Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

SERVO: SIMULATION EXPERIMENTS WITH RANDOM-VECTOR OUTPUT

Bruce W. Schmeiser

School of Industrial Engineering
Purdue University
West Lafayette, Indiana 47907-1287

Mark D. Scott

American Airlines Decision Technologies
P.O. Box 619616 / MD3345
DFW Airport, Texas 75261-9616

ABSTRACT

We discuss statistical methods, system design, and examples for SERVO, a software package that supports simulation experiments having output data that are assumed to be a sequence of identically distributed random vectors. SERVO requires two types of input: a description of the model of interest and a description of the desired experiment. From this input, SERVO generates C-language code that when executed is the simulation experiment.

The experiment automatically creates an output report with point estimators, and associated standard errors, for a variety of typical performance measures. The marginal performance measures include means, standard deviations, variances, mean squared errors, third and fourth standardized moments, and points on the distribution function; the joint performance measures are covariances and correlations.

1 INTRODUCTION

Stochastic simulation experimentation is commonly used to estimate system performance measures. Given a description of component behavior and the logical interaction among components, random realizations of system behavior are generated, data are collected, and point estimators of the performance measures are calculated. In this view, performance measures are properties of the joint distribution of the data, including marginal distribution properties such as moments and quantiles and dependence properties such as correlations.

Most industrial-engineering simulation software has focused on aiding the analyst to model complex systems such as arise in manufacturing, health, and transportation systems. Data collection and other statistical issues have been viewed as appendages. This orientation has occurred because modeling effort often dominates such simulation analyses. Moreover, the simulation data are often a set of time series, correlated but with observations occurring at random times, which results in statistical-analysis problems that are difficult to accommodate in general-purpose software.

We consider random-vector simulation in which the simulation model generates multivariate output. We assume the output observations are identically — but not necessarily independently — distributed. Each observation is a random vector, each component of which represents a random aspect of the modeled system for one replication or one time interval.

Although the simulation model changes from application to application, any code for random-vector simulation experiments includes these fundamental components: reading experiment parameters, echoing experiment parameters, looping for multiple replications and for multiple design points, obtaining observations of simulation-model performance, updating accumulators for estimators and standard errors, computing estimators and standard errors, and writing output.

In this paper we discuss SERVO (see Appendix), which is designed to simplify creating these fundamental components of random-vector simulation experiments. SERVO, developed in Scott (1990), generates C-language code to perform a simulation experiment. The code sees the simulation-model as a subroutine whose only purpose is to produce one random-vector output observation, $\underline{y} \stackrel{\text{def}}{=} (y_0, y_1, \ldots, y_{r-1})$. (The simulation-model subroutine could simply read the next $\underline{y}$ from a data file; see Example 4.3.) After a number of calls to the simulation-model subroutine, the code computes performance-measure estimates $\underline{\hat{\theta}} \stackrel{\text{def}}{=} (\hat{\theta}^0, \hat{\theta}^1, \ldots, \hat{\theta}^{p-1})$ and estimates of their standard errors for the system defined by the simulation model. SERVO reduces the responsibilities of the user to supplying the model subroutine and the simulation experimental parameters.

In Section 2 we discuss design issues for random-vector simulation software, including a preliminary example, that describe the input and output of SERVO. In Section 3 we discuss SERVO implementation issues — how the input is processed to obtain the output. Section 4 contains a continuation of the example in Section 2 and two additional examples.

2 DESIGN ISSUES

2.1 A Random-Vector Simulation Experiment

As an example of a random-vector simulation experiment, consider a circuit designer who has four network components A, B, C, and D with two possible configurations, C1 and C2, as shown in Figure 1. The reliability of a configuration is the probability that all of the components on at least one of the paths from S to T work. The designer knows the individual component reliabilities, but is interested in the reliabilities of configurations C1 and C2, θ^0 and θ^1. The difference of the configurations' reliabilities, θ^2, is also of interest.

