

Práctica Guiada 1: Señales y Espectros

Índice

| | |
|---|-----------|
| 1. Resumen y objetivos | 3 |
| 2. Señales periódicas y armónicos | 3 |
| 2.1. Representación de señales periódicas | 3 |
| 2.2. Armónicos de señales periódicas | 5 |
| 2.3. Acordes como suma de armónicos | 9 |
| 3. Espectro de una señal | 12 |
| 3.1. Ejemplo 1: Suma de tonos | 12 |
| 3.2. Ejemplo 2: Instrumentos musicales | 13 |
| 3.3. Ejemplo 3: Tono con ruido | 15 |
| 3.4. Ejemplo 4: Espectrograma | 18 |

1. Resumen y objetivos

En esta práctica se revisará la descomposición de una señal periódica en sus armónicos y su representación mediante Python. A continuación se explicará el concepto de espectro de una señal y se representarán varios ejemplos para su mejor comprensión. A este respecto, se emplearán señales de audio para facilitar la explicación de los conceptos relacionados con la frecuencia. Estas señales, además, tienen una representación gráfica sencilla debido a que son señales unidimensionales.

Para la realización de esta práctica se asume que el alumno ha asistido a la parte de Teoría de la asignatura, donde hemos visto las instrucciones básicas de representación de señales con Python. Como en las sesiones anteriores, se empleará el entorno de Spyder ya instalado en los ordenadores del laboratorio, aunque también puede usarse otro entorno en caso de utilizar un ordenador personal.

2. Señales periódicas y armónicos

Una señal periódica $x(t)$ es aquella que se repite cada cierto tiempo y por tanto cumple:

$$x(t) = x(t + nT) \quad \forall n \in \mathbb{Z}$$

donde T es el periodo.

El ejemplo más evidente de una señal periódica es una señal sinusoidal o tono. En general, todas las señales que responden a funciones trigonométricas son periódicas. Pero hay otras muchas señales periódicas, por ejemplo las señales digitales, que no tienen forma trigonométrica.

En este apartado vamos a dibujar con Python señales periódicas usando la librería SciPy, en concreto el módulo `scipy.signal`¹.

2.1. Representación de señales periódicas

Dos ejemplos sencillos de señales periódicas son la señal cuadrada y la triangular o en diente de sierra. El siguiente código Python dibuja estas dos señales con un periodo de 1 ms (o una frecuencia de 1 kHz), a partir de lo aprendido en la práctica anterior.

¹[Documentación del módulo `scipy.signal`.](#)

```

from scipy import signal
import matplotlib.pyplot as plt
import numpy as np

# Creando vector de tiempos para que se muestren varios ciclos
t = np.linspace(0,3,1000)*1e-3 # 3 ms máximo

# Parametros de las señales
f=1.0e3 # Frecuencia de 1 kHz
duty=0.5 # Ciclo de trabajo
width=0.5 # Ancho ascendente

# Generacion de las señales usando el modulo signal
y1=signal.square(2*np.pi*f*t,duty)
y2=signal.sawtooth(2*np.pi*f*t,width)

# Representacion grafica
plt.subplot(211)
plt.plot(t*1000,y1)
plt.ylabel('Amplitud')
plt.subplot(212)
plt.plot(t*1000,y2)

plt.xlabel('Tiempo (ms)')
plt.ylabel('Amplitud')

```

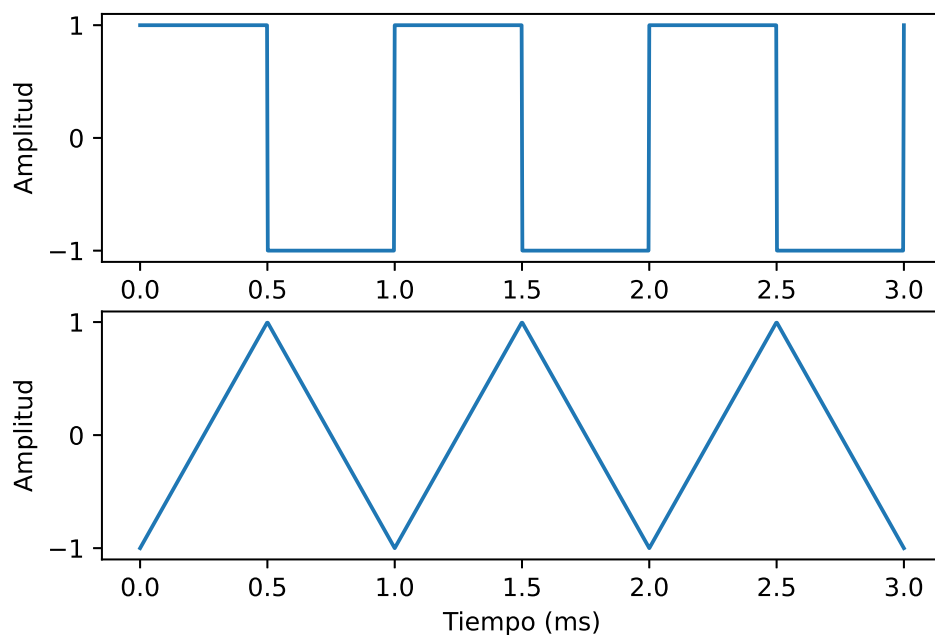



Figura 1: Señales cuadrada y triangular periódica.

El resultado obtenido se muestra en la Figura 1. Como se puede apreciar, en el código se define la variable temporal t entre 0 y 3 ms para representar 3 ciclos completos de la señal. También se definen las variables $duty$ y $width$, que son el ciclo de trabajo de la señal cuadrada y el ancho ascendente de la señal triangular. Estos parámetros se pasan como argumentos de entrada a las funciones respectivas `signal.square` y `signal.sawtooth`. Estas funciones obtienen las señales cuadrada y triangular respectivamente para el argumento ωt , del mismo modo que con una señal sinusoidal.

Ejercicio: Cambia los parámetros $duty$ y $width$ a otros valores entre 0 y 1 y observa el efecto en las señales cuadrada y triangular respectivamente. ¿Qué ocurre si los dos parámetros son 0? Si $duty=0$, la señal cuadrada está en cero siempre. Si $width=0$, la señal resultante es 

Pregunta: Si el ciclo de trabajo es de 0.5, ¿se podría generar la onda cuadrada a partir de una señal senoidal?

Se puede generar a partir de la suma de varias señales senoidales.

2.2. Armónicos de señales periódicas

Cualquier señal periódica se puede obtener como la suma de señales sinusoidales o tonos. Estos tonos presentan una frecuencia que es un múltiplo entero de la frecuencia fundamental de la señal periódica, la cual es $f_0 = 1/T$. A estos tonos se les denomina armónicos por su relación con las notas musicales. Esta teoría mediante la cual una señal temporal se puede obtener como suma de señales sinusoidales de diferentes frecuencias se le denomina análisis de Fourier.

