ASCII to Binary Conversion

```
import binascii as bina
s = "Jerry".encode()
binary = int.from_bytes(s, "big")
bbins = bin(binary)[2:]
print(s, "\n", binary, bbins)
```

```
b'Jerry'
 319529579129 100101001100101011100100111001001111001
```

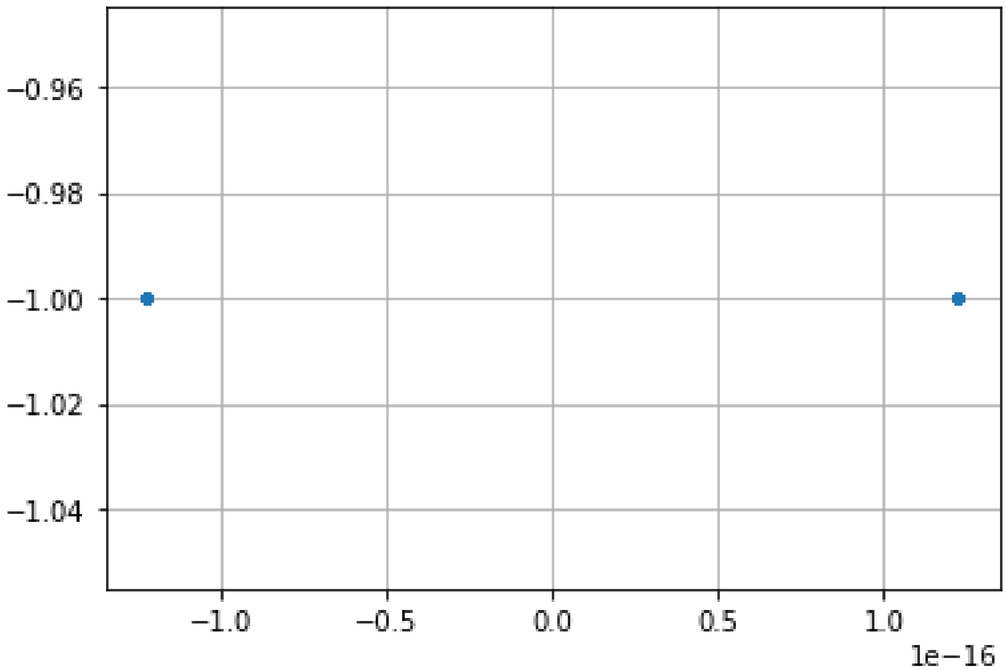
```
s = 95
s = str(s)
new = []
for i in s:
    j = int(i)+3
    x = bin(int(j))[2:].zfill(4)
    new.append(x)
new="".join(new)
print(new)
```

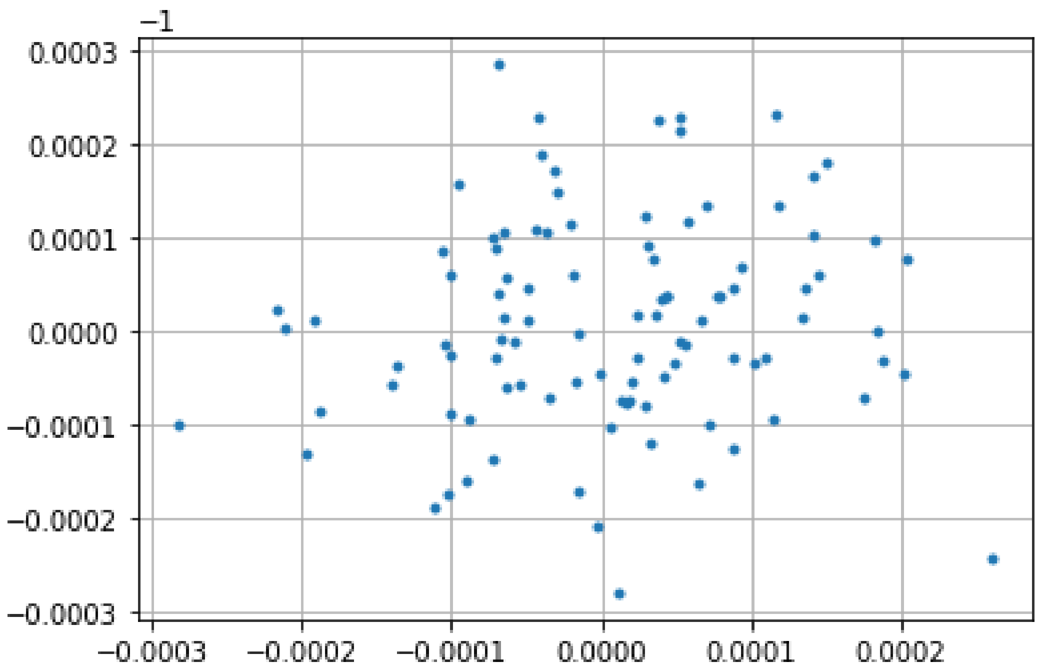
```
11001000
```

BPSK

Constellation Plot

```
import numpy as np
import matplotlib.pyplot as plt
x = 2*np.random.randint(0, 2, 100)-1
phase = x*(np.pi)
xout = np.cos(phase)+1j*np.sin(phase)
plt.plot(np.imag(xout), np.real(xout), ".")
plt.grid()
plt.show()
noise = (np.random.randn(100)+1j*np.random.randn(100))*0.0001
xout = xout+noise
plt.plot(np.imag(xout), np.real(xout), ".")
plt.grid()
plt.show()
```

