Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

A SYSTEM FOR MONTE CARLO EXPERIMENTATION

David Alan Grier
Dept. of Statistics/Computer & Information Systems
George Washington University
Washington, DC 20052 U.S.A.

ABSTRACT

A new computer system for Monte Carlo Experimentation is presented in this thesis. The new system speeds and simplifies the process of coding and preparing a Monte Carlo Experiment; it also encourages the proper design of Monte Carlo Experiments, and the careful analysis of the experimental results.

A new functional language is the core of this system. Monte Carlo Experiments, and their experimental designs, are programmed in this new language; those programs are compiled into Fortran output. The Fortran output is then compiled and executed. The experimental results are analyzed with a standard statistics package such as S, Isp, or Minitab or with a user supplied program. Both the experimental results and the experimental design may be directly loaded into the workspace of those packages.

The new functional language frees programmers from many of the details of programming an experiment. Experimental designs such as factorial, fractional factorial or latin square are easily described by the control structures and expressions of the language. Specific mathematical models, such as arima(p,n,q) models, regression models with specific collinearity properties, tabular data generated by logit or log-linear models are generated by the routines of the language. Numerous random number generators and many standard statistic routines are included. It is easy to use standard variance reduction techniques, such as common or antithetic variables, conditional Monte Carlo, weighted samples, importance sampling or control variates.

1. INTRODUCTION

Since their introduction, computers have greatly helped statisticians analyze data. But in the last fifteen years or so, they have been used by statisticians to study the statistical techniques themselves. A common way of studying a statistical procedure is the statistical simulation, often called Monte Carlo Experimentation. To study a statistic in a Monte Carlo Experiment, the computer generates a series of pseudo random data sets. It calculates the value of the statistic for each of the data sets in the series, producing a new data set of the realized values of the statistic. From the new data set, the computer can estimate the numerical properties of the statistic, such as the variance, the bias or the rejection rate.

A new software system for Statistical Monte Carlo Experimentation is presented in

this dissertation. This system does for Monte Carlo Experimentation what packages such as SAS(SAS, 1979), Minitab (Ryan, Joiner, and Ryan, 1975), and S (Becker and Chambers, 1984) do for statistical analysis. The Monte Carlo System is a collection of numerical and statistical subroutines joined in a unified programming environment with a high level language. Using this system, a researcher can prepare a Monte Carlo Experiment, execute it and prepare the output from the experiment for analysis.

There are three reasons for creating a software system for Monte Carlo Experimentation. First, such a system would speed and simplify the process of doing Monte Carlo experiments. Most statistical Monte Carlo experiments are done by custom programs, written in languages such as PASCAL or FORTRAN. These programs tend to be tedious to write and take a long time to prepare and debug. Function libraries can be used in programming experiments, but they can only provide routines to generate random numbers, sort data and calculate basic statistics; and these tasks are only the basic parts of a Monte Carlo experiment. Researchers must still write code to control the experiment, describe the experimental design, accumulate data and analyze the experimental results. A software system would transfer some of that work to the computer.

A second reason to create a software system is to unify the field of statistical Monte Carlo experimentation. Monte Carlo Experiments are done by mathematical statisticians, biostatisticians, psychologists, econometricians, and physicists, as well as researchers in other fields. They do their work by custom programs, and publish their results in journals that do not specialize in Monte Carlo Experimentation. Hence new techniques and ideas disseminate slowly through the field. Over the years, these researchers have developed many useful techniques for reducing the variance of experimental results, for reducing the amount of computer work and for generating random quantities. Yet, most of these techniques are rarely used. Researchers that could utilize a new variance reduction technique, for example, are often unaware of the existence of the technique they need. If they know of a good technique, they must often code it themselves from scratch which may not be a good use of their time. The Monte Carlo system provides a medium for distributing new ideas about Monte Carlo experimentation in a form that can be used by researchers.

A third purpose for this software system is to improve the quality of Monte Carlo experiments. In a survey of published statistical Monte Carlo experimentation, Hauck and Anderson (1984) found several weaknesses in the use of experiments. They reported that researchers often use poor algorithms for generating random numbers or for calculating statistics. Sometimes researchers choose bad experimental designs and rarely do they analyze their results properly. A Monte Carlo system would correct the problem of poor algorithms by including a collection of carefully chosen algorithms and by providing standards to test them. Knuth (1982) and Marsaglia(1984)

give a list of tests for uniform pseudo random number generators. Including them in the software, plus routines for distribution tests such as the test, the Kolomgorov-Smirnov test, and the Anderson-Darling test, would simplify the process of writing routines to generate random numbers, and would encourage careful testing. The principles of Experimental Design are well established (Box, Hunter and Hunter, 1978). A simple control structure would encourage researchers to design their experiments using these principles. An interface with the major statistical analysis packages that would load data into those packages, would encourage the careful analysis of the experimental results.

