Foreword: Scripts are supposed to be written in python and previously written py-scripts can be used as starting point. You can use as well jupyter notebook.

Convolution products with `python`

1 Box Car and Triangle Pulse

Read the demonstrations that lead to define the Fourier Transforms of a Box Car and a Triangle Pulse.

1.1. Create a time domain ranging from -30 s to 30 s, with a sampling rate of 500 Hz, see item 2.1. for ideas on how to do this.

1.2. Create the box car function. It is defined for the time domain and a given width. In the following example the default value for the width is 2 s..

```
def box_car(t, width=2):
    return np.where(np.abs(t) <= width / 2, 1, 0)
```

1.3. Create a Box Car signal of 1 s width (centered around zero) and with an amplitude of 1, as defined in the `box_car` function.

1.4. Assuming that the Box Car signal is named `box`, computes the Fourier Transform and center it:

```
dt=1.0/SR # delta_t= t[1]-t[0]
frq = np.fft.fftfreq(len(box), dt) # frequency domain
ft = np.fft.fft(box) # Fourier Transform

ft = np.fft.fftshift(ft) # center the FT
frq = np.fft.fftshift(frq) # center the frequencies
ft_magnitude = np.abs(ft)*dt # normalisation
```

1.5. Write the script lines in order to obtain a plot as similar as in Fig. 1. For the sake of clarity, the time domain is plotted between -3 and +3 s whereas the frequency domain is ranging in $[-8, +8]$ Hz.

1.6. Using the definition of the Fourier Transform of a Box Car, plot the Sine cardinal in green and compare it to the Fourier Transform of the Box Car (Fig. 1).

1.7. Write a new script in order to compute the convolution product of two identical box cars defined as in item 1.3.. Most of the preivous script can be copied into the new one, and the convolution product will be computed using

```
dt=1.0/SR # delta_t = t[1]-t[0]
conv_p = np.convolve(box, box, mode='same')*dt
```

1.8. Copy the 1st script into a new one to now perform the Fourier Transform of a Triangle Pulse. The function to create the pulse is

```
def triangle(t, width=2):
    return np.where(np.abs(t) <= width / 2, 1-(2*np.abs(t)/width), 0)
```

1.9. Show that a convolution in time is a simple product in the Fourier domain. To do this it is recommended to superimpose three curves in the Fourier domain.

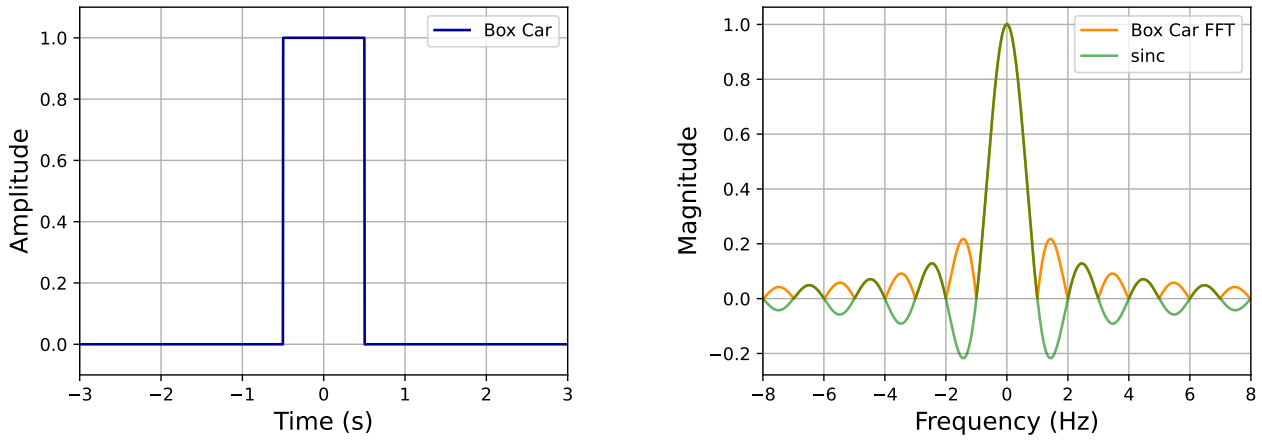

Figure 1: Fourier Transform of a Box Car. The green curve is given by the Sine cardinal function.

2 Gibbs Phenomenon

When a signal is truncated abruptly some oscillations in the Fourier domain become visible due to the convolution of the signal with a Box car (sine cardinal in the frequency domain). This phenomenon is illustrated using firstly a pure sine signal and secondly a multi frequency signal.

2.1. Create a new Python script that generates a sine wave of 0.2 Hz.

```
SR = 1000  # sampling rate (samples per second)
t0=0.0 ; lgth = 20.0  # starting time and length of signal (s)
f = 0.2  # Frequency of the sine wave (Hz)
t = np.linspace(t0, t0+lgth, int(SR*lgth), endpoint=False) # time domain
signal = np.sin(2 * np.pi * f * t) # sine wave
```

2.2. Add the Box Car function (see item 1.2.) and create a box car signal of width=40 s. Then multiply the `signal` by the `box`.

