STAT-6494 Advanced Statistical Computing with R

Homework 2

Wenjie Wang

10 March 2016

Exercise: Distribution of the Max of n Independent Normal Variables

In a clinical trial, there are k treatment groups with different dosages and one control group (think about Dr. Qiqi Deng's talk). For simplicity, suppose that the outcome of interest is normally distributed with mean μ_k and σ^2. When a statistician compares the best group out of the k groups with the control group, this is no longer a standard two-sample comparison due to the multiplicity issues from the k groups. Consider a random sample $X_1, X_2, \ldots, X_n$ from $N(\mu, \sigma^2)$. Let $Y_n = \max(X_1, \ldots, X_n)$ be the sample maximum. Although the exact distribution of Y_n can be obtained analytically (Nadarajah and Kotz 2008; Hill 2011), it is better illustrated via kernel density estimation with random draws from the distribution. Given n, μ, and σ^2, generate a large number N observations from the distribution of Y_n. Let $\mu = 0$, $\sigma = 1$, and $N = 10,000$. Use **ggplot2** to draw the histogram of the observations with $n \in \{2, 3, 4, 5, 10, 100\}$ in plot with 2×3 panels. Overlay the kernel density estimation and the density of $N(\mu, \sigma^2)$ in each panel.

Function `rMaxNorm`

The simple function `rMaxNorm` shown below is to generate random numbers from the distribution of the minimum of n independent normal distribution.

```
rMaxNorm <- function (n = 2, N = 1e4, mu = 0, sd = 1) {
    rMat <- matrix(rnorm(N * n), nrow = N, ncol = n)
    tmpList <- lapply(seq_len(n), function (j) rMat[, j])
    do.call("pmax", tmpList)
}
```

Overlaid Histogram and Density Plot by Using ggplot2.

By using the function `rMaxNorm` defined in last section, we are able to get $N = 10,000$ simulated observations with $n \in \{2, 3, 4, 5, 10, 100\}$, respectively. Then, the overlaid histogram and density plots can be plotted by **ggplot2** as follows. The output plot is shown in Figure 1.

```
library(ggplot2)
N = 1e4
x <- seq(from = - 4, to = 4, length.out = N)
dens <- dnorm(x)
ggDat0 <- data.frame(dens = dens, x = x)
set.seed(1216)
n <- c(2, 3, 4, 5, 10, 100)
ggList <- lapply(n, rMaxNorm, N = N)
ggDat <- data.frame(y = do.call("c", ggList),
                    n = factor(rep(n, each = N)),
```

```
                             levels = n, labels = paste("n =", n)))
(ggp <- ggplot(ggDat0, aes(x = x, y = dens)) + geom_line() +
     geom_histogram(data = ggDat, mapping = aes(x = y, y = ..density.., fill = n),
                    inherit.aes = FALSE, alpha = 0.25, bins = 25, ) +
     geom_density(data = ggDat, mapping = aes(x = y, color = n),
                  inherit.aes = FALSE) +
     ylab("Density") + xlab(expression(Y[n])) +
     facet_wrap("n", nrow = 2, ncol = 3) +
     theme_bw() + theme(legend.position = "none"))
```

