Journal of Econometrics 60 (1994) 65–99. North-Holland

Global optimization of statistical functions with simulated annealing*

William L. Goffe
University of Southern Missisipi, Hattiesburg, MS 39406, USA

Gary D. Ferrier
University of Arkansas, Fayetteville, AR 72701, USA

John Rogers
Memphis State University, Memphis, TN 38152, USA

Received November 1990, final version received June 1992

Many statistical methods rely on numerical optimization to estimate a model's parameters. Unfortunately, conventional algorithms sometimes fail. Even when they do converge, there is no assurance that they have found the global, rather than a local, optimum. We test a new optimization algorithm, simulated annealing, on four econometric problems and compare it to three common conventional algorithms. Not only can simulated annealing find the global optimum, it is also less likely to fail on difficult functions because it is a very robust algorithm. The promise of simulated annealing is demonstrated on the four econometric problems.

Key words: Simulated annealing; Global optimization; Estimation algorithms

Correspondence to: William L. Goffe, Department of Economics and International Business, University of Southern Missisipi, Hattiesburg, MS 39406, USA.

* We would like to thank, without implicating, Hans Amman, Gary Barinshtein, David Belsley, T. Randolph Beard, Herman Bierens, Michael Booty, David Goldberg, Stan Kerr, David Levy, Essie Maasoumi, Daniel McCaffrey, Richard Quandt, and Michael Veall for helpful suggestions and comments on previous drafts. Two anonymous referees provided many valuable suggestions. Albert Boulanger was very helpful with the literature survey. For computational help, we would like to thank Art Belmonte and Ava Horvath of IMSL. Jim Hall and Bill Courtney of IBM kindly provided a sample workstation. Sarah Barron, Jan Byrd, Robert Harkness, Luther Keeler, John Respess, and Spiros Vellas of the University of Texas Center for High Performance Computing also provided invaluable help.

The computational work for this paper was performed at the University of Texas Center for High Performance Computing.

1. Introduction

Many econometric methods, such as nonlinear least squares, the generalized method of moments, and the maximum likelihood method, rely upon optimization to estimate model parameters. Not only do these methods lay the theoretical foundation for many estimators, they are often used to numerically estimate models. Unfortunately, this may be difficult because these algorithms are sometimes touchy and may fail. For the maximum likelihood method, Cramer (1986, p. 77) lists a number of 'unpleasant possibilities': the algorithm may not converge in a reasonable number of steps, it may head toward infinitely large parameter values, or even loop through the same point time and again. Also, the algorithm may have difficulty with ridges and plateaus. When faced with such difficulties, the researcher is often reduced to trying different starting values [Cramer (1986, p. 72) and Finch et al. (1989)]. Finally, even if the algorithm converges, there is no assurance that it will have converged to a global, rather than a local, optimum since conventional algorithms cannot distinguish between the two. In sum, there is a poor match between the power of these methods and the numerical algorithms used to implement them.

A new and very different optimization algorithm, simulated annealing, potentially improves this match. It explores the function's entire surface and tries to optimize the function while moving both uphill and downhill. Thus, it is largely independent of the starting values, often a critical input in conventional algorithms. Further, it can escape from local optima and go on to find the global optimum by the uphill and downhill moves. Simulated annealing also makes less stringent assumptions regarding the function than do conventional algorithms (it need not even be continuous). Because of the relaxed assumptions, it can more easily deal with functions that have ridges and plateaus. Finally, it can optimize functions that are not defined for some parameter values.

This paper first compares the Corana et al. (1987) implementation of simulated annealing to conventional algorithms on four econometric models. Thus, these results may help guide other researchers. The first model is an example from the literature with multiple minima [Judge et al. (1985, pp. 956–957)], and the second is a rational expectations exchange rate model. The third model is an efficiency study of the banking industry using a translog cost frontier system and the fourth fits a neural network to a chaotic time series. These models vary greatly in size. The Judge model has only two parameters, the exchange rate model has 14, the neural network has 35, and the cost study has 62. All three multivariable optimization algorithms in the IMSL library represent the conventional algorithms. Since they are sensitive to starting values, 100 different sets of them are used in all cases.

This paper then introduces extensions to the Corana algorithm. One modification checks that the global optimum is indeed achieved, while another allows the researcher to restrict optimization to a subset of the parameter space (this can be very useful for understanding the function). The third extension allows the researcher to determine a critical initial parameter for the algorithm, while the final one directs the selection of parameters that control the robustness of the algorithm. This allows the researcher to minimize the execution time of the algorithm.

We find simulated annealing to be quite powerful. On the Judge function, simulated annealing always finds the global optimum, while the IMSL routines do not. For the rational expectations model, the conventional algorithms cannot optimize and do not offer any reason for their failure. When used as a diagnostic tool, simulated annealing determines the problem. Even when this problem is eliminated, the conventional algorithms only rarely find the optimum, while simulated annealing does so easily. The conventional algorithms cannot optimize the translog cost frontier model while simulated annealing does so easily. For the neural network, simulated annealing finds a much better optimum than any of the conventional algorithms. However, it is shown that the nature of this function makes it virtually impossible for any method to find the global optimum.

Essentially, simulated annealing provides more features at the cost of an increase in execution time for a single run of the algorithm. However, when compared to the multiple runs often used with conventional algorithms to test different starting values, it is competitive. In some cases, simulated annealing is even competitive with a single run of a conventional algorithm. As shown in appendix A, computer power is increasing rapidly and the resources needed for simulated annealing will soon, if not already, be available to most researchers for even the most difficult problems.

This paper is organized as follows. Section 2 describes simulated annealing in detail and compares it to conventional algorithms, while section 3 compares its performance on the four test problems to the IMSL algorithms. This section also introduces many extensions to simulated annealing. Section 4 briefly reviews these extensions and introduces another. A conclusion summarizes the strengths and weaknesses of simulated annealing. An appendix examines another global optimization algorithm, the genetic algorithm, and finds that simulated annealing is superior.

2. The algorithms

2.1. Conventional algorithms and their limitations

Conventional optimization algorithms, such as Newton–Raphson and Davidon–Fletcher–Powell, try to move uphill (maximization is assumed in the general discussion of the algorithms) in an iterative manner. In very general terms, starting from a point, they determine the best direction and step length to head uphill. After moving there, this process is repeated until some stopping criteria is achieved. Statistical packages such as SAS, TSP, and RATS use these algorithms. General reviews on them can be found in either Judge et al. (1985), Press et al. (1986), or Quandt (1983), and a more rigorous analysis is in Dennis and Schnabel (1983). Their operation is similar to a blind man walking up a hill, whose only knowledge of the hill comes from what passes under his feet. If the hill is predictable in some fashion, he will reach the top, but it is easy to imagine confounding terrain. Many conventional algorithms make the function predictable by assuming the function is approximately quadratic. Unfortunately, statistical functions may not be approximately quadratic. Implicit here is another assumption common to conventional algorithms: the function has one optimum, so that any local optimum is also the global optimum. While Kendall and Stuart (1978, p. 44) show asymptotically that likelihood functions have one optimum, the small sample properties are unknown. Taken together, it is easy to see why conventional algorithms may have difficulty finding the global maximum. As the next section shows, simulated annealing takes a very different approach to optimizing functions that involves significantly fewer limitations.

2.2. *Simulated annealing for continuous variable problems*

Simulated annealing's roots are in thermodynamics, where one studies a system's thermal energy. A description of the cooling of molten metal motivates this algorithm. After slow cooling (annealing), the metal arrives at a low energy state. Inherent random fluctuations in energy allows the annealing system to escape local energy minima to achieve the global minimum. But if cooled very quickly (or 'quenched'), it might not escape local energy minima and when fully cooled it may contain more energy than annealed metal. Simulated annealing attempts to minimize some analogue of energy in a manner similar to annealing to find the global minimum. Details can be found in Press et al. (1986, pp. 326–334).

Early simulated annealing algorithms considered combinatorial systems, where the system's state depends on the configuration of variables. Perhaps the best known is the traveling salesman problem, in which one tries to find the minimum trip distance connecting a number of cities. Combinatorial simulated annealing has been used successfully in computer and circuit design [Kirkpatrick et al. (1983) and Wong et al. (1988)], pollution control [Derwent (1988)], a special case of 0–1 programming [Drexl (1988)], neural networks [Wasserman and Schwartz (1988)], reconstruction of pollycrystalline structures [Telly et al. (1987)], and image processing [Carnevali et al. (1985)].

Other global optimization algorithms have been introduced in recent years. They include adaptive random search [Pronzato et al. (1984)], genetic algorithms [Goldberg (1989)], the filled function method [Renpu (1990)], multilevel methods [Kan et al. (1987)], and a method using stochastic differential equations [Aluffi-Pentini et al. (1988)]. Both Vanderbilt and Louie (1984) and Bohachevsky et al. (1986) have modified simulated annealing for continuous variable problems.

However, the Corana et al. implementation of simulated annealing for continuous variable problems appears to offer the best combination of ease of use and robustness, so it is used in this study [an early and very incomplete test can be found in Goffe et al. (1992)]. While a complete description can be found there, a summary follows and the algorithm in pseudo-computer code is provided in appendix B. The Fortran code of the algorithm is available from the authors and in Richard Quandt's GQOPT6 statistical optimization package. The essential starting parameters to maximize the function $f(X)$ are T_0, the initial temperature; X, the starting vector of parameters; and V, the step length for X. Note that X and V are both vectors of length n, the number of parameters of the model. Upper case refers to vectors and lower case to scalars (with the exception of temperature, T).

A function evaluation is made at the starting point X and its value f is recorded. Next, a new X, X', is chosen by varying element i of X,

$$x'_i = x_i + r \cdot v_i, \tag{1}$$

where r is a uniformly distributed random number from $[-1, 1]$ and v_i is element i of V. The function value f' is then computed. If f' is greater than f, X' is accepted, X is set to X', and the algorithm moves uphill. If this is the largest f, it and X are recorded since this is the best current value of the optimum.

