Exercises on Fast Fourier Transform

Giacomo Rizzi

March 26, 2022

1 Introduction

The numerical solutions of the following exercises are reported along with the corresponding codes, which are written in Python3 language. To each exercise there corresponds a code. All the codes make use of the Scipy `fft` function [1] and of a set of functions which are compactly grouped in a file named *"module_1.py"*, which is fully reported in Appendix (see 1). All the codes are reported on Github.

Exercise 1.1

- *Generate a dataset of 12 data points that represents a sine wave with amplitude 1 and frequency 1 Hz in the time interval 0≤t< 1 s.*

```
# Dependencies
import matplotlib.pyplot as plt
import numpy as np
from scipy.fft import fft, ifft
from module_1 import *

# Sampling
# Length of the signal
L = 1
# Sampling frequency
sf = 12
# Sampling points
sp = sf*L
# Time partition
t_part = np.linspace(0,L,sp,endpoint=False) #[s]
# Frequency partition
f_part = np.arange(0,sp/L,1/L)


# Sinewave signal parameters
freq = 1 # frequency of the signal in Hz (1/period)
A = 1. # Amplitude of the signal [V]
phi = 0. # phase shift of the signal [degree]
# Sinewave signal function
omega = 2*np.pi*freq
func = A*np.sin(omega*t_part + degree_to_rad(phi))
```

- *Apply the fft-algorithm to calculate the complex Fourier coefficients. Calculate the amplitude and the phase of the sinewave from the Fourier coefficients.*

The goal of this part calculate the amplitude (A) and the phase (ϕ) of the sinewave generated in the previous exercise from the Fourier's coefficients. First of all, let's define Discrete Fourier Transform (DFT) of the function $x[n]$ as:

$$y[k] = \sum_{n=0}^{N-1} x[n] \exp\{-2\pi i \frac{kn}{N}\}, \tag{1}$$

and the Inverse Fourier Transform (IFT)

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} y[k] \exp\{2\pi i \frac{kn}{N}\}. \tag{2}$$

The Parseval's theorem for the DFT states that

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |y[k]|^2. \tag{3}$$

At this point, let's assume that among all the Fourier's coefficients $y[k]$ there exists one that is such that $|y[\bar{k}]|^2 >> |y[k]|^2 \quad \forall k \neq \bar{k}$. Under that assumption, the summation of the term on the right collapses at $\bar{k}$ and Eq.3) rewrites as

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{2}{N} |y[\bar{k}]|^2, \tag{4}$$

where the term on the right has been multiplied by two to account for the fact that summing from 0 up to N-1 two equal values $y[\bar{k}]$ sums up since $x[n]$ is real. Then, by multiplying and dividing by N the term on the left and by recalling the definition of the room mean square we get that

$$N x_{rms}^2 = \frac{2}{N} |y[\bar{k}]|^2. \tag{5}$$

Eventually, for a sine wave it must be that $x_{rms}^2 = A^2/2$. Therefore:

$$A = \frac{1}{N/2} y[\bar{k}]. \tag{6}$$

From the knowledge of $\bar{k}$ it is also possible to evaluate the phase of the sine wave. Indeed, from the condition that absolute square value of the Fourier's coefficient associated with $\bar{k}$ is much greater than the rest of the coefficients, it follows that the leading information on the phase is carried by $y[\bar{k}]$ and thus

$$\phi = \arctan\Big(\frac{\mathrm{Im}(y[\bar{k}])}{\mathrm{Re}(y[\bar{k}])}\Big). \tag{7}$$

With regard to this latest point, it is important to note that the resulting phase ϕ has to be shifted by 90° due to the fact that the input function is a sine wave, which is negatively shifted of 90° with respect to the cosine.

```
# fourier transform of func
ft = np.empty(sp, dtype = np.complex_)
ft = fft(func) # backward normalization (i.e, no normalisation for fft, 1/sp for ifft)

# inverse fourier transform of ft
ift = np.empty(sp, dtype = np.complex_)   # *1/sp
ift = ifft(ft)


# modulus and phase spectrum
modulus = np.abs(ft[0:int(sp/2)])
phase = np.arctan2(ft.imag[0:int(sp/2)],ft.real[0:int(sp/2)])

# Index associated with the maximum Fourier's coefficient
max_mod = np.max(modulus)
index = np.where(modulus==max_mod)

# Amplitute of the signal through maximum fourier coefficient
A_max = max_mod/(sp/2)
phase_max = phase[index]

# PRINTING AMPLIUDE AND PHASE OF THE SIN FUNCTION
print("PRINTING THE VALUES OF THE AMPLITUDE AND THE PHASE OF THE SINEWAVE CALCULATED FROM THE FOURIER'S COEFFICIENTS")
print("------------------------------------------------------------------------")
print("AMPLITUTE:")
print("Expected value = %1.8f"%A)
print("Calculated from the maximum Fourier coefficient = %1.8f"%A_max)
print("----------------------------------------------------------")
print("PHASE")
print("Expected value = %1.8f"%phi)
print("Calculated from the maximum Fourier coefficients = %1.8f"%(90+rad_to_degree(phase_max)))

# Creating visualization
fig, ax = plt.subplots(3)

# plotting the sin function and the inverse fourier transform
ax[0].plot(t_part,func,"-",color = "red",label=r"$sin(\omega t)$")
ax[0].plot(t_part,ift.real,".",color = "blue",label =r"$F^{-1}[F[\nu]]$")
ax[0].legend()
ax[0].set_xlabel("Time (s)")
ax[0].set_ylabel("Signal [a.u]")

# plotting the modulus of the fourier transform
ax[1].plot(f_part[0:int(sp/2)],modulus[0:int(sp/2)], marker='o',markerfacecolor="red")
ax[1].set_xlabel("Frequency [Hz]")
ax[1].set_ylabel("Amplitude [a.u]")

# plotting the fourier transform (imaginary part)
ax[2].plot(f_part[0:int(sp/2)],rad_to_degree(phase[0:int(sp/2)])+90, marker='o',markerfacecolor="red")
ax[2].set_xlabel("Frequency [Hz]")
ax[2].set_ylabel("Phase [degree]")

plt.show()
```

