DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico

Experimentaciones sobre señales de audio

Procesamiento de Señales

Segundo Cuatrimestre de 2019

Integrante	LU	Correo electrónico
Balboa, Fernando	246/15	fbalboa95@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

1. Introducción

1.1. Resumen

El presente trabajo tratará sobre el procesamiento de diferentes muestras de audio. La idea del mismo es aplicar conceptos, algoritmos y técnicas vistas a lo largo de la materia en el campo de la acústica y observar los resultados.

Los experimentos propuestos consisten, por ejemplo, en modificar la frecuencia de una muestra, o utilizar convolución para caracterizar sistemas LTI (*Linear Time-Invariant System)*, entre otros. Las muestras utilizadas serán grabaciones en formato *.wav*, provistas por terceros.

1.2. Uso del código

El código de los experimentos asociados a este trabajo se encuentra adjunto en el repositorio. Para ejecutarlo, es necesario instalar *Jupyter Notebook* y las librerías necesarias (*NumPy, SciPy y matplotlib*). La forma recomendada de instalación es *Anaconda*, cuya distribución incluye todo lo mencionado. El código de los experimentos está en el archivo *code/Signals.ipynb*.

1.3. Conceptos básicos

El sonido puede ser analizado de la misma forma que cualquier otro tipo de señal, dado que es una onda con cierta frecuencia y amplitud, en general ambas variables a lo largo del tiempo. Estas ondas pueden ser analizadas tanto en el dominio del tiempo como en el de la frecuencia. El pasaje del primero al segundo puede lograrse utilizando la transformada de Fourier, y para volver, se utiliza su inversa. En particular, en este trabajo se utiliza un algoritmo eficiente que computa este cálculo, conocido como la transformada de Fourier rápida (*Fast Fourier Transform* o *FFT* en inglés).

Cada onda puede ser descompuesta en una suma de ondas sinusoidales de distinta frecuencia. Este procedimiento es conocido como *descomposición espectral*, y puede ser aprovechado por diferentes técnicas para lograr ciertos fines. Entre los más comunes en el campo de la música se encuentran el de filtrado de ciertas frecuencias y el cambio de timbre.

2. Experimentación

2.1. ThinkDSP

Para realizar los experimentos, se utilizó una librería del lenguaje *Python* que provee abstracciones que modelan señales y conceptos relacionados a las mismas, junto con funciones para manipularlas fácilmente. Dicha librería es parte de un libro llamado *ThinkDSP*, de Allen Downey, en el cual se explican los conceptos vistos en la materia y se aplican a diversos experimentos utilizando programación. El libro está diseñado para ser utilizado junto con el código provisto en una notebook de *Jupyter*, que permite mezclar texto con gráficos y audios fácilmente. Por esta razón, los experimentos serán presentados también en este formato, junto con los gráficos correspondientes. A continuación, se muestran algunas de las clases y funciones más relevantes utilizadas en este trabajo.

2.1.1. Signal

La clase *Signal* representa, como el nombre indica, una señal, y es la clase padre de muchos otros tipos de señales. Para crear una señal, hay que instanciarla utilizando los siguientes parámetros:

- *amp*: la amplitud de la señal, sin ninguna unidad de medición. Se asume que 1.0 es la máxima amplitud que puede registrarse.
- *freq*: la frecuencia de la señal en *Hertz* (*Hz*).
- *offset*: es el offset de fase, que determina en qué momento del período comienza la señal.

Esta clase es una representación en *Python* de una función matemática, por lo que en la mayoría de los casos, está definida para todo valor, desde $-\infty$ hasta $+\infty$. En realidad, cuando se trabaja con audio, se analizan ondas más que señales. Una señal puede ser convertida a una onda evaluándola en una secuencia de puntos en el tiempo. Cada punto en el tiempo de dicha medición se conoce como *frame* o *sample*.

2.1.2. Wave

La clase *Wave* representa exactamente una onda. Para convertir una *Signal* a una *Wave*, se puede utilizar el método *make_wave*, que toma como parámetros:

- *duration*: la duración de la onda.
- *start*: el segundo en el cual empieza la onda.

- *framerate*: un entero que indica el número de frames por segundo. Como referencia, un archivo *WAV* suele tener 44100 samples por segundo.

2.1.3. Spectrum

Como se mencionó en la introducción, las señales suelen ser analizadas tanto en el dominio del tiempo (representado por *Wave*) como en el de la frecuencia (representado por *Spectrum*). Para pasar del dominio del tiempo al de la frecuencia, se puede utilizar la *FFT*, que está encapsulada en el método *make_spectrum* de la clase *Wave*. El resultado de esta función es un objeto de la clase *Spectrum*. Para hacer lo inverso, se puede usar *make_wave* sobre un *Spectrum*. Esta clase provee métodos para manipular las frecuencias fácilmente. Por ejemplo, se puede aplicar un *low_pass* para permitir sólo el paso de frecuencias más bajas que cierto threshold.

2.2. Experimentos

2.2.1. Framerate

En música, una octava es el intervalo entre dos notas donde la segunda tiene el doble de frecuencia que la primera. Por ejemplo, en orquestas suele afinarse con un La en 440*Hz* (A4 en inglés). Al subir una octava, se llega a A5, que tiene exactamente el doble de frecuencia (880*Hz*).

El primer experimento consistió entonces en alterar el framerate de dos muestras distintas de una voz. Para la primera, se redujo a la mitad. Esto produce que la frecuencia disminuya a su vez a la mitad. El efecto audible de este experimento es que la voz se escucha una octava más abajo de lo que fue grabada. Además, la velocidad a la cual suena la palabra también se reduce a la mitad.

