lab1

August 24, 2020

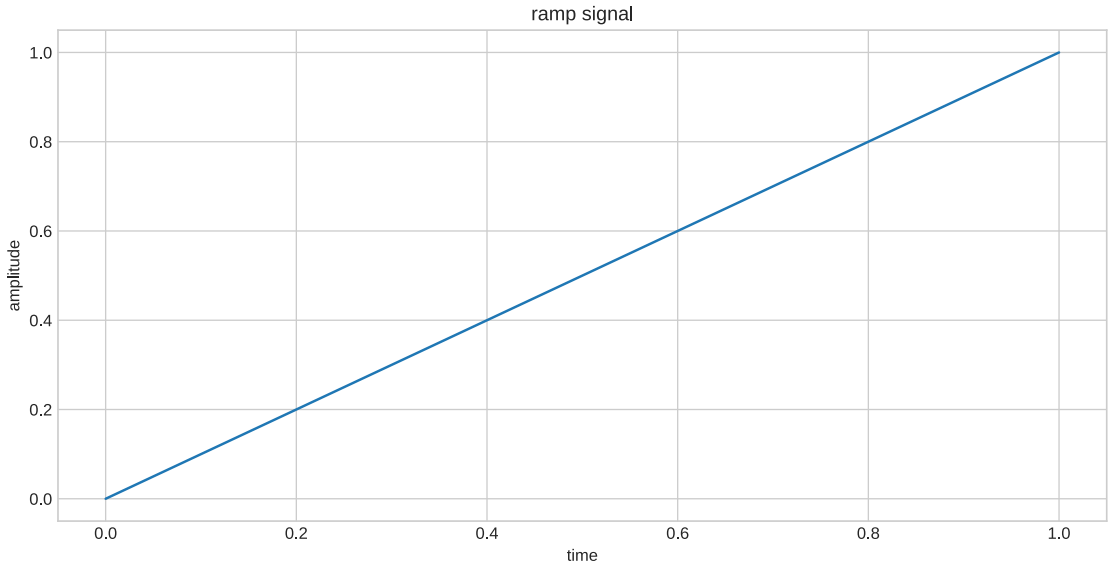
[1]:
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
pi = np.pi

def createSubplot(n):
    fig,ax = plt.subplots(n,figsize=(10,10))
    fig.tight_layout(pad=3.0)
    return fig,ax
```

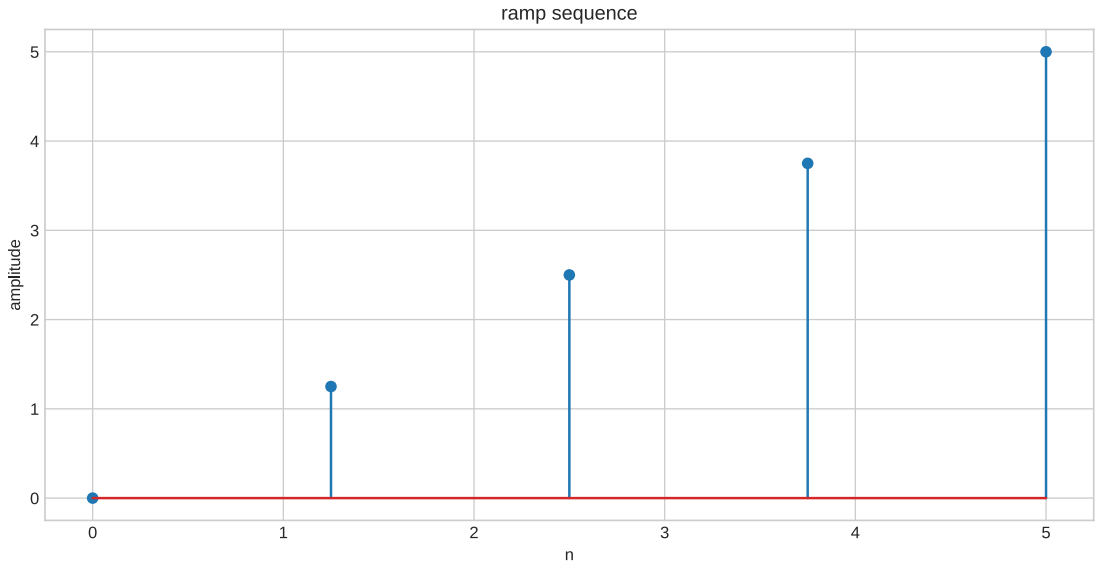
1 1 Generating Singnal

1.1 Ramp signal and Sequence

[2]:
```
def ramp(t):
    return t

fs = 5
t = np.linspace(0,1,fs)
# print(t.shape) #(5,)
fig, ax = createSubplot(2)
ramp_signal = ramp(t)
ax[0].plot(t,ramp_signal)
ax[0].set_xlabel('time')
ax[0].set_ylabel('amplitude')
ax[0].set_title('ramp signal')

n = np.linspace(0,fs,fs)
ramp_seq = ramp(n)
ax[1].stem(n,ramp_seq)
ax[1].set_xlabel('n')
ax[1].set_ylabel('amplitude')
ax[1].set_title('ramp sequence')
```

[2]:
```
Text(0.5, 1.0, 'ramp sequence')
```



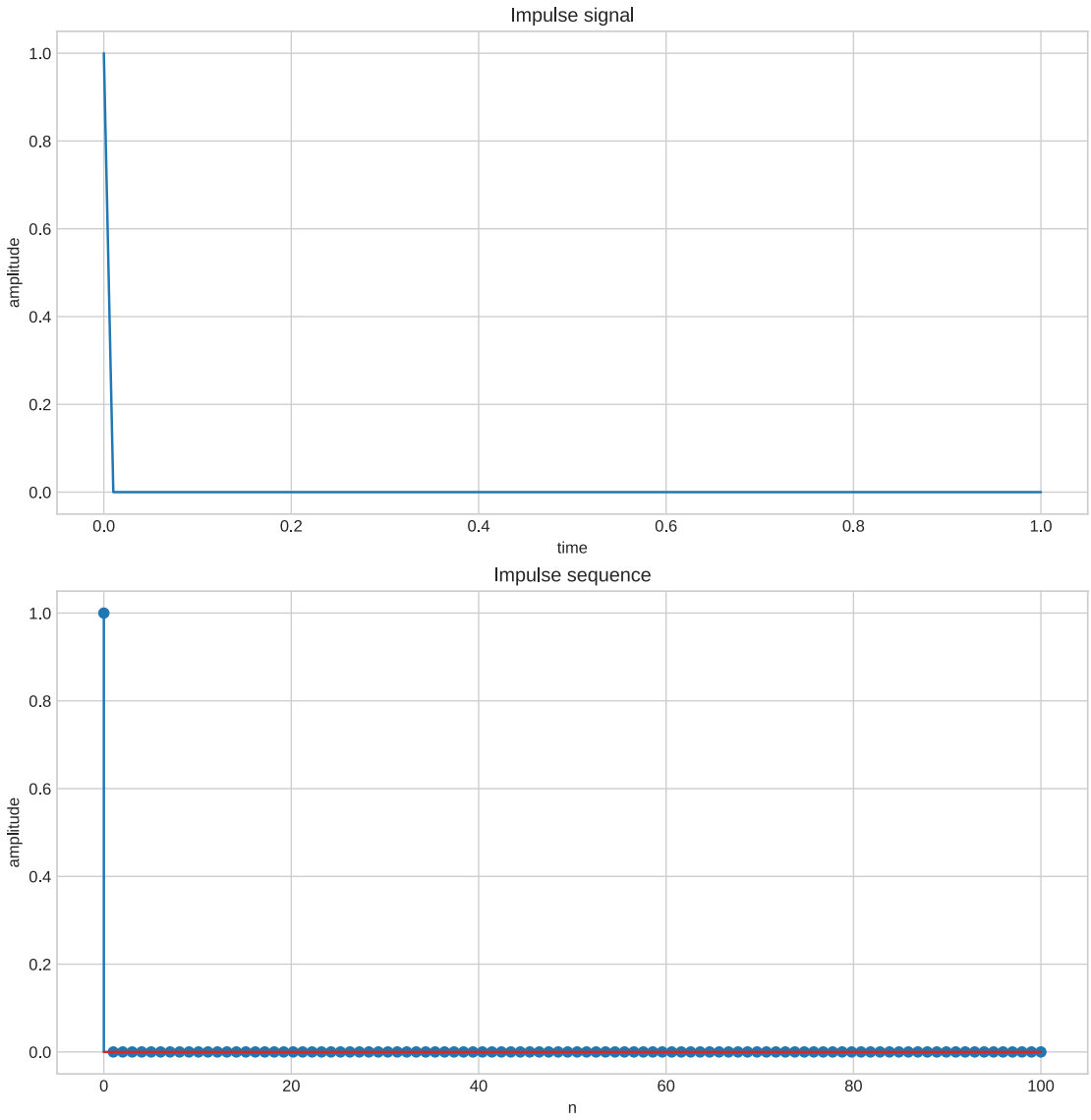


1.2 Impulse signal and Impulse sequence

[3]:
```
def impulse(t):
    sig = np.zeros(len(t))
    sig[0] = 1
    return sig

