Practical likelihood asymptotics with the `likelihoodAsy` package

Ruggero Bellio, Donald A. Pierce

1 Introduction

The `likelihoodAsy` package is designed to simplify the application of some tools for likelihood asymptotics, namely the r^* statistic and the modified profile likelihood. In order to use the methods, the main requirement for the user is the definition of two functions. The first function should return the log likelihood function at a given parameter model, whereas the second function should generate a data set under the assumed parametric statistical model. The user may decide to supply also a function that evaluates the gradient of the log likelihood function. Providing the gradient is not compulsory, but it may lead to safer computation in some models and, at times, substantial saving in computing time.

Both the function for the log likelihood function and the function to generate a data set should have just two arguments. The first argument for either function `theta` is a numeric vector, containing the value of the model parameter. The other argument `data` is a list, and it should contain all the data for the model at hand, starting from the values of the response variable and including covariate values (if any). Any additional information required to evaluate the log likelihood, such as quadrature nodes and weights, should also be included in the `data` list. In the following, we first illustrate the usage of the functions for the r^* formula, and then that for the modified profile likelihood. All the examples are taken from Pierce and Bellio (2015), with the exception of Example 6 which can be found in the help files of the package.

2 Functions for the r^* formula

Before starting with the examples, we load the package.

```
library(likelihoodAsy)
```

This would load also several dependencies.

2.1 Example 1: Inference on the survival function in Weibull regression

The data used in this example are taken from Feigl and Zelen (1965), and they are included in the `MASS` package in the `leuk` data frame.

```
library(MASS)
data(leuk)
```

The function returning the log likelihood at a given parameter point for Weibull regression is given by

```
loglik.Wbl <- function(theta, data)
{
  logy <- log(data$y)
  X <- data$X
  loggam <- theta[1]
  beta <- theta[-1]
  gam <- exp(loggam)
  H <- exp(gam * logy + X %*% beta)
  out <- sum(X %*% beta + loggam + (gam - 1) * logy - H)
  return(out)
}
```

This instance of the user-provided function assumes that the `data` list would include the components `y` and `X`. The former contains the survival times, and the latter the design matrix. Here we focus on the data subset corresponding to `ag="present"`, and take as regressor $\log_{10}$ of the white blood count for each patient of the subset considered. We then define the required list

```
X <- model.matrix(~log(wbc, base=10), data=leuk[leuk$ag=="present",])
data.fz <-list(X = X, y = leuk$time[leuk$ag=="present"])
```

Since use of such a data object is central to the package, we offer some further comments. The organization of the data list is only required to be compatible with the user-provided functions for the likelihood and for generating a dataset. For those unaccustomed to using `R` we note that reading a flat data file with `read.table()`, or more simply with `read.csv()`, results in a data frame that can be used as above. With `read.csv()` it will suffice either to have variable names in a header line, or the option `header = FALSE` results in variables named `V1, V2,...`.

We proceed to define a function for generating a data set from the Weibull regression model. The function returns a copy of the data argument with `y` replaced by a simulated vector.

```
gendat.Wbl <- function(theta, data)
{
  X <- data$X
  n <- nrow(X)
  beta <- theta[-1]
  gam <- exp(theta[1])
  data$y <- (rexp(n) / exp(X %*% beta)) ^ (1 / gam)
  return(data)
}
```

The last function defines the scalar function of inferential interest, here the log survival function for the response at 130, and corresponding to a covariate value of 4.

```
psifcn.Wbl <- function(theta)
{
  beta <- theta[-1]
  gam <- exp(theta[1])
  y0 <- 130
  x0 <- 4
  psi <- -(y0 ^ gam) * exp(beta[1] + x0 * beta[2])
  return(psi)
}
```

Now everything is in place for computing the r^* statistic for the hypothesis $\psi = \log(0.03)$, which is approximately a 95% first-order lower confidence limit. We set the random seed to a given value, here equal to 10, in order to get reproducible results.

```
rs <- rstar(data=data.fz, thetainit = c(0, 0, 0), floglik = loglik.Wbl,
            fpsi = psifcn.Wbl, psival = log(0.03), datagen = gendat.Wbl,
            trace=FALSE, seed=10, psidesc="Log survival function")
rs

psi value under testing
[1]  -3.507
Maximum likelihood estimate of psi
[1]  -2.311
Standard error of maximum likelihood estimate of psi
[1]  0.6106
r statistic
[1]  1.668
r* statistic
[1]  2.104
```

A more detailed set of results is displayed by the `summary` method

```
summary(rs)


Testing based on the r and r* statistics
------------------------------------------------------------
Parameter of interest:         Log survival function
Skovgaard covariances computed with 1000 Monte Carlo draws
psi value under testing:
[1]  -3.507
------------------------------------------------------------
Estimates
```

```
Maximum likelihood estimate of psi:
[1]  -2.311
Standard error of maximum likelihood estimate of psi:
[1]  0.6106
Maximum likelihood estimate of theta:
[1]   0.02174  -8.62676   1.12246
Maximum likelihood estimate of theta under the null:
[1]   0.1525  -8.8990   1.1210
------------------------------------------------------------
Test Statistics
Wald statistic P(r_wald<observed value; 1st order):
[1]   1.9585   0.9749
r statistic    P(r<observed value; 1st order):
[1]   1.6684   0.9524
r* statistic   P(r<observed value; 2nd order):
[1]   2.1035   0.9823
------------------------------------------------------------
Decomposition of high-order adjustment r*-r
NP adjustment  INF adjustment:
[1]   0.2749   0.1602
------------------------------------------------------------
```

