S1 Workshop 1- Signals in Time Domain

Group A03 (220046R - Asuramuni S Y and 220051D - Attanayake A Y)

Date of Lab: 01/03/2024 Date of Submission: 02/03/2024

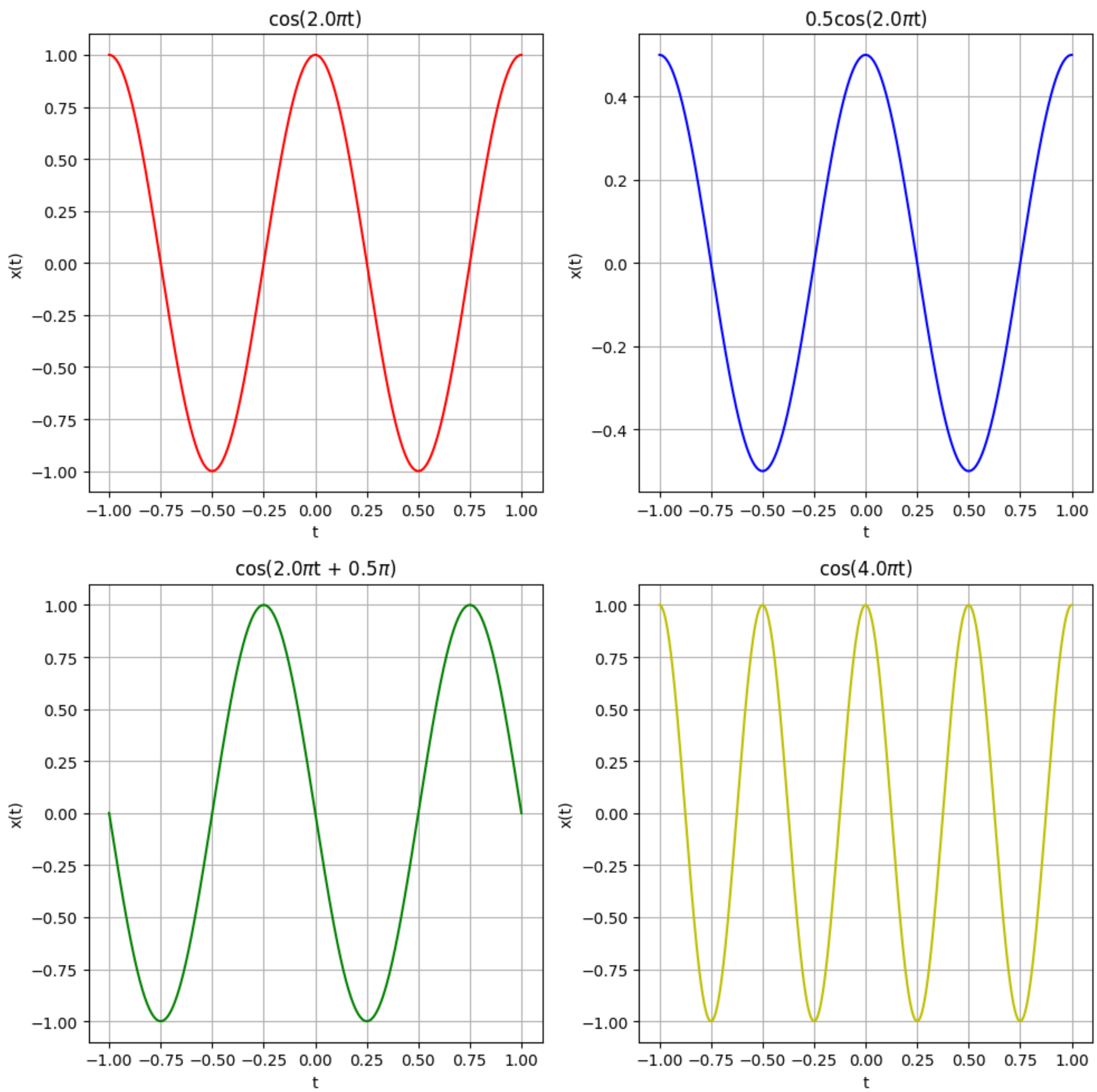
Task 1:

The following code plot will generate cosine waveforms where,

1. A = 1, w = 2*pi, phi = 0
2. A = 0.5, w = 2*pi, phi = 0
3. A = 1, w = 2*pi, phi = pi/2
4. A = 1, w = 4*pi, phi = 0

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt


sampling_frequency = 44100 #Don't change the sampling frequency unless needed
sampling_period = 1/sampling_frequency
t=np.arange(-1.,1.,sampling_period) #The time interval is from -1 to 1 with a step

def title_generator(A, w, phi):
    if A==1:
        title = 'cos(' + str(w/np.pi)+ r"$\pi$"
    else:
        title = str(A) + 'cos(' + str(w/np.pi)+ r"$\pi$"
    if phi == 0:
        title += 't)'
    else:
        title +="t + " + str(phi/np.pi) +r"$\pi$"+ ')'
    return title

def cosine_plotter(A, w, phi, row, column, ax, color="b"):

    x = A * np.cos(w * t + phi) #The cosine function required is generated here
    title = title_generator(A, w, phi)
    ax[row, column].plot(t, x, label=title, color= color)
    ax[row, column].set_title(title)
    ax[row, column].set_xlabel('t')
    ax[row, column].set_ylabel('x(t)')
    ax[row, column].grid(True)

fig, ax = plt.subplots(2, 2, figsize=(10, 10)) # Create a 2x2 grid of Axes objects

cosine_plotter(1, 2*np.pi, 0, 0, 0, ax, "r") #Plots a graph with  A = 1, w = 2*pi,
cosine_plotter(0.5, 2*np.pi, 0, 0, 1, ax) #Plots a graph with  A = 0.5, w = 2*pi, p
cosine_plotter(1, 2*np.pi, np.pi/2, 1, 0, ax, "g") #Plots a graph with  A = 1, w =
cosine_plotter(1, 4*np.pi, 0, 1, 1, ax, "y") #Plots a graph with  A = 1, w = 4*pi,
```