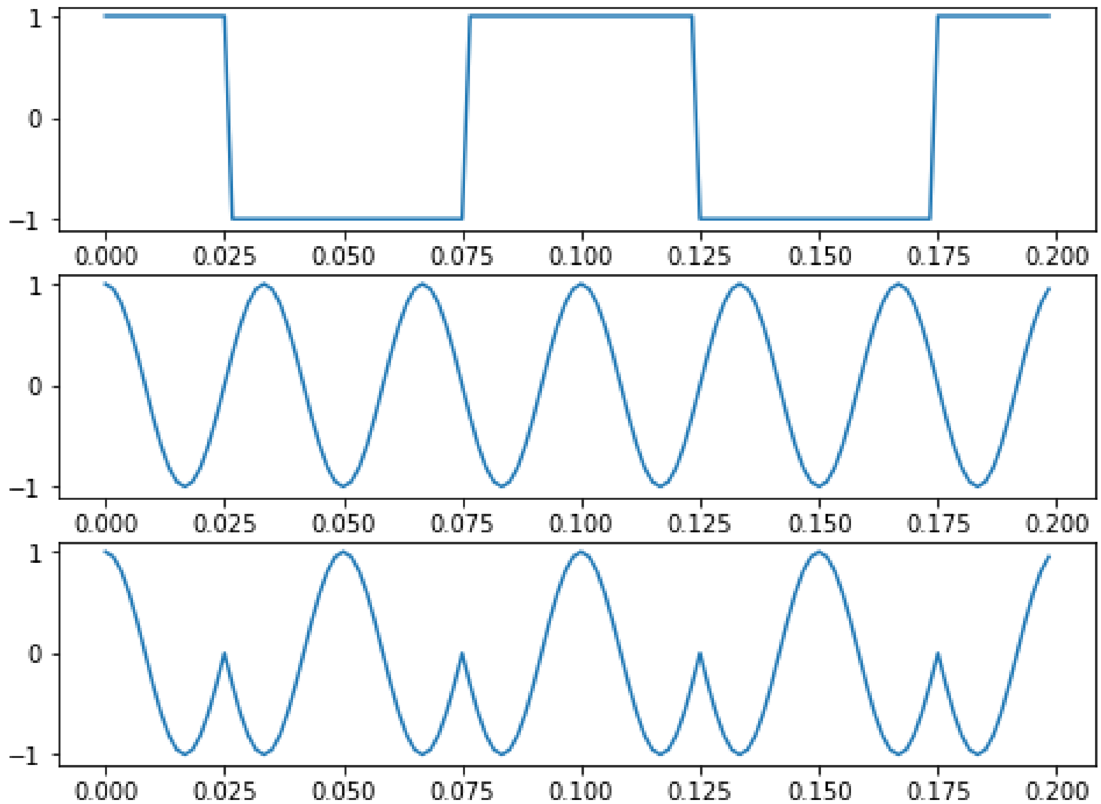


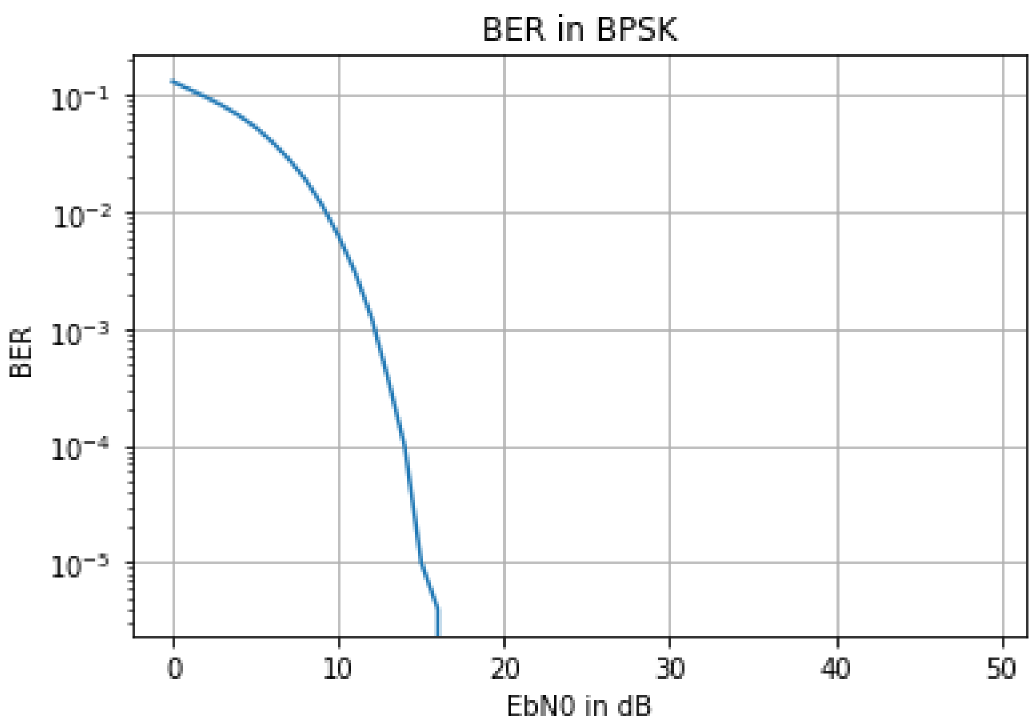


Generation of BPSK Signal

```
# # Generation of BPSK signal


import numpy as np
import matplotlib.pyplot as plt

message_frequency = 10
```

```python
carrier_frequency = 20
sampling_frequency = 30 * carrier_frequency
t = np.arange(0, 4/carrier_frequency, 1/sampling_frequency)

message = np.sign(np.cos(2 * np.pi * message_frequency * t) +
                  np.random.normal(scale=0.01, size=len(t)))
carrier = np.cos(2 * np.pi * sampling_frequency/carrier_frequency * t)
modulated_signal = carrier * message

plt.figure(figsize=(8, 6))
plt.subplot(3, 1, 1)
plt.plot(t, message)
plt.subplot(3, 1, 2)
plt.plot(t, carrier)
plt.subplot(3, 1, 3)
plt.plot(t, modulated_signal)
plt.show()


# # Monte Carlo Simulation

N = 500000
EbN0dB_list = np.arange(0, 50)

BER = []
for i in range(len(EbN0dB_list)):
    EbN0dB = EbN0dB_list[i]
    EbN0 = 10**(EbN0dB/10)
    x = 2 * (np.random.rand(N) >= 0.5) - 1
    noise = 1/np.sqrt(2 * EbN0)
    channel = x + np.random.randn(N) * noise
    received_x = 2 * (channel >= 0.5) - 1
    errors = (x != received_x).sum()
    BER.append(errors/N)


plt.plot(EbN0dB_list, BER, "-", "go")
#splt.axis([0, 14, 1e-7, 0.1])
plt.xscale('linear')
plt.yscale('log')
plt.grid()
plt.xlabel("EbN0 in dB")
plt.ylabel("BER")
plt.title("BER in BPSK")
plt.show()
```





