Parameter Estimation

Minimization and Maximization of Functions

Neba Nfonsang
University of Denver

In [1]:
```
require(ggplot2)
require(dplyr)
```

```
Loading required package: ggplot2
Loading required package: dplyr

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

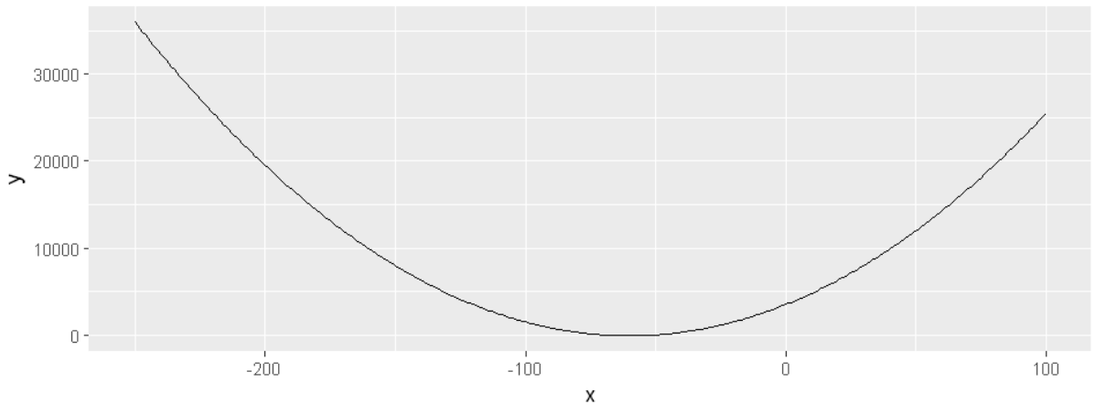
How do we Minimize Functions in R?

In [2]:
```
options(repr.plot.width=8, repr.plot.height=3)

# let's create input values
x = -250:100

# let's create the output values of a non-linear function, y
y = (x+70)*(x+50)
dat = data.frame(x, y)

# visualize the non-linear function
ggplot(data=dat, aes(x=x, y=y)) +
geom_line()
```



- Our goal is to minimize y
- We are looking for the value of x where y is minimum

In [3]:
```
# what is the minimum value of y?
min(y)
```

-100

- What is the index of the minimum value of y

- At what index position is the minimum value of y found in the vector of values ?

In [4]:
```
which.min(y)
```

191

In [5]:
```
length(y)
```

351

In [6]:
```
# the minimum value of y
y[which.min(y)]
```

-100

In [7]:
```
# this the value of x we are looking for, where y is minimum
x[which.min(y)]
```

-60

Let's use the nlm() and optimize() function in R to minimize the function

In [8]:
```
# define the objective function

