R code demo for fast computing algorithm

Shiqiang Jin and Gyuhyeong Goh

2020-06-22

Contents

Chapter 1

R code demo for fast computing algorithm

This is a supplymentary material about an R domo code for computing marginal likelihood distribution using fasting computing strategy and for-loop method

1.1 Review of Appendix B.1 Calculation

For any $i \notin \hat{\gamma}$, $|\hat{\gamma} \cup \{i\}| = k+1$, hence $s(Y|\hat{\gamma} \cup \{i\})$ in Eq.(3) can be expressed as

$$s(Y|\hat{\gamma} \cup \{i\}) = \zeta^{-\frac{m(k+1)}{2}} |X^{\mathrm{T}}_{\hat{\gamma}\cup\{i\}} X_{\hat{\gamma}\cup\{i\}} + \zeta^{-1} I_{k+1}|^{-\frac{m}{2}} |Y^{\mathrm{T}} H_{\hat{\gamma}\cup\{i\}} Y + \Psi|^{-\frac{n+\nu}{2}}. \quad (1.1)$$

Using technique fast computing algorithm, we have

$$\begin{aligned} s_+(\hat{\gamma}) &= c^+_{\hat{\gamma}} \times \left(\zeta^{-1} 1_p + diag(X^{\mathrm{T}} H_{\hat{\gamma}} X)\right)^{-m/2} \cdot & (1.2) \\ &\quad \left[1_p - \frac{diag(X^{\mathrm{T}} H_{\hat{\gamma}} Y (Y^{\mathrm{T}} H_{\hat{\gamma}} Y + \Psi)^{-1} Y^{\mathrm{T}} H_{\hat{\gamma}} X)}{\zeta^{-1} 1_p + diag(X^{\mathrm{T}} H_{\hat{\gamma}} X)}\right]^{-\frac{n+\nu}{2}} & (1.3) \end{aligned}$$

where $a^x = (a_1^x, \ldots, a_p^x)$, $a \cdot b = (a_1 b_1, \ldots, a_p b_p)$, $a/b = (a_1/b_1, \ldots, a_p/b_p)$ for generic vectors a and b, and $c^+_{\hat{\gamma}} = \zeta^{-\frac{m(k+1)}{2}} |X^{\mathrm{T}}_{\hat{\gamma}} X_{\hat{\gamma}} + \zeta^{-1} I_k|^{-\frac{m}{2}} \left|Y^{\mathrm{T}} H_{\hat{\gamma}} Y + \Psi\right|^{-\frac{n+\nu}{2}}$ is a constant with respect to $i \notin \hat{\gamma}$.

Hence,

$$\log(s_+(\hat{\gamma})) = \log(c^+_{\hat{\gamma}}) 1_p - \frac{m}{2} \log(d) - \frac{n+\nu}{2} \log(1 - \frac{u}{d}), \quad (1.4)$$

where $d = \zeta^{-1}1_p + diag(X^{\mathrm{T}}H_{\hat{\gamma}}X)$ and $u = diag(X^{\mathrm{T}}H_{\hat{\gamma}}Y(Y^{\mathrm{T}}H_{\hat{\gamma}}Y + \Psi)^{-1}Y^{\mathrm{T}}H_{\hat{\gamma}}X)$.

In the following simulation example, I will evaluate $s(Y|\mathrm{nbd}_+(\hat{\gamma}))$ by (1.1) in the "for-loop" method and by (1.4) in a single calculation.

1.2 Simulation example

In this example, we

- specify the model setting as $n = 100, p = 1000, m = 5, \zeta = \log(n), \Psi = 0.5I_m, \nu = 0.5$.

```
n <- 100
p <- 1000
m <- 5
zeta <- log(n)
Psi <- diag(0.5, m)  # Psi
v <- 0.5  # nu
```

