Studies in Nonlinear Dynamics and Econometrics

Quarterly Journal
October 1996, Volume 1, Number 3
The MIT Press

Studies in Nonlinear Dynamics and Econometrics (ISSN 1081-1826) is a quarterly journal published electronically on the Internet by The MIT Press, Cambridge, Massachusetts, 02142. Subscriptions and address changes should be addressed to MIT Press Journals, 55 Hayward Street, Cambridge, MA 02142; (617)253-2889; e-mail: journals-orders@mit.edu. Subscription rates are: Individuals $40.00, Institutions $130.00. Canadians add additional 7% GST. Prices subject to change without notice.

SIMANN: A Global Optimization Algorithm using Simulated Annealing

William L. Goffe
Department of Economics and International Business
University of Southern Mississippi
Bill.Goffe@usm.edu

Abstract. *This paper describes SIMANN, a Fortran and GAUSS implementation of the simulated annealing algorithm. The Fortran code was used in "Global Optimization of Statistical Functions with Simulated Annealing" (Goffe, Ferrier, and Rogers 1994). In that paper, simulated annealing was found to be competitive, if not superior, to multiple restarts of conventional optimization routines for difficult optimization problems. This paper compares SIMANN to the DFP algorithm on another optimization problem, namely, the maximum likelihood estimation of a rational expectations model, which was previously studied in the literature. SIMANN again performs quite well, and shows several advantages over DFP. This paper also describes simulated annealing, and gives explicit directions and an example for using the included GAUSS and Fortran code.*

Keywords. optimization, algorithm, simulated annealing, numerically intensive

Acknowledgments. I would like to thank, without implicating, an anonymous referee, Efthymios G. Tsionas for translating SIMANN from Fortran to GAUSS, and Michael Veall for providing sample code and data for the Hoffman and Schmidt (1981) example. Mark Dickie provided valuable suggestions.

1 Overview

Almost all "classical" optimization methods for multivariable problems, such as the Newton-Raphson, the Davidon-Fletcher-Powell, and the simplex methods, explicitly assume that a function has one optimum, and often assume that the function is approximately quadratic. Clearly, some functions violate these assumptions, and it is likely that as more sophisticated models are explored, more such functions will be encountered. The rather common difficulties (lack of convergence, run-on executions, etc.) that many researchers experience with classical optimization methods may stem from the violation of these assumptions. Simulated annealing makes fewer assumptions than classical optimization methods. First, it is explicitly designed for functions with multiple optima. Second, it assumes very little about the "shape" of the function. Very roughly, it explores the function's entire surface, and tries to optimize the function while moving both uphill and downhill. Thus, it is much more robust than are classical algorithms. In fact, this is the best way to think of simulated annealing. This robustness comes at a cost—greater run time. However, in an era of cheap (and increasingly cheaper) computing, this substitution of computer time for trial, error, and frustration should be welcome.

Simulated annealing's roots are in thermodynamics, where one studies a system's thermal energy. A description of the cooling of molten metal motivates the concept of the algorithm. After slow cooling (annealing), the metal arrives at a low energy state. Inherent random fluctuations in energy allow the annealing system to escape local energy minima, to achieve the global minimum. But if cooled very quickly (or "quenched"), the system might not escape local energy minima and, when fully cooled, it will contain more energy than annealed metal.

Simulated annealing attempts to minimize an analogue of the energy of a system, usually minimizing a multidimensional function. In trying to reach this minimum, simulated annealing moves both uphill and downhill. As in the metal analogue, it is trying to avoid being trapped in a local energy minimum.

This implementation of simulated annealing is based on Corana et al. (1987). It was modified and tested on a variety of econometric problems, in Goffe, Ferrier, and Rogers (1994). This paper extends Goffe, Ferrier, and Rogers (1994) in several directions. First, this paper compares the performance of simulated annealing to a classical optimization algorithm (DFP) using a different, yet fairly typical econometric problem: a maximum-likelihood version of a rational expectations model. Second, it contains explicit directions for learning to use the algorithm. Finally, it includes the companion source code, written in both Fortran and GAUSS[1] (when Goffe, Ferrier, and Rogers (1994) was written, only Fortran source from the authors was available).