fs = 100
t = np.linspace(0,1,fs)

fig, ax = createSubplot(2)
impulse_signal = impulse(t)
```

```
ax[0].plot(t,impulse_signal)
ax[0].set_xlabel('time')
ax[0].set_ylabel('amplitude')
ax[0].set_title('Impulse signal')

n = np.linspace(0,fs,fs)
impulse_seq = impulse(n)
ax[1].stem(n,impulse_seq)
ax[1].set_xlabel('n')
ax[1].set_ylabel('amplitude')
ax[1].set_title('Impulse sequence')
```

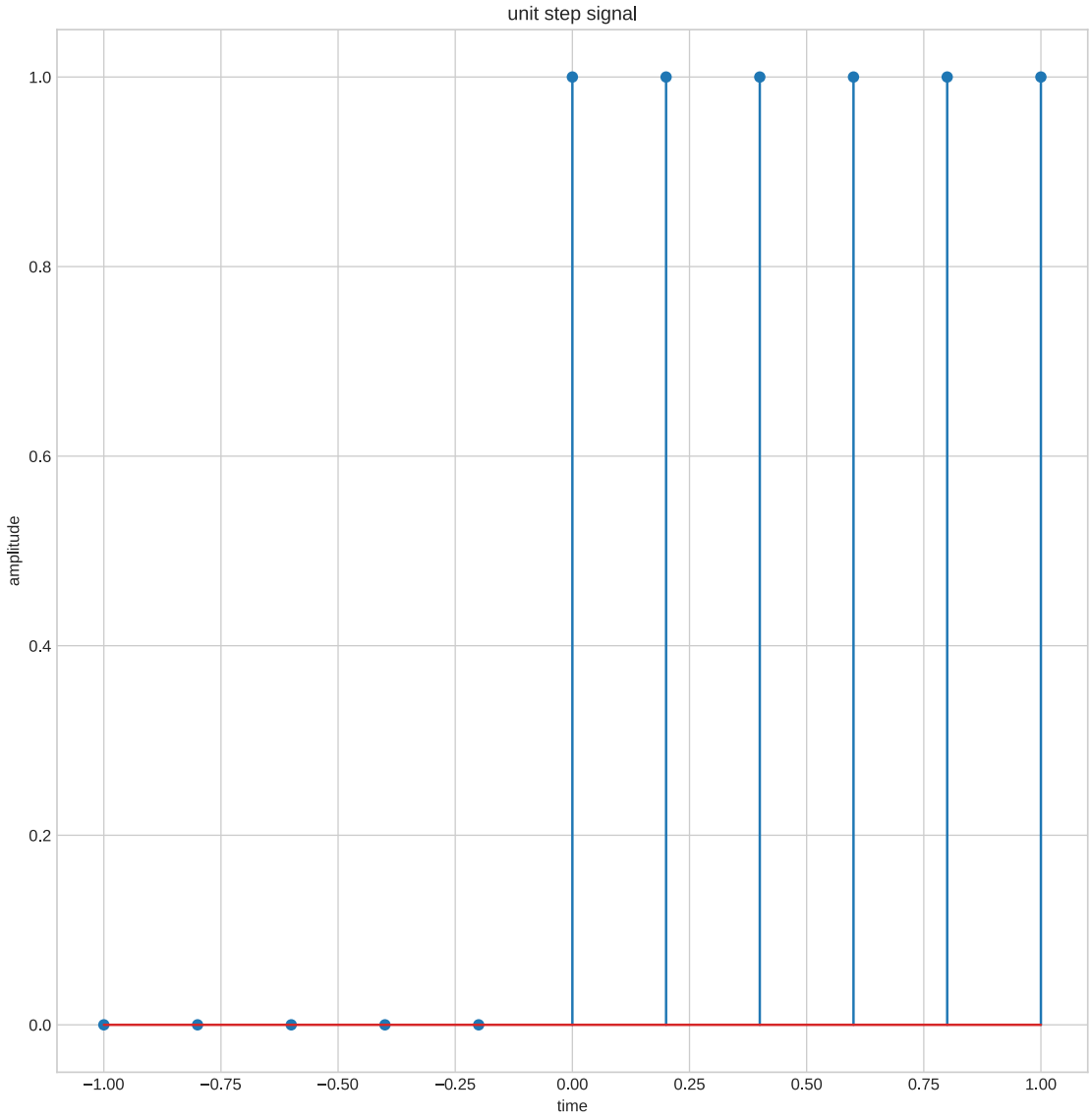
[3]: Text(0.5, 1.0, 'Impulse sequence')



1.3 unit step signal

[4]:
```
def unit(t):
    # sig = np.concatenate(np.zeros(len(t)//2),np.ones(len(t)//2))
    s = t.copy()
    s[s>=0] = 1
    s[s<0] = 0
    return s

fs = 5
t = np.linspace(-1,1,(fs*2)+1)

fig, ax = createSubplot(1)
unit_signal = unit(t)
ax.stem(t,unit_signal)
ax.set_xlabel('time')
ax.set_ylabel('amplitude')
ax.set_title('unit step signal')
```

[4]:
```
Text(0.5, 1.0, 'unit step signal')
```

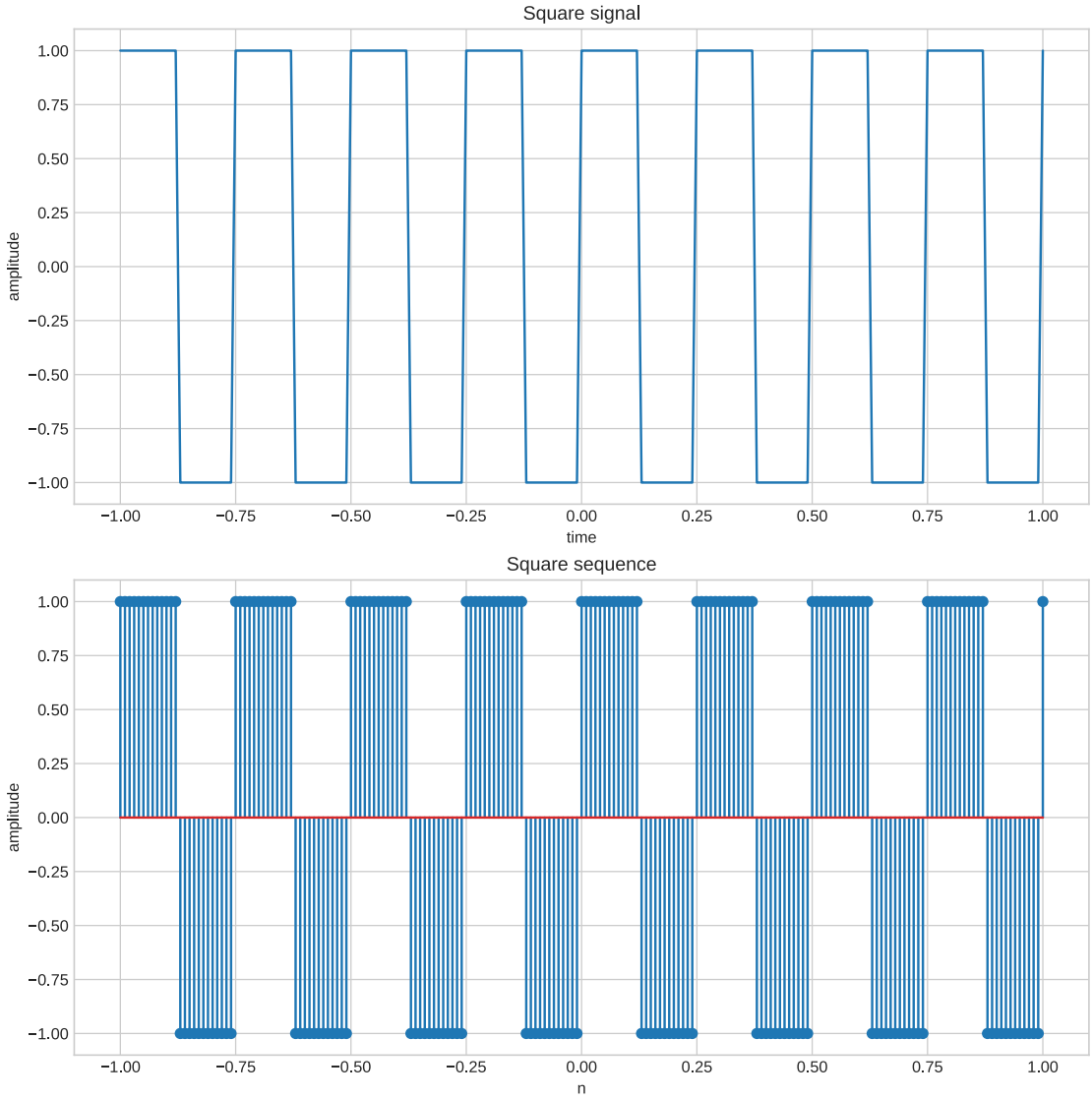


1.4 square wave signal and square wave sequence

[5]:
```
from scipy import signal
def square(t,f):
    return signal.square(2 * pi * f * t)
    # s = t.copy()
    # T = 1/f
    # s[s%T < T/2] = 1
    # s[s%T > T/2] = 0
    # return s
```

