Numerical optimization of likelihoods: Additional literature for STK2120

Geir Storvik, University of Oslo

February 21, 2011

1 Introduction

As described in section 7.2 in the textbook by Devore and Berk (2007), maximum likelihood is an important estimation method for statistical analysis. The textbook also gives several examples for which analytical expressions of the maximum likelihood estimators are available. In this note we will be concerned with examples of models where *numerical* methods are needed in order to obtain the estimates. Some of these models are "standard" in the sense that statistical software is available for direct computation. In other cases, user-made programs must be applied. In both cases the use of a computer is essential, and knowledge on using and/or developing software becomes vital. Throughout the note we will illustrate the different optimization methods through three examples. R code used in connection with these examples are given either directly in the text or in the appendix.

Example 1 (Muon decay)
The angle θ at which electrons are emitted in muon decay has a distribution with the density

$$f(x|\alpha) = \frac{1+\alpha x}{2}, \quad -1 \leq x \leq 1 \quad \text{and} \quad -1 \leq \alpha \leq 1$$

where $x = \cos\theta$ (Rice 1995, Example D, section 8.4). The following data are simulated from the given distribution:

```
 0.41040018  0.91061564 -0.61106896  0.39736684  0.37997637  0.34565436
 0.01906680 -0.28765977 -0.33169289  0.99989810 -0.35203164  0.10360470
 0.30573300  0.75283842 -0.33736278 -0.91455101 -0.76222116  0.27150040
-0.01257456  0.68492778 -0.72343908  0.45530570  0.86249107  0.52578673
 0.14145264  0.76645754 -0.65536275  0.12497668  0.74971197  0.53839119
```

We want to estimate α based on these data. The likelihood function is given by

$$L(\alpha) = \prod_{i=1}^{n} \frac{1+\alpha x_i}{2}$$

while the log-likelihood is given by

$$l(\alpha) = \sum_{i=1}^{n} \log(1+\alpha x_i) - n\log(2)$$

l is a smooth function in α, so at the max-point, the derivative should be equal to zero (if not, the max-point is at one of the end-points). Now

$$l'(\alpha) = \sum_{i=1}^{n} \frac{x_i}{1+\alpha x_i}.$$

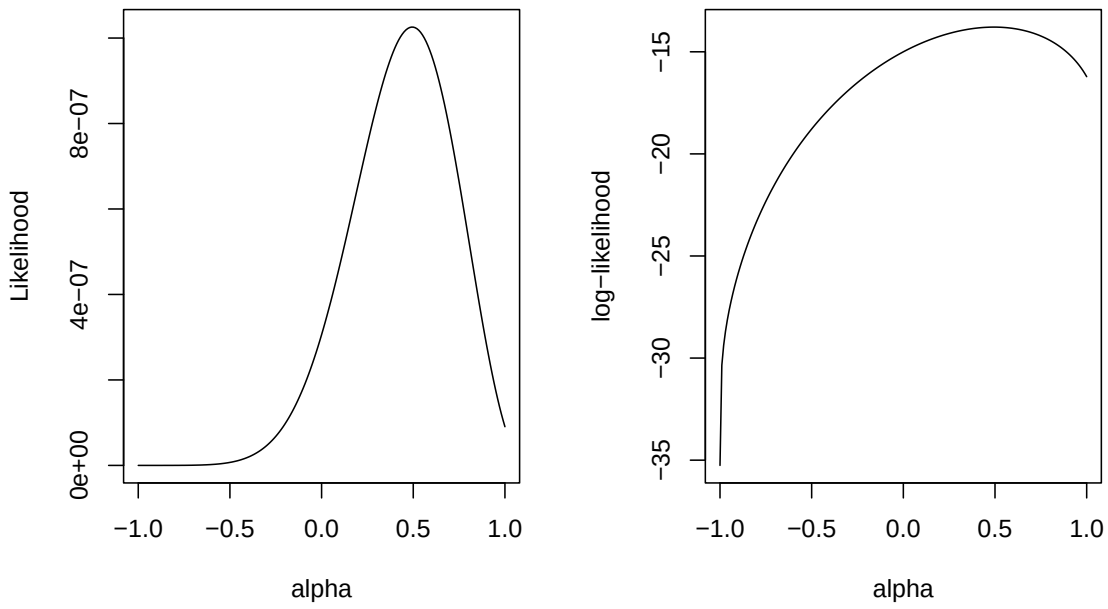

Figure 1.1: *Likelihood (left) and log-likelihood (right) for muon decay example.*

Unfortunately, no closed expression is available for the solution of the equation $l'(\alpha) = 0$. Figure 1.1 shows $\text{lik}(\alpha)$ and $l(\alpha)$ as functions of α for the actual value of $\boldsymbol{x}$. A good guess on the optimal value of α would be possible to obtain by a visual check. An automatic method would however be preferable, both in order to save work and for avoiding subjectivity. □

Example 2 (Rainfall of summer storms in Illinois)
Figure 1.2 shows a histogram of rainfall measurements of summer storms, measured by a network of rain gauges in southern Illinois for the years 1960-1964. A possible model for these data is a gamma distribution

$$f(x|\alpha, \lambda) = \frac{1}{\Gamma(\alpha)} \lambda^\alpha x^{\alpha-1} e^{-\lambda x}$$

where $\Gamma(\alpha)$ is the gamma function as defined in Devore and Berk (2007, page 190). Assuming that all observations are independent, the likelihood is in this case

$$\text{lik}(\alpha, \lambda) = \prod_{i=1}^{n} \frac{1}{\Gamma(\alpha)} \lambda^\alpha x_i^{\alpha-1} e^{-\lambda x_i}.$$

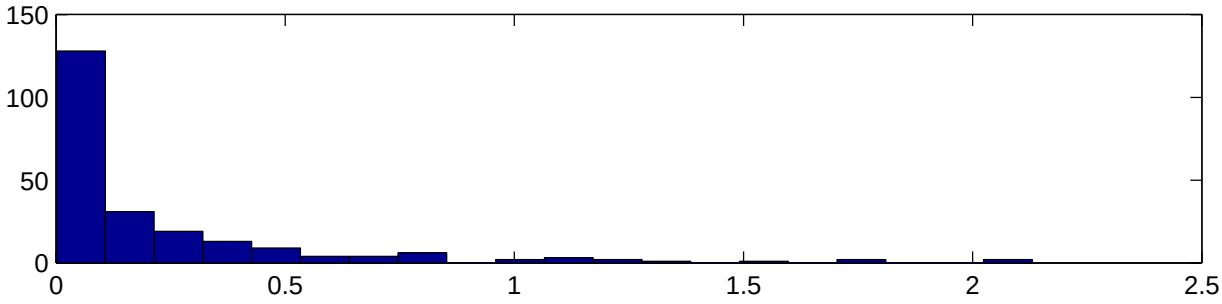

Figure 1.2: *Histogram of rainfall measurements of summer storms in Illinois.*

The log-likelihood now becomes

$$\begin{aligned} l(\alpha,\lambda) =& \sum_{i=1}^{n}[\alpha\log(\lambda)+(\alpha-1)\log(x_i)-\lambda x_i-\log(\Gamma(\alpha))] \\ =& n\alpha\log(\lambda)+(\alpha-1)\sum_{i=1}^{n}\log(x_i)-\lambda\sum_{i=1}^{n}x_i-n\log(\Gamma(\alpha)). \end{aligned} \tag{1.1}$$

Also in this case the log-likelihood is a smooth function of the parameters involved, and the derivatives are

$$\frac{\partial}{\partial\alpha}l(\alpha,\lambda) = n\log(\lambda)+\sum_{i=1}^{n}\log(x_i)-n\frac{\Gamma'(\alpha)}{\Gamma(\alpha)}; \tag{1.2}$$

$$\frac{\partial}{\partial\lambda}l(\alpha,\lambda) = n\alpha\lambda^{-1}-\sum_{i=1}^{n}x_i. \tag{1.3}$$

Putting the derivatives to zero and solving the equations is again difficult, making direct analytical solutions intractable.

In Figure 1.3, $\text{lik}(\alpha,\beta)$ and $l(\alpha,\beta)$ are plotted for different values of α and λ. These plots are produced by the commands given in Programwindow A.1.

Also in this case a good guess of the mode is possible, but a numerical procedure would be preferable. □

Example 3 (Survival times for leukemia)
In Cox and Oakes (1984) a dataset concerning the survival times for leukemia is discussed. We will consider a subset of these data, given in Table 1.1.

A distribution commonly applied to survival data is the Weibull distribution which has probability density

$$f(x;\alpha,\beta) = \beta\alpha^{-1}\left(\frac{x}{\alpha}\right)^{\beta-1}e^{-(x/\alpha)^{\beta}}.$$

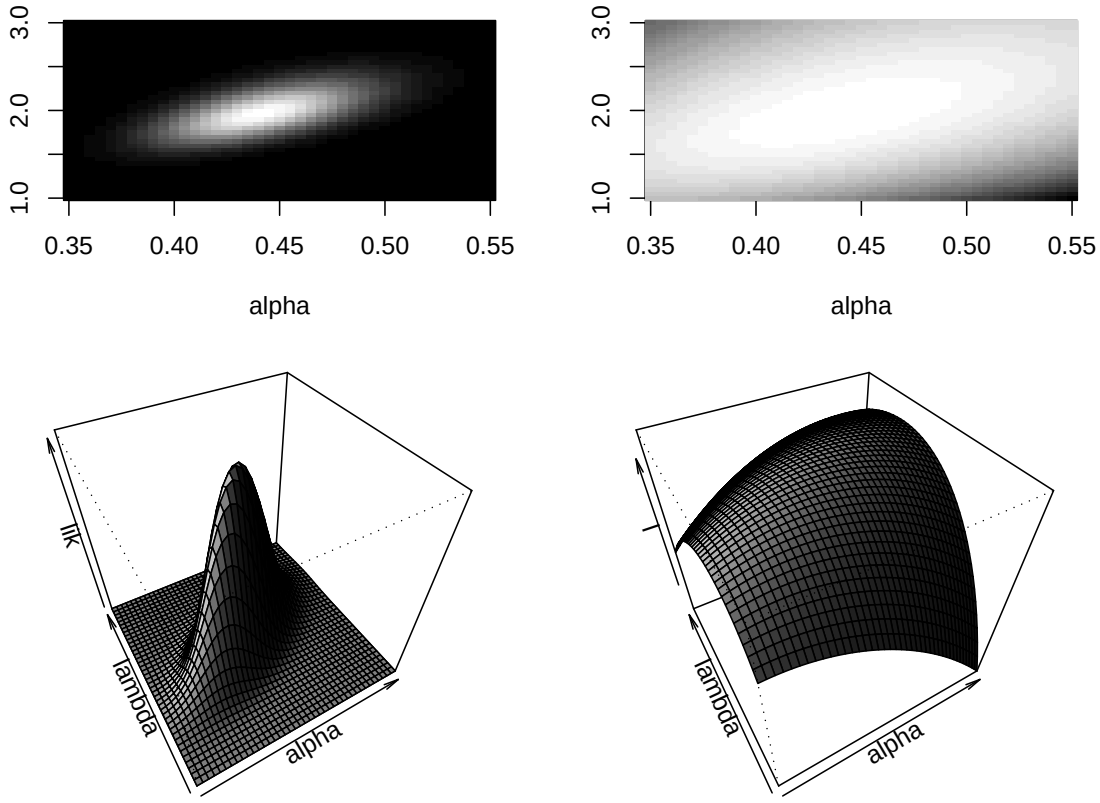

Figure 1.3: *Perspective and contour plot of likelihood function (left) and log-likelihood function (right) for rainfall measurements.*

Here α is a scale parameter while β describes the shape of the distribution. Assuming all observations $x_1, ..., x_n$ are independent, the likelihood function is

$$\text{lik}(\alpha, \beta) = \prod_{i=1}^{n} \beta \alpha^{-1} \left(\frac{x_i}{\alpha}\right)^{\beta-1} e^{-(x_i/\alpha)^\beta}$$

while the log-likelihood is

$$\begin{aligned} l(\alpha, \beta) =& \sum_{i=1}^{n} [\log(\beta) - \log(\alpha) + (\beta - 1)(\log(x_i) - \log(\alpha)) - (x_i/\alpha)^\beta] \\ =& n \log(\beta) - n\beta \log(\alpha) + (\beta - 1) \sum_{i=1}^{n} \log(x_i) - \sum_{i=1}^{n} (x_i/\alpha)^\beta. \end{aligned} \tag{1.4}$$

In Figure 1.4 $\text{lik}(\alpha, \beta)$ and $l(\alpha, \beta)$ are plotted for different values of α and β. Differentiating

56	65	17	7	16	22	3	4	2	3	8	4	3	30	4	43

Table 1.1: *Survival times for leukemia. Subset of dataset from Cox and Oakes (1984) corresponding to those individuals which responded "absent" to a given test.*

with respect to α and β, we get

$$\frac{\partial}{\partial\alpha}l(\boldsymbol{\theta}) = -\frac{n\beta}{\alpha} + \frac{\beta}{\alpha}\sum_{i=1}^{n}(x_i/\alpha)^{\beta}, \tag{1.5}$$

$$\frac{\partial}{\partial\beta}l(\boldsymbol{\theta}) = \frac{n}{\beta} - n\log(\alpha) + \sum_{i=1}^{n}\log(x_i) - \sum_{i=1}^{n}(x_i/\alpha)^{\beta}\log(x_i/\alpha). \tag{1.6}$$

As for the previous examples, maximizing the likelihood analytically is difficult. □

In this note we will discuss how numerical methods can be applied for solving statistical optimization problems. We will not go deeply into the general theory behind numerical methods. For this, we refer to other courses (i.e., MAT-INF 1100, INF-MAT 3370). Our intention is to demonstrate how such numerical methods can be applied to statistical problems, in addition to present some optimization methods which are specialized for use in statistics. We will start with some general remarks concerning optimization (section 2). Our primary application will be maximum likelihood estimation. Therefore, some properties of the log-likelihood function and the maximum likelihood estimators (section 3) will also be discussed. The remaining sections consider different optimization methods. All methods are iterative. They start with some initial guess on the parameters. Next, a new candidate to the optimum, presumable better than the initial guess is found. This new guess is used to find an even better one, and so one. Section 4 consider the Newton-Raphson method, which probably is the most used optimization method in statistics. A slight modification of this method, the scoring algorithm, is discussed in section 5. Some refinements of these algorithms are discussed in Section 6. All procedures discussed are illustrated by examples. The datasets and R code can be found on the course homepage. R routines are also given in the text.

