Solution to Question 1.1

(a) Use the following code in R to simulate random i.i.d. observations from the uniform distribution on the interval $[0,1]$ saved in an $n \times p$ matrix:

```
set.seed(1)
n <- 100
p <- 2
X <- matrix(0,n,p)
for(i in 1:n)
{
for(j in 1:p)
{
X[i,j] <- runif(1,0,1)
}
}
```

(b) Simply use:

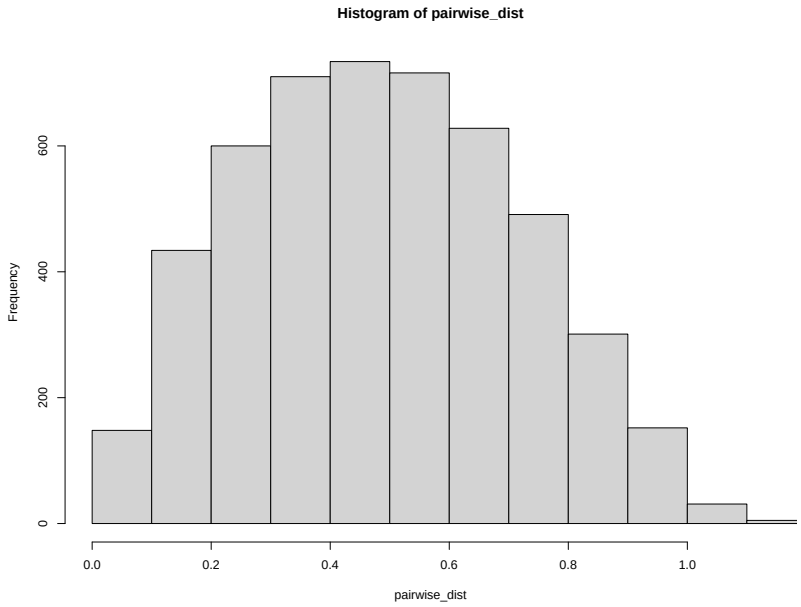
```
pairwise_dist <- dist(X)
```

Note that the R function `dist` computes and returns the distance matrix computed using the specified distance measure by the user (default is Euclidean) to compute the distances between the rows of a data matrix, which in our case are the n observations. The output is a distance matrix. Like other R functions, the online description on the internet provides further information and simple examples which can be useful.

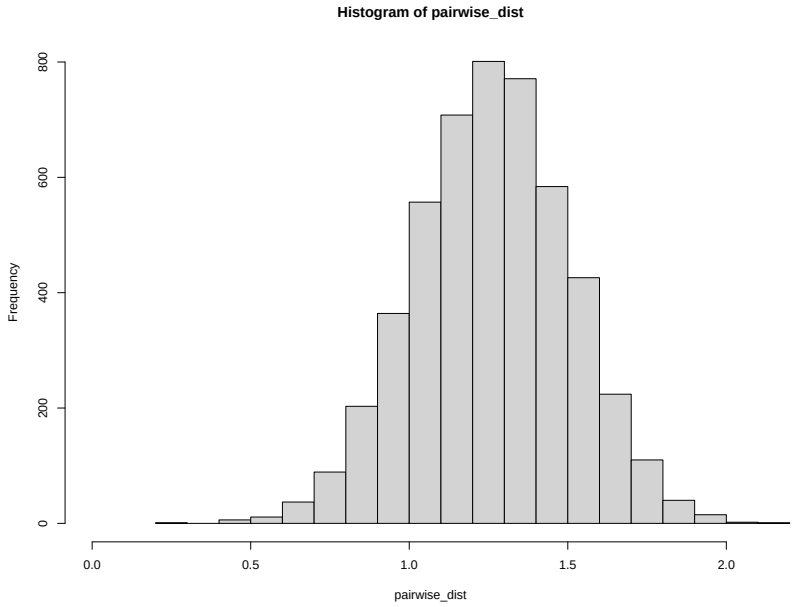
(c) Just use:

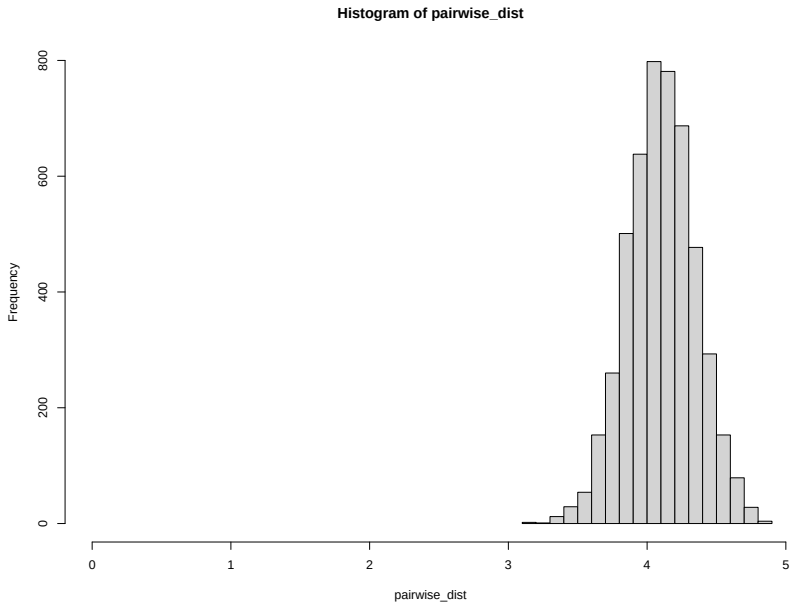
```
hist(pairwise_dist, xlim=c(0,max(pairwise_dist)))
```

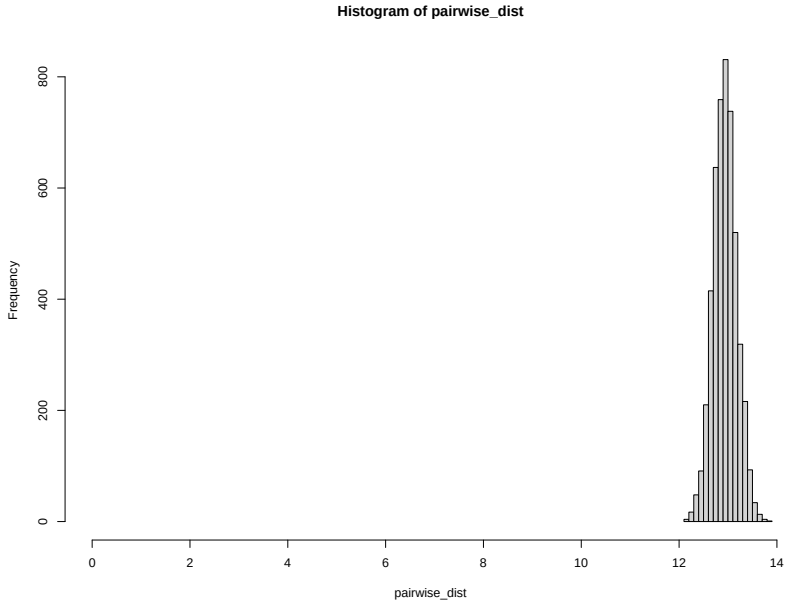
to get the following histogram.



(d) Just change the value of p in the above codes to get the following histograms for $p = 10$, $p = 100$ and $p = 1000$, respectively.



