2. (35 pts) Write Python code to produce an eye diagram (you can start with the code provided in the notes). Use it to produce eye diagrams as follows:

(a) (5 pts) SRRC with 100 % excess bandwidth (you will need to write your own SRRC function) for 4 PAM.

(b) (5 pts) SRRC with 50 % excess bandwdidth for 4 PAM.

(c) (5 pts) The real (inphase) part of 16QAM with SRRC with 100% excess band- width.

(d) (5 pts) The real (inphase) part of 16QAM with SRRC with 50% excess bandwidth.

(e) (5 pts) Plot an eye diagram for NRZ with BPSK modulation.

(f) (5 pts) Plot an eye diagram for Manchesester with BPSK modulation.

2a:

The following code was used to produce all pulses. The file was saved as 'pulses.py' (so it's clear what's being imported in other files)

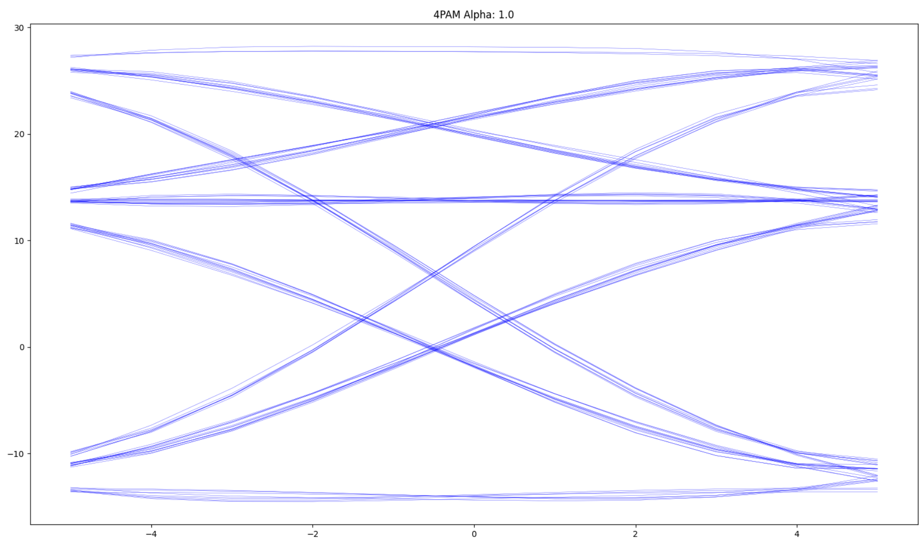
In [ ]:
```
from math import sqrt, pi, sin, cos, floor

def srrc1(alpha,N,Lp,Ts):
    """
    Return a vector of the srrc function
    alpha = excess bandwidth
    N = samples per symbol
    Lp = SRRC truncation length #Currently only supports even numbers
    Ts = sample time
    """

    times = []
    number_of_samples = int(floor(Lp/2)) # and then reflect it on the axis?
    for idx in range(number_of_samples):
        t = idx * Ts / N
        times.append(t)
        times.append(-t)
    times.sort()

    # print(len(times))
    # print(times)

    answer = []
    for t in times:
        answer.append(p_of_nT(Ts, alpha, t))

    # print(answer)
    return answer
```

