(2)

NAMRL Technical Memorandum 90-2

ANALYSIS OF NAVAL AVIATION SELECTION TEST DATA WITH NONLINEAR MODELS. PART I. PARAMETER ESTIMATION

D.J. Blower

Naval Aerospace Medical Research Laboratory
Naval Air Station
Pensacola, Florida 32508-5700

(2)

NAVAL AEROSPACE MEDICAL RESEARCH LABORATORY
NAVAL AIR STATION, PENSACOLA, FLORIDA 32508-5700

NAMRL Technical Memorandum 90-2

ANALYSIS OF NAVAL AVIATION SELECTION TEST DATA WITH NONLINEAR MODELS. PART I. PARAMETER ESTIMATION

D.J. Blower

Reviewed and approved 31 Dec 1990

J. A. BRADY, CAPT, MSC USN
Commanding Officer

This research was sponsored by the Naval Medical Research and Development Command under work unit 63706N-M0096-001-7007.

Volunteer subjects were recruited, evaluated, and employed in accordance with the procedures specified in Department of Defense Directive 3216.2 and Secretary of the Navy Instruction 3900.39 series. These instructions are based upon voluntary informed consent and meet or exceed the provisions of prevailing national and international guidelines.

The views expressed in this article are those of the authors and do not reflect the official policy or position of the Department of the Navy, Department of Defense, or the U.S. Government.

Trade names of materials and/or products of commercial or nongovernment organizations are cited as needed for precision. These citations do not constitute official endorsement or approval of the use of such commercial materials and/or products.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1990	Interim Dec 88 - Dec 89

4. TITLE AND SUBTITLE
Analysis of Naval Aviation Selection Test Data with Nonlinear Models. Part I. Parameter Estimation

5. FUNDING NUMBERS
63706N M0096 001 7007

6. AUTHOR(S)
David J. Blower

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Aerospace Medical Research Laboratory
Bldg. 1953, Naval Air Station
Pensacola, FL 32508-5700

8. PERFORMING ORGANIZATION REPORT NUMBER
NAMRL TM 90-2

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Naval Medical Research and Development Command
National Naval Medical Center, Bldg. 1
Bethesda, MD 20814-5044

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The purpose of this paper is basically tutorial in nature and, as such, describes an algorithm for estimating the parameters of a nonlinear model. This algorithm is called "simulated annealing." The actual workings of this algorithm are examined in some detail. The reason for studying this algorithm is because statistical analysis of naval aviation selection test data has always relied on the use of linear regression models. Linear models represent only a small subset of possible mathematical models that could be used as an empirical tool to predict aviator performance. Specifically, the whole class of nonlinear models has not been addressed. Recent research into neural networks and parallel distributed processing has uncovered some interesting nonlinear models. We intend to reanalyze the test scores of student naval aviators with a nonlinear model borrowed from the neural network literature. We hope that this new class of nonlinear models will be a more powerful tool in predicting aviator performance and will result in an improved naval aviator selection test battery.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Aviation Selection, Nonlinear Prediction Models, Simulated Annealing	31
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

SUMMARY

THE PROBLEM

The problem addressed in this research report concerns the estimation of parameters for a nonlinear prediction equation. Such nonlinear equations arise in selection research when performance-based tests are used as predictors of flight performance during training and a neural network formulation is employed to capture the statistical relationship between scores on the test and training performance. Software packages for desktop computers are not readily available to estimate parameters for nonlinear equations.

FINDINGS

The inner workings of an optimization technique called simulated annealing were outlined in some detail. A computer program was written in BASIC to implement the algorithm and was then applied to artificial data. These data are typical of what might arise from a neural network approach to generating a prediction equation. Simulated annealing was successful in finding solutions that, on the average, were very close to the global minimum for the defined problem. This research supports the claim that simulated annealing can be used as a heuristic tool to estimate parameters for nonlinear models.

RECOMMENDATIONS

Simulated annealing appears to be a powerful all-around tool and should be investigated further on actual selection research data. In addition to its utility as a numerical optimization technique, it has been used to investigate learning in neural networks and in artificial intelligence approaches to problem decomposition. Simulated annealing was very time consuming, and its performance on this highly simplified problem should not be construed as a blanket recommendation when scaling up to real-world problems. This research did not compare simulated annealing with other well-known techniques for nonlinear parameter estimation. When formulated as a sum-of-squares problem, these other techniques may be more suitable than simulated annealing.

ACKNOWLEDGMENTS

I would like to acknowledge the support of my colleagues in the Aviation Psychology Division and Dr. John DeLorge, Head of the Bioenvironmental Assessment Department, for providing a congenial environment in which to pursue this work.

INTRODUCTION

Statistical analysis of naval aviation selection test data has always relied on the use of linear regression models. Linear models represent only a small subset of possible mathematical models that could be used as an empirical tool to predict aviator performance. Specifically, the whole class of nonlinear models has not been addressed.

Recent research into neural networks and parallel distributed processing has uncovered some interesting nonlinear models (1). We intend to reanalyze the test scores of student naval aviators with a nonlinear model borrowed from the neural network literature. We hope that this new class of nonlinear models will be a more powerful tool in predicting aviator performance and will result in an improved naval aviator selection test battery.

A computer-based algorithm to find the parameters of the model is necessary before these new nonlinear models can be applied to selection data. Standard algorithms exist for finding the regression weights of linear regression models, and these techniques are readily available in the statistical packages we use.

The situation is not as fortunate with regard to nonlinear models. Although many techniques are described in the literature, none of them are standard features in the statistical packages.

The purpose of this paper is basically tutorial in nature and, as such, describes an algorithm for finding the parameters of a nonlinear model. This algorithm is called "simulated annealing." The actual workings of this algorithm are examined in some detail. Several computer programs written in BASIC were developed for this report. One of these is described in Appendix A. These programs generated the data analyzed and discussed in the report. These programs also form the foundation for the actual machinery of analyzing real world selection data.

Appendix B contains the mathematical background for some of the expressions used in this report, as well as the rationale for the Monte Carlo approach underlying the algorithm for finding the weights of the nonlinear model.

WHAT IS SIMULATED ANNEALING?

Simulated annealing is a term used to describe a new technique for finding the optimum value of a complicated function with many variables (2,3). The word "annealing" actually describes the physical process used in treating metals and glass by heating the material to a high temperature and then slowly cooling the material according to a temperature

schedule. The purpose of this treatment is to reach a ground state so that certain favorable properties such as a lack of brittleness in the metal or uniformity in the glass are achieved.

Simulated annealing has a goal of finding the minimum of a function of many variables by first "heating" the system being optimized at a high temperature and then lowering the temperature in slow stages until the system "freezes." The sequence of temperatures, including the melting point and freezing point, is called an annealing schedule.

Technically, simulated annealing is a stochastic-search technique with a control parameter represented by the temperature. The algorithm attempts to evaluate a complicated multidimensional integral by a Monte Carlo sampling approach. The Monte Carlo technique is necessary because the integral is too complicated to solve through conventional analytical means (4). Because of the complicated nature of the integral, the Monte Carlo approach follows a scheme designed to provide better sampling of the states where major contributions to the integral are made (5). (See Appendix B.)

