Maximum-likelihood

Vincent Lee

1/18/2020

Workshop

negative log-likelihood normal function declaration

Negative log-likelihood will minimise it

```
Negative.LL.Normal<-function(mu.Sig.parameters,Sample.Vector){
  n     <- length(Sample.Vector)
  sigma <- mu.Sig.parameters[2]
  part_a <- (n/2)*log(2*pi*sigma)
  part_b <- 1/(2*sigma)
  y      <- (Sample.Vector- mu.Sig.parameters[1])^2
  value <- part_a + part_b *(sum (y))
  return (value)
}
```

```
dataPath <- "C:/Users/vincentlee/Desktop/Non_linear_models/Week2"
Norm.Sample.Vector<-read.csv(file=paste(dataPath,"sample_for_optimization.csv",sep="/"),header=TRUE,sep=
head(Norm.Sample.Vector)
```

```
## [1] 12.6567272 10.8362391  4.1087307 10.9831141 10.7719944 -0.9756093
```

```
var(Norm.Sample.Vector)
```

```
## [1] 24.75509
```

Optimazation of negative LL

```
Optimized.Negative.Log.Likelihood.optim<-optim(c(7,4),
                                               Negative.LL.Normal,
                                               Sample.Vector=Norm.Sample.Vector,
                                               method="L-BFGS-B",
                                               hessian=TRUE,
                                               lower=c(-Inf,0),
                                               control=list(trace=1))
```

```
## iter   10 value 3024.314081
## final  value 3022.953774
## converged
```

```r
# 10 iteration
```

```r
c(Optimized.Negative.Log.Likelihood.optim$par,
Optimized.Negative.Log.Likelihood.optim$value,
Optimized.Negative.Log.Likelihood.optim$counts,
Optimized.Negative.Log.Likelihood.optim$convergence)
```

```
##                                 function   gradient
##   10.22552   24.73032 3022.95377   17.00000   17.00000    0.00000
```

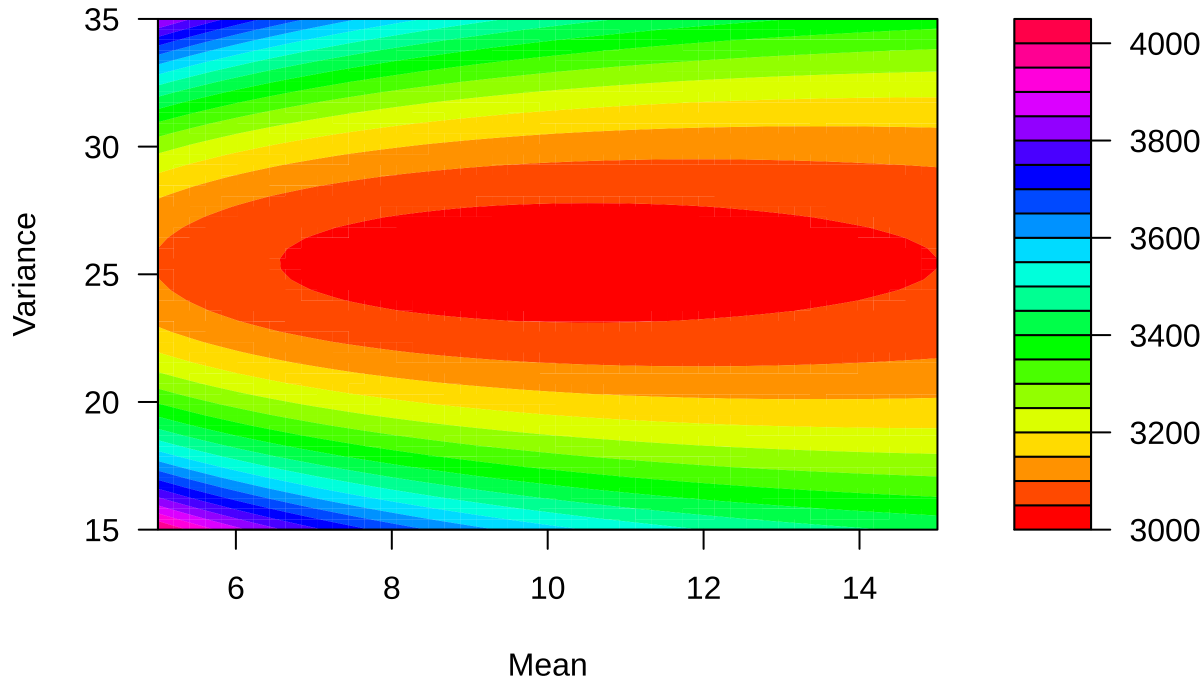
```r
Optimized.Negative.Log.Likelihood.optim$hessian
```

```
##               [,1]          [,2]
## [1,]  4.043619e+01 -1.136868e-07
## [2,] -1.136868e-07  8.175431e-01
```

```r
#Plot the objective function
Data.To.Plot.x<-seq(from=5,to=15,length.out=50)
Data.To.Plot.y<-seq(from=15,to=35,length.out=50)


Term.1<-length(Norm.Sample.Vector)/2*outer(log(2*pi*Data.To.Plot.y),
                                           rep(1,length(Data.To.Plot.y)))
Term.2<-outer(rep(1,length(Data.To.Plot.x)),
              unlist(lapply(Data.To.Plot.x,FUN=function(vector.element,Data.Vector)
                sum((Data.Vector-vector.element)^2),
                Norm.Sample.Vector)))*outer(1/2/Data.To.Plot.y,rep(1,length(Data.To.Plot.y)))

Negative.Log.Likelihood.Data<-Term.1+Term.2
filled.contour(Data.To.Plot.x,Data.To.Plot.y,Negative.Log.Likelihood.Data,
               color.palette=rainbow,nlevels=20,xlab="Mean",ylab="Variance")
```



