Likelihood theory, numerical methods, simulation methods

April 8, 2015

1 Numerical maximization of the likelihood

For many statistical models, closed form expressions for the maximum likelihood estimators can not be derived. For example, suppose that $t_1, t_2, \ldots, t_n$ is a sample from a Weibull distribution with parameters a and b. The density function for the Weibull distribution is given by

$$f(t) = \frac{a}{b}\left(\frac{t}{b}\right)^{a-1} e^{-\left(\frac{t}{b}\right)^a}. \tag{1}$$

The Weibull distribution can be derived from the assumption that the rate of mortality $\lambda(t) = (1/b)(t/b)^{a-1}$. When the parameter $a = 1$, the rate of mortality is constant and the models becomes equivalent to the exponential model. For other values of a, the rate of mortality either increases or deceases with age. Note that the parameterization used here (and by R) is somewhat differerent from the parameterization used in notat 2 in ST0103.

Given this model, the likelihood function for the observed data is thus

$$L(a,b) = \prod_{i=1}^{n} \frac{a}{b}\left(\frac{t_i}{b}\right)^{a-1} e^{-\left(\frac{t_i}{b}\right)^a} \tag{2}$$

$$= a^n b^{-an} \prod \left(\frac{t_i}{b}\right)^{a-1} e^{-\sum (t_i/b)^a}, \tag{3}$$

and the log likelihood

$$l(a,b) = n\ln a - na\ln b - (a-1)\sum \ln t_i - \sum (t_i/b)^a. \tag{4}$$

Noting that

$$\frac{\partial}{\partial a}(t_i/b)^a = \frac{\partial}{\partial a} e^{a\ln(t_i/b)} = e^{a\ln(t_i/b)}\ln(t_i/b) = (t_i/b)^a \ln(t_i/b), \tag{5}$$

and setting the partial derivatives of the log likelihood function $l(a,b)$ equal to zero yields

$$\frac{n}{a} - n\ln b - \sum \ln t_i - \sum (t_i/b)^a \ln(t_i/b) = 0 \tag{6}$$

and

$$-\frac{na}{b} + \frac{a}{b}\sum (t_i/b)^a. \tag{7}$$

These equations are non-linear in the unknowns a and b and no closed form solution can be found.

The maximum likelihood estimates must therefore instead be computed using numerical methods. This can by first defining a function in R which computes the likelihood (or the log likelihood) for given values of the unknown parameters a and b and for a given set of data. We shall then use a special function in R which finds the parameter values which maximises the likelihood. This function (the R function `optim`) works by making repeated calls to a given

function for different parameter values. Based on the evaluated value of likelihood at these parameter values, the optimisation algorithm usually finds it's way at least to a local maximum in the parameter space.

Let us start by defining a function in R which computes the likelihood. This should be a function of the unknown parameters a and b and since the likelihood also depends on the observed sample $t_1, t_2, \ldots, t_n$ we need an additional argument containing the observations. Our R function may then compute either the likelihood of the log likelihood. In practice, optimisation on the log likelihood works best. By default, `optim` minimises the given function. Thus if we let our function compute the negative log likelihood and minimise this we maximise the likelihood.

Based on equation (4) we can define the likelihood function in R as follows

```
l <- function(a,b,t) {
  n <- length(t)
  lnL <- n*log(a) - n*a*log(b) - (a-1)*sum(log(t)) - sum((t/b)^a)
  -lnL
}
```

However, since this function needs to speak to `optim` in a certain way (see `?optim`), it needs to take a vector containing the parameter values as it's first argument and not as two scalar arguments. We let the first element `p[1]` of the first argument `p` represent a and the second element `p[2]` represent b, and modify the function to

```
l <- function(p,t) {
  n <- length(t)
  lnL <- n*log(p[1]) - n*p[1]*log(p[2]) - (p[1]-1)*sum(log(t)) - sum((t/p[2])^p[1])
  -lnL
}
```

or, a somewhat easier to read version,

```
l <- function(p,t) {
  a <- p[1]
  b <- p[2]
  n <- length(t)
  lnL <- n*log(a) - n*a*log(b) - (a-1)*sum(log(t)) - sum((t/b)^a)
  -lnL
}
```

Yet another and probably the preferred way of computing the likelihood is to use R's built in function `dweibull` for evaluting the density function of the Weibull distribtuion. If we use the vector `t` containing the observed values as the first argument `dweibull` will return a vector containing $f(t_1), f(t_2), \ldots, f(t_n)$. The log likelihood is the log of the product of this or sum of the log of each term. Taking logs after having computed each $f(t_i)$, however, should be avoided since this may lead to numerical underflow. Instead, pass the `log=T` argument to `dweibull`.

```
l <- function(p,t) {
  -sum(dweibull(t,shape=p[1],scale=p[2],log=T))
}
```

Let us first simulate a sample of $n = 50$ observations simulated from a Weibull distribution with shape parameter $a = 2$ and scale parameter $b = 10$ as follows

```
tsim <- rweibull(50,shape=2,scale=10)
```

A histogram of the data and the density function of the true model generated using the commands

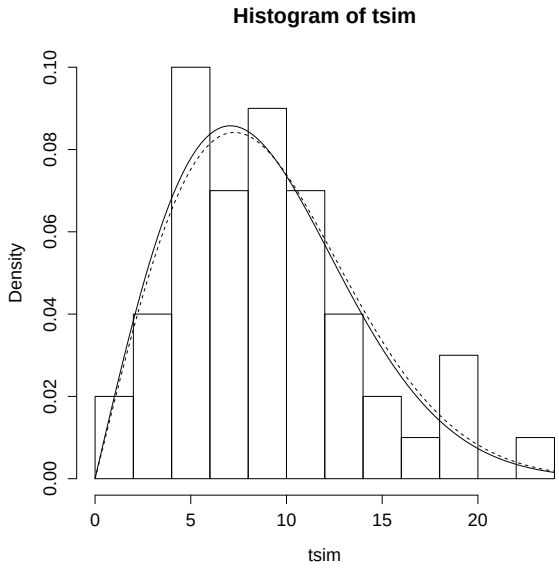

Figure 1: 50 observation simulated from a Weibull model with shape parameter $a = 2$ and scale parameter $b = 10$ (solid curve). The dashed lines represents the maximum likelihood estimate of the density function (see text).

```
hist(tsim,breaks=10,prob=T)
curve(dweibull(x,shape=2,scale=10),add=T)
```

is shown in Fig. 1.

Before proceeding, let's also make a surface plot the likelihood function. This can be done using the `persp` function which is part of the base-package. A more convenient interface to `persp` (aswell as `image` and `plot3d` in the `rgl` package) is `curve3d` function in the package `emdbook`. This function takes an expression in `x` and `y`, in our case representing the parameters a and b as it's first argument. Since `l` computes the negative log likelihood, `exp(-l( ))` gives us the log likelihood

```
library(emdbook)
curve3d(exp(-l(c(x,y),t=tsim)),
        xlim=c(1.5,2.5),ylim=c(5,15),
        sys="persp",
        theta=30,phi=30,
        tick="detailed",
        xlab="shape parameter a",ylab="scale parameter b",zlab="likelihood")
```

The other arguments specifies that the plot should be created using `persp` and the other arguments (specifying perspective etc.) are passed on to `persp` (see `?persp` for details). The resulting surface plot is shown in Fig. 2. We see that the likelihood has a maximum somewhere around the true parameter values $a = 2$ and $b = 10$.

