Maximum likelihood estimation and analysis with the `bbmle` package

Ben Bolker

June 17, 2011

Contents

Note: I have suppressed the continuation character (+) in the R examples throughout this document, as I find it easier to read/cut-and-paste where necessary.

The `bbmle` package, designed to simplify maximum likelihood estimation and analysis in R, extends and modifies the `mle` function and class in the `stats4` package that comes with R by default. `mle` is in turn a wrapper around the `optim` function in base R. The maximum-likelihood-estimation function and class in `bbmle` are both called `mle2`, to avoid confusion and conflict with the original functions in the `stats4` package. The major differences between `mle` and `mle2` are:

- `mle2` is more robust, with additional warnings (e.g. if the Hessian can't be computed by finite differences, `mle2` returns a fit with a missing Hessian rather than stopping with an error)
- `mle2` uses a `data` argument to allow different data to be passed to the negative log-likelihood function

- `mle2` has a formula interface like that of (e.g.) `gls` in the `nlme` package. For relatively simple models the formula for the maximum likelihood can be written in-line, rather than defining a negative log-likelihood function. The formula interface also simplifies fitting models with categorical variables. Models fitted using the formula interface also have applicable `predict` and `simulate` methods.
- `bbmle` defines `anova`, `AIC`, `AICc`, and `BIC` methods for `mle2` objects, as well as `AICtab`, `BICtab`, `AICctab` functions for producing summary tables of information criteria for a set of models.

Other packages with similar functionality (extending GLMs in various ways) are

- on CRAN: `aod` (overdispersed models such as beta-binomial); `vgam` (a wide range of models); `betareg` (beta regression); `pscl` (zero-inflated, hurdle models); `maxLik` (another general-purpose maximizer, with a different selection of optimizers)
- In Jim Lindsey's code repository (http://popgen.unimaas.nl/~jlindsey/rcode.html): `gnlr` and `gnlr3`

1 Example: *Orobanche*/overdispersed binomial

This example will use the classic data set on *Orobanche* germination from Crowder (1978) (you can also use `glm(...,family="quasibinomial")` or the `aod` package to analyze these data).

1.1 Test basic fit to simulated beta-binomial data

First, generate a single beta-binomially distributed set of points as a simple test.

Load the `emdbook` package to get functions for the beta-binomial distribution (random-deviate function `rbetabinom` — these functions are also available in Jim Lindsey's `rmutil` package).

```
> library(emdbook)
```

Generate random deviates from a random beta-binomial:

```
> set.seed(1001)
> x1 = rbetabinom(n=1000,prob=0.1,size=50,theta=10)
```

Load the package:

```
> library(bbmle)
```

Construct a simple negative log-likelihood function:

```
> mtmp <- function(prob,size,theta) {
   -sum(dbetabinom(x1,prob,size,theta,log=TRUE))
 }
```

Fit the model — use `data` to pass the `size` parameter (since it wasn't hard-coded in the `mtmp` function):

```
> (m0 <- mle2(mtmp,start=list(prob=0.2,theta=9),data=list(size=50)))

Call:
mle2(minuslogl = mtmp, start = list(prob = 0.2, theta = 9), data = list(size = 50))

Coefficients:
      prob      theta
 0.1030974 10.0758090

Log-likelihood: -2723.5
```

The `summary` method for `mle2` objects shows the parameters; approximate standard errors (based on quadratic approximation to the curvature at the maximum likelihood estimate); and a test of the parameter difference from zero based on this standard error and on an assumption that the likelihood surface is quadratic (or equivalently that the sampling distribution of the estimated parameters is normal).

```
> summary(m0)

Maximum likelihood estimation

Call:
mle2(minuslogl = mtmp, start = list(prob = 0.2, theta = 9), data = list(size = 50))

Coefficients:
        Estimate Std. Error z value     Pr(z)
prob   0.1030974  0.0031626  32.599 < 2.2e-16 ***
theta 10.0758090  0.6213010  16.217 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

-2 log L: 5446.995
```

Construct the likelihood profile (you can apply `confint` directly to `m0`, but if you're going to work with the likelihood profile [e.g. plotting, or looking for confidence intervals at several different α values] then it is more efficient to compute the profile once):

```
> p0 <- profile(m0)
```

Compare the confidence interval estimates based on inverting a spline fit to the profile (the default); based on the quadratic approximation at the maximum likelihood estimate; and based on root-finding to find the exact point where the profile crosses the critical level.

```
> confint(p0)
```

```
           2.5 %     97.5 %
prob   0.09709228  0.1095103
theta 8.91708221 11.3559589
```

```
> confint(m0,method="quad")
```

```
           2.5 %     97.5 %
prob   0.09689876  0.1092961
theta 8.85808147 11.2935366
```

