Due on

ECE 5630/4630 Exam II November 23, 2021 Name:

1.) In the following compare the binary modulation schemes BPSK, coherent FSK, optimum DPSK, coherent ASK, and noncoherent FSK.

5 pts. a.) Rank them in terms of the E_b/N_0 required to give $P_E = 10^{-6}$ from best to worst. Note the lowest required E_b/N_0 is the best, most power efficient, system). Show your work.

5 pts. b.) Repeat part (a) except now perform the ranking in terms of bandwidth efficiency. Assume rectangular pulse shaping. Show your work.

10 pts. 2.) It can be shown that the simplified delay-and-multiply receiver for DPSK (text Figure 9.17) has asymptotic BEP when there is a delay error $|\Delta T|$, given by

$$P_{E,|\Delta T|} = \frac{1}{2}Q(\sqrt{z}) + \frac{1}{2}Q\left(\sqrt{z\left(1 - \frac{|\Delta T|}{T_b}\right)}\right).$$

Use Python to, *estimate* the degradation in dB when $|\Delta T|/T_b = |\Delta T|R_b = 0.1$ and the desired BEP is 10^{-6}. I suggest using a numerical root finding algorithm such as `scipy.optimize.brentq()` along with `scipy.stats.norm.sf()` for $Q(\ )$.

10 pts. 3.) A channel of 200 kHz (RF bandwidth) is available. What RF modulation scheme, chosen from BPSK, coherent FSK, or QPSK, can be used to communicate through it at a data rate of (a) 50 kbps; (b) 100 kbps; (c) 200 kbps. In each case also provide $(E_b/N_0)_{\text{dB}}$ to achieve a BEP of 10^{-6}. Use Python to find the numerical answers. Show your work.

20pts. 4.) DD and DFE equalizer for QPSK (4PSK). Both DD and DFE minimum mean square error equalizers are developed for BPSK in the Jupyter notebook `ECE5630_Equalizers.ipynb`. In this problem you enhance the code for the DD and DFE version of the equalizer to work with QPSK. You might want to create a clean copy of thge notebook, e.g., `ECE5630_Equalizers_QPSK.ipynb`. As the starting point I want you to use thge MPSK Gray encode/Gray decode function shown in the sample code below:

```
# Verify that we can use dc.mpsk_gray_encode_bb with mod = 4 for Gray encoded
# QPSK modulation and dc.mpsk_gray_decode for conversion of equalized I/Q
# samples of QPSK back to Gray decoded bits

# Transmitter
pulse = 'src'
mod = 4
x,b,data = dc.mpsk_gray_encode_bb(100,16,mod,pulse,0.35,6,None)

# Matched filter
z_qpsk = signal.lfilter(b,1,x)
# dc.eye_plot(z_qpsk.real,32,0)
# Set up a QPSK constellation
DFE_A = 1
# constellation = array([1, -1])*DFE_A; # BPSK cbb constellation
constellation = array([1+1j,-1+1j,-1-1j,1-1j])/sqrt(2)*DFE_A; # QPSK cbb constellation
# Assume rect pulse shaping so max eye opening is at 8 samples of 16 per symbol
timing = 15
# Obtain the max eye openiing samples
z_eq = z_qpsk[timing::16]
```

```
# plot(z_eq.real,z_eq.imag,'.')
# axis('equal')
# Declare arrays to hold the constellation index values and the final d_hat values
data_index = zeros(len(z_eq),dtype=int)
d_hat = zeros(len(z_eq),dtype=complex)
# Loop over the symbols as is done in the equalizer code to verify that the
# resulting d_hat values can be input to dc.mpsk_gray_decode(d_hat,4) to return
# the serial bits and ultimately count bit errors
for k in range(len(z_eq)):
    DFE_error = z_eq[k] - constellation
    min_error,index = min(abs(DFE_error)), argmin(abs(DFE_error))
    data_index[k] = int(index)
    d_hat[k] = constellation[int(index)]
data_hat = dc.mpsk_gray_decode(d_hat,mod)
# Count bit errors
N_RecBits,errors= dc.bit_errors(data,data_hat)
Pe = errors/N_RecBits
print('Pe = %1.2e' % (Pe,))
```

I suggest you place the above code in a notebook cell and play with it a bit to get familiar with the inputs and outputs of `mpsk_gray` functions. Then see how the `constellation` array works with QPSK versus BPSK.

