Optimizing a model parameter

Preprocessing the data

The data is read and a new column, which is the logarithm of rate, is added to the data.

MSE function

```
myMSE <- function(lambda){
    model_temp <- loess(Y ~ X, enp.target = lambda)

    pred_temp <- predict(model_temp, newdata = Xtest)
    MSE <- (1/length(pred_temp))*sum((Ytest - pred_temp)^2)
    count <<- count + 1 ;
    return(MSE)
}

 lam <- seq(0.1,40,by=0.1)
 mse <- c()
```

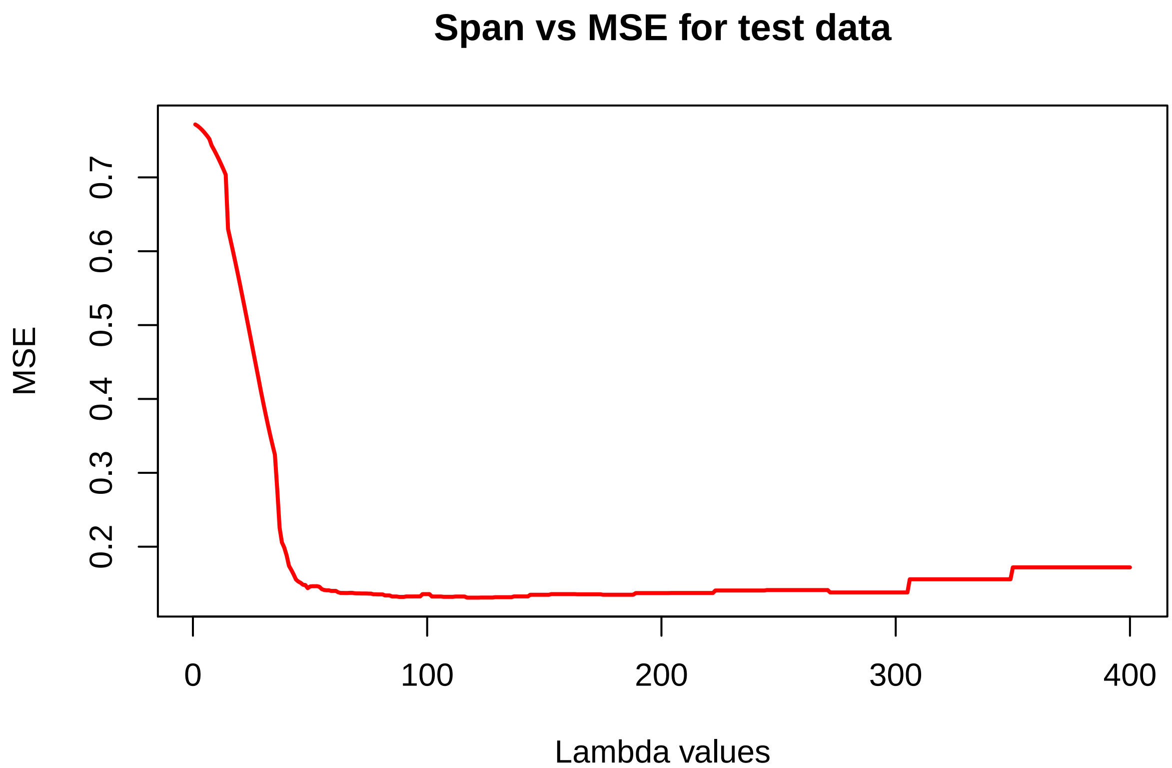
Plotting MSE vs lambda.

```
mse<- lapply(lam,myMSE)

#which.min selects the minimum mse from all the mse.
#as.numeric is used to convert list to numeric for round function
min_mse <- round(as.numeric(mse[which.min(mse)]),2)
cat(paste("\n\n Minimum MSE for the test data is =",min_mse,
    "for lambda =",lam[which.min(mse)]))
```

```
 Minimum MSE for the test data is = 0.13 for lambda = 11.7
```

```
{plot(x=1:length(lam),y=mse,xlab="Lambda values",type='l',
      ylab="MSE",col="red",lwd=2)
    title(main="Span vs MSE for test data")
}
```



Using optimize function to find lambda value

```
#Resetting count to zero to calculate iterations for optimize func
count <<- 0

#first argument is function for which we want to find the optimal value
#interval(lambda's) is the range where combination of golden section search
#and successive parabolic interpolation be done to find optimal lambda value

val <- optimise(myMSE,interval = seq(0.1,40,by=0.1))   #maximum argument is false by default,

cat(count,"Iterations were required by optimize function to find the
    optimal lambda for least MSE")
```

```
27 Iterations were required by optimize function to find the
    optimal lambda for least MSE
```

```
cat(paste("\n\n Minimum MSE found out by topimize function for the test
          data is =",round(val$objective,2),
    "for lambda =",round(val$minimum,2)))
```

```
 Minimum MSE found out by topimize function for the test
          data is = 0.13 for lambda = 10.69
```

