**Filtros 1D**

APELLIDOS, NOMBRE: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Objetivo

El objetivo de esta práctica es iniciarse en el empleo de técnicas de filtrado de señales en tiempo y en frecuencia.

La convolución discreta (suma de convolución) es una operación matemática que puede implementarse de forma sencilla en cualquier dispositivo de procesamiento digital (ordenador, microprocesador, etc.). Es por ello que la práctica comenzará realizando una función que implemente esa operación. Esa función será posteriormente utilizada para reducir el ruido de una señal.

La transformada discreta de Fourier (DFT) permite observar las componentes frecuencias de una señal. A partir de esa representación, es posible realizar el filtrado realizando la multiplicación de la DFT de la señal y la DFT del filtro.

Para realizar la práctica, se utilizará código que ya fue desarrollado en sesiones anteriores de teoría o de prácticas:

* Generación de señales básicas.
* Generación de tonos musicales como suma de cosenos.
* Generación de ruido gaussiano.

**PARTE 1**

EJERCICIO 1: Implementar la suma de convolución

Utilizando un bucle, realice una función (llamada conv) que realice la suma de convolución definida como sigue:

Tenga en cuenta lo siguiente:

* Para obtener puede utilizar list(reversed(h))
* Antes de realizar la convolución, es necesario añadir ceros antes y después de los valores de forma que se obtenga un vector extendido:

lxe = lx+lh

xe = [0] \* lxe

xe[lh-1:lh-1+lx] = x

* La señal de salida tendrá tamaño siendo la longitud de y la longitud de Es conveniente inicializar el vector de salida utilizando:

ly = lx+lh-1

y = np.zeros(ly)

* En cada iteración del bucle (es decir, para cada valor de ), se calculará el valor de como la suma del resultado de multiplicar valores de por todos los valores de .

El siguiente código genera dos señales tipo pulso y las convoluciona entre sí utilizando scipy.convolve. Añada el código que ha realizado y compruebe que obtiene la misma salida.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  import scipy  "================================================="  " Convolución de señales "  tipoconv = 1  x = np.ones(10)  Lx = len(x)  nx = np.arange(0,Lx)  h = np.ones(3)  Lh = len(h)  nh = np.arange(0,Lh)  if tipoconv == 1:  y = scipy.convolve(x,h)  elif tipoconv == 2:  "Añada su código"  Ly = len(y)  ny = np.arange(0,Ly)  plt.subplot(311)  markerline, stemlines, baseline = plt.stem(nx, x, '-.')  plt.subplot(312)  markerline, stemlines, baseline = plt.stem(nh, h, '-.')  plt.subplot(313)  markerline, stemlines, baseline = plt.stem(ny, y, '-.')  plt.show() |

|  |
| --- |
| La convolución de dos señales no es simplemente la multiplicación de las amplitudes de las señales. Lo que sucede en realidad es que la convolución es una operación matemática que combina información de ambas señales para generar una señal de salida nueva.  En la convolución, se recorre cada punto de la señal de entrada y se multiplica su valor por los valores de la otra señal en diferentes puntos. Luego, los productos resultantes se suman y se almacenan en la señal de salida. Este proceso se repite para cada punto de la señal de entrada, y cada resultado se suma para generar la señal de salida completa.  Por lo tanto, si las dos señales de entrada tienen amplitud 1, esto no significa necesariamente que la amplitud de la señal de salida resultante también sea 1. El valor de amplitud de la señal resultante dependerá del contenido de frecuencia de las señales de entrada y de cómo se superpongan.  <https://dademuch.com/2020/09/14/sumatoria-de-convolucion-convolucion-en-tiempo-discreto/>  **Copie aquí una figura donde se represente**  **y**    **Copie aquí una figura donde se represente la señal obtenida con la función de Python y otra con la suya.**    **¿Cuántos puntos distintos de cero tiene ? ¿Cómo se puede calcular a partir de** *x* y **de** **?**  10 puntos, se puede calcular a partir de la siguiente formula:  Longitud de y(n) = Longitud de x(n) + Longitud de h(n) -1  **¿Dónde comienza ? ¿Cómo se puede calcular a partir de** **y de ?**  Empieza en 0, ya que una de las señales comienza en 0  Haciendo la suma de las convoluciones  **¿Cuál es la amplitud de ? ¿Cómo se puede calcular a partir de** **y de ?**  Amplitud de x(n) \* Amplitud de h(n) \* Longitud de la señal más pequeña  Al ser dos señales unitarias, se puede usar la formula. |

Realice ahora la convolución entre y

.

