Getting started with maximum likelihood and `maxLik`

Ott Toomet

December 29, 2025

1 Introduction

This vignette is intended for readers who are unfamiliar with the concept of likelihood, and for those who want a quick intuitive brush-up. The potential target group includes advanced undergraduate students in technical fields, such as statistics or economics, graduate students in social sciences and engineering who are devising their own estimators, and researchers and practitioners who have little previous experience with ML. However, one should have basic knowledge of `R` language. If you are familiar enough with the concept of likelihood and maximum likelihood, consult instead the other vignette "Maximum Likelihood Estimation with `maxLik`".

Maximum Likelihood (ML) in its core is maximizing the *likelihood* over the parameters of interest. We start with an example of a random experiment that produces discrete values to explain what is likelihood and how it is related to probability. The following sections cover continuous values, multiple parameters in vector form, and we conclude with a linear regression example. The final section discusses the basics of non-linear optimization. The examples are supplemented with very simple code and assume little background besides basic statistics and basic `R` knowledge.

2 Discrete Random Values

We start with a discrete case. "Discrete" refers to random experiments or phenomena with only limited number of possible outcomes, and hence we can compute and tabulate every single outcome separately.

Imagine you are flipping a fair coin. What are the possible outcomes and what are the related probabilities? Obviously, in case of a coin there are only two outcomes, heads H and tails T. If the coin is fair, both of these will have probability of exactly 0.5. Such random experiment is called *Bernoulli process*. More specifically, this is *Bernoulli(0.5)* process as for the fair coin the probability of "success" is 0.5 (below we consider success to be heads, but you can choose tails as well). If the coin is not fair, we denote the corresponding process Bernoulli(p), where p is the probability of heads.

Now let us toss the coin two times. What is the probability that we end up with one heads and one tails? As the coin flips are independent,[1] we can just

[1] Events are independent when outcome of one event does not carry information about the

multiply the probabilities: 0.5 for a single heads and 0.5 for a single tails equals 0.25 when multiplied. However, this is not the whole story–there are two ways to get one heads and one tails, either H first and T thereafter or T first and H thereafter. Both of these events are equally likely, so the final answer will be 0.5.

But now imagine we do not know if the coin is fair. Maybe we are not tossing a coin but an object of a complex shape. We can still label one side as "heads" and the other as "tails". But how can we tell what is the probability of heads? Let's start by denoting this probability with p. Hence the probability of tails will be $1 - p$, and the probability to receive one heads, one tails when we toss the object two times will be $2p(1 - p)$: p for one heads, $1 - p$ for one tails, and "2" takes into account the fact that we can get this outcome in two different orders.

This probability is essentially likelihood. We denote likelihood with $\mathcal{L}(p)$, stressing that it depends on the unknown probability p. So in this example we have

$$\mathcal{L}(p) = 2\,p\,(1 - p). \tag{1}$$

p is the *model parameter*, the unknown number we want to compute with the help of likelihood.

Let's repeat here what did we do above:

1. We observe data. In this example data contains the counts: one heads, one tails.

2. We model the coin toss experiment, the data generating process, as Bernoulli(p) random variable. p, the probability of heads, is the model parameter we want to calculate. Bernoulli process has only a single parameter, but more complex processes may contain many more.

3. Thereafter we compute the probability to observe the data based on the model. Here it is equation (1). This is why we need a probability model. As the model contains unknown parameters, the probability will also contain parameters.

4. And finally we just call this probability *likelihood* $\mathcal{L}(p)$. We write it as a function of the parameter to stress that the parameter is what we are interested in. Likelihood also depends on data (the probability will look different for e.g. two heads instead of a head and a tail) but we typically do not reflect this in notation.

The next task is to use this likelihood function to *estimate* the parameter, to use data to find the best possible parameter value. *Maximum likelihood* (ML) method finds such parameter value that maximizes the likelihood function. It can be shown that such parameter value has a number of desirable properties, in particular it will become increasingly similar to the "true value" on an increasingly large dataset (given that our probability model is correct).[2] These desirable properties, and relative simplicity of the method, have made ML one of the most widely used statistical estimators.

outcome of the other event. Here the result of the second toss is not related to the outcome of the first toss.

