

Procesamiento de señales, fundamentos

.....

Maestría en sistemas embebidos MSE2020

Universidad de Buenos Aires

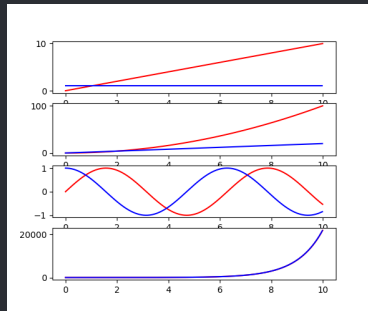
Clase 2 - Euler | Fourier

Ing. Pablo Slavkin



2.7182818284590450907955982984276488423347473144

- $f(t) = t$
- $f(t) = t^2$
- $f(t) = \sin(t)$
- $f'(t) = 1$
- $f'(t) = 2 * t$
- $f'(t) = \cos(t)$



La derivada es igual a la funcion

$$f(t) = e^t \implies f'(t) = e^t$$
$$f(t) = e^{kt} \implies f'(t) = ke^{kt}$$

$$e^{j2\pi t}$$

Euler

Pero que pasa con e^{jt} ?

La derivada es igual a la funcion

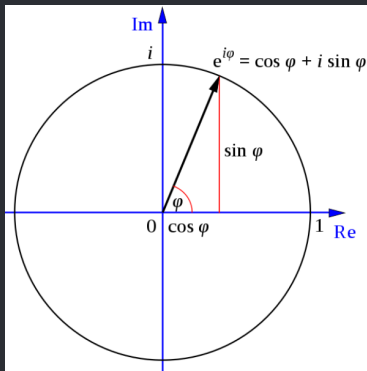
$$f(t) = e^{jt} \implies f'(t) = je^{jt}$$

$$e^{jt} = \cos(t) + j \sin(t)$$

$$e^{j\pi} = -1$$

$$e^{j\frac{\pi}{2}} = i$$

$$e^{j\frac{3\pi}{2}} = -i$$





```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

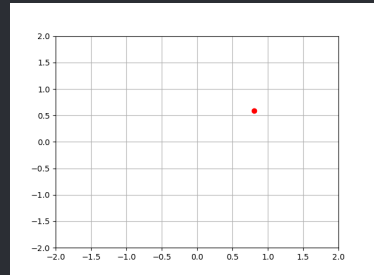
fig = plt.figure()
fs = 10
N = 10

circleAxe = fig.add_subplot(1,1,1)
circleLn, = plt.plot([],[],'ro')
circleAxe.grid(True)
circleAxe.set_xlim(-2,2)
circleAxe.set_ylim(-2,2)
circleFrec = 1

circle = lambda c,f,n: c*np.exp(-1j*2*np.pi*f*n/fs)

def update(n):
    circleLn.set_data(np.real(circle(1,circleFrec,n)),
                     np.imag(circle(1,circleFrec,n)))
    return circleLn,

ani=FuncAnimation(fig,update,N,interval=100 ,blit=False,repeat=True)
plt.show()
```





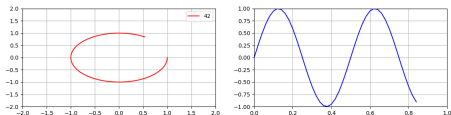
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

#-----
fig      = plt.figure()
fs       = 50
N        = 50
#-----
circleAxe = fig.add_subplot(2,2,1)
circleLn, = plt.plot([],[],'r-')
circleAxe.grid(True)
circleAxe.set_xlim(-2,2)
circleAxe.set_ylim(-2,2)
circleFreq = 1
circleData=[]
circle = lambda c,f,n: c*np.exp(-1j*2*np.pi*f*n*1/fs)
#-----
signalAxe = fig.add_subplot(2,2,2)
signalLn, = plt.plot([],[],'b-')
signalAxe.grid(True)
signalAxe.set_xlim(0,N/fs)
signalAxe.set_ylim(-1,1)
signalFreq = 2
signalData=[]
signal = lambda f,n: np.cos(2*np.pi*f*n*1/fs)
#-----
tData=[]

def update(n):
    global circleData,signalData,tData
    circleData.append(circle(1,circleFreq,n))
    circleLn.set_data(np.real(circleData),
                      np.imag(circleData))
    signalData.append(signal(signalFreq,n))
    tData.append(n/fs)
    signalLn.set_data(tData,signalData)

    if n==N-1:
        circleData=[]
        signalData=[]
        tData=[]
        circleLn.set_label(n)
        circleAxe.legend()
    return circleLn,circleAxe,signalLn