I want you to modify the DFE code and also step back to a pure DD equalizer by commenting/ uncommenting code or through the use of `if/elif/else` blocks. I have used the latter in my code. To explain how the modified interface should work consider two screen shots from my Jupyter notebook. Note a static phase rotation of 20° is also present and the SNR = 20 dB.

```
#             N=5,M=5,mu=1e-3,Nstart=500
DFE_i = DFE_inputs(5,5,0.0,Nstart=0) # 2*5+1 taps, mu = 0.005
#         def shaped_psk_DFE_EQ_sim(SNR,N_symb,mod,rot_deg,timing,DFE_i,pulse,beta=0.35,DFE=False)
Pe,errors,N_RecBits,z,DFE_o = shaped_psk_DFE_EQ_sim(20,1000,4,20,15,DFE_i,'src',beta=0.35,DFE=True)
print('Pe Est = %1.2e' % Pe)
print('B_RecBits = %d, errors = %d' % (N_RecBits,errors))
```

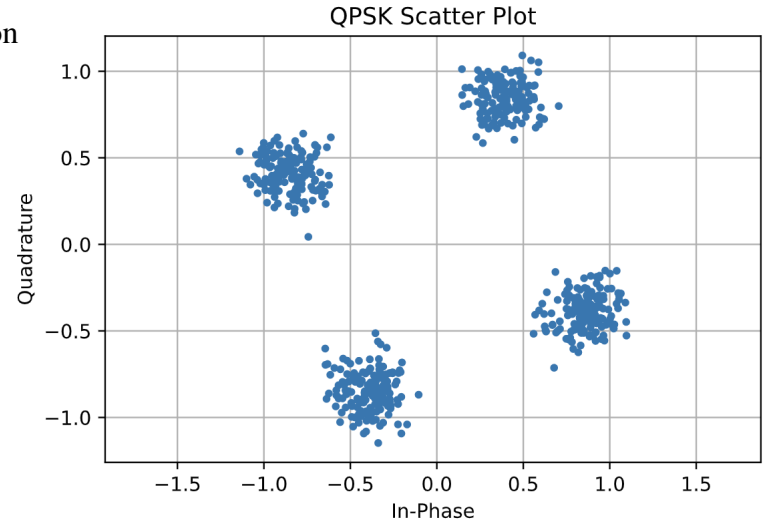
```
Pe Est = 0.00e+00
B_RecBits = 1966, errors = 0
```

Add the rotation after the channel filter

```
DFE_o.Aopt
```

```
array([0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j,
       0.+0.j, 0.+0.j, 0.+0.j])
```

No Equalization since mu = 0.0



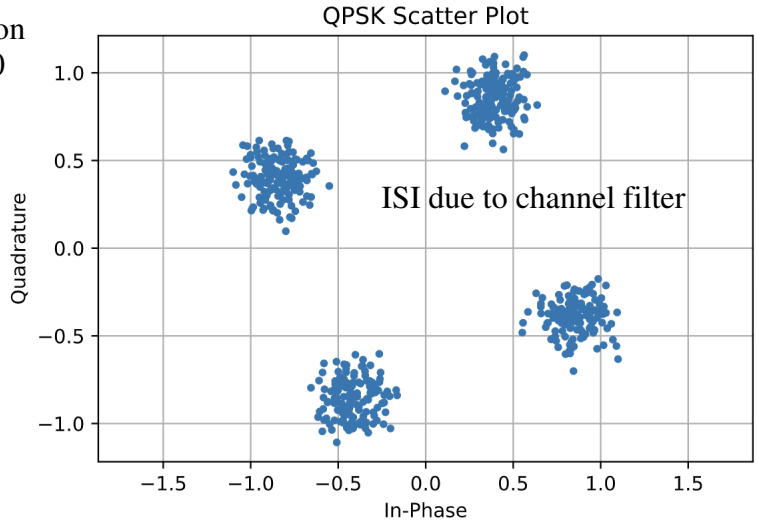
```
#              N=5,M=5,mu=1e-3,Nstart=500
DFE_i = DFE_inputs(5,5,0.0,Nstart=0) # 2*5+1 taps, mu = 0.005
#          def shaped_psk_DFE_EQ_sim(SNR,N_symb,mod,rot_deg,timing,DFE_i,pulse,beta=0.35,DFE=False)
Pe,errors,N_RecBits,z,DFE_o = shaped_psk_DFE_EQ_sim(20,1000,4,20,15,DFE_i,'src',beta=0.35,DFE=False)
print('Pe Est = %1.2e' % Pe)
print('B_RecBits = %d, errors = %d' % (N_RecBits,errors))
```

```
Pe Est = 0.00e+00
B_RecBits = 1966, errors = 0
```