This paper first briefly explains the algorithm, and then demonstrates its use on a maximum-likelihood problem. The next section contains a set of directions for learning to use SIMANN with a sample problem that comes with the program, and the final section describes the algorithm's parameters.

2 Explanation of the Algorithm

At the start, the algorithm randomly chooses a trial point within the step-length `V` (a vector of length n) of the user-selected starting point. The function is evaluated at this trial point, and is compared to its value at the initial point. In a maximization problem, all uphill moves are accepted, and the algorithm continues from an accepted trial point (note the step length is always centered on the trial point and not 0). Downhill moves may be accepted; the decision is made by the Metropolis criteria, which uses `T` (temperature), and the magnitude of the downhill move in a probabilistic manner. The higher the temperature and the smaller the downhill move, the more likely that the move will be accepted. If the trial point is rejected, another point is chosen for a trial evaluation.

Periodically, each element of `V`, the step length, is adjusted so that half of all function evaluations in that direction are accepted. A fall in temperature is imposed upon the system with the `rT` variable by $\texttt{T}_{\texttt{i+1}} = \texttt{rT} \cdot \texttt{T}_{\texttt{i}}$, where i is the i^{th} iteration. As the temperature declines, downhill moves are less likely to be accepted, and the percentage of rejections rises. Given the scheme for the selection of the step length, it falls as a result. Thus, as temperature declines, the step length falls, and simulated annealing focuses upon the most promising area for optimization.

3 Sample Problem

To compare the performance of SIMANN and a classical optimization algorithm, a rational expectations model estimated with maximum likelihood introduced in Hoffman and Schmidt (1981) and later explored in Veall (1990) was used.[2] The model's unknown parameters are γ, β_1, β_2, λ_1, λ_2, σ_0^2, σ_1^2, and σ_2^2, and the model is described by

$$Y_t = \gamma E_{t-1} Y_t + \beta_1 X_{1t} + \beta_2 X_{2t} + \epsilon_{0t}, \ |\gamma| < 1 \tag{1}$$

$$X_{1t} = \lambda_1 X_{1t-1} + \epsilon_{1t} \tag{2}$$

$$X_{2t} = \lambda_2 X_{2t-1} + \epsilon_{2t} \tag{3}$$

$$\epsilon_{it} \sim i.i.d.\ N(0, \sigma_i^2) \tag{4}$$

$$Y_t = \beta_1 X_{1t} + \beta_2 X_{2t} + (\gamma/(1-\gamma))(\beta_1\lambda_1 X_{1t-1} + \beta_2\lambda_2 X_{2t-1}) + \epsilon_{0t} \tag{5}$$

Equation (5) comes from the preceding equations, and Equations (2), (3), and (5) are estimated by maximum likelihood over the unknown parameters. The unconcentrated version of the likelihood function was employed.

The widely used Davidon–Fletcher–Powell (DFP) algorithm is used as the classical comparison to simulated annealing. Since no freely available version seems to exist (a quick check of the major on-line numerical

[1] Efthymios G. Tsionas kindly translated the Fortran to GAUSS.

[2] Michael Veall kindly supplied the code and the synthetic data used in that study.

library, Netlib, at *http://www.netlib.org/*, revealed none), the easily available and inexpensive version of DFP, dfpmin, from Press et al. (1992a) was employed. This is available on diskette from Press et al. (1992b), and can even be purchased on-line at *http://nr.harvard.edu/numerical-recipes*. Numerical derivatives were employed.

From Veall (1990), the global optimum is known. However, to simulate the researcher's uncertainty on typical problems, the starting values for the model's unknown parameters were randomly selected. Since different researchers might use different magnitudes for their starting values, different ranges for the randomly selected starting values centered on zero were employed: within ± 2.0, ± 4.0, ± 10.0, and ± 25.0. The results for 100 runs with randomly selected starting values from each of these ranges is shown in Table 1. The true optimum has a value of approximately −66.855. Runs that took more than 2,000 function evaluations were categorized as failures, as the longest otherwise-successful run took approximately 500 function evaluations.