The maximum likelihood estimates can now be computed by making a call to `optim` as follows

```
> optim(c(1,1),l,t=tsim)
$par
[1]  2.011764 10.230481

$value
```

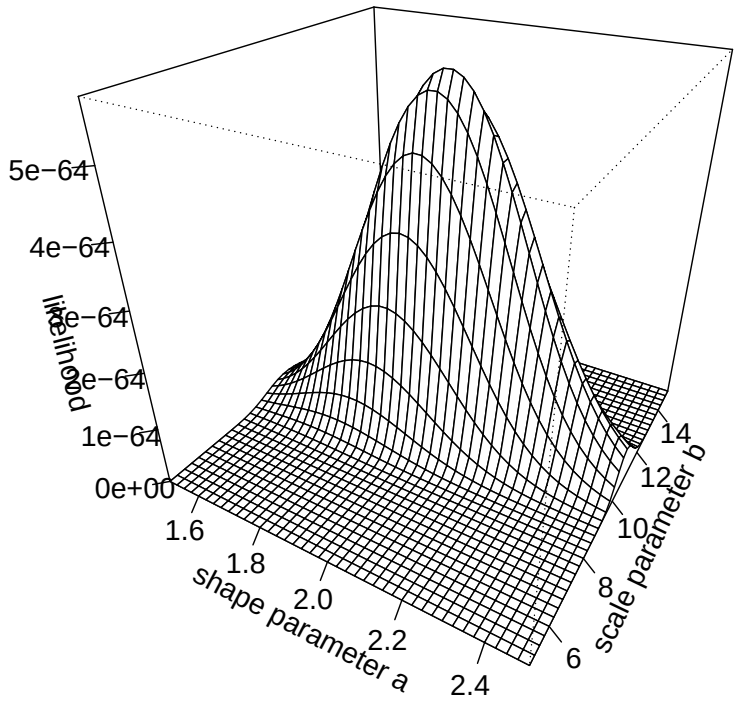

Figure 2: Surface plot of the likelihood function for the data shown in Fig. 1.

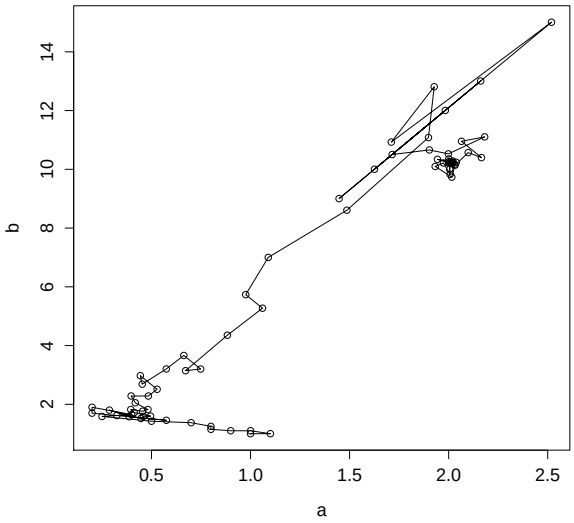

Figure 3: The trajectory of the optimisation algorithm through the parameter space from the inital values $a = b = 1$ towards the maximum likelihood estimates at $a = 2.0117$ $b = 10.2304$.

```
[1] 145.603

$counts
function gradient
      83       NA

$convergence
[1] 0

$message
NULL
```

The first argument to `optim` is a vector of inital values for the parameters. The second argument is the name of the function to be minimised. Also note the third argument, `t=tsim`. This argument does not match any of `optim`'s ordinary argument but is instead passed on to `l`, each time `optim` makes a call to `l`. See `?optim` and the defintion of `optim` for details.

`optim` works by evaluating the function specified as the second argument (in the above case `l`), at different nearby points. Based on the local gradient of the likelihood surface the algorithm then moves through the parameter space towards a the maximum, which in this case is located at $a = 2.0117$ and $b = 10.2304$. Fig. 3 shows trajactory taken by the algorithm towards this maximum if using the default optimisation method (`method="Nelder-Mead"`).

Other optimisation methods can be used; when working with log likelihood functions convergence is often faster if using `method="BFGS"` which is basically a multivariate version of Newton's method. For the above example, using this method reduces the number of function evaluations from 83 to 42.

```
> optim(c(1,1),l,t=tsim,method="BFGS")
$par
[1]  2.011848 10.228542

$value
```

```
[1] 145.603

$counts
function gradient
      42       14

$convergence
[1] 0

$message
NULL

There were 12 warnings (use warnings() to see them)

> warnings()
Warning messages:
1: In dweibull(x, shape, scale, log) : NaNs produced
2: In dweibull(x, shape, scale, log) : NaNs produced
3: In dweibull(x, shape, scale, log) : NaNs produced
...
```

The warnings occur because the algorithm used attempts to evaluate the likelihood outside the permitted ranges of the parameter values, that is, for non-positive values of a and b. In this particular case, when the likelihood is evaluated outside the permitted ranges for the parameters $a < 0$, `dweibull` produces only a warning and returns `NaN` (not a number) which propagates in further computations so that `l` also returns `NaN` as value. Both the default method used by `optim` (`"Nelder-Mead"`) aswell as (`"BFGS"`) can deal with functions returning `Inf`, `NA` or `NaN` for parameter values for which the function is not defined.

1.1 Constraints on the parameters

To be on the safe side, we can specify lower and upper limits on a and b ("box constraints") in which case `optim` uses the `"L-BFGS-B"` method.

```
> optim(c(1,1),l,t=tsim,lower=c(0,0),upper=c(Inf,Inf),method="L-BFGS-B")
$par
[1]  2.011847 10.228543

$value
[1] 145.603

$counts
function gradient
      14       14

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Unlike `"Nelder-Mead"` and `"BFGS"`, the `"L-BFGS-B"` method only works on functions which are always defined inside the box contraints specified. In the above case, certain starting values

causes an evaluation at the boundary $a = 0$ which causes the algoritm to crash since the the Weibull density function (and hence the likelihood) becomes undefined at $a = 0$:

```
> optim(c(.001,1),l,t=tsim,lower=c(0,0),upper=c(Inf,Inf),method="L-BFGS-B")
Error in optim(c(0.001, 1), l, t = tsim, lower = c(0, 0), upper = c(Inf,  :
  non-finite finite-difference value [1]
In addition: Warning message:
In dweibull(x, shape, scale, log) : NaNs produced
```

The way around the problem is to use lower boundaries on the parameters slightly larger than zero.

```
> optim(c(.001,1),l,t=tsim,lower=c(.0001,.0001),upper=c(Inf,Inf),method="L-BFGS-B")
$par
[1] 1.902166 9.440133

$value
[1] 144.5283

$counts
function gradient
      21       21

$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

The constraints above can be dealt with by reparameterisation reparameterizing the model in terms of $\theta_1 = \ln a$ and $\theta_2 = \ln b$. For a parameter q for which the likelihood is only defined for $0 < q < 1$, we may reparameterize the model in terms of $\theta = \text{logit}\, q$ and the do the optimisations with respect to θ, $-\infty < \theta < \infty$.