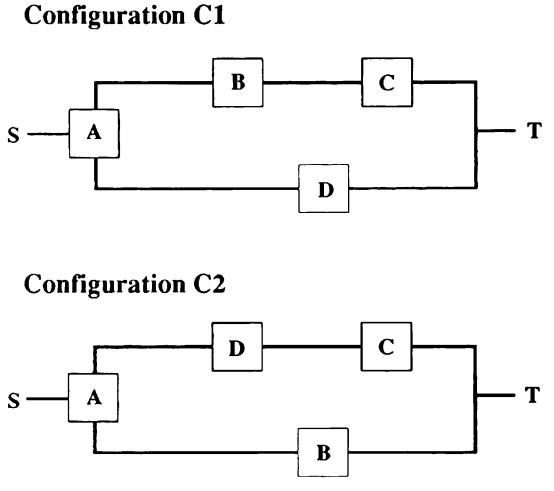

Figure 1: Reliability Networks

Here, $r = 3$ and the random vector is (Y_0, Y_1, Y_2). The random variable of interest for C1 is an indicator variable, say $Y_0 = 0$ if the system fails and $Y_0 = 1$ if the system works. Define Y_2 similarly for configuration C2. Finally, define $Y_2 = Y_0 - Y_1$, the difference of the configurations' reliabilities, which can take the values -1, 0, or 1. Observations of the random vector (y_0, y_1, y_2) are computed from realizations of the individual components, based on the given component reliabilities. The three estimators $\hat{\theta}^0$, $\hat{\theta}^1$, and $\hat{\theta}^2$ are the sample averages of Y_0, Y_1, and Y_2.

In the rest of this section, we discuss the nested-loop structure used to estimate standard errors, the design of SERVO, and a brief comparison to some related software.

2.2 Nested-Loop Structure

A primary consideration in designing simulation software is the ability to report to the user the sampling error of the point estimate $\underline{\hat{\theta}}$. A straightforward approach is to use a nested-loop structure, as discussed in Schmeiser (1990). The inner loop contains the simulation model; we refer to each of the m inner-loop executions as a microreplication. The outer loop computes point estimates $\hat{\theta}_i^l$ using the m microreplication observations; we refer to each of the k outer-loop executions as a macroreplication. The point estimate $\hat{\bar{\theta}}^l$ and its associated standard error (s.e.) are

$$\hat{\bar{\theta}}^l = \frac{\sum_{i=0}^{k-1} \hat{\theta}_i^l}{k} \quad \text{and} \quad \text{s.e.}\left[\hat{\bar{\theta}}^l\right] = \left[\frac{\sum_{i=0}^{k-1} (\hat{\theta}_i^l)^2 - k\left[\hat{\bar{\theta}}^l\right]^2}{k(k-1)}\right]^{1/2},$$

where $l = 0, 1, ..., p-1$.

Therefore, each performance-measure estimate is a function of $n \stackrel{\text{def}}{=} km$ observations. When $\hat{\theta}^l$ is not a mean, the best number of macroreplications for the point estimate $\hat{\bar{\theta}}^l$ is $k = 1$, since bias is then minimized. However, the estimate of standard error is defined only for $k \geq 2$. If k is too large (m is too small) both estimates are biased; if k is too small, the standard-error estimate has a large variance, making the standard-error estimate unreliable as a measure of precision. Typically k is chosen in the range 10 to 30. (See Schmeiser 1982).

A typical nested-loop structure with a model subroutine that returns a single random-vector observation is shown in Figure 2. Figure 2 depicts a simulation experiment in which three performance measures are being estimated for $r = 1$ random variable, $\underline{y} = (y_0)$. The accumulator variables, such as `mean`, `mean2`, and `std`, are messy and bulky even for such a small simulation. As one tries to estimate more performance measures for multiple random variables, the simulation becomes cluttered with accumulator variables that are often similar in name and function.

The nested-loop structure, peripheral accumulators, and performance-measure calculations are common in all random-vector simulation experiments. Having software to automatically handle these common components and requiring the user to define only the simulation model and experimental parameters is appealing.

```
c
c     estimate the mean, standard deviation,
c    and kurtosis for one random variable.
c    all variables are real.
c
      mean  = 0.0
      mean2 = 0.0
      std   = 0.0
      std2  = 0.0
      kurt  = 0.0
      kurt2 = 0.0
c
```

```
c    ...the macroreplication loop
c
     do 200 i = 0,k-1
        sum  = 0.0
        sum2 = 0.0
        sum3 = 0.0
        sum4 = 0.0
c
c       ...the microreplication loop.
c
        do 100 j = 0,m-1
           call model(y)
           sum  = sum  + y0
           sum2 = sum2 + y0**2
           sum3 = sum3 + y0**3
           sum4 = sum4 + y0**4
100     continue
        xmean = sum / m
        mean  = mean  + xmean
        mean2 = mean2 + (xmean*xmean)
        xstd  = f1(sum2,xmean,m)
        std   = std   + xstd
        std2  = std2  + (xstd*xstd)
        xkurt = f2(sum4,sum3,sum2,xmean,m)
        kurt  = kurt  + xkurt
        kurt2 = kurt2 + (xkurt*xkurt)
200  continue
c
c    ... compute point estimates
c
     mean_hat = mean / k
     std_hat  = std  / k
     kurt_hat = kurt / k
c
c    ...compute standard-error estimates
c
     se_hat(mean_hat) = f3(mean2,mean_hat,k)
     se_hat(std_hat)  = f3(std2,std_hat,k)
     se_hat(kurt_hat) = f3(kurt2,kurt_hat,k)
```

Figure 2: An Example of a Nested-Loop Structure

2.3 Design of SERVO

SERVO automates the development of random-vector simulation experiments. The nested-loop structure and the peripheral accumulators are automatically built around the user-provided simulation-model subroutine and the performance-measure estimates are automatically calculated. In this subsection we discuss several SERVO design decisions: code generation, multiple design points, choice of performance measures, source of randomness, and output reports and files.

