Análisis de métodos matemáticos para la física

Situación problema: ¿cuál es el nombre de esa canción?

Alberto Ruiz Biestro — A01707550*
Instituto Tecnológico y de Estudios Superiores de Monterrey

(Dated: 4 de diciembre de 2021)

Recientemente se ha popularizado el uso de aplicaciones, en smartphones y otros dispositivos móviles,que permiten identificar el nombre de una canción a partir de un fragmento de la misma, los ejemplos más populares son **Shazam** y **Soundhound**. Los algoritmos utilizados por estas aplicaciones son algoritmos patentados, sin embargo utilizan principios basados en un análisis espectral de las canciones que se desea identificar. Nosotros deseamos desarrollar nuestra propia app que nos permita identificar canciones de forma similar, con el propósito de monitorear transmisiones de radio,ya sea analógico o digital, para validar que no se están violando los derechos de autor o pago de regalías.

PRIMERA ETAPA – MÉTODOS PARA IDENTIFICAR CANCIONES

I. TONOS Y TIMBRES

Las notas musicales son representadas por una frecuencia única. Como estándar en la música occidental, se toma a la nota *La* de la cuarta octava (*A4* en Inglés) con frecuencia de 440 Hz. Las consonancias y disonancias entre las notas también son reflejadas en las frecuencias. La frecuencia de una nota se duplica por cada octava que se aumente ($f_{A5} = 2f_{A4} = 880$ Hz). A esto se le llama el *tono* del instrumento.

No obstante, un mismo tono (o frecuencia) no implica un mismo sonido. Esto se le atribuye al *timbre* o textura del instrumento, y depende de las *armónicas* (*overtones* en Inglés) que se encuentran en frecuencias por encima de la *frecuencia fundamental*. Los instrumentos tradicionales cuentan con decenas de frecuencias apiladas, mientras que los sintetizadores modernos pueden producir la forma de onda fundamental: la onda sinusoidal. La combinación de estas ondas con diferentes frecuencias dicta cómo será interpretado por el oído el instrumento. Una combinación de ondas sinusoidales puede ser vista al correr el código de abajo:

```
import numpy as np
import matplotlib.pyplot as plt
from functools import reduce

Pi = np.pi

def sines(k):
  w0 = 100
  x = np.linspace(-0.02*Pi,0.02*Pi, 500)
  return np.sin(w0 * k * x)

k = np.arange(1,20,2)

y_x = sum(map(sines, k)) # suma la funcion sines
    aplicada a cada k[i]
plt.plot(y_x)
```

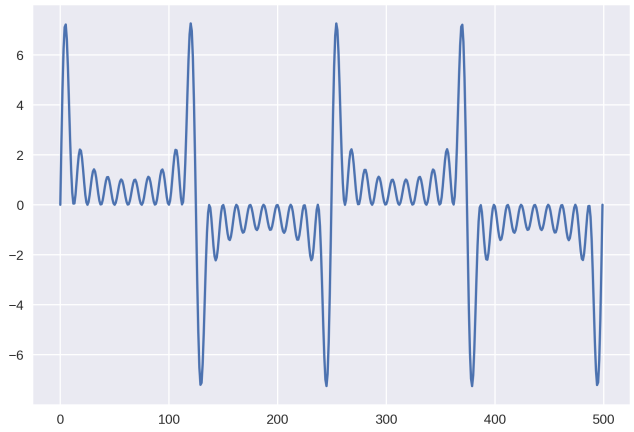


Figura 1. Suma de ondas sinusoidales con frecuencias crecientes

II. DIGITALIZACIÓN Y MUESTREO

El audio está codificado como una lista enorme de números. Se puede definir una frecuencia de muestreo, por lo usual 44.1 kHz (muestras por segundo) [1]. Como ejemplo, en 3 minutos de grabación habrá

$$3\text{ min} \times \frac{60\text{ s}}{1\text{ min}} \times \frac{44100\text{ muestras}}{1\text{ s}} = 15,876,000\text{ muestras}$$

También es importante destacar que la frecuencia de muestreo, f_s, será la misma para el canal izquierdo y derecho (*stereo*), o si es sólo un canal (*mono*).

El teorema de *Nyquist–Shannon* nos indica que hay un límite teórico en la frecuencia máxima que podemos captar mientras hacemos el muestreo, y está dictada por f_s. Para sintetizar, el teorema establece lo siguiente: Si una señal x está se encuentra en el rango (o banda) $(-B, B)$, entonces está completamente determinada por su muestra con una frecuencia de muestreo de $f_s = 2B$ [2]. En otras palabras, la frecuencia de muestreo mínima que captura la escencia de la información análoga debe de ser *al menos* 2 veces la frecuencia de la nota a digitalizar. Podemos recalcar que el tono más agudo que

