R Programming for Demographers

Optimization

Tim Riffe*

September 19, 2017

Contents

1 Introduction **1**

2 MLE of a Binomial distribution **1**
2.1 Some analytical results . . . 2
2.2 Graphical view . . . 2
2.3 Using an R optimizer . . . 5

3 A demographic example: the Gompertz distribution **10**
3.1 Optimizing in R . . . 11
3.2 Confidence Interval for the parameters . . . 12
3.3 Graphical view . . . 13

4 Exercise **16**

1 Introduction

Often demographers want to check whether a certain distribution (or model) is able to capture essential patterns in a given set of data. Assumptions and possible estimation are instruments that we need to handle with familiarity. Today's lecture deals with these concepts.

R is very good at optimizing many functions automatically. But what happens if you have a function for which no maximization is built-in and for which an analytical maximization is too tedious or even impossible? For this case, R provides the possibility to optimize your own function numerically. We start by demonstrating it for a simple case: the Maximum Likelihood Estimation (MLE) of a Binomial distribution. Then we move to a typical demographic example: estimating the Gompertz model.

2 MLE of a Binomial distribution

We have chosen this distribution since it is simple, and because you will derive the analytical outcomes during your basic statistics course, thus it is easy to compare results from R with the analytically correct ones.

*First, a big thanks to Giancarlo Camarda, who has let me recycle his impeccable material for teaching this course. Higher order thanks are also due to the very same people that Giancarlo himself thanked, especially Sabine Zinn and Roland Rau. Sabine was my teacher in this very course in 2009 (cohort 5)! Part of these lectures has been freely inspired by courses that both Roland, Sabine, and Giancarlo taught in the past. Furthermore, in recent years, Adam Lenart, Fernando Colchero, and Silvia Rizzi have aided in teaching this course and improving handouts. On the other hand, it goes without saying that I am uniquely responsible for any deficiencies and mistakes you may find in handouts.

2.1 Some analytical results

Let's assume that X has a Binomial distribution, $X \sim Bin(n, p)$:

$$B(n,p) = B_{n,p}\left(X = x\right) = \binom{n}{x} p^x \left(1-p\right)^{n-x} .$$

The associated likelihood function can be written as:

$$L(p|x) = \binom{n}{x} p^x (1-p)^{n-x}$$

To maximize this function it is more convenient to consider the log-likelihood:

$$\mathcal{L}(p|x) \;=\; ln\left[L(p|x)\right] \;\propto\; x\,\ln(p) + (n-x)\,\ln(1-p)$$

In a nutshell, given a certain value of x (and n), the aim is to find the parameter p which maximizes $\mathcal{L}$.

An advantage of logging is that multiplication is transformed to summation on the log-scale for which derivatives are easier. Moreover, in general, multiplication is computationally more expensive than addition, and logging improves numerical stability.

Analytically we can derive the function with respect to p:

$$\frac{d}{d\,p}\,\mathcal{L}(p|x) = \frac{x}{p} - \frac{n-x}{1-p}$$

set it equal to zero and find out that the fitted parameter $\hat{p}$ is:

$$\hat{p} = \frac{x}{n}$$

As illustrative example, our data shows that 13 out of 50 individuals are smokers. Assuming a binomial distribution, what is the most plausible probability of being smokers in the population?

$$\hat{p} = \frac{13}{50} = 0.26$$

2.2 Graphical view

We can use the likelihood function to look at the plausibility of different values of p to see which one is the most likely. In `R` , firstly we build up the function we would like to plot (and later on maximize). This function depends on three arguments p, x and n:

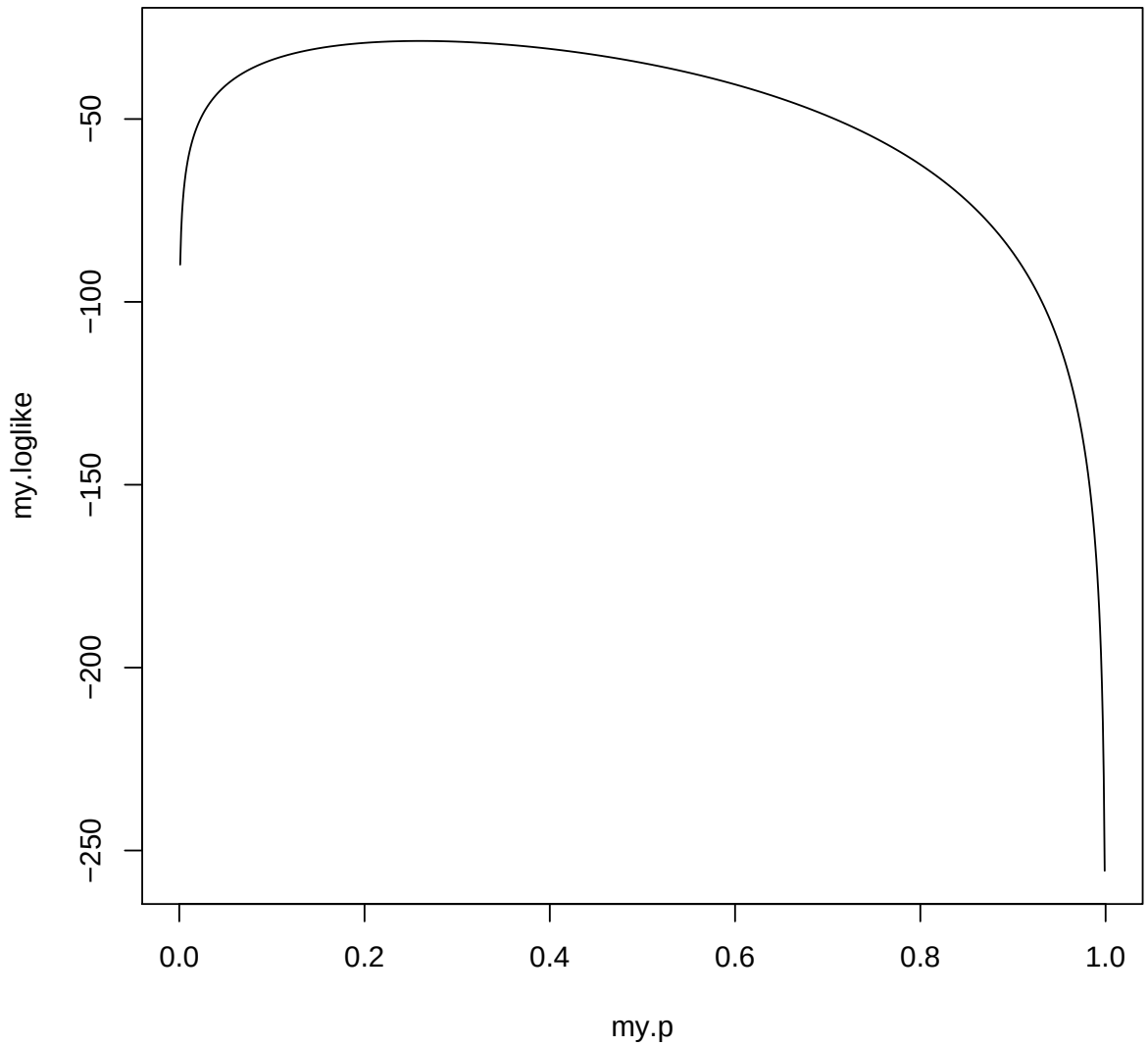
```
loglike.bin <- function(p, x, n){
  loglike <- x * log(p) + (n-x) * log(1 - p)
  return(loglike)
}
```