[2]This property is formally referred to as *consistency*. ML is a consistent estimator.

Let us generalize the example we did above for an arbitrary number of coin flips. Assume the coin is of unknown "fairness" where we just denote the probability to receive heads with p. Further, assume that out of N trials, N_H trials were heads and N_T trials were tails. The probability of this occuring is

$$\binom{N}{N_H} p^{N_H} (1-p)^{N_T} \tag{2}$$

p^{N_H} is the probability to get N_H heads, $(1-p)^{N_T}$ is the probability to get N_T tails, and the binomial coefficient $\binom{N}{N_H} = \frac{N!}{N_H!(N-N_H)!}$ takes into account that there are many ways how heads and tail can turn up while still resulting N_H heads and N_T tails. In the previous example $N = 2$, $N_H = 1$ and there were just two possible combinations as $\binom{2}{1} = 2$. The probability depends on both the parameter p and data–the corresponding counts N_H and N_T. Equation (2) is essentially likelihood–probability to observe data. We are interested how does it depend on p and stress this by writing p in the first position followed by semicolon and data as we care less about the dependency on data:

$$\mathcal{L}(p; N_H, N_T) = \binom{N}{N_H} p^{N_H} (1-p)^{N_T} \tag{3}$$

Technically, it is easier to work with log-likelihood instead of likelihood (as log is monotonic function, maximum of likelihood and maximum of log-likelihood occur at the same parameter value). We denote log-likelihood by ℓ and write

$$\ell(p; N_H, N_T) = \log \mathcal{L}(p; N_H, N_T) = \log \binom{N}{N_H} + N_H \log p + N_T \log(1-p). \tag{4}$$

ML estimator of p is the value that maximizes this expression. Fortunately, in this case the binomial coefficient $\binom{N}{N_H}$ depends only on data but not on the p. Intuitively, p determines the probability of various combinations of heads and tails, but *what kind of combinations are possible* does not depend on p. Hence we can ignore the first term on the right hand side of (4) when maximizing the log-likelihood. Such approach is very common in practice, many terms that are invariant with respect to parameters are often ignored. Hence, with we can re-define the log-likelihood as

$$\ell(p; N_H, N_T) = N_H \log p + N_T \log(1-p). \tag{5}$$

It is easy to check that the solution, the value of p that maximizes log-likelihood (5) is[3]

$$p^* = \frac{N_H}{N_H + N_T} = \frac{N_H}{N}. \tag{6}$$

This should be surprise to no-one: the intuitive "fairness" of the coin is just the average percentage of heads we get.

Now it is time to try this out on computer with `maxLik`. Let's assume we toss a coin and receive $H_H = 3$ heads and $H_T = 7$ tails:

[3] Just differentiate $\ell(p)$ with respect to p, set the result to zero, and isolate p.

```
> NH <- 3
> NT <- 7
```

Next, we have to define the log-likelihood function. It has to be a function of the parameter, and the parameter must be its first argument. We can access data in different ways, for instance through the `R` workspace environment. So we can write the log-likelihood as

```
> loglik <- function(p) {
+    NH*log(p) + NT*log(1-p)
+ }
```

And finally, we can use `maxLik` function to compute the likelihood. In its simplest form, `maxLik` requires two arguments: the log-likelihood function, and the start value for the iterative algorithm (see Section 6, and the documentation and vignette *Maximum Likelihood Estimation with* `maxLik` for more detailed explanations). The start value must be a valid parameter value (the loglik function must not give errors when called with the start value). We can choose $p_0 = 0.5$ as the initial value, and let the algorithm find the best possible p from there:

```
> library(maxLik)
> m <- maxLik(loglik, start=0.5)
> summary(m)

--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 2 iterations
Return code 1: gradient close to zero (gradtol)
Log-Likelihood: -6.108643
1  free parameters
Estimates:
     Estimate Std. error t value Pr(> t)
[1,]   0.3000     0.1449    2.07  0.0384 *
---
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
--------------------------------------------
```

As expected, the best bet for p is 0.3. Our intuitive approach–the percentage of heads in the experiment–turns also out to be the ML estimate.