* Contacto: A01707550@itesm.mx

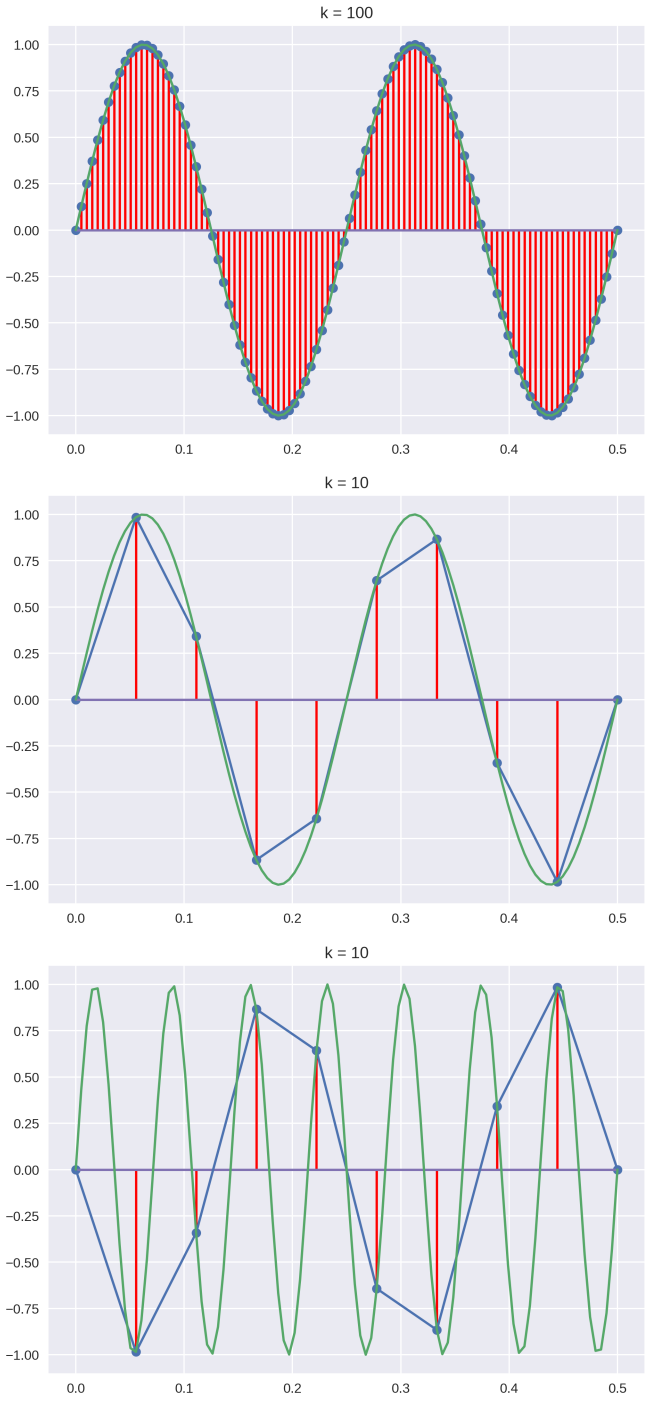


Figura 2. Visualización del teorema de Nyquist–Shannon

puede escuchar un ser humano es de 20 kHz, por lo cual es congruente que se haya escogido como mínimo los 40 kHz para la frecuencia de muestreo. El 44.1 kHz se implementó dado que se apegaba mejor a ciertos estándares de video y sonido, y se encontraba por encima de dicho límite [3].

III. MODULACIÓN POR IMPULSOS CODIFICADOS (PCM)

En los programas musicales antiguos se utilizaba (por lo usual) el número 64 como referencia para el nivel máximo de volumen. Esto es porque se acostumbraba utilizar 6 bits para codificar el volumen del sonido análogo (2^6 niveles). En inglés, a ésto se le llama *bit–depth*, y corresponde al número de bits por muestra. Un CD utiliza 16 bits ($2^{16} = 65,536$ niveles) [4].

A la representación estándar de las señales digitales se le conoce como PCM. Un archivo *.mp3* es autimáticamente convertido a una señal PCM antes de ser enviada a los audífonos [3]. Para la frecuencia de muestreo que habíamos visto (44.1 kHz), tendremos 44.1 mil muestras con un *bit–depth* de 16-bits cada segundo (2 bytes por cada canal, L-R).

IV. TRANSFORMADA DE FOURIER

En pocas palabras, la transformada de Fourier, como su nombre lo dice, *transforma una función de tiempo a una función de frecuencias*, y actúa como $f : \mathbb{R} \to \mathbb{C}$.

Del Wunsch [5], «asuma que tenemos una función absolutamente integrable $f(t)$, la cual es contínua por partes sobre cada interval finito a lo largo del eje t. Entonces podemos definir una nueva función, $F(\omega)$, llamada la *transformada de Fourier* de $f(t)$, dada por la siguiente definición»

$$F(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) e^{-i\omega t}\, dt \qquad [1]$$

Cuando la transformada de Fourier es conocida, también podemos definir la *transformada inversa de Fourier* (también llamada la representación integral-Fourier de $f(t)$:

$$f(t) = \int_{-\infty}^{\infty} F(\omega) e^{i\omega t}\, d\omega \qquad [2]$$

Ésta cuenta con la limitante en la que no se puede obtener $f(t)$ para valores de t donde $f(t)$ no es contínua. Para demostrar el método consideremos el siguiente ejemplo (también del libro de Variable Compleja de Wunsch):

1. *Ejemplo: Encuentre la transformada de Fourier de la siguiente expresión*

$$f(t) = \begin{cases} e^{-t}, & t \geq 0, \\ 0, & t < 0, \end{cases} \qquad [3]$$

Desde Eq. [1] obtenemos:

$$F(\omega) = \frac{1}{2\pi}\int_0^\infty e^{-t}e^{-i\omega t}\,dt = \frac{1}{2\pi}\int_0^\infty e^{-(1+i\omega)t}\,dt$$
$$= \frac{1}{2\pi}\frac{e^{-(1+i\omega)t}}{-1-i\omega}\Big|_0^\infty = \frac{1}{2\pi}\frac{1}{1+i\omega}$$