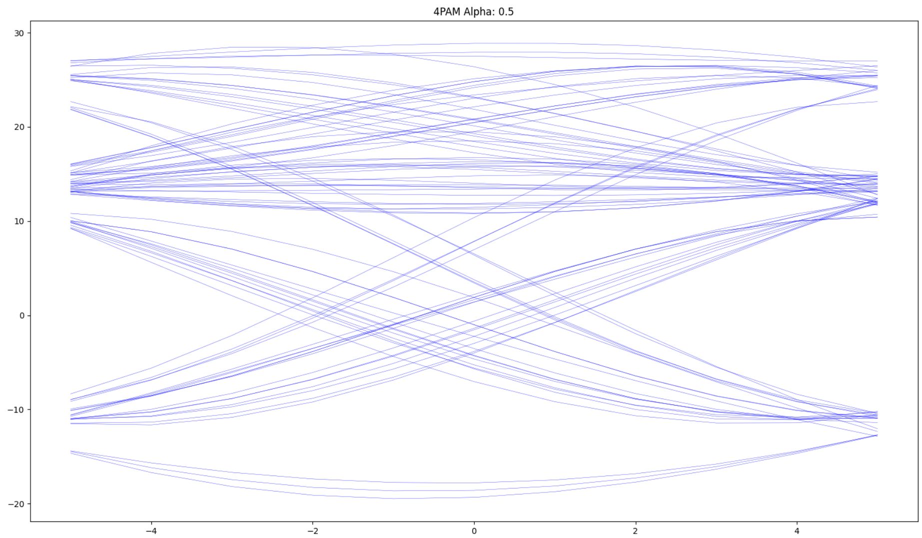
```
def p_of_nT(Ts, alpha, t):
    undefined_t_vals = [0, Ts / (4 * alpha)]
    if t in undefined_t_vals:
        return lhopital(Ts, alpha, t)
    else:
        return (1/sqrt(Ts)) * ((sin(pi*(1 - alpha) * t / Ts) + (4 * alpha *

def lhopital(Ts, alpha, t):
    numerator = (pi * (1 - alpha) / Ts) * cos(pi * (1 - alpha) * t / Ts) + (
    denominator = pi / sqrt(Ts) - (32 * pi * t / (Ts * sqrt(Ts)))
    return numerator / denominator

def NRZ(Ts):
    return [Ts]

def MANCH(Ts, Lp=60):
    return [-Ts, Ts] + [0] * Lp
```

Then this code was used, mostly the same as in the notes, to produce the following plot.

In [ ]:
```
# test eye diagram plots
import math
import numpy as np
from numpy.random import rand
from pulses import srrc1
import matplotlib.pyplot as plt

alpha = 1.0     # excess bandwidth
N = 11          # samples per symbol
Lp = 60         # SRRC truncation length
Ts = 1          # symbol time
T = Ts/N        # sample time
srrcout = srrc1(alpha,N,Lp,Ts); # get the square-root raised cosine pulse
rcout = np.convolve(srrcout,srrcout)

# peak at 2Lp
plt.figure(1); plt.clf()
plt.plot(srrcout)
plt.show()
a = 1 # PAM amplitude
LUT1d = np.array([-1,1])*a # 1-dimensional lookup table
Nsym = 100 # number of symbols
bits = (rand(Nsym)> 0.5).astype(int) # generate random bits {0,1}
ampa = LUT1d[bits] # map the bits to {+1,-1} values
print(type(ampa))
upsampled = np.zeros(N*Nsym) # make space for the upsampled data
# for i in range(0,Nsym): upsampled[N*i] = ampa[i]
# breakpoint()
upsampled = np.zeros((N*Nsym,1))
upsampled[range(0,N*Nsym,N)] = ampa.reshape(Nsym,1)
plt.figure(2); plt.clf()
plt.stem(upsampled,linefmt='b',markerfmt='.',basefmt='b-')
s = np.convolve(upsampled.reshape((N*Nsym,)),srrcout) # the transmitted sign
plt.figure(3); plt.clf()
plt.plot(s)
```

```
plt.show()
x = np.convolve(s,srrcout) # the matched filter
plt.figure(4); plt.clf()
plt.plot(x[:(2*Lp+1) + N*20])
for i in range(20):
    plt.plot(np.array([2*Lp + i*N,2*Lp + i*N]),np.array([-2,2]),color='gray'
# plt.show()
print('bits=',bits[:10])
# first peak at 2*Lp, then every N samples after that
offset = (2*Lp - np.floor(N/2)).astype(int)
# ^ ^
# 1st correlation |
# peak |
# move to center
Nsymtoss = 2*np.ceil(Lp/N) # throw away symbols at the end
nc = (np.floor((len(x) - offset - Nsymtoss*N)/N)).astype(int) # number of pc
xreshape = x[offset:offset + nc*N].reshape(nc,N)
plt.figure(5); plt.clf()
plt.plot(np.arange(-np.floor(N/2), np.floor(N/2)+1), xreshape.T,color='b',
linewidth=0.25)
plt.show()
```