In order to have a graphical view of $\mathcal{L}$ we fix in the function $x = 13$ and $n = 50$ and we compute `loglike.bin()` for different values of p which are defined between 0 and 1:

```
my.p <- seq(0, 1, length=1000)
my.loglike <- loglike.bin(my.p, x=13, n=50)
```

and then we plot the plausible p against their "plausibility":

```
plot(my.p, my.loglike, t="l")
```



At a glance, Figure 1 shows that a value about of $p = 0.26$ maximizes the log-likelihood. We can see this result as follows:

```
my.p[my.loglike==max(my.loglike)] # or my.p[which.max(my.loglike)]

## [1] 0.2602603
```

by picking the value of `my.p` at which the vector `my.loglike` maximizes.

In this case we obtain values of the likelihood function that correspond to each `my.p` in a single vector. This is possible only because the further arguments (`x` and `n`) are simple scalars. Moreover the precision of $\hat{p}$ in this graphical approach heavily depends upon the fineness of the grid in `my.p`, i.e. the finer the grid in `my.p`, the more precise the estimation

A different perspective on the log-likelihood profile could be given by plotting its first derivative with respect to the parameter p. This function should cross 0 at the maximum likelihood estimator $\hat{p} = 0.26$. We have already presented the derivative of the log-likelihood, which in `R` could be:

```
loglike1.bin <- function(p, x, n){
  loglike1 <- x / p - (n - x) / (1 - p)
  return(loglike1)
}
```