A. Transformada de Funciones Armónicas

Lo que se busca es obtener la transformada de funciones armónicas (o periódicas), dado que estas componen el sonido que percibimos como música. Una demostración no rigurosa del por qué se pueden obtener las frecuencias a partir de la Transformada ($\mathfrak{F}$) es la siguiente:

$$f(t) = A\sin(\omega_o t)$$
$$\vdots$$
$$\mathfrak{F}[f(t)] = \frac{A}{2i}\left[\frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i(\omega_o+\omega)t}\,dt - \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i(\omega-\omega_o)t}\,dt\right]$$
$$\mathfrak{F}[f(t)] = \frac{A}{2i}[\delta(\omega+\omega_o) - \delta(\omega-\omega_o)]$$

Podemos ver cómo obtenemos la frecuencia de la onda sinusoidal. Si aplicamos la Transformada de Fourier a cada función periódica $f_k(t)$ que compone la canción se espera obtener un espectrograma de las frecuencias que componen dicha canción.

B. Transformada Discreta de Fourier

Como ya lo vimos, el análisis de Fourier es elemental para expresar una función como la suma de sus componentes periódicos. Subsecuentemente se busca recuperar la función de dichos componentes. Sin embargo, se consideró el caso en el cual tanto $f(t)$ como $F(t)$ están definidas en un intervalo contínuo. Como ya vimos en la sección II, la digitalización del sonido conlelva su discretización. Es por esto que en las ciencias computacionales se reemplaza a la Transformada de Fourier por su versión discreta, *DFT* por sus siglas en Inglés [6]. Además se utiliza un algoritmo llamado *Fast Fourier Transform*, o *FFT*, para reducir la complejidad de tiempo que conllevaría utilizar la Transformada Discreta para una canción con una f_s de 44.1 kHz. Primero veremos cómo funciona la DFT.

En términos del problema, la entrada a la transformada es comúnmente la señal (la cual existe en el dominio del tiempo), mientras que la salida es el espectro (el cual existe en el dominio frecuencial) [6]. Habrá que aplicar la transformada a un canal *mono*, de lo cual había hablado en la sección II, y en caso de que sea *stereo* la entrada, habrá que convertirla por medio del promedio de la suma de cada vector (L-R), de tal forma que $S_{mono} = 0.5(S_{stereo}[0] + S_{stereo}[1])$[1]. La transformada y su inversa están dadas por las siguientes expresiones [7]:

$$X_\omega = \sum_{n=0}^{N-1} x_t \exp\left(\frac{-2i\pi\omega t}{N}\right) \quad [4]$$

$$x_t = \frac{1}{N}\sum_{k=0}^{N-1} X_\omega exp\left(\frac{2i\pi\omega t}{N}\right) \quad [5]$$

Donde X_ω representa una función discreta en el dominio frecuencial y x_t una función en el dominio temporal. Para fines prácticos, N representa el número total de muestras que componen la señal (también llamado *window*). El tamaño del intervalo (en inglés *frequency bin*) determina la resolución del espectro discreto que regresa la transformada [3], y está dado por f_s/N [Hz]. Claro que al aumentar el la resolución aumenta el tiempo de computación. En otras palabras, cunado discretizamos las frecuencias, obtenemos dichos *frequency bins*. Dado esto, cuando se discretiza la Transformada de Fourier hacemos una operación: $e^{-iw} \to e^{-2i\pi k/N}$.

Otra forma de reducir significativamente la complejidad del programa es por medio de un remuestreo, el cual **mantiene** la resolución. Este método consiste en disminuír f_s, dado que no afecta tanto nuestra percepción de una canción (ver `scipy.signal.resample` o la opción de `librosa.load`, [8]). Si pasamos de 44.1 kHz $\to 0.25 \cdot 44.1$ kHz entonces tendremos un rango de frecuencias de 0 a 5.5 kHz ($\{-B,B\} = \{-f_s/2, f_s/2\}$). A pesar de la reducción, sigue siendo posible obtener una *huella digital* precisa de la canción [3]. Antes de *submuestrear* la canción, será útil introducir un filtro para filtrar las frecuencias mayores a 20 kHz para que no corrompan el espectro a causa del teorema de Nyquist-Shannon (ver la Sección II).

De la página de SciPy podemos obtener una descripción más apegada al código de Python para la Transformada Discreta de Fourier. En este punto podemos ver fácilmente que las funciones nos regresarán un arreglo de números complejos [8] [2]:

$$y[k] = \sum_{n=0}^{N-1} x[n]\exp\left(\frac{-2\pi jkn}{N}\right) \quad [6]$$

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} y[k]\exp\left(\frac{2\pi jkn}{N}\right) \quad [7]$$

«Para N par, los elementos $\langle y[1],\ldots,y[N/2-1]\rangle$ contienen los términos de frecuencias positivas, mientras que los elementos $\langle y[N/2],\ldots,y[N-1]\rangle$ contienen las frecuencias negativas (en orden decerciente). Para N impar, se toma $\langle y[1],\ldots,y[(N-1)/2]\rangle$ y $\langle y[(N+1)/2],\ldots,y[N-1]\rangle$ para frecuencias positivas y negativas, respectivamente»[8]. Se

[1] $S_{stereo}[0] \to$ canal iquzierdo, $S_{stereo}[1] \to$ canal derecho (por convención)

[2] Se usa j para denotar $\sqrt{1}$ dado que i suele ser utilizado como un iterador para programación

acostumbra que sólo el espectro positivo sea graficado, porque además estas frecuencias son simétricas.