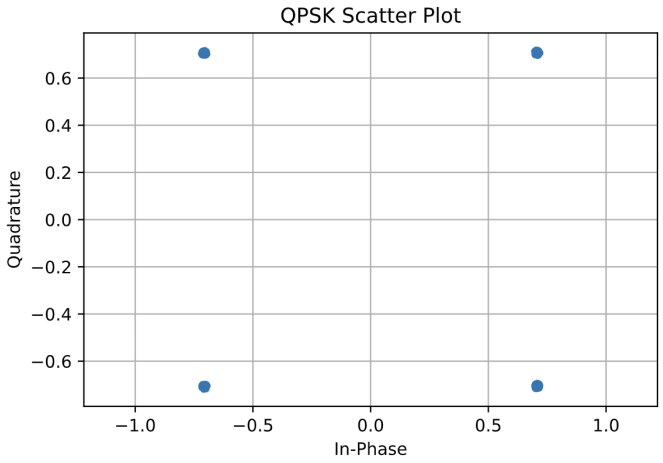
```
DFE_o.Aopt
```

```
array([0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j,
       0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j])
```

No Equalization since mu = 0.0

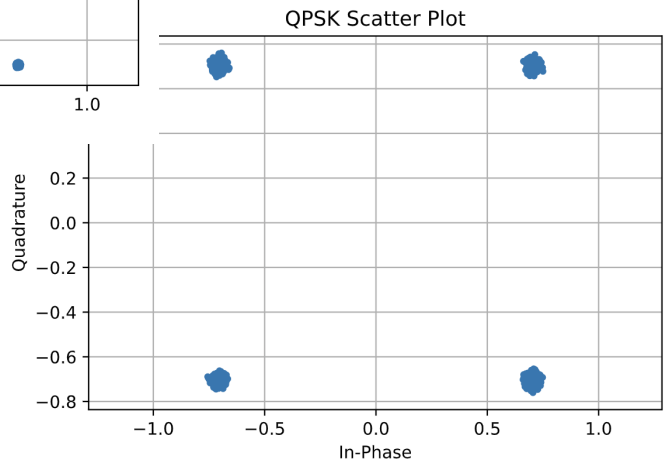


Expect the DD to perform better than the DFE at high SNR, e.g.,



DD at 50 dB, mu = 0.008

Note the equalizer removes the phase rotation.

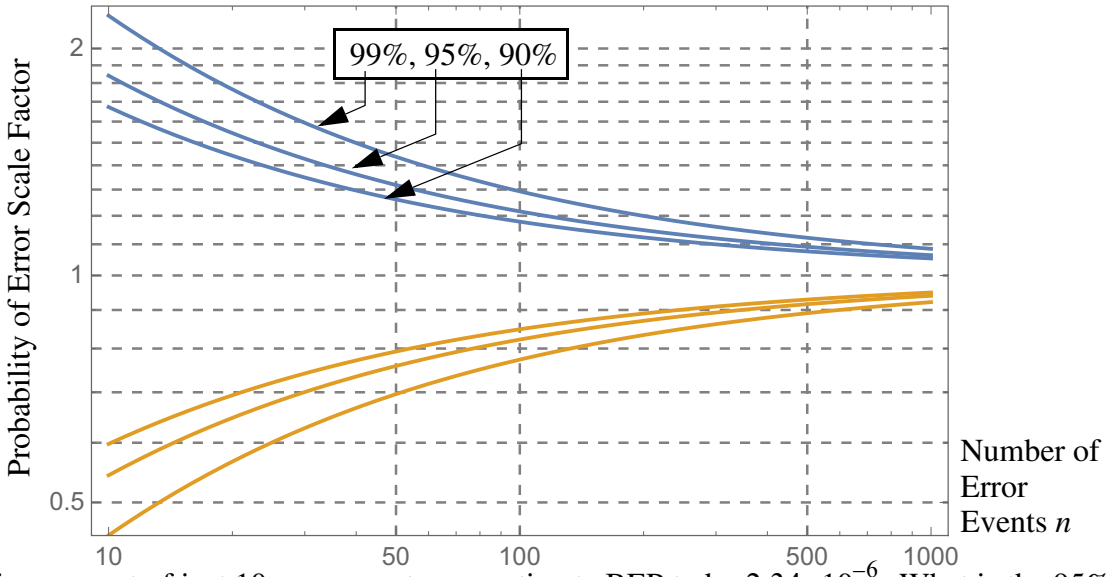


DFE at 50 dB, mu = 0.008

I expect your code to perform similarly. No need for BEP measurements in this problem.