The Metropolis algorithm (6), used for many years by statistical physicists, implements such a sampling scheme and provides the numerical method for actually calculating the needed values. When the Metropolis algorithm is augmented by the addition of a temperature parameter, it provides the computational basis for developing a computer program to carry out simulated annealing (7).

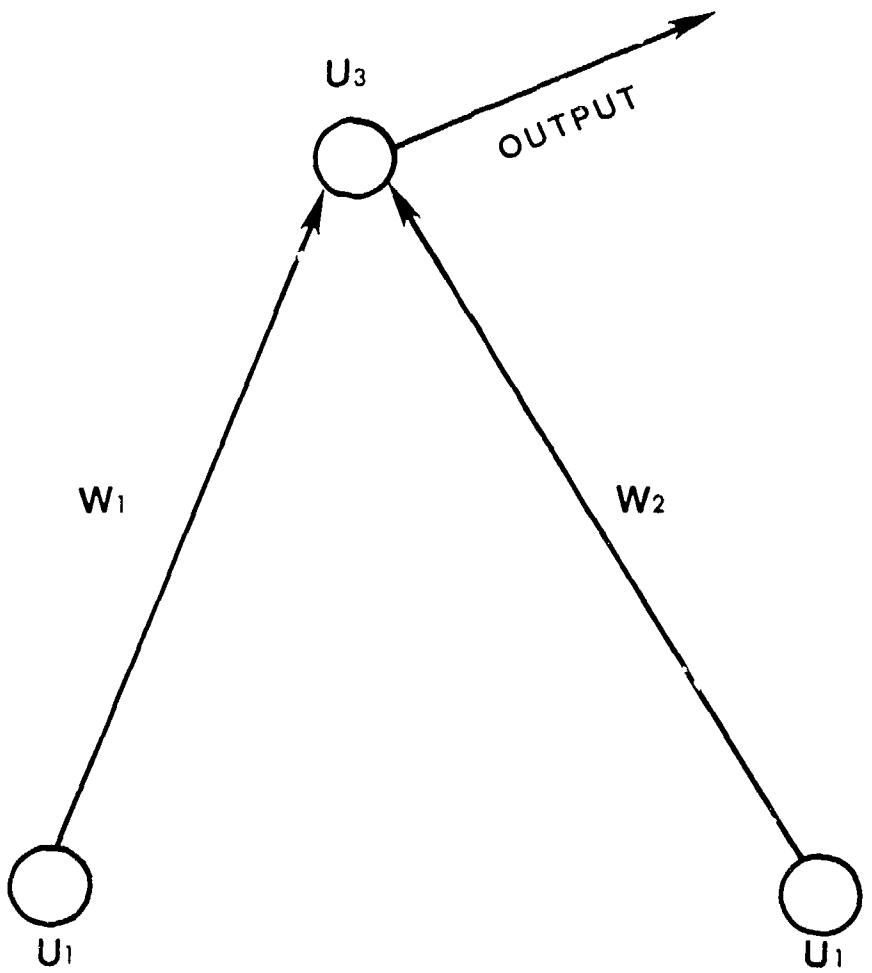
A SIMPLE NEURAL NETWORK

The class of nonlinear models in this research effort stem from neural network paradigms (1). One of the simplest neural networks possible is shown in Fig 1. This is the network that will be studied in this paper.

The three circles labelled u_1, u_2, and u_3 are called processing nodes. They are analogous to neurons. The pattern of connectivity among the processing nodes is shown in Fig. 1 with the output of unit 1 and unit 2 flowing into unit 3. The output of unit 3 is the final output of the neural network system. The weights, w_1 and w_2, modify the output from units 1 and 2. Because these weights can be positive or negative, they are analagous to the excitatory or inhibitory influence of neurons on other neurons they are connected to.

The output of processing node unit 3 is a nonlinear function of the sum of its inputs. This nonlinear relationship, when graphed, has a sigmoid shape as illustrated in Fig 2. This type of function mimics the typical firing rate of an idealized neuron. If the input to the neuron is below some threshold, output becomes negligible or is absent. As we reach the slope of the S-shaped curve, the firing rate increases with increasing input. Finally, at the top of the curve, we see that the firing rate saturates; increased input does not

$$\text{Output from Unit 3} = \frac{1}{1 + e^{-(W_1 U_1 + W_2 U_2)}}$$



SIMPLE NEURAL NET

Figure 1. A diagram of a simple neural network showing the output as a function of the two input units and the associated weights.

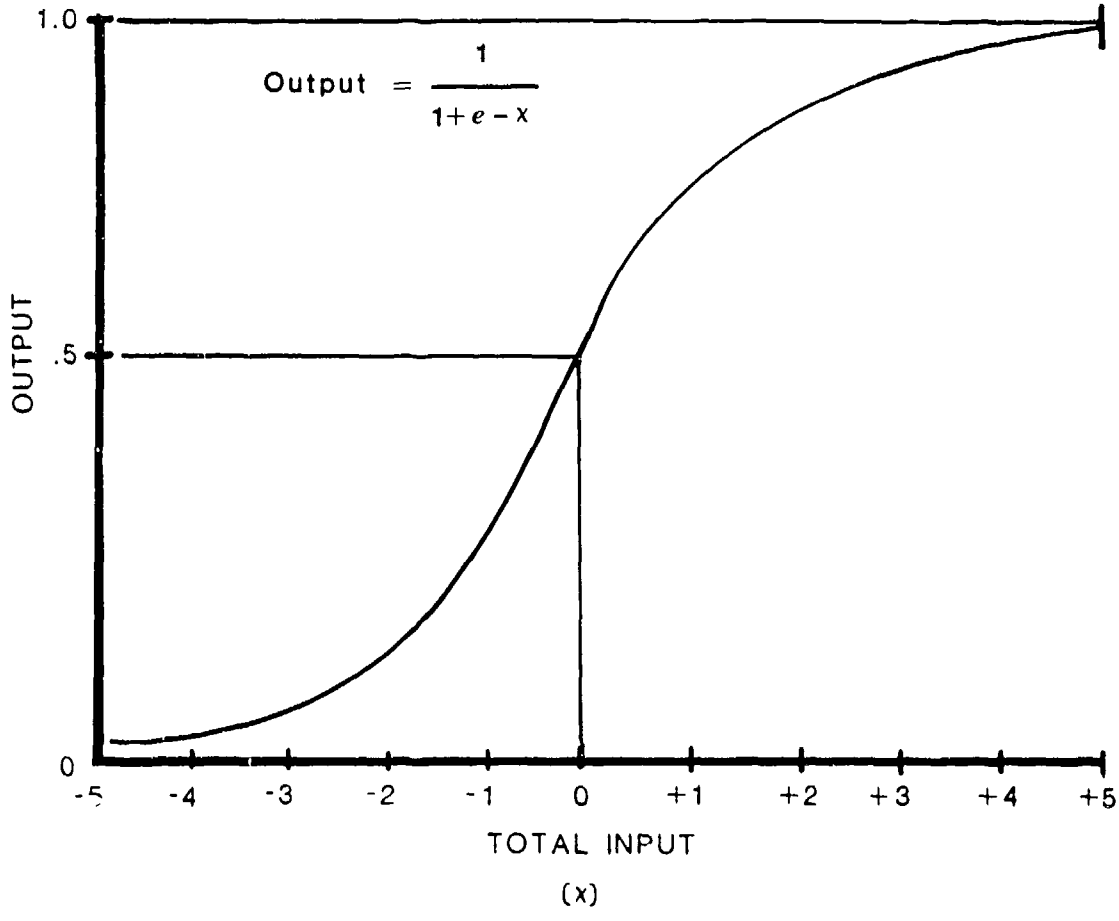


Figure 2. The general shape of the nonlinear function used to determine the output of a processing node.

result in an ever increasing output. This description essentially captures the notion of nonlinearity.

