Supercomputers, Monte Carlo simulation and regression analysis

Kleijnen, J.P.C.; Annink, B.

Publication date:
1989

Citation for published version (APA):
Kleijnen, J. P. C., & Annink, B. (1989). *Supercomputers, Monte Carlo simulation and regression analysis*. (Research memorandum / Tilburg University, Department of Economics; Vol. FEW 402). Unknown Publisher.

[redacted]

[redacted]

8

SUPERCOMPUTERS, MONTE CARLO SIMULATION AND REGRESSION ANALYSIS

Jack P.C. Kleijnen
Ben Annink

FEW 402

SUPERCOMPUTERS, MONTE CARLO SIMULATION, AND REGRESSION ANALYSIS

Jack P.C. Kleijnen

and

Ben Annink

August 1989

Correspondence should be directed to: Prof. J.P.C. Kleijnen, Department of Information Systems and Auditing, School of Business and Economics, Catholic University Brabant (Katholieke Uiversiteit Brabant), 5000 LE Tilburg, Netherlands. FAX: 013-663019. E-mail: t435klei@htikub5.

SUPERCOMPUTERS, MONTE CARLO SIMULATION, AND REGRESSION ANALYSIS

Jack P.C. Kleijnen

and

Ben Annink

Department of Information Systems and Auditing, School of Business and Economics, Catholic University Brabant (Katholieke Uiversiteit Brabant), 5000 LE Tilburg, Netherlands. E-mail: t435klei@htikub5.

Supercomputers provide a new tool for management scientists. The application of this tool requires thinking in parallel or vector mode. This mode is examined in the context of Monte Carlo simulation experiments with multivariate regression models. The parallel mode needs to exploit a specific dimension of the Monte Carlo experiment (namely the replicates of that experiment). Then Ordinary Least Squares on a CYBER 205 takes only 1.4% of the time needed on a VAX 8700. Estimated Generalized Least Squares, however, is slower on the CYBER 205 because it requires matrix inversion.

(ADDITIONAL KEYWORDS: DISTRIBUTION SAMPLING; MULTIVARIATE DISTRIBUTION; COMMON SEEDS; METAMODEL)

1. Introduction

This paper focuses on the use of supercomputers in Monte Carlo experiments with multivariate regression analysis. Regression models can be used to determine a metamodel of a simulation model; see Kleijnen (1987, 1988). So this paper may be of interest to management scientists for several reasons:

(i) Regression analysis is often used by management scientists to analyse simulation data and real-world data.
(ii) The study shows how supercomputers can be applied in Monte Carlo experiments, which are related to stochastic simulation: both use pseudorandom numbers, but simulation is dynamic (a case in point is queuing simulation) whereas Monte Carlo experiments are not; see Teichrow (1965). So Monte Carlo experiments are simpler. Our study may challenge other researchers to apply supercomputers to Monte Carlo and simulation models.

A new development in management science is the advent of *supercomputers* or vector computers such as the CYBER 205. Their pipelined architecture distinguishes these computers for traditional scalar computers (for example, the IBM 370 and the VAX series) and from truly parallel computers (such as the HYPERCUBE). The challenge now is to think in the "parallel" mode; that is, the problem is to formulate mathematical models such that the vector mode of the supercomputer can be applied. Parallel thinking can be examplified by the innerproduct of two matrices $\underset{\sim}{v}_1' \underset{\sim}{v}_2 = \Sigma_1^n v_{1j} v_{2j}$. This computation requires n scalar operations $v_{1j} v_{2j}$; these n multiplications can be done in parallel; that is, to compute the product $v_{1j} v_{2j}$ the computer does not need $v_{1(j-1)} v_{2(j-1)}$. The pipeline architecture means that a supercomputer works as an assembly line: efficiency improves drastically if a large number of identical operations can be executed independently of each other; see Levine (1982).

Our paper is organized as follows. In § 2 we summarize the well-known multivariate linear regression model and its application in simulation experiments with common pseudorandom numbers. This regression model is to be studied, using Monte Carlo experimentation. In § 3 we show how the Monte Carlo program can be "vectorized" so that it can be run in parallel; we discover a "third dimension" of Monte Carlo experiments. § 4 gives conclusions. References and appendices complete the paper.

