Informática Musical

Python. Librerías NumPy y Matplotlib

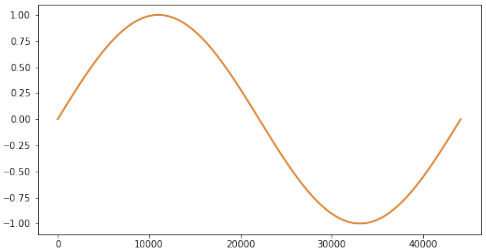
En los siguientes ejercicios utilizaremos Python3. Puede utilizarse cualquiera de los entornos de programación para Python, por ejemplo Visual Studio Code (instalado en los laboratorios de la facultad). Utilizaremos también las librerías:

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

---

1. Comenzaremos generando un segundo de ruido a 44100 Hz de frecuencia de muestreo. Para ello creamos un array de NumPy con 44100 componentes `v = np.arange(44100)` que almacenará la señal. A continuación, generamos una muestra aleatoria para cada componente, que corresponde a un instante temporal (a intervalos de 1/44100 segundos). Dibujar la señal con Matplotlib.

2. Ahora vamos a generar una onda sinusoidal de 1 Hz y 1 segundo de duración, con la misma frecuencia de muestreo, i.e., un ciclo de un segudo de esta forma:



   Como antes, crearemos un array de NumPy para almacenar las muestras. Utilizaremos la función `np.sin` para obtener esa señal teniendo en cuenta que la muestra $44100 - 1$ (los arrays se indexan desde 0) corresponde al instante $t = 1$ seg. y al argumento 2π para la función `np.sin`. Dibujar la señal con Matplotlib.

   A continuación, modificar el programa para obtener una señal de 2 Hz y 1 segundo de duración y luego para obtener una señal de 3 Hz y 2 segundos de duración. Dibujar los resultados.

3. Ahora generalizaremos aun más el ejercicio anterior. Definir una variable global *SRATE* con la frecuencia de muestreo del proyecto (por ejemplo 44100). Implementar una función *osc(f,d)* que devuelva una señal sinusoidal con frecuencia *f* y duración *d* segs. Por ejemplo, para *f=1, d=1* debe devolver un array *NumPy* de 44100 muestras con un ciclo de la función sin. Utilizar Matplotlib para dibujar la señal y probar con distintas frecuencias y duraciones. Probar también con diferentes valores para *SRATE*.

   Análogamente, implementar funciones *saw*, *square*, *triangle* que obtengan señales con las formas indicadas (véase `https://en.wikipedia.org/wiki/Waveform`) y utilizar Matplotlib para dibujarlas.

4. Implementar una función *vol(sample,vol)* que multiplique una señal dada *sample* (array de NumPy) por el volumen *factor*. Implementar otra función *modulaVol(sample,frec)* multiplique la señal dada por otro oscilador de frecuencia *frec*, que oscile en el intervalo [0,1]. ¿Qué ocurre si dejamos que oscile en el intervalo [-1,1]? Dibujar los resultados.

5. 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). Análogo con *fadeIn(sample,t)* haciendo *fadeIn* desde el inicio hasta el instante *t*.

6. 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 con 1024 por ejemplo).

Para ello implementaremos una clase *Osc*. El método constructor definirá un atributo con la frecuencia y 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.