Introducción. Señales y análisis de Fourier

En el Diccionario de la Real Academia Española, se define 'señal' –entre otras acepciones- como

> **15.** f. *Fís.* Variación de una corriente eléctrica u otra magnitud que se utiliza para transmitir información.

Tanto si la información que transmitimos es biológica (EEG, EKG, EMG) como de cualquier otra procedencia (económica, meteorológica…), las señales con una variación continua podrán ser convertidas en series de números –se dirá que se *digitalizan*- y las series de números podrán ser tratadas para extraer la información que contienen. La forma más universalmente utilizada para tratar señales, sea cual sea su procedencia es el análisis de Fourier.

El análisis de Fourier debe su nombre a Jean Baptiste Joseph Fourier (1768-1830), un matemático y físico francés. Si bien muchas personas contribuyeron a su desarrollo, Fourier es reconocido por sus descubrimientos matemáticos y su visión en el uso práctico de las técnicas. Su interés se centraba en la propagación de calor, presentando en 1807 un trabajo en el Instituto Francés sobre el uso de funciones senoidales para representar distribuciones de temperatura.

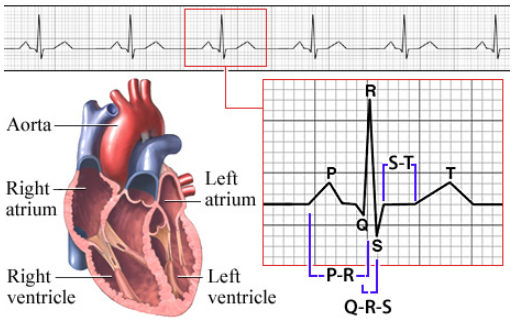
El trabajo presentaba un resultado controvertido: *que cualquier señal continua y periódica podía representarse como la suma una serie de ondas senoidales adecuadamente elegidas*. Entre los revisores de su trabajo estaban dos de los matemáticos más reputados de su época y también de la Historia, Joseph Louis Lagrange (1736-1813) y Pierre Simon de Laplace (1736-1827), que habían sido sus maestros en la Escuela Normal Superior de París.

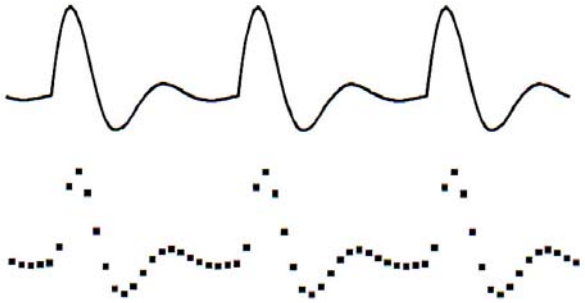
Mientras que Laplace y otros revisores votaron a favor de la publicación del trabajo, Lagrange protestó categóricamente. Durante años, Lagrange insistió en que esta aproximación no podía aplicarse a señales con 'esquinas', es decir, con cambios bruscos de pendiente como una onda triangular. El Instituto Francés cedió ante el prestigio de Lagrange y el trabajo fue rechazado, y no fue publicado hasta la muerte de éste, 15 años después. Afortunadamente, Fourier tenía muchas otras cosas que le mantenían ocupado: actividades políticas, expediciones con Napoleón a Egipto, y tratar de evitar la guillotina después de la Revolución Francesa (¡literalmente!).

¿Quién tenía razón? Pues depende. Lagrange estaba en lo cierto al decir que una serie de funciones senoidales no puede representar de manera exacta una señal con una esquina. Sin embargo se puede aproximar mucho. Tan cerca que la diferencia es irrelevante.

Muestreo de señales continuas.

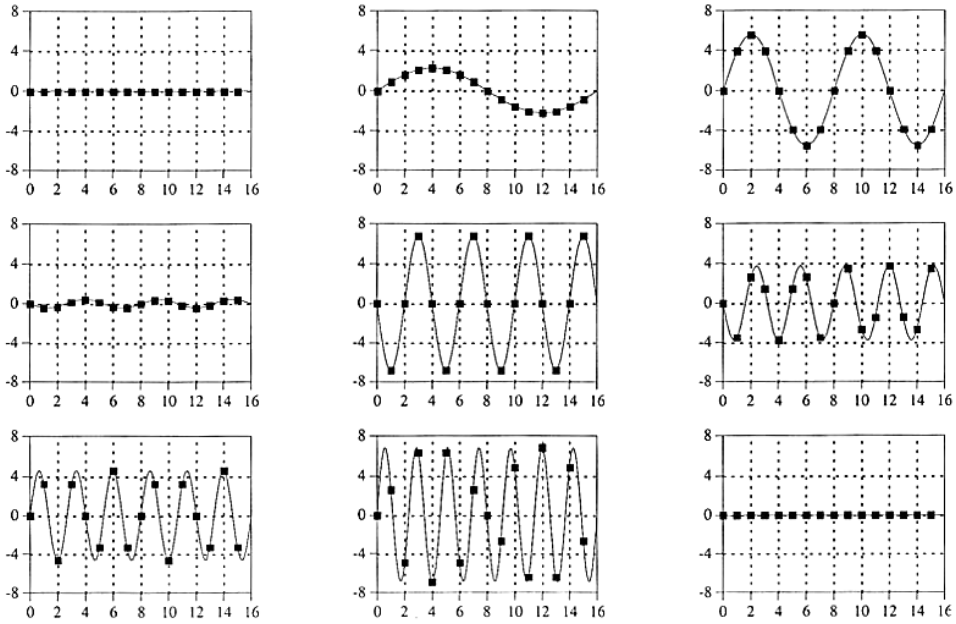
Las señales biológicas -EKG, EEG, EMG o potencial de membrana- varían de manera continua, así como su representación como señales eléctricas. Sin embargo de ese modo no pueden ser registradas en un archivo en un ordenador, ni se puede hacer ningún tratamiento con la información que contienen. Para ello deberá tomarse una serie de muestras –valores de la señal en un instante de tiempo- en forma de series de números, y ésta debe ser representativa de la señal, y no perder la información que contiene.





La forma habitual de hacer esta transformación es tomar muestras a intervalos regulares de tiempo. A este proceso lo llamaremos muestrear o digitalizar la señal. La velocidad a que se toman esas muestras, medida normalmente en número de muestras por segundo, se llamará *velocidad de muestreo*, y su elección es muy importante para no perder la información que contiene la señal, y también para no generar un archivo de datos demasiado grande.

En los gráficos de abajo pueden verse diferentes señales y diferentes puntos que representan el muestreo hecho. En algunos casos es muy claro que los puntos no representan bien la señal.