If f' is less than or equal to f, the Metropolis criterion decides on acceptance (thermodynamics motivates this criterion). The value

$$p = \mathrm{e}^{(f'-f)/T} \tag{2}$$

is computed and compared to p', a uniformly distributed random number from $[0, 1]$. If p is greater than p', the new point is accepted, X is updated with X', and the algorithm moves downhill. Otherwise, X' is rejected. Two factors decrease the probability of a downhill move: lower temperatures and larger differences in the function's value. Also note that the decision on downhill moves contains a random element.

After N_s steps through all elements of X (all such 'N' parameters are set by the user), the step length vector V is adjusted so that 50% of all moves are accepted. The goal here is to sample the function widely. If a greater percentage of points are accepted for x_i, then the relevant element of V is enlarged. For a given temperature, this increases the number of rejections and decreases the percentage of acceptances. After N_T times through the above loops, the temperature, T, is reduced. The new temperature is given by

$$T' = r_T \cdot T, \tag{3}$$

where r_T is between 0 and 1. A lower temperature makes a given downhill move less likely, so the number of rejections increase and the step lengths decline. In addition, the first point tried at the new temperature is the current optimum. The smaller steps and starting at the current optimum focuses attention on the most promising area.

The algorithm ends by comparing the last N_ε values of the largest function values from the end of each temperature reduction with the most recent one and the optimum function value. If all these differences are less than ε, the algorithm terminates. This criterion helps ensure that the global maximum is reached.

Note that simulated annealing first builds up a rough view of the surface by moving with large step lengths. As the temperature falls and the step length decreases, it slowly focuses on the most promising area. All the while, the algorithm can escape from local maxima through downhill moves. Eventually, the algorithm should converge to the function's global maximum.

Simulated annealing has several potential advantages over conventional algorithms. First, it can escape from local maxima. In thermodynamic terms, while conventional algorithms quench by simply heading up the current hill without regard to others, simulated annealing moves both uphill and downhill. Also, the function need not be approximately quadratic. In fact, it need not even be differentiable [Corana et al. (1987) successfully demonstrate simulated annealing on a parabolic function punctured with holes]. Another benefit is that the step length gives the researcher valuable information about the function. If an element of V is very large, the function is very flat in that parameter. Since it is determined by function evaluations at many points, it is a better overall measure of flatness than a gradient evaluation at a single point. Finally, simulated annealing can identify corner solutions because it can 'snuggle' up to a corner for functions that don't exist in some region. The most important advantage of simulated annealing is that it can maximize functions that are otherwise difficult or impossible to optimize. This is demonstrated in the next section with the test problems.

The sole drawback to simulated annealing is the required computational power, but this problem is disappearing or has disappeared. Appendix A describes the continuing improvements occurring in high performance computing. Briefly, the power of the first Cray supercomputer can now easily be put on a desktop, and this trend shows no sign of slowing. Thus, simulated annealing is an attractive option for difficult functions.

3. Simulated annealing compared to conventional algorithms

3.1. Comparison framework

The previous section demonstrated that simulated annealing has promise for optimizing statistical functions. To see if this promise holds, simulated annealing was compared to three conventional algorithms on four different statistical problems. The purpose here is to see if simulated annealing works on these problems, while section 4 discusses improving the algorithm.

All three multivariate optimization algorithms from the IMSL Math/Library Edition 10 were chosen for comparison with simulated annealing. This library was chosen because of its quality and availability. The routines are UMPOL, a simplex algorithm, UMCGF, a conjugate gradient algorithm with numerical derivatives, and UMINF, a quasi-Newton algorithm with numerical derivatives. The numerical derivative versions of UMCGF and UMINF were chosen because computing the analytical derivatives is very difficult since two of the four functions were quite complex (the simplex algorithm does not use derivatives).

Since conventional algorithms are sensitive to starting values, each single test used 100 different randomly selected starting values. This number gives the algorithm a good chance of finding the optimum. Plus, the typical researcher is unlikely to use more. The range used to select them should not be terribly critical since so many are used. With simulated annealing, 100 starting values were used in one problem, but computer time constraints prevented using this approach for the other three. In these cases, the results were verified in another way.

3.2. Test functions

Judge et al. (1985, pp. 956–957) provide the first test function. This nonlinear least squares problem is written in minimization form and has two local minima. It has two parameters and twenty observations. While small and artificial, it serves as a convenient test from the literature for optimization algorithms because of the two local minima. As will be seen, conventional algorithms cannot by themselves distinguish between the two, while simulated annealing has no difficulty. Note that it is optimized in minimization form.

The second function derives from the rational expectations version of the monetary theory of exchange rate determination as presented by Woo (1985). The model consists of the following structural relationships:

$$(m_t - m_t^*) - (p_t - p_t^*) = \gamma + (\varphi y_t - \varphi^* y_t^*) + \alpha(m_{t-1} - m_{t-1}^*) - \beta(r_t - r_t^*)$$

$$- \alpha(p_{t-1} - p_{t-1}^*) + (\varepsilon_t - \varepsilon_t^*),$$

$$p_t = p_t^* + e_t,$$

$$r_t = r_t^* + \mathrm{E}[(e_{t+1} - e_t) \mid I_t],$$

where m, p, e, and y are the log levels of the money stock, price level, exchange rate, and output, respectively, and where star superscripts denote variables in the domestic economy. These equations represent the standard stock-adjustment money demand equations when the interest rate elasticities and adjustment coefficients are the same in both countries. Purchasing power parity and uncovered interest parity hold. The stochastic specification of the model is completed by assuming that the exogenous variables of the system, $M_t \equiv m_t - m_t^*$, y_t, and y_t^* evolve according to a stable AR(1) representation given by

$$X_t = BX_{t-1} + V_t,$$

where $X_t \equiv [M_t y_t y_t^*]'$ and where B is a 3×3 parameter matrix such that the roots of $\det(I - BL)$ lie within the unit circle.

The solution and estimation procedure follows that outlined by Salemi (1986), which employs the stable vector ARMA representation for the variables of the system implied by the model above. To derive this representation, let $Z_t = [e_t X_t']'$, and write the model above as

$$\Sigma_0 + \Sigma_1 Z_t + \Sigma_2 Z_{t-1} + \Theta \mathrm{E}[Z_{t+1}|I_t] = U_t,$$

where Σ_0, Σ_1, Σ_2, and Θ are 5×4 parameter matrices containing the structural parameters α, β, γ, φ, and φ^*, as well as the nine elements of B, and where U_t is the vector of structural disturbances. A solution of the form

$$Z_t = \Sigma_0 + A(L)U_t = \Sigma_0 + [A_0 + A_1 L + A_2 L^2 + \ldots]U_t$$

then results in the stable ARMA representation given by

$$\Sigma_0 + [\Theta + \Sigma_1 L + \Sigma_2 L^2]Z_t = [A_0 \Theta + IL]U_t,$$

where the elements of A_0 have been restricted so as to analytically cancel the unstable root of the AR polynomial. As indicated by Salemi, the estimates of the parameters of this model may then be obtained by minimization of

$$f(\alpha,\beta,\gamma,\varphi,\varphi^*,b_{11},\ldots,b_{33}) = \sum_{i=1}^{T} [\hat{U}_t \hat{U}_t'],$$

with respect to the five system parameters (α, β, γ, φ, φ^*) and the nine AR parameters ($b_{11},\ldots,b_{33}$), subject to the complex nonlinear restrictions imposed across these parameters in the elements of A_0 [the transition from a maximum likelihood model to a model based upon the minimization of a sum of squares is described by Tunnicliffe-Wilson (1973)].

The third problem is from a study of firm production efficiency based on a system of a frontier cost function and its input share equations [see Ferrier and Lovell (1990)]. Three error terms on the cost equation capture the effects of two types of inefficiency and random error. The error terms on the share equations are linked to one of the error terms on the cost equation (errors in input share selection result in inefficiency, thus raising costs). After making assumptions concerning the various disturbance terms, estimation of the system is carried out by minimizing a function derived from the likelihood function. The estimation is carried out on real world data on U.S. depository institutions and involves 62 parameters and 575 observations.

The fourth test function comes from the neural network literature. While neural nets have many economic applications [see Baum (1988) and Kuan and White (1991) for surveys], the use here follows Gallant and White (1989). They use a neural net to approximate an unknown function, $g(x)$, given by

$$g_K(x|\beta, \gamma) = \sum_{j=1}^{K} \beta_j \cdot G\left(\sum_{i=1}^{r} x_i \cdot \gamma_{i,j} + \gamma_{0,j}\right).$$

Neural networks are organized in layers, from the input layer to the output layer. In this neural net, the x vector of length r is the input layer. The γ_j vectors, $j = 1, \ldots, K$, are weights to the next layer (note the intercept or bias term in each vector). This layer is 'hidden' – only the input and output layers are 'seen'. Since there is only one hidden layer, this is a single hidden layer neural network. The function G is the hidden unit activation function (a logistic in Gallant and White) and β, a vector of length K, is the hidden to output layer weights.

Gallant and White propose to use the neural net to approximate the unknown function that generates a chaotic series. To test this approach, a chaotic series of length n is generated by a discrete version of the Mackey–Glass equation:

$$g(x_{t-5}, x_{t-1}) = x_{t-1} + (10.5) \cdot \left[\frac{0.2 \cdot x_{t-5}}{1 + (x_{t-5})^{10}} - 0.1 \cdot x_{t-1}\right].$$

This is a useful generator of data for the study of chaotic economic and financial series as it generates data similar to those found in financial markets.

The β and the γ_j vectors are chosen to minimize

$$\sum_{t=6}^{n} [x_t - g_K(x_{t-1}, \ldots, x_{t-5})]^2.$$

Thus, the neural net parameters are chosen so that it approximates the function that generates x_t given $x_{t-1}, \ldots, x_{t-5}$. This is a difficult optimization problem because conventional wisdom has it that this function has many local minima (this assertion is quantified later).

