Asymptotic Estimate of Partition Function

Oystein Sorensen

8/27/2018

This is my working document for confirming that the `C++` implementation of the asymptotic normalizing constant is correct.

First, we take the original `R` code. Here is Zlim.

```
Zlim<-function(alpha.grid,K,niter, distance){

  nalpha <- length(alpha.grid)
  tmp <- numeric(nalpha)

  if(distance == "footrule"){
    dist <- function(x,y){return(-abs(x-y))}
  }else if(distance == "spearman"){
    dist <- function(x,y){return(-(x-y)^2)}
  }

  # IPFP procedure
  alpha <- 0
  #A <- matrix(1,ncol=min(K,niter),nrow=min(K,niter))
  A <- matrix(1,ncol=K,nrow=K)
  for(iter in 1:niter){
    A <- A/rowSums(A)
    A <- A/rep(1, nrow(A)) %*% t(colSums(A))
  }

  Z0lim <- -2*log(K)-sum(A*log(A))

  for(i in 1:nalpha){
    alpha <- alpha.grid[i]
    A <- exp(alpha*outer((1:K)/K, (1:K)/K, dist))
    for(iter in 1:niter){
      A <- A/rowSums(A)
      A <- A/rep(1, nrow(A)) %*% t(colSums(A))
    }
    tmp[i] <- alpha*sum(outer((1:K)/K, (1:K)/K, dist)*A)-2*log(K)-sum(A*log(A))
  }

  return(list("alphaGrid"=alpha.grid,"Z0lim"=Z0lim,"Zlim"=tmp))

}
```

Here is the script. Note that I set the grid kind of small to speed up the code.

```
####create a grid of alpha, for which we would later interpolate
alpha.grid<-c(seq(10,50,10))
niter<-105    ###you can run it for as long as possible, but 200 is more than enough

#Choose n - the number of items you want to compute for
n<-100
```

```
## number of factors in the function
###K = 0.4*n seems like a good choice
### but I often compute a list of normalising constants with different Ks
### and then choose one use the simulated dataset
K.try<-seq(200,206,2)


t1 <- Sys.time()
Z0lim<-c()
Z_lim<-Tranf_Zlim_2<-matrix(ncol=length(K.try),nrow=length(alpha.grid))
Z0<-0
for (x in 1:n) {
  Z0<-Z0+log(x)
}

i=1
for (K in K.try){
  #print(K)
  #print(paste("number ", which(K.try == K), "out of", length(K.try)))
  tmp<-Zlim(alpha.grid,K,niter,"footrule")
  Z0lim[i]<-tmp$Z0lim
  Z_lim[,i]<-tmp$Zlim
  Tranf_Zlim_2[,i]<-(Z_lim[,i]-Z0lim[i])/K*n+Z0
  i=i+1
}

t2 <- Sys.time()
print(t2 - t1)
```

```
## Time difference of 1.569708 secs
```

Next, I use the corresponding function from `BayesMallows`. The function is internal, so we need to use `BayesMallows:::asymptotic_partition_function` to find it. I want to compare the values in `Tranf_Zlim_2`.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(purrr)
library(tidyr)

t1 <- Sys.time()
Tranf_Zlim_Cpp <- K.try %>%
  map(~ BayesMallows:::asymptotic_partition_function(alpha_grid = alpha.grid,
                                                     metric = "footrule", K = .x,
                                                     n_iterations = niter, n_items = n))
```

```r
t2 <- Sys.time()
print(t2 - t1)
```

```
## Time difference of 0.5479369 secs
```

There is not much time difference. C++ is only twice as fast.

Now, we want to confirm that they are equal, and yes they are.

```r
Tranf_Zlim_2_NEW <- Tranf_Zlim_Cpp %>%
  bind_cols() %>%
  as.matrix()

dimnames(Tranf_Zlim_2_NEW) <- NULL
```

```r
all.equal(Tranf_Zlim_2, Tranf_Zlim_2_NEW)
```

```
## [1] TRUE
```