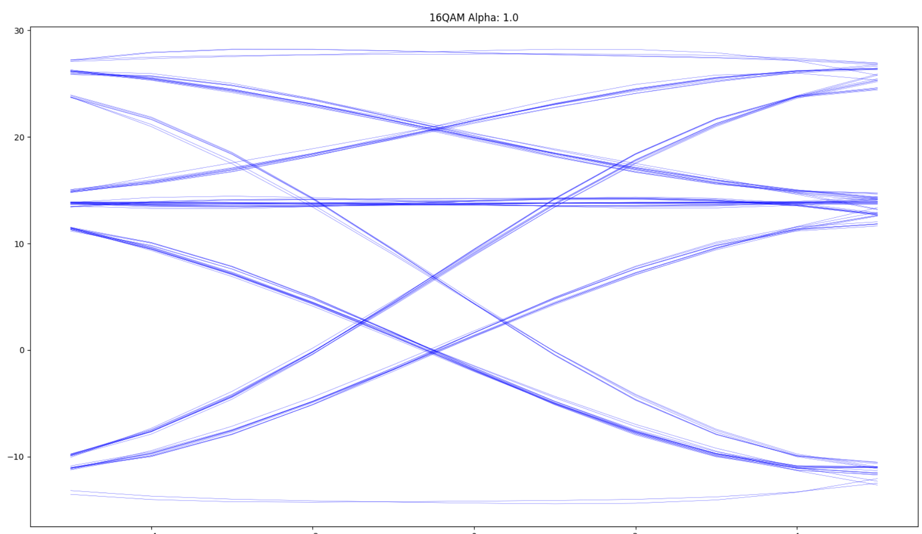


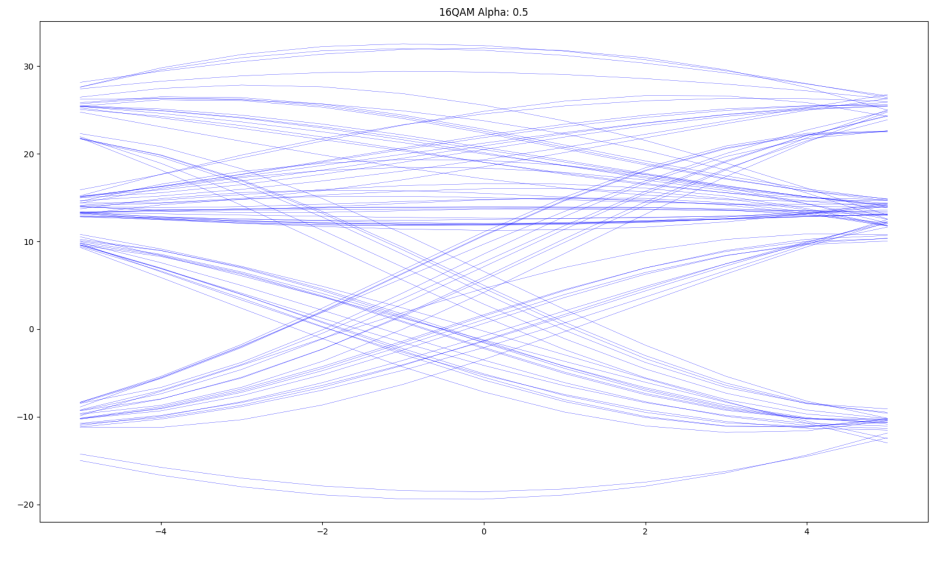


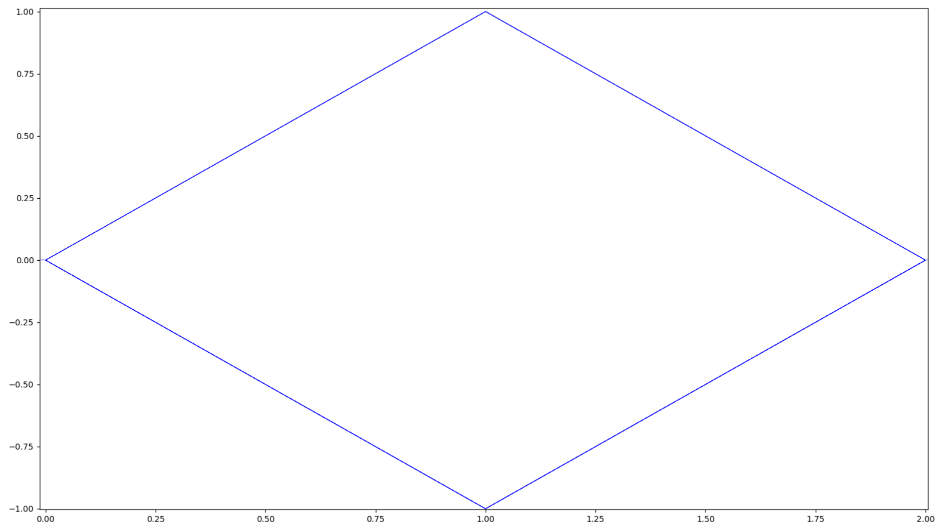
Small changes were made to the code to support inphase 16QAM