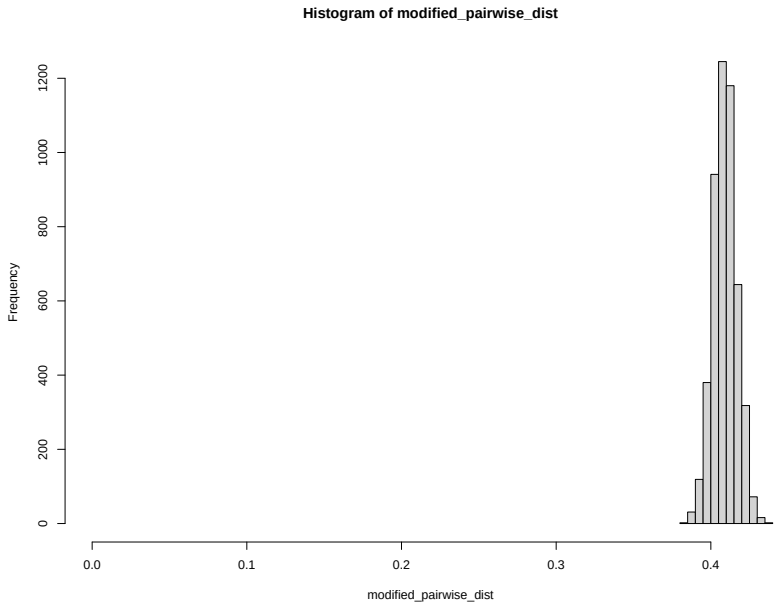




(e) For this, first scale the Euclidean distance by $p^{-1/2}$ to compute the modified pairwise distances and then plot them as follows:

```
modified_pairwise_dist <- dist(X)/sqrt(p)
hist(modified_pairwise_dist, xlim=c(0,max(modified_pairwise_dist)))
```

For example, the following histogram can be obtained with $p = 1000$.



It can be seen that using the scaled Euclidean distance (modified), the minimum distance between observations is much smaller (in fact, it stays finite and will not diverge as $p \to \infty$) which is really helpful. One might notice that the pairwise distances between observations may still look relatively close to each other which is due to small standard deviation.

Solution to Question 1.2

(a) Use the following code in R to simulate the Y_i from that regression model:

```
set.seed(1)
n <- 100
p <- 2
X <- matrix(0,n,p)
Y <- numeric(n)
beta <- numeric(p)
for(i in 1:n)
{
for(j in 1:p)
{
X[i,j] <- 3*sin(-pi*j*i/n)
```

```
}
}

for(j in 1:p)
{
beta[j] <- rnorm(1,0,j^(-4))
}

for(i in 1:n)
{
Y[i] <- t(X[i,])%*%beta+rnorm(1,0,1)
}
#an alternative way of getting simulated responses Y in matrix form
Y <- X%*%beta+rnorm(n,0,1)
```

(b) Fit a linear regression model without intercept to the simulated data using the following code:

```
LS <- lm(Y~-1+X)
summary(LS)
```

Then, use the following line to extract the estimates of regression coefficients:

```
beta_LS <- LS$coefficients
```

(c) Just use:

```
Yhat <- numeric(n)
for(i in 1:n)
{
Yhat[i] <- t(X[i,])%*%beta_LS
}
```

or more simply use:

```
Yhat <- X%*%beta_LS
```

(d) Similarly, just use:

```
Ytrue <- numeric(n)
for(i in 1:n)
{
Ytrue[i] <- t(X[i,])%*%beta
```

```
}
```