The explicit nonlinear function used by many researchers is also the one used in this paper:

$$y = 1/(1 + e^{-x}) \tag{1}$$

where y is the output from unit 3, and x equals the sum of the weighted inputs from units 1 and 2. ($x = w_1u_1 + w_2u_2$.)

A NONLINEAR MODEL FOR PREDICTION

The processing nodes u_1 and u_2 represent the scores on selection tests, the weights w_1 and w_2 represent adjustable parameters, and the output of u_3 is the predicted value of some criterion variable. Just as in linear regression models, we seek to minimize the sum of squared errors, that is, we try to find those values of w_1 and w_2 that result in the minimum value of

$$SSE = \sum_{i=1}^{N}(predicted(i) - observed(i))^2 \tag{2}$$

In keeping with the tutorial nature of this report, we choose a situation with values small enough so that the examples can be easily followed. This simple illustrative problem should not be confused with the real values of an actual problem.

Let $N = 4$ cases; that is, test results are available for two tests on four subjects, where u_1 equals the result on test 1, and u_2 equals the result on test 2. The test scores are scaled to have values between zero and one. The weights are restricted to three distinct values: -1, 0, and +1. Because two weights can each assume three values, the parameter space for this problem consists of nine possible states. These nine states are enumerated in Table 1.

Table 2 contains an example of how the sum of squared error (SSE) is calculated. The columns labelled u_1 and u_2 represent the processing nodes unit 1 and unit 2. The values under these columns are the results of the two tests. These test scores are scaled such that their values lie between 0 and 1. We consider the situation where the weights from state 4 ($w_1 = 0$ and $w_2 = -1$) are used by the model to generate the predicted values listed in the fourth column. For the purposes of this simulation the true state of the world is arbitrarily chosen to be state 9 where $w_1 = +1$ and $w_2 = +1$. When reality acts in accordance with these parameters we obtain the observed values of the fifth column. The sum of the squared discrepancies over all four subjects is calculated as .605.

Table 1: A listing of the nine parameter states with associated weights

State	w_1	w_2
1	−1	−1
2	−1	0
3	−1	+1
4	0	−1
5	0	0
6	0	+1
7	+1	−1
8	+1	0
9	+1	+1

Table 2: The calculation of the sum of squared error (SSE) when the weights of state 4 are used to predict the world, but the weights of state 9 represent the "true" observed world.

Case	u_1	u_2	Predicted	Observed	$(p-o)^2$
1	.10	.90	.289	.731	.195
2	.70	1.00	.269	.846	.333
3	.30	.40	.401	.668	.071
4	.20	.05	.488	.562	.006

$$\sum_{i=1}^{4}(p-o)_i^2 = 605$$

Table 3: A list of the sum of squared error (SSE) for all nine states.

State	w_1	w_2	SSE	Comment
1	−1	−1	.8197	
2	−1	0	.4001	
3	−1	+1	.1056	Local Minimum
4	0	−1	.6046	
5	0	0	.2049	
6	0	+1	.0208	Local Minimum
7	+1	−1	.3916	
8	+1	0	.0829	
9	+1	+1	.0000	Global Minimum

Table 3 shows the same SSE calculation for all nine possible states of the system. States 3 and 6 represent local minima, and state 9 (by definition) is the global minimum for the SSE function. The local minima in Table 3 are defined relative to the linear listing of the states and *not* in relation to the disposition of the weights in parameter space.

In linear regression models, the solution to finding the regression weights that minimize the SSE are well known and easily implemented using standard algorithms. This paper describes an algorithm, which, like the ones for linear regression models, will find values for the set of weights that result in the minimum value of the SSE function that arises from the use of a nonlinear model. Simulated annealing is a technique to handle just such a problem.

A DETAILED LOOK AT THE METROPOLIS ALGORITHM

The major component of simulated annealing is the Metropolis algorithm, named after its inventor (5). Because the algorithm was originally designed to deal with problems arising from statistical mechanics, terms such as "energy" and "temperature" are used. Such terms are retained here because of their metaphorical advantage.

At the most general level we are trying to solve an optimization problem. That is, we are trying to find the minimum value of the SSE generated from a nonlinear model. To comply with the terminology used by the Metropolis algorithm, we equate "Sum of Squared Error" (SSE) of Equation (2) with the term "Energy."

The Metropolis algorithm works in the following fashion.

1. The process starts with the system in some arbitrarily chosen state. For example, the process might start out in state 2 where w_1 = -1 and $w_2 = 0$.

2. A new state of the system is generated by applying a set of moves to the system. A transition matrix defined in Fig. 3 carries out these moves. From state 2, this transition matrix shows a 50% chance of moving into state 1 and a 50% chance of moving into state 3.

3. Move to a new system according to the transition matrix by generating a random number. If the random number is less than or equal to .50, then move to state 1, otherwise move into state 3.

4. Calculate the difference in energy (called ΔE) between the new state just entered and the old state. For example, if the random number was .37, a move into state 1 is called for. From Table 3, calculate $\Delta E = E(final) - E(initial) = .820 - .401 = .419$. If the random number had been .74, then we would have moved into state 3. In this case, $\Delta E = E(final) - E(initial) = .106 - .401 = -.295$

5. If $\Delta E \leq 0$, then we accept the move into the new state. In the case where we moved into state 3, $\Delta E \leq 0$, so we would accept this move.

6. If $\Delta E > 0$, then we accept the move into the new state with the probability $e^{-\Delta E/T}$. In the case where we moved into state 1, $\Delta E > 0$. The probability of accepting this move is $e^{-.419/T}$ where T is the temperature parameter. If T = 1, the probability of accepting the move equals .6577. We generate another random number, and if this number is less than .6577, we accept the move. Otherwise, we reject this move into the new state.

7. In moving from state to state by following the above recipe, we keep a running total of the weights attached to the states entered. At the end of a series of moves at a fixed temperature, we average these weights. These average weights, when rounded up or down, serve as the initial values of the ws for the next series of runs.

