Manual de Usuario: Técnicas de Detección de Espectro

Luis Alberto Ochoa Marín
lalberto.ochoa@udea.edu.co

Contenido

1 Introducción

1.1 Objetivo

Este manual proporciona instrucciones detalladas para ejecutar y comprender dos técnicas de detección de espectro: Detección de Energía y Detección basada en Wavelet, implementadas en los códigos EnergyDetection.py y WaveletDetection.py, respectivamente.

1.2 Herramientas Necesarias

Para ejecutar los códigos, asegúrese de tener instalados los siguientes elementos:

- Python y Spyder IDE.
- Librerías: NumPy, Matplotlib, Pandas, PyWavelets, y SciPy.

2 Instalación de Herramientas

2.1 Instalación de Python y Spyder IDE

- Descargue e instale Python desde `https://www.python.org/`.
- Descargue e instale Anaconda, la cual tiene incluida dentro de susaplicaciones Spyder IDE.

  Se recomienda realizar la descarga de la última versión disponible del Anaconda usando el siguiente enlace: `https://www.anaconda.com/download`. Posteriormente se deberá entrar mediante CLI (Command Line Interface) al directorio en el cual se encuentre ubicado el fichero descargado y se validará su correcta descarga mediante los comandos md5sum y sha256sum. Finalmente se debe ejecutar el fichero descargado mediante el comando que se describen abajo y seguir los pasos de instalación.

  ```
  cd Downloads/
  md5sum Anaconda3-2023.09-0-Linux-x86_64.sh
  sha256sum Anaconda3-2023.09-0-Linux-x86_64.sh
  bash ./Anaconda3-2023.09-0-Linux-x86_64.sh
  ```

  Por favor, asegúrate de que estás en el directorio correcto y que el archivo Anaconda está presente antes de ejecutar estos comandos.

2.2 Librerías Necesarias

Después de tener instalado Anaconda, abra Spyder IDE e instale las librerías necesarias ejecutando el siguiente comando en la consola:

```
pip install numpy matplotlib pandas pywavelets scipy
```

3 Código EnergyDetection.py

3.1 Descripción

El script `EnergyDetection.py` utiliza la técnica de Detección de Energía para analizar un archivo binario de datos de espectro. Identifica la ocupación del canal mediante el análisis de la potencia promedio en ventanas espectrales, determinando la presencia de portadoras a través de un umbral predefinido. El resultado incluye gráficas que representan el espectro y se genera un DataFrame con información detallada sobre las detecciones.

3.2 Ejecución

Para ejecutar el código, siga los siguientes pasos:

- Abra Anaconda Navigator y lance Spyder IDE.
- Dentro de Spyder, navegue hasta el directorio que contiene el archivo `EnergyDetection.py` utilizando el explorador de archivos de Spyder.
- Abra el archivo `EnergyDetection.py`.
- Ejecute el código utilizando el botón "Run" o presionando F5.
- Consulte las gráficas y la salida generada por el código en la consola de Spyder.

4 Código WaveletDetection.py

4.1 Descripción

El script `WaveletDetection.py` utiliza la Transformada Wavelet para la detección de espectro. Analiza un archivo binario de datos de espectro y descompone la señal en varios niveles de descomposición, con el fin de identificar cambios bruscos de amplitud. Luego, a partir de la Densidad Espectral de Potencia (PSD) promedio obtenida de los niveles de descomposición, se detectan picos de potencia para estimar la ocupación del espectro. El resultado incluye gráficas que representan la señal original y las PSD en diferentes niveles de descomposición, así como un DataFrame con los valores de potencia y frecuencia correspondientes a los picos detectados.

4.2 Ejecución

Para ejecutar el código, siga los siguientes pasos:

- Abra Anaconda Navigator y lance Spyder IDE.
- Dentro de Spyder, navegue hasta el directorio que contiene el archivo `WaveletDetection.py` utilizando el explorador de archivos de Spyder.
- Abra el archivo `WaveletDetection.py`..
- Ejecute el código utilizando el botón "Run" o presionando F5.
- Consulte las gráficas y la salida generada por el código en la consola de Spyder.