SERVO generates a set of code that includes a main program containing the common nested-loop structure analogous to that of Figure 2. An alternative would be to create template software that a user would modify. The advantage of having SERVO generate code is that the user needs to consider code only with respect to the user's simulation model. The disadvantage is that the several SERVO system files and user simulation-model file must be compiled and linked.

Sometimes the user needs to run an experiment at multiple design points. For example, the circuit designer may wish to estimate the reliabilities for various component reliabilities or configurations. Each run estimates the performance measures at a particular design point. SERVO performs runs at different design points by surrounding the micro and macroreplication loops with a third loop. We refer to this structure as a triply nested loop. We then have only to pass the design points, defined by the model parameters, to the simulation-model subroutine.

SERVO estimates several properties of the joint distribution of Y. For each of the r marginal distributions, the mean and standard deviation are always estimated, as are the $\binom{r}{2}$ covariance and correlation between each marginal component of Y. At the user's option, variance, skewness, and kurtosis are estimated. Also optionally, the mean-squared error $E(Y_j-\mu_j)^2$ is estimated, but this requires the user to specify the base-line values μ_j. The estimators used are standard; see, for example, Lewis and Orav (1989).

The simulation model needs random variates. Either U(0,1) random number streams or random variates could have been passed to the simulation-model subroutine. Random-number streams were chosen for simplicity; passing random variates presented possible user protocol problems. Also, having random-variate generators associated with SERVO would limit its portability. Therefore we rely on the user having access to a library of random-variate generators, such as IMSL (1987) or Tao (1990), who provides a facility to generate data sets from frequently used analytically solvable stochastic models. Random number seeds (as well as simulation-model parameters) are passed from the main program to the simulation model as subroutine parameters.

SERVO can save the raw-output data to files. Having access to each observation allows the user to perform post-processing on the data, such as performing statistical tests or generating graphs from the raw output.

The standard SERVO output report consists of the names, point estimates, and standard errors for each of the performance measures. Examples are given in Section 4.

2.4 Other Simulation Software

We briefly discuss two existing software packages that have various features shared by SERVO. Both have a more-specific context than SERVO; they estimate properties of statistical estimators.

The Monte Carlo System (Grier 1987) studies properties of estimators in designed experiments. The result of an experiment using Grier's package is an estimate of the distribution of the statistic of interest over various combinations of factor values. Features that are analogous to those of SERVO include: (1) the only sections of