Appendix C contains the results from three representative runs of a computer simulation of 40 cycles of the Metropolis algorithm for the example studied in this paper. Each table shows the initial random starting state and the temperature parameter, which was fixed at 1 for these examples. Each cycle shows the old state in which we started and the new state we moved into using the transition matrix. The next column shows ΔE. If $\Delta E \leq 0$ we accept the new state. Where $\Delta E > 0$, we calculate $e^{-\Delta E/T}$. This calculation is shown under the column labelled X. The next column over is the random number generated when deciding whether to probabilistically accept the new state when $\Delta E > 0$. If the random number is less than X, then the new state is accepted. If the random number is greater than X, reject the move into the new state and keep the old state. In the next two columns, the running total of the weights for all the states visited so far is listed. The final column indicates whether the state was accepted or rejected.

		NEW STATE								
		S1	S2	S3	S4	S5	S6	S7	S8	S9
OLD STATE	S_1	0	1/2	0	0	0	0	0	0	1/2
	S_2	1/2	0	1/2	0	0	0	0	0	0
	S_3	0	1/2	0	1/2	0	0	0	0	0
	S_4	0	0	1/2	0	1/2	0	0	0	0
	S_5	0	0	0	1/2	0	1/2	0	0	0
	S_6	0	0	0	0	1/2	0	1/2	0	0
	S_7	0	0	0	0	0	1/2	0	1/2	0
	S_8	0	0	0	0	0	0	1/2	0	1/2
	S_9	1/2	0	0	0	0	0	0	1/2	0

Figure 3. Transition Matrix showing probability of possible moves from state to state in the Metropolis algorithm.

These simulations exhibit several key aspects of the Metropolis algorithm as it has been explained above. Looking at cycle 1 of the first run, the starting state was arbitrarily chosen as state 7. A random number greater than .5 led to the decision to go to state 8 as opposed to state 6. Because ΔE was -.309 we transitioned to a state of lower energy, which we automatically accept. That is, we have found a state where the energy is lower and therefore is a step in the right direction. We then add the weights of state 8 ($w_1 = +1$ and $w_2 = 0$) to the running total of weights.

Cycle 2 starts out in state 8. The transition matrix indicates that we have a 50% chance of moving into state 9 or a 50% chance of moving back into state 7. In fact, the random number generated was less than .5 so we moved back into state 7. Unlike the first cycle, cycle 2 moves from a region of lower energy to a region of higher energy (smaller SSE to larger SSE). That is, in this cycle, $\Delta E > 0$, and we have a probability defined by $e^{-\Delta E/T}$ of either staying in the same state by rejecting the proposed move or moving into the state despite the fact that it is in a region of higher energy. This is resolved by looking at the column labelled X and finding that for this ΔE and this temperature, the probability is .734 of accepting the move with the larger energy. There is a corresponding probability of $1 - e^{-\Delta E/T} = .266$ of rejecting this move and staying in state 8. Because the random number .259 is less than .734, the move back into state 7 is accepted. The weights in state 7 are $w_1 = +1$ and $w_2 = -1$ so the running total is appropriately incremented.

This feature of accepting some states even when they lead to a higher value of the function to be minimized is different from other minimization algorithms. It allows for certain controlled "up-hill" moves in the quest for the minimum.

At cycle 16, state 6 transitions to state 5, which is a move from a state of lower energy to a state of higher energy. Even though the probability is .832 that this move will be accepted, the lower probability of a rejection prevails and the move is rejected. In this case, weights for state 6 ($w_1 = 0$ and $w_2 = +1$) are added again to the running total.

One other feature needs to be pointed out, although it was implicit in the transition matrix of Fig. 3. The states "wrap-around" so that state 9 can transition to state 1, and state 1 can transition to state 9, as in cycles 13 and 14 of Table 5.

After 40 cycles, we have sampled a number of states and have procured some information concerning the values for the weights in our nonlinear model. The values for w_1 and w_2 averaged over the 40 cycles for the 3 runs are shown in Table 4.

Because our weights are restricted to three discrete values, we determine which allowable values the weights are closest to. In run 1, -.10 rounds up to 0, and .08 rounds down to 0. These revised weights are shown in the fourth and fifth columns. The last column shows the state that corresponds to these revised weights. Performing the same procedure for the other two runs, we can see what states the algorithm has selected.

Table 4: The values for the two weights found by the Metropolis algorithm at the end of a 40 step cycle. Data are from three simulation runs.

Run	w_1	w_2	w_1'	w_2'	State
1	-.10	.08	0	0	5
2	.83	.18	1	0	8
3	-.33	.33	0	0	5

Table 5: The effect of lowering the temperature at a fixed ΔE on the probability of transitioning to a new state.

ΔE	Temperature(T)	$\Delta E/T$	Probability
.5	10.00	.05	.9512
.5	1.00	.50	.6065
.5	.50	1.00	.3679
.5	.10	5.00	.0067
.5	.01	50.00	.0000

THE TEMPERATURE PARAMETER AND SIMULATED ANNEALING

We have set up the example so that state 9 with weights $w_1 = +1$ and $w_2 = +1$ is the model generating the observed data. How can the Metropolis algorithm, as defined so far, find this desired state? This question leads us directly into a discussion of simulated annealing, which is nothing more than the use of the Metropolis algorithm at a series of temperatures ranging from very high ("hot") to very low ("cold"). In simulated annealing, we first heat the system to a very high temperature and then slowly cool it down in stages to a minimum energy state or "ground state."

The ground state will point to a minimum of the SSE function for the nonlinear model and, therefore, to the appropriate weights to use in the model. The series of temperatures employed is called the "annealing schedule," and the plot on a log-log scale of Temperature versus Energy is called the "annealing curve."

Intuitively, as the temperature is raised, states with higher energy are more likely to be accepted. Conversely, as the temperature is lowered, moving into a state of higher energy becomes more difficult. Table 5 presents some numbers to make this idea more concrete.

Choosing a ΔE of .50, which is typical for the example problem of this paper, illustrates

the effects of lowering the temperature parameter from a hot temperature of 10 to a cold temperature of .01. At a high temperature such as 10, the probability (.9512) dictates that any move where $\Delta E = .50$ will most likely be accepted. As the temperature is lowered, this probability gradually decreases so that moves to a higher energy become less and less frequent.

Another way of looking at how simulated annealing works to find the minimum through the temperature parameter is considered in the following numerical argument. Assume state 9 is reached through the usual process of sampling states with the Metropolis algorithm. Also assume that the point on the annealing curve is reached where the temperature has cooled down to a value of .1 From state 9, we can transition to either state 8 or state 1. If the system transitions to state 1, then $\Delta E = .8197$ (See Table 3) and $e^{-\Delta E/T} = .00027$. It is, therefore, highly unlikely that state 9 will ever transition to state 1 at this cold temperature. The move to state 1 will almost always be rejected, and the system will stay in state 9 where the "correct weights" ($w_1 = +1$ and $w_2 = +1$) will likewise almost always be counted.

If the other possible transition to state 8 were to take place, then $\Delta E = .0829$ and $e^{-\Delta E/T} = .4364$. The system will choose this transition less than half the time as compared to a temperature of 1 when it would be accepted 92% of the time. Of course, state 8 is a more favorable state than state 1 when trying to find the minimum of the SSE function.

Thus, at high temperatures, the system easily jumps from state to state despite high energy barriers (ΔE), and, therefore, escapes from local minima. As the temperature is lowered, the energy barriers become increasingly more difficult to overcome and the system should "freeze" into states closest to the ground state. The system will then spend most of its time in the ground state where the true global minimum is located.