2 General comments on optimization

As mentioned in the introduction, optimization problems often occur in statistics. Analytic expressions for maximum likelihood estimators in complex models are usually not easily available, and numerical methods are needed. Other types of optimization problems may also be of interest. For illustrative purpose, we will however concentrate on likelihood maximization.

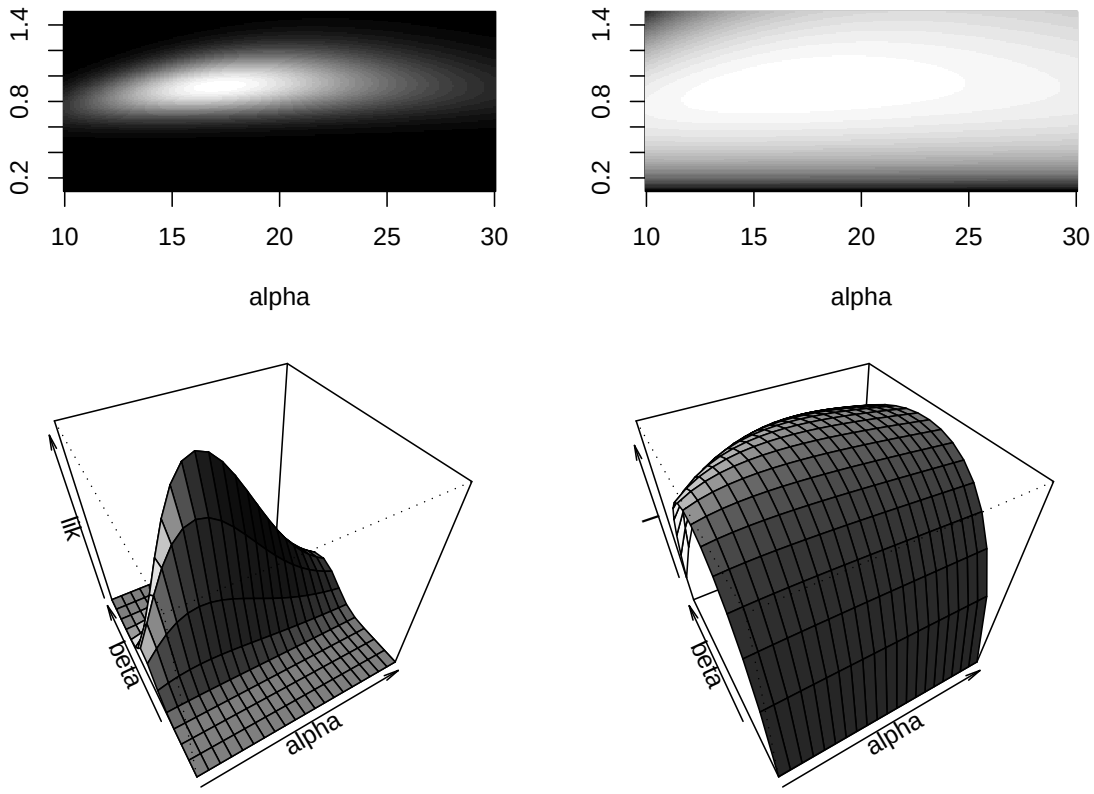

Figure 1.4: *Perspective and contour plot of likelihood function (left) and log-likelihood function (right) for leukemia data.*

Optimization is a huge field in numerical analysis and we will only focus on a few methods that are widely used in statistics. Although many of the general purpose methods are useful also for statistical problems, methods specifically designed for likelihood maximization may be better to use in certain situations. In any application, we may choose if we want to maximize or minimize, since maximization of a function l is equivalent to minimization of $-l$. Since our primary concern will be likelihood maximization, we will assume that maximization is our aim.

For a general discussion of optimization methods, we refer to Van Loan (2000). Lange (1999) is a very good reference for numerical analysis in general applied to statistics. We will only give some general comments here:

- In some applications, the function we want to maximize is well-behaved, in the sense that the function is concave and only one maximum point exists. In such cases most methods will work (although with different efficiency). For more complicated

problems, the concavity property is generally lost, and local maximum points may occur. In such cases, no guarantee is given that the numerical method will return the global maximum point. Two standard methods are then widely used:

(*a*) Find local maxima starting from widely varying starting values and then pick the maximum of these;

(*b*) Perturb a local optimum by a "small" amount, and use this as an starting point for a new run of your routine. Then see if your routine converges to a better point, or "always" to the same one.

- When searching for an appropriate method to use, you must choose between methods that need only evaluations of the function itself and methods that also require evaluations of the derivative(s) of the function. Algorithms using the derivative are somewhat more powerful, but the extra cost of calculating these *may* result in less efficiency overall. Note however, that in order to obtain information values and standard errors for the estimators, derivatives may be needed anyway (see next section).

- In many situations, application-specific algorithms can be constructed. Such algorithms can be far more efficient than the more general ones.

As already mentioned, our main concern will be on optimizing likelihood functions. The likelihood function is a function both on the random variable $\boldsymbol{X} = (X_1, ..., X_n)$ and the parameter vector $\boldsymbol{\theta}$. In general we should therefore write $\text{lik}(\boldsymbol{\theta}; \boldsymbol{X}) = f(\boldsymbol{X}; \boldsymbol{\theta})$ where $f(\boldsymbol{X}; \boldsymbol{\theta})$ is the probability density for $\boldsymbol{X}$ given $\boldsymbol{\theta}$. When estimation of $\boldsymbol{\theta}$ is to be performed, the observed value $\boldsymbol{X}$ is given, and can be considered as fixed. In those cases, we will simply write $\text{lik}(\boldsymbol{\theta})$.

The optimum of the likelihood function $\text{lik}(\boldsymbol{\theta})$ coincides with the optimum of the log-likelihood function $l(\boldsymbol{\theta})$. We will consider maximization of the log-likelihood function. In numerical literature, the function to maximize is usually denoted the *objective* function. We will follow this convention when discussing maximization algorithms in general.

3 Some properties of the likelihood function and maximum likelihood estimators

Devore and Berk (2007, sec. 7.4) discusses large sample properties of maximum likelihood estimators. Some of these results will be repeated in addition to that some extra properties important for numerical optimization of likelihood functions will be introduced. We will start by assuming only one unknown parameter θ. Afterwards we will generalize to several parameters.

3.1 The one-parameter situation

The derivative of the log-likelihood,

$$s(\theta; \boldsymbol{X}) = l'(\theta; \boldsymbol{X}) \tag{3.1}$$

is usually named the *score function*. Note that the score function is a random variable since it depends on the random observations $\boldsymbol{X}$. It can be expressed by the likelihood function itself through

$$s(\theta; \boldsymbol{X}) = \frac{1}{\text{lik}(\theta; \boldsymbol{X})} \text{lik}'(\theta; \boldsymbol{X})$$

which is obtained from (3.1) by ordinary rules on differentiation. If $l(\theta; \boldsymbol{X})$ is a smooth function in θ, the likelihood should have derivative equal to zero at the max-point. A common approach in order to find maximum points is therefore to solve the *scoring equation*

$$s(\widehat{\theta}; \boldsymbol{X}) = 0.$$

Note however that this criterion is only a necessary one, also minimum points and saddle points can be solutions to this equation. In order to evaluate if the solution actually *is* a maximum point, the second derivative must be inspected.

As is apparent from (3.1), $s(\theta; \boldsymbol{X})$ is a stochastic quantity. An important property of the scoring function is that if $\boldsymbol{X}$ has probability density $f(\boldsymbol{X}; \boldsymbol{\theta})$, then $\mathsf{E}[s(\theta; \boldsymbol{X})] = 0$. A solution of the scoring equation can therefore be seen as a value of $\widehat{\theta}$ such that the scoring function is equal to its expectation.

The variability of $s(\theta; \boldsymbol{X})$ reflects the uncertainty involved in estimating θ. The variance of s is called the *Fisher information* (sometimes it is also called the *expected information*).

The *Fisher information* $I(\theta)$ of an experiment can be written as (we will in this note use I both for the information based on one observation and for the information from a random sample, the textbook distinuish between these situations by using I_n for the latter)

$$I(\theta) = \mathsf{Var}[s(\theta; \boldsymbol{X})] = -\mathsf{E}\left[l''(\theta)\right]. \tag{3.2}$$

For proof of the properties given above for the discrete case with $n = 1$, see Devore and Berk (2007, page 365). The general case is a trivial extension. A consequence of the first equality in (3.2) is that $I(\theta)$ is always non-negative.

Given the maximum likelihood estimator $\widehat{\theta}$ of θ, an important task is to derive uncertainty properties about this estimator. The theorem at page 369 in Devore and Berk (2007) state that for a large number of observations, $\widehat{\theta}$ is approximately normal distributed with expectation θ and variance equal to $1/I(\theta)$. In the textbook a proof is sketched for the i.i.d case, but this result is also valid in more general situations.

3.2 The multi-parameter situation

Assume $\boldsymbol{\theta} = (\theta_1, ..., \theta_p)^T$ is a vector of p, say, unknown parameters. The derivate of the log-likelihood, still named the score function, now is a vector:

$$\boldsymbol{s}(\boldsymbol{\theta}; \boldsymbol{X}) = l'(\boldsymbol{\theta}; \boldsymbol{X}) = \frac{1}{\text{lik}(\boldsymbol{\theta}; \boldsymbol{X})}\text{lik}'(\boldsymbol{\theta}; \boldsymbol{X}). \tag{3.3}$$

The ith element of the vector $\boldsymbol{s}(\boldsymbol{\theta}; \boldsymbol{X})$, $s_i(\boldsymbol{\theta}; \boldsymbol{X})$, is the partial derivative of $l(\boldsymbol{\theta}; \boldsymbol{X})$ with respect to θ_i. A more mathematically correct way of expressing $\boldsymbol{s}(\boldsymbol{\theta}; \boldsymbol{X})$ would be $\frac{\partial}{\partial \boldsymbol{\theta}} l(\boldsymbol{\theta}; \boldsymbol{X})$, but we will use the simpler form $l'(\boldsymbol{\theta}; \boldsymbol{X})$.

As for the one-parameter case, each $s_i(\boldsymbol{\theta}; \boldsymbol{X})$ has expectation zero. Finding the MLE by solving the *scoring equations*

$$\boldsymbol{s}(\widehat{\boldsymbol{\theta}}; \boldsymbol{X}) = \boldsymbol{0} \tag{3.4}$$

now result in a set of p equations with p unknowns.

The expected information is now generalized to be a matrix $\boldsymbol{I}(\boldsymbol{\theta})$ with the (i, j)th entry given by

$$I_{ij}(\boldsymbol{\theta}) = \mathsf{Cov}[s_i(\boldsymbol{\theta}; \boldsymbol{X}), s_j(\boldsymbol{\theta}; \boldsymbol{X})] = -\mathsf{E}\left[\frac{\partial^2}{\partial\theta_i \partial\theta_j} l(\boldsymbol{\theta})\right]. \tag{3.5}$$

Here the second equality can be proven by a simple generalization of the argument in the one-parameter case. In the multi-parameter situation we usually name $\boldsymbol{I}(\boldsymbol{\theta})$ by the *expected information matrix* or the *Fisher information matrix*. An important property of $I(\boldsymbol{\theta})$ is that it is always positive (semi-)definite[1]. This will be of importance in the construction of the scoring algorithm in section 5.

The large sample theory described in 7.4 of Devore and Berk (2007) also generalizes to the multi-parameter case, so that the MLE is, under appropriate smoothness conditions on f, consistent. Further, each element $\widehat{\theta}_i$ of $\widehat{\boldsymbol{\theta}}$ will, for a sufficiently large number of observations, have a sampling distribution approximately normal with expectation θ_i and variance equal to $\{\boldsymbol{I}^{-1}(\boldsymbol{\theta})\}_{ii}$, the ith entry of the diagonal of $\boldsymbol{I}^{-1}(\boldsymbol{\theta})$. Further, the covariance between $\widehat{\theta}_i$ and $\widehat{\theta}_j$ is approximately given by the (i, j)th entry of $\boldsymbol{I}^{-1}(\boldsymbol{\theta})$.

Note that $\boldsymbol{I}(\boldsymbol{\theta})$ depends on the unknown quantity $\boldsymbol{\theta}$. Common practice is to insert the estimated value $\widehat{\boldsymbol{\theta}}$ for $\boldsymbol{\theta}$ giving an estimate $\boldsymbol{I}(\widehat{\boldsymbol{\theta}})$ of $\boldsymbol{I}(\boldsymbol{\theta})$.

A further complication is that the expectations in (3.5) are not always possible to compute. Then an alternative is to use the *observed information* matrix $\boldsymbol{J}(\boldsymbol{\theta})$ with (i, j)th entry given by

$$J_{ij}(\boldsymbol{\theta}) = -\frac{\partial^2}{\partial\theta_i \partial\theta_j} l(\boldsymbol{\theta}). \tag{3.6}$$

[1] A matrix $\boldsymbol{I}$ is positive semi-definite if $\boldsymbol{a}'\boldsymbol{I}\boldsymbol{a} \geq 0$ for all vectors $\boldsymbol{a}$.

As for the expected information matrix, an estimate $\widehat{\boldsymbol{\theta}}$ needs to be inserted for $\boldsymbol{\theta}$ in order to evaluate $\boldsymbol{J}(\boldsymbol{\theta})$. The ith diagonal element of $\boldsymbol{J}^{-1}(\boldsymbol{\theta})$ can then be used as an approximation for the variance of $\widehat{\theta}_i$ instead of the ith diagonal element of $\boldsymbol{I}^{-1}(\boldsymbol{\theta})$. Both these approximations will be equally valid in the sense that as the number of observations increases, the approximation error will decrease to zero. If possible to calculate, the expected information is preferable, since the observed information in some cases can be unstable. Note that in many standard models used in statistics, $\boldsymbol{I}(\boldsymbol{\theta}) = \boldsymbol{J}(\boldsymbol{\theta})$.