Analysis of the obtained estimates

Compare Optimized.Negative.Log.Likelihood.optim$par with c(mean(Norm.Sample.Vector),var(Norm.Sample.Vector))

```
rbind(Mean.Var=c(mean(Norm.Sample.Vector),var(Norm.Sample.Vector)),
      Optim.Output=Optimized.Negative.Log.Likelihood.optim$par)
```

```
##                  [,1]     [,2]
## Mean.Var     10.22552 24.75509
## Optim.Output 10.22552 24.73032
```

Why var(Norm.Sample.Vector)) is different from Optimized.Negative.Log.Likelihood.optim$par[2]?

```
n <- length(Norm.Sample.Vector)
rbind(Mean.Var=c(mean(Norm.Sample.Vector),var(Norm.Sample.Vector)),
      Optim.Output=c(Optimized.Negative.Log.Likelihood.optim$par[1],
                     Optimized.Negative.Log.Likelihood.optim$par[2]*(n/(n-1))))
```

```
##                  [,1]     [,2]
## Mean.Var     10.22552 24.75509
## Optim.Output 10.22552 24.75508
```

Fisher score, fisher information

```
Biased.Var<- var(Norm.Sample.Vector)*((n-1)/n)
```

```
sum(Norm.Sample.Vector- mean(Norm.Sample.Vector))/var(Norm.Sample.Vector)
```

```
## [1] -2.583261e-14
```

```
(-n/(2*Biased.Var))+ (sum((Norm.Sample.Vector-mean(Norm.Sample.Vector))^2)/((Biased.Var^2)*2))
```

```
## [1] 3.552714e-15
```

Observed fisher's information

```
Optimized.Negative.Log.Likelihood.optim$hessian
```

```
##               [,1]          [,2]
## [1,]  4.043619e+01 -1.136868e-07
## [2,] -1.136868e-07  8.175431e-01
```

1. Element Hessian.1.1

```
n <- length(Norm.Sample.Vector)
n/var(Norm.Sample.Vector)
```

```
## [1] 40.39574
```

2. Element Hessian 1.2 and Element Hessian 2.1

```
sum(Norm.Sample.Vector- mean(Norm.Sample.Vector))/(var(Norm.Sample.Vector)^2)
```

```
## [1] -1.043527e-15
```

3. Element Hessian 2.2

```
Hessian2.2.1<- sum((Norm.Sample.Vector- mean(Norm.Sample.Vector))^2)/(Biased.Var^3)

(-n/(2*(Biased.Var^2)))+Hessian2.2.1
```

```
## [1] 0.8175421
```

Expected fisher's information (by assuming hessian y-mu = 0)

```r
#(Biased.Var<- var(Norm.Sample.Vector)*(length(Norm.Sample.Vector)-1)/length(Norm.Sample.Vector))
rbind(c(length(Norm.Sample.Vector)/Biased.Var,0),c(0,length(Norm.Sample.Vector)/2/Biased.Var^2))
```

```
##          [,1]      [,2]
## [1,] 40.43617 0.0000000
## [2,]  0.00000 0.8175421
```

