# Tarea Corta I - Autrum

Curso: Redes (IC-7602)

Sede: Campus Tecnológico Central Cartago

Periodo: Primer semestre 2023

Profesor: Nereo Campos Araya

Estudiantes:


- Fiorella Arias Arias 2020023639
- Jarod Cervantes Gutiérrez 2019243821
- Esteban Ignacio Durán Vargas 2020388144
- Luis Diego Alemán Zúñiga 2018135386
- Leonardo David Fariña Ozamis 20200045272


**Índice**



## Instrucciones para ejecutar el proyecto

Para el funcionamiento correcto de este proyecto, se requiere una versión de Python 3.10.10 además de las siguientes librerías:

- io
- tkinter

- os
- pyaudio
- wave
- numpy
- matplotlib
- zipfile
- scipy
- time

Para instalar cada una de estas en el entorno de python se utilizó la herramienta pip de la siguiente manera:

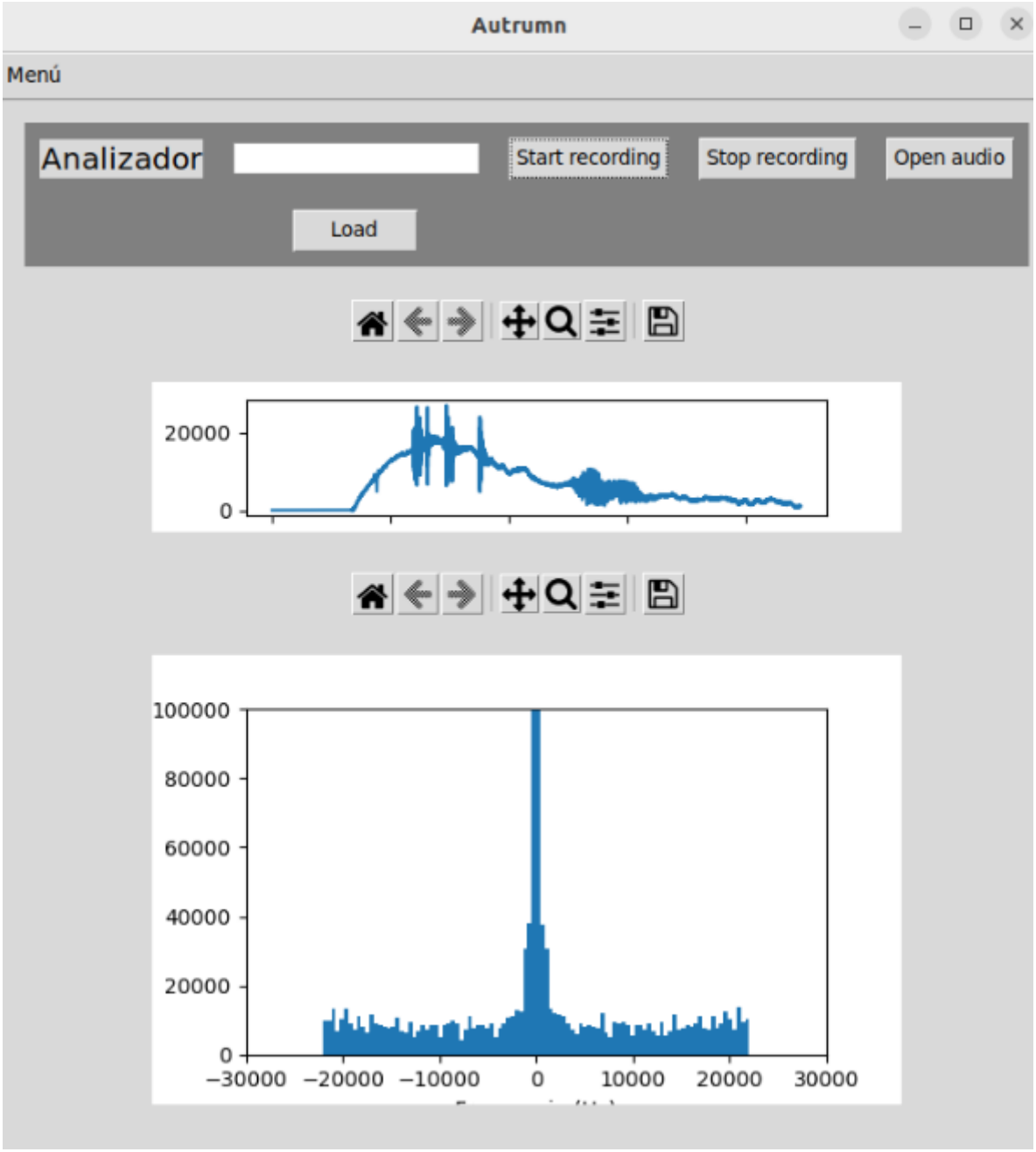
- pip install tk
- pip install wave
- pip install pyaudio
- pip install numpy
- pip install matplotlib
- pip install zip_files
- pip install scipy

