Fourier Analysis

1 Introduction to Fourier analysis in 1D

1.1 Fourier series

J.B. Fourier showed that any signal $f(t)$ could be made up by adding together a series of pure tones (sine wave) of appropriate amplitude and phase:

$$f(t) = \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2n\pi}{T}t\right) + b_n \sin\left(\frac{2n\pi}{T}t\right) \right] \tag{1}$$

where T is the period of the function and the coefficients of the series are given by:

$$an = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\cos\left(\frac{2n\pi}{T}t\right)dt, \qquad bn = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\sin\left(\frac{2n\pi}{T}t\right)dt \tag{2}$$

Using the Euler's identity: $e^{ix} = cos(x) + isen(x)$, we can write the Fourier series as:

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{2\pi i \frac{n}{T}t}, \qquad c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)e^{-2\pi i \frac{n}{T}t}dt \tag{3}$$

As an example, we are going to approximate two function using Fourier series:

Fourier serie approximation to a function.

```
import numpy as np
import scipy as sp
import scipy.integrate as itg
import matplotlib.pylab as pl
from scipy.fftpack import fft, fftfreq # for Fourier Transform

pl.ion()

# Function of frequency f modulated by an exponential.
def fun1(t,f):
    return np.cos(2*np.pi*f*t)*np.exp(-np.pi*t**2)
```

```
# Function with fundamental frecuency of f (t in seconds) and a overtone of 2*f

def fun2(t,f):
    return np.sin(2*np.pi*f*t)-1/2*np.sin(2*np.pi*2*f*t)


# We are going to work with a number of points of power of 2 (better for fast fourier transform
    )
# n2 : power of two
# T  : period of the function
# dt : bin width
# ns : number of points per period.

def inpvar(n2,f,ns):
  pn=np.arange(1,2**n2+1)
  T=1/f
  dt=1/(f*ns)
  return pn, T, dt


n2=6 # it will give 2**n2 points
f=400 # Frequency in Hz (f = 1/T )
ns=16

pn, T, dt = inpvar(n2,f,ns)
ln=len(pn)
t=np.linspace(0,(ln-1)*dt,ln)

# Aproximation to the funciton using Fourier series:
# f(t) = sum_0^inf (an*cos(2*n*pi*t/T) + bn*sin(2*n*pi*t/T))
#an=2/T*int_-T/2^T/2*f(t)*cos(2*n*pi/T)
#bn=2/T*int_-T/2^T/2*f(t)*sin(2*n*pi/T)

fun=fun2

def fan(t,T,n):
  return 2/T*fun(t,f)*np.cos(2*n*np.pi*t/T)

def fbn(t,T,n):
  return 2/T*fun(t,f)*np.sin(2*n*np.pi*t/T)

suma=0
for n in np.arange(0,21):
  pl.plot(t,fun(t,f),'-b',linewidth=2)  # original function
  an,err=itg.quad(fan,-T/2,T/2,args=(T,n))
  bn,err=itg.quad(fbn,-T/2,T/2,args=(T,n))
  suma+=an*np.cos(2*n*np.pi*t/T)+bn*np.sin(2*n*np.pi*t/T)
  pl.plot(t,suma,'or') # fourier serie
  input('Press Enter for a new plot')
  pl.clf()
```

1.2 Fourier Transform

Fourier transform converts a physical-space (or time series) representation of a function into frequency-space. The Fourier transform of a function $f(x)$ is given by:

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x k} dx \tag{4}$$

$F(k)$ gives what part of the function $f(x)$ is represented by a frequency k. The inverse of the Fourier transform gives the original function $f(x)$

$$f(x) = \int_{-\infty}^{\infty} F(k) e^{2\pi i x k} dk \tag{5}$$

1.3 Discrete Fourier Transform

For discrete data, we use the Discrete Fourier Transform (DFT), which transform the N spatial/temporal points into N frequency points:

$$Transfrom : F(k) = \sum_{n=0}^{N-1} f_n e^{-2\pi i n k/N} \tag{6}$$

And the inverse form

$$Inverse : f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{2\pi i n k/N} \tag{7}$$

1.4 Normalization

Note that in the DFT we are not using physical coordinates x, we just look at the index n itself. This assumes that the data is regularly gridded. In the index space, the smallest frequency is $1/N$. This means that if we add points to our data, then we open up higher and higher frequencies. The relation between the physical scale for the frequency is given by:

$$\nu_k = \frac{k}{N} \cdot \frac{1}{\delta x} = \frac{k}{N} \cdot \frac{N}{L} \tag{8}$$

Where L is the length of the physical interval. The lowest frequency is given by: $1/L$ and the highest frequency by $\sim 1/\delta x$ If we want to keep the physical scale back when plotting the results, from the definition of the inverse function of the Fourier Transform it follows that we have to plot F_k/N . This is equivalent to say that $F_k/N = F_k dk$

1.5 Power spectrum

The power spectrum is defined as: $P(k) = |F(k)|^2$ and represents a single number showing the importance/weight of a given wavenumber.