V. MANCHADO ESPECTRAL

Cuando se calcula la FFT (recordemos que es un algoritmo para la DFT), «el *Manchado Espectral*, o *Spectral Leakage*, ocurrirá si alguna de las componentes frecuenciales cae entre las frecuencias de referencia» [9]. En otras palabras, cualquier otra operación que resulte en la creación de nuevas frecuencias o componentes frecuenciales producirá un *leakage*. Se puede llamar *aliases*[3] del «componente del espectro original»[9].

VI. ESPECTROGRAMA

Después de haber obtenido las frecuencias de nuestra canción o señal periódica, podemos hacer un *plot* de las frecuencias contra el tiempo. Luego se aplican *tokens* obtenidas a partir de un *hashing* [4]de los valores de los *peaks* o puntos altos en el espectrograma [10]. De acuerdo con el algoritmo original desarrollado por Avery Wang para Shazam, Tanto la base de datos como la muestra de audio (obtenida del micrófono) son sometidas al análisis con la Transformada, y se obtienen dos huellas digitales, para así evaluar qué tanto se parecen mediante las *tokens* mencionadas previamente [10]. Éstas se obtienen al encontrar los picos en el espectrograma (ver Figuras VI y VI).

Los hashes generados por la base de datos deben de ser reproducibles por una grabación o muestra degradada [10]. A esto se le llama *robustez*, y debe de ser acompañada por una entropía suficiente para que no haya colisiones o identificaciones erróneas al momento de comparar una muestra con la base de datos generada. Los puntos altos (*peaks*) son una forma acertada de comparar muestras a bases de datos, dado que son los más probables a sobrevivir en un muestreo con mucho sonido [10]. El patrón para la muestra debería ser el mismo para un fragmento de la base de datos. Esto trae un nuevo problema en sí, dado que el algoritmo deberá no sólo buscar entre la base de datos entera, pero también entre intervalos de tiempo de cada canción dentro de ésta.

El *mapa de constelaciones* (ver Figura VI) es encontrado por medio de una función similar a la de `findpeaks` de MATLAB, y es utilizada para hacer un subsecuente análisis por medio de hashing.

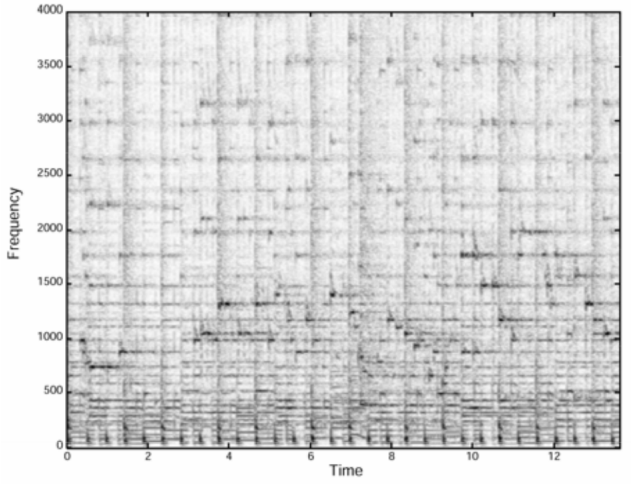


Figura 3. Espectrograma [10]

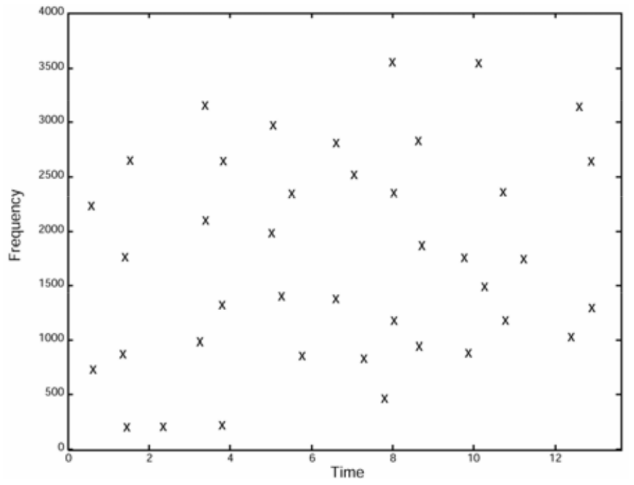


Figura 4. «Mapa de Constelaciones» [10]

VII. HASH-TABLES

Una *hash table*, o matriz asociativa, es exactamente como su nombre lo dice: una tabla a la cual cada índice le corresponde a un valor único. Este valor es nuestra canción (o el nombre de la canción), y el *hash* correspondiente es el índice en donde se encuentra. Un *hash* se encuentra al utilizar una *hash function*. Debe tener por lo menos tres fundamentos:

- Ser fácil de computar.
- Tener una distribución uniforme.
- No tener colisiones (cuando más de un elemento es transformado al mismo hash).

Lamentablemente, estos métodos no fueron implementados en el algoritmo dada la complejidad y el tiempo disponible. El intento será expuesto más adelante.

