Maximum likelihood¶

Maximum Likelihood Estimation

The maximum likelihood estimation is a method or principle used to estimate the parameter or parameters of a model given observation or observations. Maximum likelihood estimation is also abbreviated as MLE, and it is also known as the method of maximum likelihood. From this name you probably already understood that this principle works by maximizing the likelihood, therefore the key to understand the maximum likelihood estimation is to first understand what is a likelihood and why someone would want to maximize it in order to estimate model parameters.

Let's start with the definition of the likelihood function for continuous case:

$$\mathcal{L}(\theta|x) = p_\theta(x)$$

The left term means "the likelihood of the parameters θ, given data x". Now, what does that mean ? It means that in the continuous case, the likelihood of the model $p_\theta(x)$ with the parametrization θ and data x is the probability density function (pdf) of the model with that particular parametrization.

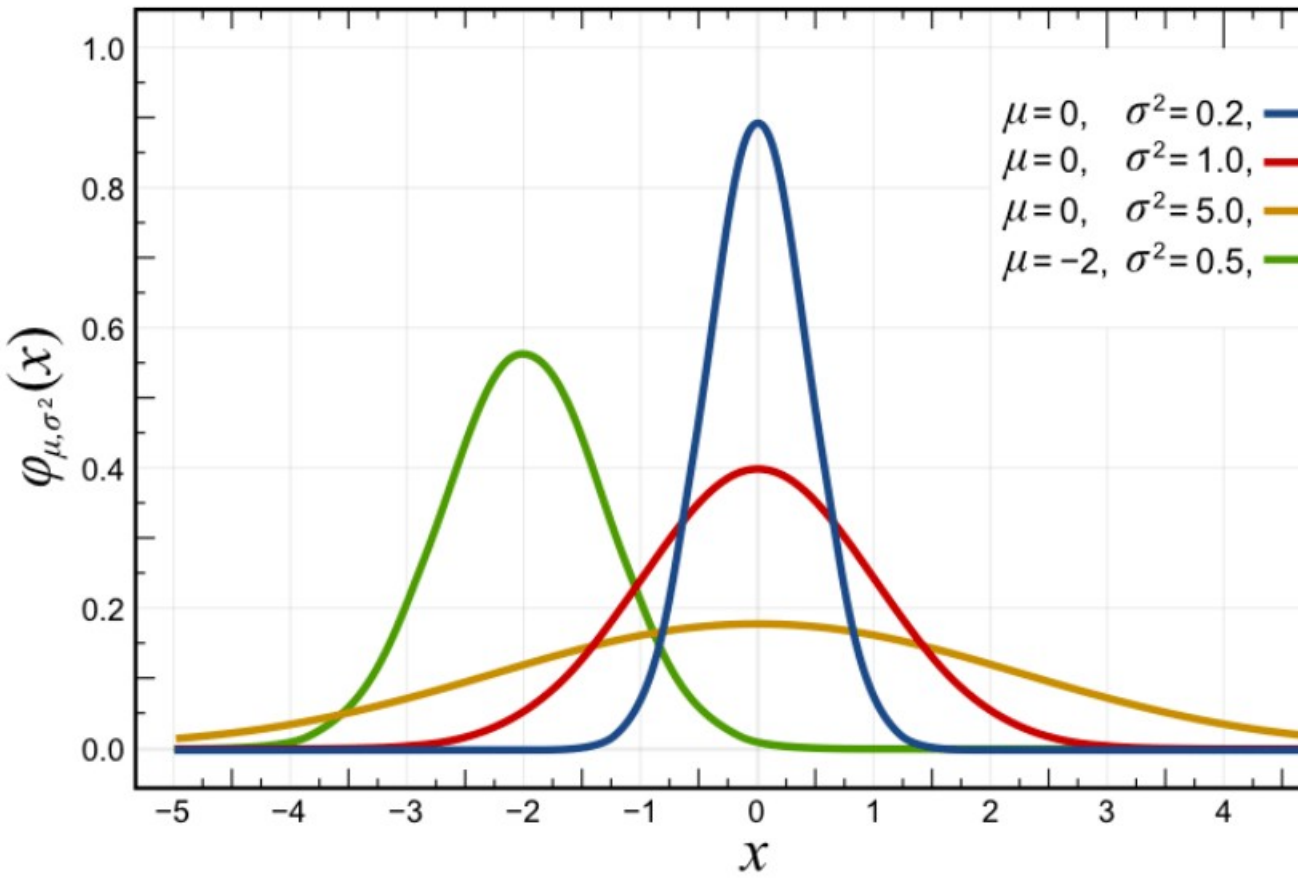
Although this is the most used likelihood representation, you should pay attention that the notation $\mathcal{L}(\cdot|\cdot)$ in this case doesn't mean the same as the conditional notation, so be careful with this overload, because it is always implicitly stated and it is also often a source of confusion. Another representation of the likelihood that is often used is $\mathcal{L}(x;\theta)$, which is better in the sense that it makes it clear that it's not a conditional, however, it makes it look like the likelihood is a function of the data and not of the parameters.

