Introducción al análisis frecuencial de señales

1. OBJETIVO

Realizar un acercamiento a los conceptos fundamentales en el análisis frecuencial de señales, y su aplicación en el análisis de bioseñales.

2. MATERIALES Y EQUIPOS

Para la realización de esta práctica se requiere del paquete computacional de Python.

3. PROCEDIMIENTO

Los ítems descritos a continuación deben ser desarrollados y entregados por los equipos de trabajo.

3.1 Señales en tiempo discreto

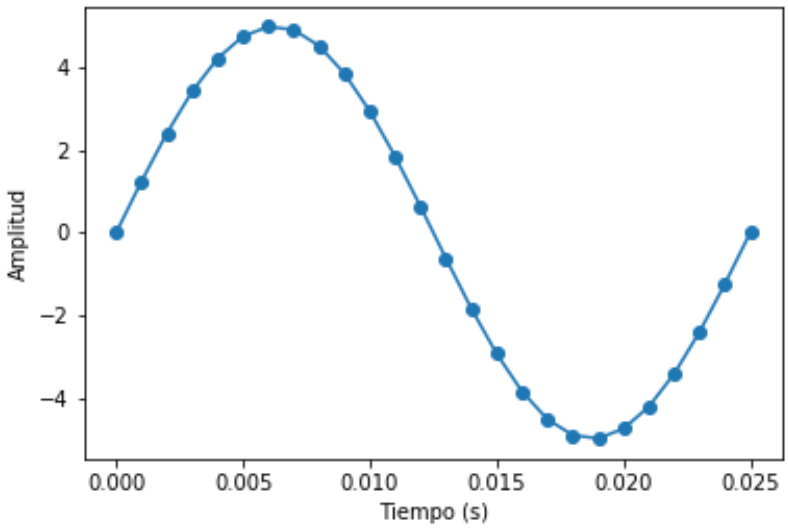
La mayoría de las señales de origen fisiológico son de naturaleza analógica, en tiempo continuo, aunque para su análisis por computador han de ser digitalizadas y convertidas a señales en tiempo discreto de valores cuantificados o discretos. Si queremos digitalizar la señal analógica $x_a(t)$ para ser procesada y analizada en computador debemos tomar muestras representativas de la señal continua a través del muestreo de la señal.
Muestrear una señal continua (analógica) consiste en remplazar la señal por sus valores en un conjunto de puntos discretos. Comúnmente, estos instantes de muestreo se distribuyen en intervalos regulares de tiempo, llamado muestreo periódico. Podemos describir esta operación matemáticamente como $x(n) = x_a(nT)$, $-\infty \leq n \leq \infty$, donde $x(n)$ es la señal en tiempo discreto obtenida tomando muestras de la señal analógica $x_a(t)$ cada T segundos. El intervalo de tiempo T entre muestras sucesivas se llama período de muestreo o intervalo de muestreo y su inverso $\frac{1}{T} = F_s$ se denomina frecuencia de muestreo (en muestras por segundo o Hz).

3.2 Representación de señales

Como ejemplo de señal discreta utilizaremos la señal analógica $x_a(t) = Asin(2\pi F_0 t)$ con $F_0 = 40Hz$ (frecuencia fundamental de la señal), $A = 5V$ (amplitud), y muestreada a $F_s = 1000Hz$ ($T = 0.001s$). Definamos estos parámetros en Python y tomemos ***un ciclo de la señal muestreada***:

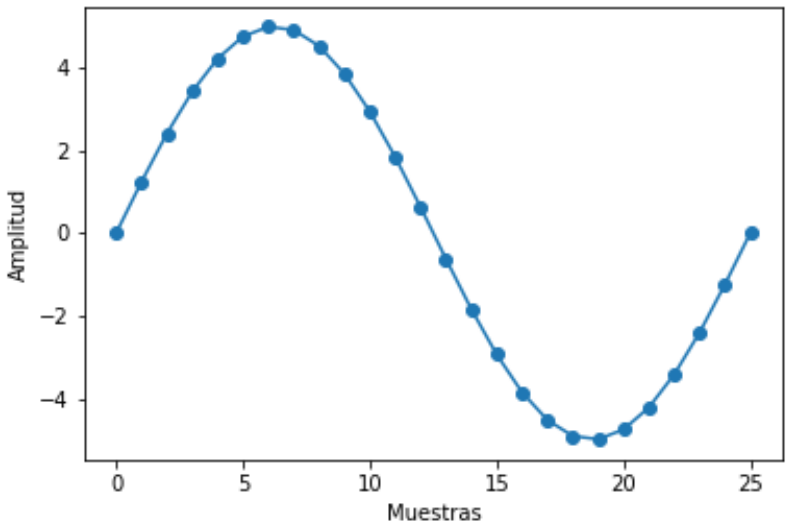
```
import matplotlib.pyplot as plt
import numpy as np

Fo = 40 # Frecuencia fundamental de la señal
Tp = 1/Fo # Periodo de la señal
```

```
Fs = 1000 # Frecuencia de muestreo
T = 1/Fs # Periodo de muestreo
t = np.arange(0, Tp+T, T) # Tiempo para un ciclo de la señal con duración de Tp más
una muestra T
A = 5
x = A*np.sin(2*np.pi*Fo*t)
plt.plot(t, x, marker='o')
plt.xlabel('Tiempo (s)')
plt.ylabel('Amplitud')
plt.show()
```



Se observa claramente que el periodo de la señal analógica o en tiempo continuo es $T_p = 0{,}025s$. Una manera alternativa de crear la señal $x(n)$ sería:

```
fo = Fo/Fs
n = np.arange(0, len(t))
x1 = A*np.sin(2*np.pi*fo*n)
plt.plot(n, x1, marker='o')
plt.xlabel('Muestras')
plt.ylabel('Amplitud')
plt.show()
```



En este caso se observa claramente que el periodo de la señal en tiempo discreto es $N = 25$ muestras. La señal $x_1(n)$ se ha generado mediante la pulsación de oscilación en tiempo discreto $f_0 = 0{,}04$ muestras/s. La única diferencia respecto al anterior es que en este último caso el período de muestreo T "manipula" la pulsación original F_0 y se considera como eje temporal la sucesión

de números enteros correspondiente a los instantes de muestreo.

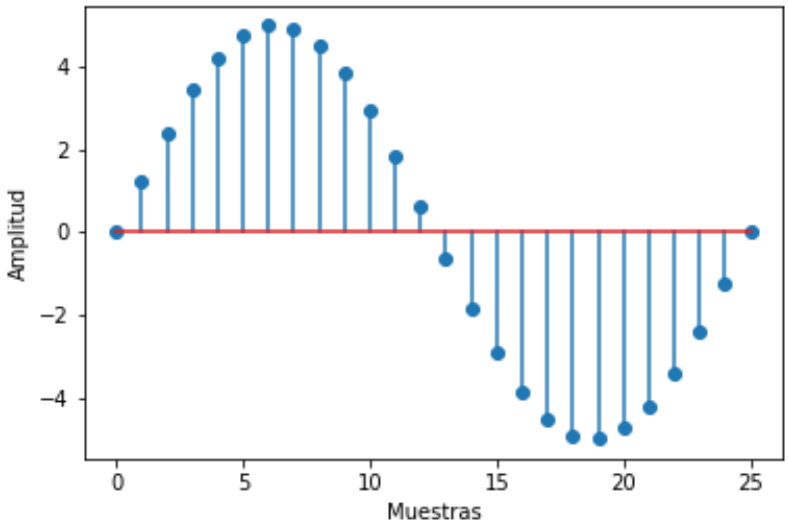
En todo caso, el resultado es el mismo, pues gráficamente se observa que ambas señales $x(t)$ y $x_1(n)$ coinciden, la única diferencia radica en que la evolución de las muestras se escala con la abscisa de tiempo en segundos $t \in$ R en el primer caso, y con la abscisa de número de muestras $n \in$ N en el segundo caso.

a) Compruebe que los valores de las primeras cinco muestras son los mismos

.