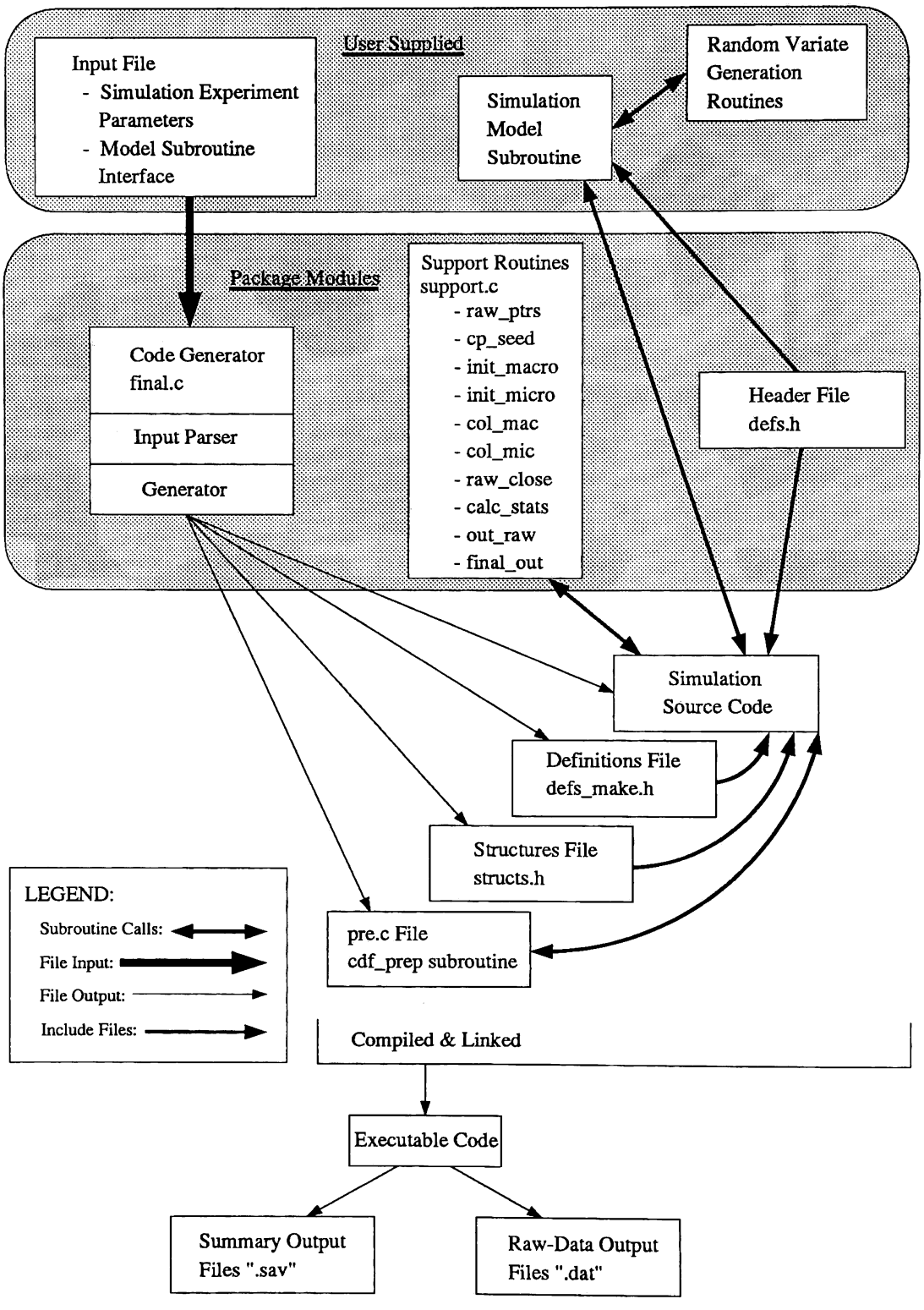

Figure 3: SERVO System-Integration Diagram

code that are application dependent are those that generate the random data set and that evaluate the statistic of study, (2) the data are collected in a doubly nested loop, and (3) multiple design points are considered. Differences are that Grier's package has predefined factors (points include sample size, degrees of freedom, and type of experimental design), considers only scalar ($r = 1$) data, and computes no standard error estimates.

SMTBPC is the simulation test bed program provided as a software supplement to Lewis and Orav (1989). Like The Monte Carlo System, SMTBPC is designed to estimate the distribution of a single random variable; like SERVO its emphasis is on the first four moments. SMTBPC design points differ only by the size of the simulated univariate data sample. In addition, SMTBPC provides estimates of the coefficients of the asymptotic bias expansion for the estimator of the mean as a function of sample size.

The Monte Carlo System and SMTBPC provide side-by-side boxplots that allow concise, graphical comparison of the estimated distributions at different design points. SERVO provides no graphical comparisons; one could create a variety of graphical analyses by passing the raw-data file to a general-purpose statistical package.

3 IMPLEMENTATION

In this section we discuss high-level programming considerations, issues of numerical error, multiple runs, the input file, and the output file.

3.1 Two Modules of C Code

Both SERVO and the generated simulation source code are written in C (see e.g., Darnell and Margolis 1988). To maintain as much portability over computers and compilers as possible, SERVO follows ANSI standards and uses only common string manipulation and math routines.

Two designs for SERVO were considered. Common to both approaches was the need for a model subroutine and some control statements. The control statements are needed to inform the package of important characteristics of the model subroutine and the desired parameters of the simulation experiment, such as the number of macroreplications and microreplications.

The first approach considered was to create a single module of code that would accept, as input, the model subroutine name and control statements. Then, while still in execution, the module would dynamically link the model subroutine, perform the simulation experiment, and output the results. This approach was appealing because it would perform the entire simulation in one step. However, a major problem with this approach was the difficulty and lack of portability in having the single module link itself to the model subroutine, and then perform the simulation.

The approach we chose requires two stages. Within stage one, the package accepts the model-subroutine's name and control statements as input and then generates source code that is capable of performing the simulation experiment. The second stage consists of compiling and executing the source code from stage one. In addition to the module that generates the simulation code in stage one, a set of support routines is needed for linking with the generated simulation code. This second approach is appealing because after the first stage, code tailored to the user's specifications has been generated. This code can then be inspected or modified by the user.

