Module 9: Simulations and Parallel Computing

Jianhui Gao

July 23, 2025

Outline

In this module, we will review

- Simulation Study
- Rationale for Simulations
- Parallel Computing in R

Simulation study

- Simulation: A numerical techniques for conducting experiments on the computer
- Monte Carlo simulation: Computer experiment involving random sampling from probability distributions

Why simulation?

To establish/validate the properties of statistical methods

- Exact analytical derivations of properties are **rarely** possible
- Large sample approximations to properties are **often possible**, but need to evaluate their relevance to (finite) sample sizes likely to be encountered in practice

Why simulation?

To establish/validate the properties of statistical methods

- Exact analytical derivations of properties are **rarely** possible
- Large sample approximations to properties are **often possible**, but need to evaluate their relevance to (finite) sample sizes likely to be encountered in practice

Moreover, analytical results may require **assumptions** (e.g., normality)

- But what happens when these assumptions are violated?
- Analytical results, even large sample ones, may not be possible

Considerations for simulation

- Is an estimator **biased** in finite samples? Is it still **consistent** under departures from assumptions? What is its **sampling variance**?
- How does it **compare** to competing estimators on the basis of bias, precision, etc.?

Considerations for simulation

- Is an estimator **biased** in finite samples? Is it still **consistent** under departures from assumptions? What is its **sampling variance**?
- How does it **compare** to competing estimators on the basis of bias, precision, etc.?
- Does a procedure for constructing a **confidence interval** for a parameter achieve the advertised **nominal level of coverage**?
- Does a **hypothesis testing** procedure attain the advertised **level** or **size**?
- If it does, what **power** is possible against different alternatives to the null hypothesis? Do different test procedures deliver different power?

Monte Carlo simulation

- Generate S independent data sets under the conditions of interest
- Compute the numerical value of the estimator/test statistic T (data) for each data set $\Rightarrow T_1, \ldots, T_S$
- If S is large enough, **summary statistics** across $T_1, \ldots, T_S$ should be good **approximations** to the true sampling properties of the estimator/test statistic under the conditions of interest

Simulations for properties of estimators

Example: Compare 3 estimators for the **mean** μ of a distribution based on i.i.d. draws $Y_1, \dots, Y_n$

- Sample mean $T^{(1)}$
- Sample 20% trimmed mean $T^{(2)}$
- Sample median $T^{(3)}$

Simulations for properties of estimators (cont'd)

Simulation procedure: For a particular choice of μ, n, and true underlying distribution

- Generate independent draws $Y_1, \ldots, Y_n$ from the distribution
- Compute $T^{(1)}$, $T^{(2)}$, $T^{(3)}$
- Repeat S times $T_1^{(1)}, \ldots, T_S^{(1)}; \quad T_1^{(2)}, \ldots, T_S^{(2)}; \quad T_1^{(3)}, \ldots, T_S^{(3)}$
- Compute for $k = 1, 2, 3$

$$\widehat{\mu} = S^{-1} \sum_{s=1}^{S} T_s^{(k)} = \bar{T}^{(k)}, \ \widehat{\text{bias}} = \bar{T}^{(k)} - \mu$$

$$\widehat{\sigma} = \sqrt{(S-1)^{-1} \sum_{s=1}^{S} \left(T_s^{(k)} - \bar{T}^{(k)} \right)^2}$$

$$\widehat{\mathrm{MSE}} = S^{-1} \sum_{s=1}^{S} \left(T_s^{(k)} - \mu \right)^2 \approx \widehat{\mathrm{SD}}^2 + \widehat{\mathrm{bias}}^2$$

Simulations for properties of estimators (cont'd)

Another important property we care about is the **relative efficiency** (RE).

- If the estimators are unbiased,

$$RE = \frac{\text{var}\left(T^{(1)}\right)}{\text{var}\left(T^{(2)}\right)}$$

- If the estimators are biased,

$$RE = \frac{\text{MSE}\left(T^{(1)}\right)}{\text{MSE}\left(T^{(2)}\right)}$$

In either case $RE < 1$ means estimator 1 is preferred (estimator 2 is inefficient relative to estimator 1 in this sense)

Set up parameters

```r
set.seed(123)
# number of simulations
S <- 1e5
# sample size
n <- 1000
# mu and sigma
mu <- 1
sigma <- sqrt(5 / 3)
# function
trimmean <- function(Y) mean(Y, 0.2)
```