Table 1
DFP Results

Starting Range	Correct Optimum	Other Optimum	Failed	Run Time (seconds)
± 2.0	57	25	18	13
± 4.0	36	41	23	17
± 10.0	9	41	50	27
± 25.0	0	10	90	43

An IBM RS/6000 model 550, a moderate-performance machine, was used in this work. It performs the Linpack 100 × 100 benchmark (inverting a 100 × 100 matrix) at 26 Mflops; a Power Mac 8500/120 does the same benchmark at 20 Mflops, and a Gateway 2000 133 MHz Pentium generated 19 Mflops. For a comparison, a Cray 1S in 1983 turned in 12 Mflops, and an IBM-PC with an 8087 math coprocessor brings up the rear at .0069 Mflop. The current record for a single processor is 576 Mflops for a Cray T94. See Dongarra (1996) for additional results and details on the Linpack benchmark.

As can be seen, the run times (measured by user time on this UNIX machine) bordered on the trivial. Unfortunately, the researcher would likely have had a difficult time quickly determining the correct optimum unless he was using multiple runs (in effect, a crude method of global optimization). Only with small starting values close to zero do even a bare majority of the runs reach the correct optimum. When the starting values are selected from the largest set, not a single one of the 100 runs found the correct optimum. Clearly, DFP is not a very good algorithm for estimating this model.

After several trial runs (described below), a single run of SIMANN correctly found the optimum with a run time of 14 s. As described above, the algorithm moves both uphill and downhill, so it didn't get "stuck" at any of the spurious points that hindered DFP. In addition, the run time is nearly trivial. The specifics are shown in Tables 2 and 3. The first table shows the major inputs to the algorithm (less important ones are described in the documentation that accompanies the program), and the second is the output; the intermediate output, which yields a variety of useful statistics, is not shown. The contents of both tables have been slightly edited for convenience.

Table 2
Inputs to SIMANN

```
INITIAL TEMP: .10E+03  RT: .75
EPS: .10E-05  NS: 20  NT: 5  NEPS: 4

                        STARTING VALUES
0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000

                      INITIAL STEP LENGTH
10.00   10.00   10.00   10.00   10.00   10.00   10.00   10.00

                           LOWER BOUND
-.10E+26 -.10E+26 -.10E+26 -.10E+26 -25.0 -25.0 -25.0 -25.0

                           UPPER BOUND
 .10E+26  .10E+26  .10E+26  .10E+26  25.0  25.0  25.0  25.0
```

The initial temperature of T = 100 was chosen after several trial runs; it ensures that the step-length V is large enough to cover the region to be optimized. At this temperature, the smallest step length (other than for

the constrained model parameters, discussed below) was about 13,000. Clearly, one could be more conservative, but, with the short run time, it is hardly worth the effort. The variable $\mathtt{rT}$, the factor by which temperature is reduced at each step, was set to .75. This is a rather conservative value, which is suitable for a function one has little experience with. A value of 10^{-6} for ϵ, and 4 for $\mathtt{N}_\epsilon$, means that the algorithm will terminate after four temperature reductions where the difference in final function values at each temperature and the optimal function value is less than 10^{-6}. The variable $\mathtt{NS}$ says that after $\mathtt{NS} \cdot \mathtt{n}$ function evaluations, the step-length $\mathtt{V}$ is adjusted so that approximately half of all function evaluations are accepted. The value of 20 should generally be used. The variable $\mathtt{NT}$ says that after $\mathtt{NT} \cdot \mathtt{NS} \cdot \mathtt{n}$ function evaluations, the temperature is reduced by the factor $\mathtt{rT}$. A value of 5 is relatively small, but is appropriate for this fairly simple function. To vary the run time and robustness of the algorithm, the values for $\mathtt{rT}$ and $\mathtt{NT}$ should be changed.

