Informática Musical

SoundDevice

Los ejercicios marcados con **(Evaluable)** deben resolverse y subirse al CV. Los archivos entregados deben contener al principio el **nombre y los apellidos de los miembros del grupo** (uno por línea).

En los siguientes ejercicios utilizaremos Notebooks de Python en cualquiera de los entornos instalados en los laboratorios (por ejemplo, VS Code con Python-anaconda). Utilizaremos también las librerías NumPy, Matplotlib y SoundDevice (si no estan instaladas, pueden instalarse con `python -m pip install ...`), así como la clase `KBHit` (para lectura no bloqueante de teclado) explicada en clase. Pueden consultarse tutoriales en:

- *NumPy*: `https://numpy.org/doc/stable/user/quickstart.html`
- *Matplotlib*: `https://matplotlib.org/stable/tutorials/index.html`
- *SoundDevice*: `https://python-sounddevice.readthedocs.io/en/0.5.1/`
- *KBHit*: `https://github.com/simondlevy/kbhit`

En la primera celda del Notebook cargaremos las librerías necesarias y definiremos algunas constantes:

```
# Nombre1 y apellidos1
# Nombre2 y apellidos2

#%%
import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt
import time # para medir tiempos de ejecución

# gráficos en el notebook
%matplotlib inline

SRATE = 441000 # Sample rate, para todo el programa
```

---

1. Comenzaremos generando una muestra de ruido blanco de una duración dada en segundos. Definiremos dos funciones:

   - `noise1(dur)`: crea un array vacío con `np.empty`. El tamaño se calcula en función de `dur` (1 segundo requiere `SRATE` muestras). A continuación rellena cada una de sus componentes con valores aleatorios del intervalo `[-1,1]` utilizando la función básica `random.random()`, que **genera valores individuales**.
   - `noise2(dur)`: genera directamente la señal en un array de tamaño adecuado utilizando la función de NumPy `random.random` o `random.uniform`, que delegan la generación del array completo directamente en NumPy.

   Dibujar la señal obtenida con Matplotlib. Para ver samples aislados puede generarse una muestra de corta duración o bien dibujar solo una parte de la señal (haciendo slicing de arrays). Asegurarse de que los valores generados están en el intervalo `[-1,1]`.

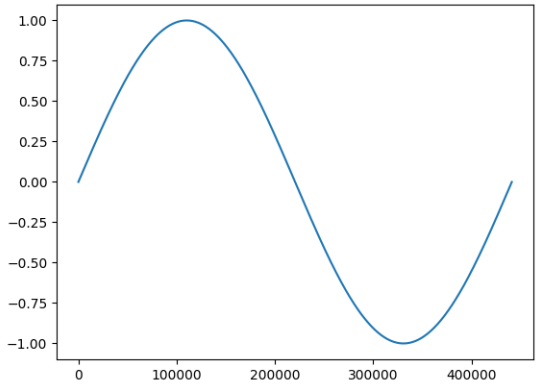
   Comparar el tiempo de ejecución de ambos métodos generando señales de suficiente duración. Para medir el tiempo de ejecución de un fragmento de código puede hacerse:

   ```
   start = time.time()
   # código ...
   print(f'time: {time.time() - start}')
   ```

   ¿Qué versión de `noise` es más eficiente?

   Generar una señal de ruido de 1 segundo de duración y reproducirla con la función `sd.play`.

2. Generar una onda sinusoidal de 1 Hz con frecuencia de muestreo `SRATE` y 1 segundo de duración, y dibujarla. Debe tener el siguiente aspecto:

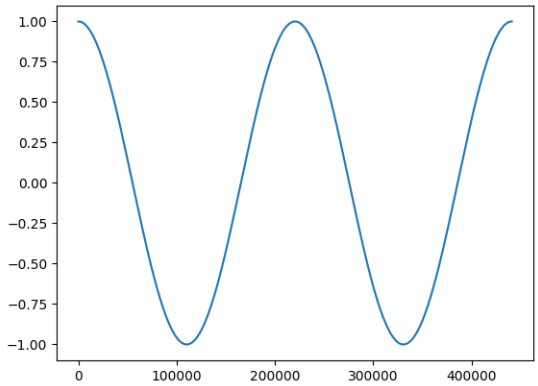


Como antes, crearemos un array de NumPy para almacenar las muestras. En este caso será útil la función `np.arange(N)` para generar el array de *soporte* y aplicar la función `np.sin` para calcular el valor de las muestras. Nótese que el instante $t = 1$ seg. corresponde al argumento 2π para la función `np.sin`.

A continuación, obtener una señal de 2 Hz y 1 segundo de duración, y otra de 3 Hz y 2 segundos de duración. Dibujar los resultados.

Generar varias señales de 1 segundo de duración con diversas frecuencias (110 Hz, 220 Hz, ...) y reproducirlas con `sd.play`. Prueba con frecuencias muy agudas y muy graves para determinar cuáles son las frecuencia mínima y máxima que puedes escuchar. Utiliza niveles moderados de volumen para no dañar tus oídos.

3. Vamos a generalizar el ejercicio anterior implementando una función `osc(freq,dur=1,amp=1, phase=0)` que devuelva una señal sinusoidal de frecuencia `freq`, duración `dur` segundos, amplitud `amp` y fase `phase` (en radianes). Por ejemplo, `osc(2,1,1,np.pi/2)` tendrá esta forma:

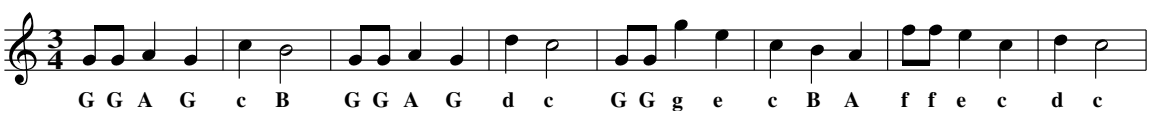


Dibujar varias señales con distintos parámetros. Prueba con diferentes valores de fase y escucha las diferencias. Probar también con diferentes valores para `SRATE`.

4. **(Evaluable)** Consideremos la partitura de la canción *Happy Birthday*:

Happy Birthday

Joe Buchanan's Scottish Tome - Page 551.3



Podemos hacer una representación de esta partitura mediante una lista (Python) de pares *(nota,duración)*. Las notas se representan con las letras `A B ...G` (la, si, do,..., sol). Para subir una octava se utilizan las minúsculas `a b ...g`[1]. La duración se representa con un número que indica las *unidades de tiempo* (la unidad se fijará en el propio programa, por ejemplo 1 segundo) De este modo, la partitura anterior quedaría:

$$[(G,0.5),(G,0.5),(A,1),(G,1),(c,1),(B,2),\ldots]$$

La tabla de frecuencias (para una octava) es la siguiente:

C	D	E	F	G	A	B	c	d ... g
523,251	587,33	659,255	698,456	783,991	880	987,767	...	...

Recordemos también que dada la frecuencia de una nota, la de la octava superior se obtiene duplicando dicha frecuencia. La de la octava inferior, dividiendo entre 2. De este modo podemos calcular la frecuencia de cualquier nota.

Implementar un programa para reproducir la partitura anterior utilizando el oscilador que acabamos de ver. Puede utilizarse un pequeño silencio al final de cada nota para diferenciar bien las notas.

Opcional: puede mejorarse aún más utilizando alguno de los generadores de ejercicios posteriores y una envolvente de volumen para cada nota.

5. Implementar una función `modulator(signal,freq)` que multiplique la señal `signal` por otra señal que hace el papel de *moduladora*. Esta moduladora será un oscilador de frecuencia `freq` y con valores en el intervalo [0,1] (no en [-1,1] como es habitual). Es decir, sube y baja el volumen de la señal entre 0 y 1.

   Utilizar esta función para modular un fragmento de ruido y señales sinusoidales de distintas frecuencias. Prueba con distintas frecuencias de modulación. Dibujar los resultados y reproducirlos con sounddevice.

   ¿Qué ocurre si dejamos que el modulador oscile en el intervalo [-1,1]?

6. **(Evaluable)** Utilizando los osciladores anteriores implementar una función `harmOsc(f,amps, dur, amp)` para simular un nota con sus armónicos $(f, 2f, 3f, \ldots, f_n)$. La función recibirá como parámetros la frecuencia fundamental `f`, una lista de amplitudes de los armónicos `amps=[a0,a1,...,an]`, la duración de la nota `dur` y la amplitud de la nota `amp`. El sonido resultante se calcula como suma de las señales sinusoidales de frecuencias $f, 2f, 3f, \ldots, f_n$ y amplitudes $a_0, a_1, \ldots, a_n$ (que se calcularán con la función osc anterior).

   Generar algunas notas con esta función con distintas configuraciones de frecuencias/amplitudes y reproducirlas. ¿Puedes encontrar combinaciones de frecuencias y amplitudes que aproximen el sonido de algún instrumento real? En ese caso, en la entrega detalla las configuraciones encontradas y los instrumentos que aproximan.

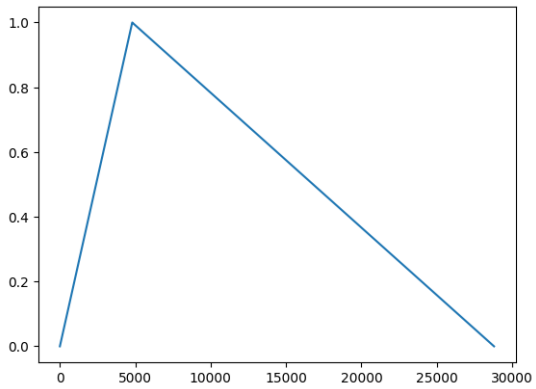
7. Con la misma idea que el oscilador `osc` que hemos visto, implementar funciones *saw*, *square*, *triangle* que obtengan señales con las formas indicadas (véase `https://en.wikipedia.org/wiki/Waveform`). Utilizar Matplotlib para dibujarlas y generar algunos sonidos con ellas.

   Exender la función `harmOsc` anterior con un argumento `shape` que permita generar notas con armónicos de las formas anteriores.

8. Implementar una función `fadeOut(sample,t)` que haga un efecto *fadeOut* con la señal `sample` desde el instante `t` hasta el final (caída lineal de volumen en tiempo `t`). Análogo con `fadeIn(sample,t)` haciendo *fadeIn* desde el inicio hasta el instante `t`.

9. Implementar una función `env(attack,release)` que genere una envolvente de volumen con un ataque de `attack` segundos y una extinción de `release` segundos. Por ejemplo, con `attack=0.1, release=0.5` debe generar esta señal:

[1] Para la siguiente octava por arriba podrían utilizarse los apostrofes (`a'`), para la siguiente doble apostrofe (`a''`), etc. Para bajar octava se podrían utilizar las comas (`A, A,,`)



Utilizando la función *osc* (o, si se prefiere, `oscs` o `harmOsc`) y esta envolvente implementar una función `playNote(freq,dur,amp,attack,decay)` que reproduzca una nota de frecuencia `freq`, duración `dur` y amplitud `amp`, con un ataque de `attack` y una extinción de `decay` (segundos).

Comprobar su correcto funcionamiento con una señal sinusoidal y una señal de ruido. Dibujar los resultados y reproducirlos con sounddevice.

10. Implementar una función `playSong(song)` que reproduzca una lista de notas como la del ejercicio 4, utilizando la función `playNote` del ejercicio anterior para reproducir cada nota.