Procesamiento de Señales, Fundamentos

Trabajo Práctico 2

Jairo Mena

Profesor:

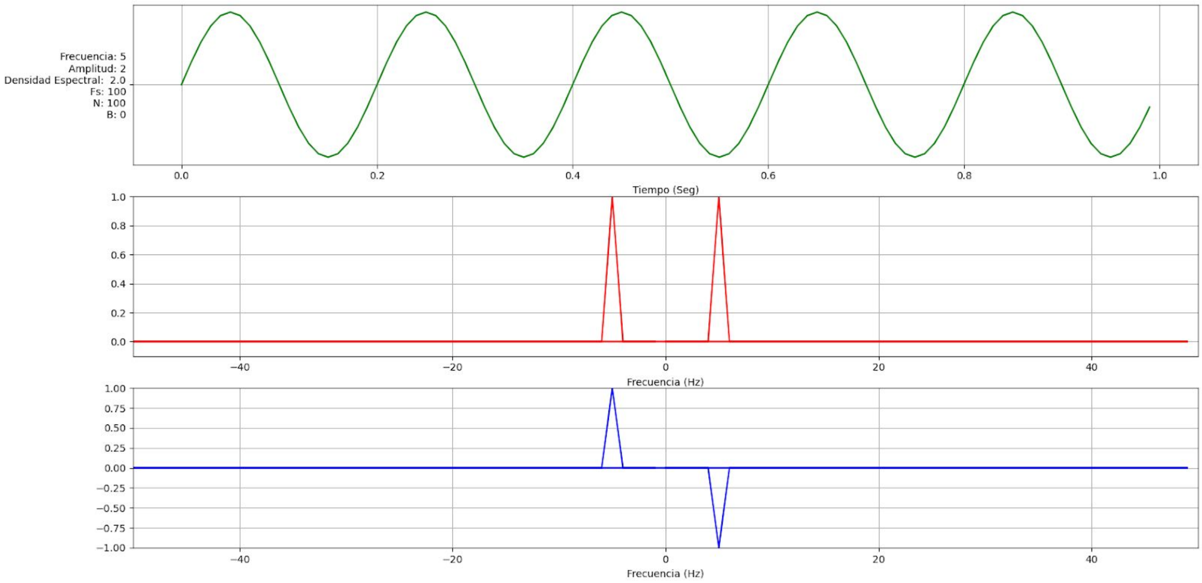
Pablo Slavkin

Universidad de Buenos Aires

MSE 5Co2020

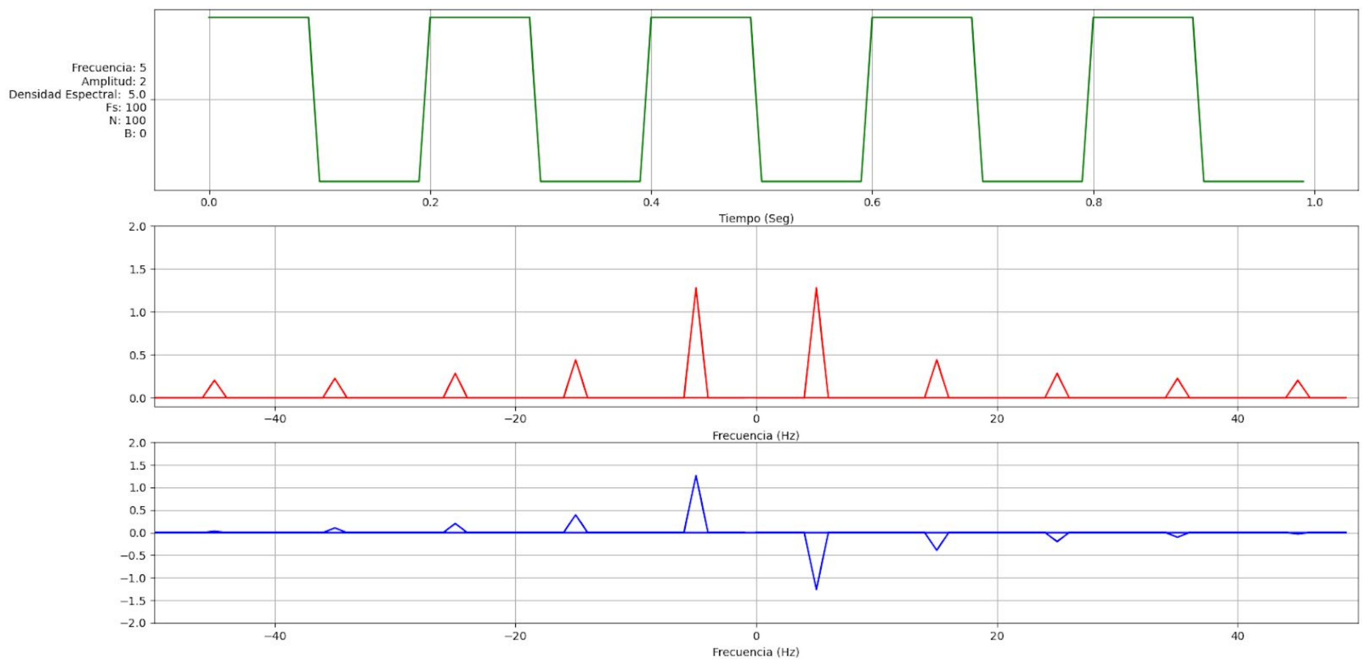
1) Se grafica las siguientes señales lado a lado con su respectivo espectro en frecuencias:
Código que grafica la señal sinusoidal con parámetros y su respectiva transformada de Fourier.

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
import scipy.signal as sc


def densidadEspectral(vector):
    den = 0
    for i in vector:
        den = den + i
    return den

#-------------------------------------
fig        = plt.figure()

freq       = 5
amp        = 2
fs         = 100
N          = 100
B          = 0

#-----------RANGE Time-------------------------
tData      = np.arange(0,N/fs,1/fs)

#-----------SIGNAL Sen-------------------------
senLn = fig.add_subplot(3,1,1)
sen = amp * np.sin(2*np.pi*tData*freq + B)
senLn.plot(tData, sen, "g-")
plt.title("Señal Senoidal")
plt.xlabel("Tiempo (Seg)")
dEsp = round(densidadEspectral(1/N *np.abs(np.fft.fft(sen))))
plt.plot("Frecuencia: " + str(freq) + "\nAmplitud: " + str(amp) +  "\nDensidad Espectral: "
         + str(dEsp) + "\nFs: " + str(fs) + "\nN: " + str(N) + "\nB: " + str(B))
senLn.grid(True)

