TPNº1- Transformada Discreta de Fourier

Autor: Pablo D. Folino
Link del repositorio: https://github.com/PabloFolino/MSE_PSF_TP1.git

Se realiza un programa(***TP1_folino_v3.py***) en donde posee funciones que generan señales cuadradas triangulares y senoidales(desde la línea 191 a la 232), se transcribe parte del código:

Importante: a las funciones se le agregan algunos parámetros extra para tener mayor flexibilidad .

```
201  def senoidal(fs,f,amp,muestras,fase):
202      n = np.arange(0, muestras, 1)/fs        # Intervalo de tiempo en segundos
203      f1=amp*np.sin(2*np.pi*f*n+fase)         # Definimos el Vector de Frecuencias
204      return f1,n
```

```
216  def cuadrada(fs,f,amp,muestras,fase):
217      n = np.arange(0, muestras, 1)/fs            # Intervalo de tiempo en segundos
218      f1 =amp*sc.square(2*np.pi*n*f+fase)         # Definimos una onda
219      return f1,n
```

```
229  def triangular(fs,f,amp,muestras,fase):
230      n = np.arange(0, muestras, 1)/fs            # Intervalo de tiempo en segundos
231      f1 =amp*sc.sawtooth(2*np.pi*f*n+fase,1)        # Definimos una onda
232      return f1,n
```

A estas funciones se le aplica la Transformada Discreta de Fourier, en sus dos versiones la manual(la cual se hace un cálculo mediante un For Next), y la otra opción utilizando la librería numpy y la Transformada Rápida de Fourier. La idea es poder comparar ambos resultados.

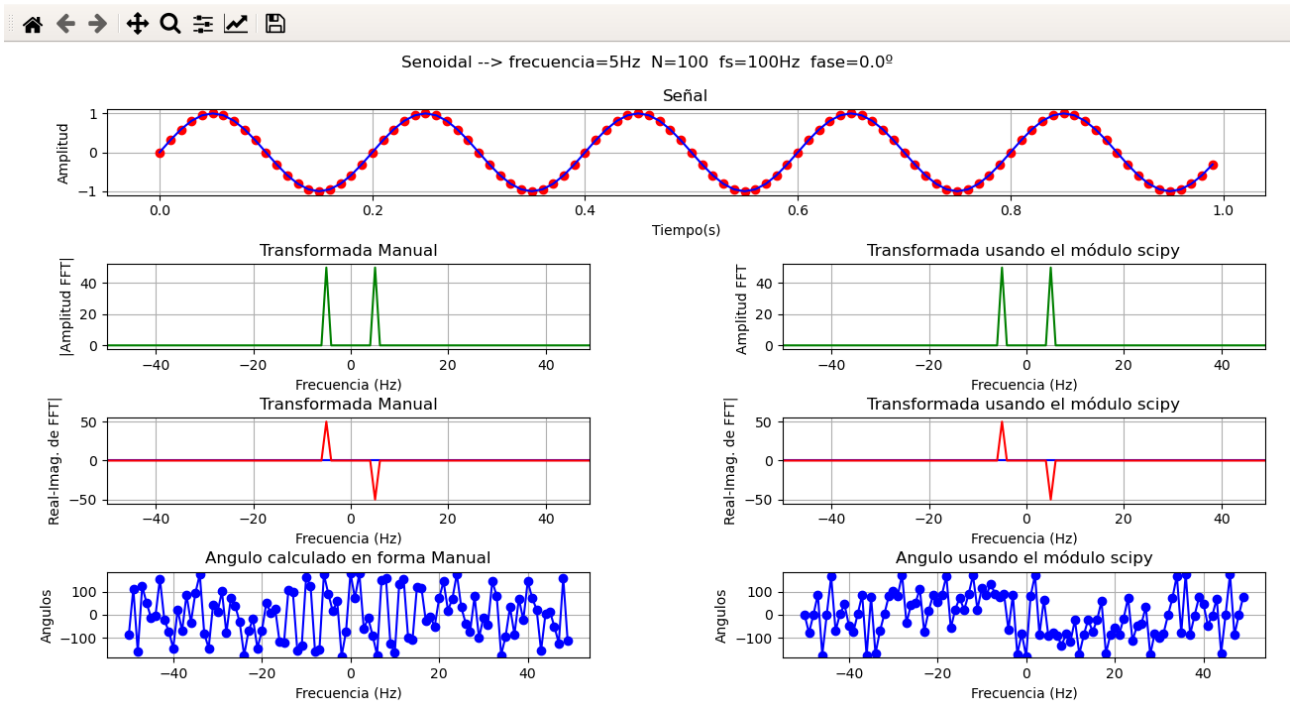
A continuación se describe el código de ambas funciones del archivo ***TP1_folino_v3.py*** :