2. Multivariate Regression Models and Simulation

Consider the well-known linear regression model

$$E(\underset{\sim}{y}) = \underset{\sim}{X}\,\underset{\sim}{\beta} \tag{2.1}$$

with $\underset{\sim}{y} = (y_1,\dots,y_i,\dots,y_n)'$, $\beta = (\beta_1,\dots,\beta_j,\dots,\beta_Q)'$ and $\underset{\sim}{X} = (x_{ij})$ where $i = 1,\dots,n$ and $j = 1,\dots Q$, This model is <u>multivariate</u> if the errors $\underset{\sim}{e}$ = $(e_1,\dots,e_i,\dots,e_n)'$ are mutually dependent (the errors are also called disturbance or noise). We assume additive errors:

$$\underset{\sim}{y} = \underset{\sim}{X}\,\underset{\sim}{\beta} + \underset{\sim}{e} \tag{2.2}$$

We further assume that $\underset{\sim}{e}$ is multivariate normally (MN) distributed:

$$\underset{\sim}{e} \in MN(\underset{\sim}{\mu}_e, \underset{\sim}{\Omega}_e) \tag{2.3}$$

where $\underset{\sim}{\mu}_e = \underset{\sim}{0}$ (a column of n zero's) and $\underset{\sim}{\Omega}_e$ equals $\underset{\sim}{\Omega}_y$ because of (2.2); $\underset{\sim}{\Omega}_y$ is assumed to be non-singular. When this model is applied to simulation data, (2.1) is called the metamodel (the regression equation is a model of the simulation computer program; see Kleijnen, 1987); in (2.3) $\underset{\sim}{\Omega}_y$ is non-diagonal if common seeds are used for the pseudorandom number generator of the simulation model; see Kleijnen (1988).

We consider <u>experimental design</u> situations only: we assume that the independent regression variables $\underset{\sim}{X}$ in (2.1) follow from an experimental design for k factors: $\underset{\sim}{D} = (d_{ih})$ with $h = 1,\dots,k$. For example for i = $1,\dots,n$ we may have $x_{i1} = 1$ (dummy), $x_{i2} = d_{i1}$ (factor 1), $x_{i3} = \log d_{i2}$ (factor 2 on log scale), $x_{i4} = d_{i1}d_{i3}$ (interaction between factor 1 and 3). Counterexamples are provided by econometrics where $\underset{\sim}{X}$ can be observed only, not controlled; see Kleijnen (1987, p. 159). Moreover, in well designed experiments it is possible to <u>replicate</u> specific factor combinations; that is, row i of $\underset{\sim}{X}$ or $\underset{\sim}{x}_i' = (x_{i1},\dots,x_{ij},\dots,x_{iQ})$ can be observed $m_i \geq 2$ times. For example, in simulation the combination i of the simulation parameters d_h is run m_i times (a terminating simulation is repeated with m_i independent pseudorandom number streams; in non-terminating or steady-state simulations m_i subruns are obtained; see Kleijnen, 1987, pp.

8-10, 63-83). If all combinations of simulation parameters use the same seed for the pseudorandom number generator, then obviously m_i becomes a constant m. Outside a simulation context, Rao (1959) assumes m independent observations on the n-variate vector $\underset{\sim}{y}$. His assumption agrees with the simulation context of Table 1, which assumes independent seeds. These observations yield the following unbiased estimators of $\sigma_{ii'} = \text{cov}(y_i, y_{i'}) = \text{cov}(y_{ir}, y_{i'r})$:

$$\hat{\sigma}_{ii'} = \frac{\sum_{r=1}^{m} (y_{ir} - \bar{y}_i)(y_{i'r} - \bar{y}_{i'})}{m-1} \qquad (i, i' = 1, \ldots, n)(m \geq 2) \tag{2.4}$$

with the averages $\bar{y}_i = \sum_{r=1}^{m} y_{ir}/m$; by definition we have $\sigma_{ii} = \sigma_i^2$. In matrix notation (2.4) becomes

$$\hat{\underset{\sim}{\Omega}}_y = \underset{\sim}{Y}\,\underset{\sim}{Y}'/(m-1) - \bar{\underset{\sim}{y}}\,\bar{\underset{\sim}{y}}'m \tag{2.5}$$

with $\hat{\underset{\sim}{\Omega}}_y = (\hat{\sigma}_{ii'})$, $\underset{\sim}{Y} = (y_{ir})$ and $\bar{\underset{\sim}{y}} = (\bar{y}_i)$.

