SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO. 754

August 1987

TESTING STRATEGIES FOR SIMULATION OPTIMIZATION

by

Russell R. Barton

This research was conducted using the Cornell National Supercomputing Facility, a resource of the Center for Theory and Simulation in Science and Engineering at Cornell University, which is funded in part by the National Science Foundation, New York State, and IBM Corporation.

TESTING STRATEGIES FOR SIMULATION OPTIMIZATION

Russell R. Barton
School of Operations Research and Industrial Engineering
Cornell University
Ithaca, NY 14853, U.S.A.

ABSTRACT

There is increasing interest in science and industry in the optimization of computer simulation models. Often these models are not Monte-Carlo simulations, but consist of systems of differential equations, or other mathematical models. These models can present special problems to numerical optimization methods. First, derivatives are often unavailable. Second, function evaluations can be extremely expensive (e.g. 1 hour on an IBM 3090). Third, the numerical accuracy of each function value may depend on a complicated chain of calculations, and so be impractical to pre-specify. This last point makes it difficult to calibrate optimization routines that use finite difference approximations for gradients. This paper presents a strategy for comparing optimization techniques for these problems, and reviews several interesting findings for quasi-Newton methods, simplex search, and others.

1. INTRODUCTION

Design engineers have seen remarkable decreases in the cost of computing over the past five years. They can now make many repeated runs of their models for the time and cost of a single run several years ago. This new opportunity has greatly increased the interest in optimization methods for computer simulation models. Models used previously for evaluation of designs can now be used iteratively as design tools themselves. These engineers want to know which optimization methods can be applied to their models, and under what circumstances.

This question is difficult to answer in a satisfactory way. Each model has different properties, and each design problem poses different goals. This paper presents some useful strategies for comparing optimization codes to asses their applicability. The pros and cons of various algorithms will not be discussed, except as examples to illustrate the algorithm testing methodology. *That is, this paper is about how to compare optimization codes, and not about optimization codes per se .*

For the prospective user, this information is important for critical review of published tests. Without this perspective it is difficult to tell whether particular test results justifiably qualify or disqualify an algorithm. One must be able to distinguish anecdotal results from compelling evidence.

For the optimization code designer or tester, the ideas here have two purposes. First, to present views of algorithm performance that can aid in algorithm design and revision. Second, to provide a useful framework for performing and presenting computational results.

The methodology is based on the statistical design of experiments. It identifies some useful test functions and measures of performance. An example comparing three optimization codes is presented as an illustration.

2. SCOPE

We assume that the problem at hand is the *unconstrained minimization* of a single output measure, f, of a simulation model. The measure f is a function of one or more simulation model input parameters represented by the vector x. The function is assumed to be expensive to evaluate relative to the optimization code's calculations. The output of the simulation program may have a component of Monte-Carlo random error and/or machine roundoff error. This implies the need for robust optimization methods, and distinguishes this study from that of numerical optimization methods. For an example of the latter, see Moré et al. (1981).

For example, the simulation model might be a partial differential equation solver for electron trajectories, and x might represent voltages and dimensions of an electron gun. The output function might be some measure of the beam size. Similar models are described by Winarsky (1987). Models of this sort exist which require many minutes of cpu time on an IBM 3090. Alternatively, consider a Monte Carlo simulation of a digital communications network, with x representing flow control parameters and f(x) providing a measure of network throughput.

The single measure, f(x), means that dynamic systems output is not available to the optimizer, so spectral techniques are not applicable. The function is assumed to be real valued; not discrete,

as are the parameters represented by the x vector. The important issues of multi-objective optimization discussed by Graver et. al. (1980) and the discrete optimization methods of Lee and Azadivar (1985) are not addressed specifically.

The discussion is further limited to tests for optimization of simulation models whose structure is unknown, or at least not exploited by the optimization algorithm. This means, for example, that the perturbation methods applied by Suri, et al.(1985) are not considered. This black box structure illustrating the problem scope is presented as Figure 1.

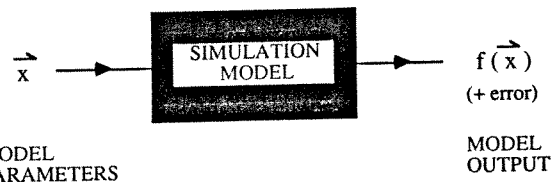

Figure 1: Scope for Minimization: Black Box Representation

So we see that this 'general' approach to testing will in fact deal with just a subset of the simulation optimization methods discussed in the literature. This is not to say that the techniques here have little value. Certainly the highest levels of the testing methodology to be discussed would also apply in more general cases. Details, such as measures of performance, would change. Furthermore, many of the optimization situations described above involve unconstrained optimization subproblems, e.g. constrained optimization using penalty functions. The kinds of optimization problems that can be modeled as shown in Figure 1 includes sequential process optimization and response surface models in settings more general than simulation optimization. See Barton (1984) for a discussion of this model.