```
f = 4 #Hz
fs = 100
t = np.linspace(-1,1,(fs*2)+1)

fig, ax = createSubplot(2)
square_signal = square(t,f)
ax[0].plot(t,square_signal)
ax[0].set_xlabel('time')
ax[0].set_ylabel('amplitude')
ax[0].set_title('Square signal')

ax[1].stem(t,square_signal)
ax[1].set_xlabel('n')
ax[1].set_ylabel('amplitude')
ax[1].set_title('Square sequence')
```

[5]: Text(0.5, 1.0, 'Square sequence')

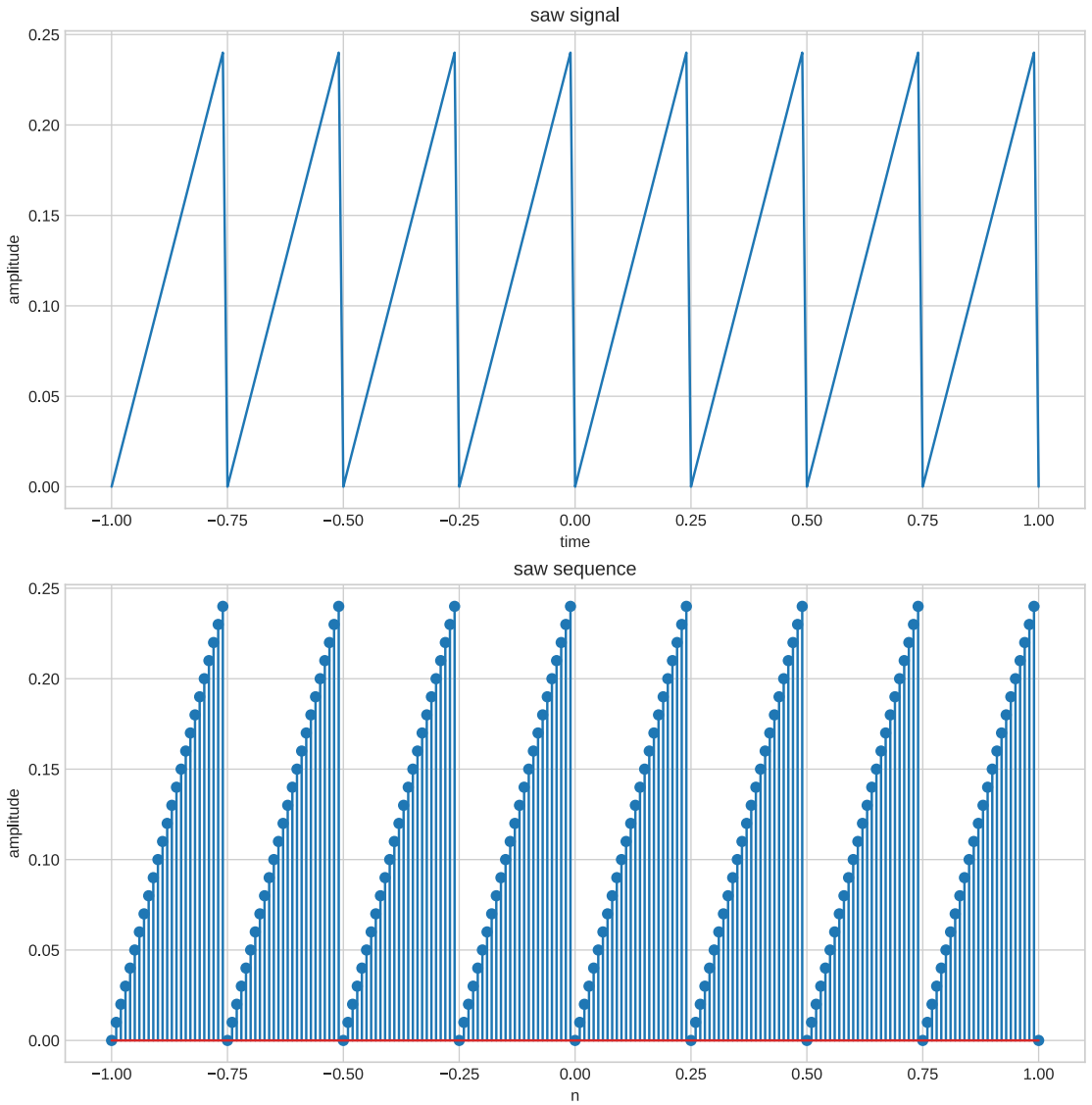


1.5 saw tooth signal and saw tooth sequence

[6]:
```
def saw(t,f):
    s = t.copy()
    T = 1/f
    s[s%T < T] = s[s%T < T]%(T)
    return s


f = 4 #Hz
fs = 100
t = np.linspace(-1,1,(fs*2)+1)
```

```
fig, ax = createSubplot(2)
saw_signal = saw(t,f)
ax[0].plot(t,saw_signal)
ax[0].set_xlabel('time')
ax[0].set_ylabel('amplitude')
ax[0].set_title('saw signal')