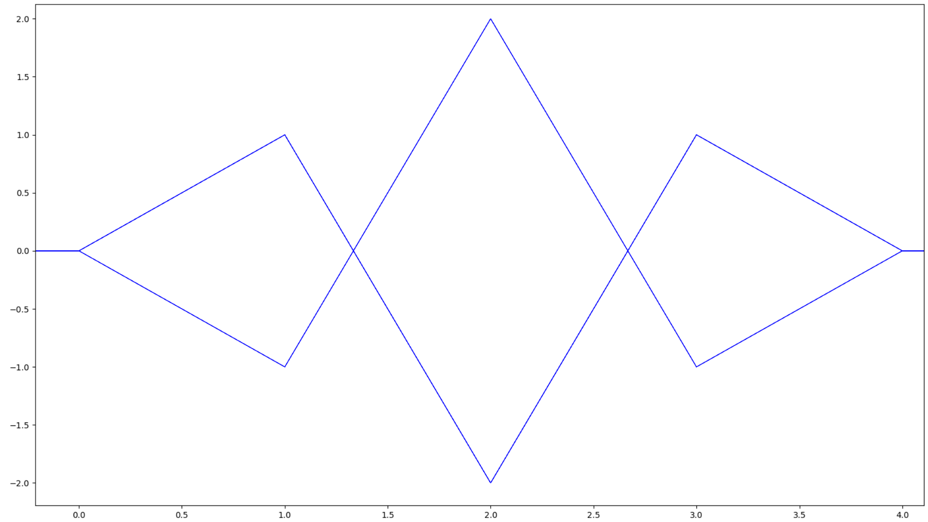
In [ ]:
```
# test eye diagram plots
import numpy as np
from numpy.random import rand, seed
from pulses import srrc1
import matplotlib.pyplot as plt

seed(12345)

alpha = 1.0     # excess bandwidth
N = 11          # samples per symbol
Lp = 60         # SRRC truncation length
Ts = 1          # symbol time
T = Ts/N        # sample time
srrcout = srrc1(alpha,N,Lp,Ts); # get the square-root raised cosine pulse
rcout = np.convolve(srrcout,srrcout)

# peak at 2Lp
plt.figure(1); plt.clf()
plt.plot(srrcout)
# plt.show()
a = 1 # PAM amplitude
LUT = np.array([-2,-1,1,2])*a # 1-dimensional lookup table
Nsym = 100 # number of symbols
bit_idx = []
bits = (rand(Nsym*2)> 0.5).astype(int) # generate random bits {0,1}
for idx in range(Nsym):   # Convert from 2 bits to one number
    bit_idx.append(LUT[bits[idx*2]*2 + bits[idx*2 + 1]])
ampa = LUT[bit_idx] # map the bits to {+1,-1} values
upsampled = np.zeros(N*Nsym) # make space for the upsampled data
# for i in range(0,Nsym): upsampled[N*i] = ampa[i]
# breakpoint()
upsampled = np.zeros((N*Nsym,1))
upsampled[range(0,N*Nsym,N)] = ampa.reshape(Nsym,1)
```

```
plt.figure(2); plt.clf()