```
plt.tight_layout()
plt.show()
```



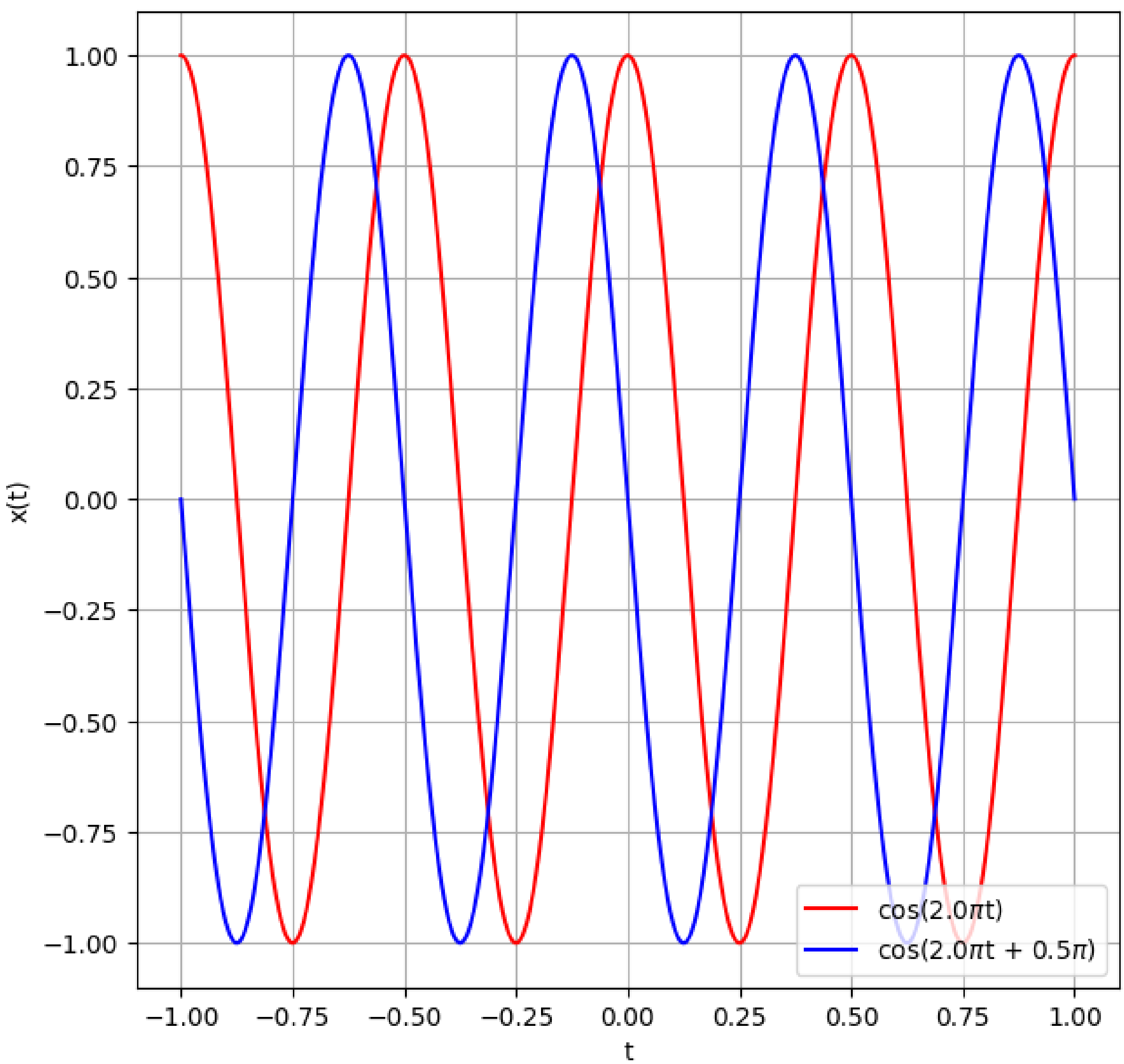
x1(t), x2(t) and x4(t) are even signals while x3(t) is an odd signal

Example 2:

Plotting 2 graphs in the same plane x(t) = A cos(w*t + phi) where phi = 0 and phi =pi/2

In [ ]:
```
samp_f = 44000 #Don't change the sampling frequency unless needed
samp_T = 1/samp_f
t=np.arange(-1.,1.,samp_T) #The time interval is from -1 to 1 with a step of sampli

def title_generator(A, w, phi):
    if A==1:
        title = 'cos(' + str(w/np.pi)+ r"$\pi$"
    else:
        title = str(A) + 'cos(' + str(w/np.pi)+ r"$\pi$"
```

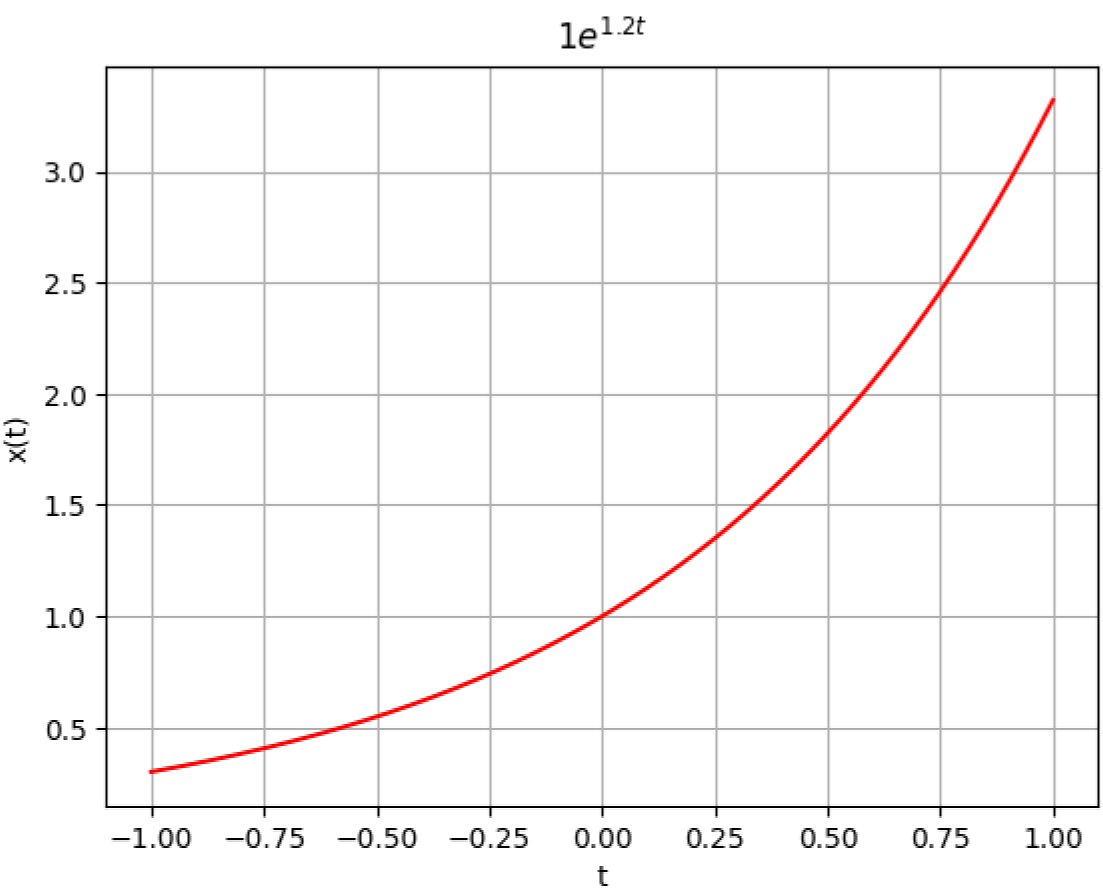
```
    if phi == 0:
        title += 't)'
    else:
        title +="t + " + str(phi/np.pi) +r"$\pi$"+ ')'
    return title