2.3. In order to study the positive frequencies, we have to take only the positive half of the frequencies and corresponding FFT values:

```
ft = np.fft.fft(signal)
dt=1.0/SR # delta_t = t[1]-t[0]
frq = np.fft.fftfreq(len(signal), dt)
positive_frq = frq[:len(frq)//2]
positive_ft = np.abs(ft[:len(ft)//2])
```

2.4. Write the script lines in order to obtain a plot as similar as in FIG. 2. For the sake of clarity, the frequency domain is plotted between 0 and 2.5 Hz.

2.5. Test that everything goes well when multiplying f by 10.

2.6. Change the sine wave frequency to $f = 0.21$ Hz. What's happen? Explain this phenomenon?

2.7. Go back to the initial value of $f = 0.2$ Hz and now change the width of the Box car from the initial value of `T=40` to `T=10`, every 5 s. What's happen? Explain this phenomenon?

2.8. Copy the previous script and instead of a pure sine signal, generate a synthetic signal composed by 12 sine waves at various frequencies between 0.2 and 6 Hz (FIG. 3). One way of doing this is to define randomly the synthetic signal characteristics:

```
np.random.seed(32)
n_frq = 12
frqs = np.random.uniform(0.2, 6, n_frq) # frq values
amp = np.random.uniform(0.5, 2, n_frq) # amplitudes
phs = np.random.uniform(0, 2 * np.pi, n_frq) # phase shifts
```

Explore the effects of a truncation by decreasing `T` from 40 s to 5 s.

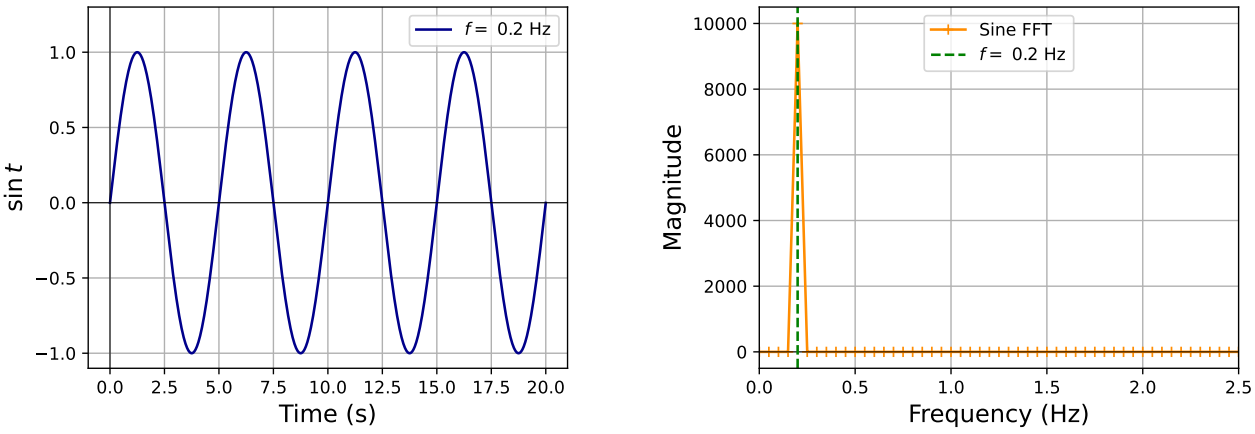

Figure 2: Fourier Transform of a sine signal.

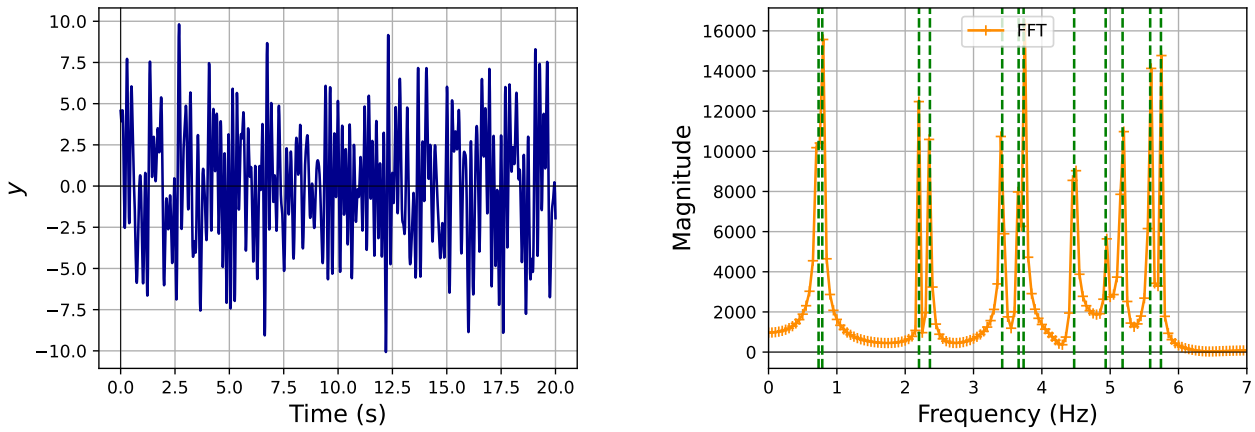

Figure 3: Fourier Transform of a synthetic signal composed by 12 sine waves.