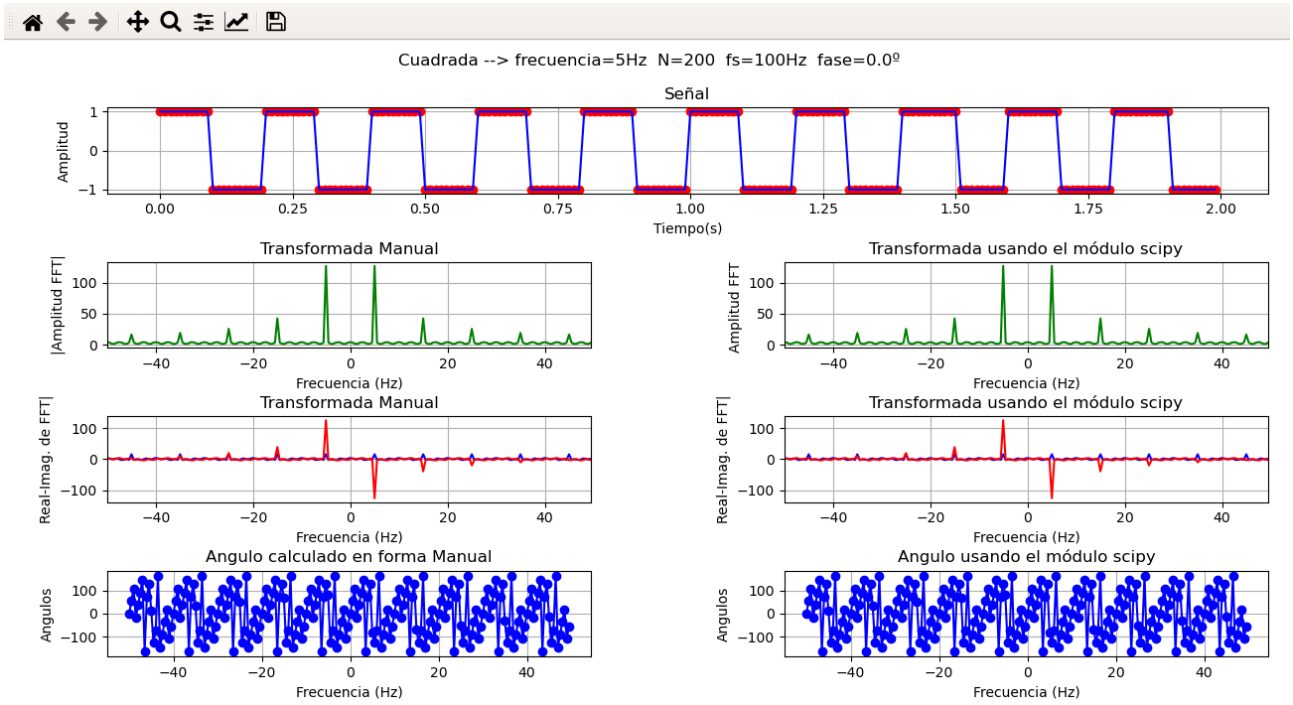
```
53  def tdf_manual(funcion,N,fs):
54      temp=[]
55      # Genero X de N posiciones
56      df=0                          # delta de frecuencia
57      for i in range(N):
58          X.append(complex(0))           # donde se guarda TDF los números complejos
59          F.append((-fs/2)+df*(fs/N))    # donde se guarda la frecuencia
60          df+=1
61      # Barro por frecuencia in range(int(-fs/2),int(fs/2-(fs/N))):
62      for frec in range(0,N,1):
63          for k in range(0,N,1):
64              X[frec]+=funcion[k]*(np.exp(-1j*2*np.pi*F[frec]*k/fs))
65      return X,F
```

