Informática Musical

Procesamiento de audio digital: NumPy, SoundDevice, SciPy.

Los ejercicios marcados con **(Evaluable)** son prácticas de laboratorio evaluables. El ejericion 1 es obligatorio; de los restantes evaluables debe seleccionarse **al menos otro** y subir ambos al CV. Los notebooks deben ser **autocontenidos** (incorporar las librerías necesarias).

Todos los archivos entregados tienen que contener el nombre y los apellidos de los miembros del grupo (uno por línea).

---

1. (**Evaluable**) Implementar una clase `Modulator` para hacer una modulación sinusoidal de una señal dada. La idea es la misma que en el modulador del ejercicio de la hoja anterior, pero ahora el modulador se encapsulará en un clase que procesa por chunks. La constructora de esta clase será de la forma

   ```
   __init__(self,signal,freq=1,v0=0,v1=1)
   ```

   donde `signal` es la señal a modular (un objeto generador de señal, que implementa el método `next` para obtener sucesivos chunks de señal), `freq` la frecuencia de modulación y `v0,v1` los valores mínimo y máximo de la amplitud de la modulación. La clase `Modulator`, será a su vez un generador de señal, es decir implementará el método `next` que devolverá la señal modulada (un chunk en cada llamada).

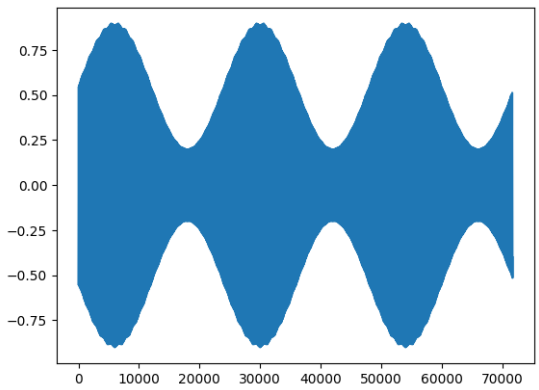
   De este modo, podremos por ejemplo moudular una señal trigular de 440 Hz con un modulador de 2 Hz y amplitud de 0.2 a 0.9:

   ```
   # señal que vamos a modular
   signal = Osc(freq=440,shape='triangle')

   # con un modulador de 2 Hz y amplitud en [0.2,0.9]
   mod = Modulator(signal,freq=2,v0=.2,v1=.9)

   # generamos 1.5 segundos de señal modulada
   time = 1.5
   chunks = int(time*SRATE/CHUNK) # número de chunks a generar
   signal = np.empty(0) # acumulador de la señal
   for i in range(chunks): # generamos los chunks
       signal = np.append(signal,mod.next())

   plt.plot(signal)
   ```

2. Extender la clase `Osc` vista en clase con los métodos *get/set* para poder obtener y modificar dinámicamente el volumen y la frecuencia del oscilador. Después, utilizaremos la librería TkInter para captura de teclas (ver ejemplo del notebook *Síntesis FM, librería TkInter, envolventes, enrutado...*) e implementaremos una interfaz para controlar el oscilador anterior: permitirá subir/bajar la frecuencia con las teclas *[f/F]* y el volumen con *[v/V]*.

3. (**Evaluable**) Extender la clase OscFM del notebook *Síntesis FM, librería TkInter, envolventes, enrutado...* con métodos get/set para poder obtener y modificar dinámicamente la frecuencia moduladora y el índice beta. Utilizando la clase librería TKinter, implementar un interfaz gráfico para controlar dinámicamente estos parámetros con las teclas *[m/M]* y *[b/B]*. Buscar algunas combinaciones de los mismos para obtener timbres realistas (o al menos interesantes).

4. En este ejercicio vamos a implementar un efecto de modulación de paneo (o balance de canales) para muestras estéreo. En primer lugar, si tenemos un array numpy con una señal estéreo, podemos obtener los canales con `left, right = bloque[:,0], bloque[:,1]`. Una vez realizadas las operaciones deseadas con cada canal, podemos recomponer la muestra estéreo con `bloque = np.column_stack((left,right))`

   El paneo es la cantidad de señal que se manda a cada canal. Hay distintas alternativas de implementación (consultar capítulo 9 de *Hack Audio: An Introduction to Computer Programming and Digital Signal Processing in MATLAB*. Eric Tarr. Routledge, 2019.)