4 The Newton-Raphson method

One of the most used methods for optimization in statistics is the Newton-Raphson method (or Newton's rule). It is based on approximating the function which we want to optimize by a quadratic one. The optimum of the approximation (which is easy to calculate) gives a guess of the optimum for the actual function. If this guess is not adequately close to the optimum, a new approximation is computed and the process repeated.

Assume for simplicity that l only involves a one-dimensional parameter and that $\bar{\theta}$ is our current best guess on the maximum of $l(\theta)$. By using a Taylor series expansion around $\bar{\theta}$, $l(\theta)$ can be approximated by

$$\tilde{l}_{\bar{\theta}}(\theta) = l(\bar{\theta}) + l'(\bar{\theta})(\theta - \bar{\theta}) + \frac{1}{2} l''(\bar{\theta})(\theta - \bar{\theta})^2. \tag{4.1}$$

When θ is close to $\bar{\theta}$, the difference $l(\theta) - \tilde{l}_{\bar{\theta}}(\theta)$ is small. Figure 4.1 shows $l(\theta)$ and $\tilde{l}_{\bar{\theta}}(\theta)$ for a specific example. We see that the maximum value of $\tilde{l}_{\bar{\theta}}(\theta)$ is closer to the maximum value of $l(\theta)$ than $\bar{\theta}$.

The *gradient* of $\tilde{l}_{\bar{\theta}}(\theta)$ at θ is

$$\tilde{l}'_{\bar{\theta}}(\theta) = l'(\bar{\theta}) + l''(\bar{\theta})(\theta - \bar{\theta})$$

and the *Hessian* or second derivative is

$$\tilde{l}''_{\bar{\theta}}(\theta) = l''(\bar{\theta}).$$

At the point $\bar{\theta}$, $l(\theta)$ and $\tilde{l}_{\bar{\theta}}(\theta)$ have equal first and second derivatives. Note that for the particular case when l is a log-likelihood function, the Hessian is equal to minus the observed information evaluated at $\theta = \bar{\theta}$, $l''(\bar{\theta}) = -J(\bar{\theta})$.

In the optimum point of the approximation, $\tilde{l}_{\bar{\theta}}(\theta)$ has a gradient equal to zero, giving the following equation:

$$l''(\bar{\theta})(\theta - \bar{\theta}) = -l'(\bar{\theta}). \tag{4.2}$$

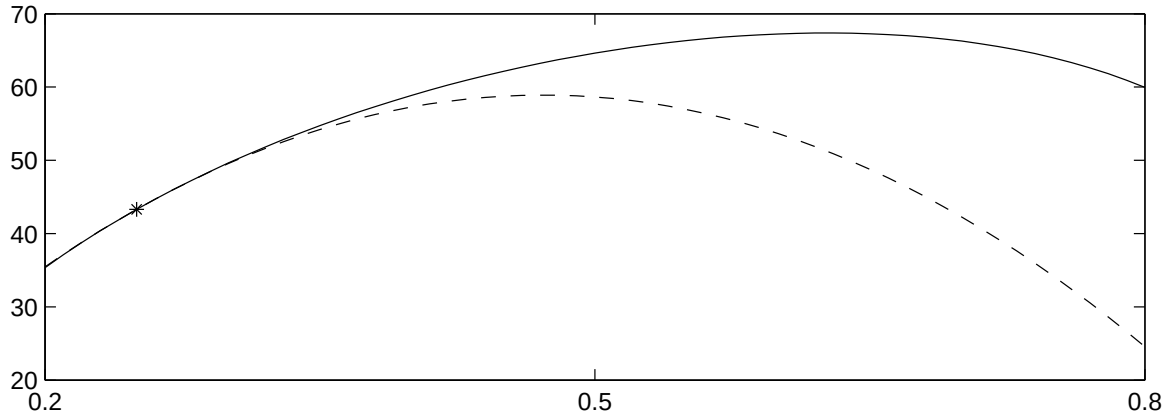

Figure 4.1: *A function $l(\theta)$ (solid line) and its quadratic approximation $\tilde{l}_{\bar{\theta}}(\theta)$ (dashed line). The point $(\bar{\theta}, l(\bar{\theta}))$ is given by a $*$.*

Solving with respect to θ, we get

$$\theta = \bar{\theta} - \frac{l'(\bar{\theta})}{l''(\bar{\theta})}. \tag{4.3}$$

This gives a procedure for optimizing $\tilde{l}_{\bar{\theta}}(\theta)$. Our aim is however to optimize $l(\theta)$. Since $\tilde{l}_{\bar{\theta}}(\theta)$ is an approximation of $l(\theta)$, our hope is then that (4.3) will give us a new value closer to the optimum of $l(\theta)$. This suggests an iterative procedure for optimizing $l(\theta)$:

$$\theta^{(s+1)} = \theta^{(s)} - \frac{l'(\theta^{(s)})}{l''(\theta^{(s)})}; \tag{4.4}$$

which is the Newton-Raphson method. The procedure is run until there is no significant difference between $\theta^{(s)}$ and $\theta^{(s+1)}$.[2]

$\theta^{(s+1)} = \theta^{(s)}$ is equivalent to $l'(\theta^{(s)}) = 0$. This demonstrates that when the algorithm has converged, we have reached a stationary point of $l(\theta)$. This point could be a maximum point, a minimum point or even a saddle point. However, if $l''(\theta^{(s)}) < 0$, the point *is* a maximum point. This should be checked in each case. Figure 4.2 shows the first four iterations of a Newton-Raphson algorithm. The results from the last two iterations are almost indistinguishable, indicating that convergence is reached.

When $l(\theta)$ is a log-likelihood function, this algorithm can be written as

$$\theta^{(s+1)} = \theta^{(s)} + \frac{s(\theta^{(s)})}{J(\theta^{(s)})}.$$

In such cases a convergence point is a maximum point if $J(\theta^{(s)}) > 0$.

[2]Other convergence criteria could also be used but will not be considered here

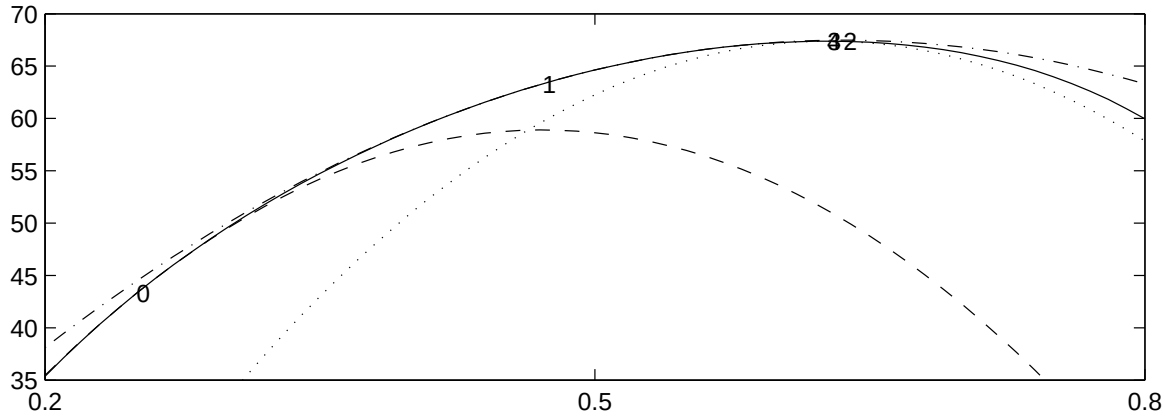

Figure 4.2: *A function $l(\theta)$ (solid line) and its quadratic approximations $\tilde{l}_{\theta^s}(\theta)$ for $s = 0$ (solid plus star), $s = 1$ (dashed), $s = 2$ (dashed-dotted) and $s = 3$ (dotted). The points $(\theta^s, l(\theta^s))$ for $s = 0, 1, 2, 3, 4$ are given by their numbers.*

The Newton-Raphson algorithm will only converge to the closest stationary point. If several such points are present, the choice of starting value becomes critical. In many cases of statistics, reasonable starting values can be found by other means, i.e. moment estimators.

Example 1 (Muon decay, cont.)
We want in this case to maximize the log-likelihood function

$$l(\alpha) = \sum_{i=1}^{n} \log(1 + \alpha x_i) - n \log(2)$$

In section 1 we found that

$$s(\alpha) = l'(\alpha) = \sum_{i=1}^{n} \frac{x_i}{1 + \alpha x_i}.$$

Differentiating once more with respect to α gives

$$l''(\alpha) = -J(\alpha) = -\sum_{i=1}^{n} \frac{x_i^2}{(1 + \alpha x_i)^2}.$$

Note $l''(\alpha)$ is always negative (and therefore $J(\alpha)$ is always positive) for all possible α values, showing that the log-likelihood function is concave and unimodal.

The iterative steps in the Newton-Raphson algorithm are given by

$$\alpha^{(s+1)} = \alpha^{(s)} + \frac{\sum_{i=1}^{n} \frac{x_i}{1+\alpha^{(s)} x_i}}{\sum_{i=1}^{n} \frac{x_i}{(1+\alpha^{(s)} x_i)^2}}.$$

```
nr.muon <- function(x,alpha0=0.6,eps=0.000001)
{
 n = length(x)
 diff = 1;
 alpha = alpha0;
 l = sum(log(1+alpha*x))-n*log(2)
 while(diff>eps)
  {
   alpha.old = alpha
   s = sum(x/(1+alpha*x))
   Jbar = sum((x/(1+alpha*x))^2)
   alpha = alpha+s/Jbar
   l = sum(log(1+alpha*x))-n*log(2)
   diff = abs(alpha-alpha.old)
 }
 list(alpha,Jbar)
}
```

Programwindow 4.1: R *code for running Newton-Raphson on muon decay example.*

Programwindow 4.1 describes a R function for running the Newton-Raphson algorithm using a stopping criteria

$$|\alpha^{(s+1)} - \alpha^{(s)}| < 0.000000000001.$$

The small value on the right hand side is much smaller than necessary, but is used in order to demonstrate properties of the Newton-Raphson algorithm. A run of this function gave the following results:

Iteration s	$\alpha^{(s)}$	$l(\alpha^{(s)})$
0	0.6000000	-19.65135
1	0.5040191	-19.58507
2	0.4944591	-19.58454
3	0.4943927	-19.58454
4	0.4943927	-19.58454

We see that after one iteration, one decimal is correct. This is increased to three decimals after 2 iterations and to at least 7 decimals after 3 iterations, illustrating the rapid increase in accuracy for the Newton-Raphson algorithm. Further, since the log-likelihood funcion is concave and unimodal, the resulting point is a maximum point.

The Fisher information is in this case

$$\begin{aligned} I(\theta) =& \mathsf{E}[J(\theta)] \\ =& n\mathsf{E}[\frac{X^2}{(1+\alpha X)^2}] \\ =& \frac{n}{2}\int_{-1}^{1}\frac{x^2}{1+\alpha x}dx \\ =& \frac{n}{2\alpha^2}[\int_{-1}^{1}(\alpha x-1)dx+\int_{-1}^{1}\frac{1}{1+\alpha x}dx] \\ =& \frac{-n}{\alpha^2}+\frac{n}{2\alpha^3}\log\left(\frac{1+\alpha}{1-\alpha}\right) \end{aligned} \tag{4.5}$$

For $\widehat{\alpha} = 0.4943927$, $\widehat{I} = 11.78355$ giving an approximative standard error equal to 0.291 for $\widehat{\alpha}$. Using the observed information, the standard error would be estimated to 0.297. □

The argument for deriving the Newton-Raphson algorithm for optimization in one dimension can be directly extended to multi-dimensional problems giving the multi-parameter Newton-Raphson method:

$$\boldsymbol{\theta}^{(s+1)} = \boldsymbol{\theta}^{(s)} - \left[\boldsymbol{l}''(\boldsymbol{\theta}^{(s)})\right]^{-1}\boldsymbol{l}'(\boldsymbol{\theta}^{(s)}) \tag{4.6}$$

where $\boldsymbol{l}'(\boldsymbol{\theta})$ now is a vector consisting of the partial derivatives while $\boldsymbol{l}''(\boldsymbol{\theta})$ is a matrix with (i,j) entry equal to the second derivative with respect to θ_i and θ_j. $\boldsymbol{l}''(\boldsymbol{\theta})$ is usually denoted as the Hessian matrix. The algorithm can be written as

$$\boldsymbol{\theta}^{(s+1)} = \boldsymbol{\theta}^{(s)} + \boldsymbol{J}^{-1}(\boldsymbol{\theta}^{(s)})\boldsymbol{s}(\boldsymbol{\theta}^{(s)}). \tag{4.7}$$

for likelihood optimization, where $s(\boldsymbol{\theta})$ is the score function while $\boldsymbol{J}(\boldsymbol{\theta})$ is the observed information matrix. In order to check if the resulting point is a maximum point, we need to see if the observed information matrix is positive definite, which is the case if all eigenvalues of $\boldsymbol{J}(\widehat{\boldsymbol{\theta}})$ are positive.

Example 2 (Rainfall data, cont.)
In this case we want to maximize

$$l(\alpha,\lambda) = n\alpha\log(\lambda) + (\alpha-1)\sum_{i=1}^{n}\log(x_i) - \lambda\sum_{i=1}^{n}x_i - n\log(\Gamma(\alpha)).$$

The first derivatives (or score functions) are given by (1.2)-(1.3) while the second derivatives are

$$\frac{\partial^2}{\partial\alpha^2}l(\alpha,\lambda) = -n\frac{\Gamma''(\alpha)\Gamma(\alpha)-\Gamma'(\alpha)^2}{\Gamma(\alpha)^2} \qquad \frac{\partial^2}{\partial\lambda\partial\alpha}l(\alpha,\lambda) = n\lambda^{-1} \tag{4.8}$$

$$\frac{\partial^2}{\partial\lambda^2}l(\alpha,\lambda) = -n\alpha\lambda^{-2} \tag{4.9}$$

A problem with implementing the Newton-Raphson algorithm in this case is that the first and second derivatives of the gamma function is not directly available. *Numerically* these can however be approximated by

$$\Gamma'(\alpha) \approx \frac{\Gamma(\alpha+h)-\Gamma(\alpha)}{h}$$
$$\Gamma''(\alpha) \approx \frac{\Gamma'(\alpha+h)-\Gamma'(\alpha)}{h} \approx \frac{\Gamma(\alpha+2h)-2\Gamma(\alpha+h)+\Gamma(\alpha)}{h^2}$$

for h small[3].

