ML estimation - TMB -

FPA2020 Lecture 7

Toshihide Kitakado

June 10, 2020

Contents


```
knitr::opts_chunk$set(echo=T, warning=F)
library(knitr)
library(rmarkdown)
library(ggplot2)
```

In this lecture, we shall learn a quick-and-dirty syntax for using "TMB", which is a fast computation tool. When we use TMB for maximization of likelihood, "TMB" also provides us with useful ouctomes and convenient by-products for your likelihoo inference.

1 Installation and introduction

1.1 Step 1: Installation of "Rtools"

The "Rtools" includes various utilities used in computation in R. This also includes "C++" libraries, which are also useful for several other statistical computation. A code is quite simple like below if you want to install from the console. You can also download from the website of R.

```
pkgbuild:: has_build_tools(debug=T)
```

If you have successfully installed it into your PC, you will see the message "TRUE"!

1.2 Step 2. Installation of "TMB"

Then you will install "TMB" into your R.

```
#install.packages("TMB")
require(TMB)
```

1.3 Procedure to use "TMB"

The rough procedure is like this:

- write your code in "cpp" file (for example, the file is saved as "xxx.cpp")
- compile "xxx.cpp" and call a dynamic link using TMB
- conduct optimization via dynamic link created by TMB
- extract results

OK, let us start with some simple examples. We shall come back TMB in later lectures for hierarchical models (with random and mixed effects) and time series models.

2 How to use "TMB"? Example 1 - IID normal distribution -

I would like to begin with a simple maximum likelihood estimation for the normal distribution.

2.1 Set-up example data

The assumption is like this.

$$Y_1, Y_2, \ldots, Y_n \sim (iid) N(\mu, \sigma^2)$$

We shall generate a set of simulation data.

```
set.seed(2020)
mu.true <- 5
sigma.true <- 2
Nsample <- 20
y <- rnorm(Nsample, mu.true, sigma.true)
data <- list(y=y)
```

2.2 Model (1): Estimation of μ only given σ

First, a known σ case.

2.2.1 Writing cpp file

```
sink(file="normal_1.cpp")
cat("
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() ()
{
 DATA_VECTOR(y);
 PARAMETER(mu);
 Type logSigma=0.0;
 Type sigma = exp(logSigma);

 Type f;
 f = (-1.0)*sum(dnorm(y,mu,sigma,true));
 return f;
}
", fill=TRUE)
sink()
```

2.2.2 Compiling "cpp" file and call "Dynamic link"

```
compile("normal_1.cpp")
```

```
[1] 0
```

```
dyn.load(dynlib("normal_1"))
```

2.2.3 Minimizing an objective function defined (optimization)

- "MakeADFun": Constructing objective functions with derivatives based on a compiled C++ template
- "nlminb": Minimization

```
parameters <- list(mu=0)
model <- MakeADFun(data, parameters, DLL="normal_1")
fit <- nlminb(model$par, model$fn, model$gr)
```

```
outer mgc:  95.90528
outer mgc:  15.90528
outer mgc:  2.220446e-15
```

2.2.4 Extracting estimates and their SEs and showing results

```
best <- model$env$last.par.best
rep <- sdreport(model)
```

```
outer mgc:  2.220446e-15
outer mgc:  0.02
outer mgc:  0.02
```

```
print(best)
```

```
      mu
4.795264
```

```
print(rep)
```

```
sdreport(.) result
   Estimate Std. Error
mu 4.795264  0.2236068
Maximum gradient component: 2.220446e-15
```

```
mean(y) #For confirmation because we know the analytical result
```

```
[1] 4.795264
```

2.3 Model (2): Estimation of μ and σ

2.3.1 Writing cpp file

```
sink(file="normal_2.cpp")
cat("
#include <TMB.hpp>
template<class Type>
Type objective_function<Type>::operator() ()
{
 DATA_VECTOR(y);
 PARAMETER(mu);
 PARAMETER(logSigma);
 Type sigma = exp(logSigma);
 ADREPORT(sigma);

 Type f;
 f = (-1.0)*sum(dnorm(y,mu,sigma,true));
 return f;
}
", fill=TRUE)
sink()
```