plt.stem(upsampled,linefmt='b',markerfmt='.',basefmt='b-')
s = np.convolve(upsampled.reshape((N*Nsym,)),srrcout) # the transmitted sign
plt.figure(3); plt.clf()
plt.plot(s)
# plt.show()
x = np.convolve(s,srrcout) # the matched filter
plt.figure(4); plt.clf()
plt.plot(x[:(2*Lp+1) + N*20])
for i in range(20):
    plt.plot(np.array([2*Lp + i*N,2*Lp + i*N]),np.array([-2,2]),color='gray'
# plt.show()
print('bits=',bits[:10])
# first peak at 2*Lp, then every N samples after that
offset = (2*Lp - np.floor(N/2)).astype(int)
# ^ ^
# 1st correlation |
# peak |
# move to center
Nsymtoss = 2*np.ceil(Lp/N) # throw away symbols at the end
nc = (np.floor((len(x) - offset - Nsymtoss*N)/N)).astype(int) # number of po
xreshape = x[offset:offset + nc*N].reshape(nc,N)
plt.figure(5); plt.clf()
plt.plot(np.arange(-np.floor(N/2), np.floor(N/2)+1), xreshape.T,color='b',
linewidth=0.25)
plt.title(f"16QAM Alpha: {alpha}")
plt.show()
```




16QAM Alpha: 0.5
30
20
10
0
−10
−20
−4
−2
0
2
4


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



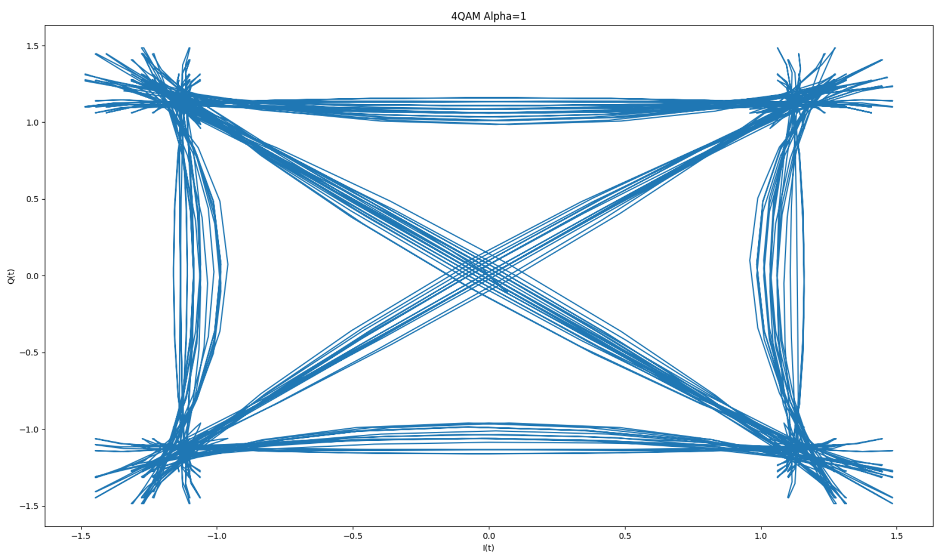
3. (15 pts) Write Python code to produce phase trajectory plots (see p. 236). Reproduce the four trajectory plots in figure 5.3.14.