BER in BPSK
BER
EbN0 in dB

Eye Diagram

```
import numpy as np
import matplotlib.pyplot as plt
from commpy.filters import rcosfilter as rcf

n = 100
nm = n
sps = 8

bit = np.random.randint(0, 2, n)*2-1
print(bit)
sampled_bit = []
for x in range(n):
    sampled_bit = np.concatenate(
        (sampled_bit, np.concatenate(([bit[x]], np.zeros(sps-1)))))

n = 100
alph = 0.35
ts = 8
# rc = 1/ts*np.sinc(t/ts)np.cos(np.pi*alph*t/ts)/(1-(2*alph*t/ts)*2)
rc = rcf(n, alph, ts, 1)[1]
plt.plot(rc)
plt.show()
# plt.plot(rc)
y = np.convolve(rc, sampled_bit)
plt.plot
plt.show()

fir = 0
for i in range(len(rc)):
    if abs(y[i]) >= 1:
        print(rc[i])
        fir = i
        break
print(len(fir))

for i in range(0, nm):
    plt.plot(y[fir+16*i:fir+16*(i+1)])
    # print(fir+sps*i,fir+sps*(i+1)," ",rc[fir+sps*i])
plt.show()
```

```
[-1  1  1 -1 -1  1  1  1 -1  1 -1  1  1 -1 -1 -1  1  1  1 -1  1 -1 -1
-1
  1 -1  1  1 -1 -1 -1 -1 -1 -1  1 -1  1 -1  1  1 -1 -1  1 -1 -1  1  1
1
  1  1 -1  1 -1 -1  1 -1 -1 -1  1  1  1  1 -1  1 -1 -1 -1  1  1  1  1
-1
  1 -1  1 -1  1  1  1  1  1  1  1 -1 -1  1  1  1  1  1 -1  1  1 -1  1
1
  1 -1  1 -1]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [6], in <cell line: 19>()
     17 ts = 8
     18 # rc = 1/ts*np.sinc(t/ts)np.cos(np.pi*alph*t/ts)/(1-(2*alph*t/ts)*2)
---> 19 rc = rcf(n, alph, ts, 1)[1]
     20 plt.plot(rc)
     21 plt.show()

NameError: name 'rcf' is not defined
```

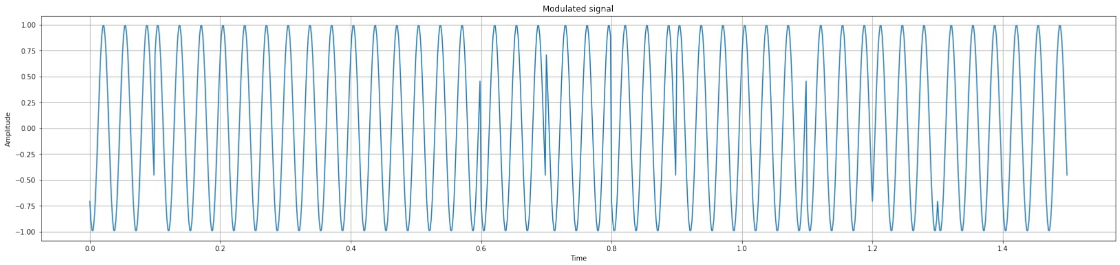
QPSK and it's Error Performance

```python
# Error Performance of QPSK # QPSK Modulation
import numpy as np
from numpy import pi
import matplotlib.pyplot as plt


def cosineWave(f, overSamplingRate, nCycles, phase):
    fs = overSamplingRate * f
    t = np.arange(0, nCycles*1/f, 1/fs)
    g = np.cos(2 * np.pi * f * t + phase)
    return list(g)


fm = 10
fc = 30
overSamplingRate = 20
fs = overSamplingRate * fc

x = np.random.rand(30) >= 0.5

str_x = [str(int(i)) for i in x]
x = "".join(str_x)
print("Message string : {}".format(x))
message = [x[2*i: 2*(i+1)] for i in range(int(len(x)/2))]
print("Message string grouped as combinations of 2 bits each: {}".format(message))

mod_00 = cosineWave(fc, overSamplingRate, fc/fm, 3*pi/4)
mod_01 = cosineWave(fc, overSamplingRate, fc/fm, pi/4)
mod_10 = cosineWave(fc, overSamplingRate, fc/fm, -3*pi/4)
mod_11 = cosineWave(fc, overSamplingRate, fc/fm, -pi/4)

modulated_signal = []
for i in message:
```

```
    if i == '00':
        modulated_signal = modulated_signal + mod_00
    if i == '01':
        modulated_signal = modulated_signal + mod_01
    if i == '10':
        modulated_signal = modulated_signal + mod_10
    if i == '11':
        modulated_signal = modulated_signal + mod_11

t = np.arange(0, (len(x)/2) * 1/fm, 1/fs)
print(len(t), len(modulated_signal))
plt.figure(figsize=(28, 6))
plt.plot(t, modulated_signal)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Modulated signal")
plt.grid(True)
plt.show()


# Error performance of QPSK

N = 500000
EbN0dB_list = np.arange(0, 50)

BER = []
for i in range(len(EbN0dB_list)):
    EbN0dB = EbN0dB_list[i]
    EbN0 = 10**(EbN0dB/10)
    x = np.random.rand(N) >= 0.5
    x_str = [str(int(i)) for i in x]
    x_str = "".join(x_str)
    message = [x_str[2*i: 2*(i+1)] for i in range(int(len(x)/2))]
    noise = 1/np.sqrt(2 * EbN0)
    channel = x + np.random.randn(N) * noise
    received_x = channel >= 0.5
    xReceived_str = [str(int(i)) for i in received_x]
    xReceived_str = "".join(xReceived_str)
    messageReceived = [xReceived_str[2*i: 2 *
                                     (i+1)] for i in
range(int(len(x)/2))]
    message = np.array(message)
    messageReceived = np.array(messageReceived)
    errors = (message != messageReceived).sum()
    print(errors)
    BER.append(errors/N)

print(BER)
plt.plot(EbN0dB_list, BER, "-", EbN0dB_list, BER, "go")
```