The model $p_\theta(x)$ can be any distribution, and to make things concrete, let's say that we are assuming that the data generating distribution is an univariate Gaussian distribution, which we define below:

$$p(x) \sim \mathcal{N}(\mu, \sigma^2)$$
$$p(x; \mu, \sigma^2) \sim \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

If you plot this probability density function with different parametrizations, you'll get something like the plots below, where the red distribution is the standard Gaussian $p(x) \sim \mathcal{N}(0, 1.0)$:

If you plot this probability density function with different parametrizations, you'll get some
like the plots below, where the red distribution is the standard Gaussian $p(x) \sim \mathcal{N}(0, 1.0)$



A selection of Normal Distribution Probability Density Functions (PDFs). Both the mean, μ, and variance, σ^2, a varied. The key is given on the graph. **Source**: Wikimedia Commons.

As you can see in the probability density function (pdf) plot above, the likelihood of x at variou given realizations are showed in the y-axis. Another source of confusion here is that usually, people take this as a probability, because they usually see these plots of normals and the likelihood is always below 1, however, the probability density function doesn't give you probabilities but densities. The constraint on the pdf is that it must integrate to one:

$$\int_{-\infty}^{+\infty} f(x)dx = 1$$

So, it is completely normal to have densities larger than 1 in many points for many different distributions. Take for example the pdf for the Beta distribution below:

So, returning to our original principle of maximum likelihood estimation, what we want is to maximize the likelihood $\mathcal{L}(\theta|x)$ for our observed data. What this means in practical terms is that we want to find the parameters θ of our model where the likelihood that this model generated our data is maximized, we want to find which **parameters of this model are most plausible** to have generated this observed data, or what are the parameters that make this sample most probable ?

For the case of our univariate Gaussian model, what we want is to find the parameters μ and σ^2, which for convenient notation we collapse into a single parameter vector:

$$\theta = \begin{bmatrix} \mu \\ \sigma^2 \end{bmatrix}$$

Because these are the statistics that completely define our univariate Gaussian model. So let's formulate the problem of the maximum likelihood estimation:

$$\begin{aligned} \hat{\theta} &= \arg\max_{\theta} \mathcal{L}(\theta|x) \\ &= \arg\max_{\theta} p_{\theta}(x) \end{aligned}$$

This says that we want to obtain the maximum likelihood estimate $\hat{\theta}$ that approximates $p_\theta(x)$ a underlying "true" distribution $p_{\theta^*}(x)$ by maximizing the likelihood of the parameters θ given data x. You shouldn't confuse a maximum likelihood estimate $\hat{\theta}(x)$ which is a realization of the maximum likelihood estimator for the data x, with the maximum likelihood estimator $\hat{\theta}$, so pay attention to disambiguate it in your head.