En el caso anterior, si la señal tiene un periodo de 1 ms, la frecuencia fundamental es de $f_0 = 1$ kHz. Por tanto, la señal periódica se puede descomponer como suma de tonos de frecuencia $f = nf_0$, es decir, 1 kHz, 2 kHz, 3 kHz, ... La amplitud de estos tonos determinará la forma de la señal. A estos tonos se les denomina armónicos y el de frecuencia $f = f_0$ es el armónico fundamental. Esta suma es conocida como serie de Fourier.

En el caso de la señal cuadrada, el desarrollo en serie de Fourier es:

$$f(t) = \frac{4}{\pi} \sum_{n=1}^{\infty} \frac{\sin[(2n-1)\omega_0 t]}{2n-1} \quad (1)$$

En esta expresión ω_0 es la pulsación de la señal obtenida como $\omega_0 = 2\pi f_0$, siendo f_0 la frecuencia fundamental.

Para comprobar que esta suma de tonos proporciona la señal cuadrada de 1 kHz, vamos a escribir un código en Python que represente la suma de los primeros armónicos de la serie.

```
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
```

```

# vector de tiempos
t = np.linspace(0,2,1000)*1e-3 # 2 ms máximo

# Parametros de las señales
f=1.0e3 # Frecuencia de 1 kHz
duty=0.5 # Ciclo de trabajo
width=0.5 # Ancho ascendente

# señal cuadrada
y = signal.square(2*np.pi*f*t,duty)

# suma de tonos
y1 = (4/np.pi)*np.sin(2*np.pi*f*t) # armonico 1
y3 = y1 + (4/(3*np.pi))*np.sin(2*np.pi*3*f*t) # + armonico 3
y5 = y3 + (4/(5*np.pi))*np.sin(2*np.pi*5*f*t) # + armonico 5

# grafica
plt.plot(t*1000,y,label='Onda cuadrada')
plt.plot(t*1000,y1,label='Armónico 1')
plt.plot(t*1000,y3,label='Armónico 1 y 3')
plt.plot(t*1000,y5,label='Armónico 1, 3 y 5')

plt.xlabel('Tiempo (ms)')
plt.ylabel('Amplitud')
plt.axis([0,2,-1.5,1.5])
plt.legend()

```

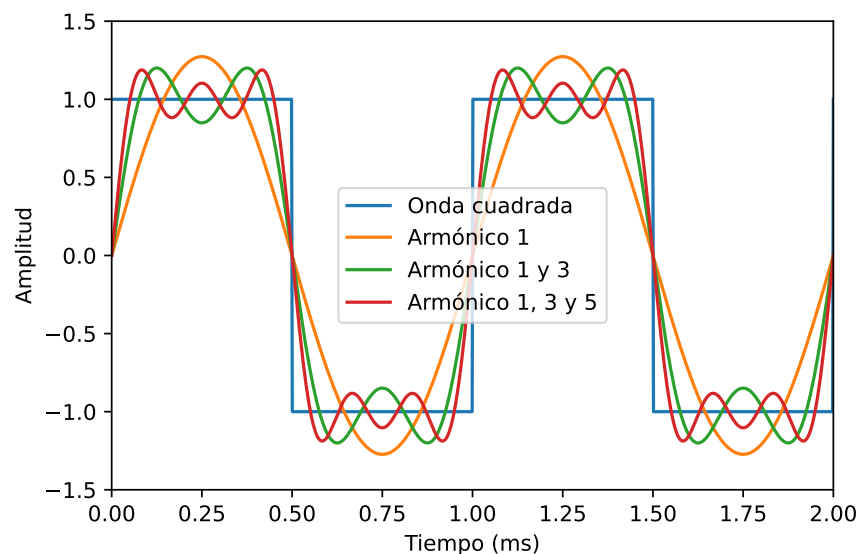


Figura 2: Suma de 3 armónicos de la señal cuadrada.

Como se puede comprobar el código calcula el primer armónico de frecuencia f_0 , al que le suma

el tercer armónico de frecuencia $3f_0$, para obtener el resultado final sumándole el quinto armónico de frecuencia $5f_0$. La amplitud de los armónicos viene dada por la expresión del desarrollo de Fourier (1). Hay que tener en cuenta que la serie sólo incluye los armónicos de orden impar de frecuencia $(2n - 1)f_0$.

El resultado de la suma progresiva de armónicos se puede ver en la Figura 2, donde también se representa la señal cuadrada. Se puede apreciar cómo, a medida que se suman armónicos, la señal se va acercando a la cuadrada, aunque 3 armónicos son muy pocos para obtener un buen resultado. Hay que tener en cuenta que la señal cuadrada presenta unos cambios muy abruptos, los cuales necesitan tonos de muy alta frecuencia.

A continuación vamos a sumar un mayor número de tonos para obtener una mejor aproximación de la señal cuadrada. Como se puede suponer, el código anterior no es eficiente para sumar un número elevado de tonos (requeriría una línea adicional por armónico), por lo que se debe recurrir a los bucles de Python. En el siguiente código se muestra la forma de sumar los 50 primeros armónicos de forma efectiva usando un bucle.

```
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np

# vector de tiempos
t = np.linspace(0,2,1000)*1e-3 # 2 ms máximo

# Parametros de la señal cuadrada
f=1.0e3 # Frecuencia de 1 kHz
duty=0.5 # Ciclo de trabajo

# senal cuadrada
y = signal.square(2*np.pi*f*t,duty)

N=50 # Numero de armonicos sumados a representar

# la variable con la suma de armonicos se inicializa a cero
y0 = np.zeros(len(t))

# Bucle con acumulador
for n in np.arange(1,N):
    y0 += 4/(np.pi)*np.sin((2*n-1)*2*np.pi*f*t)/(2*n-1)

# grafica
plt.plot(t*1000,y,label='Onda cuadrada')
plt.plot(t*1000,y0,label='Suma de armonicos')

plt.xlabel('Tiempo (ms)')
plt.ylabel('Amplitud')
```

```
plt.axis([0,2,-1.5,1.5])
plt.legend()
```

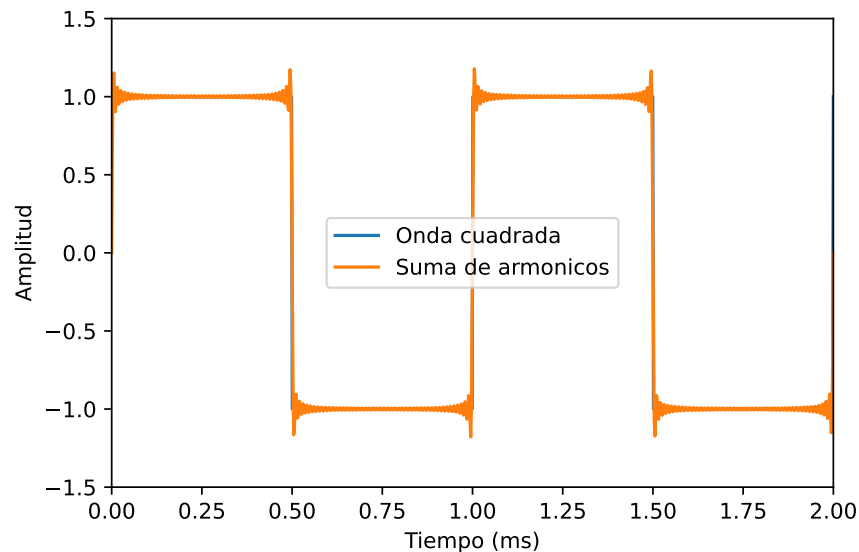


Figura 3: Suma de 50 armónicos de la señal cuadrada.