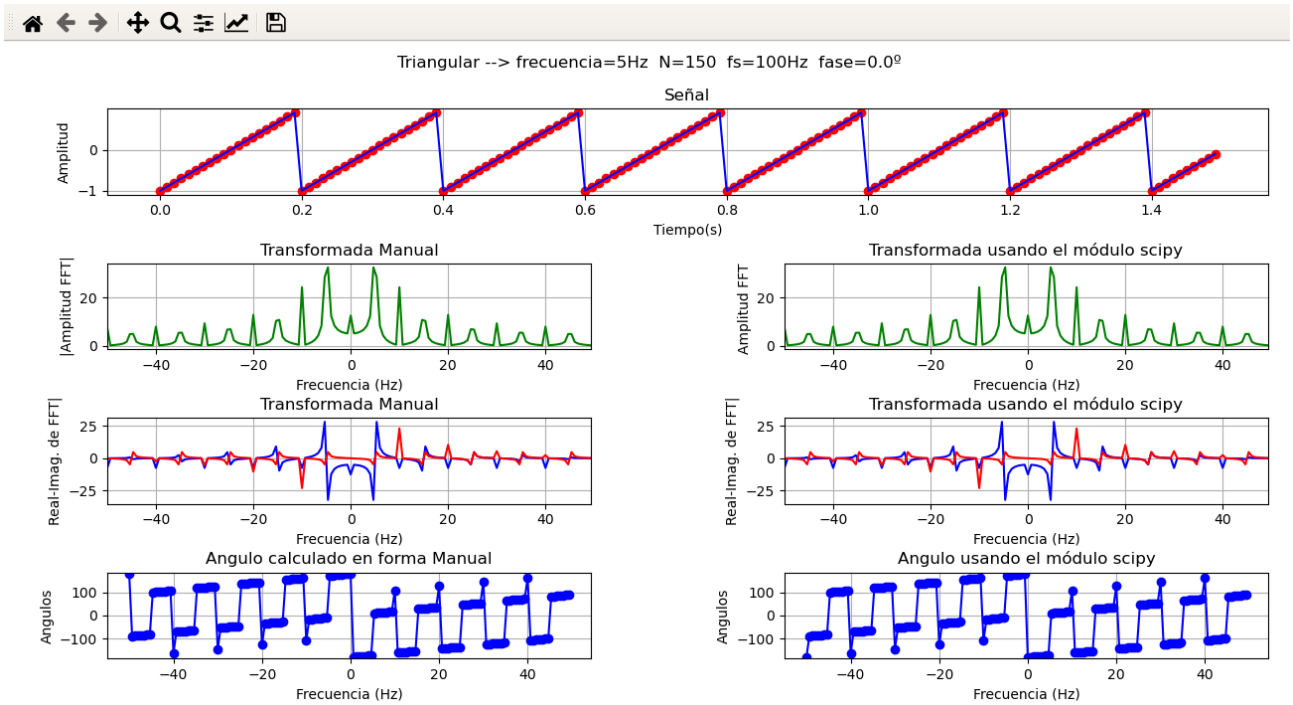
Para la transformada rápida el módulo numpy entrega un resultado con un corrimiento, por ende para representarlo en forma correcta se opto por generar una función que corrija dicho funcionamiento.

```
67  # Se usa para acomodar el orden de las listas que entrega numpy.fft
68  def rotar(lista):
69      for n1 in range(0,int(len(lista)/2)):
70          temp=lista[len(lista)-1]
71          for n2 in range(len(lista)-1,0,-1):
72              lista[n2]=lista[n2-1]
73          lista[0]=temp
74      return lista
75
76  # Transformada Rápida de Fourier
77  def  tdf(funcion,N,ts):
78      X_fft=fft(funcion)
79      X_fft=rotar(X_fft)
80      F_fft=fftfreq(len(funcion),ts)
81      F_fft=rotar(F_fft)
82      return  X_fft,F_fft
```

A continuación se muestran algunas gráficas obtenidas con el programa ***TP1_folino_v3.py***:




Cuadrada --> frecuencia=5Hz N=200 fs=100Hz fase=0.0º
Señal
Amplitud
Tiempo(s)
Transformada Manual
|Amplitud FFT|
Frecuencia (Hz)
Transformada usando el módulo scipy
Amplitud FFT
Frecuencia (Hz)
Transformada Manual
Real-Imag. de FFT|
Frecuencia (Hz)
Transformada usando el módulo scipy
Real-Imag. de FFT|
Frecuencia (Hz)
Angulo calculado en forma Manual
Angulos
Frecuencia (Hz)
Angulo usando el módulo scipy
Angulos
Frecuencia (Hz)


Triangular --> frecuencia=5Hz N=150 fs=100Hz fase=0.0º
Señal
Amplitud
Tiempo(s)
Transformada Manual
|Amplitud FFT|
Frecuencia (Hz)
Transformada usando el módulo scipy
Amplitud FFT
Frecuencia (Hz)
Transformada Manual
Real-Imag. de FFT|
Frecuencia (Hz)
Transformada usando el módulo scipy
Real-Imag. de FFT|
Frecuencia (Hz)
Angulo calculado en forma Manual
Angulos
Frecuencia (Hz)
Angulo usando el módulo scipy
Angulos
Frecuencia (Hz)