Experiment No 1 Performance of Waveform Coding Using PCM

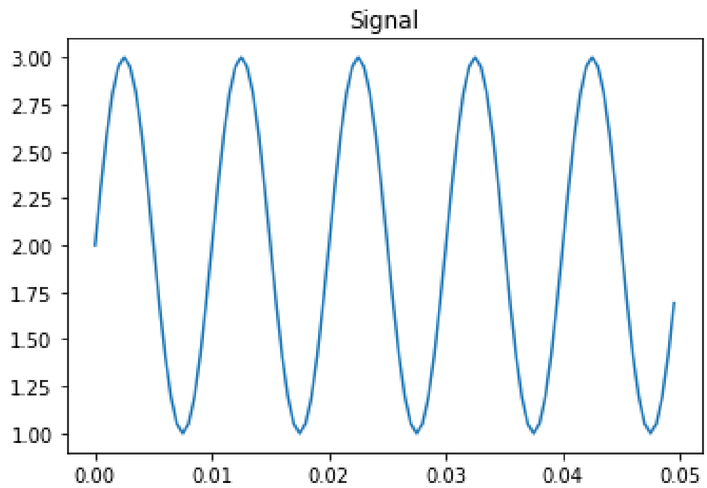
VIDYA SUNIL\ S6 ECE\ TVE19EC061
College of Engineering Trivandrum

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1. Generate a sinusoidal waveform with a DC offset so that it takes only positive amplitude value.

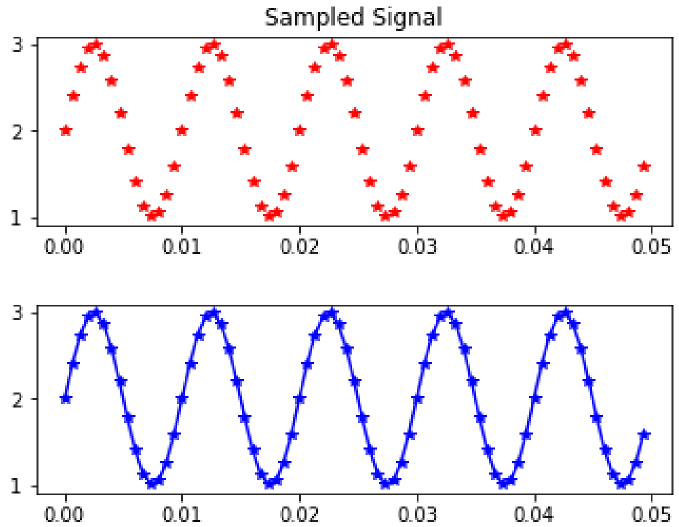
In [2]:
```
time = np.arange(0,0.05,0.0005) #Time -TVE19EC061
frequency_message=int(input("Enter frequency of sinusoidal waveform: "
)) #frequency -TVE19EC061
dc_offset = 2 #DC offset -TVE19EC061
signal = np.sin(2*np.pi*frequency_message*time) + dc_offset
plt.plot(time,signal)
plt.title("Signal")
plt.show()# -TVE19EC061
```

```
Enter frequency of sinusoidal waveform: 100
```



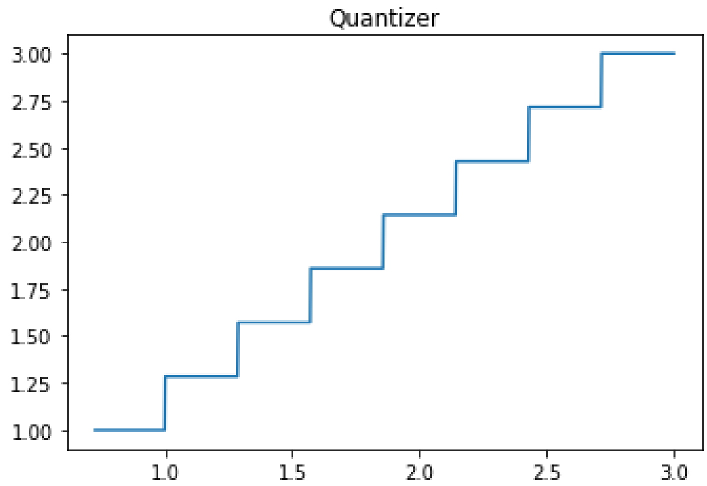
1. Sample and quantize the signal using an uniform quantizer with number of representation levels L. Vary L. Represent each value using decimal to binary encoder.