ax[1].stem(t,saw_signal)
ax[1].set_xlabel('n')
ax[1].set_ylabel('amplitude')
ax[1].set_title('saw sequence')
```

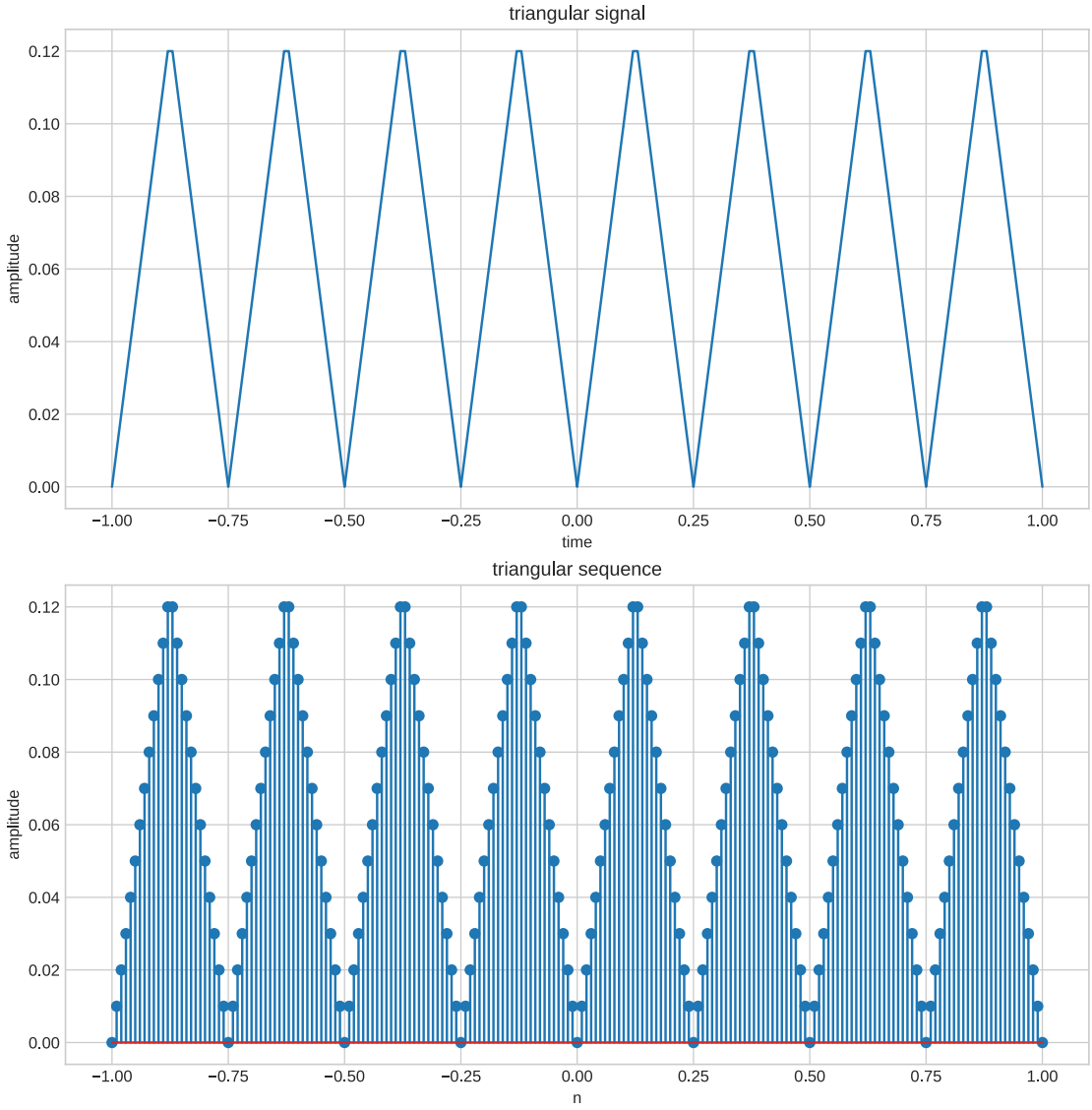
[6]: Text(0.5, 1.0, 'saw sequence')



1.6 triangular signal and triangular sequence

[7]:
```
def triangular(t,f):
    s = t.copy()
    T = 1/f
    s[s%T < T/2] = s[s%T < T/2]%(T/2)
    s[s%T > T/2] = (T/2) - s[s%T > T/2]%(T/2)
    return s


f = 4 #Hz
fs = 100
t = np.linspace(-1,1,(fs*2)+1)

fig, ax = createSubplot(2)
triangular_signal = triangular(t,f)
ax[0].plot(t,triangular_signal)
ax[0].set_xlabel('time')
ax[0].set_ylabel('amplitude')
ax[0].set_title('triangular signal')

ax[1].stem(t,triangular_signal)
ax[1].set_xlabel('n')
ax[1].set_ylabel('amplitude')
ax[1].set_title('triangular sequence')
```

[7]:
```
Text(0.5, 1.0, 'triangular sequence')
```

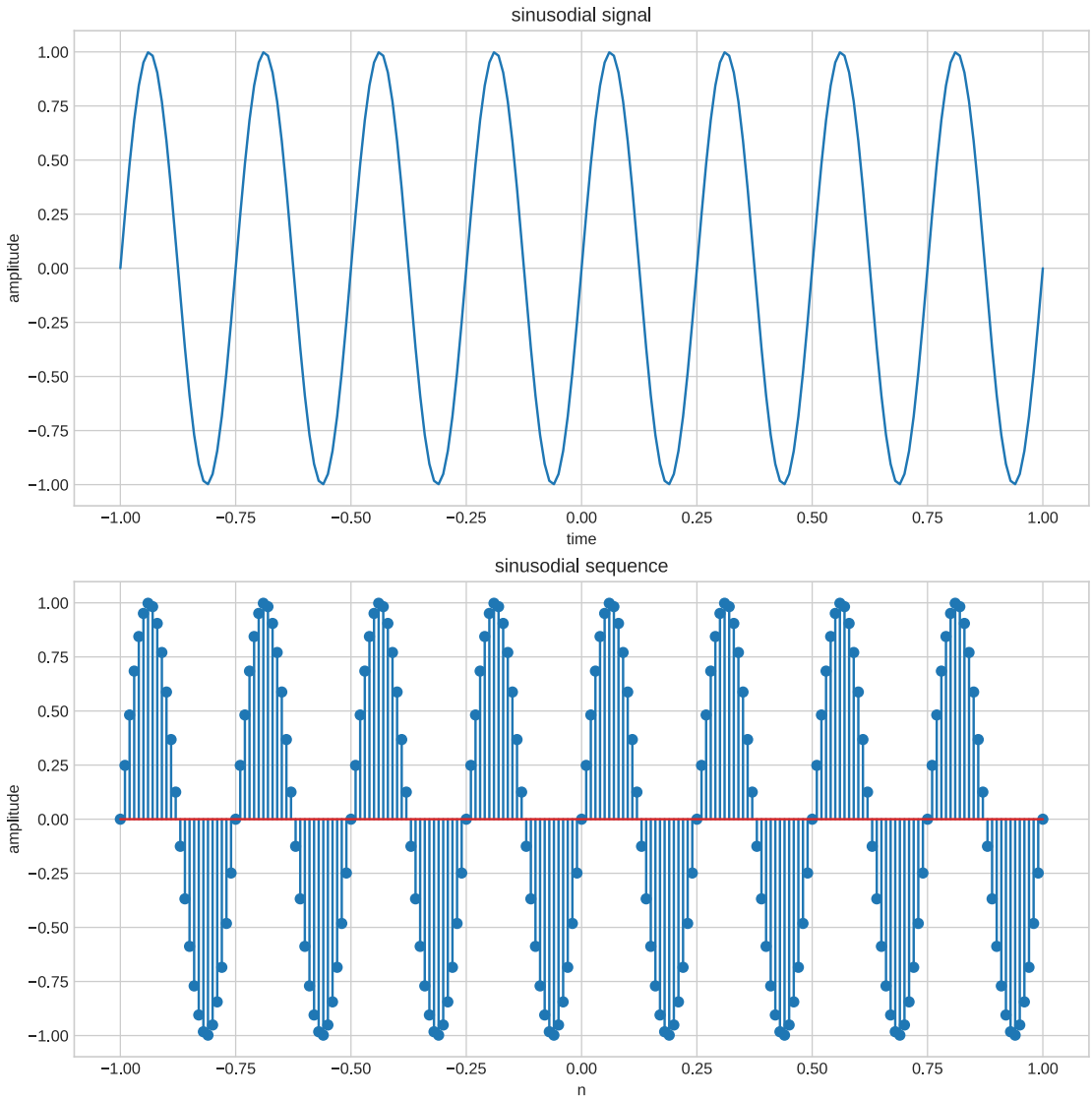


1.7 sinusoidal wave signal and sinusoidal wave sequence

[8]:
```
def sinusodial(t,f):
    s = t.copy()
    T = 1/f
    s = np.sin(t * pi * f * 2)
    return s


f = 4 #Hz
fs = 100
t = np.linspace(-1,1,(fs*2)+1)
```

```
fig, ax = createSubplot(2)
sinusodial_signal = sinusodial(t,f)
ax[0].plot(t,sinusodial_signal)
ax[0].set_xlabel('time')
ax[0].set_ylabel('amplitude')
ax[0].set_title('sinusodial signal')