# Utilización de la interfaz gráfica

## Analizador

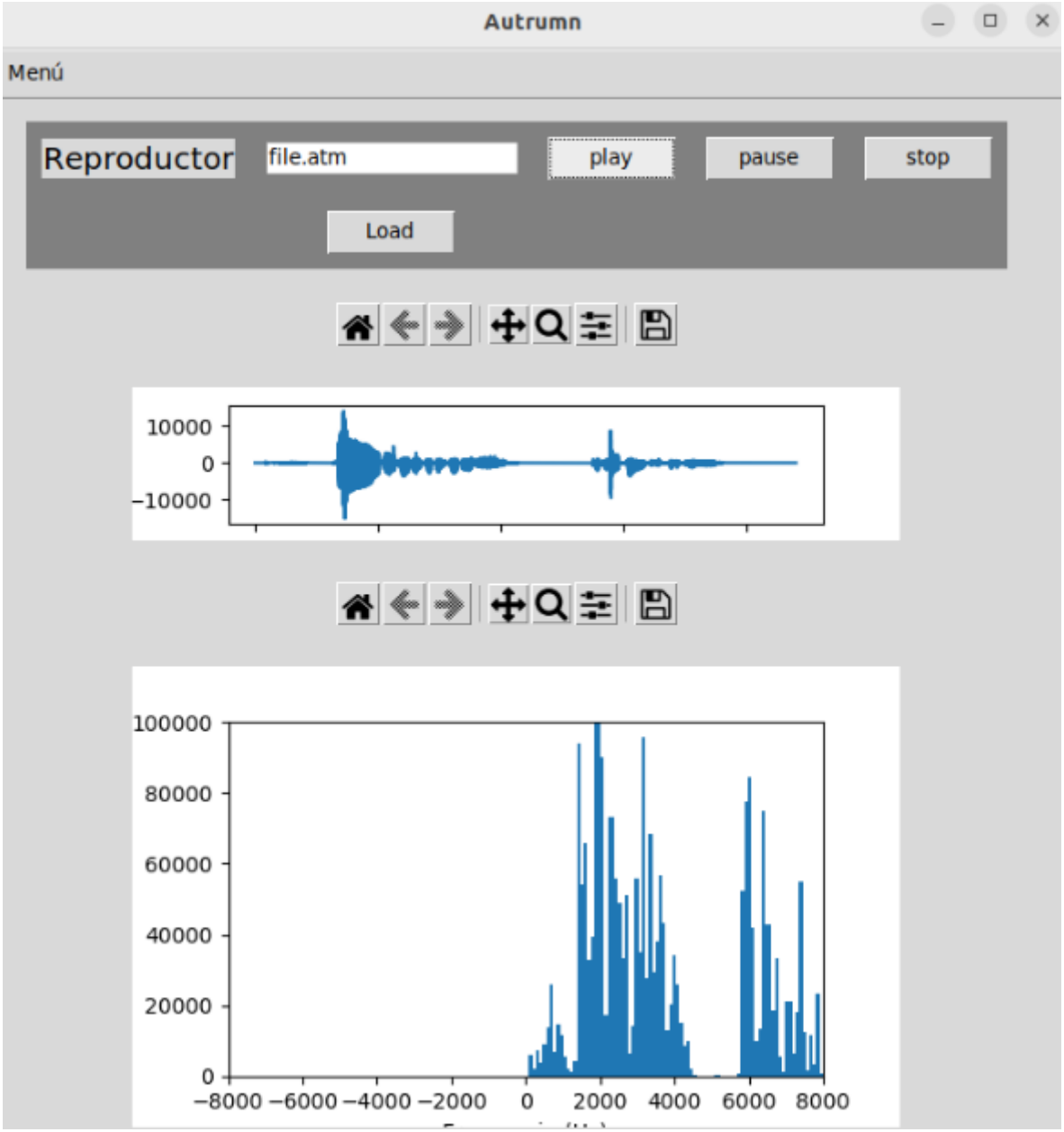

Está sección de la aplicación permite grabar audio y analizarlo mientras se está grabando, al terminar la grabación se genera un archivo WAV, llamado “output.wav” y otro archivo comprimido ATM llamado “file.atm”.

En está sección se permite escuchar un archivo WAV, se realiza escribiendo el nombre del archivo en el text box y presionando el botón “load”




## Reproductor

Esta parte de la aplicación permite abrir archivos ATM, permitiendo reproducir su audio. Para abrir el archivo se escribe el nombre en el textbox, se presiona el botón “load” para cargarlo y al reproducirlo se mostrará una gráfica del audio.




# Código

El código desarrollado se divide en dos módulos. El primero es el analizador, el cuál se encarga de la grabación de audio, analizar archivos de sonido wav y generar los archivos atm que se requieren para el siguiente módulo. El segundo módulo incluye la reproducción de archivos atm y el mismo análisis en tiempo real de estos audios. Todo el código está adaptado también a una interfaz gráfica desarrollada para activar cada una de las funcionalidades propuestas. Dentro de las explicaciones de cada módulo solo se hace referencia a las partes que interactúan directamente con las funciones, no incluye un explicación a profundidad de cada parte de la interfaz. A continuación, la explicación de cada módulo.

## Analizador

A continuación se enlistan cada una de las funcionalidades del analizador de sonidos de Autrum.

**Grabación de sonidos**

La grabación de sonido sucede principalmente en la función startRecording de la clase Analizador. Esta función no recibe parámetros. Empieza abriendo un canal de pyaudio con la información de formato, canales,

