Exercicis Tardor 2019

ÍNDEX

E1 – Pi....2
E2 – Sèries....4
E3 – odeint....6
E4 – Matrix....11
E5 – Audio....13
E6 – RGB....16

E1 – Pi

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as spi
import csv

''' ------------------------- Exercici 1.1 ------------------------- '''

llista_error = np.array([])
llista_iteracions = np.array([])

pi = 0
k = 0
while True:
    pi += (1/(16**k))* ((4/(8*k+1)) - (2/(8*k+4)) - (1/(8*k+5)) - (1/(8*k+6)))
    error = np.pi-pi
    llista_error = np.append(llista_error,error)
    llista_iteracions = np.append(llista_iteracions,k)
    k += 1
    if error<1e-14:
        break
    else:
        continue

plt.figure (num = "Error vs Iteracions")
plt.plot(llista_iteracions, llista_error)
plt.yscale('log')
plt.grid()
plt.title('BPP formula: $\pi$ error estimation')
plt.xlabel("Iteration")
plt.ylabel("Error")

with open('BPP.csv', 'w', newline='', encoding='utf-8') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerows([llista_error,llista_iteracions])

np.save('BPP', [llista_error,llista_iteracions]) #guarda tipus .npy
plt.savefig("Error vs Iteracions")

''' ------------------------- Exercici 1.2 QUAD ---------------------- '''
# Manera 1:
def integrant1(x):
    return np.exp(-x**2)
I1 = spi.quad(integrant1, - np.inf, np.inf)

def integrant2(x):
    return 1/(1+x**2)
I2 = spi.quad(integrant2, - np.inf, np.inf)

# Manera 2:
I1 = spi.quad(lambda x: np.exp(-x**2), - np.inf, np.inf)
I2 = spi.quad(lambda x: 1/(1+x**2), - np.inf, np.inf)

pi1 = I1[0]**2
pi2 = I2[0]
print ('Quad 1= '+str(pi1))
print ('Quad 2= '+str(pi2))
```

```
''' ------------------------- Exercici 1.2 SIMPS --------------------- '''

x1=np.linspace(-20,20,10001)
x2=np.linspace(-100000,100000,5000001)
y1=integrant1(x1)
y2=integrant2(x2)
pi1_simps=spi.simps(y1,x1)**2
pi2_simps=spi.simps(y2,x2)

print('Simps 1= '+str(pi1_simps))
print('Simps 2= ' + str(pi2_simps))

''' ------------------------- Exercici 1.3 ------------------------- '''
from matplotlib.animation import FuncAnimation

L = 2
R = L/2
count = 0
iteracio = 0
err = 0
ll_error = np.array([])

ll_x = np.array([])
ll_y = np.array([])

while True:
    iteracio += 1
    x = (-1)**(np.random.randint(-1,1)) * R * np.random.random() #(-1)^parell (0) o senar (1) * L/2 * num random de [0.0, 1.0)
    y = (-1)**(np.random.randint(-1,1)) * R * np.random.random()
    ll_x = np.append(ll_x,x)
    ll_y = np.append(ll_y,y)
    d = (x**2 + y**2)**0.5

    if d<L/2: # Si el punt generat cau al centre
        count +=1
    err = abs(np.pi - 4*(count/iteracio)) # pi real - pi calculat segons AreaCercle/AreaRectangle = pi·(L/2)^2 / L^2 = pi / 4
    ll_error = np.append(ll_error,err)
    if err < 1e-3:
        break
    else:
        continue

fig = plt.figure()
plt.xlim(-R, R)
plt.ylim(-R, R)
graph, = plt.plot([], [], 'o')
plt.plot(np.linspace(-R,R,100), ((R**2) - (np.linspace(-R,R,100))**2)**0.5,'r')
plt.plot(np.linspace(-R,R,100), -((R**2) - (np.linspace(-R,R,100))**2)**0.5,'r')

def animate(i):
    graph.set_data(ll_x[:i+1], ll_y[:i+1])
    plt.title('$\pi$ = '+str(np.pi-ll_error[i+1]))
    return graph

ani = FuncAnimation(fig, animate, frames=iteracio, interval=100)
plt.show()
#from matplotlib.animation import PillowWriter
#writer = PillowWriter(fps=25)
#ani.save("GifPi.gif", writer=writer)
```

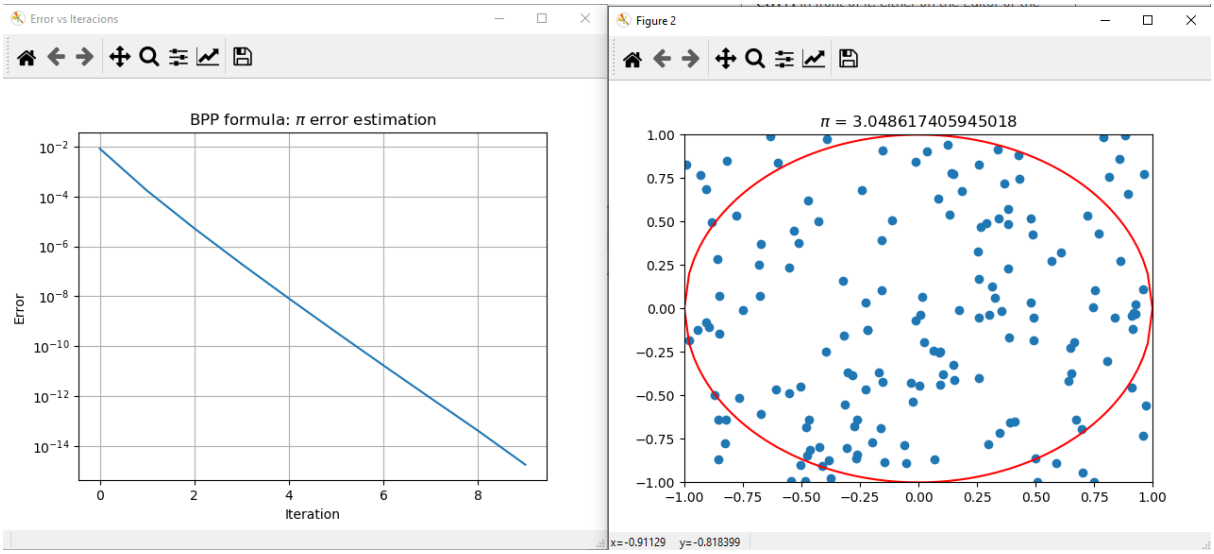
Out

Quad 1= 3.1415926535897927

Quad 2= 3.141592653589793

Simps 1= 3.1415926535897927

Simps 2= 3.1415726535897948

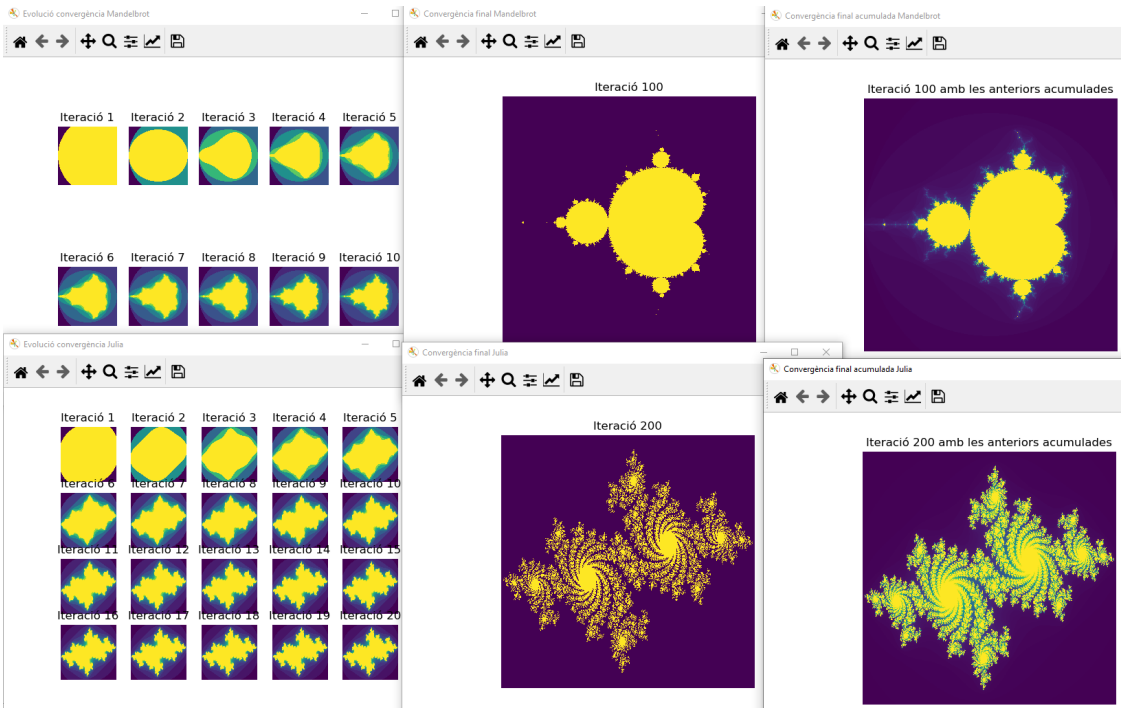


E2 – Sèries