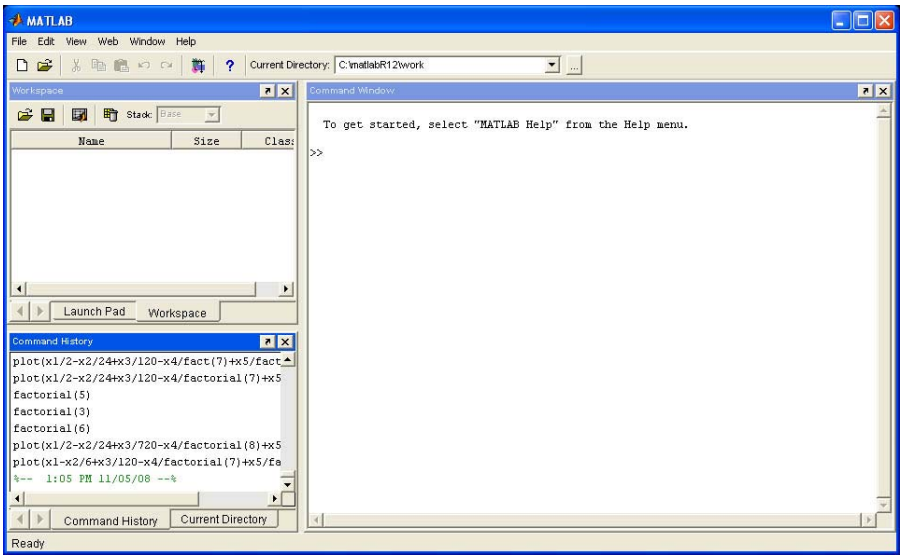
Introducción a Matlab.

MATLAB (abreviatura de MATrix LABoratory, "laboratorio de matrices") es un software matemático que ofrece un entorno de desarrollo con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows y Apple Mac OS X. Entre sus capacidades básicas están: la manipulación de tablas de datos, la representación de datos y funciones, la implementación de algoritmos, y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete MATLAB puede además ampliar sus posibilidades con las cajas de herramientas (toolboxes) específicas para un gran variedad de aplicaciones distintas, desde el tratamiento de imágenes a la bioinformática pasando por la estadística o la economía.

http://www.mathworks.com/products/product_listing/index.html

Es el software más usado en universidades y centros de investigación y desarrollo. En los últimos años ha aumentado el número de prestaciones, como la de programar directamente procesadores digitales de señal.

Matlab puede utilizarse para realizar cálculos sencillos de manera directa, como si se tratase de una calculadora, o bien programar tareas complejas para realizar otros cálculos de manera más automatizada. El interfaz básico para utilizarlo como calculadora es el de la figura de abajo, si bien difiere de unas versiones a otras



A través de la línea de comandos (que sigue al símbolo >>) teclearemos instrucciones desde las más sencillas, como una simple suma, hasta las más complejas.

>>2+3

ans = 5 (ans es el acrónimo de *answer*, la respuesta es 5)

Funciones senoidales. Representación gráfica con Matlab.

La afirmación de Fourier de que "*cualquier señal continua y periódica podía representarse como la suma una serie de ondas senoidales adecuadamente elegidas*", se tratará ahora de aplicar, pero en sentido contrario. Generaremos ondas senoidales y las sumaremos, para ver el resultado.

Matlab tiene una gran librería de funciones matemáticas, entre ellas las trigonométricas como la función seno (*sin* en inglés)

$$y = \sin(x)$$

Para obtener el seno de un número simplemente se escribe la función. Una de las ventajas del uso de Matlab es que las expresiones matemáticas se escriben de manera muy similar a como se haría sobre el papel.

```
>>sin(90)
ans = 0.8940
```

Nota: las funciones trigonométricas en Matlab (y casi en cualquier programa) utilizan como unidades para los ángulos los radianes, no los grados. Una circunferencia de 360º tiene 2 veces π radianes. Matlab utiliza las letras 'pi' para dicho número

```
>>sin(pi/2)
ans = 1
```

Otra de las ventajas fundamentales de Matlab es el trabajo sencillo con matrices o tablas de datos. Una tabla de datos (con tantas filas y columnas) se introduce tecleándola y utilizando como separadores los símbolos coma, punto y coma y corchetes.

```
>>[1,2,3]
ans =
     1     2     3

>>[1;2;3]
ans =
     1
     2
     3

>>[1,2;3,4]
ans =
     1     2
     2     4
```

En ocasiones, cuando hagamos cálculos de este modo, no querremos ver el resultado de alguno por ocupar mucho espacio de pantalla. Si finalizamos cualquier instrucción con un punto y coma ';', Matlab guardará el resultado de la operación pero no lo mostrará por pantalla.

Si aplicamos una función como el seno, a una tabla de datos, el resultado será otra tabla del mismo tamaño en la que se ha aplicado la función a cada elemento

```
>>sin([1,2,3])
ans =
        0.8415    0.9093    0.1411
```

Una serie de datos puede verse como una tabla de una única fila o columna, y podrá representarse por medio de muchos tipos diferentes de gráficos (líneas, barras, colores). Las señales biológicas, una vez digitalizadas, serán sólo eso: series de números. Los resultados en forma de valores sueltos y series o tablas de datos, pueden guardarse para no perder la información. Para ello se utilizarán *variables*. La forma de guardar un determinado contenido en una variables es el símbolo '='

```
>>a=[2,3,4]
a =
        2        3        4
```

De este modo, cuando queramos recuperar el contenido, o utilizarlo, sólo tenemos que llamarla por su nombre

```
>>a
a =
        2        3        4
```

Las variables se podrán usar dentro de las fórmulas, como veremos a continuación, pero empezaremos usando una serie especial: una serie que representará el tiempo desde 0 a 1 segundo en intervalos de una décima.

```
>>t=[0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1];
```

Nota: Matlab utiliza el punto '.' como símbolo decimal, y la coma ',' para separar valores en una tabla.
Nota: Matlab puede generar listas regulares de números escribiendo [inicio:paso:fin]. En el caso anterior, sería [0:0.1:1]

```
>>t=[0:0.1:1]
t =
        0        0.1      0.2      0.3      0.4      0.5      0.6      0.7      0.8      0.9      1
```

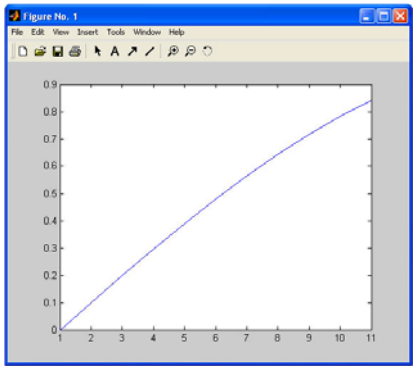
Si aplicamos ahora la función seno a nuestra variable t y guardamos el contenido en otra variable (x) tendremos una nueva tabla de valores

```
>>x=sin(t)
x =
 Columns 1 through 7
        0   0.0998   0.1987   0.2955   0.3894   0.4794   0.5646
 Columns 8 through 11
        0.6442   0.7174   0.7833   0.8415
```