In [3]:
```
# Sampling # -TVE19EC061
#f_s = int(input("Enter Sampling Frequency:"))
frequency_sample = 15*frequency_message
sampling_time = np.arange(0,0.05,1/frequency_sample)
sampled_signal = dc_offset + np.sin(2*np.pi*frequency_message*sampling_
time)# -TVE19EC061
plt.subplot(2,1,1)
plt.plot(sampling_time,sampled_signal,"r*") # -TVE19EC061
plt.title("Sampled Signal")
plt.show()
plt.subplot(2,1,2)
plt.plot(sampling_time,sampled_signal,"b*-")
plt.show() # -TVE19EC061
```

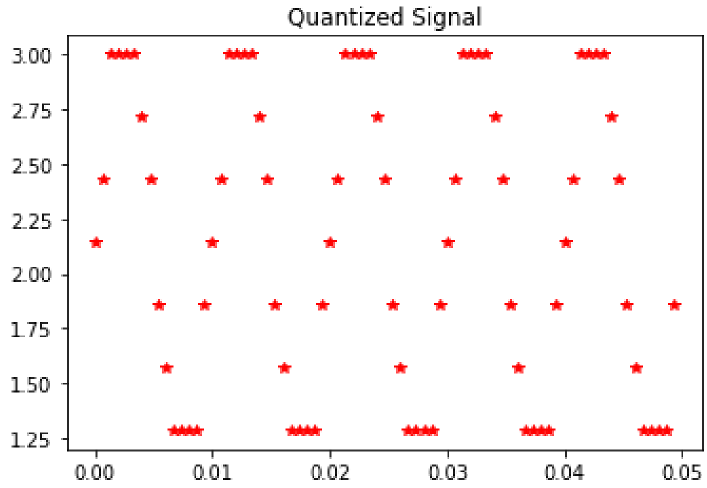


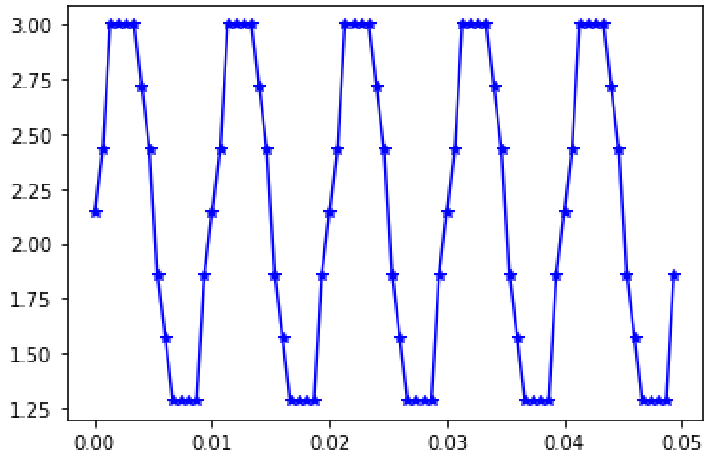
In [4]:
```
# Uniform Quantizer # -TVE19EC061
L = 8 # No.of Quantization Levels
signal_min = round(min(signal))
signal_max = round(max(signal))
Quantization_levels = np.linspace(signal_min,signal_max,L)# -TVE19EC061
print("Quantization Levels:",Quantization_levels)
q_level=[]
# -TVE19EC061
for i in np.linspace(0.725,3,1000):
    for j in Quantization_levels:
        if i <= j:
            q_level.append(j)
            break
plt.plot(np.linspace(0.725,3,1000),q_level)
plt.title("Quantizer")
plt.show() #  -TVE19EC061
```

```
Quantization Levels: [1.         1.28571429 1.57142857 1.85714286 2.142
85714 2.42857143
 2.71428571 3.        ]
```



In [5]:
```
# Quantizing with 8 Quantization Levels # -TVE19EC061

L = 8 # No.of Quantization Levels
signal_min = round(min(signal))
signal_max = round(max(signal))
Quantization_levels = np.linspace(signal_min,signal_max,L)# -TVE19EC061

quantized_signal = [] # -TVE19EC061
for i in sampled_signal:
    for j in Quantization_levels:
        if i <= j:
            quantized_signal.append(j)
            break

plt.subplot(1,1,1)
plt.plot(sampling_time,quantized_signal,"r*")
plt.title("Quantized Signal")
plt.show() # -TVE19EC061
plt.subplot(1,1,1)
plt.plot(sampling_time,quantized_signal,"b*-")
plt.show() # -TVE19EC061
```