OUTPUT

The program returns the expected an calculated amplitude and phase of the sine wave given as input. In addition to that, it plots the sine wave and the IFT together to verify that the FFT algorithm is working properly (see Fig. 3). Eventually, the modulus and the phase of the Fourier transform of the sine wave are plotted.

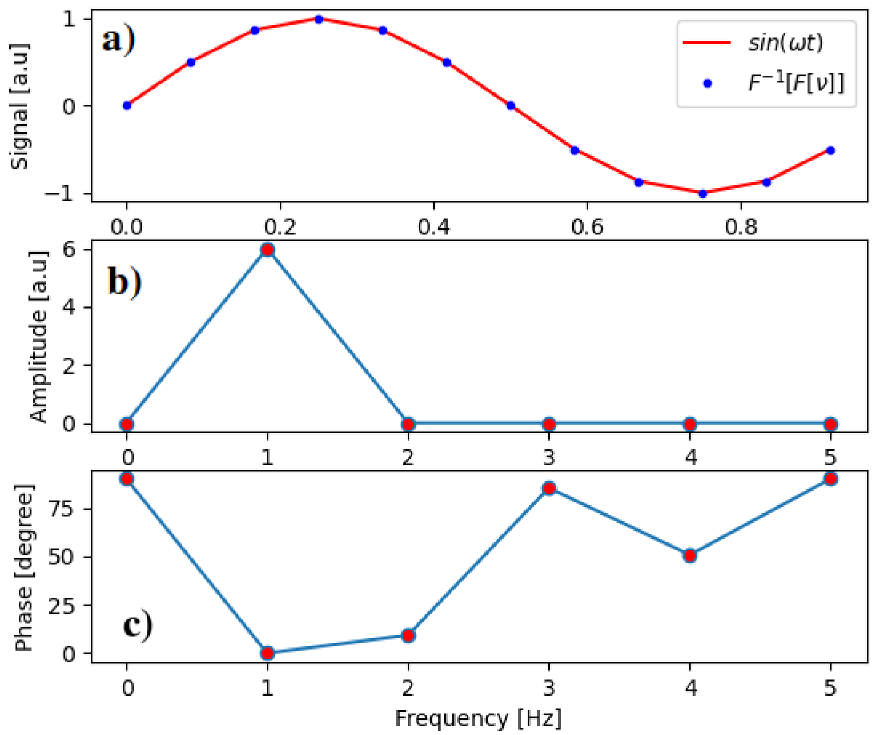


Figure 1: Output of the exercise 1.1. The plot (a) represents is the sine wave with frequency of 1 Hz in the interval 0≤t< 1 s (red line) along with the inverse Fourier points (blue dots). In the second (b) the modulus of the fourier coefficients $y[k]$ is reported (red dots, connected with a fictitious blue line). In the last figure (c) the the phase of the Fourier coefficient is reported, which goes coherently to zero.

```
PRINTING THE VALUES OF THE AMPLITUDE AND THE PHASE OF THE SINEWAVE
CALCULATED FROM THE FOURIER'S COEFFICIENTS
--------------------------------------------------------------------
AMPLITUTE:
 Expected value = 1.00000000
 Calculated from the maximum Fourier coefficient = 1.00000000
--------------------------------------------------------------------
PHASE
```

```
Expected value = 0.00000000
Calculated from the maximum Fourier coefficients = 0.00000000
```