#------------FFT Senoidal--------------------------
fftSen = np.fft.fft(sen)
fftSenFreq = np.fft.fftfreq(n=sen.size, d=1/fs)

fftSenRealLn = fig.add_subplot(3,1,2)
plt.xlim(-fs/2,fs/2)
plt.ylim(-0.1,amp/2)
fftSenRealLn.plot(fftSenFreq, 1/N * np.abs(fftSen),"r-", label= "Espectro en Frecuencias")
plt.xlabel("Frecuencia (Hz)")
fftSenRealLn.grid(True)

fftSenImagLn = fig.add_subplot(3,1,3)
plt.xlim(-fs/2,fs/2)
plt.ylim(-amp/2,amp/2)
fftSenImagLn.plot(fftSenFreq, 1/N * fftSen.imag,"b-", label= "Parte Imaginaria")
plt.xlabel("Frecuencia (Hz)")
fftSenImagLn.grid(True)
```



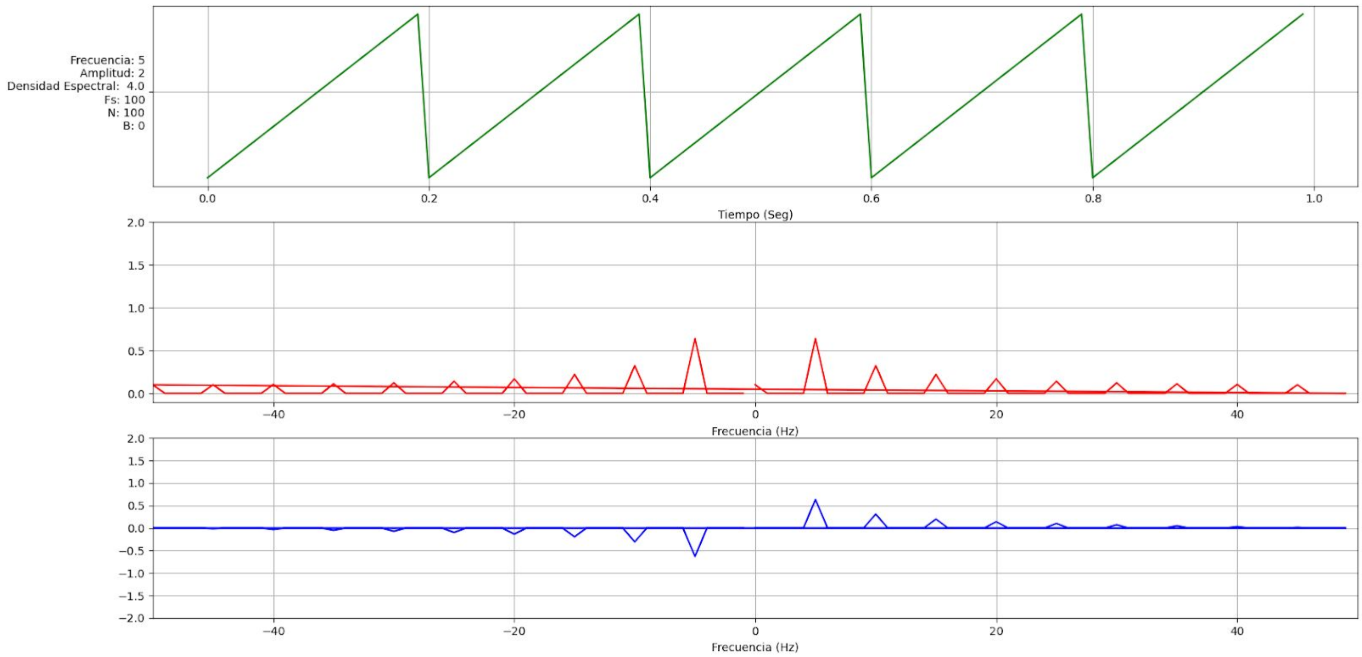
Código que grafica la señal Cuadrada con parámetros y su respectiva transformada de Fourier.

```python
In [ ]: #------------SIGNAL Cuadrada-------------------------
sqrLn = fig.add_subplot(3,1,1)
sqr =  amp * sc.square(2*np.pi*tData*freq + B,0.5)
sqrLn.plot(tData, sqr, "g-", label= "Señal Cuadrada")
plt.title("Señal Cuadrada")
plt.xlabel("Tiempo (Seg)")
dEsp = round(densidadEspectral(1/N *np.abs(np.fft.fft(sqr))))
plt.plot("Frecuencia: " + str(freq) + "\nAmplitud: " + str(amp) +  "\nDensidad Espectral: "
        + str(dEsp) + "\nFs: " + str(fs) + "\nN: " + str(N) + "\nB: " + str(B))
sqrLn.grid(True)

#------------FFT Cuadrada--------------------------
fftSqr = np.fft.fft(sqr)
fftSqrFreq = np.fft.fftfreq(n=sqr.size, d=1/fs)

fftSqrRealLn = fig.add_subplot(3,1,2)
plt.xlim(-fs/2,fs/2)
plt.ylim(-0.1,amp)
fftSqrRealLn.plot(fftSqrFreq, 1/N * np.abs(fftSqr),"r-", label= "Espectro en Frecuencias")
plt.xlabel("Frecuencia (Hz)")
fftSqrRealLn.grid(True)

fftSqrImagLn = fig.add_subplot(3,1,3)
plt.xlim(-fs/2,fs/2)
plt.ylim(-amp,amp)
fftSqrImagLn.plot(fftSqrFreq, 1/N * fftSqr.imag,"b-", label= "Parte Imaginaria")
plt.xlabel("Frecuencia (Hz)")
fftSqrImagLn.grid(True)
```



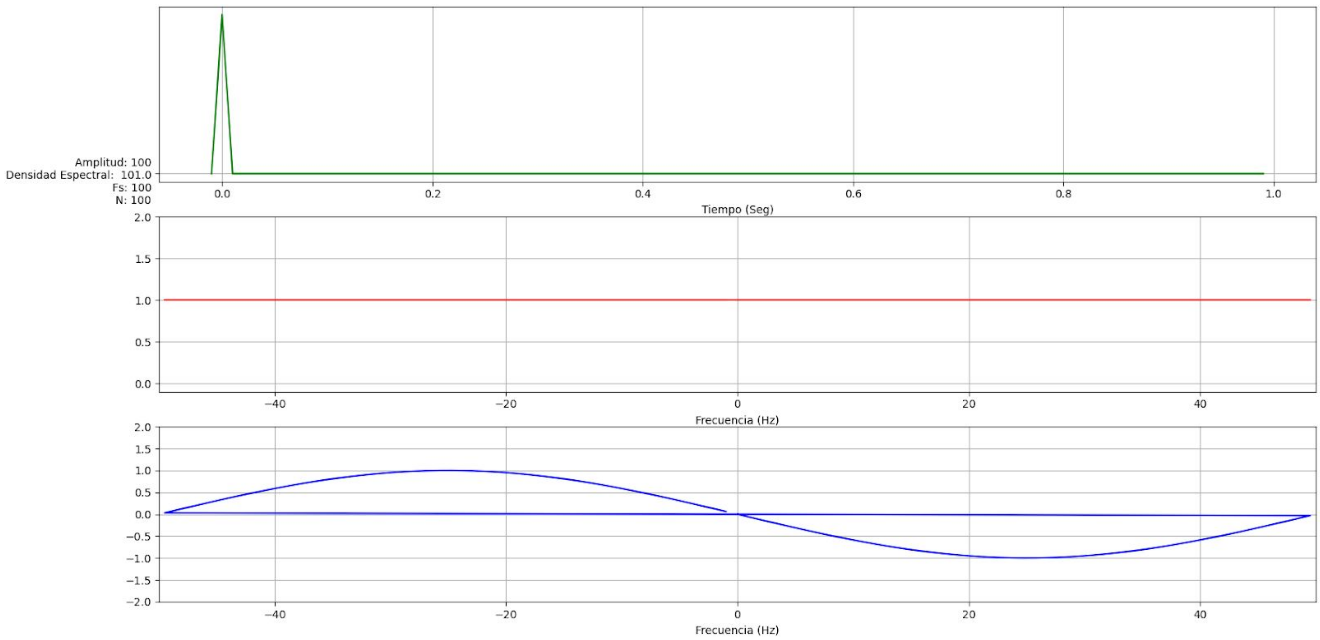
Código que grafica la señal Cuadrada con parámetros y su respectiva transformada de Fourier.

In [ ]:
```
#------------SIGNAL Triangular--------------------------
rawLn = fig.add_subplot(3,1,1)
raw =  amp * sc.sawtooth(2*np.pi*tData*freq + B,1)
rawLn.plot(tData, raw, "g-", label= "Señal Triangular")
plt.title("Señal Triangular")
plt.xlabel("Tiempo (Seg)")
dEsp = round(densidadEspectral(1/N *np.abs(np.fft.fft(raw))))
plt.plot("Frecuencia: " + str(freq) + "\nAmplitud: " + str(amp) +  "\nDensidad Espectral:  " + str(dEsp) + "\nFs: " + str(fs) +
"\nN: " + str(N) + "\nB: " + str(B))
rawLn.grid(True)

