ECEN 310: Communications Engineering  
Lab: M-PSK Error Rate Performance

We will study the Symbol Error Rate (SER) performance of PSK systems. In the first part we will confirm the theoretical results derived in class obtained with ideal assumptions of zero timing offset, perfect coherent demodulation (no phase and frequency offset).

In parts two and three we will see how timing offset and phase error, respectively, degrade the SER performance.

The SER plots will be obtained by means of a Monte Carlo simulation. Essentially, we will generate a sequence of data, simulate a transmission over a (in our case AWGN) channel, and count the number of errors at the receiver. This is a common way of evaluating error performance, especially in complex systems where simple analysis is not possible.

1. We will start with an M-PSK system operating in an AWGN channel. For part one we will assuming a Nyquist pulse shaping with perfect timing synchronisation as well as coherent demodulations (no phase nor frequency offset). In this case, we simply have the complex received signal samples given by

\[ r_k = s_k + n_k \]  (1)

where \( s_k \) are the transmitted symbols (\( \pm 1 \) for BPSK, \( e^{j2\pi m/(M-1)} \) for MPSK), \( n_k \) is the complex AWGN noise with zero mean and variance \( \sigma_n^2 = N_0/2 \) (in other words, \( n_k = \mathcal{N}(0, N_0/2) \)). We will define the Signal-to-Noise Ratio (SNR) as

\[ \text{SNR} = \frac{E_s}{N_0} \]  (2)

where \( E_s \) is the average symbol energy. Typically when simulating a digital system we set \( E_s = 1 \) and vary the SNR by adjusting \( N_0 \).

(a) Generate \( N_s = 1000 \) BPSK data symbols \( s_k \) - the simplest way to generate a sequence of symbols drawn randomly from a constellation is to use the \texttt{randsrc} command. To do this, define an alphabet, which in the case of BPSK is simply a vector \([1 \ -1]\). Corrupt them by AWGN. To do this, generate vector of \( N_s \) Gaussian distributed complex noise samples using the \texttt{randn} command. Note that \texttt{randn} will return a zero mean, unit variance sequence, so you must scale it to achieve a variance of \( N_0/2 \).

(b) Plot the transmitted signal constellation and the received symbols for SNR values of 0, 5, 10 and 15 dB (in decibels). Put all four plots on a single matlab figure (using \texttt{subplot}). Mark the transmitted and received symbols with different markers. Observe the effects of SNR on the received constellation.

(c) Repeat the above for QPSK, 8-PSK and 16-PSK modulations (\( M=4, M=8 \) and \( M=16 \)). Your alphabet will now be a set of \( M \) equally spaced points on a unit circle in a complex plane.
Let’s move on to the receiver, starting with BPSK. Implement a decision device to estimate the received symbols, which for BPSK is simply a matter of comparing it to the threshold of zero. Generate the estimates \( \hat{s}_k \) of the data symbols \( s_k \), and count the number of symbol errors. One way to do this is to use the \texttt{nnz} command on a difference vector \( \hat{s}_k - s_k \). The SER will simply be the number of errors divided by \( N_s \).

Generate SER for QPSK, 8-PSK and 16-PSK modulations. You will have to modify the decision device, as you are no longer comparing to a threshold but deciding depending on which region the received data point fall in. The easiest way to do this is to find the constellation point closest to the received symbol. To achieve this, use \texttt{min} command to find the minimum distance and the associated index of the constellation vector:

\[
[d,i]=\text{min}(\text{abs}(r(n)-\text{alphabet}))
\]

is the estimate. Then the \( \hat{s}_k \) will be the \( i \)th point in the constellation.

Now generate SER curves, that is \( \log_{10}(SER) \) versus \( SNR_{dB} \). It might be wise to put the code generated so far in a function \texttt{getSER(M,Ns,SNRdB)} that is one which takes in the MPSK order \( M \), number of data symbols \( N_s \) and the SNR in dB, and returns the SER. Run the simulation for \( N_s = 10^4 \) symbols and SNR of -4 to 8 (in steps of 2 to speed up the simulation). Plot the SERs (BPSK, QPSK, 8-PSK and 16-PSK) on a logarithmic scale versus the SNR in dB. Use the \texttt{semilogy} command.

Compare the above with the theoretical results derived in class - plot the theoretical curves using lines and simulated using symbols all on the same figure. Be sure to label it nicely!

