Available online at www.sciencedirect.com
ScienceDirect
Computational Statistics & Data Analysis 51 (2007) 2747–2752
COMPUTATIONAL STATISTICS & DATA ANALYSIS
www.elsevier.com/locate/csda

Optimization of large simulations using statistical software

Ilya Novikov*, Bernice Oberman

Gertner Institute for Epidemiology and Health Policy Research, Tel Hashomer, 52621 Ramat Gan, Israel

Received 15 November 2005; received in revised form 22 June 2006; accepted 22 June 2006
Available online 18 July 2006

Abstract

Many applications utilize time and memory intensive simulations. We demonstrate a method of decreasing necessary space to 5% or less without losing time, compared to conventional ways, applicable to any statistical software. The idea is to perform simulations in portions. Statistics are calculated for each portion and stored with the current seed(s). After running all portions, the full set of statistics is analyzed. For the same starting seed(s) we obtain the same results for any number of portions. We give examples of programs for SAS and S-PLUS.

This approach is important for performing large simulations on multi-user systems and small computers.

Keywords: Effective statistical simulation; SAS programming; S-PLUS programming; Space saving

1. Introduction

The efficient programming of simulations has become an important part of statistical work, especially when the size of the simulations is very large, as in weather systems (e.g. Zhang and Garbrecht, 2003), genetics (e.g. Griffiths and Tavaré, 1996), and other applications. For example, a popular bootstrap estimation of confidence intervals in multivariate analyses use re-sampling with replications from the initial sample, using 1000–10,000 repetitions. Each repetition generates a bootstrap sample of size equal to that of the initial sample, which may be thousands or tens of thousands of multivariate observations. Sometimes, simulations of a bootstrap procedure are needed. When doing this, hundreds or thousands of initial samples can be generated, thus increasing the computing task by orders of magnitude. Bootstrap methods have increased in importance with the advent of more powerful computers (Fan et al., 2002), and special procedures for this type of bootstrapping exist in many statistical software packages including the BOOT macro in SAS (SAS Institute, 2000) and 'bootstrap' command in S-Plus.

Statistical simulations comprise generating a set of similar 'individual' samples, performing the same first level analysis for each sample and analyzing the series of the results of these analyses to produce the final summaries. There are two basic approaches to this computing problem, denoted as A and B.

* Corresponding author. Tel.: +972 3 5303835; fax: +972 3 5349607.
E-mail address: ilian@gertner.health.gov.il (I. Novikov).

0167-9473/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2006.06.018

In approach A, one can generate and store a (sometimes huge) 'GENERAL' data set that contains all the individual samples, usually with the label (sequential number) of the individual sample. Then the first level analysis is performed for each sample and results stored in the RESULTS data set. Finally, the summary analysis is performed using the RESULTS data set. The size of the RESULTS data set is approximately equal to the size of the GENERAL data set divided by the size of an individual sample. Approach A requires intensive use of external memory for dealing with the GENERAL data set and may not even be feasible if there is insufficient memory for the GENERAL data set.

In approach B, one individual sample at a time is generated and analyzed, appending the results to a special 'RESULTS' data set and erasing the individual data set after each analysis. When the analyses of all individual data sets have been obtained, a summary analysis of the RESULTS data set is performed. Approach B is slower than approach A because it involves calling the analysis program thousands of times.

Both of these approaches are non-optimal with respect to computing time for large simulations, while approach B is sub-optimal with respect to memory.