Providing the code for the gradient of the log likelihood function may lead to some gain in the computational time. This can be done by defining another function, with the same arguments as `loglik.Wbl`

```
grad.Wbl <- function(theta, data)
{
  logy <- log(data$y)
  X <- data$X
  loggam <- theta[1]
  beta <- theta[-1]
  gam <- exp(loggam)
  H <- exp(gam * logy + X %*% beta)
  score.beta <- t(X) %*% (1 - H)
  score.nu <- sum(1 + gam * logy - gam * H * logy)
  out <- c(score.nu, score.beta)
  return(out)
}
```

This can be checked against a numerical result by using the `grad` function of the `pracma` package (Borchers, 2015), which is included in the package dependencies.

```
cbind(pracma::grad(loglik.Wbl, rs$theta.hyp, data=data.fz),
      grad.Wbl(rs$theta.hyp, data.fz))
```

```
           [,1]       [,2]
[1,] -39.956101 -39.956101
[2,]  -7.047483  -7.047483
[3,] -28.190090 -28.190090
```

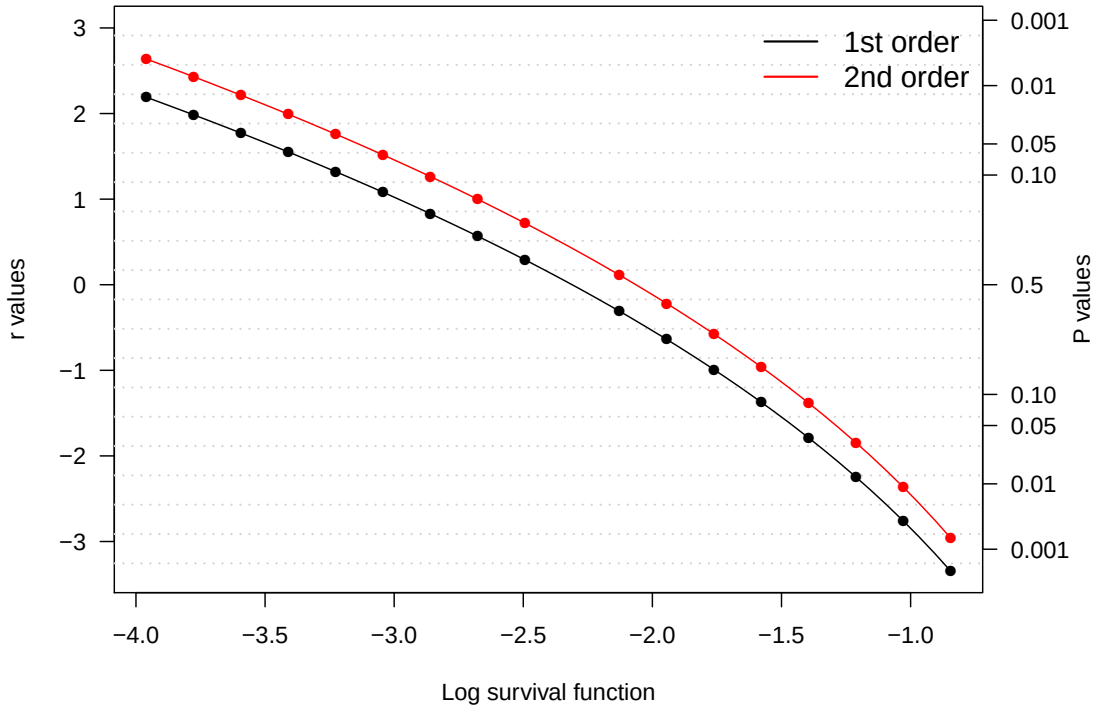
The computation of confidence intervals based on the r^* statistic can be done by calling the `rstar.ci` function.

```
rs.int <- rstar.ci(data=data.fz, thetainit = c(0, 0, 0), floglik = loglik.Wbl,
                   fpsi = psifcn.Wbl, fscore=grad.Wbl, datagen=gendat.Wbl,
                   trace=FALSE, seed=1223, psidesc="Log survival function")
rs.int

Confidence interval calculations based on likelihood asymptotics
1st-order
         90%                          95%                          99%
( -3.487  ,  -1.456 )         ( -3.755  ,  -1.324 )         ( -4.306  ,  -1.091 )
2nd-order
         90%                          95%                          99%
( -3.139  ,  -1.289 )        ( -3.384  ,  -1.169 )        ( -3.9051  ,  -0.9609 )
```

There are both `summary` and `plot` methods for the output.

```
summary(rs.int)

Confidence interval calculations based on likelihood asymptotics
--------------------------------------------------------------------------------
Parameter of interest:         Log survival function
Calculations based on a grid of 17 points
Skovgaard covariances computed with 1000 Monte Carlo draws
--------------------------------------------------------------------------------
1st-order
         90%                          95%                          99%
( -3.487  ,  -1.456 )       ( -3.755  ,  -1.324 )       ( -4.306  ,  -1.091 )
2nd-order
         90%                          95%                          99%
( -3.139  ,  -1.289 )       ( -3.384  ,  -1.169 )       ( -3.9051  ,  -0.9609 )
--------------------------------------------------------------------------------
Decomposition of high-order adjustment
Nuisance parameter adjustment (NP)
   Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
 0.2298   0.2508   0.2669   0.2631   0.2772   0.2865
Information adjustment (INF)
   Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
 0.1572   0.1605   0.1611   0.1607   0.1616   0.1618
--------------------------------------------------------------------------------
```

```
plot(rs.int)
```