3. EXPERIMENT DESIGN

The general procedure follows the basic steps of experimental design:

1. State the objectives of the study; formulate hypotheses.
2. Design an experiment to test the hypotheses.
3. Run the experiment.
4. Analyze the data to provide information on the hypotheses.
5. Revise the objectives and hypotheses, and return to 2.

This is just the scientific method, of course. Steps 1., 2., and 4. are discussed below in terms specific to the scope outlined above.

3.1. Objectives and Hypotheses

Why do we test optimization codes? Some tests are designed to validate a particular algorithm as an effective method. This usually means a set of comparative runs with a small number (or none!) of alternative codes on a small set of test problems. There is usually a single starting point for each problem, perhaps two. The author of an algorithm is obliged to perform this kind of testing in order to publish his invention.

Some engineers have very specific model forms, and test optimization codes on such a narrow class. The kind of tests motivated by this situation are too specific to discuss here. The findings of such tests may have little value for other model classes.

Some tests have been designed to compare algorithms in more general settings without the need to validate any particular algorithm. This is the situation we address, with just two primary objectives:

1. For users, we want a general assessment over a *broad set of situations* of the *effectiveness* of the optimization algorithms we would like to consider. We wish to test hypotheses about differences in effectiveness of algorithms, both overall and for particular problem classes.

2. For developers and users, identify *specific performance characteristics* for each algorithm; its strengths and weaknesses. We want to test hypotheses about algorithm sensitivity to random perturbations of the function value, badly scaled parameters, and non-quadratic response surfaces.

To determine a broad set of situations means determining what is called a **frame** in experimental design terminology. Effectiveness is defined by identifying appropriate **measures of performance**. We make these definitions and choices based on the characteristics of our problem scope. Specific performance characteristics can be examined by careful selection of both design frame elements (see the pseudo-function discussion below) and measures of performance (see the definition of RHO).

3.2. The Design Frame

The design frame consists of a set of simulation functions, optimization codes, and starting conditions. The simulation functions may be generated by actual simulation models or they may be artificial ones created explicitly as functions, with a random error component. Artificial simulation functions have two advantages; they are inexpensive to compute, and they can be designed to test specific algorithm behavior. A design frame for tests involving artificial functions has a fourth component: the kind of random/roundoff error generator.

The proposed components for the experimental design frame are shown in Figure 2. Each optimization algorithm has a set of associated parameters, some of which are standard, while others may be set by the user. Noise function families have parameters as well; for example, the upper and lower limits of a uniform distribution are its parameters. Selection for each of these are discussed below.

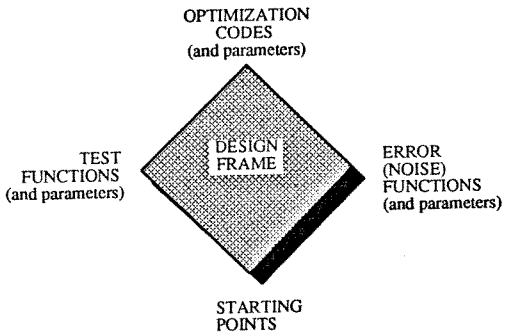

Figure 2: Design Frame for Testing Optimization Codes

Optimization Codes. Amyot and van Blokland (1985) provide nine optimization routines in their ACSL package. Any test should include a simplex/complex routine and first and second order response surface methods as described by Biles and Swain (1979).

It is important to recognize that *codes* are being tested and compared, not *algorithms*. Details of implementation may not save a poor algorithm, but if neglected they can certainly harm a good one. Publicly available versions of algorithms should be used where possible. The setting of optimization code parameters for step lengths, etc. presents a difficult situation. One approach is to use the default settings supplied with the optimization code. Another possibility is to select parameter values randomly from a limited range. This is discussed in more detail in the randomization section.

Test Functions. Ideally, we would like a set of test functions that represent well the entire population of possible simulation model response functions. We have no characterization for this population. One might approach this difficulty by testing a random sample of simulation output functions from the population we wish to study. We might classify the functions by type, e.g. PDE/ODE, general Monte Carlo, queueing simulations, etc. We would also like to classify the functions by the dimension of the x vector, e.g. small (2-10), medium (11-50), and large (50+). This not practical for several reasons. Real simulation models are often proprietary. The number of available models is too small. The computational burden in running large simulation models just to test algorithms is too great.

Actual simulation models can make up part of the design frame, but their purpose can really only be confirmatory. Artificial functions must play a role. The numerical analysis literature contains many real functions which can be used as the deterministic components of artificial simulation functions. These functions tend to appear in the literature because they are unusual and in some way difficult to optimize. One might argue that they are not representative of the difficulties present in simulation output functions; but we have no characterization for this population.

One might also argue that they are unusually difficult and atypical; that is why they have been published. This is not a drawback, if one can assume that algorithms that do well on these functions will also do well on simpler or more typical ones. In this case the published test functions provide a stratified sample that allows greater ability to resolve differences among optimization codes.

