justifica a qué se deben.

Añadiendo las siguientes líneas al programa:

```
% Filtro de mediana solo para píxeles negros o blancos
mask = (imagn.*(1-imagn)==0);
imagm2 = (1-mask).*imagn + mask.*medfilt2(imagn, [2*r+1,2*r+1] );
imwrite(imagm2, 'resultado2.png');
```

se aplica el filtro de mediana solo a los píxeles que valen cero o uno que son los únicos que pueden estar afectados por el ruido.

8) Compruebe que el nuevo filtro con `r=2` funciona mejor que el anterior para `r=1` y `r=2` con densidades algo mayores. Busca una razón para esta mejora.

3. Reducción de ruido periódico. Ahora nuestro objetivo es limpiar una imagen de ruido en forma de *tonos puros* periódicos que se superponen a los valores dando lugar a ciertas oscilaciones. En la electrónica analógica esto no es tan extraño por las posibles interferencias de componentes funcionando a baja frecuencia. Quizá te suene haber visto algo de este tipo en imágenes de televisión antiguas. En la página de mi sitio web:

`http://matematicas.uam.es/~fernando.chamizo/dark/d_periodic.html`

traté el problema y escribí un programa. La idea es que el ruido se tiene que manifestar en picos sueltos en la DCT y un pico se puede eliminar con un filtro de mediana. Lo malo es que la DCT tiene de por sí picos, por ejemplo para la frecuencia nula, y hay que buscar algo que

a ser posible no toque en absoluto estos picos buenos. Los correspondientes al ruido deben ser muy finos, porque la frecuencia está muy localizada, por ello impuse como regla para detectar un pico de ruido que el valor en un punto sea mucho mayor que la mediana de los de alrededor.

9) Por favor, relee el párrafo anterior y también lo mencionado en la página web hasta que entiendas el procedimiento. Si necesitas más detalles, mira el código de la función `denoip`, incluido al final de la página.

Considera el siguiente programa que aplica la idea para cierto `r` que da el tamaño del filtro de mediana y para cierto `alpha` que dice cuántas veces mayor debe ser un valor que la mediana para que el pico se considere malo. La imagen debe ser de tamaño 512×512.

```
1  ima = imread('original.png');
2  %ima = rgb2gray(ima);
3  %imwrite( ima ,'original_gris.png');
4  ima = im2double(ima);
5
6  % Imagen con ruido
7  t = linspace(0,1 ,512);
8  [X, Y] = meshgrid(t, t);
9  iman = ima + 0.2*sin(16*pi*(X-Y));
10 imwrite( im2double(mat2gray(iman)) ,'noisy.png' );
11
12 % Limpieza
13 r = ...
14 alpha = ...
15 res = denoip( iman, r, alpha );
16 imwrite( res ,'resultado.png' );
```

10) Escoge una imagen y experimenta con valores para los puntos suspensivos en las líneas 13 y 14 hasta que te parezca que no puedes reducir más el ruido. No estoy seguro pero sospecho que los imágenes que no tienen grandes zonas totalmente blancas o negras deberían ser más adecuadas. Si no consigues una reducción apreciable, prueba con diferentes imágenes. También, si quieres, puedes cambiar la contaminación de la línea 9 por otra.

Tarea a entregar. El documento a entregar debe comenzar la parte teórica del análisis de Fourier discreto con un ejemplo, descrita en el tercer ejercicio. Después, debes incluir los experimentos del filtrado homomorfo, el filtro de mediana y la eliminación de interferencias periódicas, todos ellos precedidos con algunas explicaciones con el grado de profundidad que prefieras.

No te pongo un límite para el tamaño del documento porque, más que en otras ocasiones, hay muchas posibilidades para modificarlo según las necesidades de tu TFG, por ejemplo escalando las imágenes o reduciendo el número de ejemplos. Para las partes de código que están en mi web, puedes dar la referencia o copiarlo, lo que prefieras.

Referencias

[Cha20] F. Chamizo. A course on signal processing. http://matematicas.uam.es/~fernando.chamizo/libreria/libreria.html, 2020.

[Sal02] D. Salomon. *A guide to data compression methods*. Springer-Verlag, 2002.

[Wal08] J. S. Walker. *A primer on wavelets and their scientific applications*. Studies in Advanced Mathematics. Chapman & Hall/CRC, Boca Raton, FL, second edition, 2008.

[Wik20] Wikipedia contributors. Salt-and-pepper noise — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Salt-and-pepper_noise&oldid=943959344, 2020. [Online; accessed 21-March-2021].