Table 6 presents the results from a simulation run to ascertain whether the simulated annealing approach is effective in finding the minimum of the SSE function. Each row represents significant data at 13 different temperatures ranging from hot (T=100) to cold (T=.01). The data in each row were generated by the same program that produced the data for Appendix C, but in this case, the detail for each cycle has been suppressed.

The second and third columns give w_1 and w_2 averaged over 50 cycles. The fourth column shows how often the proposed move was actually accepted. As the temperature decreased, the ratio of accepted moves also declined. The next-to-last column gives the energy (SSE) averaged over all 50 states visited at the listed temperature. This is the important information to be digested from this simulation run. It shows that the Energy steadily decreased as the temperature was lowered, and by the time the final temperature of .01 was reached, the global minimum of zero had been found. The last column shows the state closest to the average weights. This is the starting state which simulated annealing uses at the next lower temperature.

The last three rows illustrate the effect of taking the temperature parameter down to

Table 6: Simulated annealing finds the minimum Energy.

Temp	Avg w_1	Avg w_2	Accept. ratio	Avg Energy	State
100.00	-.20	-.10	.96	.347	5
10.00	.08	-.06	.96	.287	5
1.00	0.00	.16	.90	.276	5
.90	.14	.22	.78	.179	5
.80	.32	.16	.84	.180	5
.70	.12	.36	.72	.177	5
.60	.64	.20	.78	.150	8
.50	.40	.30	.72	.146	5
.40	.38	.30	.74	.144	5
.30	-.22	.60	.42	.132	6
.20	.58	.70	.32	.052	9
.10	1.00	.74	.32	.022	9
.01	1.00	1.00	.00	.000	9

a low value as was discussed above. At a temperature of .2, the algorithm entered state 9 after 50 cycles because that is the state the averaged weights were closest to at the end. The system started off in this state at the next lower temperature of .1. The weights at the end of the 50 cycles at this temperature were even closer to the true values of $w_1 = +1$ and $w_2 = +1$. So, at the final temperature of .01, the system started out in state 9 and never once left it on any of its 50 cycles.

In this particular case, simulated annealing worked to perfection. It found the state where the global minimum was located. Because simulated annealing is a heuristic technique, it has no known guarantees for always finding the global minimum. Nonetheless, as we have demonstrated here, this technique does show promise for getting close to the global minimum for this type of optimization problem.

DISCUSSION

The annealing curve shown in Fig. 4 encapsulates the main point of this paper. This curve shows the relationship between the objective function to be minimized and the temperature parameter as plotted on a log-log scale (8). Each point on the curve is the average of 30 simulation runs on the computer. At high temperatures (i.e., greater than 10, which is 1 on the log scale), the energy is simply the average of the energy of all the given states for our problem. All the states are essentially visited at random.

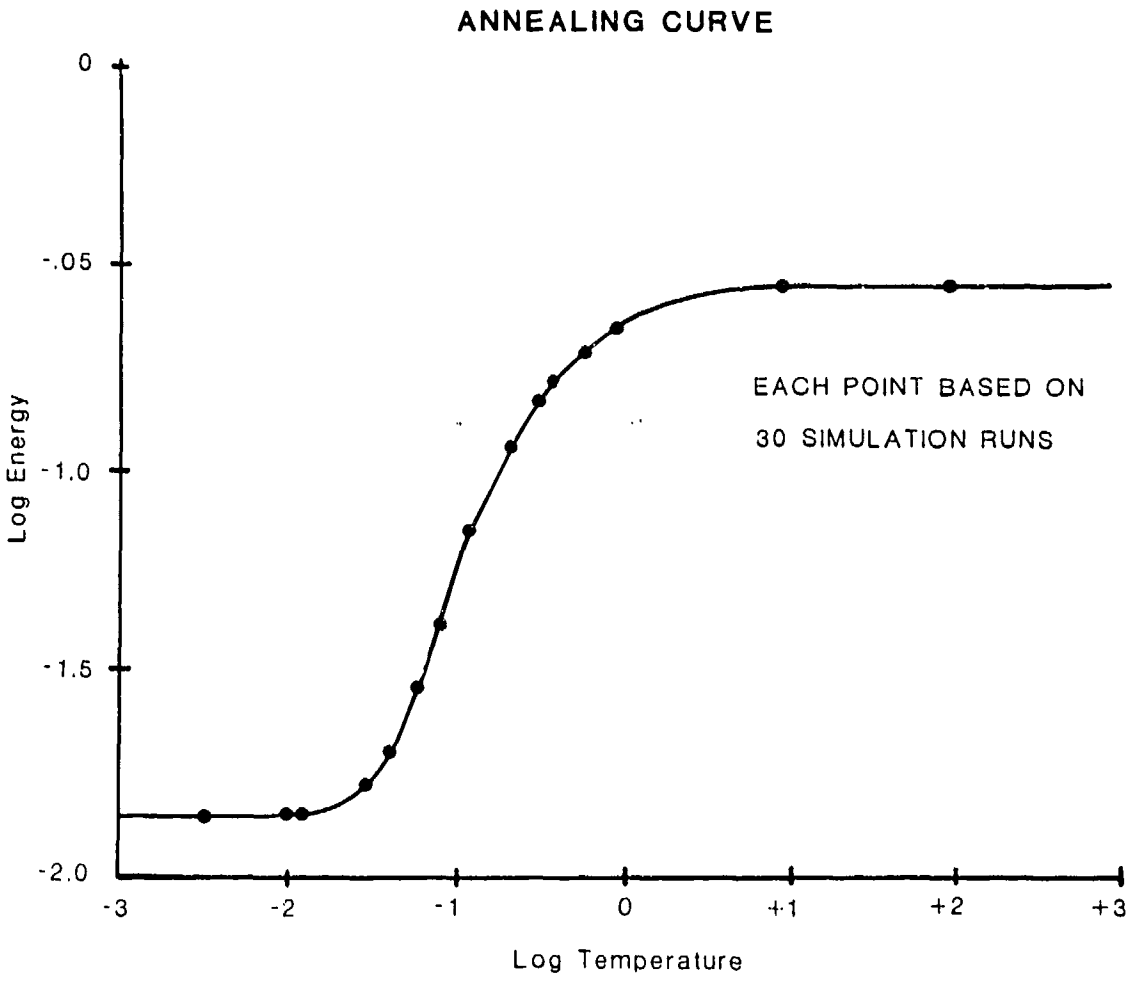


Figure 4. The annealing curve showing how the Energy function is minimized as the temperature parameter is lowered.

As the temperature slowly decreases, weights are found that result in a lower and lower energy. Finally, a temperature is reached where the energy no longer decreases. This value is about .014 in terms of energy (On the log scale this equals -1.85.) The true value of the global minimum is zero. Figure 4 shows, on the average, how close simulated annealing can come to the true global minimum. It certainly does a creditable job of finding the optimal weights we want to employ in the nonlinear model. This is basically what we require of simulated annealing as a practical tool.