4. Example simple linear regression

```r
nSample<-500
sigmaEps<-1.5
set.seed(927436)
Eps<-rnorm(nSample,0,sigmaEps)
beta1<-1
beta0<-2.5
lambda<-.5
X<-rexp(nSample,lambda)
Y<-beta0+beta1*X+Eps
# plot(X,Y)
```

```r
dtf<-data.frame(X=X,Y=Y)
head(dtf)
```

```
##          X         Y
## 1 1.980506 4.0826325
## 2 2.542126 5.2707668
## 3 1.040133 6.9997564
## 4 1.694994 0.7666407
## 5 0.102001 1.9767673
## 6 2.751328 6.4738524
```

4.2 loglikelihood function

```r
linModLL<-function(Parameters,regSample){
  # product of density of response f (y, theta)
  # L(theta, y) = sum (log f(yi, theta))
  # = sum( log(dnorm(sample, mean=bo+b1xi, sd=sigmaEps)))
  # Parameters[1] = beta0 Paramaters[2] = beta1 Sigma = Parameters[3]
  y <- regSample[,2]
  x <- regSample[,1]
  average <- Parameters[2]
  intercept <- Parameters[1]
  standev <-Parameters[3]
  log_dnorm <- log(dnorm(x = y, mean = intercept+(average*x),sd =standev))
  log_lik <- -sum(log_dnorm)
  return(log_lik)
}
```

```r
linModLL(c(Beta0=beta0+1,Beta1=beta1+1,Sigma=sigmaEps),dtf)
```

```
## [1] 2523.769
```

```r
linModLL(c(Beta0=beta0,Beta1=beta1,Sigma=sigmaEps),dtf)
```

```
## [1] 931.3951
```

4.3 FIT

```r
Optimized.linModLL.optim <-optim(c(Beta0 = beta0,Beta1= beta1,Sigma=sigmaEps),
                                    linModLL,
                                    regSample=dtf,
                                    method="L-BFGS-B",
                                    hessian=TRUE,
                                    lower=c(-Inf,0.00),
                                    control=list(trace=1))
```

```
## final  value 930.111159
## converged
```

```r
# this is optim function
# we need to use newton approach for optimization in Assignment
```

```r
Optimized.linModLL.optim$par
```

```
##     Beta0     Beta1     Sigma
## 2.4929634 0.9761973 1.5547015
```

Compare the results with linear model fit.

```r
linM<-lm(Y~X,dtf)
c(linM$coefficients,summary(linM)$sigma)
```

```
## (Intercept)           X
##   2.4929604   0.9761977   1.5578206
```

Assignment

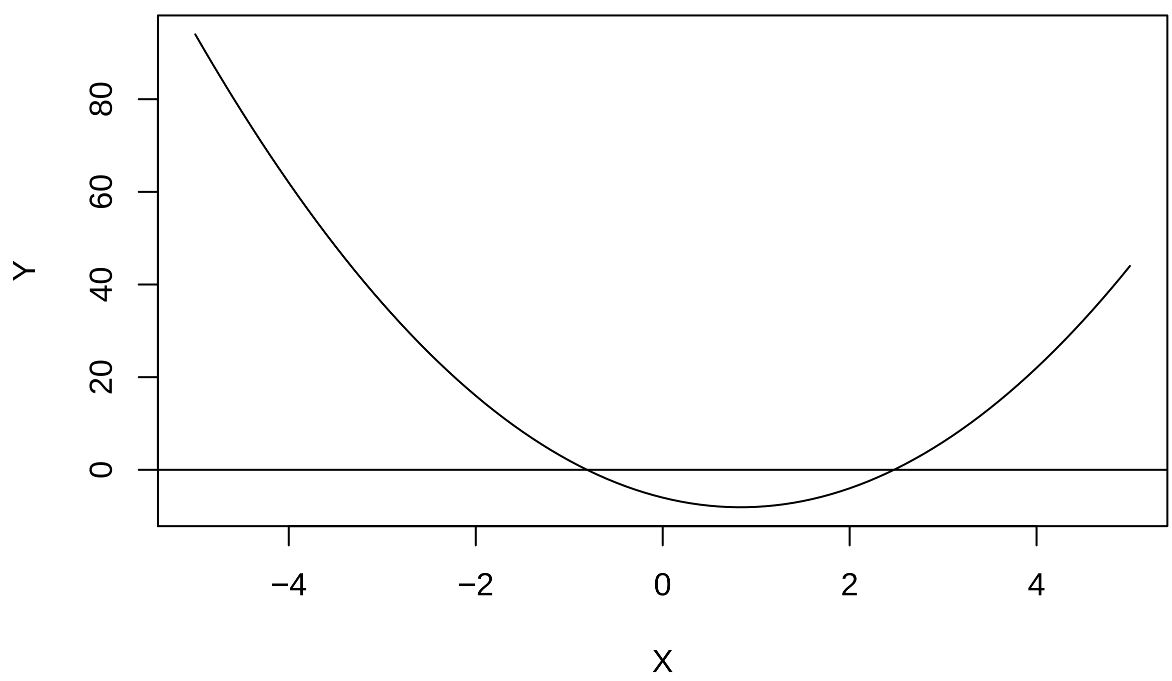
```r
my.Optimizer<-function(Start.Value,Function.To.Optimize,Epsilon,projectID){
  iteration <- 10000 #random number
   derivative   <- function(Start.Value, Epsilon, projectID) {
     # derivative function of testFunction
    return((Function.To.Optimize((Start.Value + Epsilon),projectID) -
             Function.To.Optimize((Start.Value - Epsilon), projectID)) / (2 * Epsilon))
      }
```

```
    for (i in 1:iteration){
     update.step <- (Function.To.Optimize(Start.Value,projectID))/(derivative(Start.Value, Epsilon, proj
     #once update step is smaller than Eps, print results
     if (abs(update.step) < Epsilon){
       break
     }
      Start.Value<- Start.Value - update.step
  }
  root <- Start.Value
  return (root)
}
```