5 Referencias

[1] Python. https://www.python.org/

[2] Spyder IDE. https://www.spyder-ide.org/

[3] Anaconda. https://www.anaconda.com/download

[4] NumPy. https://numpy.org/

[5] Matplotlib. https://matplotlib.org/

[6] Pandas. https://pandas.pydata.org/

[7] PyWavelets. https://pywavelets.readthedocs.io/

[8] SciPy. https://www.scipy.org/

6 Anexos

6.1 Anexo A. Código EnergyDetection.py

```
\begin{minted}{python}
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


"""
@author: Luis Ochoa
"""

import time
import pandas as pd
import pywt
from scipy.signal import find_peaks, decimate
import numpy as np
import os
import matplotlib.pyplot as plt
import scipy.signal as signal

# Cargar los datos desde el archivo binario
loaded_data = np.fromfile('/home/luis/Grupo de Investigación
↪  GITA/Practicas_Academicas/bandwidth_spectrum_Prom.bin')
#4928
#2.1401923076923075 2.1537967032967034
# Graficar los datos
plt.plot(loaded_data)
plt.show()
print(len(loaded_data))

# Feecuencias extremas de la banda
index_start = 2140192307.6923075
index_end = 2153796703.2967034

def fftSt(st, Fs, AmpNor):
    """
    Parameters
    ----------
    st : Array of Complex
        Complex signal.
    AmpNor : Boolean
        Yes(True) or not(False) FFT with Amplitud Normalized.

    Returns
    -------
    Sshift : Array of Complex
        FFT with center fequency 0 Hz
    f : Array of Float
        axis x of spectrum signal.
    df : float
        f array resolution (delta f).

    """
    N = len(st)
    if AmpNor:
        S = np.fft.fft(st)/len(st)           # Normalize amplitude
    else:
        S = np.fft.fft(st)
```

```
    Sshift = np.fft.fftshift(S)
    df = Fs/N
    sampleIndex = np.linspace(-N/2, (N/2)-1, N)
    f = sampleIndex*df

    return Sshift, f, df


##############################################################################


pathCRC = "/home/luis/Grupo de Investigación GITA/Notebook Jupyter/"

#Se carga el archivo binario
file = 'LTE_B14.5M_Fc2.1475G.bin'
data = np.fromfile(pathCRC+file, dtype=np.complex64)

# Limitar los datos a las primeras 500,000 muestras
data_subset = data[:500000]

# Tamaño de la muestra
sample_size = 50000 - 1024

# Extraer las muestras
sample2 = data_subset[1024 + sample_size : 1024 + 2 * sample_size]
sample3 = data_subset[1024 + 2 * sample_size : 1024 + 3 * sample_size]
sample4 = data_subset[1024 + 3 * sample_size : 1024 + 4 * sample_size]
sample5 = data_subset[1024 + 4 * sample_size : 1024 + 5 * sample_size]

fftSize = 8192
samp_rate = 14.50e6
bw = samp_rate
fc = 2.14755e9

# Inicializar matriz para almacenar FFT de cada muestra
all_ffts = np.zeros((4, fftSize), dtype=np.complex128)

samples = [sample2, sample3, sample4, sample5]

# Procesar cada muestra y calcular su FFT
for i, sample in enumerate(samples):
    st = sample[:fftSize]  # Limitar el tamaño de la muestra a fftSize si es necesario
    fft_result, _, _ = fftSt(st, samp_rate, True)
    all_ffts[i] = fft_result

# Calcular el promedio de las FFT de las muestras
average_fft = np.mean(all_ffts, axis=0)

# Crear el eje de frecuencias
frequency_axis = np.linspace(fc - int(bw / 2), fc + int(bw / 2), fftSize)

fft_dB = 20 * np.log10(average_fft)

sint =  fft_dB

x_f = np.linspace(fc-int(bw/2),fc+int(bw/2),fftSize)
x_f_mhz = x_f / 1e6 # Convertir Hz a MHz

ventana=55 #Tamaño de la ventana
overlap=0  #Tamaño del solapamiento
```

```
size_ventana=0    #A partir de esta longitud se toma la nueva ventana en cada interacción
vect_ocupacion = []       #vector de ocupacón
umbral_spectrum=np.mean(sint) + 5 #Potencia promedio del especteo más 5 dB por encima del piso del
↪  ruido como lo recomienda la ITU
Delta=bw/(1e3*fftSize) #separación entre cada punto de la FFT en KHz

