Mini Project - 1

Rohith D, Jay Shah, Sayan Mitra, Aravint Annamalai

March 1, 2021

1 Abstract

In this mini-project, we apply least squares model to estimate **h** according to the following equation:

$$\mathbf{y} = \mathbf{XFh} + \mathbf{n} \tag{1}$$

given **y**, **X**, **F** and **n**. Here, a set of random bits are generated and modulated as QPSK symbols to generate **X**. **h** is a multipath Rayleigh fading channel vector with an exponentially decaying power-delay profile **p** of length `L` and **F** is a 512 × L matrix performing IDFT. The various scenarios of least squares are being simulated in this mini-project.

2 Least squares with no constraints/conditions

Here, we estimate **h** using ordinary least squares, with no constraints/restrictions on the variables other than those mentioned earlier. **X** and **F** are generated as above. **n** is a complex Gaussian noise with variance = `0.1`. The true value of **h** is generated and is compared with the value of **h** estimated by using Least Squares.

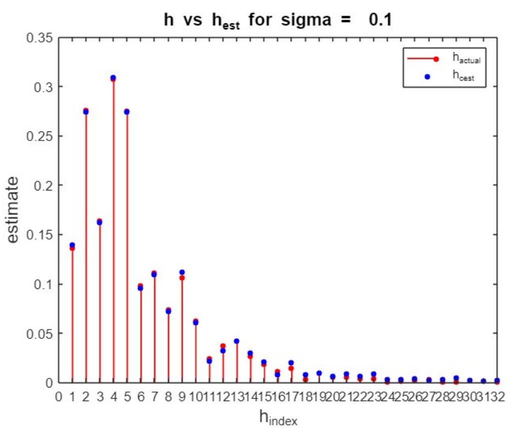


Figure 1: True value and estimated value of h

Now we perform the above computations for 10000 random trials. The plot obtained is as follows:

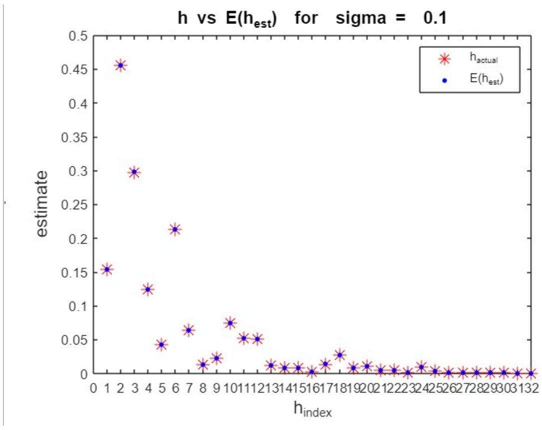


Figure 2: True value and estimated value of h over 10000 trials

3 Least squares with sparse h

Now, we consider the case where **h** is a sparse variable, with just 6 non-zero values (taps). The non-zero locations are generated at random, and the true value of **h** is made 0 at those locations. Estimating **h** is now just a constrained least squares problem, with the constraint that the estimated value of **h** should be 0 at all those locations where the true value of **h** is 0. The constrained least squares problem is solved using the method of Lagrangian multipliers and the true and estimated value of **h** is plotted.

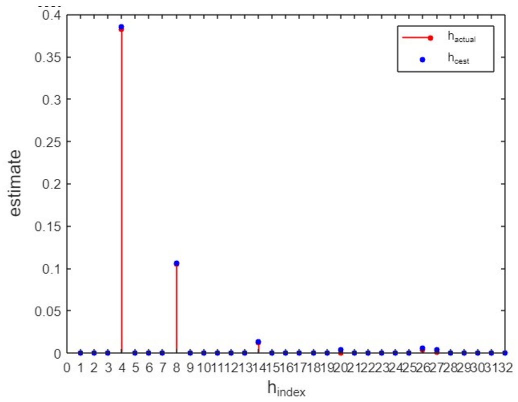


Figure 3: True value and estimated value of h, with 6 non-zero taps

Now we perform the above computations for 10000 random trials. The plot obtained is as follows:

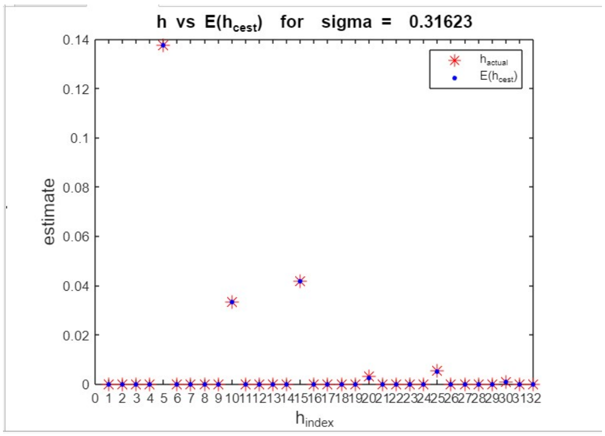


Figure 4: True value and estimated value of h, with 6 non-zero taps, over 10000 trials

4 Introduction of guard band of 180 symbols

Here, we include a 'guard band' of 180 symbols (i.e.) we make the first 180 and last 180 symbols of $\mathbf{X}$ as 0. This will basically make the $\mathbf{X}$ matrix "sparse". Now, with this constraint, we apply least squares to estimate $\mathbf{h}$. When the estimated and true value of $\mathbf{h}$ is plotted, we noticed a large error between them. This is because, least squares has an inversion operation, and the $\mathbf{X}$, which is already diagonal, is sparse as well. So, there is hardly any non-zero element in $\mathbf{X}$, and thus, it becomes nearly singular. This leads to a large error in estimating $\mathbf{h}$.

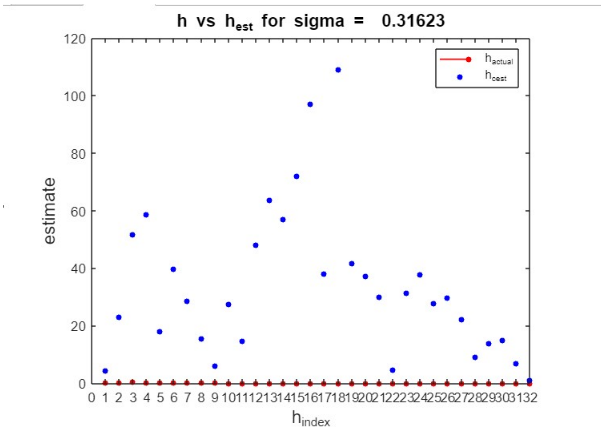


Figure 5: True value and estimated value of h, after introducing a guard band on X

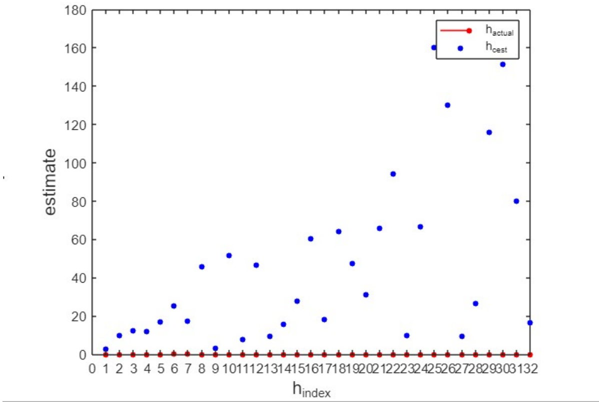


Figure 6: True value (6 non-zero taps) and estimated value of h, after introducing a guard band on X

Now to reduce the error, we use a technique called regularisation. Here, along with the objective of minimizing the squared error, we also have another objective of reducing the magnitude of the predicted **h** vector. Here, we tune a parameter α such that the MSE of the estimation is minimised. The following plots are obtained for the two scenarios.

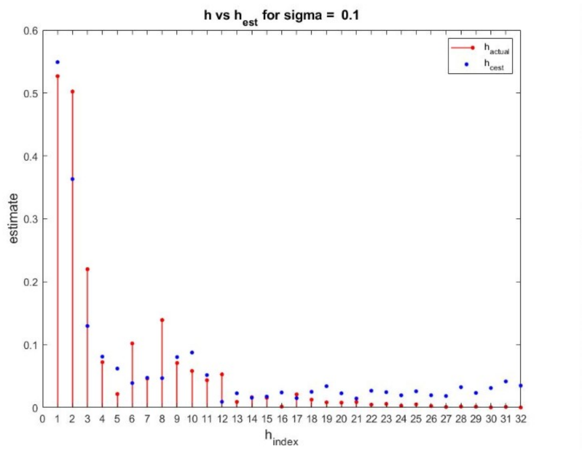


Figure 7: True and estimated value of h, after regularization

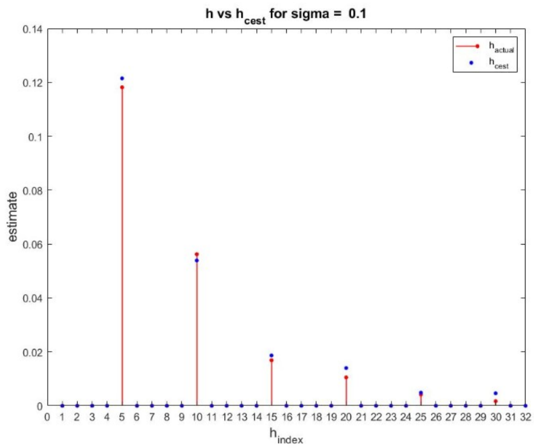


Figure 8: True and estimated value of h, after regularization , where **h** has 6 non-zero taps

Optimum value of $\alpha = 0.4$ for normal **h** and 5.5 for sparse values of **h**.

5 Constrained Least Squares

Now we introduce some contraints on the values of **h**, which are as follows:

$$\begin{aligned} h[1] &= h[2] \\ h[3] &= h[4] \\ h[5] &= h[6] \end{aligned} \tag{2}$$

If there are no other constraints, then the plot is obtained as follows:

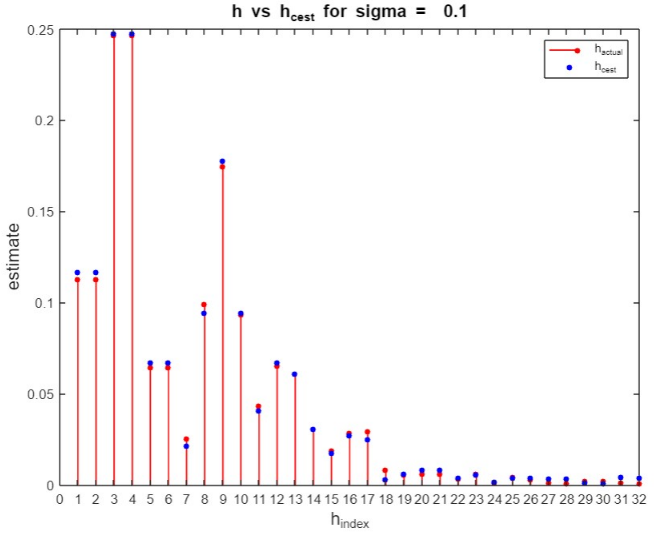
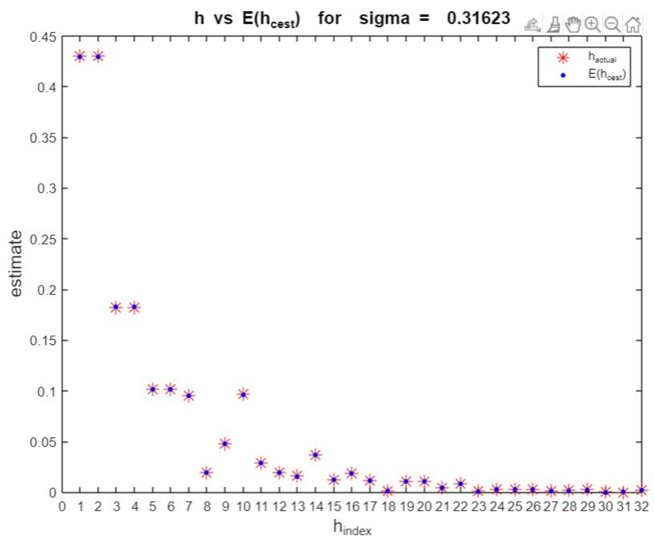


Figure 9: True value and estimated value of h, after introducing constraints



6 Algorithm to compute the non-zero locations of h

The algorithm to compute the non-zero values of **h** is explained in the code snippet below:

```
k0=6;   //number of non zero taps
S=[];   //for position of non zero taps
```

```
A=X*F;
r=y;    //residual
t=0;
for k=1:k0
    temp=-Inf;       //used to compute argmax
    for j=1:32
        val=norm((A(:,j))'*r); //argmax over j
        if val>temp
            t=j;
            temp=val;
        end
    end
    S(k)=t;    //appending tk to S
    B=A(:,S);
    P=B*pinv(B);
    r=(eye(512)-P)*y;
```

Now, having known the non-zero locations of $\mathbf{h}$, we can use normal Least Squares to estimate the values of $\mathbf{h}$.