Programwindow 4.2 describes a R function for running the Newton-Raphson algorithm using a stopping criteria

$$\sum_i [|\alpha_i^{(s+1)} - \alpha_i^{(s)}| + |\lambda_i^{(s+1)} - \lambda_i^{(s)}|] < 0.0001.$$

A run of this function, using the moment estimates for α and λ as starting values, gave the following results:

Iteration s	$\alpha^{(s)}$	$\lambda^{(s)}$	$l(\boldsymbol{\theta}^{(s)})$
0	0.3762506	1.676755	-656.1040
1	0.4306097	1.919006	-580.9499
2	0.4405084	1.963119	-567.5825
3	0.4407890	1.964370	-567.2049
4	0.4407914	1.964380	-567.2017
5	0.4407914	1.964380	-567.2017
6	0.4407914	1.964380	-567.2017

Also for this example, convergence is fast. The algorithm is however sensitive to starting values. Using $\alpha^{(0)} = \lambda^{(0)} = 1$, both $\alpha^{(1)}$ and $\lambda^{(1)}$ becomes negative, resulting in that the algorithm crashes. We will in Section 6 see how this can be avoided.

The observed information matrix $\boldsymbol{J}(\boldsymbol{\theta}) = -l''(\boldsymbol{\theta})$ is directly available from the Newton-Raphson algorithm. Note from (4.8)-(4.9) that $\boldsymbol{J}(\boldsymbol{\theta})$ do not depend on the observations, resulting in that the Fisher information matrix in this case is equal to the observed one. Inserting the estimate of $\boldsymbol{\theta}$ we get

$$\boldsymbol{I}(\widehat{\boldsymbol{\theta}}) = \begin{bmatrix} 1394.1 & -115.6 \\ -115.6 & 25.9 \end{bmatrix}, \quad \boldsymbol{I}^{-1}(\widehat{\boldsymbol{\theta}}) = \begin{bmatrix} 0.00114 & 0.0051 \\ 0.0051 & 0.0613 \end{bmatrix}$$

giving approximative standard errors 0.0337 and 0.248 for $\widehat{\alpha}$ and $\widehat{\lambda}$, respectively.

[3]Better approximations are possible, but the simple choice will be sufficient here. Note that h shouldn't be chosen too small, because round-off errors in the calculation of $\Gamma(\alpha)$ on a computer then can make the approximation bad, see Van Loan (2000, sec. 1.5.2)

```
nr.gamma <- function(x,eps=0.000001)
{
 n = length(x);sumx = sum(x);sumlogx = sum(log(x))
 diff = 1;h = 0.0000001;
 alpha = mean(x)^2/var(x);lambda=mean(x)/var(x)
 theta = c(alpha,lambda)
 while(diff>eps)
  {
   theta.old = theta
   g = gamma(alpha)
   dg = (gamma(alpha+h)-gamma(alpha))/h
   d2g = (gamma(alpha+2*h)-2*gamma(alpha+h)+
         gamma(alpha))/h^2
   s = c(n*log(lambda)+sumlogx-n*dg/gamma(alpha),
         n*alpha/lambda-sumx)
   Jbar = matrix(c(n*(d2g*g-dg^2)/g^2,-n/lambda,
                   -n/lambda,n*alpha/lambda^2),ncol=2)
   theta = theta + solve(Jbar,s)
   alpha = theta[1];lambda = theta[2]
   diff = sum(abs(theta-theta.old))
 }
 list(theta=theta,Jbar=Jbar)
}
```

Programwindow 4.2: R *code for running Newton-Raphson on gamma distributed data.*

Note that since $\boldsymbol{J}(\boldsymbol{\theta}) = \boldsymbol{I}(\boldsymbol{\theta})$, $\boldsymbol{J}(\boldsymbol{\theta})$ will be positive definite for all values of $\boldsymbol{\theta}$. The log-likelihood function is then concave, resulting in that the values of $(\widehat{\alpha}, \widehat{\lambda})$ obtained by the Newton-Raphson algorithm is equal to the global maximum. □

Example 3 (Leukemia data, cont.)
The log-likelihood is given in (1.4) while the score functions are given in equations (1.5)

and (1.6). Further

$$J_{1,1} = -\frac{\partial^2}{\partial\alpha^2} l(\boldsymbol{\theta}) = -\frac{n\beta}{\alpha^2} + \frac{\beta(\beta+1)}{\alpha^2}\sum_{i=1}^{n}(x_i/\alpha)^\beta,$$

$$J_{1,2} = -\frac{\partial^2}{\partial\alpha\partial\beta} l(\boldsymbol{\theta}) = \frac{n}{\alpha} - \frac{1}{\alpha}\sum_{i=1}^{n}(x_i/\alpha)^\beta - \frac{\beta}{\alpha}\sum_{i=1}^{n}(x_i/\alpha)^\beta \log(x_i/\alpha),$$

$$J_{2,2} = -\frac{\partial^2}{\partial\beta^2} l(\boldsymbol{\theta}) = \frac{n}{\beta^2} + \sum_{i=1}^{n}(x_i/\alpha)^\beta [\log(x_i/\alpha)]^2.$$

Programwindow 4.3 shows R code for a Newton-Raphson algorithm in this case. With starting values $\alpha^{(0)} = 20$ and $\beta^{(0)} = 1$ and a convergence criterion

$$|\alpha^{(s+1)} - \alpha^{(s)}| + |\beta^{(s+1)} - \beta^{(s)}| < 0.00001,$$

we get the following results:

Iteration s	$\alpha^{(s)}$	$\beta^{(s)}$	$l(\boldsymbol{\theta}^{(s)})$
0	10.00000	1.0000000	-138.00580
1	11.88883	0.8904244	-62.98770
2	15.09949	0.9287394	-62.22634
3	16.74320	0.9244928	-62.10186
4	17.17639	0.9220478	-62.09619
5	17.20186	0.9218854	-62.09617
6	17.20194	0.9218849	-62.09617
7	17.20194	0.9218849	-62.09617

Inserting $\widehat{\boldsymbol{\theta}}$ for $\boldsymbol{\theta}$ into $J(\boldsymbol{\theta})$ gives

$$\boldsymbol{J}(\widehat{\boldsymbol{\theta}}) = \begin{bmatrix} 0.0460 & -0.4324 \\ -0.4324 & 36.3039 \end{bmatrix}, \quad \boldsymbol{J}^{-1}(\widehat{\boldsymbol{\theta}}) = \begin{bmatrix} 24.5071 & 0.2919 \\ 0.2919 & 0.0310 \end{bmatrix}$$

giving approximate standard error 4.9505 for $\widehat{\alpha}$ and 0.1761.

Also this Newton-Raphson algorithm is sensitive to the starting values chosen. With $\alpha^{(0)} = 20$ and $\beta^{(0)} = 2$, both $\alpha^{(1)}$ and $\beta^{(1)}$ become negative, which are illegal values for these parameters. □

An advantage in using the Newton-Raphson algorithm for statistical problems, is that log-likelihoods in many cases are close to quadratic functions around their maximum points. This is connected to that maximum likelihood estimators are approximately normally distributed (the logarithm of a Gaussian density is a quadratic function). In such cases, the approximation $\tilde{l}_{\tilde{\theta}}(\theta)$ becomes a very good approximation near the maximum point.

In more complex situations, the use of a Newton-Raphson algorithm may be more problematic. In the next sections we will discuss modifications of the Newton-Raphson algorithm, making it more robust.

```
nr.weibull = function(x,theta0,eps=0.000001)
 {
  n = length(x);
  sumlogx = sum(log(x));
  diff = 1;theta = theta0;alpha = theta[1];beta = theta[2]
  while(diff>eps)
   {
    theta.old = theta
    w1 = sum((x/alpha)^beta)
    w2 = sum((x/alpha)^beta*log(x/alpha))
    w3 = sum((x/alpha)^beta*log(x/alpha)^2)
    s = c(-n*beta/alpha+beta*w1/alpha,
          n/beta-n*log(alpha)+sumlogx-w2)
    Jbar = matrix(c(-n*beta/alpha^2+beta*(beta+1)*w1/alpha^2,
                    n/alpha-w1/alpha-beta*w2/alpha,
                    n/alpha-w1/alpha-beta*w2/alpha,n/beta^2+w3),
                  ncol=2)
    theta = theta + solve(Jbar,s)
    alpha = theta[1];beta = theta[2]
    diff = sum(abs(theta-theta.old))
  }
 list(alpha=alpha,beta=beta,Jbar=Jbar)
}
```

Programwindow 4.3: R *code for running Newton-Raphson on Weibull distributed data.*

5 Fisher's scoring algorithm

Figure 5.1 illustrates one type of problem that can occur when using the Newton-Raphson algorithm. $\bar{\theta}$ is given in a point where $l(\theta)$ is convex (that is the second derivative is positive). Using (4.3) will give a *reduction* in l, since (4.3) in this case finds a minimum point of $\tilde{l}_{\bar{\theta}}(\theta)$.

This problem with the Newton-Raphson method is directly translated to the multi-parameter case. In the general case, if at least one of the eigenvalues of the Hessian matrix $l''(\boldsymbol{\theta})$ is positive, a smaller value of $l(\boldsymbol{\theta})$ can be obtained from one iteration to the other. A standard trick in numerical literature for such cases is to replace the Hessian matrix with another matrix which is negative definite. For likelihood optimization, this means replacing $\boldsymbol{J}(\boldsymbol{\theta})$ with a *positive* definite matrix. A possible candidate could be the identity matrix, but a more efficient choice is available. The Fisher information $\boldsymbol{I}(\boldsymbol{\theta})$ is equal to

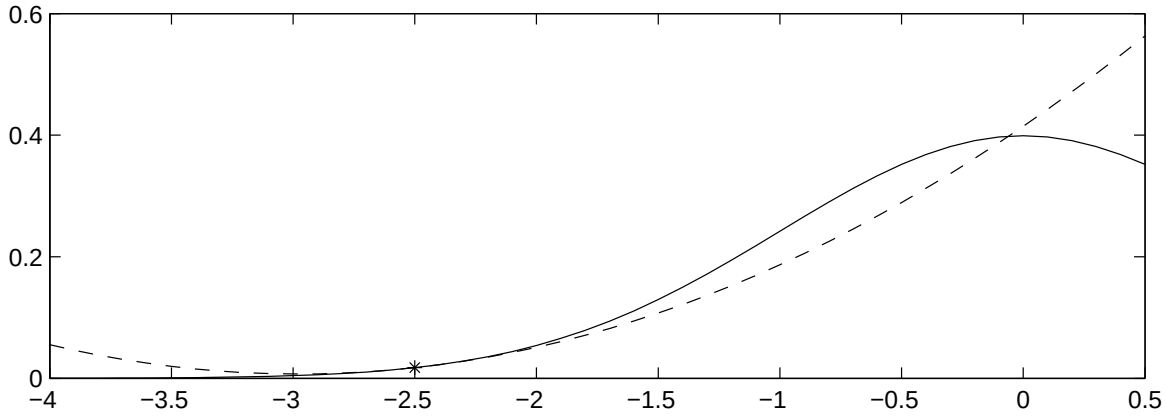

Figure 5.1: *Example of a one-dimensional function $l(\theta)$ which is to be maximized (solid line). The quadratic approximation $\tilde{l}_{\bar{\theta}}(\theta)$ defined in* (4.1) *is shown as a dashed line with the value of $\bar{\theta}$ marked by the vertical line.*

the expectation of $\boldsymbol{J}(\boldsymbol{\theta})$, indicating that these two matrices should be similar. But $\boldsymbol{I}(\boldsymbol{\theta})$ will always be positive definite, making this matrix a possibility for replacing $\boldsymbol{J}(\boldsymbol{\theta})$. This is the *Fisher's method of scoring* (or the *scoring algorithm*):

$$\boldsymbol{\theta}^{(s+1)} = \boldsymbol{\theta}^{(s)} + \boldsymbol{I}^{-1}(\boldsymbol{\theta}^{(s)})\boldsymbol{s}(\boldsymbol{\theta}^{(s)}). \tag{5.1}$$

Note that for this algorithm, the Fisher information is directly available.

Example 1 (Muon decay, cont.)
The Fisher information is in this case given by (4.5). Based on this, a scoring algorithm can be constructed as given in Programwindow 5.1. Note that the only change from the Newton-Raphson algorithm given in Programwindow 4.1 is the use of $I(\theta)$ instead of $J(\theta)$. Running this algorithm with convergence criterion

$$|\alpha^{(s+1)} - \alpha^{(s)}| < 0.000001,$$

convergence was obtained after five iterations. The $\widehat{\alpha}$ value was exactly the same as that obtained by the Newton-Raphson algorithm, and also the number of iterations needed for convergence is comparable for the two algorithms. □

In general it is not possible to say which algorithm that would be preferable *if both converges*. The next example illustrates a situation where the Newton-Raphson algorithm have problems in converging while the scoring algorithm is more robust.

Example 4 (Truncated Poisson)
We will consider an example which involves the estimation of the parameter of a *truncated* Poisson distribution given by

$$f(x;\theta) = \frac{\theta^x e^{-\theta}}{x!(1-e^{-\theta})}, \quad x = 1, 2, \ldots$$

```
scoring.muon <- function(x,alpha0=0.6,eps=0.000001)
{
 n = length(x)
 diff = 1;
 alpha = alpha0;
 l = sum(log(1+alpha*x))-n/2
 while(diff>eps)
  {
   alpha.old = alpha
   s = sum(x/(1+alpha*x))
   Ibar = n*(log((1+alpha)/(1-alpha))/(2*alpha^3)-1/(alpha^2))
   alpha = alpha+s/Ibar
   l = sum(log(1+alpha*x))-n/2
   diff = abs(alpha-alpha.old)
 }
 list(alpha,Ibar)
}
```

Programwindow 5.1: R *routine for estimation in the muon decay example using the scoring algorithm.*

x	1	2	3	4	5	6
f_x	1486	694	195	37	10	1

Table 5.1: *Grouped data from a truncated Poisson distribution, where f_x represents the frequency of x. Table 4.1 from Everitt (1987).*

Such a density might arise, for example, in studying the size of groups at parties. The data in Table 5.1 is taken from Everitt (1987) and represents samples from this distribution.