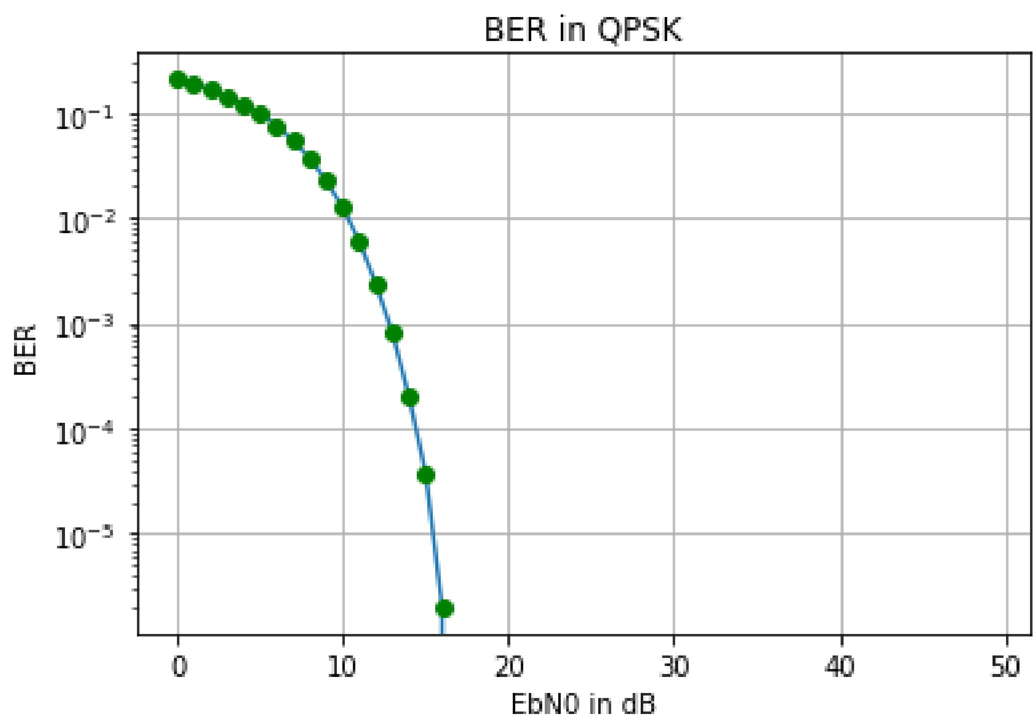
```
plt.xscale('linear')
plt.yscale('log')
plt.grid()
plt.xlabel("EbN0 in dB")
plt.ylabel("BER")
plt.title("BER in QPSK")
plt.show()
```

```
Message string : 001111111111000100111100100000
Message string grouped as combinations of 2 bits each:['00', '11',
'11', '11', '11', '11', '00', '01', '00', '11', '11', '00', '10',
'00', '00']
900 900
```




```
105388
95502
84691
72755
61215
49668
38320
27243
18770
11312
6311
3013
1182
407
100
19
1
0
0
0
0
0
0
0
0
0
0
```