#Ptencia promedio del ruido en todo el espectro:  -86.20302343554795  (dBm)

print("Tamaño de la FFT:", fftSize, " --> ",bw/1e6, "MHz. "" --> ",Delta, "KHz entre muestras.")
print("Tamaño de la ventana:", ventana, " --> ",Delta*ventana/1e3, "MHz. ")
#print("Tamaño del solapamiento:", overlap," --> ",Delta*overlap/1e3, "MHz. " "\r\n")

dataFrame=[]            #Base de datos
datos=[]                #Datos por cada ventana
ventanas=[]             #Lista con los datos de potencia [dBm] de cada ventana temporalmente
vect_frecuencias=[]     #Lista con los datos de frecuencia[Hz] de cada ventana temporalmente
frecuencias=[]          #Lista de todas las frecuencias obtenidas en los puntos máximos de potencia
diff_umbral=[]
num_ventana=[]
i=1                     #Contador de ventanas

#Escala de la grafica en el eje Y (dBm)
y_max=-75
y_min=-125

while((size_ventana)<len(sint)):
    if size_ventana==0:
        ventanas= sint[0:ventana] #se toma el primer segmento de ventana sin traslape
        vect_frecuencias=x_f[0:ventana]
        umbral=np.mean(ventanas) + 4 #Potencia promedio de las ventanas más 4 dB
        vect_ocupacion = [0 if punto < umbral else 1 for punto in ventanas]    #Se mira cuantos
        ↪  puntos estan por encima del umbral "1" y cuantos no "0"
        num_ventana.append(i)        #se obtiene el # de la ventana y se agrega a una lista

        #se agrega al Dataframe las potencias que superen un umbral definido.
        if (np.max(ventanas)>=umbral):
            datos.append(i)
            datos.append(np.mean(ventanas))                          #Potencia promedio
            datos.append(umbral)
            datos.append(np.sum([n**2 for n in ventanas])/len(sint))#Potencia acumulada por ventana
            datos.append(np.max(ventanas))
            max_index = np.argmax(ventanas)                                  #Potencia Máxima
            datos.append(vect_frecuencias[max_index]) #se obtiene la frecuencia que corresponde al
            ↪  maximo
            datos.append(0)
            frecuencias.append(vect_frecuencias[max_index]) #también se guardar las frecuencias en
            ↪  un lista a parte
            dataFrame.append(datos)                                 #Se agrega los datos obtenidos de
            ↪  cada ventana al DataFrame

            plt.figure(figsize=(7,3))
            plt.scatter(vect_frecuencias[max_index],np.max(ventanas),color="r",marker="v",label =
            ↪  "P_Máxima") #se grafica el punto maximo de potencia
            plt.axhline(umbral,color='b', linestyle='--',label = "Umbral")   #se grafica el promedio
            ↪  de la potencia
            plt.axhline(np.mean(ventanas),color='y', linestyle='--',label = "P_Promedio")   #se
            ↪  grafica el promedio de la potencia
            plt.plot(x_f[0:ventana], sint[0:ventana],color="g",label = "Espectro_Signal")  #plot(x,
            ↪  y_FTT)
            plt.ylim([y_min,y_max])
```

```
            plt.xlabel('Frequency [Hz]')
            plt.ylabel('Power [dBm]')
            plt.legend(loc = "upper right")
            print("Ventana-> [{} - {}] MHz.".format(x_f[0]/1e6,x_f[ventana]/1e6)," -> Tamaño =
            ↪  {}".format(Delta*ventana/1e3), "MHz. ")
            #Frecuencia inicial y final de la ventana
            plt.grid()
            plt.show()

        i+=1
        size_ventana+=ventana-overlap

    elif(len(sint[size_ventana:size_ventana + ventana])==ventana and
    ↪  (size_ventana+ventana)<len(sint)-1):
        ventanas= sint[size_ventana:size_ventana + ventana] #se toma el segmento de ventana con
        ↪  traslape
        vect_frecuencias=x_f[size_ventana:size_ventana + ventana]
        umbral=np.mean(ventanas) + 4 #Potencia promedio de las ventanas más 4 dB