---

[3] Se le llama *Aliasing* al efecto que causa que señales diferentes se vuelvan indistinguibles (*aliases* de una y otra) cuando son sujetas a un muestreo. Para nuestro caso, se le refiere a la distorción (relativa a la señal original) que resulta cuando una señal es reconstruída a partir de muestras. Leer más: `https://support.ircam.fr/docs/AudioSculpt/3.0/co/Aliasing.html`

[4] Transformación única de una llave o valores a otros, únicos al valor de entrada

SEGUNDA ETAPA – IMPLEMENTACIÓN

VIII. FUNCIONES

En esta sección explicaré lo que hacen las funciones del código, y subsecuentemente en la sección IX se explicará el programa principal que detecta las canciones. Sugiero ver el programa `A01707550_proceso.ipynb` para tener una mejor visualización del código y lo que hace. Las funciones de abajo son todas las que se utilizaron para el proceso, aunque no hayan sido utilizadas para el producto final.

A. `read_this(string, srate)`

Esta función lee el archivo `string` con una frecuencia de muestreo `srate`.

B. `fourier(signal, samplerate, divisor)`

Aplica la Transformada Discreta de Fourier (con el algoritmo FFT) a una señal con su respectiva f_m en un rango de tiempo de la señal. El `divisor` dice qué tantas veces es dividida la f_m, y por ende la dimensión del intervalo de tiempo en el que se aplica la Transformada. Un divisor mayor implica un tiempo menor, y por lógica un espectro con mayor resolución, pero esto no siempre es lo que queremos. La función regresa la magnitud en decibeliosl, el vector de tiempo de la canción, la magnitud en unidades arbitrarias (no se aplica el `log`), un vector para las frecuencias, y el intervalo en el que se aplicó la transformada (en unidades frecuenciales).

C. `noisify(string, srate)`

Lee una canción con una f_m diferente (se busca que sea menor) y regresa una señal con sonido blanco y menor calidad.

D. `spectrogram(matrix, time, freq, title, bartitle)`

Nos regresa un espectrograma a partir de una matriz de magnitudes (sea en decibelios o en otras unidades). Para las escalas se necesita un vector de tiempo `time`, uno de frecuencia `freq`, y títulos opcionales para las gráficas. Todos los espectros que enseño en este documento fueron hechos con esta función.

E. `constellation(matrix, time, freq, interval, pmn, title)`

Nos regresa un espectro similar al *Mapa de Constelaciones* del artículo de Avery Wang [10]. Es bastante similar a la función `spectrogram`, pero aquí se utiliza un valor de prominencia `pmn` para los *peaks* que encontrará, y un intervalo `interval` que corresponde al intervalo de tiempo en el cual se encontrarán estos puntos altos.

F. `hashfun(x, plot)`

Una función que nos calcula los *peaks* y nos regresa un vector de hashes producidos. Es demasiado lenta, y sólo sirvió para el primer intento de identificación musical. Al final, si la diferencia entre la función aplicada a la muestra (la cual tiene que ser una canción completa) y la función aplicada a la canción de la base de datos era pequeña (se calculaba el MSE), entonces se decía que eran correspondientes ambas señales. Es un proceso bastante ineficiente, pero se demuestran los resultados de todos modos.

G. `mypeaks(matrix)`

Nos divide un rango de frecuencias $0 \leq f \leq 2000$ en cuatro intervalos y subsecuentemente calcula un punto máximo en cada intervalo. Se aplica a cada división original en la cual se calculó la transformada, y se regresa un vector`peak_freq`.

H. `getscore(song_magnitude, sample_magnitude)`

Compara cada índice de un vector de *peaks* de la muestra con otro vector de *peaks*. La longitud del primero debe ser menor a la del segundo. El puntaje se calcula con la suma de los valores `True` que encuentra. Es decir, si las frecuencias son las mismas en el índice `[k_1, k_2]`, se guarda un `True`, y cuando no lo son se guarda un `False`. Regresa el puntaje obtenido (en sí, qué tanto se parecen los vectores).

I. `findmatch(goal)`

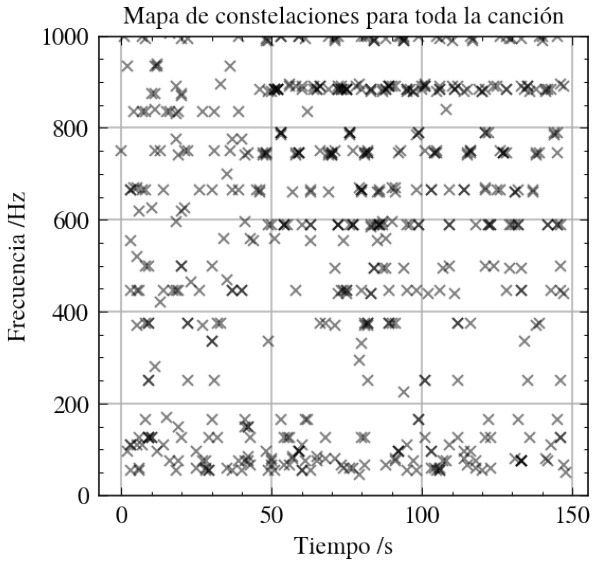
Nos encuentra un *match* para cierta grabación (nuestro *goal*). Para esto se tienen que crear bases de datos de los *peaks* calculados para la base de datos de las canciones. Se deben cargar los archivos `pairs`, el cual corresponde a los pares de `[song ID, song name]`; `div`, el cual es la división con la que se creó la base de datos de puntos clave (para así aplicar el mismo proceso a la grabación); `mag`, el cual contiene las magnitudes e índices de los *peaks* para calcular el puntaje entre cada matriz de cada canción y la grabación. Ver el pro-

grama `A01707550_implementacion.ipynb` para una mejor comprensión.