La serie de datos que hemos creado representa la función seno de t, cuando t vale 0, 0.1, 0.2… hasta 1. Para representarla gráficamente utilizaremos la función plot

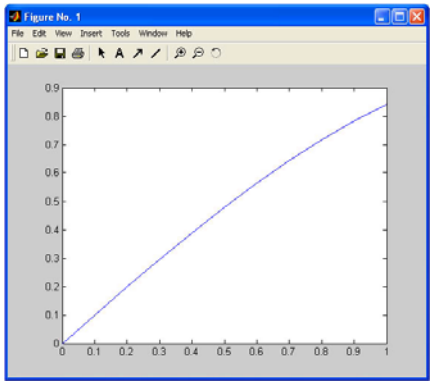
>>plot(x)

Matlab abrirá ahora una nueva ventana con el gráfico



La función seno oscila arriba y abajo. Lo que hemos representado es únicamente un tramo de la primera subida. Además Matlab ha representado los números uno a continuación de otro, y por eso en el eje de abajo aparece una escala de 1 a 11. Si queremos que represente nuestras escala de tiempo desde 0 hasta 1, escribiremos

>>plot(t,x)

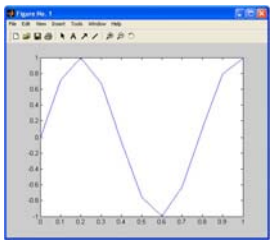


Trataremos ahora de generar diferentes ondas senoidales. Partiremos de una función sencilla e iremos modificando los valores que las caracterizan, que son la amplitud, la frecuencia y la fase (se verá que significa cada cosa). Al mismo tiempo veremos la necesidad de utilizar un número mínimo de puntos para representar bien una señal. A medida que realicemos cambios sobre la función inicial sin(t), éstos estarán marcados en amarillo.

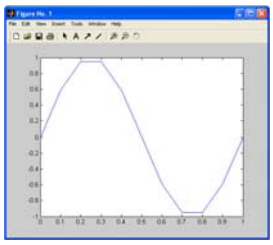
Frecuencia de la señal.

Modificaremos la frecuencia de la señal senoidal multiplicando la variable t (dentro del paréntesis del seno) por un número, por ejemplo 8, o 2π

```
>>x=sin(8*t);
>>plot(t,x)
```

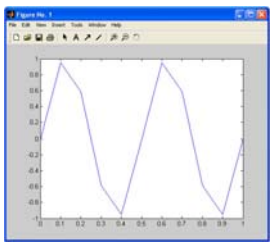


```
>>x=sin(2*pi*t);
>>plot(t,x)
```



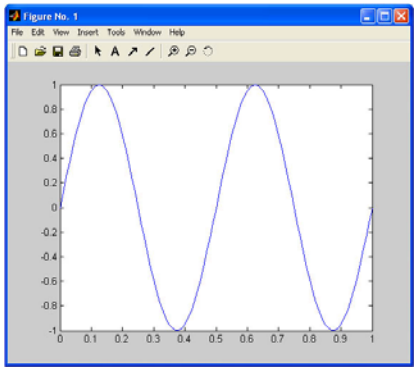
En el caso anterior hemos visto además una cosa importante: cuando el valor que multiplica a t es un múltiplo de 2π, tendremos ciclos completos de ondas.

```
>>x=sin(4*pi*t);
>>plot(t,x)
```



También podemos observar que el gráfico que une los puntos del muestreo cada vez se parece menos a la onda inicial continua. Estamos cerca de perder información. Representaremos mejor la curva con más puntos, vamos a probar con 100 intervalos en lugar de 10.

```
>>t=[0:0.01;1]
>>x=sin(4*pi*t)
>>plot(t,x)
```

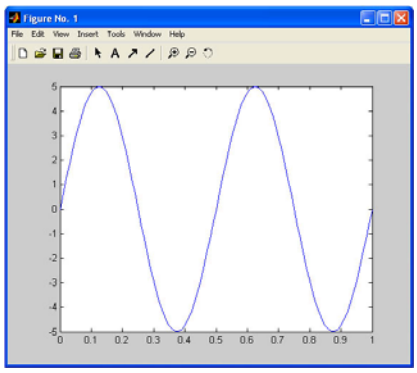


Nota: hemos repetido varias veces las mismas instrucciones. Si le damos al cursor del teclado hacia arriba, nos pondrá la lista de última órdenes para poder usarlas sin teclearlas de nuevo.

Amplitud de la señal y componente continua

Si multiplicamos el valor de la función seno por una constante, estaremos multiplicando cada uno de los valores de la señal por ese número, y estaremos cambiando la amplitud de variación de la curva

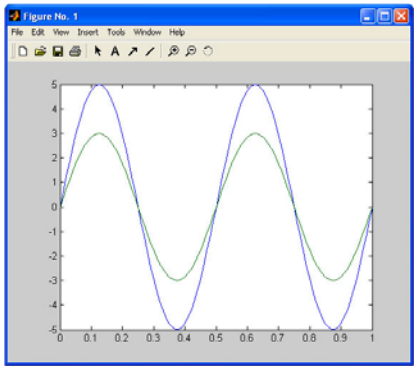
```
>>t=[0:0.01;1]
>>x=5*sin(4*pi*t)
>>plot(t,x)
```



La señal que hemos generado tiene la misma forma que la anterior, pero ahora varía entre un mínimo de -5 y un máximo de 5. Hemos cambiado su *amplitud*.

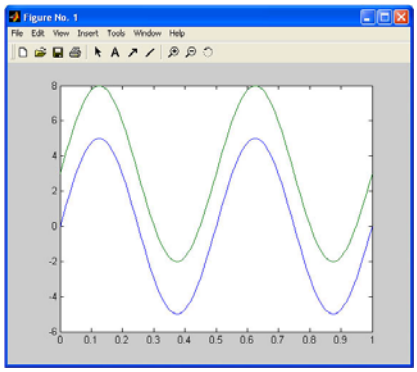
También podemos representar gráficamente varias curvas juntas usando plot(t,x1,t,x2), y es un buen momento para usar el punto y coma final para que no nos muestre toda las ristra de números

```
>>t=[0:0.01;1]
>>x1=5*sin(4*pi*t);
>>x2=3*sin(4*pi*t);
>>plot(t,x1,t,x2)
```



Si además le sumamos a todos los valores una constante, habremos variado el valor medio de la señal, que llamaremos *componente continua*. Ahora el rango de variación ha subido, tanto el mínimo como el máximo, en esa cantidad (en el ejemplo entre -2 y 8).