Assume n is the number of observations and x_i is the value of observation i. The likelihood function is given by

$$\text{lik}(\theta) = \prod_{i=1}^{n} \frac{\theta^{x_i} e^{-\theta}}{x_i!(1 - e^{-\theta})}$$

```
nr.trpo = function(x,theta0,eps=0.000001)
 {
  n = sum(x);i = 1:6;sumx = sum(x*i)
  theta = theta0;diff = 1
  while(diff> eps)
   {
    theta.old = theta
    s = sumx/theta-n/(1-exp(-theta))
    Jbar = sumx/theta^2-n*exp(-theta)/(1-exp(-theta))^2
    theta = theta+s/Jbar
    diff = abs(theta-theta.old)
  }
  list(theta,Jbar)
}
```

Programwindow 5.2: R *routine for estimation in the truncated Poisson distribution using the Newton Raphson algorithm.*

and the log-likelihood

$$\begin{aligned} l(\theta) =& \sum_{i=1}^{n} [x_i \log(\theta) - \theta - \log(x_i!) - \log(1 - e^{-\theta})] \\ =& \text{Const.} + \log(\theta) \sum_{i=1}^{n} x_i - n\theta - n \log(1 - e^{-\theta}). \end{aligned}$$

Differentiating with respect to θ, we get

$$l'(\theta) = \frac{\sum_{i=1}^{n} x_i}{\theta} - n - n \frac{e^{-\theta}}{1 - e^{-\theta}}.$$

If we put $l'(\theta)$ to zero we find that the resulting equation has no explicit solution for θ. We will apply both the Newton-Raphson and the scoring algorithms for obtaining numerical solutions. The second derivative is given by

$$l''(\theta) = -\frac{\sum_{i=1}^{n} x_i}{\theta^2} + n \frac{e^{-\theta}}{(1 - e^{-\theta})^2}.$$

In Programwindow 5.2 R code for performing the Newton Raphson algorithm is given. Using this code with starting value $\theta^{(0)} = 1.5$ gave $\widehat{\theta} = 0.8925$ after 6 iterations. On the other hand, starting at $\theta^{(0)} = 2.0$ or larger, the procedure *diverged*, demonstrating the non-robustness of the Newton-Raphson algorithm.

Turn now to the scoring algorithm. The expectation in the truncated Poisson distribution is $\theta/(1-e^{-\theta})$, which gives the Fisher information

$$I(\theta) = \frac{n}{(1-e^{-\theta})}\left[\frac{1}{\theta} - \frac{e^{-\theta}}{1-e^{-\theta}}\right].$$

In Programwindow 5.3, R code for performing the scoring algorithm is given. Note that the only change from the Newton-Raphson algorithm is the replacement of $J(\theta)$ with $I(\theta)$. Starting at $\theta^{(0)} = 1.5$, only 4 iterations were needed for convergence to $\widehat{\theta} = 0.8925$. For $\theta^{(0)} = 2.0$, convergence was also obtain, now after 6 iterations. Convergence was also obtained for all other values tried out.

Inserting $\widehat{\theta}$, we obtain $I(\widehat{\theta}) = 1750.8$ giving an approximate standard error for $\widehat{\theta}$ equal to 0.0239. □

```
scoring.trpo = function(x,theta0,eps=0.000001)
 {
  n = sum(x);i = 1:6;sumx = sum(x*i)
  theta = theta0;diff = 1
  while(diff> eps)
   {
    theta.old = theta
    s = sumx/theta-n/(1-exp(-theta))
    Ibar = n*(1/theta-exp(-theta)/(1-exp(-theta)))/(1-exp(-theta))
    theta = theta+s/Ibar
    diff = abs(theta-theta.old)
  }
  list(theta,Ibar)
}
```

Programwindow 5.3: R *routine for estimation in the truncated Poisson distribution using the Fisher scoring algorithm.*

6 Modifications of the Newton-Raphson and the scoring algorithms

Direct use of the Newton-Raphson or the Fisher scoring algorithm can in many cases be problematic. In this section we will discuss three possibilities for improving these algorithms, reduction of the optimization problem to a smaller dimension, reparametrization and smaller jumps.

6.1 Dimension reduction

For some maximum likelihood problems where full analytical solutions are impossible, some of the parameters can be partly found as functions of others. We will illustrate this through the leukemia data example.

Example 3 (Leukemia data, cont.)
As discussed before, the Newton-Raphson algorithm is very sensitive to starting values for this problem. Further, using the scoring algorithm is problematic because of difficulties in calculation of the expected information matrix. Inspecting the scoring function (1.5), we see however that solving the equation $s_1(\boldsymbol{\theta}) = 0$ with respect to α (keeping β fixed), we obtain

$$\widehat{\alpha}(\beta) = [\frac{1}{n}\sum_{i=1}^{n} x_i^{\beta}]^{1/\beta}.$$

Here we have used the notation $\widehat{\alpha}(\beta)$ to make it explicit that the optimal value of α depends on the valu of β. Now insert this solution for α into the log-likelihood function (1.4) to obtain:

$$\begin{aligned} l_\beta(\beta) =& l(\widehat{\alpha}(\beta), \beta) \\ =& n\log(\beta) - n\beta\log([\frac{1}{n}\sum_{i=1}^{n} x_i^{\beta}]^{1/\beta}) + (\beta-1)\sum_{i=1}^{n}\log(x_i) - \frac{\sum_{i=1}^{n} x_i^{\beta}}{\frac{1}{n}\sum_{i=1}^{n} x_i^{\beta}} \\ =& n\log(\beta) - n\log([\frac{1}{n}\sum_{i=1}^{n} x_i^{\beta}]) + (\beta-1)\sum_{i=1}^{n}\log(x_i) - n. \end{aligned} \tag{6.2}$$

Note that the likelihood function has been reduced to a function in *one* variable, which is much easier to maximize. The partially maximized log-likelihood function $l_\beta(\beta)$ is called the *profile log-likelihood* function and is plotted in Figure 6.2.

Maximization of $l_\beta(\beta)$ can now be performed by a Newton-Raphson algorithm. We have

$$\begin{aligned} l'_\beta(\beta) =& \frac{n}{\beta} - n\frac{\sum_{i=1}^{n} x_i^{\beta}\log(x_i)}{\sum_{i=1}^{n} x_i^{\beta}} + \sum_{i=1}^{n}\log(x_i) \\ l''_\beta(\beta) =& -\frac{n}{\beta^2} - n\frac{\sum_{i=1}^{n} x_i^{\beta}\log(x_i)^2 \sum_{i=1}^{n} x_i^{\beta} - (\sum_{i=1}^{n} x_i^{\beta}\log(x_i))^2}{(\sum_{i=1}^{n} x_i^{\beta})^2} \end{aligned}$$

Programwindow 6.1 shows R code for the Newton-Raphson. A run of this algorithm with $\beta^{(0)} = 1.0$ and convergence criterion

$$|\beta^{(s)} - \beta^{(s-1)}| < 0.000001,$$

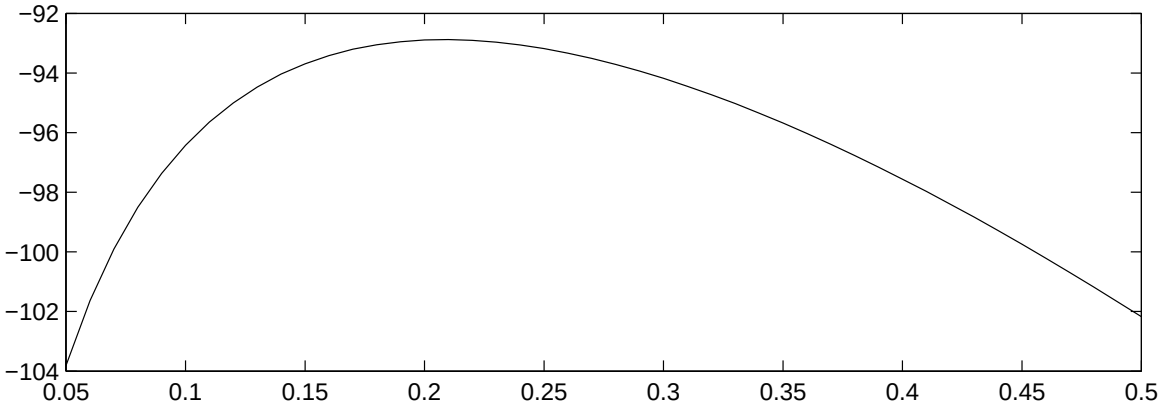

Figure 6.2: *Profile log-likelihood* $l_\beta(\beta)$ *given in* (6.2) *for the leukemia data.*

gave the following results:

Iteration s	$\alpha^{(s)}$	$\beta^{(s)}$	$l_\beta(\beta^{(s)})$
0	17.93750	1.0000000	-62.19030
1	23.32840	0.9165698	-66.60703
2	16.87444	0.9218605	-61.81284
3	17.20042	0.9218849	-62.09486
4	17.20194	0.9218849	-62.09617
5	17.20194	0.9218849	-62.09617

which is the same results obtained in Section 4, but convergence was obtained with fewer iterations.

Although a more efficient algorithm can be constructed in this way, the algorithm is still sensitive to starting values. Starting with $\beta^{(0)} = 2.0$, the following happened:

Iteration s	$\alpha^{(s)}$	$\beta^{(s)}$	$l_\beta(\beta^{(s)})$
0	26.6118	2.0000000	-73.6808
1	2.370984e-10	-0.2961171	NaN

It is possible to show that $l''_\beta(\beta) \leq 0$ for all values of β. Nevertheless, the modified Newton-Raphson algorithm (6.5) fail for some starting values. The reason in this case is the constraint on the parameter β (it needs to be positive). Running the Newton-Raphson, at the first iteration β becomes negative, and the algorithm just gives meaningless results. How to handle constraints will be discussed in the next subsection. □

6.2 Reparametrization

Many of the parameters involved have restrictions on their values. For the gamma distributed data discussed in example 2, both parameters needs to be positive. The same is

```
nr.profile.weibull = function(x,beta0,eps=0.00001)
{
 n = length(x);sumlogx = sum(log(x));
 diff = 1;beta = beta0
 while(diff>eps)
  {
   beta.old = beta
   w1 = sum(x^beta)
   w2 = sum((x^beta)*log(x))
   w3 = sum((x^beta)*log(x)*log(x))
   w4 = sum(log(x))
   l1 = n/beta-n*w2/w1+w4
   l2 = -n/(beta^2)-n*(w3*w1-w2^2)/(w1^2)
   beta = beta.old - l1/l2;
   diff = abs(beta-beta.old)
  }
 alpha = (w1/n)^(1/beta)
 list(alpha=alpha,beta=beta)
}
```

Programwindow 6.1: R *routine for optimization of the profile likelihood function* $l_\beta(\beta)$ *given in* (6.2) *with respect to* β *using the Newton-Raphson algorithm.*

true for the Weibull data considered in example 3. For both the Newton-Raphson algorithm and the Fisher scoring algorithm, negative values can occur at some iterations, and the procedures can break down. The basic idea in such cases is to use *reparametrization*.

Example 3 (Leukemia data, cont.)
Continuing on the profile likelihood discussed in the previous subsection, we will now see how the constraint $\beta > 0$ can be taken into consideration.

Define $b = \log(\beta)$. Since $\beta > 0$, b can take values on the whole real line. Rewriting $l_\beta(\beta)$ as a function of b, we get

$$l_b(b) = nb - n\log([\frac{1}{n}\sum_{i=1}^{n} x_i^{e^b}]) + (e^b - 1)\sum_{i=1}^{n}\log(x_i) - n. \tag{6.3}$$

Such a transformation of β to b we call a *reparametrization* of β. We aim at maximizing $l_b(b)$ with respect to b. Note that since $\log(\beta)$ is a monotone and invertible transformation, a maximum point for b directly gives a maximum value for β. Since $\beta = \exp(b)$, we automatically obtain $\beta > 0$.

To use Newton-Raphson, we need the derivatives:

$$
\begin{aligned}
l_b'(b) =& n\left[1 - \frac{\frac{1}{n}\sum_{i=1}^n x_i^{e^b} e^b \log(x_i)}{\frac{1}{n}\sum_{i=1}^n x_i^{e^b}} + e^b \frac{1}{n}\sum_{i=1}^n \log(x_i)\right] \\
l_b''(b) =& -ne^b\left[\frac{\frac{1}{n}\sum_{i=1}^n x_i^{e^b}\log(x_i) - \frac{1}{n}\sum_{i=1}^n \log(x_i)\frac{1}{n}\sum_{i=1}^n x_i^{e^b}}{\frac{1}{n}\sum_{i=1}^n x_i^{e^b}}\right] \\
& -ne^{2b}\left[\frac{\frac{1}{n}\sum_{i=1}^n x_i^{e^b}[\log(x_i)]^2 \frac{1}{n}\sum_{i=1}^n x_i^{e^b} - [\frac{1}{n}\sum_{i=1}^n x_i^{e^b}\log(x_i)]^2}{[\frac{1}{n}\sum_{i=1}^n x_i^{e^b}]^2}\right] + e^b\sum_{i=1}^n \log(x_i)
\end{aligned}
$$

Programwindow 6.2 shows R code for a Newton-Raphson algorithm for maximizing $l_b(b)$ with respect to b.