ax[1].stem(t,sinusodial_signal)
ax[1].set_xlabel('n')
ax[1].set_ylabel('amplitude')
ax[1].set_title('sinusodial sequence')
```

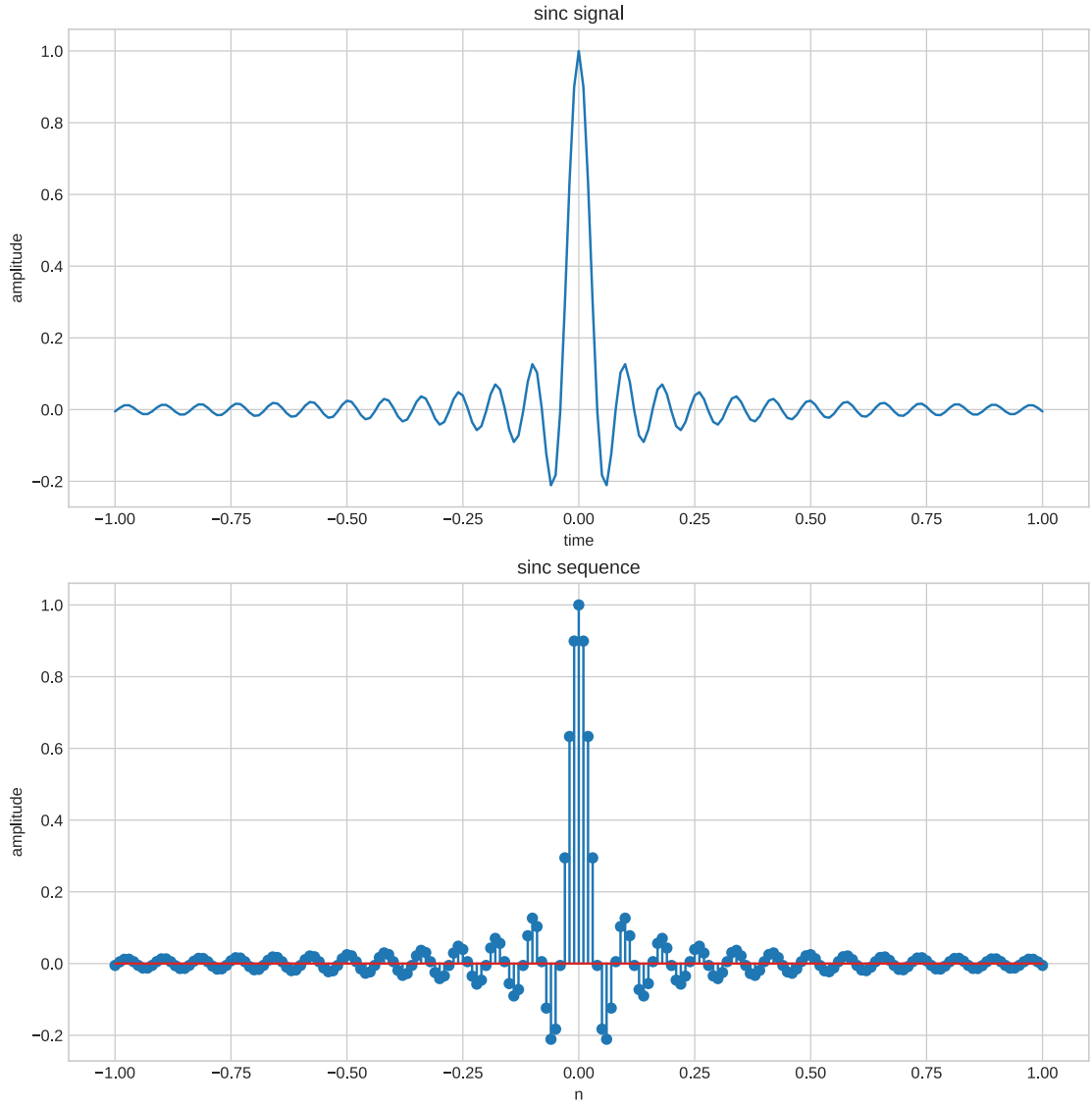
[8]: Text(0.5, 1.0, 'sinusodial sequence')



1.8 sinc signal and sinc sequence

[9]:
```
def sinc(t,f):
    s = t.copy()
    T = 1/f
    s = np.sinc(t * pi * f * 2)
    return s


f = 4 #Hz
fs = 100
t = np.linspace(-1,1,(fs*2)+1)

fig, ax = createSubplot(2)
sinc_signal = sinc(t,f)
ax[0].plot(t,sinc_signal)
ax[0].set_xlabel('time')
ax[0].set_ylabel('amplitude')
ax[0].set_title('sinc signal')

ax[1].stem(t,sinc_signal)
ax[1].set_xlabel('n')
ax[1].set_ylabel('amplitude')
ax[1].set_title('sinc sequence')
```

[9]:
```
Text(0.5, 1.0, 'sinc sequence')
```

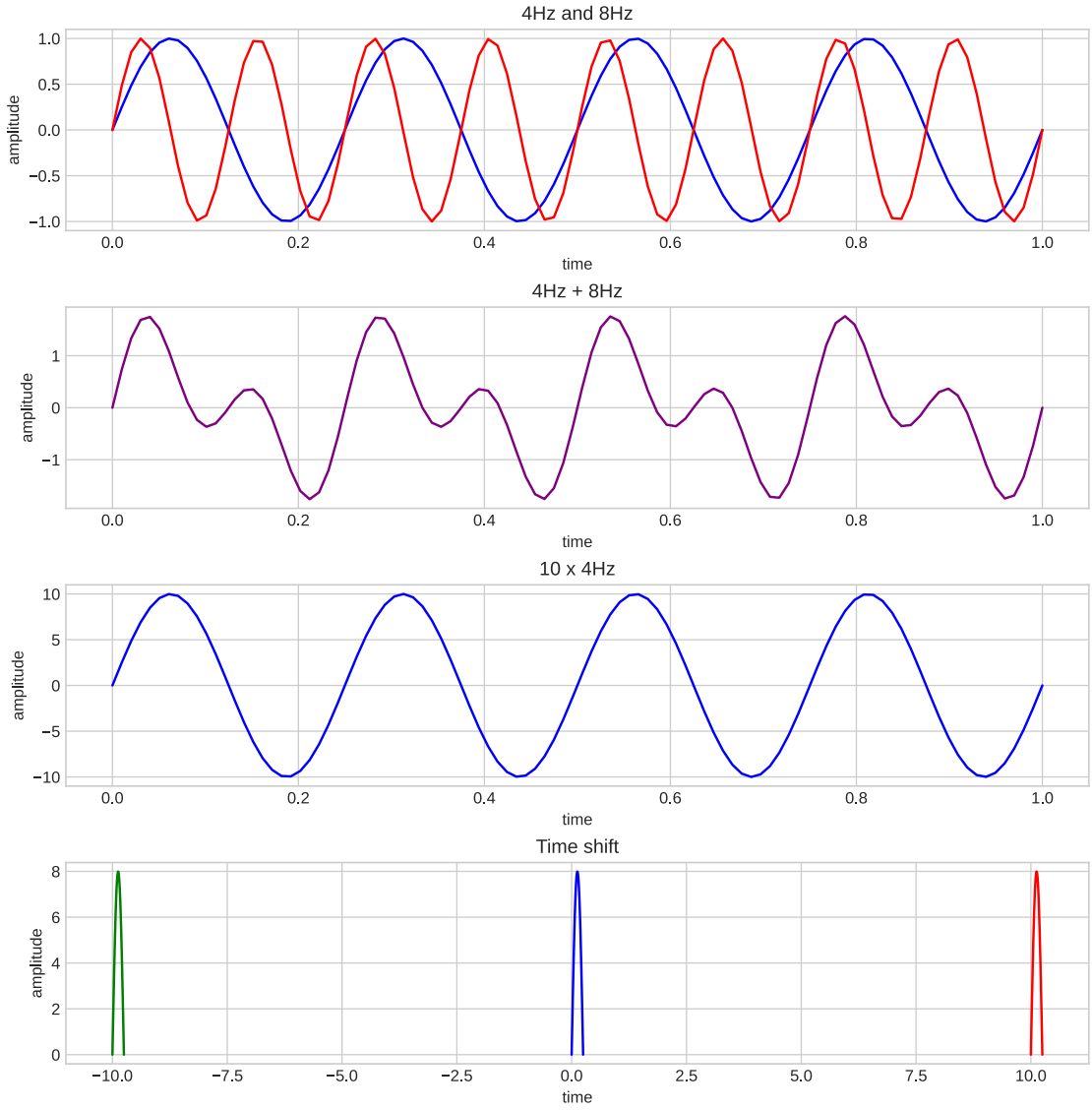


2 2 Operations on a continuous signal

[10]:
```
# 1. Generate two sine waves with frequency 4Hz and 8Hz from the below signal␣
 ↪in Matlab
# for 0 to 1s with 0.01 increments and plot both of them. A = Sin(2 f t)
fs = 100
t = np.linspace(0,1,fs)
s1 = np.sin(t * 2 * pi * 4) #4Hz
s2 = np.sin(t * 2 * pi * 8) #8Hz
fig, ax = createSubplot(4)
ax[0].plot(t,s1, color='blue')
```

