St. Pierre bank assessment model

Jonathan Babyn

March 15, 2019

Contents

Appendices

A Numerically stable logged normal cumulative distribution function derivatives in TMB

`TMB` uses Automatic Differentation (AD) to generate the first and second derivatives which are used to help with both performing MLE for parameters and integrating random effects out of the likelihood with the Laplace Approximation (LA). `TMB` accomplishes this with a mixture of forward and reverse AD accumulation modes. Further details on AD can be seen in Fournier et al[1]. `TMB` automatically tries to determine the derivatives of the user's template. However, it can not automatically determine the most numerically stable form of the derivative which can often be necessary when working with extremely small values that can commonly occur when working with probabilities. `TMB` also includes the ability to manually define the forward and reverse modes for a given function to overcome this limitation using atomic functions. By default `TMB` includes a built in atomic version of the normal CDF, `pnorm` based on the version in `R`'s `C` language math library. This built-in version does not support the ability to return a logged value of the probability and shares the same limitation of returning 0 or 1 for negative or positive values of standard normal Z scores when $|Z| >\sim 38$ due to floating point limitations. This clearly means that for any $Z < -38$, `TMB` will return a `NaN` in the gradient when trying to perform `log(pnorm(Z))` due to log(0) being negative infinity. In practice however this is actually much worse since the derivative of `log(pnorm(Z))` is

$$\frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}\left(\frac{\text{erf}\left(\frac{z}{\sqrt{2}}\right)}{2}+\frac{1}{2}\right)} \tag{1}$$

and easily results in the division of two extremely small numbers. With floating point limitations this quickly becomes undefined if not handled correctly.

I created an atomic function to specifically deal with the problem of numerical stability when trying to do `log(pnorm(Z))` and similar calculations in the objective function of SPAM such as those done when using a censored likelihood for the detection limit for fitting the observation equations for the survey indices. This requires specifying more numerically stable versions for both the forward and reverse AD modes. For the forward mode this is just the `double C++` function giving the log of the normal CDF and all that requires is calling the version of `pnorm` in `R`'s `C` math library with the `give.log` flag turned on. For the reverse mode just using Equation 1 is not numerically stable due to the division. It can easily be seen that Equation 1 is equivalent to

$$\exp\left(\log\left(\frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}}\right)-\log\left(\frac{\text{erf}\left(\frac{z}{\sqrt{2}}\right)}{2}+\frac{1}{2}\right)\right). \tag{2}$$

Working in log space is much less likely to result in under or overflow of floating point values when performing division of small numbers. Using numerically stable versions of the logged normal PDF & CDFs and the form in Equation 2 results in a numerically stable reverse mode derivative for `log(pnorm(Z))`. The atomic function was wrapped in the function `pnorm4` and made available for others to use in the `C++` header file `pnorm4.hpp`. Using `pnorm4(Z)` in place of `log(pnorm(Z))` allows for running one-sided censor likelihood bounds without the need for using two runs of optimization.

It's accuracy was checked by comparing the results of the gradient and hessian generated by `TMB` for `pnorm4` against the numerical first and second derivatives of `pnorm4` done by the `R` software

package `numDeriv` along with the brute force first and second derivatives of the normal log(CDF) calculated by the open-source computer algebra system `Maxima` with 5000 digits of floating point precision.

A.1 Extending stability to censored bounds

Censored likelihood bounds pose a similar problem to the above. When using the built-in functions to add normally distributed censored log-likelihood bounds this can be done like `log(pnorm(ZU))-log(pnorm(ZU))` where `ZU` & `ZL` are Z scores of the upper and lower bounds respectively. Just replacing the calls to `pnorm` with the more stable `pnorm4` and using the brute force approach or `logspace_sub` to perform the subtraction is not stable either. This is because for $Z > 38$ `pnorm` will return 1, or 0 for the logged version which will then result in trying to take the log of 0, again leading to `NaN` in the gradient.

The normal CDF of the upper and lower bounds can be thought of as $\Phi(ZU) = 1 - \epsilon_u$ and $\Phi(ZL) = 1 - \epsilon_l$ where the ϵs are some value greater than zero and the censored log-likelihood bound is

$$\log(\Phi(ZU) - \Phi(ZL)) = \log(1 - \epsilon_u - (1 - \epsilon_l)) = \log(\epsilon_l - \epsilon_u)) \tag{3}$$

References

[1] David A. Fournier et al. "AD Model Builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models". *Optimization Methods and Software* 27.2 (Apr. 2012), pp. 233–249. ISSN: 1029-4937. DOI: 10.1080/10556788.2011.597854. URL: http://dx.doi.org/10.1080/10556788.2011.597854.