Next, we look at an example with continuous outcomes.

3 Continuous case: probability density and likelihood

In the example above we looked at a discrete random process, a case where there were only a small number of distinct possibilities (heads and tails). Discrete cases are easy to understand because we can actually compute the respective probabilities, such as the probability to receive one heads and one tails in

our experiment. Now we consider continuous random variables where the outcome can be any number in a certain interval. Unfortunately, in continuous case we cannot compute probability of any particular outcome. Or more precisely–we can do it, but the answer is always 0. This may sound a little counter-intuitive but perhaps the following example helps. If you ask the computer to generate a single random number between 0 and 1, you may receive 0.444768540561199. What is the probability to get the same number again? You can try, you will get close but you won't get exactly the same number.[4] But despite the probability to receive this number is zero, we somehow still produced it in the first place. Clearly, zero probability does not mean the number was impossible. However, if we want to receive a negative number from the same random number generator, it will be impossible (because we chose a generator that only produces numbers between 0 and 1). So probability 0-events may be possible and they may also be impossible. And to make matter worse, they may also be more likely and less likely. For instance, in case of standard normal random numbers (these numbers are distributed according to "bell curve") the values near 0 are much more likely than values around -2, despite of the probability to receive any particular number still being 0 (see Figure 1).

The solution is to look not at the individual numbers but narrow interval near these numbers. Consider the number of interest x_1, and compute the probability that the random outcome X falls into the narrow interval of width δ, $[x_1 - \delta/2,\, x_1 + \delta/2]$, around this number (Figure 1). Obviously, the smaller the width δ, the less likely it is that X falls into this narrow interval. But it turns out that when we divide the probability by the width, we get a stable value at the limit which we denote by $f(x_1)$:

$$f(x_1) = \lim_{\delta \to 0} \frac{\Pr(X \in [x_1 - \delta/2,\, x_1 + \delta/2])}{\delta}. \tag{7}$$

In the example on the Figure the values around x_1 are less likely than around x_2 and hence $f(x_1) < f(x_2)$. The result, $f(x)$, is called *probability density function*, often abbreviated as *pdf*. In case of continuous random variables, we have to work with pdf-s instead of probabilities.

Consider the following somewhat trivial example: we have sampled two independent datapoints x_1 and x_2 from normal distribution with variance 1 and mean (expected value) equal to μ. Say, $x_1 = 2.976$ and $x_2 = -0.711$. Assume we do not know μ and use ML to estimate it. We can proceed in a similar steps as what we did for the discrete case: i) observe data, in this case x_1 and x_2; ii) set up the probability model; iii) use the model to compute probability to observe the data; iv) write the probability as $\ell(\mu)$, log-likelihood function of the parameter μ; v) and finally, find μ^*, the μ value that maximizes the corresponding log-likelihood. This will be our best estimate for the true mean.

As we already have our data points x_1 and x_2, our next step is the probability model. The probability density function (pdf) for normal distribution with mean μ and variance 1 is

$$f(x;\mu) = \frac{1}{\sqrt{2\pi}}\, \mathrm{e}^{-\frac{1}{2}(x-\mu)^2} \tag{8}$$

[4] As computers operate with finite precision, the actual chances to repeat any particular random number are positive, although small. The exact answer depends on the numeric precision and the quality of random number generator.

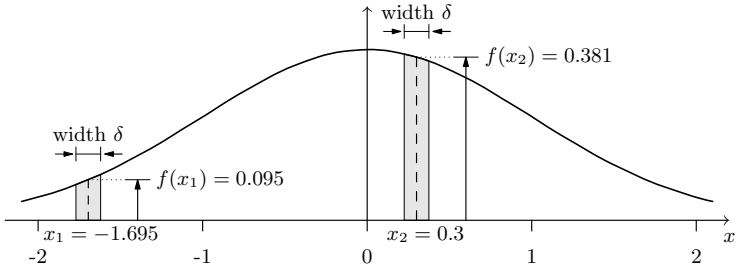


Figure 1: Standard normal probability density (thick black curve). While $\Pr(X = x_1) = 0$*, i.e. the probability to receive a random number exactly equal to* x_1 *is 0, the probability to receive a random number in the narrow interval of width* δ *around* x_1 *is positive. In this example, the probability to get a random number in the interval around* x_2 *is four times larger than for the interval around* x_1*.*

