Parallel Computations of Response Surface Regression in R

Hao Yu

Department of Statistical & Actuarial Sciences, The University of Western Ontario

August 2, 2010

Abstract

- Many test statistics under null hypothesis don't have closed forms of finite-sample distributions and hence must reply on asymptotic distributions to find critical values or p-values. This often leads to inaccurate assessment of the hypothesis tests when sample sizes are relative small.
- One solution is to use response surface regression (RSR) technique. It requires mass simulation to estimate quantiles of the finite-sample distribution or asymptotic approximation with replications for each of different sample sizes.
- We use R package Rmpi to implement RSR. Rmpi is a wrapper to MPI, the most used parallel computing tool. Rmpi has its own interactive master and slaves environment. It can be run under Windows, Mac OS X, and various Linux platforms.
- Jarque-Bera normality test statistic is used as an example.

Objectives

- Introduce RSR.
- Introduce Rmpi and its various parallel apply functions.
- Implement general procedures of RSR with Rmpi.
- Find RSR for Jarque-Bera normality test.

Background on RSR

- Let $T(\underline{X})$ be a statistic of estimating a unknown parameter θ, with sample $\underline{X} = (X_1, \ldots, X_n)$. Except some simple cases, we don't have the closed distribution form of $T(\underline{X})$. Often we reply on its asymptotic result for proper inference

$$a_n(T(\underline{X}) - \theta) \xrightarrow{\mathcal{D}} W,$$

  where $a_n > 0$ and $a_n \to 0$.
- Under a null hypothesis of interest, we use W to find critical values and/or p-values. This leads to inaccurate assessment of the hypothesis test when sample sizes are small.
- MacKinnon (2002) proposed to use RSR simulation technique to find quantiles of $T(\underline{X})$ under the null hypothesis.
- Simulation steps in RSR. We assume that $\underline{X}$ can be simulated under a null hypothesis repeatedly.
  - Choose a proper set of sample sizes, say, $n = 10, 20, \ldots, 90, 100, 200, \ldots, 900, 1000, \ldots, 5000$.
  - Choose a proper set of probabilities, say, $probs = 0.90, 0.95, 0.99$.
  - For each sample size n, compute N replications of $T(\underline{X})$ and find its corresponding quantiles at $probs$. Repeat the same procedure M times in order to find local variations of those quantiles.
- Estimating RSR steps.
  - For each $probs$ and sample size n, compute sample mean and variance of quantiles, denote them as $\bar{q}(n, probs)$ and $v\bar{a}r(n, probs)$.
  - Set up the regression line

$$\bar{q}(n, probs) = k_0 + k_1 n^{-1/2} + k_2 n^{-1} + k_3 n^{-3/2} + k_4 n^{-2} + \text{error}.$$

  - Use R's lm to find RSR with $weights = 1/v\bar{a}r(n, probs)$ (weighted least squares).
- In order to estimate quantiles reliably, N should be at least 10,000 (prefer 100,000). For the same reason, M should be at least 100 (prefer 200). With $N = 100,000$ and $M = 200$, total simulations at each simple size is $N * M = 20,000,000$.
- Regression terms in RSR may differ for different types of statistics. For example, for unit root test, regression terms n^{-1}, n^{-2}, n^{-3} are preferred.
- Significance checking of regression terms can be carried out in lm and should be uniformly selected for all different quantiles.
- When simulation is carried out on a parallel cluster, it is important to choose and activate a proper parallel random number generator.