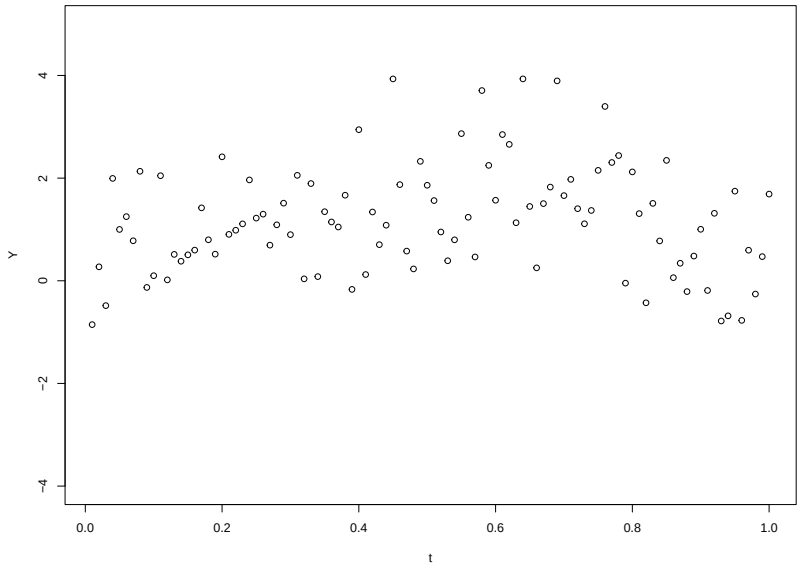
or more simply use:

```
Ytrue <- X%*%beta
```

(e) Use the following code to get a simple plot of Y_i versus t_i:

```
t <- (1:n)/n
plot(x=t,y=Y, ylim=c(-4,5))
```

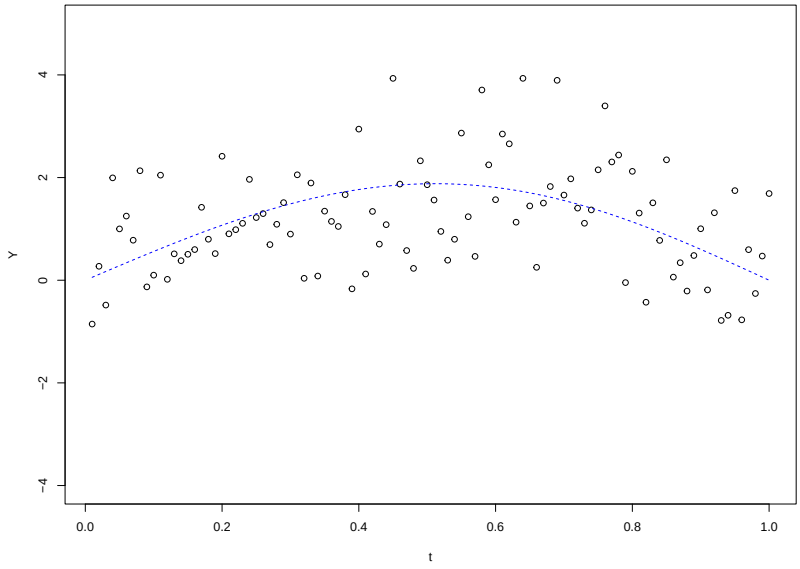
The plot is shown below.



(f) Append the following line to the above R code to add the true regression curve (signal curve) to the plot of Y_i versus t_i:

```
points (x=t, y=Ytrue, type="l", lty=2, col="blue")
```

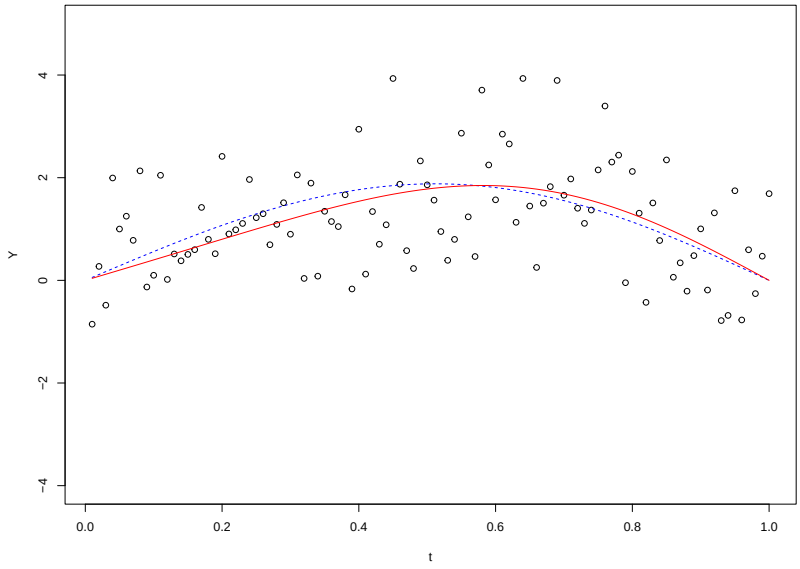
The resulting plot is shown below.



