Graduate School Form 9
(Revised 7/94)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis Acceptance

This is to certify that the thesis prepared

By Aaron Fletcher Stanton

Entitled PIVOT METHODS FOR GLOBAL OPTIMIZATION

Complies with University regulations and meets the standards of the Graduate School for originality and quality

For the degree of Doctor of Philosophy

Signed by the final examining committee:

[illegible], chair

[illegible]

[illegible]

[illegible]

Approved by: [illegible] 4/23/99
Department Head Date

This thesis ☐ is [✓ is not] to be regarded as confidential. [illegible]
Major Professor

Format Approved by:

______________________ or Rita F. Biederstedt
Chair, Final Examining Committee Thesis Format Adviser

PIVOT METHODS FOR GLOBAL OPTIMIZATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Aaron Fletcher Stanton

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 1999

Dedicated

to

My Wife

Christina Lynn Fletcher Stanton

for

Being selfish enough to

Help me succeed

ACKNOWLEDGMENTS

The author would like to thank several people for their contributions to this work over the course of the past several years. Firstly, I thank Dr. Richard E. Bleil for helping me get all this started, and brainstorming out the first incarnation of this method. I thank Dr. Pablo Serra for his help with the generalized simulated annealing, the nearest neighbor concept, and the q distribution. I would like to thank Pablo Nigra next for actually using this method actively in his research. Stephen D. Belair deserves special thanks for putting up with me during the writing of this thesis. I've dedicated this to my wife, but she deserves many thanks for her help in putting this together as well as putting up with me.

Most of all I would like to thank my advisor, Professor Sabre Kais, for insisting I do this right and expecting from me what I am capable of.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Stanton, Aaron Fletcher Ph.D., Purdue University, May, 1999. Pivot Methods for Global Optimization. Major Professor: Sabre Kais.

A new algorithm is presented for the location of the global minimum of a multiple minima problem. It begins with a series of randomly placed probes in phase space, and then uses an iterative redistribution of the worst probes into better regions of phase space until a chosen convergence criterion is fulfilled. The method quickly converges, does not require derivatives, and is resistant to becoming trapped in local minima. Comparison of this algorithm with others using a standard test suite demonstrates that the number of function calls has been decreased conservatively by a factor of about three with the same degrees of accuracy.

Two major variations of the method are presented, differing primarily in the method of choosing the probes that act as the basis for the new probes. The first variation, termed the lowest energy pivot method, ranks all probes by their energy and keeps the best probes. The probes being discarded select from those being kept as the basis for the new cycle. In the second variation, the nearest neighbor pivot method, all probes are paired with their nearest neighbor. The member of each pair with the higher energy is relocated in the vicinity of its neighbor.

Both methods are tested against a standard test suite of functions to determine their relative efficiency, and the nearest neighbor pivot method is found to be the more efficient. A series of Lennard-Jones clusters is optimized with the nearest neighbor method, and a scaling law is found for cpu time versus the number of particles in the system. The two methods are then compared more explicitly, and finally a study in the use of the pivot method for solving the Schroedinger equation is presented. The nearest neighbor method is found to be able to solve the ground state of the quantum harmonic oscillator from a pure random initialization of the wavefunction.

A New Approach to Global Minimization

A. F. Stanton, R. E. Bleil, and S. Kais, J. Comp. Chem. 18, 594 (1997).

Abstract

A new algorithm is presented for the location of the global minimum of a multiple minima problem. It begins with a series of randomly placed probes in phase space, and then uses an iterative Gaussian redistribution of the worst probes into better regions of phase space until all probes converge to a single point. The method quickly converges, does not require derivatives, and is resistant to becoming trapped in local minima. Comparison of this algorithm with others using a standard test suite demonstrates that the number of function calls has been decreased conservatively by a factor of about three with the same degrees of accuracy. A sample problem of a system of seven Lennard-Jones particles is presented as a concrete example.[1, 2, 3]

CHAPTER I. INTRODUCTION

Rationale

The recent explosion of publications relating algorithms for the location of the global minimum in a multiple minima function demonstrates both the importance of this problem in practically all fields of science and that further development toward such a method is still needed. Such an algorithm would find applications in fields such as drug design, molecular modeling, quantum mechanical calculations, and mathematical biological calculations.[4, 5, 6, 7, 8, 9, 10, 11, 12, 13] Unfortunately, the problem of locating the global minimum in nearly all cases is hindered by the presence of local minima.

All minimization algorithms face the difficulty of locating the global minimum in the presence of a series of local minima. Once a local minimum is converged on, there is no sure method to determine if another deeper minimum might exist somewhere else without leaving the comfort of the minimum located. The problem becomes more harrowing as one increases the dimensionality of the problem. Fortunately, several newer methods[14] for function minimization are showing vast improvements in the ability to locate global versus local minima.

Deterministic Methods

Some of the older minimization algorithms are local deterministic methods, primarily based on extensions of Newton's methods.[15, 16, 17] These work by following the terrain of the function, either utilizing the numerical steepest downhill approach, or some derivative of the function that indicates the best path to follow. These methods were notorious for finding themselves locked in local minima.

Stochastic Methods

More recently, deterministic approaches have been usurped by other methods resistant to becoming trapped in local minima. These new methods have been compared in several review articles,[18] and show improving accuracy and efficiency. Several new methods worth noting are the tabu search,[18] tunneling,[19, 20] renormalization group,[21] J-walking,[22] and convexity,[23] however two powerful and popular methods deserve some in depth attention, especially considering the debt the method presented here owes to each of them.

Simulated Annealing

Simulated annealing[24, 25, 26] is based on a statistical mechanics approach, wherein a probe is moved and allowed to overcome small maxima in the beginning, thereby having the ability to move out of the local minima. It begins with only one representation of the system being investigated, and as "time" elapses, it varies that single representation according to some scheme.

In the original version of simulated annealing, the single representation, or "probe" as will be used in this work, is varied according to a Gaussian distribution. If the energy of the new probe is lower than the old, it is accepted. If it is higher than the old, it is accepted with a probability based on a Boltzmann distribution from the difference in energies and the "temperature." If the temperature does not change with time, this is simply statistical mechanics, but if one allows the system to "cool" logarithmically over time, one can reach the minimum energy state of the system.

In "fast" simulated annealing, Szu and Hartley[27] allowed nonlocal jumps to occur, enabling the system to get out of local minima wells more quickly than otherwise possible. They accomplish this by replacing the Gaussian distribution with a Cauchy-Lorentz distribution, which has some nonlocal character to it. In general, this variation does substantially improve the performance of the simulated annealing algorithm.

Most recently, Tsallis and Stariolo[28] have created generalized simulated annealing. They have developed a distribution function with a free parameter, designed such that for certain values of this parameter one recovers the more familiar Gaussian or Cauchy-Lorentz distributions. Additionally, they have generalized the logarithmic cooling scheme to be more appropriate to the new possibilities of different distribution functions. They have found that by setting their adjustable parameter, q, to 2.5 - 2.7, they receive drastic improvements in the rate of convergence for simulated annealing. It can be seen easily that their distribution function has a great deal of nonlocal character for values of q in that region, and that seems to be key in escaping local wells.

Genetic Algorithm

The genetic algorithm[29, 30, 31, 32] starts with several random guesses as to where the minimum might be, but then changes some of the variables of the worst guesses until they converge. There are many variations on this theme available, but it is worth noting that, unlike simulated annealing, the genetic algorithm uses many guesses at once. This parallel processing, along with allowing those guesses to interact with each other, makes the genetic algorithm very powerful.

Some of the more common ways to vary these "probes" include simply keeping the very best, generating completely random ones, varying good probes slightly (again, usually with a Gaussian distribution function), and mixing two probes in one of several ways. Some of the ways of "mating" two are a "one point crossover," in which a point is chosen at random inside the probe and all information past that point is exchanged between the two, and a "two point crossover," where two points are chosen inside the probes and everything between those two points is exchanged. From this it is clear what an "N point crossover" might be.

As time passes in the system, the probes are ranked, and the probes that rank poorly get thrown out. The system gradually explores the potential in question, and ideally converges upon the best possible result for the space in question. Conceptually a simple algorithm, it is fairly powerful and easy to implement, but its lack of sophistication makes it miss global minima occasionally. Our method incorporates ideas from both this method and generalized simulated annealing.

The Pivot Methods

The algorithm being introduced shows a great improvement in efficiency of the location of the global minimum. It begins with a series of random guesses as to where the minimum might be, then redistributes the worst guesses into better regions of phase space until a desired convergence criterion is reached. This new method locates the global minimum in a multiple minima problem, without the necessity of including computationally expensive estimates for the derivative of the function, is generally applicable, avoids entrapment in local minima, and is conceptually simple and easy to program.

Lowest Energy Pivot Method

In Chapter II, the lowest energy pivot method is presented in detail. This is the first incarnation of the pivot methods, and the original data for comparing this to several other optimization methods is presented. It is shown that this method improves on previously existing methods by at least a factor of five.

This most clearly resembles the genetic algorithm in how it ranks probes and discards bad ones, but its streamlined approach is likely what allows it to be more efficient than the genetic algorithm.

Nearest Neighbor Pivot Method

Chapter III introduces the nearest neighbor pivot method, which uses a new method for selecting pivots. Rather than merely keep all good probes regardless of location, it serves to keep the collection of probes more free of duplicates by searching for probes near each other and moving the worse of the two. Moderate improvements are gained by doing this when using our test suite of functions, and still more improvements appear when we no longer vary probes by the Gaussian distribution, but instead vary them by a more nonlocal one, the q distribution. The original data is presented here for comparison.

Comparison of Pivot Methods

In Chapter IV, I go over once again the basics of both versions of the methods and provide a systematic comparison of both the lowest energy method and the nearest neighbor method. The most recent data is presented, and it is found that the nearest neighbor method solves the test suite of functions more efficiently than does the lowest energy method. With this in mind, clusters of six to twenty Lennard-Jones particles are optimized, and a general scaling law is derived. It is found that the nearest neighbor method scales substantially better than the genetic algorithm for systems of this type.

Solution of Schroedinger's Equation

It is detailed in Chapter V how to go about implementing the needed computations for solving the Schroedinger equation via an optimization routine. An efficient method for speeding up the calculations is presented, and results for the pivot methods are tabulated. It is found that a nonlocal variation of the wavefunction is necessary to solve the quantum harmonic oscillator, and the lowest energy pivot method proves to be more efficient than the nearest neighbor pivot method for this particular problem.

Conclusions and Recommendations

The conclusions, Chapter VI, summarize the results in a non-quantitative fashion, presenting to the reader the knowledge of what has been accomplished without undue repetition of facts that are detailed elsewhere. Also, suggestions are made on future uses of the pivot methods, along with ideas on how to implement them for more complicated quantum mechanical systems.

References and Appendices

Lastly, a list of references used in this work are given along with a set of appendices containing relevant information. Included in the appendices are the equations used in the suite of test functions used to compare the pivot methods with other methods and with each other, a complete implementation of both methods to solve one of the test functions, and the necessary files to implement the solution of the quantum harmonic oscillator. Finally, my Vita is included at the end of this work.

CHAPTER II. LOWEST ENERGY PIVOT METHOD

Methodology

Phase space is defined to be the set of points bounded by the minimum and maximum values of all variables in the system. The dimensionality of the problem is the number of variables. The task is to find the global minimum within the defined phase space, min $f(s)\colon s \in S$ where $f(s)$ is the function to be minimized and s is a point in phase space S. This begins with the random placement of probes within phase space, which are used as a means of sampling regions of the entire space. For convenience, a flowchart, Figure 1, has been included to illustrate this method step by step.

The probes are themselves complete sets of all variables in the phase space, and thus each probe has the specific value of the function to be minimized at that location. Initially, these probes are randomly placed within phase space, and the function value at the location of each probe is computed. The number of probes chosen is arbitrary, but with enough probes to get a good statistical distribution. With too many probes, the algorithm runs too slowly, but without enough probes, one is likely to find a local rather than global minimum. A balance between these two extrema is required.

with more probes necessary for more sharply varying functions. Once the probes are initially placed, one begins sampling even more of the phase space by the relocation of the worst of these probes to be nearer to the best.