f <- function(x){
    return ((x+70)*(x+50))
}
```

- The function to be optimized is called the objective function
- Minimize the objective function using nlm(),
- The nlm() takes an objective function (f) and the initial value (p)
- If there is only one parameter to be estimated, pass one initial value into the nlm()
- If there are many parameters to be estimated, pass as many initial values are there are parameters to be estimated.

In [9]:
```
# run the minimization
nlm(f, p = 0)
```

$minimum
-100
$estimate
-60
$gradient
0
$code
1
$iterations
2

- As we can see from these results, the estimate is -60, and this happens at the minimum value of y = -100

Let's use the optimize() function instead

In [10]:
```
# specify the interval where the minimum and maximum values are
optimize(f = f, interval = c(max(x), min(x)))
```

$minimum
-60
$objective
-100

- This results are the same as the results we obtained using the nlm() function

Difference between nlm() and optimize()

- The optimize() function can maximimize and minimize an objective function, but the nlm() only minimizes the objective function.
- An objective function should be minimized if it is convex, or if its shape looks like a valley. For example, $y = x^2$ is a convex function.
- An objective function should be maximized if it is concave, or if it's shape looks like a mountain. For example $y = -x^2$ is a concave function.
- Since the nlm() is a minimization function, we cannot not use it directly to maximize a function. If you want to maximize a function using the nml() function, you should instead minimize the negative of the function because minimizing the negative of a function is the same as maximizing the function.
- If you have more than one parameter to estimate, use the nlm() instead. The optimize() function is good for finding a single parameter estimate.

Parameter Estimation Using Maximum Likelihood

Problem: Out of a total of 5 flights from Denver to Durango, only two planes arrived on time. What is the probability that any flight from Denver to Durango will arrive on time?

Let's use a Bernoulli or Binomial Approach

Solution:

- Each data point represents the outcome of an independent Bernoulli trial
- The Bernouli outcomes look like this: value = c(0, 0, 0, 1, 1), where 0 represents late arrival and 1's represent arrival on time.
- This problem can be modelled as a Bernoulli problem where we are looking for the parameter p (the probability of success for a single trial or the probability that any flight arrives on time)
- If we use a Bernoulli or Binomial approach, we will arrive at the same answer because the likelihood function we are maximizing in a Bernoulli distribution is the same as the likelihood function in a Binormial distribution.

Likelihood function for a Bernoulli distribution:

- For a Bernoulli random variable that takes on possible values $x_i = [0, 1]$, the likelihood function can be specified as:
  $L(\theta) = \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i}$
  $= p^{\Sigma_1^n x_i}(1-p)^{\Sigma_1^n 1-x_i}$
  $= p^x(1-p)^{n-x}$
  where x = number of successes
- It is probably better to use k = number of successes here instead of x to distinguish that from the x in the Bernoulli trial.
- n = total number of trials

Likelihood function for the Binomial distribution:

- For a Binomial distribution, the likelihood function is:

$$L(\theta) = nC_x\ p^x(1-p)^{n-x},$$

- The expression on the right has no product sign because x in a Binomial density represents the number of successes and is a **single value** for a given set of data or outcomes from independent Bernouli trials. x can be calculated by summing the outcomes of the independent Bernoulli trials.
- To maximize the likelihood function, since nC_x is a constant, we can drop it, which means we will need to maximize:
  $p^x(1-p)^{n-x}$, which is the same expression we are maximizing when using the Bernoulli distribution

In [11]:
```
# Goal: maximize the likelihood or log likelihood function
# with n=5, and x=2.
# This is the same as minimizing the negative log likelihood function

#specify the likelihood function for the binomial distribution
n = 5
x = 2
neg.like<- function(p){

   neg.lik <- -(p^x)*(1-p)^(n-x)
    # we want to return the negative likelihood function
    # because we will be using the nlm() which is a minimization functio
    return (neg.lik)
}
```

In [12]:
```
# let's first optimize the negative likelihood function using optimize()
# the optimize requires the interval where to search
# for the optimal parameter value
optimize(neg.like, interval=c(0, 1))
```

$minimum
0.399999427550837
$objective
-0.034559999999882

- The optimize function returned the parameter value, which is 0.39999 ≈ 0.4
- Also note that, with the optimize function is a minimization and maximization function, so the objective function or the negative of the objective function should give the same results. The likelihood function, log likelihood, negative likelihood or negative log likelihood, should give the same results with the optimize() function.
- If you are using the nlm() for maximum likelihood estimation, your objective function should be the negative likelihood function or the negative log likelihood function since you are maximizing the objective function but the nlm() is a minimization function.

In [13]:
```
# now, let's use the nlm() function
# since there is only one parameter to estimate,
# provide only one starting value, avoid using 0
nlm(neg.like, p=.1)
```

$minimum
-0.0345599999999045
$estimate
0.39999948506851
$gradient
-1.07622244449601e-08
$code
1
$iterations
5

In [14]:
```
# we could also solve the problem using the formula for the
# probability of a single trial directly as p = k/n
values = c(0, 0, 0, 1, 1)
sum(values)/length(values)
```

0.4

A Normal Distribution Problem using MLE

Problem: Given that the data: 30, 40, 50, 60 were drawn from a normal distribution, use the maximum likelihood method to estimate the parameters of the distribution that generated this data?

In [15]:
```
X = c(30, 40, 50, 60)
# let's specify the objective function to be optimized
# the first parameters specified in the function
# should be the parameters to be estimated
neg.loglik <- function(thetha, sample){
    # use dnorm() and turn on log=TRUE to get the log of the densities
    # then sum the log of the densities to get the loglikelihood
    # then return the negative of the loglikehood
    loglike <- sum(dnorm(x = sample, mean =thetha[1] ,
                         sd =thetha[2], log=TRUE))
    return (-loglike)
}
```

- The negative log likelihood is specified in this situation, just because it would be easier to use the negative log likelihood with the nlm() minimization function.
- Also, each time we specify the log likelihood or negative log likelihood, we are summing the logs of the densities of the data points which may be easier to specify compared to using the likelihood where we multiply the densities of the data points.
- Maximizing likelihood gives the same results as maximizing log likelihood, minimizing likelihood or minimizing log likelihood.
- When we place a negative sign to the log likelihood, we get the negative log likelihood.
- Also note that this objective function above could be specified using the formula of the normal density but the built-in function in R is used, which is much easier.

In [16]:
```
# run the minimization
# inside the nlm() function, pass the values of known parameters
# such as sample=X
nlm(f=neg.loglik, p=c(min(X), max(X)), sample=X)
```

$minimum
19.8160873095294
$estimate
31.7481004535896 53.6728688687699

$gradient
-0.0184004306730313 0.0667486843605149

$code
4
$iterations
100

In [17]:
```
# let's check the parameters using the mean() and var()
# functions directly
mean(X)
sd(X)
```

45

12.9099444873581

- The estimate values from the nlm() function do not match the mean and variance computed from the mean() and var() function.
- The initial values selected in the nlm() function determine what final estimate values we get, so we need to select initial values carefully.
- It is a good idea to try different initial values, then select the initial values that give the smallest minimum value of the function. We can use a nested for loop to acheive this in a situation where we have two parameters to estimate. If there is a single parameter to estimate, a single for loop should be fine.

Let's find the best initial starting values for the nlm() function

In [18]:
```
# again, here is the function we want to optimize
X = c(30, 40, 50, 60)
neg.loglik <- function(thetha, sample){
    loglike <- sum(dnorm(x = sample, mean =thetha[1],
                         sd =thetha[2], log=TRUE))
    return (-loglike)
}
```

In [19]:
```
options( warn = -1 ) # supress warning