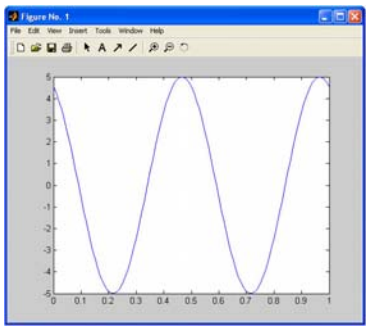
```
>>t=[0:0.01;1]
>>x1=5*sin(4*pi*t)
>>x2=3+5*sin(4*pi*t)
>>plot(t,x)
```



Modificación de la fase.

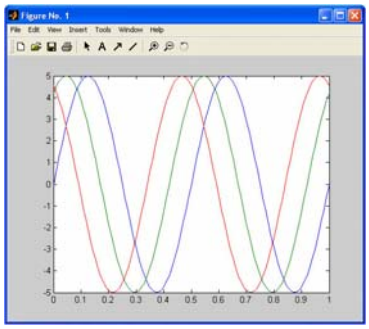
Si ahora sumamos un valor constante a la escala de tiempo (dentro del paréntesis de la función seno) veremos que estamos desplazando la señal, que ya no comenzará en cero. Esa diferencia temporal la llamaremos fase (a veces también desfase)

```
>>t=[0:0.01;1]
>>x=5*sin(4*pi*t+2)
>>plot(t,x)
```



O bien, para comparar los cambios

```
>>x1=5*sin(4*pi*t);
>>x2=5*sin(4*pi*t+1);
>>x3=5*sin(4*pi*t+2);
>>plot(t,x1,t,x2,t,x3)
```



Valores característicos de una señal senoidal.

Hemos visto como modificar la frecuencia, la amplitud y la fase de una señal senoidal. En el caso más general, una onda senoidal vendrá dada únicamente por esos tres números.

$$x = A \cdot \sin(2\pi ft + \delta)$$

O bien para Matlab

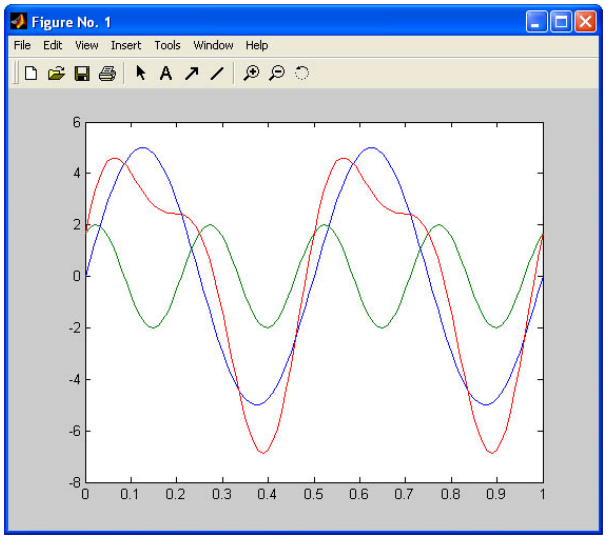
```
>>x=A*sin(2*pi*f*t+delta)
```

A será nuestra amplitud, f la frecuencia o número de ciclos por segundo, y delta la fase. La última que hemos dibujado tiene Amplitud 5, Frecuencia 2 y Fase 2. Y como siempre t puede ser un número o bien una tabla de valores para la que queremos ver el resultado y dibujar un gráfico.

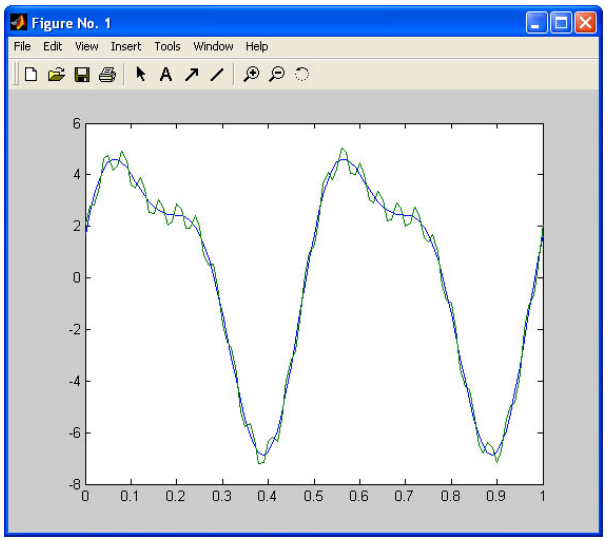
Suma de ondas senoidales.

Al igual que sumamos números, en Matlab podemos sumar tablas. La única condición es que deben tener el mismo número de elementos, para así ser sumados uno a uno. Si las tablas representan señales con la misma escala de tiempo (ojo! esto último es muy importante), estaremos sumando señales. Vamos a ver dos ejemplos

```
>>t=[0:0.01:1];
>>x1=5*sin(4*pi*t);
>>x2=2*sin(8*pi*t+1);
>>x3=x1+x2;
>>plot(t,x1,t,x2,t,x3)
```




```
>>t=[0:0.01:1];
>>x1=5*sin(4*pi*t);
>>x2=2*sin(8*pi*t+1);
>>x3=0.5*sin(50*pi+1);
>>plot(t,x1+x2,t,x1+x2+x3)
```



En el segundo ejemplo se ha indicado a Matlab que represente la suma de las dos primeras, y la suma de las tres juntas. Se ha hecho así a propósito para ver la que podría ser la suma de dos señales que contienen información más una tercera de alta frecuencia que podría representar ruido.

El objetivo final del análisis de Fourier será separar cualquier señal en componentes más sencillas –senoidales- y en su caso, eliminar las que nos sobran, como el ruido.

Ejercicios

Se tratará ahora de representar señales que son suma de varias senoidales a partir de una tabla, que tendrá la amplitud, frecuencia y fase de cada componente de la suma. Se representarán 2 segundos de tiempo. Como sugerencia, se puede representar en el gráfico la suma de una componente, luego dos y así sucesivamente para ver que aporta cada una a la suma total

Ejercicio 1

A	f	delta
1	1	0
0.3333	3	0
0.2	5	0
0.1429	7	0

Ejercicio 2

A	f	delta
1	1	0
1/9	3	pi
1/25	5	0
1/49	7	pi
1/81	9	0

http://www.youtube.com/watch?v=DzjwjDt2W1I

La transformación de Fourier y la FFT.