IX. IMPLEMENTACIÓN FORMAL Y RESULTADOS

Se creó el programa `A01707550_implementacion.ipynb` para demostrar tres posibles casos de la implementacion del algoritmo a grabaciones. Aconsejo visitar el archivo `A01707550_implementacion.html` para escuchar las grabaciones y las canciones que identifica el programa de una manera más interactiva. En este se obtienen primero las magnitudes para las bases de datos de las canciones, y se guardan en eldirectorio respectivo. Por órden alfabético se leen las canciones y se les asigna un `id` correspondiente, lo cual es guardado en una matriz `song ID, song name`. Cabe notar que a veces el programa asigna los ids por órden de creación, lo cual no debería afectar en nada.

Después de la creación de la base de datos, se lee un fragmento de una canción, y se obtiene un score relativo a cada una de las canciones en la base de datos. A continuación se demuestran los resultados.



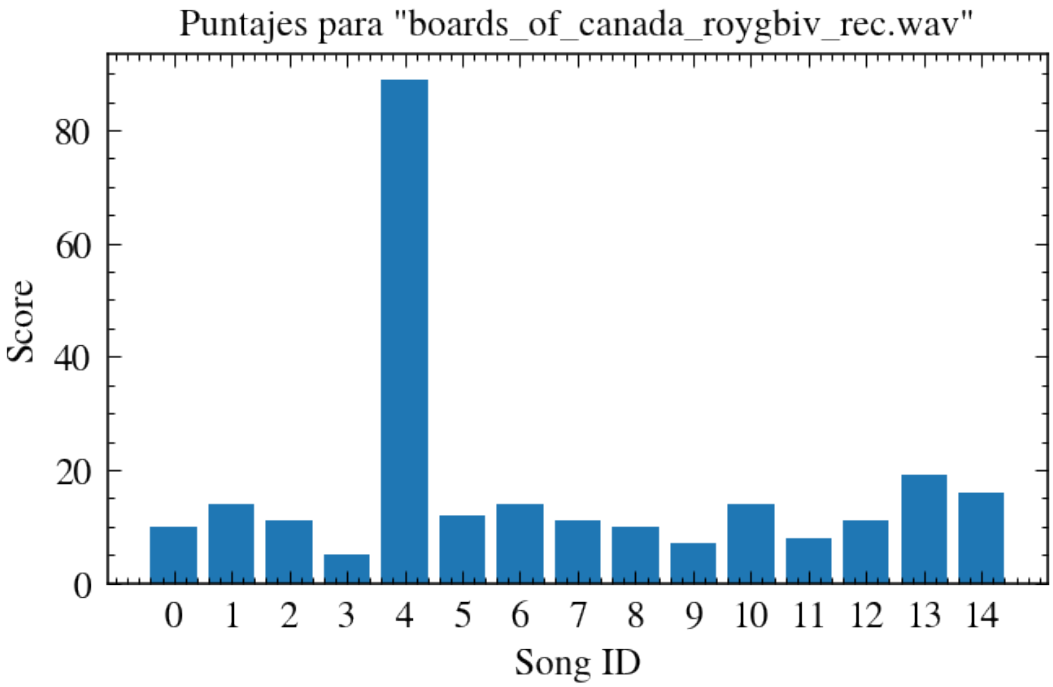


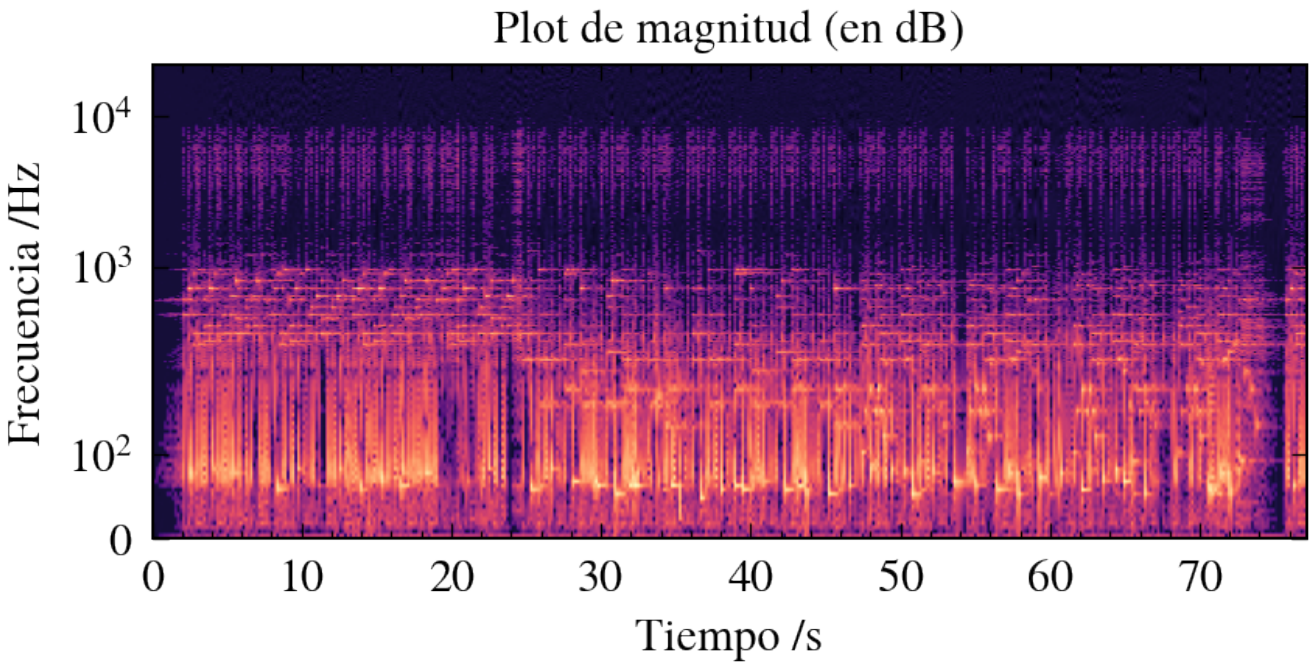
Figura 5. Ejemplo de una identificación acertada de una muestra de audio (el `song ID` 4 corresponde a la canción correcta)