The resulting plot represents the behavior of $r(\psi)$ and $r^*(\psi)$ as a function of the parameter of interest ψ, here defined in the `psifcn.Wbl` function. Note that both confidence intervals based on the signed likelihood ratio test computed by employing either the first-order or the second-order asymptotic formula are invariant to interest-preserving parameterization. For example, the 95% confidence interval based on the r^* formula for the survival function (rather than the log survival) at the same point is simply given by

```
print(exp(rs.int$CIrs[2,]), digits=3)

[1] 0.0339 0.3106
```

As a final variation, we consider the case where some observations might be censored. This is readily handled by modifying the log likelihood function (and, possibly, the gradient) and the function for generating data sets. The change required in the former case is simple, as we need to introduce a censoring indicator in the data object and do a minor change to the log likelihood computation, namely

```
loglik.Wbl.cens <- function(theta, data)
{
  logy <- log(data$y)
  X <- data$X
  f <- data$f          ### binary censoring indicator: 0=censored, 1=observed
  loggam <- theta[1]
  beta <- theta[-1]
  gam <- exp(loggam)
  H <- exp(gam * logy + X %*% beta)
  out <- sum(f * (X %*% beta + loggam + (gam - 1) * logy) - H)
  return(out)
}
```

The change required for the data-generating function is more delicate, as we need to specify a censoring model. Here we assume Type II censoring, assuming that the largest 5 failure times are censored at the just-preceding failure time. This is carried out by the following function

```
gendat.Wbl.cens <- function(theta, data)
{
  X <- data$X
  n <- nrow(X)
  beta <- theta[-1]
  gam <- exp(theta[1])
  y <- (rexp(n) / exp(X %*% beta)) ^ (1 / gam)
  maxv <-  n - 5            ### the five largest observation are censored
  ymaxv <- sort(y)[maxv]
  data$y <- ifelse(y < ymaxv, y, ymaxv)
  data$f <- ifelse(y < ymaxv, 1, 0)
  return(data)
}
```

For running the example, we also need to modify the data list:

```
data.fz.cens <-list(X = X, y = leuk$time[leuk$ag=="present"], f=rep(1,nrow(X)))
data.fz.cens$y <- ifelse(data.fz$y <  sort(data.fz$y)[12], data.fz$y,
                         sort(data.fz$y)[12])
data.fz.cens$f <- ifelse(data.fz$y <  sort(data.fz$y)[12], 1,  0)
```

Finally, we compute the confidence intervals for the same parameter of interest considered without censoring. We note in passing that here the log likelihood function is harder to maximize under the null hypothesis, and we employ for the task the constrained optimizer made available by the `alabama` package. This is envoked by setting the argument `constr.opt` to `"alabama"`.

```
rs.int.cens <- rstar.ci(data=data.fz.cens, thetainit = c(0, 0, 0),
                        floglik = loglik.Wbl.cens, fpsi = psifcn.Wbl,
```

```
                        datagen=gendat.Wbl.cens,  constr.opt="alabama",
                    trace=FALSE, seed=1223, psidesc="Log survival function")
summary(rs.int.cens)

Confidence interval calculations based on likelihood asymptotics
-------------------------------------------------------------------------------
Parameter of interest:         Log survival function
Calculations based on a grid of 17 points
Skovgaard covariances computed with 1000 Monte Carlo draws
-------------------------------------------------------------------------------
1st-order
         90%                          95%                          99%
( -2.1110  ,  -0.5947 )       ( -2.3353  ,  -0.5118 )       ( -2.8107  ,  -0.3742 )
2nd-order
         90%                          95%                          99%
( -1.8633  ,  -0.5142 )       ( -2.0632  ,  -0.4409 )       ( -2.5005  ,  -0.3217 )
-------------------------------------------------------------------------------
Decomposition of high-order adjustment
Nuisance parameter adjustment (NP)
   Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.09025  0.12520  0.15000  0.14520  0.16810  0.18090
Information adjustment (INF)
   Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
 0.1864   0.2013   0.2103   0.2074   0.2151   0.2187
-------------------------------------------------------------------------------
```

2.2 Example 2: Autoregressive model of order 1

For this example, the required functions are given as follows. As no covariates are involved, they are relatively simple to code. The log likelihood function is given by

```
likAR1 <- function(theta, data)
{
  y <- data$y
  mu <- theta[1]
  sigma2 <- exp(theta[2] * 2)
  rho <- theta[3]
  n <- length(y)
  Gamma1 <- diag(1 + c(0, rep(rho^2, n-2), 0))
  for(i in 2:n)
    Gamma1[i,i-1]<- Gamma1[i-1,i] <- -rho
  lik <- -n/2 * log(sigma2) + 0.5 * log(1 - rho^2) - 1 / (2 * sigma2) *
          mahalanobis(y, rep(mu,n), Gamma1, inverted = TRUE)
  return(lik)
}
```

and we note that the inverse of the covariance matrix (`Gamma1`) has been coded explicitly. Here `theta[2]` corresponds to the log standard deviation of the error term, so the variance is recovered by `exp(theta[2] * 2)`. It would be preferable to use a different parameterization for the correlation parameter as well, and actully the help file for the `rstar` function illustrates the same example where the Fisher's z-transform is employed. The gradient of the log likelihood function is coded as follows.

```
grAR1 <- function(theta, data)
{
  y <- data$y
  mu <- theta[1]
  sigma2 <- exp(theta[2] * 2)
  rho <- theta[3]
  n <- length(y)
  Gamma1 <- diag( 1 + c(0, rep(rho^2, n-2), 0))
  DGamma1 <- diag(c(0, rep( 2 * rho, n-2), 0))
  for(i in 2:n)
  {
    Gamma1[i,i-1]<- Gamma1[i-1,i] <- -rho
    DGamma1[i,i-1] <- DGamma1[i-1,i] <- -1
  }
  out <- rep(0, length(theta))
  out[1] <-  1 / sigma2 * t(rep(1,n)) %*% Gamma1 %*% (y-mu)
  out[2] <-  -n / (2 * sigma2) + 1 / (2 * sigma2^2) *
             mahalanobis(y, rep(mu,n), Gamma1, inverted = TRUE)
  out[2] <- out[2] * sigma2 * 2
  out[3] <-  -rho / (1 - rho^2) - 1 / (2 * sigma2) *
             mahalanobis(y, rep(mu,n), DGamma1, inverted = TRUE)
  return(out)
}
```