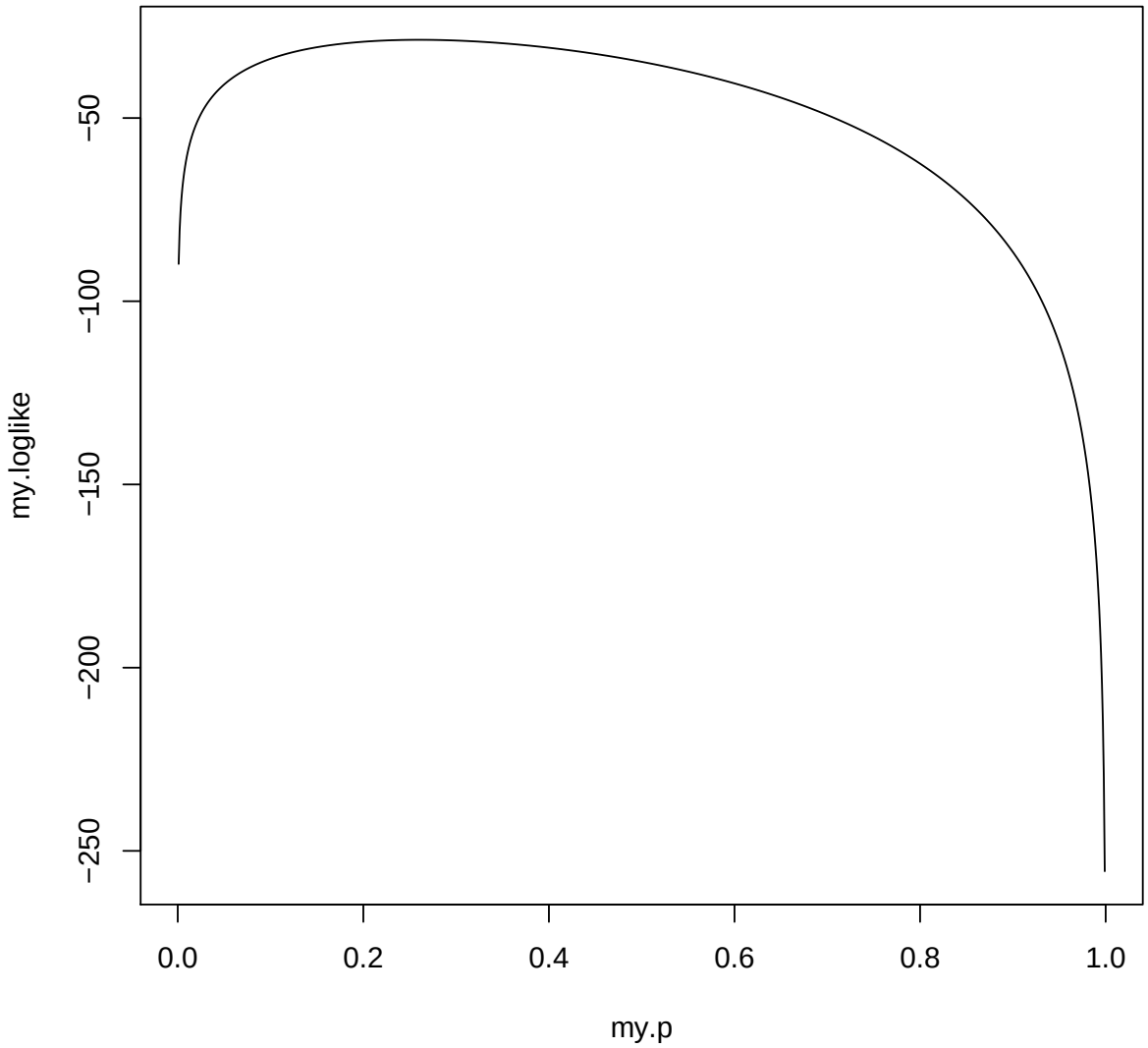


Figure 1: Log-likelihood function of a Binomial distribution with $x = 13$ and $n = 50$

We now evaluate this function for all the elements in `my.p`

```
my.loglike1 <- loglike1.bin(my.p, x = 13, n = 50)
```

and we plot the profile of derivative of the likelihood function as follows: We restricted the limits of the axes (Figure 2) for better focusing about the MLE $\hat{p}$.

2.3 Using an R optimizer

So far we have dealt with a simple distribution with a known analytical solution, and with a single parameter which can be plotted against the log-likelihood function itself. What about a "harder" distribution?

R offers the possibility to maximize any function numerically and as you can imagine, the harder the function, the more tricks you may need to optimize it. So let's start with the known simple binomial distribution.

Though several commands are available for performing optimization, we will present `optim()`, probably the most flexible one. In its simplest form you just have to "feed" a starting value for the parameter (or a set of parameters) and the additional arguments your objective function may need.

Let's assume as a starting value the middle of all possible p: 0.5. Obviously a wiser choice would be needed for more complex optimization problems. Then we simply use this command:

```
optim(par=0.5, fn=loglike.bin, x=13, n=50)

## $par
## [1] 1
##
## $value
## [1] -1359.262
##
## $counts
## function gradient
##      209       NA
##
## $convergence
## [1] 10
##
## $message
## NULL
```

You see in the first entry of the output (`$par`, which stands for the optimal/estimated parameter), that R did not find the correct result of $\hat{p} = 0.26$. Actually the optimal value is equal to one. This gives you a first indication, that numerical optimization is sometimes (very often?) a tricky business. If you read the help page of `optim()`, you might quickly recognize one possible problem. In the section "Details" it mentions:

```
"By default this function performs minimization ..."
```

The problem with the log-likelihood function above was therefore that R was "walking" into 1, where it actually has its minimum (see Fig. 1).

We could get the right behavior by using some other optional arguments to `optim()`, or, more intuitively, modify `loglike.bin()`: maximizing a function is equivalent to minimizing the negative of the function itself. Therefore we add a - sign in front of the final return object:

```
loglike.bin <- function(p, x, n){
    loglike <- - (x * log(p) + (n-x) * log(1 - p))
    return(loglike)
}
```

Now we try again with `optim()`:

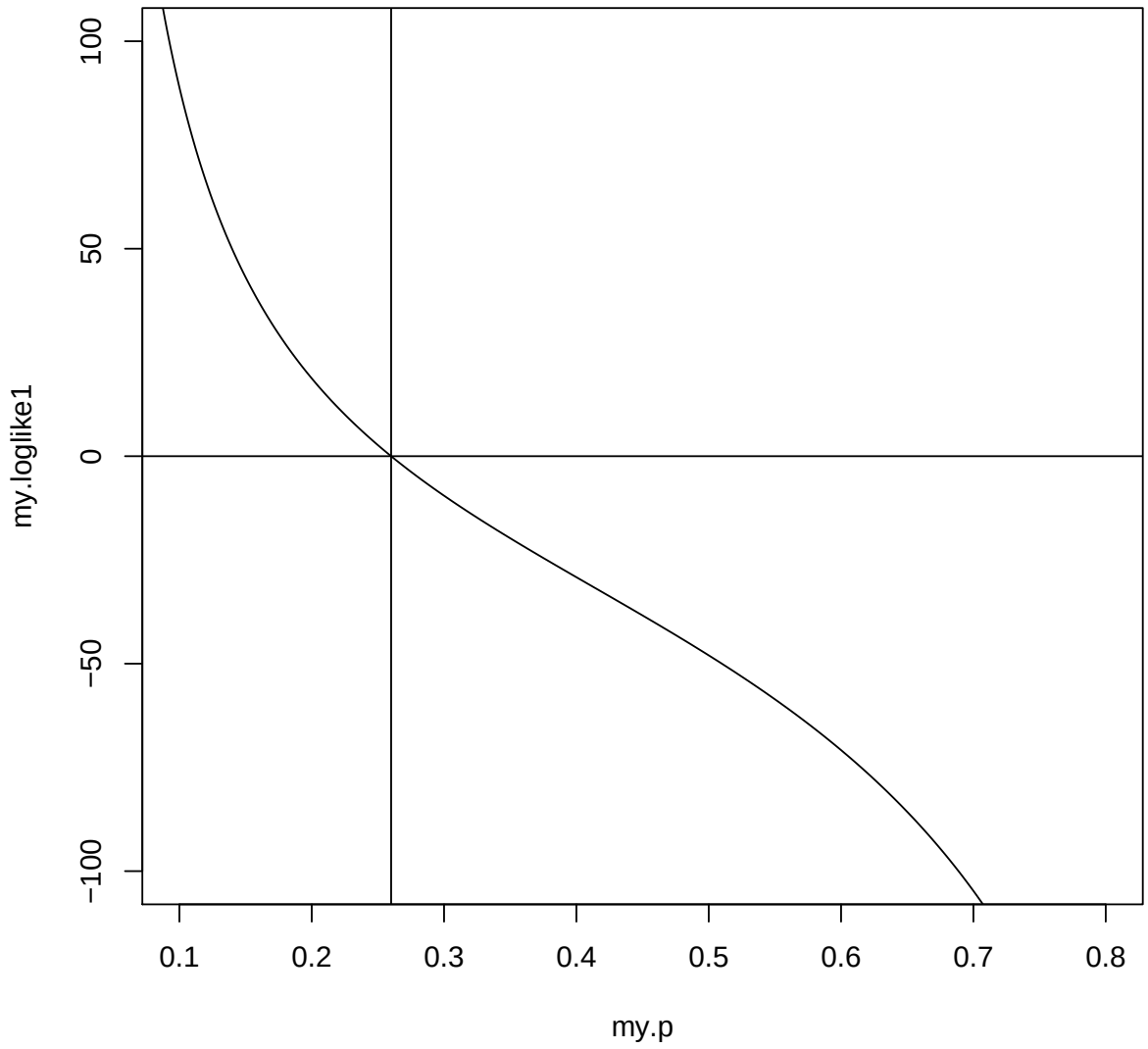


Figure 2: First derivative of the log-likelihood function with respect to the parameter p of a Binomial distribution with $x = 13$ and $n = 50$

```
optim(par=0.5, fn=loglike.bin, x=13, n=50)

## Warning in optim(par = 0.5, fn = loglike.bin, x = 13, n = 50):  one-dimensional optimization
by Nelder-Mead is unreliable:
## use "Brent" or optimize() directly

## $par
## [1] 0.2599609
##
## $value
## [1] 28.65285
##
## $counts
## function gradient
##       26       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

So, R is successfully finding the maximum. Nevertheless we should acknowledge the fact that R warns us every time in the last line of the output that the default setting of `optim()` has problems with optimizing. Such warning can depend on the R version you are running on your own machine. For this case it can appear something like:

```
Warning: one-dimensional optimization by Nelder-Mead is unreliable:
use "Brent" or optimize() directly
```

Instead of changing the optimizer (`optimize()`), we follow the other approach. The warning informs that the default `Nelder-Mead` method is prone to problems if the optimization runs over one dimension (one parameter) and it would be better to proceed with the method `Brent`. Additionally, from the documentation we can see five different optimization methods available:

- Nelson-Mead
- BFGS
- CG
- L-BFGS-B (a generalization of BFGS for bounding the optima)
- SANN
- Brent (internally it uses `optimize()` and the user needs to provide lower/upper bounds)

If you want to learn more about the internal algorithms for finding optima, references are provided in the help page. Here we will just show how simple is to change the `method` within `optim()`. Moreover an important feature of R is that we can assign any output we have obtained from previous procedure to an object. The same can be done for the output of an optimization procedure.

In the following we run all possible methods and we will assign the results to different objects:

```
optBFGS  <- optim(0.5, loglike.bin, x = 13, n = 50, method = "BFGS")
optCG    <- optim(0.5, loglike.bin, x = 13, n = 50, method = "CG")
optSANN  <- optim(0.5, loglike.bin, x = 13, n = 50, method = "SANN")
optBrent <- optim(0.5, loglike.bin, x = 13, n = 50, method = "Brent",
                  lower = 0, upper = 1)
```

Despite other warning messages due to computational problems skipped in this material[1], we can check that all optimization methods return the same results by extracting the values of $\hat{p}$ which is the estimated parameter (i.e. where, given our data, `loglike.bin()` is minimum):

[1] In the searching procedure `optim()` is taking $p = 0$ and/or $p = 1$, `loglike.bin()` is thus sometimes returning infinite outcomes. This issue can be overcome with some tricks that go beyond the objectives of this course

```
c(optBFGS$par,optCG$par,optSANN$par,optBrent$par)

## [1] 0.2600007 0.2600003 0.2600062 0.2600000
```

As with many R -object coming from implemented functions, the output of `optim()` has different values. You can get their names with:

```
names(optBFGS)

## [1] "par"         "value"       "counts"      "convergence" "message"
```

For instance we can check whether our optimization reached its goal by typing:

```
optBFGS$convergence

## [1] 0
```

From the documentation we can read that `0 indicates successful completion`, so in this case we should not worry. Additionally the number of iterations executed by the internal algorithm is extracted as follows:

```
optBFGS$counts[1]

## function
##       41
```

Sometimes it is important to know the value of the objective function evaluated in the fitted parameters. In other words $-\mathcal{L}(\hat{p}|x)$:

```
optBFGS$value

## [1] 28.65285
```

Of course, this value can be computed just inserting the `optBFGS$par` and our data in `loglike.bin()`:

```
loglike.bin(p=optBFGS$par, x=13, n=50)

## [1] 28.65285
```

Finally we can graphically present the outcomes of our MLE for the binomial distribution: we again plot the log-likelihood profile from `my.loglike` and then add a vertical line at $\hat{p}$ from `optBFGS` (see Fig. 3):

Additional to the objective function, the R -command `optim()` accepts the first derivative of the objective function by the argument `gr` which stands for "gradient". By default finite-difference approximations are computed, alternatively we can provide the analytic solution of the gradient by a function. This would speed up and enhance the search of the optimum and it is possible only for the `"BFGS"`, `"CG"` and `"L-BFGS-B"` methods.

For the binomial example, we have already programmed the first derivatives of the log-likelihood function with respect to p. Nevertheless we are minimizing minus the log-likelihood, therefore we need to change the sign of the gradient too:

```
loglike1.binN <- function(p, x, n){
  loglike1 <- -loglike1.bin(p, x, n)
  return(loglike1)
}
```

We now add this information in the optimization process:

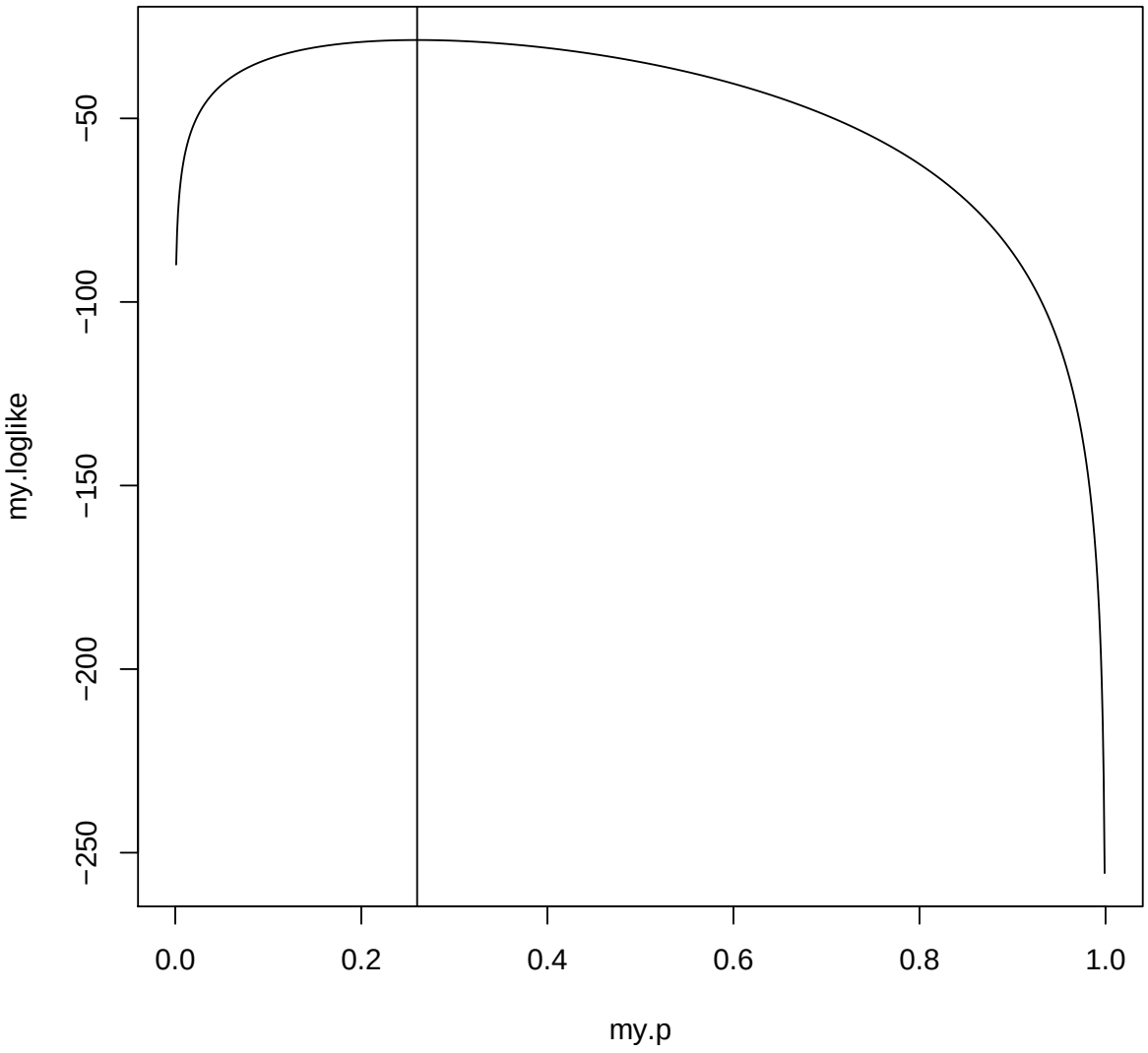


Figure 3: Log-likelihood function of a Binomial distribution with $x = 13$ and $n = 50$. The vertical line show the MLE of p.

```
optBFGSgr <- optim(0.5, loglike.bin, x = 13, n = 50,
                   method = "BFGS",
                   gr = loglike1.binN)

## Warning in log(p):  NaNs produced
## Warning in log(p):  NaNs produced
## Warning in log(p):  NaNs produced
## Warning in log(p):  NaNs produced
## Warning in log(p):  NaNs produced
## Warning in log(p):  NaNs produced

optBFGSgr$par

## [1] 0.26
```

Apparently nothing has changed, and the estimated parameters are equal to the previous ones. Nevertheless internally the function has found the minima much more easily. Let's check the number of iterations:

```
optBFGSgr$counts[1]