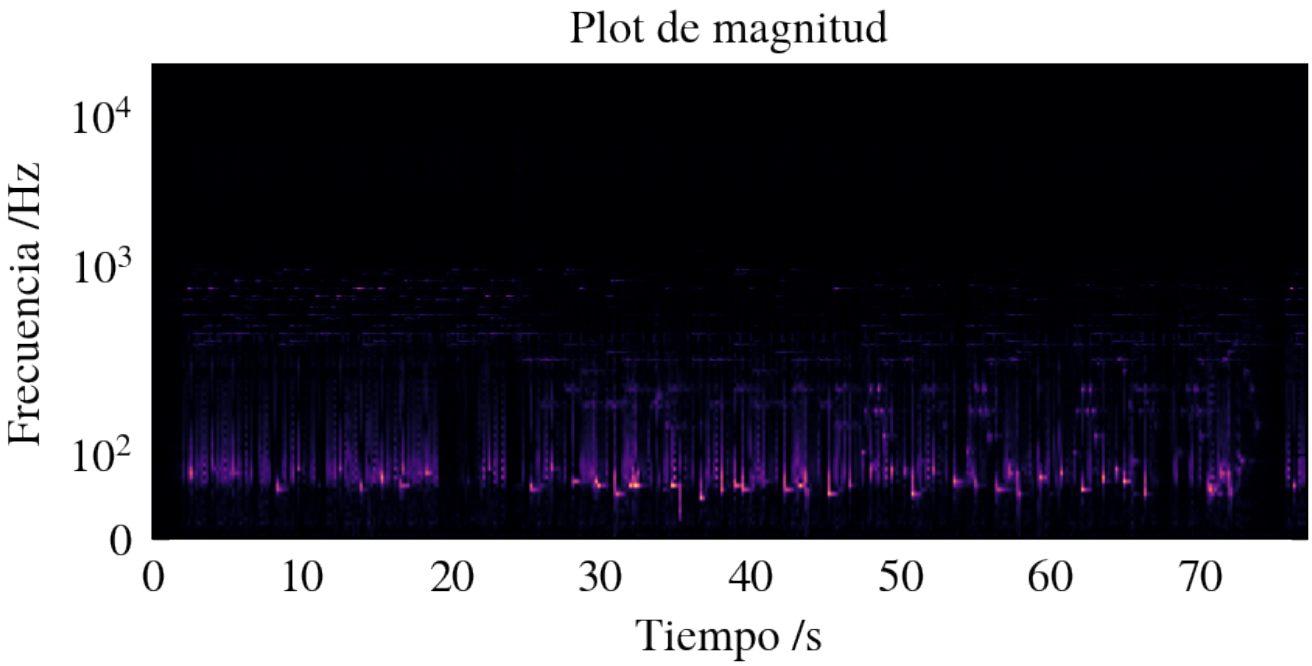
X. CONCLUSIÓN

No hizo falta implementar una *window-function* de manchado espectral, dado que funcionó relativamente bien para los casos presentados. Invito al lector a ver el archivo `.html` que demuestra las grabaciones y las canciones que se logra identificar. Se utilizó el algoritmo de puntajes dado que uno con hashes no funcionó como se pensaba, y el relacionado a *Fast Combinatorial Hashing* necesitaba más tiempo para refinar. Dado esto, las limitaciones del programa son meramente de memoria (sólo para la creación de bases de datos). Se aconseja utilizar un lenguaje compilado y *bitwise operations* para reducir la carga del programa.

[1] Will Drevo. Audio fingerprinting with python and numpy.
[2] Victor E Cameron. 10.2: Sampling theorem, Feb 2021.
[3] Christophe Kalenzaga. How does shazam work, Aug 2015.
[4] Wikipedia contributors. Audio bit depth — Wikipedia, the free encyclopedia, 2021.
[5] A. David Wunsch. *Complex variables with applications*. Pearson, 2005.
[6] Numpy reference, Jun 2021.
[7] Lars Wanhammar. 3 - Digital Signal Processing. In Lars Wanhammar, editor, *DSP Integrated Circuits*, Academic Press Series in Engineering, pages 59–114. Academic Press, Burlington, 1999.
[8] Scipy documentation, Aug 2021.
[9] Wikipedia. Manchado espectral — wikipedia, la enciclopedia libre, 2019. [Internet; descargado 18-noviembre-2021].
[10] Avery Wang. An industrial strength audio search algorithm. *Shazam Entertainment, Ltd.*, 2003.