```python
import numpy as np
import matplotlib.pyplot as plt
import imageio
plt.close('all')
# ----------------------------------------------------
## Conjunt de Mandelbrot
# ----------------------------------------------------
# Dimensió de la malla
Lx = 1000
Ly = 1000

# Eixos
Eixx = np.linspace(-2,1,Lx)
Eixy = np.linspace(-1.5,1.5,Ly)

# Variable complexa c
Ax,Ay = np.meshgrid(Eixx,Eixy)
c = Ax + 1j*Ay

# Inicialització de variables
Convergencia = np.array([])
Acumulada = np.zeros([])
z=0
images = []


for i in range(100):
    # Sèrie
    z = z**2 + c
    # Mask
    mask = np.abs(z) > 1e10
    z = (1-mask)*z + mask*1e10
    # Convergència
    Convergencia = np.abs(z) < 2
    # Matriu acumulada
    Acumulada = Acumulada + Convergencia
    # Representació de les convergències amb les iteracions
    if i<10:
        plt.figure('Evolució convergència Mandelbrot')
        plt.subplot(2,5,i+1)
        plt.title('Iteració ' + str(i+1))
        plt.axis('off')
        plt.imshow(Acumulada)
    # Guardar un GIF
    images.append(np.uint8(Convergencia*255))
imageio.mimsave('GifMandelbrot.gif', images, duration = 0.04)

#Representació final
plt.figure('Convergència final Mandelbrot')
plt.title('Iteració 100')
plt.axis('off')
plt.imshow(Convergencia)

plt.figure('Convergència final acumulada Mandelbrot')
plt.title('Iteració 100 amb les anteriors acumulades')
plt.axis('off')
plt.imshow(Acumulada)
```

```
# ------------------------------------------------------
## Conjunt de Julia
# ------------------------------------------------------

# Dimensió de la malla
lx = 1000
ly = 1000

# Eixos
eixx = np.linspace(-1.5,1.5,lx)
eixy = np.linspace(-1,1,ly)

# Variable complexa zj
ax,ay = np.meshgrid(eixx,eixy)
zj = ax + 1j*ay


# Inicialització de variables
convergencia = np.array([])
acumulada = np.zeros([])

for i in range(200):
    zj = zj**2 - 0.7 + 0.27015*1j
    mask = np.abs(zj) > 1e10
    zj = (1-mask)*zj + mask*1e10
    convergencia = np.abs(zj) < 2
    acumulada = acumulada + convergencia
    if i<20:
        plt.figure('Evolució convergència Julia')
        plt.subplot(4,5,i+1)
        plt.title('Iteració ' + str(i+1))
        plt.axis('off')
        plt.imshow(acumulada)

#Representació final
plt.figure('Convergència final Julia')
plt.title('Iteració 200')
plt.axis('off')
plt.imshow(convergencia)

plt.figure('Convergència final acumulada Julia')
plt.title('Iteració 200 amb les anteriors acumulades')
plt.axis('off')
plt.imshow(acumulada)
```

Out



E3 – odeint

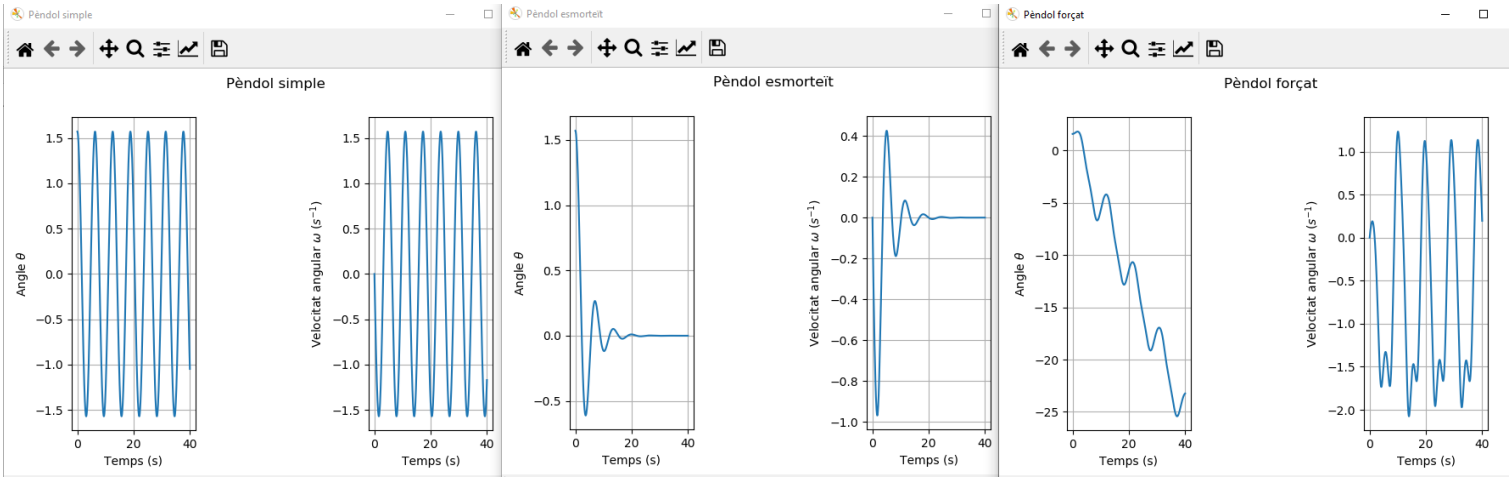
```python
# Imports necessaris en tots els exercicis. En el 3.4, a més, se n'afegeix un.
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

#-------------------------------------------------------------------------------
#       EXERCICI 3.1
#-------------------------------------------------------------------------------

# Definicó d'equacions diferencials
''' On:
    y[0]: omega
    y[1]: theta '''

def eqdif_simple(y,t):
    return -g*y[1]/L, y[0]
def eqdif_esmorteit(y,t):
    return -g*np.sin(y[1])/L - b*y[0]/(m*L**2), y[0]
def eqdif_forçat(y,t):
    return -g*np.sin(y[1])/L + (-b*y[0]+A*np.cos(OMEGA*t))/(m*L**2), y[0]

# Definició de constants
A = 1.35 # m
m = 1 # kg
g = 1 # m/s^2
L = 1 # m
b = 0.5 # (kg·m^2)/s
OMEGA = 0.666 # s^-1

# Definició de condicions inicials
y0_simple = [0, np.pi/2] #omega_inicial=0, theta_inicial=pi/2
y0_esmorteit = [0, np.pi/2]
y0_forçat = [0, np.pi/2]

# Definició de temps
t = np.linspace(0, 40, 800)

# Solució d'equacions diferencials
sol_simple = odeint(eqdif_simple, y0_simple, t)
sol_esmorteit = odeint(eqdif_esmorteit, y0_esmorteit, t)
sol_forçat = odeint(eqdif_forçat, y0_forçat, t)