Finally, the following function generates a data set.

```
genDataAR1 <- function(theta, data)
{
  out <- data
  mu <- theta[1]
  sigma <- exp(theta[2])
  rho <- theta[3]
  n <- length(data$y)
  y <- rep(0,n)
  y[1] <- rnorm(1, mu, s = sigma * sqrt(1 / (1 - rho^2)))
  for(i in 2:n)
    y[i] <- mu + rho * (y[i-1] - mu) + rnorm(1) * sigma
  out$y <- y
  return(out)
```

```
}
```

For an illustrative example, we consider the `lh` data set from the `MASS` library, like done in Lozada-Can and Davison (2010).

```
data.AR1 <- list( y = as.numeric(lh) )
```

We proceed to test the hypothesis $H_0 : \rho = 0.765$ by means of the `rstar` function, with the value under testing being the upper limit of a 1st-order 95% level confidence level based on r.

```
rsAR1 <- rstar(data=data.AR1, thetainit = c(0, 0, 0), floglik = likAR1,
               fpsi = function(theta) theta[3], fscore=grAR1,
            psival = 0.765, datagen=genDataAR1, trace=FALSE, seed=10121,
            psidesc="Autocorrelation parameter")
summary(rsAR1)


Testing based on the r and r* statistics
------------------------------------------------------------
Parameter of interest:         Autocorrelation parameter
Skovgaard covariances computed with 1000 Monte Carlo draws
psi value under testing:
[1]  0.765
------------------------------------------------------------
Estimates
Maximum likelihood estimate of psi:
[1]  0.5739
Standard error of maximum likelihood estimate of psi:
[1]  0.1162
Maximum likelihood estimate of theta:
[1]   2.4133  -0.8110   0.5739
Maximum likelihood estimate of theta under the null:
[1]   2.4299  -0.7879   0.7650
------------------------------------------------------------
Test Statistics
Wald statistic P(r_wald<observed value; 1st order):
[1]   -1.64429    0.05006
r statistic    P(r<observed value; 1st order):
[1]   -1.64272    0.05022
r* statistic   P(r<observed value; 2nd order):
[1]   -1.155    0.124
------------------------------------------------------------
Decomposition of high-order adjustment r*-r
NP adjustment  INF adjustment:
[1]   0.3575   0.1298
------------------------------------------------------------
```

For a comparison, we can also test the same hypothesis by means of parametric bootstrap, which is a practical route to validate the result of likelihood asymptotics. For example, we may use 50,000 bootstrap trials, employing the `rstar` function with argument `ronly` set to `TRUE`.

```
rvals <- rep(0, 50000)
set.seed(107)
for(i in 1:length(rvals))
{
  data.boot <- genDataAR1(rsAR1$theta.hyp, data.AR1)
  if(i%%1000==0) cat("i=",i,"\n")
  r.boot <- rstar(data=data.boot, thetainit = rsAR1$theta.hyp, floglik = likAR1,
            fpsi = function(theta) theta[3], fscore=grAR1,
            psival = 0.765, datagen=genDataAR1, trace=FALSE, ronly=TRUE)
  rvals[i] <- r.boot$r
}
```

The computation (not shown) takes a few minutes on most machines. The bootstrap-based p-value agrees with that based on r^*, as can be found by running the command

```
 c(mean(rvals < rsAR1$r), pnorm(rsAR1$rs) )
```

which returns, with the given random seeed, the values 0.1313 and 0.1240 respectively.

2.3 Example 3: Binomial overdispersion

The data for this example (Finney, 1947) are included in the `finndat` data frame. After loading it, we can define the `data.binOD` list, required for usage of the package routines. The log likelihood function will be approximated by Gauss-Hermite quadrature, therefore the quadrature nodes and weights are also included in the `data.binOD` list. The quadrature nodes and weights are obtained from the function `gauss.quad` from the `statmod` package (Smyth et al., 2015).

```
data(finndat)
z <- scale(finndat$z * 10, scale=FALSE)
X <- cbind(rep(1,length(z)), z)
data.binOD  <- list(X=X, den = finndat$den, y = finndat$y,
                    gq=gauss.quad(40,"hermite"))
```

Now we can define the log likelihood function deriving from the assumption of a Gaussian random effect on the linear predictor with logit link function.

```
loglik.binOD <- function(theta, data)
{
  p.range<- function(p, eps=2.22e-15)
   {
      out <- p
      out[p<eps] <- eps
```

```
      out[p>(1-eps)] <- (1-eps)
      return(out)
   }
  y <- data$y
  den <- data$den
  X <- data$X
  gq <- data$gq
  n <- length(y)
  p <- ncol(X)
  beta <- theta[1:p]
  sigma <-  exp(theta[p+1])
  linpred <- X %*% beta
  L <- rep(0,n)
  for (i in 1:n)
  {
    prob <- p.range(plogis(linpred[i] + gq$nodes * sqrt(2)*sigma))
    likq <- y[i] * log(prob) + (den[i] - y[i]) * log(1-prob)
    L[i] <- sum(gq$weights * exp(likq) ) / sqrt(2 * pi)
    }
   return(log(prod(L)))
}
```