   Una forma razonable para conseguir *equal power panning* es utilizar la fórmula *square-law panning*. En este caso, dado un valor de paneo $p \in [0,1]$, la amplitud de cada canal viene dada por:

   $$\begin{aligned} a_{left} &= \sqrt{1-p} \\ a_{right} &= \sqrt{p} \end{aligned}$$

   De este modo, con $p = 0$ irá toda la señal al canal izquierdo (nada al derecho), con $p = 1$ irá toda la señal al derecho y con $p = 0{,}5$ irá la misma señal a uno y otro. En nuestro caso, queremos modular ese coeficiente (que varíe con el tiempo). Para conseguirlo puede utilizarse la clase `Modulator` del ejercicio anterior o implementar ad-hoc dicho modulador.

5. Implementar un programa que permita grabar y reproducir a la vez utilizando la clase *Stream*, con callbacks (en el mismo método callback puede gestionarse la reproducción y la grabación). Para probarlo, tomar un wav de entrada para la reproducción y a la vez grabar otro, que se guardará en un archivo.

6. Implementar un reproductor de *wav* que *normalice* el volumen de la salida, procesando por chunks. Para cada chunk calculará el valor máximo de las muestras en valor absoluto. Con ese coeficiente es fácil calcular el máximo valor por el que puede incrementar el volumen sin saturar la señal (todas las muestras deben quedar en el rango $[-1, 1]$).

   Como mejora, el coeficiente multiplicador puede modificarse *suavemente* entre chunks para no generar cambios abruptos de dinámica en el sonido de salida.

7. **(Evaluable)** Implementar una clase `Delay` para hacer una (línea de retardo): recibe una señal de entrada y devuelve esa misma señal, pero retardada en el tiempo una cantidad prefijada de tiempo. La constructora recibirá como argumento el tiempo de retardo en segundos. Tanto la entrada como la salida serán chunks de audio e internamente deberá gestionarse un buffer para producir el retardo.

   Para probar el funcionamiento vamos a implementar un *idiotizador*[1]. Para ello utilizaremos un Stream de entrada/salida y reenviaremos la señal de entrada a la salida, con un retardo temporal utilizando la línea de retardo del ejercicio anterior.

8. (**Evaluable**) El *theremin* es un instrumento electrónico basado en osciladores, que consta de dos antenas metálicas que detectan la posición relativa de las manos del intérprete (*thereminista*). Esa posición determina la frecuencia y la amplitud del tono (la frecuencia se controla con una mano y la amplitud (volumen) con la otra). Puede verse un vídeo en `https://www.youtube.com/watch?v=K6KbEnGnymk`

   En este ejercicio se implementará un *theremin* con un oscilador básico. Las antenas se simularán con el ratón: la posición horizontal determina la frecuencia (en un rango prefijado, como [50,1500] por) y la vertical, la amplitud en [0,1].

   El instrumento debe incluir un pequeño interfaz gráfico con TkInter, que permite detectar fácilmente la posición del ratón en en pantalla:

[1] Véase `https://www.youtube.com/watch?v=zhUDQGWM8kM`

```
import tkinter as tk

WIDTH, HEIGHT = 800, 600
root = tk.Tk()
root.geometry(f"{WIDTH}x{HEIGHT}")

def motion(event):
    x = root.winfo_pointerx()
    y = root.winfo_pointery()
    print(f'{x}, {y}')

root.bind('<Motion>', motion)root.mainloop()
```

La reproducción ser hará mediante callbacks (no bloqueantes) en SoundDevice.

Extensiones:

- Utilizar otros modelos de generación, como la síntesis FM. Deberán evitarse o suavizarse los pops al cambiar la frecuencia/volumen.
- Incorporar un *autotune* para obtener solo frecuencias de un conjunto determinado (correspondientes a una escala) y utilizar la posición del ratón para aproximar la nota más cercana de ese conjunto.