def cosine_plotter(A, w, phi, ax, color="b"):

    x = A * np.cos(2 * w * t + phi) #The cosine function required is generated here
    title = title_generator(A, w, phi)
    ax.plot(t, x, label=title, color= color)
    ax.set_xlabel('t')
    ax.set_ylabel('x(t)')
    ax.grid(True)

fig, ax = plt.subplots(1, 1, figsize=(7, 7)) # Create a 2x2 grid of Axes objects
cosine_plotter(1, 2*np.pi, 0, ax, "r") #Plots a graph with  A = 1, w = 2*pi, phi =
cosine_plotter(1, 2*np.pi, np.pi/2 , ax) #Plots a graph with  A = 0.5, w = 2*pi, ph
plt.legend(loc='lower right')
plt.show()
```



1.2: Real CT Exponential

The following code will generate a signal x(t) = C * exp(alpha * t) with C = 1.0 and alpha = 1.2

In [ ]:
```
samp_f = 44000 #Don't change the sampling frequency unless needed
samp_T = 1/samp_f
t=np.arange(-1.,1.,samp_T) #The time interval is from -1 to 1 with a step of sampli

def expo_graph_generator(c, alpha):
    x = c * np.exp(alpha * t)
    title = r"${} e^{{{}t}}$".format(c, alpha)
    plt.plot(t, x, label=title, color="r")
    plt.title(title)
    plt.xlabel('t')
    plt.ylabel('x(t)')
    plt.grid(True)

expo_graph_generator(1, 1.2)
plt.show()
```



When the value of alpha is:

- Negative - exponentially decaying function is generated

- Positive - exponentially growing function is generated

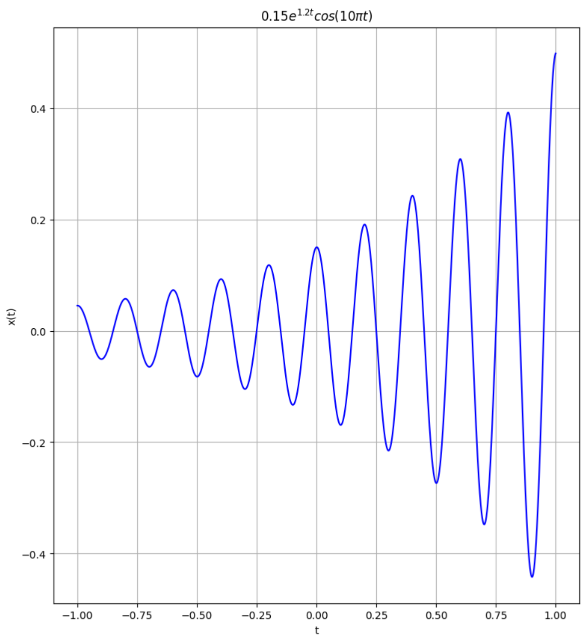
1.3 Growing Sinusoidal Signal

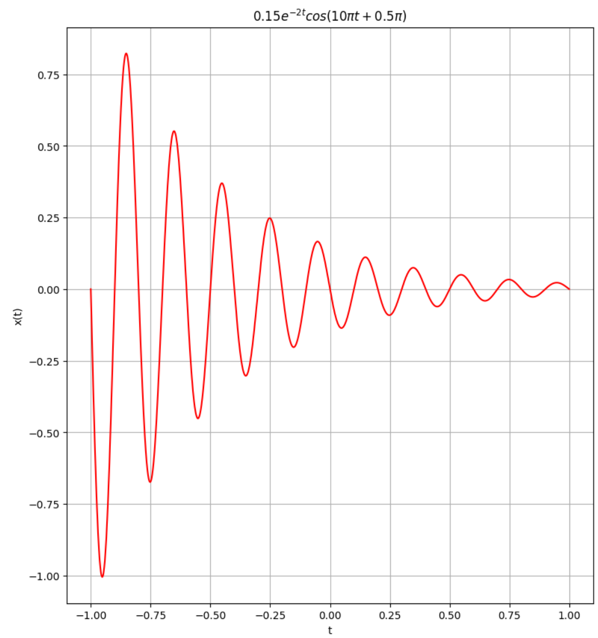
The following code will generate a signal with $x(t) = C\ exp(rt)$ *cos$(\omega t + \theta)$:

1. $C = 0.15$, $r = 2$, and $\theta = 0$.
2. $C = 0.15$, $r = -2$, and $\theta = \pi/2$

In [ ]:

```
samp_f = 44000 #Don't change the sampling frequency unless needed
samp_T = 1/samp_f
t=np.arange(-1.,1.,samp_T) #The time interval is from -1 to 1 with a step of sampli

f = 5 #Frequency of the signal is 5 Hz

def title_generator(C, R, f, theta):
    if theta == 0:
        title = r"${} e^{{{}t}} cos( {} \pi  t)$".format(C, R, 2*f)
    else:
        title = r"${} e^{{{}t}} cos( {} \pi  t + {} \pi)$".format(C, R, 2*f, theta/

    return title


def expo_sin_generator(C , R, theta, column, ax, co='b'):
    x = C * np.exp(R * t) * np.cos(2 * np.pi * f * t + theta)
    title = title_generator(C , R, f, theta)
    ax[column].plot(t, x, label=title, color=co)
    ax[column].set_title(title)
    ax[column].set_xlabel('t')
    ax[column].set_ylabel('x(t)')
    ax[column].grid(True)

fig, ax = plt.subplots(1, 2, figsize=(20, 10)) # Create a 2x2 grid of Axes objects