CONCLUSION

In order to break new ground in developing mathematical models for predicting naval aviator performance, we have investigated certain nonlinear models inspired by neural network research. The estimation of parameters for these nonlinear models involves more advanced techniques. These numerical algorithms for parameter estimation are not readily available in our current statistical packages.

Therefore, this report has attempted to show how one such technique can be used as a heuristic device to estimate parameters. This parameter estimation technique uses a Monte Carlo approach that has been refined by statistical physicists over the past 35 years.

The paper went into some detail on the actual workings of the Metropolis algorithm and simulated annealing to illustrate just how successful this heuristic can be. The value of simulated annealing still remains to be verified for more realistic problems with much larger number of parameters.

REFERENCES

1. Rumelhart, D. and McClelland, J., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol 1: Foundations.* MIT Press, Cambridge, MA, 1986.

2. Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P., "Optimization by Simulated Annealing." *Science*, Vol. 220, No. 4598, pp. 671-680, 1983.

3. Hajek, B., "A Tutorial Survey of Theory and Applications of Simulated Annealing." *Proceedings of the 24th Conference on Decision and Control*, Ft. Lauderdale, FL, December 1985.

4. Valleau, J.P. and Whittington, S.G., "A Guide to Monte Carlo for Statistical Mechanics: 1. Highways." In *Statistical Mechanics Part A: Equilibrium Techniques* Berne, J. (Ed.), Plenum Press, New York, NY, 1977.

5. Hastings, W.K., "Monte Carlo Sampling Methods using Markov Chains and their Applications." *Biometrika*, Vol. 57, pp. 97-109, 1970.

6. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E., "Equation of State Calculations by Fast Computing Machines." *Journal of Chemical Physics*, Vol. 21, No. 6, pp. 1087-1092, 1953.

7. Koonin, S.E., *Computational Physics.* Benjamin/Cummings, Menlo Park, CA, 1986.

8. White, S.W., *Concepts of Scale in Simulated Annealing*, Research Report RC10661 (#47856), IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1984.

9. Pincus, M., "A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems." *Operations Research*, Vol. 18, pp. 1225-1228, 1970.

10. Newman, T.G. and Odell, P.L., "The Generation of Random Variates." *Number 29 of Griffin's Statistical Monographs and Courses.* In Alan Stuart (Ed.), Hafner, New York, NY, 1971.

11. Reif, F., *Statistical Physics*, McGraw-Hill, New York, NY, 1964.

12. Hammersley, J. and Handscomb, D., *Monte Carlo Methods*, Methuen & Co., Ltd., London, UK, 1964.

APPENDIX A

This Appendix contains a program written in GW BASIC, Version 3.15, running on a Zenith Z-248 (IBM 286 compatible machine) under MS-DOS Version 3.1. This program was used to generate the data in Appendix C. The lines of code that print out the various headings and the information for each cycle are omitted to make the program easier to follow. The documentation of the program follows.

Lines	Description
20	A matrix for the two weights for each of the nine states is set up.
30	The user inputs the temperature parameter.
40	The user inputs the number of cycles.
60-150	Reads in the two test scores for the four subjects and the true value of the output based on the model.
160-210	Reads in the weights.
220	Chooses a random starting state between 1 and 9.
280-380	Loops through the required number of cycles.
300	Calls subroutine 3000 to implement the calculations for each cycle. Subroutine 3000 calls subroutines 1000 and 2000.
1000	Subroutine to calculate the sum of squared errors (SSE) or what has been called the energy of the state. Determines energy based on four subjects over two tests for the initial state.
2000	Subroutine to implement the transition matrix of Fig. 3.
3010	Calculates energy for initial state.
3020	Transitions to new state.
3030	Calculates energy of the new state.
3040	Calculates ΔE, the difference in energy between the initial state and the transitioned state.

Lines	Description
320	Calculates $e(-\Delta E/T)$ and stores values in X.
350	Implementation of the Metropolis algorithm. If $\Delta E < 0$ or a random number < X, then the new state is accepted . (Subroutine 5000) If $\Delta E > 0$ and the random number > X, then the old state is retained (Subroutine 6000).
5000	Subroutine for accepting a new state. Calls subroutine 4000. The new state is now the old state for the next move.
6000	Subroutine for retaining the old state. Calls subroutine 4000. The old state remains as the old state for the next move.
4000	Subroutine to average the weights of either the new state or the old state as determined by the Metropolis algorithm.

PROGRAM LISTING

The following BASIC program was used to generate the data appearing in Tables 4, 5, and 6. The documentation appears on the previous page of this appendix. The program lines which output information to the printer have been omitted for the sake of clarity.

```
10 RANDOMIZE TIMER
20 OPTION BASE 1:DIM W(9,2)
30 INPUT "Temperature = ";TEMP
40 INPUT "Number of cycles =";NC
50 DEF FNCALC(D,T)=EXP (-D/T)
60 FOR I=1 TO 4
70 FOR J=1 TO 2
80 READ X(I,J)
90 NEXT J
100 READ TRUE(I)
110 NEXT I
120 DATA .1,.9,.7310586
130 DATA .7,1.0,.8455348
140 DATA .3,.4,.6681878
150 DATA .2,.05,.5621765
160 FOR K=1 TO 9
170 FOR J=1 TO 2
180 READ W(K,J)
190 NEXT J
```

```
200 NEXT K
210 DATA -1,-1,-1,0,-1,1,0,-1,0,0,0,1,1,-1,1,0,1,1
220 K=INT(9*RND+1):OLDK=K
280 FOR N=1 TO NC
300 GOSUB 3000
320 X=FNCALC(DELTAE,TEMP)
340 RN=RND
350 IF DELTAE < 0 OR RN < X THEN GOSUB 5000 ELSE GOSUB 6000
380 NEXT N
400 END
1000 REM Subroutine CalcEnergy
1010 ENERGY=0
1020 FOR I=1 TO 4
1030 SUM=0
1040 FOR J=1 TO 2
1050 SUM=SUM+X(I,J)*W(K,J)
1060 NEXT J
1070 PRED(I)=1/(1+EXP (-SUM))
1080 ENERGY=ENERGY+(PRED(I)-TRUE(I))↑ 2
1090 NEXT I
1100 RETURN
2000 REM Subroutine ChangeK
2010 RN=RND
2020 IF RN ≤ .5 THEN GOTO 2050
2030 IF K=9 THEN K=1:GOTO 2070
2040 K=K+1:GOTO 2070
2050 IF K=1 THEN K=9:GOTO 2070
2060 K=K-1
2070 RETURN
3000 REM Subroutine Cycle
3010 K=OLDK:GOSUB 1000:OLDENERGY=ENERGY
3020 GOSUB 2000
3030 GOSUB 1000:NEWK=K:NEWENERGY=ENERGY
3040 DELTAE=NEWENERGY-OLDENERGY
3050 RETURN
4000 REM Subroutine to average the move taken
4010 FOR J=1 TO 2
4020 AVG(J)=AVG(J)+W(K,J)
4030 NEXT J
4040 RETURN
5000 REM Subroutine for accepting New State
```

```
5010 K=NEWK
5020 GOSUB 4000
5030 OLDK=NEWK
5040 FLAG=1
5050 ASUM=ASUM+1
5060 RETURN
6000 REM Subroutine for rejecting new state and keeping old state
6010 K=OLDK
6020 GOSUB 4000
6030 FLAG=0
6040 RSUM=RSUM+1
6050 RETURN
```

APPENDIX B

This appendix provides some mathematical background for the equations that appear in simulated annealing and discusses why the Monte Carlo approach is necessary.