SERVO consists of two code modules. The module `final.c` parses the input file and generates the simulation source code, and the module `support.c` contains support routines to be compiled with the simulation source code. Figure 3 provides a detailed interaction picture of SERVO. The figure is more detailed than the following discussion.

The generated simulation source code calls many support subroutines to collect data, calculate estimates, and produce output files. The module `support.c`, which contains all of these support routines, is `#included` in the generated simulation source code. An example of the core section of the generated source code is shown in Figure 4.

```
/** BEGIN THE MAIN PART OF PROGRAM **/
cp_seed(iseed,seeds,&data);
for(i=0;i<NUM_RUNS;i++)
{
  data.run = i;
  raw_ptrs(&data);
  cdf_prep(arr,&data,seeds);
  init_macro(arr,&data);
  for(j=0;j<NUM_MACRO;j++)
  {
    init_micro(arr,&data);
    data.mac_rep = j;
    for(k=0;k<NUM_MICRO;k++)
    {
      data.mic_rep = k;
      sub_model(data.params[i],seeds,
                        observations);
      out_raw(arr,&data,observations);
      col_mic(arr,&data,observations);
    }
    col_mac(arr,&data);
  }
  raw_close(&data);
  calc_stats(arr,&data);
  final_out(arr,&data,iseed,seeds);
  cp_seed(seeds,iseed,&data);
}
```

Figure 4: Core of Generated Simulation Source Code written to `SIMULATION FILE`

The simulation source code contains the doubly or triply nested-loop structure and is never larger than 22 lines. The concise code, which allows the user to see the simulation experiment structure, is feasible because all of the collection and analysis of the simulation output data are performed in support subroutines.

3.2 Numerical Error

To reduce numerical error, double-precision float variables were used for all real variables. Also, the first-value-offset (FVO) method was implemented. The FVO method removes the order of magnitude from the estimation of the performance measure by approximately centering the distribution around zero, where computer arithmetic can be performed with greatly diminished numerical error. We record the first observation sampled from a distribution and call it the *offset*. We subtract the offset from every observation including the first, which shifts the distribution by the constant offset. The only moment affected by this shift is the mean. At the end of the analysis, the sample mean is corrected by adding the constant offset.

3.3 Multiple Runs

The user may decide that the experiment needs to be run at various design points. This multiple-run experiment then necessitates a triply-nested loop. Since the run is performed in the inner two loops, the outer-loop of the multiple-run experiment is responsible for keeping track of the correct parameters, or design points.

The model subroutine receives seeds from the main routine for use in random-variate generation routines, and returns a vector of observations. Also, the user has an option to pass a vector of parameters to the model subroutine. The parameters may be used for any purpose, such as defining design points, within the subroutine. Thus, for each run, the user may pass a vector of parameters to the model subroutine. However, within any given run of a multiple-run experiment, the same vector of parameters is passed to the subroutine.

3.4 Input File

Figure 5 is a template of the user input. The purpose of the input file is to specify user's desired random-vector simulation experiment to SERVO. Two sets of information are encoded in the input file. First is information pertaining to the interface between the user's model subroutine and the main routine yet to be generated. The second set of information defines the simulation experiment itself. Although there are two distinct sets of information, the set of input commands cannot be separated since some commands pertain to both sets of information.

There are two delimiters in the input file: commas and semicolons. Commas separate each of the input fields and semicolons end each input line. Any characters following a semicolon are treated as a comment.

The eleven input commands are defined below.

1. `*NAME` identifies the user. Its only field, `name`, is for the alphanumeric user's name.

2. `*DATE` identifies the date. Its only field, `date`, is used for the alphanumeric current date.

3. `*MODEL` has two input fields. The first field, `model_name`, is the alphanumeric simulation-model subroutine name. There can be no spaces in the name. The second field, `fortran_or_c`, identifies whether `model_name` is written in C or FORTRAN.

4. `*RANDOM VARIABLES` specifies an input structure. In the command line, `*RANDOM VARIABLES`, there are two numeric fields: the number of random variables, r, and number of raw output files, f. If the second field is blank or negative, no raw output files are produced.

   Each of the r lines below the `*RANDOM VARIABLES` input command describes one random variable. Each random variable line contains three fields; the random-variable name, its true value if available, and raw-output file numbers corresponding to raw-output files in which the user would like to save the raw data.

5. `*RUNS` is also an input structure. The `*RUNS` command line has three fields. The first is the number of runs w. (Recall that a run is the simulation of a design point, which requires an entire doubly-nested loop simulation experiment.) If w is greater than one, the user can change design points between runs. The second field informs the package how many design-point parameters p are passed to the model subroutine. If left blank, the number of parameters is set to zero. The third field allows the user to choose whether or not common random numbers are used between multiple runs. The w lines after the command line `*RUNS` each corresponds to one run and should contain p parameters, each separated by a comma.