## function
##       36
```

By specifying the gradient function, we reduced the number of iterations from 41 to 36.

For the estimation of a MLE of a Binomial distribution we cannot appreciate the advantage of adding the gradient in `optim()`, the importance of this feature rises along with the complexity of the optimization problem at hand.

3 A demographic example: the Gompertz distribution

As already metioned, a well-known distribution in demography is the Gompertz distribution. In particular Gompertz (1825) showed that the hazard (or in demography, force of mortality) of a particular individual i, $h(x_i)$, often follows an exponential function such as:

$$h(x_i) = \alpha e^{\beta x_i}$$

So, given an actual distribution of independent deaths by age from a population $(x_i, i = 1, \ldots, n)$, and assuming that this population follows a Gompertz distribution, our aim is to estimate the (most likely) parameters α and β, which would have produced the actual data.

Following the same path we used for the Binomial distribution, we look for plausible pairs $[\alpha, \beta]$ to see which one is the most likely to have generated the actual data. While for the Binomial case the actual data were six heads out of ten tosses, for the Gompertz distribution the data in hands are the deaths by age x_i.

Also in this case, a suitable tool for fitting a distribution is Maximum Likelihood Estimation. The difference is that for Gompertz we have two parameters and, consequently, it is slightly harder to get an analytical maximum.

Nevertheless we can derive the formula for the likelihood function as product of the density functions since we assume that deaths occur independently (for further details see Klein and Moeschberger (1997)). Moreover we aim to maximize it:

$$\begin{aligned} L(\boldsymbol{\theta}|x_i, i = 1, \ldots, n) &= \prod_{i=1}^{n} f(\boldsymbol{\theta}|x_i) = \\ &= \prod_{i=1}^{n} [h(\boldsymbol{\theta}|x_i) \cdot S(\boldsymbol{\theta}|x_i)] = \\ &= \prod_{i=1}^{n} \left[\alpha e^{\beta x_i} \cdot e^{\frac{\alpha}{\beta}(1-e^{\beta x_i})}\right] \end{aligned} \tag{1}$$

where $\boldsymbol{\theta}$ is the vector (α, β).

Remember always that it is easier to work with the log-likelihood such that products are transformed in sums. In formula:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}|x_i, i = 1, \ldots, n) &= \sum_{i=1}^{n} \ln[f(\boldsymbol{\theta}|x_i)] = \\
&= \sum_{i=1}^{n} \{\ln[h(\boldsymbol{\theta}|x_i)] + \ln[S(\boldsymbol{\theta}|x_i)]\} = \\
&= \sum_{i=1}^{n} \left\{ \ln[\alpha e^{\beta x_i}] + \ln[e^{\frac{\alpha}{\beta}(1-e^{\beta x_i})}] \right\} = \\
&= \sum_{i=1}^{n} \left\{ \ln(\alpha) + \beta x_i + \frac{\alpha}{\beta}(1 - e^{\beta x_i}) \right\} \qquad (2)
\end{aligned}$$

3.1 Optimizing in R

We can code in `R` the equation (2) remembering that `optim` searches for the minimum. Then our objective function is -log-likelihood:

```
loglike.gom <- function(theta, x){
    alpha    <- theta[1]
    beta     <- theta[2]
    loglk    <- log(alpha) + beta*x + alpha/beta*(1 - exp(beta*x))
    sumloglk <- -sum(loglk)
    return(sumloglk)
}
```

This function has two arguments, the vector of unknown parameters $\boldsymbol{\theta} = (\alpha, \beta)$ and the actual data x_i.

Note that `optim` can optimize over a **single** argument, which could be a scalar or a vector: it is our task to specify, inside the objective/log-likelihood function, how this argument should work.

We use for this example the data we generate in the previous module: `lt`. An instance of this simulated data are saved in the file `ltGomp.txt`:

```
lt <- read.table("ltGomp.txt")
```

Since we have simulated them, we know the true parameter ($\alpha = 0.00003$ and $\beta = 0.1$):

```
true.a <- 0.00003
true.b <- 0.1
```

Nevertheless let's assume that we want to estimate them. In other words, our aim is to find, given those data, the most plausible parameters of the Gompertz distribution, i.e. given `lt`, we need to minimize `loglike.gom`.

In order to find $\hat{\alpha}$ and $\hat{\beta}$, we can simply use `optim`, give some starting values, and assign the outcomes to an object:

```
opt.gomp <- optim(par = c(0.001, 0.05), loglike.gom, x = lt)

## Warning in log(alpha):  NaNs produced
## Warning in log(alpha):  NaNs produced
## Warning in log(alpha):  NaNs produced
## Warning in log(alpha):  NaNs produced
```

This time we may have some warning messages regarding the logarithm, but the optimization procedure seems OK. To be sure let's have a look at `opt.gomp`

```
opt.gomp

## $par
## [1] 3.351282e-05 9.867918e-02
##
## $value
## [1] 7777.426
##
## $counts
## function gradient
##      137       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

In particular we are interested in the values of $\hat{\alpha}$ and $\hat{\beta}$. Let's extract them:

```
opt.gomp$par

## [1] 3.351282e-05 9.867918e-02
```

They are fairly similar to the true parameters we have used in the simulation:

```
true.a

## [1] 3e-05

true.b

## [1] 0.1
```

3.2 Confidence Interval for the parameters

One way of checking the uncertainty of the estimated parameters is to construct their confidence internals.

Briefly, we can compute the standard errors of the estimated parameters by taking the square root of the variance-covariance matrix of your maximum likelihood estimator, which can be computed by inverting the Hessian matrix evaluated at the estimated parameters. In `R` we can extract the Hessian matrix using the routine `fdHess()` from the package `nlme`:

```
library(nlme)
H <- fdHess(pars = opt.gomp$par, loglike.gom, x = lt)$Hessian
```

Then we invert this matrix (more about this operation in a later module) with the function `solve()`