#-------------FFT Triangular---------------------------
fftRaw = np.fft.fft(raw)
fftRawFreq = np.fft.fftfreq(n=raw.size, d=1/fs)

fftRawRealLn = fig.add_subplot(3,1,2)
plt.xlim(-fs/2,fs/2)
plt.ylim(-0.1,amp)
fftRawRealLn.plot(fftRawFreq, 1/N * np.abs(fftRaw),"r-", label= "Espectro en Frecuencias")
plt.xlabel("Frecuencia (Hz)")
fftRawRealLn.grid(True)

fftRawImagLn = fig.add_subplot(3,1,3)
plt.xlim(-fs/2,fs/2)
plt.ylim(-amp,amp)
fftRawImagLn.plot(fftRawFreq, 1/N * fftRaw.imag,"b-", label= "Parte Imaginaria")
plt.xlabel("Frecuencia (Hz)")
fftRawImagLn.grid(True)
```



Código que grafica la señal Delta en T=0 con parámetros y su respectiva transformada de Fourier.

```
In [ ]: #------------SIGNAL Delta T=0-------------------------
ampDel = 100
tDataDel      = np.arange(-1/fs,N/fs,1/fs)

u = lambda t: np.piecewise(t,t>=0,[1,0])
dt = 1/fs

u0 = np.piecewise(tDataDel,tDataDel>=0,[1,0])
udt = np.piecewise(tDataDel,tDataDel>=(0+dt),[1,0])
u0 = u(tDataDel)
udt = u(tDataDel-dt)
impulso = u0 - udt

deltaLn = fig.add_subplot(3,1,1)
delta = ampDel  * impulso
deltaLn.plot(tDataDel, delta, "g-", label= "Señal Delta en T=0")
plt.title("Señal Delta en T=0")
plt.xlabel("Tiempo (Seg)")
dEsp = round(densidadEspectral(1/N *np.abs(np.fft.fft(delta))))
plt.plot("\nAmplitud: " + str(ampDel) +  "\nDensidad Espectral:  " + str(dEsp) + "\nFs: " + str(fs) + "\nN: " + str(N))
deltaLn.grid(True)

#-------------FFT Delta T=0--------------------------
fftDelta = np.fft.fft(delta)
fftDeltaFreq = np.fft.fftfreq(n=delta.size, d=1/fs)

fftDeltaRealLn = fig.add_subplot(3,1,2)
plt.xlim(-fs/2,fs/2)
plt.ylim(-0.1,amp)
fftDeltaRealLn.plot(fftDeltaFreq, 1/N * np.abs(fftDelta),"r-", label= "Espectro en Frecuencias")
plt.xlabel("Frecuencia (Hz)")
fftDeltaRealLn.grid(True)

fftDeltaImagLn = fig.add_subplot(3,1,3)
plt.xlim(-fs/2,fs/2)
plt.ylim(-amp,amp)
fftDeltaImagLn.plot(fftDeltaFreq, 1/N * fftDelta.imag,"b-", label= "Parte Imaginaria")
plt.xlabel("Frecuencia (Hz)")
fftDeltaImagLn.grid(True)

plt.show()
```

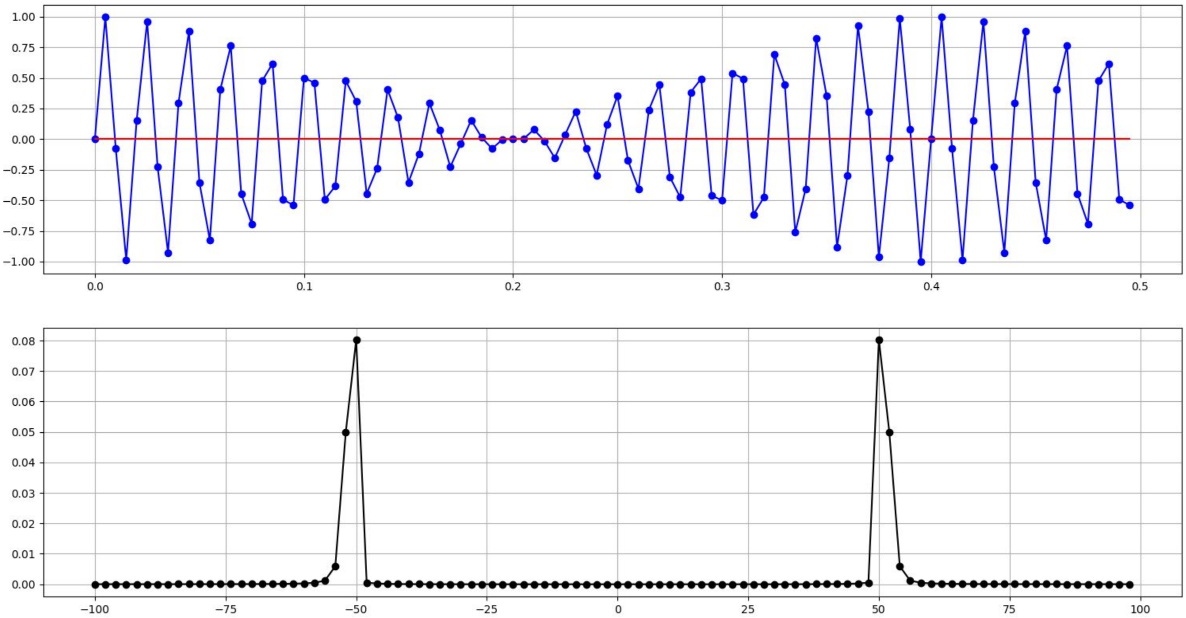


NOTA: La práctica es consistente con lo visto en la teoría en todas las señales y en todos los espectros.

2) Se encuentra el contenido espectral de la siguiente señal. N = 100 y Fs = 200.