rate y cuántos frames se leen a la vez (chunk). El input true es para recibir los datos de un micrófono.

```
stream = p.open(format=FORMAT,
                channels=CHANNELS,
                rate=RATE,
                input=True,
                frames_per_buffer=CHUNK)
```

Una vez abierto el canal empieza un ciclo, el cual lee un chunk desde el buffer del micrófono y lo concatena a un arreglo llamado frames. y este continúa hasta que la variable global recording cambie a falso. Esto sucede con una línea de código que se ejecuta al presionar el botón stop recording de la interfaz gráfica.

```
while(recording):
    data = stream.read(CHUNK)
    numpydata = np.frombuffer(data, dtype=np.int16)
    frames.append(numpydata)
```

Finalmente, se cierra el stream de datos y el canal de pyaudio. Para conservar este audio recién grabado, se utiliza la biblioteca wave para crear un nuevo archivo wav. Se configuran sus características de la misma manera que se hizo al abrir el canal de py audio y se unen escriben los frames grabados. Estos se escriben en bytes como se ve en la línea wf.writeframes y se procede a llamar a la función to_atm la cual guarda tanto la variable frames y el archivo wav en un archivo comprimido atm.

```
#Cierra el input de datos del micrófono
stream.stop_stream()
stream.close()
p.terminate()

#crea un wav con el ncon el nombre output.wav
#toda la información se guarda en
wf = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
wf.setnchannels(CHANNELS)
wf.setsampwidth(p.get_sample_size(FORMAT))
wf.setframerate(RATE)
wf.writeframes(b''.join(frames))
wf.close()
to_atm(frames, WAVE_OUTPUT_FILENAME)
```