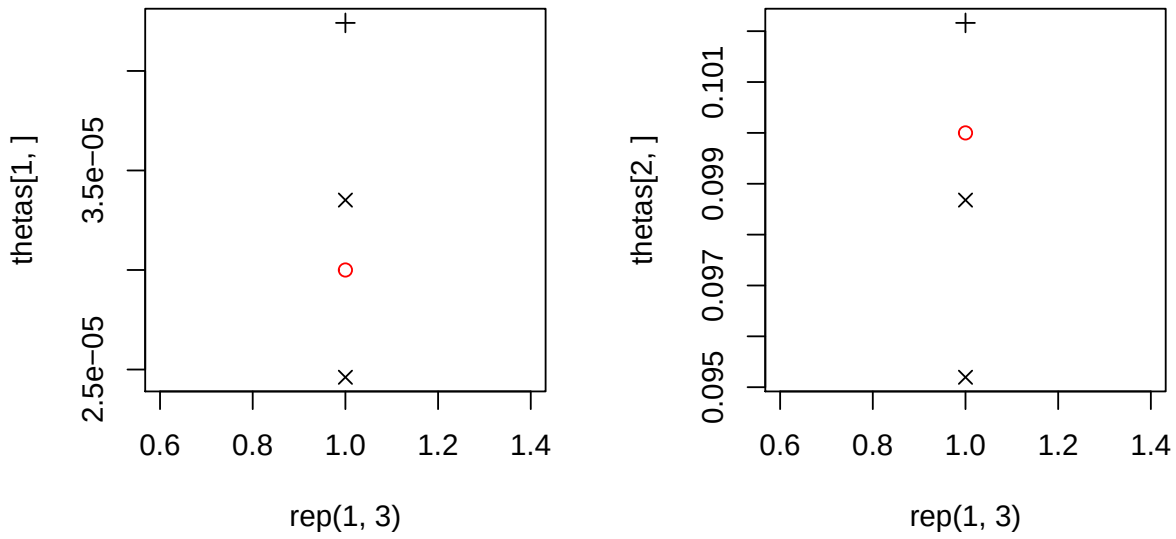
```
vcov <- solve(H)
```

Finally, we compute the standard errors for our $\hat{\boldsymbol{\theta}} = [\hat{\alpha}, \hat{\beta}]$ as follows:

```
se.theta <- sqrt(diag(vcov))
```

The 95% confidence intervals can be computed by adding and substracting 2 times the standard errors from the estimated parameters. To keep everything in a single object, we write:

```
par(mfrow=c(1,2))
plot(rep(1,3), thetas[1, ], pch = c(3, 4, 4))
points(1, true.a, col=2)
plot(rep(1,3), thetas[2, ], pch = c(3, 4, 4))
points(1, true.b, col=2)
```




```
par(mfrow=c(1,1))
```

Figure 4: True and estimated parameters from a Gompertz distribution along with the estimated 95% confidence intervals. Simulated life-times.

```
thetas <- cbind(opt.gomp$par + 2 * se.theta,
                opt.gomp$par,
                opt.gomp$par - 2 * se.theta)
```

A plot of true parameters along with the estimated ones and their 95% confidence intervals is useful to check the model fit. Results are shown in Figure 4.

3.3 Graphical view

Unlike for the log-likelihood function of the Binomial distribution which was quite simple to plot against different values of π, the Gompertz distribution raises additional issues. We are in a two-dimensional setting, hence the log-likelihood is no longer a line depending on a single parameter, but a surface depending on two parameters.

In other words instead of a 2-D graph (e.g. π vs. $\mathcal{L}(\pi|x)$), we need to plot a 3-D surface where the three axes are:

- different values for the first parameter, α
- different values for the second parameter, β
- log-likelihood evaluated for each combination of the two parameters, $\mathcal{L}(\pi|\alpha, \beta)$.

This can be easily coded in R for the Gompertz case. Like for the π in the Binomial case, we first have to decide which plausible values for α and β we want:

```
alphas <- seq(0.00002, 0.00005, length = 100)
betas  <- seq(0.05, 0.13, length = 100)
```

then, remembering that we are in a 3-D setting, we create a matrix which should be filled with the values of the `loglike.gomp` evaluated for each couple of `alphas` and `betas`. This can be done with a double `for`-loop, for instance:

```
my.like <- matrix(0, nrow = length(alphas), ncol = length(betas))
for(i in 1:length(alphas)){
  for(j in 1:length(betas)){
    my.like[i, j] <- loglike.gom(c(alphas[i], betas[j]), x = lt)
  }
}
```

Once we have the grid of possible -(log-likelihood) values, we plot them using the function `image`[2]. We need to change the breaks in order to get a better idea of the minimum:

Figure 5 clearly shows the minimum, and which α and β generated it. These are based on what we fit using `optim()`:

```
opt.gomp$par

## [1] 3.351282e-05 9.867918e-02
```

[2]For a better explanation of this kind of plot, type `?image`.

```
my.breaks <- quantile(my.like,
                      prob = c(0, 0.005, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1))
image(x = alphas,
      y = betas,
      z = my.like,
      breaks = my.breaks,
      col = gray(seq(0, .9, length = 9)))
abline(v = opt.gomp$par[1], col = "red", lty = 2)
abline(h = opt.gomp$par[2], col = "red", lty = 2)
points(opt.gomp$par[1], opt.gomp$par[2], pch = 16, col = "red")
```