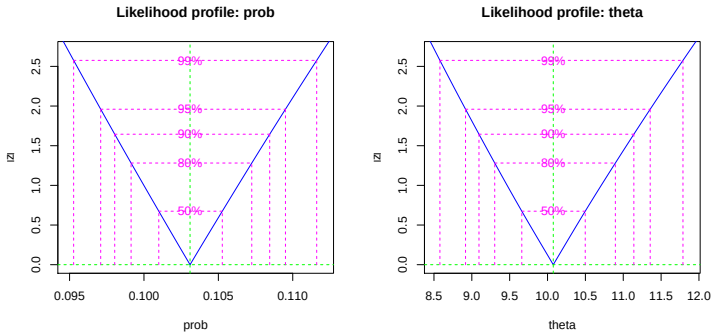
```
> confint(m0,method="uniroot")
```

```
           2.5 %     97.5 %
prob   0.09709185  0.1095099
theta 8.91691020 11.3559746
```

All three types of confidence limits are similar.

Plot the profiles:

```
> par(mfrow=c(1,2))
> plot(p0,plot.confstr=TRUE)
```



By default, the plot method for likelihood profiles displays the square root of the the deviance difference (twice the difference in negative log-likelihood from the best fit), so it will be V-shaped for cases where the quadratic approximation works well (as in this case). (For a better visual estimate of whether the profile is quadratic, use the `absVal=FALSE` option to the `plot` method.)

You can also request confidence intervals calculated using `uniroot`, which may be more exact when the profile is not smooth enough to be modeled accurately by a spline. However, this method is also more sensitive to numeric problems.

Instead of defining an explicit function for `minuslogl`, we can also use the formula interface. The formula interface assumes that the density function given (1) has `x` as its first argument (if the distribution is multivariate, then `x` should be a matrix of observations) and (2) has a `log` argument that will return the log-probability or log-probability density if `log=TRUE`. Some of the extended functionality (prediction etc.) depends on the existence of an `s-` variant function for the distribution that returns (at least) the mean and median as a function of the parameters (currently defined: `snorm`, `sbinom`, `sbeta`, `snbinom`, `spois`).

```
> m0f <- mle2(x1~dbetabinom(prob,size=50,theta),
              start=list(prob=0.2,theta=9),data=data.frame(x1))
```

Note that you must specify the data via the `data` argument when using the formula interface. This may be slightly more unwieldy than just pulling the data from your workspace when you are doing simple things, but in the long run it makes tasks like predicting new responses much simpler.

It's convenient to use the formula interface to try out likelihood estimation on the transformed parameters:

```
> m0cf <- mle2(x1~dbetabinom(prob=plogis(lprob),size=50,theta=exp(ltheta)),
              start=list(lprob=0,ltheta=2),data=data.frame(x1))
> confint(m0cf,method="uniroot")

           2.5 %    97.5 %
lprob  -2.229963 -2.095757
ltheta  2.187950  2.429744

> confint(m0cf,method="spline")

           2.5 %    97.5 %
lprob  -2.229963 -2.095756
ltheta  2.187948  2.429742
```

In this case the answers from `uniroot` and `spline` (default) methods barely differ.

1.2 Real data (*Orobanche*, Crowder (1978))

Data are incorporated in the `aod` package:

```
> library(aod)

Package aod, version 1.2

> summary(orob1)

  dilution       n                y
 1/1  :6   Min.   :  7.00   Min.   : 0.00
 1/25 :5   1st Qu.: 17.50   1st Qu.: 8.00
```

```
 1/625:5   Median : 47.50   Median :13.50
           Mean   : 44.25   Mean   :27.19
           3rd Qu.: 57.50   3rd Qu.:46.25
           Max.   :104.00   Max.   :90.00
```

Now construct a negative log-likelihood function that differentiates among groups:

```
> ML1 <- function(prob1,prob2,prob3,theta,x) {
   prob <- c(prob1,prob2,prob3)[as.numeric(x$dilution)]
   size <- x$n
   -sum(dbetabinom(x$y,prob,size,theta,log=TRUE))
 }
```

Results from Crowder (1978):

model	prob1	prob2	prob3	theta	sd.prob1	sd.prob2	sd.prob3	NLL
prop diffs	0.132	0.871	0.839	78.424	0.027	0.028	0.032	−34.991
full model								−34.829
homog model								−56.258

```
> (m1 <- mle2(ML1,start=list(prob1=0.5,prob2=0.5,prob3=0.5,theta=1),
     data=list(x=orob1)))

Call:
mle2(minuslogl = ML1, start = list(prob1 = 0.5, prob2 = 0.5,
    prob3 = 0.5, theta = 1), data = list(x = orob1))

Coefficients:
     prob1      prob2      prob3      theta
 0.1318187  0.8706259  0.8382504 73.7968323

Log-likelihood: -34.99

Warning: optimization did not converge (code 1: )
```