In this analysis, $r = 5$ (this yields six elements for each γ_j vector since the intercept term is not included in the summation), $K = 5$, and $n = 1000$. These values of r and K yield 35 parameters to 'train' or fit the neural net to the function. These values were chosen because Gallant and White found they perform well in fitting the function to this series length and it is a problem of moderate size.

3.3. Computing environment

All computing was done at the University of Texas Center for High Performance Computing in Austin on Cray X-MP/24 and Cray Y-MP/864 supercomputers (all computing was done at one site even though this level of performance was not always needed). All programs were written in Fortran 77 with some Fortran 90 (which includes many features that simplify writing programs with arrays) extensions using the CFT77 compiler, versions 2.0, 3.0, and 4.0. A 64-bit representation was used for all floating point numbers; while single precision on a Cray, this represents double precision in the Vax, IBM mainframe, and PC worlds.

The Cray presented one difficulty. When a floating point error occurs on it, the program terminates. This is different from the IBM mainframe world where a software correction often takes place and execution continues. Thus, it would appear that results generated on a Cray may not carry over to other computers. However, the Cray handles an unusually wide range of numbers, from 10^{2466} to 10^{-2466}. Thus, floating point errors are only likely to occur if the algorithm is beyond the likely region of a solution. In these situations, a computer with a fixup for floating point errors is unlikely to do better. Thus, the Cray's lack of a fixup for floating point errors should not limit the lessons learned here.

3.4. Results with Judge's function

Table 1 shows the results of all algorithms for the Judge model. All algorithms were run 100 times with different starting values for the model parameters (the same ones were used by all algorithms). A uniformly distributed random number generator selected them from -100 to 100 to simulate the researcher's uncertainty about the best starting value. For the IMSL routines, their suggestions for algorithm inputs were used where given, except the maximum number of function evaluations was set to a very large value to avoid early termination.

Table 1

Comparison of all algorithms on Judge's function; bounds of starting values: $-100 < \Theta_i < 100$ ($i = 1, 2$); number of runs: 100.

Algorithm		UMPOL (simplex)	UMCGF (conjugate gradient)	UMINF (quasi-Newton)	SA (simulated annealing)
Solutions[a]	@20.482	40	48	48	0
	@16.082	60	52	52[b]	100
Mean number of function evaluations		152.54	—[c]	31.25	845,241[d]
Execution time (X-MP)		1.96 sec.	0.408 sec.	0.683 sec.	1632.8 sec.

[a] Solutions are categorized by the minimum at which they terminate. The minimum with the value of 16.082 is the global minimum.

[b] The algorithm twice reports: 'The last global step failed to locate a lower point than the current X value. The current X may be an approximate local minimizer and no more accuracy is possible or the step tolerance may be too large where $STEPTL$ = 3.696522E − 10'. All other terminations from all algorithms are reported as normal.

[c] UMCGF does not report the number of function evaluations.

[d] This number is much smaller when less conservative values for the parameters for simulated annealing were employed. See section 4.

Otherwise, neutral values were chosen. The inputs for simulated annealing are reported in table 2.

The top of table 1 lists the results from the algorithms. In all cases they converged to one of the two local minima (the global minimum has a value of 16.082, while a nearby local minimum has a value of 20.482). The conventional algorithms were split between them, while simulated annealing always found the global minimum. Of course, it could be foreseen that the conventional algorithms would split between the two optima given their design and the different starting values. However, this does demonstrate that simulated annealing can find the global optima. In effect, this is an early and very simple comparison.

The remaining part of table 1 lists the mean number of function evaluations and mean execution time for the 100 runs. As expected, the simplex algorithm takes longer than the other conventional algorithms [Press et al. (1986, p. 289)]. The much longer time for simulated annealing is largely due to using Corana et al.'s very conservative suggestions for parameters (which are appropriate for very difficult functions). In fact, a grid search with a resolution of about 0.2 could be accomplished in the same number of function evaluations (note that such a fine resolution depends upon the low dimensionality of this problem). In practice, a coarser grid search coupled with a conventional algorithm could be used. However, section 4 shows that this number of function evaluations can be reduced by more than 99% to 3789. Still, simulated annealing requires more execution time than conventional algorithms. Simulated annealing's real promise is on more difficult functions.

Table 2

Simulated annealing on the Judge function; sample output from selected temperatures.

Initial parameters for the algorithm

Initial parameters for the objective function: − 13.722, − 77.314
Resulting function value: 1.228806E + 08

T_0:	5,000,000	ε:	1.0E − 8	N_S:	20	C_i:	2.0 (i = 1, 2)
r_T:	0.85	N_ε:	4	N_T:	100	V_i:	100.0 (i = 1, 2)

Temperature	4,250,000.00
Current optimal parameters	5.8319, − 2.9451
Current optimal function value	523.7317281388
Downhill, accepted uphill, & rejected moves	1011, 1016, 1973
Stepsize	1118.4, 166.51
Temperature	18.37500
Current optimal parameters	0.87094, 1.2255
Current optimal function value	16.08428345998
Downhill, accepted uphill, & rejected moves	981, 1001, 2018
Stepsize	1.6176, 1.7525
Temperature	0.7944483E − 4
Current optimal parameters	0.86479, 1.2357
Current optimal function value	16.08173013579
Downhill, accepted uphill, & rejected moves	992, 1009, 1999
Stepsize	0.50138E − 2, 0.24662E − 2
Temperature	0.2414776E − 08
Current optimal parameters	0.86479, 1.2357
Current optimal function value	16.08173013296
Downhill, accepted uphill, & rejected moves	1012, 994, 1994
Stepsize	0.23341E − 4, 0.22194E − 4
Return from simulated annealing	Normal
Solution for parameters	0.8644876261063, 1.235748322078
Optimal function value	16.08173013296
Final stepsize	0.23341E − 4, 0.22194E − 4
Final temperature	2.4147761862962E − 9
Number of accepted moves	434,015
Number of function evaluations	868,001
Execution time (X-MP)	16.0518 sec.

Table 2 shows the results of one simulated annealing run. It first lists the values for the input parameters. With the exception of T_0, V, and ε, all are values suggested by Corana et al. The initial temperature, T_0, was chosen so that the step length was approximately 100 in both parameters (different temperatures were tried until this was achieved). This size ensures that the function is well searched. Correspondingly, the initial value of V is 100 (since the algorithm adjusts V, this initial parameter is not very important). The parameter ε was chosen to ensure that the solution has converged to the global maximum. It is slightly larger than the final tolerances used in the quasi-Newton and simplex algorithms (the conjugate gradient documentation is silent on this matter).

Table 2 next lists some of the intermediate output as the temperature falls. For each of the selected temperatures (from more than 200), the current best parameter values along with the resulting function value are shown. Also listed are the number of downhill and uphill moves. Downhill evaluations are always accepted, while uphill moves are accepted according to the Metropolis criteria. These results indicate that the step length is chosen correctly since about half of all functions evaluations are accepted (recall the step length adjusts to ensure this). The number of function evaluations at each temperature is $N_S \cdot N_T \cdot n$, or 4000. Finally, the step length for both parameters is shown.

As the temperature falls, one can observe the algorithm closing in on the global minimum. The falling temperature along with the Metropolis criteria causes fewer uphill moves to be accepted. This rise in rejections in turn shrinks the step length. In conjunction with the algorithm starting out at the current optimum with each new temperature, the smaller step length focuses the algorithm on the most promising area. The final part of table 2 shows the results produced by a successful termination of the algorithm.

This function shows the promise of simulated annealing as it found the global minimum 100 out of 100 times. Still, these results are not conclusive because, other than having trouble finding the global minimum, the conventional algorithms performed well. If faced with such a function, the researcher would only have to try several different starting values to find the global minimum. The next three functions provide much more difficult tests.

3.5. Results with the rational expectations model

The objective function for the rational expectations exchange rate model is difficult to minimize, in part because it effectively does not exist for some parameter values. In these regions, the function value is either complex or the elements of the covariance matrix go to infinity. When these regions were encountered, the objective function value was set to about 10^{2000} to force termination of the conventional algorithms (the effect on simulated annealing is described below). Unfortunately, this seems to be the only way for the IMSL routines to terminate when these regions were encountered. Values slightly different from 10^{2000} marked the reason for failure.

The conventional algorithms experienced great difficulty here. Of the 100 runs with the simplex and quasi-Newton algorithms, about half terminated due to floating point errors, which indicated that the algorithm is beyond the likely region of a solution due to the large values needed to cause a floating point error on the Cray. After accounting for the runs that reached regions where the function did not exist, the simplex algorithm found only two possible solutions and the quasi-Newton algorithm found six (all of which had smaller objective function values than the simplex algorithm produced). However, none of these six possible solutions had zero gradients and the values of five of the model parameters varied greatly (these were behavioral parameters, while the others are AR coefficients in a forecasting equation).

The conjugate gradient algorithm performed slightly better. Only one run resulted in a floating point error, while 53 ended because the algorithm entered a proscribed region. Of the other runs, seven reported objective function values just slightly larger than those reported by the quasi-Newton algorithm (the other values were quite a bit larger). Like the quasi-Newton result, the five behavioral parameters varied greatly in value and the gradients were not quite zero. Thus, the conventional algorithms failed to optimize this function since no two runs resulted in the same set of parameter values and no solution had a zero gradient. Further, they indicate that the function appears to be flat in a subset of the model's parameters. Conventional wisdom has it that this is a feature of some rational expectations models.