Otra manera de graficar las señales muestreadas es mediante la función stem:

```
plt.stem(n, x1, marker='o')
plt.xlabel('Muestras')
plt.ylabel('Amplitud')
plt.show()
```



De todas las opciones de las que disponemos para visualizar en Python, en esta guía utilizaremos la primera, pues brinda información sobre lo que sucede en el mundo analógico de forma rápida y directa. De esta manera podemos pensar claramente en la señal analizada $x_a(t)$ y cómo evoluciona en tiempo continuo, a pesar de que esté muestreada. La conversión mental de uno a otro formato ha de ser clara para el estudiante. Además, en señales no trigonométricas, la visualización mediante el comando stem resultaría muy engorrosa y poco manejable.

3.3 Energía y potencia en el dominio del tiempo

La energía, medida en V^2 ó $W.s$ en un ciclo de la señal $x(n)$ se puede hallar de la siguiente manera:

```
energia = sum(x**2)
```

La potencia media de la señal medida en W es:

```
potencia = energia/(len(t)-1)
```

El valor cuadrático medio es la raíz cuadrada de la potencia:

```
rms = np.sqrt(potencia)
```

Para hallar la energía, así como la potencia media de la señal considerando 10 ciclos de la sinusoide (250ms):

```
t10 = np.arange(0,10*Tp+T,T)
x10 = A*np.sin(2*np.pi*Fo*t10)
energia10 = sum(x10**2)
```

```
potencia10 = energia10/(len(t10)-1)
rms10 = np.sqrt(potencia10)
```

b) La potencia es la misma que en $x(n)$. ¿Por qué?

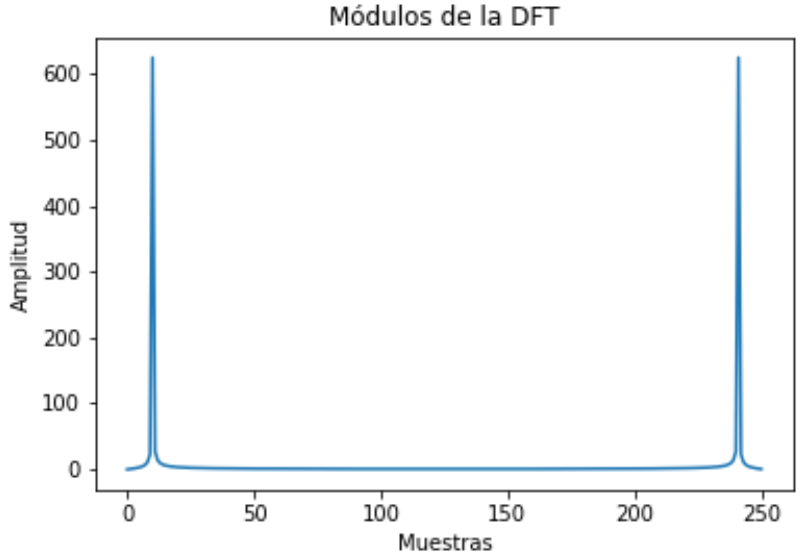
3.4 Análisis de Fourier en tiempo discreto

La transformada Discreta de Fourier (o de forma abreviada, DFT, por sus siglas en inglés) de una señal $x(n)$ se calcula en Python mediante el comando *np.fft.fft* (ver el help de Python). Vamos a calcular la DFT de la señal

```
X10 = np.fft.fft(x10)
```

Se puede observar que la DFT obtenida tiene el mismo número de puntos como muestras tiene la señal $X_{10}(n)$, 250 en total. El cálculo eficiente de la DFT se realiza mediante el algoritmo de la *Fast Fourier Transform* o FFT, cuyo comando en Python acabamos de utilizar, podemos representar los resultados:

```
plt.plot(abs(X10))
plt.title('Módulos de la DFT')
plt.xlabel('Muestras')
plt.ylabel('Amplitud')
plt.show()
```



Podemos observar que efectivamente la DFT tiene 250 puntos y que su módulo presenta simetría circularmente par (porque la señal $x(n)$ toma valores reales únicamente). También vemos que el valor máximo que toman los coeficientes es de 625.