|  |
| --- |
| **Copie aquí una figura donde se represente**  **y**    **Copie aquí una figura donde se represente la señal obtenida con la función de Python y otra con la suya.**    **¿Cuántos puntos distintos de cero tiene ? ¿Cómo se puede calcular a partir de** *x* y de **?**  Tiene 7 puntos distintos de ceros y se pueden calcular con la siguiente formula, restándole 1:  Longitud de y(n) = Longitud de x(n) + Longitud de h(n) -1  **¿Dónde comienza ? ¿Cómo se puede calcular a partir de** **y de ?**  Empieza en 0  y(n) se puede calcular haciendo la suma de las convoluciones entre x(n) y h(n)  y(n) = x(n)\*h(n)    **¿Cuál es la amplitud de ? ¿Cómo se puede calcular a partir de** **y de ?**  La Amplitud de y(n) se puede calcular:  Amplitud de x(n) \* Amplitud de h(n) \* Longitud de la señal más pequeña |

EJERCICIO 2: Filtrado de media

En este apartado se realizará un filtro para minimizar el impacto que el ruido tiene en una señal. Debe generar la señal utilizando lo que ya ha realizado en la PRÁCTICA 1 (señales),

A esa señal, súmele ruido gaussiano como se muestra a continuación:

xn = np.random.normal(media, sigma, lx)

x = xcos + xn

Para reducir el ruido, se utilizará un filtro de media que tiene como respuesta al impulso Lh valores iguales a 1 (es decir, es un pulso de longitud Lh ).

Dado que el filtro es un sistema LTI, la salida será la convolución de y (es decir, la salida se obtendrá utilizando la función conv que ha desarrollado en el EJERCICIO 1). Para comprobar que funciona correctamente, utilice también scipy.convolve.

Para realizar las simulaciones, considere los siguientes parámetros:

valini = 0 # Valor inicial

lx = 512 # Num muestras

fs = 22100 # Frecuencia de muestre oHz

f = 100 # Frecuencia fundamental

Na = 2 # Numero de armonicos

media = 0 # Media del ruido

sigma = 0.2 # Desviacion tipica

|  |
| --- |
| **Para** lh=10:   * **Copie aquí una figura donde se represente y** * **Copie aquí una figura donde se represente la señal**   **Para** lh=50:   * **Copie aquí una figura donde se represente y** * **Copie aquí una figura donde se represente la señal**   **¿Qué efecto tiene aumentar lh?**  **¿Qué efecto tiene aplicar un filtro de media?** |

**PARTE 2**

EJERCICIO 3: DFT

El siguiente código calcula la DFT de una señal y representa la parte correspondiente a las frecuencia positivas (en Hz).

|  |
| --- |
| from scipy.fft import fft, fftfreq  N = lx # Numero de puntos  T = 1.0 / fs # Separacion entre puntos  Xf = fft(x)  frec = fftfreq(N, T)  plt.plot(frec[:N//2], 2.0/N \* np.abs(Xf[0:N//2]))  plt.grid()  plt.show() |

Utilizando ese código, calcule y represente la DFT de la señal EJERCICIO 2 SIN ruido utilizando como frecuencia fundamental **f = 1000 Hz.**

|  |
| --- |
| **Copie la figura de la DFT**  **¿Cuántas señales “delta” aparecen en frecuencia? ¿En qué frecuencias están?** |

Utilizando ese código, calcule y represente la DFT de la señal EJERCICIO 2 CON ruido utilizando como frecuencia fundamental **f = 1000 Hz.**.

|  |
| --- |
| **Copie la figura de la DFT**  **¿Cuántas señales “delta” aparecen en frecuencia? ¿En qué frecuencias están?**  **¿Qué efecto tiene el ruido?** |

EJERCICIO 4: Filtrado en frecuencia

Para realizar el filtrado en frecuencia, puede emplearse un filtro ideal que tiene 1s en las frecuencias deseadas y 0s en el resto. El siguiente código permite generar un filtro paso bajo ideal que deja pasar solamente la componente de la frecuencia fundamental:

|  |
| --- |
| fcorte = 1500  HPB = (frec > -fcorte) & (frec < fcorte) |

Para aplicar ese filtro realizaremos los siguiente:

|  |
| --- |
| Yf = np.multiply(Xf, HPB)  y = ifft(Yf)  ly = len(y)  ny = np.arange(0,ly)/fs |

Para hacer la representación, puede utilizar

|  |
| --- |
| Yf = npplt.plot(frec[:N//2], 2.0/N \* np.abs(Yf[0:N//2]))  plt.grid()  plt.show()  plt.subplot(313)  markerline, stemlines, baseline = plt.stem(ny, y, '-.')  plt.show() |

Ejecute el código para dejar pasar solamente la señal con la frecuencia fundamental **f = 1000 Hz.**.

|  |
| --- |
| **Copie la figura de la DFT de la salida** |

Cambie el filtro para que deje pasar el coseno de la frecuencia fundamental y el primer armónico.

|  |
| --- |
| **Copie la figura de la DFT de la salida** |

Cambie el filtro para que deje pasar SOLAMENTE el último armónico. Tenga en cuenta que ahora no puede utilizar un filtro paso bajo.

|  |
| --- |
| **Copie la figura de la DFT de la salida** |