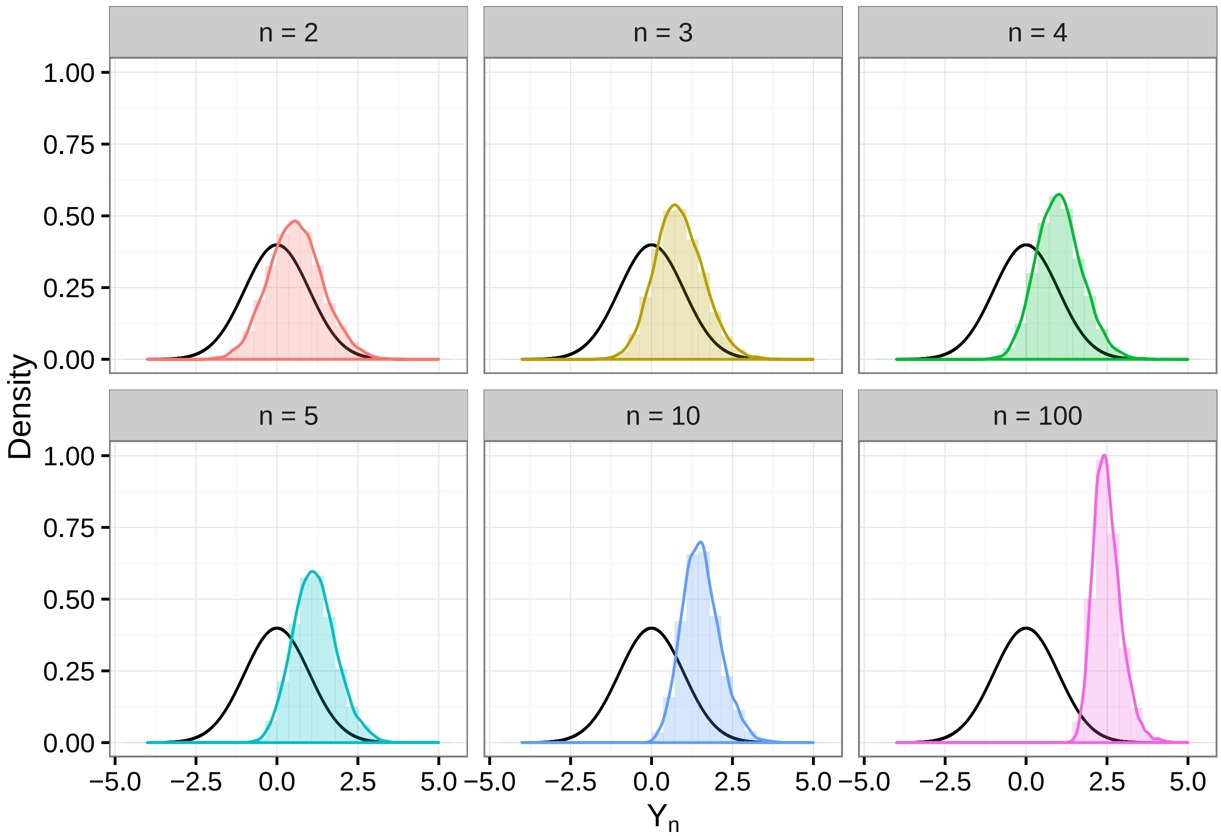

Figure 1: Overlaid histogram and density plots.

Exercise: Linear Regression with RcppArmadillo

Consider the linear regression model with response vector Y and design matrix X. Write a function in **C++** using the facilities available in package **RcppArmadillo** to do linear regression fit. The function takes two inputs `Y` and `X`, and outputs three components: `coefficients` for estimated regression coefficients; `stderr` for standard error of the estimates; and `df.residuals` for the degrees of freedom of the residuals. Compare the performance with function `lm.fit` in **R** on a simulated large dataset.

Function `armaLm`

The function `armaLm` is slightly revised from an example given by Eddelbuettel (2013) and implemented with the help of R package **inline** fitting linear regression model. It takes design matrix `X`, response vector

Y, and returns **coefficients** for estimated coefficients, **stderr** for standard error of the estimates, and **df.residuals** for the degrees of freedom of the residuals.

```
src <- '
Rcpp::NumericMatrix Xr(Xs);
Rcpp::NumericVector Yr(Ys);
int n = Xr.nrow(), k = Xr.ncol();
arma::mat X(Xr.begin(), n, k, false);
arma::colvec Y(Yr.begin(), Yr.size(), false);
int df = n - k;
// fit model Y ~ X, extract residuals
arma::colvec coef = arma::solve(X, Y);
arma::colvec res = Y - X * coef;
double s2 = std::inner_product(res.begin(),
                               res.end(), res.begin(), 0.0) / df;
// std.errors of coefficients
arma::colvec sderr =
  arma::sqrt(s2 * arma::diagvec(arma::pinv(arma::trans(X) * X)));
return Rcpp::List::create(Rcpp::Named("coefficients") = coef,
Rcpp::Named("stderr") = sderr,
Rcpp::Named("df.residuals") = df);
'

armaLm <- inline::cxxfunction(signature(Xs = "numeric", Ys = "numeric"),
                              body = src, plugin = "RcppArmadillo")
```

Test on a Large Simulated Dataset

We first generate a simulated data from linear regression model. Then we fit the model by `armaLm` and get the results of interest. The response `Y` is set as a vector with length 100,000. The design matrix is a 100,000 by 10 matrix.

```
## generate simulation data
betaVec <- c(0.5, 1, 2, 0.8, 1.2, 2.3, 0.6, 1.5, 0.9) # true beta
p <- length(betaVec)
sigMat <- matrix(0.1, ncol = p - 1, nrow = p - 1) +
    diag(0.9, nrow = p - 1, ncol = p - 1)
set.seed(1216)
X <- cbind(1, MASS::mvrnorm(n = 1e5, mu = rep(0, p - 1), Sigma = sigMat))
resVec <- rnorm(1e5)
Y <- X %*% betaVec + resVec

## fit linear model by `armaLm`
armaLm(X, Y)
```

```
## $coefficients
##            [,1]
##  [1,] 0.5037954
##  [2,] 1.0000462
##  [3,] 2.0087218
##  [4,] 0.7993188
##  [5,] 1.2001423
##  [6,] 2.2990620
```

```
## [7,] 0.6054398
## [8,] 1.5047199
## [9,] 0.8999855
##
## $stderr
##              [,1]
## [1,] 0.003169161
## [2,] 0.003242726
## [3,] 0.003242648
## [4,] 0.003228458
## [5,] 0.003243005
## [6,] 0.003238661
## [7,] 0.003243101
## [8,] 0.003248527
## [9,] 0.003252709
##
## $df.residuals
## [1] 99991
```

Performance Comparison with `lm.fit`

We compare the computing performance of `armaLm` with `lm.fit` from package **stats** with the help of package **microbenchmark** over the simulated `Y` and `X` as follows:

```r
library(microbenchmark)
microbenchmark(lm.fit(X, Y), armaLm(X, Y), times = 100)
```

```
## Unit: milliseconds
##          expr      min       lq     mean   median       uq      max neval
##  lm.fit(X, Y) 22.43148 23.48385 33.56906 25.62882 27.31743 66.94347   100
##  armaLm(X, Y) 20.01297 20.16291 20.50800 20.35720 20.61012 22.67137   100
##  cld
##    b
##   a
```

From the comparison, we may find that `armaLm` runs faster than `lm.fit`.

Reference

Eddelbuettel, Dirk. 2013. *Seamless R and C++ Integration with Rcpp*. Springer.

Hill, Joshua E. 2011. "The Minimum of N Independent Normal Distributions."

Nadarajah, Saralees, and Samuel Kotz. 2008. "Exact Distribution of the Max/Min of Two Gaussian Random Variables." *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 16 (2). IEEE: 210–12.