Simulated annealing, like the conventional algorithms, also experienced some difficulty. It converged to different optima for different starting values and seeds for the random number generator. The function was then modified to search a restricted region of the parameter space to find an explanation. For parameter Θ_i, let the region's boundary be marked by α_i and β_i. If simulated annealing chooses a value for Θ_i inside this range, it proceeds as usual. If outside, the objective function was not calculated and a very large value was returned instead, which caused the point to be rejected. This limits the algorithm to be inside the boundary. The previously mentioned modification to the objective function for regions in which it does not exist excludes the algorithm from those regions as well. This ability to focus on one area for minimization is a distinct advantage of simulated annealing. When this region was enlarged over successive runs, an interesting phenomena occurred: the optimal value of one parameter (the interest rate elasticity of money demand) often followed its upper bound. This suggested that the function decreased in this parameter (i.e., the algorithm found a minimum near a saddlepoint). To explore this situation, several of these minima were plotted with the elasticity varying and the other parameters held constant. These plots showed that the interest rate elasticity parameter achieved a minimum at the boundary. Thus, the function appeared to be a ditch in this parameter: as the boundary expanded, the minimum point of the function followed this wandering ditch. This explains the problems of the conventional algorithms. Thus, simulated annealing is useful as a diagnostic tool for difficult functions. (It should be noted that it may be possible to identify problems like this with conventional algorithms by examining the Hessian. Unfortunately, this idea could not be done here because these IMSL routines do not allow the user to recover the Hessian.)

Table 3

Conventional algorithms on the modified rational expectations model; bounds on starting values: $-1 < \Theta_i < 1$ ($i = 1, \ldots, 13$); number of runs: 100.

Algorithm	UMPOL (simplex)	UMCGF (conjugate gradient)	UMINF (quasi-Newton)
Results by the final value of the objective function			
Complex	0	0	0
Huge covariance matrix elements	10	12	12
Other (possible solution)	10	84	10
Total nonfloating point error returns	20	96	22
Floating point error returns	80	4	78
Minimum value found	6.487554836067[a]	6.486982467843	6.486982467804
Program reported results (covers all nonfloating point errors)			
Normal	20	57[b]	18[c]
Line search abandoned (UMCGF)[d]		39[e]	
Failed to find lower value (UMINF)			4[f,g]
Execution time (X-MP)	2 min., 21 sec.	33 min., 25 sec.	14 min., 12 sec.

[a] Of the ten possible solutions, the final values of the objective function ranged from this to about 6.945.

[b] Of these 57 returns, 44 produced objective values less than 6.486983. The corresponding gradients are approximately zero and the parameter values are nearly identical. The slight variation in objective values is likely due to the termination criteria.

[c] Of these 18 returns, 12 are due to the huge elements in the covariance matrix. The other six had identical objective values, near-zero gradients, and approximately equal parameter values.

[d] The algorithm reports: 'The line search of an integration was abandoned. An error in the gradient may be the cause.'

[e] Only one value was near the presumed minimum; the rest were some distance away.

[f] The algorithm reports: 'The last global step failed to locate a lower point than the current X value. The current X may be an approximate local minimizer and no more accuracy is possible or the step tolerance may be too large where $STEPTL = 3.696522\text{E} - 10$.'

[g] The final values of the parameters, objective values, and gradient were indistinguishable from the 'good' normal results.

To continue the comparison of the algorithms, the interest rate elasticity parameter was set to 0.25 and the above experiments were repeated. This reduces the number of parameters in the model to 13. Table 3 contains the results from the conventional algorithms on this modified function. The first category lists outcomes by the final value of the objective function. This includes errors in the function evaluation and solutions. The sum of these possibilities yields the number of nonfloating point error returns because floating point errors immediately terminate the program and the function value is unknown. The second category lists the number of floating point error returns. Since the Cray supports such a wide range of floating point values, these are likely caused by an out of control algorithm (i.e., floating point errors indicate that the algorithm was very far removed from any likely solution given the size of values that produce floating point errors on the Cray). The number of runs in this and the previous category sums to 100, the total number of runs. The next category reports the minimum value found (by default, from the 'other' possibility). The final category lists the results reported by the algorithm.

The simplex algorithm again performed poorly with many floating point error terminations and only a few terminations near the presumed minimum. The quasi-Newton algorithm did somewhat better by finding the presumed minimum a total of ten times. However, four of these ten were reported as errors. Thus, even with a moderate number of runs, the researcher would have difficulty interpreting the results obtained with either of these algorithms.

The conjugate gradient algorithm found the correct minimum 44 times. All objective function values were within 10^{-6} of each other in this count and the model parameters were very similar if not identical. Thus, this algorithm would have proved fairly useful to the researcher for this function since about half of all runs terminated successfully.

Finally, the simulated annealing algorithm was employed. One hundred different starting values could not be used to check the algorithm's results due to the computational load with this function, so another method was used. The algorithm was run twice with different starting values for the model's parameters and a different seed for the random number generator inside the algorithm. This generator selects points inside the step length for analysis and selects uphill moves through the Metropolis criteria. Thus, the algorithm follows a completely different path in these two runs. If it terminates at the same point, one has some assurance that simulated annealing has indeed found the global minimum. The only other change from the runs with Judge's function are in N_T (set to 20) and ε (discussed below). The selection of the initial temperature is an important consideration. If the initial temperature is too low, the step length will be too small and the area containing the optimum may be missed. If too high, then the step length is too large and an excessively large area is searched. If the step length is extremely large, floating point errors may result from running very large values through the function.

The following method was employed to find T_0. It was set to 10^7 and the temperature reduction parameter, r_T, was 0.10. Rather than finding the minimum, the goal was to quickly find the temperature at which the step length began to decline. This occurred with a temperature of 100. Upon further work, an initial temperature of 10 was chosen because the step length was only slightly smaller and the lower temperature allowed a shorter execution time. Interestingly, even at a temperature of 10^7, some parameters had a step length less than 1, which implies that the function is quite narrow in these parameters.

The function was then modified to include bounds. The lower bound was set to -25 and the upper to 25 for each model parameter, a range which should

Table 4

Simulated annealing on the modified rational expectations model.

T_0:	10.0	ε:	1.0E − 11	N_S:	20	C_i:	2.0 (i = 1, 13)
r_T:	0.85	N_ε:	4	N_T:	20	V_i:	100.0 (i = 1, 13)

Final step lengths

0.15678E − 07	0.97505E − 06	0.76930E − 05	0.97651E − 05	0.15193E − 05
0.44804E − 05	0.33611E − 05	0.10882E − 05	0.31521E − 05	0.15193E − 05
0.11529E − 05	0.21567E − 05	0.22594E − 05		

Final function value: 6.4869
Final temperature: 1.03058E − 12
Number of accepted moves: 477,446
Number of function evaluations: 956,801
Execution time (X-MP): 40 min., 18 sec.

include any economically plausible values. The model was later run without any bounds and the same solution was found (bounds were first used because experience with the translog cost frontier model, discussed in the next section, showed that floating point errors could occur without them due to very large parameter values being run through the function). The initial runs on the Cray were in a job classification that allowed the jobs to execute about 22 minutes, which corresponds approximately to $\varepsilon = 10^{-6}$. The next longer job classifications had long turn-around times (sometimes days at that time), so exploratory runs were in this category. In two runs with different random number generator seeds and starting values, the algorithm converged to approximately the same objective function value. Further, the difference in the reported optimal model parameters was on the order of the step length, so both runs were minimizing in the same region. This implies that both were converging to the same point. To make certain, a run was then made in the next job classification with $\varepsilon = 10^{-11}$, which resulted in a solution in the same region. Further, the successful runs from the conventional algorithms found this same point. The results are shown in table 4. To further test the algorithm, the starting values from one of the above mentioned shorter runs was used with the random number seed from the other shorter run with $\varepsilon = 10^{-11}$. When run in the extended job category, the same point was found. Thus, three different paths were taken and they all found the same optimum. In addition, the elements of the gradient at this point were approximately zero. Also, note that the optimal function value differs from the best one from the quasi-Newton algorithm by 10^{-12}, an insignificant amount. Finally, it can be seen that about half of all function evaluations were accepted as expected.

Simulated annealing demonstrated two advantages with the rational expectations model. First, it was able to identify, with a modification introduced here, the reason the conventional algorithms could not minimize the function.

Second, with that problem eliminated, simulated annealing was much more consistent in finding the minimum (it was 3 for 3, while the conventional algorithms were, with generous accounting, 64 for 300). This was particularly true when the simplex and quasi-Newton algorithms are considered (generously, 20 for 200). While a single run of simulated annealing requires substantially greater execution time, this is ameliorated by the large number of runs with a conventional algorithm a researcher would have to make to be sure of the robustness of estimated results. Section 4 shows that the execution time of simulated annealing can be substantially reduced for this model to the point where it is competitive with a single run of a conventional algorithm.

3.6. Results with the translog cost frontier model

The objective function for the translog cost frontier model was easier to work with than the rational expectations function because the only errors in its evaluation were floating point errors. Table 5 reports the results from the conventional algorithms (its format follows table 3).

The first part of the table shows that only the quasi-Newton algorithm was unstable because only it experienced floating point errors. Unfortunately, run-on executions were more of a problem. For instance, one simplex run executed (unintentionally) for 1 hour, 47 minutes for more than 2 million function evaluations! Clearly, the starting values here were not useful. Rather than exploring a few runs with such extended execution times, which may simply be exhibiting pathological behavior, the number of function evaluations was limited. The number of function evaluations for the simplex algorithm was limited to 25,000, which yields a maximum execution time of about 58 seconds. A similar choice was made for the conjugate gradient algorithm, but since few runs resulted in run-on executions, a limit was chosen that constrained execution time for any one run to about 12 minutes, surely enough time to find an optimum. No constraint on function evaluations was applied to the quasi-Newton algorithm since it did not experience this problem.