```
ax[0].plot(t,s2, color='red')
ax[0].set_xlabel('time')
ax[0].set_ylabel('amplitude')
ax[0].set_title('4Hz and 8Hz')

# 2. Plot the addition and multiplication of these signals with each other.
ax[1].plot(t,s1 + s2, color='purple')
ax[1].set_xlabel('time')
ax[1].set_ylabel('amplitude')
ax[1].set_title('4Hz + 8Hz')

# 3. Plot the amplified signal for 4Hz if the scaling factor k is 10.
ax[2].plot(t,10*s1, color='blue')
ax[2].set_xlabel('time')
ax[2].set_ylabel('amplitude')
ax[2].set_title('10 x 4Hz')

# 4. Plot these time shifted signals for 0 to  with 0.01 increments with 2Hz.␣
↪A = 8sin(2 f t)B =
# 8sin(2 f(t + 10))C = 8sin(2 f(t - 10))
t = np.arange(0,0.26,0.01)
# print(t)
f = 2 #Hz
A = 8 * np.sin(2 * pi * f * t)
B = 8 * np.sin(2 * pi * f * (t+10))
C = 8 * np.sin(2 * pi * f * (t-10))
ax[3].plot(t,A, color='blue')
ax[3].plot(t+10,B, color='red')
ax[3].plot(t-10,C, color='green')
ax[3].set_xlabel('time')
ax[3].set_ylabel('amplitude')
ax[3].set_title('Time shift')
```

[10]: Text(0.5, 1.0, 'Time shift')

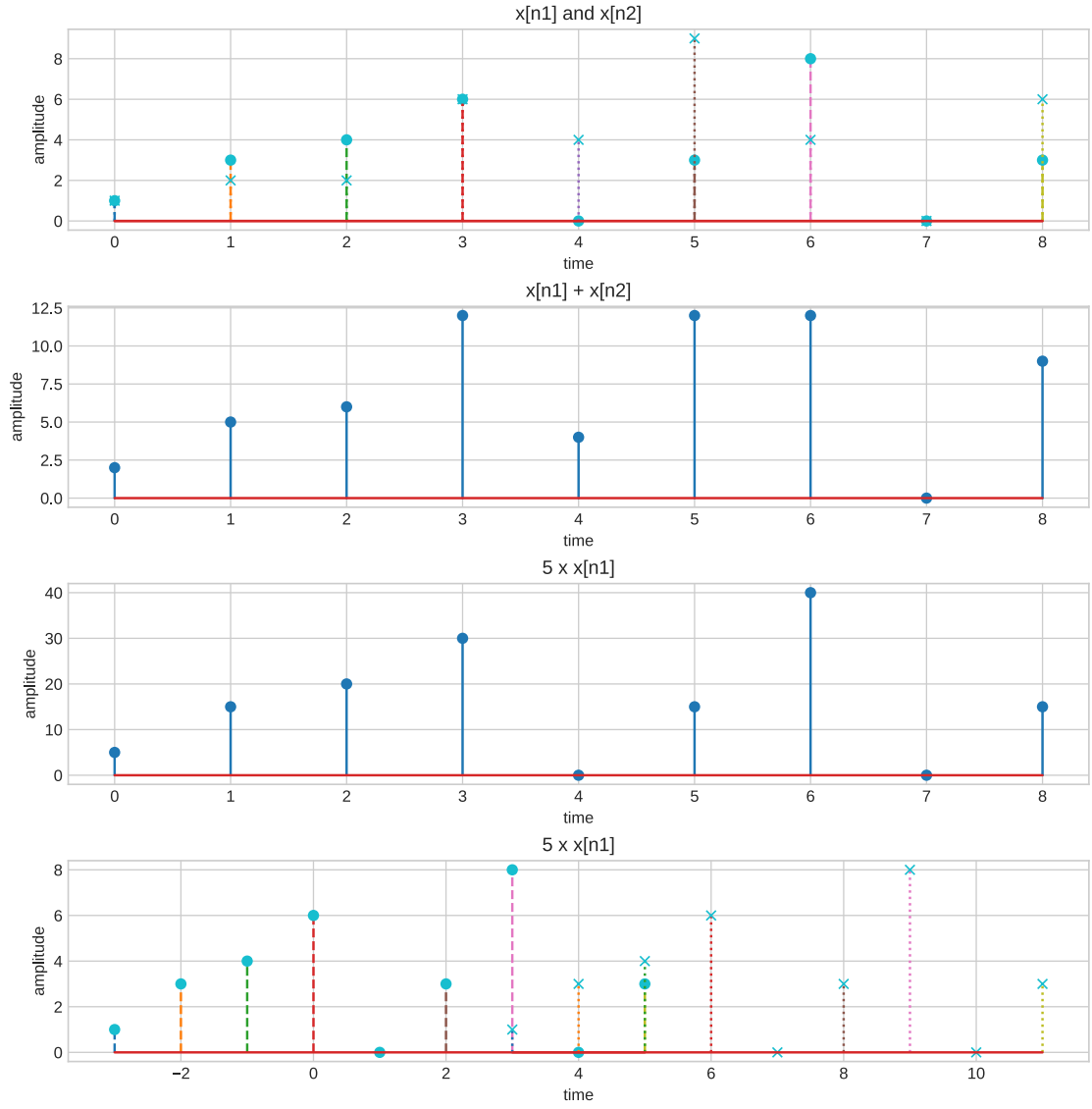


3 4.3 Operations on a discrete signal

[11]:
```
# 1. Plot x[n1]= 1 3 4 6 0 3 8 0 3 and x[n2]= 1 2 2 6 4 9 4 0 6 sequence.
fig, ax = createSubplot(4)

x1 = np.array([1,3,4,6,0,3,8,0,3])
x2 = np.array([1,2,2,6,4,9,4,0,6])
n = np.arange(0,len(x1))
ax[0].stem(n,x1,linefmt='--',markerfmt='o')
ax[0].stem(n,x2,linefmt=':',markerfmt='x')
ax[0].set_xlabel('time')
```

```
ax[0].set_ylabel('amplitude')
ax[0].set_title('x[n1] and x[n2]')

# 2. Plot the addition and multiplication of these signals with each other.
ax[1].stem(n,x1+x2)
ax[1].set_xlabel('time')
ax[1].set_ylabel('amplitude')
ax[1].set_title('x[n1] + x[n2]')

# 3. Plot the amplified signal for x[n1] if the scaling factor k is 5.
ax[2].stem(n,5 * x1)
ax[2].set_xlabel('time')
ax[2].set_ylabel('amplitude')
ax[2].set_title('5 x x[n1]')

# 4. Plot the time shifted signal x[n1 - 3] and x[n1+3].
ax[3].stem(n-3,x1,linefmt='--',markerfmt='o')
ax[3].stem(n+3,x1,linefmt=':',markerfmt='x')
ax[3].set_xlabel('time')
ax[3].set_ylabel('amplitude')
ax[3].set_title('5 x x[n1]')
```

[11]: `Text(0.5, 1.0, '5 x x[n1]')`



4 4.4 Energy and Power of a Signal

[12]:
```
# Write down a Matlab program to compute the energy and power of a discrete␣
↪signal taking the user input sequence. Eg - for sequence 1 3 5 6 you should␣
↪get the energy as 71 and power as 17.75