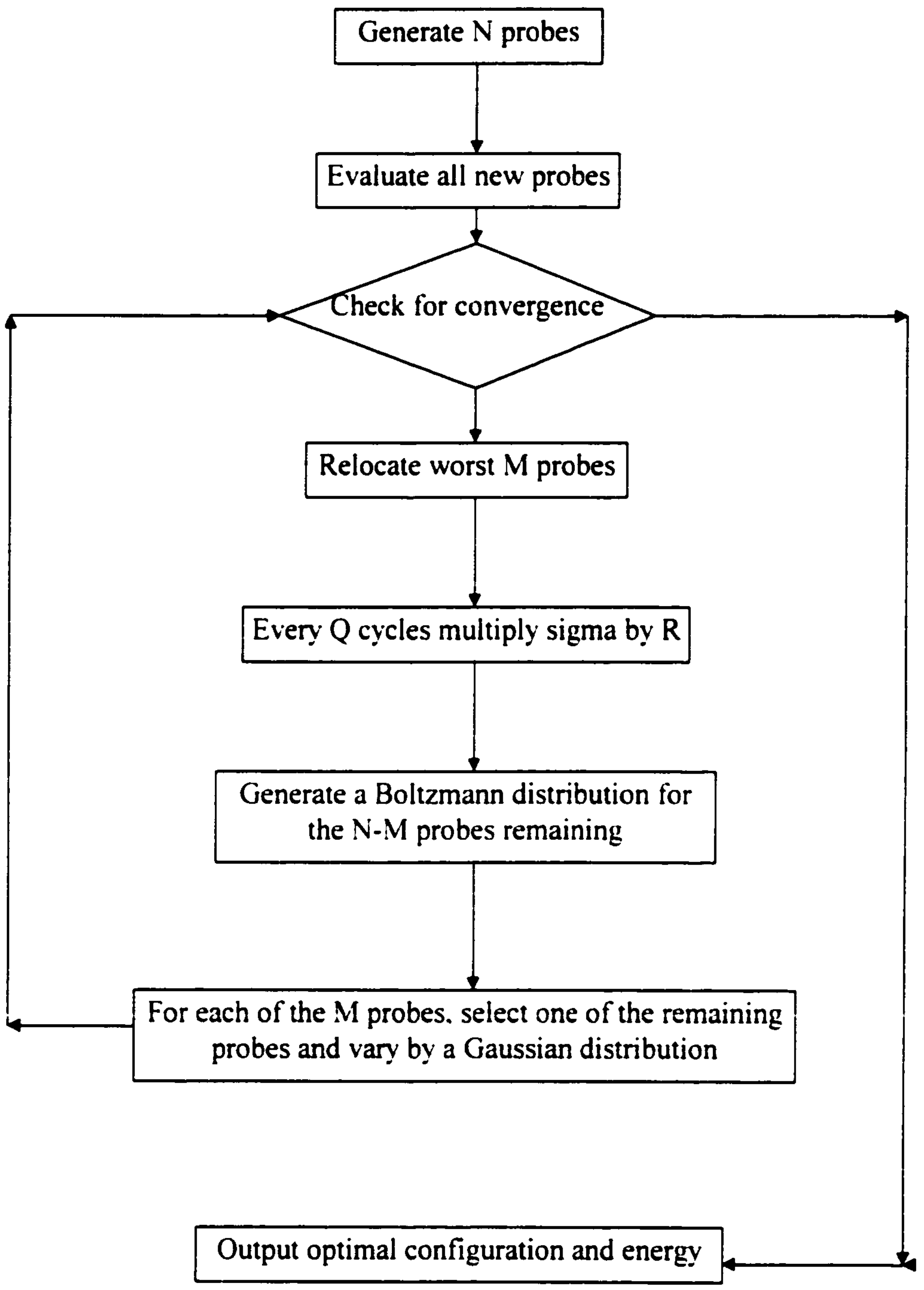
Generate N probes
Evaluate all new probes
Check for convergence
Relocate worst M probes
Every Q cycles multiply sigma by R
Generate a Boltzmann distribution for the N-M probes remaining
For each of the M probes, select one of the remaining probes and vary by a Gaussian distribution
Output optimal configuration and energy

Figure 1 Flowchart for the Lowest Energy Pivot Method

Before one can choose where to locate the worst probes, it first becomes necessary to choose how many probes should be relocated. It is then necessary to rank the existing probes according to their values of the function to be minimized. It must be noted that this method borrows conceptually from the genetic algorithm in this part, but that it substantially differs in how new probes are chosen, as will be detailed. The number of probes to be relocated is also arbitrary. A satisfactory amount is roughly one third of the total number of probes. At each iteration of the algorithm, one then replaces a certain number of the probes, choosing to replace the worst guesses at all times, keeping the best guesses unchanged. The larger the percentage of probes to be relocated, the more calculations that are required at each iteration, but fewer iterations should be required. This saves computational time on reranking the probes at each step. Furthermore, it is necessary to keep enough of the best probes to avoid the possibility that one is deleting the probe closest to the global minimum because a second probe happened to be sitting lower in a local minima well. The probes to be relocated are placed near one of the better probes, called a pivot probe, in a random fashion.

Recognizing the problem that one may be too close to a local minimum in the first iteration, one chooses the pivot probe in a random fashion with a probability based on the value of each probe. To do this, one assigns a probability to each of the probes not to be relocated in a Boltzmann distribution; that is, for probe *i*, its probability is

$$P_i = \frac{\exp(-f(i))}{P}.$$

where

$$P = \sum_{i=1}^{n-m} \exp(-f(i)) \ .$$

n is the total number of probes, m is the number of probes relocated at each iteration, and $f(i)$ is the function value of probe i. Each probe to be relocated is placed randomly near one of these remaining probes, with the best probe getting most of the relocated probes near it, but it is important to note that all remaining probes have a finite chance of being used as a pivot probe. Each probe to be relocated chooses its own pivot probe based on these probabilities. Typically, in a single iteration, there is more than one pivot probe chosen, depending on how sharply varying the function is, how many choices of pivot probes there are, and how many probes will be relocated. In this way, one is always moving probes to better regions of phase space, but hopefully slowly enough so as not to overlook a global minimum. The distance of the location of the relocated probe from the pivot probe is a decision that must be carefully weighed.

Much like simulated annealing, it is best to begin relocating probes in a large distribution from the chosen pivot of each. The relocated probes are placed in a Gaussian distribution centered on the pivot probes with a standard deviation input as an arbitrarily adjustable parameter, σ. This standard deviation defines a space around each pivot probe, where the relocated probes are found inside this region half of the time, and outside this region the rest of the time. Initially, this size should be chosen to be large, much like simulated annealing, so that a fairly large portion of phase space is covered by the pivot probes. This process is repeated for several iterations.

The number of iterations for any given standard deviation value can be varied. The more steps taken, the better phase space is sampled, but the longer it will take for the algorithm to converge. Once a predetermined number of iterations have elapsed using this standard deviation, this value is then decreased at some given rate. This rate, R, decreases the standard deviation σ to some new value, σ'. The rate used for the functions tested was 0.466. At each new standard deviation, the algorithm repeats as many times as with the initial standard deviation, until σ is decreased again. In effect, this means that the probes will in time converge on some given small point, and, with a good sampling of phase space, this point should be the global minimum. The choice of stopping criteria therefore becomes important.

There are several possible choices of stopping criteria. One is to choose a small given range of values for the function, or some given standard deviation of the average values of the parameters of phase space. By choosing a stopping criterion based on the standard deviation of the phase space variables, it is to be hoped that one avoids finding several local minima wells with equal energies. Unfortunately, it is also possible for the system to become trapped in several minima and, if the cooling rate is too high, to simply never leave those wells, effectively resulting in an infinite loop. It is always a good idea to set a maximum number of iterations to avoid such a loop. In the end, however, it is common practice just to look at the value of the best probe, with the hope that the probe is closest to the true global minimum. If a value for the global minimum is known, one can stop once the best probe within a given error of this value. Of course, the implementation of this algorithm depends on what is known of the function initially.

If the behavior of the function is known to be smoothly varying, it would be best to choose few pivot probes and a large number of probes to be relocated at each iteration. If it is sharply varying, it is best to use a larger number of pivot probes with a smaller number to be relocated at each iteration. If the behavior of the function is not known at all, it is best to repeat the algorithm at least twice, increasing the number of probes. If a different minimum is found, then it is a sharply varying function and yet another run would be warranted. For the test cases, the values of the global minima were known and I chose to use stopping criteria appropriate to this knowledge in order to measure the efficiency of the method in terms of function calls.

Testing

In the test cases, several well established functions were used for comparison with established methods of optimization. These functions included Goldstein-Price (GP), Branin (BR), Hartman three- and six-dimensional variants (H3 and H6), and Shubert (SH).[33] The full details of the functions tested can be found in Appendix A. Notice that all of these functions except for the H3 and the H6 are two dimensional and that all of these have similar results. The stopping criteria selected was for the probe with the best value to be at worst within 3% of the known global minimum, or for it to stop if the number of iterations exceeded a certain amount. This latter criterion was set high enough that if it were the cause for stopping the algorithm it could be safely stated that the probes were trapped in a local minima. For the H6 function, repeating boundary conditions were introduced, which essentially cause the probes to "wrap around" the area being searched rather than be pushed to the edges of phase space. It was determined that this enabled more of phase space to be searched, and significantly improved the results. Note that this technique was not used on the other functions tested, but the results obtained from H6 strongly indicate that this would not detract in any way from their efficiency and should in fact improve the algorithm overall.

As a rule of thumb, a number of probes equal to 5-15 times the dimensionality of the function was used, and a third of them were moved at each iteration. Clearly, with fewer probes the method runs much more quickly, but one also has a greater chance of being caught in a local minimum. The selected cooling rate came from some preliminary results in attempting to optimize the search parameters and was used throughout for consistency. The initial phase was 10 iterations, as was the number of iterations between each reduction of the standard deviation as described above. In my specific example in the next section, I demonstrate that it can be useful to vary the number of iterations between cooling. The step size taken is crucial. It is very important that the steps taken are not too small initially, as that will cause probes to never leave the region of a local minima. It is advisable to begin with large steps and cool to much smaller ones. In general, the initial step size should be on the same order of magnitude as the phase space being searched. This especially holds true if one is using repeat boundary conditions, as this insures that all of phase space will be adequately searched. As a specific example, for H6, which has a range of 0-1 for each variable, an initial standard deviation of 1 was used, which implies that half of the time the probe was relocated in such a way that repeating boundary conditions was invoked. This in turn shows that the entire space was searched.

Table 1 Number of Function Evaluations for LEP[a]

Method[b]	GP	BR	H3	H6	SH
PRS	5125	4850	5280	18,090	6700
MS	4400	1600	2500	6000	-
SA1	5439	2700	3416	3975	241,215
SA2	563	505	1459	4648	780
TS	486	492	508	2845	727
LEP	112	144	122	1536	281

a The majority of this data is taken directly from Reference 18. The results of the lowest energy pivot method have been added for comparison.

b The methods listed here are pure random search (PRS), multistart (MS), simulated annealing type 1 and 2 (SA1 and SA2, respectively), and tabu search (TS). The references for these methods can be found in Reference 18.

Lennard-Jones Clusters

The previous sampling of the performance of this new algorithm using a standard test suite of mathematical functions suffices to demonstrate its usefulness for functions of low dimensionality. In general, however, one is interested in a function of more than only six dimensions. Therefore, to illustrate this method's usefulness to more realistic problems, I present a simple Lennard-Jones cluster of seven particles. While this is clearly very far from anything new, its very simplicity can be used to implement this method relatively easily.

Specifically, one must establish an array containing the coordinates of seven points in three-dimensional space, and a function that yields the standard Lennard-Jones energy when that array is passed to it.[11] One must also establish an array capable of holding an arbitrary number of these sets of seven points, represented by an array of twenty-one variables. Each set of twenty-one variables is referred to as a "probe" of the space in question. In my particular implementation I chose to use 150 of these "probes" to explore the potential.

One must then place each of these "probes" in the space to be searched. I chose to confine my search to a region of three-dimensional space defined by a cube extending from -2 to 2 in each direction, and any "probe" that moved outside this region was placed back into it by using repeating boundary conditions. Each "probe" is placed randomly, which is to say that for each of my 150 "probes" I randomly place seven points inside the cube from -2 to 2.

Next, each "probe" has an energy assigned to it by using the Lennard-Jones function previously mentioned, and the whole set of 150 is sorted from best to worst (lowest to highest).

At this point, I chose to discard the worst 50 sets of points and choose 50 new sets of points based on the remaining 100. Each of these "probes" is assigned a probability of being chosen according to a Boltzmann distribution, as explicitly described in the previous section. Each of the 50 new "probes" then separately chooses a "probe" upon which to be based. Note that it is entirely possible for two or more of the new sets of points to be based on the same "probe". I then vary the selected set of points by a Gaussian distribution around it, and the new values of coordinates for the set of points become the new "probe". I initially chose for the width of this Gaussian distribution to be 2, or one half the width of the cube being searched. This serves to establish that all of the space is being searched properly.

At this point, all 50 of the new "probes" must have their energy evaluated, and then the entire set of 150 is resorted. Again, the worst 50 are selected out, rechosen, reevaluated, and reranked. This entire process continues 100 times in my particular example.

After this occurs 100 times, I multiply the width of the Gaussian distribution by a factor of 0.9, which serves to narrow the region of space being searched. I wish to confine my search to regions that I believe likely to yield the global minimum. At this point I also determined the statistical deviation of the energies of all 150 probes. If the deviation is below 10^{-7}. I decide that the system has converged sufficiently to assume that I am finished. If the deviation is insufficiently low. I repeat another 100 iterations of selecting out and choosing new "probes". and at the end of this time the Gaussian is once again contracted and statistics are once again performed.

Clearly. I could have used the known global minimum value of -16.505[11] for this particular problem as the stopping criteria. but I wish to demonstrate that this method is fully capable of finding an unknown global minimum quite as effectively as a known one. Having run this entire routine 100 times. this method found the global minimum of -16.505 a total of 75 times. In addition. it took this method an average of 390.383 calls to the potential for each of the times that it successfully found that minimum. Admittedly. this does not represent a 90% convergence. but it suffices to prove that this method will find a global minimum without previously knowing exactly what that number is. One can improve the convergence rate if one is willing to sacrifice speed in so doing simply by increasing the number of probes or by increasing the number of cycles between cooling.

This simple demonstration of this method is quite easily done in any programming language capable of performing simple mathematical operations. One run of this particular implementation takes approximately 7-10 minutes on a 486DX/33 with 8 megabytes of RAM. and takes substantially less time on an IBM RS/6000. This should demonstrate adequately that this method is quite fast.

Results and Discussion

The results given in Table 1 reflect this algorithm converging a minimum of 90% of the time to within. at most. 3% if the known global minimum of the function in question. In many cases the error was significantly less than 3%. and the convergence exceeds 90% on a few. This is identical with the established reliability of the tabu search method.[18] The number of function calls listed is the average value of the times that the algorithm converged. This indicates a two- to four-fold improvement over the best method found for optimization. the tabu search method. for all functions tested. It must be noted. however. that Kan and Timmer[34] have released results that are particularly noteworthy for a similar test suite of functions. but a lack of clearly staged convergence criteria prevents one from using their results as a basis of comparison. Had such criteria been established. their results would be included in the table to indicate the relative merit of the various methods presented. I also have presented a concrete example that should be very easy for anyone familiar with programming to implement. which enables users to see directly the value of this method.

This method of searching for a global minimum of an arbitrary function has proven to be efficient. seldom becomes trapped in a local minimum. does not require the computationally expensive use of derivatives. and is easy to implement.

CHAPTER III. NEAREST NEIGHBOR PIVOT METHOD

Methodology

The task is to locate a global minimum of a given function $f(x)$ within a predetermined phase space Ω defined by the maximum and minimum values of all parameters $x(i)$ of the function. A globally minimum value for this function is assumed to exist within the defined phase space at the corresponding phase space point $f_0 = f(x_0) = \min_{\vec{x} \in \Omega} f(x)$, where x_0 is a phase-space point with the global minimum value f_0 for the function $f(x)$.