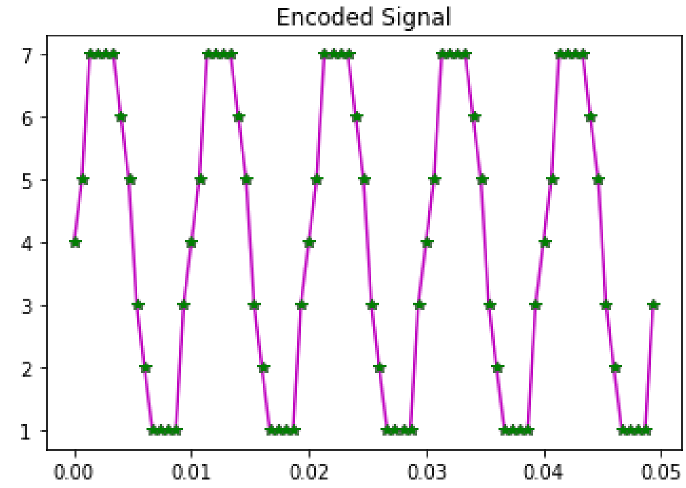


In [6]:

```
# Encoding with 8 quantization Levels # -TVE19EC061
#quantization levels # -TVE19EC061
L = 8 # No.of Quantization Levels
signal_min = round(min(signal))
signal_max = round(max(signal))
Quantization_levels = np.linspace(signal_min,signal_max,L)

#Quantization Levels Mapping to decimal # -TVE19EC061
count = 0
quantization_levels_mapping = {}
for i in Quantization_levels:
    quantization_levels_mapping[i] = quantization_levels_mapping.get(i,
count) # -TVE19EC061
    count+=1

# Decimal to binary bits # -TVE19EC061
binary_code ={}
bit_no = int(np.log2(L))
for i in range(L):# -TVE19EC061
    val = bin(i).replace("0b", "")
    if len(val) < bit_no:
        f_bit =""
        for j in range(bit_no-len(val)):
            f_bit += "0"
        val = f_bit + val
    binary_code[i] = binary_code.get(i,val)

print("Quantization Levels Mapping:",quantization_levels_mapping) # -TV
E19EC061
print("\nBinary Code:",binary_code) # -TVE19EC061

# encoded signal # -TVE19EC061
encoded_signal=[]
for k in quantized_signal:
    encoded_signal.append(quantization_levels_mapping[k])
plt.plot(sampling_time, encoded_signal, "m*-",sampling_time, encoded_si
gnal, "g*") # -TVE19EC061
plt.title("Encoded Signal")
plt.show() # -TVE19EC061

# Binary Coding # -TVE19EC061
binary_coded_signal = []
for k in encoded_signal:
    binary_coded_signal.append(binary_code[k])
print("Binary Coded Signal:", binary_coded_signal)
# -TVE19EC061
```

Quantization Levels Mapping: {1.0: 0, 1.2857142857142856: 1, 1.5714285714285714: 2, 1.8571428571428572: 3, 2.142857142857143: 4, 2.4285714285714284: 5, 2.7142857142857144: 6, 3.0: 7}

Binary Code: {0: '000', 1: '001', 2: '010', 3: '011', 4: '100', 5: '101', 6: '110', 7: '111'}

