Supercomputers for Monte Carlo simulation

Kleijnen, J.P.C.

Published in:
Bootstrapping and related techniques

Publication date:
1992

SUPERCOMPUTERS FOR MONTE CARLO SIMULATION:

CROSS-VALIDATION VERSUS RAO'S TEST

IN MULTIVARIATE REGRESSION

Jack P.C. Kleijnen
Katholieke Universiteit Brabant (Tilburg University)
Postbox 90153, 5000 LE Tilburg, Netherlands
Bitnet: kleijnen@kub.nl Fax: +3113-663072

Abstract

Part I covers vector computers, in the context of Monte Carlo experiments wit regression models. These computers should exploit a specific dimension of th Monte Carlo experiment, namely its replicates. The resulting code computes Ordi nary Least Squares (OLS) estimates on a CYBER 205 in 2% of the time needed on Vax 8700. For Generalized Least Squares, however, the code runs slower on th CYBER 205 if the regression model is small; for large models the CYBER runs muc faster. Part II covers regression models with correlated errors. To test th validity of the specified regression model, Rao (1959) generalized the F statis tic for lack of fit, whereas Kleijnen (1983) proposed cross-validation usin Student's t statistic and Bonferroni's inequality. A large Monte Carlo experi ment compares these two methods, for normal and non-normal errors. Under norma lity, cross-validation is conservative, whereas Rao's test realizes its nomina type I error and has high power. Several confidence interval procedures fo regression parameters are also compared. Under lognormality, only cross-valida tion with OLS works.

Part I: Vector computers

Part I is based on Kleijnen and Annink (1990). It illustrates three important points about "supercomputers":

(i) Efficient supercomputing requires that algorithms be adjusted to take advantage of the specific architecture of the computing hardware.

(ii) For certain problems, expensive supercomputers may be slower than general purpose machines are.

(iii) The increased speed of the supercomputers may not outweigh the burden of constructing the specialized code: the researcher's time is valuable too.

Vector computers are to be distinguished from traditional scalar computers and truly parallel computers. Traditional computers such as the IBM 370 and the VAX series, execute one instruction after the other; so they operate sequentially. Truly parallel computers such as the HYPERCUBE, have many Central Processing Units (CPU's) that can operate independently of each other; this is called coarse grain parallelism. Vector computers such as the CRAY 1 and the CYBER 205, have a "vector processing" capability: fine grain parallelism. Consider, for example, the computation of the inner product of two vectors: $v_1' v_2 = \Sigma_{j=1}^{n} v_{1j} v_{2j}$. This computation requires n identical scalar operations $v_{1j}\ v_{2j}$. The vector processor starts computing $v_{1j}\ v_{2j}$ while the computation of the predecessors $v_{1(j-1)}\ v_{2(j-1)}$, $v_{1(j-2)}\ v_{2(j-2)}$, ... is still in process! So a vector computer works as an assembly line. A technical condition is that the scalar operations do not depend on each other; in the example the computation of the scalar product $v_{1j}\ v_{2j}$ does not need the other scalar products. Such an architecture is called a pipeline. The pipeline or assembly line requires a fixed set up cost; consequently a vector computer works efficiently only if a "large" number of identical scalar operations can be executed independently of each other. In Part I we show how to formulate the Monte Carlo model such that a vector computer can be applied efficiently. Technical details on the new generation of "supercomputers" are given by Levine (1982); more references are provided in Kleijnen and Annink (1989).

1. Regression Models and Simulation

Consider the well-known linear regression model

$$E(\mathbf{y}) = \mathbf{X}\,\beta \qquad (1.1)$$

with $\mathbf{y} = (y_1,\ldots,y_i,\ldots,y_n)'$, $\beta = (\beta_1,\ldots,\beta_j,\ldots,\beta_Q)'$ and $\mathbf{X} = (x_{ij})$ where i 1,...,n and j = 1,...Q. We assume additive errors $\mathbf{e} = (e_1,\ldots,e_i,\ldots,e_n)'$:

$$\mathbf{y} = \mathbf{X}\,\beta + \mathbf{e}. \qquad (1.2$$

We further assume that **e** is n-variate normally (N_n) distributed:

$$\mathbf{e} \sim N_n\,[0_n,\ \mathbf{cov}(\mathbf{e})], \qquad (1.3$$

where 0_n denotes a column of n zeros; **cov(e)** denotes the variance-covariance matri of **e**; **cov(e)** equals **cov(y)** because of (1.2); **cov(y)** is assumed to be nonsingular.