(This is the thick curve in Figure 1). We write it as $f(x;\mu)$ as pdf is usually written as a function of data. But as our primary interest is μ, we also add this as an argument. Now we use this pdf and (7) to find the probability that we observe a datapoint in the narrow interval around x. Here it is just $f(x;\mu)\cdot\delta$. As x_1 and x_2 are independent, we can simply multiply the corresponding probabilities to find the combined probability that both random numbers are near their corresponding values:

$$\Pr\Big(X_1 \in [x_1-\delta/2, x_1+\delta/2] \quad \text{and} \quad X_2 \in [x_2-\delta/2, x_2+\delta/2]\Big) =$$

$$= \underbrace{\frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-\frac{1}{2}(x_1-\mu)^2}\cdot\delta}_{\text{First random value near } x_1} \times \underbrace{\frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-\frac{1}{2}(x_2-\mu)^2}\cdot\delta}_{\text{Second random value near } x_2} \equiv$$

$$\equiv \tilde{\mathcal{L}}(\mu; x_1, x_2). \quad (9)$$

The interval width δ must be small for the equation to hold precisely. We denote this probability with $\tilde{\mathcal{L}}$ to stress that it is essentially the likelihood, just not written in the way it is usually done. As in the coin-toss example above, we write it as a function of the parameter μ, and put data x_1 and x_2 after semicolon. Now we can estimate μ by finding such a value μ^* that maximizes the expression (9).

But note that δ plays no role in maximizing the likelihood. It is just a multiplicative factor, and it cannot be negative because it is a width. So for our maximization problem we can just ignore it. This is what is normally done when working with continuous random variables. Hence we write the likelihood as

$$\mathcal{L}(\mu; x_1, x_2) = \frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-\frac{1}{2}(x_1-\mu)^2} \times \frac{1}{\sqrt{2\pi}}\,\mathrm{e}^{-\frac{1}{2}(x_2-\mu)^2}. \quad (10)$$

We denote this by $\mathcal{L}$ instead of $\tilde{\mathcal{L}}$ to stress that this is how likelihood function for continuous random variables is usually written.

Exactly as in the discrete case, it is better to use log-likelihood instead of likelihood to actually compute the maximum. From (10) we get log-likelihood

as

$$\ell(\mu; x_1, x_2) = -\log\sqrt{2\pi} - \frac{1}{2}(x_1 - \mu)^2 + (-\log\sqrt{2\pi}) - \frac{1}{2}(x_2 - \mu)^2 =$$

$$= -2\log\sqrt{2\pi} - \frac{1}{2}\sum_{i=1}^{2}(x_i - \mu)^2. \quad (11)$$

The first term, $-2\log\sqrt{2\pi}$, is just an additive constant and plays no role in the actual maximization but it is typically still included when defining the likelihood function.[5]

One can easily check by differentiating the log-likelihood function that the maximum is achieved at $\mu^* = \frac{1}{2}(x_1 + x_2)$. It is not surprising, our intuitive understanding of mean value carries immediately over to the normal distribution context.

Now it is time to demonstrate these results with `maxLik` package. First, create our "data", just two normally distributed random numbers:

```
> x1 <- rnorm(1)  # centered around 0
> x2 <- rnorm(1)
> x1

[1] 1.026914

> x2

[1] 0.575752
```

and define the log-likelihood function. We include all the terms as in the final version of (11):

```
> loglik <- function(mu) {
+    -2*log(sqrt(2*pi)) - 0.5*((x1 - mu)^2 + (x2 - mu)^2)
+ }
```

We also need the parameter start value–we can pick 0. And we use `maxLik` to find the best μ:

```
> m <- maxLik(loglik, start=0)
> summary(m)

--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 2 iterations
Return code 1: gradient close to zero (gradtol)
Log-Likelihood: -1.888764
1  free parameters
Estimates:
     Estimate Std. error t value Pr(> t)
[1,]   0.8013     0.7071   1.133   0.257
--------------------------------------------
```

[5] Additive or multiplicative constants do not play any role for optimization, but they are important when comparing different log-likelihood values. This is often needed for likelihood-based statistical tests.

