Basics of optimization/likelihood minimization

7 Oct 2024

```
library(RTMB)
library(bbmle)
```

References

Bolker (2008), Bolker et al. (2013)

Minimization in statistics

- fitting models to data
- 'best fit'
- *objective function* or *loss function*; differs by application/problem type
- least squares, minimum absolute deviation, cross-entropy …
- many of these are special cases of **negative log-likelihood**

Maximum likelihood

- nice properties (efficient, consistent, asymptotically unbiased)
- unifying principle
- usually minimize negative log-likelihood instead
- "when it can do the job, it's rarely the best tool for the job but it's rarely much worse than the best" (S. Ellner)

Minimization

- closed form solution of *score equations* (direct or via linear algebra)
- iteratively reweighted least squares
- gradient descent
- more complex iterative solutions
  - Nelder-Mead
  - quasi-Newton methods (BFGS, L-BFGS)
- automatic differentiation

Minimization in R

- `optim()` (`nlminb`)
- Nelder-Mead
- various quasi-Newton methods (Press et al. 2007)

```
X <- model.matrix(~wool*tension, data=warpbreaks)
y <- warpbreaks$breaks
nll <- function(beta) {
    mu <- exp(X %*% beta)
    -sum(dpois(y, mu, log=TRUE))
}
par0 <- rep(0, ncol(X))
opt1 <- optim(par0, nll, control = list(maxit = 1000))
par2 <- coef(glm(breaks~wool*tension, data=warpbreaks, family=poisson))
all.equal(opt1$par, unname(par2), tolerance = 1e-3)
```

```
[1] TRUE
```

with `mle2`

- wrapper for `optim`, variant of `stats4::mle()`
- `data` argument
- better accessor methods (`coef()`, `vcov()`, `broom::tidy()`, `profile()`, `confint()`, `predict()`, …)

```
names(par0) <- parnames(nll) <- paste0("beta", 1:ncol(X))
confint(mle2(nll, par0))
```

```
             2.5 %     97.5 %
beta1  3.69723021  3.8930351
beta2 -0.61480841 -0.3002960
beta3 -0.78557346 -0.4545355
beta4 -0.76139971 -0.4328077
beta5  0.39936387  0.8783493
beta6 -0.06653957  0.4428584
```

formula notation

```r
mle2(breaks~dpois(exp(eta)),
     parameters = list(eta ~ wool*tension),
     start = list(eta = 0),
     data = warpbreaks)
```

```
Call:
mle2(minuslogl = breaks ~ dpois(exp(eta)), start = list(eta = 0),
    data = warpbreaks, parameters = list(eta ~ wool * tension))

Coefficients:
   eta.(Intercept)           eta.woolB        eta.tensionM        eta.tensionH
         3.7966810          -0.4565402          -0.6186293          -0.5957747
eta.woolB:tensionM eta.woolB:tensionH
         0.6381522          0.1882803

Log-likelihood: -228.48
```

`mle2` notes

- uses `BFGS` (with finite difference gradients!) by default; may want Nelder-Mead
- probably want to provide a link function for dispersion parameters, e.g. fit negative binomial with `size = exp(logsize)`

Template Model Builder and autodiff

- everything is better with gradients
- **finite difference** gradients (R default) are terrible (expensive and inaccurate)

- FD vs symbolic vs automatic/algorithm
- magic/chain rule; *cheap gradient* principle (Kristensen et al. 2016)

deriv() in base R

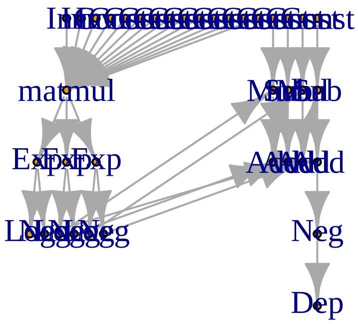
Useful, illustrative, but limited

```
deriv(expression(cos(x*sin(x^2))), "x")
```

```
expression({
    .expr1 <- x^2
    .expr2 <- sin(.expr1)
    .expr3 <- x * .expr2
    .value <- cos(.expr3)
    .grad <- array(0, c(length(.value), 1L), list(NULL, c("x")))
    .grad[, "x"] <- -(sin(.expr3) * (.expr2 + x * (cos(.expr1) *
        (2 * x))))
    attr(.value, "gradient") <- .grad
    .value
})
```

RTMB package

```
data("prussian", package = "pscl")
prussmin <- prussian[1:3,]
X2 <- model.matrix(~ factor(year), data = prussmin)
par2 <- list(beta = rep(0, ncol(X2)))
f2 <- function(pars) {
    getAll(pars)  ## this is like with() or attach()
    mu <- exp(X2 %*% beta)
    -sum(dpois(prussmin$y, lambda = mu, log = TRUE))
}
```




```r
ff2 <- RTMB::MakeADFun(f2, par2)
ff2$fn()
```

```
[1] 328.2462
```

```r
ff2$gr()
```

```
outer mgc:  84

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]   84    9    7    5    4   -4    8    0    3     5     9     3    -1     8
     [,15] [,16] [,17] [,18] [,19] [,20]
[1,]     3    -3     2    -1     6    10
```

Can use `f2$fn()`, `f2$gr()` to get function and gradient vector efficiently.

References

Bolker, Benjamin M. 2008. *Ecological Models and Data in R*. Princeton, NJ: Princeton University Press.

Bolker, Benjamin M., Beth Gardner, Mark Maunder, Casper W. Berg, Mollie Brooks, Liza Comita, Elizabeth Crone, et al. 2013. "Strategies for Fitting Nonlinear Ecological Models in R, AD Model Builder, and BUGS." *Methods in Ecology and Evolution* 4 (6): 501–12. https://doi.org/10.1111/2041-210X.12044.

Kristensen, Kasper, Anders Nielsen, Casper W. Berg, Hans Skaug, and Bradley M. Bell. 2016. "TMB : Automatic Differentiation and Laplace Approximation." *Journal of Statistical Software* 70 (5). https://doi.org/10.18637/jss.v070.i05.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 2007. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd ed. Cambridge University Press.