This regression model can be applied to analyze the results of a simulatic experiment that uses the *same* pseudorandom number sequence; see Kleijnen (1987) ar Kleijnen (1988). In a well designed simulation experiment it is easy to *replicate* each factor combination; that is, row i of **X** is observed m ≥ 2 times. Unbiased esti mators of $\sigma_{ih} = \mathrm{cov}(y_i,y_h) = \mathrm{cov}(y_{ir},y_{hr})$ where y_{ir} is the r^{th} replication of th i^{th} factor combination, are:

$$\hat{\mathbf{cov}}(\mathbf{y}) = (\mathbf{Y}\,\mathbf{Y}' - \bar{\mathbf{y}}\,\bar{\mathbf{y}}'\mathbf{m})/(\mathbf{m}-1), \qquad (1.4$$

with $\hat{\mathbf{cov}}(\mathbf{y}) = (\hat{\sigma}_{ih})$, $\mathbf{Y} = (y_{ir})$, $\bar{\mathbf{y}} = (\bar{y}_i)$ with the averages $\bar{y}_i = \Sigma_{r=1}^{m}\, y_{ir}/m$; b definition we have $\sigma_{ii} = \sigma_i^2$. It is simple to prove that $\hat{\mathbf{cov}}(\mathbf{y})$ is singular for m n.

We consider two different point estimators for $\underline{\beta}$. The classic estimator use *Ordinary Least Squares* or OLS:

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\bar{\mathbf{y}}\ , \qquad (1.5$$

which assumes n ≥ Q and rank (**X**) = Q. If **cov(y)** were known, then a better estimat would use Generalized Least Squares (GLS). Since **cov(y)** is unknown in practice, replace it by the estimator $\hat{\mathbf{cov}}(\mathbf{y})$ of (1.4), and use *Estimated Generalized Lea Squares* or EGLS:

$$\tilde{\beta} = (\mathbf{X}'\hat{\mathbf{cov}}(\mathbf{y})^{-1}\mathbf{X})^{-1}\mathbf{X}\ \hat{\mathbf{cov}}(\mathbf{y})^{-1}\bar{\mathbf{y}}\ , \qquad (1.6$$

which assumes that $\hat{\mathbf{cov}}(\mathbf{y})$ is non-singular.

2. Vectorizing the Monte Carlo program

We wish to construct a Monte Carlo program to compare the OLS and EGLS estimators of (1.5) and (1.6). We might use the vector mode to compute an individual element y_{ir} of **Y**; see (1.2). But **X** is n x Q; typically n and Q range between n = 8 and Q = 4 and n = 32 and Q = 22. Vector computers are inefficient if the number of parallel operations is "small", say, smaller than 50; see Levine (1982) and SARA (1984). So it is inefficient to vectorize the computation of an individual y_{ir}.

Next we consider the vectorization of either the rows or the columns of **Y**. Since there are only n rows (factor combinations), vectorization is again inefficient. Because the m columns of **Y** are statistically independent (see § 1), vectorization is possible. In practice, however, m will be small (the minimum is m = n + 1; otherwise, $\hat{\text{cov}}(y)$ is singular). So vectorizing the columns of **Y** is also inefficient.

The Monte Carlo experiment is replicated L = 100 times, as we shall see in Part 2. (These Monte Carlo replicates ℓ with $\ell = 1,\ldots,L$, must be distinguished from the simulation replicates $r = 1,\ldots,m$.) The L Monte Carlo replicates are statistically independent; they can be vectorized as we show now. Note that the more replicates we wish to obtain, the more efficient the vector computer becomes. Vectorization may be compared to filling a three-dimensional box in parallel with errors $e_{ir\ell}$ with $i = 1,\ldots,n$; $r = 1,\ldots,m$; $\ell = 1,\ldots,L$.

