TRABAJO PRÁCTICO 2

FUNDAMENTOS MATEMÁTICOS DEL PROCESAMIENTO DE SEÑALES

23 Diciembre 2021

El objetivo de este trabajo práctico es manipular y comprender las herramientas de procesamiento de señales digitales vistos en el primer capítulo del curso. No programaremos algoritmos por el momento, solo haremos visualizaciones de señales en el dominio espectral y frecuencial.

Este trabajo práctico se realizará en Python 3 y se aconseja fuertemente utilizar el entorno de programación Anaconda e implementar las diferentes secciones en Jupyter Notebooks, con comentario y notas en cada ítem.

Se espera que el código entregado pueda ser descargado desde github y ejecutado (figuras reproducibles) y que cada resultado este acompañado de una breve discusión en formato *markdown*, el cual permite escribir ecuaciones dentro de Jupyter.

En este trabajo dirigido usaremos bibliotecas Numpy/Scipy para manipular datos numéricos y Matplotlib para graficarlos.

También necesitaremos tener acceso a algunos métodos de las librerias scipy.signal y scipy.io. El submódulo wavfile que también debe importarse.

```
import numpy as np
import scipy as sp
import scipy.signal
import scipy.io.wavfile
```

1 Generación y muestreo de señales

En esta sección generaremos algunas señales digitales y veremos el efecto del muestreo en términos de reconstrucción. Finalmente veremos cómo generar señales de audio y guardarlas en formato *.wav*.

1.1 Generación de señales

Generaremos muestras a partir de la siguiente señal continua

$$x(t) = \sin(2\pi f_0 t) + \cos(2\pi f_1 t) \tag{1}$$

donde $f_0 = 2\text{Hz}$ y $f_1 = 3f_0$.

1. Implemente una función `def` $x(t)$ que devuelve los valores de $x(t)$ de una lista de valores en formato numpy.
2. Genere una señal con un muestreo fino a una frecuencia de muestreo $f_{s0} = 1000\text{Hz}$ sobre $N_0 = 1024$ muestras:
   - Genere un arreglo t_0 de N muestras temporales de valores $t = \frac{n}{f_{s0}}$.
   - Evalue la función x en el vector de tiempo t_0 y almacene los valores resultantes $x[n]$ en un vector x_0.
   - Trace la señal con el eje de tiempo correcto en segundos.
3. Genere una señal x_n con una frecuencia de muestreo $f_s = 20$ sobre $N = f_s$ muestras (muestreo de 1 segundo).
4. Grafique simultáneamente x_0 y x_n. Para x_n, use el estilo de trazado "-o" para ver la posición de las muestras.

1.2 Reconstrucción de señales

1. ¿Cuál es la frecuencia de muestreo necesaria f_s para garantizar que la señal $x(t)$ pueda reconstruirse?
2. Programe una función `def recSinc` (x_s, t_s, f_s, t) que reconstruya una señal en el tiempo t a partir de muestras x_s, t_s a la frecuencia f_s.
3. Grafique simultáneamente x_0 y la interpolación de x_n en t_0. ¿Qué pasa en el borde de la ventana de muestreo?
4. Modifique la frecuencia de muestreo de $f_s = 20$ a $f_s = 10$. ¿Qué sucede con la reconstrucción?

1.3 Generación de señales de audio

En esta parte trabajaremos con secuencias de audio. Para hacer eso usaremos `scipy.io.wavefile` para cargar y guardar archivos en formato *.wav*. También es posible escuchar audio directamente en Python usando la librería de manipulación audio `sounddevice` que se puede instalar con *pip* o *conda*. En esta sección las señales generadas solo serán escuchadas y trazadas, pero estudiaremos sus componentes de frecuencia en la siguiente sección.

1. Genere 1 segundo de una onda sinusoidal de magnitud 0.5 y de frecuencia $f_0 = 425$Hz muestreada a $f_s = 8000$Hz. Guárdelo como un archivo wave y escuche el archivo o escuche directamente la señal desde python. Este es el tono de marcado de los teléfonos europeos.
2. Es posible generar notas musicales a partir de su número MIDI m donde la frecuencia se expresa como

$$f_m = 440 \cdot 2^{\frac{m-69}{12}} .$$

   Vemos que hay 12 semitonos para pasar de una nota a su octava.

   La nota MIDI $m = 69$ es la LA 4 en notación y es el estándar de tono utilizado para afinar instrumentos para conciertos.

   La lista de notas y sus correspondientes nombres y frecuencias puede encontrarse en línea[1].