Debemos tener en cuenta que la DFT $X(k)$ está relacionada con los coeficientes c_k de la serie de Fourier de la señal $x_p(n)$, siendo $x_p(n)$ la que se obtendría por repetición de $x(n)$. Como $x(n)$ contiene un número entero de periodos, $x_p(n)$ es la señal sinusoidal originada de duración infinita, y entonces se cumple la relación $X(k) = N.c_k$.

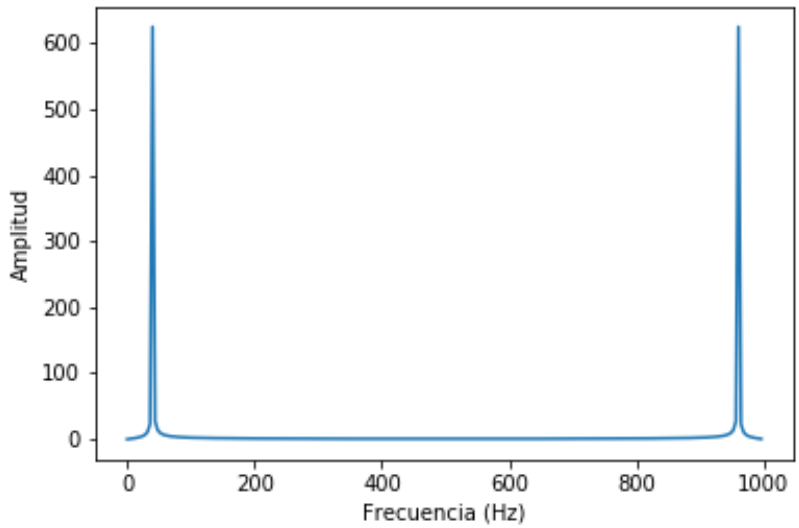
Es decir, hay un escalado por *N* (números de muestras, sin contabilizar los ceros añadidos que pudiera haber). La señal sinusoidal de amplitud unitaria tiene unos coeficientes de Fourier asociados a la frecuencia de oscilación ${c_k}' = \frac{1}{2}j$ es decir de módulo $0{,}5$. Hay que tener en cuenta que la amplitud de $x(n)$ es 5 y por lo tanto los coeficientes de la señal $x_p(n)$ serán $5 \cdot 0{,}5 = 2{,}5$. Entonces, $X(k) = N.c_k = 250 \cdot 2{,}5 = 625$ justo en la frecuencia de oscilación.

Como la señal $x_{10}(n)$ con $0{,}25s$ de duración fue muestreada a $F_s = 1000Hz$ entonces los índices que representan los 250 valores de Fourier k de $X(k)$ deben escalarse para llevarlos a escala de

frecuencia f. Para ello simplemente hallamos $f = \frac{k}{N}$, siendo $k = 0 \dots N$.

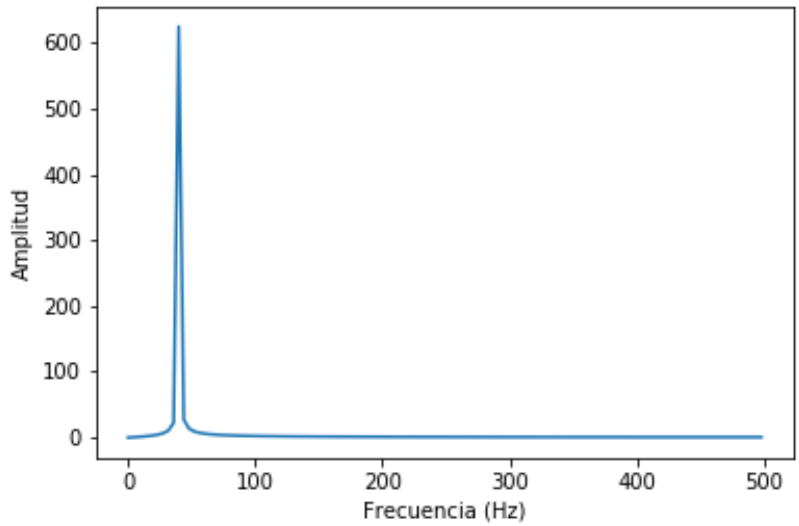
Sin embargo, si lo que nos interesa es relacionar directamente la DFT con las frecuencias F de tiempo continuo para extraer información y conclusiones de la señal original $x_a(t)$, deberemos trabajar con $F = \frac{(k.Fs)}{N}$.

```
N = len(X10)
F = np.arange(0,N)*Fs/N
plt.plot(F,abs(X10))
plt.xlabel('Frecuencia (Hz)')
plt.ylabel('Amplitud')
plt.show()
```