Reparameterization may also be useful for problems involving contraints on the parameters that can not be dealt with as "box-constraints". For example, the likelihood function for the blood type data is only defined for $0 < p_1 < 1$, $0 < p_2 < 1$ and $0 < p_3 < 1$ and $\sum p_i = 1$. The latter two condition translates to the constraint $p_1 + p_2 < 1$ on the parameters which the likelihood is maximised with respect to, that is, p_1 and p_2. Such contraints can be dealt with by reparameterization of the model in terms of an additive log ratio transformation (Bolker (2008). This transforms p_1, p_2, p_3 to two new parameters, θ_1 and θ_2 given by

$$\theta_1 = \ln \frac{p_1}{p_3}, \qquad \theta_2 = \ln \frac{p_2}{p_3}. \tag{8}$$

After some algebra, we find that the back-transformation is given by

$$p_1 = \frac{e^{\theta_1}}{1 + e^{\theta_1} + e^{\theta_2}}, \qquad p_1 = \frac{e^{\theta_2}}{1 + e^{\theta_1} + e^{\theta_2}}, \qquad p_1 = \frac{1}{1 + e^{\theta_1} + e^{\theta_2}}. \tag{9}$$

Using the alternative parameterization of the model given by (8), and letting `par` represent (θ_1, θ_2) instead of p_A and p_B, the model for the blood type data in handout 2 can be fitted using as follows without the need for any contraints on the new parameters.

```
multinomialprobs <- function(par) {
  pA <- exp(par[1])/(1+exp(par[1])+exp(par[2]))
```

```
  pB <- exp(par[2])/(1+exp(par[1])+exp(par[2]))
  p0 <- 1/(1+exp(par[1])+exp(par[2]))
  c(pA^2 + 2*pA*p0, pB^2 + 2*pB*p0, 2*pA*pB, p0^2)
}

lnL <- function(par,x) {
  n <- sum(x)
  -dmultinom(x,prob=multinomialprobs(par),log=T)
}

> fit <- optim(c(-10,20),lnL,x=x)
> fit$par
[1] -1.528548 -1.996059
> multinomialprobs(fit$par)
[1] 0.26271241 0.15859208 0.03220303 0.54649247
```

In contrast, the original code might fail if bad starting values are used.

Finally it should be noted that if the function being maximised has several maxima, the algorithm may only find a local and not the global maximum but this is seldom the case when we are maximising a likelihood function. Furthermore, for a model containing more parameters than can be estimated from the data, the likelihood surface may take the shape of a flat ridge with no maximum. Similarly, problems due to linear separation (see handout 4) may lead to a misbehaved likelihood function.

2 Some asymptotic theory

2.1 Approximate standard errors

The likelihood function plotted in Fig. 2 looks very similar to a two-dimensional density function. However, in ordinary frequentist statistics, we can not make this interpretation since the parameters a and b are unknown fixed constants and not random variables. We shall see, however, that approximate standard errors can be computed based on the curvature (a matrix of second order partial derivatives) of the log likelihood function.

Let us first consider a simple example involving a single unknown parameter. If $X \sim \text{bin}(n, p)$ and we observe x successes out of n trials, the likelihood function is

$$L(p) = \binom{n}{x} p^x (1-p)^{n-x}, \tag{10}$$

the log likelihood

$$l(p) = \ln \binom{n}{x} + x \ln p + (n-x) \ln(1-p), \tag{11}$$

and the value of p which maximises the log likelihood, the solution of the equation

$$\frac{dl}{dp} = \frac{x}{p} - \frac{n-x}{1-p} = 0 \tag{12}$$

is

$$\hat{p} = x/n. \tag{13}$$

Since X has a binomial distribution $\text{Var}\, X = np(1-p)$ and the variance of the estimator $\hat{p}$ is

$$\text{Var}\, \hat{p} = \text{Var}\, X/n = \frac{p(1-p)}{n}. \tag{14}$$

Now consider the second derivative of l,

$$\frac{d^2l}{dp^2} = -\frac{x}{p^2} + \frac{n-x}{(1-p)^2}. \tag{15}$$

At the maximum likelihood, that is, for $p = \hat{p} = x/n$, the second derivative takes the value

$$\left.\frac{d^2l}{dp^2}\right|_{p=x/n} = -\frac{xn^2}{x^2} + \frac{(n-x)n^2}{(n-x)^2} = -\frac{n}{\hat{p}(1-\hat{p})}. \tag{16}$$

Thus, in this case, we see that

$$\left(-\left.\frac{d^2l}{dp^2}\right|_{p=\hat{p}}\right)^{-1} \tag{17}$$

gives us an estimate of the variance (14) of the maximum likelihood estimator $\hat{p}$. Note how the inverse relationship means that a large second derivative gives a small estimate of the variance.

More generally, for a model with k parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_k)$, an estimate of the variance-covariance matrix of the maximum likelihood estimates is given by

$$-\left[\begin{array}{cccc} \frac{\partial^2 l}{\partial\theta_1^2} & \frac{\partial^2 l}{\partial\theta_1\partial\theta_2} & \cdots & \frac{\partial^2 l}{\partial\theta_1\partial\theta_k} \\ \frac{\partial^2 l}{\partial\theta_2\partial\theta_1} & \frac{\partial^2 l}{\partial\theta_2^2} & & \\ \vdots & & \ddots & \\ \frac{\partial^2 l}{\partial\theta_k\partial\theta_1} & & & \frac{\partial^2 l}{\partial\theta_k^2} \end{array}\right]^{-1}_{\theta=\hat{\theta}} \tag{18}$$

where -1 denotes the matrix inverse.

This result follows from asymptotic efficiency of maximum likelihood estimators and the so called Cramér-Rao lower bound. It is only exact asymptotically as the sample size n goes to infinity. For small sample sizes it may provide a poor approximation to the variance-covariance matrix of the estimators.

In practice, if the maximum likelihood estimates are computed by maxisining the likelihood numerically using the method in subsection 1, the above matrix containing all the second order derivatives is computed by `optim` if specifying the additional argument `hessian=TRUE`. Going back to the Weibull example, the variance-covariance matrix of the maximum likelihood estimates can be computed by inverting this matrix using `solve` as follows.

```
> fit <- optim(c(1,1),l,t=tsim,hessian=TRUE)
> fit$hessian
          [,1]      [,2]
[1,] 22.851009 -2.103949
[2,] -2.103949  1.932290
> solve(fit$hessian)
           [,1]       [,2]
[1,] 0.04863777 0.05295861
[2,] 0.05295861 0.57518390
```

Since we are working with the negative log likelihood we don't need to change the sign of each element of the matrix.

Thus, (estimates of) the variance of the estimators based on the above asymptotic theory are $\operatorname{Var}\hat{a} = 0.048$ and $\operatorname{Var}\hat{b} = 0.57$. Taking the square root of the elements along the diagonal we obtain the stanard errors

```
> sqrt(diag(solve(fit$hessian)))
[1] 0.2205397 0.7584088
```

So the estimates of the shape and scale parameters are $\hat{a} = 2.01 \pm 0.22$ and $\hat{b} = 10.23 \pm 0.75$. An estimate of the correlaton is

```
> cov2cor(solve(fit$hessian))
          [,1]      [,2]
[1,] 1.0000000 0.3166258
[2,] 0.3166258 1.0000000
```

2.2 Asymptotic properties of likelihood ratio tests

Another useful approximate result concerns the the change in the maximum likelihood when some model H_0 is extended to some more general model H_1, say by adding an extra explanatory variable. In general, H_0 needs to be nested inside H_1, that is, H_0 must be a special case of H_1 for certain parameter values. In such cases, the maximum likelihood (the probability of the observed data) will always be larger for the more general flexible model just like the residual sum of squares and the deviance is always reduced when an additional term is added to the model is added to a linear or generalized linear model.

Thus, the ratio between the maximum likelihood under H_1 and H_0 or the difference between the maximum log likelihoods can be used a test statistic. In practice, two times this,

$$2(\ln L_1 - \ln L_0) \tag{19}$$

where L_0 and L_1 is the maximum likelihood under H_0 and H_1 is used. This statistic is approximately chi-square distributed with degrees of freedom equal to the difference $p_1 - p_0$ between the number of estimated parameters under H_0 and H_1.