2. Now we will study the effects of a carrier phase error on the SER performance. Assume the carrier at the receiver given by \( c(t) = \cos(\omega_c t + \phi_e) \), where \( \phi_e \) is the phase error relative to the transmitter. Assuming perfect timing and frequency estimation, the effect of \( \phi_e \) is simply a rotation, that is

\[
r_k = s_k e^{j\phi_e} + n_k
\]

(a) Modify your \texttt{getSER} function to include the phase error \( \phi_e \). For SNR of 5 dB, plot the constellation diagrams for BPSK and 8-PSK using \( \phi_e = \pi/16 \) and \( \phi_e = \pi/8 \).

(b) Now generate SER plot for BPSK (SNR of -4:2:8) for \( \phi_e \) of 0, \( \pi/32 \), \( \pi/16 \), \( \pi/8 \).

(c) Compare to the theoretical curve, given by

\[
\text{SER} = Q\left(\sqrt{\frac{2E_s}{N_0}} \cos \phi_e \right)
\]

(d) Repeat b for 8-PSK. What happens for \( \phi_e = \pi/8 \)?

3. Having considered the effect of a phase offset, let us now see how a timing offset impacts the SER performance. In order to do this we will have to implement Nyquist pulse shaping, specifically a Raised Cosine filter. Typically the filtering would be split between the transmitter and a receiver, each using a root raised cosine filter. To simplify matters, we will filter the sequence of transmitted symbols using the overall raised cosine filter.

(a) Generate a raised cosine pulse. You can use \texttt{rcosfir(b,D,Rs,T)} where \( b \) is the rolloff (excess bandwidth) factor, \( D \) is half the duration of the filter (measured in symbol intervals), \( Rs \) is the upsampling rate and \( T \) is the symbol duration. For simplicity let us set \( T = 1 \). Thus, \texttt{rcosfir} will then return a vector of length \( 2DR_s + 1 \). Plot (using \texttt{stem}) the filter response for a rollof of 0.5, \( D = 2 \) and \( R_s = 16 \)

(b) Modify the \texttt{getSER} function you wrote to include pulse shaping and sampling. Temporarily, comment out the AWGN code, as we will start with noiseless transmission. Starting with a small sequence of \( N_s = 10 \) BPSK symbols, filter a sequence \( s_k \) using the RC filter obtained above. Note that you will have to upsample (insert zeros between the symbols) as you are using
a filter sampled at $T_s = T/R_s$. Using the \texttt{upsample} function, upsample the data symbols by $R_s$. Then use \texttt{filter}(g,1,x) to pass the upsampled sequence $x$ through the RC filter, denoted by $g$. Generate a stemplot of the filtered data stream.

(c) Now implement the sampler at the receiver by downsampling the data by $R_s$ using \texttt{downsample}(y,Rs,Te). Here, $y$ is the received data stream, and $Te$ is the \textit{timing offset}. Start with $Te = 0$ and examine the received data sequence. Note that

- Filtering (ie convolution) resulted in a sequence of length $N_s + D$ (following sampling), with the first $D$ symbols invalid.
- Since the RC filter is necessarily causal, its peak is delayed by $D$ symbol intervals. Thus, our received samples are delayed by $D$. In addition, we also 'lose' the last $D$ samples.

(d) Our solution to the above will be to transmit an additional $D$ data symbols, that is $N = N_s + D$. After downsampling, strip off the first $D$ symbols in the received sequence and the last $D$ symbols in the transmitted sequence: $s=s(1: end-D)$ and $r=r(1+D: end)$. Now you can put the transmitted and received symbols side by side in a column vector and easily compare the sequences. Make sure that you can obtain perfect data at the receiver (with no noise and timing error) before proceeding.

(e) Now compare the received samples for a fixed timing offset. Set $Te$ in the \texttt{downsample} function to 1, 2, 4. Assuming $Rs = 16$, this corresponds to a sampling error of $T/16$, $T/8$ and $T/4$.

(f) Following the sampling, you can add the AWGN, as in parts 1 and 2. Our decision device and error counter stays the same. Note that you are still using $N_s$ symbols to compute the SER.

(g) Add $Rs$ and $Te$ as parameters to your \texttt{getSER} function and generate SER plots for BPSK for a sampling error of $T/16$, $T/8$ and $T/4$. Run the simulation for $N_s = 1e4$ symbols.

(h) Repeat for QPSK and 8-PSK.