The starting values for the unknown parameters of the model are set to zero; they matter little with SIMANN, since it will subsequently make both uphill and downhill moves from this point. The initial step-length $\mathtt{V}$ also matters little, as it rapidly adjusts so that at the current temperature, about half of all moves are accepted. Finally, the upper and lower bounds, which restrict the search space, are set. Note that the first five values of both are quite large and basically imply no restriction, but the final three imply a width of 50 when searching for the last three of the model's unknown parameters. The reason is described below.

Table 3
Output from SIMANN

```
 SA ACHIEVED TERMINATION CRITERIA. IER = 0.

 ****   RESULTS AFTER SA   ****

                        SOLUTION
.76715 .50097 2.7918 5.2857 -.35397 -3.9846 -9.3502 -7.3199

                        FINAL STEP LENGTH
.134E-03 .181E-03 .223E-02 .283E-02 .115E-02 50.0 50.0 50.0

OPTIMAL FUNCTION VALUE:  -66.85595
NUMBER OF FUNCTION EVALUATIONS:          56801
NUMBER OF ACCEPTED EVALUATIONS:          39259
NUMBER OF OUT OF BOUND EVALUATIONS:      11428
FINAL TEMP:   .1795925998138E-06  IER:   0
```

SIMANN denotes a successful termination with $\mathtt{IER} = 0$. Next, it reports its solution to the model's parameters and, then, the final step length, $\mathtt{V}$. Note that the first five are relatively small, but the final three (for the model's variances) are the width of the bounds given at the start. In several previous runs of SIMANN, these step lengths ballooned to very large values even as the temperature fell. This indicates that the likelihood function is extremely flat in these three variables: SIMANN is continually increasing the step length in search of one where only half of all evaluations will be accepted. This flatness is not apparent with DFP, except when different solutions are examined (one sees an identical function value, but different values for the variances). As this flatness was easily spotted with the first runs of SIMANN, the parameter space was limited to the range $\pm$ 25.0 in the variances in the hope that a very shallow minimum might be found with more emphasis on the likely values.

Next, the optimal function value is reported and, then, some statistics on the number and types of function evaluations. When the search space is not constrained, about half of all function evaluations will be accepted.

Finally, to ensure that the global optimum has been achieved, it is a good idea to rerun SIMANN with different seeds to its random-number generator. If the same optimum is reached after an entirely different set of trial points and decisions on downhill moves, one can be fairly certain that the global optimum has indeed been reached.

SIMANN showed its superiority to DFP twice. First, it easily found the optimum; many runs with DFP never found it. On modern computers, the run time was hardly long, and certainly substitutes quite well for the researcher's time. Second, SIMANN clearly informed the user of an important feature of the likelihood function—its extreme flatness in the variance parameters.

4 Learning to Use SIMANN

It is highly recommended that potential users first gain experience with the example enclosed with the code. In a short time, you will build up sufficient human capital to use the algorithm on difficult problems. The example, from Judge et al. (1985), is a small two-dimensional function with two optima.

1. Run the program as is, to make sure it operates correctly. Look at the intermediate output to see how the algorithm optimizes as temperature falls. Notice how the optimal point is reached, and how falling temperature reduces the step length.

2. To see how the algorithm selects points and makes decisions about uphill and downhill moves, set `iprint = 3` (very detailed intermediate output) and `maxevl` = 100 (use 100 function evaluations to limit output).

3. To see the importance of different temperatures, try starting with a very low one (say `T` = 10^{-5}). You will see: (i) the algorithm never escapes from the local optima (in annealing terminology, it quenches), and (ii) the step length will be quite small. This is a key part of the algorithm: as temperature falls, so does step length. In a minor point, note how the step length is quickly reset from its initial value. Thus, the input step length is not very important.

4. To see the effects of different parameters and their effects on the speed of the algorithm, try $r_T = .95$ and $r_T = .1$. Note the vastly different speed for optimization. Also try $N_S = 20$. This sample function is quite easy to optimize, so it will tolerate big changes in these parameters. To adjust the run time of the algorithm and its robustness, the values of r_T and N_T should be modified.