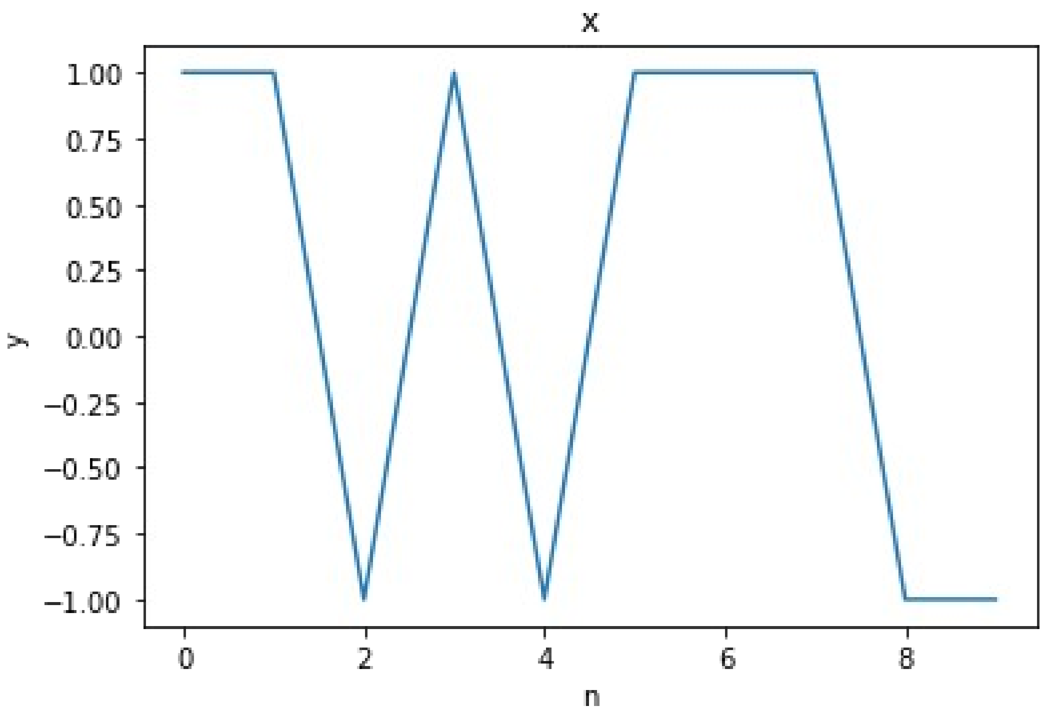
```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
[0.210776, 0.191004, 0.169382, 0.14551, 0.12243, 0.099336, 0.07664,
0.054486, 0.03754, 0.022624, 0.012622, 0.006026, 0.002364, 0.000814,
0.0002, 3.8e-05, 2e-06, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

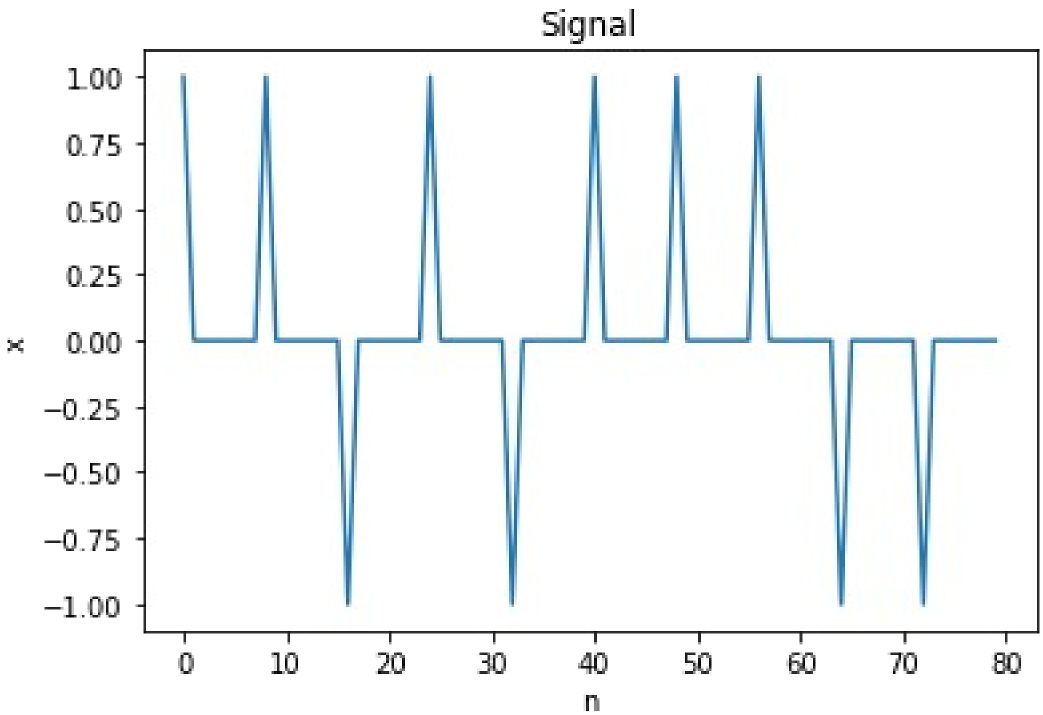


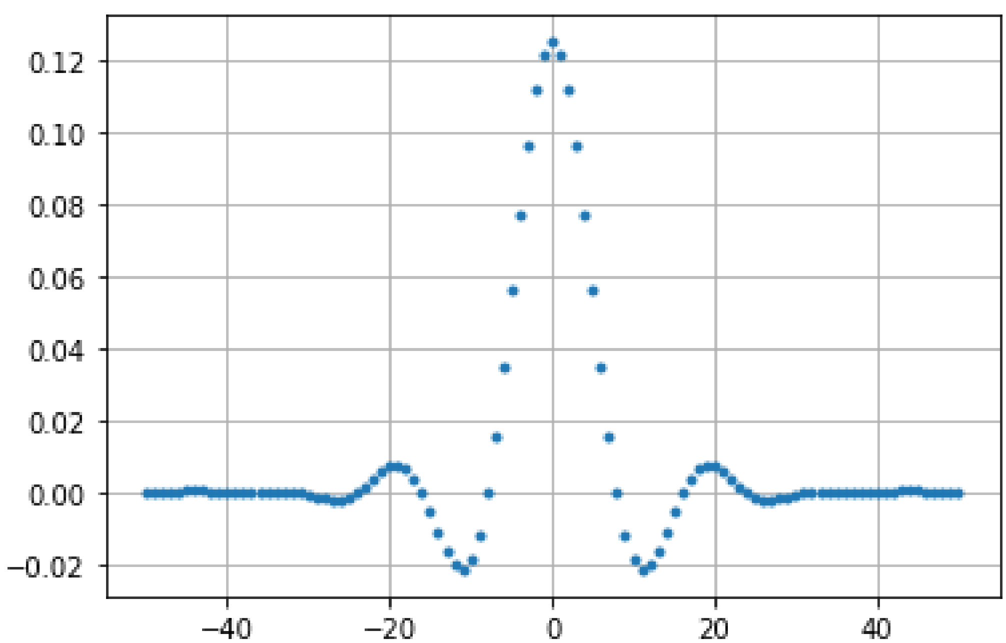
Matched Filter

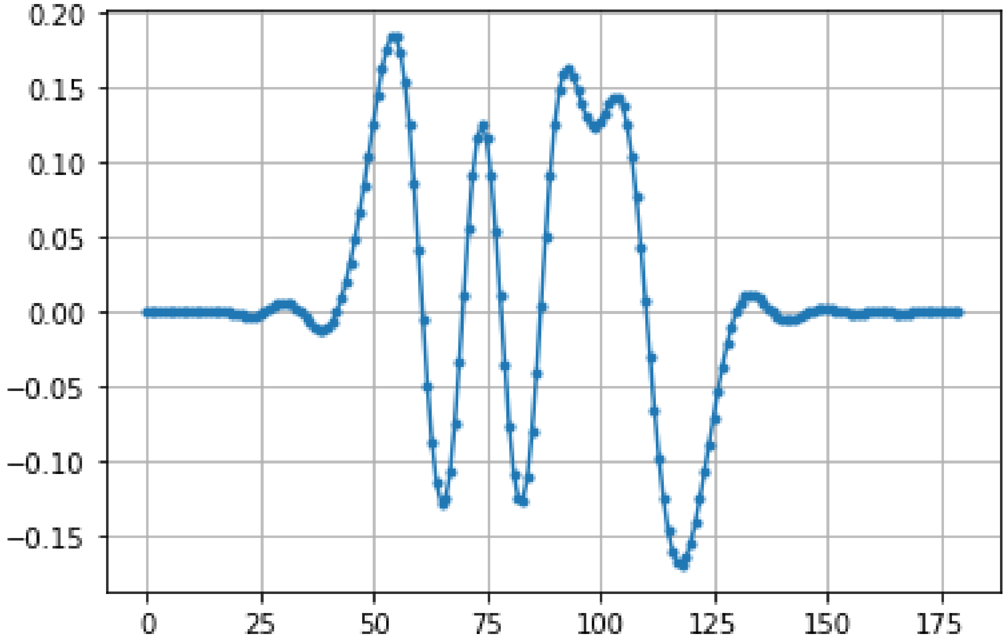
```
import numpy as np
import random
import matplotlib.pyplot as plt