TABLE 1

Regression Data

Combination i (effects: $\beta_1 \ldots \beta_j \ldots \beta_Q$)	Responses y_{ir} (seed 1)...(seed r)...(seed m)	Average response $\bar{y}_i$	Estimated (co)variances $\hat{\sigma}_{ii'}$
$x_{11} \ldots x_{1j} \ldots x_{1Q}$	$y_{11} \quad \ldots \quad y_{1r} \quad \ldots \quad y_{1m}$	$\bar{y}_1$	$\hat{\sigma}_1^2 \hat{\sigma}_{12} \ldots \hat{\sigma}_{1n}$
$x_{21} \ldots x_{2j} \ldots x_{2Q}$	$y_{21} \quad \ldots \quad y_{2r} \quad \ldots \quad y_{2m}$	$\bar{y}_2$	$\hat{\sigma}_2^2 \ldots \hat{\sigma}_{2n}$
$x_{i1} \ldots x_{ij} \ldots x_{iQ}$	$y_{i1} \quad \ldots \quad y_{ir} \quad \ldots \quad y_{im}$	$\bar{y}_i$	$\hat{\sigma}_i^2 \ldots \hat{\sigma}_{in}$
$x_{n1} \ldots x_{nj} \ldots x_{nQ}$	$y_{n1} \quad \ldots \quad y_{nr} \quad \ldots \quad y_{nm}$	$\bar{y}_n$	σ_n^2

Kleijnen (1988, p.67) proposes two different point estimators for the regression parameters $\underset{\sim}{\beta}$. The first estimator uses <u>Ordinary Least Squares</u> or OLS:

$$\hat{\underset{\sim}{\beta}} = (\underset{\sim}{X}'\underset{\sim}{X})^{-1}\underset{\sim}{X}'\bar{\underset{\sim}{y}} \ , \tag{2.6}$$

which assumes n > Q. The second estimator uses <u>Estimated Generalized Least Squares</u> or EGLS:

$$\hat{\tilde{\underset{\sim}{\beta}}} = (\underset{\sim}{X}'\hat{\underset{\sim}{\Omega}}^{-1}\underset{\sim}{X})^{-1}\underset{\sim}{X}'\hat{\Omega}_{y}^{-1}\bar{\underset{\sim}{y}} \ , \tag{2.7}$$

which assumes that $\hat{\underset{\sim}{\Omega}}_{y}$ is non-singular; also see (2.3). The estimated covariance matrices of these two estimators are

$$\hat{\underset{\sim}{\Omega}}_{\hat{\beta}} = (\underset{\sim}{X}'\underset{\sim}{X})^{-1}\underset{\sim}{X}'\hat{\Omega}_{y}\underset{\sim}{X}(\underset{\sim}{X}'\underset{\sim}{X})^{-1}/m \tag{2.8}$$

and

$$\hat{\underset{\sim}{\Omega}}_{\hat{\beta}} \approx (\underset{\sim}{X}'\hat{\underset{\sim}{\Omega}}_{y}^{1}\underset{\sim}{X})^{-1}/m \tag{2.9}$$

where the symbol ≈ means that the equality holds only asymptotically. Obviously we have

$$\hat{\underset{\sim}{\Omega}}_{\bar{y}} = \hat{\underset{\sim}{\Omega}}_{y}/m \tag{2.10}$$

Monte Carlo experimentation enables us to study the statistical behavior of different regression procedures. For example, we may estimate the α and β errors of a test devised to detect a misspecified regression model; see Kleijnen (1988, p. 71). In this paper, however, we focus on the cumputer generation of the observations $Y = (y_{ir})$ and the estimators $\hat{\underset{\sim}{\beta}}$ and $\hat{\tilde{\underset{\sim}{\beta}}}$; see (2.2) through (2.7). The computation of other statistics is then straightforward. The Monte Carlo experiment is reblicated L times, say L = 100 (taking L = 100 means that estimated α and β errors have standard errors smaller than 0.05, since $\hat{\alpha}$ and $\hat{\beta}$ are binomially distributed).