Se puede apreciar que el código emplea un bucle que recorre los números enteros de 1 a N y en cada iteración suma un nuevo armónico con la amplitud correspondiente (1). La suma se almacena en un acumulador que se inicializa a cero antes de comenzar el bucle.

El resultado se visualiza en la Figura 3. Se puede comprobar que se produce un solapamiento prácticamente perfecto entre la señal cuadrada y la suma de los 50 armónicos. La mayor discrepancia se produce en las discontinuidades (saltos) de la señal cuadrada, donde se muestran las mayores oscilaciones. Este hecho tiene cierta lógica ya que el salto presenta una pendiente infinita, la cual sólo sería replicable por un tono de frecuencia infinita. Este comportamiento se conoce como fenómeno de Gibbs.

Ejercicio: Cambia el número de armónicos N y visualiza el resultado. ¿Mejora la forma de la suma de armónicos si N aumenta? ¿Cuál es el mínimo número de armónicos para obtener una señal similar a la cuadrada? **Mayor N, más cuadrado. El mínimo es 5.**

Ejercicio: Modifica el código anterior para obtener la suma de los primeros 10 armónicos de una señal triangular y representarla gráficamente. El desarrollo en serie de Fourier de la señal triangular viene dado por la expresión (2) ¿La aproximación es mejor o peor (visualmente) que la de la señal cuadrada?

$$g(t) = -\frac{8}{\pi^2} \sum_{n=1}^{\infty} \frac{\cos[(2n-1)\omega_0 t]}{(2n-1)^2} \quad (2)$$

Es mucho mejor, se necesitan menos sumandos para que la representación sea buena.

2.3. Acordes como suma de armónicos

Como es bien conocido, las notas musicales corresponden a señales sinusoidales de una determinada frecuencia, es decir, a señales periódicas. De la misma forma, un acorde musical se puede considerar como la suma de varios tonos simultáneos, un concepto similar a lo que hemos visto en el apartado anterior.

En este apartado vamos a ver cómo generar un acorde ideal en Python, representarlo gráficamente y reproducirlo. Comenzaremos generando un tono correspondiente a la nota C4, cuya frecuencia es 261.6 Hz, y de duración 3 segundos. Como habitualmente, debemos generar un vector de tiempos con un determinado número de muestras. En este caso tomamos una frecuencia de muestreo sr de 5000 Hz, es decir, una muestra cada 0.2 ms. Hay que tener en cuenta que según el teorema de Nyquist la frecuencia de muestreo debe ser al menos del doble de la frecuencia máxima de la señal. El siguiente código muestra cómo generar el tono y representarlo gráficamente.

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

T0 = 3.0 # segundos
f0 = 261.6 # frecuencia del tono
sr = 5000 # frecuencia de muestreo

t = np.arange(0, T0, 1/sr) # vector de tiempo
x = np.sin(2*np.pi*f0*t) # señal sinusoidal

plt.figure(figsize=(15,7))
plt.subplot(211)
plt.plot(t,x)
plt.ylabel('Amplitud')

plt.subplot(212)
plt.plot(t[:200],x[:200])

plt.xlabel('Tiempo (s)')
plt.ylabel('Amplitud')
```

La Figura 4 muestra el resultado obtenido. En el eje superior se ha representado el tono en los 3 segundos de duración mientras que en el inferior únicamente se muestran las 200 primeras muestras para una mejor visualización. Recuerda que para tomar las N primeras muestras de un vector x se escribe $x[:N]$.

El tono generado se puede reproducir empleando la función `Audio` del módulo `IPython.display`, y pasándole como argumento la señal y su frecuencia de muestreo. Para ello, ejecuta lo siguiente:

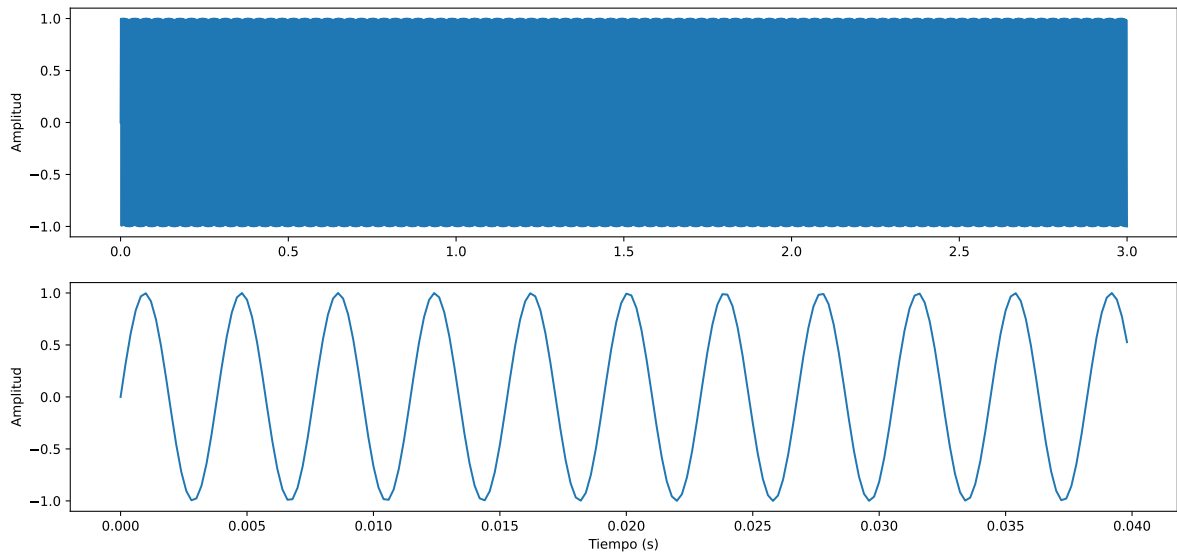


Figura 4: Tono de 261.6 Hz.