Or:

```
> ## would prefer ~dilution-1, but problems with starting values ...
> (m1B <- mle2(y~dbetabinom(prob,size=n,theta),
              param=list(prob~dilution),
              start=list(prob=0.5,theta=1),
     data=orob1))
```

The result warns us that the optimization has not converged; we also don't match Crowder's results for θ exactly. We can fix this by setting `parscale` appropriately.

```
> (m2 <- mle2(ML1,start=as.list(coef(m1)),
           control=list(parscale=coef(m1)),
           data=list(x=orob1)))

Call:
mle2(minuslogl = ML1, start = as.list(coef(m1)), data = list(x = orob1),
    control = list(parscale = coef(m1)))

Coefficients:
     prob1      prob2      prob3      theta
 0.1322123  0.8708913  0.8393195 78.4227905

Log-likelihood: -34.99
```

Calculate likelihood profile (restrict the upper limit of θ, simply because it will make the picture below a little bit nicer):

```
> p2 <- profile(m2,prof.upper=c(Inf,Inf,Inf,theta=2000))
```

Get the curvature-based parameter standard deviations (which Crowder used rather than computing likelihood profiles):

```
> round(stdEr(m2),3)

 prob1  prob2  prob3  theta
 0.028  0.029  0.032 74.223
```

We are slightly off Crowder's numbers — rounding error?

Crowder also defines a variance (overdispersion) parameter $\sigma^2 = 1/(1+\theta)$.

```
> sqrt(1/(1+coef(m2)["theta"]))

    theta
0.1122089
```

Using the delta method (via the `deltavar` function in the `emdbook` package) to approximate the standard deviation of σ:

```
> sqrt(deltavar(sqrt(1/(1+theta)),meanval=coef(m2)["theta"],
          vars="theta",Sigma=vcov(m2)[4,4]))

[1] 0.0524311
```

Another way to fit in terms of σ rather than θ is to compute $\theta = 1/\sigma^2 - 1$ on the fly in a formula:

```
> m2b <- mle2(y~dbetabinom(prob,size=n,theta=1/sigma^2-1),
             data=orob1,
             parameters=list(prob~dilution,sigma~1),
             start=list(prob=0.5,sigma=0.1))
> round(stdEr(m2b)["sigma"],3)
```

```
sigma
0.052
```

```
> p2b <- profile(m2b,prof.lower=c(-Inf,-Inf,-Inf,0))
```

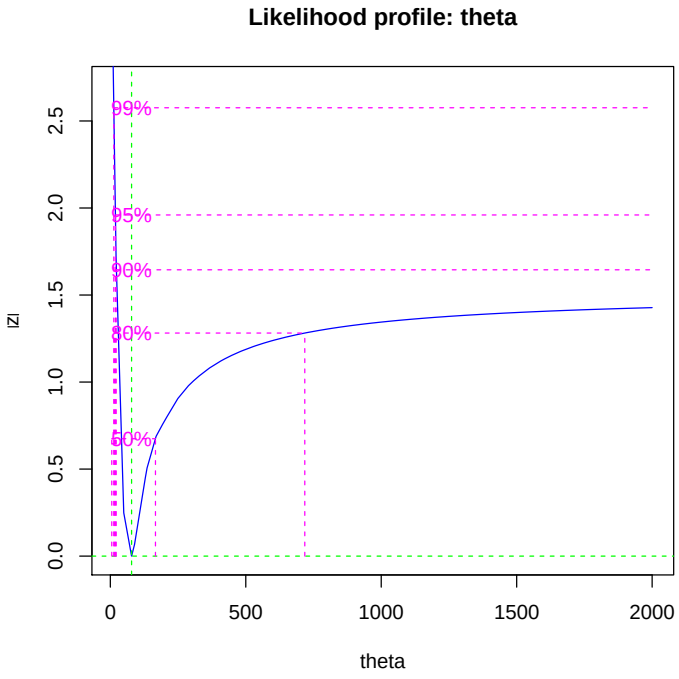
As might be expected since the standard deviation of σ is large, the quadratic approximation is poor:

```
> r1 <- rbind(confint(p2)["theta",],
              confint(m2,method="quad")["theta",])
> rownames(r1) <- c("spline","quad")
> r1
```

```
           2.5 %   97.5 %
spline  19.67216       NA
quad   -67.05101 223.8966
```