The pivot method begins with a given number of probes placed initially within the phase space Ω. If something is known of the problem, it is most convenient to use this information in the initial determination of probe location, which will be demonstrated in the Lennard-Jones example to follow. Each probe is simply a set of values for the parameters $x(i)$ of the problem within the boundaries of the phase space and therefore with a given functional value equal to the value of the function $f(x)$ at the

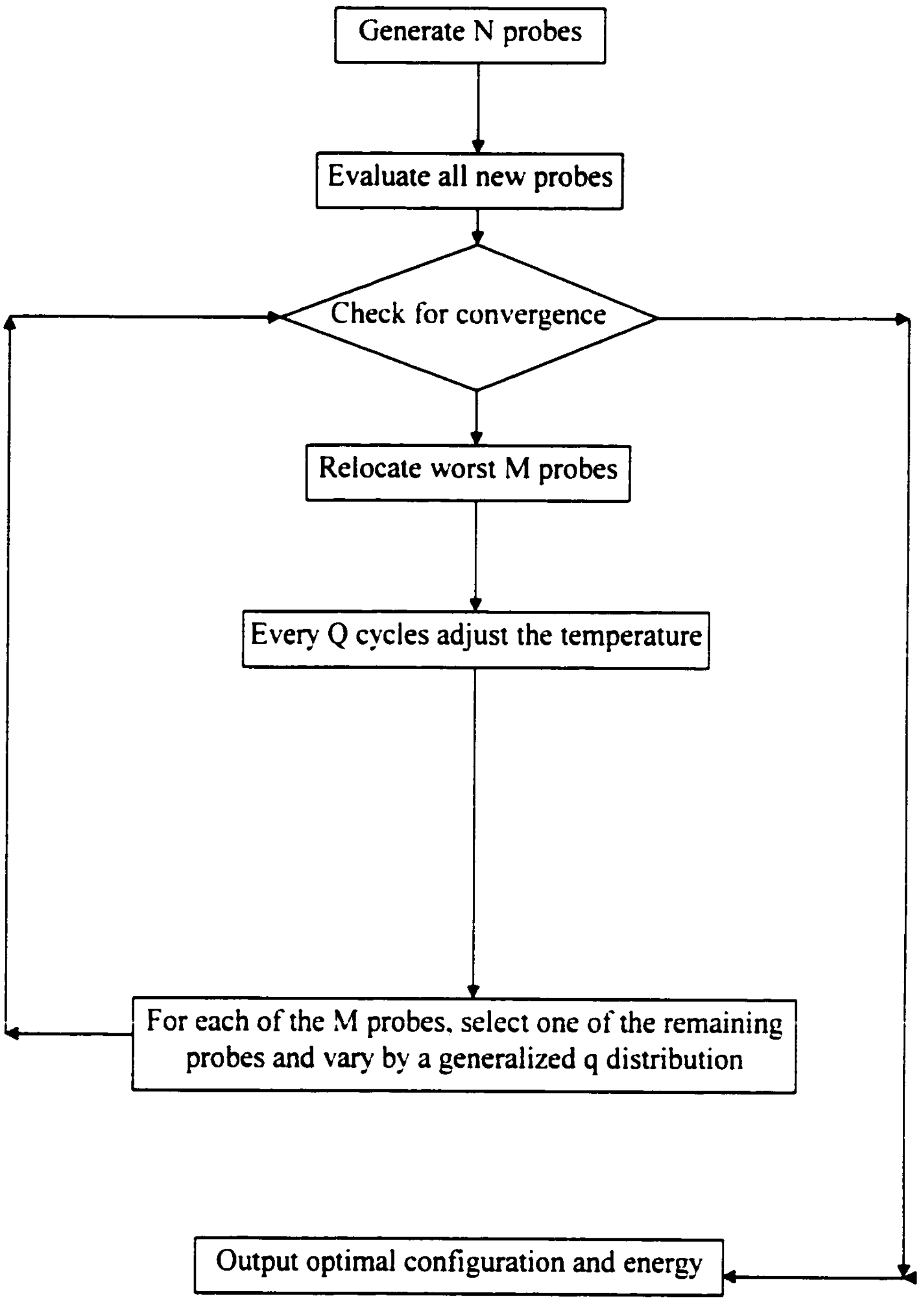
Generate N probes
Evaluate all new probes
Check for convergence
Relocate worst M probes
Every Q cycles adjust the temperature
For each of the M probes, select one of the remaining probes and vary by a generalized q distribution
Output optimal configuration and energy

Figure 2 Flowchart for the Nearest Neighbor Pivot Method

probe point. In the current method, we start with $N = 2m$ initial probes of which m probes will act as the pivot probes, and the remaining m probes will be relocated.

A selection of the m pivot probes begins with a search at each probe for its nearest neighbor, based on the distance of the probes. Once we have paired the probes, the probe with the lower value for the function $f(x)$ is defined as the pivot probe, the other probe being the probe that will be relocated. There are several methods possible to do this pairing, but in my particular case I began with finding the nearest neighbor for probe 1 and removed those two probes from further consideration. This was repeated until all points had been paired.

For each pivot probe with parameter values $x_{B,i}$, we explore phase space by placing the probe to be relocated near the pivot probe by changing its parameters $x_{R,i}$ as $x_{R,i} = x_{B,i} + \Delta x_i$ where Δx_i is a randomly generated vector according to an exploring distribution $g_e(\Delta x)$. We have tested several distributions, such as the standard Gaussian distribution used in the pivot method, Chapter II, and in simulated annealing[24] as well as the Cauchy-Lorentz distribution proposed by Szu and Hartley for fast simulated annealing method.[27] Finally, we have tested a generalized q distribution based on the Tsallis entropy[35] and recently used with good results in the generalized simulated annealing method.[36, 28, 37, 38] We chose to use the general q distribution for the placement of the probes near the pivot probes. This distribution is defined to be

$$g_q(x) = \sqrt{\frac{q-1}{\pi}} \frac{\Gamma\left(\frac{1}{q-1}\right)}{\Gamma\left(\frac{1}{q-1} - \frac{1}{2}\right)} \frac{[\beta(t)]^{\frac{1}{(3-q)}}}{\left(1 + (q-1)\left\{[\beta(t)]^{\frac{1}{(3-q)}} x\right\}^2\right)^{\frac{1}{(q-1)}}}$$

where β is defined to be $\frac{1}{T}$ and T is an artificial temperature given by[28]

$$T(t) = \frac{2^{q-1} - 1}{(1+t)^{q-1} - 1} T(1), t = 1,2,3,\ldots.$$

where $T(1)$ is the initial temperature and t is the discrete time corresponding to the computer iterations.[39]

The most relevant fact about this distribution is the introduction of a new parameter q. Special cases that should be noted are the limit $q \to 1$, where the general q distribution approaches the Gaussian distribution, and $q = 2$, where the q distribution is equal to the Cauchy-Lorentz distribution. The second moment of this distribution diverges for $q \geq \frac{5}{3}$ and the distribution becomes non-normalizable for $q \geq 3$. As in general simulated annealing,[28] we have found $q = 2.5$ to be a good value for my global optimization method. A detailed comparison with other values of q and other distributions will be given in Chapter IV.

Testing

In my standard test suite of functions we include several well-known functions for comparison with established methods of optimization. These are the Goldstein-Price (GP), Branin (BR), Hartman three- and six-dimensional variants (H3 and H6), and Shubert (SH) functions.[33] The full details of these functions can be found in Appendix A. Note that all of the two-dimensional functions have similar results. The stopping criterion chosen was for the best probe to have a value no farther from the known global minimum than 3% or to stop if the number of iterations exceeded a certain value. This stopping criterion is the same as that in Ref. 18 and therefore allows an objective standard for comparison. The latter criterion was set sufficiently high to establish that the system was trapped in a local minimum if this criterion was triggered. A different number of probes was found to be optimal for each of the varying test functions. We have included in Table 2 the results to illustrate the improvements over previous methods. This represents an average of function calls over the successful runs. One thousand runs were done for each function. The improvement for two-dimensional methods is a factor of 3.2-7.2, whereas the H3 and H6 functions show improvement of 9.8 and 12, respectively. Figure 3 illustrates the efficiency of using a value of $q = 2.5$ graphically by plotting the number of times the Branin function was called upon in the minimization against values of q ranging from 1 to 3. Note that this represents a successful convergence rate in excess of 95%. A similar behavior was observed for the remainder of the functions in the test suite.

Table 2 Number of Function Evaluations for NNP[c]

Method[d]	GP	BR	H3	H6	SH
PRS	5125	4850	5280	18,090	6700
MS	4400	1600	2500	6000	-
SA1	5439	2700	3416	3975	241,215
SA2	563	505	1459	4648	780
TS	486	492	508	2845	727
NNP	153	68	52	237	159

[c] The majority of this data is taken directly from Reference 18. The results of the lowest energy pivot method have been added for comparison.

[d] The methods are pure random search (PRS), multistart (MS), simulated annealing type 1 and 2 (SA1 and SA2, respectively), and tabu search (TS). The references for these methods can be found in Reference 18.

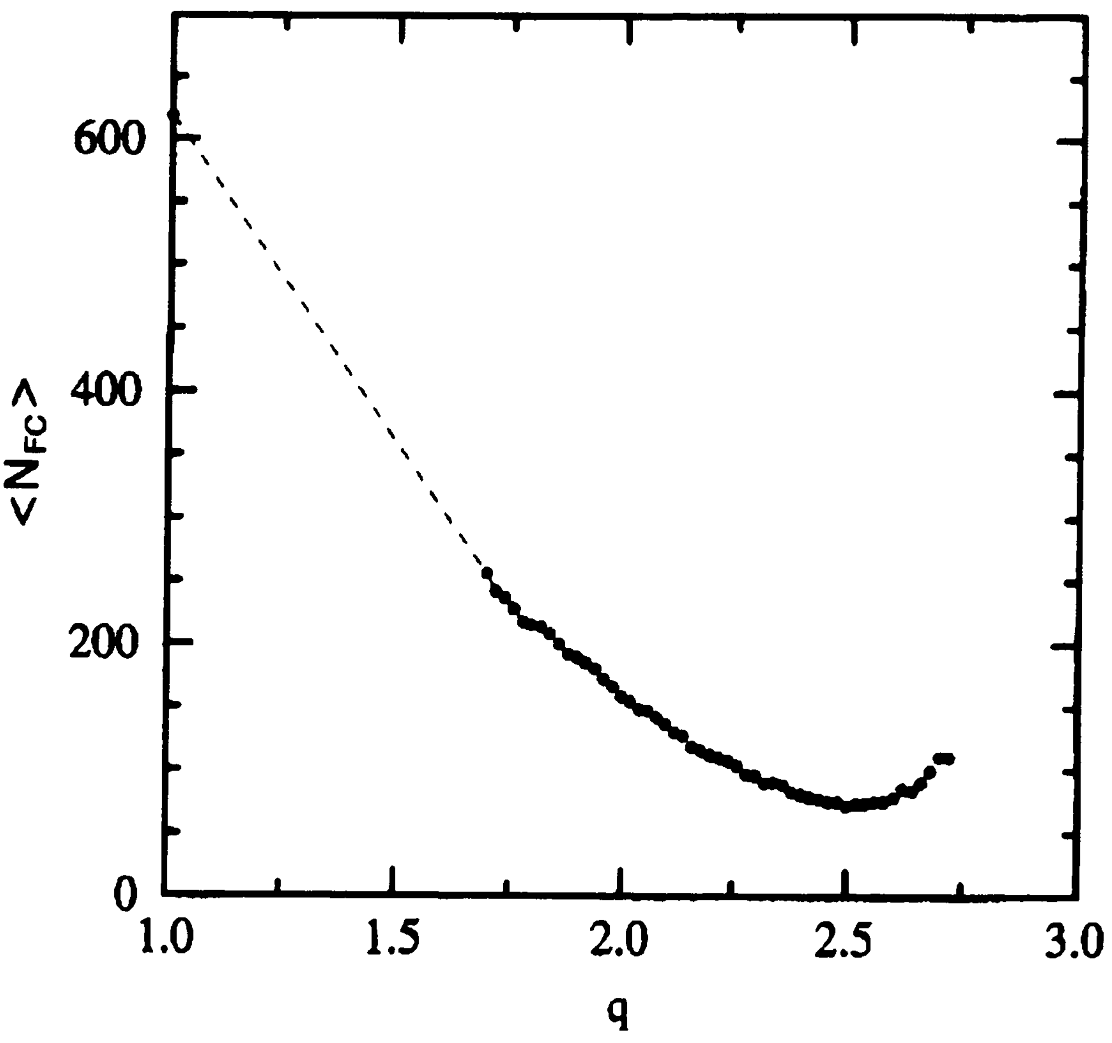

Figure 3 Average Number of Function Calls versus q for the Branin Function

Lennard-Jones Clusters

We now apply the method to Lennard-Jones cluster of size 6-20 (phase-space dimension 12-54). Lennard-Jones clusters are excellent for testing the efficiency of global optimization algorithms. Regular Lennard-Jones clusters have well-established minima and minimum-energy structures for very large clusters.[40, 41] However, the number of local minima apparently grows as $\exp(N^2)$.[42] Wille and Vennik have shown that to find the global minimum in Lennard-Jones clusters is an NP hard problem.[43] Several global optimization methods have been applied to the energy function of Lennard-Jones clusters. These include simulated annealing,[44] genetic algorithm,[11] diffusion equation methods,[45] quantum annealing,[46] and others.[47] The total energy for a Lennard-Jones cluster of N particles is $E_N = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} V_{LJ}(r_{ij})$ where r_{ij} is the distance between the i th and the j th particles and $V_{LJ}(r)$ is the Lennard-Jones two-body potential $V_{LJ}(r) = \frac{1}{r^{12}} - \frac{2}{r^6}$.