This Gaussian quadrature with 40 points yields results that are adequate for numerical differentiation (and many of the more standard routines do not). Anyway, this is an example where gradient code could be essential, though it is not in this instance because of the attention paid to the quadrature method. The gradient is coded as follows.

```
grad.binOD <- function(theta,data)
{
  p.range<- function(p, eps=2.22e-15)
   {
      out <- p
      out[p<eps] <- eps
      out[p>(1-eps)] <- (1-eps)
      return(out)
   }
  y <- data$y
  den <- data$den
  X <- data$X
  gq <- data$gq
  n <- length(y)
  p <- ncol(X)
  beta <- theta[1:p]
  sigma <-  exp(theta[p+1])
  linpred <- X %*% beta
```

```
  L <- rep(0,n)
  LB <- matrix(0, nrow=n, ncol=p+1)
  out <- rep(0,p+1)
  for (i in 1:n)
  {
    prob <- p.range(plogis(linpred[i]+gq$nodes*sqrt(2)*sigma))
    likq <- y[i] * log(prob) + (den[i] - y[i]) * log(1-prob)
    score <- (y[i] - den[i] * prob)
    L[i] <- sum(gq$weights * exp(likq) ) / sqrt(2 * pi)
    LB[i,1] <- sum(gq$weights * exp(likq) * score) / sqrt(2 * pi)
    LB[i,2] <- sum(gq$weights * exp(likq) * score * X[i,2] ) / sqrt(2 * pi)
    LB[i,3] <- sum(gq$weights * exp(likq) * score * gq$nodes *
                   sqrt(2)) / sqrt(2 * pi) * sigma
    out <- out + LB[i,] / L[i]
    }
   return(out)
}
```

The function that generates a data set is as follows.

```
gendat.binOD <- function(theta, data)
 {
   out <- data
   den <- data$den
   X <- data$X
   p <- ncol(X)
   n <- length(data$y)
   beta <- theta[1:p]
   sigma <-  exp(theta[p+1])
   u <- rnorm(n) * sigma
   linpred <- X %*% beta + u
   out$y <- rbinom(n, size=den, prob=plogis(linpred))
   return(out)
 }
```

Now we can apply the `rstar` function for testing the hypothesis that the slope is equal to one. For this model, it seems that 500 Monte Carlo trials are enough for stable results, meaning that the variation in the results is rather limited across repetitions with different random seed.

```
rs <- rstar(data=data.binOD, thetainit=c(0, 0, 0),  floglik=loglik.binOD,
            fscore=grad.binOD,  fpsi=function(theta) return(theta[2]), seed=110,
            trace=FALSE, R=500, psival=1 ,datagen=gendat.binOD,
            psidesc="Regression slope")
summary(rs)
```

```
Testing based on the r and r* statistics
------------------------------------------------------------
Parameter of interest:         Regression slope
Skovgaard covariances computed with 500 Monte Carlo draws
psi value under testing:
[1]  1
------------------------------------------------------------
Estimates
Maximum likelihood estimate of psi:
[1]  1.765
Standard error of maximum likelihood estimate of psi:
[1]  0.388
Maximum likelihood estimate of theta:
[1]   0.4447   1.7645  -0.1740
Maximum likelihood estimate of theta under the null:
[1]   0.48544   1.00000  -0.02455
------------------------------------------------------------
Test Statistics
Wald statistic P(r_wald<observed value; 1st order):
[1]   1.9704   0.9756
r statistic    P(r<observed value; 1st order):
[1]   2.1938   0.9859
r* statistic   P(r<observed value; 2nd order):
[1]   1.9862   0.9765
------------------------------------------------------------
Decomposition of high-order adjustment r*-r
NP adjustment  INF adjustment:
[1]   -0.3198    0.1122
------------------------------------------------------------
```

2.4 Example 4: 2×2 contingency table

This further example shows a simple application to Poisson models for counts, in the special case of a 2×2 table. The log likelihood and the function to simulate a data set are easily defined, by representing the table as a vector of length 4. Note the usage of a continuity correction in the log likelihood function, with each cell of the table perturbed by 0.5.

```r
loglik.Pois <- function(theta, data)
{
  y <- data$y
  y <- y + 0.50 * c(-1,1,1,-1) ### continuity correction
  mu <- exp(data$X %*% theta)
  el <- sum(y * log(mu) - mu)
  return(el)
}
```

```
gendat.Pois <- function(theta, data)
{
  out <- data
  mu <-  exp(data$X %*% theta)
  out$y <- rpois(n=4, lam=mu)
  return(out)
}
```

Let us now define the list representing the observed table `c(15, 9, 7, 13)`.

```
rowf <- c(1, 0, 1, 0)
colf <- c(1, 1, 0, 0)
intf <- c(0, 0, 0, 1)
X <- cbind( rep(1, 4), rowf, colf, intf)
data.2x2  <- list(y = c(15, 9, 7, 13), X=X)
```

The p-value for independence based on the r^* statistic is quickly obtained.

```
rs <- rstar(data=data.2x2, thetainit = c(0, 0, 0, 0), floglik = loglik.Pois,
            fpsi = function(theta) theta[4], psival = 0, datagen=gendat.Pois,
            trace=FALSE, R=50, psidesc="Independence test")
summary(rs)


Testing based on the r and r* statistics
------------------------------------------------------------
Parameter of interest:         Independence test
Skovgaard covariances computed with 50 Monte Carlo draws
psi value under testing:
[1]  0
------------------------------------------------------------
Estimates
Maximum likelihood estimate of psi:
[1]  0.9337
Standard error of maximum likelihood estimate of psi:
[1]  0.6225
Maximum likelihood estimate of theta:
[1]  1.5920  0.4229  0.6592  0.9337
Maximum likelihood estimate of theta under the null:
[1]   2.303e+00   9.755e-07   1.823e-01  -1.529e-20
------------------------------------------------------------
Test Statistics
Wald statistic P(r_wald<observed value; 1st order):
[1]   1.4998   0.9332
r statistic    P(r<observed value; 1st order):
```

```
[1]   1.5208   0.9358
r* statistic   P(r<observed value; 2nd order):
[1]   1.4940   0.9324
-------------------------------------------------------------
Decomposition of high-order adjustment r*-r
NP adjustment  INF adjustment:
[1]   -0.017587   -0.009129
-------------------------------------------------------------
```