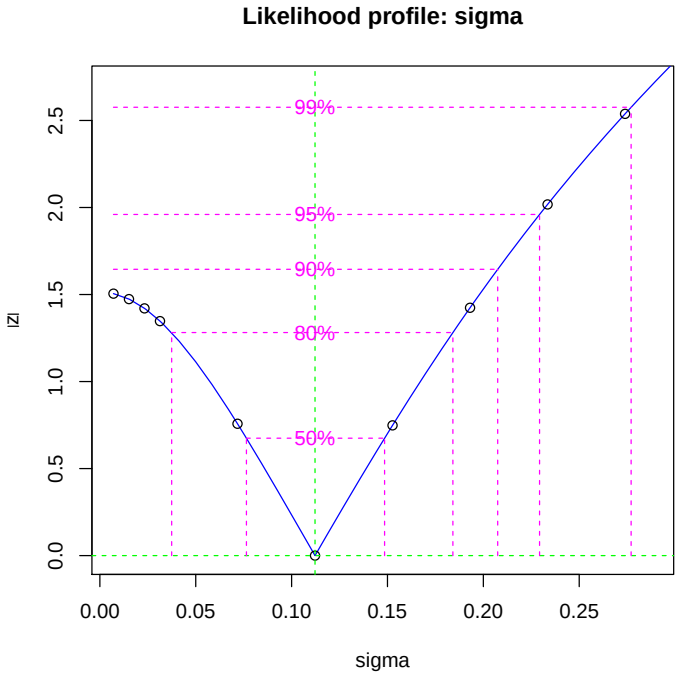
Plot the profile:

```
> plot(p2,which="theta",plot.confstr=TRUE)
```



What does the profile for σ look like?

```
> plot(p2b,which="sigma",plot.confstr=TRUE,
      show.points=TRUE)
```



Now fit a homogeneous model:

```
> ml0 <- function(prob,theta,x) {
   size <- x$n
   -sum(dbetabinom(x$y,prob,size,theta,log=TRUE))
 }
> m0 <- mle2(ml0,start=list(prob=0.5,theta=100),
           data=list(x=orob1))
```

The log-likelihood matches Crowder's result:

```
> logLik(m0)

'log Lik.' -56.25774 (df=2)
```

It's easier to use the formula interface to specify all three of the models fitted by Crowder (homogeneous, probabilities differing by group, probabilities and overdispersion differing by group):

```
> m0f <- mle2(y~dbetabinom(prob,size=n,theta),
             parameters=list(prob~1,theta~1),
             data=orob1,
             start=list(prob=0.5,theta=100))
> m2f <- mle2(y~dbetabinom(prob,size=n,theta),
```

```
             parameters=list(prob~dilution,theta~1),
             data=orob1,
             start=list(prob=0.5,theta=78.424))
> m3f <- mle2(y~dbetabinom(prob,size=n,theta),
             parameters=list(prob~dilution,theta~dilution),
             data=orob1,
             start=list(prob=0.5,theta=78.424))
```

anova runs a likelihood ratio test on nested models:

```
> anova(m0f,m2f,m3f)

Likelihood Ratio Tests
Model 1: m0f, y~dbetabinom(prob,size=n,theta): prob~1, theta~1
Model 2: m2f, y~dbetabinom(prob,size=n,theta): prob~dilution, theta~1
Model 3: m3f, y~dbetabinom(prob,size=n,theta): prob~dilution,
          theta~dilution
  Tot Df Deviance   Chisq Df Pr(>Chisq)
1      2  112.515
2      4   69.981 42.5341  2  5.805e-10 ***
3      6   69.981  0.0008  2     0.9996
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The various ICtab commands produce tables of information criteria, optionally sorted and with model weights.

```
> AICtab(m0f,m2f,m3f,weights=TRUE,delta=TRUE,sort=TRUE)

    dAIC df weight
m2f  0.0 4  0.881
m3f  4.0 6  0.119
m0f 38.5 2  <0.001

> BICtab(m0f,m2f,m3f,delta=TRUE,nobs=nrow(orob1),sort=TRUE,weights=TRUE)

    dBIC df weight
m2f  0.0 4  0.9412
m3f  5.5 6  0.0588
m0f 37.0 2  <0.001

> AICctab(m0f,m2f,m3f,delta=TRUE,nobs=nrow(orob1),sort=TRUE,weights=TRUE)

    dAICc df weight
m2f  0.0  4  0.99222
m3f  9.7  6  0.00778
m0f 35.8  2  < 0.001
```

2 Example: reed frog size predation

Data from an experiment by Vonesh (Vonesh and Bolker, 2005)