2. EXISTING SYSTEMS

There are numerous simulation computer languages and some of these languages have existed for over twenty-five years. A recent catalogue of these languages in the October, 1984 issue of **Simulation** (Vol. 43, No. 4), lists over 400 different simulation languages. Most of the languages are designed for the narrow purpose of simulating a specific physical model. There are some general purpose simulation languages such as Simscript (Kiviat et al., 1975), (Dahl and Nygaard, 1967), Slam (Prisker and Pegden, 1979) and GPSS (Schriber, 1974).

There are also some simulation languages that were originally general purpose computer languages to which some extensions have been added. These languages usually contain new random number generators plus procedures and data structures to handle queues, calendars or networks. The languages GASP (Prisker, 1974), based on Fortran, Simpas (Seila, 1981), based on Pascal, and Simpli (Rubin, 1981), based on PLI, are examples of this type of language. They are often compiled by preprocessors that translate language statements into subprogram calls.

Virtually all of the simulation languages are concerned with the problems of simulating a physical, economic or psychological system, instead of simulating the mathematical problems of interest to statisticians. These languages are concerned with simulating events that occur over time and use queueing models or network models to control the timing of events and have special syntaxes for handling the queues or networks. The special syntaxes and concepts can rarely be used in statistical Monte Carlo Experimentation and these languages have nothing more than the normal computer language control structures for handling either experimental design or the standard form of most Monte Carlo Experiment programs.

Another problem with the simulation languages involves the fact that most Monte Carlo experiments make very specific distributional assumptions and the simulation languages don't provide the routines to support those assumptions. For example, an assumption might be that data in a certain stochastic model comes from a contaminated Normal distribution. If the contamination is small, it may take thousands of observations to verify that the data from a contaminated Normal distribution rather than a

simple Normal distribution (Mosteller and Tukey, 1977). Since they are designed to simulate a physical system rather than a mathematical one, the simulation languages usually only provide generators for the uniform, poisson, normal and exponential and often the normal generator will produce data from a distribution that is only approximately normal (Brately, Fox and Schrage, 1985). This lack of routines to generate random data further limits the usefulness of the simulation languages for statistical Monte Carlo Experiments.

None of the simulation languages have been widely used for Statistical Monte Carlo Experimentation (Friedman and Friedman, 1984). Most statistical problems cannot make use of the control structures for queues and networks. They need a greater range of probability distributions for the generation of random numbers than is often provided. In addition, many statistical problems require mathematical software that is not available to the simulation languages and they could benefit from control structures to collect data and design the experiments.

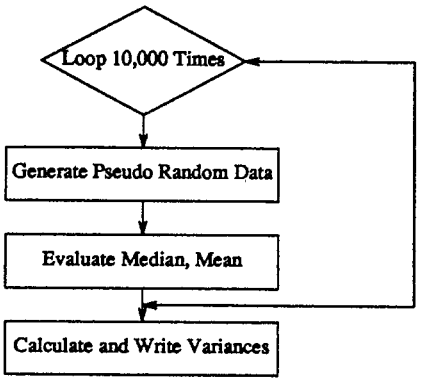
3. THE NATURE OF THE PROBLEM

Monte Carlo Experimentation is used to study the properties of statistics. It is most commonly used in cases where assumptions about the distribution of the data make it difficult or impossible to compute the probability distribution of a statistic. A common example is the problem of finding the small sample variance of an estimator; another example is the problem of finding the true, small sample coverage probability of a confidence set. Either of these properties could be computed if the probability distribution of the estimator or of the confidence set were known. When the true distribution of a statistic cannot be known because of the distribution of data, it can be estimated by generating random data sets, computing the value of the statistic for each of those data sets and calculating the empirical cumulative distribution function. Then the researcher can estimate the expected value, the variance, the true rejection rate or any other numerical property of the statistic.

Consider a simple example. Suppose that we wish to study the properties of the median and the 10% trimmed mean as estimators of location, both for distributions that are somewhat heavy tailed and for small sample sizes. Andrews et al. (1972) studied this type of problem in the Princeton robustness study. We want to compare the variance of our two estimators for samples of size 15; and we will generate our data from the contaminated normal family of distributions. (In the contaminated normal family of distributions, a given data point comes from a standard normal distribution N(0,1) with probability $1-\gamma$ and from a normal distribution with variance σ^2 with probability γ for $0 \leq \gamma \leq 1$. Since we are interested in heavy tailed distributions, we choose γ to be 0.2 and to be 50. We will replicate this experiment 10,000 times. The value of 10,000 was chosen to make the variance of our results, the variance of the sample variance, small. In performing this experiment, the

computer will generate 10,000 data sets from the contaminated normal distribution of length 15, compute the 10% trimmed mean and the median of each of those 10,000 data sets and then compute the sample variances of those 10,000 105% trimmed means and 10,000 medians. The sample variances will be our estimates of the variances of the 10% trimmed mean and the median.