        vect_ocupacion += [0 if punto < umbral else 1 for punto in ventanas]    #Se mira cuantos
        ↪  puntos estan por encima del umbral "1" y cuantos no "0"
        max_index = np.argmax(ventanas)
        #se cuentan las frecuencias repetidas
        if vect_frecuencias[max_index] not in frecuencias:
            #se agrega al Dataframe las potencias que superen un umbral definido.
            num_ventana.append(i)
            if (np.max(ventanas)>=umbral):
                datos=[]
                datos.append(i)
                datos.append(np.mean(ventanas))
                datos.append(umbral)
                datos.append(np.sum([n**2 for n in ventanas])/len(sint)) #Potencia acumulada por
                ↪  ventana
                datos.append(np.max(ventanas))                          #
                datos.append(vect_frecuencias[max_index]) #se obtiene la frecuencia que corresponde
                ↪  al maximo
                datos.append(0)    #para conservar el tamaño de la lista y poder agregarla al
                ↪  dataFrame
                frecuencias.append(vect_frecuencias[max_index]) #también se guardar las frecuencias
                ↪  en un lista a parte
                dataFrame.append(datos)

                plt.figure(figsize=(7,3))
                plt.scatter(vect_frecuencias[max_index],np.max(ventanas),color="r",marker="v",label
                ↪  = "P_Máxima") #se grafica el punto maximo de potencia
                plt.axhline(umbral,color='b', linestyle='--',label = "Umbral")   #se grafica el
                ↪  promedio de la potencia
                plt.axhline(np.mean(ventanas),color='y', linestyle='--',label = "P_Promedio")   #se
                ↪  grafica el promedio de la potencia
                plt.plot(x_f[size_ventana:size_ventana + ventana],sint[size_ventana:size_ventana +
                ↪  ventana],color="g",label = "Espectro_Signal")  #plot(x, y_FTT)
                plt.ylim([y_min,y_max])
                plt.xlabel('Frequency [Hz]')
                plt.ylabel('Power [dBm]')
                plt.legend(loc = "upper right")
                print("Ventana-> [{} - {}] MHz.".format(x_f[size_ventana]/1e6,x_f[size_ventana +
                ↪  ventana]/1e6)," -> Tamaño = {}".format(Delta*ventana/1e3), "MHz. ") #Se imprime
                ↪  la frecuencia inicial y final de cada ventana
                plt.grid()
                plt.show()
```

```
        else:
            datos[0]=i   #las ventanas repetidas  también se enúmeran para poder mantener el orden
            datos[-1]+=1 #se cuentan los valores de potencia maximo repetidos

        i+=1
        size_ventana+=ventana-overlap

    elif(size_ventana<len(sint)-1):     #si las muestras que faltan es menor a tamaño de la ventana
    ↪   se grafican a parte
        ventanas= sint[size_ventana:len(sint)]
        vect_frecuencias=x_f[size_ventana:len(sint)]
        umbral=np.mean(ventanas) + 4 #Potencia promedio de las ventanas más 4 dB
        vect_ocupacion += [0 if punto < umbral else 1 for punto in ventanas]    #Se mira cuantos
        ↪   puntos estan por encima del umbral "1" y cuantos no "0"
        max_index = np.argmax(ventanas)
        #se cuentan las frecuencias repetidas
        if vect_frecuencias[max_index] not in frecuencias:
            num_ventana.append(i) #se obtine el # de la ventana y se agrega a una lista
            #se agrega al Dataframe las potencias que superen un umbral definido.
            if (np.max(ventanas)>=umbral):
                datos=[]
                datos.append(i)
                datos.append(np.mean(ventanas))
                datos.append(umbral)
                datos.append(np.sum([n**2 for n in ventanas])/len(sint)) #Potencia acumulada por
                ↪   ventana