Hemos visto que una señal periódica –que se repite en el tiempo como un EKG- puede representarse como suma de ondas senoidales. Las funciones a sumar pueden ser muy diferentes, al igual que 1+4 y 2+3 suman 5. Las que nos interesarán especialmente, y en eso se basa el análisis de Fourier son las que tienen unas frecuencias determinadas. Las elegidas son, la frecuencia que caracteriza la señal que queremos analizar (por ejemplo 80 por minuto podría ser un valor típico de un EKG), y sus múltiplos: el doble, el triple…

$$f, 2f, 3f, 4f, 5f\ldots$$

Y así, cualquier señal podrá descomponerse en una suma como la siguiente

$$A_0 + A_1 \sin(2\pi ft + \delta_1) + A_2 \sin(2\pi 2\, ft + \delta_2) + A_3 \sin(2\pi 3\, ft + \delta_3) + A_4 \sin(2\pi 4\, ft + \delta_4) + \ldots$$

Esta forma de descomponer una señal se llama la Transformación de Fourier

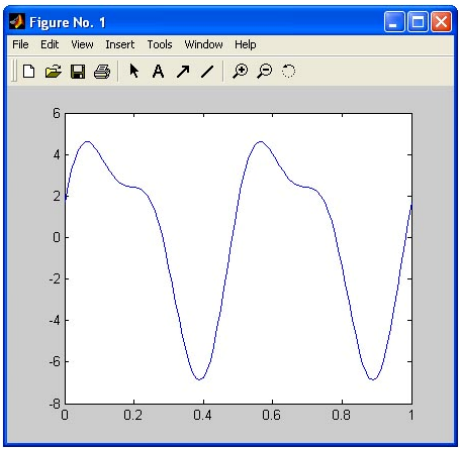
La ventaja de elegir estas funciones, que se llamarán *armónicos*, es que analizar una señal cualquiera para ver sus componentes con esas frecuencias es sencillo. La formulación matemática para hacerlo se sale de este curso, pero sí se verá la capacidad de Matlab para obtener las componentes no ya de una señal continua, sino de la serie de datos que la representa –bien o mal- en forma de muestreo. Esto se llamará la Transformación de Fourier Discreta (DFT), y hay muchas formas de calcularla. La más eficiente es la Transformada Rápida de Fourier, la FFT.

Matlab tiene implementada una función para calcular directamente una FFT, con hacer simplemente la instrucción

```
>>fft(x)
```

En donde x es nuestra serie de datos. El resultado numérico de la transformación no se interpreta directamente. Se trata de números complejos, que tendrán un módulo –que será la amplitud- y un ángulo –la fase- de cada una de las componentes. Además estarán repetidos, dando una serie de datos simétrica. Nos interesa sólo la mitad. Y además estarán multiplicados por $N/2$, siendo N el número de datos de nuestra serie. Si recuperaremos la última señal que generamos.

```
>>t=[0:0.01:1];
>>x1=5*sin(4*pi*t);
>>x2=2*sin(8*pi*t+1);
>>x=x1+x2;
>>plot(t,x)
```

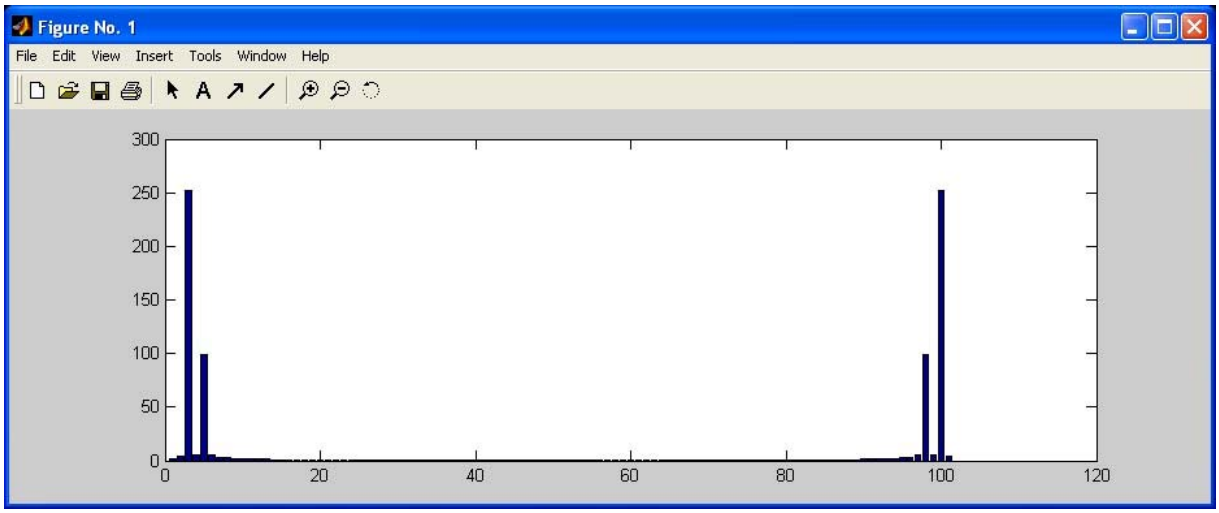


Si nuestro espacio de tiempo es 1 seg, nuestra frecuencia *f* es 1/1seg = 1Hz. Si hacemos ahora la fft, y nos quedamos sólo con las amplitudes (con la función abs) podemos escribir

>>y=abs(fft(x));

Lo representaremos ahora con un nuevo tipo de gráfico: un gráfico de barras –*bar graph*- con la función bar

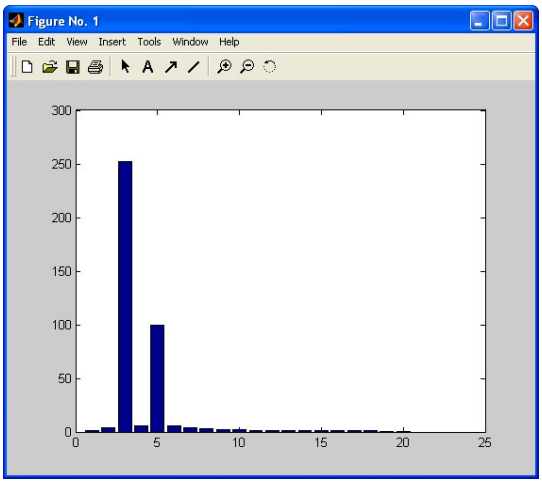
>>bar(y)



Tal y como se dijo, tenemos los datos repetidos en un gráfico simétrico, nos interesan sólo la mitad, y para ser más concretos, los primeros, porque a partir de un determinado valor son muy pequeños

Podemos representar una parte del gráfico escribiendo

>>bar(y(1:20))



Nuestro gráfico tiene dos barras mucho mayores que las anteriores, vamos a ver lo que representa cada una.