Apéndice A: Espectrogramas y otras imágenes





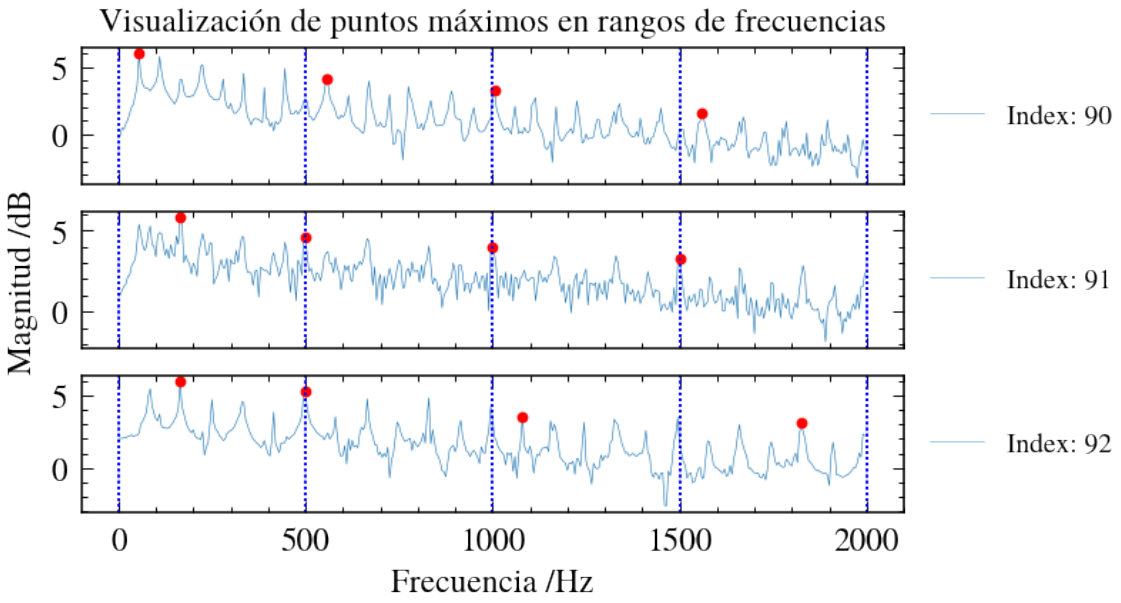


Figura 6. Un punto máximo por cada intervalo

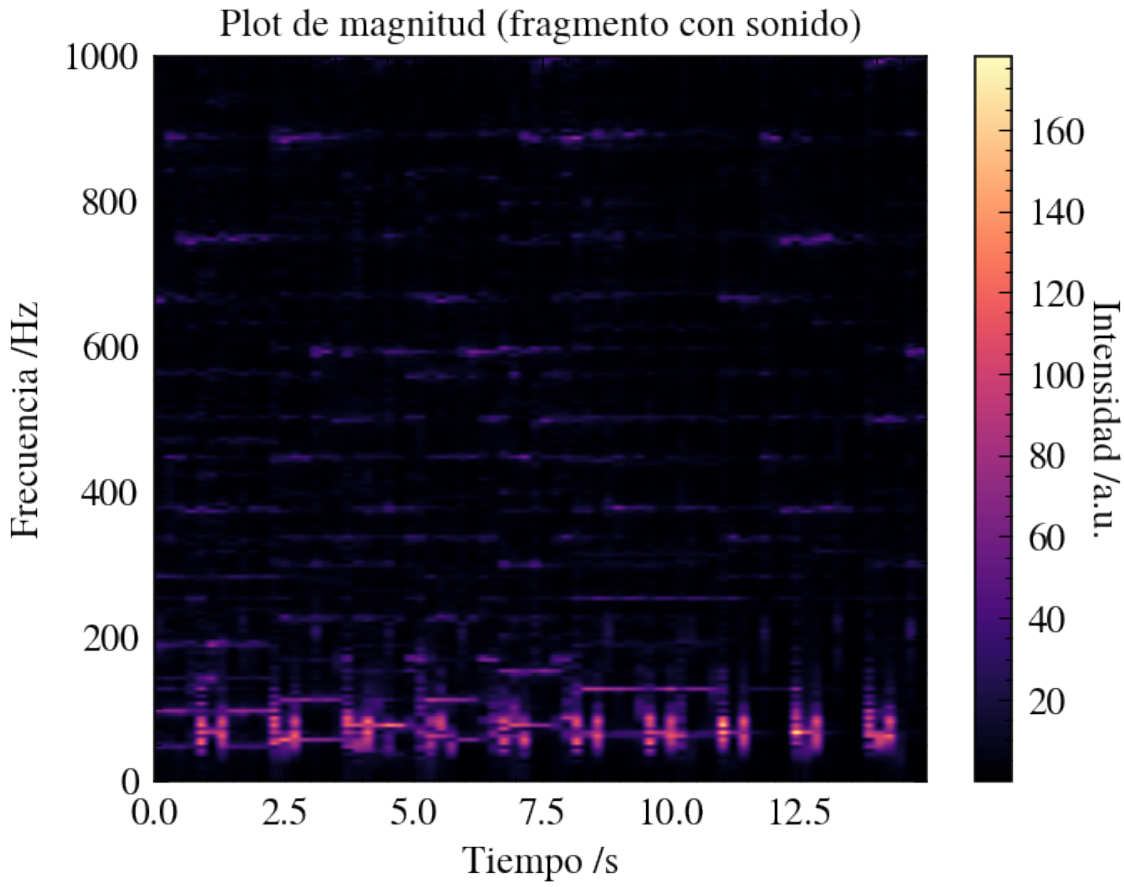


Figura 7. Espectro para una muestra