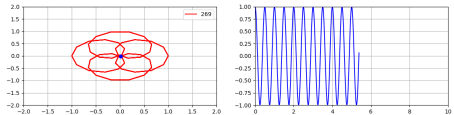
ani=FuncAnimation(fig,update,N,interval=10 ,blit=False,repeat=True)
plt.show()
Ing. Pablo Slavkin
```





```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

#-----
fig      = plt.figure()
fs       = 50
N        = 500
#-----
circleAx = fig.add_subplot(2,2,1)
circleLn, promLn = plt.plot([],[], 'r-', [], [], 'bo')
circleAx.grid(True)
circleAx.set_xlim(-2,2)
circleAx.set_ylim(-2,2)
circleFrec = 3
circleData = []
prom = 0
circle     = lambda c, f, n: c*np.exp(-1j)*2*np.pi*f*n*1/fs)
#-----
signalAx = fig.add_subplot(2,2,2)
signalLn, = plt.plot([],[], 'b-',)
signalAx.grid(True)
signalAx.set_xlim(0, N/fs)
signalAx.set_ylim(-1,1)
signalFrec = 2
signalData = []
signal     = lambda f, n: np.cos(2*np.pi*f*n*1/fs)
#-----
tData = []
def update(n):
    global circleData, signalData, tData, promData
    circleData.append(circle(1, circleFrec, n)*signal(signalFrec, n))
    prom = np.average(circleData)
    promLn.set_data(np.real(prom),
                    np.imag(prom))
    circleLn.set_data(np.real(circleData),
                     np.imag(circleData))
    signalData.append(signal(signalFrec, n))
    tData.append(n/fs)
    signalLn.set_data(tData, signalData)
    if n==N-1:
        circleData = []
        signalData = []
        tData      = []
        prom       = 0
        circleLn.set_label(n)
        circleAx.legend()
    return circleLn, circleAx, signalLn, promLn
ani=FuncAnimation(fig, update, N, interval=10 ,blit=False, repeat=True)
plt.show()
```



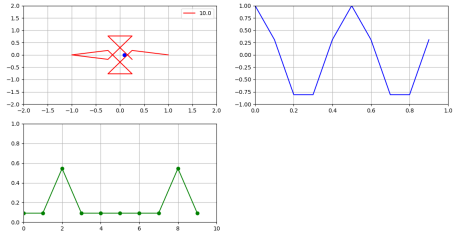


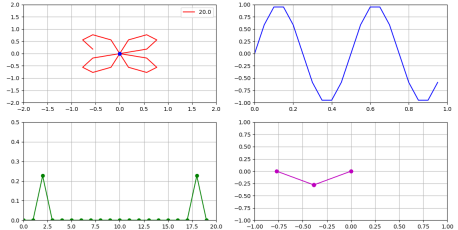
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

#-----
fig = plt.figure()
fs = 10
N = 100
#-----
circlekw = fig.add_subplot(2,2,1)
circlea, proma = plt.plot([],[], 'r-', [], [], 'b-')
circlekw.grid(True)
circlekw.set_xlim(-2,2)
circlekw.set_ylim(-2,2)
circlefrec = 0
circledata = []
prom = []
fractier = 0
circle = lambda x,f,A: c*np.exp(1j*2*np.pi*f*x+1j/fx)
#-----
signalkw = fig.add_subplot(2,2,2)
signalkw = plt.plot([],[], 'b-')
signalkw.grid(True)
signalkw.set_xlim(0,fs)
signalkw.set_ylim(-1,1)
signalfrec = 2
signaldata[]
signal = lambda f,s: sp.cos(2*np.pi*f*f*x+1j/fx)
#-----
fourierkw = fig.add_subplot(2,2,3)
fourierkw = plt.plot([],[], 'g-')
fourierkw.grid(True)
fourierkw.set_xlim(0,fs)
fourierkw.set_ylim(0,1)
fourierdata[]
#-----
thata[]
thataw[]

def update(n):
    global circledata, signaldata, theta, promdata, fractier, circlefrec, fourierdata, thata
    circledata.append(circlea)
    circlefrec, circlea = signal(signalfrec, n)
    proma = average(circledata)
    proma = np.imag(proma)
    circlekw.set_data(sp.real(circledata), sp.imag(circledata))
    signaldata.append(signal(signalfrec, n))
    thata.append(n/fx)
    signalkw.set_data(thata, signaldata)
    if n==N-1:
        circledata = []
        signaldata = []
        thata = []
        fourierdata.append(np.real(proma))
        thata.append(circlefrec)
        fourierkw.set_data(thata, fourierdata)
        prom = 0
        fractier = 0
        if fractier == N:
            repeat = False
        circlefrec = fractier+fs/N
        circlekw.set_label(circlefrec)
        circlekw.legend()
    return circlea, circlekw, signala, proma, fractier

anim = FuncAnimation(fig, update, N, interval=100, _init=False, repeat=True)
plt.show()
```