These published functions do not provide a good means for meeting our second objective, the characterization of algorithm strengths and weaknesses. For this purpose we propose a class of subroutines called *pseudofunctions* (Barton, 1986).

Pseudofunctions. The difficulties are illustrated by the well known Rosenbrock (1960) function. The function has a descending curved valley, but this feature is not identified and followed unless the starting point is suitably chosen. A second shortcoming is that, the non-quadratic nature of the valley disappears as the optimum is approached(true for any analytic function). Figure 3 shows the region of function values that are two through four orders of magnitude below the value of 24.2 observed at the usual starting point: (-1.2,1). The contours that the algorithm sees at this level have very little curvature. The contours are shown after applying a linear transformation to the x vector. Third, it is difficult to assess particular aspects of algorithm performance because a variety of contours will be traversed in an unpredictable way.

We would like to be able to expose an optimization code to particular function topographies in a controllable way. This would allow characterization of an algorithm's strengths and weaknesses and perhaps suggest modifications. Pseudofunction subroutines provide this opportunity. The motivation for developing pseudofunctions was inspired by Lyness (1979).

In two dimensions, one can think of a pseudofunction as made

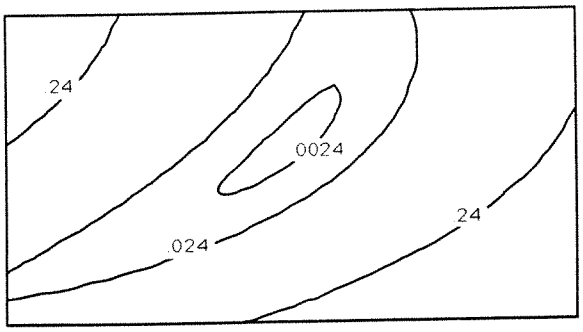

Figure 3: Rescaled Contours of Rosenbrock's Function Near the Optimum

up of two components; a descending spiral part, plus a penalty function for deviating from the spiral:

$$f(x) = u(x) + \alpha\theta \quad , \qquad (1)$$

where u(x) is a penalty function and θ is the cumulative angle of progress around the spiral. In higher dimensions, these functions can be constructed using parameterized curves and potential functions. These subroutines do not provide true functions, because the returned value depends on how many times the search trajectory has proceeded around the spiral. This allows them to forever remain non-quadratic They are locally well defined, however; derivatives of any order can be derived analytically. In addition, by substituting u(Ax) for u(x), where A is a nonsingular matrix, elliptical search trajectories can be generated.

A contour plot for one revolution of a circular pseudofunction is shown in Figure 4. Note that the algorithm cannot escape the curved non-quadratic nature of the contours. These constructs have no minimum, but rather permit infinite reduction.

Error Functions. For artificial simulation functions, a distribution function for the error term must be chosen. Uniformly distributed errors are appropriate when one wants can assume little about the nature of the random error. Normally distributed error terms may be appropriate for simulation output functions where the output is the sum of many terms, so that the Central Limit Theorem holds.

Starting Points. Hillstrom (1977) discussed the importance of using multiple starting points for each test function. For most test functions, this provides the optimization routine with varied terrains. Furthermore, *random* selection of starting points provides an opportunity to make probabilistic assertions about the relative performance of optimization algorithms.

3.3. Design Issues: Randomization

To produce comprehensive rather than anecdotal results, we need to test over the full range of the frame. Randomization, blocking, and fractional designs must be used to do this efficiently. These statistical techniques are covered in statistics texts, for example in Box, Hunter, and Hunter (1978). Without these techniques, the number of possible experimental runs can rapidly become overwhelming.

Randomized sampling plays a key role in any experiment; some believe that it provides the only justification for statistical inference. For testing optimization codes it provides both statistical inference and efficiency. By randomly selecting starting points in our frame, we can make statements about the likely outcome of a comparison of two algorithms *for any starting point in the sampling region.* This is a powerful concept.

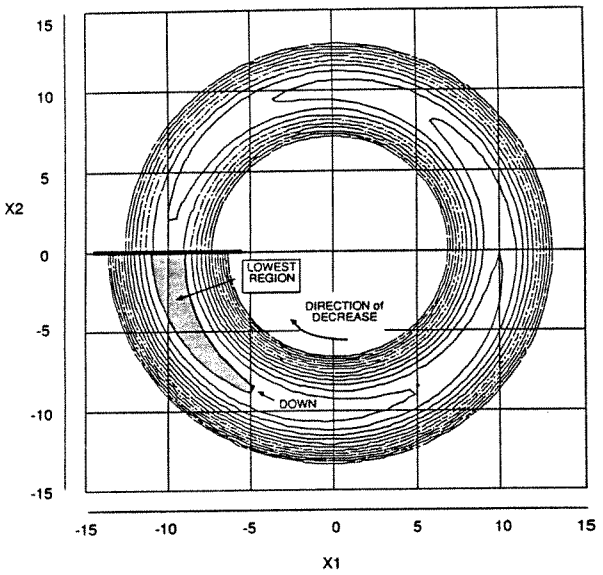

Figure 4: Contour Map for One Revolution of a Pseudofunction

Randomization can also provide some general results when faced with a very large set of design parameters. This may happen, for example, if the optimization algorithms have many tuning parameters that one wishes to include in the design. Noise distribution and test function parameters can add further to the complexity. In this situation one may choose to randomly select starting points and algorithm, noise, or test function parameters.