Exercise 1.2

- *Generate a signal of 2s length with sampling frequency 500 Hz that contains (i) a white background noise as produced by a resistance of 1MW at 300K and a bandwidth of 10 kHz, (ii) a sine wave signal at frequency 80 Hz with amplitude 12 mV, (iii) a sine wave signal at frequency 170 Hz with amplitude 6 mV.*

```
# Dependencies
import matplotlib.pyplot as plt
import numpy as np
from scipy.fft import fft,ifft
from module_1 import *

# Sampling
# length of the signal
L = 2
# sampling frequency
sf = 500
# sampling points
sp = sf*L

# time partition
t_part = np.linspace(0,L,sp,endpoint=False) #[s]
# frequency partition
f_part = np.arange(0,sp/L,1./L)

#--------------------------------------------------------------
# Signal 1: white noise
from numpy.random import normal

def white_noise(N,A = 1):
    return A*normal(0,1,N)

# noise parameters
T = 300    #Temperature [K]
R = 1e6    #Resistance [Ohm]
df = 1e4 #Bandwidth [Hz]
kb = 1.38064852e-23 #[m2 kg s-2 K-1] Boltzmann constant
Vrms = np.sqrt(4*kb*T*R*df) #Johnson-Nyquist noise

V_1 = white_noise(sp,Vrms)
#--------------------------------------------------------------
# Signal 2: sinewave

A2 = 12e-6 # Signal amplitude [V]
freq2 = 80  # Frequency of the sinewave [Hz]
V_2 = A2*np.sin(2*np.pi*freq2*t_part)
#--------------------------------------------------------------
# Signal 3: sinewave
A3 = 6e-6
freq3 = 170
V_3 = A3*np.sin(2*np.pi*freq3*t_part)
```

```
#------------------------------------------------------------

# Combining all the three signals
V_comb = V_1 + V_2 + V_3
```

- *Calculate the power-spectrum from the signal in units of V2/Hz and dB. Test if the area of the power spectrum is equal to the rms value of the signal.*

  The standard deviation of the set of data $\{x[n]\}$, is defined as:

$$\sigma = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}(x[n]-\mu)^2}, \tag{8}$$

  Where μ is the arithmetical mean of $\{x[n]\}$. In the case under exam we are summing three signals whose mean is equal to zero, thus $\mu = 0$ and the standard deviation reduces to the root mean square of the function $x[n]$. The square of this value can compared with the integral of the power spectrum of $x[n]$.

```
# Combining all the three signals
V_comb = V_1 + V_2 + V_3

# Evaluating the power spectrum of the combined signal in V^2/Hz and in dB
P_V_comb = power_spectrum(V_comb,sp,L)
P_V_comb_dB = dB(P_V_comb)

# Evaluating the V_rms from statistical argument
V_stat = np.std(V_comb)

# Integrating over the power spectrum by means of the trapezoidal formula
f_min = 0.
f_max = f_part[int(sp/2)]
area_P_comb = trapezoidal(P_V_comb[0:int(sp/2)],f_min,f_max,int(sp/2))

print("COMBINED SIGNAL: V_1 + V_2 + V_3")
print("Statistical V_rms^2",V_stat**2)
print("V_rms^2 from the trapezoidal formula",area_P_comb)
```

- *Determine from the power spectrum the power of the combined signal and the power of each of its single constituents. Compare to the theoretical values.*

  In this part of the exercise the root mean squared value of V_{rms} signal is calculated in three different modes:

  1. Theoretically, by adding up the squared V_{rms} of the single signal components calculated from their amplitudes.
  2. By integrating the power spectrum of the combined signal.
  3. By integrating the power spectrum of each signal component and then summing up the results.

```
# Evaluating the V_rms^2 from the Power spectrum
totP_V_comb = tot_power(P_V_comb,sp,L)

# Evaluating the theoretical total power of the combined signal
totP_th = Vrms**2+A2**2/2+A3**2/2  # theoretical value of the power for the three (uncorrelated) signals

# Calculating the power spectrum in V^2/Hz of the single components
P_V1 = power_spectrum(V_1,sp,L)
P_V2 = power_spectrum(V_2,sp,L)
P_V3 = power_spectrum(V_3,sp,L)

# Evaluating the total power of the three signal
totP_V1 = tot_power(P_V1,sp,L)
totP_V2 = tot_power(P_V2,sp,L)
totP_V3 = tot_power(P_V3,sp,L)

# Printing results
print("SIGNAL 1")
print("theoretical V_rms:",Vrms**2)
print("V_rms calculated from the sum over the power spectrum component:",totP_V1)
print("------------------------------------------------------------")
print("SIGNAL 2")
print("theoretical V_rms:",A2**2/2)
print("V_rms calculated from the sum over the power spectrum component:",totP_V2)
print("------------------------------------------------------------")
print("SIGNAL 3")
print("theoretical V_rms:",A3**2/2)
print("V_rms calculated from the sum over the power spectrum component:",totP_V3)
print("------------------------------------------------------------")
print("COMBINED SIGNAL: V_1 + V_2 + V_3")
print("Theoretical power:",totP_th)
print("Power from the power spectrum of the combined signal",totP_V_comb)
print("Power from the sum of the power of the single components",totP_V1+totP_V2+totP_V3)
```

- *Investigate how the signal in the power-spectrum at 170 Hz depends on the sampling frequency and on the signal length. Demonstrate your results with a figure.*

  Let's call f_{max} the maximum frequency component of a signal; the Nyquist–Shannon sampling theorem guarantees that, if the signal is sampled with a frequency f_s greater or equal twice the value of f_{max}, then all the pieces of information of the original signal can be restored.
  In this section, we investigate the dependence of the sine wave (at 170 Hz) on the sampling frequency and on the signal length. From the Nyquist–Shannon theorem we should sample the signal with frequencies at least of 340 Hz.

```
# sampling frequency
sf = [100,300,500,750,1000] # Hz
# length of the signal
L = [0.5,2.5,5,7.5,10]