```
[ 0.00000000e+00  9.98458667e-01 -7.82172325e-02 -9.86184960e-01
  1.54508497e-01  9.61939766e-01 -2.26995250e-01 -9.26320082e-01
  2.93892626e-01  8.80202983e-01 -3.53553391e-01 -8.24724024e-01
  4.04508497e-01  7.61249282e-01 -4.45503262e-01 -6.91341716e-01
  4.75528258e-01  6.16722682e-01 -4.93844170e-01 -5.39229548e-01
  5.00000000e-01  4.60770452e-01 -4.93844170e-01 -3.83277318e-01
  4.75528258e-01  3.08658284e-01 -4.45503262e-01 -2.38750718e-01
  4.04508497e-01  1.75275976e-01 -3.53553391e-01 -1.19797017e-01
  2.93892626e-01  7.36799178e-02 -2.26995250e-01 -3.80602337e-02
  1.54508497e-01  1.38150398e-02 -7.82172325e-02 -1.54133313e-03
  1.83758918e-15  1.54133313e-03  7.82172325e-02 -1.38150398e-02
 -1.54508497e-01  3.80602337e-02  2.26995250e-01 -7.36799178e-02
 -2.93892626e-01  1.19797017e-01  3.53553391e-01 -1.75275976e-01
 -4.04508497e-01  2.38750718e-01  4.45503262e-01 -3.08658284e-01
 -4.75528258e-01  3.83277318e-01  4.93844170e-01 -4.60770452e-01
 -5.00000000e-01  5.39229548e-01  4.93844170e-01 -6.16722682e-01
 -4.75528258e-01  6.91341716e-01  4.45503262e-01 -7.61249282e-01
 -4.04508497e-01  8.24724024e-01  3.53553391e-01 -8.80202983e-01
 -2.93892626e-01  9.26320082e-01  2.26995250e-01 -9.61939766e-01
 -1.54508497e-01  9.86184960e-01  7.82172325e-02 -9.98458667e-01
  5.63708916e-15  9.98458667e-01 -7.82172325e-02 -9.86184960e-01
  1.54508497e-01  9.61939766e-01 -2.26995250e-01 -9.26320082e-01
  2.93892626e-01  8.80202983e-01 -3.53553391e-01 -8.24724024e-01
  4.04508497e-01  7.61249282e-01 -4.45503262e-01 -6.91341716e-01
  4.75528258e-01  6.16722682e-01 -4.93844170e-01 -5.39229548e-01]
```