For small clusters numbers of Lennard-Jones particles, $(N \leq 6)$, the global energy minimum was located very quickly (less than 1 CPU second on an IBM RS/6000). For larger clusters, we incorporated the partial knowledge that we had by starting with the structure of the smaller clusters and adding additional particles at random. In any "growing" problem, such as minimum-energy configuration of clusters, self-avoiding walks, and protein folding,[40] this systematic approach to solving the structure of large clusters can be incorporated. One of the powerful features of this algorithm is that information such as this can be built into the initialization of the probes.

For a cluster of size N, we begin with $m \times k$ initial pivot probes chosen as follows: $m, N-1$ clusters + one random atom, $m, N-2$ clusters + 2 random atoms, ..., $m, N-k+1$ clusters + k-1 random atoms, and finally m completely random pivot probes (in our calculations we set $k = \frac{N}{2}$). With this set of the initial pivot probes, if the size N cluster has a similar structure to a smaller cluster, the algorithm converges faster than purely random initial points. If the size N cluster has a much different structure than the $N-k$ structure or has one or more local minima near the global minimum, as in the $N = 18$ Lennard-Jones cluster, then the method works no less efficiently than it would with initial pivot probe locations chosen completely at random.

Results and Discussion

Figure 4 illustrates how this method scales with the number of Lennard-Jones particles to be minimized. Using a log-log scale we show that this method scales approximately as $N^{2.9}$, compared to the recently reported modified genetic algorithm, which scales as $N^{4.7}$.[11] One should note that the probability of finding that the global

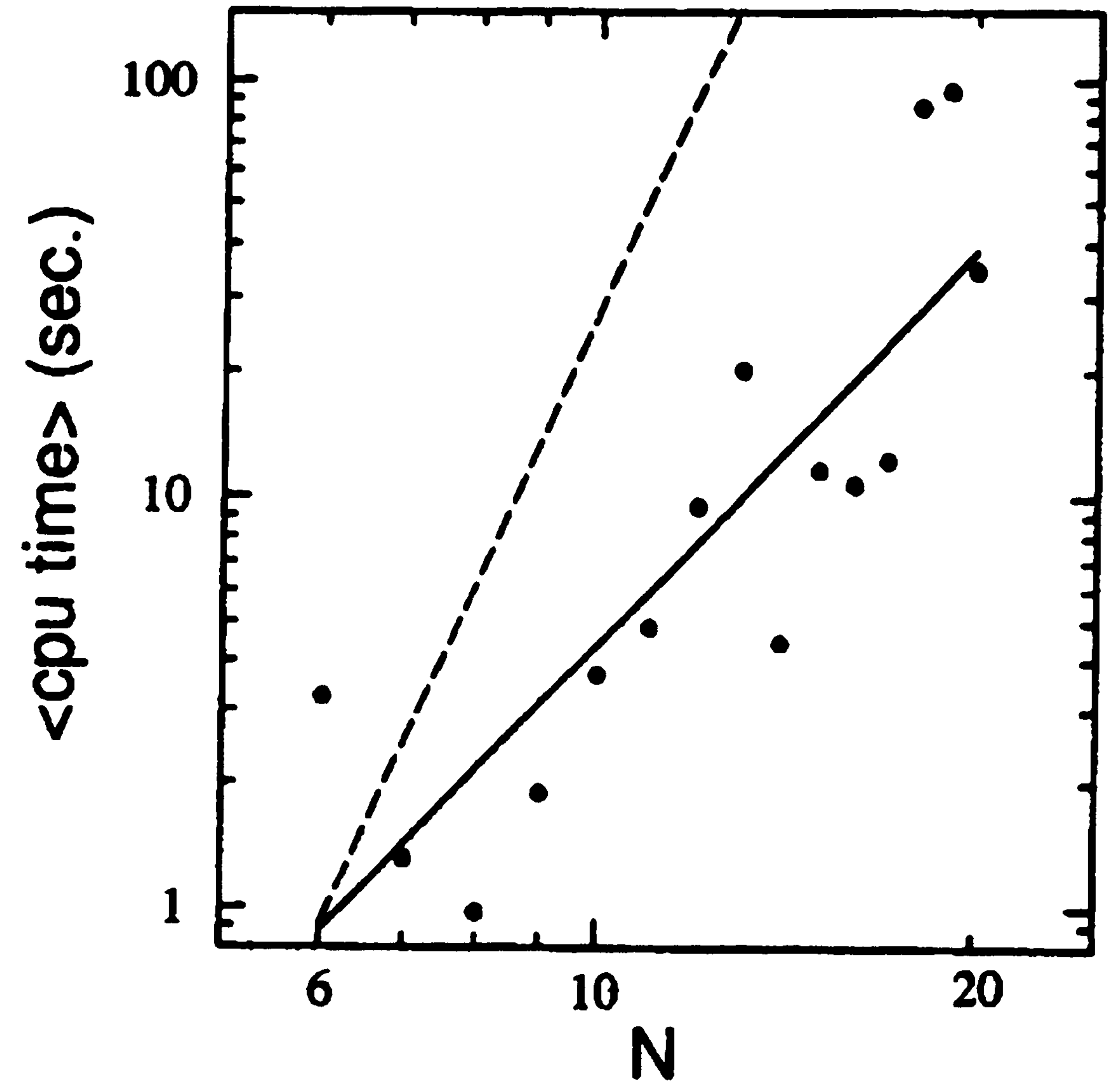

Figure 4 Log-log graph of cpu time versus number of Lennard-Jones particles

energy minimum was strongly correlated with the size of the cluster. It was relatively difficult to obtain the global minimum for "magic number" clusters such as $N = 6$ and $N = 18$. This implies that the difficulty (CPU time) does not scale in a simple way with N but depends on the characteristics of the potential energy hypersurface.[48] In order to reach the exact minimum for the Lennard-Jones clusters, a gradient descent minimization was used once my method met its convergence criteria. Our CPU time given includes this gradient minimization. In this case, we found that a value of $q = 2.7$ worked better than $q = 2.5$ for these criteria. The CPU time on this chart has been experimentally determined on an IBM RS/6000-580 for the method, the genetic algorithm[11] was included for comparison of scaling, and the exact CPU time was not determined. A comparison such as this shows that for extremely large systems it would allow considerable savings in time to use the present method.

We have presented a general method of optimization of arbitrary functions that has shown itself to be easy to implement, does not get easily trapped in local minima, and is extremely fast. Any initial knowledge of the behavior of the function in question is easy to incorporate in the initial conditions of the search, thus lending additional versatility. A twelvefold improvement over previous methods has been demonstrated for the six-dimensional Hartman function. CPU time has been shown to scale approximately as $N^{2.9}$, compared to the modified genetic algorithm,[11] which scales as $N^{4.7}$ for Lennard-Jones clusters. This method was empirically found to be extremely useful, and as of yet there does not exist a rigorous mathematical proof for convergence to the global minimum or for the rate of change of the temperature. This method is currently being used to find minimum energy structures of water clusters.[49]

CHAPTER IV. COMPARISON STUDY

We now compare two implementations of a new algorithm called the pivot method for the location of the global minimum of a multiple minima problem. The pivot method uses a series of randomly placed probes in phase space, moving the worst probes to be near better probes iteratively until the system converges. The original implementation, called the "lowest energy pivot method," Chapter II, chooses the pivot probes with a probability based on the energy of the probe. The second approach, called the "nearest neighbor pivot method," Chapter III, chooses the pivot probes to be the nearest neighbor points in the phase space. We examine the choice of distribution by comparing the efficiency of the methods for Gaussian versus generalized q-distribution, based on the Tsallis entropy in the relocation of the probes. The two implementations of the method are tested with a series of test functions and with several Lennard-Jones clusters of various sizes. It appears that the nearest neighbor pivot method using the generalized q-distribution is superior to previous methods.

Introduction

In Chapters II and III, we have developed two optimization methods, both based on pivot moves through phase space. In the original method, Chapter II, the pivots were chosen based on their energies, while in a more efficient version[2] the pivots were chosen as the nearest neighbor point. The major difference between the methods is the way in which the pivot points are chosen. In effect, phase space is visited in a very different way for the two methods.

To determine what circumstances favor one method over the other, I have run a series of test functions, as well as Lennard-Jones clusters, using both methods in order to better compare the relative strengths and weaknesses of the two approaches to the pivot method.

Figure 5 illustrates visually the differences between the Gaussian (q=1), Cauchy (q=2), and generalized q=2.5 distributions. From this, one can easily see the transition from a localized distribution to a long tail one. Figure 6, Figure 7, and Figure 8 demonstrate a two-dimensional random walk taken by each of these three distributions. Again, one can see that for a Gaussian distribution the majority of the walk is confined to one region of phase space, the Cauchy distribution occasionally has large "jumps" from one region to another, yet spends a large amount of time focusing on specific areas, and the q=2.5 distribution has a high nonlocal character. Such a high nonlocal character insures that all of phase space is being adequately searched on a continuing basis, even at a low temperature.

We tested two "pivot acceptance criteria." The first is the Metropolis algorithm,[24] where the acceptance probability $P_A(x_t \to x_{t+1})$ is given by

$$P_A(x_t \rightarrow x_{t+1}) = \begin{cases} 1 \mathit{if}\ f(x_{t+1}) < f(x_t) \\ e^{-\beta(t)[f(x_{t+1})-f(x_t)]} \mathit{if}\ f(x_{t+1}) \geq f(x_t) \end{cases}$$

where β is a fictitious inverse temperature.

The second criterion used is an absolute acceptance criterion. which is identical to the Metropolis algorithm in the limit $T \rightarrow 0$

$$P_A(x_t \rightarrow x_{t+1}) = \begin{cases} 1 \mathit{if}\ f(x_{t+1}) < f(x_t) \\ 0 \mathit{if}\ f(x_{t+1}) \geq f(x_t) \end{cases}$$

It has been determined that there is no significant difference between these two acceptance criteria. and we use the absolute criterion in the present work.

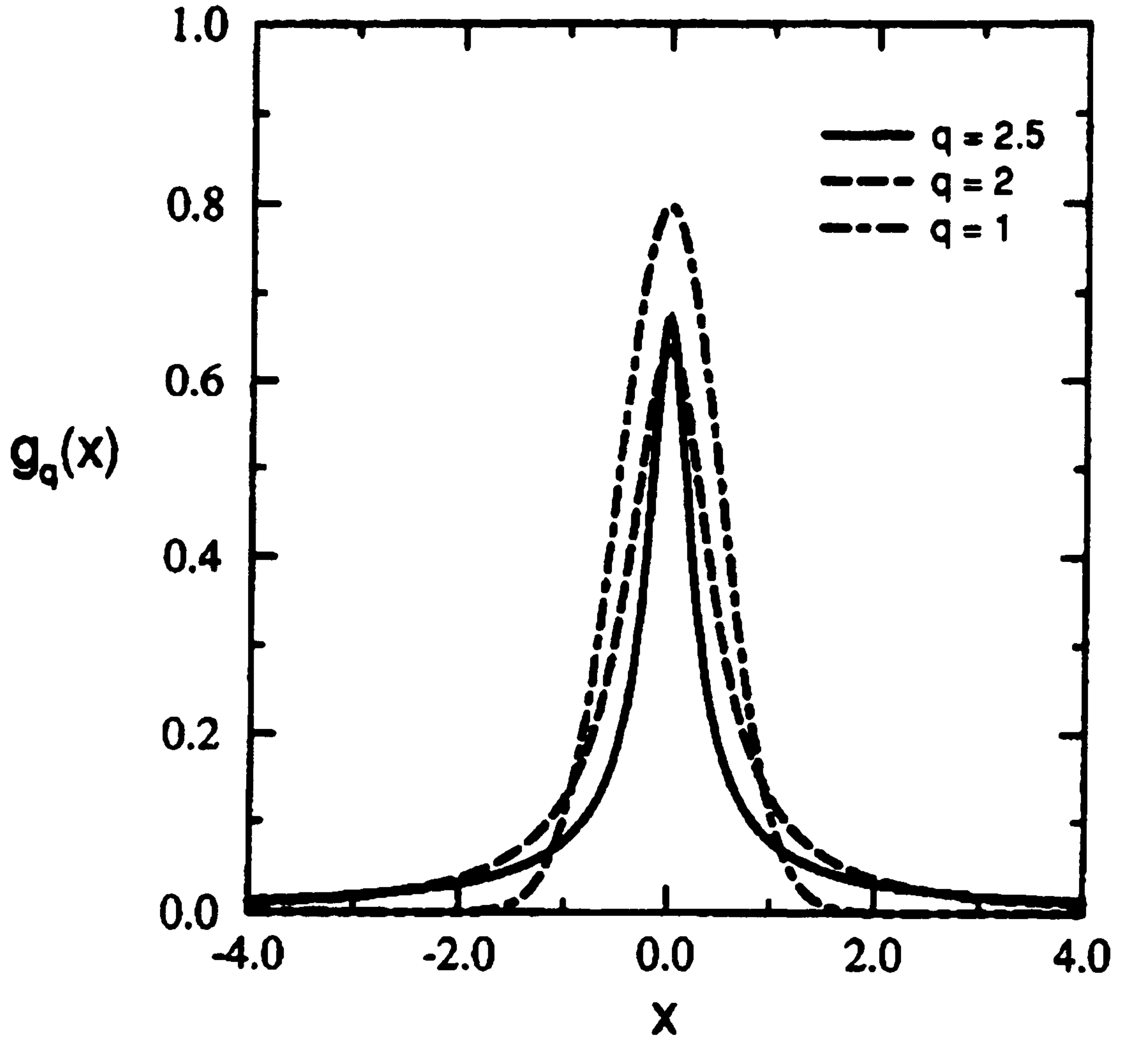

Figure 5 Generalized q Distribution as a Function of x for q=1, q=2, and q=2.5.