Here it is important to note that the model of this example is a full exponential family model, for which the r^* statistic has a close-form analytic expression, and Monte Carlo computation is not required for its computation. The `rstar` function however does not attempt to detect such instances, and the general algorithm (employing Monte Carlo computation) is used nevertheless, even if the outcome of the Monte Carlo computation will cancel out at the end. In such cases, we recommend to set the value of the `R` argument to a small yet not null value, such as the value 50 used here, in order to avoid any numerical problem that may occur in the Monte Carlo computation.

2.5 Example 5: logistic regression

This example features a large-dimensional nuisance parameter. The data are the famous "crying babies dataset", already employed by many authors. The data can be found in the `cond` package (Brazzale, Davison and Reid, 2007).

```
library(cond)
data(babies)
```

We first fit a standard logistic regression model, with fixed effects for `lull` and `day`, and proceed with the definition of the list with all the data information.

```
mod.glm <- glm(formula = cbind(r1, r2) ~ day + lull - 1, family = binomial,
               data = babies)
data.obj <- list(y = babies$r1, den = babies$r1 + babies$r2,
                 X = model.matrix(mod.glm))
```

The data definition is compatible with the functions providing the log likelihood and data simulation. For this model, coding the gradient of the log likelihood is straightforward.

```
loglik.logit<- function(theta, data)
{
  y <- data$y
  den <- data$den
  X <- data$X
  eta <- X %*% theta
  p <- plogis(eta)
  l <- sum(y * log(p) + (den - y) * log(1-p))
```

```
  return(l)
}

grad.logit<- function(theta, data)
{
  y <- data$y
  den <- data$den
  X <- data$X
  eta <- X %*% theta
  p <- plogis(eta)
  out <- t(y - p * den) %*% X
  return(drop(out))
}


gendat.logit<- function(theta, data)
{
  X <- data$X
  eta <- X %*% theta
  p <- plogis(eta)
  out <- data
  out$y <- rbinom(length(data$y), size = data$den, prob = p)
  return(out)
}
```

Here we obtain confidence intervals for the coefficient of `lull`. For the sake of comparison, we do it twice, with and without employing the coded gradient, and record the time spent for the computation.

```
time.with <- system.time( rs.int <- rstar.ci(data=data.obj,
                          thetainit = coef(mod.glm),
                          floglik = loglik.logit, fpsi = function(theta) theta[19],
                          fscore=grad.logit, datagen=gendat.logit, trace=FALSE,
                           psidesc="Coefficient of lull") )
time.without <- system.time( rs.int.no <- rstar.ci(data=data.obj,
                          thetainit = coef(mod.glm),
                          floglik = loglik.logit, fpsi = function(theta) theta[19],
                          datagen=gendat.logit, trace=FALSE,
                          psidesc="Coefficient of lull") )
```

We now have a look at the obtained intervals, and also at computing times.

```
summary(rs.int)

Confidence interval calculations based on likelihood asymptotics
```

```
--------------------------------------------------------------------------------
Parameter of interest:         Coefficient of lull
Calculations based on a grid of 18 points
Skovgaard covariances computed with 1000 Monte Carlo draws
--------------------------------------------------------------------------------
1st-order
         90%                          95%                          99%
( 0.3196  ,  2.7850 )       ( 0.1228  ,  3.0856 )       ( -0.2521  ,   3.7122 )
2nd-order
         90%                          95%                          99%
( 0.1932  ,  2.4770 )       ( 0.01063  ,  2.75520 )       ( -0.3372  ,   3.3455 )
--------------------------------------------------------------------------------
Decomposition of high-order adjustment
Nuisance parameter adjustment (NP)
    Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.28530  -0.27390  -0.24000  -0.21590  -0.17050  -0.07383
Information adjustment (INF)
    Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.07281  -0.06686  -0.05897  -0.05845  -0.05031  -0.04248
--------------------------------------------------------------------------------
```

```
time.with

   user  system elapsed
   1.53    0.04    1.60

time.without

   user  system elapsed
 14.445   0.538  15.270
```

There is a close agreement between the results obtained here and those provided by the `cond` package. The latter are readily computed.

```
res.cond <- cond(object = mod.glm, offset = lullyes)
summary(res.cond)


 Formula:  cbind(r1, r2) ~ day + lull - 1
 Family:  binomial
 Offset:  lullyes

          Estimate   Std. Error
uncond.       1.432       0.7341
cond.         1.270       0.6888
```

```
Confidence intervals
--------------------
 level = 95 %
                                          lower two-sided upper
Wald pivot                               -0.006514           2.871
Wald pivot (cond. MLE)                   -0.080180           2.620
Likelihood root                           0.122800           3.086
Modified likelihood root                  0.010690           2.755
Modified likelihood root (cont. corr.)   -0.152300           3.097

Diagnostics:
-----------
    INF      NP
0.07596 0.28882

 Approximation based on 20 points
```