The answer is the same as sample mean:

```
> (x1 + x2)/2

[1] 0.8013328
```

4 Vector arguments

The previous example is instructive but it does have very few practical applications. The problem is that we wrote the probability model as normal density with unknown mean μ but standard deviation σ equal to one. However, in practice we hardly ever know that we are dealing with unit standard deviation. More likely both mean and standard deviation are unknown. So we have to incorporate the unknown σ into the model.

The more general normal pdf with standard deviation σ is

$$f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}}\frac{1}{\sigma}\,\mathrm{e}^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}}. \tag{12}$$

Similar reasoning as what we did above will give the log-likelihood

$$\ell(\mu,\sigma;x_1,x_2) = -2\log\sqrt{2\pi} - 2\log\sigma - \frac{1}{2}\sum_{i=1}^{2}\frac{(x_i-\mu)^2}{\sigma^2}. \tag{13}$$

We write the log-likelihood as function of both parameters, μ and σ; the semicolon that separates data x_1 and x_2 shows that though the log-likelihood depends on data too, we are not much interested in that dependency for now. This formula immediately extends to the case of N datapoints as

$$\ell(\mu,\sigma) = -N\log\sqrt{2\pi} - N\log\sigma - \frac{1}{2}\sum_{i=1}^{N}\frac{(x_i-\mu)^2}{\sigma^2} \tag{14}$$

where we have dropped the dependency on data in the notation. In this case we can actually do the optimization analytically, and derive the well-known intuitive results: the best estimator for mean μ is the sample average, and the best estimator for σ^2 is the sample variance.

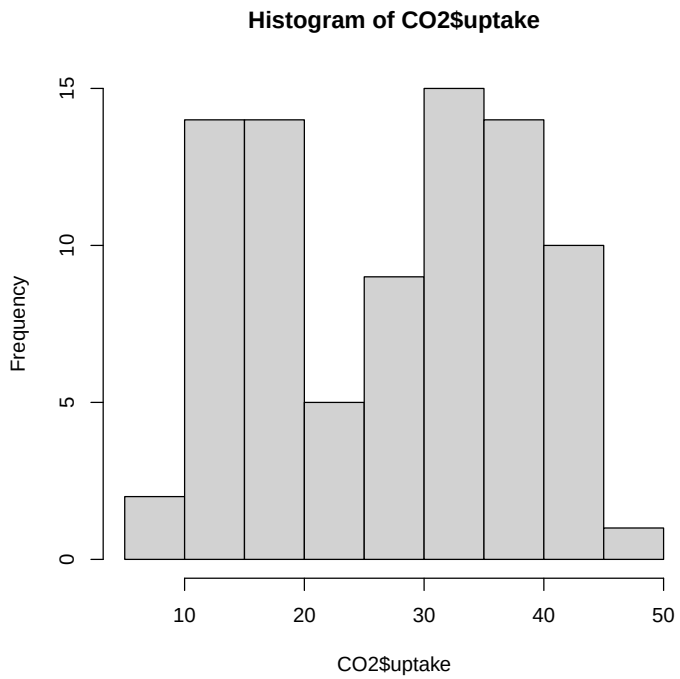
However, in general the expression cannot be solved analytically. We have to use numeric optimization to search for the best μ and σ combination. The common multi-dimensional optimizers rely on linear algebra and expect all the parameters submitted as a single vector. So we can write the log-likelihood as

$$\ell(\boldsymbol{\theta}) \quad \text{where} \quad \boldsymbol{\theta} = (\mu,\sigma). \tag{15}$$

Here we denote both parameters μ and σ as components of a single parameter vector $\boldsymbol{\theta}$. (Traditionally vectors are denoted by bold symbols.) We have also dropped dependency on data in notation, but remember that in practical applications log-likelihood always depends on data. This notation can be converted to computer code almost verbatim, just remember to extract the parameters μ and σ from $\boldsymbol{\theta}$ in the log-likelihood function.