Starting now with $\beta^{(0)} = 2.0$, the following happened:

Iteration s	$\alpha^{(s)}$	$\beta^{(s)}$	$l_\beta(\beta^{(s)})$
0	26.61179	2.000000	-73.68079
1	23.92023	1.662872	-68.31253
2	21.33190	1.366104	-64.63275
3	19.15131	1.129239	-62.71561
4	17.71581	0.976462	-62.14272
5	17.20004	0.921683	-62.09617
6	17.20225	0.921918	-62.09617
7	17.20189	0.921879	-62.09617
8	17.20195	0.921886	-62.09617
9	17.20194	0.921885	-62.09617
10	17.20194	0.921885	-62.09617

Also for more extreme starting values, convergence was obtained in this case. □

6.3 Smaller jumps

Formally, the Newton-Raphson algorithm only finds solutions to the equations

$$l'(\boldsymbol{\theta}) = 0 \tag{6.4}$$

which is equal to the scoring equations (3.4). From a pure numerical point of view, optimization through searching for solutions of (6.4) is not recommended. By only using the derivatives of the function, and not the function itself, the algorithm may converge to any stationary point, not distinguishing between maxima and minima. However, if the function

l is concave and unimodal, the Newton-Raphson algorithm can be slightly modified such that convergence can be guaranteed. Consider the iterations

$$\boldsymbol{\theta}^{(s+1)} = \boldsymbol{\theta}^{(s)} - \delta^{(s)} \left[\boldsymbol{l}''(\boldsymbol{\theta}^{(s)}) \right]^{-1} \boldsymbol{l}'(\boldsymbol{\theta}^{(s)}) \tag{6.5}$$

where $\delta^{(s)} \leq 1$. At each iterations, start with $\delta^{(s)} = 1$. If $l(\theta^{(s+1)}) < l(\theta^{(s)})$, divide $\delta^{(s)}$ by two, and use (6.5) again. Repeat this until $l(\theta^{(s+1)}) \geq l(\theta^{(s)})$. Because l is concave, it is always possible to find an $\delta^{(s)}$ small enough such that this will be satisfied. In many situations of statistics, the log-likelihood function satisfy this property, making such a procedure operational.

A similar modification can be made for the scoring algorithm:

$$\boldsymbol{\theta}^{(s+1)} = \boldsymbol{\theta}^{(s)} + \delta^{(s)} \boldsymbol{I}^{-1}(\boldsymbol{\theta}^{(s)}) \boldsymbol{s}(\boldsymbol{\theta}^{(s)}). \tag{6.6}$$

Since $\boldsymbol{I}(\boldsymbol{\theta}^{(s)})$ is positive definite, such an algorithm is guaranteed to converge to a (local) maxima, even if $l(\boldsymbol{\theta})$ is not concave.

Example 2 (Rainfall data, cont.)
For this example, $\boldsymbol{J}(\boldsymbol{\theta}) = \boldsymbol{I}(\boldsymbol{\theta})$, implying that $\boldsymbol{J}(\boldsymbol{\theta})$ is positive (semi-) definite while $l''(\boldsymbol{\theta})$ is negative (semi-) definite. An implementation of a Newton-Raphson algorithm (6.5) allowing for smaller jumps is given in Programwindow 6.3. In order to make negative values of the parameters illegal, we put the likelihood value in such cases very small (a better implementation would be to reparametrize, as described in section 6.2).

Running this algorithm with $\alpha^{(0)} = \lambda^{(0)} = 1$, convergence now was reached after seven iterations. Note that a smaller δ was only necessary in the first iteration. When $(\alpha^{(s)}, \lambda^{(s)})$ gets closer to the optimal value, no modification of the ordinary Newton-Raphson algorithm is necessary.

Iteration s	$\alpha^{(s)}$	$\beta^{(s)}$	$l(\boldsymbol{\theta}^{(s)})$	$\delta^{(s)}$
0	1.0000000	1.000000	-50.9370	
1	0.3841504	0.578052	146.8740	0.25
2	0.3518249	0.912280	171.4139	1.00
3	0.4019210	1.423652	182.4078	1.00
4	0.4319168	1.822000	185.1676	1.00
5	0.4402049	1.954300	185.3468	1.00
6	0.4407877	1.964326	185.3477	1.00
7	0.4407914	1.964380	185.3477	1.00

□

```
nr.profile.weibull2 = function(x,beta0,eps=0.000001)
 {
  n = length(x);sumlogx = sum(log(x))
  diff = 1;b = log(beta0)
  beta = exp(b)
  w1 = sum(x^beta)
  l = n*log(beta)-n*log(w1/n)+(beta-1)*sumlogx-n
  alpha = (w1/n)^(1/beta)
  it = 0
  while(diff>eps)
   {
    it = it+1
    b.old = b
    w1 = sum(x^(exp(b)))
    w2 = sum((x^(exp(b))*log(x)))
    w3 = sum((x^(exp(b))*(log(x))^2))
    l1 = n-n*exp(b)*w2/w1+exp(b)*sumlogx
    l2 = -n*exp(2*b)*(w2+(w3*w1-w2*w2)/w1)/w1 +
         exp(b)*sumlogx
    b = b - l1/l2
    diff = sum(abs(b-b.old))
    beta = exp(b)
    diff = abs(beta-exp(b.old))
    w1 = sum(x^beta)
    alpha = (w1/n)^(1/beta)
    l = n*log(beta)-n*log(w1/n)+(beta-1)*sumlogx-n
  }
  alpha = (w1/n)^(1/beta)
  list(alpha=alpha,beta=beta)
}
```

Programwindow 6.2: R *routine for optimization of the profile likelihood function* $l_b(b)$ *given in* (6.3) *with respect to* b *using the Newton-Raphson algorithm.*

```
nr.gamma.mod = function(x,theta0=NULL,eps=0.000001)
 {
  n = length(x);sumx = sum(x);sumlogx = sum(log(x))
  h = 0.0000001;diff = 1
  if(is.null(theta0))
    {alpha = mean(x)^2/var(x);lambda=mean(x)/var(x)}
  else
    {alpha = theta0[1];lambda=theta0[2]}
  theta = c(alpha,lambda)
  l = n*alpha*log(lambda)+(alpha-1)*sumlogx-
      lambda*sumx-n*log(gamma(alpha))
  while(diff>eps)
   {
    theta.old = theta;l.old = l
    g = gamma(alpha)
    dg = (gamma(alpha+h)-gamma(alpha))/h
    d2g = (gamma(alpha+2*h)-2*gamma(alpha+h)+
           gamma(alpha))/h^2
    s = c(n*log(lambda)+sumlogx-n*dg/gamma(alpha),
          n*alpha/lambda-sumx)
    Jbar = -matrix(c(-n*(d2g*g-dg^2)/g^2,n/lambda,
                     n/lambda,-n*alpha/lambda^2),ncol=2)
    l = l.old-1;delta = 1
    while(l < l.old)
      {
       theta = theta.old + delta*solve(Jbar,s)
       alpha = theta[1];lambda = theta[2]
       if ((alpha < 0) || (lambda < 0))
          l = -9999999
       else
         {
          l = n*alpha*log(lambda)+(alpha-1)*sumlogx-
              lambda*sumx-n*log(gamma(alpha))
        }
       delta = delta/2
     }
    diff = sum(abs(theta-theta.old))
   }
 list(alpha,lambda,Jbar)
}
```

Programwindow 6.3: R *code for running Newton-Raphson on rainfall data.*

7 Non-linear regression

In this section we will consider maximum likelihood estimation in non-linear regression models. The general formulation of the model will be

$$Y_i = g(\boldsymbol{x}_i, \boldsymbol{\beta}) + e_i, \quad i = 1, ..., n$$

where $\boldsymbol{x}_i$ is a vector of explanatory variables, $\boldsymbol{\beta}$ is a p-dimensional vector of unknown regression parameters and e_i is a noise term. We will make the standard assumptions about these noise terms:

(*a*) $\mathsf{E}[e_i] = 0$.

(*b*) $\mathsf{Var}[e_i] = \sigma^2$.

(*c*) $e_1, ..., e_n$ are uncorrelated.

(*d*) The e_i's are normally distributed.

Multiple linear regression is the special case where

$$g(\boldsymbol{x}_i, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_{i,1} + \cdots + \beta_{p-1} x_{i,p-1}.$$

We will however in this section allow for nonlinear g functions.

Assume that $\{(y_i, \boldsymbol{x}_i), i = 1, ..., n\}$ are observed (y_i is the observed value of Y_i). Under the assumptions above, the likelihood function is given by

$$\begin{aligned} L(\boldsymbol{\beta}, \sigma^2) =& \prod_{i=1}^{n} f(y_i; \boldsymbol{x}_i, \boldsymbol{\beta}, \boldsymbol{\sigma}^2) \\ =& \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2} \end{aligned}$$

while the log-likelihood is

$$\begin{aligned} l(\boldsymbol{\beta}, \sigma^2) =& \sum_{i=1}^{n} [-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}(y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2] \\ =& -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2. \end{aligned} \tag{7.1}$$

not possible to obtain, and numerical methods have to be applied. For notational simplicity, define

$$g'_k(\boldsymbol{x}_i, \boldsymbol{\beta}) = \frac{\partial}{\partial \beta_k} g(\boldsymbol{x}_i, \boldsymbol{\beta}) \tag{7.2}$$

and

$$g''_{k,l}(\boldsymbol{x}_i, \boldsymbol{\beta}) = \frac{\partial^2}{\partial\beta_k \partial\beta_l} g(\boldsymbol{x}_i, \boldsymbol{\beta}). \tag{7.3}$$

The partial derivatives of $l(\boldsymbol{\beta}, \sigma^2)$ with respect to $\boldsymbol{\beta}$ and σ^2 are then given by the score function $\boldsymbol{s}(\boldsymbol{\beta}, \sigma^2)$ with elements

$$\begin{aligned}
\boldsymbol{s}_k(\boldsymbol{\beta}, \sigma^2) =& \frac{\partial}{\partial\beta_k} l(\boldsymbol{\beta}, \sigma^2) \\
=& \frac{1}{\sigma^2} \sum_{i=1}^{n} (y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta})) g'_k(\boldsymbol{x}_i, \boldsymbol{\beta}), && (7.4) \\
\boldsymbol{s}_{p+1}(\boldsymbol{\beta}, \sigma^2) =& \frac{\partial}{\partial\sigma^2} l(\boldsymbol{\beta}, \sigma^2) \\
=& -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_{i=1}^{n} (y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2, && (7.5) \\
& && (7.6)
\end{aligned}$$

and the observed information matrix $\boldsymbol{J}(\boldsymbol{\beta}, \sigma^2)$ with elements

$$\begin{aligned}
J_{k,l}(\boldsymbol{\beta}, \sigma^2) =& -\frac{\partial^2}{\partial\beta_k \partial\beta_l} l(\boldsymbol{\beta}, \sigma^2) \\
=& \frac{1}{\sigma^2} \sum_{i=1}^{n} [g'_k(\boldsymbol{x}_i, \boldsymbol{\beta}) g'_l(\boldsymbol{x}_i, \boldsymbol{\beta}) - (y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta})) g''_{k,l}(\boldsymbol{x}_i, \boldsymbol{\beta})], && (7.7) \\
J_{k,p+1}(\boldsymbol{\beta}, \sigma^2) =& -\frac{\partial^2}{\partial\beta_k \partial\sigma^2} l(\boldsymbol{\beta}, \sigma^2) \\
=& \frac{1}{\sigma^4} \sum_{i=1}^{n} (y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta})) g'_k(\boldsymbol{x}_i, \boldsymbol{\beta}), && (7.8) \\
J_{p+1,p+1}(\boldsymbol{\beta}, \sigma^2) =& -\frac{\partial^2}{\partial\sigma^2 \partial\sigma^2} l(\boldsymbol{\beta}, \sigma^2) && (7.9) \\
=& -\frac{n}{2\sigma^4} + \frac{1}{\sigma^6} \sum_{i=1}^{n} (y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2, && (7.10)
\end{aligned}$$

where $k, l = 1, ..., p$. These quantities can be directly imputed into the general Newton-Raphson algorithm (4.7).

A more efficient algorithm can be obtained by utilizing that for given $\boldsymbol{\beta}$, an analytical expression for the maximum likelihood estimate $\widehat{\sigma}^2$ for σ^2 can be obtained. From (7.1),

$$\frac{\partial}{\partial\sigma^2} l(\boldsymbol{\beta}, \sigma^2) = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_{i=1}^{n} (y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2$$

and the solution $\widehat{\sigma}^2$ to the equation $\frac{\partial}{\partial \sigma^2} l(\boldsymbol{\beta}, \sigma^2) = 0$ is given by

$$\widehat{\sigma}^2(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2. \tag{7.11}$$

Inserting this into (7.1), we obtain the profile log-likelihood

$$\begin{aligned} l_{\boldsymbol{\beta}}(\boldsymbol{\beta}) =& l(\boldsymbol{\beta}, \widehat{\sigma}^2(\boldsymbol{\beta})) \\ =& -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\frac{1}{n}\sum_{i=1}^{n}(y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2) - \frac{n}{2}. \end{aligned}$$

Maximizing $l_{\boldsymbol{\beta}}(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ is equivalent to minimizing

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}))^2,$$

showing that similarly to ordinary linear regression, maximizing the likelihood is equivalent to least squares estimation when we assume normal error terms. A Newton-Raphson algorithm for minimizing $S(\boldsymbol{\beta})$ can be directly constructed.

