EE4140 Assignment 2 : L-MMSE based Equalisation

Kishore R, EE20B064

1 Introduction

In this assignment, we will compare the performances of a Linear Equaliser setup using 2 methods: Stastical Weiner Filter (SWF) and Time-Averaged Weiner Filter (TAWF) across various orders (N) and decoding delays (Δ), with a Decision Feedback Equaliser of the same order and also with the Viterbi decoder implemented in the previous assignment.

2 Part (a) : Linear Equalizers

Linear MMSE equalizers are used to remove the distortion effects of ISI created by the channel. In order to estimate the taps of this equalizer, we minimise the mean square error metric, solving which gives us the Weiner Filter. Here, we consider the Stastical Weiner Filter for parts (a1)-(a5) and the Time-Averaged Weiner Filter for parts (a6)-(a7) for the defined channel $F(z) = \frac{1}{\sqrt{2}}(0.8 - z^{-1} + 0.6z^{-2})$ at SNR=10dB.

2.1 Statistical Weiner Filter (SWF)

- SWF uses the **true statistics** of the transmitted symbols and noise to setup the correlation matrices R and p.
- The *auto-correlation matrix* $\mathbf{R} = E[\mathbf{y}(k)\mathbf{y}^T(k)]$ is an NxN symmetric toeplitz matrix, only dependent on the channel taps, signal and noise variances (s_var and n_var), and is independent of the decoding delay (Δ). We generate R for our case of L=3, using the `toeplitz()` command in MATLAB, as shown:

```
t1 = filter(1)*filter(2) + filter(2)*filter(3);
t2 = filter(1)*filter(3);
R = (toeplitz([s_var+n_var, t1*s_var, t2*s_var, zeros(1, N-L)])).';
```

- The *cross-correlation matrix* $\mathbf{p} = E[d(k)\mathbf{y}(k)]$ is an Nx1 matrix which depends on the filter taps, signal variance and the decoding delay (Δ). We generate p as shown:

```
p = [zeros(1, del-L+1), fliplr(filter)*s_var, zeros(1, N-del-1)].';
p = p(end-N+1:end);
```

(a1) N = 3, $\Delta = 0$

We have the auto-correlation and cross-correlation matrices R and p as:

$$R = \begin{bmatrix} 1.1 & -0.7 & 0.24 \\ -0.7 & 1.1 & -0.7 \\ 0.24 & -0.7 & 1.1 \end{bmatrix} \qquad p = \begin{bmatrix} 0.5657 \\ 0 \\ 0 \end{bmatrix}$$

Hence, we get the optimal weights for the Wiener filter as:

$$w_{opt} = R^{-1}p = \begin{bmatrix} 0.9587 \\ 0.8016 \\ 0.3009 \end{bmatrix}$$

Thus, $J_{min} = \sigma_I^2 - p^T w_{opt} = \mathbf{0.4577}$

(a2) N = 10, $\Delta = 0$

Similarly, we get:

$$w_{opt} = R^{-1}p = \begin{bmatrix} 0.9816 \\ 0.8417 \\ 0.3130 \\ -0.0821 \\ -0.2003 \\ -0.1371 \\ -0.0340 \\ 0.0270 \\ 0.0350 \\ 0.0164 \end{bmatrix} \qquad J_{min} = \sigma_I^2 - p^T w_{opt} = \mathbf{0.4447}$$

(a3) N = 10, $\Delta = 5$

Similarly, we get:

$$w_{opt} = R^{-1}p = \begin{bmatrix} 0.0518 \\ 0.1826 \\ 0.2953 \\ 0.1755 \\ -0.3610 \\ 0.5894 \\ 0.6439 \\ 0.3051 \\ 0.0190 \\ -0.0545 \end{bmatrix} \qquad J_{min} = \sigma_I^2 - p^T w_{opt} = \mathbf{0.3369}$$

(a4) Best N and Δ

- From the discussions in class, we know that as N and Δ are increased, J_{min} achieves a minimum value in between and then starts rising again.
- We could have simulated for a large set of values and found the best combination, but the question asks us to use trial and error, hence noting the above trend is important.
- Upon varying the values of N and Δ over a large range, we notice that a global minimum value of J_{min} occurs when N = 82 and Δ = 41
- However, this is a huge value and is highly overmodelling the channel which is merely 6 taps in length. Also, the matrix inverse operation becomes more expensive as the order increases.
- Hence, if we limit the value of N to 20, we get minimum J_{min} **= 0.3315** for **N = 20** and **Δ = 9**.
- The general trend is that the minimum J_{min} occurs for Δ = N/2.
- Also note that, the value of J_{min} remains the same (ie: 0.3315) even for higher N (=82), if we consider precision upto four decimal places.