3. Parallel Design of the Monte Carlo Program

In § 1 we emphasized that supercomputers work efficiently only if many parallel operations can be identified. An individual element y_{ir} in Table 1, defined by (2.2), can be computed in vector mode, but this mode is inefficient since typical values for n and Q are as small as 4 and 3. Alternatively, we may consider the parallel computation of either n rows (factor combinations) or m columns (independent observations or seeds). Let us first consider these two dimensions and then a third dimension.

The errors within a column are statistically dependent: they are n-variate normal. We first discuss programming for n = 2. We then sample the independent univariate standard normal variates z_1 and z_2, and compute the linear transformation $e_1 = \sigma_1 z_1$ and $e_2 = \sigma_2(\rho\, z_1 + (1-\rho^2)^{\frac{1}{2}} z_2)$. For general n the sampling subroutine for multivariate normal $\underset{\sim}{e}$ with covariance matrix $\underset{\sim}{\Omega}_y$ is

$$\underset{\sim}{e} = \underset{\sim}{C}\,\underset{\sim}{z}, \tag{3.1}$$

where $\underset{\sim}{z} = (z_1, \ldots, z_i, \ldots, z_n)'$ with independent $z_i \in N(0,1)$, and with $\underset{\sim}{C}$ a lowertriangular matrix defined by

$$\underset{\sim}{C}\,\underset{\sim}{C}' = \underset{\sim}{\Omega}_y, \tag{3.2}$$

which is computed by Choleski's technique; see Naylor et al. (1966. pp. 97-99) and standard software libraries such as IMSL and NAG. But (3.1) defines a *recursive* relation, and such relations are not efficiently handled by supercomputers. So we do not compute this dimension in parallel.

The columns in Table 1 are statistically independent, by definition (see the text above (2.4)). Hence a supercomputer can calculate these m observations in parallel. But it is well-known that supercomputers become efficient only if the number of parallel operations is "large", say, $m \geq 50$. In the Monte Carlo experiment we wish to study m equal to 2, 10, 25 and 50. So in most cases, parallel computation would be slower than scalar computation; also see Levine (1982) and SARA (1984).

But there is a *third dimension* in this problem! The Monte Carlo experiment is repeated 100 times (see § 2). We speak of "MC replicates" ℓ

with $\ell = 1,\ldots,L$ and $L = 100$ (which must be distinguished from the replicates $r = 1,\ldots,m$). These replicates are statistically independent and can be computed in parallel, as we shall see. The more replicates we wish to obtain (higher L), the more efficient the vector computer becomes. We may visualize our problem as follows. There is a three-dimensional box to be filled in parallel with errors $e_{ir\ell}$ with $i = 1,\ldots,n$; $r = 1,\ldots,m$; $\ell = 1,\ldots,L$. This box is filled in steps 1 through 3 below. In step 4 statistics such as $\hat{\Omega}_{\tilde{y}}$ are computed.

Step 1: Sample pseudorandom numbers x in parallel

Kleijnen (1989) evaluates several procedures for the parallel generation of pseudorandom number $x \in U(0,1)$. Kleijnen and Annink (1989) recommend the following standard scalar generator. Take a multiplicative congruential generator, since the statistical properties of such a generator are well-known. To initialize the parallel version of this generator, first generate - in vector mode - a vector of J successive pseudorandom integers $\tilde{x} = (x_0, x_1, x_2, \ldots, x_{J-2}, x_{J-1})$ with seed x_0 and $x_j = (a\ x_{j-1}) \bmod m$ for $j = 1,2,\ldots,\ J-1$. Once and for all compute a scalar multiplier: $(a^J) \bmod m$. Multiplication of the vector $\tilde{x}$ with this scalar multiplier gives a new vector: $(x_J, x_{J+1}, \ldots, x_{2J-2,2J-1})'$. In this way the pseudorandom numbers are generated 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 may be continued later on.

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

There is a computational problem: overflow occurs when computing $(a^J) \bmod m$. This problem is solved, using controlled integer overflow and the CYBER 205's two's complement representation of negative integers. Appendix 1 gives the compter program.