```

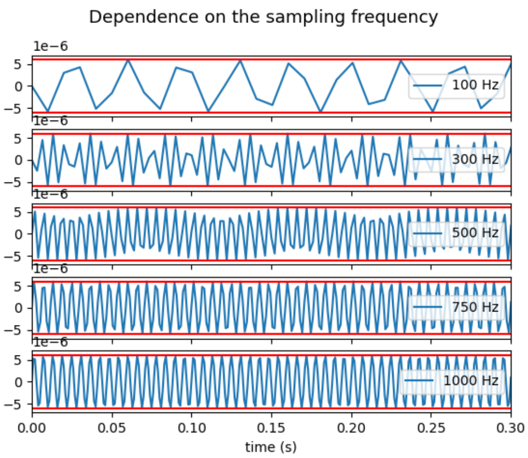
```
     # Creating visualization to observe the dependence on the sampling frequency
1006 fig5 , ax5 = plt.subplots(len(sf),sharex = True)
     fig5.suptitle('Dependence on the sampling frequency', fontsize=13)
1008 ax5[len(sf)-1].set_xlabel("time (s)")

1010 # the length is fixed and the sampling frequeny varies
     length = 2 #s
1012 # iterating over the various sampling frequency
     for i,freq in enumerate(sf):
1014     sp = int(freq*length)
         t = np.linspace(0,length,sp,endpoint=False)
1016     V_3 = A3*np.sin(2*np.pi*freq3*t)
         ax5[i].plot(t,V_3, label= str(freq)+" Hz")
1018     ax5[i].axhline(-A3, color = "red")
         ax5[i].axhline(A3,color = "red")
1020     ax5[i].set_ylim(-7e-6,7e-6)
         ax5[i].set_xlim(0,0.3)
1022     ax5[i].legend(loc = "right")

1024 # Creating visualization to observe the dependence on the sampling frequency
     fig6 , ax6 = plt.subplots(len(L),sharex = True)
1026 fig6.suptitle('Dependence on the signal length', fontsize=13)
     ax6[len(L)-1].set_xlabel("time (s)")
1028
     # the sampling frequency is fixed and the length of the signal varies
1030 freq = 500 # Hz
     # iterating over the various length of the signal
1032 for i,length in enumerate(L):
         sp = int(freq*length)
1034     t = np.linspace(0,length,sp,endpoint=False)
         V_3 = A3*np.sin(2*np.pi*freq3*t)
1036     ax6[i].plot(t,V_3, label= str(length)+" sec")
         ax6[i].axhline(-A3, color = "red")
1038     ax6[i].axhline(A3,color = "red")
         ax6[i].set_ylim(-7e-6,7e-6)
1040     ax6[i].set_xlim(0,0.3)
         ax6[i].legend(loc = "right")
1042 plt.show()
```

OUTPUT

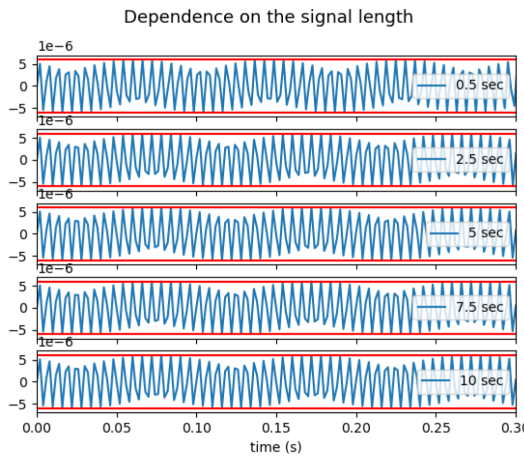
The program returns both graphical and numerical results. The numerical ones are:

- The V_{rms}^2 of the combined signal calculated from the standard deviation (statistical) and from the integration (trapezoidal). Lines: (1000-1002)
- The V_{rms}^2 of the single signals, which constitute the combined signal, calculated theoretically as well as from their power spectra. Lines: (1004-1014)
- The V_{rms}^2 calculated from the sum of the single signal V_{rms}^2 (theoretical), the integral of the power spectrum and the sum of the integrals of the power spectra of the single constituents. Lines: (1016-1019)

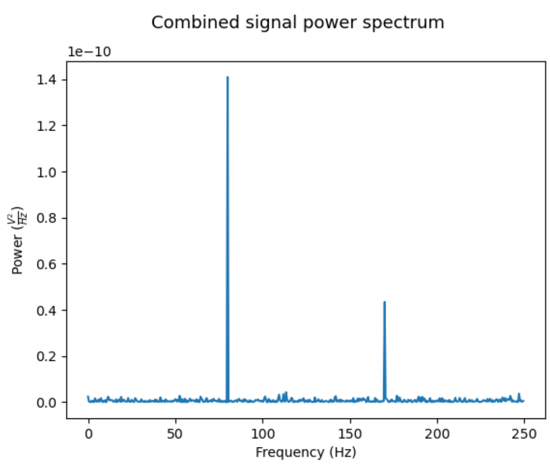
The graphical outputs are the following plots:



(a) Dependence of the signal shape on the sampling frequency. The signal (blue) suffers from aliasing for frequency smaller than 1 KHz. This can be easily noticed by considering that the maximum of the signal should occur on the horizontal red lines, which are drawn at the amplitude of the planewave.