2. Our approach to simulations

We propose making use of an intermediate approach in which the $N(=kM)$ individual samples are simulated in k portions. We generate a number, M (smaller than N) of samples, store them in a GENERAL data set, perform the first level analysis on each, and save the results by appending them to the RESULTS data set. The GENERAL data set is then emptied and refilled with the next portion of M samples. For continuity in the series of analyses, one has to store the seed(s) and (sometimes, for specific purposes) the SERIAL number of the sample at the end of each partial run, and use them for the generation and analysis of the next portion. When the predefined number, N, of samples has been generated and analyzed, the final analysis is performed on the RESULTS data set. It includes N observations, one per sample, each containing the statistics for one sample. In approach A, the GENERAL data set will comprise $k * M * n_{\text{obs}}$ observations, where n_{obs} is the number of observations in one sample, thus being approximately n_{obs} times larger than the RESULTS data set. Thus, the size of the RESULTS data set is negligible compared to the GENERAL data set for approach A, especially for problems with large samples. Therefore, by performing the simulations in k portions we use only about $1/k$ of the space that is required for the GENERAL data set in approach A. Surprisingly, there is a simultaneous gain in run time when k is not too big (20–100).

SAS (SAS Institute Inc.) and S-PLUS are popular statistical packages. They are quite different in their approach to programming in general and simulations in particular. However, the most general principles are valid for both of them and for other packages. We present two realizations of our proposal, in SAS and in S-PLUS. Since we are not interested in comparing the performance of the two packages, all run times shown are relative to the time of a simulation with $k = 1$ (approach A), thus making all comparisons only within each package. All system parameters used were the default ones.

3. Example

The goals of this example are simply to demonstrate how to perform the simulations using our approach and to show the gain in space and time of execution that may be obtained depending on the parameters k and M. Although we have mentioned the bootstrap as one of the procedures for which our proposed method may confer benefits, our example is a simpler one involving a simulation to test the outliers of a random number generator. We would characterize the example as involving a simulation of only medium size, but still big enough to show the effect of our approach.

Consider a program that tests the symmetry of outliers in a large pseudo-random normally distributed series, generated with the SAS or S-PLUS random number generators (RANNOR for SAS and *rnorm* for S-PLUS). This is of interest since extreme values are especially important for some applications and because most commercial random number generators do not satisfy some tests for randomness (Klimasauskas, 2002). For each generated sample of 4000 observations, the maximum and minimum values were stored, and then, over 2520 simulations of such samples, the equality $|\max| = |\min|$ was tested. The number 2520 $(=5 * 7 * 8 * 9)$ was chosen to allow comparison of several divisions of the simulations into k sets of M samples, where k and M are whole numbers. In SAS we used Macro (SAS Institute, 1999) and BY techniques (Novikov, 2003), for running the analyses. In S-PLUS we used matrix operations

Table 1
Relative times for 2520 simulations for various numbers of portions k and number of samples on one portion M, such that $k * M = 2520$, taking time for $k = 1$ as the unit

Number of portions (k)	Number of samples in one portion (M)	Relative time versus approach A	
		SAS	S-PLUS
(A)			
1	2520	1	1
2	1260	1.010184	0.967853
3	840	1.011328	0.968427
4	630	0.989282	0.962687
5	504	1.006560	0.964983
6	420	0.961858	0.960390
7	360	1.010756	0.961538
8	315	0.952361	0.964409
9	280	1.080479	0.964983
10	252	1.005378	0.971297
12	210	1.045885	0.970723
15	168	0.978564	0.960964
18	140	1.003585	0.953502
20	126	0.814364	0.975316
24	105	0.909452	0.950057
30	84	0.934472	0.961538
40	63	0.803951	0.963835
56	45	0.705584	0.963261
72	35	0.719925	0.974168
105	24	0.712182	0.990241
168	15	0.806927	0.995408
252	10	0.958921	1.029277
504	5	1.477992	1.033295
630	4	1.748570	1.044202
840	3	2.224693	1.081515
1260	2	3.119231	1.130310
(B)			
2520	1	5.773019	1.270379

(S-PLUS 2000 User's Guide, 1999). We found that both RANNOR in SAS (paired t-test p-value $= 0.33$) and rnorm in S-PLUS (paired t-test p-value $= 0.37$) performed well.

The total number of observations was $2520 * 4000 = 10,080,000$. In SAS we kept two variables in each observation (the SERIAL number and the value itself). The GENERAL data set for $k = 1$ in SAS used 161.28 MB. In S-PLUS, the same GENERAL data set occupies 80.64 MB since we did not include the SERIAL variable.