An alternative modification is to replace the observed information matrix $\boldsymbol{J}(\boldsymbol{\beta}, \sigma)$ by its expectation, that is the Fisher information matrix $\boldsymbol{I}(\boldsymbol{\beta}, \sigma)$, similar to the approach in section 5. Replacing y_i by Y_i in (7.7)-(7.10) and using that $E[Y_i] = g(\boldsymbol{x}_i, \boldsymbol{\beta})$, we get

$$I_{k,l}(\boldsymbol{\beta}, \sigma) = \frac{1}{\sigma^2} \sum_{i=1}^{n} g_k'(\boldsymbol{x}_i, \boldsymbol{\beta}) g_l'(\boldsymbol{x}_i, \boldsymbol{\beta}), \tag{7.12}$$

$$I_{k,p+1}(\boldsymbol{\beta}, \sigma) = 0, \tag{7.13}$$

$$I_{p+1,p+1}(\boldsymbol{\beta}, \sigma) = \frac{n}{2\sigma^4} \tag{7.14}$$

for $k, l = 1, ..., p$. It can be shown that the $\boldsymbol{I}(\boldsymbol{\beta}, \sigma)$ *is* negative definite. By using this matrix instead of $\boldsymbol{J}(\boldsymbol{\beta}, \sigma)$ in the Newton-Raphson algorithm we obtain the scoring algorithm. Both the Newton-Raphson algorithm and the scoring algorithm are easy to implement. Note however that the scoring version is somewhat simpler since it only involves first derivatives. Further, defining $\boldsymbol{\theta} = (\boldsymbol{\beta}, \sigma^2)$, $\boldsymbol{s}_{\boldsymbol{\beta}}(\boldsymbol{\beta}, \sigma^2)$ to be the first p elements of $\boldsymbol{s}(\boldsymbol{\beta}, \sigma)$, $s_{\sigma^2}(\boldsymbol{\beta}, \sigma^2)$ to be the last element of$\boldsymbol{s}(\boldsymbol{\beta}, \sigma)$, $\boldsymbol{I}_{\boldsymbol{\beta}}(\boldsymbol{\beta}, \sigma^2)$ to be the upper left $p \times p$ submatrix of $\boldsymbol{I}(\boldsymbol{\beta}, \sigma)$ and $I_{\sigma^2}(\boldsymbol{\beta}, \sigma^2)$ to be the $(p+1, p+1)$ element of $\boldsymbol{I}(\boldsymbol{\beta}, \sigma)$, the scoring algorithm update can be written as

$$\begin{aligned} \boldsymbol{\theta}^{s+1} =& \begin{bmatrix} \boldsymbol{\beta}^{s+1} \\ (\sigma^2)^{s+1} \end{bmatrix} \\ =& \boldsymbol{\theta}^s + \begin{bmatrix} \boldsymbol{I}_{\boldsymbol{\beta}}(\boldsymbol{\beta}, \sigma^2) & \boldsymbol{0} \\ \boldsymbol{0} & I_{\sigma^2}(\boldsymbol{\beta}, \sigma^2) \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{s}_{\boldsymbol{\beta}}(\boldsymbol{\beta}, \sigma^2) \\ s_{\sigma^2}(\boldsymbol{\beta}, \sigma^2) \end{bmatrix} \\ =& \begin{bmatrix} \boldsymbol{\beta}^s + \boldsymbol{I}_{\boldsymbol{\beta}}(\boldsymbol{\beta}, \sigma^2)^{-1} \boldsymbol{s}_{\boldsymbol{\beta}}(\boldsymbol{\beta}, \sigma^2) \\ (\sigma^2)^s + I_{\sigma^2}(\boldsymbol{\beta}, \sigma^2)^{-1} s_{\sigma^2}(\boldsymbol{\beta}, \sigma^2) \end{bmatrix} \end{aligned}$$

Noticing now that $\boldsymbol{s}_{\boldsymbol{\beta}}(\boldsymbol{\beta}, \sigma^2)$ and $\boldsymbol{I}_{\boldsymbol{\beta}}(\boldsymbol{\beta}, \sigma^2)$ only depend on σ^2 through the common factor $\frac{1}{2\sigma^2}$, we see that the updating of $\boldsymbol{\beta}$ is independent of σ^2, a reasonable property since as noted above the optimal value of $\boldsymbol{\beta}$ is unaffected of σ^2.

It can further be shown that at any iteration,

$$(\sigma^2)^{s+1} = \frac{1}{n}\sum_{i=1}^{n}(y_i - g(\boldsymbol{x}_i, \boldsymbol{\beta}^s))^2,$$

corresponding to the optimal solution given in (7.11). In practice this means hat we only need to update $\boldsymbol{\beta}$ through the scoring algorithm, and after convergence, the estimate of σ^2 can be obtaind directly through (7.11).

As usual, uncertainty estimates of our estimates are of interest, and can be obtained through the information matrices. Either the observed or the expected (Fisher) information matrices can be used.

Using the Fisher information matrix, for large sample sizes, the covariance matrix for $(\widehat{\boldsymbol{\beta}}, \widehat{\sigma}^2)$ is given by

$$\boldsymbol{I}^{-1}(\boldsymbol{\beta}, \sigma^2) = \begin{bmatrix} \sigma^2 \boldsymbol{I}_{\boldsymbol{\beta}}^{-1}(\boldsymbol{\beta}) & \boldsymbol{0} \\ \boldsymbol{0} & \frac{2\sigma^4}{n} \end{bmatrix} \tag{7.15}$$

The $\boldsymbol{0}$ part on the off-diagonal of $\boldsymbol{I}^{-1}(\boldsymbol{\beta}, \sigma^2)$ imply that $\widehat{\boldsymbol{\beta}}$ and $\widehat{\sigma}^2$ are independent for large sample sizes.

Example 5 (Weight loss programme)
Venables and Ripley (1999) contain a dataset (originally from Dr. T Davies) describing weights (y_i) of obese patients after different number of days (x_i) since start of a weight reduction programme. The data, also available from the course home page, is plotted in Figure 7.1. Venables and Ripley (1999) suggests the following model for this dataset:

$$y_i = \beta_0 + \beta_1 e^{-\beta_2 x_i} + e_i.$$

So $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2)$ contains 3 unknown regression parameters in this case. In order to implement the Newton-Raphson or scoring algorithm, we need the derivatives of the g-function. We have

$$g'(x_i, \boldsymbol{\beta}) = \begin{pmatrix} 1 & e^{-\beta_2 x_i} & -\beta_1 e^{-\beta_2 x_i} x_i \end{pmatrix}$$
$$g''(x_i, \boldsymbol{\beta}) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -e^{\beta_2 x_i} x_i \\ 0 & -e^{-\beta_2 x_i} x_i & \beta_1 e^{-\beta_2 x_i} x_i^2 \end{pmatrix}$$

In Programwindow 7.1 R code for maximum likelihood estimation based on Newton-Raphson is given. Running this algorithm with start values given in the first row of the table below, convergence was reached after 7 iterations.

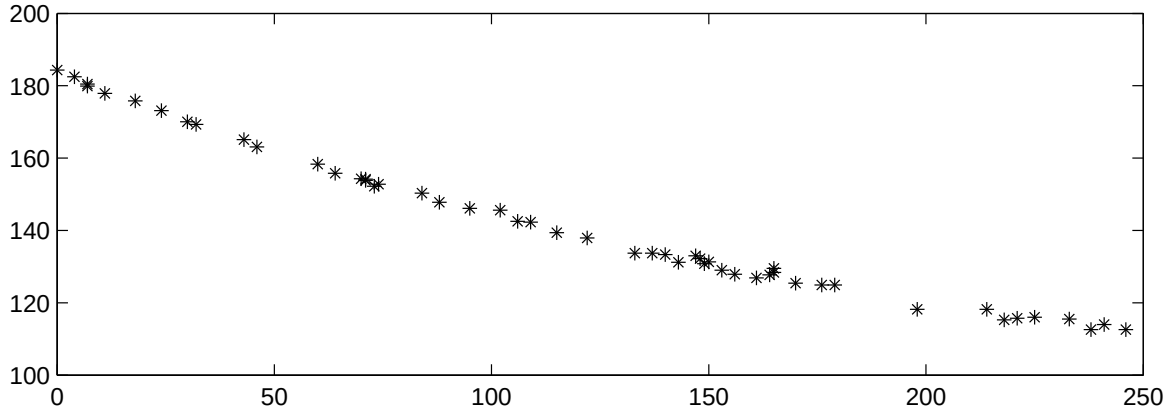

Figure 7.1: *Weight loss from an obese patient*

Iteration s	$\beta_0^{(s)}$	$\beta_1^{(s)}$	$\beta_2^{(s)}$	$(\sigma^2)^{(s)}$	$l(\boldsymbol{\beta}^{(s)}, (\sigma^2)^{(s)})$
0	90.00000	95.0000	0.005000000	209.38680	-143.25920
1	84.33882	100.5511	0.005199136	0.7286630	-69.66973
2	76.34951	107.1314	0.004415786	1.4737980	-78.82676
3	76.80056	106.8413	0.004541713	0.6565196	-68.31437
4	81.66417	102.3930	0.004880670	0.6063372	-67.28066
5	81.39949	102.6619	0.004886570	0.5695842	-66.46777
6	81.37380	102.6841	0.004884399	0.5695808	-66.46769
7	81.37382	102.6841	0.004884401	0.5695808	-66.46769

In Programwindow 7.2 R code for maximum likelihood estimation based on scoring algorithm is given. Running this algorithm with start values given in the first row of the table below, convergence was reached after 4 iterations.

Iteration s	$\beta_0^{(s)}$	$\beta_1^{(s)}$	$\beta_2^{(s)}$	$(\sigma^2)^{(s)}$	$l(\boldsymbol{\beta}^{(s)}, (\sigma^2)^{(s)})$
0	90.00000	95.0000	0.005000000	209.3868	-143.25920
1	81.40014	102.6569	0.004875966	0.5758058	-66.60900
2	81.37434	102.6836	0.004884439	0.5695808	-66.46769
3	81.37381	102.6841	0.004884401	0.5695808	-66.46769
4	81.37382	102.6841	0.004884401	0.5695808	-66.46769

In Table 7.1 standard errors based on large sample approximations (first row) is given. An alternative method for estimating the variability of the parameter estimates is bootstrapping. We will consider bootstrapping in the conditional inference setting. This means that we consider the explanatory variables to be fixed while the randomness appears from the noise terms. Bootstrap samples of $Y_1, ..., Y_n$ can be obtained by

$$Y_i^* = g(\boldsymbol{x}_i, \widehat{\boldsymbol{\beta}}) + e_i^*, \quad i = 1, ..., n$$

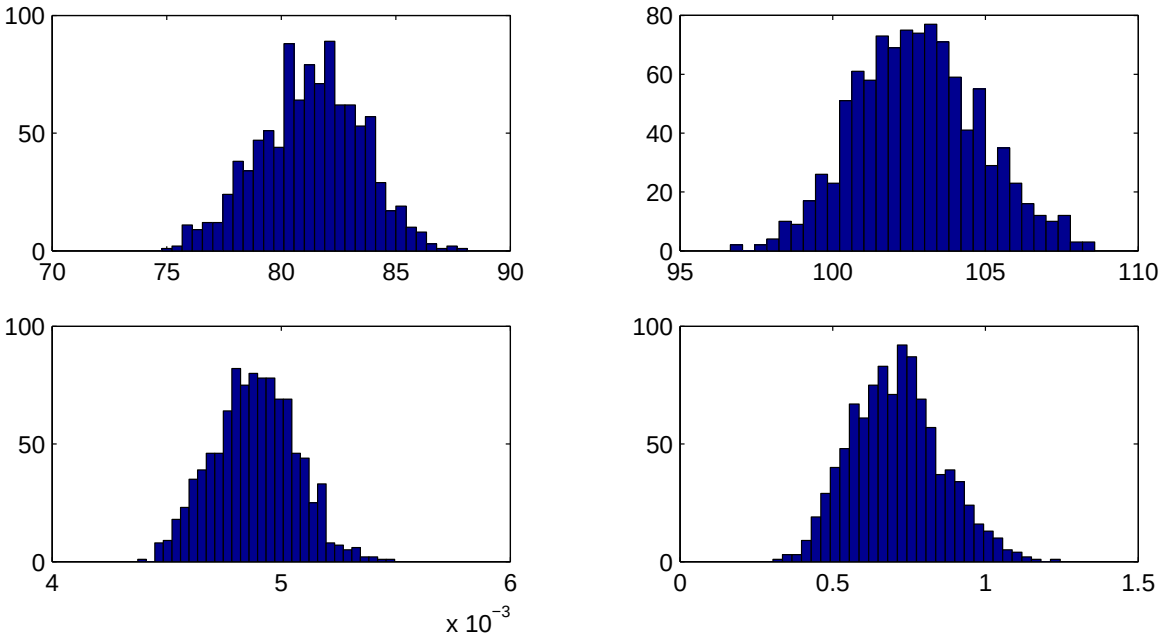

Figure 7.2: *Histogram of bootstrap simulations of* $\widehat{\beta}_0$ *(upper left),* $\widehat{\beta}_1$ *(upper right),* $\widehat{\beta}_2$ *(lower left) and* $\widehat{\sigma^2}$ *(lower right) for weight loss data.*

Parameter	$\mathsf{Std}[\widehat{\beta}_0]$	$\mathsf{Std}[\widehat{\beta}_1]$	$\mathsf{Std}[\widehat{\beta}_2]$	$\mathsf{Std}[\widehat{\sigma}^2]$
Large sample approx	2.5354	2.3273	0.00018	0.1480
Fixed x, Parametric bootstrapping	2.1880	2.0118	0.00017	0.1473
Random x, Nonparametric bootstrapping	2.1800	1.9927	0.00018	0.1502
Fixed x, Nonparametric bootstrapping	2.2760	2.0776	0.00018	0.1533

Table 7.1: *Standard errors based on large sample approximation, parametric and nonparametric bootstrapping for weight loss data.*

where $\boldsymbol{e}^* = (e_1^*, ..., e_n^*)$ are bootstrap samples of the noise terms. Two main alternatives for sampling $\boldsymbol{e}^*$ is possible. In *parametric* bootstrapping, the model assumption $e_i \sim N(0, \sigma^2)$ is used and we simulate e_i^* through $e_i^* \sim N(0, s^2)$. For *nonparametric* bootstrapping, the normal assumption is relaxed, and the e_i^*'s are samples from an estimate of the distribution for e_i. This can be peformed by sampling $e_1^*, ..., e_n^*$ from $(e_1, ..., e_n)$ *with replacement*.

The second and third rows in Table 7.1 shows standard errors estimated by parametric and non-parametric bootstrapping. Figure 7.2 shows histograms of the 1000 nonparametric bootstrap simulations. They all are close to normal distributions, confirming the large sample theory and also explaining the similarities of the standard errors obtained from the two methods. The parametric simulations (not shown) looks very similar. Programwindow 7.3 shows R code for performing parametric and nonparametric bootstrap simulations.