#Representació dels gràfics
plt.figure('Pèndol simple')
plt.suptitle('Pèndol simple')
plt.subplot(1,3,1)
plt.plot(t, sol_simple[:,1])
plt.grid()
plt.xlabel('Temps (s)')
plt.ylabel('Angle $\\theta$')
plt.subplot(1,3,3)
plt.plot(t, sol_simple[:,0])
plt.grid()
plt.xlabel('Temps (s)')
plt.ylabel('Velocitat angular $\omega$ ($s^{-1}$)')
```

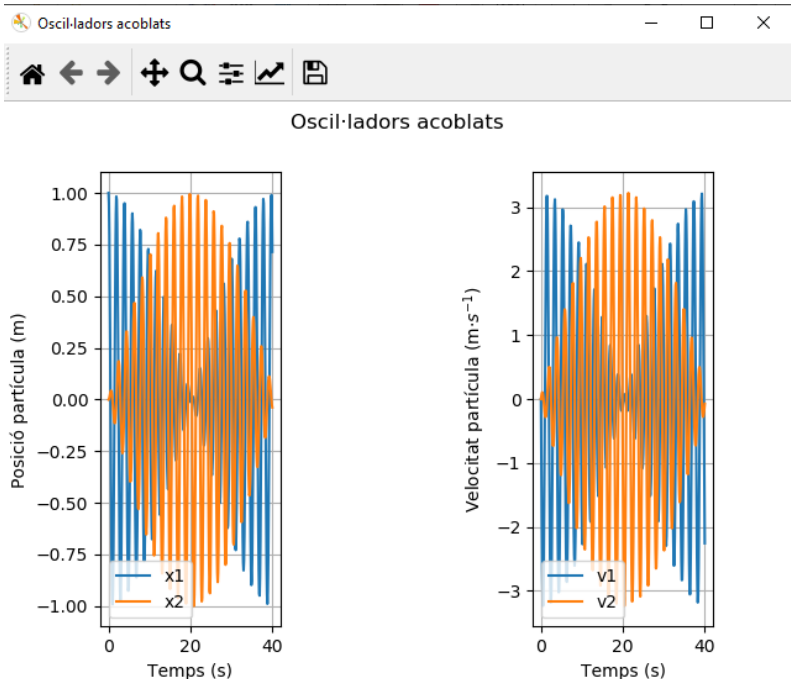
```
plt.figure('Pèndol esmorteït')
plt.suptitle('Pèndol esmorteït')
plt.subplot(1,3,1)
plt.plot(t, sol_esmorteit[:,1])
plt.grid()
plt.xlabel('Temps (s)')
plt.ylabel('Angle $\\theta$')
plt.subplot(1,3,3)
plt.plot(t, sol_esmorteit[:,0])
plt.grid()
plt.xlabel('Temps (s)')
plt.ylabel('Velocitat angular $\omega$ ($s^{-1}$)')

plt.figure('Pèndol forçat')
plt.suptitle('Pèndol forçat')
plt.subplot(1,3,1)
plt.plot(t, sol_forçat[:,1])
plt.grid()
plt.xlabel('Temps (s)')
plt.ylabel('Angle $\\theta$')
plt.subplot(1,3,3)
plt.plot(t, sol_forçat[:,0])
plt.grid()
plt.xlabel('Temps (s)')
plt.ylabel('Velocitat angular $\omega$ ($s^{-1}$)')

#-------------------------------------------------------------------------------
#       EXERCICI 3.2
#-------------------------------------------------------------------------------

# Definicó d'equacions diferencials
''' On:
    y[0]: v1
    y[1]: x1
    y[2]: v2
    y[3]: x2'''
def eqdif_acoblat(y,t):
    m = 1 # kg
    k_1 = 10 # kg·s^-2
    k_2 = 0.5 # kg·s^-2
    eq_1 = (0 - (k_1 + k_2)*y[1] + k_2*y[3])/m
    eq_2 = (0 - (k_1 + k_2)*y[3] + k_2*y[1])/m
    return (eq_1, y[0], eq_2, y[2])

# Definició de condicions inicials
y0 = [0, 1, 0, 0]

# Definició de temps
t = np.linspace(0, 40, 400)

# Solució d'equació diferencial
sol_acoblat = odeint(eqdif_acoblat, y0, t)

#Representació dels gràfics
plt.figure('Oscil·ladors acoblats')
plt.suptitle('Oscil·ladors acoblats')
plt.subplot(1,3,1)
plt.plot(t, sol_acoblat[:,1], label='x1')
plt.plot(t, sol_acoblat[:,3], label='x2')
plt.legend(loc='lower left')
plt.grid()
plt.xlabel('Temps (s)')
plt.ylabel('Posició partícula (m)')
plt.subplot(1,3,3)
plt.plot(t, sol_acoblat[:,0], label='v1')
plt.plot(t, sol_acoblat[:,2], label='v2')
plt.legend(loc='lower left')
plt.grid()
plt.xlabel('Temps (s)')
plt.ylabel('Velocitat partícula (m·$s^{-1}$)')
```

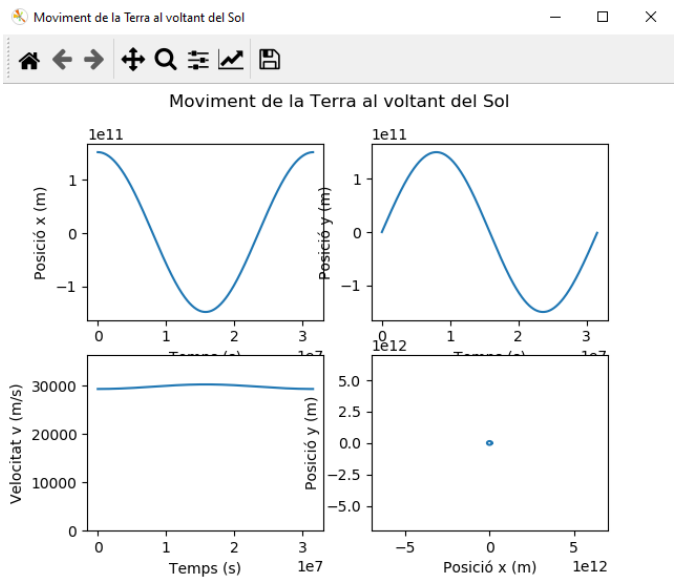
```
#-------------------------------------------------------------------------
#       EXERCICI 3.3
#-------------------------------------------------------------------------

# Definicó d'equacions diferencials
''' On:
    y[0]: v_x
    y[1]: x
    y[2]: v_y
    y[3]: y '''
def eqdif_gravitatori(y,t):
    G = 6.67e-11 # m^3 kg^-1 s^-2
    M = 1.9891e30 # kg
    eq_1 = -(G*M*y[1]) / (y[1]**2 + y[3]**2)**(1.5)
    eq_2 = -(G*M*y[3]) / (y[1]**2 + y[3]**2)**(1.5)
    return (eq_1, y[0], eq_2, y[2])

# Definició de condicions inicials
y0_T = [0, 152.1e9, 29300, 0] # Terra
y0_H = [0, 35.082*1.49598e11, 869, 0] # Halley

# Definició de temps (temps inicial, periode, nombre de dies en periode)
t_T = np.linspace(0, 3.15576e7, 365) # Terra
t_H = np.linspace(0, 3.15576e7*75, 365*75) # Halley

# Solució d'equació diferencial
sol_Terra = odeint(eqdif_gravitatori, y0_T, t_T)
sol_Halley = odeint(eqdif_gravitatori, y0_H, t_H)