```python
import numpy as np
import matplotlib.pyplot as plt
#--------------------------------------
fig         = plt.figure()
fs          = 200
N1           =100
N2           =0
N=N1+N2
frecIter   = 0
signalFrec = 50
#--------------------------------------
tData      = np.arange(0,N/fs,1/fs)
n1Data      = np.arange(0,N1,1)
n2Data      = np.arange(N1,N1+N2,1)
circleFrec = np.arange(-fs/2,fs/2,fs/N)
#------------SIGNAL--------------------------
signalData1 = 0.5*np.sin(2*np.pi*signalFrec*n1Data*1/fs)+0.5*np.sin(2*np.pi*(2.5+signalFrec)*n1Data*1/fs)
signalData2 = np.zeros(abs(N2))
signalData=np.concatenate((signalData1,signalData2))
print(signalData)
signalAxe  = fig.add_subplot(2,1,1)
signalRLn,signalILn,= plt.plot(tData,np.real(signalData),'b-o',tData,np.imag(signalData),'r-')
signalAxe.grid(True)
#------------FFT IFFT-----------------------
fftData  = np.fft.fft(signalData)
ifftData = np.fft.ifft(fftData)
fftData  = np.concatenate((fftData[N//2:N],fftData[0:N//2]))/N
#-----------FFT--------------------------
fftAxe                 = fig.add_subplot(2,1,2)
fftAbsLn = plt.plot(circleFrec,np.abs(fftData)**2,'k-o')
fftAxe.grid(True)
#-----------------------------------------------------
plt.show()
```



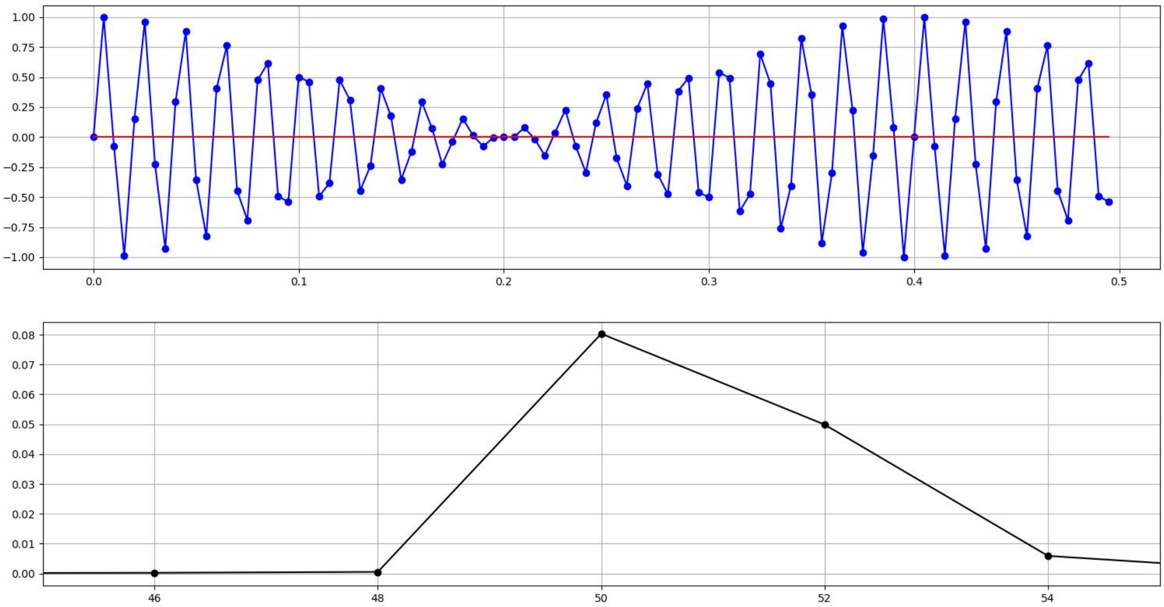
La resolución espectral se la encuentra en dividir la Frecuencia de muestreo Fs sobre el número de muestras de la señal (N).

$$Resolución\ Espectral\ =\ \frac{Fs}{N}$$

Fs = 200Hz y N = 100

$$Resolución\ Espectral\ =\ \frac{200Hz}{100}$$

Resolución espectral es igual a **2Hz.**

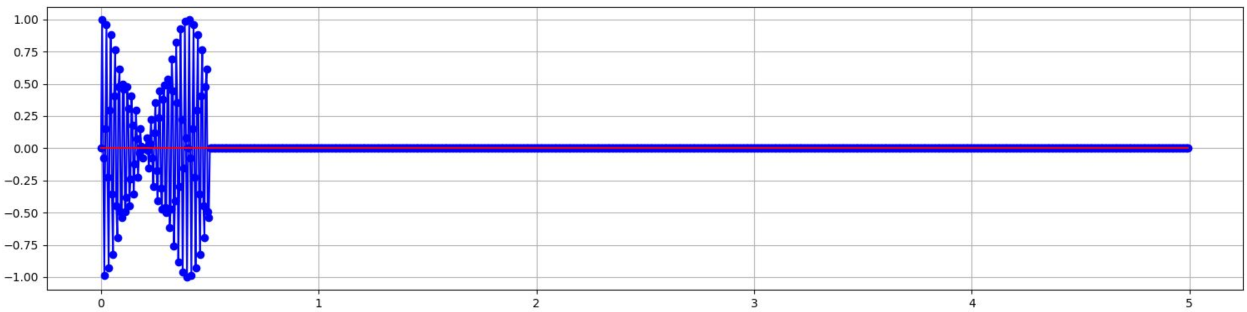


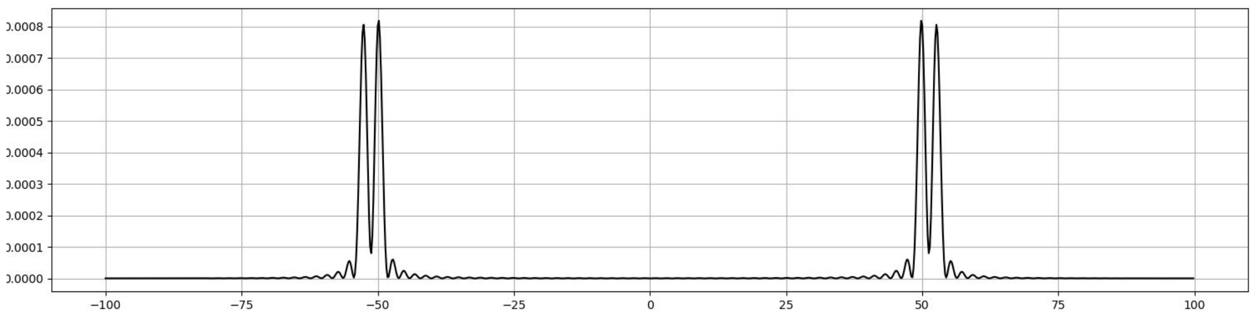
Como se muestra en la figura anterior haciendo zoom sobre la gráfica del espectro, se puede visualizar de manera explícita la resolución espectral de 2Hz.

Para mejorar la resolución espectral se aplica la técnica de rellenado de ceros (zero padding).

Se realiza un relleno de 900 ceros.

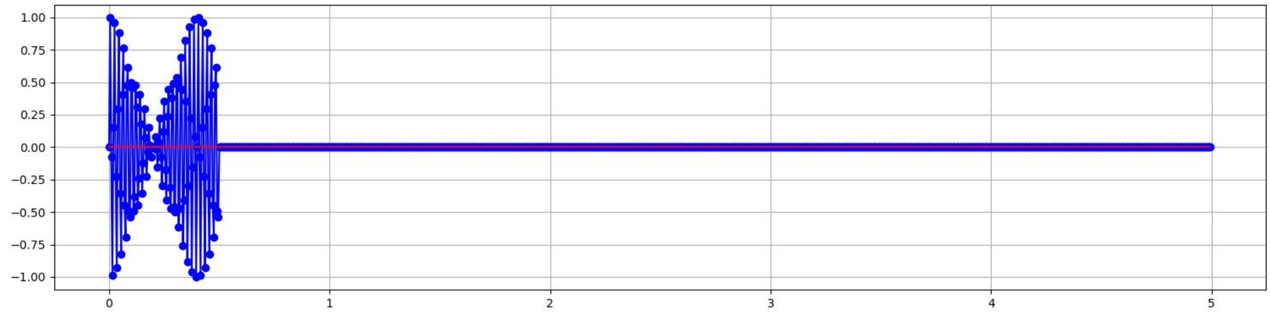
```
import numpy as np
import matplotlib.pyplot as plt
#--------------------------------------
fig        = plt.figure()
fs         = 200