Run Simulation (for loop)

```r
start_time <- Sys.time()
t1 <- t2 <- t3 <- c()
for (s in 1:S) {
  # generate data
  dat <- rnorm(n, mu, sigma)
  # calculate T1
  t1 <- c(t1, mean(dat))
  # calculate T2
  t2 <- c(t2, trimmean(dat))
  # calculate T3
  t3 <- c(t3, median(dat))
}
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 39.86446 secs
```

Bias?

```r
mean(t1 - 1)
```

```
## [1] 0.0002025304
```

```r
mean(t2 - 1)
```

```
## [1] 0.0001590339
```

```r
mean(t3 - 1)
```

```
## [1] 6.529179e-05
```

- All estimators are shown minimal bias, why?

Sample Variance?

```r
var(t1)
```

```
## [1] 0.001661072
```

```r
var(t2)
```

```
## [1] 0.001902612
```

```r
var(t3)
```

```
## [1] 0.00261475
```

Relative Efficiency?

```r
cat("T1 vs T2", (mean(t2 - 1)^2 + var(t2)) /
  (mean(t1 - 1)^2 + var(t1)), "\n")
```

```
## T1 vs T2 1.145398
```

```r
cat("T1 vs T3", (mean(t3 - 1)^2 + var(t3)) /
  (mean(t1 - 1)^2 + var(t1)), "\n")
```

```
## T1 vs T3 1.574098
```

```r
cat("T2 vs T3", (mean(t3 - 1)^2 + var(t3)) /
  (mean(t2 - 1)^2 + var(t2)), "\n")
```

```
## T2 vs T3 1.374279
```

Run Simulation (lapply)

```
start_time <- Sys.time()
t <- lapply(1:S, function(s) {
  # generate data
  dat <- rnorm(n, mu, sigma)
  # calculate T1
  t1 <- mean(dat)
  # calculate T2
  t2 <- trimmean(dat)
  # calculate T3
  t3 <- median(dat)
  c(t1, t2, t3)
})
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 6.725847 secs
```

```
# convert t to a dataframe with column t1, t2, t3
t_final <- do.call(rbind, t)
```

Run Simulation (Vectorize)

```r
generate.normal <- function(S, n, mu, sigma) {
  dat <- matrix(rnorm(n * S, mu, sigma), ncol = n, byrow = T)
  out <- list(dat = dat)
  return(out)
}
```

```r
start_time <- Sys.time()
out <- generate.normal(S, n, mu, sigma)
out_mean <- apply(out$dat, 1, mean)
out_trimmean <- apply(out$dat, 1, trimmean)
out_median <- apply(out$dat, 1, median)
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 8.394579 secs
```

Introduction to Embarrassing Parallelism

- `for loop` execute each task sequentially
- Modern computers are built in with multiple cores that allows you do the above jobs in parallel
- Rise of high performance computing (HPC) cluster
- The improvement is not linear!

Parallel in local computer (foreach)

- most intuitive parallel algorithm, just like `for loop`
- need to set-up the local cluster

```
library(doParallel)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```
detectCores()
```

```
## [1] 14
```

```
cl <- makeCluster(8)
registerDoParallel(cl)
start_time <- Sys.time()
t <- foreach(s = 1:S, .combine = "rbind") %dopar% {
  # generate data
  dat <- rnorm(n, mu, sigma)
  # calculate T1
  t1 <- mean(dat)
  # calculate T2
  t2 <- trimmean(dat)
  # calculate T3
  t3 <- median(dat)

  c(t1, t2, t3)
}
end_time <- Sys.time()
end_time - start_time
```

Parallel in local computer (mclapply)

- The `mclapply()` function essentially parallelizes calls to `lapply()`

```r
library(parallel)

start_time <- Sys.time()
t <- mclapply(1:S, function(s) {
  # generate data
  dat <- rnorm(n, mu, sigma)
  # calculate T1
  t1 <- mean(dat)
  # calculate T2
  t2 <- trimmean(dat)
  # calculate T3
  t3 <- median(dat)
  c(t1, t2, t3)
}, mc.cores = 4)
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 2.246978 secs
```

```r
# convert t to a dataframe with column t1, t2, t3
t_final <- do.call(rbind, t)
```