# Energy = sum of signal**2
# Power = Energy / num

def disEnergy(sig):
    return sig @ sig
```

```
def disPower(sig):
    energy = disEnergy(sig)
    return energy/(len(sig))

s = np.array([1,3,5,6])
print("Energy: ", disEnergy(s))
print("Power: ", disPower(s))
```

```
Energy:  71
Power:  17.75
```

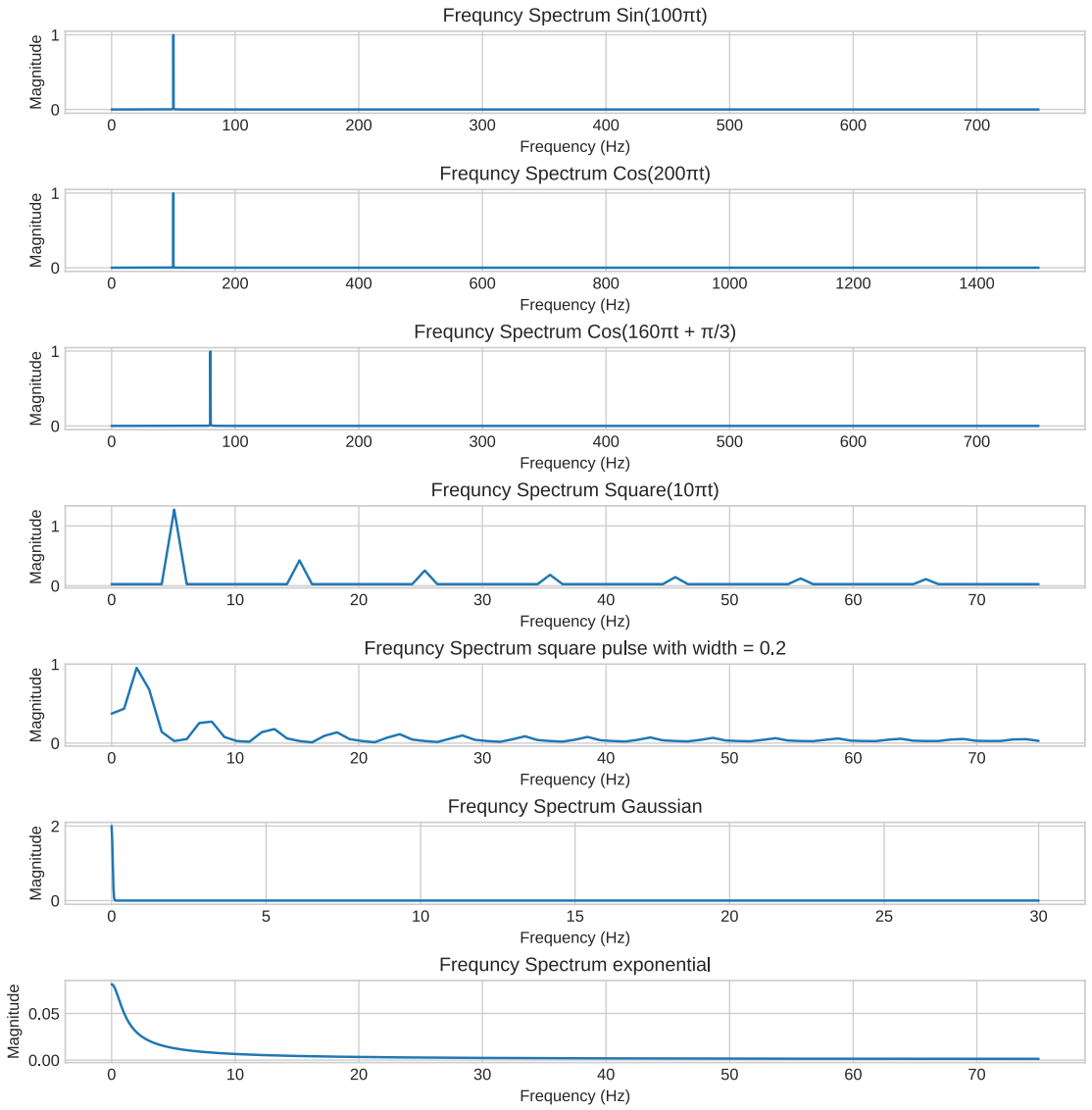
5 4.5 Magnitude spectrum in frequency domain

[13]:
```
from numpy.fft import fft
fig, ax = createSubplot(7)
# 1. Sin(100 t) with sampling frequency Fs=1500 Hz and for a duration of 10s
fs = 1500 # Hz
duration = 10 # second
sampling = fs*duration
t = np.linspace(0,duration,sampling)
s1 = np.sin(100*pi*t)
z1 = np.abs(fft(s1))[0:(sampling)//2]
f1 = np.linspace(0,fs/2, sampling//2)
ax[0].plot(f1,z1/(sampling/2))
ax[0].set_xlabel('Frequency (Hz)')
ax[0].set_ylabel('Magnitude')
ax[0].set_title('Frequncy Spectrum Sin(100 t)')

# 2. Cos(200 t) with sampling frequency Fs=3000 Hz and for a duration of 10s
fs = 3000 # Hz
duration = 10 # second
sampling = fs*duration
t = np.linspace(0,duration,sampling)
s1 = np.cos(200*pi*t)
z1 = np.abs(fft(s1))[0:(sampling)//2]
f1 = np.linspace(0,fs/2, sampling//2)
ax[1].plot(f1,z1/(sampling/2))
ax[1].set_xlabel('Frequency (Hz)')
ax[1].set_ylabel('Magnitude')
ax[1].set_title('Frequncy Spectrum Cos(200 t)')

# 3. Cos(160 t + /3) with sampling frequency Fs=1500 Hz and for a duration of␣
 ↪10s
fs = 1500 # Hz
duration = 10 # second
```

```
sampling = fs*duration
t = np.linspace(0,duration,sampling)
s1 = np.cos((160*pi*t) + (pi/3))
z1 = np.abs(fft(s1))[0:(sampling)//2]
f1 = np.linspace(0,fs/2, sampling//2)
ax[2].plot(f1,z1/(sampling/2))
ax[2].set_xlabel('Frequency (Hz)')
ax[2].set_ylabel('Magnitude')
ax[2].set_title('Frequncy Spectrum Cos(160 t + /3)')

# 4. Square(10 t) with sampling frequency Fs=150 Hz and for a duration of 1s
fs = 150 # Hz
duration = 1 # second
sampling = fs*duration
t = np.linspace(0,duration,sampling)
s1 = signal.square(10 * pi * t)
z1 = np.abs(fft(s1))[0:(sampling)//2]
f1 = np.linspace(0,fs/2, sampling//2)
ax[3].plot(f1,z1/(sampling/2))
ax[3].set_xlabel('Frequency (Hz)')
ax[3].set_ylabel('Magnitude')
ax[3].set_title('Frequncy Spectrum Square(10 t)')

# 5. A square pulse with width = 0.2 and Fs = 150Hz
# width = T/2 = 0.2 => T = 4/10
# f = 2.5
fs = 150 # Hz
duration = 1 # second
sampling = fs*duration
t = np.linspace(0,duration,sampling)
s1 = signal.square(5 * pi * t)
z1 = np.abs(fft(s1))[0:(sampling)//2]
f1 = np.linspace(0,fs/2, sampling//2)
ax[4].plot(f1,z1/(sampling/2))
ax[4].set_xlabel('Frequency (Hz)')
ax[4].set_ylabel('Magnitude')
ax[4].set_title('Frequncy Spectrum square pulse with width = 0.2')

# 6. A Gaussian pulse Fs = 60Hz
fs = 60 # Hz
duration = 50 # second
sampling = fs*duration
t = np.linspace(-50,duration,sampling)
s1 = []
for i in t:
    s1.append((1/np.sqrt(2*pi*0.01)) * np.exp((-1 * (i * i))/2*0.01  ))
z1 = np.abs(fft(s1))[0:(sampling)//2]
```

```
f1 = np.linspace(0,fs/2, sampling//2)
# ax[5].plot(t,s1)
ax[5].plot(f1,z1/(sampling/2))
ax[5].set_xlabel('Frequency (Hz)')
ax[5].set_ylabel('Magnitude')
ax[5].set_title('Frequncy Spectrum Gaussian')