expo_sin_generator(0.15, 1.2, 0, 0, ax)
expo_sin_generator(0.15, -2, np.pi/2, 1, ax,'r')
plt.show()
```





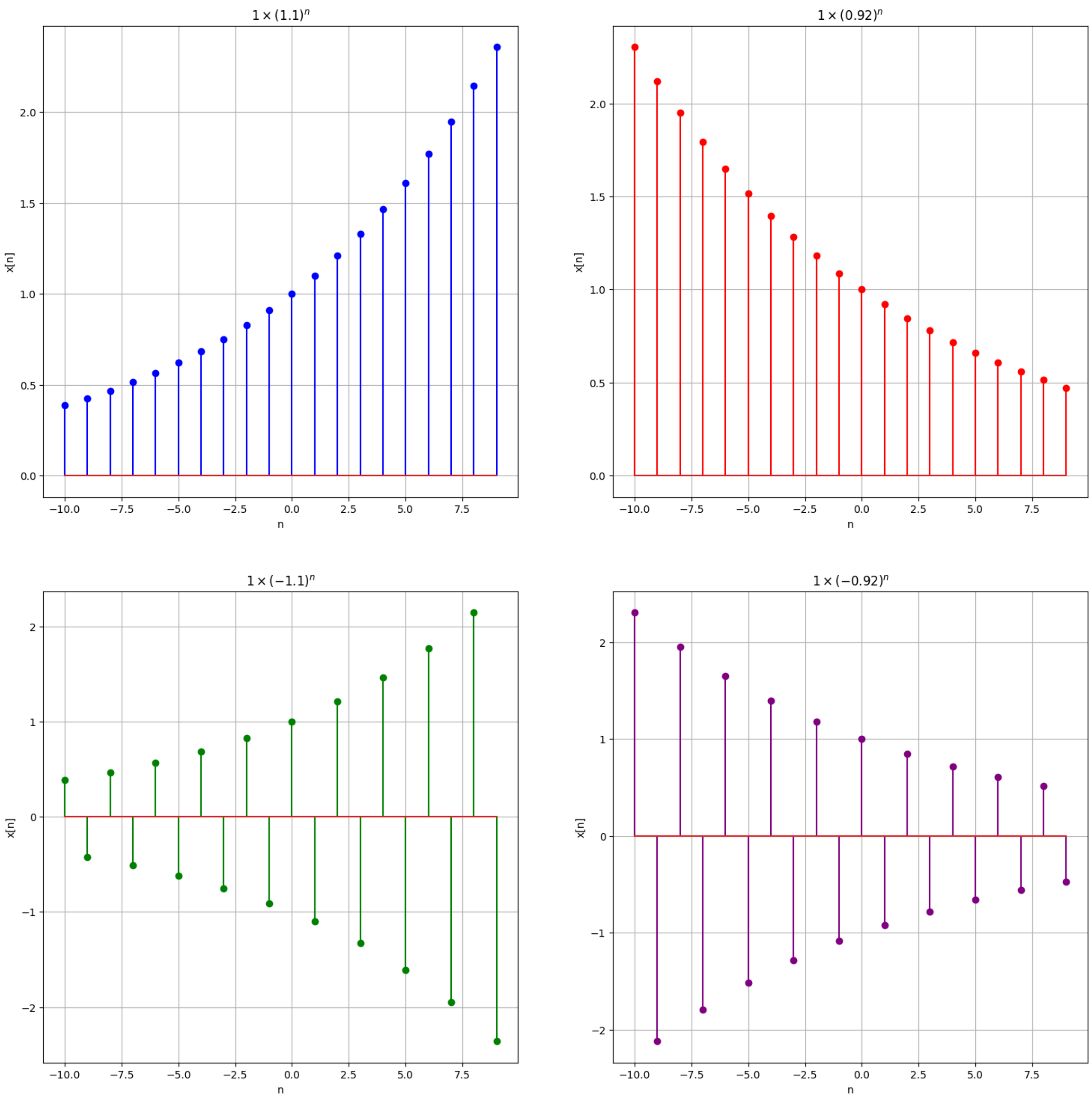
1.4: Real DT Exponential

The following code will generate the discrete-time signal $x[n] = C\alpha\text{^}n$ for:

1. $C = 1$, $\alpha = 1.1$
2. $C = 1$, $\alpha = 0.92$
3. $C = 1$, $\alpha = -1.1$
4. $C = 1$, $\alpha = -0.92$

In [ ]:

```python
n=np.arange(-10,10,1) #The n interval is from -10 to 10 with a step of 1

def expo_graph_generator_dt(c, alpha,row,column, ax, color):
    x = c * alpha ** n
    title = r"${} \times ({})^n$".format(c, alpha)
    ax[row,column].stem(n, x, linefmt=color)
    ax[row,column].set_title(title)
    ax[row,column].set_xlabel('n')
    ax[row,column].set_ylabel('x[n]')
    ax[row,column].grid(True)

fig, ax = plt.subplots(2,2, figsize=(18,18))

expo_graph_generator_dt(1, 1.1, 0, 0, ax, 'b')
expo_graph_generator_dt(1, 0.92, 0, 1, ax, 'r')
expo_graph_generator_dt(1, -1.1, 1, 0, ax, 'g')
expo_graph_generator_dt(1, -0.92, 1, 1, ax, 'purple')

plt.show()
```



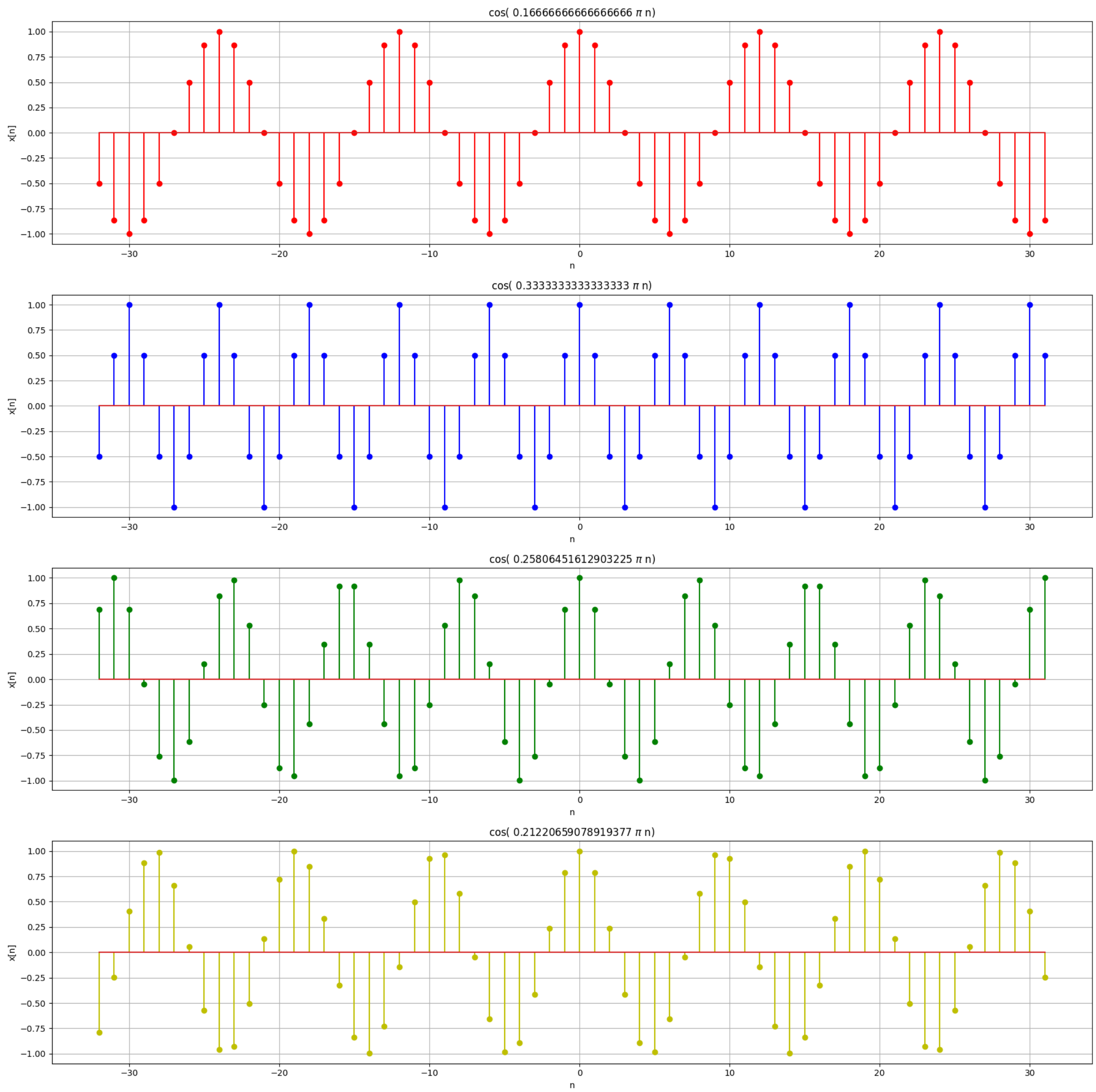
1.5: DT Sinusoids

The following code will generate the discrete-time signal $x[n]$ as:

1. $x1[n] = A\cos(\omega n)$ with $A = 1$ and $\omega = 2\pi/12$.
2. $x2[n] = A\cos(\omega n)$ with $A = 1$ and $\omega = 2\pi/6$.
3. $x3[n] = A\cos(\omega n)$ with $A = 1$ and $\omega = 8\pi/31$.
4. $x4[n] = A\cos(\omega n)$ with $A = 1$ and $\omega = 1/1.5$

In [ ]:
```
n=np.arange(-32.,32., 1) #The n interval is from -32 to 32 with a step of 1
def title_generator(A, w):
    if A==1:
        title = 'cos( ' + str(w/np.pi)+ r" $\pi$"
    else:
        title = str(A) + 'cos( ' + str(w/np.pi)+ r" $\pi$"
    title += ' n)'
```

