# MATH4P06 Midterm Report

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

Queenie Zeng (supervisor: Dr. 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. The source code can be viewed here: https://github.com/queezzz/qzmle.

## 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 $n$ 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 nonlinear 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 the Hessian matrix), and the corresponding $p$-values based on the $z$-scores of the parameters.

```
library(qzmle)
dd <- data.frame(y = rpois(100, lambda=3))
fit <- mle(y~dpois(lambda=ymean),start=list(ymean=mean(dd$y)),data=dd)
summary(fit)
```

```
## Maximum likelihood estimation
##
## Call:
## mle(form = y ~ dpois(lambda = ymean), start = list(ymean = mean(dd$y)),
##     data = dd)
##
## Coefficients:
##       Estimate Std. Error z value     Pr(z)
## ymean  2.62000    0.16187  16.186 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## -2 log L: 362.9782
```

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 the implementation of R packages and have built the basic mle function. The next goal is to develop 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 uses 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.