However, we need to incorporate multiple observations in this formulation, and by adding multiple observations we end up with a complex joint distribution:

$$\hat{\theta} = \arg\max_{\theta} p_\theta(x_1, x_2, \ldots, x_n)$$

That needs to take into account the interactions between all observations. And here is where w make a strong assumption: we state that the **observations are independent**. Independent random variables mean that the following holds:

$$p_\theta(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p_\theta(x_i)$$

Which means that since $x_1, x_2, \ldots, x_n$ don't contain information about each other, we can write the joint probability as a product of their marginals.

Another assumption that is made, is that these random variables are **identically distributed**, which means that they came from the same generating distribution, which allows us to model it with the same distribution parametrization.

Given these two assumptions, which are also known as IID (independently and identically distributed), we can formulate our maximum likelihood estimation problem as:

$$\hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{n} p_\theta(x_i)$$

Note that MLE doesn't require you to make these assumptions, however, many problems will appear if you don't to it, such as different distributions for each sample or having to deal with joint probabilities.

Given that in many cases these densities that we multiply can be very small, multiplying one by the other in the product that we have above we can end up with very small values. Here is where the logarithm function makes its way to the likelihood. The log function is a strictly monotonically increasing function, that preserves the location of the *extrema* and has a very nice property:

$$\log ab = \log a + \log b$$

Where the logarithm of a product is the sum of the logarithms, which is very convenient for us, so we'll apply the logarithm to the likelihood to maximize what is called the log-likelihood:

$$\begin{aligned}\hat{\theta} &= \arg\max_{\theta} \prod_{i=1}^{n} p_\theta(x_i) \\ &= \arg\max_{\theta} \sum_{i=1}^{n} \log p_\theta(x_i)\end{aligned}$$

As you can see, we went from a product to a summation, which is much more convenient. Another reason for the application of the logarithm is that we often take the derivative and solve it for the parameters, therefore is much easier to work with a summation than a multiplication.

We can also conveniently average the log-likelihood (*given that we're just including a multiplication by a constant*):

$$\begin{aligned}\hat{\theta} &= \arg\max_{\theta} \sum_{i=1}^{n} \log p_\theta(x_i) \\ &= \arg\max_{\theta} \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(x_i)\end{aligned}$$

This is also convenient because it will take out the dependency on the number of observations. We also know, that through the law of large numbers, the following holds as $n \to \infty$:

$$\frac{1}{n} \sum_{i=1}^{n} \log\, p_\theta(x_i) \approx \mathbb{E}_{x \sim p_{\theta^*}(x)} \left[\log\, p_\theta(x)\right]$$

As you can see, we're approximating the expectation with the **empirical expectation** defined by our dataset $\{x_i\}_{i=1}^{n}$. This is an important point and it is usually implictly assumed.

The weak law of large numbers can be bounded using a Chebyshev bound, and if you are interested in concentration inequalities, I've made an article about them here where I discuss the Chebyshev bound.

To finish our formulation, given that we usually minimize objectives, we can formulate the same maximum likelihood estimation as the minimization of the negative of the log-likelihood:

$$\hat{\theta} = \arg\min_{\theta} -\mathbb{E}_{x \sim p_{\theta^*}(x)} \left[\log p_{\theta}(x)\right]$$

Which is exactly the same thing with just the negation turn the maximization problem into a minimization problem.

Basic MLE parameter estimation simulation¶

https://towardsdatascience.com/maximum-likelihood-estimation-in-r-b21f68f1eba4

Now, there are many ways of estimating the parameters of your chosen model from the data you have. The simplest of these is the method of moments — an effective tool, but one not without its disadvantages (notably, these estimates are often biased).

Another method you may want to consider is Maximum Likelihood Estimation (MLE), which tends to produce better (ie more unbiased) estimates for model parameters. It's a little more technical, but nothing that we can't handle. Let's see how it works.

What is likelihood?¶

The likelihood — more precisely, the likelihood function — is a function that represents how likely it is to obtain a certain set of observations from a given model. We're considering the set of observations as fixed — they've happened, they're in the past — and now we're considering under which set of model parameters we would be most likely to observe them.