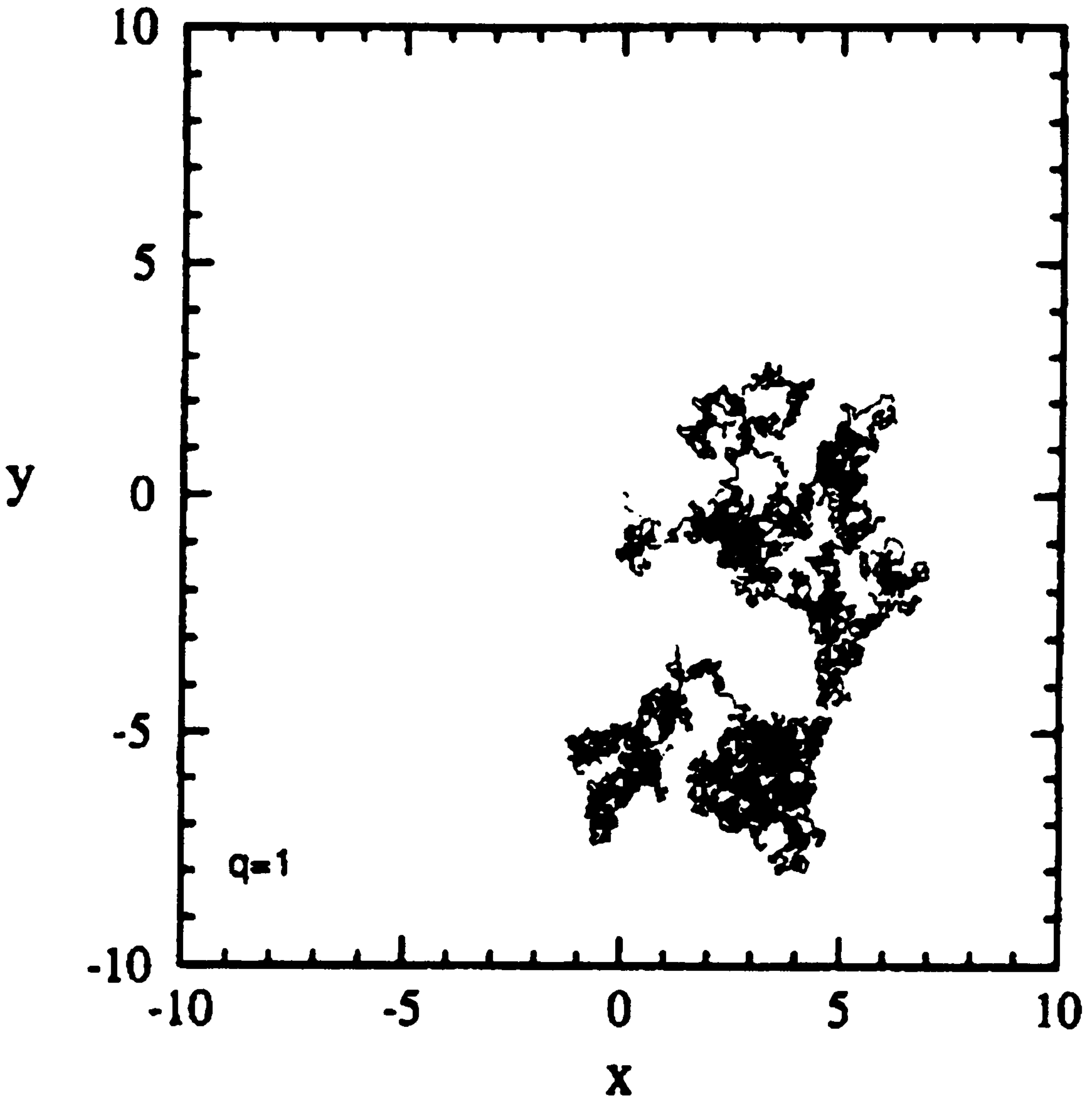

Figure 6 Two Dimensional Random Walk with a Gaussian Distribution

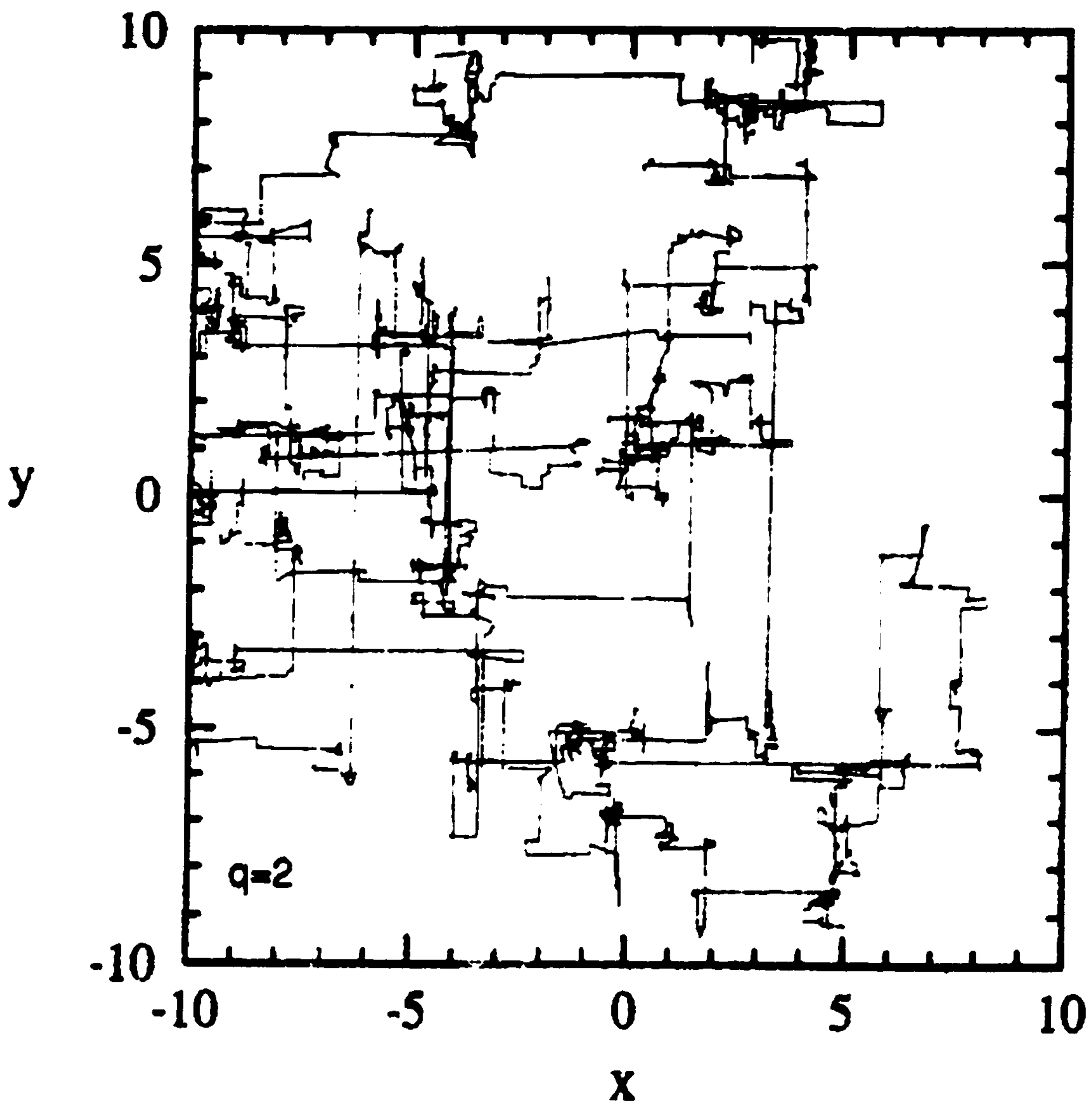

Figure 7 Two Dimensional Random Walk with a Lorentz Distribution

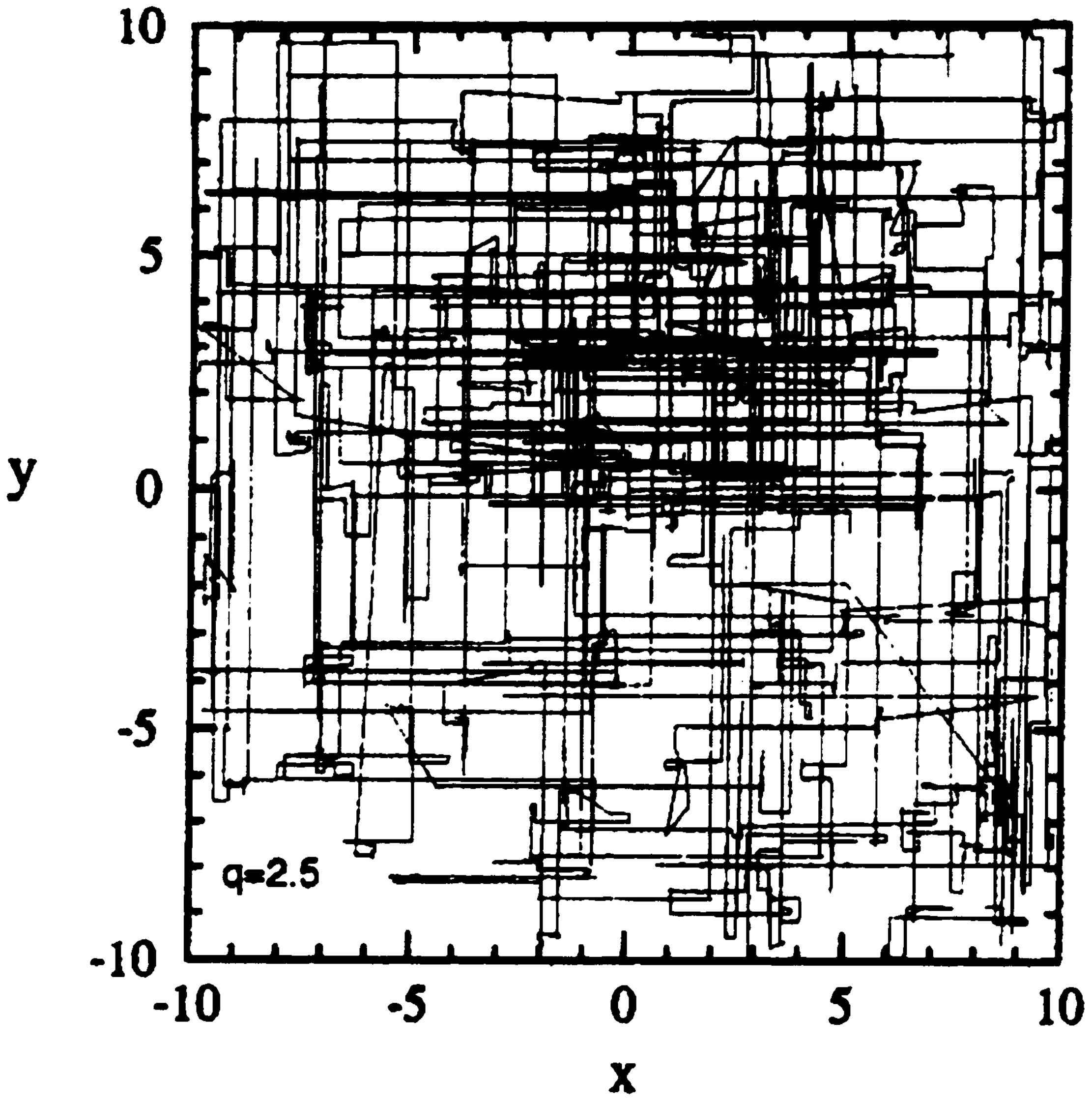

Figure 8 Two Dimensional Random Walk with q Distribution. q=2.5

To empirically determine which of the various possible values of q might work best for my optimization method, we must explicitly test the various distributions against an established set of test functions.

Test Functions

As a preliminary test for the use of the generalized q-distribution, we begin by trying to find the optimal value of q that minimized the number of function calls necessary in the minimization procedure. We have used the Tsallis distribution for all test functions. We have compared the two approaches (LEP and NNP) with one another, both with the Gaussian distribution function as well as NNP with the generalized q-distribution function. The goal is to determine the optimal value of q for each method and to demonstrate how the two approaches compare with one another. To do so, we have run each approach on several well-established functions. These functions included the Goldstein-Price (GP) equation, the Branin (BR), the Hartman three- and six-dimensional functions (H3 and H6), and the Shubert function (SH).[33] Each of these functions is explicitly stated in Appendix A. All of these functions were tested using both the LEP and NNP methods, and each method was tested using the Gaussian distribution and the NNP with generalized q-distribution for the relocation of the probes that are moved. To compare these in a machine-independent fashion, we report the number of function calls for both the Gaussian and the q-distribution function of the NNP approach, and the Gaussian distribution function of the LEP method in Table

3. For comparison, we included in Table 3 the results of pure random search (PRS), simulated annealing types 1 and 2 (SA1 and SA2, respectively) and tabu search (these results are taken directly from Ref. 18). Several comments should be made about these results.

The number of function calls for the LEP approach are different from reported in Chapter II. In the previous chapter, I tried to keep all parameters of the problem identical for all test functions. In the current publication, I have used four of these parameters (number of initial probes, number of probes moved, number of search iterations per standard deviation, σ, and rate of contraction, R) as free parameters. These parameters were varied to minimize the number of function calls for each of the test functions. As a specific example, for the GP function, the best values were found to be 15 initial probes, 5 probes move at each step, 4 steps were taken at any given value of σ, and a rate of contraction of 0.385. It can be clearly seen from Table 3 that the LEP method, when used with my optimal parameters, yields a generous improvement of a factor of approximately 4 over previous methods. This represents the bare method. As we shall see, when modifications such as the use of nearest neighbor selection or a generalized q-distribution are added, additional improvements are gained.

Similarly, the parameters for the NNP approach have been optimized as well. I am therefore comparing the best results in Table 3. Initially I used a Gaussian distribution with the nearest neighbor pivot variation, and improvement was only seen for the H3 and H6 methods, and this suggested to us that perhaps varying the

Table 3 Number of Function Evaluations for Comparison of LEP and NNP[c]

Method[f]	GP	BR	H3	H6	SH
PRS	5125	4850	5280	18,090	6700
MS	4400	1600	2500	6000	-
SA1	5439	2700	3416	3975	241,215
SA2	563	505	1459	4648	780
TS	486	492	508	2845	727
LEP (Gaussian)	112	144	122	1536	281
NNP (Gaussian)	575	358	60	411	360
NNP (q=2.5)	153	68	52	237	114

[c] The majority of this data is taken directly from Reference 18. The results of the lowest energy method and the nearest neighbor method have been included for comparison.