# 7. An exponential function with x = 2e^-5t and Fs= 150Hz
fs = 150 # Hz
duration = 10 # second
sampling = fs*duration
t = np.linspace(0,duration,sampling)
s1 = []
for i in t:
    s1.append( 2 * np.exp(-5*i) )
z1 = np.abs(fft(s1))[0:(sampling)//2]
f1 = np.linspace(0,fs/2, sampling//2)
# ax[6].plot(t,s1)
ax[6].plot(f1,z1/(sampling/2))
ax[6].set_xlabel('Frequency (Hz)')
ax[6].set_ylabel('Magnitude')
ax[6].set_title('Frequncy Spectrum exponential')
```

[13]: Text(0.5, 1.0, 'Frequncy Spectrum exponential')



6 Identify string

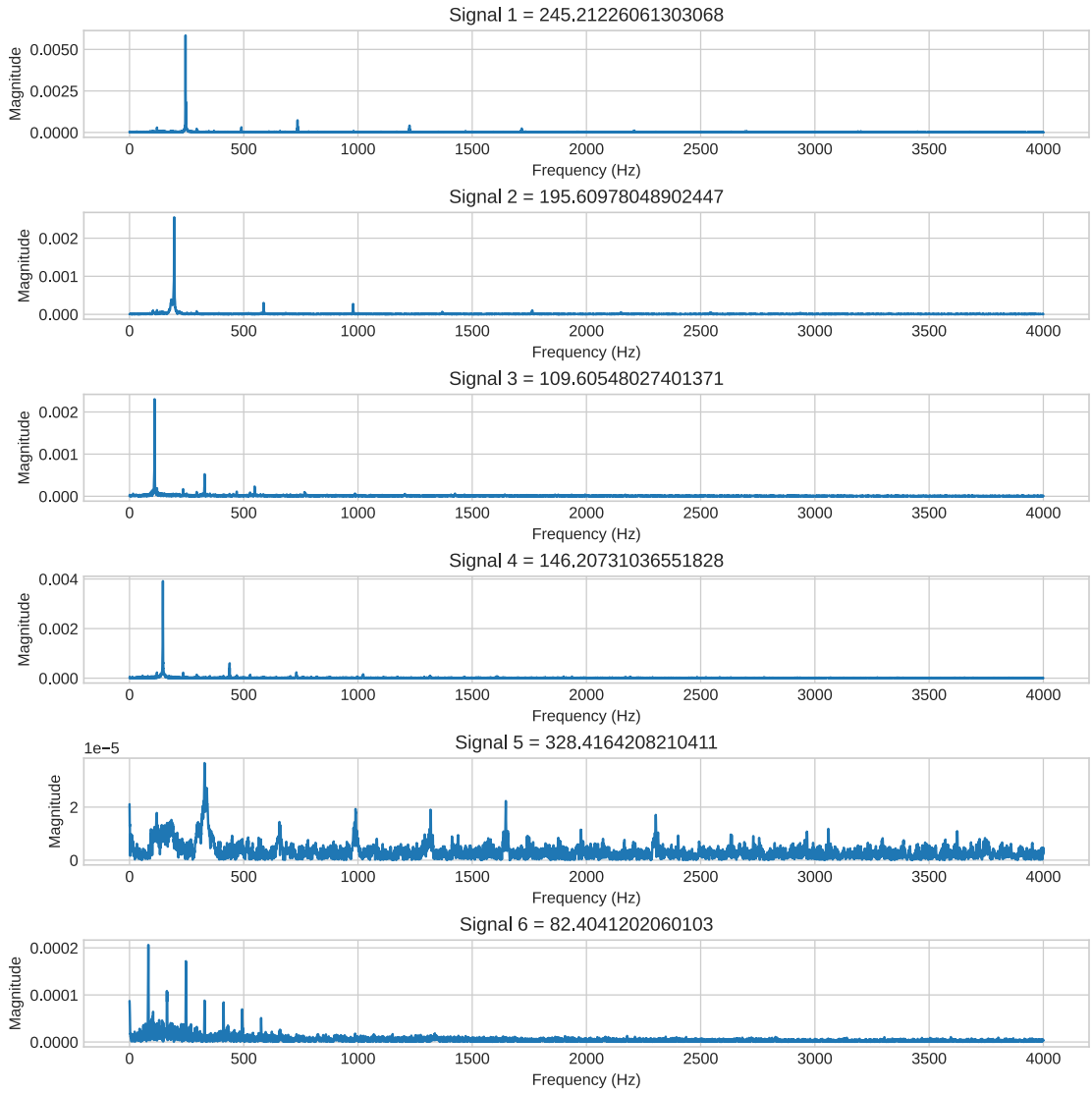
- high E – 330 Hz
- B – 247 Hz
- G – 196 Hz
- D – 147 Hz
- A – 110 Hz
- low E – 82 Hz

[14]:
```
# Note: Each signal is acquired by playing single string from guitar and then␣
↪recording through microphone using a sampling rate of 8000 samples/sec and␣
↪for the duration of 5 seconds
```

```
fs = 8000 # Hz
duration = 5 # s

fig, ax = createSubplot(6)

# signal1
signal1 = np.loadtxt('signal1.txt', dtype='double')
sampling = fs * duration
z1 = np.abs(fft(signal1))[0:(sampling)//2]
f1 = np.linspace(0,fs/2, sampling//2)
ax[0].plot(f1,z1/(sampling/2))
ax[0].set_xlabel('Frequency (Hz)')
ax[0].set_ylabel('Magnitude')
ax[0].set_title(f'Signal 1 = {f1[np.argmax(z1)]}')

signal2 = np.loadtxt('signal2.txt', dtype='double')
sampling = fs * duration
z2 = np.abs(fft(signal2))[0:(sampling)//2]
f2 = np.linspace(0,fs/2, sampling//2)
ax[1].plot(f2,z2/(sampling/2))
ax[1].set_xlabel('Frequency (Hz)')
ax[1].set_ylabel('Magnitude')
ax[1].set_title(f'Signal 2 = {f2[np.argmax(z2)]}')

signal3 = np.loadtxt('signal3.txt', dtype='double')
sampling = fs * duration
z3 = np.abs(fft(signal3))[0:(sampling)//2]
f3 = np.linspace(0,fs/2, sampling//2)
ax[2].plot(f3,z3/(sampling/2))
ax[2].set_xlabel('Frequency (Hz)')
ax[2].set_ylabel('Magnitude')
ax[2].set_title(f'Signal 3 = {f3[np.argmax(z3)]}')

signal4 = np.loadtxt('signal4.txt', dtype='double')
sampling = fs * duration
z4 = np.abs(fft(signal4))[0:(sampling)//2]
f4 = np.linspace(0,fs/2, sampling//2)
ax[3].plot(f4,z4/(sampling/2))
ax[3].set_xlabel('Frequency (Hz)')
ax[3].set_ylabel('Magnitude')
ax[3].set_title(f'Signal 4 = {f4[np.argmax(z4)]}')

signal5 = np.loadtxt('signal5.txt', dtype='double')
sampling = fs * duration
z5 = np.abs(fft(signal5))[0:(sampling)//2]
f5 = np.linspace(0,fs/2, sampling//2)
ax[4].plot(f5,z5/(sampling/2))
```

```
ax[4].set_xlabel('Frequency (Hz)')
ax[4].set_ylabel('Magnitude')
ax[4].set_title(f'Signal 5 = {f5[np.argmax(z5)]}')

signal6 = np.loadtxt('signal6.txt', dtype='double')
sampling = fs * duration
z6 = np.abs(fft(signal6))[0:(sampling)//2]
f6 = np.linspace(0,fs/2, sampling//2)
ax[5].plot(f6,z6/(sampling/2))
ax[5].set_xlabel('Frequency (Hz)')
ax[5].set_ylabel('Magnitude')
ax[5].set_title(f'Signal 6 = {f6[np.argmax(z6)]}')
```

[14]: `Text(0.5, 1.0, 'Signal 6 = 82.4041202060103')`



- Signal 1 is B
- Signal 2 is G
- Signal 3 is A
- Signal 4 is D
- Signal 5 is high E
- Signal 6 is low E