In [ ]:

In [2]:

```
#A simple coin-flipping example
set.seed(22)
```

```
# Generate an outcome, ie number of heads obtained, assuming a fair coin was used for
the 100 flips
heads <- rbinom(1,100,0.5)
heads
```

52

For the purposes of generating the data, we've used a 50/50 chance of getting a heads/tails, although we are going to pretend that we don't know this for the time being. For almost all real world problems we don't have access to this kind of information on the processes that generate the data we're looking at — which is entirely why we are motivated to estimate these parameters!

> Under our formulation of the heads/tails process as a binomial one, we are supposing that there is a probability p of obtaining a heads for each coin flip. Extending this, the probability of obtaining 52 heads after 100 flips is given by:

$$\frac{52!}{48! \cdot 52!} p^{52}(1-p)^{48}$$

This probability is our likelihood function — it allows us to calculate the probability, ie how likely it is, of that our set of data being observed given a probability of heads p. You may be able to guess the next step, given the name of this technique — we must find the value of p that maximises this likelihood function.

In [3]:

```
# To illustrate, let's find the likelihood of obtaining these results if p was 0.6—that
is, if our coin was biased in such a way to show heads 60% of the time.
biased_prob <- 0.6
# Explicit calculation
choose(100,52)*(biased_prob**52)*(1-biased_prob)**48

# Using R's dbinom function (density function for a given binomial distribution)
dbinom(heads,100,biased_prob)
```

0.0214877567069514
0.0214877567069513

Back to our problem — we want to know the value of p that our data implies. For simple situations like the one under consideration, it's possible to differentiate the likelihood function with respect to the parameter being estimated and equate the resulting expression to zero in order to solve for the MLE estimate of p. However, for more complicated (and realistic) processes, you will probably have to resort to doing it numerically.

Luckily, this is a breeze with R as well! Our approach will be as follows:

1. Define a function that will calculate the likelihood function for a given value of p; then
2. Search for the value of p that results in the highest likelihood.

Starting with the first step:

In [4]:

```
likelihood <- function(p){
  dbinom(heads, 100, p)
}
# Test that our function gives the same result as in our earlier example
likelihood(biased_prob)
```

0.0214877567069513

And now considering the second step. There are many different ways of optimising (ie maximising or minimising) functions in R — the one we'll consider here makes use of the *nlm* function, which stands for non-linear minimisation. If you give nlm a function and indicate which parameter you want it to vary, it will follow an algorithm and work iteratively until it finds the value of that parameter which minimises the function's value.

You may be concerned that I've introduced a tool to *minimise* a function's value when we really are looking to *maximise* — this is maximum likelihood estimation, after all! Fortunately, *maximising a function is equivalent to minimising the function multiplied by minus one.* If we create a new function that simply produces the likelihood multiplied by minus one, then the parameter that minimises the value of this new function will be exactly the same as the parameter that maximises our original likelihood.

As such, a small adjustment to our function from before is in order:

In [5]:

```
negative_likelihood <- function(p){
  dbinom(heads, 100, p)*-1
}
# Test that our function is behaving as expected
negative_likelihood(biased_prob)
# -0.0214877567069513
```

-0.0214877567069513

In [6]:

```
#Excellent — we're now ready to find our MLE value for p.

nlm(negative_likelihood,0.5,stepmax=0.5)
```

$minimum

-0.0796525598193163

$estimate

0.519999499998771

$gradient

-2.77555756156289e-11

$code

1

$iterations

4

In [ ]:

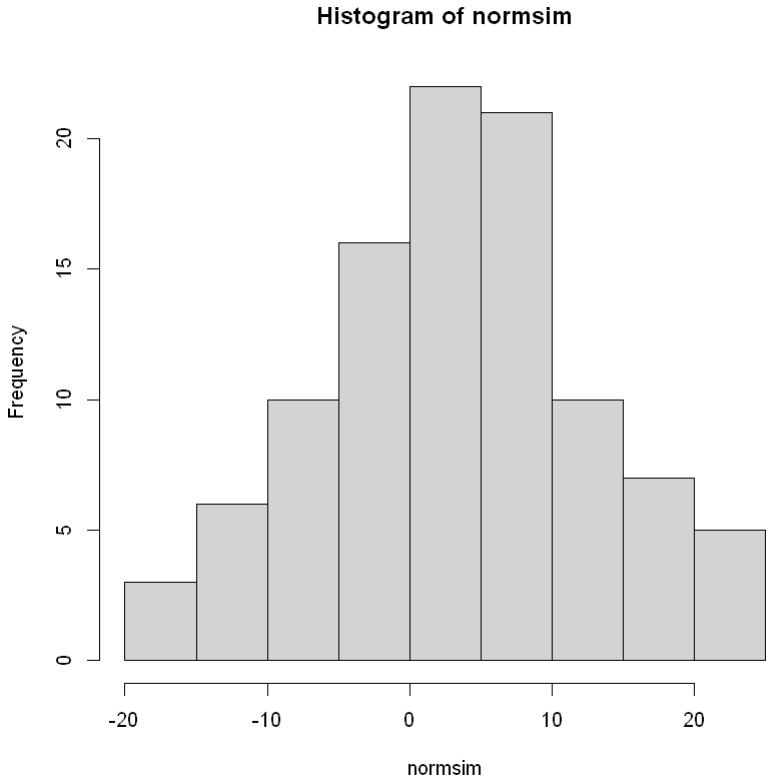
Normal distirbution example¶

In [8]:

```
normsim <- rnorm(100,5,10)
```

In [9]:

```
hist(normsim)
```



If we assign an statistical model to the random population, any particular value (let's call it x_i) sampled from the population will have a probability density according to the model (let's call it $f(x_i)$). If we then assume that all the values in our sample are statistically independent (i.e. the probability of sampling a particular value does not depend on the rest of values already sampled), then the *likelihood* of observing the whole sample (let's call it $L(x)$) is defined as the product of the probability densities of the individual values (i.e. $L(x) = \prod_{i=1}^{i=n} f(x_i)$ where n is the size of the sample).

For example, if we assume that the data were sampled from a Normal distribution, the likelihood is defined as:

$$L(x) = \prod_{i=1}^{i=n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

Thus we make the assumption that the variables are independent

Let's start with the *probability density function* for one observation x from normal random variable with mean μ and variance σ^2,

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{1}$$

For a set of n replicate independent observations $x_1, x_2, \cdots, x_n$ the joint density is

$$f(x_1, x_2, \cdots, x_n|\mu,\sigma) = \prod_{i=1}^{n} f(x_i|\mu,\sigma) \tag{2}$$

We interpret this as follows: given the values of μ and σ, equation (2) tells us the relative probability of different possible values of the observations. Maximum likelihood turns this around by defining the *likelihood function*

$$\mathcal{L}(\mu,\sigma|x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} f(x_i|\mu,\sigma) \tag{3}$$

In [10]:

```
#the product of the density functions for each observation

prod(dnorm(normsim))
```

0

One complication of the MLE method is that, as probability densities are often smaller than 1, the value of $L(x)$ can become very small as the sample size grows. For example the likelihood of 100 values sampled from a standard Normal distribution is very small:

```
set.seed(2019)
sample = rnorm(100)
prod(dnorm(sample))
## [1] 2.23626e-58
```

When the variance of the distribution is small it is also possible to have probability densities higher than one. In this case, the likelihood function will grow to very large values. For example, for a Normal distribution with standard deviation of 0.1 we get:

```
sample_large = rnorm(100, sd = 0.1)
prod(dnorm(sample_large, sd = 0.1))
## [1] 2.741535e+38
```

The reason why this is a problem is that computers have a limited capacity to store the digits of a number, so they cannot store very large or very small numbers. If you repeat the code above but using sample sizes of say 1000, you will get `0` or `Inf` instead of the actual values, because your computer will just give up. Although it is possible to increase the amount of digits to be stored per number, this does not really solve the problem, as it will eventually come back with larger samples. Furthermore, in most cases we will need to use numerical optimization algorithms (see below) which will make the problem even worse. Therefore, we cannot work directly with the likelihood function.