2.2 Time-Averaged Weiner Filter (TAWF)

- Assuming ergodicity, we obtain time-averaged estimates of the correlation matrices R and p by averaging over 'P' samples or snapshots.
- We use $\mathbf{R} = \frac{1}{P}\sum_{k=1}^{P} \mathbf{y}(k)\mathbf{y}^T(k)$ to obtain the auto-correlation matrix and $\mathbf{p} = \frac{1}{P}\sum_{k=1}^{P} d(k)\mathbf{y}(k)$ to obtain the cross-correlation matrix, for three different values of P = 20, 100 and 500, as shown:

```
for k = 1:P
        y_k = fliplr(y(k:k+N-1)).';
        R = R + (y_k*y_k.')/P;
        p = p + (y_k*dec(k))/P;
end
```

- Here, y_k is a randomly generated 4-PAM symbol sequence of length P with gaussian noise of appropriate variance added to it.
- Please note that since this approach uses Monte Carlo simulations for a small number of samples, the values of J_{min} obtained will vary a lot every time the code is run.

(a6) N = 10, Δ = 5

For the given values of N and Δ, at SNR = 10dB, using the above code to generate the TAWF and J_{min} values for various values of P, we get:

P	20	100	500
J_{min}	0.0639	0.3067	0.3170

Clearly, as P increases, we will get better estimates of J_{min} that are closer to the theoretical value obtained from the SWF approach in part (a3) ie: $J_{min} = 0.3369$.

2.3 SER plots for Linear Equalizers

(a5) SWF, (a7) TAWF

Here, we obtain the SER plots for the SWF and TAWF (P = 500) Linear Equalizers implemented in parts (a4) and (a6) for 50,000 symbols obtained from a 4-PAM constellation by varying SNR from 0dB to 20dB in steps of 2dB.