[f] The methods are pure random search (PRS), multistart (MS), simulated annealing type 1 and 2 (SA1 and SA2, respectively), and tabu search (TS). The references for these methods can be found in Reference 18.

distribution may yield some more improvement. Tests were done to determine what the best value of q might be for various functions, and the results of this can be seen in Figure 9, Figure 10, and Figure 11. One can readily see that a value of roughly q=2.5 yields the lowest number of function calls for nearly all of the test functions. This appears to be the best "compromise" between searching locally and nonlocally. The results of these test functions indicate that the NNP method is superior over previous methods, with a four- to twelvefold improvement over the tabu search method[18] for all functions tested.

Lennard-Jones Clusters

As was mentioned in Chapter III, I tested varying values of q to make certain that q=2.5 was still a good value for Lennard-Jones clusters of varying sizes. Figure 12 represents the rate of convergence for different values of q for a cluster of 13 particles, and it can be seen that this convergence reaches a maximum at or very near to q=2.5. Each point plotted represents an average over 100 runs. Figure 13 is a similar illustration for 18 particles in a Lennard-Jones cluster, and again it can be seen that there is a maximal value near q=2.5. This indicates fairly strongly that a value of q near 2.5 seems to be a somewhat universal property of the generalized distribution function for my method of optimization.

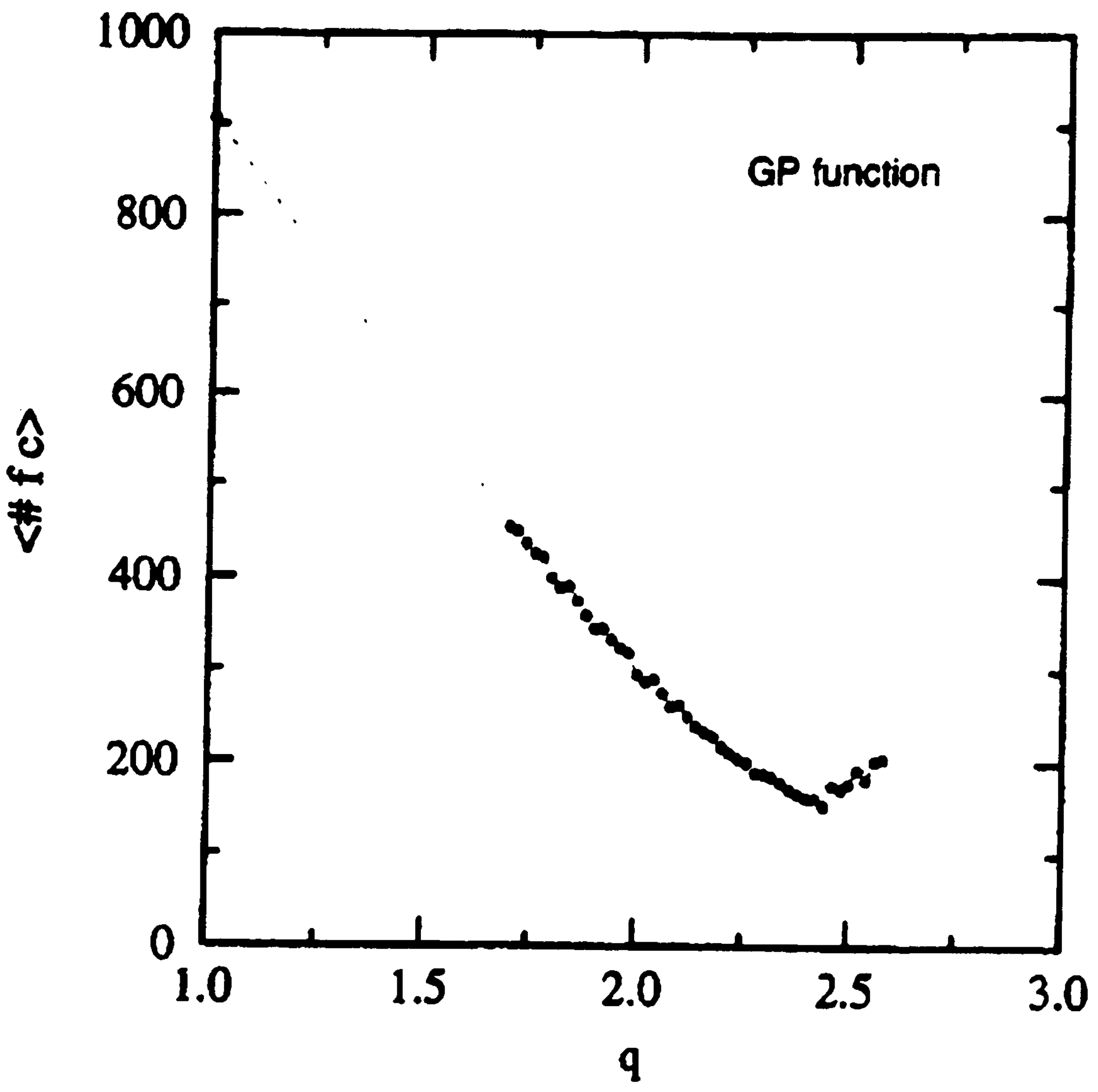

Figure 9 Average Number of Function Calls versus q for the Goldstein-Price Function

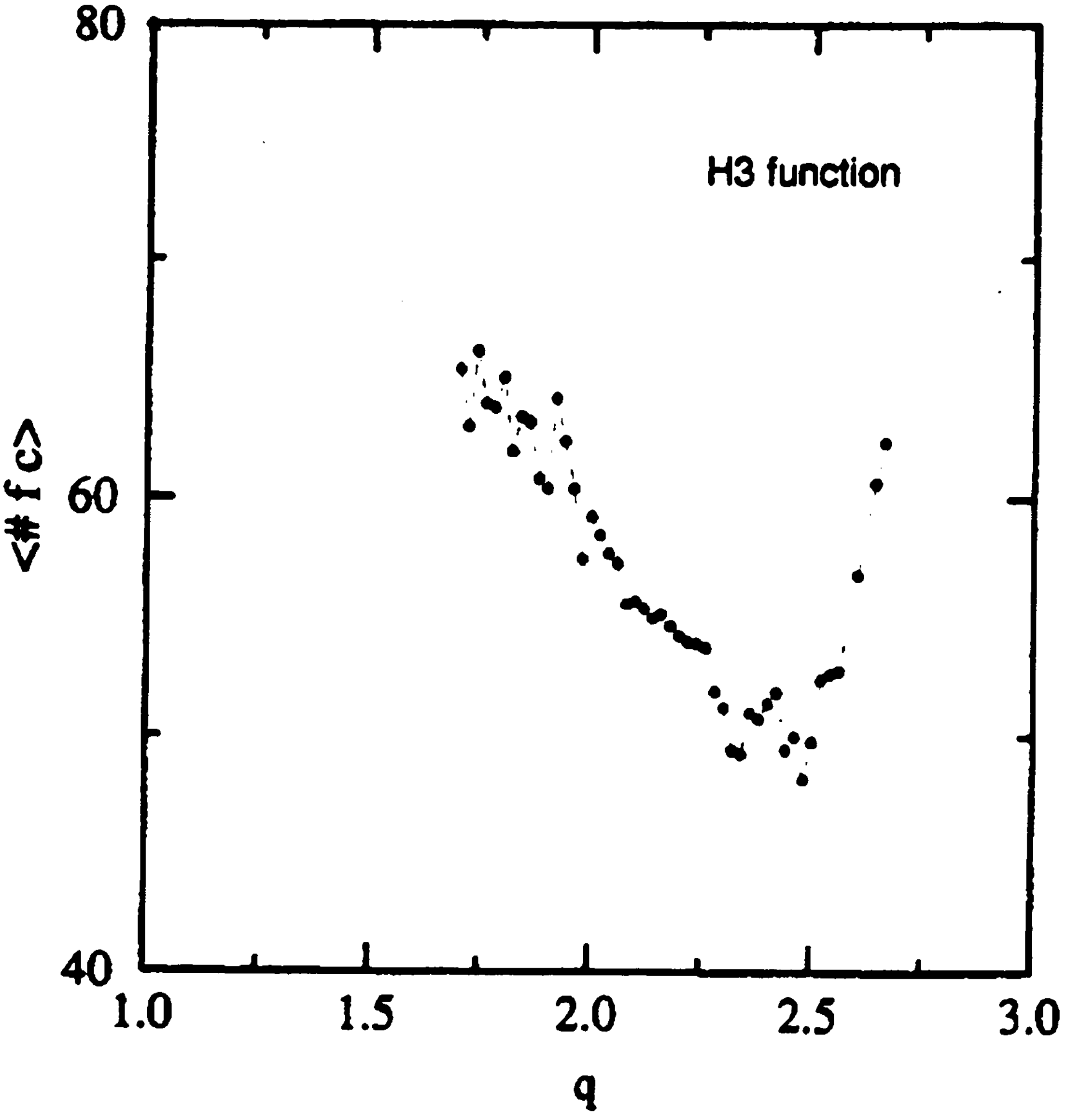

Figure 10 Average Number of Function Calls versus q for the Three Dimensional Hartman Function

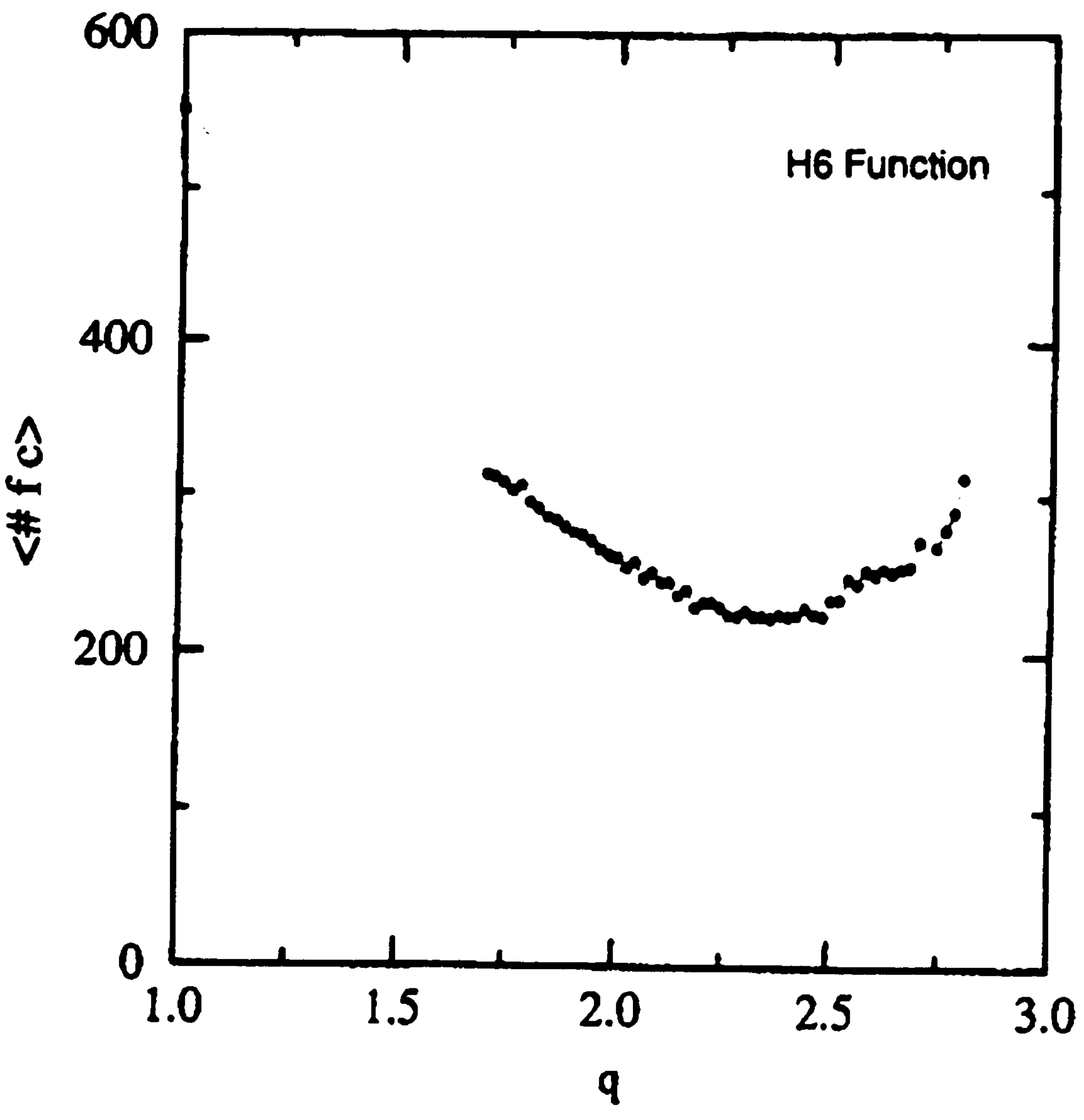

Figure 11 Average Number of Function Calls versus q for the Six Dimensional Hartman Function