- and generate data $Y = XC + E$ with $E \sim \mathcal{N}(0, \Omega)$; The true model is $\gamma^* = (1, 2, 3, 4, 7, 8, 9, 10)$ and the current model $\hat{\gamma} = (1, 2, 3, 4, 7, 8, 9)$ with model size $|\hat{\gamma}| = 7$. $\Omega = 0.2^{|i-j|}$ and X is generated from $\mathcal{N}(0, \Sigma)$ with $\Sigma = 0.2^{|i-j|}$.

```
# Generate data
library(mvtnorm)
set.seed(1314)
true.model <- c(1:4, 7:10)  # true model
r <- c(1:4, 7:9)  # current model
k <- length(r)  # current model size
rho_e <- 0.2
Omega <- rho_e^(abs(matrix(1:m, m, m) - t(matrix(1:m, m, m))))
rho_x <- 0.2
Sig_x <- rho_x^(abs(matrix(1:p, p, p) - t(matrix(1:p, p, p))))
seq.p <- c(1:p)
len.true.model <- length(true.model)
# generate random coefficient matrix C
c0 <- sample(seq(-1, 1, 0.2), size = len.true.model * m, replace = TRUE)
C <- matrix(0, p, m)
C[true.model, ] <- matrix(c0, len.true.model, m)
X <- rmvnorm(n, rep(0, p), Sig_x, method = "chol")
E <- rmvnorm(n, mean = rep(0, m), sigma = Omega)
Y <- as.numeric(X %*% C) + E
```

To better understand R code and corresponding notations, we list a cross-reference table for some of them as follows:

I_n	I_k1	log.s.plus1 or log.s.plus2	rUi	X.rUi	H.rUi
I_n	I_{k+1}	$\log(s(Y\|\text{nbd}_+(\hat{\gamma}))\hat{\gamma} \cup i$		$X_{\hat{\gamma}\cup i}$	$H_{\hat{\gamma}\cup i}$
log.s.Y.rUi	I_k	X.r	X_r	H.r	colSums(H.r%*%X_r*X_r)
$\log(s(Y\|\hat{\gamma}\cup i))$	I_k	$X_{\hat{\gamma}}$	$X_{-\hat{\gamma}}$	$H_{\hat{\gamma}}$	$diag(X_{-\hat{\gamma}}^{\mathrm{T}} H_{\hat{\gamma}} X_{-\hat{\gamma}})$
YHX_r					
$Y^{\mathrm{T}} H_{\hat{\gamma}} X_{-\hat{\gamma}}$					

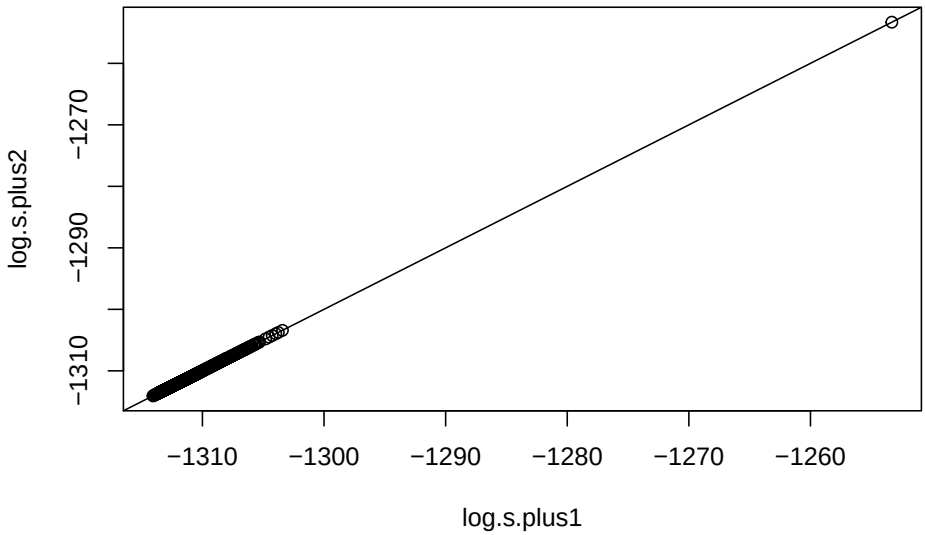
```r
# For loop method
I_n <- diag(1, n)  # n-dimension identity matrix
I_k1 <- diag(1, k + 1)
p_r <- setdiff(seq(1, p), r)  # p-k vector
log.s.plus1 <- rep(NA, length(p_r))
j <- 1
for (i in p_r) {
    rUi <- sort(c(r, i))  # add one index from p_r
    X.rUi <- X[, rUi]  # model in addition neighbor
    XtX <- crossprod(X.rUi) + 1/zeta * I_k1
    H.rUi <- I_n - X.rUi %*% solve(XtX) %*% t(X.rUi)
    # logarithm of Eq (1.1)
    log.s.Y.rUi <- -m * (k + 1)/2 * log(zeta) - m/2 * log(det(XtX)) - (n + v)/2 * log(det(t(Y) %*
    log.s.plus1[j] <- log.s.Y.rUi
    j <- j + 1
}

# Proposed Method
I_k <- diag(1, k)  # k-dimension identity matrix
X.r <- X[, r]
X_r <- X[, p_r]  # n by p-k m sub-matrix of X
H.r <- I_n - X.r %*% solve(crossprod(X.r) + 1/zeta * I_k) %*% t(X.r)  # n by n matrix
d <- 1/zeta + colSums(H.r %*% X_r * X_r)  # p-k dimension vector
YHX_r <- t(Y) %*% H.r %*% X_r  # p-k by m matrix
YHY_1 <- solve(t(Y) %*% H.r %*% Y + Psi)  # m by m matrix
u <- colSums(YHY_1 %*% YHX_r * YHX_r)  # p-k dimension vector

# logarithm of Eq (1.3)
log.s.plus1.approx <- -m/2 * log(d) - (n + v)/2 * log(1 - u/d)
# log(c)
log.c <- -0.5 * m * (k + 1) * log(zeta) - 0.5 * m * log(det(crossprod(X.r) +
    1/zeta * I_k)) - (n + v)/2 * log(det(t(Y) %*% H.r %*% Y + Psi))
log.s.plus2 <- log.c + log.s.plus1.approx  # logarithm of Eq (1.2)
```