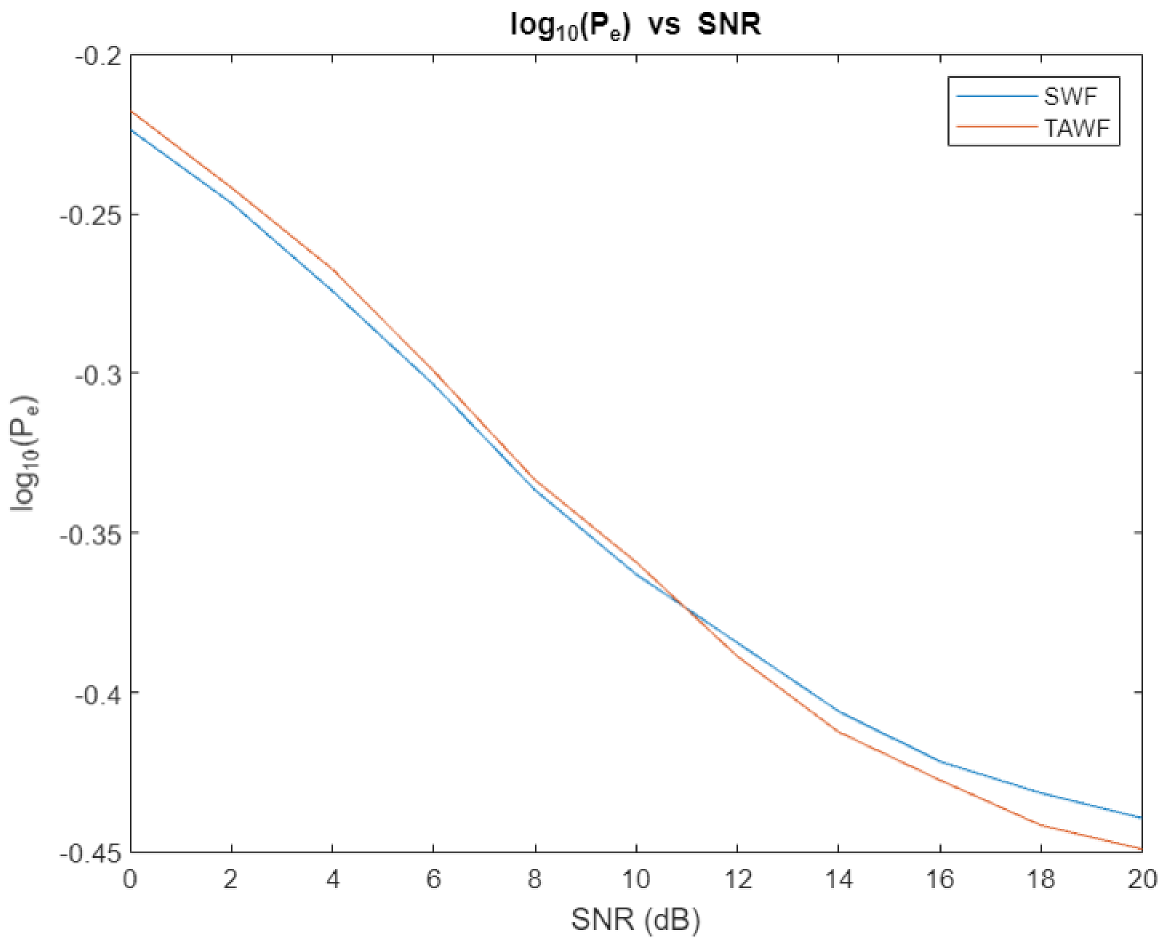

Figure 1: SER plots: SWF and TAWF Linear Equalizers

Clearly, the TAWF plot is very close to the SWF plot, which tells us that our assumptions of ergodicity are indeed right. Also these SER values are quite high compared to the VA plot from the last assignment, which makes sense, since equalization is a sub-optimal approach for MLSE estimation. Note that we are using equalizers defined for one SNR value across all the SNRs, hence the performance is so poor. And this is fine since changing the equalizer taps according to the changing SNR every time, doesn't make practical sense.

3 Part (b) : Decision Feedback Equalizers

A decision feedback equalizer consists of 2 filters:

- **Feedforward filter**: Same as the previously discussed linear equalizer, except that it only handles the precursor portion of ISI
- **Feedback filter**: Removes the postcursor ISI with the help of symbol decisions

Now, we must redefine the correlation matrices. We define the weights matrix as a concatenation of the feedforward and feedback weights, and we expand the output matrix y(k) to make the "appended data vector", which is defined as:

$$\mathbf{a}(k) = \begin{bmatrix} y(k) & y(k-1) & \cdots & y(k-N_1-1) & \vdots & I(k-\Delta-1) & \cdots & I(k-\Delta-N_2+1) \end{bmatrix}^T$$

Thus, we can now define our correlation matrices as: $\mathbf{R} = E[\mathbf{a}(k)\mathbf{a}^T(k)]$ and $\mathbf{p} = E[\mathbf{a}(k)I(k-\Delta)]$.

To generate the $\mathbf{R}$ matrix and make our code simpler, we can continue to think of $\mathbf{a}$ and $\mathbf{a}^T$ as augmented matrices with the y(k) and I(k) samples as separate N_1x1 and N_2x1 matrices, and carry out the calculations in 4 separate parts as described below:

- The auto-correlation matrix
$$R = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}$$
where:
- R_{11} is a symmetric N_1xN_1 toeplitz matrix which is generated using the code from part (a).
- R_{12} is an asymmetric N_1xN_2 toeplitz matrix defined as:

$$R_{12} = -\sigma_I^2 \begin{bmatrix} f_{\Delta+1} & f_{\Delta+2} & \cdots & \cdots & f_{\Delta+N_2-1} & f_{\Delta+N_2} \\ f_{\Delta} & f_{\Delta+1} & \cdots & \cdots & f_{\Delta+N_2-2} & f_{\Delta+N_2-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ f_{\Delta-N_1+3} & f_{\Delta-N_1+4} & \cdots & \cdots & f_{\Delta-N_1+N_2+1} & f_{\Delta-N_1+N_2+1} \\ f_{\Delta-N_1+2} & f_{\Delta-N_1+3} & \cdots & \cdots & f_{\Delta-N_1+N_2} & f_{\Delta-N_1+N_2+1} \end{bmatrix}$$