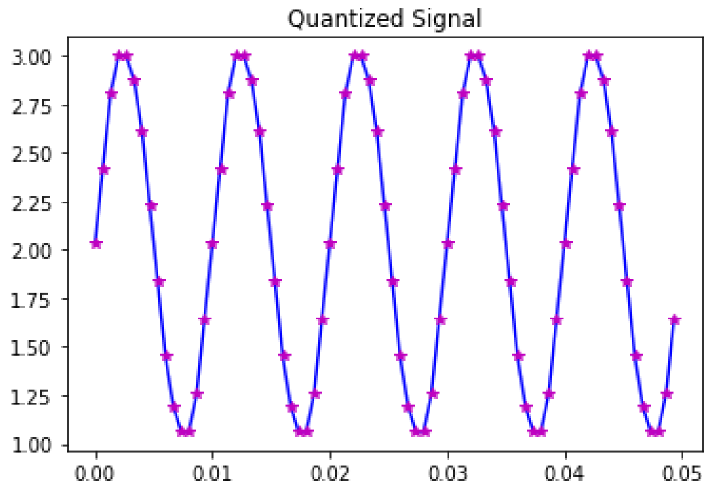


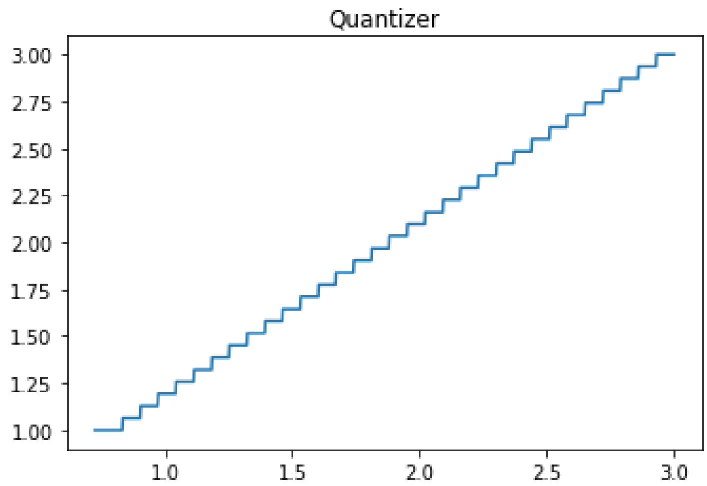
Binary Coded Signal: ['100', '101', '111', '111', '111', '111', '110', '101', '011', '010', '001', '001', '001', '001', '011', '100', '101', '111', '111', '111', '111', '110', '101', '011', '010', '001', '001', '001', '001', '011', '100', '101', '111', '111', '111', '111', '110', '101', '011', '010', '001', '001', '001', '001', '011', '100', '101', '111', '111', '111', '111', '110', '101', '011', '010', '001', '001', '001', '001', '011', '100', '101', '111', '111', '111', '111', '110', '101', '011', '010', '001', '001', '001', '001', '011']

In [7]:

```
# Quantizing with L levels # -TVE19EC061

L = int(input("Enter no.of quantization levels: "))
signal_min = round(min(signal))
signal_max = round(max(signal))
Quantization_levels1 = np.linspace(signal_min,signal_max,L) # -TVE19EC0
61
quantized_signal1 = []
for i in sampled_signal:
    for j in Quantization_levels1:
        if i <= j:
            quantized_signal1.append(j)
            break
plt.subplot(1,1,1)
plt.plot(sampling_time,quantized_signal1,"b",sampling_time,quantized_si
gnal1,"m*")
plt.title("Quantized Signal")
plt.show() # -TVE19EC061

#Quantizer # -TVE19EC061
q_level=[]
for i in np.linspace(0.9,3,1000):
    for j in Quantization_levels1:
        if i <= j:
            q_level.append(j)
            break
plt.subplot(1,1,1)
plt.plot(np.linspace(0.725,3,1000),q_level) # -TVE19EC061
plt.title("Quantizer")
plt.show() # -TVE19EC061
```

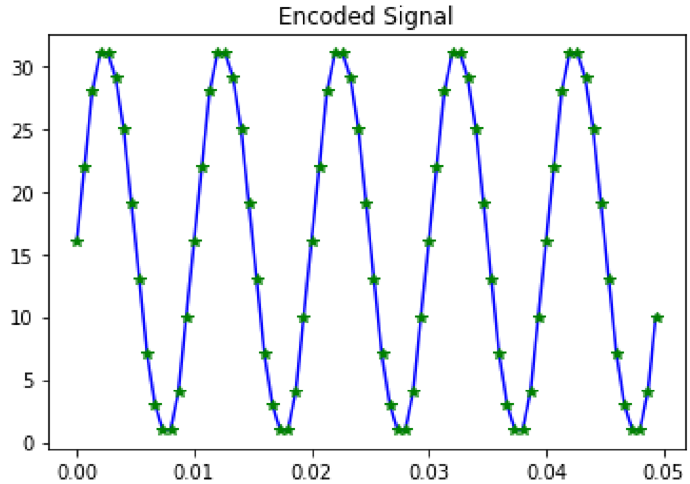
```
Enter no.of quantization levels: 32
```