```
# libreria para reproducir audio
import IPython.display as ipd
ipd.Audio(x, rate=sr)
```

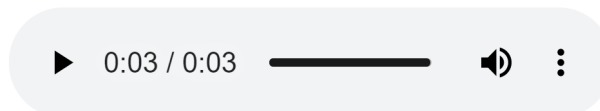


Figura 5: Reproductor de audio del módulo *IPython.display*.

Al ejecutarlo aparece un reproductor de audio integrado en el interfaz de Jupyter que permite escuchar la señal repetidas veces. Se puede apreciar que el sonido reproducido carece del timbre de un instrumento musical y suena muy artificial, ya que es un tono puro sintetizado digitalmente.

A partir de este código es muy simple generar acordes como combinación de tonos de diferentes frecuencias. A continuación se muestra cómo generar el acorde de los tonos C4 (261.6 Hz), A4 (440 Hz), C5 (523.2 Hz) y A5 (880 Hz), representarlo y reproducirlo.

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

T0 = 3.0 # segundos
sr = 5000 # frecuencia de muestreo

t = np.arange(0, T0, 1/sr) # vector de tiempo
```

```

x_c4 = np.sin(2*np.pi*261.6*t) # tono C4
x_a4 = np.sin(2*np.pi*440*t) # tono A4
x_c5 = np.sin(2*np.pi*523.2*t) # tono C5
x_a5 = np.sin(2*np.pi*880*t) # tono A5

chord = x_c4 + x_a4 + x_c5 + x_a5 # acorde

plt.figure(figsize=(15,5))
plt.plot(t[:200],chord[:200])

plt.xlabel('Tiempo (s)')
plt.ylabel('Amplitud')

import IPython.display as ipd
ipd.Audio(chord, rate=sr)

```

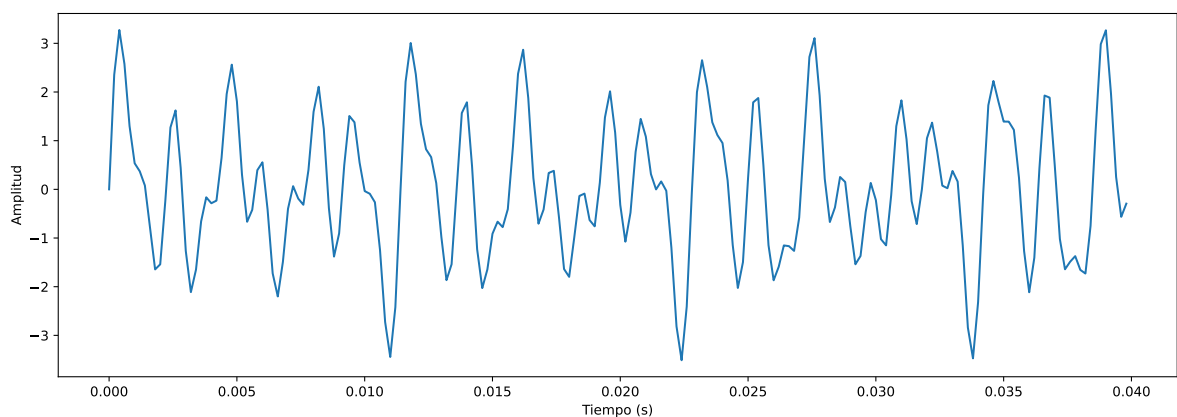


Figura 6: Acorde C4-A4-C5-A5.

Como se puede apreciar en la Figura 6, el acorde es una suma de tonos de diferentes frecuencias. La señal estaría más definida si se escogiera una frecuencia de muestreo mayor, aunque no es necesario para su reproducción.

Ejercicio: Escucha el acorde y compáralo con el tono. Se puede comprobar que el acorde es más rico musicalmente y está compuesto por la suma de varias notas.

Ejercicio: Prueba a cambiar la amplitud de cada tono en el acorde multiplicándolo por una constante y escucha cómo cambia el resultado. De esta forma se le puede dar más énfasis a las notas de alta o de baja frecuencia.

3. Espectro de una señal

Como se ha ido introduciendo en los puntos anteriores, una señal puede definirse a partir de las componentes frecuenciales que la forman. A la función que proporciona la amplitud (y fase) de las componentes en frecuencia de la señal se le denomina *espectro*. Si la señal es periódica el espectro son unas líneas espectrales discretas a múltiplos de la frecuencia fundamental. La base teórica del espectro de una señal se estudiará en asignaturas posteriores durante el grado. Aquí vamos a ver algunos ejemplos para entender conceptualmente su utilidad.

3.1. Ejemplo 1: Suma de tonos

En Python es muy sencillo obtener el espectro de una señal empleando la función denominada `magnitude_spectrum` del módulo `matplotlib.pyplot`². Como ejemplo representamos el espectro del acorde generado en el apartado anterior, mediante esta simple instrucción:

```
plt.magnitude_spectrum(chord, Fs=sr)
```

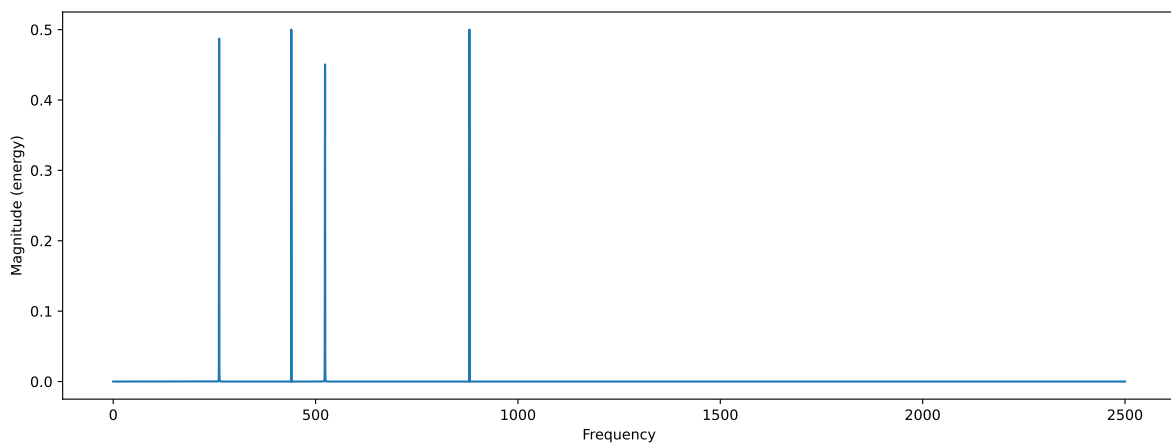


Figura 7: *Espectro del acorde C4-A4-C5-A5.*

El resultado se puede apreciar en la Figura 7. Como era de esperar, la señal está formada principalmente por 4 componentes frecuenciales correspondientes a las notas que componen el acorde. La contribución del resto de frecuencias a la señal es prácticamente nula. Se puede comprobar que el eje de abscisas llega hasta 2500 Hz, ya que ésta es la frecuencia máxima para cumplir el criterio de Nyquist.