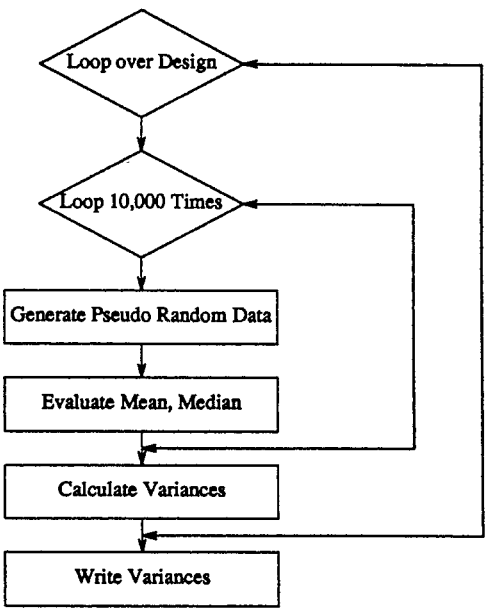
The basic program for this experiment is a simple, single loop program. Inside a loop, we generate a contaminated normal data set of length 15 and apply the median and 10% trimmed mean. We will execute this loop 10,000 times. After the execution of the loop, we will calculate the variance of our two estimators. The basic structure for this kind of program is seen in Figure 1.



Flowchart for Simple Monte Carlo Experiment
Figure 1

By expanding the experiment, we would like to determine how the size of the data sets and the heaviness of the tails of the original distributions affect the variances of our estimators. For this, we will have to design a more complicated experiment. We could look at sample sizes of 15, 25, 50 and 100, γ's of 0.05, 0.1, 0.2 and 0.3, and σ's of 10, 50, 100 and 500. We will combine these three different factors in a factorial design, performing our basic experiment 10,000 times for each possible combination of those factors.

The structure of the program we will need to write to produce the Monte Carlo experiment would then be a double nested loop program. It would have a outer design loop and an inner replication loop. The outer loop would loop over every point in the design loop and an inner replication loop. The outer loop would loop over every point in the design. The inner loop would be the simple Monte Carlo program pictured in Figure 1 above. The program needed for this experiment is pictured in Figure 2 below.



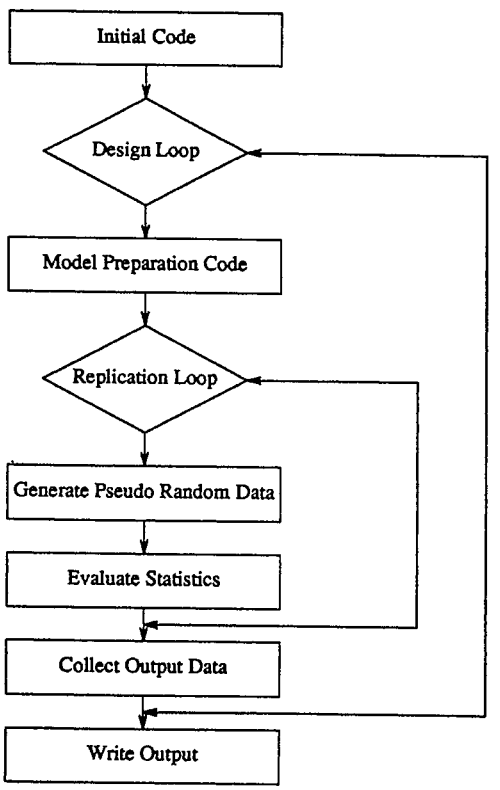
Flowchart for Typical Monte Carlo Experiment
Figure 2

Like the program in Figure 2, the general structure of the program for any statistical Monte Carlo experiment will contain two nested loops. The inner structure will have to generate data by repeatedly producing random numbers and calculating statistics. The outer structure will control the design of the experiment. Some experiments may lack either one of those loops and some might invert the loops to take advantage of a variance reducing swindle, such as a common variate swindle. Others, such as a bootstrap experiment, might require more than two nested loops. However, the basic program structure, seen in Figure 2, will handle most Monte Carlo Experiments. The Monte Carlo system, described in this dissertation, will treat a slightly more general version of the basic Monte Carlo program. This more general version is shown in Figure 3.