   Implemente una función `def getNote (m, fs, l)` que devuelve la nota m tocada durante l segundos a la frecuencia f_s.
3. Guarde la nota $m = 69$ en un archivo llamado"A4.wav". Escuche varias otras notas MIDI. ¿Qué pasa para $m = 117$ (LA8) cuando se guarda a la frecuencia de muestreo $f_s = 8000$Hz?
4. Codifique una secuencia de concatenación de notas $[70, 72, 68, 56, 63]$ (1 segundo cada una) y guarde la secuencia como archivo "seq.wav". ¿Le suena la secuencia?
5. La saturación puede ocurrir cuando los amplificadores alcanzan su máxima amplitud. El efecto de la saturación puede ser reproducido usando un "clipeo" en un seno.

   Compare la señal de la nota $m = 69$ a 440Hz para diferentes valores de recorte. Guarde la nota usando el "clipeo" en un archivo "A4clip.wav". ¿Cual es el efecto de la saturación en el contenido frecuencial de la señal?
6. Genere la señal

$$x(t) = \sin\left(2\pi\left(\omega_0 t + \frac{c}{2}t^2\right)\right)$$

   con $f_0 = 100$Hz y $c = 500$ durante 1 segundo a una frecuencia de muestreo $f_s = 8000$Hz. Esta señal es llama "chirp" y corresponde a una modulación de frecuencia. Guarde la señal en el archivo "chirp.wav".

[1] https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies

2 Transformada de Fourier discreta (DFT) y Transformada de Fourier Rápida (FFT)

2.1 Funciones básicas y matriz de transformada discreta de Fourier

2.2 Transformada de Fourier Rápida (FFT)

3 Interpretación de las señales

En esta sección estudiaremos varios filtros digitales y los aplicaremos a las señales.

3.1 Filtraje Ideal

1. Cargue la señal en el archivo "stairwayb.wav". Intentaremos atenuar el ruido presente en el audio cortando toda la banda de frecuencia donde hay ruido.

2. Calcule la FFT de la señal y grafique su magnitud en el dominio de Fourier. Seleccione una frecuencia de corte f_c para un filtro de paso bajo ideal.

3. Aplique un filtro ideal "low-pass" con una frecuencia de corte f_c. Escuche la señal filtrada. Tenga en cuenta que guardar un archivo wav en formato flotante recorta los valores entre -1 y 1, por lo que la señal se debe escalar correctamente para evitar la saturación.

4. Utilice un filtro ideal para seleccionar solo la nota con la frecuencia más baja en la señal "seq.wav". Escuche la señal filtrada para comprobar que solo queda una nota.

5. Calcule la transformada de Fourier inversa de la respuesta de frecuencia ideal del filtro de paso bajo para obtener su respuesta de impulso de convolución circular. Haga lo mismo con el filtro de paso alto ideal. Compruebe ambos que la frecuencia 0 se pasa y se corta respectivamente calculando la ganancia estática.

3.2 Diseño de filtros digitales

En aplicaciones de la vida real, a menudo es necesario diseñar filtros causales. Esto se hace estimando coeficientes para un Filtro FIR o IIR de orden finito que se aproxima a sistemas de tiempo continuo como Butterworth de Chebychev filtros.

1. Calcule los coeficientes de un filtro Butterworth FIR discreto para una frecuencia de corte normalizada de $f_c = 0.2$ (para una frecuencia de muestreo de 1) de orden $n = 2$.

2. Implemente una función `def freqResp (a, b, f)` que devuelve la respuesta de frecuencia a un filtro IIR a, b para obtener una lista de frecuencias f. Grafique la respuesta de frecuencia para el filtro Butterworth de órdenes $n = 1, 2, 3, 4$.

3. Aplique el filtro a la señal "stairwayb.wav". ¿Cual es el equivalente frecuencia de corte en Hz? ¿En qué orden el filtro es lo suficientemente fuerte como para atenuar bien el ruido?

4. Calcule los coeficientes de un filtro Chebychev FIR discreto de tipo 1 para una frecuencia de corte normalizada de $f_c = 0.2$ (para una frecuencia de muestreo de 1), de orden $n = 2$ y permitiendo ondulaciones de 1dB en el paso de banda. Grafique la respuesta de frecuencia de ambos filtros Buterworth y Chebychev del mismo orden en el misma gráfico.

5. Aplique el filtro Chebychev a la señal con ruido "stairwayb.wav". ¿Qué sucede con un orden de $n = 50$? ¿Ocurre lo mismo con el filtro Butterworth de igual orden?

3.3 Separación de fuentes y eliminación de ruido

1. Diseñe filtros IIR o ideales que permitan una separación grosera de las diferentes fuentes en señales: "drum.wav", "seq.wav".

2. Diseñe filtros para las señales "ecg.npz" y "conso.npz" para apreciar mejor la señal primaria (respectivamente los latidos del corazón y el uso de energía).