Using optimize function with search range and accuracy

Using optimise function with accuracy to 0.01 converges fast than without giving the accuracy parameter. Using `accuracy` argument to 0.01 takes only 18 iterations to find the parameter whereas 27 iterations of function is executed to find the same value for the parameter without accuracy argument.

```
#tol - the degree of accuracy required.
count <<- 0
val1 <- optimise(myMSE,interval = seq(0.1,40,by=0.1),tol=0.01)

cat(count,"Iterations were required by optimize function with accuracy 0.01 to find the
    optimal lambda for least MSE")
```

```
18 Iterations were required by optimize function with accuracy 0.01 to find the
    optimal lambda for least MSE
```

```
cat(paste("\n\n Minimum MSE found out by topimize function for the test
          data is =",round(val1$objective,2),
    "for lambda =",round(val1$minimum,2)))
```

```


 Minimum MSE found out by topimize function for the test
          data is = 0.13 for lambda = 10.69
```

Using optim()function and BFGS to find lambda value

```
count <<-0
optim(par=35,fn=myMSE,method="BFGS")
```

```
$par
[1] 35

$value
[1] 0.1719996

$counts
function gradient
       1        1

$convergence
[1] 0

$message
NULL
```

Maximum Likelyhood

Log likelyhood and deriving maximum likelyhood estimator

$$log(x_1, x_2...x_{100}|\mu, \sigma) = -\frac{1}{2 * \sigma^2} * \sum_{i=1}^{100}(x_i - \mu)^2 - \frac{100}{2}(log2\pi\sigma^2)$$

Let P = $log(x_1, x_2...x_{100}|\mu, \sigma)$

$$\frac{\partial P}{\partial \mu} = \frac{1}{\sigma^2} * \sum_{i=1}^{100} (x_i - \mu)$$

$$\frac{\partial P}{\partial \sigma} = \frac{\sum_{i=1}^{100} (x_i - \mu)^2}{\sigma^3} - \frac{100}{\sigma}$$

We can maximize the estimator μ and σ by equating both the above derivative to zero. .

$$\mu = \frac{\sum_{i=1}^{100} (x_i)}{100}$$

$$\sigma^2 = \frac{\sum_{i=1}^{100} (x_i - \mu)^2}{100}$$

```
Below parameters estimates the normal distribution from
where the sample is taken:


Mean(mu): 1.28

Sigma: 2.01
```

Optimizing the minus log likelihood function

Why it is a bad idea to maximize likelihood rather than maximizing log likelihood?

Likelyhood terms are multiplicative and most of the distribution for whose finding the parameter is of keen interest includes exponential terms. Maximizing the likelyhood by finding a derivative and equating to zero would require cumbersome mathematics and is often computationally expensive because the right hand side of the equation contains lot of multiplicative and exponential term. Since log is monotonically increasing function, the parameter that maximizes the log of a function would be same as the one that maximizes the likelyhood. Log tranforms the multiplicative terms into addition terms and exponential term into multiplicative term which is less computationally intensive. Moreover the log reduces the number in interest and computer work much better with smaller numbers thereby reducing the precision error as well.

```r
#Minus log likelyhood
minusloglikely <- function(par) {
    mu = par[1]
    sigma = par[2]
    1/((2*sigma^2))*sum((data-mu)^2)+(length(data)/2)*(log(2*pi*sigma^2))
}

#Optimizing negative log likelyhood using conjugate gradient and BFGS method
val1 <- optim(par=c(0,1),minusloglikely,method="CG")
val2 <- optim(par=c(0,1),minusloglikely,method="BFGS")


compare_data <- rbind(compare_data,c(round(val1$par[1],2),
                                     round(val1$par[2],2),val1$counts[1],val1$counts[2]))
compare_data <- rbind(compare_data,c(round(val2$par[1],2),
                                     round(val2$par[2],2),val2$counts[1],val2$counts[2]))
rownames(compare_data) <- c("Maximum Likelyhood","Conjugate Gradient","BFGS")
```

Analysis

Maximizing log likelyhood,conjugate gradient and BFGS, converges the mean and sigma estimators for the data to same value to 2 decimal precision. Below table represent the number of function and gradient evaluations done by the algorithm.

Table 1: Comparison of result by different methods

	Mean	Sigma	Count_function_eval	Count_gradient_eval
Maximum Likelyhood	1.28	2.01	-	-
Conjugate Gradient	1.28	2.01	297	45
BFGS	1.28	2.01	37	15