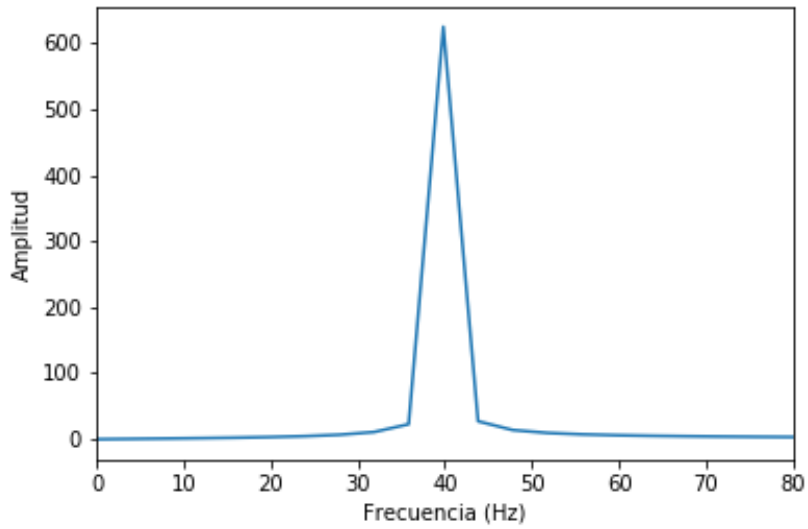
Aunque el comando de Python genera la DFT completa, habitualmente sólo se manipula y grafica la mitad de los puntos, los cuales van hasta la mitad de la frecuencia de muestreo, ya que por simetría la otra mitad se deduce fácilmente. Este valor "mitad" de la frecuencia de muestreo también recibe el nombre de frecuencia Nyquist.

```
Nmitad = int(np.ceil(N/2))
Fmitad = np.arange(0,Nmitad)*Fs/N
X10mitad = X10[0:Nmitad]
plt.plot(Fmitad,abs(X10mitad))
plt.ylabel('Amplitud')
plt.xlabel('Frecuencia (Hz)')
plt.show()
```



Realizando un zoom se observa que la componente frecuencial se encuentra ubicada a 40 Hz.

```
plt.plot(Fmitad,abs(X10mitad))
plt.ylabel('Amplitud')
plt.xlabel('Frecuencia (Hz)')
plt.xlim(0,80)
plt.show()
```



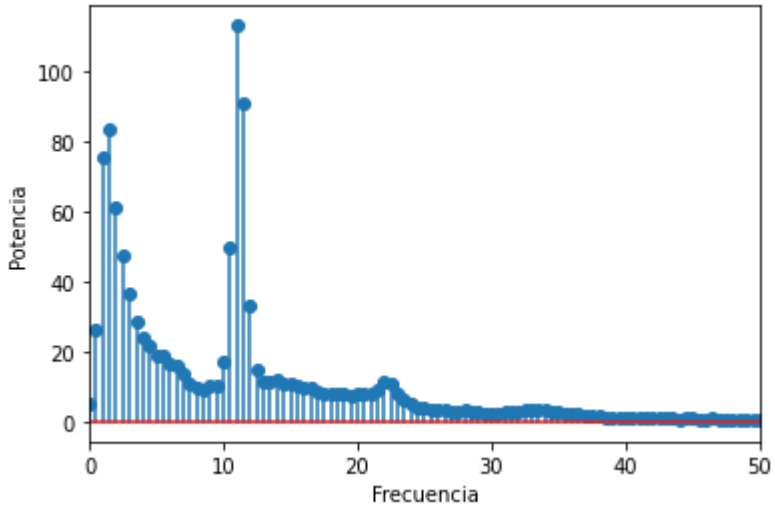
3.5 Periodogramas

El análisis de frecuencia de señales discretas y finitas también puede llevarse a acabo a través de periodogramas que se calculan usando la transformada discreta de Fourier y una ventana corta centrada en un valor específico para estimar el contenido de potencia de las diferentes frecuencias en ese intervalo. Si la ventana es rectangular se habla de periodograma y si la ventana es cualquier otra se habla de periodograma modificado.