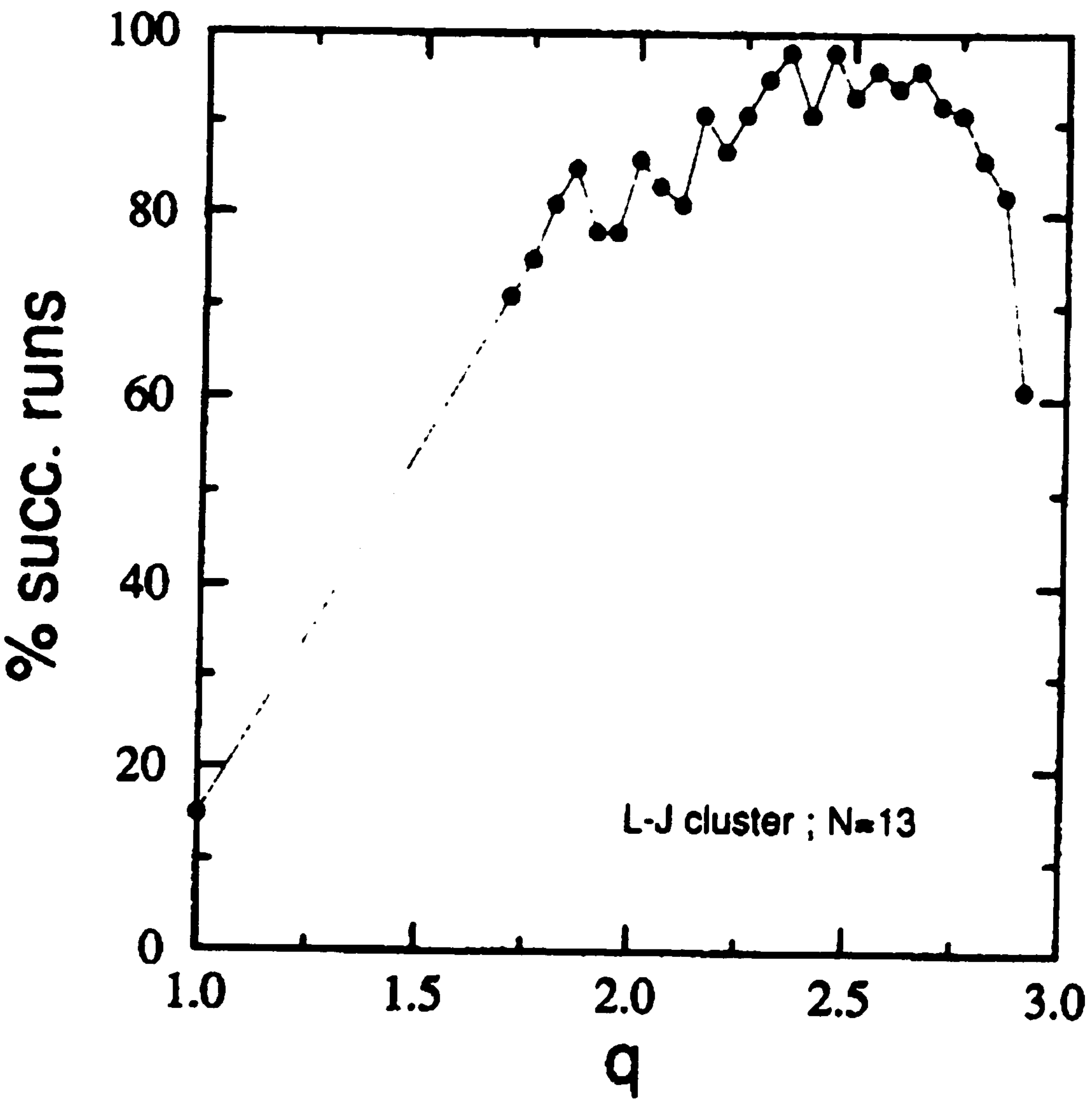

Figure 12 Percentage of Successful Runs vs. q for N=13 Lennard-Jones System

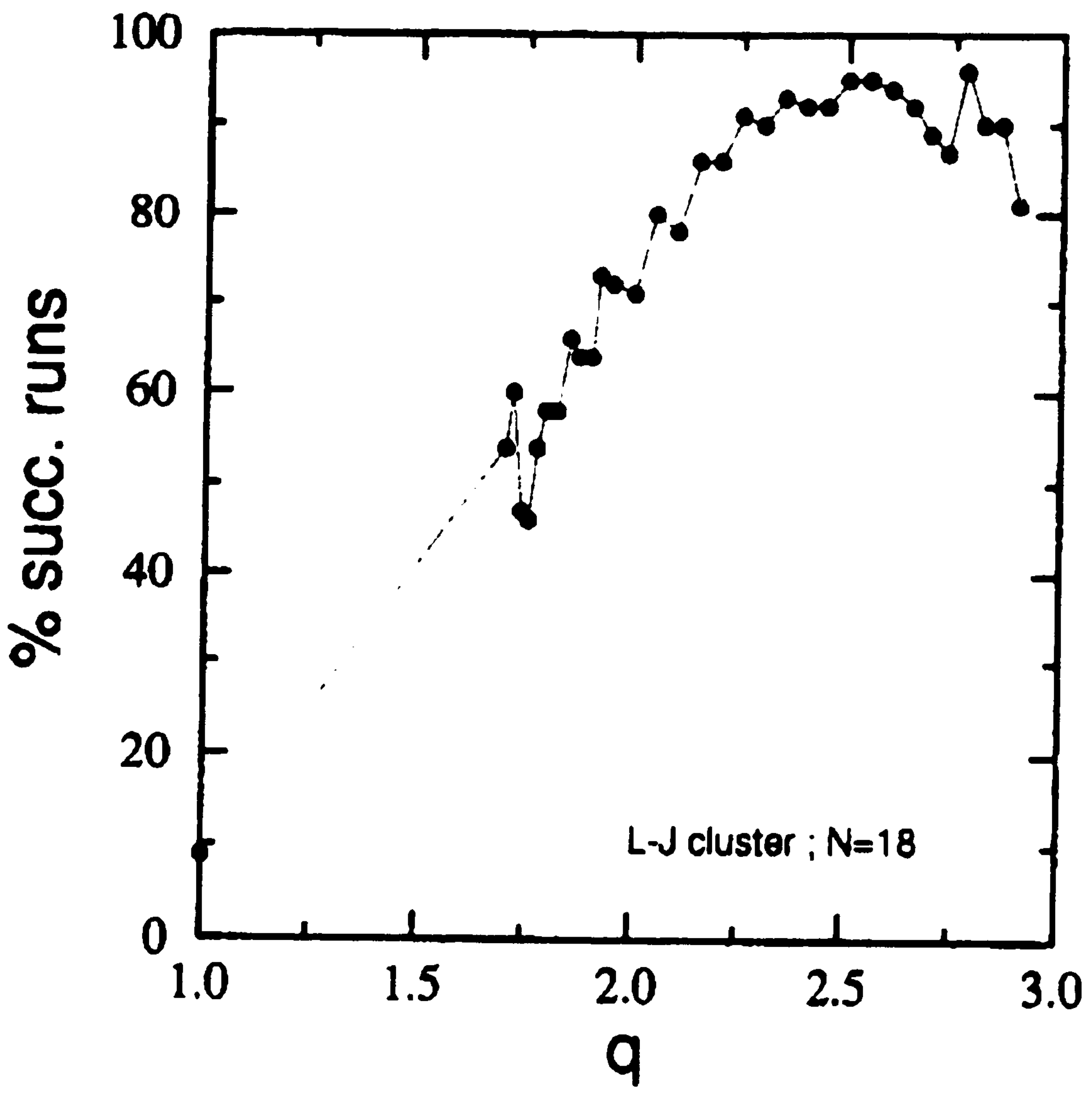

Figure 13 Percentage Of Successful Runs vs. q for N=18 Lennard-Jones System

Discussion

In this section I have presented two different pivot methods and compared them, the lowest energy pivot method (LEP) and the nearest neighbor method (NNP). Using a Gaussian distribution it appears that the lowest energy pivot is good for low dimensional functions, while for higher order potentials the nearest neighbor method shows its strength. Once a generalized q-distribution is used in place of a Gaussian distribution, the nearest neighbor pivot method shows even more substantial improvement, especially for the high-dimensional functions. Having tested this with a range of values for q indicates that the value of q approximately equal to 2.5 gives the best results for this method. I concluded that when using the generalized q-distribution the NNP method is superior to previous methods.

As an explicit demonstration of the improvements our method can yield over previous methods, Table I shows that for the simple LEP method using a Gaussian distribution, one can obtain roughly a fourfold improvement for most potentials. More interesting than that, however, is the roughly twelvefold improvement one sees by comparing the NNP method with a q-distribution using q=2.5.

Continuing in my exploration of the pivot methods for optimizing functions, we have investigated the minimization of Lennard-Jones clusters of particles and found our method to be highly satisfactory.

In short, I have presented a new method of optimizing functions that has proven itself to be quite efficient, very flexible, rarely becomes trapped in a local minima, does not require computationally expensive derivatives, and is quite easy to implement. Flexibility is further enhanced by the ability to incorporate any previous knowledge of the potential under investigation into the optimization. As a specific example, we used smaller optimized Lennard-Jones clusters as the starting point for the larger ones. For very large systems, one could use a crystalline structure as a starting point and optimize from there. Although there are established methods[40] for large Lennard-Jones clusters, they require a homogeneous regular structure as a basis, and my method has no such restriction. This method is currently being used for small clusters of water molecules.[49]

CHAPTER V. SOLUTION OF SCHROEDINGER'S EQUATION

Rationale

Modern quantum chemistry computations are generally carried out using only three types of basis sets: Slater orbitals, Gaussian orbitals, and plane waves, the last being reserved primarily for extended systems in solid state. Each of these has their advantages and disadvantages, and are discussed in Szabo and Ostlund in detail,[50] however I present here an abbreviated version of their critique. Slater orbitals behave analytically in a proper fashion, having a non-zero derivative at the origin and dying off slowly as the distance from the center increases. Though molecular wave functions are known to behave in this fashion, it is difficult to perform the integrations needed for quantum mechanical calculations using orbitals of this type. Gaussian orbitals, therefore, are generally used for molecular systems, as the integrals for multicenter Gaussian terms are relatively easy to compute analytically. Gaussian orbitals have the disadvantage of having a derivative of zero at the origin and dying off very quickly as the distance increases. It therefore requires many Gaussian orbitals to properly simulate one Slater orbital, which means that even though each individual computation is cheap in terms of computer time, there must be many such computations done to achieve a high degree of accuracy. Plane waves have their own unique advantages and disadvantages, those being that their periodic nature simulates very well the electronic structure of crystalline solids, but this same periodic nature effectively limits their use to those systems alone.

It is the purpose of this chapter to explore the possibility of eliminating the step of using basis functions in these computations entirely by solving the Schroedinger equation directly via a pivot method of optimization. A method for computing the energy of a given wavefunction is described, and the pivot method is then applied to this in the form of the quantum harmonic oscillator. It is shown that the nearest neighbor pivot method is capable of solving the quantum harmonic oscillator with initial wavefunctions generated entirely at random. There is no knowledge of the solution of the system implicit in the initialization, and this demonstrates the possibility of applying optimization to the solution of more complex systems.

Methodology

In attempting to solve any problem, it helps considerably to have a clear plan for the methods to use, and Schroedinger's equation is certainly no exception. It is best to select a well known system to work with, and in this case the quantum harmonic oscillator makes a good choice. Zeiri, Fattal, and Kosloff have done some work on solving differential equations with the genetic algorithm,[51] as have Finnila etc.,[46] and have provided the groundwork for building this area of research.

The most difficult part of solving the Schroedinger equation is on the decision on how to compute the energy of the system. The Laplacian of the wavefunction is desired in order to obtain the kinetic energy of the system in question, but that is a quite expensive operation to perform. Rather than do this, it is much more useful to recall that a wavefunction can also have a representation in momentum space. Once one makes the decision to handle the wavefunction in momentum space, computation of the energy follows in a natural way.

We can numerically compute the kinetic energy of our wavefunction as follows. By taking the inverse Fourier transform of the wavefunction, we can now multiply the square of the wavefunction in momentum space by the square of the momentum spectra, and then simply divide by twice the mass of the "particle" to get the kinetic energy.

Conceptually this is not very difficult to follow, but the specifics may present a problem. We must divide space into discrete components in order to fairly represent the wavefunction. This does present the problem of deciding how wide the wavefunction should be in physical space, because by limiting the width of the wavefunction we are not only using the desired potential we wish to study, we are also imposing a "particle in a box" condition. Clearly, if we overconfine the wavefunction, the solution will not closely approximate the real answer. It is suggested that several values for the width of the wavefunction be used, and once the system is sufficiently wide, the minimal energy found should no longer decrease.

Consideration must also be given on how finely to divide the space we are investigating. As position and momentum are conjugate variables, the smaller the division in real space we use, the larger the maximum momentum will be. For a solution that accurately resembles the true solution. one must divide space fairly finely. The combination of a wide region of real space with many divisions will yield the most accurate results, as well as the slowest.

It is suggested that one use a number of divisions equal to a power of two. as that enables one to use the fast Fourier transform. A reasonably good implementation of the FFT is presented in Numerical Recipes,[52] and several routines from that have been used in this work. Other libraries almost certainly have the needed subroutines as well.

We now move on to the specifics of how to compute the energy for a given wavefunction. First, the wavefunction must be properly normalized. This is a simple enough task: By dividing each component of the wavefunction by the square root of the sum of the squares of all the components. we are assured that the wavefunction is properly normalized. whether we are requiring it to exist with only real components or we allow both real and imaginary parts. The implementation I use in this work allows both real and imaginary parts. as that is the proper way to do quantum mechanics.

Second, we can compute the effect of the external potential on the wavefunction. As we have selected the width of the system and the number of divisions previously. we can easily compute the location in physical space of every component of the wavefunction as long as we also take into account a possible shift away from the origin. The particular implementation used here has a width of 5.0. shifted to be symmetric around the origin, and has 32 divisions in space. If we can compute the external potential for every point necessary. we can multiply that by the square of the wavefunction at those points and sum over all space. A similar procedure can be done for the kinetic energy operator, keeping in mind that one must be using the normalized momentum representation of the wavefunction.

Lastly, one must include any self-interactions the wavefunction may have. The simple quantum harmonic oscillator does not have any such problem, but if one is attempting to solve a many body problem with optimization, each body in question must be represented in the wavefunction somehow. Each such part must be properly normalized from the other parts, which presents a minor inconvenience at worst.

One way to speed up the computation of the energy of the system is by precomputing any parts which do not explicitly depend on the wavefunction itself.[53] By computing the value of the external potential for all points in space and placing those values into an array, it turns multiple calculations into a single lookup for the computer. A similar computation can be done for the kinetic energy operator. While the momentum does depend on the wavefunction, the value of the momentum, and hence the kinetic energy, at each point in momentum space does not. In a sense, kinetic energy takes the form of a harmonic oscillator in momentum space. It is advised that one take care when initializing the kinetic energy operator, as at least one implementation of the FFT does not return the components in order from lowest momentum to highest, but in a reversed sort of order. The end positions have the lowest absolute momentum, while the central positions have the highest.