#Representació dels gràfics
plt.figure('Moviment de la Terra al voltant del Sol')
plt.suptitle('Moviment de la Terra al voltant del Sol')
plt.subplot(2,2,1)
plt.plot(t_T,sol_Terra[:,1])
plt.xlabel('Temps (s)')
plt.ylabel('Posició x (m)')
plt.subplot(2,2,2)
plt.plot(t_T,sol_Terra[:,3])
plt.xlabel('Temps (s)')
plt.ylabel('Posició y (m)')
plt.subplot(2,2,3)
plt.plot(t_T,np.sqrt(sol_Terra[:,0]**2 + sol_Terra[:,2]**2))
plt.ylim(0, 1.20*max(np.sqrt(sol_Terra[:,0]**2 + sol_Terra[:,2]**2)))
plt.xlabel('Temps (s)')
plt.ylabel('Velocitat v (m/s)')
plt.subplot(2,2,4)
plt.plot(sol_Terra[:,1], sol_Terra[:,3])
plt.xlim(-7*1e12, 7*1e12) # Perquè estigui en la mateixa escala Terra i Halley.
plt.ylim(-7*1e12, 7*1e12)
plt.xlabel('Posició x (m)')
plt.ylabel('Posició y (m)')

plt.figure('Moviment del Cometa Halley al voltant del Sol')
plt.suptitle('Moviment del Cometa Halley al voltant del Sol')
plt.subplot(2,2,1)
plt.plot(t_H,sol_Halley[:,1])
plt.xlabel('Temps (s)')
plt.ylabel('Posició x (m)')
plt.subplot(2,2,2)
plt.plot(t_H,sol_Halley[:,3])
plt.xlabel('Temps (s)')
plt.ylabel('Posició y (m)')
plt.subplot(2,2,3)
plt.plot(t_H,np.sqrt(sol_Halley[:,0]**2 + sol_Halley[:,2]**2))
plt.xlabel('Temps (s)')
plt.ylabel('Velocitat v (m/s)')
plt.subplot(2,2,4)
plt.plot(sol_Halley[:,1], sol_Halley[:,3])
plt.xlim(-7*1e12, 7*1e12) # Perquè estigui en la mateixa escala Terra i Halley.
plt.ylim(-7*1e12, 7*1e12)
plt.xlabel('Posició x (m)')
plt.ylabel('Posició y (m)')
```

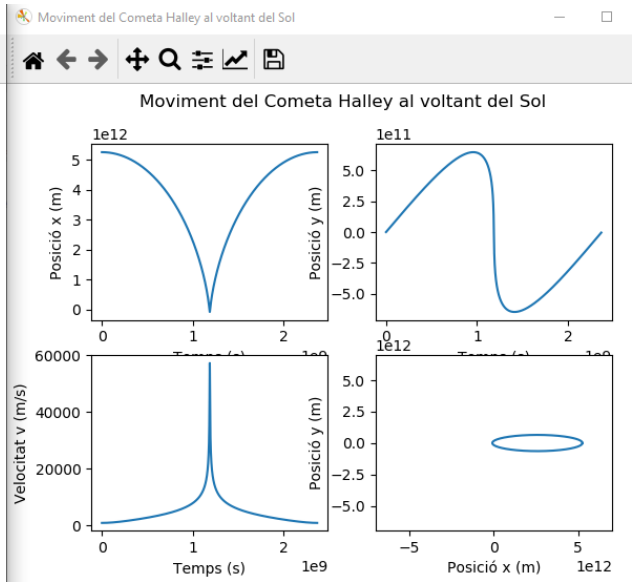
```
#       EXERCICI 3.4
#-------------------------------------------------------------------------------
from scipy import special

# Definicó de la funció d'Hermite
def func_hermite(n,x):
    return (((2**n)*np.math.factorial(n)*np.sqrt(np.pi))**(-1/2)) * np.exp(-x**2/2) * special.eval_hermite(n,x)

# Definicó d'equacions diferencials
''' On:
    y[0]: Psi prima
    y[1]: Psi '''
def eqdif_hermite_k0(y,x):
    k = 0
    B = 1
    return (B*x**2-(2*k+1))*y[1], y[0]
def eqdif_hermite_k2(y,x):
    k = 2
    B = 1
    return (B*x**2-(2*k+1))*y[1], y[0]
def eqdif_hermite_k4(y,x):
    k = 4
    B = 1
    return (B*x**2-(2*k+1))*y[1], y[0]

# Definició de x
x = np.linspace(-5, 5, 100)

# Definició de condicions inicials
y0_0 = [0, 0.751126]
y0_2 = [0, -0.531125]
y0_4 = [0, 0.459969]

# Solució d'equació diferencial
sol_0 = odeint(eqdif_hermite_k0, y0_0, x)
sol_2 = odeint(eqdif_hermite_k2, y0_2, x)
sol_4 = odeint(eqdif_hermite_k4, y0_4, x)