[illegible]



```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

#-----
fig = plt.figure()
Nt = 200
N = 200
#-----
circles = fig.add_subplot(2,2,1)
circles.grid(True)
circles.set_xlim(-2,2)
circles.set_ylim(-2,2)
circles.freq = 0
circles.data = []
prom = 0
fractier = 0
circle = lambda c,f,s: c*np.exp(-1/2*np.pi**2*f**2/s**2)
circles = lambda c,f,s: c*np.exp(-1/2*np.pi**2*f**2/s**2)
#-----
signals = fig.add_subplot(2,2,2)
signals.grid(True)
signals.set_xlim(-1,1)
signals.set_ylim(-1,1)
signals.freq = 2
signals.data = []
signal = lambda f,s: 0.5*np.cos(2*np.pi*f**2/s**2)+0.5*np.sin(2*np.pi*f**2/s**2)
#-----
cmap=plt.cm.hot("cmap_vir")
M=100000
signal= lambda f,s: cmap[s]
#-----
fourierdata = fig.add_subplot(2,2,3)
fourierdata.grid(True)
fourierdata.set_xlim(0,5)
fourierdata.set_ylim(0,5)
fourierdata.data = []
#-----
inversedata = fig.add_subplot(2,2,4)
inversedata.grid(True)
inversedata.set_xlim(-1,1)
inversedata.set_ylim(-1,1)
inversedata.data = []
#-----
pseudata = []
#-----
thetas = []

#-----
def update(f):
    global f,data,vectorData,pseudata,fourierData
    # if not isinstance(f,int):
    #     return
    vectorData.append(vectorData[-1]+circles(f,data,f,f))
    pseudata.append(vectorData[-1])
    pseudata.set_data(real(pseudata),np.imag(pseudata))
    return pseudata,vectorData

def update(f):
    global circles,signals,data,pseudata,fractier,circlesfreq
    # fourierData,data
    circlesData.append(circles(f,circlesfreq,s)*signal(signalFreq,s))
    prom=np.average(circlesData)
    pseudata.set_data(real(prom),np.imag(prom))
    circlesData.set_data(real(circlesData),np.imag(circlesData))
    signalsData.append(signal(signalFreq,s))
    data.append(f**2)
    signals.set_data(data,np.real(signalsData))

    if s==M-1:
        circlesData = []
        signalsData = []
        thetas = []
        fourierData.append(prom)
        data.append(circlesfreq)
        fourierData.set_data(data,np.sin(fourierData**2))
        prom = 0
        fractier = 0
        if fractier == 0:
            self.reset_value
            circlesfreq = fractier**f**2
            circlesData.set_data(real(circlesData),real(circlesData))
        return circlesData,signalsData,prom,fourierData

self.FuncAnimation(fig,update,f,interval=10,blit=False,repeat=True)
self.FuncAnimation(fig,update,f,interval=100,blit=False,repeat=True)
plt.show()
```