3.4. Measures of Performance

Performance measures should be based on practical considerations: what must a "good" simulation optimizer be able to do well? Certainly it must economize on function evaluations; each one entails a simulation run. Two obvious measures of this economy are: a) the number of function evaluations to reach a specified (reduced) function value or b) the reduced level of the function given a fixed number of function evaluations.

These two measures are related in the following way. If method I has fewer function evaluations than method II to reach level λ, and this occurs after n_λ function evaluations, then I also outperforms II by measure b) if the fixed number of function evaluations is set to n_λ.

PERED. We would like to be able to combine performance data over several test functions. To do this we can't use a) or b) above without standardizing the scale of measurement of f. For artificial simulation functions the minimum function value is known (excluding pseudofunctions) and so we can normalize the value to a percentage of the *gap* between the starting point function value and the optimal value. We write the percent reduction after the i^{th} function evaluation (see Barton, 1984) as:

$$\text{PERED}_i = (f_{start}-f_i) / (f_{start}-f_{opt}) \qquad (2)$$

The normalized value in (2) is easy to interpret in a practical way. For example suppose we wished to optimize the conversion efficiency of a solar cell simulation model, and we had achieved efficiencies of 5% before starting the optimization. Suppose further that the theoretical maximum cell efficiency was 15%. Then each percentage point gained in efficiency would correspond to a .1 increase in (2). Often engineers are satisfied with an order of magnitude improvement; PERED=.9. Sometimes achieving just .5 is noteworthy!

RHO. A different kind of performance measure has been largely overlooked in the literature. Box and Draper (1969) discuss this in their text on Evolutionary Operation: the excursions of the optimization iterates from the currently best point. An algorithm with large excursions can have overflow or stability problems with some simulation models. It can also encounter more constraints if used as a subroutine for constrained optimization. Algorithms with large excursions on the other hand, are more likely to avoid local minima in their search for optimality.

We define a measure here that has meaning only for pseudofunctions of the kind described above. Let R denote the radius of the spiral, and let r_i be the radius (norm) of the i^{th} iterate, x_i. Then define RHO after the i^{th} function evaluation as:

$$\text{RHO}_i = R - r_i \qquad (3)$$

A bias in the average value of (3) may indicate inertial behavior for an algorithm. Combined with bias, the dispersion of RHO values indicates how well an optimizer follows the curved valley.

Gap/σ. One way of reducing the complexity of the design frame is to introduce a measure that characterizes a class of test functions. Two such measures are discussed in Barton (1984). One of these is particularly relevant when the goal is to reduce the *gap* described above by only one or two orders of magnitude.

This difficulty in optimization depends on a signal-to-noise like quantity, which is the size of the starting gap divided by the standard deviation of noise contamination. It is easy to get PERED values of .9 or .99 for functions with gap/σ values of .001, but quite difficult if gap/σ is 1.

Quasi-Newton methods can be very efficient for optimization situations where gap/σ is small. All methods based on Taylor approximations to the underlying functions run into difficulties as gap/σ increases. This is due to conflicting pressures of errors in approximation introduced by large finite difference step size versus errors in gradient estimates from noise and small finite difference steps.

Measurement Technique. It is important to try to measure different algorithms using the same yardstick. There have been comparisons in the literature based on a 'final' objective function and a 'final' number of function evaluations for each method, where neither the function value nor the iteration counts were the same. It is hard to compare algorithms when they have not been run for the same amount of time, and simultaneously have generated different results. Unless, of course, the one with the shorter running time also had better results.

These uneven runs have occurred, I suspect, because testers have let the codes run to termination. Because our scope is limited to cpu-intensive simulations, letting algorithms run until they stop is not a useful performance measure. In practical situations an a-priori limit is set, either on the number of function evaluations, or on the percent reduction of the starting gap (PERED). Optimization test measurement procedures should reflect this. This usually does not require modifications to the optimization code.

4. TOOLS for ANALYSIS

Using the design techniques described above will generate a set of useful data, to which one can apply statistical tools and compare optimization codes. The output trajectory and function values for a single run of an optimization code on a particular test problem and starting point can be thought of as a p dimensional random performance vector P, that is the dependent variable in the analysis model. Values of the test function parameters, noise, algorithm, and starting point make up the v components of the independent variable vector, V. We can represent the statistical model as:

$$P = G(V) + \text{error} \qquad (4)$$

where G is an unknown function from R^v to R^p.

4.1. Statistical Tools

Statistical tools can be used to elicit information about the model in (4) from a given set of output data. These tools fall into three broad classes: graphical displays, summary statistics, and confidence intervals/tests of hypotheses.