Step 2: Sample independent standard normal variates z in parallel

There are several techniques for generating $z \in N(0,1)$; see Devroye (1986). We take a procedure that fits a vector computer:

$$z_1 = (-2 \ln x_1)^{\frac{1}{2}} \cos 2\pi x_2 \tag{3.3.a}$$

$$z_2 = (-2 \ln x_1)^{\frac{1}{2}} \sin 2\pi x_2, \tag{3.3.b}$$

where the mutually independent pair x_1 and x_2 yields the mutually independent pair z_1 and z_2. To compute the functions $\ln$, cos and sin for a *vector* of numbers, we use FORTRAN 200's vector functions VLN, VCOS, and VSIN. So, 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; 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 CDC (1986), SARA (1984, p. 17).

FIGURE 1

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

```
        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)
```

Note that Petersen (1988) generates z in parallel, using not (3.3) but Teichroew's procedure described in Naylor et al. (1966, p. 94).

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

Step 3: Sample n-variate $\underset{\sim}{e}$

The error vector $\underset{\sim}{e} = (e_1,\ldots,e_i,\ldots,e_n)'$ is multivariate normal with mean zero and covariance matrix $\underset{\sim}{\Omega}_y$; see (2.3). To generate $\underset{\sim}{e}$ we linearly transform the n-variate vector of independent standard normal variates $\underset{\sim}{z} = (z_1,\ldots,z_n)'$; see (3.1). This transformation uses the lower triangular matrix C of (3.2). This yields

$$e_i = \sum_{j=1}^{i} c_{ij} z_i \qquad (i=1,\ldots,n). \tag{3.3}$$

To obtain M observations and L Monte Carlo replicates of $\underset{\sim}{e}$, we might apply the naive FORTRAN program of Figure 2, where M denotes the maximum value of m in the experiment (here M = 50) and E(I,R,LL) is zero initially. Note that $\underset{\sim}{C}$ or C(I.J) does not vary over seeds (R) and Monte Carlo replicates (LL); it does vary over the Monte Carlo experiments defined by $\underset{\sim}{\Omega}_y$.

FIGURE 2

Naive FORTRAN program for $\underset{\sim}{e}$.

```
      DO    10    LL = 1,L
         DO    10    R = 1,M
            DO    10    I = 1,N
               DO    10    J = 1,I
10                E(I,R,LL) = E(I,R,LL) + C(I,J) * Z(J,R,LL)
```

To vectorize this naive program we should make the <u>inner</u> DO loop long; therefore we move the LL loop; moreover we should store the columns of the array columnwise; see SARA (1984, pp. 15, 20-21, 33). These two guidelines yield Figure 3. (Note that the inner loop forms a so-called "linked triad"; hence it can be vectorized; see SARA, 1984, pp. 18-19.)

FIGURE 3

Vectorized FORTRAN program for $\underset{\sim}{e}$.

```
         DO    20    I = 1,N
            DO    20    J = 1,I
               DO    20    R = 1,M
                  DO    20    LL = 1,L
20                  E(LL,R,I) = E(LL,R,I) + C(I,J) * Z(LL,R,J)
```

We point out that m and n vary with the Monte Carlo experiments. So an experiment may use only part of the pseudorandom numbers stored in the "box" E(LL,R,I). Implementing Figure 3 not only saves computer time, but it also runs experiments with common seeds.

Note that we generate M*L (instead of L) elements in parallel, if we replace two loops - namely the loops for R and LL - in Figure 3 by a single loop - namely LR = 1,..., M*L - which yields the two-dimensional array E(LR,I). Then, however we have to rearrange this array into the three-dimensional array E(LL,R,I) because the latter array is needed for the computation of statistics such as $\hat{\underset{\sim}{\Omega}}_y$, as we see now.