#Representació dels gràfics
plt.figure('Funcions d´Hermite')
plt.suptitle('Funció i eq. d´Hermite per n i k iguals a 0, 2 i 4')
plt.subplot(1,2,1)
plt.plot(x, sol_0[:,1]/np.max(abs(sol_0[:,1])), label='k=0') # solució normalitzada amb el màxim
plt.plot(x, sol_2[:,1]/np.max(abs(sol_2[:,1])), label='k=2')
plt.plot(x, sol_4[:,1]/np.max(abs(sol_4[:,1])), label='k=4')
plt.legend(loc='lower right')
plt.xlabel('Posició x (m)')
plt.ylabel('Equació diferencial d´Hermite')
plt.xlim(-5, 5)
plt.ylim(-1, 1)
plt.subplot(1,2,2)
plt.plot(x, func_hermite(0,x)/np.max(func_hermite(0,x)), label='n=0')
plt.plot(x, -func_hermite(2,x)/np.max(func_hermite(2,x)), label='n=2')
plt.plot(x, func_hermite(4,x)/np.max(func_hermite(4,x)), label='n=4')
plt.legend(loc='lower right')
plt.xlabel('Posició x (m)')
plt.ylabel('Funció d´Hermite')
plt.xlim(-5, 5)
plt.ylim(-1, 1)
```

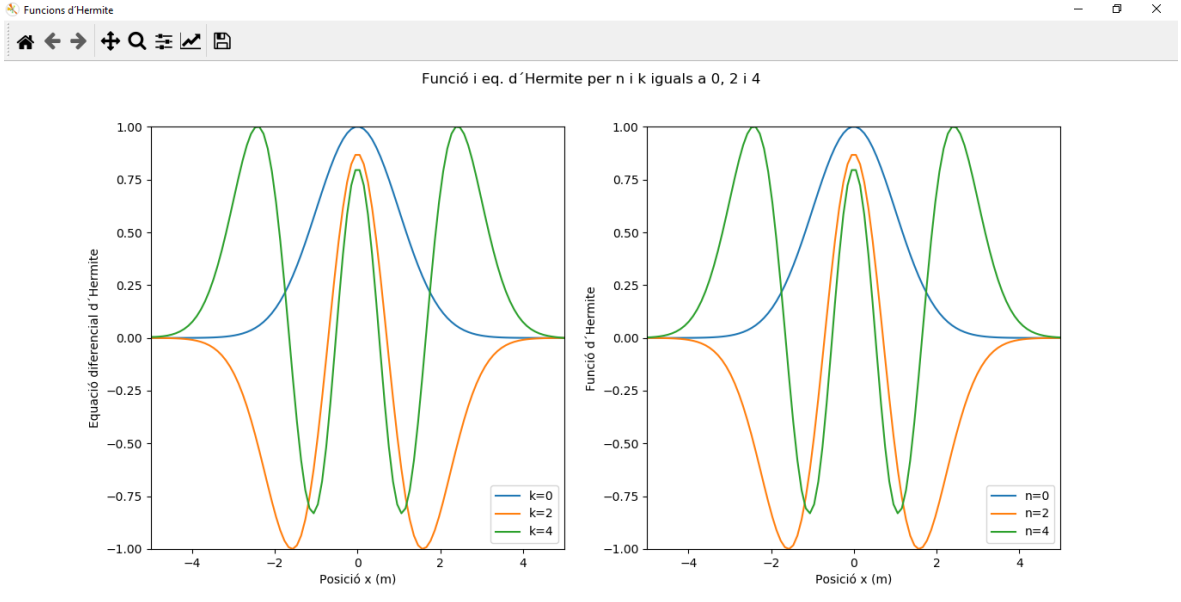
Out




Oscil·ladors acoblats
Oscil·ladors acoblats
Posició partícula (m)
Velocitat partícula (m·s⁻¹)
x1
x2
v1
v2
Temps (s)
Temps (s)


Moviment de la Terra al voltant del Sol
Moviment de la Terra al voltant del Sol
Posició x (m)
Posició y (m)
Velocitat v (m/s)
Posició y (m)
Temps (s)
Posició x (m)


Moviment del Cometa Halley al voltant del Sol
Moviment del Cometa Halley al voltant del Sol
Posició x (m)
Posició y (m)
Velocitat v (m/s)
Posició y (m)
Temps (s)
Posició x (m)


Funcions d'Hermite
Funció i eq. d´Hermite per n i k iguals a 0, 2 i 4
Equació diferencial d´Hermite
Funció d´Hermite
k=0
k=2
k=4
n=0
n=2
n=4
Posició x (m)
Posició x (m)

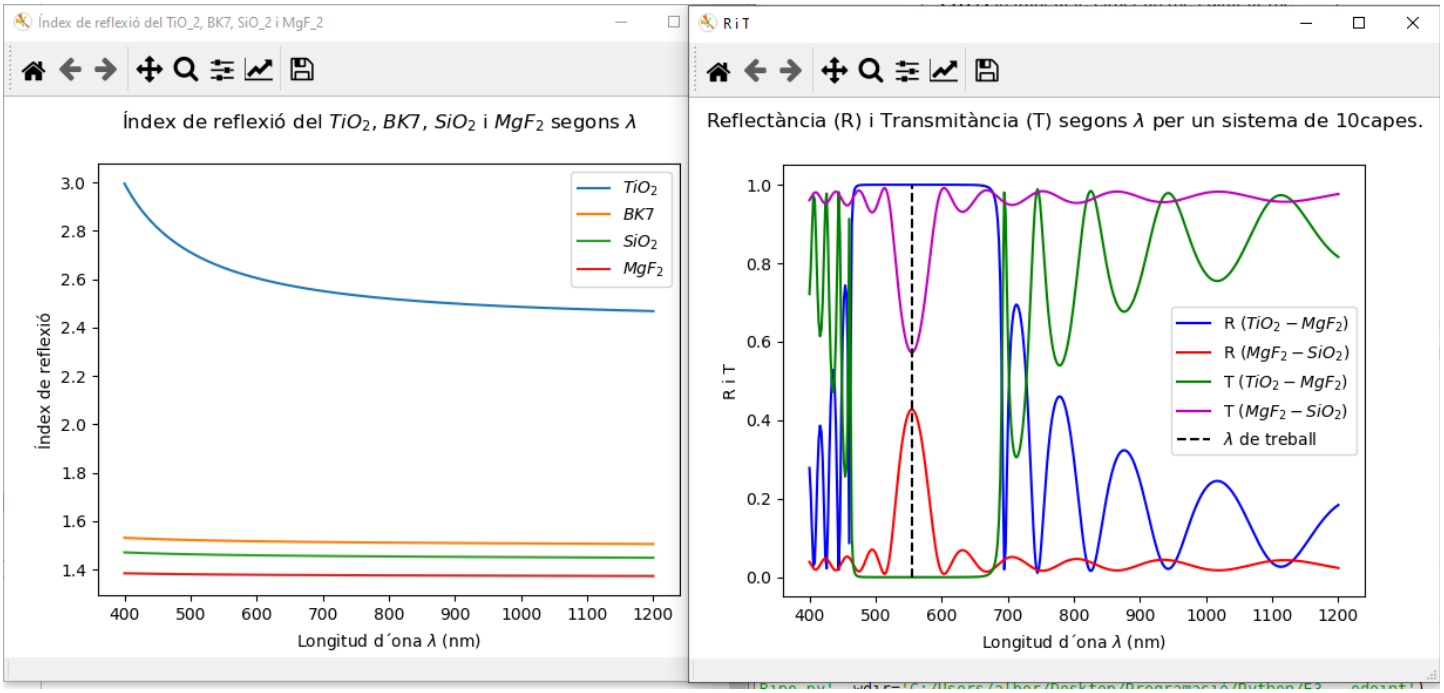
E4 – Matrix

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

# Dependència índex de reflexió amb la longitud d'ona de cada material. Extreta de: refractiveindex.info
def nTiO_2 (x):
    n=(5.913+0.2441/(x**2-0.0803))**.5
    return n
def nMgF_2 (x):
    n=(1+0.48755108/(1-(0.04338408/x)**2)+0.39875031/(1-(0.09461442/x)**2)+2.3120353/(1-(23.793604/x)**2))**.5
    return n
def nBK7 (x):
    n=(1+1.03961212/(1-0.00600069867/x**2)+0.231792344/(1-0.0200179144/x**2)+1.01046945/(1-103.560653/x**2))**.5
    return n
def nSiO_2 (x):
    n=(1+0.6961663/(1-(0.0684043/x)**2)+0.4079426/(1-(0.1162414/x)**2)+0.8974794/(1-(9.896161/x)**2))**.5
    return n
punts = 400
longitud_ona = np.linspace(0.4,1.2,punts) #micròmetres

plt.figure('Índex de reflexió del TiO_2, BK7, SiO_2 i MgF_2')
plt.suptitle('Índex de reflexió del $TiO_2$, $BK7$, $SiO_2$ i $MgF_2$ segons $\lambda$')
plt.plot(longitud_ona*1000, nTiO_2(longitud_ona), label='$TiO_2$')
plt.plot(longitud_ona*1000, nBK7(longitud_ona), label='$BK7$')
plt.plot(longitud_ona*1000, nSiO_2(longitud_ona), label='$SiO_2$')
plt.plot(longitud_ona*1000, nMgF_2(longitud_ona), label='$MgF_2$')
plt.xlabel('Longitud d´ona $\lambda$ (nm)')
plt.ylabel('Índex de reflexió')
plt.legend()
```

```python
longitud_ona_0 = 0.555 #Longitud d'ona de treball
Ncapes = 10 # Nombre de blocs de dues capes
 # expressió M apartat 4.1
def MTiO_2 (x):
    d = longitud_ona_0/(4*nTiO_2(longitud_ona_0))
    a11 = np.cos( (2*np.pi/x)*nTiO_2(x)*d )
    a22 = np.cos( (2*np.pi/x)*nTiO_2(x)*d )
    a12 = 1j/nTiO_2(x) * np.sin( (2*np.pi/x)*nTiO_2(x)*d )
    a21 = 1j*nTiO_2(x) * np.sin( (2*np.pi/x)*nTiO_2(x)*d )
    return np.array([[a11,a12],[a21,a22]])
def MMgF_2 (x):
    d = longitud_ona_0/(4*nMgF_2(longitud_ona_0))
    a11 = np.cos( (2*np.pi/x)*nMgF_2(x)*d )
    a22 = np.cos( (2*np.pi/x)*nMgF_2(x)*d )
    a12 = 1j/nMgF_2(x) * np.sin( (2*np.pi/x)*nMgF_2(x)*d )
    a21 = 1j*nMgF_2(x) * np.sin( (2*np.pi/x)*nMgF_2(x)*d )
    return np.array([[a11,a12],[a21,a22]])
def MBK7 (x):
    d = longitud_ona_0/(4*nBK7(longitud_ona_0))
    a11 = np.cos( (2*np.pi/x)*nBK7(x)*d )
    a22 = np.cos( (2*np.pi/x)*nBK7(x)*d )
    a12 = 1j/nBK7(x) * np.sin( (2*np.pi/x)*nBK7(x)*d )
    a21 = 1j*nBK7(x) * np.sin( (2*np.pi/x)*nBK7(x)*d )
    return np.array([[a11,a12],[a21,a22]])
def MSiO_2 (x):
    d = longitud_ona_0/(4*nSiO_2(longitud_ona_0))
    a11 = np.cos( (2*np.pi/x)*nSiO_2(x)*d )
    a22 = np.cos( (2*np.pi/x)*nSiO_2(x)*d )
    a12 = 1j/nSiO_2(x) * np.sin( (2*np.pi/x)*nSiO_2(x)*d )
    a21 = 1j*nSiO_2(x) * np.sin( (2*np.pi/x)*nSiO_2(x)*d )
    return np.array([[a11,a12],[a21,a22]])
```

```
70 R1 = np.zeros([punts], dtype =np.float64)
71 R2 = np.zeros([punts], dtype =np.float64)
72 T1 = np.zeros([punts], dtype =np.float64)
73 T2 = np.zeros([punts], dtype =np.float64)
74 no_ns = [[1],[nBK7(longitud_ona_0)]] #expressió 4.2 part dreta
75 A1 = np.zeros([2,1], dtype =np.float64) #expressió 4.2 part esquerra cas 1 (TiO_2)
76 A2 = np.zeros([2,1], dtype =np.float64) #expressió 4.2 part esquerra cas 2 (SiO_2)
77 i=0
78 for ona in longitud_ona:
79     MT1 = np.linalg.matrix_power(np.dot(MTiO_2(ona),MMgF_2(ona)),Ncapes) #expressió 4.1
80     MT2 = np.linalg.matrix_power(np.dot(MSiO_2(ona),MMgF_2(ona)),Ncapes) #expressió 4.1
81     A1 = np.dot(MT1,no_ns) #expressió 4.2
82     A2 = np.dot(MT2,no_ns) #expressió 4.2
83     R1[i] = (abs((A1[0]-A1[1])/(A1[0]+A1[1])))**2 #expressió 4.3
84     R2[i] = (abs((A2[0]-A2[1])/(A2[0]+A2[1])))**2 #expressió 4.3
85     T1[i] = 1-R1[i] #expressió 4.4
86     T2[i] = 1-R2[i] #expressió 4.4
87     i += 1
```