2.3.2 Compiling "cpp" file and ....

```
compile("normal_2.cpp")
```

```
[1] 0
```

```
dyn.load(dynlib("normal_2"))

parameters <- list(mu=0, logSigma=0)
model <- MakeADFun(data, parameters, DLL="normal_2")
fit <- nlminb(model$par, model$fn, model$gr)
```

```
outer mgc:  594.4889
outer mgc:  61.14155
outer mgc:  47.06547
```

```
outer mgc:  27.54712
outer mgc:  12.16396
outer mgc:  8.290393
outer mgc:  5.22705
outer mgc:  2.815733
outer mgc:  0.2983192
outer mgc:  0.01912785
outer mgc:  0.000141723
outer mgc:  2.524653e-07
```

```
rep <- sdreport(model)
```

```
outer mgc:  2.524653e-07
outer mgc:  0.002587612
outer mgc:  0.002587107
outer mgc:  0.03995992
outer mgc:  0.04004013
outer mgc:  2.780268
```

```
#print(rep)
summary(rep)
```

```
         Estimate Std. Error
mu       4.795264  0.6216867
logSigma 1.022547  0.1581138
sigma    2.780268  0.4395988
```

```
mean(y)
```

```
[1] 4.795264
```

```
sd(y)/sqrt(length(y))
```

```
[1] 0.6378371
```

3 How to use "TMB"? Example 2 - Normal regression -

3.1 Simple regression model

Now we handle with a simple regression model as

$$Y_i = a + bx_i + \varepsilon_i, \quad \varepsilon_i \sim N(0, \sigma^2).$$

3.2 Data generation (Y=0+1*x+N(0,1))

Assume $a = 0$, $b = 1$ and $\sigma = 1$ for generating 10 observations. Here, we generate covariates as $x_i = i$ for simplicity.

```
set.seed(2020)
data <- list(y = rnorm(10) + 1:10, x=1:10)
data.frame(data)
```

```
          y  x
1  1.376972  1
2  2.301548  2
3  1.901977  3
```

```
4   2.869594  4
5   2.203466  5
6   6.720573  6
7   7.939121  7
8   7.770622  8
9  10.759131  9
10 10.117367 10
```

```
#plot(data$x, data$y)
parameters <- list(a=0, b=0, logSigma=0)
data.frame(parameters)
```

```
  a b logSigma
1 0 0        0
```

3.3 Point estimation

3.3.1 Writing cpp file

```
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  DATA_VECTOR(x);
  DATA_VECTOR(y);
  int n = y.size();

  PARAMETER(a);
  PARAMETER(b);
  PARAMETER(logSigma);
  Type sigma = exp(logSigma);
  ADREPORT(sigma);

  vector<Type> yfit(n);

  Type f = 0.0;

  yfit = a + b*x;
  f = (-1.0)*sum(dnorm(y, yfit, exp(logSigma), true));

  return f;
}
```

3.3.2 Compiling "cpp" file, call "Dynamic link" and optimization

```
compile("linreg.cpp")
```

```
[1] 0
```

```
dyn.load(dynlib("linreg"))
obj = MakeADFun(data, parameters, DLL="linreg")
opt = nlminb(obj$par, obj$fn, obj$gr)
```

```
outer mgc:  400.5989
outer mgc:  28.24634
outer mgc:  12.85502
outer mgc:  8.413373
outer mgc:  10.98643
outer mgc:  1.404259
outer mgc:  1.310238
outer mgc:  0.8597652
outer mgc:  0.06675507
outer mgc:  0.001319113
outer mgc:  0.0001287523
outer mgc:  6.704565e-06
```

```
summary(sdreport(obj))
```

```
outer mgc:  6.704565e-06
outer mgc:  0.04064794
outer mgc:  0.04063453
outer mgc:  0.2844953
outer mgc:  0.2844819
outer mgc:  0.01997977
outer mgc:  0.02002026
outer mgc:  1.163317
```

```
           Estimate Std. Error
a        -0.8351474  0.7946965
b         1.1329427  0.1280769
logSigma  0.1512750  0.2236067
sigma     1.1633166  0.2601254
```

3.3.3 Extracting estimates and their SEs and showing results

```
rep <- sdreport(obj)
```

```
outer mgc:  6.704565e-06
outer mgc:  0.04064794
outer mgc:  0.04063453
outer mgc:  0.2844953
outer mgc:  0.2844819
outer mgc:  0.01997977
outer mgc:  0.02002026
outer mgc:  1.163317
```