Aunque en la gráfica de la Figura 7 se puede intuir la posición de las componentes frecuenciales, no es posible medirlas con precisión. Para ello, se puede recurrir a una librería de representación

²[Documentación de la función `matplotlib.pyplot.magnitude_spectrum`.](#)

gráfica más avanzada como Plotly³. Si no se encuentra instalada, esta librería se puede buscar e instalar desde el menú Environments en Anaconda Navigator. Ejecutamos el siguiente código para obtener el nuevo gráfico:

```
spec = plt.magnitude_spectrum(chord, Fs=sr) # cálculo del espectro

import plotly.express as px
px.line(x =spec[1], y=spec[0], labels={'x':'Frecuencia [Hz]','y':'Amplitud'})
```

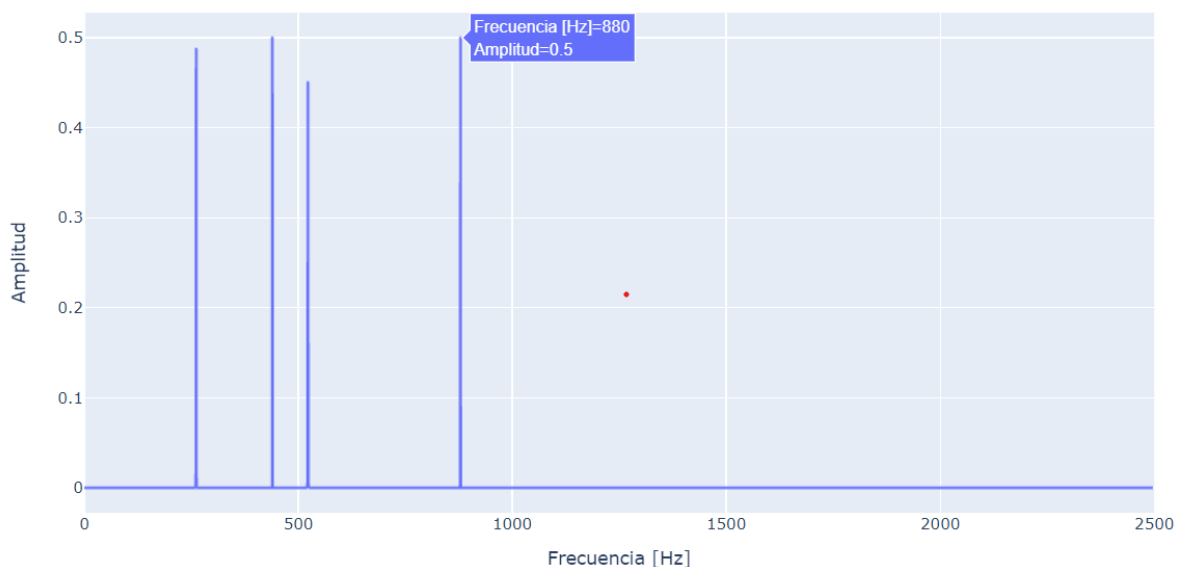


Figura 8: Gráfico interactivo con el espectro del acorde C4-A4-C5-A5.

La gráfica obtenida se muestra en la Figura 8. Para su representación se ha obtenido el espectro como previamente y almacenado su valor en la variable `spec`. A continuación se ha empleado la función `line` del módulo `plotly.express` pasándole como argumento el espectro (primer elemento de `spec`) y la frecuencia (segundo elemento de `spec`). En la gráfica generada se puede emplear el ratón para leer los valores de forma interactiva.

Ejercicio: A partir de la gráfica anterior, obtén las frecuencias de las cuatro componentes frecuenciales y su amplitud. Deberían corresponderse a los tonos que forman el acorde.

3.2. Ejemplo 2: Instrumentos musicales

En Python también es posible cargar un fichero de audio `.wav` y representar la señal temporal de sonido o su espectro. Para ello se va a usar el módulo `scipy.io`⁴ y la clase `wavfile`. A

³[Documentación de la librería Plotly.](#)

⁴[Documentación de la librería scipy.io.](#)

continuación se muestra un ejemplo de cómo cargar el fichero `clarinet_c6.wav`, el cual se debe copiar en el directorio de trabajo. Este fichero corresponde al sonido de un clarinete tocando la nota C6, de 1046.50 Hz.

```
import matplotlib.pyplot as plt
from scipy.io import wavfile # para leer y grabar audio
import IPython.display as ipd # para reproducir audio

AudioFile = "clarinet_c6.wav" # archivo de audio

fs, Audiodata = wavfile.read(AudioFile) # lectura del fichero de audio

print(f'Duracion = {Audiodata.shape[0]/fs} seg, Frecuencia de Muestreo = ' \
      f'{fs} muestras/seg, Wav format = {Audiodata.dtype}')

plt.figure(figsize=(15,5))
plt.plot(Audiodata) # representa la señal

ipd.Audio(AudioFile) # reproduce el fichero de audio
```

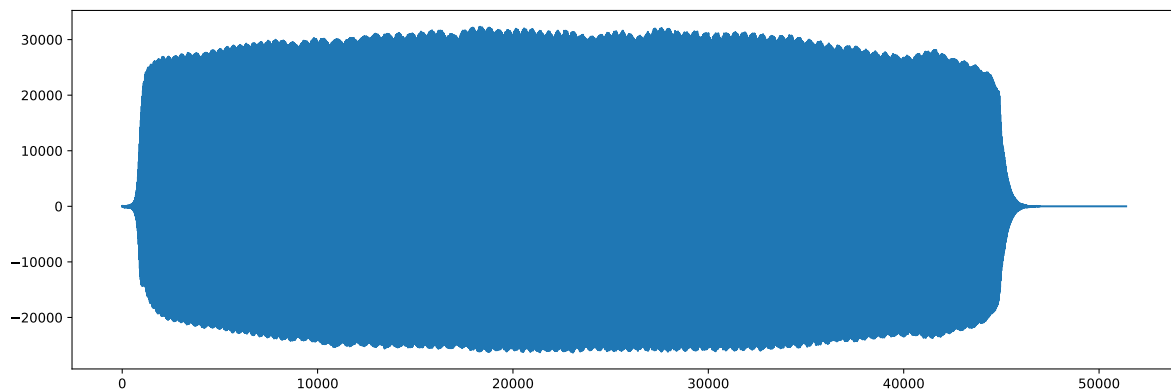


Figura 9: Nota C6 tocada por un clarinete.

Ejecuta el código y escucha el sonido. La representación de la señal temporal se muestra en la Figura 9, donde el eje de abscisas es el número de muestra. Como se aprecia en el código, la función `wavfile.read` devuelve la frecuencia de muestreo y las muestras de audio a partir del nombre del fichero. Con esta información se puede saber la profundidad de bit y la duración del audio. Estos aspectos se tratarán en la siguiente práctica. Observa cómo la función `ipd.Audio` acepta directamente el nombre del fichero para su reproducción.

Ejercicio: Representa el espectro del audio anterior y comprueba que la componente fundamental se encuentra en 1046.50 Hz, correspondiente a la nota C6. Observa cómo, además de esa componente, aparecen una serie de armónicos que son los que definen el timbre del instrumento, haciéndolo diferente del resto. El espectro obtenido se puede encontrar en la Figura 10.