Graphical Displays. It is a well worn phrase, but worth repeating: one picture is worth a thousand words. Figure 5 shows the search trajectory for one run of the simplex code on the pseudo function shown in Figure 4. The points evaluated by the simplex code are connected sequentially by line segments. The source file of 350 pairs of coordinates does not reveal this structure. Useful diagnostic plots, in addition to the trajectory plots, include contour plots for the functions and plots of PERED vs iteration. These plots show different views of the multidimensional response vector that includes trajectory and function values.

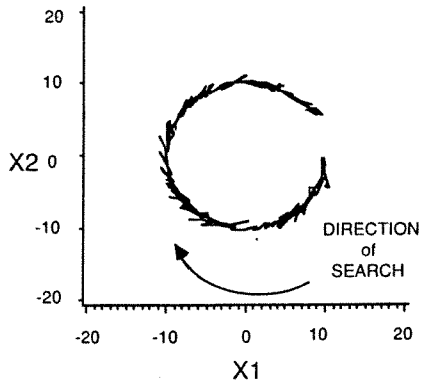

Figure 5: Plot of the Sequential Values of X Requested by the Nelder Mead Simplex for the Pseudofunction

The performance vector P generates a number of statistics, such as the average function reduction per iteration, the number of iterations to reach a PERED of .5, and the standard deviation of RHO. These statistics can be examined graphically using the tools of Exploratory Data Analysis: Box plots, Stem and Leaf plots, and Probability plots (Velleman and Hoaglin, 1981). Box plots show the general nature of the distribution of the statistic. Stem and Leaf plots present the numerical values in a histogram form. Normal probability plots indicate how the distribution of the statistic differs from a normal distribution. Normally distributed data appears as a straight line on this plot.

Summary Statistics. We are interested in the distribution of these performance statistics over a randomly selected set of starting points, and perhaps for other randomly selected frame parameters as well. These distributions can be characterized by a variety of summary statistics: population mean and median, percentiles, standard deviation and higher moments. The kurtosis statistic gives some indication of the weight of the tails of the distribution. A kurtosis value of 1 matches the weight of the tails of the normal distribution.

Confidence Intervals and Tests of Hypotheses. These tests allow us to make probabilistic inference about algorithm performance beyond the actual test runs for which we have data. Their validity is based on the randomization that was performed as part of the experiment design.

Two probability models can be used to test hypotheses and develop confidence intervals. First, individual test runs may be grouped and averaged. The Central Limit Theorem can be invoked to allow tests and confidence intervals based on normal theory. Unfortunately, this usually means making LOTS of test runs to get enough observations to group, and also have enough groups to make reasonably powerful inferences. Alternatively, the original data may be analyzed without grouping, using nonparametric methods. The latter approach is illustrated for the examples below.

An Analysis of Variance or Analysis of Covariance is used to test hypotheses about which algorithms are best under what circumstances. Factors and covariates can include gap/σ, algorithm parameters, algorithm type, and so forth.

4.2. Software Tools

Performing these graphical and statistical analysis would be impractical without modern graphical and statistical software. Trajectory plots like Figure 5 can be generated easily with FORTRAN subroutine libraries such as DISSPLA by ISSCO and DI3000 by Precision Visuals. Both of these vendors offer high level language graphics environments as well. SAS provides a

high level graphics environment called SAS/GRAPH. All of the figures in this presentation were generated using SAS.

The statistical tools described above are available in many packages. All are provided, for example, by SAS. Univariate analyses, analyses of variance, and graphical displays can each be invoked with one or two lines of commands.

Finally, one needs software tools to exercise the optimization codes and provide the test data to analyze. There are no high level packages available for this task. We have used our own package written in FORTRAN and UNIX scripts to automatically run factorial designs over function, algorithm, starting point, and noise parameters. The data for the example below was generated with this package.

5. AN EXAMPLE

The purpose of this example is to illustrate the testing methodology, not to present specific test results. The optimization algorithms that are presented are for tutorial purposes. They are not production codes. This is in keeping with the theme of this paper, which is about testing, not about algorithms. Nevertheless they are working codes based on popular methods, and the analysis does provide interesting results.

5.1. Experiment Design

This design involves testing three algorithms on two test functions at three levels of random noise. Ten starting points were chosen randomly for each function. This is a $2^1 3^2$ factorial design.

The test functions chosen were Rosenbrock's function and a pseudofunction of the form (1). These illustrate the calculation of statistics for the two classes of artificial simulation output functions. The error distribution was chosen to be uniform, with levels of ±0, ±1 and ±2 for the Rosenbrock function and ±0, ±.1, and ±.2 for the pseudofunction. The starting points for the Rosenbrock function were chosen from the square (±2, ±2) and for the pseudofunction from the square ((+1,+20), (+1,+20)).

One of the algorithms included in the runs was a simple first order a gradient step method. This was to serve as a baseline. Also tested were two direct search methods: a modified Nelder-Mead (1965) simplex search and the Hooke and Jeeves (1961) method.