In [8]:
```
# Encoding with L quantization Levels # -TVE19EC061
#quantization levels
#L = 8
signal_min = round(min(signal))
signal_max = round(max(signal))
Quantization_levels1 = np.linspace(signal_min,signal_max,L) # -TVE19EC0
61

#Quantization Levels Mapping to decimal # -TVE19EC061
count = 0
quantization_levels_mapping1 = {}
for i in Quantization_levels1:
    quantization_levels_mapping1[i] = quantization_levels_mapping1.get(
i,count)# -TVE19EC061
    count+=1

# Decimal to binary bits # -TVE19EC061
binary_code1 ={}
bit_no = int(np.log2(L))
# -TVE19EC061
for i in range(L):
    val = bin(i).replace("0b", "")
    if len(val) < bit_no:
        f_bit =""
        for j in range(bit_no-len(val)):
            f_bit += "0"
        val = f_bit + val
    binary_code1[i] = binary_code1.get(i,val)# -TVE19EC061

print("Quantization Levels Mapping:",quantization_levels_mapping1) # -T
VE19EC061
print("\nBinary Code:",binary_code1)

# encoded signal # -TVE19EC061
encoded_signal1=[]
for k in quantized_signal1:
    encoded_signal1.append(quantization_levels_mapping1[k])
plt.plot(sampling_time,encoded_signal1,"b",sampling_time,encoded_signal
1,"g*") # -TVE19EC061
plt.title("Encoded Signal")
plt.show()

# Binary Coding  # -TVE19EC061
binary_coded_signal1 = []
for k in encoded_signal1:
    binary_coded_signal1.append(binary_code1[k])
print("Binary Coded Signal:",binary_coded_signal1) # -TVE19EC061
```

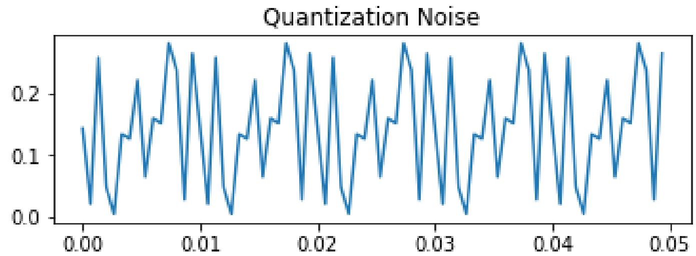
```
Quantization Levels Mapping: {1.0: 0, 1.064516129032258: 1, 1.129032258
064516: 2, 1.1935483870967742: 3, 1.2580645161290323: 4, 1.322580645161
2903: 5, 1.3870967741935485: 6, 1.4516129032258065: 7, 1.51612903225806
45: 8, 1.5806451612903225: 9, 1.6451612903225805: 10, 1.709677419354838
7: 11, 1.7741935483870968: 12, 1.8387096774193548: 13, 1.90322580645161
3: 14, 1.967741935483871: 15, 2.032258064516129: 16, 2.096774193548387:
17, 2.161290322580645: 18, 2.225806451612903: 19, 2.290322580645161: 2
0, 2.354838709677419: 21, 2.4193548387096775: 22, 2.4838709677419355: 2
3, 2.5483870967741935: 24, 2.6129032258064515: 25, 2.6774193548387095:
26, 2.741935483870968: 27, 2.806451612903226: 28, 2.870967741935484: 2
9, 2.935483870967742: 30, 3.0: 31}

Binary Code: {0: '00000', 1: '00001', 2: '00010', 3: '00011', 4: '0010
0', 5: '00101', 6: '00110', 7: '00111', 8: '01000', 9: '01001', 10: '01
010', 11: '01011', 12: '01100', 13: '01101', 14: '01110', 15: '01111',
16: '10000', 17: '10001', 18: '10010', 19: '10011', 20: '10100', 21: '1
0101', 22: '10110', 23: '10111', 24: '11000', 25: '11001', 26: '11010',
27: '11011', 28: '11100', 29: '11101', 30: '11110', 31: '11111'}
```

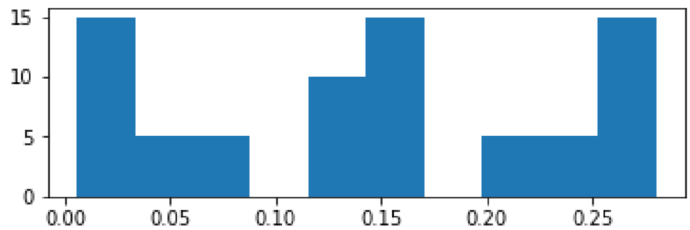



```
Binary Coded Signal: ['10000', '10110', '11100', '11111', '11111', '111
01', '11001', '10011', '01101', '00111', '00011', '00001', '00001', '00
100', '01010', '10000', '10110', '11100', '11111', '11111', '11101', '1
1001', '10011', '01101', '00111', '00011', '00001', '00001', '00100',
'01010', '10000', '10110', '11100', '11111', '11111', '11101', '11001',
'10011', '01101', '00111', '00011', '00001', '00001', '00100', '01010',
'10000', '10110', '11100', '11111', '11111', '11101', '11001', '10011',
'01101', '00111', '00011', '00001', '00001', '00100', '01010', '10000',
'10110', '11100', '11111', '11111', '11101', '11001', '10011', '01101',
'00111', '00011', '00001', '00001', '00100', '01010']
```