(g) Append the following line to the above R lines to also add the fitted regression curve (least squares curve) to the plot of Y_i versus t_i:

```
points (x=t, y=Yhat, type="l", lty=1, col="red")
```

The resulting plot is shown below.

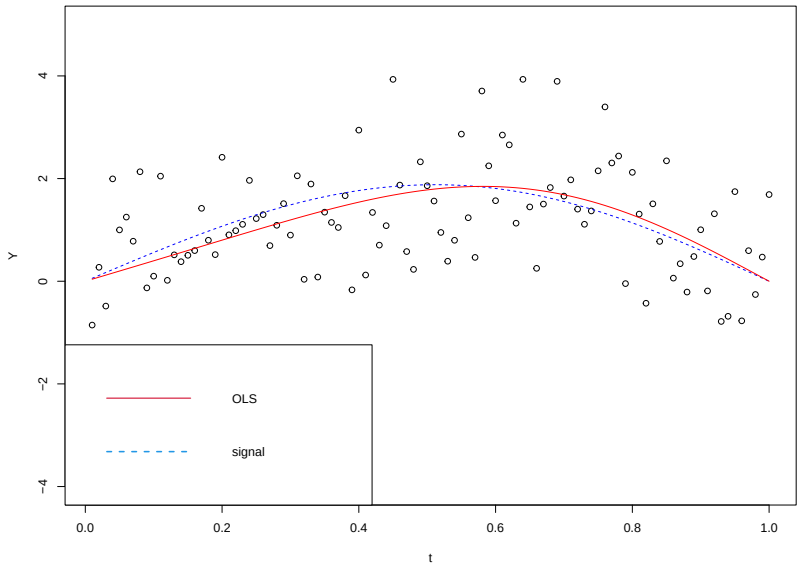


Appending the following code to the above will add a legend to the plot making its visualisation better.

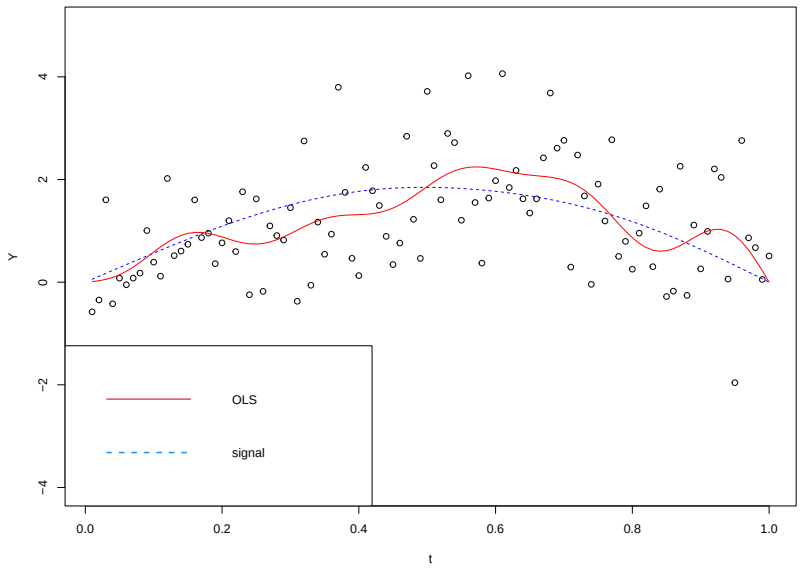
```
legend(x="bottomleft",          # Position
legend=c("LS","signal"),  # Legend texts
```