5. Try constraining the search space of the algorithm with either or both of the upper and lower bounds, `lb` and `ub`.

5 Algorithm Parameters

The previous sections described the algorithm with some generality. This section defines each of the inputs, outputs, etc. in considerable detail for the Fortran version (the GAUSS version differs slightly in several places). In several instances, the notation in this section is slightly different than the previous sections. Integer variables are denoted by I, logical variables by L, double-precision variables by DP, and vectors of length N by (N) following the type of variable.

5.1 Input Parameters

N—The number of variables in the function to be optimized. (I)

X—The starting values for the variables of the function to be optimized. (DP(N))

MAX—Denotes whether the function should be maximized or minimized. A true value denotes maximization, while a false value denotes minimization. Intermediate output (see IPRINT) takes this into account. (L)

RT—The temperature-reduction factor. The value suggested by Corana et al. (1987) is .85, but it is quite adjustable to suit the problem at hand. (DP)

EPS—The error tolerance for termination. If the final function values from the last NEPS temperatures differ from the corresponding value at the current temperature by less than EPS, and the final function value at the current temperature differs from the current optimal function value by less than EPS, execution terminates and IER = 0 is returned. (DP)

NS—The number of cycles. After NS·N function evaluations, each element of VM is adjusted so that approximately half of all function evaluations are accepted. The suggested value is 20. (I)

NT—The number of iterations before temperature reduction. After NT·NS·N function evaluations, temperature (T) is changed by the factor RT. Corana et al. (1987) suggest max(100, 5·N), but it is quite adjustable to suit the problem at hand. (I)

NEPS—The number of final function values used to decide upon termination. See EPS. The suggested value is 4. (I)

MAXEVL—The maximum number of function evaluations. If it is exceeded, IER = 1. (I)

LB—The lower bound for the allowable solution variables. (DP(N))

UB—The upper bound for the allowable solution variables. If the algorithm chooses X(I) < LB(I) or X(I) > UB(I), I = 1, N, a point from inside is randomly selected. This focuses the algorithm on the region inside UB and LB. Unless the user wishes to concentrate the search to a particular region, UB and LB should be set to very large positive and negative values, respectively. Note that the starting vector X should be inside this region. Also note that LB and UB are fixed in position, while VM is centered on the last accepted trial set of variables that optimizes the function. (DP(N))

C—The vector that controls the step-length adjustment. The suggested value for all elements is 2.0. (DP(N))

IPRINT—Controls printing inside SA. (I)
Values:

0 Nothing printed.

1 Function value for the starting value, and summary results before each temperature reduction. This includes the optimal function value found so far, the total number of moves (broken up into uphill, downhill, accepted, and rejected), the number of out-of-bounds trials, the number of new optima found at this temperature, the current optimal X, and the step-length VM. Note that there are N·NS·NT function evaluations before each temperature reduction. Finally, notice is also given upon achieving the termination criteria.

2 Each new step length (VM), the current optimal X (XOPT), and the current trial X (X). This gives the user some idea about how far X strays from XOPT, as well as how VM is adapting to the function.

3 Each function evaluation, its acceptance or rejection, and new optima. For many problems, this option will likely require a small tree if hard copy is used. This option is best used to learn about the algorithm. A small value for MAXEVL is thus recommended when using IPRINT = 3.

Suggested value: 1.
Note: For a given value of IPRINT, the lower valued options (other than 0) are utilized.

ISEED1—The first seed for the random-number generator RANMAR. 0 < ISEED1 < 31328. (I)

ISEED2—The second seed for the random-number generator RANMAR. 0 < ISEED2 < 30081. Different values for ISEED1 and ISEED2 will lead to an entirely different sequence of trial points and decisions on downhill moves (when maximizing). This can be used to test the results of SA. (I)

5.2 Input/Output Parameters

T—On input, the initial temperature. On output, the final temperature. (DP)

VM—The step-length vector. On input it should encompass the region of interest given the starting value X. For point X(I), the next trial point is selected from X(I) − VM(I) to X(I) + VM(I). Since VM is adjusted so that about half of all points are accepted, the input value is not very important (i.e., if the value is off, SA adjusts VM to the correct value). (DP(N))

5.3 Output Parameters

XOPT—The variables that optimize the function. (DP(N))

FOPT—The optimal value of the function. (DP)

NACC—The number of accepted function evaluations. (I)

NFCNEV—The total number of function evaluations. In a minor point, note that the first evaluation is not used in the core of the algorithm; it simply initializes the algorithm. (I).