Para la segunda muestra, el experimento fue duplicar el framerate, y por lo tanto la frecuencia. Esto produce el efecto contrario que en el primer caso. El tono de la voz sube una octava, escuchándose considerablemente más agudo y rápido.

2.2.2. Filtros de frecuencias

Para este experimento, primero se generaron tres señales periódicas con diferentes frecuencias, donde cada una se correspondía con una nota musical diferente (A4, D5 y A5). Luego, se mezclaron las tres para formar una única señal compuesta.

Al graficar el espectro de esta nueva señal, se vieron claramente tres picos de amplitud en las frecuencias 440*Hz* (A4), 587.33 *Hz*(D5) y 880*Hz* (A5). Luego, se aplicó un filtro para eliminar las frecuencias inferiores a 500*Hz*, conocido como *high pass filter*, dado que sólo permite el paso de frecuencias mayores a un threshold. De esta forma, se eliminó A4 de la onda mixta. Por último, se aplicó un filtro para atenuar las frecuencias mayores a 800*Hz*. Dado que el mix estaba compuesto por una sola señal mayor a esta frecuencia, sólo se alteró la amplitud de la nota A5. Al escuchar el resultado luego de los filtros, se nota fácilmente que efectivamente A4 no está presente, y D5 predomina sobre A5, que fue atenuada.

2.2.3. Caracterización de LTIs

En el contexto de procesamiento de señales, un *sistema* es una representación abstracta de algo que toma una señal de input y produce otra de output. Un ejemplo de un sistema puede ser una habitación, que toma como input el sonido en el lugar en el que se produce, y produce como output un sonido diferente en el lugar donde se escuche.

Un *Linear Time-Invariant System* (*LTI*) se define como un sistema que cumple dos propiedades:

1. **Linearidad**: si dos inputs distintos entran al sistema al mismo tiempo, el resultado es la suma de sus outputs. Matemáticamente, si un input x_1 produce un output y_1 y otro input x_2 produce y_2, entonces $ax_1 + bx_2$ produce $ay_1 + by_2$, donde a y b son escalares.

2. **Invariancia en el tiempo**: el efecto del sistema no varía en el tiempo, ni depende del estado del sistema. Si los inputs x_1 y x_2 difieren por un intevalo de tiempo, sus outputs y_1 e y_2 difieren por el mismo intervalo, pero son idénticos en todo otro aspecto.

Utilizando estas propiedades, el siguiente algoritmo computa el efecto de un sistema *LTI* sobre una señal:

1. Expresar la señal como una suma de señales sinusoidales.

2. Para cada componente de input, calcular el output correspondiente.

3. Sumar los outputs.

Encontrar el efecto que produce un sistema sobre cada componente de la señal de input se conoce como **caracterización**. Para aplicar el algoritmo mencionado anteriormente, primero es necesario caracterizar el sistema *LTI*. En el caso de sistemas mecánicos, como una habitación, una posiblidad es generar un **impulso** y grabar la respuesta.

Si se calcula el espectro de un impulso, se ve que consiste en la suma de componentes donde todas tienen la misma magnitud. Al testear el sistema con un impulso, se hace para todas las frecuencias al mismo tiempo. La linearidad del sistema asegura que los tests no interfieren entre sí.

El experimento realizado consistió entonces en caracterizar una habitación con eco a partir de una grabación de un impulso en la misma, y aplicar la transformación del sistema a otra grabación de un instrumento. El impulso en cuestión utilizado es un sólo sonido seco (tal vez un aplauso o un disparo) que retumba en la habitación. La grabación corresponde en realidad a la respuesta de dicho impulso, y contiene en su espectro toda la información necesaria para caracterizar la habitación.

2.2.4. Fades

El último experimento consistió en aplicar dos efectos muy comunes en la música, conocidos como fade-in y fade-out, a una grabación de estudio de una canción. El fade-in se utiliza para suavizar la entrada de uno o varios instrumentos, haciendo que el sonido aumente de forma gradual hasta alcanzar su nivel normal. En general, el volumen comienza en 0 y aumenta lineal o cuadráticamente durante un intervalo de tiempo hasta quedar constante. El fade-out es análogo al fade-in, y se suele utilizar para suavizar el

cierre de canciones, donde los instrumentos siguen sonando hasta el final, cada vez con menor volumen, hasta volverse inaudibles.

Para aplicar el fade-in lineal, se multiplicó la amplitud de cada sample de la onda durante cierto intervalo de tiempo por un número entre 0 y 1, de forma linearmente creciente. Por ejemplo, si suponemos que el fade-in dura 3 segundos, y el audio tiene 44100 frames por segundo, entonces se tiene un total de 3 * 44100 = 132300 muestras. La amplitud de la primera muestra sería multiplicada por un factor de 0, volviéndola inaudible. Si se tratara de un fade-in lineal como el implementado en el experimento, la muestra del medio (66150) sería multiplicada por un factor de 0.5, y la última (132300) por 1, dejándola igual a la muestra original.

Para la implementación, se hizo uso de la función *linspace* de la librería *NumPy*, que genera un vector con cierta cantidad de elementos entre uno inicial y otro final, con intervalos regulares. En el caso del fade-in, los números fueron entre 0 y 1 (creciente), y para el fade-out, entre el 1 y el 0 (decreciente). De hecho, la implementación del fade-out se realizó a partir de la del fade-in. Al ser el efecto contrario, el vector utilizado para aplicar el efecto es exactamente el inverso al del fade-in. También fue necesario tener en cuenta que los fades deberían dejar igual la porción del audio que no afectan, por lo que fue necesario agregar un padding de 1s hasta alcanzar la misma longitud que la del track completo antes de hacer la multiplicación entre la grabación y el vector de fade. Al multiplicar cada amplitud por 1, la muestra queda exactamente igual a la original.