6. `*SIMULATION FILE` has one field, which is the alphanumeric name of the file into which the generated simulation source code is to be written.

7. `*MICRO` has one field, which is the number of microreplications.

8. `*MACRO` has one field, which is the number of macroreplications.

9. `*SEEDS` has $h+1$ integer fields, where h is the number of seeds. The first field specifies h, and the remaining fields are for seed 0 through seed h-1.

```
;   Input File Template
*NAME,name;
*DATE,date;
*MODEL,model_name,fortran_or_c;
*RANDOM VARIABLES,num of random variables,num raw output files;
              rv_name_0,true value,file1,file2,...;
              rv_name_1,true value,file1,file2,...;
                    ...
              rv_name_r-1,true value,file1,file2,...;
*RUNS,num of runs,num of parameters,common random numbers(y/n);
              param0 of run0,param1 of run0,...,param p-1 of run0;
              param0 of run1,param1 of run1,...,param p-1 of run1;
                    ...
              param0 of run w-1,param1 of run w-1,...,param p-1 of run w-1;
*SIMULATION FILE,name of file in which to write simulation source code;
*MICRO,number of microreplications;
*MACRO,number of macroreplications;
*SEEDS,number of seeds, seed 0, seed 1, ..., seed h-1;
*COLLECT,all,variance,mse,skewness,kurtosis;
*CUMULATIVE DISTRIBUTION FN,num of variates,num of cutoff points;
```

Figure 5: Input File Template

random variables:

p_hat_1:	Estimate	Std. Error
Mean	.44245	(0.003797)
Std. Dev.	.496645	(0.0004591)
Variance	.246661	(0.0004553)
Skewness	.231791	(0.0155128)
Kurtosis	1.05636	(0.007529)
Cov(*,p_hat_2)	.191352	(0.001175)
Corr(*,p_hat_2)	.79676	(0.003925)
Cov(*,diff)	0.0553088	(0.001112)
Corr(*,diff)	.35583	(0.004036)

Figure 7: Sample from *rel_sim.sav0*

10. **COLLECT* has up to five fields. SERVO estimates means, standard deviations, covariances, and correlations automatically. However, *variance*, *mse*, *skewness*, and *kurtosis* are all optional. If *all* is specified, all four of the aforementioned performance measures are estimated for all of the random variables.

11. **CUMULATIVE DISTRIBUTION FN* has two numeric input fields. To estimate the marginal cdfs of all of the random variables, SERVO needs reasonable estimates of the minimum and maximum of each of the random-variable distributions. The first field is the number of random variates, c, to be sampled to estimate the minimum and maximum values for each of the random-variable distributions. The estimates of the maximum and minimum, $\hat{\max}$ and $\hat{\min}$, are the maximum and minimum of the c variates, respectively. The minimum number of variates is 100.

The second field is t, the desired number of cutoff points at which the cdf value is estimated. The cutoff points d_s, which define the cumulative probabilities D_s, are

$$d_s = \hat{\min} + s\left[\frac{\hat{\max} - \hat{\min}}{t-1}\right] \quad \text{for } s = 0,1,...,t-1$$

**CUMULATIVE DISTRIBUTION FN* is the only optional input command. If it is present, the cdf will be estimated.

3.5 Simulation-Model Subroutine

This section discusses the interface between the SERVO-generated code and the C or Fortran user-written simulation-model subroutine.

There are three sets of information that must be passed between the two modules of code: parameters, seeds, and observations. Both the parameters and observations are placed in vectors that are of type double precision real or float, while the seeds are passed in a vector of long integers. These vectors are passed by pointer, so changes made to these vectors in the model subroutine are reflected in the main routine. Of course, this is the desired effect for the seeds and observations vectors. We want to collect the observations after each call to the model subroutine, and the random-variate generation routines need different seeds each time they are called.

Parameters are numbers that are passed from the main routine to the model subroutine, usually for defining design points during multiple-runs. For example, if in a bank-teller simulation multiple-runs of the entire doubly nested loop experiment are desired with different teller service-time distributions, the user specifies the teller service-time distributions as parameters. In many applications, the user wishes to vary more than one parameter over the multiple runs, so the generated main routine passes a vector of parameters to the model subroutine. Passing the vector of parameters is optional and is specified in the **RUNS* input command. If no parameters are specified in the input field *number of parameters*, the interface between the subroutine and the main routine contains no parameters vector.

The seeds vector, which is not optional, has length and initial state defined in the **SEEDS* command. The random-seed streams, each of which is defined as an element of the seeds vector, can be used for random-variate generation routines within the model subroutine.