N1          =100
N2          =900
N=N1+N2
frecIter   = 0
signalFrec = 50
#--------------------------------------
tData      = np.arange(0,N/fs,1/fs)
n1Data      = np.arange(0,N1,1)
n2Data      = np.arange(N1,N1+N2,1)
circleFrec = np.arange(-fs/2,fs/2,fs/N)
#------------SIGNAL-------------------------
signalData1 = 0.5*np.sin(2*np.pi*signalFrec*n1Data*1/fs)+0.5*np.sin(2*np.pi*(2.5+signalFrec)*n1Data*1/fs)
signalData2 = np.zeros(abs(N2))
signalData=np.concatenate((signalData1,signalData2))
print(signalData)
signalAxe  = fig.add_subplot(2,1,1)
signalRLn,signalILn,= plt.plot(tData,np.real(signalData),'b-o',tData,np.imag(signalData),'r-')
signalAxe.grid(True)
#------------FFT IFFT----------------------
fftData  = np.fft.fft(signalData)
ifftData = np.fft.ifft(fftData)
fftData  = np.concatenate((fftData[N//2:N],fftData[0:N//2]))/N
#----------FFT-------------------------
fftAxe                 = fig.add_subplot(2,1,2)
fftAbsLn = plt.plot(circleFrec,np.abs(fftData)**2,'k')
fftAxe.grid(True)
#--------------------------------------
plt.show()
```

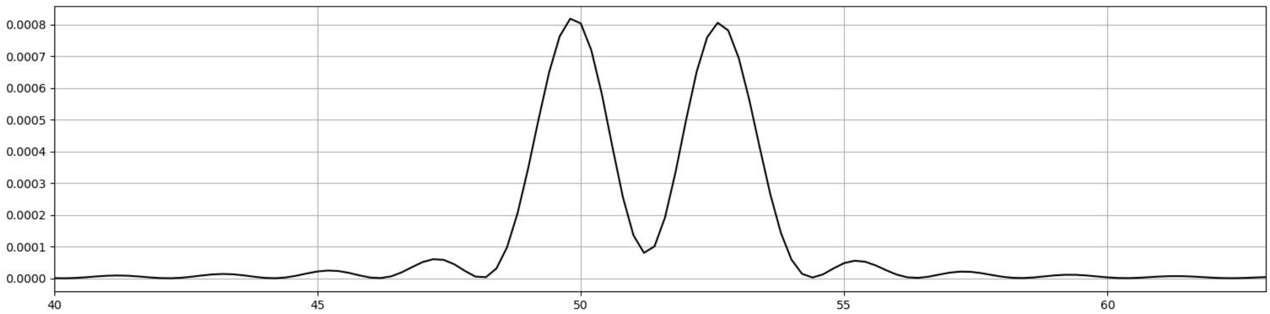




Se puede visualizar que la resolución espectral mejoró de forma sustancial, ahora se puede visualizar las dos componentes de la señal, a pensar de estar muy cerca.

Realizando zoom sobre la señal de la frecuencia se puede visualizar mejor los resultados.





La resolución espectral se calcula de nuevo con el número de muestras de la señal N = N1 + N2 que es igual a 100 + 900 = 1000 y con una frecuencia de muestreo Fs = 200Hz.

$$Resolución\ Espectral\ =\ \frac{Fs}{N}$$

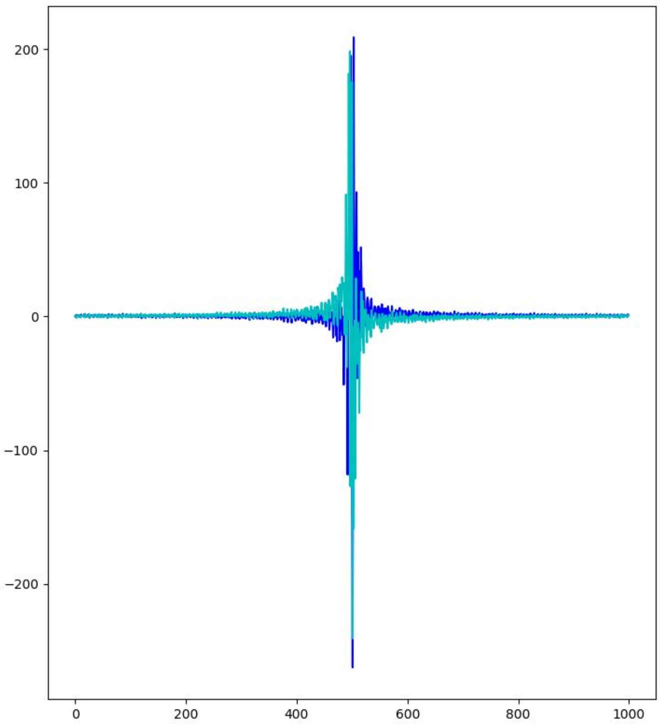
Fs = 200Hz y N = 1000

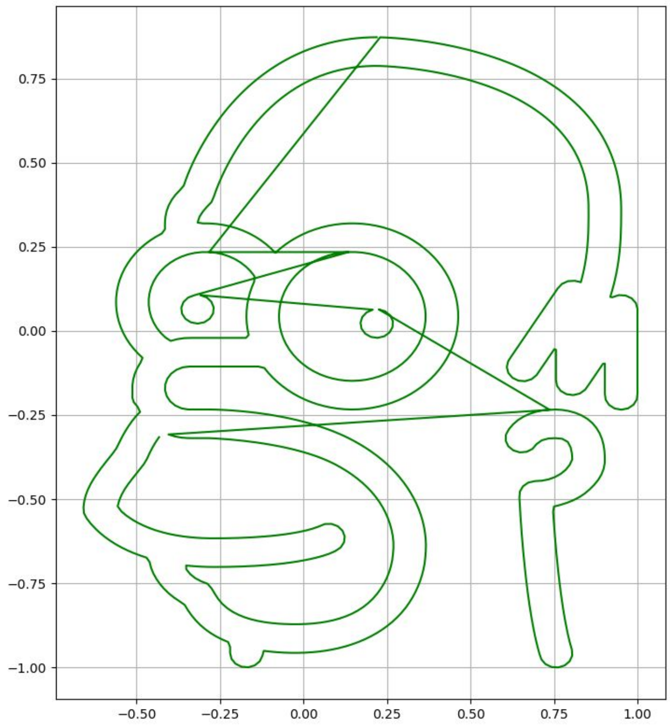
$$Resolución\ Espectral\ =\ \frac{200Hz}{1000}$$

Resolución espectral actual es igual a **0.2Hz.**

3) Después de realizar la transformada de Fourier de las señales o de la señal en formato de número imaginario del archivo (*fft_hjs.npy*) se obtiene la información bidimensional del rostro de Homero Simpson.

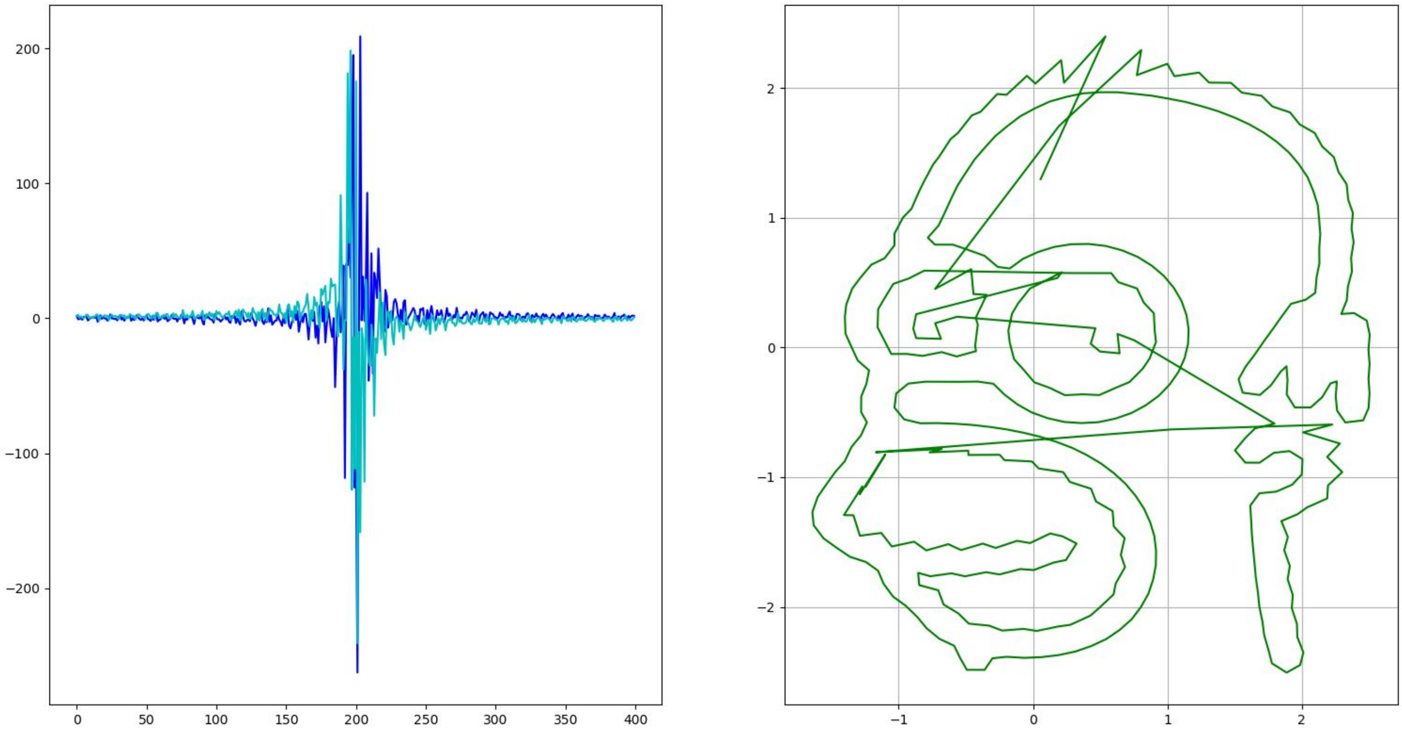
```
import numpy as np
import matplotlib.pyplot as plt