10 pts. 5.) A two-path channel consists of a direct path and one delayed path. The delayed path has a delay of 5 μs. The channel can be considered flat fading if the phase shift introduced by the delayed path is 10 degrees or less. Determine the maximum bandwidth for which this flat-fading assumption holds. Show your work.

6.) In this problem you answer some questions about confidence intervals as described on Page 5-143–146 of the notes. Use the plot of probability of error scale factor versus the number error events counted shown below to find your answers. Create your own version of the chart in a Jupyter notebook if you wish.



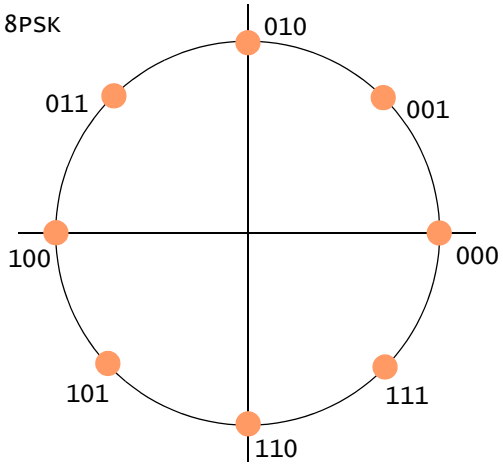
5 pts. a.) Using a count of just 10 error events you estimate BEP to be 2.34×10^{-6}. What is the 95% confidence interval surrounding P_E? Show your work.

5 pts. b.) Using a count of 300 error events you estimate BEP to be $4.22e^{-5}$. What is the 99% confidence interval surrounding P_E? Show your work.

10 pts. 7.) Explain in words the significance of Nyquist pulse shaping. Explain also the significance of the pulse shape parameter $\alpha = \beta$ (in the text).

10 pts. 8.) Using the Gray coding scheme defined in Z&T text problem 9.32, Gray code the natural bit assignments in the 8-PSK constellation shown below. Show also that the Gray encoded 3-bit words can be decoded using a similar scheme, namely

$$b_1 = g_1 \qquad b_i = g_i \oplus b_{i-1}$$



10 pts. 9.) Quadrature multiplexing (QM) is the basis for QPSK and other, yet to be discussed, higher-order digital modulation schemes. Give one pro and one con of QPSK over simple BPSK.

Comments

1.) No comments at this time.

2.) No comments at this time.

3.) No comments at this time.

4.) Depending upon DD or DFE mode I have arrange the Aopt array and corresponding combined filter states as follows:

DD: Aopt(initial) = [{N zeros} 1 {N zeros} {M zeros, but not used for DD}]

DFE: Aopt(initial) = [{N zeros} 1 {M zeros used for DFE}]

5.) No comments at this time.

6.) The accuracy of the included graph is OK for working this problem.

7.) No comments at this time.

8.) No comments at this time.

9.) Paper analysis is sufficient for this scheme, but if you want to have Python Gray encode and decode, consider the helper functions below which work back and forth between decimal representations of *B*-bit binary words and Numpy binary arrays

```
def to_bin(data, width):
    """
    Convert an unsigned integer to a Numpy binary array with the first
     element the MSB and the last element the LSB.
    """
    data_str = bin(data & (2**width-1))[2:].zfill(width)
    return [int(x) for x in tuple(data_str)]


def from_bin(bin_array):
    """
    Convert binary array back a nonnegative integer. The array length is the bit width.
    The first input index hold the MSB and the last holds the LSB.
    """
    width = len(bin_array)
    bin_wgts = 2**np.arange(width-1,-1,-1)
        return int(np.dot(bin_array,bin_wgts))
```

```
>>> M = 32
>>> # Get a bit array from a decimal index in a modulator or
>>> bit_array = to_bin(7,int(log2(M)))
>>> bit_array

[0, 0, 1, 1, 1]

>>> # For encoding you may grab log2(M) bits from a serial bit stream
>>> # For decoding you may start with a decimal value from a detector
>>> # Convert back to a bit array for Gray decoding and ultimately re-serializing
>>> from_bin(bit_array)

7
```