                datos.append(np.max(ventanas))
                datos.append(vect_frecuencias[max_index]) #se obtiene la frecuencia que corresponde
                ↪   al maximo
                datos.append(0)      #para conservar el tamaño de la lista y poder agregarla al
                ↪   dataFrame
                frecuencias.append(vect_frecuencias[max_index]) #también se guardar las frecuencias
                ↪   en un lista a parte
                dataFrame.append(datos)

                plt.figure(figsize=(7,3))
                plt.scatter(vect_frecuencias[max_index],np.max(ventanas),color="r",marker="v",label
                ↪   = "P_Máxima") #se grafica el punto maximo de potencia
                plt.axhline(umbral,color='b', linestyle='--',label = "Umbral")   #se grafica el
                ↪   promedio de la potencia
                plt.axhline(np.mean(ventanas),color='y', linestyle='--',label = "P_Promedio")   #se
                ↪   grafica el promedio de la potencia
                plt.plot(x_f[size_ventana:len(sint)], sint[size_ventana:len(sint)],color="g",label =
                ↪   "Espectro_Signal")  #plot(x_f, y_FTT)
                plt.ylim([y_min,y_max])
                plt.xlabel('Frequency [Hz]')
                plt.ylabel('Power [dBm]')
                plt.legend(loc = "upper right")
                print("Ventana-> [{} - {}]
                ↪   MHz.".format(x_f[size_ventana]/1e6,x_f[len(sint)-1]/1e6)," -> Tamaño =
                ↪   {}".format(Delta*ventana/1e3), "MHz. ")  #Frecuencia inicial y final de la
                ↪   ventana
                plt.grid()
                plt.show()
        else:
            datos[0]=i
            datos[-1]+=1

        size_ventana=len(sint)
```

```
#Se crea el DataFrame
columnas = ['#Ventana','P_Promedio (dBm)','Umbral (dBm)',"P_Acumulada (dBm)", 'P_Maxima
↪  (dBm)','Frecuencia (Hz)','#Repeticiones']
df = pd.DataFrame(dataFrame, columns=columnas)
df = df.apply(lambda x: x.apply(lambda y: y.real if isinstance(y, complex) else y)).astype(float)


df = df.round(3)  # se reondea todos los números a 2 cifras decimales.
df.index = df.index + 1 #Para que el indice del dataFrame empiece en 1 y no en 0

print("Potencia de Umbral: ",umbral_spectrum)
print("P_Promedio del espectro: ",np.mean(sint),' (dBm)')
print("P_Promedio de las P_Maximas: ",np.mean(df['P_Maxima (dBm)']),' (dBm)')
#print("Varianza: ",df['P_Maxima (dBm)'].var()) #Del los puntos de potencia máxima
print("Desviació estandar P_Maximas: ",np.sqrt(df['P_Maxima (dBm)'].var()), ' (dBm)''\r\n') #Del los
↪  puntos de potencia máxima


plt.rcParams["figure.figsize"] = (14, 10) #tamano de la grafica

# se cre una lista de etiquetas para enumerar los picos detectados
for i, (x,y) in enumerate(zip(df['Frecuencia (Hz)'], df['P_Maxima (dBm)'])):
    plt.annotate(f"{i+1}", (x,y))


plt.axhline(np.mean(sint),color='k', linestyle='--',label = "P_Promedio")   #se grafica el promedio
↪  de la potencia de toda la señal
plt.axhline(np.mean(df['P_Maxima (dBm)']),color='g', linestyle='--',label = "P_Máxima_Promedio")
↪  #se grafica el promedio de la potencia maxima de cada ventana
plt.scatter(df['Frecuencia (Hz)'],df['P_Maxima (dBm)'],color="r",marker="v",label = "P_Detectada")
plt.plot(x_f,sint,color='b',label = "Espectro_Signal")  #plot(x_f, y_FTT)
#plt.axhline(umbral ,color='r', linestyle='--',label = "Umbral") #Umbral

plt.ylim([y_min,y_max])   #Escala eje Y
plt.xlabel('Frecuencia [MHz]', fontsize=15)
plt.ylabel('Potencia [dBm]', fontsize=15)
plt.xticks(fontsize=14)   #Permite cambiar el tamaño de la fuente del eje X
plt.yticks(fontsize=14)
plt.legend(loc = "upper right",ncol=2) #leyendas
plt.grid()
plt.show()

#Se genener una imagen.png del espectro
#plt.savefig("Espectro_Detect.png")
print(df)
```