□

```
nr.weight = function(x,y,beta.start,eps=0.000001)
 {
  #Note: beta[i+1] correspond to beta_i
  n = length(x)
  diff = 1;beta = beta.start
  y.pred = beta[1]+beta[2]*exp(-beta[3]*x)
  res = y-y.pred
  while(diff>eps)
   {
    beta.old = beta
    s = c(sum(y-y.pred),
          sum((res)*exp(-beta[3]*x)),
          -beta[2]*sum((res)*exp(-beta[3]*x)*x))
    Jbar =
      matrix(c(n,sum(exp(-beta[3]*x)),-beta[2]*sum(exp(-beta[3]*x)*x),
               sum(exp(-beta[3]*x)),sum(exp(-2*beta[3]*x)),
          sum((res-beta[2]*exp(-beta[3]*x))*exp(-beta[3]*x)*x),
           -beta[2]*sum(exp(-beta[3]*x)*x),
       sum((res-beta[2]*exp(-beta[3]*x))*exp(-beta[3]*x)*x),
       -beta[2]*sum((res-beta[2]*exp(-beta[3]*x))*
                           exp(-beta[3]*x)*x*x)),ncol=3)
    beta = beta.old + solve(Jbar,s)
    y.pred = beta[1]+beta[2]*exp(-beta[3]*x)
    res = y-y.pred
    diff = sum(abs(beta-beta.old))
   }
 sigma = mean((res)^2)
 list(beta=beta,sigma=sigma,Jbar=Jbar)
}
```

Programwindow 7.1: R *code for running Newton-Raphson on weight loss data.*

```
scoring.weight = function(x,y,beta.start,eps=0.000001)
 {
  #Note: beta[i+1] correspond to beta_i
  n = length(x)
  diff = 1;beta = beta.start
  y.pred = beta[1]+beta[2]*exp(-beta[3]*x)
  while(diff>eps)
   {
    beta.old = beta
    s = c(sum(y-y.pred),
          sum((y-y.pred)*exp(-beta[3]*x)),
          -beta[2]*sum((y-y.pred)*exp(-beta[3]*x)*x))
    Ibar = matrix(c(n,sum(exp(-beta[3]*x)),
                    -beta[2]*sum(exp(-beta[3]*x)*x),
                    sum(exp(-beta[3]*x)),sum(exp(-2*beta[3]*x)),
             -beta[2]*sum(exp(-2*beta[3]*x)*x),
               -beta[2]*sum(exp(-beta[3]*x)*x),
               -beta[2]*sum(exp(-2*beta[3]*x)*x),
             beta[2]*beta[2]*sum(exp(-2*beta[3]*x)*x*x)),ncol=3)
    beta = beta.old + solve(Ibar,s)
    y.pred = beta[1]+beta[2]*exp(-beta[3]*x)
    diff = sum(abs(beta-beta.old))
   }
 sigma = mean((y-y.pred)^2)
 list(beta=beta,sigma=sigma,Ibar=Ibar)
}
```

Programwindow 7.2: R *code for running the scoring algorithm on weight loss data.*

```
n = dim(wtloss)[1]
fit = scoring.weight(wtloss$days,wtloss$weight,c(90,95,0.005))
beta = fit$beta
sigma = fit$sigma
ypred=beta[1]+beta[2]*exp(-beta[3]*wtloss$days)
res=wtloss$weight-ypred
B = 1000
beta.star = matrix(NA,nrow=B,ncol=3)
sigma.star = rep(NA,B)
for(b in 1:B)
 {
  res.star = sqrt(sigma)*rnorm(n)
  weight.star = ypred+res.star;
  fit = scoring.weight(wtloss$days,weight.star,beta);
  beta.star[b,] = fit$beta
  sigma.star[b] = fit$sigma
 }
```

```
n = dim(wtloss)[1]
fit = scoring.weight(wtloss$days,wtloss$weight,c(90,95,0.005))
beta = fit$beta
sigma = fit$sigma
ypred=beta[1]+beta[2]*exp(-beta[3]*wtloss$days)
res=wtloss$weight-ypred
B = 1000
beta.star = matrix(NA,nrow=B,ncol=3)
sigma.star = rep(NA,B)
for(b in 1:B)
 {
  res.star <- sample(res,n,replace=T)
  weight.star = ypred+res.star
  fit = scoring.weight(wtloss$days,weight.star,beta);
  beta.star[b,] = fit$beta
  sigma.star[b] = fit$sigma
 }
```

Programwindow 7.3: R *code for parametric (upper) and nonparametric (lower) bootstrapping on weight loss data.*

8 Logistic regression

In linear regression the response is usually assumed to be on a continuous scale. Many other types of responses do however exist, making the need for different regression methods. In Table 8.2, a data set is given where the response is whether a beetle given a dose of poison has died or not, i.e., a *binary* response. The explanatory variable is the amount of poison. The data are grouped since many beetles are given the same dose.

Dose	Number of insects	Numbers killed
1.6907	59	6
1.7242	60	13
1.7552	62	18
1.7842	56	28
1.8113	63	52
1.8369	59	53
1.8610	62	61
1.8839	60	60

Table 8.2: *The mortality of beetles against dose of poison.*

Assume Y_i is a binary response while x_i is a explanatory variable. In linear regression, the expected response is modeled as a linear function of the exploratory variable:

$$E[Y_i] = \beta_0 + \beta_1 x_i.$$

Note however that in the case of binary response, the expectation is equal to the probability for a beetle to die, a number between zero and one. The linear regression model is surely inappropriate since the expected value may vary from $-\infty$ to ∞.

In *logistic* regression, the response is modeled by

$$\begin{aligned} Y_i &\sim \text{binom}(1, p(x_i, \boldsymbol{\beta})), \\ p(x_i, \boldsymbol{\beta}) &= \frac{\exp\{\beta_0 + \beta_1 x_i\}}{1 + \exp\{\beta_0 + \beta_1 x_i\}}, \end{aligned} \tag{8.16}$$

By making the usual assumption that all observations are independent, the likelihood function becomes

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} p(x_i, \boldsymbol{\beta})^{y_i} (1 - p(x_i, \boldsymbol{\beta}))^{1-y_i}.$$

As usual, we consider the log-likelihood:

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{n} [y_i \log(p(x_i, \boldsymbol{\beta})) + (1 - y_i) \log(1 - p(x_i, \boldsymbol{\beta}))].$$

Elementary calculations show that the scoring function is equal to

$$\boldsymbol{s}(\boldsymbol{\beta}) = \begin{bmatrix} \frac{\partial l(\boldsymbol{\beta})}{\partial \beta_0} \\ \frac{\partial l(\boldsymbol{\beta})}{\partial \beta_1} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} [y_i - p(x_i, \boldsymbol{\beta})] \\ \sum_{i=1}^{n} [y_i - p(x_i, \boldsymbol{\beta})] x_i \end{bmatrix}$$

while the observed information matrix is given by

$$\boldsymbol{J}(\boldsymbol{\beta}) = \begin{bmatrix} \sum_{i=1}^{n} p(x_i, \boldsymbol{\beta})(1 - p(x_i, \boldsymbol{\beta})) & \sum_{i=1}^{n} p(x_i, \boldsymbol{\beta})(1 - p(x_i, \boldsymbol{\beta}))x_i \\ \sum_{i=1}^{n} p(x_i, \boldsymbol{\beta})(1 - p(x_i, \boldsymbol{\beta}))x_i & \sum_{i=1}^{n} p(x_i, \boldsymbol{\beta})(1 - p(x_i, \boldsymbol{\beta}))x_i^2 \end{bmatrix} \tag{8.17}$$

Note that $\boldsymbol{J}(\boldsymbol{\beta})$ do not depend on the random observations y_i, showing that the expected information matrix $\boldsymbol{I}(\boldsymbol{\beta})$ is equal to $\boldsymbol{J}(\boldsymbol{\beta})$. This implies that $\boldsymbol{J}(\boldsymbol{\beta})$ is always positive definite, making the log-likelihood function concave with only one (global) maxima.

In Programwindow 8.3, an R routine for optimizing the log-likelihood for the logistic model is given. Although the log-likelihood is unimodal, a modification of the ordinary Newton-Raphson algorithm allowing for smaller jumps (as described in section 6.3) is needed in order to make the algorithm robust towards starting values.

Example 6 (Beetle data)
To illustrate logistic regression, we will analyze the data given in Table 8.2. Note that these data are grouped. In order to use the expressions derived above, these data needs to be converted to individual data. This can be performed by the following R commands:

```
beetle = read.table("beetle.dat",col.names=c("dose","n","y"))
m = dim(beetle)[1]
x = NULL
y = NULL
for(j in 1:m)
  {
    x = c(x,rep(beetle$dose[j],beetle$n[j]))
    y = c(y,rep(1,beetle$y[j]),rep(0,beetle$n[j]-beetle$y[j]))
  }
beetle2 = data.frame(dose=x,resp=y)
```

Running the routine described in Programwindow 8.3, convergence was reached after 6 iterations to $\widehat{\beta}_0 = -60.72$ and $\widehat{\beta}_1 = 34.27$. The values at different iterations are shown below.

Iteration s	$\beta_0^{(s)}$	$\beta_1^{(s)}$	$l(\boldsymbol{\beta}^{(s)})$	$\delta^{(s)}$
0	2.00000	1.00000	-721.3648	
1	-104.29550	57.96621	-248.0056	0.25
2	-45.92656	25.95912	-191.0286	0.50
3	-57.76158	32.60580	-186.4047	1.00
4	-60.58140	34.19359	-186.2358	1.00
5	-60.71715	34.27016	-186.2354	1.00
6	-60.71745	34.27033	-186.2354	1.00
7	-60.71745	34.27033	-186.2354	1.00

Concerning the uncertainty involved in these estimates, the large sample approximation to the covariance matrix is given directly from (8.17) with $\boldsymbol{\beta} = \widehat{\boldsymbol{\beta}}$ inserted. This gave estimated covariance matrix

$$\mathsf{Var}[\widehat{\boldsymbol{\beta}}] \approx \begin{bmatrix} 26.8398 & -15.0822 \\ -15.0822 & 8.4806 \end{bmatrix}.$$

and standard errors 5.1807 and 2.9121 for $\widehat{\beta}_0$ and $\widehat{\beta}_1$, respectively. □

9 Discussion

In this note we have discussed several different numerical procedures for optimization. Although our primary concern has been on maximum likelihood problems, the procedures could just as well have been applied to other optimization problems.

In general it is difficult to give recommendations on which procedure to use. Some comparative remarks (Titterington et al. 1985) can be made though:

(a) Direct procedures will in many cases work, but convergence could be extremely slow.

(b) The Newton-Raphson method and the Method of Scoring are usually more complicated, and there is no guarantee of monotonicity.

(c) If the Newton-Raphson method converges, it converges fast (second order).

(d) For the Newton-Raphson method, the observed information matrix is directly given as part of the algorithm, while for direct maximization some further calculations are needed in order to obtain this.

(e) In general, no method is guaranteed to give the global optimum. The algorithms should therefore be run with different starting values.

```
nr.logist = function(data,beta.start,eps=0.000001)
 {
  x=data[,1];y=data[,2];n=length(x)
  diff=1;beta=beta.start
  p = exp(beta[1]+beta[2]*x)/(1+exp(beta[1]+beta[2]*x))
  l = sum(y*log(p)+(1-y)*log(1-p))
  while(diff>eps)
   {
    beta.old = beta
    l.old = l
    p = exp(beta[1]+beta[2]*x)/(1+exp(beta[1]+beta[2]*x))
    s = c(sum(y-p),sum((y-p)*x))
    Jbar = matrix(c(sum(p*(1-p)),sum(p*(1-p)*x),
                    sum(p*(1-p)*x),sum(p*(1-p)*x*x)),ncol=2)
    l=l.old-1;delta=1
    while(l<l.old)
     {
      beta = beta.old + delta*solve(Jbar,s)
      p = exp(beta[1]+beta[2]*x)/(1+exp(beta[1]+beta[2]*x))
      l = sum(y*log(p)+(1-y)*log(1-p))
      delta=delta/2
     }
    diff = sum(abs(beta-beta.old))
  }
 list(beta=beta)
}
```

Figure 8.3: R *code for running Newton-Raphson algorithm for logistic regression.*

A R code

```
x = scan("ILLRAIN.DAT",na.strings="*")
x = x[!is.na(x)]
alpha  = seq(0.35,0.55,0.005);lambda = seq(1,3,0.05)
loglik = matrix(nrow=length(alpha),ncol=length(lambda))
n = length(x);sumx=sum(x);sumlogx = sum(log(x));
for(i in 1:length(alpha))
for(j in 1:length(lambda))
 loglik[i,j] = n*alpha[i]*log(lambda[j])+(alpha[i]-1)*sumlogx-
               lambda[j]*sumx-n*log(gamma(alpha[i]))
par(mfrow=c(1,2))
#image(alpha,lambda,exp(loglik),col=gray((0:32)/32))
#image(alpha,lambda,loglik,col=gray((0:32)/32))
persp(alpha,lambda,exp(loglik),theta=330,phi=45,shade=1,zlab="lik")
persp(alpha,lambda,loglik,theta=330,phi=45,shade=1,zlab="l")
```

Programwindow A.1: R *code for plotting likelihood function (figure 1.3) for rainfall data.*

References

D. R. Cox and D. Oakes. *Analysis of Survival Data*. Chapman & Hall, London, 1984.

J.L. Devore and K.N. Berk. *Modern mathematical statistics with applications*. Duxbury Pr, 2007. ISBN 0534404731.

B. S. Everitt. *Introduction to optimization methods and their applications in statistics*. Chapman and Hall, London, 1987.

K. Lange. *Numerical Analysis for Statisticians*. Statistics and Computing. Springer Verlag, 1999.

J. A. Rice. *Mathematical statistics and data analysis*. Duxbury Press, Belmont, California, second edition, 1995.

D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical Analysis of Finite Mixture Distributions*. New York: John Wiley & Sons, 1985.

C. F. Van Loan. *Introduction to scientific computing*. Prentice Hall, Upper Saddle River, NJ 07458, second edition, 2000.

W. N. Venables and B. D Ripley. *Modern Applied Statistics with S-plus*. Statistics and Computing. Springer Verlag, third edition, 1999.