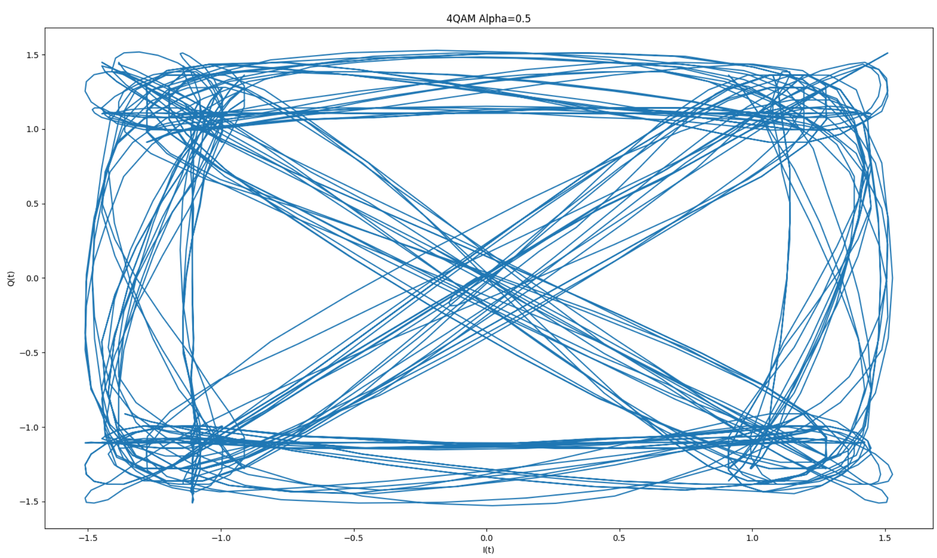
Just a note, this page number is wrong in my book (I followed the link given in the syllabus to order the book from Amazon, ISBN: 9798680369920) It's now page 244. The figure number is the same.

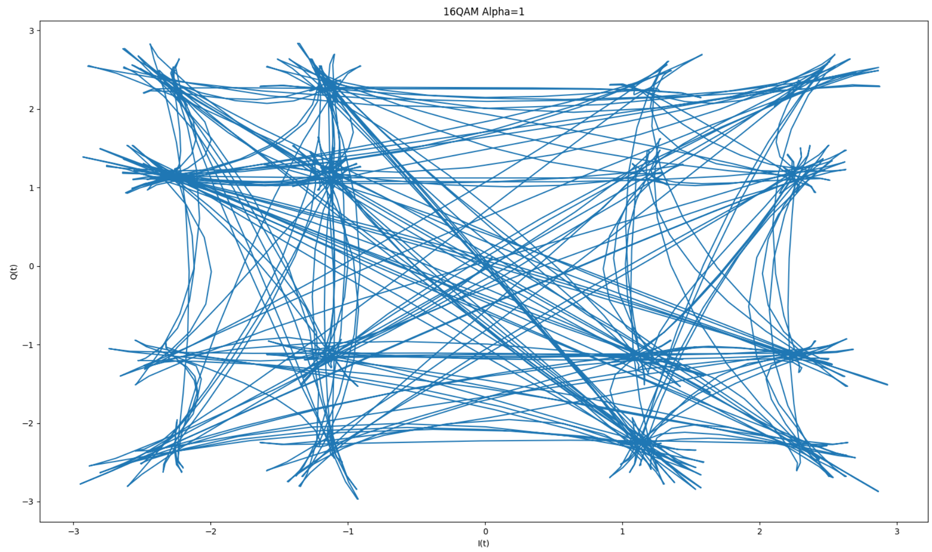
The following code was used, which is modified from the notes. The point is that all the adjustments can be made in the 'main' function at the bottom to produce all the plots.