##-------------------------------------
fig        = plt.figure()
fft_hjs = np.load("fft_hjs.npy")
fftAxe       = fig.add_subplot(1,2,1)
sRLn,sILn,= plt.plot(np.fft.fftshift(np.real(fft_hjs)) ,'b-', np.fft.fftshift(np.imag(fft_hjs)) ,'c-')

ifft_hjs = np.fft.ifft(fft_hjs)
fft_hjs_shift = np.fft.fftshift(ifft_hjs)

print(fft_hjs_shift)
ifftAxe       = fig.add_subplot(1,2,2)
penRLn, = plt.plot(np.imag(fft_hjs_shift),np.real(fft_hjs_shift),'g-')
ifftAxe.grid(True)
#-------------------------------------
plt.show()
```



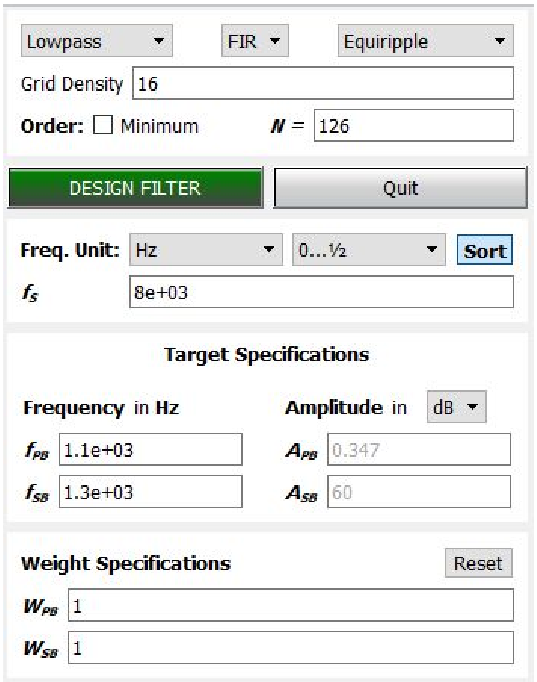


Al realizar un recorte del ancho de banda del espectro donde la variable ***center*** significa el centro del vector original y la variable ***band*** representa el número de muestras de la banda dividido entre 2. Como se muestra en el código y en la gráfica siguiente se puede vislumbrar un rostro de Homero Simpson hasta un ancho de banda de 400 datos, sin que sufra una deformación sustancial de la información utili, teniendo la original de 1000 datos.

```
import numpy as np
import matplotlib.pyplot as plt

##-------------------------------------
fig        = plt.figure()
fft_hjs = np.load("fft_hjs.npy")

fft_hjs_shift = np.fft.fftshift(fft_hjs)

print(fft_hjs_shift.size)

center = 500
band = 200

fft_hjs_shift_sub = fft_hjs_shift[center-band:center+band]

print(fft_hjs_shift_sub.size)

fftAxe       = fig.add_subplot(1,2,1)
sRLn,sILn,= plt.plot(np.real(fft_hjs_shift_sub) ,'b-', np.imag(fft_hjs_shift_sub) ,'c-')

ifft_hjs = np.fft.ifft(np.fft.fftshift(fft_hjs_shift_sub))