To check your optimizer create a test function that needs to be optimized. In this project we use one-dimensional optimization, i.e. optimization with respect to only one variable. Add one more argument to the function, called projectID. The meaning of it will become clear in section Test. The function should cross x-axis at least in one point.

```
my.Function<-function(my.X,projectID) {
  my.X^2*3-my.X*5-6
}
```

```
X<-seq(from=-5,to=5,by=.1)
Y<-my.Function(X)
plot(X,Y,type="l")
abline(h=0)
```



You can also test the optimizer by running uniroot().

```
uniroot(my.Function,lower=-5,upper=+1, tol=0.0001)
```

```
## $root
## [1] -0.8081429
##
## $f.root
## [1] -1.138463e-06
##
## $iter
## [1] 9
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 5e-05
```

```
my.Optimizer(-5, my.Function,0.0001,656)
```

```
## [1] -0.808143
```

Test

```
testFunction<-readRDS(file=paste(dataPath,"Week2_TestFunction.rds",sep="/"))$Week2_Test_Function

#project 656
testFunction(0, 656)
```

```
## [1] -5
```

```
#readRDS(file=paste(dataPath,"Week2_TestFunction.rds",sep="/"))
```

Make sure that declaration of your optimizer function contains projectID argument.

```
my.Optimizer(Start.Value=-100,
             Function.To.Optimize = testFunction,
             Epsilon=0.0001,
             projectID=656)
```

```
## [1] -5.000001
```

where:

Start.Value is initial guess for the optimizer, testFunction is the name of the test function that needs to be optimized, Epsilon is stopping criterion (set Epsilon=0.0001), projectID is individual project ID.

Find root (my.Optimizer.root) of the test function using your optimizer.

```r
my.Optimizer(Start.Value=-100,
             Function.To.Optimize = testFunction,
             Epsilon=0.0001,
             projectID=656)
```

```
## [1] -5.000001
```

Find root (uniroot.root) using uniroot(). Use Epsilon=0.0001 as tolerance parameter (tol = 0.0001) of uniroot()

```r
(uniroot.val<- uniroot(testFunction, c(-100, 0),tol = 0.0001, projectID=656))
```

```
## $root
## [1] -5
##
## $f.root
## [1] 1.007208e-07
##
## $iter
## [1] 14
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 5e-05
```

```r
(my.optimizer.root<-optim(par=-100,
                          my.Optimizer,
                          Function.To.Optimize=testFunction,
                          Epsilon = 0.0001,
                          projectID = 656,
                          method="L-BFGS-B",
                          hessian=TRUE,
                          lower=c(-1e10,0),
                          control=list(trace=1)))
```

```
## final  value -5.000001
## converged
```

```
## $par
## [1] -100
##
## $value
## [1] -5.000001
##
## $counts
## function gradient
##        2        2
##
## $convergence
```

```
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
##
## $hessian
##               [,1]
## [1,] -2.065015e-08
```

```
res <- list(Start.Value = -100,
            my.Optimizer.root =my.optimizer.root$value,
            uniroot.root = uniroot.val$root,
            uniroot.lower = -5,
            uniroot.upper = 0)
```

```
write.table(res, file = paste(dataPath,'result.csv',sep = '/'), row.names = F)
```