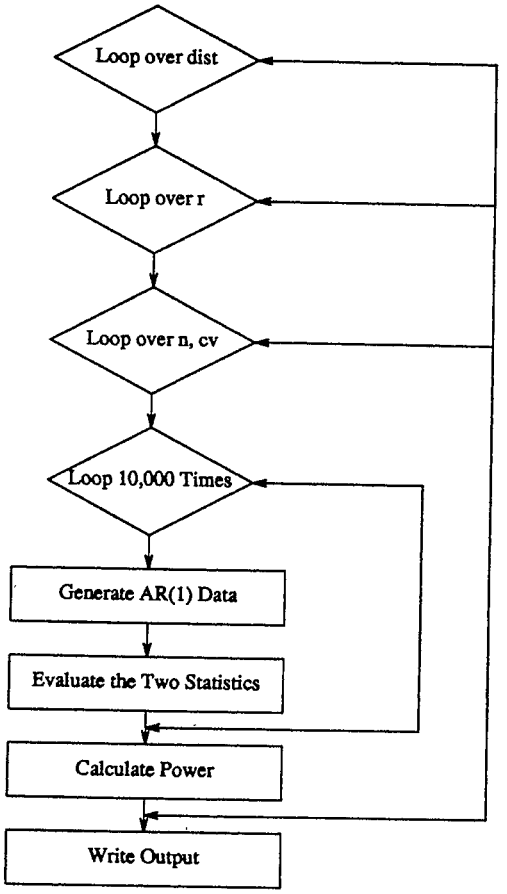
4. AN EXAMPLE

The following example is based on a Monte Carlo experiment by Bartels (1982). In that article, the author compares the small sample power of the von Neumann ratio test for autocorrelation, the rank version of the same test and a runs test. We will restrict ourselves to the von Neumann test and the rank version of the von Neumann test, and perform an experiment to calculate the power of these two statistics under various deviations from the null hypothesis of no correlation in the data. The tests will be performed at the asymptotic .05 level.

Using a first order autoregressive model, we will generate data sets with autocorrelation ranging from 0 to -.8. The data will come from four different distributions, the Normal, the Cauchy, the Scale Contaminated Normal with contamination parameter .05 and contamination standard deviation 10, and the double exponential. We will compare these two statistics on data sets of length 10, 20 and 50; and we will do 10,000 replications of the experiment, in order that our results may be significant to .01.



General Flowchart for Typical Monte Carlo Experiment
Figure 3



Flowchart for Bartel's Example
Figure 4

The basic experiment is a factorial experiment with three factors, dist, n and r. The factor dist is the distribution; n is the sample size; and r is the autocorrelation. To simplify the programming, we will introduce another factor, cv. The factor cv is the critical values for the tests. It is a function of the sample size and the critical values for the rank test differ from the values for the conventional test. It could be coded as an additional loop or a function. Instead, we will program it as a factor that is blocked with the factor n, the sample size. It will be an array of two values, the first for the conventional test and the second for the rank test. To do this coding, we will use the block expression to block the factor n and the factor cv together. The basic flow chart for this program is seen in Figure 4.

We will also modify the design the reduce the amount of computer work and to decrease the variance of comparisons across sample sizes. We will use the common expression to reuse the random variables generated for the smaller samples for the larger samples. The random data sets created for the experiment with the factor n equal to 10 will be reused when the value of n goes to 25 and later when it goes to 50. This will save the work required to generate 35 random variables for each interaction of the experiment. It will also introduce a positive correlation in the results across factor sizes. This positive correlation will reduce the variance of any comparisons across sample sizes with common other factors.

The code below presents two versions of the experiment. The first version is the simple version of the experiment without the extra code to reuse the random numbers. The second version is the more complicated version in which random quantities are reused. The text following pound signs, #, and continuing to the ends of the line are comments. Reserved words of the language are written in **bold**.

```
design(factor(                                  # factorial design
    dist(n) = ({rnorm(n)},                      # distributions for data
              {rcauchy(n)},
              {rcnorm(n,.05,10)},
              {rdexp(n)}),

    r = (c(0, -.1, -.3, -.5, -.8)),             # autocorrelation factor

block( n = (10,25,50),                          # sample size and
       cv = (c(1.36,1.04),                      # critical values
             c(1.45,1.36),
             c(1.59,1.54))
             )                                  # end of block
    )                                           # end of factorial design
){                                              # end of design

export() rep(mean,10000) {                      # 10,000 replications

      x := mkarma(dist(n),r,1);                 # make data

      vn := ifelse(sum(diff(x)**2) / (var(x) - 1)   # von Neumann
            > cv[1], 1, 0);

      rvn := ifelse(sum(diff(rank(x))**2) /     # rank von Neumann
            ((len(x)**2 - 1)/12) > cv[2], 1, 0);


      c(vn,rvn)}                                # return array of results
                                                # end of rep expression
      }                                         # end of design loop
```

Bartel's Example (Standard Version)
Figure 5

The second example is identical to the first except for the addition of a common block to the design. This **common** block reuses the pseudo random values.

```
design(factor(                                  # factorial design
    dist(n) = ({rnorm(n)},                      # distributions for data
              {rcauchy(n)},
              {rcnorm(n,.05,10)},
              {rdexp(n)}),

    r = (c(0, -.1, -.3, -.5, -.8)),             # autocorrelation factor

common( block( n = (10,25,50),                  # sample size and
       cv = (c(1.36,1.04),                      # critical values
             c(1.45,1.36),
             c(1.59,1.54))
             ))                                 # end of block and common
    )                                           # end of factorial design
){                                              # end of design

export() rep(mean,10000) {                      # 10,000 replications

      x := mkarma(dist(n),r,1);                 # make data

      vn := ifelse(sum(diff(x)**2) / (var(x) - 1)   # von Neumann
            > cv[1], 1, 0);

      rvn := ifelse(sum(diff(rank(x))**2) /     # rank von Neumann
            ((len(x)**2 - 1)/12) > cv[2], 1, 0);


      c(vn,rvn)}                                # return array of results
                                                # end of rep expression
      }                                         # end of design loop
```