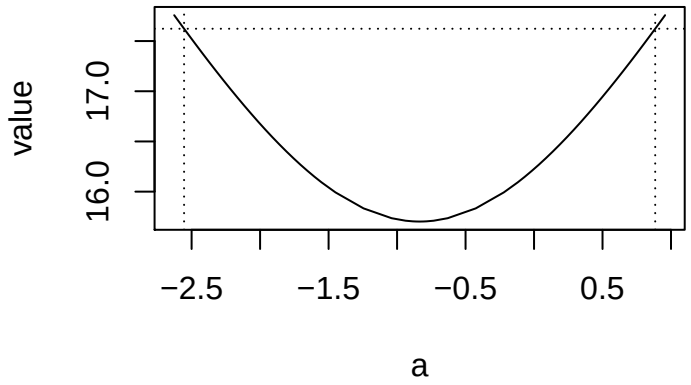
```
print(rep)
```

```
sdreport(.) result
           Estimate Std. Error
a        -0.8351474  0.7946965
b         1.1329427  0.1280769
logSigma  0.1512750  0.2236067
Maximum gradient component: 6.704565e-06
```

3.4 Profile likelihood functions and confidence intervals

3.4.1 Drawing profile likelihood functions for a

```
## Profile wrt. a:
prof <- tmbprofile(obj,"a", trace=F)
plot(prof)
```

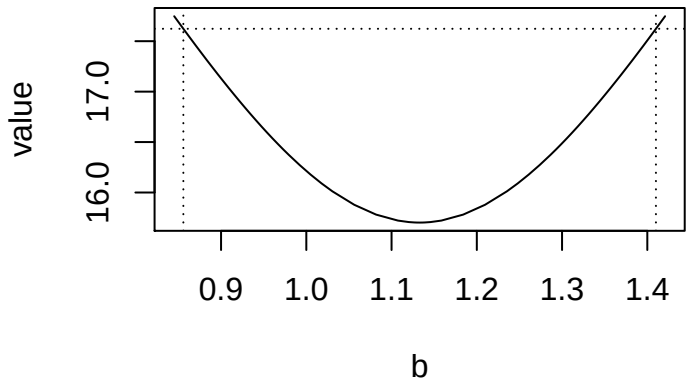



```
confint(prof)
```

```
      lower    upper
a -2.554933 0.884637
```

3.4.2 Drawing profile likelihood functions for b

```
## Profile wrt. b:
prof <- tmbprofile(obj,"b", trace=F)
plot(prof)
```

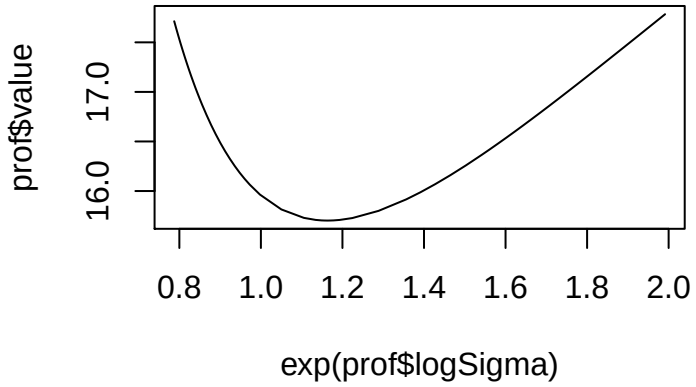


```
confint(prof)
```

```
      lower    upper
b 0.8557693 1.410116
```

3.4.3 Drawing profile likelihood functions for σ

```
## Profile wrt. Sigma:
prof <- tmbprofile(obj, "logSigma", trace=F)
plot(exp(prof$logSigma),prof$value,type="l")
```




```
CI <- exp(confint(prof))
row.names(CI) = "Sigma"
CI
```

```
          lower    upper
Sigma 0.7935187 1.942628
```

4 Bayesian analysis using TMB outcomes (you can skip, perhaps in later lectures)

TMB can be linked with Bayesian estimation with MCMC (not Gibbs sampling, but no U-turn MCMC like in stan's Hamiltonian MCMC). I will explain this in later lectures after instrucion of basics of Bayesian estimation. Having said that, here is for your reference.