One trick is to use the natural logarithm of the likelihood function instead ($log(L(x))$). A nice property is that the logarithm of a product of values is the sum of the logarithms of those values, that is:

$$\log(L(x)) = \sum_{i=1}^{i=n} \log(f(x_i))$$

Also, the values of log-likelihood will always be closer to 1 and the maximum occurs for the same parameter values as for the likelihood. For example, the likelihood of the first sample generated above, as a function of μ (fixing σ) is:

Enough with the theory. Let's estimate the values of μ and σ from the first sample we generated above. First, we need to create a function to calculate NLL. It is good practice to follow some template for generating these functions. An NLL function should take two inputs: (i) a vector of parameter values that the optimization algorithm wants to test (`pars`) and (ii) the `data` for which the NLL is calculated. For the problem of estimating μ and σ, the function looks like this:

In [12]:

```
NLL = function(pars, data) {
  # Extract parameters from the vector
  mu = pars[1]
  sigma = pars[2]
  # Calculate Negative Log-LIkelihood
  -sum(dnorm(x = data, mean = mu, sd = sigma, log = TRUE))
}
```

We can now minimize the NLL using the function `optim` . This function needs the initial values for each parameter (`par`), the function calculating NLL (`fn`) and arguments that will be passed to the objective function (in our example, that will be `data`). We can also tune some settings with the `control` argument. I recommend to set the setting `parscale` to the absolute initial values (assuming none of the initial values are 0). This setting determines the scale of the values you expect for each parameter and it helps the algorithm find the right solution. The `optim` function will return an object that holds all the relevant information and, to extract the optimal values for the parameters, you need to access the field `par` :

```
mle = optim(par = c(mu = 0.2, sigma = 1.5), fn = NLL, data = sample,
            control = list(parscale = c(mu = 0.2, sigma = 1.5)))
mle$par
##          mu       sigma
## -0.07332745  0.90086176
```

It turns out that this problem has an analytical solution, such that the MLE values for μ and σ from the Normal distribution can also be calculated directly as:

```
c(mu = mean(sample), sigma = sd(sample))
##         mu      sigma
## -0.0733340  0.9054535
```

In [18]:

```
mle = optim(par = c(mu = 3, sigma = 5), fn = NLL, data = normsim,
```

```
            control = list(parscale = c(mu = 1, sigma = 1)))
mle$par
```

mu
3.03543943144177
sigma
9.33691424541757

ANOTHER SIMULATED EXAMPLE¶

https://rpubs.com/cbw1243/mle

In [ ]:

```
n <- 1000
x <- rnorm(n,2,3) # with mean = 2, sd = 3. (true parameter)
```

In [30]:

```
LL <- function(beta, sigma){
  R = dnorm(x, beta, sigma, log = TRUE)
  -sum(R)
}
```

In [ ]:

```
library(bbmle)
```

In [31]:

```
fit_norm <- mle2(LL, start = list(beta = 0, sigma = 1), lower = c(-Inf, 0), upper =
c(Inf, Inf), method = 'L-BFGS-B')
```

```
Warning message in fix_order(call$lower, "lower bounds", -Inf):
"lower bounds not named: rearranging to match 'start'"
```

In [32]:

```
summary(fit_norm)
```

```
Maximum likelihood estimation

Call:
mle2(minuslogl = LL, start = list(beta = 0, sigma = 1), method = "L-BFGS-B",
    lower = c(-Inf, 0), upper = c(Inf, Inf))

Coefficients:
      Estimate Std. Error z value     Pr(z)
```

```
beta  1.818889   0.091969  19.777 < 2.2e-16 ***
sigma 2.908327   0.065032  44.721 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

-2 log L: 4973.033
```