**Análisis de wav externo**

Para analizar un wav externo, se desarrolló la función load_wav. En esta se abre un stream de pyaudio con las características del mismo wav, proveídas por las funciones get_forma_from_width, getnchannels y f.getframerate de la biblioteca wave. Además se establece output como True para reproducirlo. Posteriormente se leen los chunks del wav y se almacenan en data. Este arreglo resultante, será recorrido dentro de un ciclo en su totalidad y reproducirá el sonido con stream.write. Además, se almacenan todos los datos reproducidos en un buffer, que los añade a un arreglo frames para su futura graficación.

```
f = wave.open(self.entry.get(), 'rb')
global external_wav_path
external_wav_path = self.entry.get()
p = pyaudio.PyAudio()
global frames
frames=[]
global vec_fourier_frames
vec_fourier_frames = []
#Abre stream de pyaudio
stream = p.open(format = p.get_format_from_width(f.getsampwidth()),
                channels = f.getnchannels(),
                rate = f.getframerate(),
                output = True)
#Lee los chunks del wav y los almacena en data
data = f.readframes(chunk)
i=0
#Abre un ciclo para cada elemento en data
while data:
    stream.write(data)
    data = f.readframes(chunk)

    numpydata = np.frombuffer(data, dtype=np.int16)
    frames.append(numpydata)
```

**Graficación de señal en dominio del tiempo**

Durante el ciclo de grabación y el uso de un wav externo, se pretende que el gráfico de tiempo se actualice en tiempo real y que el usuario vea los cambios de lo que graba. Esto se realiza mediante un condicional dentro del ciclo de grabación o reproducción, en el cual existe un condicional que después de una cantidad de segundos de grabación (definidos en RECORD_SECONDS; que para este proyecto se definieron como 2) entra y llama a la función updatetimecanvas de la clase Analizador. Esta recibe como parámetro los frames a graficar, pero el array de frames tiene múltiples dimensiones, lo cual implica demasiadas líneas para una graficación que ocurre tan frecuente. Entonces se utiliza el comando de la biblioteca Numpy hstack, que convierte ese arreglo en un equivalente unidimensional.

```
if(i>=int(RATE / CHUNK * RECORD_SECONDS)):
    self.updatetimecanvas(np.hstack(frames)) #hstack vuelve el array de frames unidimensional
    self.updatefouriercanvas(fft_data,freqs)
    i=0

i+=1
```

Finalmente, en la función updatetimecanvas, se llama a la Figure de Tkinter que contiene el gráfico (que es una variable de clase llamada fig1) la cual limpia la gráfica actual.

```
def updatetimecanvas(self, timeframe):
    self.fig.clear()
    self.fig.add_subplot(111).plot(timeframe)  # Refrezca el plot con los frames actuales
    self.canvas.draw_idle()
```

## Transformada de Fourier

El código utilizado principalmente para la transformada de fourier se muestra a continucación:

```
fft_data = np.fft.fft(numpydata)
            freqs = np.fft.fftfreq(len(numpydata), d=1/RATE)
            fourier_frames.append(fft_data)  #datos  de la transformada
            vec_fourier_frames.append(freqs)  #datos de frecuencia de la
transformada
```

Para calcular la transformada de fourier, se utilizan funciones de la librería de numpy. En el código implementado se está utilizando la Transformada Rápida de Fourier (FFT) con el objetivo de calcular la transformada de fourier de la señal de audio dada, la cual se representa como un array de datos de tipo numpy.

Se utiliza la función nn.fft.fft de numpy, que toma como entrada el array de datos numpydata y calcula su transformada de Fourier. El resultado de esta operación es un nuevo array de números complejos, que contiene los coeficientes de la transformada de Fourier.