Bartel's Example with Common Random Variates
Figure 6

5. ANALYZING THE RESULTS OF THE EXPERIMENT

We shall look at the results from the experiment described in Section B and attempt to determine the circumstances that cause the rank von Neumann test to be more powerful than the standard von Neumann test when for testing for negative autocorrelation, and we shall use the data from Bartel's original experiment (Bartels, 1982). To help analyze how the factors affect the power curves of the two statistics, we shall fit two linear additive models to the experimental results, the Monte Carlo estimates of the power curves of the two tests. The results from both models show that the rank von Neumann test has more power than the standard von Nuemann test in situations where the data is both heavy tailed and has small autocorrelation. The standard test slightly outperforms the rank test in situations where the data has autocorrelation of -.8. The sample size had the same small effect on the two statistics.

The first model fit to the was a simple linearadditive model with only main effects. The equation for the model is shown in Figure 7, along with a brief explanation of the parameters. The results from fitting the model are shown in Figure 8. The values for the coefficients are the contribution to the power curve made by each factor over the baseline case of the Normal distribution, sample size 10 and uncorrelated data. The fitted value of the baseline case is given by the coefficient labelled "Intercept" and is just the estimated level of the test for 10 data points from the Normal distribution. For example, the coefficient for

the sample size factor equal to 25 (labelled "N=25" in the figure), equal to .07591526 gives the increase of power over the baseline case for changing the sample size to 25. The factors labelled "diff", called the difference coefficients, give the difference between a factor for the rank test and the same factor for the standard test. A positive difference coefficient implies that factor contributes more to the power of the rank test than to the power of the standard test. A negative value implies that the factor contributes more to the standard test.

	Coef	Std Err	t Value
Intercept	-0.04578855	0.03305038	-1.385416
cauchy	-0.01509677	0.02684095	-0.5624531
cnorm	-0.006198847	0.02418737	-0.2562845
exp	-0.001781554	0.01619159	-0.1100296
N=25	0.09911107	0.03395068	2.919266
N=50	0.1549979	0.03275891	4.731475
r=-.1	0.06713232	0.03802494	1.765481
r=-.3	0.3785103	0.05344310	7.082491
r=-.5	0.7698574	0.04166798	18.47599
r=-.8	0.8915017	0.02752684	32.38663
diff	-0.03891959	0.00388887	-10.00793
cauchy diff	0.03522234	0.00248226	14.18961
cnorm diff	0.01243551	0.00232291	5.353403
exp diff	0.007591526	0.00182823	4.152372
N=25 diff	0.07063844	0.00398118	17.74308
N=50 diff	0.05166333	0.00385107	13.41530
r=-.1 diff	0.05669427	0.00482690	11.74547
r=-.3 diff	0.1157636	0.00604214	19.15935
r=-.5 diff	0.05890969	0.00431559	13.65044
r=-.8 diff	-0.02057531	0.00318795	-6.454082

Residual Standard Error = 0.3968147
Multiple R-Square = 0.978354
N = 120
F Value = 237.8806 on 19, 100 df

Results of Fit of First Model
Figure 8

$$p_{ijkl} = \mu + \alpha_i + \beta_j + \gamma_k + \mu_l^{diff} + \alpha_{il}^{diff} + \beta_{jl}^{diff} + \gamma_{kl}^{diff} + \varepsilon_{ijkl}$$

i = norm, cauchy, cnorm, exp
j = 10, 25, 50
k = 0, −.1, −.3, −.5, −.8
l = 0, rank

p_{ijkl} Power of the statistic

μ Level of the test for normal data of length 10
μ_l^{diff} Difference between rank and von Neumann level
$\mu_0^{diff} = 0$

α_i Effect due to the distribution
α_{il}^{diff} Difference in effect due to distribution
$\alpha_{i0}^{diff} = 0$ for all i

β_j Effect due to sample size
β_{jl}^{diff} Difference in effect due to sample size
$\beta_{j0}^{diff} = 0$ for all j

γ_k Effect due to autocorrelation
γ_{kl}^{diff} Difference in effect due to autocorrelation
$\gamma_{k0}^{diff} = 0$ for all k

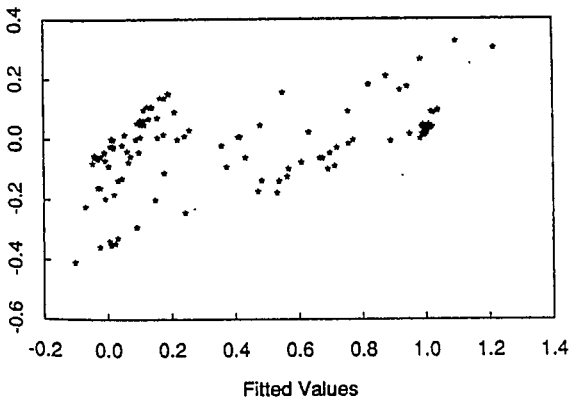
ε_{ijkl} Unexplained error term
$\text{var}(\varepsilon_{ijkl})$ is proportional to $p_{ijkl}(1-p_{ijkl})$