Step 4: Compute statistics $\underset{\sim}{\Omega}_y$, $\hat{\underset{\sim}{\beta}}$ *and* $\hat{\tilde{\underset{\sim}{\beta}}}$

Once we have the three-dimensional array $\underset{\sim}{E}$, we can easily compute estimates such as $\hat{\underset{\sim}{\Omega}}_y$ defined in (2.5). This equation can also be computed as

$$\hat{\underset{\sim}{\Omega}}_y = \underset{\sim}{e}\,\underset{\sim}{e}'/(m-1) - \bar{\underset{\sim}{e}}\,\bar{\underset{\sim}{e}}'m, \tag{3.4}$$

FIGURE 4

Vectorizable FORTRAN program for $\bar{\underset{\sim}{e}}$.

```
      DENOM = 1.0/m
      DO    10    I = 1,N
         DO    10    R = 1,M
            DO    10    LL = 1,L
10             EBAR(LL,I) = EBAR(LL,I) + E(LL,R,I)
                  DO    20    I = 1,N
                     DO    20    LL = 1,L
20                      EBAR(LL,I) = EBAR(LL,I) * DNOM
```

where $\bar{\underset{\sim}{e}} = (\bar{e}_1,\ldots,\bar{e}_i,\ldots,\bar{e}_n)'$ with $\bar{e}_i = \sum_{r=1}^{m} e_{ir}/m$. Figure 4 shows the vectorizable FORTRAN program for the computation of $\bar{\underset{\sim}{e}}$. This program can be compiled and vectorized automatically. Alternatively we can use special FORTRAN 200 instructions such as Q8SSUM that computes sums like Σe_{ir}. The computation of $\hat{\underset{\sim}{\Omega}}_y$ in (3.4) can be programmed analogous to Figure 4. Alternatively we can program innerproducts (e'e and $\bar{e}\ \bar{e}'$) through the special function Q8SDOT; see SARA (1984, pp. 22,30).

A problem arises when computing the *inverse* $\hat{\underset{\sim}{\Omega}}_y^{-1}$, which is needed to compute the EGLS estimator $\hat{\tilde{\underset{\sim}{\beta}}}$ in (2.7). The trick in the preceding steps was to make the inner loop long; that is, we made the LL loop the inner loop. The instruction in that loop is executed in parallel, provided that instruction contains *no function or subroutine references* except for basic functions such as sine. So the computer cannot calculate L inverses in parallel, since calculating an inverse requires a subroutine call; see SARA (1984, p. 23).

So $\hat{\underset{\sim}{\Omega}}_y^{-1}$ must be computed in scalar mode. Once this inverse is available, some matrix multiplications follow (such as $\hat{\underset{\sim}{\Omega}}^{-1}\underset{\sim}{X}$), but these matrices are small; hence parallellization is not efficient. To quantify these ideas, we compute the OLS and the EGLS point estimators of (2.6) and (2.7). Into (2.6) we substitute

$$\underset{\sim}{W} = (\underset{\sim}{X}' \underset{\sim}{X})^{-1} \underset{\sim}{X}' \tag{3.5}$$

and into (2.7)

$$\underset{\sim}{V} = (\underset{\sim}{X}' \hat{\underset{\sim}{\Omega}}_y^{-1} \underset{\sim}{X})^{-1} \underset{\sim}{X}' \hat{\underset{\sim}{\Omega}}_y^{-1} . \tag{3.6}$$

$\underset{\sim}{W}$ needs to be computed only once, but V needs to be computed L = 100 times (since $\hat{\underset{\sim}{\Omega}}_y$ changes every time). For the computations we select n = 4, Q = 3, and m = 10. To improve the accuracy of our timing data we repeat the computation 100 times. Appendix 2 gives the computer program. This yields Table 2.

TABLE 2

Total CPU times (in microseconds)
(n=4, Q=3, m=10, L=100)

Computer	Estimator of $\underset{\sim}{\beta}$	
	OLS	EGLS
VAX 8700	920	25,580
CYBER 205		
scalar mode	734	33,206
vector mode	13	27,613

Table 2 shows that computation of inverses (using the NAG routine Fo1AAF) is inefficient on the CYBER 205; this supercomputer is even slower than the VAX 8700! If no subroutine calls interfere with parallelization, then the CYBER 205 is very fast: the OLS estimator $\hat{\underset{\sim}{\beta}}$ requires only 1.4% of the time needed on the VAX 8700. (Appendix 3 gives some more programming tricks for improving the efficiency of supercomputers.)