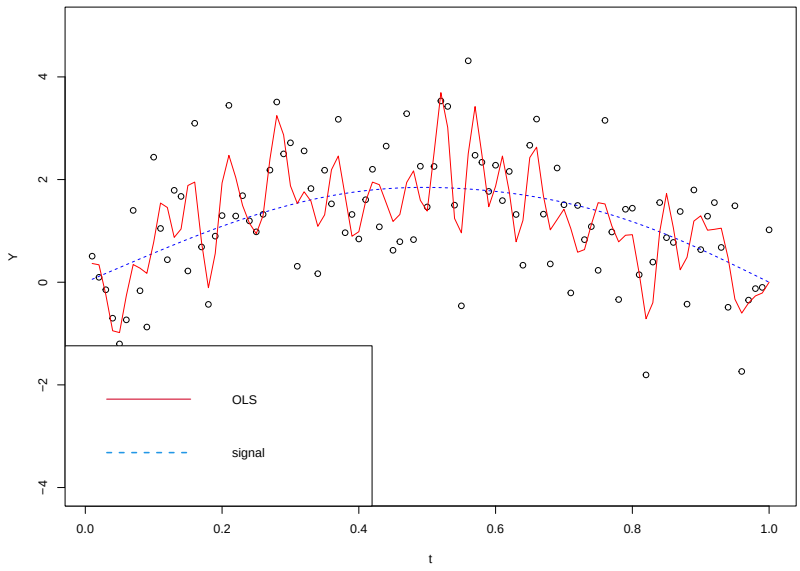
```
lty=c(1, 2),            # Line types
col=c(2, 4),            # Line colors
lwd=2)                  # Line width
```

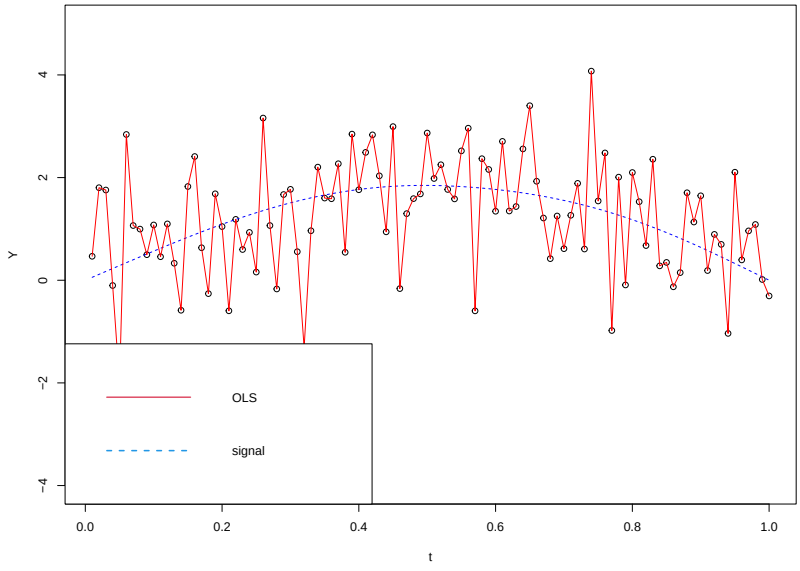
The plot with the above legend is shown below.



(h) Just change the value of p in the above codes to get the following plots for $p = 10$, $p = 50$ and $p = 100$, respectively. From these plots, one can observe that as p gets larger the mean squared error of the OLS method increases. In fact, when $p = 100 = n$ (see the last plot) the OLS curve goes through all the observations, which is clearly overfitting (model has too many parameters).







(i) Since $n = 100$, if $p = 101$ or larger then the OLS method fails to work. To see this, use $p = 101$ or $p = 200$ in the above codes and try:

```
LS <- lm(Y~-1+X)
summary(LS)
```

There will be no estimates for parameters but only for the first 100 parameters which may not be even reliable.