(b) Dependence of the signal on the length in the interval from 0 up to 30 s. No substantial dependences are observed.



(c) Power spectrum of the combined signal: two peaks are observed in correspondence of the frequencies of the sine waves. In addition, an overall background noise is present as a consequence of the white noise.

Figure 2: Output of the exercise 1.2

```
COMBINED SIGNAL: V_1 + V_2 + V_3
 Statistical V_rms^2 2.544797727868169e-10
V_rms^2 from the trapezoidal formula 2.5448380427660406e-10
--------------------------------------------------------------------
SIGNAL 1
theoretical V_rms: 1.6567782239999995e-10
V_rms calculated from the sum over the power spectrum component: 1.6570848268156656e
    -10
---------------------------------------------------------------------------
SIGNAL 2
theoretical V_rms: 7.200000000000001e-11
V_rms calculated from the sum over the power spectrum component: 7.199999999999992e
    -11
--------------------------------------------------------------------------
SIGNAL 3
theoretical V_rms: 1.8000000000000002e-11
V_rms calculated from the sum over the power spectrum component: 1.800000000000002e
    -11
--------------------------------------------------------------------------
COMBINED SIGNAL: V_1 + V_2 + V_3
 Theoretical power: 2.5567782239999997e-10
Power from the power spectrum of the combined signal 2.5447965300343255e-10
Power from the sum of the power of the single components 2.557084826815665e-10
```

Exercise 1.3

- *The asci-dataset "Impedance" contains 3 columns: Column 1 and 2 represent time and voltage data of the input signal that contains 5 principal frequencies. Column 3 shows the current of the output signal. Use the data to calculate a Bode Plot that shows the magnitude of impedance and phase angle of the measured circuit. Estimate R and C values from the data.*

In "*impedance*" data contains voltage and current as a function of time. It is possible to expand them in Fourier series, to obtain:

$$V(t) = \frac{1}{N}\sum_{k=0}^{N-1} \tilde{V[k]} \exp 2\pi i \frac{kt}{N},$$
$$I(t) = \frac{1}{N}\sum_{k=0}^{N-1} \tilde{I[k]} \exp 2\pi i \frac{kt}{N}.$$