1. Compute the signal-to-noise ratio in dB

In [9]:
```
# Quantization Noise # -TVE19EC061
# L = 8
quantization_noise = quantized_signal-sampled_signal # -TVE19EC061
plt.subplot(2,1,1)
plt.plot(sampling_time,quantization_noise)
plt.title("Quantization Noise")
plt.show() # -TVE19EC061
plt.subplot(2,1,2)
plt.hist(quantization_noise)
plt.show() # -TVE19EC061
```





In [10]:
```
#SNR  # -TVE19EC061
def power(s):
    p = 0
    for i in s:
        p += i**2
    P = p/len(s)
    return P
# -TVE19EC061
p_signal = power(signal)
p_noise = power(quantization_noise) # -TVE19EC061
snr = p_signal/p_noise
snr_db = 20*np.log10(snr)
print("Signal-to-Noise ratio in dB: ", snr_db) # -TVE19EC061
```

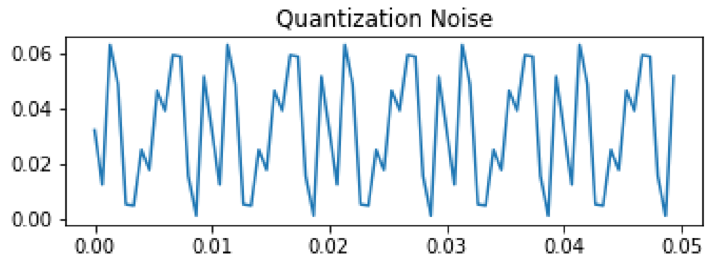
```
Signal-to-Noise ratio in dB:  43.92496604223551
```

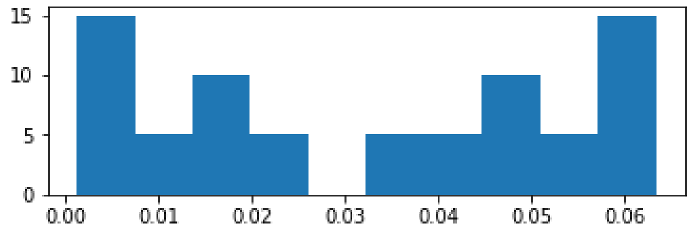
In [11]:
```
# SNR by equation # -TVE19EC061
# R = int(np.log2(L)) no.of Bits per sample # -TVE19EC061
s_min = round(min(signal))
s_max = round(max(signal))
L = 8
step_size = (s_max-s_min)/L
power_noise = (step_size**2)/3
power_signal = power(signal)
snr = power_signal/power_noise # -TVE19EC061
snr_db = 20*np.log10(snr)
print("Signal-to-Noise ratio in dB: ", snr_db) # -TVE19EC061
```

```
Signal-to-Noise ratio in dB:  46.68907502301862
```

In [12]:

```
# Quantization Noise # -TVE19EC061
signal = np.sin(2*np.pi*frequency_message*time) + dc_offset
s_min = round(min(signal))
s_max = round(max(signal))
L = 32
quantization_noise1 = quantized_signal1-sampled_signal # -TVE19EC061
plt.subplot(2,1,1)
plt.plot(sampling_time,quantization_noise1) # -TVE19EC061
plt.title("Quantization Noise")
plt.show() # -TVE19EC061
plt.subplot(2,1,2)
plt.hist(quantization_noise1)
plt.show()
#SNR # -TVE19EC061
p_signal = power(signal)
p_noise = power(quantization_noise1) # -TVE19EC061
snr = p_signal/p_noise
snr_db = 20*np.log10(snr)
print("Signal-to-Noise ratio in dB: ", snr_db) # -TVE19EC061
```






```
Signal-to-Noise ratio in dB:  69.6175465986973
```

In [13]:

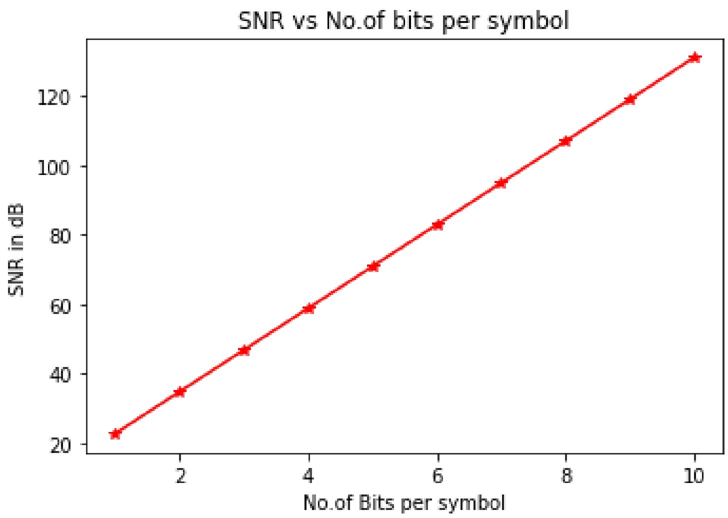
```
# SNR by equation # -TVE19EC061
# R = int(np.log2(L)) no.of Bits per sample # -TVE19EC061
s_min = round(min(signal))
s_max = round(max(signal))
L = 32
step_size = (s_max-s_min)/L
power_noise = (step_size**2)/3
power_signal = power(signal)
snr = power_signal/power_noise # -TVE19EC061
snr_db = 20*np.log10(snr)
print("Signal-to-Noise ratio in dB: ", snr_db) # -TVE19EC061
```

```
Signal-to-Noise ratio in dB:  70.77147467613712
```

1. Plot the SNR versus number of bits per symbol. Observe that the SNR increases linearly.

In [14]:
```
# -TVE19EC061
def power(s):
    p = 0
    for i in s:
        p += i**2
    P = p/len(s)
    return P
time = np.arange(0,0.05,0.0005) #Time # -TVE19EC061
signal = np.sin(2*np.pi*frequency_message*time) + 2  # -TVE19EC061
s_min = round(min(signal))
s_max = round(max(signal))
power_signal = power(signal)
snr_db=[] # -TVE19EC061
for i in range(1,11):
    R = i
    L = 2**R
    step_size = (s_max-s_min)/L
    power_noise = (step_size**2)/3
    snr = power_signal/power_noise # -TVE19EC061
    snr_db.append(20*np.log10(snr))

plt.plot(range(1,11),snr_db,"r*-")
plt.xlabel("No.of Bits per symbol") # -TVE19EC061
plt.ylabel("SNR in dB")
plt.title("SNR vs No.of bits per symbol") # -TVE19EC061
```

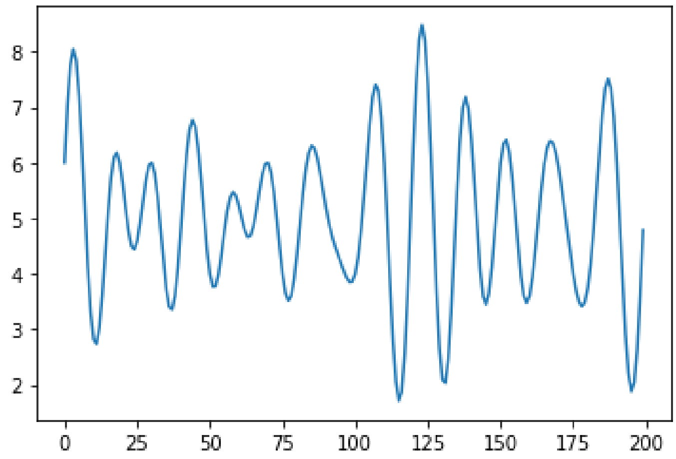
Out[14]:
```
Text(0.5, 1.0, 'SNR vs No.of bits per symbol')
```



Quantization of a signal

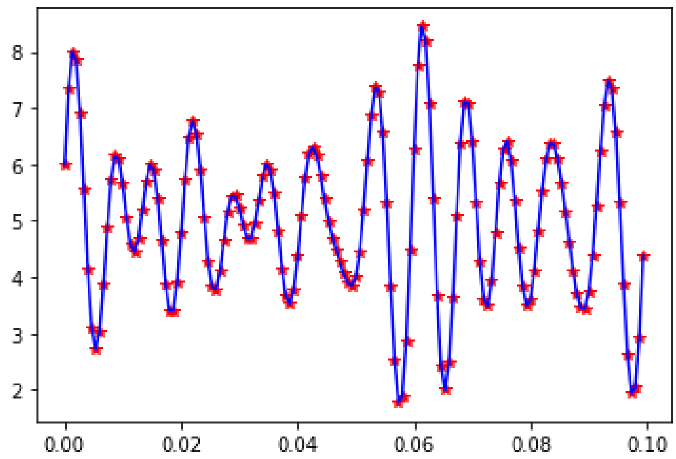
In [15]:
```
time = np.arange(0,0.1,0.0005) #Time # -TVE19EC061
message_signal = np.sin(2*np.pi*100*time) + np.sin(2*np.pi*120*time) +
np.sin(2*np.pi*150*time)+ np.cos(2*np.pi*130*time)+5
plt.plot(message_signal) # -TVE19EC061
```