Let us illustrate this using the *CO2* dataset (in package *datasets*). It describes CO_2 uptake (μmol/m^{2}sec, variable *uptake*) by different grasses in various conditions. Let us start by plotting the histogram of uptake:

```
> data(CO2)
> hist(CO2$uptake)
```



Let us model the uptake as a normal random variable with expected value μ and standard deviation σ. We code (14) while keeping both parameters in a single vector as in (15):

```
> loglik <- function(theta) {
+     mu <- theta[1]
+     sigma <- theta[2]
+     N <- nrow(CO2)
+     -N*log(sqrt(2*pi)) - N*log(sigma) -
+        0.5*sum((CO2$uptake - mu)^2/sigma^2)
+ }
```

The function is similar to the function `loglik` we used in Section 3. There are just two main differences:

- both arguments, μ and σ are passed as components of $\boldsymbol{\theta}$, and hence the function starts by unpacking the values.
- instead of using variables `x1` and `x2`, we now extract data directly from the data frame.

Besides these two differences, the formula now also includes σ and sums over all observations, not just over two observations.

As our parameter vector now contains two components, the start vector must also be of length two. Based on the figure we guess that a good starting value might be $\mu = 30$ and $\sigma = 10$:

```
> m <- maxLik(loglik, start=c(mu=30, sigma=10))
> summary(m)

--------------------------------------------
Maximum Likelihood estimation
Newton-Raphson maximisation, 5 iterations
Return code 8: successive function values within relative tolerance limit (reltol)
Log-Likelihood: -318.6817
2  free parameters
Estimates:
      Estimate Std. error t value Pr(> t)
mu     27.2131     1.1633   23.39  <2e-16 ***
sigma  10.7498     0.8389   12.81  <2e-16 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
--------------------------------------------
```

Indeed, our guess was close.

5 Final Example: Linear Regression

Now we have the main tools in place to extend the example above to a real statistical model. Let us build the previous example into linear regression. We describe CO_2 uptake (variable *uptake*) by CO_2 concentration in air (variable *conc*). We can write the corresponding regression model as

$$uptake_i = \beta_0 + \beta_1 \cdot conc_i + \epsilon_i. \tag{16}$$

In order to turn this regression model into a ML problem, we need a probability model. Assume that the disturbance term ϵ is normally distributed with mean 0 and (unknown) variance σ^2 (this is a standard assumption in linear regression). Now we can follow (13) and write log of pdf for a single observation as

$$\ell(\sigma; \epsilon_i) = -\log\sqrt{2\pi} - \log\sigma - \frac{1}{2}\frac{\epsilon_i^2}{\sigma^2}. \tag{17}$$

Here we have replaced x_i by the random outcome ϵ_i. As the expected value $\mu = 0$ by assumption, we do not include μ in (17) and hence we drop it also from the argument list of ℓ. We do not know ϵ_i but we can express it using linear regression model (16):

$$\epsilon_i = uptake_i - \beta_0 - \beta_1 \cdot conc_i. \tag{18}$$

This expression depends on two additional unknown parameters, β_0 and β_1. These are the linear regression coefficients we want to find.

Now we plug this into (17):

$$\ell(\beta_0, \beta_1, \sigma; uptake_i, conc_i) = \\ = -\log\sqrt{2\pi} - \log\sigma - \frac{1}{2}\frac{(uptake_i - \beta_0 - \beta_1 \cdot conc_i)^2}{\sigma^2}. \tag{19}$$