num_symbols = 10
sps = 8
x = random.choices([0, 1], k=10)
for i in range(len(x)):
    if x[i] == 0:
        x[i] = -1

plt.plot(x)
plt.xlabel("n")
plt.ylabel('y')
plt.title("x")
plt.show()


signal = np.array([])
for value in x:
    pulse = np.zeros(sps)
    pulse[0] = value
    signal = np.concatenate((signal, pulse))

plt.plot(signal)
plt.xlabel("n")
plt.ylabel('x')
plt.title("Signal")
plt.show()


num_taps = 101
beta = 0.35
Ts = sps
t = np.arange(-50, 51)
h = 1/Ts * np.sinc(t/Ts) * np.cos(np.pi * beta * t/Ts) / \
    (1 - (2 * beta * t/Ts)**2)
plt.plot(t, h, ".")
plt.grid(True)
plt.show()

shaped_signal = np.convolve(signal, h)
plt.plot(shaped_signal, '.-')
plt.grid(True)
plt.show()

final_signal = np.convolve(h, shaped_signal)
plt.plot(final_signal)
```

```
plt.grid(True)
plt.show()

x_shaped = np.convolve(signal, h)
for i in range(num_symbols):
    plt.plot([i*sps+num_taps//2+1, i*sps+num_taps//2+1],
             [0, x_shaped[i*sps+num_taps//2+1]])