As table 5 shows, no floating point errors occurred with the simplex algorithm. However, only 14 runs terminated normally, while the other 86 reached the function evaluation limit. The normal returns yielded objective function values of approximately $1.67 \cdot 10^{10}$. It appears that there is a plateau of about this height since the model parameter values varied dramatically. The category for too many function evaluations included the best value found for the objective function, 4363.81. A slightly smaller value was found in an earlier trial with unlimited function evaluations. In 144,284 function evaluations, a value of 2543.79 was found. The same starting value lead to 4721.51 when the number of function evaluations was constrained to 25,000. This exceedingly slow progress toward the minimum may indicate that the function has steep valleys, something

Table 5

Conventional algorithms on the translog cost frontier model; bounds on starting values: $-20 < \Theta_i < 20$ ($i = 1, \ldots, 62$); number of runs: 100.

Algorithm	UMPOL (simplex)	UMCGF (conjugate gradient)	UMINF (quasi-Newton)
Results by the final value of the objective function			
Total nonfloating point error returns	100	100	49
Floating point error returns	0	0	51
Minimum value found	4363.817614080[a]	1015.176621710[b]	− 1581017798364[c]
Program reported results (covers all nonfloating point errors)			
Normal	14	0	7[d]
Too many function evaluations	86[e]	6[f]	
Line search abandoned (UMCGF)[g]		94	
Failed to find lower value (UMINF)[h]			42
Execution time (X-MP)	85 min., 52 sec.	76 min., 58 sec.	26 min,. 1 sec.

[a] This value occurred with an iteration that terminated because the number of function evaluations exceeded 25,000. The execution time was approximately 58 seconds.

[b] Most gradients were small, but several were greater than 100 in absolute value. The routine terminated because it exceeded 5000 function evaluations.

[c] This value is an anomaly since many parameter values were greater than 1000 in absolute value. Evidently, away from 'reasonable' parameter values, the function becomes nonsensical. The next best value was 4019.03995809.

[d] None of these values contained a zero gradient. Typically, the gradient elements were either zero or ranged from 10E + 8 to 10E + 22.

[e] As described in footnote a, the number of function evaluations was limited to 25,000.

[f] As described in footnote b, the number of function evaluations was limited to 5000. These six iterations took 97.1% of the execution time for this run. All resulting objective values ranged from 1015.17 to 1308.62. The gradient values were again quite large.

[g] The algorithm reports: 'The line search of an integration was abandoned. An error in the gradient may be the cause.'

[h] The algorithm reports: 'The last global step failed to locate a lower point than the current X value. The current X may be an approximate local minimizer and no more accuracy is possible or the step tolerance may be too large where $STEPTL$ = 3.696522E − 10.' Typical gradient values were approximately 1.0E + 8.

the simplex algorithm may have difficulty traversing [Press et al. (1986, p. 290)]. Confirmation of this hypothesis is demonstrated below.

Like the simplex algorithm, the conjugate gradient algorithm experienced no floating point errors and only six runs ended due to too many function evaluations. These six runs produced the best function values, with the least being 1015.17 (values ranged up to 1308.62). In every case at least some elements of the gradient were larger than 100 in absolute value (from this point forward, all reported gradient values are in absolute value terms). These six runs consumed 97.1% of the execution time of the 100 runs. The other 94 terminated

Table 6

Simulated annealing on the translog cost frontier model.

T_0:	500.0	ε:	1.0E − 10	N_S:	20	C_i:	2.0 ($i = 1, 62$)
r_T:	0.85	N_ε:	4	N_T:	10	V_i:	100.0 ($i = 1, 62$)

		Final step length		
0.10593E − 12	0.27917E − 14	0.10735E − 13	0.23835E − 14	0.34958E − 14
0.24298E − 14	0.27545E − 13	0.46932E − 14	0.68794E − 14	0.11937E − 13
0.43830E − 14	0.13357E − 13	0.17099E − 14	0.25643E − 14	0.29042E − 14
0.57918E − 15	0.17286E − 14	0.31509E − 14	0.41301E − 14	0.75095E − 15
0.14656E − 14	0.10222E − 14	0.11749E − 14	0.20317E − 14	0.15261E − 14
0.96374E − 15	0.66411E − 15	0.11723E − 14	0.62383E − 13	0.20851E − 14
0.59387E − 13	0.60386E − 14	0.42017E − 15	0.25412E − 14	0.19055E − 14
0.21703E − 14	0.33987E − 13	0.41447E − 14	0.48380E − 13	0.15540E − 13
0.15090E − 14	0.10994E − 14	0.59865E − 14	0.39168E − 14	0.16177E − 13
0.19995E − 14	0.40851E − 14	0.24939E − 14	0.24520E − 14	0.21529E − 14
0.57125E − 15	0.16523E − 14	0.16464E − 14	0.18056E − 14	0.11392E − 14
0.22389E − 07	0.32855E − 08	0.93383E − 15	0.11984E − 15	0.86365E − 13
20.564	13.396			

Final function value: − 1601.08
Final temperature: 1.443007875941E − 12
Number of accepted moves: 1,253,777
Number of function evaluations: 2,554,401
Execution time (X-MP): 53 min., 20 sec.

with reported problems with the gradient. Since in these runs at least some elements of the gradient were quite large (values for some elements in each ranged from 10^8 to 10^{20}), this report is not surprising. This confirms the explanation offered for the problem with the simplex algorithm.

The quasi-Newton algorithm yielded 51 terminations due to floating point errors, indicating that the algorithm was very far removed from any likely solution given the size of values that produce floating point errors on the Cray. The other 49 runs resulted in 7 returns reported as normal (though every gradient had elements of either 0.0 or 10^8 and larger) and 42 in which the algorithm terminated abnormally. Here too, gradient values were 10^8 and above. The best objective function value was an anomaly: it was negative, but since all parameter values were greater than 10^3 in absolute value, this result can be discarded on the grounds that it is nonsensical. The next best value was 4019.03, but again, elements of the gradient were large (some were more than 10^3).

Table 6 shows the results with simulated annealing. To avoid floating point errors due to large parameter values being run through the algorithm, the parameter space for this function was restricted: the lower bound was set to − 50 and the upper to 50. An initial temperature of 500 produced an initial step length vector with elements that averaged in the teens, a reasonable span given the expected parameters values. As with the rational expectations model, two preliminary runs with different starting values and seeds for the random number generator produced minima in the same range. A run with a longer execution time and $\varepsilon = 10^{-8}$ then produced a refined estimate of this minimum. As a check, the gradient at this point was examined. Its values were typically 10^{-2}, except for Θ_{59} which was 0.26. While these values were much smaller than any produced by the conjugate gradient or quasi-Newton algorithms, the value of 0.26 was a cause for concern. Thus, a smaller ε of 10^{-10} was chosen and the algorithm was rerun.

The results are shown in table 6. Note that the value of the objective function was -1601.08. All gradient elements were approximately $0.5 \cdot 10^{-3}$ or so, except for Θ_{59} which was 0.24. A plot showed that the estimated value for Θ_{59} was indeed the minimum. This gradient value occurred because the gradient algorithm had difficulty with the extreme narrowness and slight asymmetry of the function in this area.

Simulated annealing was able to find the minimum, while 300 runs with three different conventional algorithms were unable to find this minimum even once. This offers strong evidence that simulated annealing can be a very useful algorithm for the optimization of statistical functions. Further, note that the execution times for simulated annealing and the conventional algorithms are comparable.

3.7. Results with the neural network

Table 7 shows the results with the conventional algorithms on the neural net, while table 8 contains the results obtained using simulated annealing. The starting values were chosen from the range of -600 to 600 (preliminary work showed optimal values in this range). The top of table 7 shows that, like the translog cost frontier model, only the quasi-Newton algorithm experienced a problem with floating point errors. As run-on executions were also a problem, limits were placed on the number of function evaluations to ensure that 100 sets of starting values could be run in a reasonable time.

All 100 runs with the simplex algorithm terminated because of too many function evaluations (more than 20,000). The minimum function value found was 229.74. The conjugate gradient algorithm also yielded a simple, but different story: all 100 runs terminated due to problems with the line search. The minimum function value found was 281.08. The quasi-Newton algorithm generated the most varied results. Five runs terminated with floating point errors, which indicates a severe problem with the algorithm (floating point errors occur with such large exponents that the algorithm is surely away from any region of possible interest). Of the 95 runs that terminated without floating point errors, 58 reported resulted normal terminations, 35 reported possible problems with the step tolerance, and two terminated with too many function evaluations. The

Table 7

Conventional algorithms on the neural net; bounds on starting values: $-600 < \Theta_i < 600$ ($i = 1, \ldots, 35$); number of runs: 100.

Algorithm	UMPOL (simplex)	UMCGF (conjugate gradient)	UMINF (quasi-Newton)
Results by the final value of the objective function			
Total nonfloating point error returns	100	100	95
Floating point error returns	0	0	5
Minimum value-found	229.7440[a]	281.0888[b]	228.2120[c]
Program reported results (covers all nonfloating point errors)			
Normal	0	0	58
Too many function evaluations	100[d]	0	2[e]
Line search abandoned (UMCGF)[f]		100	
Failed to find lower value (UMINF)[g]			35
Execution time (Y-MP)	55 min., 17 sec.	30 min., 26 sec.	76 min., 8 sec.

[a] This value occurred with an iteration that terminated because the number of function evaluations exceeded 20,000.

[b] Most elements of the gradient were small and the largest element was 0.245 in absolute value.

[c] The largest element of the gradient was 12.61 in absolute value. The routine terminated normally.

[d] As described in footnote a, the number of function evaluations was limited to 20,000.

[e] The maximum number of function evaluations was 10,000. Even though more function evaluations could have likely been used, these iterations were not promising since they had function values of 381.993 and 522.54.

[f] Each time, the algorithm reported: 'The line search of an integration was abandoned. An error in the gradient may be the cause.'

[g] The algorithm reports: 'The last global step failed to locate a lower point than the current X value. The current X may be an approximate local minimizer and no more accuracy is possible or the step tolerance may be too large where $STEPTL = 1.000000\text{E} - 14$.'

minimum function value was 228.21, the best for any of the conventional algorithms.