Out[15]: [<matplotlib.lines.Line2D at 0x7fd25ffb36a0>]



In [16]:
```
sampling_frequency = 10*150 # -TVE19EC061
sampling_time2 = np.arange(0,0.1,1/sampling_frequency)
sampled_signal2 = np.sin(2*np.pi*100*sampling_time2) + np.sin(2*np.pi*1
20*sampling_time2) + np.sin(2*np.pi*150*sampling_time2)+ np.cos(2*np.pi
*130*sampling_time2) + 5
plt.plot(sampling_time2,sampled_signal2,"r*",sampling_time2,sampled_sig
nal2,"b")
print(sampled_signal2) # -TVE19EC061
```

```
[6.         7.33163983 8.0018253  7.83734924 6.91642063 5.5402652
 4.13321853 3.10503716 2.7219465  3.02952356 3.85204869 4.86603575
 5.71989327 6.15626492 6.09394659 5.64203952 5.04424359 4.57660551
 4.43834807 4.67680429 5.17371169 5.69652274 5.99386272 5.89793973
 5.39455301 4.6339746  3.87850411 3.4063881  3.40811363 3.9138123
 4.77876826 5.73233954 6.47215982 6.76902378 6.54439627 5.89311649
 5.04442152 4.27708511 3.82080186 3.78000797 4.10688351 4.63117424
 5.13308925 5.42995321 5.44323096 5.22123174 4.9106172  4.68977334
 4.69149887 4.94592539 5.3660254  5.7810175  6.0041733  5.90825031
 5.47904776 4.82628831 4.1476252  3.6595389  3.52128145 3.78018591
 4.35796048 5.08162392 5.74584811 6.18221976 6.30953476 6.14795131
 5.79490594 5.37594046 4.9928498  4.69121096 4.4597348  4.25914987
 4.06476379 3.90028773 3.84393067 4.         4.4453408  5.1731202
 6.06081725 6.88227726 7.36735611 7.29301844 6.57453711 5.32120363
 3.83033143 2.51378747 1.77582625 1.88061392 2.8548252  4.46167434
 6.26007351 7.7338042  8.45071062 8.20001407 7.06157303 5.38276861
 3.66939366 2.42590721 1.99758104 2.46636914 3.6339746  5.09595945
 6.38015472 7.10165018 7.08236852 6.39680225 5.33180021 4.27788073
 3.59206277 3.48007302 3.93682129 4.76205477 5.6413585  6.26075453
 6.41351573 6.06317871 5.34435647 4.5058242  3.82000624 3.49262928
 3.60319775 4.09320198 4.80046285 5.52195831 6.07961105 6.3660254
 6.35806035 6.10030593 5.67197976 5.15503584 4.61723139 4.11399748
 3.70209896 3.45140241 3.4417663  3.73992649 4.36275515 5.24306176
 6.21727305 7.04860325 7.48621253 7.34523907 6.5809094  5.32757592
 3.88255207 2.63264389 1.94215223 2.03706972 2.92476677 4.37908869]
```



In [17]:

```
# Quantizing with L levels # -TVE19EC061
L = int(input("Enter no.of quantization levels: "))
signal_min2 = min(message_signal)
signal_max2 = max(message_signal)
Quantization_levels2 = np.linspace(signal_min2,signal_max2,L) # -TVE19E
C061
quantized_signal2 = []
for i in sampled_signal2:
    for j in Quantization_levels2:
        if i <= j:
            quantized_signal2.append(j)
            break
plt.subplot(1,1,1)
plt.plot(sampling_time2,quantized_signal2,"b",sampling_time2,quantized_
signal2,"m*") # -TVE19EC061
plt.title("Quantized Signal")
plt.show() # -TVE19EC061

#Quantizer # -TVE19EC061
q_level=[]
for i in np.linspace(signal_min2,signal_max2,1000): # -TVE19EC061
    for j in Quantization_levels2:
        if i <= j:
            q_level.append(j)
            break
plt.subplot(1,1,1)
plt.plot(np.linspace(signal_min2,signal_max2,1000),q_level)# -TVE19EC06
1
plt.title("Quantizer")
plt.show() # -TVE19EC061
```

```
Enter no.of quantization levels: 128
```