plt.grid(True)
plt.show()
```

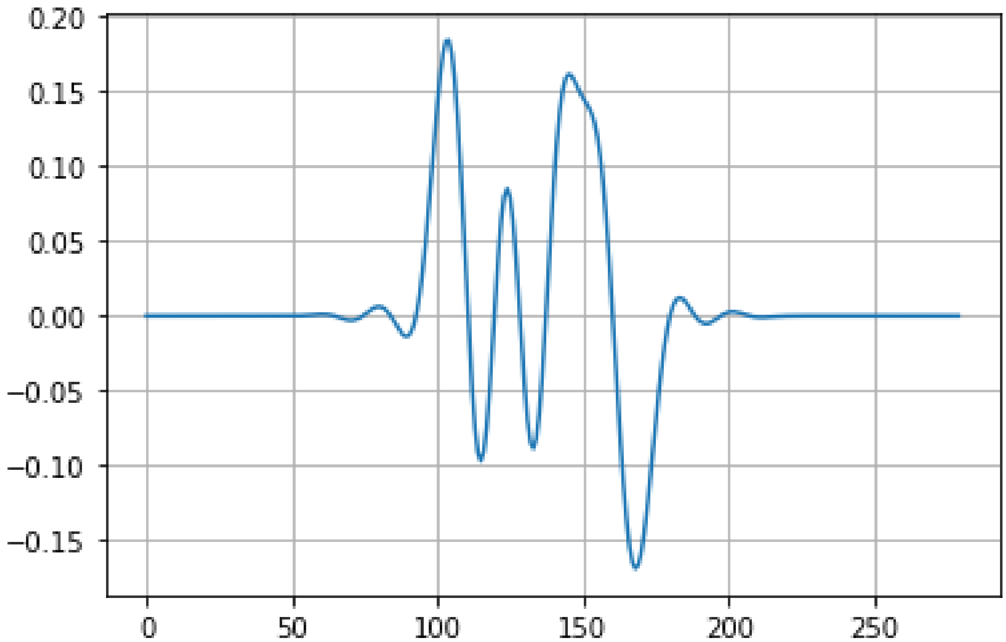



Signal
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x
0
10
20
30
40
50
60
70
80
n

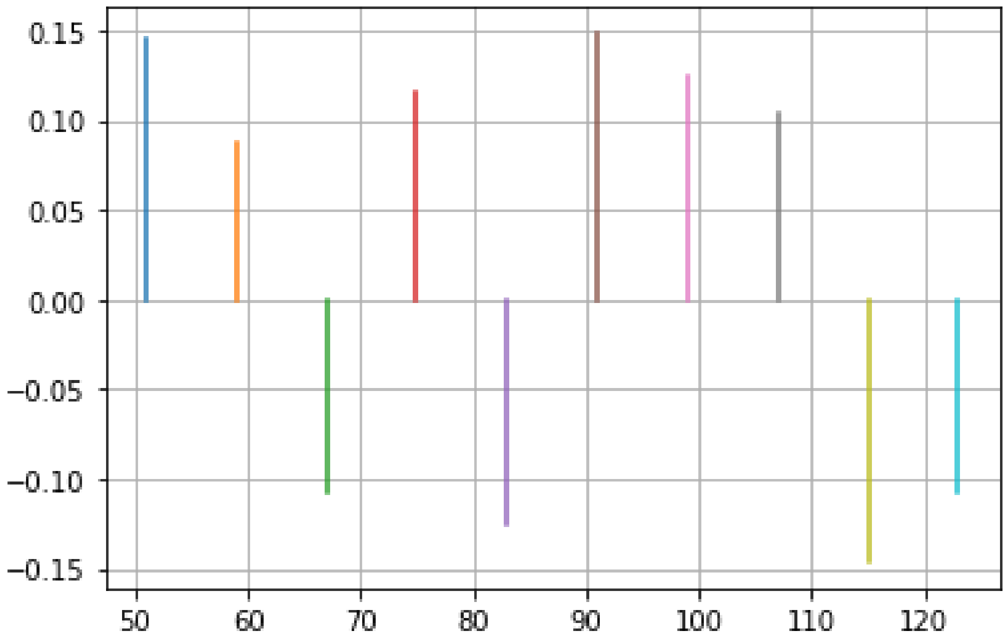

0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
−40
−20
0
20
40



0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
25
50
75
100
125
150
175

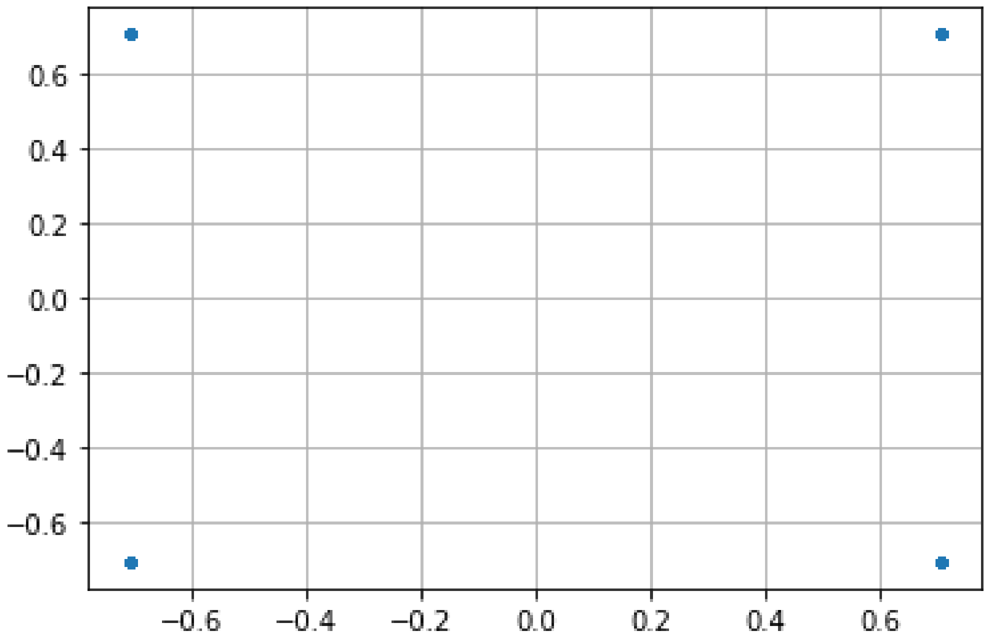


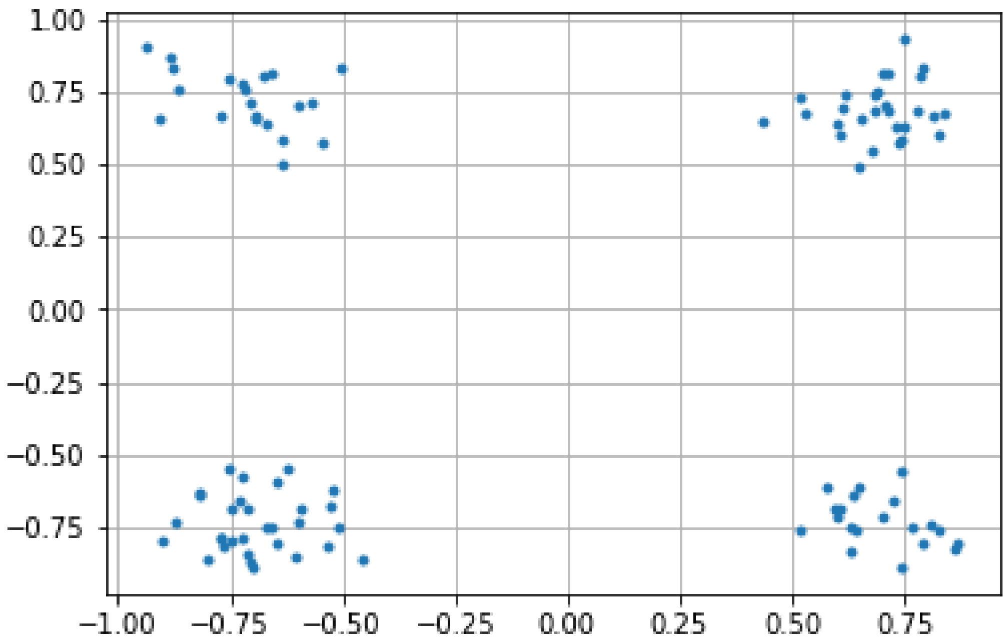
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
50
100
150
200
250



QPSK Constellation Plot