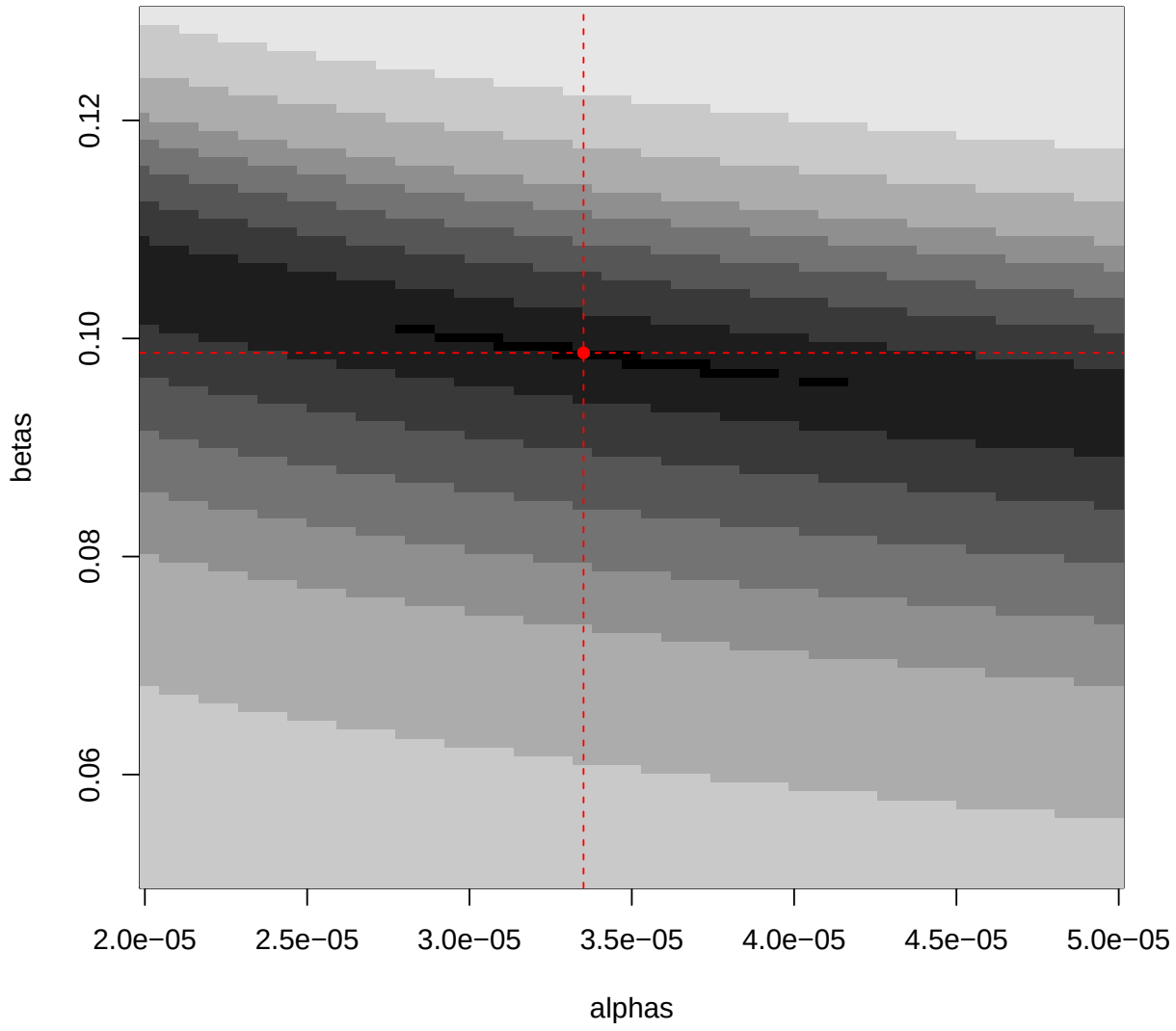


Figure 5: A contour plot of a -(log-likelihood) of a Gompertz.

4 Exercise

The species *bunny bunny bavariae* (the "Wolpertinger", or the North-American "Jackalope") is a rarely observed animal. Usually it is active only during the nights in remote, rural, areas of Bavaria (the southern part of Germany). In Figure 6 you can see a picture of it. Due to being very shy, not much is known about the animal. We were able, however, to estimate the age at death of some Wolpertingers which have been hit by cars. We assume that the lifetimes of the Wolpertinger follow an exponential distribution with rate parameter λ.

Figure 6: A Magnificent Specimen of a Wolpertinger (*bunny bunny bavariae*) - before the accident

In formulas we can define the probability density function of an Exponential distribution as follows:

$$f(x;\lambda) = \begin{cases} \lambda \exp(-\lambda x) & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

We can thus derive the cumulative distribution function:

$$F(x;\lambda) = \begin{cases} 1 - \exp(-\lambda x) & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

with $\lambda > 0$, rate parameter of the distribution.

The data set `wolp.txt` consists of observed values of Wolpertingers which have been hit by cars. This data set has two variables:

`id`: an identification numbers for each hit Wolpertinger

`ageatdeath`: the estimated age (in years) of the Wolpertinger

For all cases, we assume that the death of one Wolpertinger occurred independently from any other death.

The task is to estimate the rate parameter λ of the lifetime distribution of the Wolpertingers in Bavaria using Maximum Likelihood Estimation. As we have done for the Binomial and for the Gompertz distributions, we can derive the the likelihood-function for n individuals using the probability density function of the lifetime distribution of the Wolpertingers:

$$L(\lambda \mid x) = \prod_{i=1}^{n} \lambda \exp(-\lambda x_i),$$

where x_i is the lifetime of the ith Wolpertinger.

Of course we need to remember that the final likelihood function is the joint probability function of all data points.

In other words you are asked to:

(i) derive the log-likelihood-function $\mathcal{L}(\lambda \mid x) = ln[L(\lambda \mid x)]$.

(ii) Translate this log-likelihood function $\mathcal{L}(\lambda \mid x)$ into a R-function: `loglike.exp(lambda, x)`.

(iii) Minimize the -(log-likelihood) `loglike.exp(lambda,x)` for obtaining the unknown rate parameter `lambda`. Use for this task the R-function `optim()`. Hint: in this case, the syntax of `optim()` would be

```
optim ( par = 1/12,
        function = loglike.exp(lambda,x),
        x = ageatdeath
       )
```

At this `par` $= \frac{1}{12}$ is arbitrarily chosen as the starting value for numerical optimization. Pay attention to the fact that the negative log-likelihood function must be used as input argument.

(iv) Plot the values of function `loglike.exp(lambda,x)` for $\lambda = [0.0001, 0.2]$ and x_i. Label the axis of the plot and give a title to the graphic.

(v) Are you able to numerically find $\hat{\lambda}$ without using the function `optim`? Hint: see next task.

(vi) Are you able to compute $\hat{\lambda}$ without grid search and `optim`?

(vii) Are you able to find $\hat{\lambda}$ without any numerical approximation?

Do feel free to work together on this exercise, as it requires some thinking on the math/stats side.

References

Gompertz, B. (1825). On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. *Philosophical transactions of the Royal Society of London 115*, 513–583.

Klein, J. and M. Moeschberger (1997). Survival analysis: Techniques for censored and truncated regression.