Initial runs with simulated annealing used $T_0 = 1.0$ and $r_T > 1$ (with $r_T > 1$ temperature rises, which in turn increases the step lengths) to find the initial temperature such that each element of the step length vector was 1200 to cover the region of likely values (-600 to 600). These bounds for the search avoided both the floating point errors and the corner solutions that were observed in some initial runs. These runs found that $T_0 = 3.5 \cdot 10^8$ produced the desired initial step lengths. The first optimizing run used this initial temperature. In addition, the number of function evaluations was limited to conserve execution time, but was set in conjunction with r_T and the other parameters to reduce the likelihood that the algorithm would quench but also achieve a small terminal step length (table 8 shows that this was achieved). Standard values were used for the other parameters for simulated annealing. Even given its preliminary nature,

Table 8

Simulated annealing on the neural net.

T_0:	3.5E8	ε:	1.0E − 10	N_S:	20	C_i:	2.0 (i = 1, 35)
r_T:	0.85	N_ε:	4	N_T:	20	V_i:	100.0 (i = 1, 35)

γ_j solution vectors (one vector per row; $\gamma_{1,j}, \ldots, \gamma_{5,j}, \gamma_{0,j}$)

− 5.13	0.42	1.06	10.98	553.7	246.2
− 15.08	− 3.90	− 7.57	− 7.96	504.1	− 250.1
− 8.30E − 2	2.43E − 3	2.06E − 4	3.23E − 2	4.98	− 0.10
5.85	4.20	− 2.99	1.76	406.19	− 433.3
20.21	0.32	− 5.67	− 13.86	515.31	551.7

β solution vector

0.34	0.29	2.05	− 1.13	− 1.13

Final function value: 22.845
Final temperature: 3.5098E − 4
Number of accepted moves: 1,671,870
Number of function evaluations: 2,380,002
Execution time (Y-MP): 42 min., 51 sec.

table 8 shows that the optimal value of the function was 22.845, about an order of magnitude smaller than any of the 300 runs with the conventional algorithms. In addition, this execution time is comparable to the total execution time (i.e., for 100 different sets of starting values) of any of the three conventional algorithms.

To check this result, a different starting value and a different seed were used in another run. Unfortunately, while a similar function value was found, the γ's were very different. Thus, as anticipated, there appear to be multiple optima in this function. In an effort to find the global optimum, the parameter space in the next run was reduced by shrinking the bounds with information gained from these solutions. The optimal β vector had values of less than 20 in absolute value, so the bounds for β were reduced. This allowed $T_0 = 4000$ because the function is much steeper in β than in γ. The combination of a smaller T_0, the same execution time, and the same desired terminal temperature allowed for a larger r_T, which makes quenching less likely. Even so, the same optima could not be found twice. Finally, patterns were noticed in the solutions and these were exploited to further search for the optimum. In four of the five γ_j vectors, the fifth and sixth elements were quite large (typically more than 300 in absolute value), while the remaining γ_j have small elements (typically less than 10 in absolute value). Table 8 demonstrates the pattern. Even when the bounds were adjusted to search in this much smaller region, the same optima was never found twice. However, the best optima, a value of 10.649, was found in one of these runs.

Further analysis of the neural net revealed an interesting phenomenon: the large values in the optimal γ_j vectors swamped the other values in the $x \cdot \gamma_j$ term to create such large absolute values that the logistic function $G(\cdot)$, which is bounded by 0 and 1, was approximately either 0 or 1. In fact, for the best optima, 78.9% of the $G(\cdot)$'s were either less than 10^{-6} or greater than $1-10^{-6}$. Thus, to a considerable extent, the neural net was reduced to a linear combination of the elements of the β vector.

Even though simulated annealing found a much better optimum than any of the conventional algorithms and in less time than all but one, it was distressing that it was incapable of finding the global optimum. To understand this outcome, an analysis was made of the neural net. First, a rough lower bound on the number of local optima was calculated, then the nature of these local optima was examined.

Since enumeration of the local optima is impractical, the following method was used. A measure of the volume [more accurately called content since there are more than three dimensions; see Kendall (1961)] of the average local optima was determined with a local optimizer. Then, the content of the total solution space is determined. Division of the content of the two spaces should yield a rough estimate of the number of local optima.

The quasi-Newton algorithm was used to find local optima since it produced the most error-free optima. Different length vectors in random directions were then drawn from each local optima and the end of that vector was used to restart the quasi-Newton algorithm. The length of the vectors were increased until the algorithm no longer converged to the starting point of the vector. The longest vector that led to convergence back to the local optima was used as the radius of convergence. To be conservative, four different vectors of the same length were drawn at a time and the length increased until only one of the four converged to the starting point of the vector. Of the seven local minima examined (this number was chosen due to floating point and computer time constraints), the average radius of convergence was found to be 32.53. For a hypersphere of n dimensions with a radius of r, the content is

$$\frac{2 \cdot r^n \cdot \pi^{n/2}}{n \cdot \Gamma(n/2)}$$

[see Kendall (1961, p. 35)]. This yields a content of the average local optima of $2.84 \cdot 10^{46}$.

The most conservative possible method was used to calculate the content of the solution space. It is taken to be the content of the hyperrectangle created by the two closest local optima (all bounding hyperplanes are perpendicular to the axes in Cartesian space). While this surely underestimates the true content, it is easy to find with a cluster algorithm (PROC CLUSTER in SAS® was used). The content is $8.29 \cdot 10^{65}$. Dividing this and the content of the average optima yields on the order of 10^{19} local optima.

This many optima might not pose a problem if each local optima is just a dimple on a larger convex surface (like an egg carton tilted up at the ends). With such a surface, a global optimization algorithm could use some average of the surface to find the global minimum. Unfortunately, this does not seem to be the case with the neural net. The average value of the function, when sampled over 100,000 points with all parameters randomly sampled from -600 to 600 was $3.75 \cdot 10^{8}$ (even when the bounds on the β vector were set to -3.5 and 3.5, as the simulated annealing results suggest, the average value was 13,427). The largest optima found by the three conventional algorithms in their combined total of 300 runs was 633.20 and the smallest was the previously reported 228.21. Given this small range and the large average value of the function, the neural net surface appears to be more like that of an egg carton that has not been tilted or curved to any appreciable degree, rather than one that has been distorted.

Given the relative values of the optima, it would appear that each local optima would need to be examined to find the global optima. Given the number of local optima, this hardly appears likely given current or foreseen computers.

4. Improvements to simulated annealing

4.1. Review of previously introduced improvements

Three improvements to simulated annealing were introduced above. One allows the researcher to test if simulated annealing has indeed found the global optimum. By rerunning the algorithm with a different starting value and a different seed for the random number generator, an entirely different sequence of points is selected by the algorithm. If the same optimum is found, there is a higher degree of confidence in the global optimum. This modification was very useful for understanding the unmodified rational expectations model and the neural net.

A second improvement restricts the search area to a subset of the parameter space. This was useful as a diagnostic tool for difficult functions and to restrict the parameters chosen by the algorithm so that very large values are not run through the statistical function, which may cause floating point errors. This is also useful for functions that are very flat in some variables where a low initial temperature might restrict the search in other variables. When minimizing a function, one can think of this as putting the function in a very deep well. Points selected outside the well yield very large values and are rejected. This forces the algorithm inside the well and to the region of interest. This modification was essential for the translog cost frontier and rational expectations models.

The final improvement allows the researcher to determine the initial temperature, an essential parameter for the algorithm. It was also shown to be quite useful.

4.2. Tailoring the algorithm to the function

Section 3 showed simulated annealing to be clearly superior to conventional algorithms for some difficult statistical optimization problems. However, this benefit does not come without cost because simulated annealing may require substantially more execution time. This section introduces a modification to minimize execution time. Essentially, it tailors simulated annealing to the minimum level of robustness required to optimize the function.

This approach chooses the appropriate r_T and N_T for the function at hand. These two values greatly influence the robustness and number of function evaluations since they control how quickly the temperature declines and the number of function evaluations performed at each temperature. Much smaller values for r_T and N_T than those suggested by Corana et al. [0.85 and the maximum of (100, $5 \cdot n$), respectively] are chosen and used in one or several runs. These values are then increased and the algorithm is run again with a different random number generator seed and starting value, resulting in an entirely different sequence of sampled points. If the same optimum is found, one has a reasonable degree of assurance that the global optimum is reached. If a different optimum is achieved, then the procedure is repeated with larger values for r_T and N_T. In these runs, one should carefully monitor the intermediate results. The following results are a bit artificial since the optimum is known, but they should be useful as a guide.

This approach is first illustrated with the Judge function. With $N_T = 1$ and $r_T = 0.05$, 14 of 100 runs with different starting values and seeds found the local and not the global minimum (i.e., the algorithm quenched). When N_T was increased to 2, only two runs terminated incorrectly, while with $N_T = 5$, all terminated correctly. This required an average of 3789 function evaluations, a decrease of 99.6% from the 852,001 function evaluations used with the Corana et al. parameter suggestions (see table 1). Since temperature fell very quickly and relatively few evaluations were made at each temperature, only 38% of the function evaluations were accepted. Unless C (which controls how fast the step length is adjusted) is increased, this limits the values of N_T and r_T since successful evaluations are essential to the operation of the algorithm.

The rational expectations function yielded the same results. With ten runs and $N_T = 2$ and $r_T = 0.1$, five runs terminated correctly and five failed. The failures were easy to detect because all terminated at different optimal values and after a few initial temperature reductions; there were no uphill moves (it appears that the algorithm was caught in a local optimum and it couldn't escape given these parameter values). When r_T was increased to 0.25, nine runs terminated normally. The abnormal run had the same characteristics as the previous ones. The successful runs required an average of 13,521 function evaluations, a reduction of 98.6% from the 956,801 in table 4 (where N_T was 20 and r_T was 0.85). This produced an execution time of 69 seconds, better than any of the sets of 100 runs reported in table 3 and comparable to some of the single runs.