Nuestras barras representan las amplitudes de la serie

$$A_0 + A_1 \sin(2\pi ft + \delta_1) + A_2 \sin(2\pi 2\, ft + \delta_2) + A_3 \sin(2\pi 3\, ft + \delta_3) + A_4 \sin(2\pi 4\, ft + \delta_4) + \ldots$$

La que hemos generado nosotros es

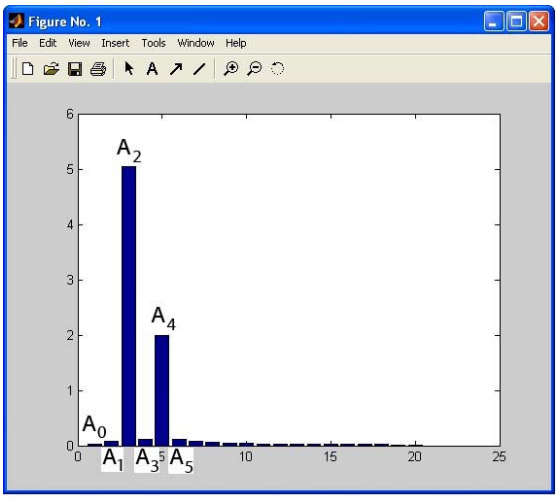
5*sin(4*pi*t)+2*sin(8*pi*t+1)

La primera barra representa A_0, la componente continua. Nuestra señal no tenía.

La segunda barra representa la amplitud A_1, correspondiente frecuencia fundamental f que tampoco existía en la señal que nos hemos generado. Tampoco tenemos en A_3.

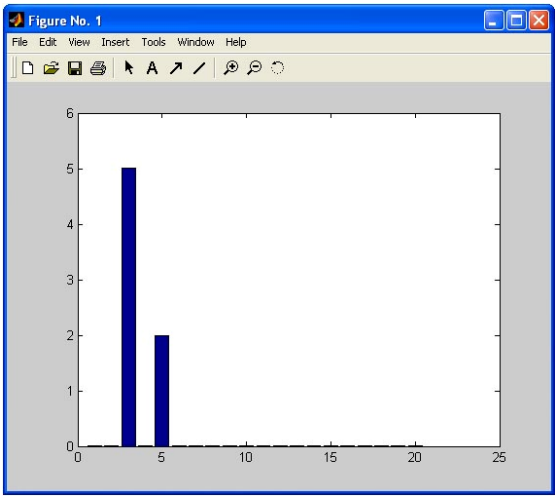
En cambio si tenemos componentes con frecuencias $2f$ y $4f$, y eso es lo que nos muestra el gráfico. Como hemos dicho, los valores de amplitudes en el gráfico están multiplicados por N/2, siendo N el número de puntos (100 en nuestro caso). Si queremos verlos sólo hay que dividir por ese número

```
>>t=[0:0.01:1];
>>x1=5*sin(4*pi*t);
>>x2=2*sin(8*pi*t+1);
>>x=x1+x2;
>>y=abs(fft(x))/50;
>>bar(y(1:20))
```



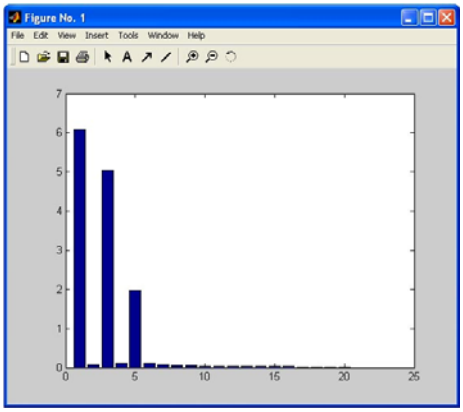
Los pequeños valores (que deberían ser cero) en otros armónicos se deben a utilizar pocos puntos para muestrear la señal. El mismo resultado con 1000 puntos en lugar de 100 es el del siguiente gráfico

```
>> t=[0:.001:1];
>> x1=5*sin(4*pi*t);
>> x2=2*sin(8*pi*t+1);
>> x=x1+x2;
>> y=abs(fft(x))/500;
>> bar(y(1:20))
```



Vamos a probar ahora con la misma señal pero con una componente continua

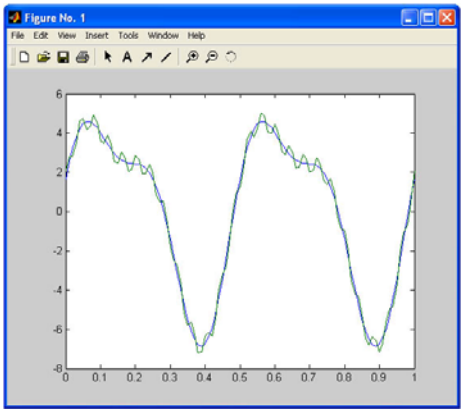
```
>>t=[0:0.01:1];
>>x1=5*sin(4*pi*t);
>>x2=2*sin(8*pi*t+1);
>>x=3+x1+x2;
>>y=abs(fft(x))/50;
>>bar(y(1:20))
```



Podemos verla en la barra A_0 ahora. Los valores de amplitud de la componente continua están multiplicados por *N* en lugar de *N/2*. Por eso aparece con valor 6.

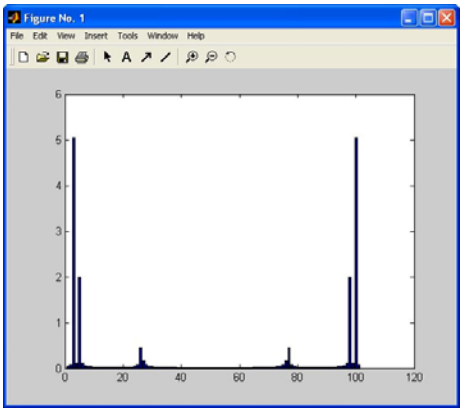
Si ahora en lugar de una componente continua añadimos el término de alta frecuencia que vimos que podía representar ruido

```
>>t=[0:0.01:1];
>>x1=5*sin(4*pi*t);
>>x2=2*sin(8*pi*t+1);
>>x3=0.5*sin(50*pi+1);
>>x=x1+x2+x3;
>>plot(t,x)
```



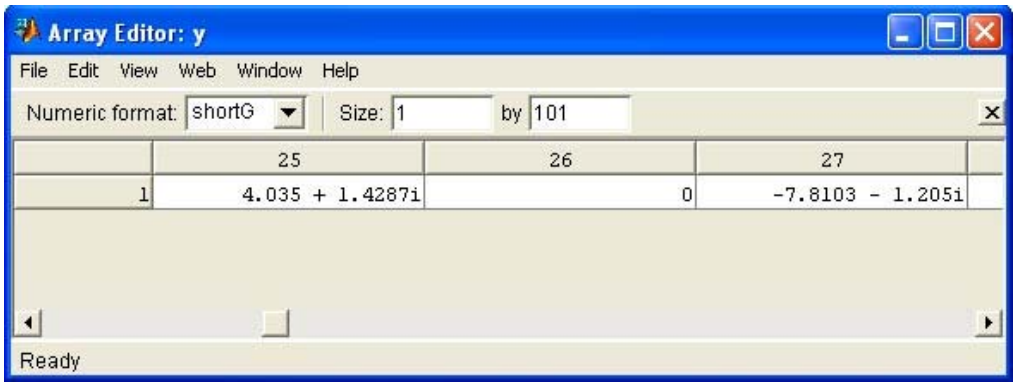
Y ahora vemos sus componentes en la FFT

```
>>y=fft(x);
>>A=abs(y)/50;
>>bar(A)
```