Además, se utiliza la función np.fft.fftfreq para calcular las frecuencias correspondientes a los coeficientes de la transformada de Fourier. Esta función toma como entrada la longitud de la señal (len(numpydata)) y la tasa de muestreo (RATE), y devuelve un array de números que representan las frecuencias en hertzios.

Los resultados de aplicar varias funciones se agregan en dos listas separadas, es decir, el array que contiene los coeficientes de la transformada de fourier se concatenan con los datos existentes en la lista “fourier_frame”, y el array de las frecuencias se almacena junto a los otros datos de frecuencias en la lista “vec_fourier_frame”.

**Graficación de componentes de frecuencia**

Para graficar los componentes de frecuencia, se utilizan los datos calculados en la transformada de fourier, Una vez que se tienen listos los componentes de frecuencia calculados, se llama a la función “updatefouriercanvas(fourier_frames[:, 0], vec_fourier_frames)” , la cual toma como entrada los primeros elementos de fourier_frames y vec_fourier_frames, que corresponden a los datos de la transformada de Fourier para el primer canal de audio.

Una vez dentro de la función ”updatefouriercanvas”, se procede a borrar cualquier gráfico previo en el canvas utilizado para graficar los datos de frecuencia, luego se agrega un nuevo plot, y utilizando los datos de la transformada de fourier y las frecuencias correspondientes, se crea un histograma donde el eje “x” corresponde a las frecuencias y el eje “y” corresponde con el valor absoluto de los componentes de la transformada de fourier. Es importante indicar que se utilizan las funciones set_xlim y set_ylim de matplotlib para poder definir las dimensiones del gráfico y así visualizar de una forma más precisa los datos del mismo.

## Reproductor

A continuación se enlistan cada una de las funcionalidades del reproductor de sonidos de Autrum.

## Reproducción del audio

Para este fin se creó una clase llamada AudioPlayer para hacer una implementación de un reproductor de audio que utiliza la biblioteca pyaudio. A continuación, una descripción de cada uno de los atributos y métodos de la clase:

**Atributos:**

- filename: una cadena que representa el nombre de archivo de salida.
- chunk: un número entero que representa el tamaño de cada fragmento de audio.
- paused: un valor booleano que indica si la reproducción está en pausa o no.
- stopped: un valor booleano que indica si la reproducción se detuvo o no.
- wave_file: un objeto de archivo de onda que contiene los datos de audio.
- p: un objeto PyAudio que se utiliza para interactuar con el controlador de audio.
- stream: un objeto que representa el flujo de audio que se está reproduciendo.
- timer: un número que representa el tiempo transcurrido desde que comenzó la reproducción.
- fig: un objeto que representa la figura de la forma de onda de tiempo.
- fig2: un objeto que representa la figura de la forma de onda de frecuencia.
- canvas: un objeto que representa el lienzo en el que se mostrará la forma de onda.

**Métodos:**

- *_init_*: el constructor de la clase que inicializa todos los atributos necesarios para la reproducción de audio.
- updatetimecanvas: actualiza el lienzo de la forma de onda de tiempo.
- check_realtime: comprueba el tiempo transcurrido desde que comenzó la reproducción.
- load: carga los datos de audio en el objeto wave_file y configura el objeto p para la reproducción.
- play: reproduce el audio y actualiza el temporizador mientras el audio se está reproduciendo.
- pause: detiene temporalmente la reproducción de audio y almacena el tiempo transcurrido en el temporizador.
- resume: reanuda la reproducción de audio después de haber sido pausada.
- stop: detiene la reproducción de audio y cierra todos los objetos y flujos de audio.