To begin, we state a theorem (9) which is important because it tells us about the existence of a minimum for real-valued, continuous functions. It therefore sheds light on the problem of how to find the weights in the neural network model which minimize the sum of squared error function.

We shall re-cast the theorem in a form that reflects the notation used in the example of the main body of the report. The theorem states that the value of the weights, w_1* and w_2*, which result in the minimum value of the function g(w) is given by

$$w_i* = \lim_{\lambda \to \infty} \frac{\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} w_i e^{-\lambda g(w_1, w_2)} \, dw_1 \, dw_2}{\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-\lambda g(w_1, w_2)} \, dw_1 \, dw_2} \tag{3}$$

where

$$g(w_1, w_2) = \sum_{i=1}^{4} \left(\frac{1}{1 + e^{w_1 u_i 1 + w_2 u_i 2}} - y_i \right)^2 \tag{4}$$

$g(w_1, w_2)$ is the sum of squared error function (Energy) which we sought to minimize via simulated annealing.

Equation (3) can be calculated numerically by the following six steps.

Step 1. Recall the definition of the mean or expected value of a discrete variable X from elementary statistics.

$$E(X) = \sum_{i=1}^{N} X_i p(X_i) \tag{5}$$

A simple example is the expected value of the throw of a six-sided die. Using formula (5)

$$\begin{aligned} E(X) = \quad & 1*1/6 + 2*1/6 + 3*1/6 \\ & +4*1/6 + 5*1/6 + 6*1/6 \\ = \quad & 3.5 \end{aligned}$$

Step 2. To estimate the expected value of the throw of the die by Monte Carlo simulation use random numbers generated by a computer over a finite number of trials (100 trials, for example). When the random number is between 0 and 1/6, add 1 to a running total; between 1/6 and 2/6, add 2 to the running total; ...; between 5/6 and 1, add 6 to the running total. At the end of 100 trials, divide the total by 100 to form an estimate of E(X).

Step 3. For a continuous variable, use the integral in the statistical definition of the expected value.

$$E(X) = \int_{-\infty}^{\infty} xP(x)dx \tag{6}$$

The Monte Carlo method is still a valid tool in forming an estimate for $E(X)$. Draw numbers as dictated by the density function $P(x)$. and, as in Step 2, add the numbers over some finite number of trials and divide by the number of trials to obtain an estimate of $E(X)$.

The important point here is that the Monte Carlo method can be viewed as a statistical approach to finding $E(X)$ or, *as a numerical method for finding the value of an integral.* They are both one and the same thing. Newman and Odell (10) state quite clearly, "Thus it is immaterial whether we consider our problem from the point of view of integration or instead consider it as a statistical problem of estimating the expected value at least as far as the numerical result is concerned ."

No one actually uses the Monte Carlo method to evaluate a one-dimensional integral, which are either solved analytically or by using more efficient numerical techniques. The Monte Carlo method does come into its own, however, when a multidimensional integral does not yield to analytical means, and a numerical approach is warranted. The Monte Carlo method is competitive with other numerical techniques in this situation and, in many cases, is the preferred technique.

Step 4. The multiple integral,

$$\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f(x_1, x_2, \ldots, x_n) p(x_1, x_2, \ldots, x_n) dx_1, dx_2, \ldots, dx_n$$

may be untractable analytically, but statistically it is still quite simply a problem of estimating a mean. So, as in Step 3, generate vectors $f(x_1, x_2, \ldots, x_n)$ according to the probability distribution $p(x_1, x_2, \ldots, x_n)$, add them up and divide by the number of trials

to form an estimate of the mean. The result is a numerical estimate of the value of this complicated integral.

Step 5. Armed with this knowledge, equation (3) is not quite as fearsome as it first appeared. If the term,

$$\frac{e^{-\lambda g(w_1,w_2)}}{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-\lambda g(w_1,w_2)} dw_1\, dw_2} = p(w_1, w_2)$$

represents a weight or probability, then it can be tied it to what has been learned in the previous steps. This is where the connection to statistical physics enters. The so-called canonical distribution (11) is defined as

$$P(State\ r) = \frac{e^{-\beta * Energy(State\ r)}}{\int\limits_{S} e^{-\beta * Energy(State\ S)}} \tag{7}$$

where the integral in the denominator represents a multidimensional integral over all possible states S.

With the following notational changes, $g(w_1, w_2)$ = Energy(State r), and $\beta = \lambda = 1/T$ we can invoke the theorem and see that the weight vector that minimizes the Energy function is the expected value of the weight vector distributed according to the canonical distribution as β goes to infinity or as the temperature goes to zero. Step 4 showed that the Monte Carlo approach can be employed to numerically evaluate the complicated integral of equation (3).

Step 6. The following equation brings home the fact that the mean of our weight vector distributed according to such a probability distribution is also the minimum value which we seek.

$$E(w_1, w_2) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (w_1, w_2) p(w_1, w_2) dw_1\, dw_2 \tag{8}$$

It is to be understood that the temperature parameter contained in the probability term goes to zero. The form of this equation is the same as in Step 4 and can be numerically estimated.

Since P(State r) is a probability, the value of the integral in the theorem can be numerically calculated via a Monte Carlo approach according to the reasoning outlined in the above steps . A complication exists in that our probability term includes a multidimensional integral in the denominator. In fact, things are simplified tremendously by taking the ratio of the probability of two "energy" states (two different SSE functions arising from using different weight parameters). By taking the ratio of two states the desired goal of eliminating the integral in the denominator of equation (7) is accomplished. This derivation follows Hammersley and Handscomb (12).

$$\pi(S_j) = \frac{e^{-\beta * Energy(S_j)}}{\int_S e^{-\beta * Energy(S)} dS}$$

$$\pi(S_i) = \frac{e^{-\beta * Energy(S_i)}}{\int_S e^{-\beta * Energy(S)} dS}$$

$$\frac{\pi(S_j)}{\pi(S_i)} = \frac{e^{-[\beta * Energy(S_j)]}}{e^{-[\beta * Energy(S_i)]}}$$

$$= e^{-\beta * [Energy(S_j) - Energy(S_i)]}$$

If $\Delta E = Energy(S_j) - Energy(S_i)$ and $\beta = 1/T$, then

$$\frac{\pi(S_j)}{\pi(S_i)} = e^{-(\Delta E/T)}$$

With these manipulations, we have come to the term which figured so prominently in the discussion of the Metropolis algorithm. Hammersley and Handscomb go on to show how this term is used in a Markov chain approach to generate transition probabilities. This is the same technique which is explained in the text and which produced the data contained in Appendix C.