```
> frogdat <- data.frame(
    size=rep(c(9,12,21,25,37),each=3),
    killed=c(0,2,1,3,4,5,rep(0,4),1,rep(0,4)))
> frogdat$initial <- rep(10,nrow(frogdat))
```

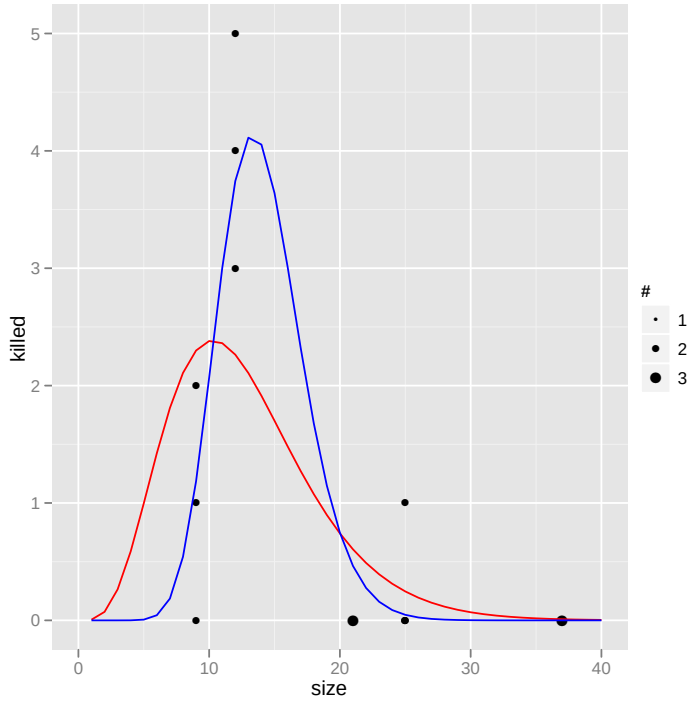
```
> library(ggplot2)
```

```
> gg1 <- ggplot(frogdat,aes(x=size,y=killed))+geom_point()+
        stat_sum(aes(size=factor(..n..)))+
        labs(size="#")+scale_x_continuous(limits=c(0,40))
```

```
> m3 <- mle2(killed~dbinom(prob=c*(size/d)^g*exp(1-size/d),
    size=initial),data=frogdat,start=list(c=0.5,d=5,g=1))
> pdat <- data.frame(size=1:40,initial=rep(10,40))
> pdat1 <- data.frame(pdat,killed=predict(m3,newdata=pdat))
```

```
> m4 <- mle2(killed~dbinom(prob=c*((size/d)*exp(1-size/d))^g,
    size=initial),data=frogdat,start=list(c=0.5,d=5,g=1))
> pdat2 <- data.frame(pdat,killed=predict(m4,newdata=pdat))
```

```
> print(gg1 + geom_line(data=pdat1,colour="red")+
        geom_line(data=pdat2,colour="blue"))
```



```
> coef(m4)
```

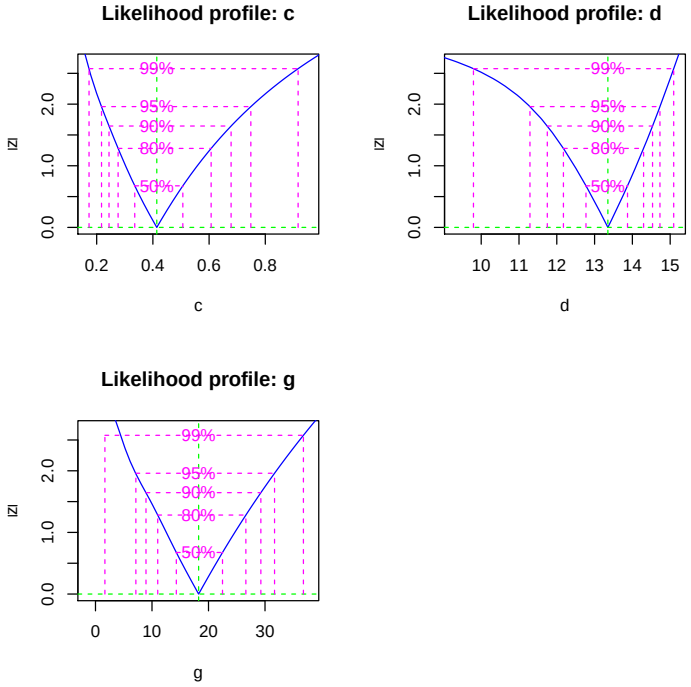
```
         c          d          g
 0.4138847 13.3517574 18.2511264
```