First Linear Model for Example
Figure 7

All the difference coefficients are positive, except for the factor "r= -.8" and the factor labelled simply "diff" which indicates the difference between the baseline values of the two statistics. The negative value of the "diff" coefficient indicates that the standard test has a slightly higher level than the rank test. The largest difference coefficient is the "r= -.2 diff" with a value of about .116. The next four largest are, in decreasing order of magnitude, "N=25 diff", "r= -.5 diff", "r= -.3 diff", "r= -1. diff". They are all bigger than the coefficient for "diff", indicating, that for those factors, the rank test has more power than the standard test.

However, this model does not fit the data well. The value of the intercept, which should be positive because it is supposed to give the level fo the standard test for Normal data, sample size 10, is negative. A quick glance at a residual plot, Figure 9, shows a clear trend in the residuals. There

is more structure in the data than this model can explain.



Residual Plot From First Model
Figure 9

The second model includes both main effects and first order interaction terms. It points to the same conclusions, albeit more specifically, implied by the first model but it fits the data better. The equation for this model is shown in Figure 10. The results from fitting it are given in Figure 11 and a residual plot is given in Figure 12. This residual plot has less discernible structure than the plot in Figure 9, implying a better fit than the first model. As an additional check, the Intercept coefficient is both positive and close to the theoretical level of the standard von Neumann test.

$$p_{ijkl} = \mu + \alpha_i + \beta_j + \gamma_k + \delta_{ij} + \zeta_{ik} + \eta_{jk} + \mu_l^{diff} + \alpha_{il}^{diff} + \beta_{jl}^{diff} + \gamma_{kl}^{diff} + \delta_{ijl}^{diff} + \zeta_{ikl}^{diff} + \eta_{jkl}^{diff} + \varepsilon_{ijkl}$$

i = norm, cauchy, cnorm, exp
j = 10, 25, 50
k = 0, −.1, −.3, −.5, −.8
l = 0, rank

p_{ijkl} Power of the statistic

μ Level of the test for normal data of length 10
μ_l^{diff} Difference between rank and von Neumann level
$\mu_0^{diff} = 0$

α_i Effect due to the distribution
α_{il}^{diff} Difference in effect due to distribution
$\alpha_{i0}^{diff} = 0$ for all i

β_j Effect due to sample size
β_{jl}^{diff} Difference in effect due to sample size
$\beta_{j0}^{diff} = 0$ for all j

γ_k Effect due to autocorrelation
γ_{kl}^{diff} Difference in effect due to autocorrelation
$\gamma_{k0}^{diff} = 0$ for all k

δ_{ij} Effect due to distribution and sample size
δ_{ijl}^{diff} Difference in effect due to distribution and sample size
$\delta_{ij0}^{diff} = 0$ for all i, j

ζ_{ik} Effect due to distribution and autocorrelation
ζ_{ikl}^{diff} Difference in effect due to distribution and autocorrelation
$\zeta_{ik0}^{diff} = 0$ for all i, k

η_{jk} Effect due to sample size and autocorrelation
η_{jkl}^{diff} Difference in effect due to sample size and autocorrelation
$\eta_{jk0}^{diff} = 0$ for all j, k

ε_{ijkl} Unexplained error term
$\text{var}(\varepsilon_{ijkl})$ is proportional to $p_{ijkl}(1 - p_{ijkl})$