Out



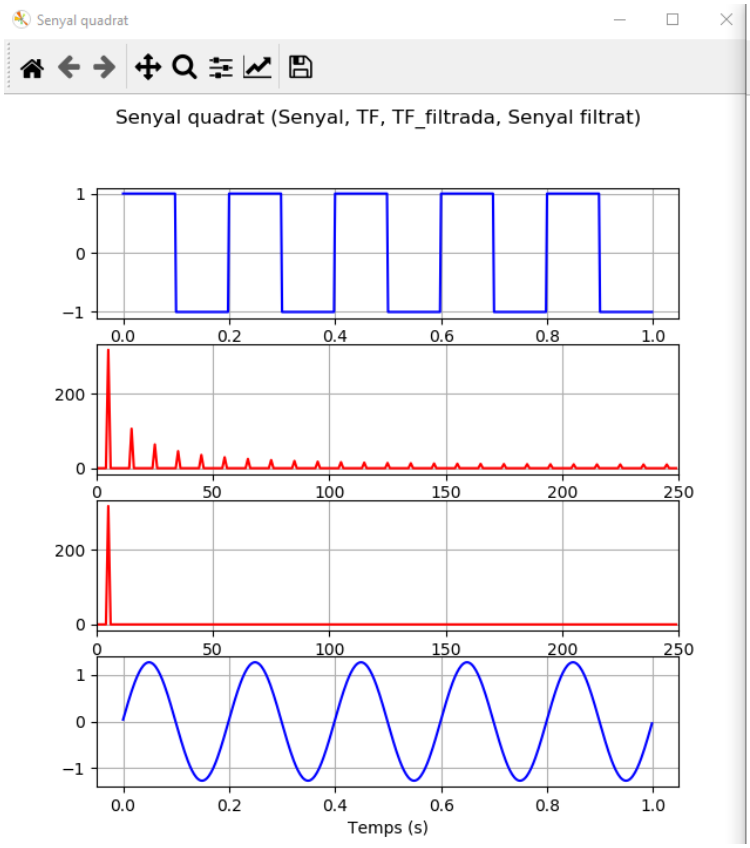
E5 – Audio

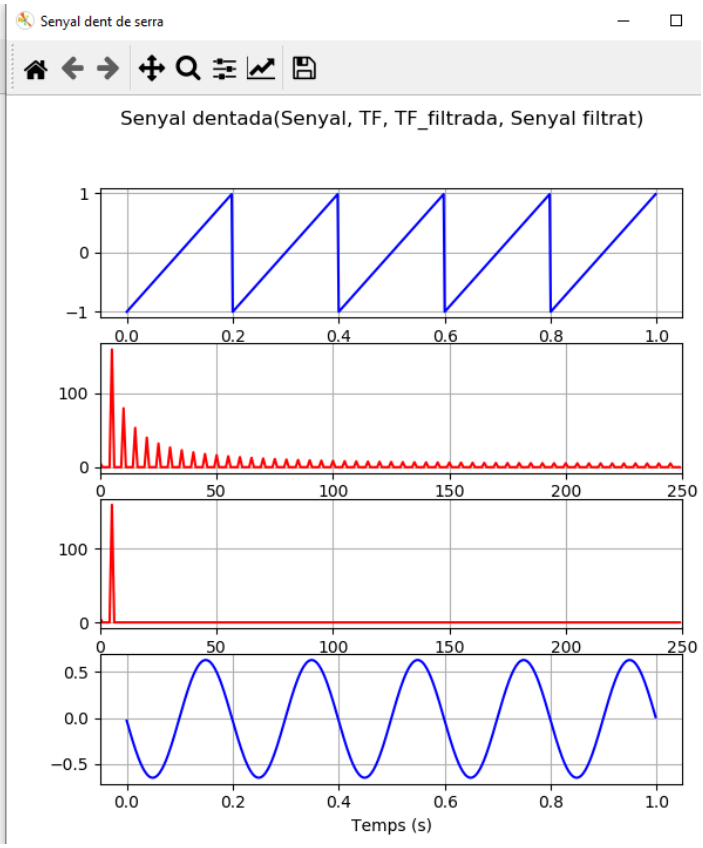
```python
import numpy as np
from numpy.fft import fft, fftshift, ifft
import matplotlib.pyplot as plt
from scipy.io.wavfile import read, write
from scipy.signal import square, sawtooth

#--------------------------------------------------------------------------
#       EXERCICI 5.1
#--------------------------------------------------------------------------

L = 1 # Duració del senyal (segons)
N = 500 # Nombre de mostres/segon
w = 5 # Freqüència del senyal (Hz)
fc = N/(2*L) # Freqüència de Nyquist (Hz).(Eq. 5.2)

t = np.linspace(0, L, N*L, endpoint=False) # Línia del temps dels senyals
f = np.linspace(-fc, fc, N*L, endpoint=False) # Línia de freqüències dels senyals

"""Transformades de Fourier dels senyals"""
TF_quadrada = fftshift(fft(square(2 * np.pi * w * t)))
TF_dentada = fftshift(fft(sawtooth(2 * np.pi * w * t)))

"""Filtre Passabaixos"""
passabaixos = np.zeros(N*L)
passabaixos[(N*L//2)-6:(N*L//2)+6] = 1 # Creació del filtre
# El filtre passabaixos s'ha realitzat per quedar-se amb les freqüències més baixes.
# S'ha triat el valor 6 Hz donat els pics al en la TF al voltant de 5 Hz
# (Descomentar línies 53 i 75 per observar-ho)
TF_quadrada_filtrada = TF_quadrada * passabaixos # Aplicació del filtre
TF_dentada_filtrada = TF_dentada * passabaixos # Aplicació del filtre

"""Antitranformem"""
iTF_quadrada_filtrada = ifft(fftshift(TF_quadrada_filtrada))
iTF_dentada_filtrada = ifft(fftshift(TF_dentada_filtrada))

plt.figure("Senyal quadrat")
plt.suptitle('Senyal quadrat (Senyal, TF, TF_filtrada, Senyal filtrat)')
plt.subplot(4,1,1)
plt.plot(t,square(2 * np.pi * w * t),'b')
plt.grid(True)
plt.xlabel('Temps (s)')
plt.subplot(4,1,2)
plt.plot(f,abs(TF_quadrada),'r')
plt.xlim(0,fc) # Només cal representar valors positius
plt.grid(True)
#plt.xlim(-8,8)
plt.xlabel('Freqüència (Hz)')
plt.subplot(4,1,3)
plt.plot(f,abs(TF_quadrada_filtrada),'r')
plt.xlim(0,fc) # Només cal representar valors positius
plt.grid(True)
plt.xlabel('Freqüència (Hz)')
plt.subplot(4,1,4)
plt.plot(t,iTF_quadrada_filtrada,'b')
plt.grid(True)
plt.xlabel('Temps (s)')
```