APPENDIX C

This appendix contains three simulation runs of the Metropolis algorithm in which the temperature parameter was set to T=1. By following the explanation in the text, these tables allow the interested reader a detailed look at the heuristics of the algorithm as it goes about its job of searching for the optimal weights.

Computer simulation data to illustrate the workings of the Metropolis algorithm. Run #1.

Cycle	Old State	New State	ΔE	X	RN	w_1	w_2	Acc
1	7	8	-.309	***	***	1	0	A
2	8	7	0.309	.734	.259	2	-1	A
3	7	6	-.371	***	***	2	0	A
4	6	5	0.184	.832	.894	2	1	R
5	6	7	0.371	.690	.532	3	0	A
6	7	6	-.371	***	***	3	1	A
7	6	5	0.184	.832	.564	3	1	A
8	5	4	0.400	.671	.755	3	1	R
9	5	4	0.400	.671	.402	3	0	A
10	4	5	-.400	***	***	3	0	A
11	5	4	0.400	.671	.152	3	-1	A
12	4	5	-.400	***	***	3	-1	A
13	5	4	0.400	.671	.551	3	-2	A
14	4	5	-.400	***	***	3	-2	A
15	5	6	-.184	***	***	3	-1	A
16	6	5	0.184	.832	.953	3	0	R
17	6	5	0.184	.832	.892	3	1	R
18	6	7	0.371	.690	.823	3	2	R
19	6	7	0.371	.690	.058	4	1	A
20	7	8	-.309	***	***	5	1	A
21	8	7	0.309	.734	.978	6	1	R
22	8	9	-.083	***	***	7	2	A
23	9	1	0.820	.441	.108	6	1	A
24	1	2	-.419	***	***	5	1	A
25	2	3	-.295	***	***	4	2	A
26	3	4	0.499	.607	.294	4	1	A
27	4	5	-.400	***	***	4	1	A
28	5	6	-.184	***	***	4	2	A
29	6	5	0.184	.832	.802	4	2	A
30	5	4	0.400	.671	.075	4	1	A
31	4	3	-.499	***	***	3	2	A
32	3	4	0.499	.607	.306	3	1	A
33	4	3	-.499	***	***	2	2	A
34	3	4	0.499	.607	.499	2	1	A
35	4	3	-.499	***	***	1	2	A
36	3	2	0.295	.744	.828	0	3	R
37	3	2	0.295	.744	.260	-1	3	A
38	2	1	0.419	.658	.608	-2	2	A
39	1	2	-.419	***	***	-3	2	A
40	2	3	-.295	***	***	-4	3	A

Computer simulation data to illustrate the workings of the Metropolis algorithm. Run #2.

Cycle	Old State	New State	ΔE	X	RN	w_1	w_2	Acc
1	7	6	-.371	***	***	0	1	A
2	6	7	0.371	.690	.483	1	0	A
3	7	8	-.309	***	***	2	0	A
4	8	9	-.083	***	***	3	1	A
5	9	8	0.083	.920	.917	4	1	A
6	8	9	-.083	***	***	5	2	A
7	9	8	0.083	.920	.690	6	2	A
8	8	9	-.083	***	***	7	3	A
9	9	8	0.083	.920	.646	8	3	A
10	8	9	-.083	***	***	9	4	A
11	9	1	0.820	.441	.887	10	5	R
12	9	1	0.820	.441	.808	11	6	R
13	9	1	0.820	.441	.154	10	5	A
14	1	9	-.820	***	***	11	6	A
15	9	1	0.820	.441	.521	12	7	R
16	9	8	0.083	.920	.560	13	7	A
17	8	7	0.309	.734	.354	14	6	A
18	7	8	-.309	***	***	15	6	A
19	8	9	-.083	***	***	16	7	A
20	9	1	0.820	.441	.433	15	6	A
21	1	9	-.820	***	***	16	7	A
22	9	8	0.083	.920	.276	17	7	A
23	8	7	0.309	.734	.671	18	6	A
24	7	6	-.371	***	***	18	7	A
25	6	7	0.371	.690	.184	19	6	A
26	7	8	-.309	***	***	20	6	A
27	8	7	0.309	.734	.201	21	5	A
28	7	8	-.309	***	***	22	5	A
29	8	9	-.083	***	***	23	6	A
30	9	8	0.083	.920	.439	24	6	A
31	8	9	-.083	***	***	25	7	A
32	9	8	0.083	.920	.850	26	7	A
33	8	7	0.309	.734	.248	27	6	A
34	7	6	-.371	***	***	27	7	A
35	6	7	0.371	.690	.128	28	6	A
36	7	8	-.309	***	***	29	6	A
37	8	9	-.083	***	***	30	7	A
38	9	1	0.820	.441	.941	31	8	R
39	9	8	0.083	.920	.383	32	8	A
40	8	7	0.309	.734	.637	33	7	A

Computer simulation data to illustrate the workings of the Metropolis algorithm. Run #3.

Cycle	Old State	New State	ΔE	X	RN	w_1	w_2	Acc
1	2	1	0.419	.658	.569	-1	-1	A
2	1	9	-.820	***	***	0	0	A
3	9	8	0.083	.920	.462	1	0	A
4	8	9	-.083	***	***	2	1	A
5	9	1	0.820	.441	.145	1	0	A
6	1	2	-.419	***	***	0	0	A
7	2	3	-.295	***	***	-1	1	A
8	3	2	0.295	.744	.968	-2	2	R
9	3	2	0.295	.744	.839	-3	3	R
10	3	2	0.295	.744	.395	-4	3	A
11	2	1	0.419	.658	.389	-5	2	A
12	1	2	-.419	***	***	-6	2	A
13	2	1	0.419	.658	.836	-7	2	R
14	2	3	-.295	***	***	-8	3	A
15	3	4	0.499	.607	.767	-9	4	R
16	3	4	0.499	.607	.988	-10	5	R
17	3	2	0.295	.744	.898	-11	6	R
18	3	4	0.499	.607	.885	-12	7	R
19	3	4	0.499	.607	.464	-12	6	A
20	4	3	-.499	***	***	-13	7	A
21	3	4	0.499	.607	.865	-14	8	R
22	3	2	0.295	.744	.741	-15	8	A
23	2	3	-.295	***	***	-16	9	A
24	3	2	0.295	.744	.571	-17	9	A
25	2	1	0.419	.658	.155	-18	8	A
26	1	9	-.820	***	***	-17	9	A
27	9	1	0.820	.441	.912	-16	10	R
28	9	1	0.820	.441	.550	-15	11	R
29	9	8	0.083	.920	.266	-14	11	A
30	8	7	0.309	.734	.005	-13	10	A
31	7	6	-.371	***	***	-13	11	A
32	6	5	0.184	.832	.347	-13	11	A
33	5	6	-.184	***	***	-13	12	A
34	6	5	0.184	.832	.555	-13	12	A
35	5	6	-.184	***	***	-13	13	A
36	6	5	0.184	.832	.423	-13	13	A
37	5	6	-.184	***	***	-13	14	A
38	6	5	0.184	.832	.317	-13	14	A
39	5	4	0.400	.671	.563	-13	13	A
40	4	5	-.400	***	***	-13	13	A