The observations vector is used to pass observations $\underline{y}$ from the model subroutine to the main routine for collection and analysis. The length of this vector is r, as defined in the **RANDOM VARIABLES* input command by the field. Each time the model subroutine is engaged, SERVO assumes that each element of the observation vector is assigned a value corresponding to the correct observation.

To summarize the interface between the main routine and the simulation-model subroutine (*SMS*), there are either two or three vectors comprising the argument list of the subroutine. The optional parameters vector is first in the subroutine argument list, the mandatory seeds vector is second, and the mandatory observations vector is last. If no parameters are specified in the **RUNS* input command, the parameters vector does not appear in the argument list. An example call to *SMS* is
SMS(params,seeds,observations).

3.6 Output Files

SERVO produces two types of output files: summary files and (optionally) output-data files. The names of both types of files begin with the character string specified in *SIMULATION FILE; for example, SF. Each filename suffix indicates the type of file and relates it to the run number, $i = 0,1,...,w-1$. For example, the name of the summary report from run (design point) 6 is SF.sav5. Similarly, the fourth output-data file for run 6 is named SF.dat5_3.

4 EXAMPLES

4.1 Circuit Reliability

Recall the example in Section 2.1. Figure 6 shows the input file used to create the simulation source code. The name of the model subroutine is *reliability* and it is written in FORTRAN. We are interested in three random variables, *p_hat_1*, *p_hat_2*, and *diff*. The random variables *p_hat_1* and *p_hat_2* represent the reliability of configurations C1 and C2, respectively, whereas *diff* represents the difference in the two configurations' reliabilities. We have chosen to

save the raw data in two files. In the first file, raw-data file 0, we have only asked to save `p_hat_1` and `p_hat_2`, whereas we have asked to save all three random variables' raw data into raw-data file 1. The generated source code is written to the file `rel_sim`. The simulation experiment consists of one run that performs 1000 microreplications and 20 macroreplications. Three seeds are defined in the `*SEEDS` input command. Finally, we wish to collect all of the optional performance measures.

```
;       Circuit Example
*NAME,Mark;
*DATE,11/30/90;
*MODEL,reliability,fortran;
*RANDOM VARIABLES,3,2;
        p_hat_1,,0,1;
        p_hat_2,,0,1;
        diff,,1;
*RUNS,1;
*SIMULATION FILE,rel_sim;
*MICRO,1000;
*MACRO,20;
*SEEDS,3,12345678,98765432,55555555;
*COLLECT,all;
```

Figure 6: Input File for the Circuit Example

A partial summary output file is shown in Figure 7; only the summary table for random variable `p_hat_1` is shown. Individual component reliabilities corresponding to this example are below.

$$P[A \text{ works}] = 0.9$$
$$P[B \text{ works}] = 0.3$$
$$P[C \text{ works}] = 0.8$$
$$P[D \text{ works} \mid C \text{ worked}] = 0.2$$
$$P[D \text{ works} \mid C \text{ did not work}] = 0.7$$

The deterministically calculated reliability for configuration 1 is 0.4428, which is within one standard error of the `p_hat_1` from Figure 7.

4.2 Correlated Data

Suppose that a user of our system has two columns of data in a file named `data`. The first column represents total time to process a batch of parts, and the second column represents the corresponding percent rework for that batch. Since the batch size remains constant, it would seem that there should be non-zero correlation between the two sets of data.

A quick way to estimate the marginal distributions of process time and percent rework and to estimate their correlation is to read these data into SERVO, one observation at a time. The model subroutine's observations vector contains two elements. The first element is a process time, and the second is its corresponding percent rework. The generated simulation source code passes one seed, as an indicator, to the model subroutine. Each time the model subroutine is called, it checks whether the seed is equal to one. If it is, it opens the file `data`. Then it increments the seed and reads one observation of process time and one observation of percent rework out of the data file. No subsequent call to the subroutine will attempt to open the file `data`, but every call to the model subroutine will read an observation.

```
;       Correlated Data Example
*NAME,Mark;
*DATE,11/30/90;
*MODEL,scan,c;
*RANDOM VARIABLES,2;
        proc_time;
        %_rework;
*RUNS,1;
*SIMULATION FILE,data;
*MICRO,1500;
*MACRO,20;
*SEEDS,1,1;
*COLLECT,skewness,kurtosis;
```

Figure 8: Input File for Correlated-Data Example

Figure 8 shows the input file for this example. We ask for 1500 microreplications and 20 macroreplications, since there are 30,000 pairs of data in the file `data`.

4.3 Time Slicing

SERVO can be used to analyze data generated by other simulation software. For example, one of us (MS) has helped Pritsker Corporation adapt SERVO to analyze data generated with SLAM II (Pritsker 1986). The simulation-produced output data, for example various performance measures for each eight-hour shift (time slice), are written to a file. The model subroutine reads one observation y from the file each time it is called by SERVO. The SERVO main routine then calculates the desired point estimates and standard errors as if the model routine were the simulation model.