```
    return title

def cosine_plotter_dt(A, w, row, ax, color="b"):

    x = A * np.cos(w * n) #The cosine function required is generated here
    title = title_generator(A, w)
    ax[row].stem(n, x, linefmt= color)
    ax[row].set_title(title)
    ax[row].set_xlabel('n')
    ax[row].set_ylabel('x[n]')
    ax[row].grid(True)

fig, ax = plt.subplots(4, 1, figsize=(18, 18)) # Create a 2x2 grid of Axes objects

cosine_plotter_dt(1, np.pi/6, 0, ax, "r") #Plots a graph with  A = 1, w = pi/6
cosine_plotter_dt(1, np.pi/3, 1, ax) #Plots a graph with  A = 1, w = pi/3
cosine_plotter_dt(1, 8*np.pi/31, 2, ax, "g") #Plots a graph with  A = 1, w = 8*pi/3
cosine_plotter_dt(1, 1/1.5, 3, ax, "y") #Plots a graph with  A = 1, w = 1/1.5




plt.tight_layout()
plt.show()
```



About the period for the periodic signals:

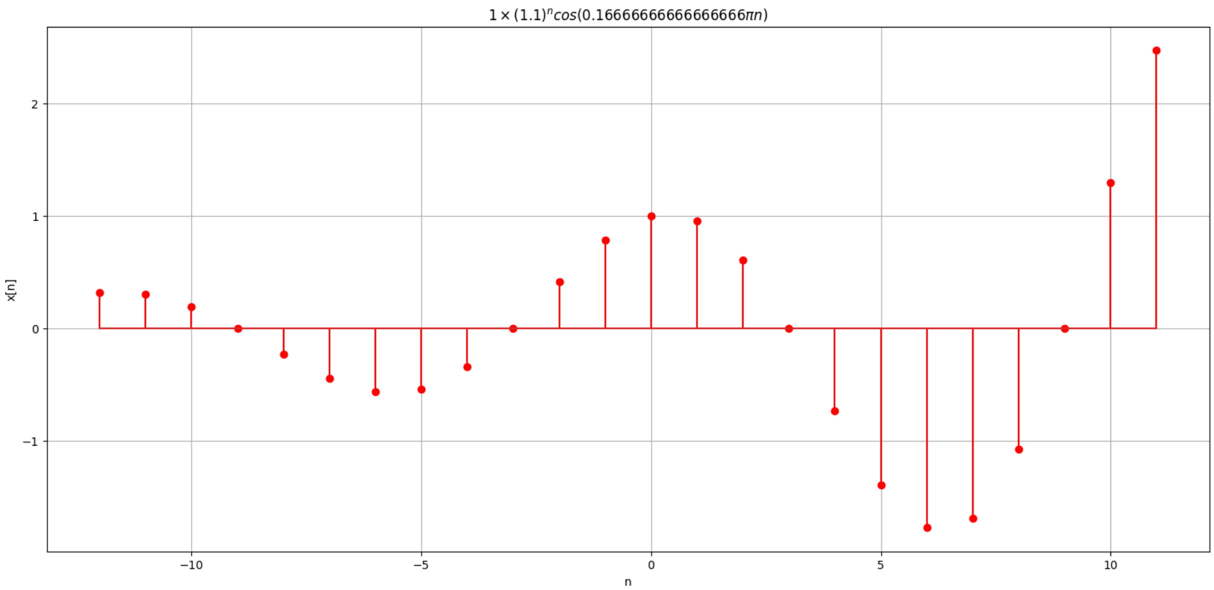
1. $x1[n]$ - is periodic; T = 12
2. $x2[n]$ - is periodic; T = 6
3. $x3[n]$ - is periodic; T = 31
4. $x4[n]$ - non periodic

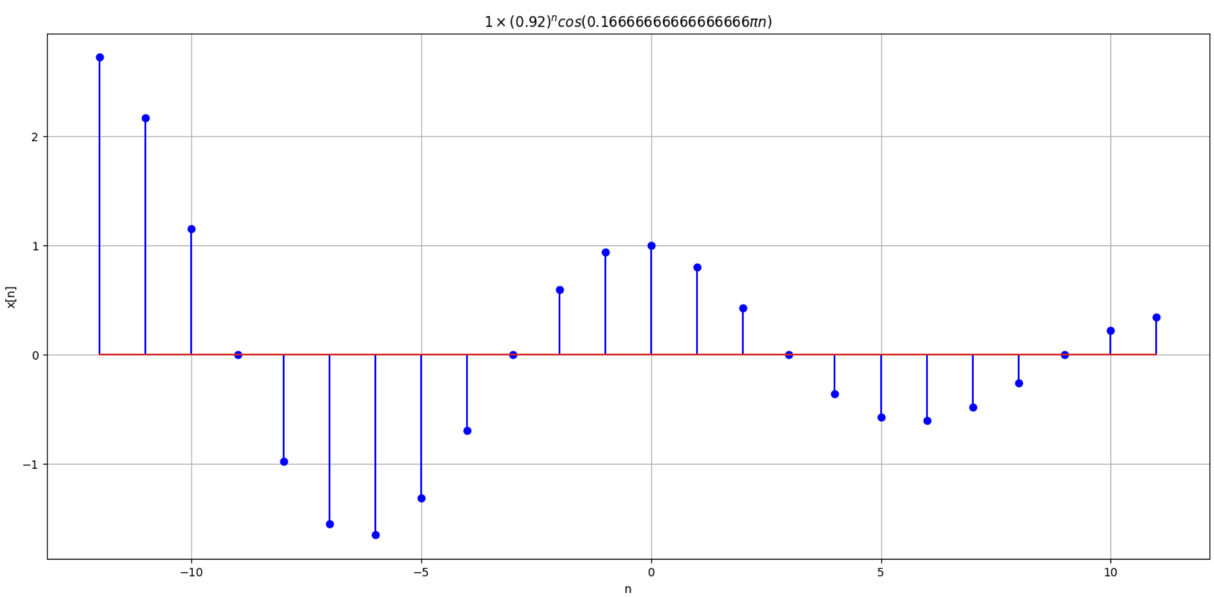
1.6: General Complex Exponential Signals

The following code will plot the real part of $x[n] = C\alpha$^n with $C = |C|e$^$(j\theta)$, $\alpha = |\alpha|\ e$^$(j\omega)$ for following C, α, and ω values.

1. $C = 1$, $\alpha = 1.1$, $\omega = 2\pi/12$
2. $C = 1$, $\alpha = 0.92$, $\omega = 2\pi/12$

```
In [ ]: n = np.arange(-12, 12, 1) #The n interval is from -12 to 12 with a step of 1

fig, ax = plt.subplots(2, 1, figsize=(18, 18)) # Create a 2x1 grid of Axes objects

def real_expo_generator(c, alpha, w, row, ax, color='b'):
    x = c * (abs(alpha) **n) * np.cos( w * n)
    ax[row].stem(n, x, linefmt= color)
    title = r"${} \times ({})^n cos ({} \pi n)$".format(c, alpha, w/np.pi)
    ax[row].set_title(title)
    ax[row].set_xlabel('n')
    ax[row].set_ylabel('x[n]')
    ax[row].grid(True)

real_expo_generator(1, 1.1, np.pi/6, 0, ax, 'r') #Plots a graph with  C = 1, alpha
real_expo_generator(1, 0.92, np.pi/6, 1, ax) #Plots a graph with  C = 1, alpha = 0.
```



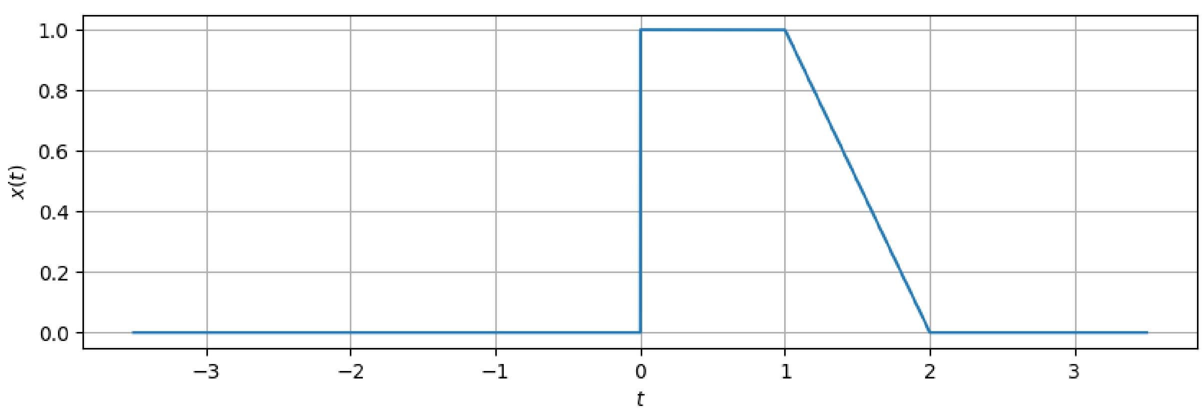


When alpha is a negative value, there will not be any change in the wave form as in the equation alpha is within a modulus.

1.7: Transformation of the Independent Variable

The following code was provided which plots x(t):

In [ ]:
```
def x(t):
    if (t < 0.):
        return 0.
    elif (t < 1.):
        return 1.
    elif (t < 2.):
        return 2. - t
    else:
        return 0.

fs = 4.4e4 # 44,000-Hz sampling frequency
ts = 1/fs
t = np.arange(-3.5, 3.5, ts) # A linearly-spaced array with step ts
fig, axes = plt.subplots(1,1, sharey='all', figsize=(10,3))
# x(t)
axes.plot(t, [x(t_) for t_ in t])
axes.set_xlabel('$t$')
axes.set_ylabel('$x(t)$')
axes.grid(True)
# Your code goes here
plt.show()
```

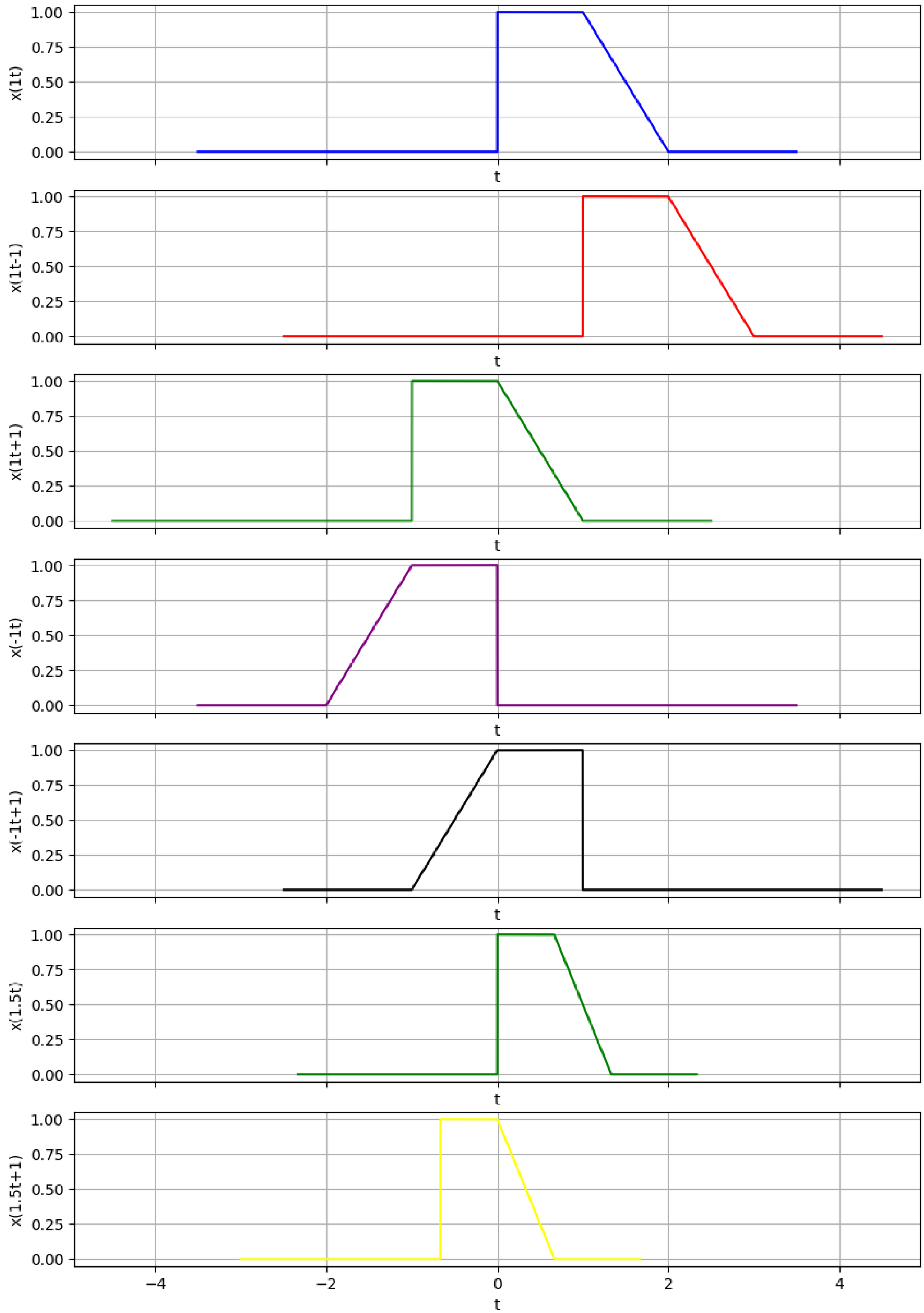


The updated code is given below where the following functions are to be plotted:

1. $x(t-1)$
2. $x(t+1)$
3. $x(-t)$
4. $x(-t+1)$
5. $x(3t/2)$
6. $x(3t/2+1)$

In [ ]:

```
def x(t):
    if (t < 0.):
        return 0.
    elif (t < 1.):
        return 1.
    elif (t < 2.):
        return 2. - t
    else:
        return 0.

fs = 4.4e4 # 44,000-Hz sampling frequency
ts = 1/fs
t = np.arange(-3.5, 3.5, ts) # A linearly-spaced array with step ts
fig, axes = plt.subplots(7,1, sharex='all', figsize=(10,15))
# x(t)

def title_generator(tt,shift):
    if shift<0:
        title= "x({}t{})".format(tt,shift)
    elif shift>0:
        title= "x({}t+{})".format(tt,shift)
    else:
        title= "x({}t)".format(tt)

    return title

def x_transformer(cof_t,shift, row, color='r'):
    axes[row].plot((t/cof_t)-(1/cof_t)*shift, [x(t_) for t_ in t],color=color)
    title = title_generator(cof_t,shift)
    axes[row].set_xlabel("t")
    axes[row].set_ylabel(title)
    axes[row].grid(True)

x_transformer(1,0,0,'b') #plots the original x(t) graph
x_transformer(1,-1,1) #plots x(t-1) graph
x_transformer(1,1,2,'g') #plots x(t+1) graph
x_transformer(-1,0,3,'purple') #plots x(-t) graph
x_transformer(-1,1,4, 'black') #plots x(-t+1) graph
x_transformer(3/2,0,5,'green') #plots x(1.5t) graph
x_transformer(3/2,1,6, 'yellow') #plots x(1.5t+1) graph


plt.show()
```

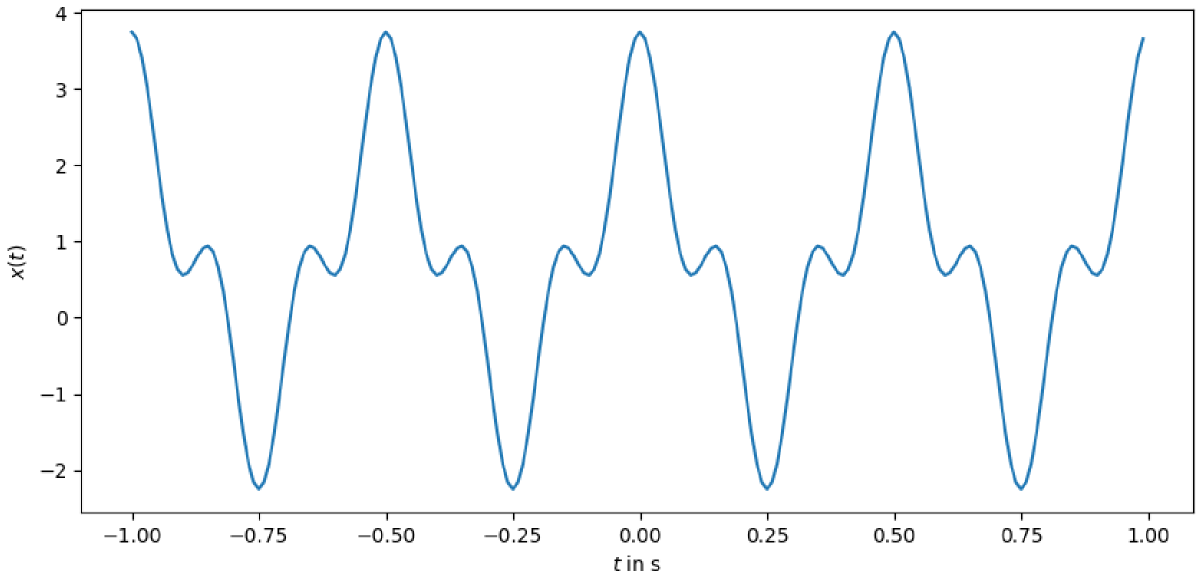

x(1t)
x(1t-1)
x(1t+1)
x(-1t)
x(-1t+1)
x(1.5t)
x(1.5t+1)
1.00
0.75
0.50
0.25
0.00
−4
−2
0
2
4
t

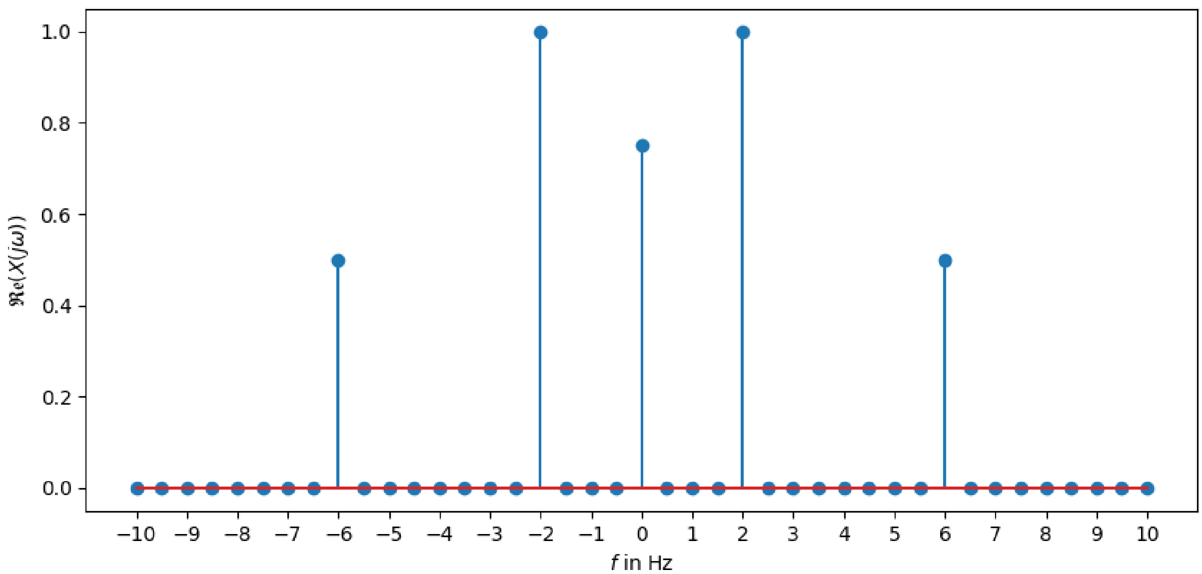
1.8: Observing a Signal in Frequency Domain

The following code was provided:

In