- R_{21} is just the transpose of R_{12}.
- R_{22} is a diagonal matrix of order N_2 with each of its diagonal elements $= \sigma_I^2$.

```
% Generating matrix R11 (for L=6)
t1 = filter(1)*filter(2) + filter(2)*filter(3) + filter(3)*filter(4) +
    filter(4)*filter(5) + filter(5)*filter(6);
t2 = filter(1)*filter(3) + filter(2)*filter(4) + filter(3)*filter(5) +
    filter(4)*filter(6);
t3 = filter(1)*filter(4) + filter(2)*filter(5) + filter(3)*filter(6);
t4 = filter(1)*filter(5) + filter(2)*filter(6);
t5 = filter(1)*filter(6);
R_11 = (toeplitz([s_var+n_var, t1*s_var, t2*s_var, t3*s_var, t4*s_var,
    t5*s_var, zeros(1, N1-L)])).';

% Generating matrix R12
row = zeros(1,N2);
col = zeros(N1,1);
for i=1:N2
    if(del+i>=0 && del+i<L)
        row(i) = -s_var*filter(del+i+1);
    end
end
for i=1:N1
    if(del-i+2>=0 && del-i+2<L)
```

```
        col(i) = -s_var*filter(del-i+3);
    end
end
R_12 = toeplitz(col, row);

% Generating matrix R21
R_21 = R_12.';
% Generating matrix R22
R_22 = s_var*eye(N2);

% Generating Autocorrelation matrix R
R = [R_11 R_12; R_21 R_22];
```

- The cross-correlation matrix r is defined in the same way as we did in part (a), with the only change that N is replaced by N_1+N_2, as shown:

```
p = [zeros(1, del-L+1), fliplr(filter)*s_var, zeros(1, N1+N2-del-1)].';
p = p(end-(N1+N2)+1:end);
```

(b1) SWF LE for N = 20, Δ = 9

Using code similar to that of part (a), we get:

$$w_{opt} = R^{-1}p = \begin{bmatrix} 0.0332 \\ 0.0527 \\ -0.0008 \\ -0.1142 \\ -0.1694 \\ -0.0587 \\ 0.1421 \\ 0.1668 \\ -0.1610 \\ 0.8104 \\ 0.6359 \\ 0.2702 \\ -0.1074 \\ -0.1549 \\ 0.0149 \\ 0.1437 \\ 0.1233 \\ 0.0220 \\ -0.0409 \\ -0.0327 \end{bmatrix} \qquad J_{min} = \sigma_I^2 - p^T w_{opt} = \mathbf{0.2647}$$

(b2) DFE for $N_1 = 15$, $N_2 = 5$, $\Delta = 1$

Using code similar to that of part (a), we get:

$$w_{opt} = R^{-1}p = \begin{bmatrix} -0.1860 \\ 1.0756 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.7474 \\ 0.3419 \\ 0.1331 \\ -0.1318 \\ -0.0721 \end{bmatrix} \qquad J_{min} = \sigma_I^2 - p^T w_{opt} = \mathbf{0.1604}$$

The J_{min} value, as expected, is lesser than the value obtained in (b1). Our DFE is of the same order (N=20) as the LE. It uses 15 taps for the feedforward filter and 5 taps for the feedback filter which removes the postcursor ISI, allowing for better performance even with a lower decoding delay ($\Delta = 1$).

(b4) Viterbi Decoder with 2^5 states, $\Delta = 30$

For this, we use the Viterbi Algorithm implemented in Assignment 1, for MLSE estimation of 50,004 2-PAM symbols over the given range of SNR values.

3.1 SER plots for Performance Comparison

Here, we compare the performance of the Linear Equalizer, Decision Feedback Equalizer and the Viterbi Decoder implemented in parts (b1), (b2) and (b4), for 50,000 symbols obtained from a 2-PAM constellation using the SER vs SNR plots.

(b3) SER plots: VA vs DFE vs LE

Comments

- Clearly, the VA decoder (Viterbi Algorithm) performs the best, since it is an optimal MLSE decoder, whereas equalizers are sub-optimal.
- The next best performance is shown by the Decision Feedback Equalizer. As explained before, this is because the feedback filter removes the post-cursor ISI, which allows the feedforward part of this equalizer to perform better.
- Lastly, we have the Linear Equalizer, which doesn't have any such feedback filter and must depend solely on its feedforward weights to minimize ISI.
- We can also see that, the performance of both equalizers is almost the same at low SNR, but the DFE starts performing much better at higher SNR values, when compared to the LE. This is because the decision errors that are being subtracted, (which have distortion effects resulting from both ISI and noise), have more effect at high SNR values, since the signal power is now more than that of the noise, and the DFE is able to get rid of this post-cursor ISI, whereas the LE doesn't have this advantage.
- Hence, for the same order, the Decision Feedback Equalizer shows much better performance than the Linear Equalizer.