Table 1 presents the relative run times for selected values of k. The first row ($k = 1$) corresponds to approach A, and the last row ($k = 2520$) corresponds to approach B. The results show that for k between 20 and 100, not only was memory saved, but also run time was gained for both packages. For SAS, the relative time decreased from 1 for $k = 1$ when the number of portions is 20–250, reaching a minimum of 0.70 for $k = 56$ and then increased up to 5.77 for $k = 2520$ (approach B). For S-PLUS the effects on time were not dramatic, but relative run times were slightly decreased in the range of k below 168, with the minimum value of 0.95 at $k = 24$, and then slightly increased up to the maximum value of 1.27, this again occurring for approach B.

There were some other differences between SAS and S-PLUS in Table 1. In SAS there was a practically linear growth of relative time when k increased from 50 to 2520 with a coefficient of about 0.00257. This is the relative time necessary for performing all actions for one loop, including generation of the k copies of the text, compiling the text and for working with external memory to build a GENERAL data set, reading it for calculations, saving results in an intermediate data set and appending them to the RESULTS data set. This demonstrates the inefficiency of approach A (see also Novikov, 2003). For S-PLUS the growth is more rapid than linear; however, the linear coefficient is much lower, being 0.00011 at the last segment from $k = 1260$ to $k = 2520$.

We repeated our calculations using various computers with various default settings. The times were frequently different but the general picture we have described always remained the same.

4. Conclusion

The proposed approach follows general ideas used in system programming. As far as we know it has not been applied generally to statistical simulations. However, the bootstrap function in S-PLUS permits use of our approach for bootstrapping since it includes a parameter 'block' which is equivalent to our parameter k.

In practice, we recommend using between 20–50 portions, which will save about 95% of space and should not increase and may even simultaneously decrease the run time in comparison with using one GENERAL data set.

We have used our approach in different simulation problems, including regression models, and obtained results similar to those given in the example.

We believe that our approach is a quite general, simple, and effective method that can be widely used for simulations using any statistical software. This method may be particularly useful for performing very large simulations on any machine, and for medium to large simulations on small personal computers or in a multi-user environment, such as a university.

Acknowledgments

The authors thank Havi Murad for her encouragement in publishing this communication and Laurence Freedman for help in improving the text.

Appendix A. SAS program

```
* Testing symmetry of RANNOR by comparisons abs(MAX) and abs(MIN);
* nobs = number of objects in one sample;
* M = number of individual data sets in one portion;
* k = number of portions, seed = initial number for random number generator;
*************************************************************;
options nosource nonotes noprintmsglist ;          * otherwise log may be overflowed   ;
%macro symm (nobs,M,seed,k) ;
   %let start=12 : 00 : 00.000 ;                   * initializing start variable       ;
   data getstart ;                                 * fixing time of start              ;
      sta = time();
      star = put(sta,time12.3) ;
      call symput('start',trim(right(star)));      * saving time of start              ;
   data RESULTS ;                                  * initialization of the results     ;
      minx = .; maxx = .; SERIAL = .; _
      TYPE_ = . ; _FREQ_ = . ;
   run;
   %let rep = 0 ;
   %do ik = 1 %to &k;                              * external level                    ;
      data GENERAL ; keep SERIAL x ;               * generation of one part            ;
      seedx = %eval(&seed);
      SERIAL = %eval(&rep);
      do i = 1 to &M ;                             * internal level                    ;
         SERIAL + 1 ;
         do j = 1 to &nobs ;                       * one series                        ;
            call rannor(seedx,x) ; output ;
         end ;                                     * end of one series                 ;
      end;                                         * end of the portion                ;
```

```
        call symput('seed',trim(right(seedx)));          * saving seed and SERIAL      ;
        call symput('rep', trim(right(SERIAL)));
    run;
    proc means data = GENERAL noprint ;                  * calculations for one portion ;
    var x;
    output out = b min(x) = minx max(x) = maxx;          * saving current results     ;
    by SERIAL ;
run;
proc append base=RESULTS force;                          * adding the portion of results ;
%end;                                                    * end of external level      ;
data RESULTS ;                                           * summarizing calculations   ;
    set RESULTS end=eof;
    where (minx^= .) ;
    diff = abs(maxx)-abs(minx) ;                         * difference = | max |-| min | ;
run;
proc means n mean max min probt;                         * Testing symmetry           ;
    var diff ;
data final ;                                             * estimation of running time ;
    set getstart;
    file print ;
    finish = time() ;                                    * fixing time of the end     ;
    delta = finish-sta ;                                 * delta = running time       ;
    put " k=&k START = &start FINISH = " finish
          time12.3 " delta = "  delta time12.3 ;
        run;
    %mend symm ;

%symm(4000,2520,1234,1) ;
```