Parallel in local computer (parLapply)

```r
cl <- makeCluster(8)
registerDoParallel(cl)
start_time <- Sys.time()
t <- parLapply(cl = cl, X = 1:S, function(s) {
  # generate data
  dat <- rnorm(n, mu, sigma)
  # calculate T1
  t1 <- mean(dat)
  # calculate T2
  t2 <- trimmean(dat)
  # calculate T3
  t3 <- median(dat)
  c(t1, t2, t3)
})
```

```
## Error in checkForRemoteErrors(val): 8 nodes produced errors; first error: object 'n' not found
```

```r
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 0.007097006 secs
```

```r
# convert t to a data frame with column t1, t2, t3
t_final <- do.call(rbind, t)
```

parLapply continued

- need to export the environment

```r
clusterExport(cl, varlist = c("n", "mu", "sigma", "trimmean"))
start_time <- Sys.time()
t <- parLapply(cl = cl, X = 1:S, function(s) {
  # generate data
  dat <- rnorm(n, mu, sigma)
  # calculate T1
  t1 <- mean(dat)
  # calculate T2
  t2 <- trimmean(dat)
  # calculate T3
  t3 <- median(dat)
  c(t1, t2, t3)
})
end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 0.9950922 secs
```

```r
# convert t to a data frame with column t1, t2, t3
t_final <- do.call(rbind, t)
```

Error Handling (foreach)

```r
t <- foreach(
  i = 1:1e4, .combine = "rbind"
) %dopar% {
  # generate data
  A <- matrix(data = rbinom(4, 1, 0.5), nrow = 2)
  solve(A)
}
```

```
## Error in {: task 1 failed - "Lapack routine dgesv: system i
```

- The error may only occur occasionally
- You want to ignore the error and finish your job

Error Handling (foreach)

```r
t <- foreach(
  i = 1:1e4,
  .errorhandling = "pass"
) %dopar% {
  # generate data
  A <- matrix(data = rbinom(4, 1, 0.5), nrow = 2)
  solve(A)
}

head(t, 2)
```

```
## [[1]]
##      [,1] [,2]
## [1,]    0    1
## [2,]    1    0
##
## [[2]]
## <simpleError in solve.default(A): Lapack routine dgesv: sys
```

Error Handling (foreach)

```r
t <- foreach(
  i = 1:1e4,
  .errorhandling = "remove"
) %dopar% {
  # generate data
  A <- matrix(data = rbinom(4, 1, 0.5), nrow = 2)
  solve(A)
}
head(t, 2)
```

```
## [[1]]
##      [,1] [,2]
## [1,]    0    1
## [2,]    1   -1
##
## [[2]]
##      [,1] [,2]
## [1,]   -1    1
## [2,]    1    0
```

Error Handling (tryCatch)

- `tryCatch` enables you to handle **errors** and **warnings**

```r
t <- parLapply(cl, X = 1:1e4, fun = function(x) {
  # generate data
  tryCatch(
    {
      A <- matrix(data = rbinom(4, 1, 0.5), nrow = 2)
      solve(A)
    },
    error = function(e) {
      # code that will be executed in the event of an error
      return(NA)
    }
  )
})

head(t, 2)
```

```
## [[1]]
## [1] NA
##
## [[2]]
##      [,1] [,2]
## [1,]    1    0
## [2,]   -1    1
```

Error Handling (tryCatch)

- Often times, warning messages are not outputed in the parallel process

```r
sigma <- -1
t <- parLapply(cl = cl, X = 1:S, function(s) {
  # generate data
  dat <- rnorm(n, mu, sigma)
  # calculate T1
  t1 <- mean(dat)
  # calculate T2
  t2 <- trimmean(dat)
  # calculate T3
  t3 <- median(dat)
  c(t1, t2, t3)
})
head(t, 2)
```

```
## [[1]]
## [1] 0.9195931 0.8962642 0.8693469
##
## [[2]]
## [1] 0.9633860 0.9890214 1.0077568
```

Error Handling (tryCatch)

```r
sigma <- -1
t <- parLapply(cl = cl, X = 1:S, function(s) {
  tryCatch(
    {
      # generate data
      dat <- rnorm(n, mu, sigma)
      # calculate T1
      t1 <- mean(dat)
      # calculate T2
      t2 <- trimmean(dat)
      # calculate T3
      t3 <- median(dat)
      c(t1, t2, t3)
    },
    warning = function(w) {
      # code that will be executed in the event of a warning
      return(w)
    }
  )
})
head(t, 2)
```

```
## [[1]]
## [1] 0.9910793 0.9903015 0.9856814
##
## [[2]]
## [1] 1.025379 1.039457 1.003340
```