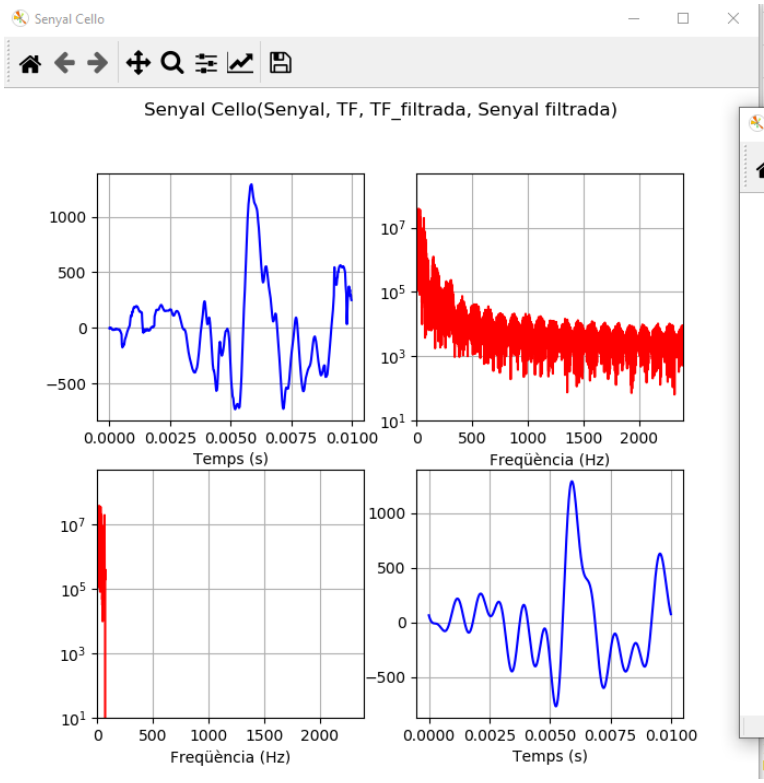
```
plt.figure("Senyal dent de serra")
plt.suptitle('Senyal dentada(Senyal, TF, TF_filtrada, Senyal filtrat)')
plt.subplot(4,1,1)
plt.plot(t,sawtooth(2 * np.pi * w * t),'b')
plt.grid(True)
plt.xlabel('Temps (s)')
plt.subplot(4,1,2)
plt.plot(f,abs(TF_dentada),'r')
plt.xlim(0,fc) # Només cal representar valors positius
plt.grid(True)
#plt.xlim(-8,8)
plt.xlabel('Freqüència (Hz)')
plt.subplot(4,1,3)
plt.plot(f,abs(TF_dentada_filtrada),'r')
plt.xlim(0,fc) # Només cal representar valors positius
plt.grid(True)
plt.xlabel('Freqüència (Hz)')
plt.subplot(4,1,4)
plt.plot(t,iTF_dentada_filtrada,'b')
plt.grid(True)
plt.xlabel('Temps (s)')

#-------------------------------------------------------------------------------
#       EXERCICI 5.2
#-------------------------------------------------------------------------------

L = 5 # Duració del senyal (segons)
N = 44100 # Nombre de mostres/segon
w = 440 # Freqüència del senyal (Hz)

t =  np.linspace(0, L, N*L, endpoint=False)
sinusoidal = np.sin(2 * np.pi * w * t)

write('diapaso.wav', N, np.float32(sinusoidal))

#-------------------------------------------------------------------------------
#       EXERCICI 5.3 i 5.4
#-------------------------------------------------------------------------------

lectura = read('cello.wav')

N = lectura[0] # Nombre de mostres/segon
NTotal = len(lectura[1]) # Nombre de mostres totals
L = NTotal/N # Longitud del senyal (en segons)
T = L/N # Durada de cada mostra

"""S'agafa només el senyal a partir de quan comença a ser diferent de zero."""
cello_data = lectura[1]
index = 0
for data in cello_data:
    if data!=0:
        break
    index = index+1
Lectura_1 = cello_data[index:index+N] #Senyal de 1 segon, a partir de quan el senyal és diferent a zero.
Lectura_001= Lectura_1[0:800] # Senyal de 800 punts. Aquesta quantitat de punts és representable per pantalla.
"""Temps i freqüències en el primer segon i en els primers 800 punts (0.01 segons aprox)."""
fc = 1/(2*T) # Freqüència de Nyquist (Hz).(Eq. 5.2)

t_1 = np.linspace(0,1,N, endpoint=False)
t_001 = np.linspace(0,0.01,800, endpoint=False)
f_1 = np.linspace(-fc,fc,N, endpoint=False)
f_001 = np.linspace(-fc,fc,800, endpoint=False)

"""Transformades de Fourier"""
TF_1 = fftshift(fft(Lectura_1))
TF_001 = fftshift(fft(Lectura_001))

"""Filtre"""
f_filtratge = 70 # Hz
# No es deixa passar freqüències majors a la freq. de filtratge
TF_1_Filtrada = TF_1 * (np.abs(f_1)<=f_filtratge)
TF_001_Filtrada = TF_001 * (np.abs(f_001)<=f_filtratge)
```

```
"""Antitransformada"""
iTF_1_Filtrada = np.real(ifft(fftshift(TF_1_Filtrada)))
iTF_001_Filtrada = np.real(ifft(fftshift(TF_001_Filtrada)))

plt.figure("Senyal Cello")
plt.suptitle('Senyal Cello(Senyal, TF, TF_filtrada, Senyal filtrada)')
plt.subplot(2,2,1)
plt.plot(t_001, Lectura_001 ,'b')
plt.grid(True)
plt.xlabel('Temps (s)')
plt.subplot(2,2,2)
plt.plot(f_1,abs(TF_1),'r')
plt.xlim(0,fc) # Només cal representar valors positius
plt.ylim(10,5e8)
plt.grid(True)
plt.yscale('log')
plt.xlabel('Freqüència (Hz)')
plt.subplot(2,2,3)
plt.plot(f_1,abs(TF_1_Filtrada),'r')
plt.xlim(0,fc) # Només cal representar valors positius
plt.ylim(10,5e8)
plt.grid(True)
plt.yscale('log')
plt.xlabel('Freqüència (Hz)')
plt.subplot(2,2,4)
plt.plot(t_001,iTF_001_Filtrada,'b')
plt.grid(True)
plt.xlabel('Temps (s)')

plt.figure('Comparació cello original i filtrat')
plt.plot(t_001,Lectura_001,'b',label = 'senyal original')
plt.plot(t_001,iTF_001_Filtrada,'r',label = 'senyal filtrat')
plt.axis([0,0.01,-1500,1500])
plt.xlabel('Temps(s)')
plt.legend()
plt.grid(True)

write('cello_filtrat.wav',N,np.float32(iTF_1_Filtrada))
```

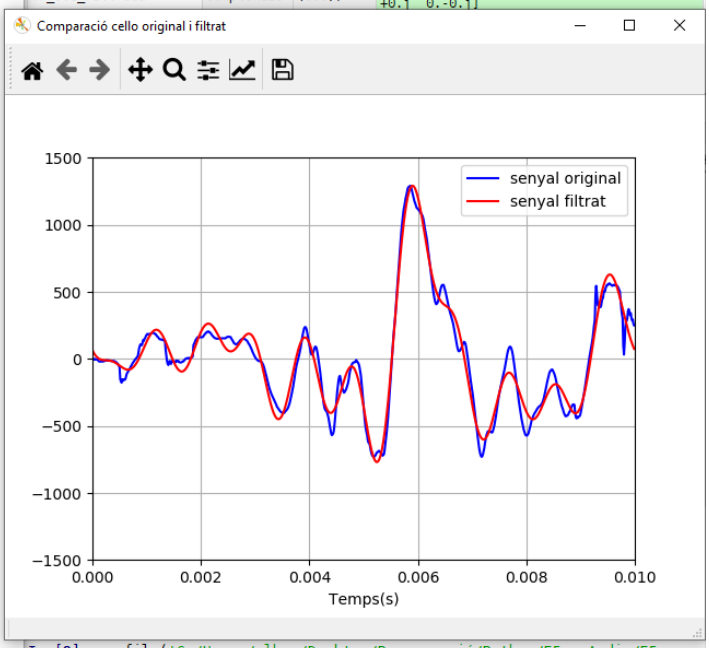
Out







NTotal	int	1	405891
T	float	1	0.00020870470637234485
TF_001_Filtrada	complex128	(800,)	[-0.+0.j -0.+0.j 0.+0.j ... 0.+0.j 0.+0.j 0.-0.j]




```
In [8]: runfile('C:/Users/alber/Desktop/Programació/Python/E5 - Audio/E5.
Ripo_3.py', wdir='C:/Users/alber/Desktop/Programació/Python/E5 - Audio')
```