Returning to the Weibull example we may be interested in testing the simpler null hypothesis H_0 that the shape parameter $a = 1$ versus the alternative model H_1 in which a can take any positve value. These models are then nested since H_0 is a special case of H_1 and (19) applies. Under H_0 each observation is now simply exponentially distributed with parameter b and the maximum likelihood estimate of b is $\hat{b} = \bar{t}$. The observed maximum log likelihood (4) at this point in the parameter space simplifies to

$$\begin{aligned} \ln L(1, \hat{b}) &= -n \ln \hat{b} - \sum_{i=1}^{n} \frac{t_i}{\hat{b}} \\ &= -n \ln \bar{t} - \sum_{i=1}^{n} \frac{t_i}{\bar{t}}. \end{aligned} \tag{20}$$

For our simulated dataset this evaluates to

```
> n <- length(tsim)
> -n*log(mean(tsim))-sum(t/mean(t))
[1] -210.1440
```

From the `$value` component the list returned by `optim`, the maximum likelihood under H_1 is -145.603 so the observed value of our test statistic (19) is thus

```
> -146.603 - (-210.144)
[1] 63.541
```

that is, much greater than the critical value of

```
> qchisq(.95,df=1)
[1] 3.841459
```

so we can reject $H_0 : a = 0$ (the exponential model) in favour of the more general Weibull model which is not surprising given that the data was simulated from model for which the true value of shape parameter $a = 2$.

3 More examples

3.1 Generalized linear models

Different quite complex models can easily be fitted using the above method. For example, consider a generalized linear model with a binomially distributed response variable y. Suppose that

$$\operatorname{logit} p = \beta_0 + \beta_1 x. \tag{21}$$

This model contains two parameters β_0 and β_1. Given the observations $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, the (negative log) likelihood can be computed as follows

```
lnLglm <- function(par,y,x) {
  beta0 <- par[1]
  beta1 <- par[2]
  eta <- beta0 + beta1*x
  p <- 1/(1+exp(-eta))
  -sum(dbinom(y,size=1,prob=p,log=T))
}
```

The first two lines are there to increase the readability of the code. In the third line, the predicted values on the logit scale for the current values of the parameters is computed for each value of the explanatory variable x contained in `x`, and in the second last line, the corresponding predicted probabilities. In the last line, since each Y_i is binomially distributed with parameters $n = 1$ and p's given by the vector `p`, the total log likelihood can be computed as the sum of the log of probabilities of each observed y_i'value.

Using the data `juul.girl` (see Dalg. p. 240), the maximum likelihood estimates and standard errors of the parameters β_0 and β_1 can now be found as follows.

```
> attach(jull.girl)
> y <- c(0,1)[menarche]
> fit <- optim(c(0,0),lnLglm,x=age,y=y,hessian=TRUE)
> fit
$par
[1] -20.014137   1.517378

$value
[1] 100.3321

$counts
function gradient
      83       NA

$convergence
[1] 0

$message
```

```
NULL

$hessian
          [,1]      [,2]
[1,]  30.65301  401.1491
[2,] 401.14905 5291.7033

> sqrt(diag(solve(fit$hessian)))
[1] 2.0284609 0.1543851
```

This is exactly the estimates and standard errors returned when fitting the model with R's built in function for fitting generalized linear model.

```
> summary(glm(y~age,fam=binomial))

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -20.0132     2.0284  -9.867   <2e-16 ***
age           1.5173     0.1544   9.829   <2e-16 ***
--

    Null deviance: 719.39  on 518  degrees of freedom
Residual deviance: 200.66  on 517  degrees of freedom
AIC: 204.66

Number of Fisher Scoring iterations: 7
```

Also recall that the deviance of a generalized linear model is defined as two times the differene in log likelihood between the fitted and the full model. For the full model involving as many parameters as observations, the maximum likelihood estimates of each p is simply $\hat{p}_i = y_i$ since all the p_i's are essentially free parameters. At the maximum likelihood estimates of all the p_i's, the proability of the observed data (the maximum likelihood) is thus simply one and the maximum log likelihood under the full model is zero. Thus the deviance become $2(0 - (-100.33) = 200.66$ as reported in the summary.

While, standard generalized linear models encompass a large number of interesting models, models involving non-standard assumptions are sometimes of interest and we need to resort to the numerical techniques introduced above as we shall see in assignment 10.

In such cases it may still be of interest to compute the deviance of a given fitted model (see assignment 11). In particular when each observation y_i is based on many Bernoulli trials n_i this will allow us to assess the goodness-of-fit of a given model. In such cases, the maximum likelihood estimates of each p_i under the full model (a total of n parameters) then becomes

$$\hat{p}_i = \frac{y_i}{n_i} \tag{22}$$

and the corresponding maximum log likelihood becomes

$$\ln L_{full} = \sum \ln \binom{n_i}{y_i} + \sum y_i \ln \hat{p}_i + \sum (n_i - y_i) \ln(1 - \hat{p}_i) \tag{23}$$

The result that the deviance, that is,

$$D = 2(\ln L_{\text{full}} - \ln L) \tag{24}$$

is approximately chi-square distributed with $n - p$ degrees of freedom, as we noted in handout 4, follows from the more generally theorem (19) introduced above, since a given model and the full model are two nested alternative models involving p and n parameters respectively.

3.2 Biased sampling (optional material)

Family size example...

3.3 More computationally intensive likelihoods (optional material)

Suppose that the weight of different individuals in a population of cod follows a log-normal distribution such that logtransformed weights X are normally distributed with parameters μ and σ. We sample n individuals with weights $X_1, X_2, \ldots, X_n$ from the population at random but using a fishing net such that only individuals above a certain size $X > a$ are captured...

4 Simulation based methods

For many models, including both standard and non-standard models, only various approximate results are available from which we can compute standard errors of parameter estimates and test different alternative hypotheses. When the amount of data is small, these approximation may perform poorly. In such cases, we may need to assess the properties such as the bias and variance of the estimators used using simulations. We begin by a brief recap of basic probability theory.

The sample space S of a random experiment is the set of all distinct outcomes s of the experiment. An event A is subset of the sample space S. The probability of an event A is usually defined as the limiting value of the long run frequency of the event if we imagine that we repeat the experiment many times,

$$P(A) = \lim_{m\to\infty} \frac{n_A}{m}. \tag{25}$$

On the computer, we can easily generate realisations of given random experiments. If we then count number of times n_A a given event A occurs and divide this by the number of realisations generated m, we obtain an estimate of $P(A)$. By increasing m, the estimate can be made arbitrarily precise.

On given sample spaces we also define random variables. The expected value of contineous random variables is defined as

$$EX = \int x f(x) dx \tag{26}$$

and for discrete random variables, as

$$EX = \sum xp(x). \tag{27}$$

The theoretical expecation EX of a random variable X can be thought of as the limiting value of the long run average if we imagine that we repeat the experiment many times.

Again, on the computer, we can easily simulate many realisations of such random variables, say realisations $X^{(1)}, X^{(2)}, \ldots, X^{(m)}$. An estimate of the theoretical expecation can computed simply by taking the average of the simulated values,

$$\bar{X} = \frac{1}{m}\sum_{i=1}^{m} X^{(i)}. \tag{28}$$

By increasing the number of realisations m, the theoretical mean EX can be estimated with arbitrarily precision. Similarly,

$$\operatorname{Var} X = E(X - EX)^2, \tag{29}$$

that is, the expected value of the random variable $W = (X - EX)^2$, can be estimated by

$$\frac{1}{m}\sum_{i=1}^{m} W^{(i)} = \frac{1}{m}\sum_{i=1}^{m} (X^{(i)} - EX)^2 \tag{30}$$

In practice, if we don't know the mean EX, the sample variance

$$S^2 = \frac{1}{m-1}\sum_{i=1}^{m}(X^{(i)} - \bar{X})^2 \tag{31}$$

may be used.