Second Linear Model for Analysis
Figure 10

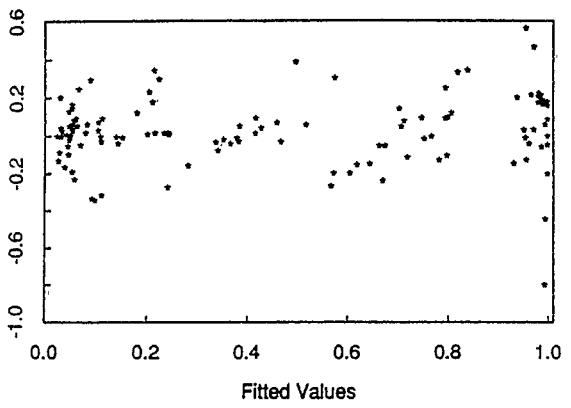
	Coef	Std Err	t Value
Intercept	0.04566148	0.01569252	2.909760
cauchy	-0.008863008	0.01929450	-0.4593540
cnorm	-0.01523599	0.01937728	-0.7862810
exp	-0.007019783	0.02041993	-0.3437711
r=-.1	0.03427143	0.02181895	1.570718
r=-.3	0.09629255	0.02888086	3.334130
r=-.5	0.3170538	0.02835428	11.18187
r=-.8	0.6532093	0.02482697	26.31047
N=25	0.006643924	0.01907946	0.3482240
N=50	0.006415711	0.01856728	0.3455385
(r=-.1 × cauchy)	-0.02770493	0.02452011	-1.129886
(r=-.3 × cauchy)	0.05610760	0.03385195	1.657440
(r=-.5 × cauchy)	0.02949658	0.02694342	1.094760
(r=-.8 × cauchy)	0.02051979	0.01826561	1.123411
(r=-.1 × cnorm)	-0.01370912	0.02528103	-0.5422692
(r=-.3 × cnorm)	0.03323159	0.03437202	0.9668210
(r=-.5 × cnorm)	0.02419550	0.02637002	0.9175384
(r=-.8 × cnorm)	0.01303348	0.01824976	0.7141729
(r=-.1 × exp)	0.007321722	0.02655699	0.2756984
(r=-.3 × exp)	0.02958349	0.03529438	0.8381927
(r=-.5 × exp)	0.01463342	0.02684701	0.5450671
(r=-.8 × exp)	0.01546604	0.01785500	0.8662024
(r=-.1 × N=25)	0.02812739	0.01998212	1.407628
(r=-.3 × N=25)	0.2589798	0.02969102	8.722497
(r=-.5 × N=25)	0.4014144	0.03060300	13.11683
(r=-.8 × N=25)	0.2681467	0.02451935	10.93612
(r=-.1 × N=50)	0.06697737	0.02134691	3.137567
(r=-.3 × N=50)	0.5567748	0.02816203	19.77040
(r=-.5 × N=50)	0.5823371	0.02610882	22.30423
(r=-.8 × N=50)	0.2937377	0.02347770	12.51135
(cauchy × N=25)	-0.01785738	0.02183406	-0.8178684
(cnorm × N=25)	-0.005245424	0.02208397	-0.2375218
(exp × N=25)	-0.01143195	0.02272980	-0.5029500
(cauchy × N=50)	-0.02174458	0.02111022	-1.030050
(cnorm × N=50)	-0.003050006	0.02130844	-0.1431360
(exp × N=50)	-0.008535272	0.02188685	-0.3899726
diff	0.01336847	0.001825398	7.323594
cauchy (diff)	0.003383649	0.002343732	1.443701
cnorm (diff)	0.000824492	0.002292900	0.3595850
exp (diff)	0.001191323	0.002402421	0.4958841
r=-.1 (diff)	-0.03949700	0.002533177	-15.59188
r=-.3 (diff)	-0.03055829	0.003285929	-9.299742
r=-.5 (diff)	-0.01559731	0.003221641	-4.841419
r=-.8 (diff)	-0.009648632	0.002839378	-3.398149
N=25 (diff)	-0.02498178	0.002200787	-11.35129

Results of Second Model Fit
Figure 11

	Coef	Std Err	t Value
N=50 (diff)	-0.01344390	0.002158256	-6.229059
(r=.1 × cauchy) (diff)	0.1563420	0.003130644	49.93924
(r=.3 × cauchy) (diff)	0.1915934	0.003825223	50.08686
(r=.5 × cauchy) (diff)	0.04909303	0.002897434	16.94362
(r=.8 × cauchy) (diff)	-0.01091001	0.002071304	-5.267222
(r=.1 × cnorm) (diff)	0.05146074	0.003012369	17.08315
(r=.3 × cnorm) (diff)	0.06291419	0.003937791	15.97702
(r=.5 × cnorm) (diff)	0.02599824	0.002944333	8.829927
(r=.8 × cnorm) (diff)	-0.01195592	0.002091582	-5.716208
(r=.1 × exp) (diff)	0.04132482	0.003133697	13.18723
(r=.3 × exp) (diff)	0.07076290	0.004011545	17.63981
(r=.5 × exp) (diff)	0.02618377	0.003009096	8.701542
(r=.8 × exp) (diff)	-0.01577092	0.002074789	-7.601218
(r=.1 × N=25) (diff)	0.03600698	0.002533349	14.21319
(r=.3 × N=25) (diff)	0.02173366	0.003457098	6.286678
(r=.5 × N=25) (diff)	0.03205910	0.003439215	9.321634
(r=.8 × N=25) (diff)	0.01998501	0.002799418	7.138988
(r=.1 × N=50) (diff)	0.08699303	0.002720082	31.98177
(r=.3 × N=50) (diff)	0.001257015	0.003214491	0.3910465
(r=.5 × N=50) (diff)	-0.005481657	0.002964340	-1.849199
(r=.8 × N=50) (diff)	0.009562719	0.002713774	3.523771
(cauchy × N=25) (diff)	0.03920404	0.002560384	15.31178
(cnorm × N=25) (diff)	0.02986895	0.002553679	11.69644
(exp × N=25) (diff)	0.03032771	0.002621921	11.56698
(cauchy × N=50) (diff)	0.01759541	0.002482373	7.088144
(cnorm × N=50) (diff)	0.01674535	0.002466194	6.789961
(exp × N=50) (diff)	0.01487447	0.002530054	5.879110