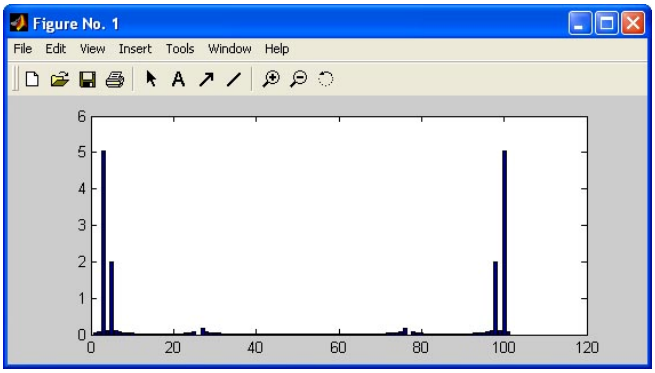
En el gráfico de barras podemos ver ahora la componente de ruido. Esa componente podrá ser eliminada ahora. La forma más directa es borrarla en la variable y.

Para ello podemos editarla dentro de Matlab haciendo doble click en ella. Entraremos en el editor de tablas, con un aspecto similar a una hoja de cálculo, y en él podemos borrar los elementos que representan el ruido, el 26 y el 77. En un caso real para "limpiar" mejor las componentes de ruido podríamos eliminar también los cercanos (24 al 28, y 75 al 79, se deja como ejercicio).



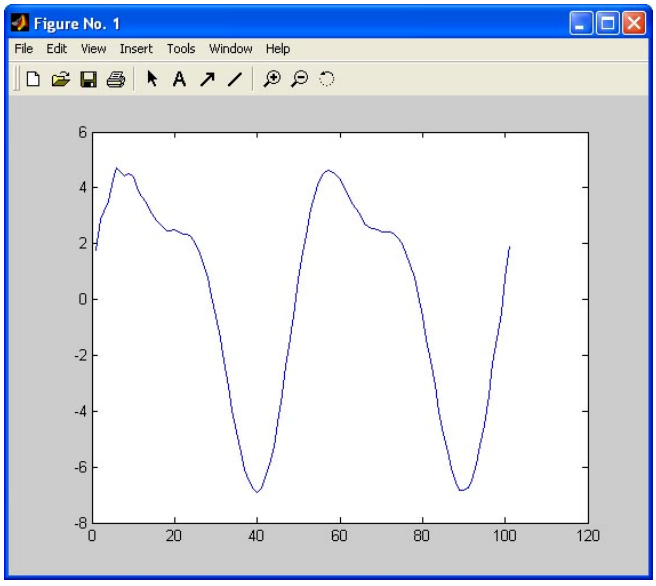
La nueva fft será ahora

```
>>A=abs(y)/50;
>>bar(A)
```



Una vez eliminados podemos invertir la transformación de Fourier con el comando ifft. El resultado, de nuevo es un número complejo. Nos interesa sólo su parte real

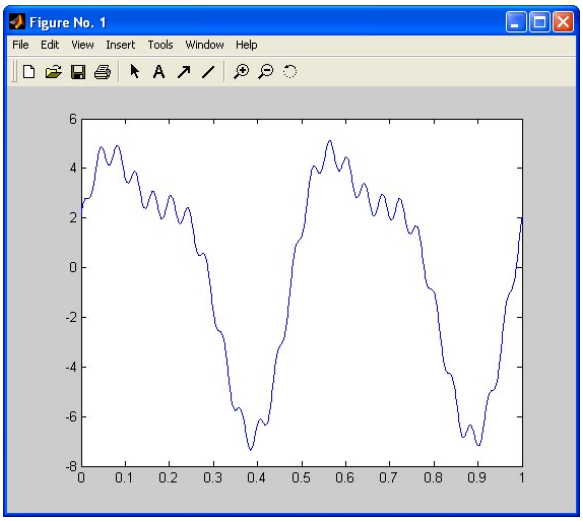
```
>>z=ifft(y);
>>plot(real(z))
```



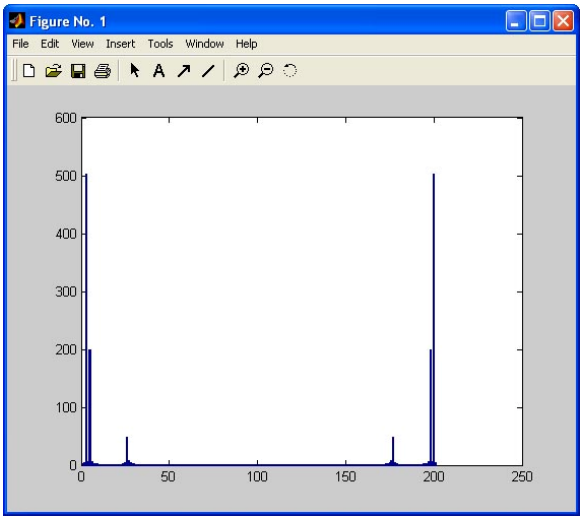
¡Hemos filtrado la señal!

Si el muestreo lo hubiésemos hecho con más puntos el resultado sería mejor, pero el obtenido es suficientemente bueno para la mayor parte de las necesidades.

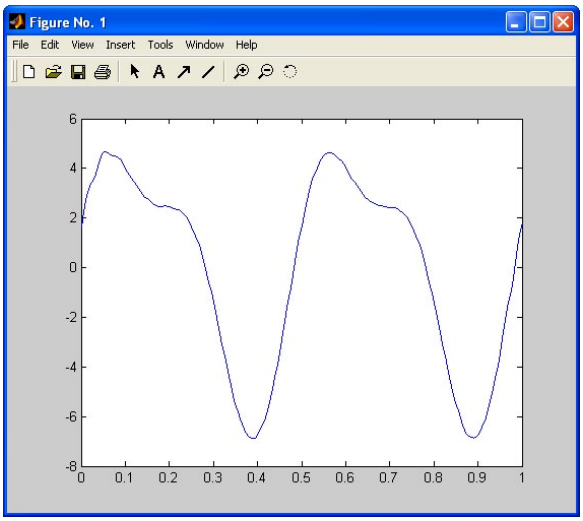
```
>>t=[0:0.005:1];
>>x1=5*sin(4*pi*t);
>>x2=2*sin(8*pi*t+1);
>>x3=0.5*sin(50*pi+1);
>>x=x1+x2+x3;
>>plot(t,x)
```




```
>>y=fft(x);
>>bar(abs(y))
```




```
>>y(24:26)=[0,0,0,0,0];
>>y(175:179)=[0,0,0,0,0];
>>z=real(ifft(y));
>>plot(z);
```