In [34]:

```
# create a df with multiple sigma and beta values
library(tidyverse)
b=seq(0.1,to=4,by=0.1)
m=seq(0.1,to=4,by=0.1)

df=expand.grid(b,m) %>%as.data.frame()
names(df)=c("beta","sigma")
```

```
Warning message:
"package 'tidyverse' was built under R version 4.1.2"
-- Attaching packages
---------------------------------------------------------------------------------
tidyverse 1.3.1 --

v ggplot2 3.3.6     v purrr   0.3.4
v tibble  3.1.2     v dplyr   1.0.7
v tidyr   1.1.3     v stringr 1.4.0
v readr   1.4.0     v forcats 0.5.1

Warning message:
"package 'ggplot2' was built under R version 4.1.3"
-- Conflicts
------------------------------------------------------------------------------------
tidyverse_conflicts() --
x dplyr::filter()  masks stats::filter()
x purrr::flatten() masks jsonlite::flatten()
x dplyr::lag()     masks stats::lag()
x dplyr::slice()   masks bbmle::slice()
```

In [40]:

```
dfres=cbind(df,mapply(LL,df$beta,df$sigma))
names(dfres)[3]="LL"
##LL is to be minimized
```

In [39]:

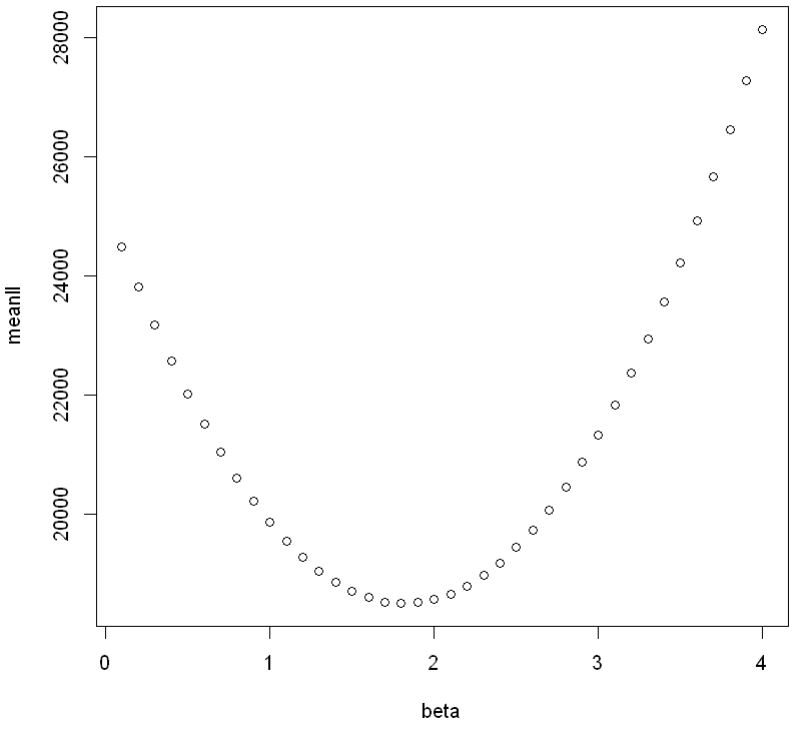
```
dfres %>% arrange(LL) %>% head
```

A data.frame: 6 × 3

	beta	sigma	LL
	<dbl>	<dbl>	<dbl>
1	1.8	2.9	2486.546
2	1.9	2.9	2486.916
3	1.7	2.9	2487.365
4	1.8	3.0	2487.480
5	1.9	3.0	2487.826
6	1.8	2.8	2488.017

In [41]:

```
dfres %>% group_by(beta) %>% summarize(meanll=mean(LL)) %>% plot
```



In [42]:

```
#thus the minimization on average (sigma) happens arround 1.8
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: