EE4140 Assignment 1 Report

Kishore Rajendran [EE20B064]

Q1) In this question, we observe the output samples obtained by passing a BFSK input sequence through a Raised Cosine (RC) filter.

The RC pulse is a widely used pulse, as it satisfies the Nyquist Criterion for zero ISI (Inter Symbol Interference). But the catch here, is that we're only considering a finite set of samples for this pulse.

1.1 and 1.2 are the same $x(kT_s)$ plots except that L = number of samples taken for the RC filter (2LT symbol durations) is taken as 2 and 4.

↳ Clearly, as L increases, the pulse shape's bandwidth becomes narrower, reducing spectral leakage. Hence this lower BW is better for (higher L). This is also true because, as we increase L, the pulse shape used is more closer to the actual RC pulse, hence reducing ISI.

This is clearly visible from the obtained symbol and ISI plots shown in <Fig 1>.

1.3, we vary the excess BW factor $\beta \Rightarrow 0, 0.5, 1$. We know from the eq's that a higher β means faster decay in time domain. This is evident from the plots, as the signal looks flatter around the reqd. samples as β increases. Also the plot for $\beta = 1$ remains bounded between -1 and 1 unlike the other two, which allows for better decoding. (Zero ISI)

Please refer to <Fig 2>

NOTE) In the time domain definition of the RC pulse, at $t=0$, sinc must be defined as 1 and the points where the denominator $(1-(2\beta t)^2)$ goes to 0 must also be redefined with the limit in order to avoid 'inf' & 'nan' values in our simulation.

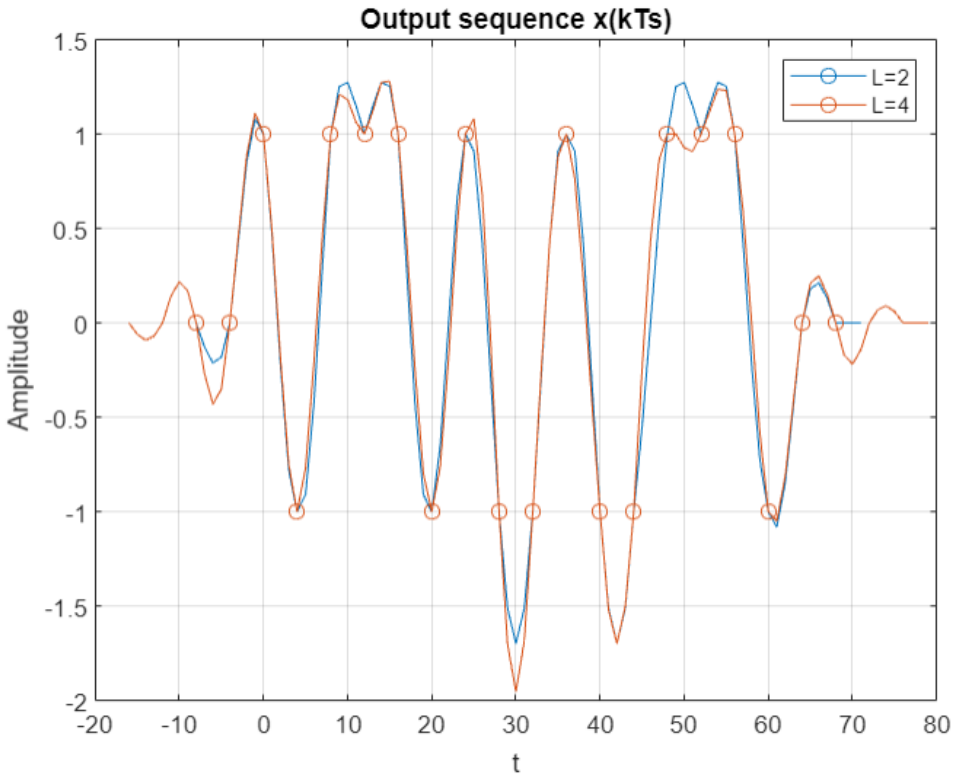
1.4, Periodogram

↳ This is a computationally economic way of estimating the PSD. Instead of computing the auto-correlation of the sequence, we use Monte-Carlo simulations and averaging to estimate the PSD of our output sequence.

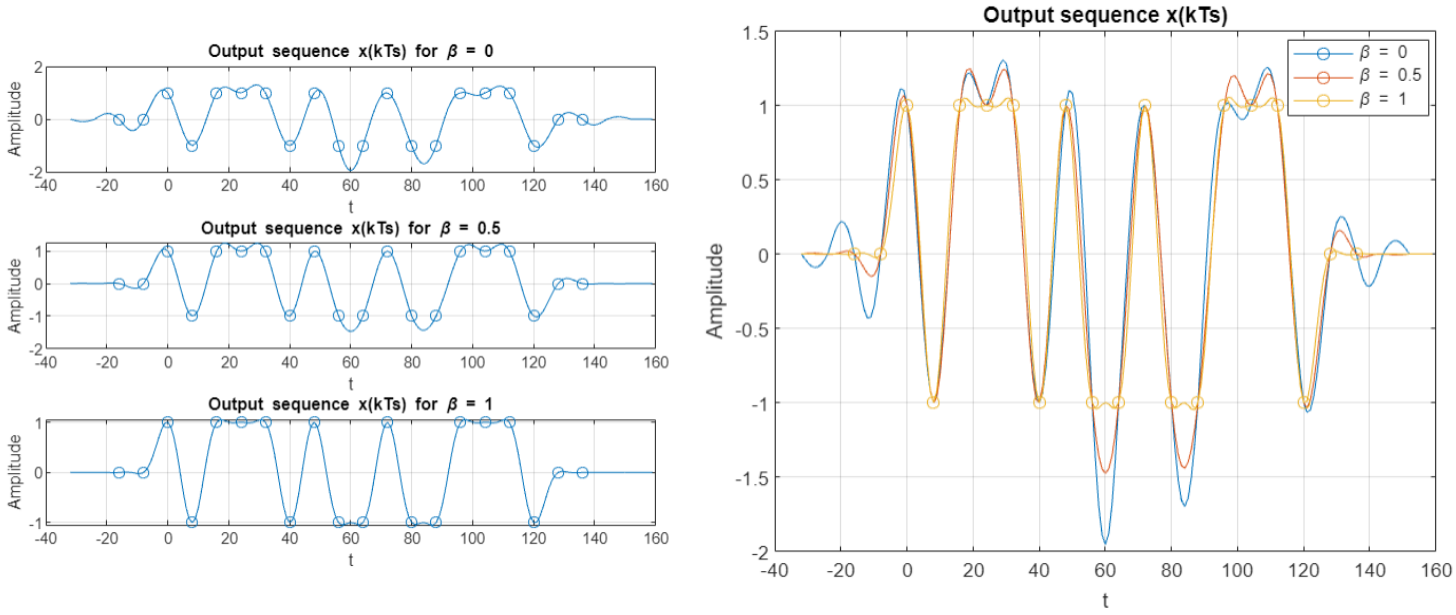
The steps taken to obtain this plot are mentioned in the Question very clearly.

The plots are as expected, PSD of the rectangular filter looks like the sinc^2 function with one primary peak and other small peaks across the spectrum. The PSD of the RC pulse with $\beta=0.5$, on the other hand, is smoother and has a limited Bandwidth. Please refer to <Fig 3>

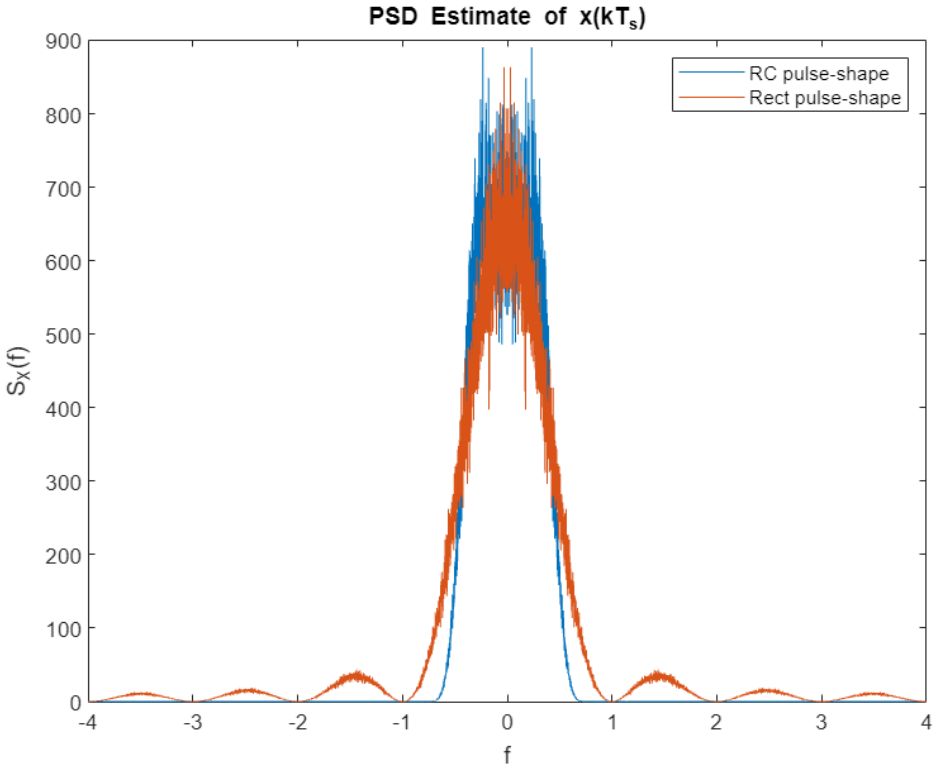
EE4140 Assignment-1 Plots



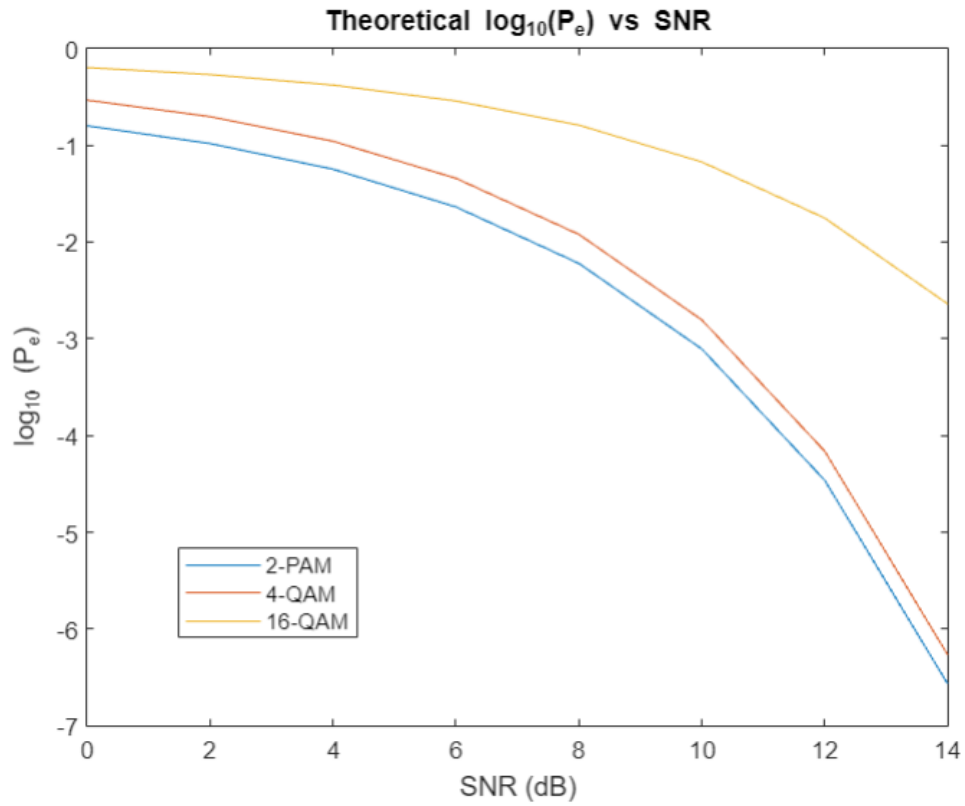
[Fig 1: Output Sequence x(kTs) for L = 2, 4]



[Fig 2: Output Sequence x(kTs) for β = 0, 0.5, 1]



[Fig 3: PSD estimate of the Output Sequence x(kTs)]



[Fig 4: Theoretical log10(Pe) vs SNR for BPSK, QPSK, 16-QAM]

Q2) In this question, we compare P_e computed using various approximations, simulated SER with the theoretical values for 2-PAM, 4-QAM, 16-QAM

* Theoretical P_e (Considering $E_b = 1$ for all signal sets)

2-PAM	4-QAM	16-QAM
$\mathcal{E}_{s_2} = d_1^2$	$\mathcal{E}_{s_4} = 2d_2^2$	$\mathcal{E}_{s_{16}} = 10\, d_3^2$
$\mathcal{E}_{b_2} = d_1^2$	$\mathcal{E}_{b_4} = d_2^2$	$\mathcal{E}_{b_{16}} = 10/4\, d_3^2$

$\mathcal{E}_{b_2} = \mathcal{E}_{b_4} = \mathcal{E}_{b_{16}} = 1 \Rightarrow d_1 = 1,\ d_2 = 1,\ d_3 = \sqrt{0.4}$

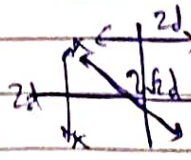
$P_e = q$	$P_e = 2q - q^2$ $= 1 - (1-q)^2$	$P_e = 3q - 2.25q^2$ $= 1 - \frac{1}{16}\left(4(1-q)^2 + 8(1-q)(1-\frac{3}{4}q) + 4(1-\frac{3}{4}q)^2\right)$

where $q = \frac{1}{2}\operatorname{erfc}\left(\frac{d}{\sqrt{2}\sigma}\right)$ with d as calculated above

These expressions are plotted over a range of SNR values by varying the variance σ^2 in <Fig 4>

2.2 Now, we explore the various approximations used to obtain bounds on P_e for 4-QAM

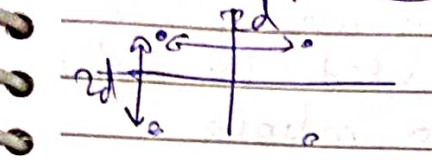
<2.2.1> Union Bound using all pairwise symbol errors



$$P_e = \frac{1}{4}\left(4 \times \left[\frac{1}{2}\operatorname{erfc}\left(\frac{d}{\sqrt{2}\sigma}\right) + \frac{1}{2}\operatorname{erfc}\left(\frac{d}{\sqrt{2}\sigma}\right) + \frac{1}{2}\operatorname{erfc}\left(\frac{d\sqrt{2}}{\sqrt{2}\sigma}\right)\right]\right)$$

$$= \operatorname{erfc}\left(\frac{d}{\sqrt{2}\sigma}\right) + \frac{1}{2}\operatorname{erfc}\left(\frac{d\sqrt{2}}{\sqrt{2}\sigma}\right)$$

2.2.2 Union bound using only the nearest neighbours



$$P_e = \frac{1}{4} \times \left(4 \times \left[2 \times \frac{1}{2} \operatorname{erfc}\left(\frac{d}{\sqrt{2}\sigma}\right)\right]\right)$$

For 4-QAM

$$= \operatorname{erfc}\left(\frac{d}{\sqrt{2}\sigma}\right)$$

→ do note that this d is taken acc. to prev. pg.

2.2.3 Chernoff Bound

$$P_e = \operatorname{erfc}\left(\frac{d}{\sqrt{2}\sigma}\right) = e^{-\frac{d^2}{2\sigma^2}}$$ .... The chernoff bound on erfc f^n

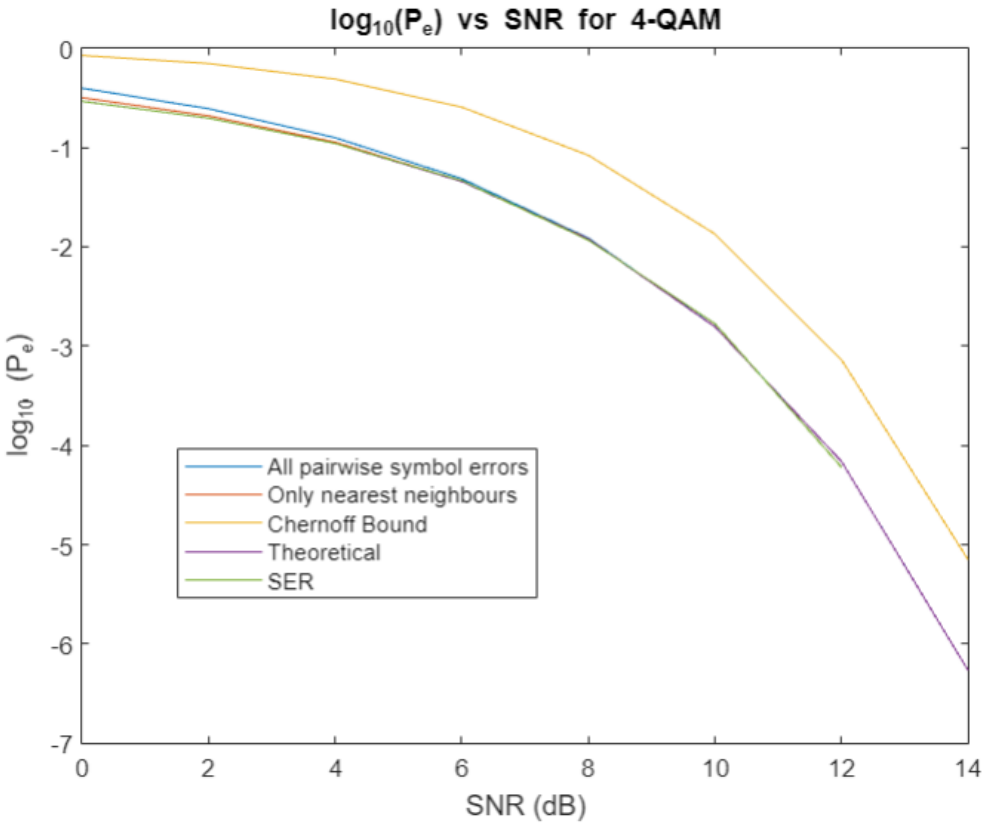
These bounds are plotted across values of SNR as asked in <fig. 5>

All these bounds, except the Chernoff bound are almost the same and very close to the theoretical value. We can only make out a difference at low values of SNR where the nearest neighbours is the tightest bound and then the full pairwise bound. Chernoff bound is a very loose bound compared to these.

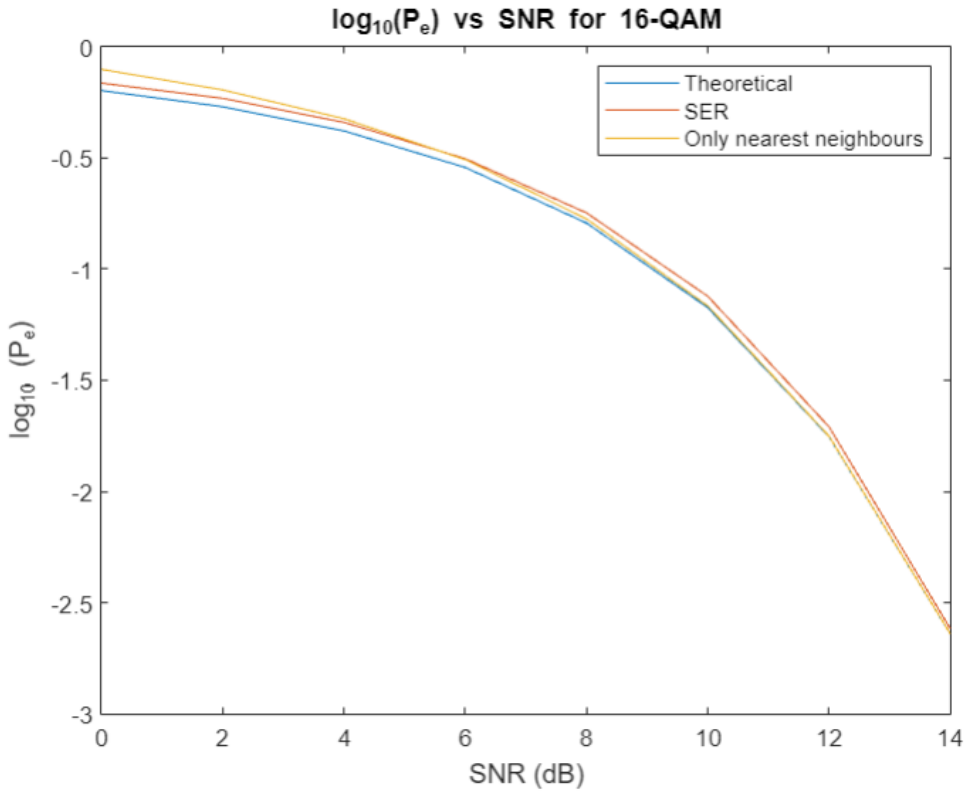
NOTE) MATLAB can calculate the erfc f^ for us, that's how these plots are obtained.

2.3 SER simulation

We generate 10^5 tx samples among $(\pm d \pm jd)$ using uniform RV's between $[0, 1]$ and map a value between 0 – 0.5 to $-d$ and 0.5 – 1 to $+d$, for both real and imaginary parts separately



[Fig 5: Various Pe bounds and simulated SER for 4-QAM]



[Fig 6: Nearest Neighbours Pe bound and simulated SER for 16-QAM]

2.3 cont'd

Next, we generate 2 separate gaussian r.v's with variances according to the SNR's we are plotting for, to obtain noise which is added to both real and imaginary parts of the Tx symbols.

Then, we decode these received symbols using the ML decision boundaries ⇒ (1 line eqn Gaussian Noise) (Bisector eqn equiprobable)

ie) Decisions are made based on the axes in this case, as they are the perpendicular bisectors between our symbols. This can easily be done by comparing the signs of the real & imaginary parts of the R_x symbols.

We then compare these with the original Tx symbols to obtain an error count and hence the SER.

NOTE) In the plots, when after a certain SNR, the plot disappears, it's because P_e has gone to 0.

This SER matches with our previous P_e plots as can be seen in <Fig 5>.

2.4 For 16-QAM

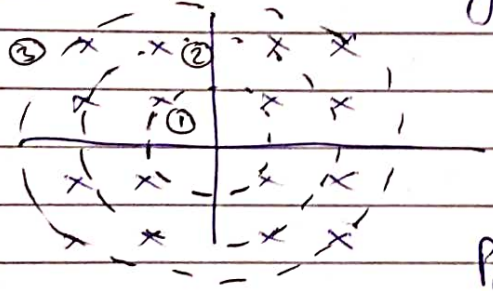
We repeat the same but for 16-QAM now. Here, we have 4 sets of values possible in both the real and imaginary parts.

⇒ Thus, we take 2 uniform r.v's like before (bet^n 0 to 1). If $0 \le rv < 0.25 \Rightarrow -3d$. $0.25 \le rv < 0.5 \Rightarrow -d$; $0.5 \le rv < 0.75 \Rightarrow d$. $0.75 \le rv < 1 \Rightarrow 3d$ for both real and img. parts separately.

Once we generate the Tx symbols, we add noise to get the Rx symbols, which we again decode using the ML decision boundaries.

The obtained plot is shown in <Fig 6>.

2.5 Nearest Neighbour Bound for 16-QAM



As shown, there are 3 different sets of points, each having a different no. of nearest neighbours.

$$P_{e,nn} = \frac{2q \times 4 + 3q \times 8 + 4q \times 4}{16} = 3q$$

where q is defined similar to previous parts of the Q.

This is plotted in <Fig 6> as well, and we can see that all the obtained P_e plots are very nearby and accurate values.

Q3) In this question, we're trying to analyse the performance of an MLSE based approach to decode the samples distorted by an ISI channel, which is known to us.

We implement the **Viterbi Algorithm**, which is an optimal MLSE estimator. Here are the steps I took to implement this algorithm:

① Precalculate $\hat{x}$, (the noiseless hypotheses), as they do not change as we move ahead in the trellis.

② For the first L-1 steps (L = filter length), compute the transition metrics and hence the cumulative metric, then populate the survivor sequence's first L-1 columns.

③ Now, run a loop for the now constant sized M^{L-1} trellis, to find the transition metric (TM) & cumulative metric (CM) using CM = CM + TM.

④ I'm first finding TM for all the branches exiting each node, adding the corresponding CM, to give the array 'comp', which I then rearrange to give the array 'comp_enter' which holds the TM+CM metric of all the branches entering a node.

⑤ Hence, we can easily compare these branches, find the minimum CM+TM, update the survivor sequence for each node and the CM array accordingly.

⑥ Now, to implement decoding delay, I maintain an array ss_final which is the MLSE sequence being computed with a delay δ, without having to wait till the very end.

⑦ Once we cross a length of del in the trellis, we can start populating ss_final and pruning the survivor sequences stored for each node. This decision for ss_final is taken by comparing the cumulative metrics.

⑧ We repeat the above steps for a range of noise variances and a range of decoding delays to obtain the plot (Fig. 7).

*Comments

① The tail symbols are used to obtain the sequence estimate for the last 'δ' symbols, as this approach is better than picking the least CM sequence

② Implementing decoding delay 'δ', hugely improves run-time of the MATLAB code, since we're now only maintaining a fixed size of survivor sequences by pruning the rest. Although these are sub-optimal decisions, the difference isn't huge (in error probabilities) but runtime is significantly improved.

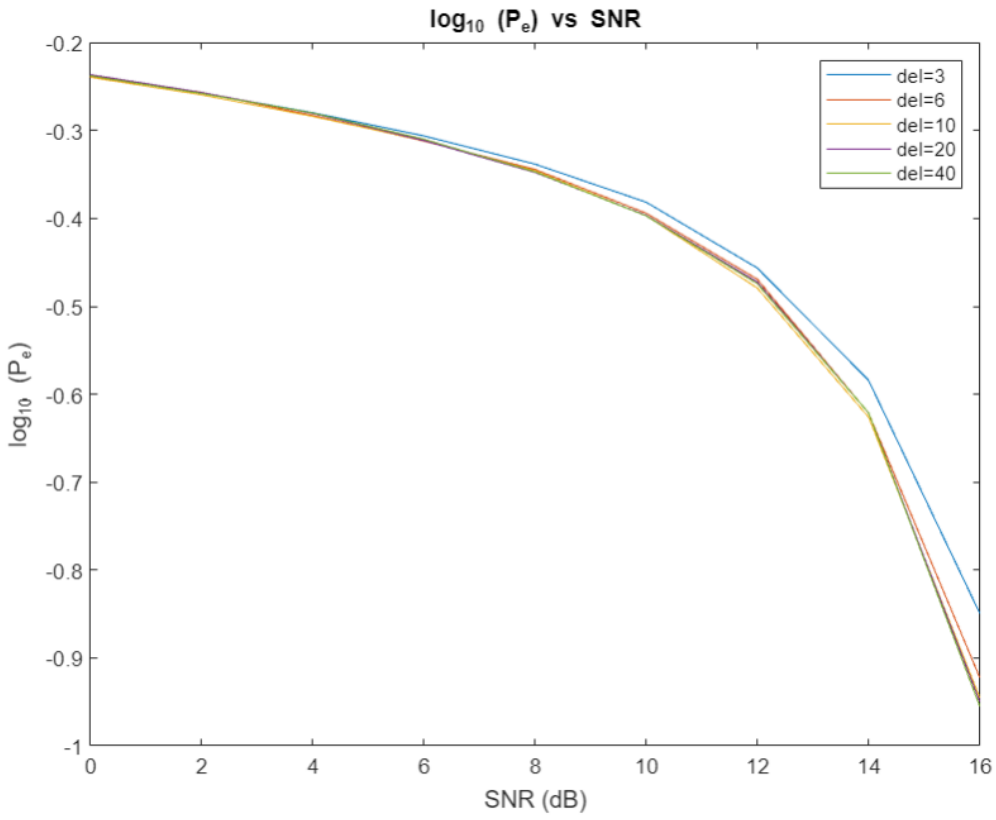
③ As expected, higher the decoding delay 'δ', better the performance of our estimator.

NOTE) SNR is given by $1/\sigma_v^2$ here, because SNR $= \frac{E[|I(k)|^2]}{\sigma_v^2}$ and $E[|I(k)|^2] = 1$ (given) $\Rightarrow \therefore$ SNR $= \frac{1}{\sigma_v^2}$

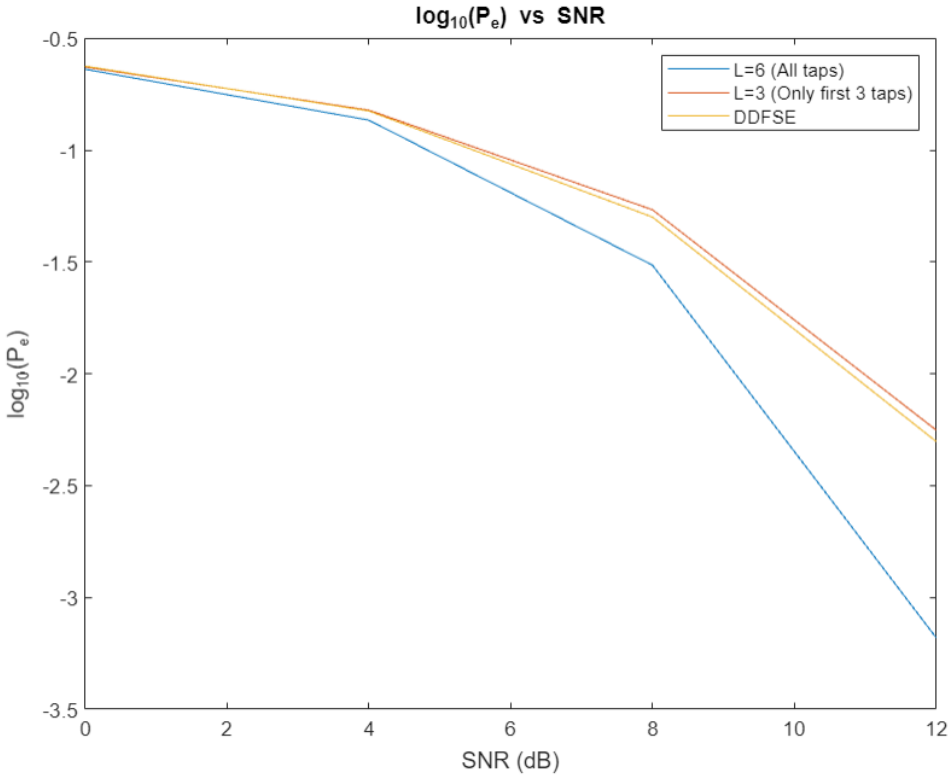
④ SER, calculated over the unknown symbols only (ie: ignoring the tail symbols), clearly decreases with an increase in SNR. This is quite obvious, since with an increase in SNR, our noise reduces, allowing for better performance in decoding by our VA estimator.

⑤ We're able to use an MLSE based approach to estimate this sequence only because the noise is gaussian and the channel is being modeled as an FIR filter. The Viterbi algorithm allows us to implement an MLSE estimator by maintaining only the top M^{L-1} sequences instead of comparing the M^N possible sequences!

NOTE) I have generalized the code for the Viterbi Algorithm to run for any M-PAM and a filter of L taps.



[Fig 7: SER for various decoding delays in the Viterbi Algorithm]



[Fig 8: SER for Under-modeled taps and DDFSE in the Viterbi Algorithm]

Q4) Here, we implement the same Viterbi algorithm for a generalized case of M-ary PAM and an L-tap filter.

(1) Normalizing the channel gain to unity:

$$\frac{1}{C^2}\left(1^2 + 0.9^2 + 0.5^2 + 0.1^2 + 0.2^2 + 0.1^2\right) = 1$$

This gives $\boxed{C = 1.4916}$

In parts (2) and (3), we compare the performance of a regular 2^5 state VA ($M^{L-1} = 2^5$) and a VA with a truncated impulse response (considering only the first 3 taps, since the other 3 taps are weaker), both with a decoding delay (δ) = 30 for a 2-ary PAM.

* Clearly, ignoring the 3 taps will give a worse result, but since the VA now only has 2^2 states instead of 2^5, the runtime of the MATLAB code improves significantly. This is a sub-optimal approach, but gives decent SER.

* Since in (3), we ignore these last 3 taps completely for our decoding process, it can be treated as noise! But this noise starts dominating over $V(k)$ as SNR increases, hence as SNR increases, the 3-tap response starts performing worse, which justifies the increasing gap between the 2 plots of SER for $L=6$ and $L=3$. (Fig 8)

(4.4) Now, we Implement a decision feedback mechanism.

Instead of ignoring the last 3 taps completely, we can use the current survivor sequence at every node, to get an estimate of the contribution of these taps, which can be subtracted from the actual received samples. Again, this is a sub-optimal approach.

In our case, the decision feedback $\mathcal{E}_l^{(k)}$ is given as $\mathcal{E}_l^{(k)} = \sum_{i=4}^{6} f_i \hat{I}_l(k-i)$ where $\hat{I}_l$ = survivor sequence at a node.

We do $r(k) - \mathcal{E}_l^{(k)} \Rightarrow$ To get $\bar{r}_l(k)$ which is now only dependent on the first 3 taps which is being used in our Viterbi decoder.

As expected, the DDFSE approach works better than ignoring the last 3 taps for reasons mentioned above and the fact that this method is obviously better than undermodelling. Again obviously, the result obtained in (2) where we consider all the taps, will be much better, but will be more computationally expensive, as discussed before.