NOBDS—The total number of trial-function evaluations that would have been out-of-bounds of LB and UB. Note that a trial point is randomly selected between LB and UB. (I)

IER—The error return number. (I)
Values:

0 Normal return; termination criteria achieved.

1 Number of function evaluations (NFCNEV) is greater than the maximum number (MAXEVL).

2 The starting value (X) is not inside the bounds (LB and UB).

3 The initial temperature is not positive.

99 Should not be seen; only used internally.

5.4 Work Arrays

Each must be dimensioned in the calling routine.
RWK1 (DP(NEPS)) (FSTAR in SA)
RWK2 (DP(N)) (XP in SA)
IWK (INT(N)) (NACP in SA)

5.5 Required Functions

EXPREP—Replaces the function EXP to avoid underflows and overflows. It may have to be modified for non-IBM-type mainframes. (DP)

RMARIN—Initializes the random-number generator RANMAR.

RMARAR—Must run first (SA does this). It produces uniform random numbers on [0,1]. See Marsaglia and Zaman (1987) for the original routine, and James (1990) for the modification used here.

Each of these functions are included with the code.

5.6 Required Subroutines

PRTVEC—Prints vectors.

PRT1 . . . PRT10—Prints intermediate output.

FCN—Function to be optimized. The form is

```
SUBROUTINE FCN(N,X,F)
INTEGER N
DOUBLE PRECISION  X(N), F
...
function code with F = F(X)
...
RETURN
END
```

Note: This is the same form used in the multivariable minimization algorithms in the IMSL edition 10 library.

Each of these subroutines, with the exception of FCN, are included.

5.7 Machine-Specific Features

- EXPREP may have to be modified for some machines. Watch for underflows and overflows in EXPREP.
- Some FORMAT statements use G25.18; this may be excessive for some machines.
- RMARIN and RANMAR are designed to be portable; they should not cause any problems.

References

Corana, A., M. Marchesi, C. Martini, and S. Ridella (1987). "Minimizing Multimodal Functions of Continuous Variables with the 'Simulated Annealing' Algorithm." *ACM Transactions on Mathematical Software*, 13(3):262–280.

Dongarra, Jack J. (1996). "Performance of Various Computers using Standard Linear Equations Software." Technical Report CS-89-95. Computer Science Department, University of Tennessee, Knoxville. Available at http://*www.netlib.org/benchmark/performance.ps*.

Goffe, William L., Gary D. Ferrier, and John Rogers (1994). "Global Optimization of Statistical Functions with Simulated Annealing." *Journal of Econometrics*, 60(1/2):65–99.

Hoffman, Dennis L., and Peter Schmidt (1981). "Testing the Restrictions Implied by the Rational Expectations Hypothesis." *Journal of Econometrics*, 15(2):265–287.

James, Frederick (1990). "A Review of Pseudorandom Number Generators." *Computer Physics Communications*, 60(3):329–344.

Judge, George G., W. E. Griffiths, R. Carter Hill, Helmut Lutkepoh, and Tsoung-Chao Lee (1985). *The Theory and Practice of Econometrics*, 2^{nd} ed. New York: Wiley, pp. 956–957.

Marsaglia, George, and Arif Zaman (1987). "Toward a Universal Random Number Generator." Technical Report FSU-SCRI-87-50, Florida State University.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery (1992a). *Numerical Recipes in FORTRAN, The Art of Scientific Computing*, 2^{nd} ed. New York: Cambridge University Press.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery (1992b). *Numerical Recipes in FORTRAN, The Art of Scientific Computing, 2^{nd} ed., Diskette V2.0*. New York: Cambridge University Press.

Veall, Michael (1990). "Testing for a Global Maximum in an Econometric Context." *Econometrica*, 58(6):1459–1465.