This technique produced no reduction in N_T and r_T for the translog cost frontier model, so there was no reduction in the 2,554,401 function evaluations reported in table 6. Even a small reduction in N_T and r_T produced runs that, with only a few temperature reductions, lead to no uphill moves – exactly the same phenomenon that occurred with the rational expectations functions. Given the extremely steep gradients of this function, it is not surprising the algorithm got stuck in a local optimum. This method was not used with the neural net since the global optimum was not found.

Thus, of the functions with a global optimum that can be found, this section shows that only the translog cost frontier model requires a high performance computer for successful optimization and that the number of function evaluations needed by simulated annealing may be quite reasonable for some functions. Thus, simulated annealing is immediately useful to researchers for many functions.

5. Conclusion

This paper tests a recently developed algorithm, simulated annealing, on four different statistical models that require optimization. Compared to the three conventional optimization algorithms from the IMSL library, simulated annealing is shown to have several advantages. First, it can be used as a diagnostic tool to understand how conventional algorithms fail. Second, it can 'step around' regions in the parameter space for which the function does not exist. But, most importantly, it can optimize functions that conventional algorithms have extreme difficulty with or simply cannot optimize at all.

The first test model, a synthetic example from the literature, has two minima. Simulated annealing correctly differentiated between the global and local minima, while the conventional algorithms did not. The second test function was a rational expectations exchange rate model. All conventional algorithms failed on it, and simulated annealing was able to identify the reason. After correcting the problem, the conventional algorithms found the optimum only 21% of the time, with most of the successes from only one algorithm. Simulated annealing was able to find the optimum easily. The third test function was from an efficiency study of the banking industry using a translog cost frontier model. None of the conventional algorithms were able to optimize it, but simulated annealing did so easily. For the fourth test function, the neural network, simulated annealing found a much better optimum than any of the conventional algorithms. While simulated annealing was not able to find the global optimum, it was shown that the nature of the function makes it virtually impossible for any method to find the global optimum.

We introduced a number of extensions to simulated annealing. Perhaps the most important one allows the researcher to tailor the algorithm to the function to minimize the execution time of the algorithm. This allows simulated annealing to find the global optimum for actual econometric problems on commonly available computers.

Appendix A: Computer performance

The most useful widely used benchmark of computer performance for econometric work is probably the Linpack benchmark. In its most common form, a system of 100 linear equations are solved and the number of floating point operations per second are counted (mflops for a million of them). This is an appropriate benchmark since it involves floating point operations on arrays.

Table 9 [from Dongarra (1991)] shows the number of floating point operations per second on the Linpack benchmark for some large computers using one processor. The Cray Y-MP is currently the top of Cray's line; it replaced the X-MP. The neural network described above ran on a Y-MP and the other work for this paper ran on an X-MP. The IBM ES/9000 is the top of IBM's line. The Cray 1S was Cray's first widely used machine and is still used as a reference. The IBM 3081 was the top of IBM's mainframe line in the early 1980s. Note that all but this last machine are so-called vector processors that are designed to quickly perform mathematical operations that occur in long loops.

Table 10 [from AIXTIPS (1992) for RS/6000's, Varhol (1991) for Club America, and Dongarra (1991) for IBM PC] shows the performance of some desktop computers. The IBM RS/6000 models 220 and 350 are the low and high ends, respectively, of IBM's desktop workstation line. The model 220 comes

Table 9

Computer	Mflops[a]
Cray Y-MP	161.000
Cray X-MP	121.000
IBM ES/9000 Model 900 VF	60.000
Cray 1S	12.000
IBM 3081 K	2.100

[a] Millions of floating point operations per second.

Table 10

Computer	Mflops[a]
IBM RS/6000 model 350	18.6
IBM RS/6000 model 220	6.5
Club America Hawk III	1.46
IBM PC (with 8087 math coprocessor)	0.012

[a] Millions of floating point operations per second.

Table 11

Computer	Execution time (seconds)
Cray Y-MP	13.11
IBM RS/6000 model 220	35.17
IBM 3081 D	61.80
486 33 Mhz clone (PC AGE)	236.0

fully equipped with an academic price of approximately $7,000. It is of particular interest because the core of its CPU chip will be used in machines Apple and IBM will be introducing in 1993 as part of their joint venture [Linderholm and Marshall (1992)]. With their higher clock speed, these machines may perform at approximately ten mflops on the Linpack benchmark. The street price of the Apple machines is estimated to be $1,000 to $2,000 [Linderholm and Marshall (1992)]. The Club America Hawk III is one of the fastest Intel 486 computers. The IBM PC with its math coprocesser, the Intel 8087, is well known.

In comparing tables 9 and 10, one can see that desktop machines have drastically increased in power over the last decade and some current machines approach or surpass the Cray 1S. Also note that a 486 clone comes close to the leading IBM mainframe of the early 1980s. Further, this trend seems certain to continue; Gelsinger et al. (1989) estimate the peak floating point performance of a microprocessor in the year 2000 to be 2,000 mflops (a single processor of a Cray Y-MP has a peak of 334 mflops).

To further compare different machines' performance, our simulated annealing code with 1,000,000 evaluations of the Judge function was run on the machines shown in table 11. While the structure of the Judge function is more favorable to the RS/6000 than the Y-MP (the small size of the function should fit inside the instruction and data caches of the RS/6000 processor, while the relatively short loop length of 20 doesn't allow the Cray's vector processor to show its full speed), the relatively small difference is striking. The 486 clone would likely have done better with a 32-bit compiler, but none was available.

This appendix shows that dramatic increases in affordable computer power have occurred in recent years and show no evidence of slowing down anytime soon. Such easily available high performance computing makes simulated annealing (and other computationally intensive methods) quite attractive and affordable.

Appendix B: Simulated annealing algorithm

```
SET initial parameters and values
        set $C$      (controls how fast $V$ adjusts)
        set $X$      (starting values for model parameters)
        set $V$      (should cover the entire range of interest in $X$)
        set $\varepsilon$      (convergence criteria)
        set $r_T$     (temperature reduction factor)
        set $T_0$     (initial temperature)
        set $N_\varepsilon$     (# times $\varepsilon$ tolerance is achieved before termination)
        set $N_S$     (# times through function before $V$ adjustment)
        set $N_T$     (# times through $N_S$ loop before $T$ reduction)

CALCULATE $f(X)$
        $X_{opt} = X$
        $f_{opt} = f$

DO UNTIL convergence
   DO $N_T$ times
      DO $N_S$ times
         DO $i = 1, n$
            $x_i' = x_i + r \cdot v_i$ {$r$ is uniform random number on $[-1, 1]$}
            CALCULATE $f'(X')$
            IF $f' \leq f$ THEN
               apply Metropolis criteria
               IF accepted: $X = X'$ & $f = f'$
            END IF
            IF $f' > f$ THEN
               $X = X'$ & $f = f'$
            END IF
            IF $f' > f_{opt}$ THEN
               $X = X', f = f', X_{opt} = X'$, & $f_{opt} = f'$
            END IF
         END DO
      END DO
      ADJUST $V$ such that half of all trials are accepted
   END DO
```

```
  IF change in $f_{opt} < \varepsilon$ last $N_\varepsilon$ iterations & $|f - f'| < \varepsilon$ THEN
    REPORT $X_{opt}, f_{opt}$, & $V$
    STOP
  ELSE
    $X = X_{opt}$ {start on current best optimum}
    $T = r_T \cdot T$ {reduce $T$}
  END IF

CONTINUE
```

Appendix C: Experience with the genetic algorithm

Genetic algorithms mimic natural selection for function optimization. The value of a function corresponds to the fitness of an organism, while the parameters for the function correspond to the genes of that organism. The genes that produce the fittest function values from one generation 'mate' with each other and pass on their successful traits to the next generation. Through a sequence of generations, the function should be optimized, much as an organism is for its environment. As described by Goldberg (1989), from which this discussion follows, genetic algorithms have been used successfully in many areas, including pipeline operation, structural optimization, job shop scheduling, and filter design. Most of the work, however, has been for optimizing noncontinuous functions. It should be noted while Goldberg was not the originator of the genetic algorithm, he and his workers developed it to such a degree that he was a NSF Presidential Young Investigator in 1985.

Like simulated annealing, the genetic algorithm uses probabilistic decision rules and can escape local optima. It also slowly optimizes the function, as opposed to 'greedily' optimizing like conventional algorithms. However, as described below, there are numerous differences between simulated annealing and the genetic algorithm.

In the genetic algorithm, the parameters of the function to be optimized are coded as a character string, most commonly as binary digits. For instance, the integers from 8 to -7 can be coded with the binary string $b_1 b_2 b_3 b_4$, $b_i = \{0, 1\}$, as $b_1 \cdot 2^3 + b_2 \cdot 2^2 + b_3 \cdot 2^1 + b_4 \cdot 2^0 - 7$. More useful codings clearly follow. One of the strengths of the genetic algorithm is that short substrings, called schemata, contain important information. With this coding example, if one is maximizing the function $f(x) = x^3$ on $[8, -7]$, the substring $b_1 = 1$ would be quite important. Although the analysis is rather involved, it turns out that for a population of size n, there are on the order of n^3 schemata. Thus, many different characteristics of a population can be explored with a relatively small population.

The algorithm starts with a population of n strings, with the elements of each string assigned a random value. The strings are then decoded and the function to be optimized is evaluated. The fittest strings (those that produce the largest function value if maximizing the function) are selected and mated with other fit strings. There are many actual selection procedures, including those with and without replacement, as well as stochastic and deterministic methods. The one used below comes from Goldberg: a string's probability for selection is weighted by $f_i/\sum f_i$, where f_i is the function value generated by the ith string and $\sum f_i$ is the sum of all function values in a generation. The most basic mating mechanism is simple crossover: two strings interchange after a randomly selected point. For instance, with binary strings $b_1b_2b_3b_4b_5b_6$ and $B_1B_2B_3B_4B_5B_6$, interchange after the fourth element yields $b_1b_2b_3b_4B_5B_6$ and $B_1B_2B_3B_4b_5b_6$. Mating and selection continues until all members are chosen for the next generation. There is also a small probability of mutation during mating. With a binary coded string, this is implemented as a nonzero probability that a bit may switch. It turns out that mutation is not very important, and that too high a rate of mutation actually hinders the optimization process by adding unnecessary noise to the system.