4.1 Computing probabilites

Suppose that the number of goals scored by Denmark and Norway X_1 and X_2 in a given match is Poisson distributed with expectations $\lambda_1 = 1.2$ and $\lambda_2 = 1.5$ respectively. What is the probability of that Denmark wins, the probability of a draw and the probability that Norway wins the game?

Given the assumptions, a realisation of the experiment can be simulated by simulating two realisations from Poisson distributions with the appropriate means. If we do this $m = 1000$ times and count the number of realisations which leads to a victory for Denmark, a draw and a victory for Norway, estimates of the above probabilities can be computed by dividing the respective counts by m as follows.

```
footballprob <- function(lambda1,lambda2,m=1000) {
  n.team1 <- n.team2 <- n.draws <- 0
  for (i in 1:m) {
    x1 <- rpois(1,lambda1)
    x2 <- rpois(1,lambda2)
    if (x1==x2) {
       n.draws <- n.draws + 1
    } else {
       if (x1 > x2)
          n.team1 <- n.team1 + 1
       else
          n.team2 <- n.team2 + 1
    }
  }
  list(team1=n.team1/m,draw=n.draws/m,team2=n.team2/m)
}
```

Plugging in the Poisson means for each team, the probabilities are

```
> footballprob(1.2,1.5)
$team1
[1] 0.288

$draw
[1] 0.245

$team2
[1] 0.467
```

4.2 Computing coverage probabilities of approximate confidence intervals

Suppose that we observe $X = x$ successes out in a total of n indepenent Bernoulli trials so that X is binomially distributed. In most introductory statistics textbooks, the approximate $(1-\alpha)$-confidence interval

$$(\hat{p} - z_{\alpha/2}\sqrt{\hat{p}(1-\hat{p})/n}, \hat{p} + z_{\alpha/2}\sqrt{\hat{p}(1-\hat{p})/n}) \tag{32}$$

for the parameter p is derived based on the approximate standard normal distribution of

$$\frac{\hat{p}-p}{\sqrt{\hat{p}(1-\hat{p})/n}}. \tag{33}$$

Also, this interval do not involve a continuity correction for the fact that X (and hence (33)) is discrete and not continuous.

In R, a confidence intervals involving fewer approximations and a continuity correction is computed by the function `prop.test`. Suppose that the true value of $p = 0.1$ and that $n = 100$. We may then simulate on realisation of the data and compute the interval as follows.

```
> p <- 0.1
> x <- rbinom(1,size=n,prob=p)
> x
13
> prop.test(x,n)$conf.int
[1] 0.07376794 0.21560134
```

Note how we can refer the confidence interval computed by `prop.test` (a vector of length 2) by referring to the `$conf.int` component of the list returned by `prop.test`. Also note that this interval is assymetric around $\hat{p} = 0.1$.

In contrast, the textbook interval (32) is assymetric. Let's first program a function which computes this interval. This function will need to take x and n as arguments. As we shall see later, life will be easier if our function computing the textbook interval returns the interval the same way as `prop.test`, that is, as a vector as part of a list. The upper $\alpha/2$-quantile, $z_\alpha/2$ is computed with `qnorm`.

```
textbook.ci <- function(x,n,alpha=0.05) {
  phat <- x/n
  z <- qnorm(alpha/2,lower.tail=FALSE)
  lower <- phat - z*sqrt(phat*(1-phat)/n)
  upper <- phat + z*sqrt(phat*(1-phat)/n)
  list(conf.int=c(lower,upper))
}
```

Testing this on the same simulated data we get the following.

```
> textbook.ci(x,n)
$conf.int
[1] 0.06408574 0.19591426
```

While the nominal confidence level of both intervals are 95%, the real coverage, that is,

$$P(A < p < B) \tag{34}$$

where A and B is the lower and upper limit of the confidence interval, may differ from this. The above probability may be computed by simulating say, $m = 1000$ realisations of the experiment, and then counting the number of times the confidence limits contains the true parameter value p. Using a for-loop, this can be done as follows in R.

```
p <- 0.1
n <- 100
nA <- 0
for (i in 1:1000) {
   x <- rbinom(1,size=n,prob=p)
```

```
   ci <- prop.test(x,n)$conf.int
   if (ci[1]<p & ci[2]>p)
      nA <- nA + 1
}
```

Based on these simulations, the estimate of the above probability becomes

```
> nA/1000
[1] 0.967
```

This can be turned into a more general function which can find the coverage of confidence intervals computed by different functions `fn`.

```
coverage <- function(n,p,fn=prop.test,m=1000) {
  nA <- 0
  for (i in 1:m) {
     x <- rbinom(1,size=n,prob=p)
     ci <- fn(x,n)$conf.int
     if (ci[1]<p & ci[2]>p)
        nA <- nA + 1
  }
  nA/m
}
```

Comparing the real coverage of the above two intervals is now straightforward.

```
> coverage(n=100,p=.1,prop.test)
[1] 0.977
> coverage(n=100,p=.1,textbook.ci)
[1] 0.935
```

More accurate estimates can be obtained by generating more than the default number of simulations.

```
> coverage(n=100,p=.1,prop.test,m=10000)
[1] 0.9726
> coverage(n=100,p=.1,textbook.ci,m=10000)
[1] 0.9324
```

For the parameter values used, we see that `prop.test` computes a more conservative interval whereas the true coverage of the textbook interval in fact is smaller than the nominal level of 95%.

4.3 Assessing bias and variance of estimators

Suppose that $X_1, X_2, \ldots, X_n$ is a random sample from an exponential distribution with density function

$$f(x) = \lambda e^{-\lambda x} \tag{35}$$

The MLE of λ is then

$$\hat{\lambda} = \frac{n}{\sum X_i} \tag{36}$$

Suppose that the true value of $\lambda = 1$. A single realisation of the above random sample of size $n = 10$ can no be simulated as follows

```
n <- 10
lambda <- 1
X <- rexp(n,rate=lambda)
```

and the corresponding value of $\hat{\lambda}$ becomes

```
> lambdahat <- n/sum(x)
> lambdahat
[1] 0.1818182
```

To assess the bias of this estimator we simulate 1000 realisations of the experiment and store all the simulated values of $\hat{\lambda}$ in a vector we call `lambdasim`.

```
lambdasim <- NULL
for (i in 1:1000) {
   X <- rexp(n,rate=lambda)
   lambdasim[i] <- n/sum(X)
}
```

The first line creates `lambdasim` as an empty vector. This is necessary since assigning values to elements of non-existing vectors will cause an error:

```
> myvector[3] <- 5
Error in myvector[3] <- 5 : object 'myvector' not found
```

An estimate of $E(\hat{\lambda})$ can now be obtained by taking the average of the simulated values.

```
> mean(lambdasim)
[1] 1.114902
```

This indicates that $\hat{\lambda}$ overestimates λ by about 10%.