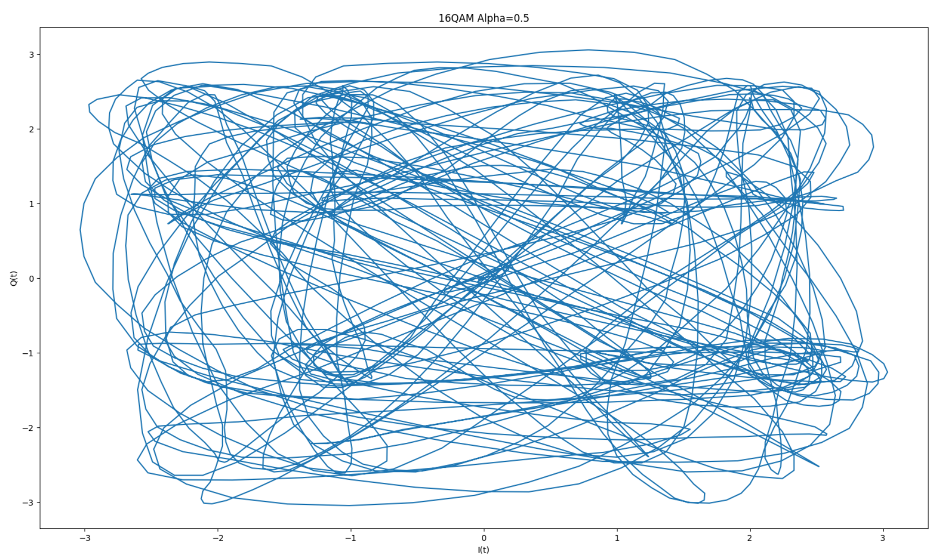
In [ ]:
```
import numpy as np
from numpy.random import rand
import matplotlib.pyplot as plt
from pulses import *

def get_filter_output(pulse_func, LUT=np.array([-1,1]), alpha:float=1.0, N:i
    T = Ts/N
    srrcout = pulse_func(alpha,N,Lp,Ts); # get the square-root raised cosine
    rcout = np.convolve(srrcout,srrcout)

    pam_len = int(len(LUT)/2)
    bit_idx = []
    bits = (rand(Nsym*(pam_len))> 0.5).astype(int) # generate random bits {0
    for idx in range(Nsym):   # Convert from 2 bits to one number
        sum = 0
        for bit in range(pam_len):
            sum = sum << 1 | bits[idx * pam_len + bit]
        bit_idx.append(sum)
    ampa = LUT[bit_idx] # map the bits to {+1,-1} values
    upsampled = np.zeros(N*Nsym) # make space for the upsampled data
    upsampled = np.zeros((N*Nsym,1))
    upsampled[range(0,N*Nsym,N)] = ampa.reshape(Nsym,1)
```

```
    s = np.convolve(upsampled.reshape((N*Nsym,)),srrcout) # the transmitted
    return s

if __name__ == '__main__':
    Nsym=200
    alpha=1
    LUT=np.array([-1,1])
    It = get_filter_output(srrc1, alpha=alpha, LUT=LUT, Nsym=Nsym)
    Qt = get_filter_output(srrc1, alpha=alpha, LUT=LUT, Nsym=Nsym)
    plt.plot(It, Qt)
    plt.title(f"{len(LUT)**2}QAM Alpha={alpha}")
    plt.xlabel("I(t)")
    plt.ylabel("Q(t)")
    plt.show()
```






16QAM Alpha=1
Q(t)
I(t)


16QAM Alpha=0.5
Q(t)
I(t)