6.2 Anexo B. Código WaveletDetection.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: Luis Ochoa
"""
import time
import pandas as pd
import pywt
from scipy.signal import find_peaks, decimate
import numpy as np
import os
import matplotlib.pyplot as plt
import scipy.signal as signal

def fftSt(st, Fs, AmpNor):
    """
    Parameters
    ----------
    st : Array of Complex
        Complex signal.
    AmpNor : Boolean
        Yes(True) or not(False) FFT with Amplitud Normalized.

    Returns
    -------
    Sshift : Array of Complex
        FFT with center fequency 0 Hz
    f : Array of Float
        axis x of spectrum signal.
    df : float
        f array resolution (delta f).

    """
    N = len(st)
    if AmpNor:
        S = np.fft.fft(st)/len(st)           # Normalize amplitude
    else:
        S = np.fft.fft(st)
    Sshift = np.fft.fftshift(S)
    df = Fs/N
    sampleIndex = np.linspace(-N/2, (N/2)-1, N)
    f = sampleIndex*df

    return Sshift, f, df

######################################################################

# Cargar los datos desde el archivo binario
pathCRC = "/home/luis/Grupo de Investigación GITA/Notebook Jupyter/"

file = 'LTE_B14.5M_Fc2.1475G'
data = np.fromfile(pathCRC+file, dtype=np.complex64)

# Limitar los datos a las primeras 500,000 muestras
data_subset = data[:500000]

# Tamaño de la muestra
sample_size = 50000 - 1024
```

```
# Extraer las muestras
sample2 = data_subset[1024 + sample_size : 1024 + 2 * sample_size]
sample3 = data_subset[1024 + 2 * sample_size : 1024 + 3 * sample_size]
sample4 = data_subset[1024 + 3 * sample_size : 1024 + 4 * sample_size]
sample5 = data_subset[1024 + 4 * sample_size : 1024 + 5 * sample_size]

fftSize = 4096
samp_rate = 14.50e6
bw = samp_rate
fc = 2.14755e9

# Inicializar matriz para almacenar FFT de cada muestra
all_ffts = np.zeros((4, fftSize), dtype=np.complex128)

samples = [sample2, sample3, sample4, sample5]

# Procesar cada muestra y calcular su FFT
for i, sample in enumerate(samples):
    st = sample[:fftSize]  # Limitar el tamaño de la muestra a fftSize si es necesario
    fft_result, _, _ = fftSt(st, samp_rate, True)
    all_ffts[i] = fft_result