By defining the impedance of the circuit as

$$Z(k) = \frac{\tilde{V[k]}}{\tilde{I[k]}}, \tag{9}$$

it is evident that a sensible radio involves only the Fourier components associated with the frequencies that constitute the signal, while the remaining ones can be discarded. Therefore, a function that finds the peaks of £$V(t)$ and $I(t)$ is needed and the impedance will be calculated only in correspondence of these peaks.

```python
# Dependencies
import matplotlib.pyplot as plt
import numpy as np
from scipy.fft import fft, ifft
from scipy.optimize import curve_fit
import pandas as pd

def PeaksFinder(arr,std):
    """
    Calculates the multiple peaks (if any) of arr, along with the indeces at which they occur.
    ----------------------------------------------------------------------------
    input:
        arr: REAL, N-dimensional. Array whose maxima have to be computed.
        std: REAL. Standard deviation of the array calculated in the region without peaks.
    output:
        pos: list, INTEGER.
    """
    n = len(arr)
    peaks = []  # contains the values of arr in correspondence of the peaks
    pos = []   # contains the indeces of arr in correspondence of the peaks
    # First element
    if (arr[0]-arr[1] >= 3*std):
        peaks.append(arr[0])
        pos.append(0)

    # from the second to the one before the last
    for i in range(1, n-1):
        if (arr[i] - arr[i-1] >= 3*std and arr[i] - arr[i+1]>= 3*std):
            peaks.append(arr[i])
            pos.append(i)
    # last element
    if (arr[n-1] - arr[n-2] >= 3*std):
        peaks.append(arr[n-1])
        pos.append(n-1)
    return pos, peaks


def Z_RC_par_mod(freq,R,C):
    """
    Calculates the modulus of the complex impedance of an RC parallel circuit
    ---------------------------------------------------------------------------------
    input:
        freq: REAL, N-dimensional. An array with the frequencies in Hz
        R: REAL. Resistance of the circuit
        C: REAL. Capacitance of the circuit
    output:
        modulus of the impedance
    """
    return R / np.sqrt(1+(2*np.pi*C*R*freq)**2)

def Z_RC_par_phase(freq,tau):
    """
    Calculates the phase of the complex impedance of an RC parallel circuit
    ---------------------------------------------------------------------------------
    input:
        freq: REAL, N-dimensional. An array with the frequencies in Hz
```

```python
        tau: REAL. tau = RC, is the characteristic time.
    output:
        phase of the impedance
    """
    return np.arctan(-2*np.pi*freq*tau)

#selecting the x-range to fit
def select_range(df,header,x_min,x_max):
    """
    Returns a logic array whose components are TRUE when the elements of df[header]
    are within [xmin,xmax], FALSE elsewhere.
    ----------------------------------------------------------------------
        df: Pandas Dataframe object. (NXM), with M number of columns and N number of rows
        header: String object. Contains the name of the column of df to which xmin, xmax are referred
        xmin,xmax: REAL. Extremes of a subinterval of df[header]
    output:
        selected_df: array, N-dimensional. Its elements are FALSE where df[header] is not within [xmin,xmax]
    """
    selected_df = df.loc[(df[header] >= x_min) & (df[header] <= x_max)]
    return selected_df

#Importing the excel data through the pandas function read_excel
data = pd.read_excel("../../Data/data_DAQ.xls")

# Interpreting and converting data into numpy arrays
time = np.array(data["time(s)"])
volt = np.array(data["input(V)"])
amp = np.array(data["output(A)"])

# sampling point
sp = len(time)

#Defining length of the signal (L) and sampling frequency (sf)
L = time[-1]-time[0]
sf = 1/(time[1]-time[0])

# Defining frequency partition
f_part = np.arange(0,(sp)/L,1./L)

#Calculting DFT of volt and amp
v_dft = fft(volt)
a_dft = fft(amp)

# Calculating stardard deviation of voltage and current in a FLAT interval
# 1. Creating a new Pandas DataFrame that contains the FFT of volt, amp
df_freq = pd.DataFrame(columns =["volt", "amp", "freq"])
df_freq["volt"] = v_dft
df_freq["amp"] = a_dft
df_freq["freq"] = f_part
# 2. Selecting the "flat" range where the standard deviation is computed
x_min = 1e4
x_max = 1.8e4
df_flat = select_range(df_freq,"freq",x_min,x_max)
# 3. Using  std() method to calculate the standard deviation of v_dft and a_dft
std_values = df_flat.std()
v_std = std_values[0]
```

```
a_std = std_values[1]

# Finding the peaks position ror v_dft
pos, _ = PeaksFinder(v_dft[0:int(sp/2)],v_std)

# calculating the impedence
Z = v_dft/a_dft
Z_peak = Z[pos]

# Calculating phase and modulus of the impedance
Z_phase = np.arctan2(Z_peak.imag,Z_peak.real)
Z_modulus = np.abs(Z_peak)

# Guessing the value of resistance R and capacitance C
R = 1e5    # R = 100 KOhm
C = 1e-9   # C = 1nF
tau = R*C

# Defining a frequency array for plotting
freq_arr = np.linspace(pos[0],20000,1000)

# Fitting the phase of the impedance
par_phase, covs_phase = curve_fit(Z_RC_par_phase,f_part[pos],Z_phase,
                                              p0=[tau])

covs_phase = np.sqrt(np.diag(covs_phase))
# Fitting the modulus of the impedace
par_mod, covs_mod = curve_fit(Z_RC_par_mod,f_part[pos], Z_modulus,
                                              p0=(R,C))
covs_mod = np.sqrt(np.diag(covs_mod))

print("Printing results of the curve fittings on the phase and modulus of the
    impedance")
print("")

print("CURVE FITTING OF THE PHASE OF THE IMPEDANCE")
print("Parameter: tau")
print("Tau: ",par_phase[0],"+-",covs_phase[0],"s")
print("---------------------------------------------")

tau_mod = par_mod[0]*par_mod[1]
err_tau_mod = np.sqrt((par_mod[1]*covs_mod[0])**2+(par_mod[0]*covs_mod[1])**2)

print("CURVE FITTING OF THE PHASE OF THE IMPEDANCE")
print("Parameters: Resistance, Capacitance")
print("Resistance: ",par_mod[0],"+-",covs_mod[0],"Ohm")
print("Capacitance: ",par_mod[1],"+-",covs_mod[1],"F")
print("Tau:",tau_mod,"+-",err_tau_mod,"s")
print("---------------------------------------------")

tau_phase_mod = abs(par_phase[0]-tau_mod)
err_tau_phase_mod = np.sqrt(err_tau_mod**2 + covs_phase[0]**2)

print("COMPATIBILITY OF THE TWO TAU")
print("|Tau(phase)- Tau(modulus)|=", tau_phase_mod ,"+-",err_tau_phase_mod,"s")
fit_phase = Z_RC_par_phase(freq_arr,par_phase)
fit_modulus = Z_RC_par_mod(freq_arr,par_mod[0],par_mod[1])

fig7,ax7 = plt.subplots(2,sharex=True)
```

```
ax7[0].semilogx(f_part[pos],rad_to_degree(Z_phase),".",markersize =10 ,color = "red", label ="data_DAQ")
ax7[0].semilogx(freq_arr,rad_to_degree(fit_phase), color = "blue", label ="Fitting")
ax7[0].set_title("Impedance phase")
ax7[0].legend()
ax7[0].set_ylabel(r'$atan2(\frac{Im(Z)}{Re(Z)}) \ [degree]$')
ax7[1].set_ylabel('|Z| [dB]')
ax7[1].semilogx(f_part[pos],20*np.log10(Z_modulus),".", markersize = 10,color = "red", label ="data_DAQ")
ax7[1].semilogx(freq_arr, 20*np.log10(fit_modulus), color = "blue", label ="Fitting")
ax7[1].set_title("Impedance modulus")
ax7[1].legend()
ax7[1].set_xlabel("Frequency [Hz]")
fig7.tight_layout()

plt.show()

plt.show()
```