Unlike the first two examples, the random vectors in time slicing can be dependent through time. For example, if the average length of a queue in time period n is long, the chances are high that the length of that queue in time period $n+1$ is also long. Despite this dependency, the micro/macroreplication structure produces reasonable standard-error estimates if the number of microreplications, m, is substantially bigger than the number of

observation lags required for dependency to become negligible and the number of macroreplications, k, is not too small, as discussed earlier.

ACKNOWLEDGEMENTS

This research was supported by NSF Grant DMS-8717799 to Purdue University. We thank Barry Nelson, Alan Pritsker, Ronald Rardin, and James Reed Wilson for helpful discussions, and H. Philip Walton for the use of his C-language expertise.

APPENDIX: Naming the Software Package

Every software package needs a name. Ideally the name is a meaningful acronym as well as an image-evoking word. In addition to being good examples, CACI's SIMSCRIPT, Pritsker Corporation's SLAM, Schruben's SIGMA, and System Modeling's SIMAN indicate that such a name should also begin with an "S". Both authors agreed that "S" is a fine first letter for a name.

Being in an engineering school, we evoke the engineer image and name this software package SERVO. *Simulation Experiment with Random-Vector Output* is a descriptive phrase capturing the essence of our context.

Having already satisfied sufficient criteria for a reasonable name, we go out on a limb over thin ice to argue that this software package *is* a SERVO, which is short for *servomechanism*, whose entry on our dictionary is

> ser.vo.mech.a.nism: an automatic device for controlling large amounts of power by means of very small amounts of power and automatically correcting performance of a mechanism.

Automatic device seems reasonable, since our purpose is to automatically generate the simulation-experiment code. *Large amounts of power* is the information obtained by the simulation experiment. *Very small amounts of power* is the information provided by the user. *Automatically correcting performance* stretches us yet further, but why stop now? A fundamental design consideration has been that we wanted to estimate standard error for every point estimator. The purpose of the estimated standard error is to provide feedback on estimator precision to the analyst. The analyst is then part of the *control* loop that guarantees sufficient point-estimator precision.

Ok, a twig over open water.

REFERENCES

Darnell, P.A. and P.E. Margolis. 1988. *Software engineering in C.* New York: Springer-Verlag.

Grier, D.A. 1987. Systems for Monte Carlo work. In *Proceedings of the Winter Simulation Conference*, eds. A. Thesen, H. Grant, W. D. Kelton, 428-433.

IMSL STAT/LIBRARY. 1987. *User's Manual.* Version 1.0, Houston: IMSL Inc.

Lewis, P.A.W. and E.J. Orav. 1989. *Simulation methodology for statisticians, operations analysts, and engineers: Vol. 1.* Pacific Grove, California: Wadsworth & Brooks/Cole.

Pritsker, A.A.B. 1986. *Introduction to simulation and SLAM II.* Third Edition. New York: Halsted Press.

Schmeiser, B. 1982. Batch size effects in the analysis of simulation output. *Operations Research* **30:** 556-568.

Schmeiser, B. 1990. Simulation experiments. In: *Handbooks in operations research and management science. Vol. 2: Stochastic models*, eds. D.P. Heyman and M.J. Sobel, 295-330. Amsterdam: North-Holland.

Scott, M.D. 1990. A code generator for random-vector simulation experiments. Master's Thesis, School of Industrial Engineering, Purdue University, West Lafayette, Indiana.

Tao, Y. 1990. Simulation test bed. Master's Thesis, Department of Industrial and Systems Engineering, The Ohio State University, Columbus, Ohio.

AUTHOR BIOGRAPHIES

BRUCE SCHMEISER is a professor in the School of Industrial Engineering at Purdue University. He received his Ph.D. from the School of Industrial and Systems Engineering at Georgia Tech in 1975. His undergraduate degree in the mathematical sciences and master's degree in industrial and management engineering are from The University of Iowa.

Professor Schmeiser is active in a number of professional service roles. He is the current Simulation Area Editor of *Operations Research* and member of the Council of the Operations Research Society of America. He is an active participant in the Winter Simulation Conference, including being Program Chairman in 1983 and Chairman of the Board of Directors during 1988-1990.

MARK D. SCOTT is a consultant with American Airlines Decision Technologies. He is involved primarily with the airport consulting group, which simulates both air and land sides of airports. He received his M.S. in Industrial Engineering from Purdue University in 1990 and his B.S. in Computer and Systems Engineering from Rensselaer Polytechnic Institute in 1988.