3 Functions for the Modified Profile Likelihood (MPL)

The `likelihoodAsy` package has also two functions for the Modified Profile Likelihood and the Profile Likelihood, the `logMPL` and `logPL` functions respectively. Both the two functions evaluate the value of the target log likelihood at a given value of the parameter of interest, of dimension possibly larger than one. The two functions return the value of the log likelihood at a given value parameter of interest. Either function value can be multiplied by -1 to ease usage with general-purposes optimizers, which typically performs minimization rather than maximization. Indeed, differently from the functions for the r^* statistic, their optimization and graphical display are left to the user, who can employ for this task the functionality available within `R`. The only difference in the design of the functions with respect to the functions for the r^* computation lies in the way the parameter of interest are represented in the input argument. These functions handle only the case where the parameter of interest is a subset of the vector representing the model parameters, with no need to define a specific function for the parameter of interest like in the `rstar` function. A numeric vector `indpsi` containing the indexes (coordinates) of the parameter of interest is used instead.

3.1 Example 5: logistic regression

Let us consider again the crying babies dataset. Here the parameter is scalar, so we are able to plot the two profile log likelihoods. The data list is the same `data.obj` defined above. We first proceed to the numerical optimization of both functions, by means of the `nlminb` optimizer.

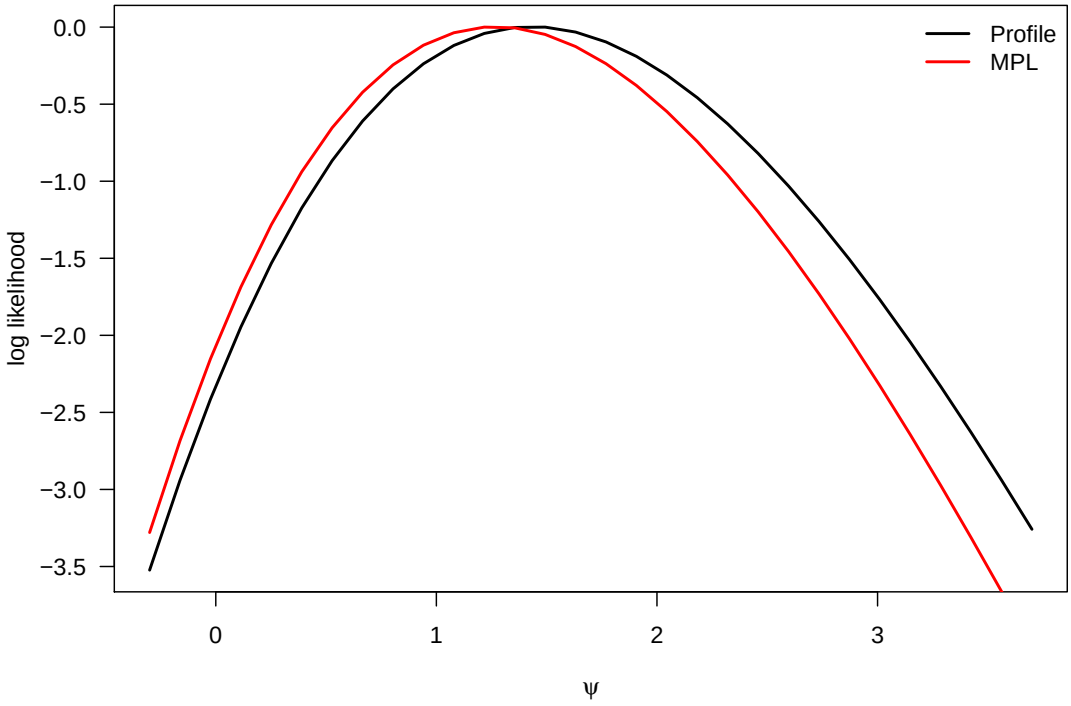
```
max.prof <- nlminb(0, logPL, data=data.obj, thetainit=coef(mod.glm),
                   floglik=loglik.logit, fscore=grad.logit, indpsi=19, trace=FALSE,
                   minus=TRUE)
max.mpl <- nlminb(0, logMPL, data=data.obj, mle=coef(mod.glm),
```

```
                  floglik=loglik.logit, fscore=grad.logit, datagen=gendat.logit,
                  indpsi=19, R=50, seed=2020, trace=FALSE, minus=TRUE)
c(max.prof$par, max.mpl$par)

[1] 1.432371 1.269897
```

Like before, theory of exponential family models suggests that the MPL formula would not require any Monte Carlo computation, but the software does not recognize this fact. Once again, there is no need to employ a large simulation sample size. The final lines of code obtain the plot the two log likelihoods.

```
psi.vals <- seq(-0.3, 3.7, l=30)
obj.prof <- sapply(psi.vals, logPL, data=data.obj, thetainit=coef(mod.glm),
                floglik=loglik.logit, fscore=grad.logit, indpsi=19)
obj.mpl <- sapply(psi.vals, logMPL, data=data.obj, mle=coef(mod.glm),
                floglik=loglik.logit, fscore=grad.logit, datagen=gendat.logit,
                indpsi=19, R=50, seed=2020)
```

```
par(pch="s")
plot(psi.vals, obj.prof - max(obj.prof), type="l", xlab=expression(psi),
     ylab="log likelihood", lwd=2, las=1)
lines(psi.vals, obj.mpl  - max(obj.mpl), col="red", lwd=2)
legend("topright", col=c(1, 2), lty=1, lwd=2, legend=c("Profile","MPL"), bty="n")
```



Again, by plotting the `res.cond` object it is possible to verify the agreement with the results provided by the `cond` package.