We have designed log-likelihood formula for a single linear regression observation. It depends on three parameters, β_0, β_1 and σ. For N observations we have

$$\ell(\beta_0, \beta_1, \sigma; \boldsymbol{uptake}, \boldsymbol{conc}) = \\ = -N \log \sqrt{2\pi} - N \log \sigma - \frac{1}{2} \sum_{i=1}^{N} \frac{(uptake_i - \beta_0 - \beta_1 \cdot conc_i)^2}{\sigma^2} \quad (20)$$

where vectors ***uptake*** and ***conc*** contain the data values for all the observations. This is a fully specified log-likelihood function that we can use for optimization. Let us repeat what we have done:

- We wrote log-likelihood as a function of parameters β_0, β_1 and σ. Note that in case of linear regression we typically do not call σ a parameter. But it is still a parameter, although one we usually do not care much about (sometimes called "nuisance parameter").
- The likelihood function also depends on data, here the vectors ***uptake*** and ***conc***.
- The function definition itself is just sum of log-likelihood contributions of individual normal disturbance terms, but as we do not observe the disturbance terms, we express those through the regression equation in (19).

Finally, we combine the three parameters into a single vector $\boldsymbol{\theta}$, suppress dependency on data in the notation, and write

$$\ell(\boldsymbol{\theta}) = -N \log \sqrt{2\pi} - N \log \sigma - \frac{1}{2} \sum_{i=1}^{N} \frac{(uptake_i - \beta_0 - \beta_1 \cdot conc_i)^2}{\sigma^2}. \quad (21)$$

This is the definition we can easily code and estimate. We guess start values $\beta_0 = 30$ (close to the mean), $\beta_1 = 0$ (uptake does not depend on concentration) and $\sigma = 10$ (close to sample standard deviation). We can convert (21) into code almost verbatim, below we choose to compute the expected uptake μ as an auxiliary variable:

```
> loglik <- function(theta) {
+    beta0 <- theta[1]
+    beta1 <- theta[2]
+    sigma <- theta[3]
+    N <- nrow(CO2)
+    ## compute new mu based on beta1, beta2
+    mu <- beta0 + beta1*CO2$conc
+    ## use this mu in a similar fashion as previously
+    -N*log(sqrt(2*pi)) - N*log(sigma) -
+       0.5*sum((CO2$uptake - mu)^2/sigma^2)
+ }
> m <- maxLik(loglik, start=c(beta0=30, beta1=0, sigma=10))
> summary(m)

--------------------------------------------
Maximum Likelihood estimation
```

```
Newton-Raphson maximisation, 6 iterations
Return code 8: successive function values within relative tolerance limit (reltol)
Log-Likelihood: -307.409
3  free parameters
Estimates:
       Estimate Std. error t value  Pr(> t)
beta0 19.500306   1.978074   9.858  < 2e-16 ***
beta1  0.017731   0.003681   4.817 1.46e-06 ***
sigma  9.399847   0.744019  12.634  < 2e-16 ***
---
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
--------------------------------------------
```

These are the linear regression estimates: $\beta_0 = 19.5$ and $\beta_1 = 0.018$. Note that `maxLik` output also provides standard errors, z-values and p-values, hence we see that the results are highly statistically significant.

One can check that a linear regression model will give similar results:

```
> summary(lm(uptake ~ conc, data=CO2))

Call:
lm(formula = uptake ~ conc, data = CO2)

Residuals:
    Min      1Q  Median      3Q     Max
-22.831  -7.729   1.483   7.748  16.394

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 19.500290   1.853080  10.523  < 2e-16 ***
conc         0.017731   0.003529   5.024 2.91e-06 ***
---
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 9.514 on 82 degrees of freedom
Multiple R-squared:  0.2354,        Adjusted R-squared:  0.2261
F-statistic: 25.25 on 1 and 82 DF,  p-value: 2.906e-06
```

Indeed, the results are close although not identical.

6 Non-linear optimization

Finally, we discuss the magic inside `maxLik` that finds the optimal parameter values. Although not necessary in everyday work, this knowledge helps to understand the issues and potential solutions when doing non-linear optimization. So how does the optimization work?

Consider the example in Section 4 where we computed the normal distribution parameters for CO_2 intake. There are two parameters, μ and σ, and

`maxLik` returns the combination that gives the largest possible log-likelihood value. We can visualize the task by plotting the log-likelihood value for different combinations of μ, σ (Figure 2).