# Calcular el promedio de las FFT de las muestras
average_fft = np.mean(all_ffts, axis=0)

# Crear el eje de frecuencias
frequency_axis = np.linspace(fc - int(bw / 2), fc + int(bw / 2), fftSize)

fft_dB = 20 * np.log10(average_fft)

sint =  fft_dB
# Niveles de descomposición
nDesc = 3

# Se puede ajustar el tipo de wavelet según tu preferencia
# haar, bior1.1, bior1.3, db1, sym2, sym3.
waveletname = 'haar'

x_f = np.linspace(fc-int(bw/2),fc+int(bw/2),fftSize)
x_f_mhz = x_f / 1e6 # Convertir Hz a MHz
# Crear una figura para visualizar los coeficientes de la Transformada Wavelet
fig, axs = plt.subplots(nDesc+2, 1, figsize=(20, 18))

axs[0].plot(x_f_mhz,sint, '#00008B')
axs[0].set_title("Señal Original", fontsize=21)
axs[0].set_xlabel('Frecuencia (MHz)', fontsize=18)
axs[0].set_ylabel('Magnitud', fontsize=18)
axs[0].tick_params(axis='x', labelsize=15)  # Tamaño de la fuente para el eje x
axs[0].tick_params(axis='y', labelsize=15)

# Aplicar la transformada wavelet
coeffs = pywt.wavedec(sint, waveletname, level=nDesc)

# Inicializar una lista para almacenar las PDS
pds_list = []

for i, coeff in enumerate(coeffs):
    #x_f_2= np.linspace(frequencies.min(), frequencies.max(), len(coeff))
    x_f_2 =  np.linspace(fc-int(bw/2),fc+int(bw/2),len(coeff))
```

```
    x_f_2 = x_f_2/1e6

    # No calcular la PSD para el primer nivel (i=0)
    if i > 0:
        positive_coeff = np.maximum(coeff.real, 0)
        power_spectrum = np.abs(positive_coeff) ** 2

        # Graficar la PDS y el nivel de descomposición
        axs[i].plot(x_f_2, coeff,'#00008B')
        axs[i].plot(x_f_2, power_spectrum, '#ff7f0e')
        axs[i].set_title(f'PSD - DWT Nivel {i}', fontsize=21)
        axs[i].set_xlabel('Frecuencia (MHz)', fontsize=18)
        axs[i].set_ylabel('Magnitud', fontsize=18)

        # Interpolar la PDS a 2048 muestras, excepto para el último nivel
        if i < nDesc:
            power_spectrum = np.interp(np.linspace(0, 1,fftSize//2 ), np.linspace(0, 1,
            ↪  len(power_spectrum)), power_spectrum)

        else:
            axs[i].plot(x_f_2, coeff, '#00008B')
            axs[i].plot(x_f_2, power_spectrum, '#ff7f0e')
            axs[i].set_title(f'PSD - DWT Nivel {i}', fontsize=21)
            axs[i].set_xlabel('Frecuencia (MHz)', fontsize=18)
            axs[i].set_ylabel('Magnitud', fontsize=18)

        pds_list.append(power_spectrum) #Se agregan las PSD promedio a una lista

# Calcular el promedio de todas las PSD
pds_avg = np.mean(pds_list, axis=0)

# Graficar el promedio de las PDS
axs[i+1].plot(x_f_2, pds_avg, '#FF6347')
axs[i+1].set_title('Promedio Densidad Espectral de Potencia (PSD) - DWT', fontsize=21)
axs[i+1].set_xlabel('Frecuencia (MHz)', fontsize=18)
axs[i+1].set_ylabel('Magnitud', fontsize=18)

# Ajustar el layout
plt.tight_layout()

# Cambiar el tamaño de los números en los ejes x e y
for j in range(len(axs)-1):
    axs[j+1].tick_params(axis='x', labelsize=17)  # Tamaño de la fuente para el eje x
    axs[j+1].tick_params(axis='y', labelsize=17)  # Tamaño de la fuente para el eje y

# Guardar la figura como PNG
plt.savefig("wavelet_PSD.png")
plt.show()

# Detectar los picos en la PSD promedio
peaks, _ = find_peaks(pds_avg, height=50)

# Crear un DataFrame con los valores de potencia y frecuencia correspondientes a los picos
df = pd.DataFrame({
    'Frecuencia (MHz)': x_f_2[peaks],
    'Magnitud': pds_avg[peaks]
})
#Se imprime el dataframe
print(df)
```