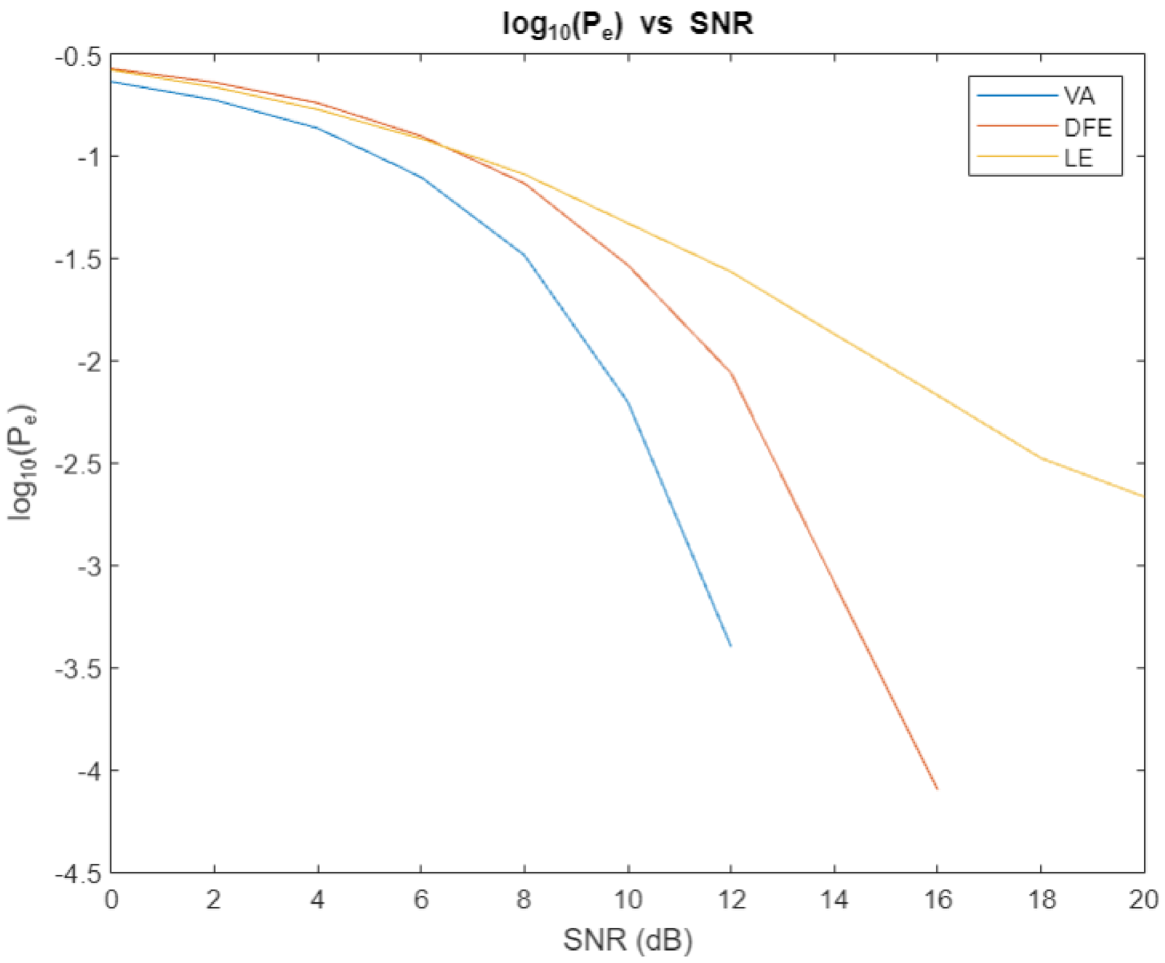

Figure 2: Performance Comparison: SER plots for VA, DFE and LE

- Although not visible in this plot, from theory, we know that the DFE's plot is an upper bound for the VA decoder with DDFSE implemented in the previous assignment, since both work on the same principle.
- Another example of subotimal equalizers are adaptive equalizers which are sometimes preferred over these methods. Since, their performance is not that worse compared to LE and DFE equalizers, and because they are computationally inexpensive compared to the inverse matrix operation involved in generating these LE and DFE equalizers.