# create a variable to track minimum values of the function
mini = rep(NA, 100)
# create variables to track initial means and sds
initial.means = rep(NA, 100)
initial.sds = rep(NA, 100)

counter = 1
for (i in 1:10){
    for (j in 1:10){
       # track the minimum values, initial means and sds
       m <- nlm(neg.loglik, c(i, j), sample=X)$minimum
       mini[counter] <- m
       initial.means[counter] <- i
       initial.sds[counter] <- j

       counter = counter + 1
    }
}

# index of best minimum value
which.min(mini)
```

68

In [20]:
```
# best minimum value of function
mini[which.min(mini)]
```

15.3323816074233

In [21]:
```
# compute the best initial means and sds
best.initial.mean <- initial.means[which.min(mini)]
best.initial.sd <-initial.sds[which.min(mini)]

best.initial.mean
best.initial.sd
```

7

8

In [22]:
```
# run the optimization and check the estimate values
nlm(neg.loglik, c(best.initial.mean, best.initial.mean),
    sample=X)
```

$minimum
15.3323816074311
$estimate
44.9999794557662 11.1803341594842
$gradient
6.26067385221393e-08 -8.73852469643192e-09
$code
1
$iterations
23

- The values of the estimate, 44.99998 and 11.18033 are very close to the values (45 and 12) we got from mean(X) and var(X)

Estimating the Parameters of a Simple Linear Regression

In [23]:
```
# let's use the cars data in R
head(cars)
```

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10

In [24]:
```
# specify the sum of square error function to be minimized
sse <- function(thetha, x, y){
    return (sum((y-thetha[1]-thetha[2]*x)^2))

}
```

In [25]:
```
# let's first find best initial parameter values to use
mini = rep(NA, 100)
counter = 1
I = rep(NA, 100)
J = rep(NA, 100)
for (i in 0:10){
    for (j in 0:10){
       m <- nlm(sse, c(i, j), x=cars$speed, y=cars$dist)$minimum
       mini[counter] <- m
       I[counter] <- i
       J[counter] <- j

       counter = counter + 1
    }
}
which.min(mini)
```

24

In [26]:
```
I[which.min(mini)]
```

2

In [27]:
```
J[which.min(mini)]
```

1

In [28]:
```
nlm(sse, c(2, 1), x=cars$speed, y=cars$dist)
```

$minimum
11353.5210510949
$estimate
-17.5790947370236 3.93240874139822

$gradient
-1.19497616009811e-05 -0.000232586262527155

$code
1
$iterations
9

- Let's check the estimates using the lm() function

In [29]:
```
x = cars$speed
y = cars$dist
lm(y~x, data=cars)
```

```
Call:
lm(formula = y ~ x, data = cars)

Coefficients:
(Intercept)            x
    -17.579        3.932
```

- The estimates are very similar to the estimates obtained through the nlm() minimization function

So, in the nutshell, if you know the cost or loss function of a model, you can always optimize it to get the parameter estimates.