This process of creating new generations from old ones by selection, mating, and mutation continues until some termination criterion comes into play. Simple implementations may run interactively and the researcher terminates the process when appropriate.

We obtained the Pascal code for the simple genetic algorithm outlined in Goldberg (1989) and modified it for functions with vector inputs. A variation of the above coding scheme for integers was implemented with a user-defined floating point. A full 32 or 64 bits for each element was not implemented because the lower-order bits would be useless until the optimal higher-order bits are determined. Let the vector of inputs be $X = (x_1, \ldots, x_n)$. As the above discussion illustrates, with crossover mating key information should be in short strings, so the largest part of all x_i's are coded next to each other, then the next largest are next to each other, and so on.

To gain experience with the algorithm before using it on any of the test functions, a unimodal function of two variables was optimized:

$$f(x_1, x_2) = \mathrm{e}^{-\beta \cdot (x_1^2 + x_2^2)}.$$

Running for 40 generations with a population of 30 in each generation and with $\beta = 1$, the simple genetic algorithm experienced difficulty. In no generation did the average value of the function exceed 0.84 (the maximum value of this function is 1.0). In fact, after quick initial improvement for the first few generations, the average value varied between 0.70 and 0.84.

In this problem, the average function value in a generation is only slightly worse than the best function value, while the worst is substantially worse (for instance, in the tenth generation the average was 0.73, the best 0.85, and the worst 0.21). Thus, with the usual selection rules, the genes that produce average values are almost as likely to be selected as the best genes and little improvement occurs.

To surmount this problem, Goldberg recommends using a scaling function, which transforms a generation's function values into a distribution more likely to lead to better performance. In this case, a scaling function should stretch the distance between the average and largest values so more superior genes are selected. Unfortunately, linear scaling does not work with this population. We developed a third scaling function, based on a quadratic, that did provide some improvement in average function value performance.

With $\beta = 0.1$, the simple genetic algorithm reaches a higher average function value (to a maximum of 0.96) without scaling because there is less variation between the average, minimum, and maximum function values in each generation. In part, this is because with $\beta = 0.1$ the function is much broader than with $\beta = 1$, so smaller values are less likely. Thus, different scaling functions are needed for different functions. Further, as the $\beta = 1$ example shows, it may be hard to find the appropriate scaling function.

Next, the Judge function was explored. This function is written in minimization form. In this form all values are positive (recall that it is the sum of squares), and for values far from the minimum the function values are substantial. However, the simple genetic algorithm maximizes functions and its selection rule requires positive function values. With these considerations in mind and f as the function value, we maximized $10^6 - f$.

The simple genetic algorithm experienced difficulty when nearing the optimum. The problem stems from the selection rule, where the probability of selecting the ith string is weighted by $f_i/\sum f_i$. Near the optimum, f_i is approximately 999,984 (recall the unadjusted minimum f is 16.0817). Thus, near the optimum, there is little relative difference between the function values produced by good and bad strings. As a result, there is very little preference given to good strings in selection for mating.

While this problem might seem specific to this function, it actually stems from the selection rule, which has difficulty selecting between close function values. Thus, it will experience difficulty with any relatively flat surface. At this point it became apparent that for continuous problems, the genetic algorithm is in need of further development. With each function we optimized, we encountered difficulty with the algorithm and were forced to modify it. While a very knowledgeable user might be able to correct these problems, the typical user of an optimization algorithm should not be expected to modify the algorithm to suit the function. Perhaps with more work, the genetic algorithm will become more usable by less experienced users for continuous function problems. It should be noted that very few of the references given by Goldberg for continuous function optimization with the genetic algorithm comes from peer-reviewed sources. One particular refinement Goldberg is currently working on is with so-called messy genetic algorithms. These algorithms more closely mimic natural selection and, in particular, they have the capability of finding isolated optima, something that proves troublesome for many algorithms.

References

AIXTIPS NEWS 92015, IBM AIX Field Support Center, Munich, January 22, 1992.

Aluffi-Pentini, Filippo, Valerio Parisi, and Francesco Zirilli, 1988, A global optimization algorithm using stochastic differential equations, ACM Transactions on Mathematical Software 14, 344–365.

Baum, Eric B., 1988, Neural nets for economists, in: Philip W. Anderson, Kenneth J. Arrow, and David Pines, eds., The economy as an evolving complex system (Addison-Wesley, New York, NY).

Bohachevsky, Ihor O., Mark E. Johnson, and Myron L. Stein, 1986, Generalized simulated annealing for function optimization, Technometrics 28, 209–217.

Carnevali, P., L. Coletti, and S. Patarnello, 1985, Image processing by simulated annealing, IBM Journal of Research and Development 29, 569–579.

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, 262–280.

Cramer, J.S., 1986, Econometric applications of maximum likelihood methods (Cambridge University Press, New York, NY).

Dennis, J.E. and Robert B. Schnabel, 1983, Numerical methods for unconstrained optimization and nonlinear equations (Prentice-Hall, Englewood Cliffs, NJ).

Derwent, Dick, 1988, A better way to control pollution, Nature 331, 575–578.

Dongarra, Jack J., 1991, Performance of various computers using standard linear equations software, Sept. (Computer Science Department, University of Tennessee, Knoxville, TN).

Drexl, A., 1988, A simulated annealing approach to the multiconstraint zero–one knapsack problem, Computing 40, 1–8.

Ferrier, Gary D. and C.A. Knox Lovell, 1990, Measuring cost efficiency in banking: Econometric and linear programming evidence, Journal of Econometrics 46, 229–245.

Finch, Stephen J., Nancy R. Mendell, and Henry C. Thode, Jr., 1989, Probabilistic measures of adequacy of a numerical search for a global maximum, Journal of the American Statistical Association 84, 1020–1023.

Gallant, A. Ronald and Halbert White, 1989, On learning the derivatives of an unknown mapping with multilayer feedforward networks, Neural Networks 4.

Gelsinger, Patrick P., Paolo A. Gargini, Gerhard H. Parker, and Albert Y.C. Yu, 1989, Microprocessors circa 2000, IEEE Spectrum 26, no. 10, 43–47.

Goffe, William L., Gary D. Ferrier, and John Rogers, 1992, Simulated annealing: An initial application in econometrics, Computer Science in Economics and Management 4.

Goldberg, David E., 1989, Genetic algorithms in search, optimization and machine learning (Addison-Wesley, Reading, MA).

Judge, George G., W.E. Griffiths, R. Carter Hill, Helmut Lütkepohl, and Tsoung-Chao Lee, 1985, The theory and practice of econometrics, 2nd ed. (Wiley, New York, NY).

Kan, A.H., G. Rinnooy, and G.T. Timmer, 1987, Stochastic global optimization methods, Part II: Multilevel methods, Mathematical Programming 39, 57–78.

Kendall, M.G., 1961, A course in the geometry of *n* dimensions (Hafner, New York, NY).

Kendall, Maurice and Alan Stuart, 1978, The advanced theory of statistics, 4th ed. (Macmillan, New York, NY).

Kirkpatrick, S., C.D. Gelatt, Jr., and M.P. Vecchi, 1983, Optimization by simulated annealing, Science 220, 671–680.

Kuan, Chung-Ming and Halbert White, 1991, Artificial neural networks: An econometric perspective, Working paper (Economics Department, University of Illinois, Urbana-Champaign, IL).

Linderholm, Owen and Trevor Marshall, 1992, In the PowerPC chips, Byte 17, no. 2, 100.
Press, William H., Brian Flannery, Saul A. Teukolsky, and William T. Vetterling, 1986, Numerical recipes: The art of scientific computing (Cambridge University Press, New York, NY).
Pronzato, Luc, Eric Walter, Alain Venot, and Jean-Francois Lebruchec, 1984, A general-purpose optimizer: Implementation and applications, Mathematics and Computers in Simulation 24, 412–422.
Quandt, Richard E., 1983, Computational problems and methods, in: Zvi Griliches and M.D. Intriligator, eds., Handbook of econometrics, Vol. 1 (North-Holland, New York, NY).
Renpu, G.E., 1990, A filled function method for finding a global minimizer of a function of several variables, Mathematical Programming 46, 191–204.
Salemi, Michael K., 1986, Solution and estimation of linear rational expectations models, Journal of Econometrics 31, 41–66.
Telly, H., Th.M. Liebling, and A. Mocellin, 1987, Reconstruction of polycrystalline structures: A new application of combinatorial optimization, Computing 38, 1–11.
Tunnicliffe-Wilson, G., 1973, Estimation of parameters in multiple time series models, Journal of the Royal Statistical Society B 20, 76–85.
Vanderbilt, David and Seven G. Louie, 1984, A Monte Carlo simulated annealing approach to optimization over continuous variables, Journal of Computational Physics 56, 259–271.
Varhol, Peter D., 1991, Fast 486 performance and compatible processors, Personal Workstation 3, 67–69.
Wasserman, Philip D. and Tom Schwartz, 1988, Neural networks, Part 2: What are they and why is everybody so interested in them now?, IEEE Expert, Spring, 10–15.
Wong, D.F., H.W. Leong, and C.L. Liu, 1988, Simulated annealing for VLSI design (Kluwer Academic Publishers, Boston, MA).
Woo, Wing T., 1985, The monetary approach to exchange rate determination under rational expectations: The dollar–deutschmark rate, Journal of International Economics 18, 1–16.