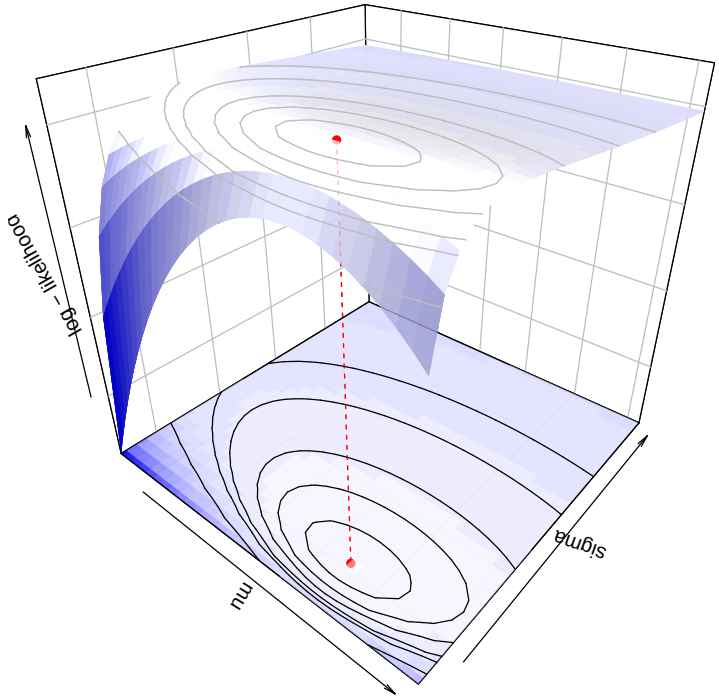


Figure 2: Log-likelihood surface as a function of μ and σ. The optimum, denoted as the red dot, is at $\mu = 27.213$ and $\sigma = 10.75$. The corresponding countour plot is shown at the bottom of the figure box.

So how does the algorithm find the optimal parameter value $\boldsymbol{\theta}^*$, the red dot on the figure? All the common methods are iterative, i.e. they start with a given start value (that's why we need the start value), and repeatedly find a new and better parameter that gives a larger log-likelihood value. While humans can look at the figure and immediately see where is its maximum, computers cannot perceive the image in this way. And more importantly–even humans cannot visualize the function in more than three dimensions. This visualization is so helpful for us because we can intuitively understand the 3-dimensional structure of the surface. It is 3-D because we have two parameters, μ and σ, and a single log-likelihood value. Add one more parameter as we did in Section 5, and visualization options are very limited. In case of 5 parameters, it is essentially impossible to solve the problem by just visualizations.

Non-linear optimization is like climbing uphill in whiteout conditions where you cannot distinguish any details around you–sky is just a white fog and the ground is covered with similar white snow. But you can still feel which way the ground goes up and so you can still go uphill. This is what the popular algorithms do. They rely on the slope of the function, the gradient, and follow the direction suggested by gradient. Most optimizers included in the `maxLik` pack-

age need gradients, including the default Newton-Raphson method. But how do we know the gradient if the log-likelihood function only returns a single value? There are two ways: i) provide a separate function that computes gradient; ii) compute the log-likelihood value in multiple points nearby and deduce the gradient from that information. The first option is superior, in high dimensions it is much faster and much less error prone. But computing and coding gradient can easily be days of work. The second approach, numeric gradient, forces the computer to do more work and hence it is slower. Unfortunately importantly, it may also unreliable for more complex cases. In practice you may notice how the algorithm refuses to converge for thousands of iterations. But numeric gradient works very well in simple cases we demonstrated here.

This also hints why it is useful to choose good start values. The closer we start to our final destination, the less work the computer has to do. And while we may not care too much about a few seconds of computer's work, we also help the algorithm to find the correct maximum. The less the algorithm has to work, the less likely it is that it gets stuck in a wrong place or just keeps wandering around in a clueless manner. If this is the case, you may see how the algorithm gets slow, does not converge (returns the "maximum number of iterations exceeded" message), how the results look weird, or standard errors are extremely large.