Informática Musical

Python. Librerías NumPy y Matplotlib

Los ejercicios marcados con **(Evaluable)** deben resolverse y subirse al CV.

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 y Matplotlib (si no estan instaladas, pueden instalarse con `python -m pip install ...`). Pueden consultarse tutoriales en:

- *NumPy*: `https://numpy.org/doc/stable/user/quickstart.html`
- *Matplotlib*: `https://matplotlib.org/stable/tutorials/index.html`

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

```
#%%
import matplotlib.pyplot as plt
import numpy as np
import time

# gráficos en el notebook
%matplotlib inline

SRATE = 4410000 # 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 la función `empty` (véase ayuda de NumPy). 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 `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(N)` (genera un array de tamaño `N` con valores aleatorios).

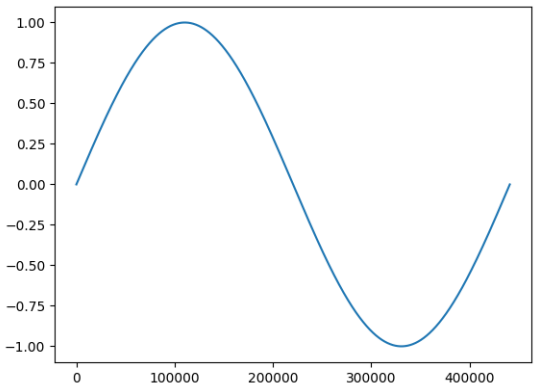
   Dibujar la señal obtenida con Matplotlib. Para ver samples aislados puede generarse una muestra de corta duración o bien dibujar solo una parte de la señal (haciendo slicing de arrays). Asegurarse de que genera valores del intervalo `[-1,1]`.

   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?

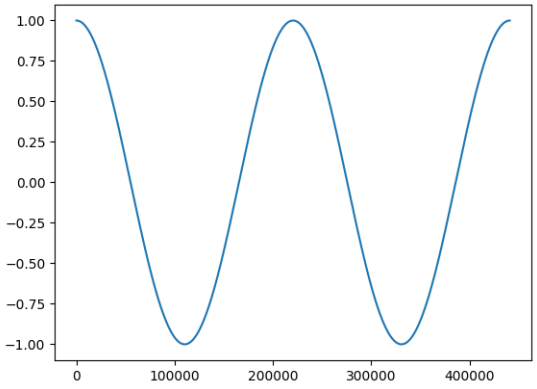
2. Generar una onda sinusoidal de 1 Hz, con el mismo `SRATE` y 1 segundo de duración, y dibujarla. Debe tener el siguiente aspecto:



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`. D

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

3. Vamos a generalizar el ejercicio anterior. Implementar una función `osc(freq,dur=1,amp=1,phase=0)` que devuelva una señal sinusoidal de frecuencia `freq`, duración `dur` segs, 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. Probar también con diferentes valores para `SRATE`.

4. **(Evaluable)** Implementar una función `modulator(sample,freq)` que multiplique la señal `sample` por un oscilador de frecuencia `freq`, que oscile en el intervalo [0,1]. Utilizar un fragmento de ruido como entrada y dibujar el resultado con un modulador de 2Hz. ¿Qué ocurre si dejamos que el modulador oscile en el intervalo [-1,1]?

5. Utilizando los osciladores anteriores implementar una función para simular un nota con sus armónicos: suma de osciladores de frecuencias $f, 2f, 3f, \ldots, nf$ (n y f dados) y amplitudes $v, v/2 \ldots v/n$ respectivamente (v dado).

6. Con la misma idea que el ejercicio anterior implementar funciones *saw*, *square*, *triangle* que obtengan señales con las formas indicadas (véase `https://en.wikipedia.org/wiki/Waveform`) y utilizar Matplotlib para dibujarlas.

7. 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`.

8. En este ejercicio vamos a implementar un oscilador sinusoidal con una filosofía diferente, generando la señal por bloques (*chunks*). Es decir, no generaremos la señal completa sino sucesivos fragmentos de la misma de tamaño prefijado. El tamaño de los bloques vendrá determinado por una variable global `BUF_SIZE` (que podemos inicializar a 1024, por ejemplo).

   Para ello implementaremos una clase `Osc`. El método constructor definirá atributos para la frecuencia, amplitud y phase. Otro método `next` devolverá el siguiente *chunk* (de tamaño `BUF_SIZE`) con las siguientes muestras de la señal.

   Concatenar varios chunks consecutivos y utilizar Matplotlib para verificar que se obtiene la señal esperada, sin saltos entre chunks.