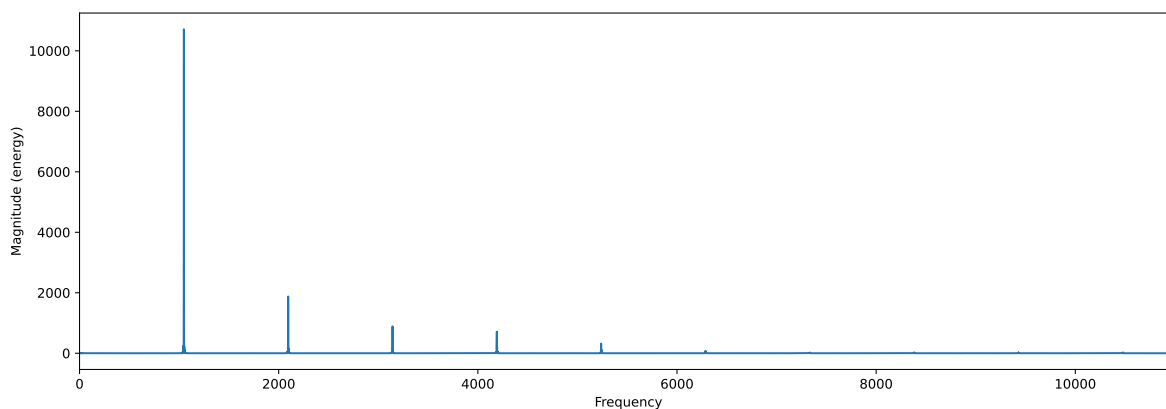


Figura 10: *Espectro de la nota C6 tocada por un clarinete.*

Ejercicio: En la Figura 10 apenas se pueden apreciar los armónicos de orden superior. Para una mejor lectura de los niveles bajos se puede emplear la escala logarítmica, representando el valor del espectro en dB. Para ello, a la función `magnitude_spectrum` se le pasa como argumento `scale='dB'`, obteniendo así la gráfica de la Figura 11.

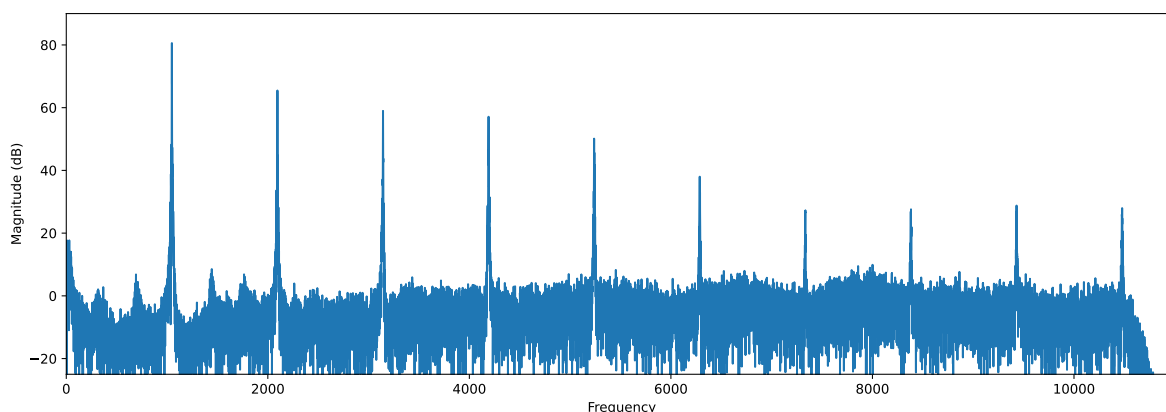


Figura 11: *Espectro en dB de la nota C6 tocada por un clarinete.*

Ejercicio: Calcula ahora el espectro del oboe también tocando la nota C6, cuyo audio se encuentra en el fichero `oboe_c6.wav`. Compara el espectro del oboe, del clarinete y de un tono puro a la frecuencia de 1046.50 Hz. En la Figura 12 se muestra un ejemplo con los tres espectros comparados. A su vez, escucha los 3 audios para comprobar las diferencias a nivel sonoro.

3.3. Ejemplo 3: Tono con ruido

El ruido es una señal aleatoria que aparece junto a la señal deseada y provoca una distorsión. Aunque existen diversos tipos de ruido, el más común es el ruido blanco, el cual presenta un espectro de amplitud uniforme en todas las frecuencias. Este nombre viene de su analogía con la luz blanca. En el contexto de una señal de audio, un ruido demasiado elevado compromete la calidad de la sensación sonora. Se debe tener en cuenta que el ruido está siempre presente

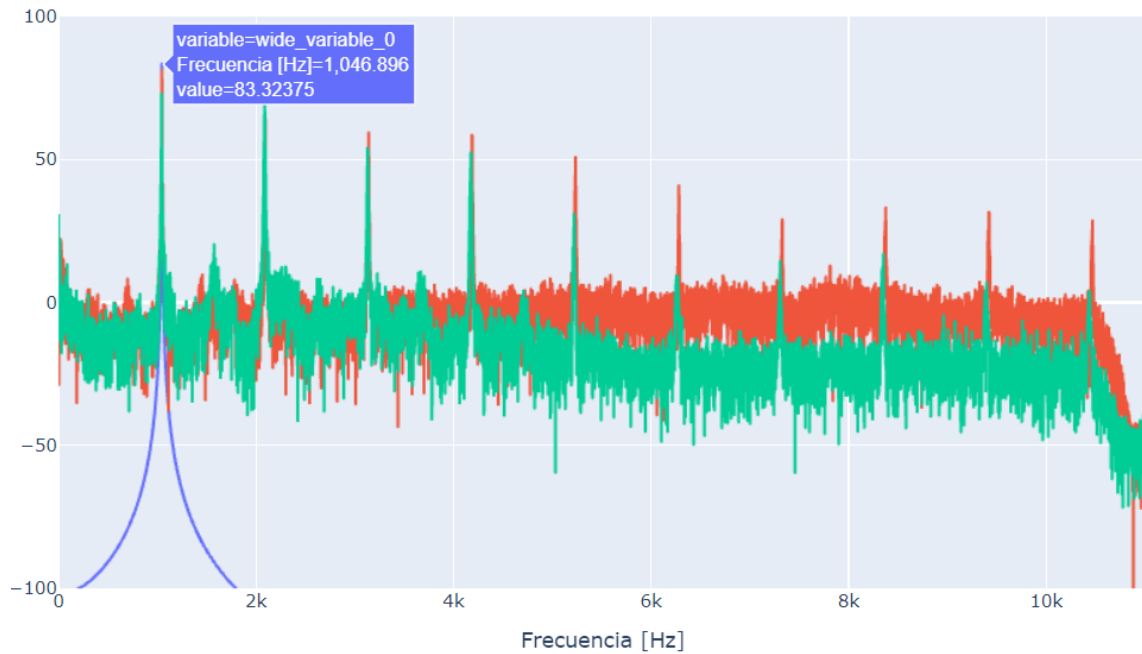


Figura 12: Comparación del espectro en dB de la nota C6 con clarinete y oboe.

en cualquier señal capturada, almacenada o reproducida, siendo muy difícil de eliminar debido a su naturaleza aleatoria. Para minimizar su impacto, se debe mejorar la calidad de los diferentes elementos que capturan y reproducen la señal: micrófonos, cables, altavoces, ...

A modo ilustrativo, en este ejemplo se va a generar un tono, al que se le va a sumar ruido blanco de una determinada potencia. Este ruido se puede generar con la función `normal` del módulo `numpy.random`, cuyos parámetros son la media del ruido (igual a cero), su desviación estándar y el tamaño de la señal. A continuación se muestra el código empleado.

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

T0 = 3.0 # segundos
sr = 20000 # frecuencia de muestreo

t = np.arange(0, T0, 1/sr) # vector de tiempo
x_a5 = np.sin(2*np.pi*880*t) # tono A5

std_noise = 0.01 # desviacion estandar del ruido (potencia de ruido)
noise=np.random.normal(0, std_noise, x_a5.shape[0]) # ruido blanco
x_a5_noise=x_a5+noise; # señal + ruido

plt.figure(figsize=(12,8))

# señal con y sin ruido
plt.subplot(211)
```



```

plt.plot(t[:200],x_a5[:200])
plt.plot(t[:200],x_a5_noise[:200])
plt.xlabel('Tiempo (s)')
plt.ylabel('Amplitud')

# espectro con y sin ruido
plt.subplot(212)
plt.magnitude_spectrum(x_a5, Fs=sr, scale='dB')
plt.magnitude_spectrum(x_a5_noise, Fs=sr, scale='dB')
plt.axis([0,2500,-100,0])

import IPython.display as ipd
ipd.Audio(x_a5_noise, rate=sr)

```

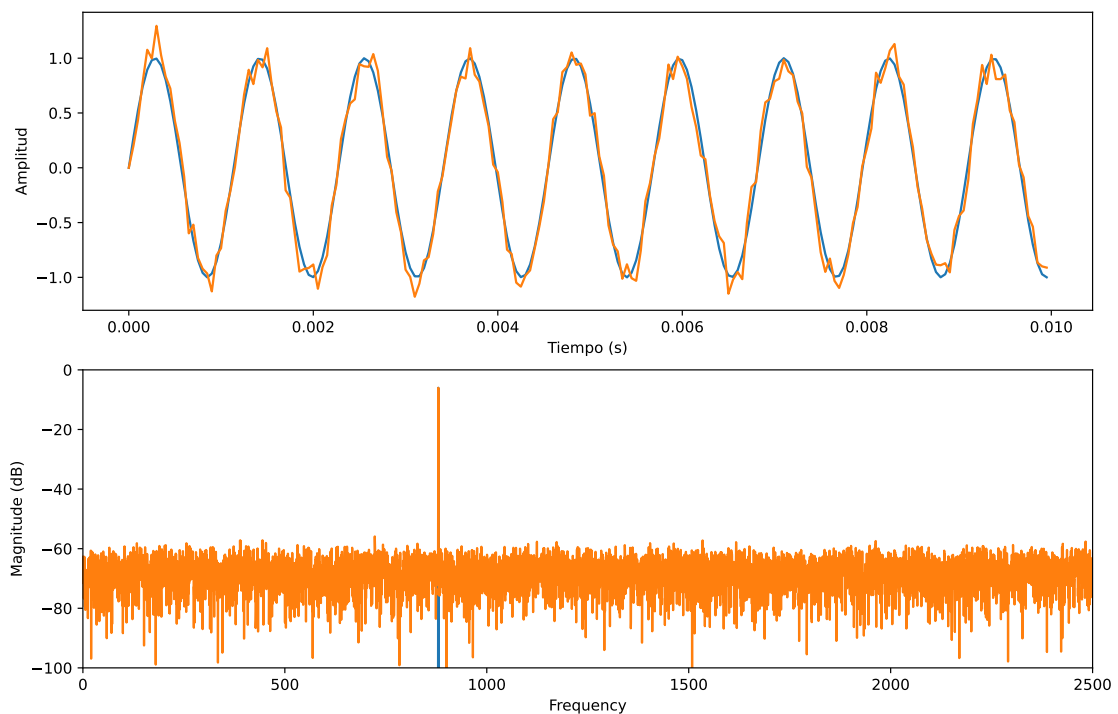


Figura 13: Tono de 880 Hz con ruido y su espectro.

Ejercicio: Escucha el audio y comprueba si se escucha el ruido de fondo. Aumenta la potencia del ruido, incrementando el valor de la variable `std_noise`, hasta que sea perceptible. Observa la diferencia entre el tono original y el tono con ruido, tanto en el tiempo como en la frecuencia.

Por ejemplo, en la Figura 13 se muestra el resultado para una desviación estándar del ruido de 0.1. En la subfigura superior se puede apreciar la variación aleatoria del ruido superpuesta al tono original. En la subfigura inferior se puede ver el espectro del tono a la frecuencia de 880 Hz, junto con el ruido presente en todas las frecuencias.

3.4. Ejemplo 4: Espectrograma

Hasta ahora hemos analizado señales que apenas cambian su comportamiento durante el tiempo, es decir, que se podría decir que las componentes frecuenciales son las mismas en toda su duración temporal. Para otro tipo de señales más complejas, existe una representación que permite mostrar cómo varía el espectro en diferentes instantes de la señal sonora. A este tipo de representación se le denomina espectrograma. En la siguiente práctica lo analizaremos con más detalle. Aquí veremos un ejemplo.

Analizaremos una señal de audio contenida en el fichero `twinkle.wav`. Copia este fichero al directorio de trabajo y reproduce el audio. Representa esta señal en el dominio del tiempo, obteniendo una gráfica similar a la de la Figura 14. Se puede apreciar que el audio es una secuencia temporal de tonos crecientes en frecuencia y luego decrecientes. Se puede hacer un zoom entre las muestras 3500 y 4500 con `plt.xlim` para observar la transición.

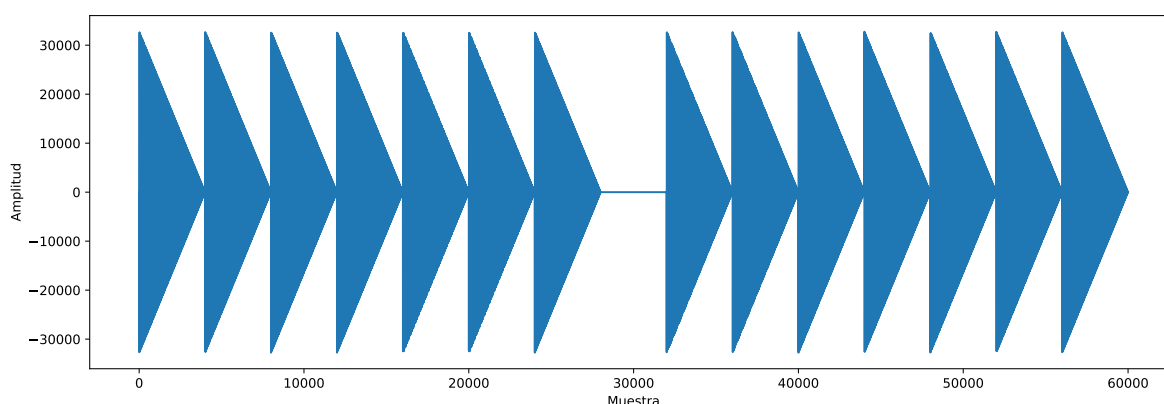


Figura 14: Señal de audio del fichero `twinkle.wav`.

Calcula el espectro de la señal de audio para obtener las componentes frecuenciales. Se obtendrá algo similar a la gráfica de la Figura 15. Como se puede observar, la señal presenta 6 componentes frecuenciales representativas, las 6 notas que se tocan en el audio. Pero obviamente, el espectro no nos aporta información acerca de la secuencia temporal de esas notas ya que se calcula a partir la señal completa.

Una representación más adecuada para apreciar la secuencia de notas es el espectrograma. Esta gráfica se obtiene mediante la función `specgram` de `matplotlib.pyplot`. Para ello, ejecuta el siguiente código una vez cargado el fichero de audio:

```
plt.figure(figsize=(15,6))
plt.specgram(Audiodata, Fs=fs); plt.colorbar(); plt.title('Espectrograma')
```

El espectrograma obtenido se muestra en la Figura 16. En él se representa con una escala de color la amplitud de las componentes frecuenciales en función del tiempo (o muestras). El

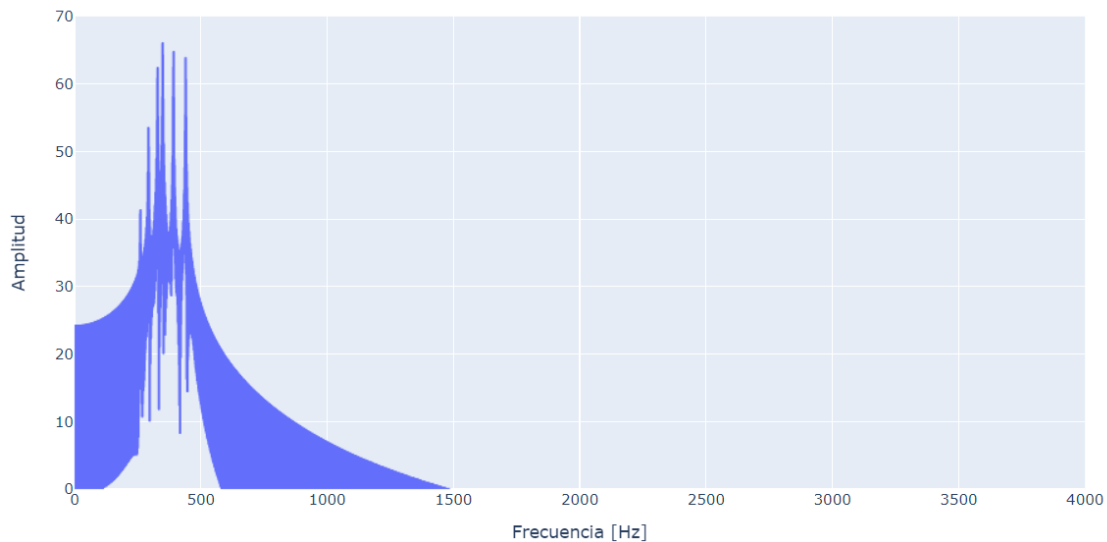


Figura 15: Espectro de la señal de audio del fichero *twinkle.wav*.

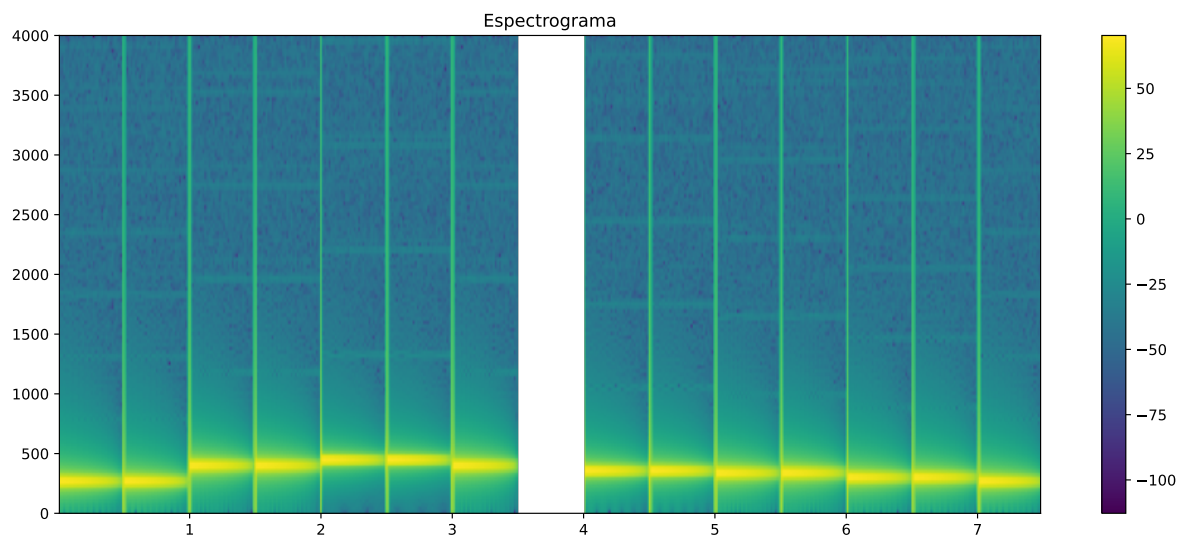


Figura 16: Espectrograma de la señal de audio del fichero *twinkle.wav*.

eje temporal es el horizontal y el frecuencial el vertical. Se puede observar cómo la frecuencia va creciendo a medida que avanza el tiempo hasta la mitad de la duración del audio, para luego decrecer. Como se puede comprobar, esta representación es muy útil para obtener de forma visual más información sobre la señal de audio.

Ejercicio: Obtener el espectrograma del fichero de audio *Cscale.wav* e interpretar el resultado. Escuchar el audio para comprobar lo deducido a partir del espectrograma.

Aquí finaliza el contenido de esta sesión de prácticas donde se han revisado conceptos relacionados con el espectro de una señal. Se recomienda al alumno salvar el cuaderno en un fichero .ipynb donde se guarden todos los ejemplos vistos en la práctica, con diferentes celdas tipo *Markdown* que sirvan para organizar el contenido.