Output

The outputs of the exercise concern the results of the fit performed on the impedance data, which by inspection are found to be collected by an RC circuit in parallel.

```
CURVE FITTING OF THE PHASE OF THE IMPEDANCE
Parameter: tau
Tau:  0.00028804076418770323 +- 4.385292427427088e-06 s
----------------------------------------------------------
CURVE FITTING OF THE PHASE OF THE IMPEDANCE
Parameters: Resistance, Capacitance
Resistance:  99971.02780246854 +- 276.1548302116769 Ohm
Capacitance:  2.982773236952017e-09 +- 2.5324627676331495e-11 F
Tau: 0.00029819090619978913 +- 2.662357157229908e-06 s
-------------------------------------------------------------
COMPATIBILITY OF THE TWO TAU
|Tau(phase)- Tau(modulus)|= 1.0150142012085904e-05 +- 5.130198369137658e-06 s
```

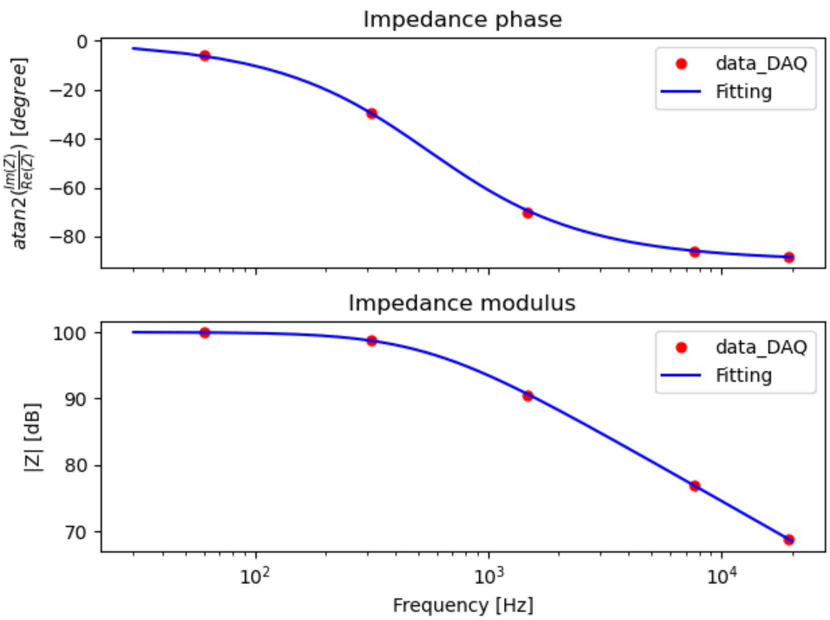


Figure 3: Output of the exercise 1.3. Above: Experimental data of the phase of the impedance (red dots) and fit (blue line). Below: Experimental data (red dots) of the modulus of the impedance along with the fit (blue line).

Exercise 1.4

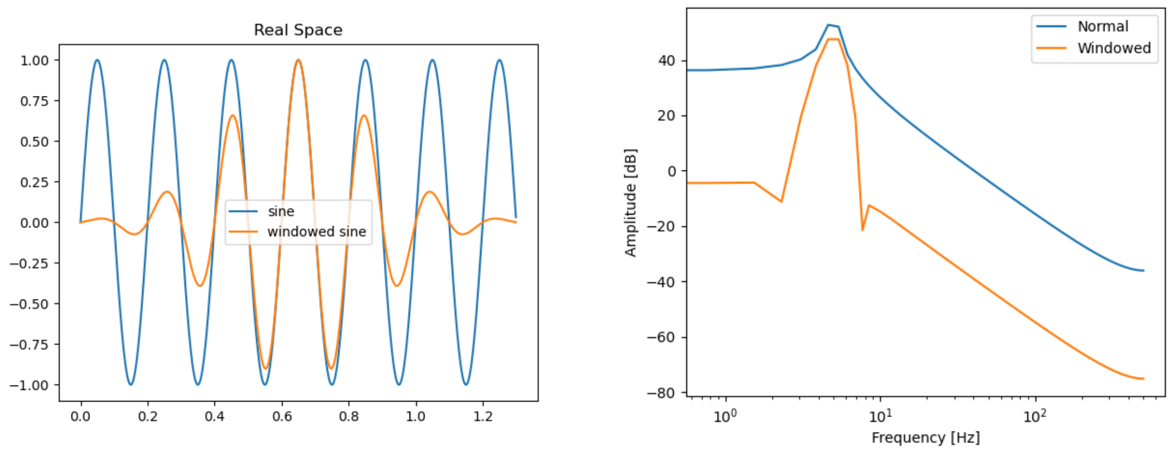
- *Create a numerical example that demonstrates how windowing can improve a power spectrum.*

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.fft import fft, ifft

def gaussian(x,y0,A,x0,width):
    """
    Return a Gaussian function.
    ---------------------------
    INPUT:
    x: x-data array
    y0: offset
    A: amplitude
    x0: shift
    width: standard deviation
    """
    return y0 + A*np.exp(-(x-x0)**2/(2*width**2))