4.1 adnuts (tmbstan) estimation

```
#install.packages('adnuts')
#install.packages('shinystan')
library(adnuts)
library(rstan)
rstan_options(auto_write = TRUE)
library(shinystan)
library(coda)
library(bayesplot)
```

4.1.1 Posterior simulation using NUTS HMC

```
init <- function() list(a=0, b=0, logSigma=0)
#init <- as.list(opt$par)
fit <- sample_tmb(
        obj=obj,
        init=init,
        chains=3,
        iter=15000,
        warmup=5000,
        thin=10
)
```

4.1.2 HMC posteriors

```
post <- extract_samples(fit, as.list=T)
post_vec <- extract_samples(fit)
Median <- apply(post_vec, 2, median)
SD <- apply(post_vec, 2, sd)
data.frame(Median, SD)
```

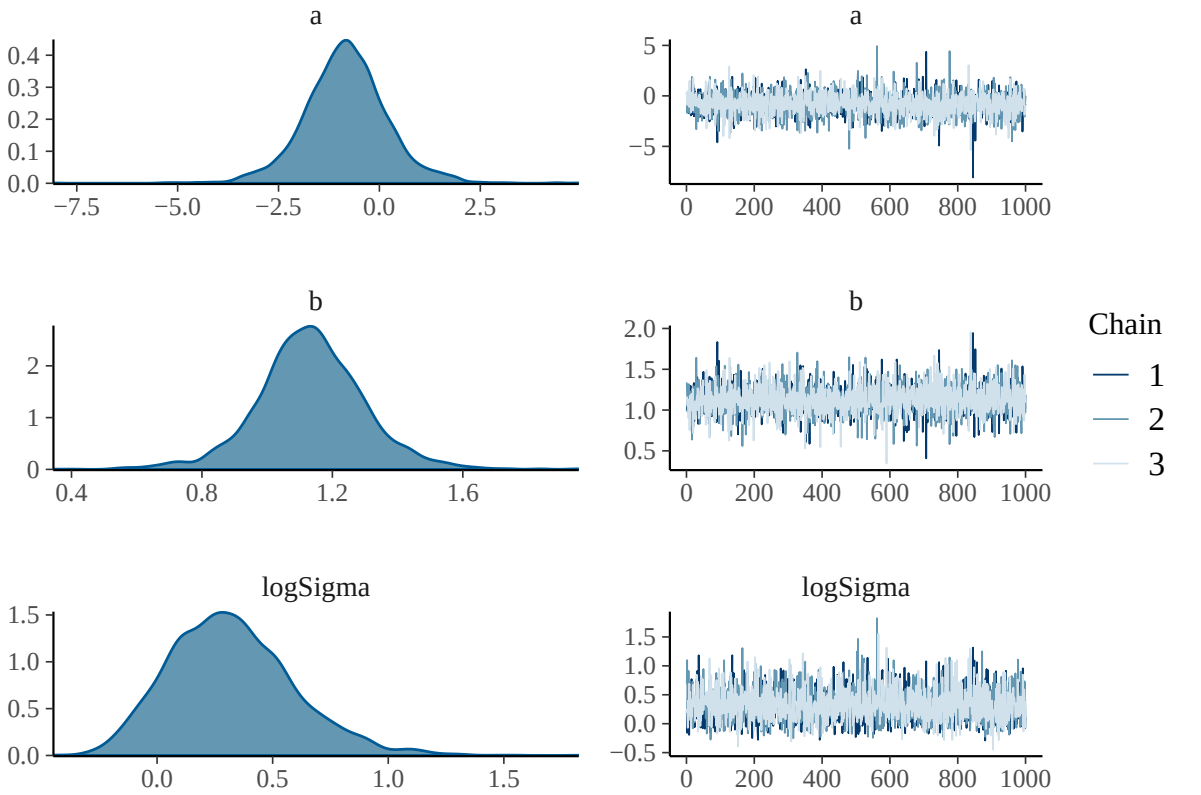
```
             Median        SD
a        -0.8371677 1.0325414
b         1.1316473 0.1649020
logSigma  0.3002783 0.2688152
```

4.1.3 Shiny

```
launch_shinytmb(fit)
```

‘ ### Diagnostics

```
mcmc_combo(post)
```




```
mon <- rstan::monitor(fit$samples, print=FALSE)
Rhat <- mon[,"Rhat"]
ess <- mon[, 'n_eff']
data.frame(Rhat, ess)
```

```
             Rhat  ess
a       1.0028327 2258
b       1.0027079 2187
logSigma 1.0011229 2099
lp__    0.9997739 2196
```

5 HW (submission due is June 16, 2020)

Assume an iid distribution from one of thme below and estimate the parameter by a generating simulation data set (this time, single parameter model):

- Poisson distribution
- binomial distribution
- exponential distribution