The bias may of course depend on λ which in practice will be unknown aswell as the sample size. To investigate how, let's first turn the above code into a more convinent function which computes an estimate of $E\hat{\lambda}$ aswell as $\operatorname{Var}\hat{\lambda}$ for given values of λ and n.

```
lambdahat.properties <- function(lambda=1,n,m=1000) {
   lambdasim <- NULL
   for (i in 1:1000) {
      X <- rexp(n,rate=lambda)
      lambdasim[i] <- n/sum(X)
   }
   list(mean=mean(lambdasim),var=var(lambdasim))
}
```

A few numerical experiments now suggests that the bias in general seems to be proportional to the true value of λ which is perhaps not very surprising.

```
> lambdahat.properties(1,10)
$mean
[1] 1.103104

$var
[1] 0.1469100

> lambdahat.properties(100,10)
$mean
```

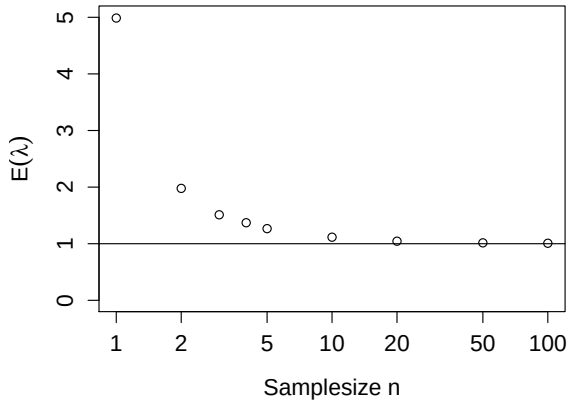

Figure 4: The expected value of the MLE of λ in the exponential model (second axis) estimated by simulating $m = 1000$ random samples of different size n (first axis) using $\lambda = 1$ (solid line).

```
[1] 110.2236

$var
[1] 1530.352

> lambdahat.properties(10000,10)
$mean
[1] 11273.91

$var
[1] 16601025
```

How does the bias depend on the sample size n? A plot (Fig. 4) showing how the estimate of $E\hat{\lambda}$ change with increasing sample size can be generated as follows.

```
nn <- c(1,2,3,4,5,10,20,50,100)
ll <- NULL
for (i in 1:length(nn)) {
  ll[i] <- lambdahat.properties(1,n=nn[i])
}
plot(nn,ll,ylim=c(1,5),log="x",xlab="Samplesize n",ylab=expression(E(hat(lambda))))
```

The bias is thus very large for small sample size but quickly decline to an acceptable level once the sample size reaches about 20 observations.

Next we consider a slightly more elaborate example. For a simple linear regression model,

$$Y_i = \alpha + \beta x_i + \epsilon_i, \tag{37}$$

where $\epsilon_i \sim N(0, \sigma^2)$, it is relatively easy to show that the estimator of the slope,

$$\hat{\beta} = \frac{\sum Y_i(x_i - \bar{x})}{\sum (x_i - \bar{x})^2} \tag{38}$$

is

$$N(\beta, \frac{\sigma^2}{\sum (x_i - \bar{x})^2}). \tag{39}$$

The normality of $\hat{\beta}$ follows from the fact that $\hat{\beta}$ is a linear combination of the Y_i's.

Let's see if we can verify this using simulations. To do this we need to need to simulate realisations of our experiment multiple times. A realisation of the experiment does in this case consist of realisations of the response Y_i for each given value of the explanatory variable x_i.

Suppose that the true value of the intercept $\alpha = 5$, the true value of the intercept is $\beta = 0.5$ and the true value of $\sigma^2 = 1$.

```
alpha <- 5
beta <- .5
simga2 <- 1
```

Suppose also that we are measuring the response for the following values of the explanatory variable x_i.

```
x <- 1:10
n <- length(x)
```

We can now simulate a realisation of the experiment (realisation of the $n = 10$ Y_i's) as follows.

```
Y <- rnorm(n,mean = alpha + beta*x, sd=sqrt(sigma2))
```

Note how this is done by letting the second argument be a vector specifying the expected value of each Y_i.

Having simulated one realisation, the corresponding value of $\hat{\beta}$ can be computed using equation (38) as follows

```
betahat <- sum(Y*(x-mean(x)))/sum((x-mean(x))^2)
```

Alternatively, an approach that can be used more generally is to fit a linear regression using `lm` and extract the estimate from the fitted model object using `coef` which returns a vector of parameters, the second one being the slope.

```
betahat <- coef(lm(Y~x))[2]
```

`coef` returns a named vectors so we can get at the estimate of the regressio coefficient for a particular explanantory by referring to it by it's name.

```
> coef(lm(Y~x))
(Intercept)           x
  5.0181893   0.5232656
> coef(lm(Y~x))["x"]
        x
0.5232656
```

Now let's repeat the above simulation of a single realisation a 1000 times by putting the code inside a for-loop and store all the simulated $\hat{\beta}$'s into a vector called `betasim`.

```
betasim <- NULL
for (i in 1:1000) {
  Y <- rnorm(n,mean = alpha + beta*x, sd=sqrt(sigma2))
  betasim[i] <- coef(lm(Y~x))["x"]
}
```

A histogram of the simulated $\hat{\beta}$'s with the theoretical density function of β given by (39) added on top of the histogram generated using the code

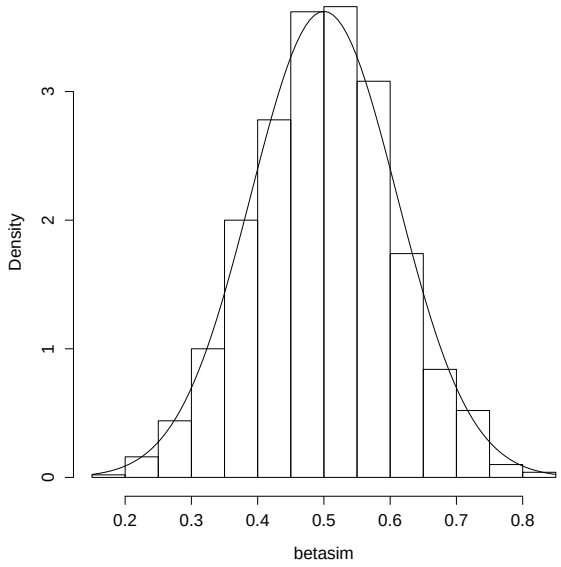

Figure 5: Histogram of simulated values of the estimator of the slope $\hat{\beta}$ in a linear regression and the theoretical normal density function.

```
hist(betasim,freq=F)
varbetahat <- sigma2/sum((x-mean(x))^2)
curve(dnorm(x,mean=beta,sd=sqrt(varbetahat)),add=T)
```

is shown in Fig. 5.

The mean and variance of the simulated realisations also conforms with the theory.

```
> mean(betasim)
[1] 0.4977819
> beta
0.5
> var(betasim)
[1] 0.01116444
> varbetahat
[1] 0.01212121
```

4.4 How well do the asymptotic approximations work? (Optional material)

In section 2.2 we used a general theorem from asymptotic theory stating that the change in two times the log likelihood is approximately chi-square when nested model alternatives are considered. We applied this theorem in a test between two particular nested models; the exponential model and the more general Weibull model.