Appendix B. S-PLUS program

```
function(k, m, nobs, seed)
{
    start <- proc.time()                                  #record time at beginning
    set.seed(seed)                                        #set initial seed
    results <- matrix(0, nrow = as.numeric(k * m), ncol = 3)
                                                          #set up matrix for results
    serial <- 0                                           #initialise serial
        for(ik in 1:k) {
                                                          #loop on portion
        GENERAL <- matrix(0, nrow = as.numeric(m) * nobs, ncol = 1)
                                                          #set up temporary matrix for each portion
            for(i in 1:m) {
                                                          #loop on dataset
                serial <- serial +1                       #increment serial number
                GENERAL[(((i - 1) * nobs) + 1):(i * nobs), 1] <- rnorm(nobs)
                                                          #enter random numbers from normal distribution into
                                                          #temporary matrix
```

```
      results[i + (m * (ik - 1)), 1] <- min(GENERAL[(((i - 1) * nobs) + 1):(i * nobs), 1])
                                                                  #find min for the generated normal numbers
      results[i + (m * (ik - 1)), 2] <- max(GENERAL[(((i - 1) * nobs) + 1):(i * nobs), 1])
                                                                  #find max for the generated normal numbers
    results[i + (m * (ik - 1)), 3] <- serial                      #save serial
    set.seed <- .Random.seed[1]
                                                                  #ensure that seed gets last value of random seed
    }
  }
  temp <- abs(results[, 2]) - abs(results[, 1])                   #find differences for entire file
  n <- length(temp)                                               #find statistics
  minimum <- min(temp)
  maximum <- max(temp)
  average <- mean(temp)
  end2 <- (proc.time() - start)                                   #find end time
  return(start, end2, n, minimum, maximum, average)
                                                                  #return statistics
}
```

References

Fan, X., Felsövályi, Á., Sivo, S.A., Keenan, S.C., 2002. SAS® for Monte Carlo Studies: A Guide for Quantitative Researchers. SAS Institute Inc., Cary, NC.

Griffiths, R.C., Tavaré, S., 1996. Monte Carlo inference methods in population genetics. Math. Comput. Model. 23, 141–158.

Klimasauskas, C.C., 2002. Not knowing your random number generator could be costly: random generators—why they are important. PCAI 16 (3), 52–58.

Novikov, I., 2003. A remark on efficient simulations in SAS. Statistician 52 (Part 1), 83–86.

SAS Institute Inc. SAS Language Reference, Version 8 (1999). SAS Institute Inc., Cary, NC.

SAS Institute Inc. Sample 479: Jackknife and Bootstrap Analyses, 2000 ⟨http://support.sas.com/ctx/samples/index.jsp?sid = 479#ref⟩.

S-PLUS 2000 User's Guide, 1999. Data Analysis Products Division, Mathsoft, Seattle, WA.

Zhang, X.C., Garbrecht, J.D., 2003. Evaluation of CLIGEN precipitation parameters and their implication on WEPP runoff and erosion prediction. Trans. ASAE 46 (2), 311–320.