Step 1: Sample pseudorandom numbers x in parallel

Kleijnen (1989) evaluates several procedures for the parallel generation of pseudorandom numbers $x \sim U(0,1)$. Kleijnen and Annink (1989) recommend the following generator. Take a scalar multiplicative congruential generator with a multiplier that gives acceptable statistical behavior. To initialize the vector version of this generator, first generate - in scalar mode - a vector of J successive pseudorandom integers $\mathbf{x} = (x_0, x_1, x_2, \ldots, x_{J-2}, x_{J-1})'$ with seed x_0 and $x_j = (a\, x_{j-1}) \bmod m$ for $j = 1,2,\ldots, J-1$. To obtain numbers between zero and one, divide by m. Once and for all, compute a scalar multiplier $(a^J) \bmod m$. Vector multiplication of the vector **x** with this scalar multiplier gives a new vector: $(x_J, x_{J+1}, \ldots, x_{2J-2}, x_{2J-1})'$. In this way the pseudorandom numbers are generated in parallel, in exactly the same order as they would have been produced in scalar mode. At the end of the Monte Carlo experiment the vector of the last J numbers should be stored, so that the experiment can be continued later on.

We mentioned that vector computers become more efficient as the number of parallel operations increases. For the CYBER 205, however, there is a technical upper limit: $J = 2^{16} - 1 = 65\,535$ (since this computer uses 16 bits for addressing; see SARA, 1984, p. 26).

There is a computational problem: overflow occurs when computing $(a^J) \bmod m$. This problem is solved through the techniques of "controlled integer overflow" and the CYBER 205's "two's complement" representation of negative integers; see Kleijnen and Annink (1989).

Step 2: Sample independent standard normal variates z in parallel

There are many techniques for generating normal variates. We select the following procedure that fits a vector computer:

$$z_1 = (-2 \ln x_1)^{\frac{1}{2}} \cos 2\pi x_2 \qquad (2.1.a)$$

$$z_2 = (-2 \ln x_1)^{\frac{1}{2}} \sin 2\pi x_2, \qquad (2.1.b)$$

where the mutually independent pair x_1 and x_2 with $x \sim U(0,1)$ yields the mutually independent pair z_1 and z_2 with $z \sim N(0,1)$. To compute the functions $\ln$, cos, and sin for a *vector* of numbers, we use FORTRAN 200's vector functions VLN, VCOS, and VSIN. Given a vector of L independent pseudorandom numbers x, we use the first half to compute L/2 independent parallel realizations of $\ln x_1$, and the second half to compute cos $(2\pi x_2)$ and sin $(2\pi x_2)$: Figure 1 gives a pseudo-FORTRAN program where π is computed through the arccosine function; see SARA (1984, p. 13). To convert this pseudo-FORTRAN into a FORTRAN 200 program, we can replace DO loops by the special syntax of FORTRAN 200; the supercomputer can also automatically translate the FORTRAN program of Figure 1 provided we add CONTINUE statements; see SARA (1984, p. 17).

FIGURE 1

Parallel computation of L variates $z \sim N(0,1)$.

```
    L2 = L/2; PI = ACOS(-1.0); C = 2 * PI
    DO   20   LL = 1, L2
20      HELP1(LL) = SQRT(-2 * LOG(X(LL)))
    DO   30   LL = 1, L2
        HELP2(LL) = COS(X(LL + L2) * C)
        HELP3(LL) = SIN(X(LL + L2) * C)
        Z(LL) = HELP1(LL) * HELP2(LL)
30      Z(L2 + LL) = HELP1(LL) * HELP3(LL)
```

Above we saw that we wish to fill a three-dimensional "box" with $e_{ir\ell}$. So we store the *vectors* z (with L elements) of Figure 1 into a three-dimensional *array*

PD Dr. Karl-Heinz Jöckel
Bremen Institute for Prevention Research
and Social Medicine (BIPS)
Grünenstraße 120, W-2800 Bremen, FRG

Prof. Dr. Günter Rothe
Hauptverband der gewerblichen Berufsgenossenschaften e. V.
Alte Heerstraße 111, W-5205 Sankt Augustin, FRG

Prof. Dr. Wolfgang Sendler
University of Trier, Abteilung für angewandte Mathematik
Postfach 38 25, W-5500 Trier, FRG

ISBN 3-540-55003-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-55003-8 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
42/3140-543210 - Printed on acid-free paper