MATH4P06 Midterm Report

Incorporating Symbolic and Automatic Differentiation in a General-Purpose Likelihood Optimization Function

Queenie Zeng - Ben Bolker

12/15/2020

Objective

The primary objective of this project is to re-implement a completed R package `qzmle` for maximum likelihood estimation and analysis that follows the principles and includes all functionality in the current R package `bbmle`, while providing more extensive methods and reduced computing time. It is aimed to be more robust and inclusive than the `mle()` function that comes with R by default.

Current Work

When given a set of sample observations $y_1, y_2, ..., y_n$ that follows some parametric distribution such that $y \sim (f_\theta|\theta)$ with n number of parameters $(\theta_1, \theta_2, ..., \theta_n)$, the maximum likelihood estimate (MLE) $\hat{\theta}$ of any given distribution parameter is the value that makes the observed data the most probable. This can be achieved by maximizing the likelihood function of the given distribution with respect to the give parameter $(L_n(\theta|y))$, or more often by the convention of minimizing the negative log-likelihood function such that $\hat{\theta} = \text{argmin}_\theta(-\log L_n(\theta|y))$. We implemented this process in the wrapper function `mle()` (the function name is not finalized) around the general-purpose optimization function `optim()` in base R. Here is an example of fitting a simple model and applying the `summary` method for `qzmle` objects to show the parameters, standard errors (approximated using the inverse of hessian), and the corresponding p-values based on the z-scores.

```
d <- data.frame(x=0:10,y=c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8))
fit <- qzmle::mle(y~dnorm(mean=ymean, sd=2),start=list(ymean=mean(d$y), ysd=2),data=d)
summary(fit)
```

```
## Maximum likelihood estimation
##
## Call:
## qzmle::mle(form = y ~ dnorm(mean = ymean, sd = 2), start = list(ymean = mean(d$y),
##     ysd = 2), data = d)
##
## Coefficients:
##       Estimate Std. Error z value     Pr(z)
## ymean 11.54545    0.60302  19.146 < 2.2e-16 ***
## ysd    2.00000    0.00000     Inf < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## -2 log L: 157.1477
```

It is worth mentioning that this implementation automatically computes the gradients to subsequently derive the hessian with convenience, instead of using the hessian option in `optim`. The reason for this is to reduce the cost of run-time and offer the flexibility for the users to omit deriving the hessian. The procedure for generating the gradients leverages the symbolic differentiation method facilitated by `Deriv` in the `Deriv` package, and also considering situations when the parameters of the distribution depends on other covariates such that $\theta_i = g(w_1, w_2, ..., w_m)$. Applying the chain-rule, the gradient of the negative log-likelihood function with respect to each covariate is $\frac{\partial L}{\partial w_j} = \sum_{k=1}^{k} \frac{\partial L}{\partial \theta_i}(y_k) \cdot \frac{\partial \theta_i}{\partial w_j}(y_k)$.

To-do for next term

So far we have grasped a good understanding of architecture of `bbmle` and have built the basic mle function. The nearest goal is to developed all functionality and methods in `bbmle`, such as constructing likelihood profiles and confidence intervals for multiple parameters in a model.

The next priority is to set up the Template Model Builder (TMB) as the back end. The TMB is an integration of compiled C++ scripts to perform faster and more robust calculations. For instance, we want to incorporate automatic differentiation to evaluate more complex objective functions and derivatives without sacrificing computing time. Furthermore, TMB also allows for greater flexibility to perform other comprehensive analysis, such as fitting mixed-effects models and importance sampling.