```
import numpy as np
import matplotlib.pyplot as plt

x_int = np.random.randint(0, 4, 100)  # 0 to 3
x_degrees = x_int*360/4.0 + 45  # 45, 135, 225, 315 degrees
x_radians = x_degrees*np.pi/180.0  # sin() and cos() takes in radians
# this produces our QPSK complex symbols
x_symbols = np.cos(x_radians) + 1j*np.sin(x_radians)
plt.plot(np.real(x_symbols), np.imag(x_symbols), '.')
plt.grid(True)
plt.show()
noise = (np.random.randn(100)+1j*np.random.randn(100))*0.1
x_symbols = x_symbols+noise
plt.plot(np.real(x_symbols), np.imag(x_symbols), '.')
plt.grid(True)
plt.show()
```





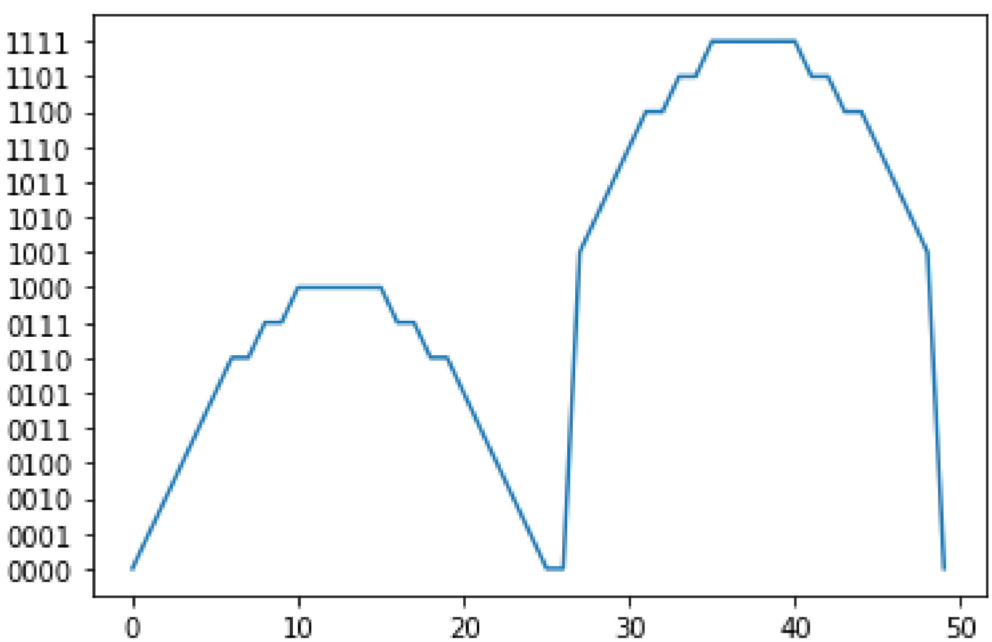
Sine Wave Quantization

```
import numpy as np
import matplotlib.pyplot as plt


def find_nearest(array, value):
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return array[idx]
```

```
def quantization(ys, l):
    quantarr = []
    quantl = np.linspace(min(ys), max(ys), l)
    # here we compare each value in y to quantl and see if value is
there in array that indices is taken and appeneded
    # quantarr= is that quantised sig
    # quantsig= is basically representing that straight line and
finding nearest in each case
    # quantised-sampled sig=noise
    for i in ys:
        quantarr.append(find_nearest(quantl, i))
    # ax[1][0].plot(quantarr)
    quantisedsig = []
    for i in np.linspace(min(ys), max(ys), 200):
        quantisedsig.append(find_nearest(quantl, i))
    return(quantarr, quantisedsig)


def encode(quantisedsig):
    width = np.log2(l)
    print(width)
    quantsigdic = {}
    count = 0
    sets = set(quantisedsig)
    for i in sets:
        quantsigdic[i] = quantsigdic.get(i, count)
        count += 1
    for i in quantsigdic:
        quantsigdic[i] = np.binary_repr(quantsigdic[i], int(width))
    quantbinaries = []
    for i in quantisedsig:
        quantbinaries.append(quantsigdic[i])
    print(quantbinaries)
    plt.plot(quantbinaries)
    plt.show()
    return(quantbinaries)


def snr(qsig):
    qnoise = qsig-ys
    noisepwr = sum(np.array(qnoise)**2)
    sigpwr = sum(np.array(y)**2)
    print(noisepwr, sigpwr)
    return(qnoise)


fm = 100
```

```
fs = 50*fm
fc = 300
dc = 2
t = np.arange(0, 3*1/fc, 1/fm)
y = np.sin(2*np.pi*fm*t)+dc
ts = np.arange(0, 3*1/fc, 1/fs)
ys = np.sin(2*np.pi*fm*ts)+dc

l = 16


qsig, qgraphsig = quantization(ys, l)
pcm = encode(qsig)
qnoise = snr(qsig)
fig, ax = plt.subplots(2, 2)
ax[0][0].plot(t, y)
ax[0][1].stem(ts, ys)
ax[1][0].plot(qgraphsig)
ax[1][1].plot(qnoise)
plt.show()
```

```
4.0
['0000', '0001', '0010', '0100', '0011', '0101', '0110', '0110',
'0111', '0111', '1000', '1000', '1000', '1000', '1000', '1000',
'0111', '0111', '0110', '0110', '0101', '0011', '0100', '0010',
'0001', '0000', '0000', '1001', '1010', '1011', '1110', '1100',
'1100', '1101', '1101', '1111', '1111', '1111', '1111', '1111',
'1111', '1101', '1101', '1100', '1100', '1110', '1011', '1010',
'1001', '0000']
```



0.07166680890015185 4.0