3.2 Example 6: random intercept model

As a further example we consider a simple linear mixed model, with only random intercepts. The log likelihood function is taken from Wood (2006), and for speeding up the computation we code the gradient as well. The data generation function is the simplest of the three.

```
logLikLme<- function(theta, data)
{
  X <- data$X
  Z <- data$Z
  y <- data$y
  beta <- theta[1:ncol(X)]
  sigma.b <- theta[ncol(X)+1]
  sigma <- theta[ncol(X)+2]
  n <- nrow(X)
  V <- tcrossprod(Z) * sigma.b^2 + diag(n) * sigma^2
```

```
  L <- chol(V)
  XL <- backsolve(L, X, transpose=TRUE)
  yL <- backsolve(L, y, transpose=TRUE)
  out<- - sum(log(diag(L))) - sum( (yL-XL %*% beta)^2) / 2
  return(out)
}


gradLikLme <- function(theta, data)
{
  X <- data$X
  Z <- data$Z
  y <- data$y
  beta <- theta[1:ncol(X)]
  sigma.b <- theta[ncol(X)+1]
  sigma <- theta[ncol(X)+2]
  n <- nrow(X)
  V <- tcrossprod(Z) * sigma.b^2 + diag(n) * sigma^2
  L <- chol(V)
  XL<- backsolve(L, X, transpose=TRUE)
  yL<- backsolve(L, y, transpose=TRUE)
  out <- rep(0, length(theta))
  out[1:ncol(X)] <-  t(yL-XL %*% beta)  %*% XL
  ni<- as.vector(t(Z) %*% rep(1,n))
  Zv<- matvec(Z, sqrt(1/(sigma^2 + sigma.b^2 * ni)))
  V1 <- diag(n) / sigma^2 - tcrossprod(Zv) * sigma.b^2 / sigma^2
  Vb <- tcrossprod(Z) * 2 * sigma.b
  Vs <- diag(n) * 2 * sigma
  Mb <- V1 %*% Vb
  Ms <- V1 %*% Vs
  r <- as.vector(y - X %*% beta)
  out[ncol(X)+1] <- -sum(diag(Mb)) / 2  +
                    as.numeric( t(r) %*% Mb %*% V1 %*% r) / 2
  out[ncol(X)+2] <- -sum(diag(Ms)) / 2  +
                    as.numeric( t(r) %*% Ms %*% V1 %*% r) / 2
  return(out)
}


genDataLme <- function(theta, data)
{
  out <- data
  X <- data$X
  Z <- data$Z
  y <- data$y
```

```
  beta <- theta[1:ncol(X)]
  sigma.b <- theta[ncol(X)+1]
  sigma <- theta[ncol(X)+2]
  n <- nrow(X)
  mu <- X %*% beta
  b <- rnorm(ncol(Z), s=sigma.b)
  e <- rnorm(nrow(Z), s=sigma)
  out$y <- mu + e + Z %*% b
  return(out)
}
```

We take the `sleepstudy` data from the `lme4` package (Bates et al., 2014) for this example.

```
library(lme4)
fm1R <- lmer(Reaction ~ Days + (1|Subject), sleepstudy)
sleepdata <- list(X=model.matrix(Reaction ~ Days, sleepstudy),
                  Z=model.matrix(Reaction ~ factor(Subject)-1, sleepstudy),
                  y=sleepstudy$Reaction)
```

We start by computing the maximum likelihood estimates.

```
mleFull <- optim( c(250, 10, 30, 30), logLikLme, gr=gradLikLme,
                  data=sleepdata, method="BFGS",
                  control=list(fnscale=-1))
```

Then we maximize the MPL, employing just 100 Monte Carlo simulations for its computation. The number of trials seems to suffice, due to the curved exponential family structure of linear mixed models.

```
mleM <- optim(mleFull$par[3:4], logMPL, data=sleepdata, mle=mleFull$par,
              floglik=logLikLme, fscore=gradLikLme, minus=TRUE,
              indpsi=3:4, datagen=genDataLme, trace=FALSE, seed=11, R=100)
```

The optimization takes up to a few minutes on most machines. The result (not shown) agrees well with what found by the `lmer` function, using the default REML estimation method.

References

[1] Bates, D., Maechler, M., Bolker, B. and Walker, S. (2014). `lme4`: Linear mixed-effects models using `Eigen` and `S4`. `R` package version 1.1-7. `http://CRAN.R-project.org/package=lme4`.

[2] Borchers, H. W. (2015). `pracma`: Practical Numerical Math Functions. `R` package version 1.8.3. `http://CRAN.R-project.org/package=pracma`

[3] Brazzale, A.R., Davison, A.C. and Reid, N. (2007). *Applied Asymptotics: Case Studies in Small-Sample Statistics*. Cambridge University Press, Cambridge. `http://statwww.epfl.ch/AA/`

[4] Lozada-Can, C. and Davison, A.C. (2010). Three examples of accurate likelihood inference. *The American Statistician*, **64**, 131–139.

[5] Feigl, P. and Zelen, M. (1965). Estimation of exponential survival probabilities with concomitant information. *Biometrics*, **21**, 826–838.

[6] Finney, D.J. (1947). *Probit Analysis: A Statistical Treatment of the Sigmoid Response Curve*. Cambridge University Press, London and New York.

[7] Pierce, D.A. and Bellio, R. (2017). Modern likelihood-frequentist inference. *International Statistical Review*, **85**, 519–541.

[8] Smyth, G., Hu, Y., Dunn, P., Phipson, B. and Chen, Y. (2015). `statmod`: Statistical Modeling. `R` package version 1.4.21. `http://CRAN.R-project.org/package=statmod`

[9] Wood, S. (2006). *Generalized Additive Models: An Introduction with R*. Chapman & Hall/CRC, Boca Raton.