The design matrix contains 18 rows, with 10 replications for each row. A total of 180 simulation optimization runs were performed. Each method was run for 350 function evaluations for the pseudofunction test problem, and 100 evaluations for the Rosenbrock function.

5.2. Results and Analysis

The three methods performed quite differently one the test functions. This is illustrated by looking at the trajectory plots. Not all 180 trajectories were plotted; just one per replication set. Two of these 18 plots are shown in Figure 6. They are sized to be approximately the same scale. These are the trajectories for the gradient step and Hooke and Jeeves methods for one pseudo-function run. Compare these with the simplex trajectory of Figure 5.

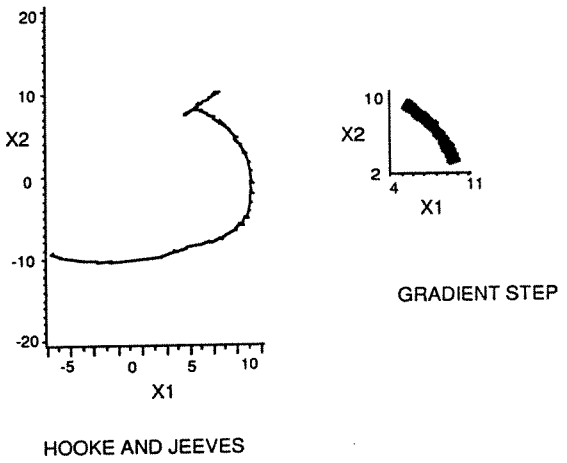

Figure 6: Pseudofunction Trajectories for Two Algorithms

The Hooke and Jeeves trajectory is very steady, but it wanders substantially out of the valley. The gradient step method makes slow zig-zagging progress, but it remains centered over the spiral. All three plots show the same number of function evaluations, so the simplex method is clearly most effective for this run. These visual findings are supported by the statistical analyses reported below.

The statistical analysis considered different sets of performance measures for the two classes of functions (pseudofunctions and true functions). For pseudofunctions, average function reduction per iteration, average RHO, and std.dev. RHO were calculated for each run. They were compared graphically using grouped Boxplots, with a separate Boxplot for each level of noise. The plot for average RHO and uniform ±.2 noise is shown in Figure 7. The algorithms are denoted by hjm (Hooke and Jeeves), ngs(first order negative gradient step) and nmsm (Nelder and Mead simplex

method). The wanderings of the Hooke and Jeeves method are apparent in the large variation in bias over the 10 replications. Boxplots of std.dev. RHO show similar characteristics.

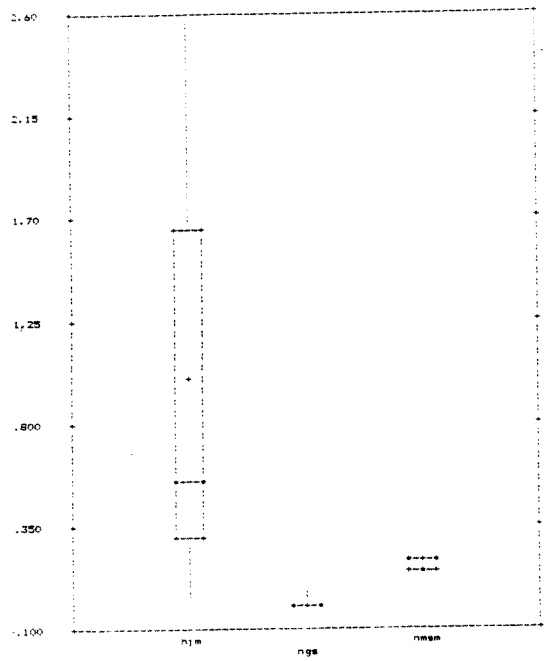

Figure 7: Comparative Boxplots for Average RHO value, Uniform(-1,1) Noise

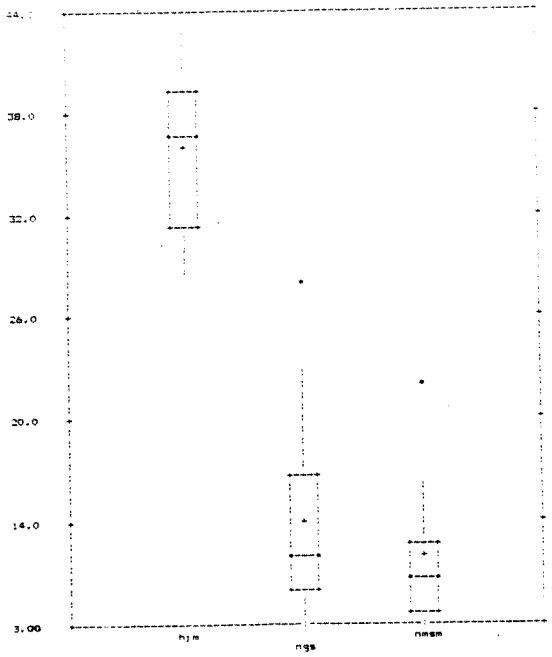

Figure 8: Comparative Boxplots for Iterations to Achieve 99% Gap Reduction, Uniform(-1,1) Noise

The measures for the Rosenbrock function were based on PERED. The number of iterations to achieve reductions of 50%, 90%, and 99% were shown in comparative Boxplots for each noise level. The plot for the uniform ±1 noise distribution runs is shown in Figure 8.