E6 – RGB

```
 8 import numpy as np
 9 import matplotlib.pyplot as plt
10 import scipy.integrate as spi
11
12 # Lectura de les dades
13 arxiu_problema = np.loadtxt('arxiu_problema.txt')
14 l = arxiu_problema[:,0]
15 x_l = arxiu_problema[:,1]
16 y_l = arxiu_problema[:,2]
17 z_l = arxiu_problema[:,3]
18 E_l = arxiu_problema[:,4]
19
20 # Representació x_l (l), y_l (l), z_l (l) i E_l (l)
21 plt.figure("x(lambda), y(lambda), z(lambda) i E(lambda)")
22 plt.subplot(1,2,1)
23 plt.title('x($\lambda$), y($\lambda$), z($\lambda$)')
24 plt.plot(l, x_l, 'r', label='x($\lambda$)')
25 plt.plot(l, y_l, 'g', label='y($\lambda$)')
26 plt.plot(l, z_l, 'b', label='z($\lambda$)')
27 plt.grid()
28 plt.xlim(380, 650)
29 plt.ylim(0, 2)
30 plt.xlabel('$\lambda$ (nm)')
31 plt.ylabel('x, y i z (u.a.)')
32 plt.legend(loc='best')
33 plt.subplot(1,2,2)
34 plt.title('E($\lambda$)')
35 plt.plot(l, E_l, 'k', label='E($\lambda$)')
36 plt.grid()
37 plt.xlim(380, 650)
38 plt.ylim(0, 2)
39 plt.xlabel('$\lambda$ (nm)')
40 plt.ylabel('E (u.a.)')
41 plt.legend(loc='best')
```

```
# Calcul de les integrals
x = spi.simps(E_l * x_l, (l)) # Eq. 6.1
y = spi.simps(E_l * y_l, (l)) # Eq. 6.2
z = spi.simps(E_l * z_l, (l)) # Eq. 6.3

# Normalitzacio
X = x/(x+y+z) # Eq. 6.5
Y = y/(x+y+z) # Eq. 6.6
Z = z/(x+y+z) # Eq. 6.7

# Matriu del canvi lineal
M_canvilineal = np.array([[3.240479,-1.537150,-0.498535],
                [-0.969256,1.875992,0.041556],[0.055648,-0.204043,1.057311]])

# Calcul de RGB amb el canvi lineal
[R,G,B] = np.dot(255. , np.dot(M_canvilineal,np.array([X,Y,Z])) )  # Eq. 6.9
[R,G,B] = np.uint8([R,G,B])

# Color de la pantalla
c1 = np.ones([250,250,3], dtype='uint8')
c1[:,:,0] = R
c1[:,:,1] = G
c1[:,:,2] = B

print ('Resultats per a E(lambda) de arxiu_problema: \n', 'x = ',x,'y = ',y,
  'z = ',z, 'X = ',X,'Y = ',Y,'Z = ',Z,'R = ', R,'\t\t\tG = ',G,'\t\t\tB = ',B)

plt.figure("Color per a E(lambda) de arxiu_problema")
plt.suptitle('Color per a E($\lambda$) de arxiu_problema')
plt.imshow(c1)

# Es repeteixen els punts anteriors per E_l (l) = 1:
x = spi.simps(1 * x_l, (l)) # Eq. 6.1
y = spi.simps(1 * y_l, (l)) # Eq. 6.2
z = spi.simps(1 * z_l, (l)) # Eq. 6.3
X = x/(x+y+z) # Eq. 6.5
Y = y/(x+y+z) # Eq. 6.6
Z = z/(x+y+z) # Eq. 6.7
[R,G,B] = np.dot(255. , np.dot(M_canvilineal,np.array([X,Y,Z])) ) # Canvi lineal, Eq. 6.9
[R,G,B] = np.uint8([R,G,B])
c1 = np.ones([250,250,3], dtype='uint8')
c1[:,:,0] = R
c1[:,:,1] = G
c1[:,:,2] = B

print ('Resultats per a E(lambda) =1: \n', 'x = ',x,'y = ',y,'z = ',z, 'X = ',X,
       'Y = ',Y,'Z = ',Z,'R = ', R,'\t\t\tG = ',G,'\t\t\tB = ',B)

plt.figure("Color per a E(lambda)=1")
plt.suptitle('Color per a E($\lambda$)=1')
plt.imshow(c1)
```

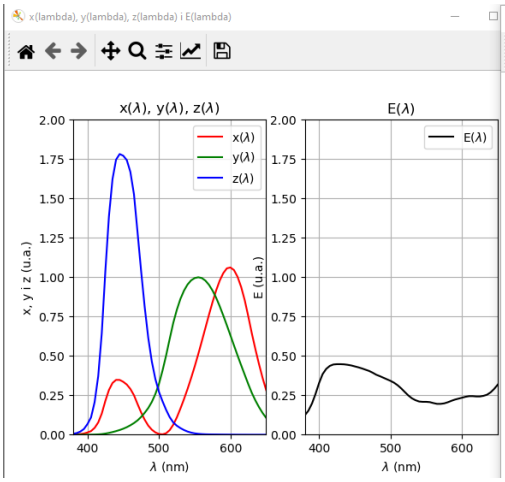
Out

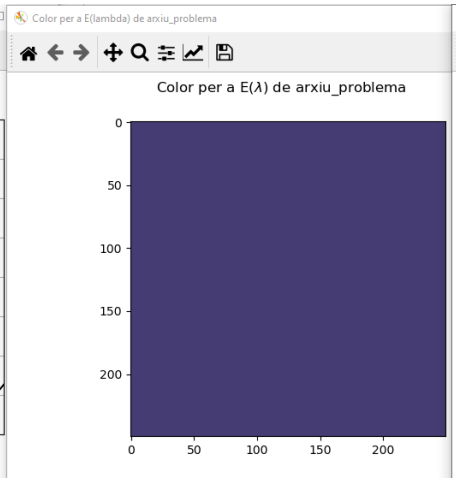
Resultats per a E(lambda) de arxiu_problema:

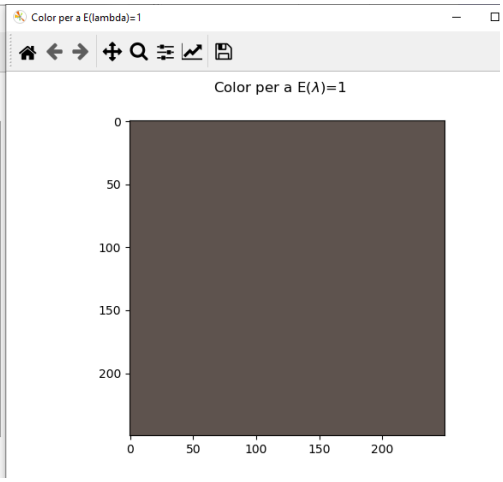
x = 26.898798456133335 y = 25.235278297333334 z = 44.58302728810833 X = 0.2781183196363134 Y = 0.26091846470594954 Z = 0.4609632156577371 R = 68 G = 60 B = 114

Resultats per a E(lambda) =1:

x = 102.11869606666667 y = 105.10399266666668 z = 106.87651768833334 X = 0.3251160588084264 Y = 0.33462036999090156 Z = 0.3402635712006719 R = 94 G = 83 B = 78


x(λ), y(λ), z(λ)
E(λ)
x(λ)
y(λ)
z(λ)
E(λ)
x, y i z (u.a.)
E (u.a.)
λ (nm)
λ (nm)


Color per a E(λ) de arxiu_problema


Color per a E(λ)=1