1.6 Filtering

In computation, we use de Fast Fourier Transform (FFT) which is equivalent to the DFT but faster because it exploit special symmetries of the function in performing the sums, which increases the velocity of the calculation. Now we are going to calculate the Fourier transform of one of the functions defined previously:

Fourier serie approximation to a function.

```
pl.plot(t,fun(t,f),'k-')
pl.plot(t,fun(t,f),'ro')
pl.clf()


Y = fft(fun(t,f)) / ln  # Gives Fast Fourier Transform normalized (because we divide it by the
    number of intervals).
F = fftfreq(ln,dt)      # All Frequencies sampled.
pl.vlines(F, 0, Y.imag)  # Imaginary part gives the frequency involves.
pl.annotate(s=u'f = 400 Hz', xy=(400.0, -0.5), xytext=(400.0 + 1000.0, -0.5 - 0.35), arrowprops
    =dict(arrowstyle = "->"))
pl.annotate(s=u'f = -400 Hz', xy=(-400.0, 0.5), xytext=(-400.0 - 2000.0, 0.5 + 0.15),
    arrowprops=dict(arrowstyle = "->"))
pl.annotate(s=u'f = 800 Hz', xy=(800.0, 0.25), xytext=(800.0 + 600.0, 0.25 + 0.35), arrowprops=
    dict(arrowstyle = "->"))
pl.annotate(s=u'f = -800 Hz', xy=(-800.0, -0.25), xytext=(-800.0 - 1000.0, -0.25 - 0.35),
    arrowprops=dict(arrowstyle = "->"))
pl.ylim(-1, 1)
pl.xlabel('Frequency (Hz)')
pl.ylabel('Im($Y$)')

# This is awesome, but what if we are not carefull sampling the data:

n2 = 2**5

t2 = np.linspace(0, 0.012, n2)  # Time interval
dt2 = t2[1] - t2[0]
y2 = np.sin(2 * np.pi * f * t2) - 0.5 * np.sin(2 * np.pi * 2 * f * t2)  # Signal sampling

Y2 = fft(y2) / n2  # Fast Fourier Transform normalized
frq2 = fftfreq(n2, dt2) # Frequencies

fig = pl.figure(figsize=(6, 8))

ax1 = fig.add_subplot(211)
ax1.plot(t2, y2)
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('$y_2(t)$')

ax2 = fig.add_subplot(212)
ax2.vlines(frq2, 0, Y2.imag)
pl.xlabel('Frequency (Hz)')
pl.ylabel('Im($Y_2$)')
```

```
# There are new frequencies not expected. This is know as 'leaking' and is because the signal
    is not sampled in a whole period. We can extend our sampling by adding zeros to our data and
     increasing the resolution in frequency. This is known as zero-padding but it has the
    problem that the 'leaking effect is larger:


t3 = np.linspace(0, 0.012 + 9 * dt2, 10 * n2)  # Time inteval
y3 = np.append(y2, np.zeros(9 * n2))  # Signal

Y3 = fft(y3) / (10 * n2)  # Fast Fourier Transform normalized
frq3 = fftfreq(10 * n2, dt2) # Frequencies

fig = pl.figure(figsize=(6, 8))

ax1 = fig.add_subplot(211)
ax1.plot(t3, y3)
ax1.set_xlabel('Time (s)')
ax1.set_ylabel('$y_3(t)$')

ax2 = fig.add_subplot(212)
ax2.vlines(frq3, 0, Y3.imag)
pl.xlabel('Frequency (Hz)')
pl.ylabel('Im($Y_3$)')

# There is a way to reduce the leaking effect by using a window-function. Window-functions are
    function which are null for all values except within an interval. Usually are used to smooth
     out or filtering a signal. We can use some of the numpy's built-in window function, as for
    example: kaiser, bartlett, blackman, hamming, and hanning

# Plot how looks like these window functions:

pl.clf()
pl.plot(np.arange(20), np.kaiser(20,3.5))
pl.plot(np.arange(20), np.bartlett(20))
pl.plot(np.arange(20), np.blackman(20))
pl.plot(np.arange(20), np.hamming(20))
pl.plot(np.arange(20), np.hanning(20))


# If we use blackman window:
n4 = 2 ** 8

t4 = np.linspace(0, 0.05, n4)
dt4 = t4[1] - t4[0]
y4 = np.sin(2 * pi * f * t4) - 0.5 * np.sin(2 * pi * 2 * f * t4)
y5 = y4 * np.blackman(n4)

t4 = np.linspace(0, 0.12 + 4 * dt4, 5 * n4)
y4 = np.append(y4, np.zeros(4 * n4))
y5 = np.append(y5, np.zeros(4 * n4))

Y4 = fft(y4) / (5 * n4)
Y5 = fft(y5) / (5 * n4)
frq4 = fftfreq(5 * n4, dt4)




fig = pl.figure(figsize=(6, 8))
```

```
ax1 = fig.add_subplot(411)
ax1.plot(t4, y4)
pl.xlabel('Frequency (Hz)')
pl.ylabel('$y_4(t)$')

ax2 = fig.add_subplot(412)
ax2.vlines(frq4, 0, abs(Y4))  # Amplitude spectrum
pl.xlabel('Frequency (Hz)')
pl.ylabel('Abs($Y_4$)')

ax3 = fig.add_subplot(413)
ax3.plot(t4, y5)
plt.xlabel('Frequency (Hz)')
plt.ylabel('$y_5(t)$')

ax4 = fig.add_subplot(414)
ax4.vlines(frq4, 0, abs(Y5))  # Amplitude spectrum
plt.xlabel('Frequency (Hz)')
plt.ylabel('Abs($Y_5$)')
```

2 Exercises.

- Calculate the FFT of the following functions:
- $\sin(2\pi f_0 x)$: pure imaginary at a single wavenumber
- $\cos(2\pi f_0 x)$: pure real at a single wavenumber
- $\sin(2\pi f_0 x + \pi/4)$: equal magnitude real and imaginary parts at a single wavenumber