The graphical results were confirmed by nonparametric one-way analyses of variance. Two of these tables are shown below. For the pseudofunction, the Nelder Mead method maintained the closest proximity to the spiral, when considered over all starting points and all three noise levels. Table 1 shows the nonparametric test for the null hypothesis of equal average RHO values for the three algorithms. A statistical test of no difference between methods failed with a very high significance level: a p-value of less than .0001. The Kruskal Wallace test is based on ranks, the order of the standard deviations of RHO for all tests combined (all three algorithms. If there is no difference among algorithms, the order (ranks) of the observed standard deviations should be intermixed. The higher rank sums for the Hooke and Jeeves method (hjm) indicate higher standard deviations.

Table 1: Nonparametric ANOVA for the Standard Deviation of RHO, for all Pseudofunction Runs

ANALYSIS OF NELDER-MEAD, HOOKE-JEEVES, AND FIRST ORDER METHODS

ANALYSIS FOR VARIABLE SDRHO CLASSIFIED BY VARIABLE ALGF1

AVERAGE SCORES WERE USED FOR TIES

WILCOXON SCORES (RANK SUMS)

LEVEL	N	SUM OF SCORES	EXPECTED UNDER HO	STD DEV UNDER HO	MEAN SCORE
hjm	30	2209.00	1365.00	116.83	73.63
ngs	30	1421.00	1365.00	116.83	47.37
nmsm	30	465.00	1365.00	116.83	15.50

KRUSKAL-WALLIS TEST (CHI-SQUARE APPROXIMATION)
CHISQ= 74.50 DF= 2 PROB) CHISQ=0.0001

Table 2 shows that the Nelder-Mead method and the gradient step method were both efficient at achieving a 90% reduction in the Rosenbrock objective function gap. This result again is over all starting points and three noise levels. There were not 30 observations per cell in this one-way analysis of variance because the algorithms had difficulty reducing the gap by 90% for one of the starting points.

Table 2: Nonparametric ANOVA for the Number of Function Evaluations to Reduce the Gap by 90%, for all Pseudofunction Runs

ANALYSIS OF NELDER-MEAD, HOOKE-JEEVES, AND FIRST ORDER METHODS

ANALYSIS FOR VARIABLE N90 CLASSIFIED BY VARIABLE ALGP1

AVERAGE SCORES WERE USED FOR TIES

WILCOXON SCORES (RANK SUMS)

LEVEL	N	SUM OF SCORES	EXPECTED UNDER HO	STD DEV UNDER HO	MEAN SCORE
hjm	27	1836.00	1120.50	100.33	68.00
ngs	27	787.50	1120.50	100.33	29.17
nmsm	28	779.50	1162.00	101.23	27.84

KRUSKAL-WALLIS TEST (CHI-SQUARE APPROXIMATION)
CHISQ= 50.91 DF= 2 PROB > CHISQ=0.0001

Only a fraction of the statistical analyses were presented here, but they span the kinds of tests and expose some findings of interest. In summary, the major findings of the example experimental design and analysis were:

a) The Nelder-Mead method is the best overall performer for these tests.
b) The gradient method is more sensitive to increasing levels of noise than the Nelder-Mead method.
c) The Hooke and Jeeves method is most strongly affected by noise.

5.3. Implications for Comprehensive Testing

What is the domain of applicability of these findings? Statistical inference allows us to say that these conclusions hold with high probability over the entire range of starting points considered, but with some serious limitations. The codes compared had fixed code parameter settings. For example, the delta step size for the Hooke and Jeeves method was set at .01 for the Rosenbrock function. We have no justification to assume that these findings apply to the method with a delta step size of .5 or even .2. We used only two test functions - and these were both on R^2. Certainly this does not provide an acceptable coverage of the simulation function population.

This 180 run experiment falls far short of answering the questions the design engineers are asking. This, of course, was not the intent. It is certainly possible to do a better job by designing an experiment with varying algorithm parameters and more test functions. Furthermore, a useful test should employ publicly available code for all of the most frequently used and cited methods, including complex/simplex, first and second order response surface methods, quasi-Newton methods, and the too-frequently used one-at-a-time variation method. Adding all these factors could result in an enormous design that could not be carried out. The techniques of randomization and fractional designs are needed to keep the number of runs to a manageable level.