How well this approximation works will depend on the particular models we are considering. In practice, it is relatively easy to simulate the distribution of this test statistic. Let's consider the Weibull example again. The theorem is a statement about the distribution of the test statistic under H_0. We thus need to simulate data under the null hypothesis that the data come from an exponential distribution. We can only find the distribution for given values of other parameters of the model, in the exponential case, the scale parameter b (we choose to work with the parametesation $f(t) = (1/b)e^{-t/b}$). We have seen earlier how the maximum log likelihood both under H_0 (the exponential model) and under H_1 (the Weibull model) can be computed

(the latter using `optim`). The following function simulates many realisations from H_0, computes the log likelihoods under both model alternatives and stores the value of $2(\ln L_1 - \ln L_0)$ in a vector which is returned as it's value.

```
l <- function(p,t) {
  -sum(dweibull(t,shape=p[1],scale=p[2],log=T))
}

asympt.chisq <- function(n,b=1,m=1000) {
   teststat <- NULL
   for (i in 1:m) {
       t <- rexp(n,rate=1/b) # one realisation of the random sample under H0

       bhat.H0 <- mean(t)    # MLE of b under H0
       lnL.H0 <- -l(c(1,bhat.H0),t=t) # maximum loglik under H0

       lnL.H1 <- -optim(c(1,bhat.H0),l,t=t,lower=c(.001,.001))$value # maximum loglik under
       teststat[i] <- 2*(lnL.H1 - lnL.H0)
   }
   teststat
}
```

A histogram of the simulated distribution of $2(\ln L_1 - \ln L_0)$ for sample sizes of $n = 5, 10, 100$ along with the theoretical chi-square distribution with one degree of freedom generated with the code

```
par(mfrow=c(1,3))
for (n in c(5,10,100)) {
   hist(asympt.chisq(n=n,m=1000),freq=F,main="",
        xlab="Change in two times loglik",
        breaks=seq(0,20,by=.5),xlim=c(0,10))
   curve(dchisq(x,df=1),add=T)
   legend("topright",leg=n,bty="n")
}
```

is shown in Fig. 6. As we see, the approximation works remarkably well also for samples size as small as $n = 10$. For $n = 5$, the true distribution appear to have a heavier right tail than the theoretical chi-square.

4.5 Computing power

For some standard statistical tests such as a one- and two-sample t-tests and tests of equality of two proportions we have seen (Dalgaard, ch. 9) that closed form expression for the power γ of the test are available. If we require a certain power γ, we can then find for example the necessary sample size n by solving the power equation either analytically or using numerical methods.

In some cases we do not have an expression for the power of the test and we need to compute this by other means. Recall that the power of a given test is defined as the probability of rejecting the null hypothesis H_0 given that the alternative hypothesis is correct H_1. Rejection of the null hypothesis is the event that the test statistic falls in the critical region. If we can simulate many realistations of the model under H_1 and compute the corresponding value of the test statistic for each realisation, the power can be computed approximately by dividing the number of rejections of H_0 by the number of simulations.

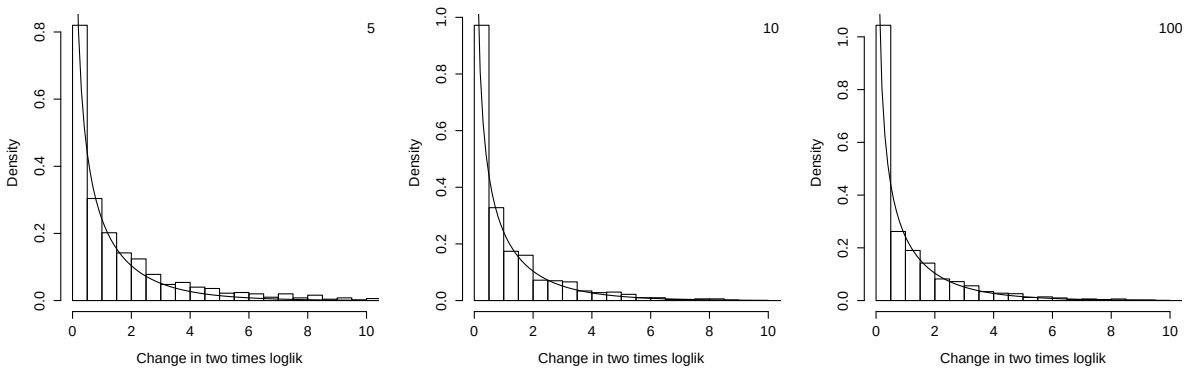

Figure 6: Simulated distribution of two times the change in log maximum likelihood for a test of the exponential against the more general Weibull model for samples sizes of $n = 10, 100, 1000$. The solid line represents the theoretical asymptotic or approximate chi-square distribution of the test statistic.

Consider the logistic regression model

$$\operatorname{logit} p = \beta_0 + \beta_{\text{ab}}\text{ab} + \beta_{\text{age}}\text{age} \tag{40}$$

from assignment 6 for the risk om malaria using age and the log of antibody level as explanatory variables.

```
> malaria
    subject age   ab mal
1         1  15  546   0
2         2  14  268   0
3         3  12  284   0
4         4  15   38   0
5         5  14  827   0
6         6  12  252   0
7         7  12   24   1
8         8  13 1740   0
9         9  13   76   0
10       10  14   83   0
11       11  13   67   0
12       12  15   31   0
13       13  14 1407   1
14       14  15 1949   0
```

The estimate of the regression coefficient for the effect of age was $\beta_{\text{age}} = -0.06$ which corresponds to an oddsratio of malaria per year of increasing age of 0.936, that is, the odds of malaria decrease by approximately 6% per year of increasing age. This effect was not statistically significant from zero, however, as can be seen from the large P-value in the summary of the model.

```
> summary(glm(mal~log(ab)+age,binomial))

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.57234    0.95184   2.702 0.006883 **
log(ab)     -0.68235    0.19552  -3.490 0.000483 ***
```

```
age         -0.06546    0.06772  -0.967 0.333703
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The nullhypothesis that $\beta_{\text{age}} = 0$ can be tested in two ways, either based on an approximate standard normal distribution (under H_0) of $Z = \hat{\beta}_{\text{age}}/\widehat{SE(\beta_{\text{age}})}$ or on the approximate chi-square distribution of the change in deviance when removing age from the model (computed by `drop1()`). Suppose we base our test on the first approximation.

We may now for example ask how the power of the test of this nullhypothesis depends on the true value of β_{age}, say, a more negative value. Note, however, that the overall risk of malaria will go down if we make β_{age} more negative and keep other parameters fixed, notably the intercept which represent the risk of malaria at age= 0 for the current model. The way around this is to introduce a new age variable centered around the mean age of all individuals in the sample and use this as the second explanatory variable in the model.

```
> age2 <- age-mean(age)
> age2
  [1]  6.14  5.14  3.14  6.14  5.14  3.14  3.14  4.14  4.14  5.14  4.14  6.14
...
 [85] -5.86 -4.86 -4.86 -5.86 -4.86 -5.86 -5.86 -4.86 -5.86 -4.86 -4.86 -5.86
 [97] -5.86 -5.86 -4.86 -3.86
> summary(glm(mal~log(ab)+age2,binomial))

Call:
glm(formula = mal ~ log(ab) + age2, family = binomial)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.99236    0.85556   2.329 0.019874 *
log(ab)     -0.68235    0.19552  -3.490 0.000483 ***
age2        -0.06546    0.06772  -0.967 0.333703
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

This is really only a kind of reparameterization of the model which only changes the interpreation of the intercept to the risk of malaria on the logit-scale at an age equal to mean age of the individuals in the sample.