4. Conclusions

Supercomputers provide a new challenge for management scientists, since their application requires a new way of thinking, namely "thinking in parallel mode". This paper examined supercomputing in Monte Carlo experiments with multivariate regression models. Because the matrix of independent variables $\underset{\sim}{X}$ is relatively small, supercomputers are inefficient if applied straightforwardly. Monte Carlo experiments, however, are replicated many times, say 100 times. Exploiting this dimension of the problem makes supercomputers efficient in some applications, for example, in Ordinary Least Squares. If, however, matrix inversion is needed - as is the case in Estimated Generalized Least Squares - then supercomputers seem slower than scalar computers such as the VAX 8700.

Appendix: FORTRAN 200 program for the pseudorandom number generator

```
      PROGRAM VARIANT4
      IMPLICIT REAL (U-Z), INTEGER (A-T)
      PARAMETER (N1=5,N4=65535,K=1)
      PARAMETER (A3=37772072706109)
      INTEGER MVAST
      BIT BVAST
      DESCRIPTOR MVAST, BVAST
      DIMENSION T(N4), S1(N1)
      DIMENSION X1(N1)
      DATA MINT / X'0000800000000000' /
      CALL RANSET (K)
      DO 5 I=1,N4
       U=RANF()
       CALL RANGET(T(I))
    5 CONTINUE  C ! N=5
C     ! SCALAR
      S1(1;N1)=T(1;N1)
      ZPU1=SECOND()
      DO 10 I=1,N1
       S1(I)=A1*S1(I)
       IF (S1(I).LT.0) S1(I)=S1(I)-MINT
       X1(I)=S1(I)/MINT
   10 CONTINUE
      ZPU2=SECOND()
      U1=ZPU2-ZPU1
C     ! VECTOR
      ASSIGN MVAST,.DYN.N1
      ASSIGN BVAST,.DYN.N1
      S1(1;N1)=T(1;N1)
      ZPU1=SECOND()
      S1(1;N1)=A1*S1(1;N1)
      BVAST=S1(1;N1).LT.0
      MVAST=S1(1;N1)-MINT
      S1(1;N1)=Q8VCTRL(MVATS,BVAST;S1(1;N1))
      X1(1;N1)=S1(1;N1)/MINT
      ZPU2=SECOND()
      Z1=ZPU2-ZPU1
      FREE
      PRINT *. 'BEGIN:  GEVEKTORISEERD  SCALAR
      PRINT *, 'N=    5 '.Z1,'  ',U1
      END
```

Appendix 2: FORTRAN 200 program for the OLS and EGLS estimators

OLS ESTIMATOR FOR BETA

```
      CALL MXM(XT,X,XTX)
      CALL INVERSE(XTX,XTXI)
      CALL MXM(XTXI,XT,W)
       DO 5 I=1,N
        DO 5 J=1,M
         YGEM(1,I;LL)=YGEM(1,I;LL)+Y(1,J,I;LL)
 5    CONTINUE
      DO 10 I=1,R
       DO 10 J=1,N
        BETA(1,I;LL)=BETA(1,I;LL)+W(I,J)*YGEM(1,J;LL)
10    CONTINUE
```