Parallel in HPC

- Using Niagara cluster (Compute Canada) as an example, it contains 2024 nodes, each with 40 cores, for a total of 80,640 cores.
- Say if you want to request 20 cores, there are two ways to request it
  - 1 node and all 20 cores on the node
  - different nodes

One node Prallel

```r
cl <- makeCluster(20)
registerDoParallel(cl)
clusterExport(cl, varlist = c("n", "mu", "sigma", "trimmean"))
t <- parLapply(cl = cl, X = 1:S, function(s) {
  # generate data
  dat <- rnorm(n, mu, sigma)
  # calculate T1
  t1 <- mean(dat)
  # calculate T2
  t2 <- trimmean(dat)
  # calculate T3
  t3 <- median(dat)
  c(t1, t2, t3)
})

# convert t to a data frame with column t1, t2, t3
t_final <- do.call(rbind, t)

# save the results
saveRDS(t_final, "t_final.rds")
```

save the R script as `example.R`

One node Prallel

Use `module spider r` to check the requirement for loading R

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
#SBATCH --time=0-01:30           # time (DD-HH:MM)
module load gcc/9.3.0 r/4.0.2

Rscript example.R
```

Save it as `submit.sh`

Submit the job by `sbatch submit.sh`

Multiple Nodes

- things are much more complicated
- sometimes cannot be avoided, say if you want to request 800 cores
- need to use `OpenMPI`

Multiple Nodes

```r
cl <- makeCluster(800, type = "MPI")
registerDoParallel(cl)
t <- parLapply(cl = cl, X = 1:S, function(s) {
  # generate data
  dat <- rnorm(n, mu, sigma)
  # calculate T1
  t1 <- mean(dat)
  # calculate T2
  t2 <- trimmean(dat)
  # calculate T3
  t3 <- median(dat)
  c(t1, t2, t3)
})

# convert t to a data frame with column t1, t2, t3
t_final <- do.call(rbind, t)

# save the results
saveRDS(t_final, "t_final.rds")
```

save the R script as `example.R`

Multiple Nodes

```
#!/bin/bash
#SBATCH --nodes=20
#SBATCH --ntasks-per-node=40
#SBATCH --time=0-01:30           # time (DD-HH:MM)
module load gcc/9.3.0 openmpi/4.0.3 r/4.0.2

R_PROFILE=${HOME}/R/x86_64-pc-linux-gnu-library/4.0/snow/
RMPISNOWprofile;
export R_PROFILE
mpirun -np 800 -bind-to core:overload-allowed R CMD BATCH
--no-save example.R
```

Save it as `submit.sh`

Submit the job by `sbatch submit.sh`

Passing argument

- sometimes you may want to run for a set of arguments
- e.g. n = c(100, 200, 300, 400)

```
args <- commandArgs(TRUE)
n <- args[1]

cl <- makeCluster(800, type = "MPI")
registerDoParallel(cl)
clusterExport(cl, varlist = c("n", "mu", "sigma", "trimmean"))

t <- parLapply(cl = cl, X = 1:S, function(s) {
  # generate data
  dat <- rnorm(n, mu, sigma)
  # calculate T1
  t1 <- mean(dat)
  # calculate T2
  t2 <- trimmean(dat)
  # calculate T3
  t3 <- median(dat)
  c(t1, t2, t3)
})

# convert t to a data frame with column t1, t2, t3
t_final <- do.call(rbind, t)
# save the results
saveRDS(t_final, "t_final.rds")
```

Passing argument

```
#!/bin/bash
#SBATCH --nodes=20
#SBATCH --ntasks-per-node=40
#SBATCH --time=0-01:30           # time (DD-HH:MM)
module load gcc/9.3.0 openmpi/4.0.3 r/4.0.2

R_PROFILE=${HOME}/R/x86_64-pc-linux-gnu-library/4.0/snow/RMPISNOWprofile; export R_PROFILE
mpirun -np 800 -bind-to core:overload-allowed R CMD BATCH --no-save "--args $n" example.R
```

Save it as `submit.sh`

Submit the job by

```
for n in 100 200 300 400
do
  sbatch --export=n=$n submit.sh
done
```