Residual Standard Error = 0.0884716
Multiple R-Square = 0.999483
N = 120
F Value = 1308.284 on 71, 48 df

Results of Second Model Fit
Figure 11 (Continued)



Residual Plot from Second Model
Figure 12

The biggest difference coefficients are not the main effects but the coefficients for the interactions. The six largest difference coefficients, in decreasing order of magnitude, are the "(r= -.3xcauchy)(diff)", "(r= 0.1xcauchy)(diff)", "(r= -.1xN=50)(diff)", "(r= -.3xexp)(diff)", "(r= -.3xcnorm)(diff)", and "(r- -.2xcnorm)(diff)". The coefficients for the main effect difference factors are close to zero. These interaction coefficients dominate them and, again, indicate that the improvement in power of the rank test over the standard von Neumann test is greatest in the situations where the data has both a low autocorrelation and a comes from a heavy tailed distribution.

6. SUMMARY

The example given above shows how the features of the Monte Carlo System speed and simplify the coding of a Monte Carlo Experiment as well as the careful planning of the experiment and the proper analysis of its results. A complete description of the system, including a formal definition of the Monte Carlo Language, may be found in the author's Ph.D. thesis (Grier, 1986), available from the University of Washington, Department of Statistics, Seattle, WA 98195.

ACKNOWLEDGEMENTS

The author wishes to thank Richard Kronmal, Andreas Baja and Peter Guttorp for their help.

REFERENCES

Andrews,D.F. et al., Robust Estimates of Location: Survey and Advances, Princeton University Press, Princeton,NJ 1972.

Bartels, Robert, "The Rank Version of von Neumann's Ratio Test for Randomness", JASA, 77:377 (March 1982), 40-46.

Becker,Richard A. and Chambers,John M., S: An Interactive Environment for Data Analysis and Graphics, Wadsworth, Belmont, CA, 1984.

Box,George E.P., Hunter,William G., Hunter,J. Stuart, Statistics for Experimenters, John Wiley, New York, 1978.

Bratley,Pual, Fox,Bennett L., Schrage, Linus E., A Guide to Simulation, Springer Verlag, New York, 1983.

Dahl,O.-J. and Nygaard,K., Simula: A Language for Programming and Description of Discrete Event Systems, Norwegian Computing Center, Oslo, Norway, 1967.

Friedman,Linda Weiser and Friedman, Hersheyll, "Simulation State of the Art", Statistical Computing and Simulation, 19:3 (1984), 237-256.

Grier, David Alan, "A System for Monte Carlo Experimentation", Ph.D. Thesis, Department of Statistics, University of Washington, Seattle WA, 98195; 1986.

Hauck,Walter W. and Anderson,Sharon, "A Survey Regarding the Reporting of Simulation Studies", JASA, August 1984, 38:3, 214-216.

Kiviat,P.J., Villanueva,R., and Markowitz, H.M., Simscript II.5 Programming Language, CACI Inc., Los Angeles, CA, 1975.

Knuth,Donald, The Art of Computer Programming, Volume 2, Addison Wesley, Reading MA, 1981 (Second edition).

Marsaglia,George, "A current view of random number generators", Proceedings Statistics and Computer Science: Sixteenth Symposium on the Interface, North Holland, 1984.

Mosteller,Frederick and Tukey,John, Data Analysis and Regression, Addison-Wesley, Reading, MA, 1977.

Prisker,A. Alan B., GASP IV Simulation Language, Wiley, NY, 1974.

Prisker,A. Alan B. and Pedgenm, Claude Dennis, Introduction to Simulation and SLAM, Halsted, New York, 1979.

Ryan,T., Joiner,B., and Ryan,B., Minitab II, Reference Manual, University Park: Department of Statistics, The Pennsylvania State University, 1975.

Rubin, Jerrold, "A Tutorial on SimPLI, Proceedings of the 1981 Winter Simulation Conference, Oren, Delfone and Shub eds., IEEE, Computer Society Press, Silver Spring, MD, 1981.

....., SAS User's Guide, SAS Institue, Cary, North Carolina, 1979.

Schriber,T.J., Simulation Using GPSS, Wiley, New York, 1974.

Seila,Andrew, "A Tutorial on SIMPAS", Proceedings of the 1981 Winter Simulation Conference, Oren, Delfone and Shub ed., IEEE, Computer Society Press, Silver Spring, MD, 1981.

AUTHOR'S BIOGRAPHY

David Alan Grier is an Assistant Professor in the Department of Statistics/ Computer and Information Systems at George Washington University. He received a B.A. in mathematics from Middlebury College in 1978 and M.S. and Ph.D. degrees from the University of Washington (1983 and 1986 respectively). He was involved in computer simulation and design while an employee of Burroughs Corp. (1977 to 1983). His current research interests include Monte Carlo Simulation, queueing models, compiler design, and Empirical Bayes methodology. He is a member of ACM, ASA and MAA.

David Alan Grier
Department of Statistics/
Computer & Information Systems
George Washington University
Washington, DC 20052
(202) 676-6359