A realisation of the experiment for parameter values equal to the parameter estimates computed from the observed data but using a slightly more negative value for β_{age} can now be simulated as follows

```
n <- nrow(malaria)
beta0 <- 1.99
betaab <- -0.68
betaage <- -0.1  # slightly more negative effect
eta <- beta0 + betaab*log(ab) + betaage*age2 # the linear predictor
p <- 1/(1+exp(-eta)) # prob. of malaria
malsim <- rbinom(n,size=1,prob=p)
```

The vector `malsim` now contains a simulated sample.

```
> malsim
  [1] 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 0 1
 [38] 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1
 [75] 1 0 0 0 0 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 0 1 1 1 0 0
```

Refitting the model to the simulated data, the actual value of the test statistic of interest can be extracted from the `$coefficients` components of the list returned by `summary.glm` (see `?summary.glm`).

```
> malmod <- glm(malsim ~ log(ab) + age,binomial)
> summary(malmod)$coef
             Estimate Std. Error   z value    Pr(>|z|)
(Intercept)  2.3313313  0.9038219  2.579414 0.009896796
log(ab)     -0.7431750  0.2043223 -3.637268 0.000275545
age2        -0.2336564  0.0772452 -3.024865 0.002487438
```

To get at the actual Z-value in this named matrix, we refer to the column and row of interest by their names.

```
> summary(malmod)$coef["age2","z value"]
[1] -3.024865
```

Thus for this particular simulated realisation of the experiment using the above parameter values, the test statistic of the test of interest falls below the lower critical value of the test.

```
> qnorm(.025)
[1] -1.959964
```

The power of the test can be estimated by doing the above say a 1000 times and counting the number of rejections. It makes sense to put everything together as a function as follows.

```
power.logistic.regression <- function(beta0=1.99,betaab=-0.68,betaage=-0.1, m=1000) {
   n <- nrow(malaria)
   eta <- beta0 + betaab*log(ab) + betaage*age2 # the linear predictor
   p <- 1/(1+exp(-eta)) # prob. of malaria
   n.rejections <- 0
   for (i in 1:m) {
      malsim <- rbinom(n,size=1,prob=p)
      malmod <- glm(malsim ~ log(ab)+age2,binomial)
      zsim <- summary(malmod)$coef["age2","z value"]
      if (abs(zsim) > qnorm(0.975))
         n.rejections <- n.rejections + 1
   }
   n.rejections/m
}
```

Note how only certain operations need to be inside the for-loop.

That is all, we can now compute the power for different true values of β_{age}.

```
> power.logistic.regression(betaage=-.1)
[1] 0.291
> power.logistic.regression(betaage=-.2)
[1] 0.836
> power.logistic.regression(betaage=-.3)
[1] 0.983
> power.logistic.regression(betaage=0)
[1] 0.049
```

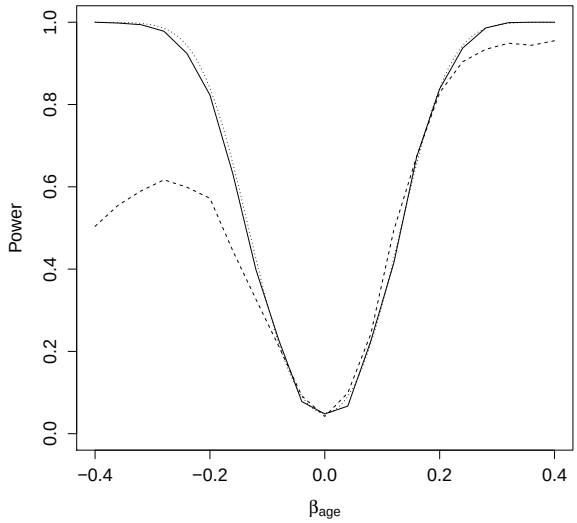

Figure 7: Power of a z-test of $H_0 : \beta_{\text{age}} = 0$ vs $H_0 : \beta_{\text{age}} \neq 0$ for the logistic regression analysis of the malaria data (solid line). Also shown is the power if intercept in the model with non-centered age is kept constant (dashed line) and an approximation based on normality (dotted line).

This appears to work correctly, note that we reject H_0 given that H_0 is true ($\beta_{\text{age}} = 0$) (type I error) with the correct probability.

The above can be turned into a nice graph (Fig. 7) as follows.

```
bb <- seq(-.4,.4,length=21)
pp <- NULL
for (i in 1:length(bb)) {
   pp[i] <- power.logistic.regression(betaage=bb[i])
}
plot(bb,pp,xlab=expression(beta[age]),ylab="Power",type="l",ylim=c(0,1))
```

4.6 Robustness of the t-test

Many statistical procedures such as the t-test, linear regression and analysis of variance, relies on the assumption that the data has a normal distribution. In practice, we seldom know if this assumption holds. Many of these methods are robust, however, in the sense that they behave approximately "the way they should" also if the data is non-normal. More specifically, the t-test is constructed such that the probability of type I error (incorrectly rejecting H_0) is equal to the level of significance α based on the assumption that each observation follows a normal distribution. If this assumption is incorrect, we may get a higher or lower rate of type I error. If the real probability of type I error is close to the nominal level of α we say that the method used is robust.

Suppose that we observe the lifespans of individuals each sex of a given organism and that the observed lifespans $X_1, X_2, \ldots, X_n$ and $Y_1, Y_2, \ldots, Y_n$ in each sex comes from exponential distributions with parameters λ_1 and λ_2, respectively. We are interested in testing if the is any difference between the sexes in mean lifespan. In the absense of any better idea, we use a two-sample t-test to test if there is any difference between the sexes in mean life span.

The real probability of type I error can now be found by simulating exponentially distributed data from the real model under the null hypothesis of no sex difference, then performing a t-test

on the data and counting the number of times we reject H_0.

```
robustness <- function(n,lambda=1,m=1000) {
   n.rejections <- 0
   for (i in 1:m) {
      x <- rexp(n,rate=lambda) # both rates are lambda
      y <- rexp(n,rate=lambda) # since we are simulating from H0
      teststat <- t.test(x,y,var.equal=TRUE)$statistic
      if (abs(teststat)>qt(.975,df=2*n-2))
         n.rejections <- n.rejections + 1
   }
   n.rejections/m
}
```

Simulations for particular sample sizes n (number of observations of each sex) gives the following.

```
> robustness(n=5)
[1] 0.047
> robustness(n=10)
[1] 0.044
> robustness(n=20)
[1] 0.051
```

Note, however, that there is considerable uncertainty in these estimates if they are based on only $m = 1000$ simulations. For example, for $n = 5$, a 95%-confidence interval for true probability of type I error becomes

```
binom.test(47,1000)$conf
[1] 0.03473506 0.06201263
attr(,"conf.level")
[1] 0.95
```

which implies that the true probabilility of type I error, based on these simulations, may be equal to the nominal level of 0.05.

Fig. 8 generated using the code

```
nn <- c(2,3,4,5,10,20,50,100)
pp <- NULL
for (i in 1:length(nn))
   pp[i] <- robustness(n=nn[i],m=100000)
plot(nn,pp,xlab="Sample size n",ylab="Prob. of type I error",ylim=c(.03,.06))
abline(h=.05,lty=2)
```

shows how the probability of type I error depends on n for $2 \leq n \leq 20$ based on more extensive simulations.

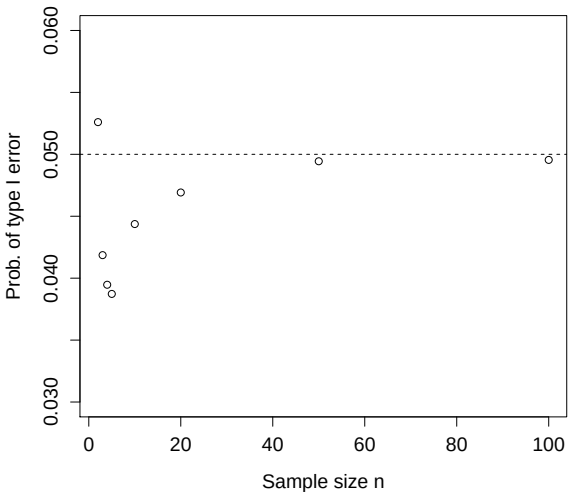

Figure 8: Real probability of type I error for a two-sample t-test for non-normal (exponentially) distributed observations).