EGLS ESTIMATOR FOR BETA

```
      DO 5 I=1,N
       DO 5 J=1,M
        YGEM(1,I;LL)=YGEM(1,I;LL)+Y(1,J,I;LL)
 5    CONTINUE
      DO 10 I=1,N
       DO 10 J=1,N
        DO 10 K=1,M
         S(1,I,J;LL)=S(1,I,J;LL)+((Y(1,K,I,;LL)-
      YGEM(1,I;LL))*(Y(1,K,J;LL)-YGEM(1,J;LL)))
10    CONTINUE
```

```
      DO 15 I=1,
       DO 15 J=1,N
        S(1,J,I;LL)=S(1,I,J;LL)
15    CONTINUE
      DO 18 K=1,LL
       DO 20 I=1,N
        DO 20 J=1,N
         DU4(J,1)=S(K,J,I)
20     CONTINUE
       CALL INVERSE(DU4,MY4,N)
       DO 25 I=1,N
        DO 25 J=1,N
         SI(K,J,I)=MY4(J,I)
25     CONTINUE
18    CONTINUE
      DO 30 I=1,R
       DO 30 J=1,N
        DO 30 K=1,N
         XTSI(1,I,J;LL)=XTSI(1,I,J;LL)+XT(I,K*SI(1,K,J;LL)
30    CONTINUE
      DO 35 I=1,R
       DO 35 J=1,R
        DO 35 K=1,N
       XTSIX(1,I,J;LL)=XTSIX(1,I,J;LL)+XTSI(1,I,K;LL)*X(K,J)
35    CONTINUE
       DO 40 K=1,LL
        DO 45 I=1,R
         DO 45 J=1,R
          DU3(J,I)=XTSIX(K,J,I)
45      CONTINUE
        CALL INVERSE(DU3,MY3,R)
        DO 50 I=1,R
        DO 50 J=1,R
         XTSIXI(K,J,I)=MY3(J,I)
50     CONTINUE
40    CONTINUE
```

```
      DO 55 I=1,R
       DO 55 J=1,N
        DO 55 K=1,R
       V(1,I,J;LL)=V(1,I,J;LL)+XTSIXI(1,I,K;LL)*XTSI(1,K,J;LL)
55    CONTINUE
      DO 60 I=1,R
       DO 60 K=1,N
        BETA(1,I;LL)=BETA(1,I;LL)+V(1,I,K;LL)*YGEM(1,K;LL)
60    CONTINUE
```

Appendix 3: Programming tricks

There are several "tricks" for improving the efficiency of supercomputers. These tricks should be applied in any computer program, not only Monte Carlo experiments:

1. Scalar divides take relatively much time (54 cycles versus 5 cycles for multiplication; 1 cycle takes 20 nanoseconds); the computation of denominators like 1/m (see Figure 4) and 1/(m-1) (see eq. 3.4) should therefore be separated by several lines of code; SARA (1984, pp. 5,7).
2. Double precision is slow and excludes vector mode; SARA (1984, p. 6).
3. There are special vectorized instructions so-called V-functions and Q8-functions. We saw some examples above; also see SARA (1984, pp. 27,30).
4. The compiler can optimize the standard FORTRAN program; next special programs (like SPY and CIA) can measure which parts of the program take most time during execution and are candidates for customized optimization.

Acknowledgement

The first author was sponsored by the Supercomputer Visiting Scientist Program at Rutgers University, The State University of New Jersey, during July 1988. In 1989 computer time on the CYBER 205 in Amsterdam was made available by SURF/NFS.

References

CDC, *FORTRAN 200 Version 1 Reference Manual*, Publicatio no. 60480200, Control Data Corporation, Sunyvale, California 94088-3492, December 1986.

Devroye, L., *Non-Uniform Random Variate Generation*, Springer-Verlag, New York, 1986.

Kleijnen, J.P.C., *Statistical Tools for Simulation Practitioners*, Marcel Dekker, Inc., New York, 1987.

Kleijnen, J.P.C., Analyzing Simulation Experiments with Commmon Random Numbers, *Management Science, 34, 1(1988), 65-74.*

Kleijnen, J.P.C., Pseudorandom number generation on supercomputers, *Supercomputer (1989) (accepted for publication).*

Kleijnen, J.P.C. and B. Annink, *Pseudorandom number generators revisited*, Katholieke Universiteit Brabant (Catholic University Brabant), May 1989.

Levine, R.D., Supercomputers, *Scientific American*, January 1982, 112-125.

Naylor, T.H., J.L. Balintfy, D.S. Burdick and K. Chu, *Computer Simulation Techniques*. John Wiley & Sons, New York, 1966.

Petersen, W.P., Some vectorized random number generators for uniform, normal, and Poisson distributions for CRAY X-MP. *The Journal of Supercomputing*, 1. (1988), 327-335.

Rao, C.R., Some problems involving linear hypotheses in multivariate analysis, Biometrika, 46 (1959), 49-58.

SARA, *Cyber 205 user's guide; part 3, Optimization of FORTRAN programs.* SARA (Stichting Academisch Rekencentrum Amsterdam/ Foundation Academic Computer Centre Amsterdam), Amsterdam, Nov. 1984.

Teichroew, D., A history of distribution sampling prior to the era of the computer and its relevance to simulation. *Journal American Statistical Association*, March 1965, 27-49.