Sin embargo, estimar el contenido de frecuencia en una señal de larga duración, a través de una sola ventana de corta duración, puede no dar cuenta real del comportamiento de la señal. Como alternativa se tiene lo periodogramas por el método de Welch donde se calcula la densidad espectral de potencia en una ventana que se desplaza a lo largo de la señal, y finalmente se promedian los espectros de potencia para obtener un periodograma de Welch.

Ejemplo en Python para una señal cualquiera

```
#@title Default title text
#importamos la rutina de welch
from scipy.signal import welch as pwelch

#%% USANDO WELCH
fs = 1000 # frecuencia de muestreo
nperseg = 2000 #ancho de la ventana
noverlap = int(nperseg/2) #solapamiento de las ventanas
f, Pxx = pwelch(senal[0,:], fs, 'hanning', nperseg, noverlap)
#graficamos frecuencia vs potencia
plt.stem(f, Pxx)
plt.xlabel('Frecuencia')
plt.ylabel('Potencia')
plt.xlim([0, 50]) #rango para el eje X
plt.show()
```



4. APLICACIÓN (100%)

Los ítems descritos a continuación deben ser desarrollados y entregados por los equipos de trabajo. Adjuntar conclusiones y referencias. Recuerde entregar el archivo ipynb, con el desarrollo punto a punto de esta aplicación, incluyendo los enunciados.

4.1 Cree una señal que sea la suma de tres componentes sinusoidales con frecuencias de 40, 80 y 160 Hz. Defina la frecuencia de muestreo mínima necesaria para representar la señal, y utilice la frecuencia de muestreo necesaria para representarla apropiadamente. (10%)

4.2 Calcule la transformada de Fourier de la señal y grafique el espectro de frecuencia. Identifique en el espectro las frecuencias que componen la señal (trabaje con 10 periodos de la señal). (10%)

4.3 Consulte la función que permite realizar el cálculo de la transformada inversa de Fourier. Aplíquelo a la señal anterior. (10%)

4.4 ¿Podría decir que las siguientes líneas aplican un filtro? ¿Por qué? (10%)

```
F1 = np.fft.fft(x);
F2 = np.zeros((len(F1)));
F2[9:13] = F1[9:13];
xr = np.fft.ifft(F2);
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(t, np.real(xr))
ax.set(xlabel='Tiempo (s)', ylabel='Amplitud (V)');
```

plt.show()
Extraiga de manera similar cada una de las componentes de la señal.

4.5 El archivo adjunto (senecg.mat) contiene una señal de ECG adquirida a una frecuencia de muestreo de 250 Hz. Realice un análisis en frecuencia de la señal usando periodogramas de Welch y determine si es necesario eliminar ruido, en tal caso, elimínelo de la forma que se mostró antes y compruebe que lo haya realizado, calcule la transformada inversa de Fourier de la señal filtrada y compruebe con el periodograma. (Usar una ventana Hanning, y justificar la elección del solapamiento y el ancho de la ventaja) (25%)

4.6 Realice un análisis de frecuencia a través del periodograma de welch de una señal de EEG de un paciente Sano y otra de un paciente con Parkinson de algún canal que en el proyecto 1 hayan identificado que presentaba diferencias, si no encontró diferencias, elija un par al azar. Realice un análisis comparativo de las señales según lo que le indiquen los periodogramas. (Usar una ventana Hanning, y justificar la elección del solapamiento y el ancho de la ventaja) (25%)

4.7 Conclusiones y referencias (10%)

5. ARCHIVOS ADJUNTOS O ANEXOS

- senecg.mat
- Datos de EEG ya los tienen

6. BIBLIOGRAFÍA

(1) J. G. Proakis, D.G. Manolakis, “Digital Signal Processing. Principles and Applications”, 4th ed, pag. 75-.77, Ed. Prentice Hall, 2007.