```
> prof4 <- profile(m4)
```

Three different ways to draw the profile:
(1) Built-in method (base graphics):

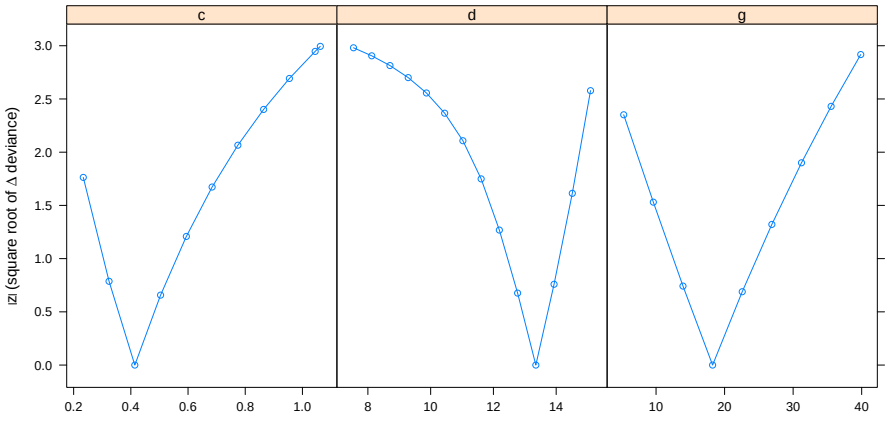
```
> print(plot(prof4))
```

```
NULL
```



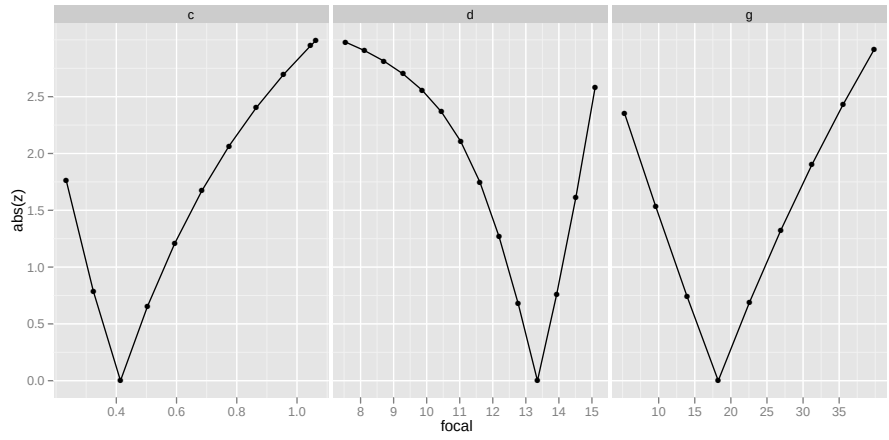
(2) Using `xyplot` from the `lattice` package:

```
> prof4_df <- as.data.frame(prof4)
> library(lattice)
> print(xyplot(abs(z)~focal|param,data=prof4_df,
        subset=abs(z)<3,
        type="b",
        xlab="",
        ylab=expression(paste(abs(z),
            " (square root of ",Delta," deviance)")),
        scale=list(x=list(relation="free")),
              layout=c(3,1)))
```



(3) Using `ggplot` from the `ggplot2` package:

```
> print(ggplot(subset(prof4_df,abs(z)<3),
       aes(x=focal,y=abs(z)))+geom_line()+
      geom_point()+
      facet_grid(~param,scale="free_x"))
```



Additions/enhancements/differences from `stats4::mle`

- `anova` method
- warnings on convergence failure
- more robust to non-positive-definite Hessian; can also specify `skip.hessian` to skip Hessian computation when it is problematic
- when profiling fails because better value is found, report new values

- can take named vectors as well as lists as starting parameter vectors
- added `AICc`, `BIC` definitions, `ICtab` functions
- added `"uniroot"` and `"quad"` options to `confint`
- more options for colors and line types etc etc. The old arguments are:

```
> function (x, levels, conf = c(99, 95, 90, 80, 50)/100, nseg = 50,
            absVal = TRUE, ...) {}
```

  The new one is:

```
> function (x, levels, which=1:p, conf = c(99, 95, 90, 80, 50)/100, nseg = 50,
            plot.confstr = FALSE, confstr = NULL, absVal = TRUE, add = FALSE,
            col.minval="green", lty.minval=2,
            col.conf="magenta", lty.conf=2,
            col.prof="blue", lty.prof=1,
            xlabs=nm, ylab="score",
            show.points=FALSE,
            main, xlim, ylim, ...) {}
```

  `which` selects (by character vector or numbers) which parameters to plot: `nseg` does nothing (even in the old version); `plot.confstr` turns on the labels for the confidence levels; `confstr` gives the labels; `add` specifies whether to add the profile to an existing plot; `col` and `lty` options specify the colors and line types for horizontal and vertical lines marking the minimum and confidence vals and the profile curve; `xlabs` gives a vector of x labels; `ylab` gives the y label; `show.points` specifies whether to show the raw points computed.

- `mle.options()`
- `data` argument
- handling of names in argument lists
- can use alternative optimizers (`nlminb`, `nlm`, `constrOptim`, `optimx`, `optimize`)
- uses code from `numDeriv` package to compute Hessians rather than built-in optimizer code
- by default, uses `MASS::ginv` (generalized inverse) rather than `solve` to invert Hessian (more robust to positive-semidefinite Hessians ...)
- can use `vecpar=TRUE` (and `parnames()`) to use objective functions with parameters specified as vectors (for compatibility with `optim` etc.)

3 Newer stuff

To do:

- use `predict`, `simulate` etc. to demonstrate different parametric bootstrap approaches to confidence and prediction intervals
  - use `predict` to get means and standard deviations, use delta method?
  - use `vcov`, assuming quadratic profiles, with `predict(...,newparams=...)`
  - prediction intervals assuming no parameter uncertainty with `simulate`
  - both together ...

4 Technical details

4.1 Profiling and confidence intervals

This section describes the algorithm for constructing profiles and confidence intervals, which is not otherwise documented anywhere except in the code. * indicates changes from the version in `stats4:::mle`

4.1.1 Estimating standard error

In order to construct the profile for a particular parameter, one needs an initial estimate of the scale over which to vary that parameter. The estimated standard error of the parameter based on the estimated curvature of the likelihood surface at the MLE is a good guess.

- if `std.err` is missing, extract the standard error from the summary coefficient table (ultimately computed from `sqrt(diag(inverse Hessian))` of the fit)
- * a user-set value of `std.err` overrides this behavior unless the value is specified as `NA` (in which case the estimate from the previous step is used)
- * if the standard error value is still `NA` (i.e. the user did not specify it and the value estimated from the Hessian is missing or `NA`) use `sqrt(1/diag(hessian))`. This represents a (fairly feeble) attempt to come up with a plausible number when the Hessian is not positive definite but still has positive diagonal entries
- if all else fails, stop and * print an error message that encourages the user to specify the values with `std.err`

There may be further tricks that would help guess the appropriate scale: for example, one could guess on the basis of a comparison between the parameter values and negative log-likelihoods at the starting and ending points of the fits.

On the other hand, (a) this would take some effort and still be subject to failure for sufficiently pathological fits and (b) there is some value to forcing the user to take explicit, manual steps to remedy such problems, as they may be signs of poorly defined or buggy log-likelihood functions.

4.1.2 Profiling

Profiling is done on the basis of a constructed function that minimizes the negative log-likelihood for a fixed value of the focal parameter and returns the signed square-root of the deviance difference from the minimum (denoted by z). At the MLE $z = 0$ by definition; it should never be < 0 unless something has gone wrong with the original fit. The LRT significance cutoffs for z are equal to the usual two-tailed normal distribution cutoffs (e.g. $\pm \approx 1.96$ for 95% confidence regions).

In each direction (decreasing and increasing from the MLE for the focal parameter):

- fix the focal parameter
- adjust control parameters etc. accordingly (e.g. remove the entry for the focal parameter so that the remaining control parameters match the non-fixed parameters)
- controls on the profiling (which can be set manually, but for which there is not much guidance in the documentation):
  - `zmax` Maximum z to aim for. (Default: `sqrt(qchisq(1-alpha/2, p))`) The default maximum α (type I error) is 0.01. *I don't understand this criterion. It seems to expand the size of the univariate profile to match a cutoff for the multivariate confidence region of the model. The χ^2 cutoff for deviance to get the $(1-\alpha)$ multivariate confidence region (i.e., on all p of the parameters) would be* `qchisq(1-alpha,p)` *— representing a one-tailed test on the deviance. Taking the square root makes sense, since we are working with the square root of the deviance, but I don't understand (1) why we are expanding the region to allow for the multivariate confidence region (since we are computing univariate profiles) [you could at least argue that this is conservative, making the region a little bigger than it needs to be]; (2) why we are using $1-\alpha/2$ rather than $1-\alpha$.* For comparison, `MASS::profile.glm` (written by Bates and Venables in 1996, ported to R by BDR in 1998) uses `zmax=sqrt(qchisq(1-alpha,1))` *(this makes more sense to me ...)* . On the other hand, the profiling code in `lme4a` (the `profile` method for `merMod`, in `profile.R`) uses `qchisq(1-alphamax, nptot)` ...