Rmpi package for R

- There are many parallel computational tools. Among them, Message Passing Interface (MPI) is the most widely used framework for parallel computing.
- MPI is not a new programming language; rather it is a collection of over 200 functions in either C(++) or Fortran.
- Rmpi (Yu, 2002) is an R interface (wrapper) to MPI.
- Rmpi is one of two core packages used for R's High-Performance Computing (HPC).
- Rmpi can be run under Windows, Mac OS X, and various Linux platforms though it does take a considerable time to setup initially.
- Additional information of parallel packages used for R's HPC can be found at `http://probability.ca/cran/web/views/HighPerformanceComputing.html`.
- Rmpi implements a set of MPI API extensions specifically designed for R or R slave environments.
  - MPI C-level functions are ported as R functions.
  - Rmpi loads MPI environment automatically.
  - Each running R (master or slave) is a stand alone process with MPI capabilities.
  - Rmpi is capable of exchanging complicated datasets.
  - Rmpi can call either rlecuyer or rsprng package to activate parallel random number generator.
  - Rmpi allows lower level programming or interacts with other non R processes.
- Rmpi has a set of parallel apply functions that can be used without excessive MPI codes.
  - mpi.parApply used as apply.
  - mpi.parLapply used as lapply.
  - mpi.parRapply used as rapply.
  - mpi.parSapply used as sapply.
  - mpi.parReplicate used as replicate.
- An example of using mpi.parReplicate. Assume that myfun() will be run repeatedly n times.

```
mpi.setup.rngstream()         #activiate random number generator
mpi.bcast.Robj2slave(myfun)   #send myfun object to slaves
mpi.parReplicate(n, myfun())  #repeat myfun() n times
```

  This will send myfun() to all available R slaves with jobs equally divided among slaves. A useful option is $job.num$ ($\geq$ total number of slaves). It splits n jobs into $job.num$ junks. This allows faster slaves to do more jobs (load balancing).

Rmpi implementation of RSR

- Implementation of RSR involves many N replications (jobs) which can be run independently on any of available R slaves. This is a typical "embarrassing parallel" job which is common for many Monte Carlo simulations.
- R master and slaves must coordinate with each other to avoid any deadlocks or race conditions.
- From R master point of view, jobs can be carried out through "push" them to slaves or "pull" them from salves. "push" is used to implement RSR.
- When R master pushes jobs to slaves, load balancing must be considered. To achieve load balancing for RSR, master will push jobs with largest sample sizes first to slaves.
- Functions used in implementing RSR.
  - mpi.rsr is the main function used by end users.
  - slaves.rsr is used by all slaves.
  - mpi.bcast.Robj2slave is used to send all necessary R objects to slaves.
- Flow charts of the function mpi.rsr
  - Tell slaves to run the function slave.rsr.
  - Send all necessary R objects to slaves.
  - Send first round of jobs with the largest sample sizes to slaves.
  - Have a while loop sending jobs to slaves. During a loop, collect a result and send out a new job with first come first reply policy (load balancing).
  - Collect final round of results and send signals to terminate each slave.
  - Return computed results.
- Sketch of the function slave.rsr.

```
slave.rsr= function(){
    ... #codes to collect R objects sent by master
    #get the sample size for the first job
    loop = mpi.recv(integer(1), type=1, source=0, tag=88)
    while (loop > 0) {
        out=try(replicate(N, do.call(".tmp.statistic",
            c(list(do.call(".tmp.rand.gen",
            c(list(n[loop]), rand.arg))), stat.arg))))
     out2=try(apply(out,1, quantile, probs=probs))
     mpi.send.Robj(out2, dest=0, tag=loop)
     loop = mpi.recv(integer(1), type=1, source=0, tag=88)
    }
}
```

- Rmpi codes for RSR can be obtained at `www.stats.uwo.ca/faculty/yu/Rmpi`.

Jarque-Bera normality test

- In econometrics, Jarque-Bera normality test is commonly used to test if data are from normal distribution population.
- Let $X_1, \ldots, X_n$ be iid with d.f. F. Then sample skewness and kurtosis are defined as