**Graficación de señal en dominio del tiempo en tiempo real**

En el ciclo de la función play del reproductor, se calcula cuando el tiempo actual de reproducción (end_time - start_time) que dice por qué segundo se encuentra la reproducción. Esta reproducción cada cierto número de segundos (especificados en PLAY_RANGE y para este proyecto siendo 5) se llama la función check_realtime para comenzar a graficar la función de tiempo. Este control de tiempo se lleva a cabo mediante la variable multiplier que indica el múltiplo de PLAY_RANGE que sigue antes de volver a graficar.

```
multiplier = 0
self.check_realtime(multiplier)
multiplier +=1
while data != b'' and not self.stopped:
    if not self.paused:
        start_time = time.time()
        self.stream.write(data)
        data = self.wave_file.readframes(self.chunk)
        end_time = time.time()
        self.timer += end_time - start_time

        ## Función de tiempo
        if self.timer >= multiplier*PLAY_RANGE: #si el tiempo ya es mayor que el multiplicador por el rango de segundos
            self.check_realtime(multiplier)
            multiplier += 1 #aumentar el multiplicador en 1.
```

Con la función check_realtime, que recibe un multiplicador, se calcula la porción actual del audio que se reproduce que se quiere mostrar. Esto se realiza calculando un límite inferior o superior. El condicional presente evita que el límite superior se salga del rango del arreglo, de manera que si se calcula que sería mayor se reemplaza por el largo menos uno. Para este se calculó que el samplerate guardaba 43 frames cada segundo, número que se multiplica por cuantos segundos se requiere graficar a la vez. En este caso, este valor se encuentra en PLAY_RANGE y se estableció como 5 para 5 segundos. Finalmente este número se multiplica por el punto actual del audio que es el multiplicador. Finalmente, se llama a la función updatetimecanvas con un sub arreglo limitado por estos valores.

```
def check_realtime(self, multiplier):
    global frames

    if (43 * (multiplier + 1) * PLAY_RANGE <= len(frames)):
        self.updatetimecanvas(np.hstack(frames[43*multiplier*PLAY_RANGE:43 * (multiplier + 1) * PLAY_RANGE]))
    else:
        self.updatetimecanvas(np.hstack(frames[43 * multiplier * PLAY_RANGE:len(frames)-1]))
```

La graficación sucede de manera similar que en la grabación, con una función updatetimecanvas, que recibe el timeframe unidimensional de los frames a graficar. Este limpia el Figure que contenga a esta gráfica y crea un plot dentro de ella con matplotlib y la información requerida.

```
def updatetimecanvas(self, timeframe):


    self.fig.clear() #limpia el gráfico actual
    self.fig.add_subplot(111).plot(timeframe)  # Genera un plot con los valores especificado
    self.canvas.draw_idle()
```

**Graficación de señal en dominio de frecuencia en tiempo real**

El proceso de graficación es muy similar a lo explicado en la sección del analizador. Se toma como entrada la lista de arrays que contienen los datos del audio, y se le calcula la transformada de fourier mediante las funciones fft y fftfreq, además, cabe aclarar que del resultado de la función fft(que se almacena en la variable de fourier_frames), se toma solo la primer parte. Seguidamente, se procede a llamar a la función de updatefouriercanvas que se encarga de graficar los componentes de frecuencia enviados. Esta función se puede llamar de dos formas distintas:

```
if (43 * (multiplier + 1) * PLAY_RANGE <= len(frames)):
    self.updatefouriercanvas(fourier_frames[43*multiplier*PLAY_RANGE:43 *
(multiplier + 1) * PLAY_RANGE],vec_fourier_frames[43*multiplier*PLAY_RANGE:43 *
(multiplier + 1) * PLAY_RANGE])
else:
    self.updatefouriercanvas(fourier_frames[43 * multiplier *
PLAY_RANGE:len(frames)-1],vec_fourier_frames[43 * multiplier *
PLAY_RANGE:len(frames)-1])
```

Con esto se busca que no se grafique más allá del total de frames disponibles, por lo que si el próximo rango de graficación excede el largo del frame, se utiliza el largo del frame como límite superior.

Para encontrar una medición exacta de los 5 segundos a lo largo de frames, se calculó que hay 43 frames por segundo. Estos 43 se multiplica por un PLAY_RANGE que sería de 5 para 5 segundos, y para encontrar el punto actual se multiplica por el multiplicador actual para buscar el frame 10, 15, etc. Para el límite superior, se le suma 1 al multiplicador para que dé el siguiente punto.

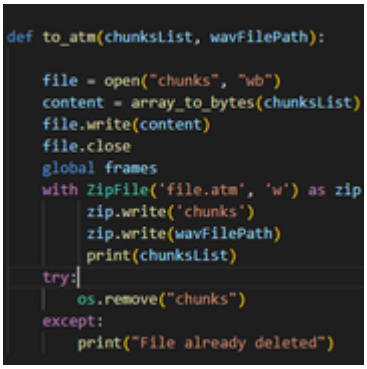
## Generación de archivos ATM

Los archivos ATM en esencia es un archivo comprimido el cual contiene dos de los componentes necesarios para que el programa pueda reproducir el audio y los gráficos nuevamente sin tener que generarlos de cero. Estos componentes son:

- El archivo WAV con el cual se reproducirá el audio.
- Un archivo binario el cual contiene un arreglo de numpy el cual es generado por el programa a partir del audio, este es guardado para evitar recalcular de nuevo con el audio.

Para ello se utilizó la librería ZipFile, la cual permite realizar la compresión y descompresión de los archivos en python y BytesIO convertir el arreglo numpy a bytes y viceversa.

**Generar archivos ATM**



```
def to_atm(chunksList, wavFilePath):

    file = open("chunks", "wb")
    content = array_to_bytes(chunksList)
    file.write(content)
    file.close
    global frames
    with ZipFile('file.atm', 'w') as zip
        zip.write('chunks')
        zip.write(wavFilePath)
        print(chunksList)
    try:
        os.remove("chunks")
    except:
        print("File already deleted")
```

Para ello se tiene la función to_atm, la cual recibe una la lista generada por el programa “chunksList” y la ruta al archivo wav, con eso se convierte a bytes el arreglo y se guarda en un archivo de nombre chunks. Para el archivo wav comprime el archivo indicado en la ruta suministrada.

**Lectura de los archivos ATM**

```python
def from_atm(filepath):
    with ZipFile(filepath) as zip:
        files = zip.namelist()
        for i in range(0,len(files)):
            if(".wav" in files[i]):
                global wavFile
                zip.extract(files[i])
                wavFile = open_wav_file(files[i])
            elif(files[i] == "chunks"):
                global frames
                frames = bytes_to_array(zip.read(files[i]))
```

Se tiene la función from_atm que recibe la ruta al archivo .atm a ser cargado. Utilizando la librería ZipFile se carga el archivo, se obtiene la una lista con los archivos que están comprimidos y se buscan los dos componentes del atm, un archivo cuyo nombre contenga la extensión .wav y el archivo chunks, cualquier otro elemento existente en el comprimido es ignorado. En el caso del .wav se descomprime y se carga en memoria con la librería wave, mientras que chunks es cargado directamente utilizando la función de FileZIp read.

## Recomendaciones

En este trabajo una de las mayores dificultades que se presentaron fué analizar los gráficos generados, es por ello que para futuros trabajos se podría mejorar la graficación de los datos e incluso profundizar más sobre el espectro de frecuencias que se quiera estudiar de modo que permita centrarse mejor en ese espacio y reducir el ruido. Por otro lado si se quiere utilizar la herramienta para comparaciones, lo ideal sería utilizar un mismo equipo de grabación en un entorno controlado para evitar que ruidos externos o incluso las diferencias entre un micrófono y otro afecten los resultados.

## Conclusiones

En conclusión, el programa realizado permite de manera exitosa observar las componentes de frecuencia que componen los sonidos que se le ingresen. Hay puntos de mejora para el proyecto que podrían hacer que su utilidad para análisis aumente, pero cumple con los objetivos planteados para este proyecto.