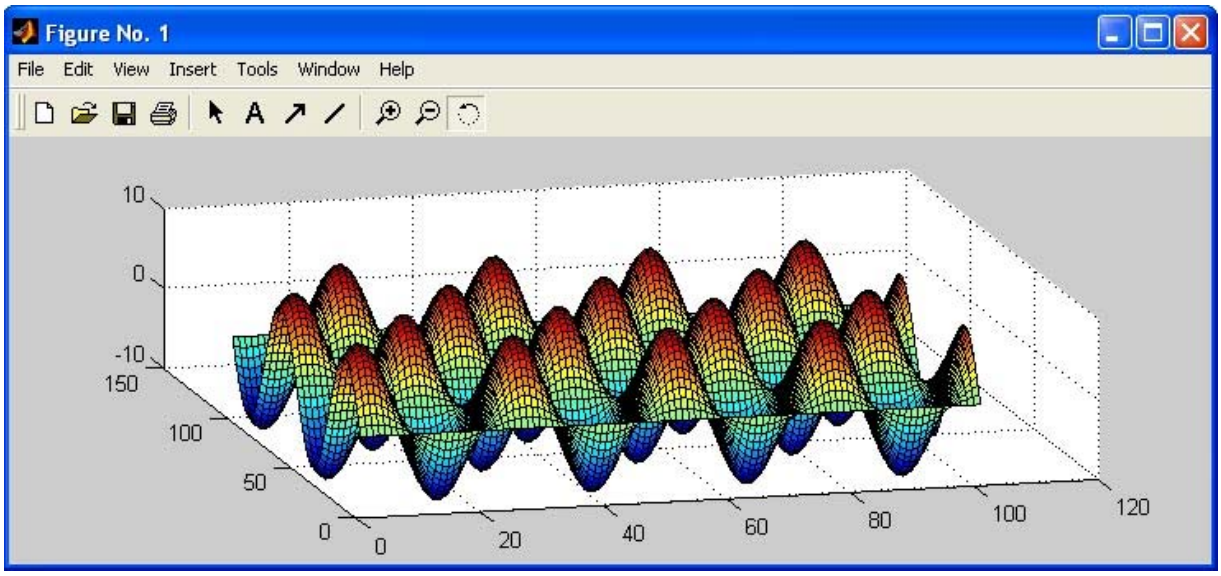


FTTs en más de una dimensión.

Haremos ahora una breve introducción a la transformación de Fourier en dos dimensiones, que no es más que una extensión de los mismos conceptos.

Al igual que cualquier señal puede representarse como suma de ondas sinusoidales, cualquier superficie puede representarse como suma de ondas sinusoidales en dos dimensiones. Dibujaremos una de estas superficies para ver de qué estamos hablando

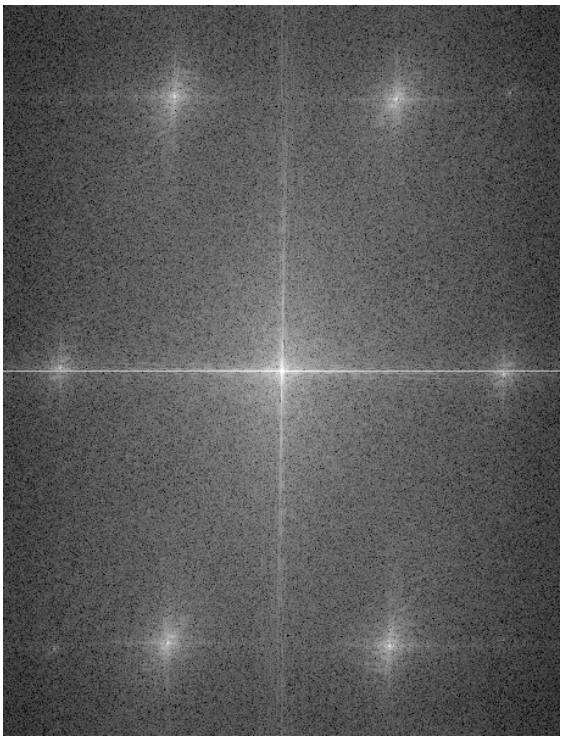
```
>>t=[0,0.01,1];
>> x1=5*sin(4*pi*t);
>> x2=2*sin(8*pi*t+1);
>> z=x1'*x2;
>> surf(z)
```



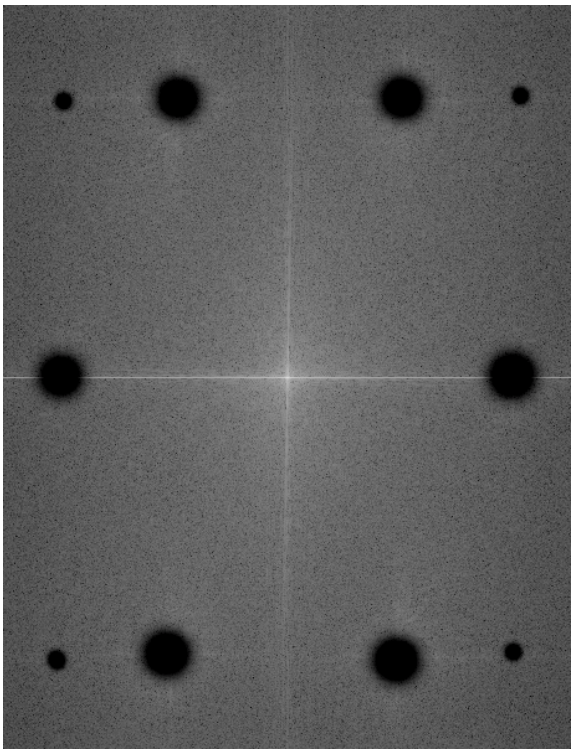
Y al igual que antes, si la superficie está dada por un tabla de datos, podrá ser transformada, haciendo ahora una FFT en dos dimensiones. Esta transformación dará como resultado otra tabla de amplitudes y fases de ondas como la del dibujo, también en 2D. Se verá ahora una aplicación con Photoshop, al que se le puede añadir un complemento (plugin) para que haga FFTs.

Partimos de una imagen que tiene una trama típica del escaneado de fotos antiguas

Haremos una FFT en 2D, la amplitud estará ahora representada en tonos de gris en una imagen 2D



Las estrellas blancas corresponden al tramado, y podemos eliminarlas pintando directamente sobre imagen de tonos de gris



Si invertimos ahora la transformación de Fourier, el resultado es el siguiente, en donde la trama ha desaparecido. Sin entrar en los detalles matemáticos, en realidad hemos hecho la misma operación de filtrado