```
fs = 100# 100-Hz sampling frequency
ts = 1/fs
t = np.arange(-1., 1., ts) # A linearly-spaced array with step ts
fig, axes = plt.subplots(2,1, figsize=(10,10))
f = 2 # 2 Hz
omega0 = 2*np.pi*f

xt = 0.75 + 2.*np.cos(omega0*t) + 1.*np.cos(3*omega0*t)

Xf = np.fft.fft(xt)
freq = np.fft.fftfreq(t.shape[-1], d=ts)

axes[0].plot(t,xt)
axes[0].set_xlabel('$t$ in s')
axes[0].set_ylabel('$x(t)$')

valsubrange = np.concatenate((np.arange(0,21,1), np.arange(-1,-21,-1)))
freqsubrage = np.concatenate((np.arange(0,21,1), np.arange(-1,-21,-1)))

axes[1].stem(freq[freqsubrage], Xf.real[valsubrange]/len(t))
axes[1].set_xlabel('$f$ in Hz')
axes[1].set_ylabel(r'$ \mathfrak{Re}(X(j\omega))$')

plt.xticks(np.arange(-10,11))
plt.show()
```





Observing the frequency domain representation, it is noted that we can see spikes at 0 Hz (a dc offset), 2 Hz and 6 Hz and their mirroring at negative values

1.9: Simple Audio Effects

The following code is provided:

In [ ]:
```
import numpy as np
import pyaudio
import wave
from IPython.display import Audio
# import utility
CHUNK = 8820 # 44100 = 1 s

wf = wave.open("anthem.wav", 'r')
p = pyaudio.PyAudio()
```

```
nchannels=wf.getnchannels()
stream = np.array(np.zeros(nchannels), dtype=np.int16) # init stream

data = wf.readframes(CHUNK)
dtype = '<i2' # little-endian two-byte (int16) signed integers
sig = np.frombuffer(data, dtype=dtype).reshape(-1, nchannels)
signal_chunk = np.asarray(sig)
delayed = np.zeros(signal_chunk.shape, dtype=dtype)

i=0
alpha = 1.0
while data != "" and signal_chunk.shape[0] == CHUNK and i<120:
    i+=1
    modified_signal_chunk = alpha*signal_chunk + (1. - alpha)*delayed
    modified_signal_chunk_int16 = modified_signal_chunk.astype(np.int16)
    stream = np.vstack((stream, modified_signal_chunk_int16)) # append modified to
    delayed = signal_chunk
    data = wf.readframes(CHUNK)
    sig = np.frombuffer(data, dtype=dtype).reshape(-1, nchannels)
    signal_chunk = np.asarray(sig)

stream = stream[1:] # pop stream init
byte_stream = stream.tobytes() # np array to bytes
p.terminate()
wf.close()
wfo = wave.open("anthem_modified.wav", 'wb') # writing the bot stream to a output w
wfo.setnchannels(nchannels)
wfo.setsampwidth(wf.getsampwidth())
wfo.setframerate(wf.getframerate())
wfo.writeframes(byte_stream)
wfo.close()
Audio('anthem_modified.wav')
```

Out[ ]:

0:00 / 0:24

The code is modified as followed to slow down by 1.5 times and the time limit was elongated to get in the full audio :

In [ ]:

```
import numpy as np
import pyaudio
import wave
from IPython.display import Audio

CHUNK = 18820 # 44100 = 1 s

wf = wave.open("anthem.wav", 'r')
p = pyaudio.PyAudio()

nchannels=wf.getnchannels()
stream = np.array(np.zeros(nchannels), dtype=np.int16) # init stream

data = wf.readframes(CHUNK)
dtype = '<i2' # little-endian two-byte (int16) signed integers
sig = np.frombuffer(data, dtype=dtype).reshape(-1, nchannels)
```

```python
signal_chunk = np.asarray(sig)
delayed = np.zeros(signal_chunk.shape, dtype=dtype)

i=0
alpha = 1.0
while data != b"" and signal_chunk.shape[0] == CHUNK and i<120:
    i+=1
    modified_signal_chunk = alpha*signal_chunk + (1. - alpha)*delayed
    modified_signal_chunk_int16 = modified_signal_chunk.astype(np.int16)
    stream = np.vstack((stream, modified_signal_chunk_int16)) # append modified to
    delayed = signal_chunk
    data = wf.readframes(CHUNK)
    sig = np.frombuffer(data, dtype=dtype).reshape(-1, nchannels)
    signal_chunk = np.asarray(sig)

stream = stream[1:] # pop stream init
byte_stream = stream.tobytes() # np array to bytes
p.terminate()
wf.close()

wfo = wave.open("anthem_modified.wav", 'wb') # writing the bot stream to a output w
wfo.setnchannels(nchannels)
wfo.setsampwidth(wf.getsampwidth())
wfo.setframerate(int(wf.getframerate()/1.5)) # Slow down by 1.5 times
wfo.writeframes(byte_stream)
wfo.close()

Audio('anthem_modified.wav')
```

Out[ ]:

0:00 / 0:47

Here, the initial framerate is divided 1.5 to get the final output framerate from the code line:

wfo.setframerate(int(wf.getframerate() / 1.5)) # Slow down by 1.5 times

This slows down the audio file

And the CHUNK size was changed to a larger value which increases the time limit to be able the read the full audio file