# Length of the signal
L = 1.3    #[s]
# sampl. freq
fs = 1000    #[Hz]
# sampling period
T = 1/fs
# sam. points
sp = int(fs*L)
```

```
# time partition
t_part = np.linspace(0,L,sp,endpoint=False)
# frequency partitio
f_part = np.arange(0,sp/L,1/L)

# sinewave definition
freq = 5    #[Hz]
sine = np.sin(2*np.pi*freq*t_part)

# Fourier transform of sinewave
sine_ft = fft(sine)

# Creating the windowing function.
# A Gaussian function with standard deviation (s) equals to L/6 and centred on the L/2, so that the 3*s falls at the edges of the interval [0, L]
sigma = L/6.
y0 = 0.
x0 = L/2.
A = 1     # to avoid shrinking/enhancing the amplitude of the sinewave function at the centre of [0,L]
window = gaussian(t_part,y0,A,x0,sigma)

# product of the two functions
sine_wind = sine*window

# Fourier transform of sine_wind
sine_wind_ft = fft(sine_wind)

# Creating visualization
fig, ax = plt.subplots()
# Plotting sine and sine_wind in real space
ax.set_title("Real Space")
ax.plot(t_part,sine,label = "sine")
ax.plot(t_part,sine_wind,label = "windowed sine")
ax.legend()

# Creating visualiaztion
fig1,ax1 = plt.subplots()
# Plotting sine_ft and sine_wind_ft in reciprocal space
ax1.set_xlabel("Frequency [Hz]")
ax1.set_ylabel("Amplitude [dB]")
ax1.semilogx(f_part[0:int(sp/2)],20*np.log10(abs(sine_ft[0:int(sp/2)])), label="Normal")
ax1.semilogx(f_part[0:int(sp/2)],20*np.log10(abs(sine_wind_ft[0:int(sp/2)])), label="Windowed")
ax1.legend()

plt.show()
```

OUTPUT

The graphical outputs are the following plots:



(a) Sine wave (blue) and sine wave multiplied by the Gaussian function, i.e windowed sine wave (orange), in real space.

(b) Amplitude (in dB) of the Fourier coefficients of the sine wave (blue) and of the windowed sine wave (orange). It can be seen that the amplitude of the coefficients different from 5 Hz are much smaller in the windowed setup.

Figure 4: Output of the exercise 1.4

Appendix

In this appendix the auxiliary functions that are used in all the four exercises are reported. Writing all the functions inside a single module has the value of increasing the readability of the main code.

```
import numpy as np
from scipy.fft import fft, ifft


def power_spectrum(Vin,N,T):
    """
    calculate the power spectrum of a signal Vin
    -------------------------------------------------
    input:
        -Vin: N-dimensional, REAL or COMPLEX. Input signal
        -N: INTEGER. number of sampling points
        -T: REAL or INTEGER. Length of the signal
    """
    tmp = fft(Vin)
    Vft  = np.empty(int(N/2), dtype = np.complex_)
    Vft = tmp[0:int(N/2)]    # selecting only half of the fourier transform spectrum
    return np.abs(Vft)**2*T/(2*(N/2)**2)


def dB(P_in, P_ref = 1):
    """
    Calculates the power spectrum in unit of dB
    -------------------------------------------
    input:
        -P_in:COMPLEX, N-dimensional. Power spectrum of the signal under test
        -P_ref: REAL/COMPLEX, optinal. Reference power that normalizes P_in
    """
```

```
    return 10*np.log10(abs(P_in)/P_ref)

def trapezoidal(f,xmin,xmax,N):
    """
    Evaluate the integral of the function f within the interval [xmin,xmax],
    over a number of points N
    ----------------------------------------------------------------------
    input:
        -f: function object.
        -xmin,max. Real. Extremes of the interal [xmin,xmax]
        -N: number of points
    """
    delta_k = (xmax-xmin)/(N-1)
    Trap = 0.
    for k in range(1,N):
        Trap += delta_k*(f[k]+f[k-1])/2.0
    return Trap

def tot_power(Power,N,T):
    """
    Calculate the area of the power spectrum, i.e, the total power in V^2 of a
   signal
   ---------------------------------------------------------------------------------
    input:
        - Power: COMPLEX, N-dimensional. Power spectrum of the signal
        - N: INTEGER. Number of sampling points
        - T: REAL or INTEGER: number of sampling points
    output:
        - REAL. Total power
    """
    tmp = 0
    for i in range(1,int(N/2)):
        tmp+=Power[i]
    return tmp/T

def rad_to_degree(rad_angle):
    """
    Returns the angle in degree if provided in radiant.
    INPUT:
      - rad_angle: REAL. Angle [rad]
    OUTPUT: angle [degree]
    """
    return 180./np.pi*rad_angle

def degree_to_rad(degree_angle):
    """
    Returns the angle in radiants if provided in degree.
    INPUT:
      - degree_angle: REAL. Angle [degree]
    OUTPUT: angle [rad]
    """
    return np.pi/180*degree_angle
```

2 References

1. Scipy library website: scipy.fft