  - `del` Step size (scaled by standard error) (Default: `zmax`/5.) Presumably (?) copied from `MASS::profile.glm`, which says (in `?profile.glm`): "[d]efault value chosen to allow profiling at about 10 parameter values."

  - `maxsteps` Maximum number of profiling steps to try in each direction. (Default: 100)

- While `step<maxsteps` and `abs(z) < zmax`, set the value of the focal parameter to its MLE + `sgn*step*del*std.err` where `sgn` represents the direction, `step` is the current (integer) step, and `del` and `std.err` are the step size scaling factor and standard error estimate discussed above (i.e. take steps of size (`del*std.err`) in the appropriate direction); evaluate z
- Stop the profiling:
  - if z doesn't change from the previous step (`stop_flat`)
  - * stop if z is less than `tol.newmin` (default: 0.001) units *better* than the MLE fit, i.e. $z < -$`tol.newmin` (if $-$`tol.newmin` $< z < 0$, set z to zero) (`newpars_found`)
  - if z is `NA` (`stop_na`) *(for greater robustness, should we try to keep going anyway?)*
  - if z is beyond `zmax` (i.e., we have reached our goal: `stop_cutoff`)
  - if `step==maxsteps`
  - if the focal parameter has hit its upper/lower bound (`stop_bound`)
- if we have hit the maximum number of steps but not reached the cutoff (`stop_maxstep` but not `stop_cutoff`), "try a bit harder": go *almost* one more `del*std.err` unit out (in intervals of 0.2, 0.4, 0.6, 0.8, 0.9) (*also seems reasonable but don't know where it comes from*)
- * if we violated the boundary but did not reach the cutoff (`!stop_cutoff && stop_bound`), evaluate z at the boundary
- if we got to the cutoff in < 5 steps, try smaller steps: start at `step=0.5` and proceed to `mxstep-0.5` in unit increments (rather than the original scale which went from 0 to `mxstep`). (*Again, it seems reasonable, but I don't know what the original justification was ...*)

4.1.3 Confidence intervals

We are looking for the values where z (signed square root deviance difference) is equal to the usual two-tailed normal distribution cutoffs for a specified α level, e.g. $z = \pm 1.96$ for 95% confidence intervals (this is equivalent to a one-tailed test on the deviance difference with the cutoff value for χ^2_1).

- If necessary, construct the profile
- * If the profile of the signed square root is non-monotonic, warn the user and revert to linear approximation on the profiled points to find the cutoffs:

- Otherwise, build an interpolation spline of z (signed square root deviance difference) based on profiled points (the default is $n = 3 \times L$ where L is the length of the original vector). Then use linear approximation on the y (z) and x (focal parameter value) of the spline to find the cutoffs (*Why construct a spline and then interpolate linearly? Why not use* `backSpline` *as in the profile plotting code?*)

It is also possible to use `method="quad"` (this is obsolete, should be replaced by documentation pointing to `confint.default`) and `method="uniroot"` (undocumented for now; rather than approximating from the profile, try to use `uniroot` with calls to the negative log-likelihood function to identify the confidence intervals as precisely and robustly as possible (should be more accurate, but slower — must also be computed for each specified α level separately).

4.1.4 Profile plotting

Plot the signed (or unsigned) square root deviance difference, and $(1-\alpha)$ confidence regions/critical values designated by `conf` (default: $\{0.99, 0.95, 0.9, 0.8, 0.5\}$).

- * If the (signed) profile is non-monotonic, simply plot computed points with `type="l"` (i.e., with the default linear interpolation)
- Construct the interpolation spline (using `splines:::interpSpline` rather than `spline` as in the confidence interval method (*why this difference?*)
- attempt to construct the inverse of the interpolation spline (using `backSpline`)
- * if this fails warn the user (assume this was due to non-monotonicity) and try to use `uniroot` and `predict` to find cutoff values
- otherwise, use the inverse spline to find cutoff values

Why is there machinery in the plotting code to find confidence intervals? Shouldn't this call `confint`*, for consistency/fewer points of failure?*

Bugs, wishes, to do

- **WISH**: further methods and arguments: `subset`, `predict`, `resid`: `sim`?
- **WISH**: extend ICtab to allow DIC as well?
- minor **WISH**: better methods for extracting `nobs` information when possible (e.g. with formula interface)
- **WISH**: better documentation, especially for S4 methods
- **WISH**: variable-length chunks in argument list
- **WISH**: limited automatic differentiation (add capability for common distributions)

References

Crowder, M. J. (1978). Beta-binomial Anova for proportions. *Applied Statistics 27*, 34–37.

Vonesh, J. R. and B. M. Bolker (2005). Compensatory larval responses shift tradeoffs associated with predator-induced hatching plasticity. *Ecology 86*(6), 1580–1591.