Procesamiento de señales, TP1

Parte 3 - simulaciones:

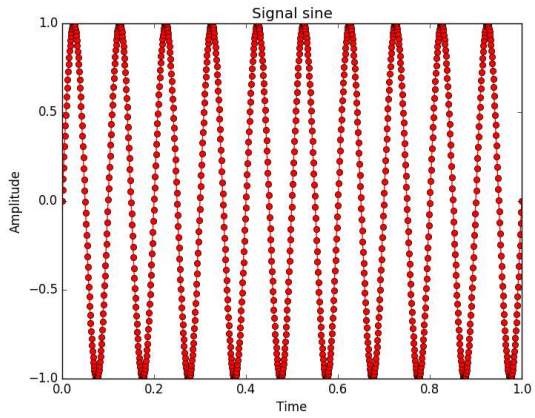
3-1

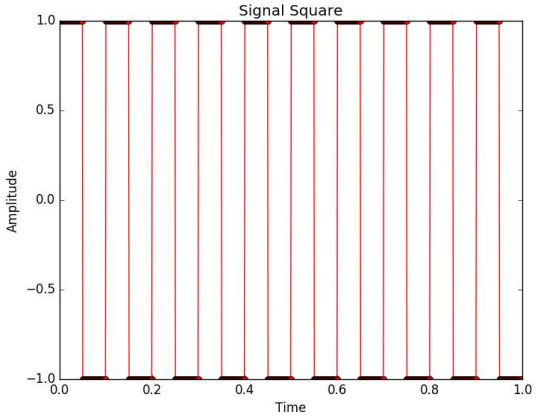
Se implementó el módulo signals.py, donde se definen 3 funciones que devuelven las señales de Sin, Cuadrada, y Triangular.

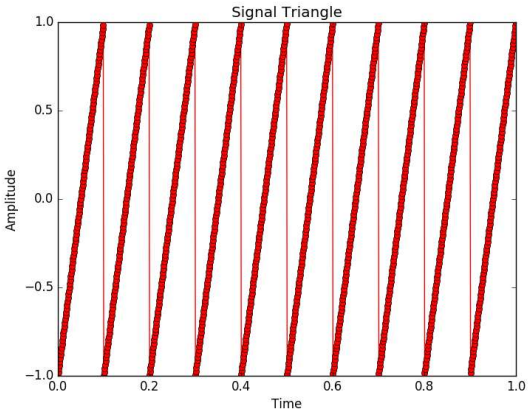
```python
import numpy   as     np
from scipy import signal
import matplotlib.pyplot    as     plt
from   matplotlib.animation import FuncAnimation
np.set_printoptions(precision=3, suppress=False)

# Sintetizar senales senoidal, cuadrada y triangular
# Frecuencia de muestreo: fs  (HZ) # Frecuencia: f0 (HZ) # Fase en radianes: fase
# # Amplitud: amp = [0,1] # cantidad de muestras: N

#generators con funciones
def signalSine(fs,f0, amp, N, fase):
    i = 0
    # inicializo array
    s = np.zeros((N ,), dtype=float)
    m = np.empty((2,N), float)
    # linspace asegura la cantidad de valores (inicio, parada, cantidad de puntos)
    t = np.linspace(0, 1, N, endpoint=True)
    for  i in range(N):
        s[i] = amp * np.sin(2*np.pi*fs*t[i]*1/f0 + fase)
        i+=1
    m[0] = np.transpose(s)
    m[1] = t
    return m

def signalSquare(fs,f0, amp, N):
    i = 0
    # inicializo array
    s = np.zeros((N ,), dtype=float)
    m = np.empty((2,N), float)
    # linspace asegura la cantidad de valores (inicio, parada, cantidad de puntos)
    t = np.linspace(0, 1, N, endpoint=True)
    for  i in range(N):
        s[i] = amp * signal.square(2*np.pi*fs*t[i]*1/f0)
        i+=1
    m[0] = np.transpose(s)
    m[1] = t
    #plt.plot(s,t,'ro-')
    return m
```

```python

def signalTriangle(fs,f0, amp, N):
    i = 0
    # inicializo array
    s = np.zeros((N ,), dtype=float)
    m = np.empty((2,N), float)
    # linspace asegura la cantidad de valores (inicio, parada, cantidad de puntos)
    t = np.linspace(0, 1, N, endpoint=True)
    for  i in range(N):
        s[i] = amp * signal.sawtooth(2*np.pi*fs*t[i]*1/f0)
        i+=1
    m[0] = np.transpose(s)
    m[1] = t
    #plt.plot(s,t,'ro-')
    return m
```

El correcto funcionamiento de las mismas se puede apreciar en las siguientes figuras:



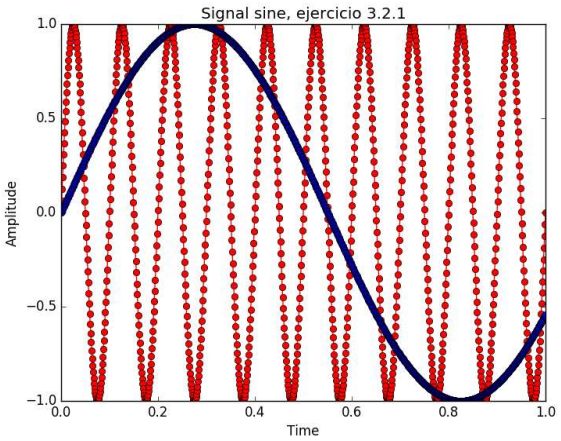




3-2

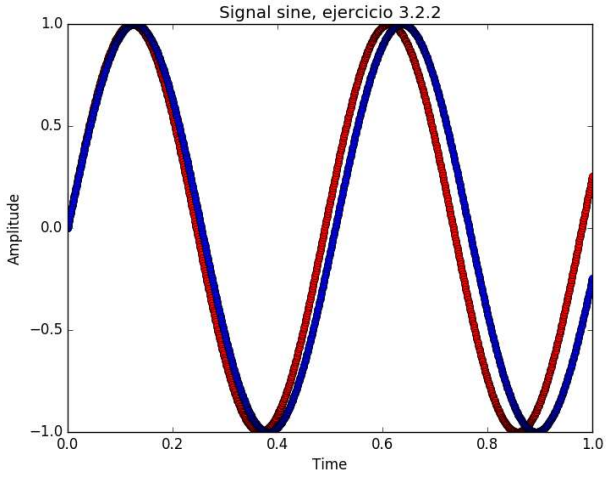
2_1

En la siguiente figura se muestra el resultado obtenido, estas señales no es posible diferenciarlas, hay que agregar un filtro antialiasing para no dejar que entre al sistema la señal no deseada.



2_2

En la siguiente figura se muestra el resultado obtenido, como se observa ambas señales tienen la misma frecuencia, pero se encuentran desfasadas.



Parte 5 - CIAA

5_1

Dentro de la carpeta TP1_5_1 se encuentra el código en C, donde, dado q7_t a= 0x040 y q7_t b=0x23, se calcula q7_t c = a*b e imprime el resultado. Se obtuvo como resultado en decimal de c=17.

Con respecto al redondeo, como el resultado de a*b es de 16 bits, el método fue un corrimiento a la derecha, perdiendo la parte menos significativa, la cual no se tiene en cuenta.