ifftAxe    = fig.add_subplot(1,2,2)
penRLn, = plt.plot(np.imag(ifft_hjs),np.real(ifft_hjs),'g-')
ifftAxe.grid(True)
#-------------------------------------
plt.show()
```

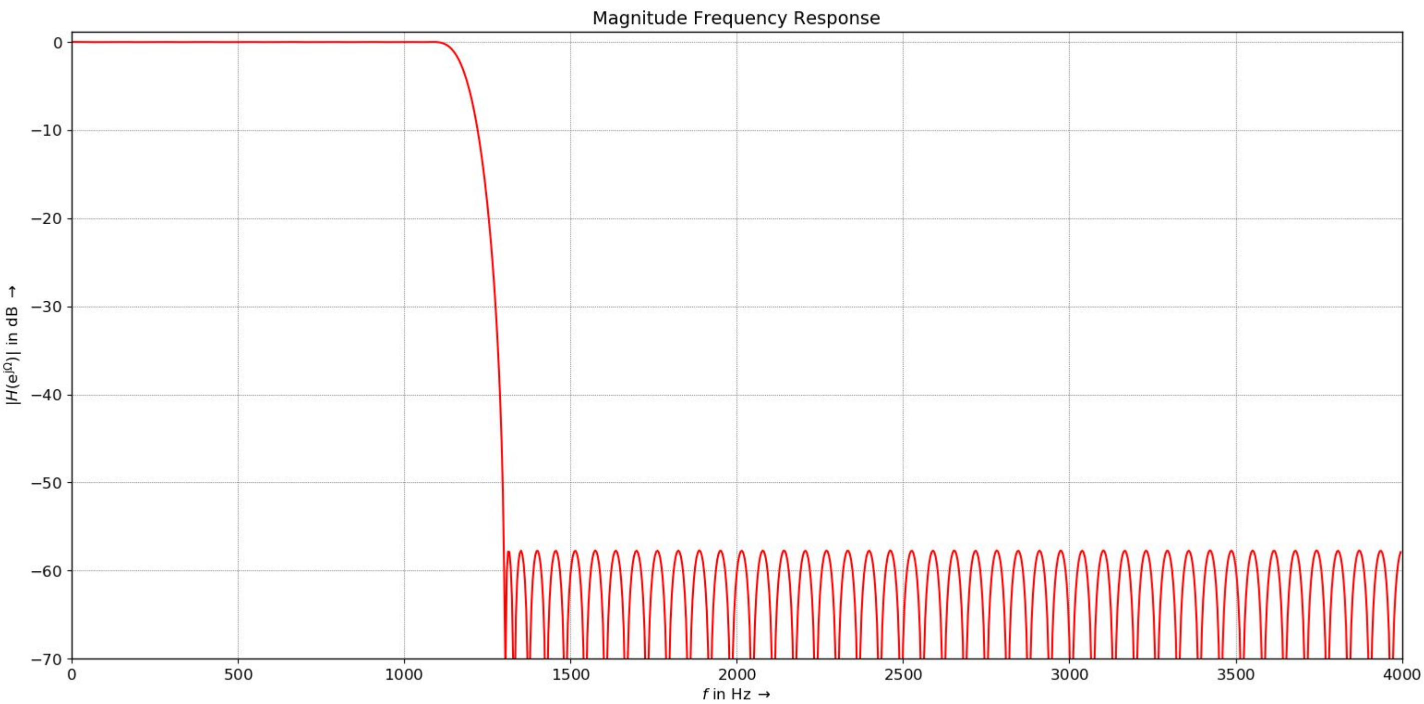


4) Dado el segmento de audio en el archivo ***chapu_noise.npy*** con fs = 8000 y sumergido en ruido de alta frecuencia se diseña un filtro en Pyfda como se muestra en la siguientes figuras.



Especificaciones del filtro:

- Filtro pasabajo FIR con un N de 126 muestras.
- fs = 8000
- Pass Frecuency de 1100Hz
- Stop Frecuency de 1300Hz
- Zona de Transición de 200Hz
- Y una atenuación cercana a los 60dB

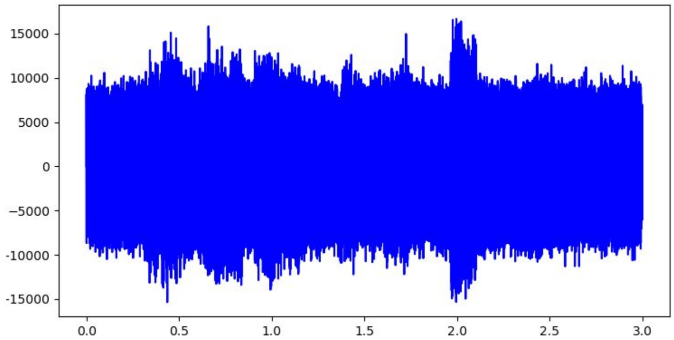


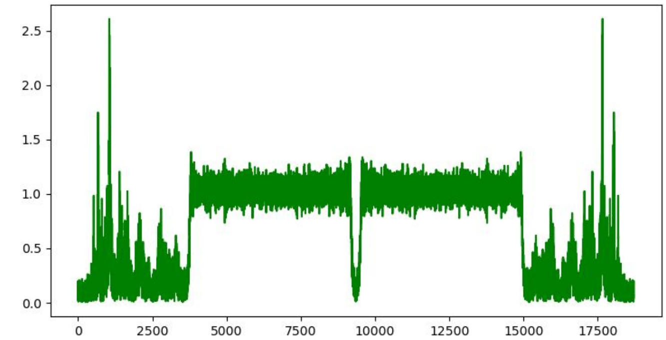
Se aplica la convolución de la señal original con ruido con la señal del filtro diseñado anteriormente, al generar el audio se puede apreciar claramente al Chapulín Colorado diciendo su famosa frase *"No contaban con mi astucia"* .

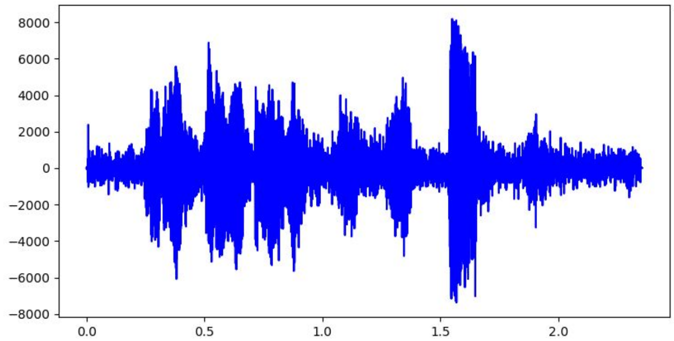
Se realiza las gráficas de las señal antes y después de aplicar el filtrado con sus respectivas transformadas de Fourier.

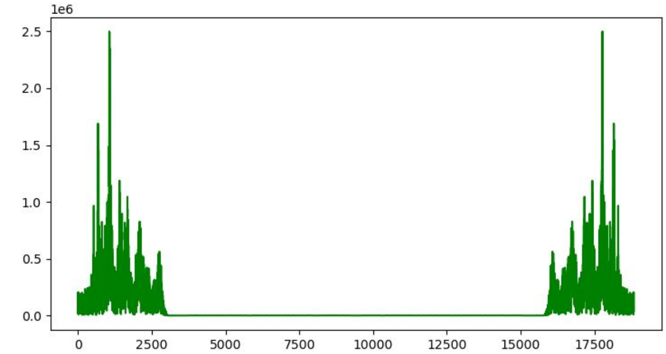
```
import numpy as np
import simpleaudio as sa
import matplotlib.pyplot as plt

fig        = plt.figure()
fs  = 8000

def noise(n):
    return ((2**13)+np.random.normal(scale=1000))*np.sin((1600+500*n)*n*(2*np.pi))

a=np.load("chapu_noise.npy")

play_obj = sa.play_buffer(a, 1, 2, fs)
play_obj.wait_done()

fft=np.fft.fft(a)

lo_pass,=np.load("low_pass_filter.npy").astype(float)
out=np.convolve(a,lo_pass).astype(np.int16)

audioAxe  = fig.add_subplot(2,2,1)
audioLn,= plt.plot(np.linspace(0,3,len(a)),a,'b-')

fftAxe  = fig.add_subplot(2,2,2)
fftLn,= plt.plot(np.arange(0,len(fft),1),np.abs(fft),'g-')

signalAxe  = fig.add_subplot(2,2,3)
signalLn,  = plt.plot(np.linspace(0,len(out)/fs,len(out)),out,'b-')

fft_out=np.fft.fft(out)
fftAxe  = fig.add_subplot(2,2,4)
fftLn,= plt.plot(np.arange(0,len(fft_out),1),np.abs(fft_out),'g-')

play_obj = sa.play_buffer(out, 1, 2, fs)
play_obj.wait_done()
plt.show()
```

En la gráfica se puede apreciar como la convolución de la señal con el filtro elimina totalmente las frecuencias de ruido en el espectro bajo, dejando pasar solamente la señal que contiene la información útil, en este caso, el audio del Chapulín Colorado diciendo su famosa frase.


15000
10000
5000
0
−5000
-10000
-15000
0.0
0.5
1.0
1.5
2.0
2.5
3.0


2.5
2.0
1.5
1.0
0.5
0.0
0
2500
5000
7500
10000
12500
15000
17500


8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
0.0
0.5
1.0
1.5
2.0


1e6
2.5
2.0
1.5
1.0
0.5
0.0
0
2500
5000
7500
10000
12500
15000
17500