How manageable was the level of experimentation performed in the example? The 180 simulation runs were calculated on a SUN 3-50 UNIX workstation. The elapsed time for all 180 runs was approximately 40 minutes. The output files totaled approximately 4 megabytes. These files were analyzed using SAS on an IBM 3090. The cpu time for the SAS job was approximately 30 seconds. The tests and graphs for this analysis amounted to approximately 70 pages. The total computing cost was less than $50, including SAS code development and preliminary analyses. No costs were attributed to the use of the SUN, which is in a computing laboratory. The cost for a diskless SUN 3-50 for academic institutions is now on the order of $4000.

Given this computing and software environment, a reasonable upper bound on a practical design is one order of magnitude more *total function evaluations* than were performed in the example. These relate directly to storage space and execution time. Of course, one could get more than one order of magnitude increase in experimental runs by decreasing the number of iterations per run. This is probably reasonable. Furthermore, one could get more factors into the design by randomization or by carrying out a fraction of the factorial runs.

6. CONCLUSIONS

It is not easy to do comprehensive algorithm testing, but based on the above discussion it appears to be possible. And it is getting easier. The information provided by a carefully designed experiment is now worth the effort required to generate the data and do the analyses. The advances in computer hardware and software that have led engineers to simulation optimization can lead us to more comprehensive testing procedures.

The days of anecdotal algorithm testing should be ending. I once attended a lecture by Horace Andrews on the design of experiments. He said that scientific knowledge was based on carefully designed experiments with many observations. "One or

two data points provide no basis for conviction..." he said. I remember asking him whether he would believe I had a serum to turn dogs blue if he saw a single application of it. I had to admit though that, outside of my imagination, I had never seen such a powerful serum.

ACKNOWLEDGEMENTS

James Lyness provided many useful suggestions during the course of this work. The author acknowledges the past support of his managers at the RCA David Sarnoff Research Center. At Cornell this work has been supported by the NSF sponsored Cornell National Supercomputer Facility and the Departments of Computer Science and Operations Research. Juan Medero developed the automatic factorial algorithm running software. Robin Barton has provided much appreciated support.

REFERENCES

Amyot, J. R. and van Blokland, G. (1985). Parameter optimization with ACSL. *Proceedings of the 1985 Summer Computer Simulation Conference*. The Society for Computer Simulation, San Diego, California, 63-68.

Barton, R. R. (1984). Minimization algorithms for functions with random noise. *American Journal of Mathematical and Management Sciences* **4**, 109-138.

Barton, R. R. (1986). A new test function for unconstrained optimization. In: *Committee on Algorithms Newsletter* , **14** , (R. Meyer and J. Telgen, eds.), The Mathematical Programming Society, August.

Biles, W. E. and Swain, J. J. (1979). Mathematical programming and the optimization of computer simulation models. In: *Mathematical Programming Study 11: Engineering Optimization* (M. Avriel and R. S. Dembo, eds.), North Holland, Amsterdam, 189-207.

Box, G. E. P., and Draper, N. R. (1969). *Evolutionary Operation* . Wiley and Sons, New York.

Box, G. E. P., Hunter, W. G., and Hunter, J. S. (1978). *Statistics for Experimenters* . Wiley and Sons, New York.

Graver, M., Timmel, G. and Pollmer, L. (1980). Comparison of some search algorithms for efficient points in polyoptimization problems. In: *Systems Analysis and Simulation* , (A. Sydow, ed.) Akademie-Verlag, Berlin, 165-169.

Hillstrom, K. E. (1977). A simulation test approach to the evaluation of nonlinear optimization algorithms. *ACM Transactions on Mathematical Software* **3**, 305-315.

Hooke, R. and Jeeves, T. A. (1961). Direct search solution of numerical and statistical problems. *Journal of the Association for Computing Machines* , **8**, 212-229.

Lee, Y. H. and Azadivar, F. (1985). An application of optimization-by-simulation to discrete variable systems. In: *Proceedings of the 1985 Winter Simulation Conference* (D. Gantz, G. Blais, and S. Solomon eds.). Institute of Electrical and Electronics Engineers, San Francisco, California, 173-177.

Lyness, J. N. (1979). A bench mark experiment for minimization algorithms. *Mathematics of Computation* , **33**, 249-264.

Moré, J. J., Garbow, B. S., and Hillstrom, K. E. (1981). Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, **7**, 17-41.

Nelder, J. A., and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal* , **7**, 308-313.

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. The Computer Journal **3**, 175-184.

Suri, R., Diehl, G. W., and Ho, Y. C. (1985). Optimization of manufacturing systems simulations using perturbation analysis and SENSE. *Proceedings of the 1985 Winter Simulation Conference* (D. Gantz, G. Blais, and S. Solomon, eds.). Institute of Electrical and Electronics Engineers, San Francisco, California, 178-184.

Velleman, P. F., and Hoaglin, D. C. (1981). *Applications, Basics, and Computing of Exploratory Data Analysis*. Duxbury Press, Boston.

Winarsky, N. D. (1987). Applied mathematics at the David Sarnoff Research Center. In: *SIAM News*, **20**, no. 4, 15.