I compute mean absolute percentage error MAPE $= \frac{1}{n}\Sigma_{t=1}^{n}|\frac{A_t - F_t}{A_t}|$ to measure the accuracy of the fast computing algorithm.

```
# Mean absolute percentage error
MAPE <- mean(abs(log.s.plus1 - log.s.plus2)/abs(log.s.plus1))
print(paste("MAPE =", MAPE))
```

```
## [1] "MAPE = 8.12298413315687e-17"
```

```
plot(log.s.plus1, log.s.plus2)
abline(a = 0, b = 1)
```



From the plot and MAPE, $\log(s(Y|\text{nbd}_+(\hat{\gamma})))$ computed by (1.1) and (1.4) are the same. But the time costs are different.

1.3 Time cost comparison for proposed method and for loop method

Note that when doing the model selection, as $\log(c_{\hat{\gamma}}^{+})$ is a constant with respect to $i \notin \hat{\gamma}$, in R code I only compute `log.s.plus2.approx`. I use R package `microbenchmark` to do the simulation and the default replication is 100 times.

```
library(microbenchmark)
timecost <- microbenchmark("for_loop" = {
  log.s.plus1 <- rep(NA, length(p_r))
  j <- 1
  for (i in p_r) {
    rUi <- sort(c(r, i))  # add one index from p_r
```

```
    X.rUi <- X[, rUi]  # model in addition neighbor
    XtX <- crossprod(X.rUi) + 1/zeta * I_k1
    H.rUi <- I_n - X.rUi %*% solve(XtX) %*% t(X.rUi)
    # logarithm of Eq (1.1)
    log.s.Y.rUi <- -m * (k + 1)/2 * log(zeta) -
      m/2 * log(det(XtX)) - (n + v)/2 * log(det(t(Y) %*% H.rUi %*% Y + Psi))
    log.s.plus1[j] <- log.s.Y.rUi
    j <- j + 1
  }
},
"Proposed" = {
  X.r <- X[, r]
  I_k <- diag(1, k)  # k-dimension identity matrix
  X_r <- X[, p_r]  # n by p-k m sub-matrix of X
  H.r <- I_n - X.r %*% solve(crossprod(X.r) + 1/zeta * I_k) %*% t(X.r)  # n by n matrix
  d <- 1/zeta + colSums(H.r %*% X_r * X_r)  # p-k dimension vector
  YHX <- t(Y) %*% H.r %*% X_r  # p-k by m matrix
  YHY_1 <- solve(t(Y) %*% H.r %*% Y + Psi)  # m by m matrix
  u <- colSums(YHY_1 %*% YHX * YHX)  # p-k dimension vector
  # logarithm of Eq (3)
  log.s.plus2.approx <- -m/2 * log(d) - (n + v)/2 * log(1 - u/d)
}
)
timecost
```

```
## Unit: milliseconds
##      expr        min         lq      mean     median         uq       max neval
##  for_loop 131.419268 138.245982 155.65647 144.917884 156.258279 317.48266   100
##  Proposed   1.247057   1.320723   1.74544   1.378074   1.528487  10.30982   100
```

Looking at the median of time cost, the proposed method is about 100 times faster than the for-loop method.