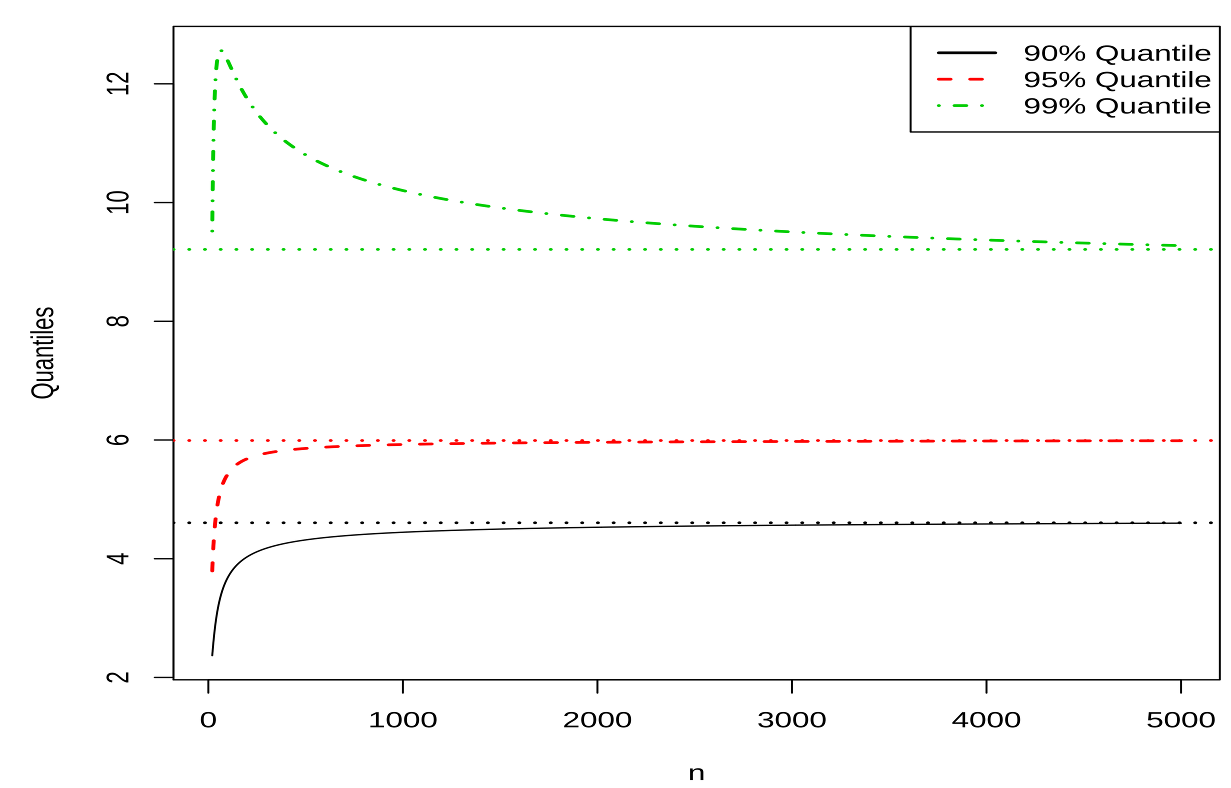
$$\hat{\gamma}_n = \frac{\sum_{t=1}^{n}(X_t - \bar{X})^3/n}{\sigma_n^3} \text{ and } \hat{\kappa}_n = \frac{\sum_{t=1}^{n}(X_t - \bar{X})^4/n}{\sigma_n^4},$$

  where $\bar{X} = \sum_{t=1}^{n} X_t/n$ and $\sigma_n^2 = \sum_{t=1}^{n}(X_t - \bar{X})^2/n$.
- Then Jarque-Bera statistic is defined as

$$JB = \frac{n}{6}\hat{\gamma}_n^2 + \frac{n}{24}(\hat{\kappa}_n - 3)^2.$$

- Under $H_0: F$ =normal, $JB \to \chi^2(2)$ in distribution as $n \to \infty$.
- On a cluster with 56 CPUs (Intel Xeon E5345 2.33GHz), it took about 4 hours to compute RSR for JB. Total CPU hours = 4*56 = 224 hours ≈ 9 days.
- 95% quantile RSR for JB

$$q(n, .95) = 6.01755 - 1.42389/\sqrt{n} - 50.25321/n + 54.359752/n^{3/2}.$$



References

[1] MacKinnon, J. G. (2002). Computing numerical distribution functions in econometrics. In proceedings of High Performance Computing Systems and Applications, edited by Pollard, A., Mewhort, D. J. and Weaver, D. F. Springer US. Vol. 451, 455–471.

[2] Yu, H. (2002). Rmpi: Parallel Statistical Computing in R. R News. Vol. 2, 10–14.