Once the needed operators are initialized and the energy is placed into a function, one can then attempt to minimize the energy by any desired method. It was found that the pivot methods work quite well by varying the wavefunction in momentum space as well as real space. If they vary the wavefunction only in real space, it is difficult for them to solve the system at all, even when using a linear descent method at the end to find the lowest energy available.

Results

I chose to initialize the initial wavefunctions with completely random values in order to demonstrate that the pivot methods are capable of solving a simple system with no prior knowledge. One could expect the optimizations to run even more quickly if one initialized the system close to the known solution.

In short, it was found that both the lowest energy pivot method and the nearest neighbor pivot method can solve the quantum harmonic oscillator, so long as one varies the components of the wavefunction with a generalized q distribution, where q=2.5, and both appear unable to solve the system when restricted to using the Gaussian distribution as a method for varying the system. Table 4 illustrates this by showing the results for one hundred runs of each combination of LEP and NNP with the Gaussian and generalized q distributions. The time listed is in user seconds on a Silicon Graphics O2 system with 128 MB of RAM. Performance certainly may vary depending on the system and the compiler used, but this shows quite plainly that even though the q distribution appears to take about 2.5 times longer than the Gaussian to run, it does give substantially better results. It is interesting to note that the lowest energy method ran in about 66% of the time that the nearest neighbor method did, and when using the q distribution there was little change in the results.

Conclusion

I have shown here that it is possible to solve a simple quantum system by means of the pivot methods for optimization. The generalized q distribution appears to be critically better than the Gaussian distribution for solving this type of problem. Despite the superior performance of the nearest neighbor method over the lowest energy method earlier on, it is interesting to see that in this sample problem, the lowest energy method outperforms the nearest neighbor method.

These results suggest that it may be possible to solve more complex systems directly by these methods where more conventional means may prove intractable. Initializing systems with a better initial guess than pure noise, can only lead to a quicker convergence of the system to a desired minima. Ideas on how this methodology may be extended to other quantum systems such as Helium, Lithium, and simple molecular systems will be presented in the final section of this work.

Table 4 Average Times and Energies for the Quantum Harmonic Oscillator

	Gaussian Distribution	q Distribution (q=2.5)
LEP average time computer seconds	62.23958	146.42696
average energy atomic units	1.002401	0.5039192
NNP average time computer seconds	93.68348	224.79778
average energy atomic units	0.8082680	0.5025765

CHAPTER VI. CONCLUSIONS AND RECOMMENDATIONS

As finding optimal values of functions is a very important problem in many fields. it is useful to find new and better ways for doing just this. In this work I have presented a new method. along with some variations. that appears to not only be extremely effective for systems of low dimensionality. it scales well to systems of higher dimensionality.

I have shown the evolution of the method. from the most basic form of merely discarding bad probes and creating new ones based on the better. to more sophisticated methods of eliminating duplicated efforts by moving probes near each other to further apart. It appears that nonlocal searching of potential energy surfaces is very useful in avoiding wells that might otherwise trap the optimization.

In a concrete comparison between the methods. we have seen that for our test suite of functions. the nearest neighbor energy method outperforms the lowest energy method. and it also outperforms the genetic algorithm quite nicely when finding minimal energy structures of Lennard-Jones systems of particles. This latter result is not trivial, as many people use Lennard-Jones type potentials to model several different types of clusters.

Appendix B consists of FORTRAN source code for the solution of the three dimensional Hartman potential. This code has been tested, and is an efficient implementation of both the lowest energy and the nearest neighbor pivot methods. To properly use the code given, one must use some routines from Numerical Recipes[52] or modify the code to use similar routines in a different library. A minimal amount of modification from the source code will allow one to use either the lowest energy method or the nearest neighbor method, and equally simple changes will allow one to choose the value for q for use in the generalized distribution, or to simply use a Gaussian distribution.

Appendix C consists of code for solving the quantum harmonic oscillator, and any routines not present in Appendix C can either be found in Appendix B or Numerical Recipes[52] as before. Slightly greater changes must be made to the routines in Numerical Recipes, most notably that prior to the evaluation of the energy of the wavefunction in question, one must normalize it. This is a simple change to make to the code, but if it is not done, erroneous results can occur.

When attempting to solve the trial system of the quantum harmonic oscillator, we once again find that nonlocal searching seems to be a key element, but we discover that the nearest neighbor method is outperformed by the lowest energy method. It appears that both variations are potentially useful, and one should not arbitrarily discard either method out of hand.

It is suggested that this line of research be continued to determine if more complex systems will fall to this new method. Three dimensional quantum systems such as the three dimensional harmonic oscillator, the hydrogen atom, and the helium atom are obvious choices to be explored. An N-dimensional FFT routine is easily found,[52] and the method of initializing the operators prior to use can still be used to speed up energy computations. As with the one dimensional system, one should take care when setting up those operators. It will help matters to investigate the method by which FORTRAN lays out arrays in memory. Even though the operators are one dimensional arrays, they can be treated as three dimensional arrays with the proper nesting of DO loops and the EQUIVALENCE statement. One may run into the problem of inadequate memory in the system, as the FFT only accepts powers of two along every axis. A cube divided eightfold in every direction has 512 real elements and 512 imaginary elements in space, which is a tractable if slow problem, but one divided sixteenfold has 4096 real and 4096 imaginary elements. Dividing space well enough to get satisfactory answers may yield problems for even the most efficient of optimization schemes.

When initializing systems such as the helium atom, it is strongly suggested that one use an approximation that is reasonably close to the true solution. While the pivot methods should be able to converge from pure random initialization, as shown for the one dimensional quantum harmonic oscillator, for systems with as many parameters as was just described, a great savings of time can be had with an appropriate initialization. Recall that for solving clusters of Lennard-Jones particles in Chapter III we used the solution of smaller clusters as the basis for larger ones. Such a method is a first order approximation to the true answer, and it should work well here as well. For systems that lack a simple approximation that is easy to implement. it is suggested that one could perhaps use established tables of Gaussian expansion of wavefunctions for purposes of solving for lithium and other elements. or even for molecules. It is strongly recommended that in attempting to solve molecules with this method that one use the Born-Oppenheimer approximation and treat the nuclei as fixed. If one does this. the potential of the nuclei can be placed into an operator array to speed up computation.

Another interesting field of research still open to explore with the pivot methods is the problem of minimal energy structures for clusters of carbon. water. noble gases. and many other systems. For the particularly ambitious. the problem of protein folding is one very much worth solving. and it is hoped that these methods can lay a solid foundation for such lofty goals.

REFERENCES

REFERENCES

[1] A. F. Stanton, R. E. Bleil. and S. Kais, J. Comp. Chem. 18, 594 (1997).

[2] P. Serra, A. F. Stanton, and S. Kais, Phys. Rev. E 55, 1162 (1997).

[3] P. Serra, A. F. Stanton, R. E. Bleil, and S. Kais, J. Chem. Phys., 106, 17 (1997).

[4] K. B. Lipkowitz and D. B. Boyd, Eds., Reviews in Computational Chemistry, Vol. III, VCH, New York, 1992.

[5] L. C. W. Dixon and G. P. Szego, Towards Global Optimization, Vols. 1 and 2, North-Holland, New York, 1975.

[6] D. R. Herschbach, J. Avery, and O. Goscinski, Dimensional Scaling in Chemical Physics, Kluwer, Dordrecht, 1993.

[7] T. L. Blundell, B. L. Sibanda, M. J. E. Sternberg, and J. M. Thornton, Nature, 326, 347 (1987).

[8] K. A. Dill, S. Bromberg, K. Yue, K. M. Fiebig, D. P. Yee, D. D. Thomas, and H. S. Chan, Prot. Sci., 4, 561 (1995).

[9] M. Oresic and D. Shalloway, J. Chem. Phys., 101, 9844 (1994).

[10] L. Piela, J. Kostrowicki, and H. A. Scheraga, J. Phys. Chem., 93, 3339 (1989).

[11] S. K. Gregurick, M. H. Alexander, and B. Hartke, J. Chem. Phys., 104, 2684 (1996).

[12] R. S. Berry, Int. J. Quantum Chem., 58, 657 (1996); R. S. Berry, J. Jellinek, and G. Natanson, Phys. Rev. A 30, 919 (1984); D. J. Wales and R. S. Berry, J. Chem. Phys. 92, 4283 (1990).

[13] H. B. Schlegel, in Ab Initio Methods in Quantum Chemistry-I, edited by K. P. Lawley (Wiley, New York, 1987).

[14] C. Floudas and P. M. Pardalos, Recent Advances in Global Optimization, Princeton University Press, Princeton, NJ, 1991.

[15] J. E. Dennis and R. B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, (Prentice-Hall, Englewood Cliffs, NJ, 1983).

[16] J. C. Gilbert and C. Lemarechal, Math. Prog. 45, 407 (1989); X. Zou, I. M. Navon, F. X. Le Dimet, A. Nouailler, and T. Schlick, SIAM J. Opt. 3, 582 (1993).

[17] M. H. Wright, Acta Numerica, 1, 341 (1991); J. Nocedal, Acta Numerica, 1, 199 (1991).

[18] D. Cvijovic and J. Klinowski, Science, 267, 664 (1995).

[19] A. V. Levy and A. Montalvo, SIAM J. Sci. Stat. Comput. 6, 15, (1985).

[20] A.V. Levy and S. Gomez, The tunneling method applied to global optimization, in Numerical Optimization 1984, P. T. Boggs, R.H. Byrd, and R.B. Schnabel, eds., SIAM, Philadelphia, 213 (1985).

[21] D. Shalloway, J. Global Opt. 2, 281 (1992).

[22] D. D. Frantz, D. L. Freeman, and J. D. Doll, J. Chem. Phys. 93, 2769 (1990).

[23] P.M. Pardalos and J.B. Rosen, Constrained Global Optimization: Algorithms and Applications, Lecture Notes in Computer Science, 268, Springer, Berlin (1987).

[24] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, J. Chem. Phys 21, 1087 (1953).

[25] K. S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, Science, 220, 671 (1983).

[26] K. S. Kirkpatrick, J. Stat. Phys. 34, 975 (1984).

[27] H. Szu and R. Hartley, Phys. Lett. A 122, 157 (1987).

[28] C. Tsallis and D. A. Stariolo, Physica A 233, 395 (1996).

[29] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor (1975).

[30] L. Davis, editor, Handbook of Genetic Algorithms, Van Nostrand-Reinhold, London (1991).

[31] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, Reading, MA, 1989.

[32] D.B. Hibbert, Chemom. Intell. Lab Syst. 19 (1993) 277-293 / Tutorial.

[33] A. Torn and A. Zilniskas, Global Optimization (Princeton University Press, Princeton, NJ, 1991).

[34] A. Rinooy Kan and G. Timmer, in Numerical Optimization, 1984, P. Boggs, R. Byrd, and R. Schnabel, Eds., SIAM, Philadelphia, PA, 1985, p. 245.

[35] C. Tsallis, J. Stat. Phys. 52, 479 (1988).

[36] D. A. Stariolo and C. Tsallis, in Annual Review of Computational Physics II, edited by D. Stauffer (World Scientific, Singapore, 1995), p. 343.

[37] T. J. P. Penna, Phys. Rev. E 51, R1 (1995).

[38] I. Andrichioaei and J. Straub, Phys. Rev. E 53, R3055 (1996).

[39] The FORTRAN code for the random number generator of the q distribution is given in Ref 28.

[40] J. Gu and B. Du, in DIMACS 23, Global Optimization of Nonconvex Energy Functions: Molecular Conformation and Protein Folding, edited by P. M. Pardalos, D. Shalloway, and G. Xue (American Mathematical Society, Providence, 1996), p. 65.

[41] R. S. Berry, T. L. Beck, H. L. Davis, and J. Jellinek, in Advances in Chemical Physics, edited by I. Prigogine and S. A. Rice (Wiley, New York, 1988), Vol. 70B, p. 75.

[42] M. R. Hoare, Adv. Chem. Phys. 40, 49 (1979).

[43] L. T. Wille and J. Vennik, J. Phys. A. 18, L419 (1985).

[44] L. T. Wille, Chem. Phys. Lett. 133, 405 (1987).

[45] J. Kostrowicki, L. Piela, B. J. Cherayil, and H. A. Scheraga, J. Phys. Chem. 95, 4113 (1991).

[46] A. B. Finnila, M. A. Gomez, C. Sebenik, C. Stenson, and J. D. Doll, Chem. Lett. 219, 343 (1994).

[47] P. M. Pardalos and G. L. Xue, J. Global Opt. 4, 117 (1994).

[48] J. Ma and J. E. Straub, J. Chem. Phys. 101, 533 (1994).

[49] P. Nigra and S. Kais, Chem. Phys. Letters(submitted).

[50] A. Szabo and N. S. Ostlund, Modern Quantum Chemistry (Dover, Mineola, 1996).

[51] Y. Zeiri, E. Fattal, and R. Kosloff, J. Chem. Phys., 102, 1859 (1995).

[52] W.H Press, S.A. Teukolsky, W.T. Vetterling, and B. P. Flannery, Numerical Recipes in FORTRAN, Second Edition (Cambridge University Press, 1992).

[53] S.D. Belair, personal conversation, Feb. 1999.

APPENDICES

APPENDICES

Appendix A: Test Functions

Following are the test functions used in this paper, along with relevant tables of parameters for those functions.

Goldstein-Price

Equation 1: The Goldstein-Price Function

$$f(x_1,x_2)=\left[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)\right]\times$$
$$\left[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)\right]$$

where $-2\le x_i\le 2$. The global minimum is equal to 3 and the minimum point is located at $(0,-1)$. There are four local minima in the region of interest.

Branin

Equation 2: The Branin Function

$$f(x_1,x_2)=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos(x_1)+10$$

where $-5 \le x_1 \le 10$, $0 \le x_2 \le 15$. The global minimum is approximately 0.398 and is reached at three points: $(-3.142, 12.275)$, $(3.142, 2.275)$, and $(9.425, 2.425)$.

Hartman

Equation 3: The Hartman Function

$$f(x_i) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{n} a_{ij}\left(x_j - p_{ij}\right)^2\right],\ 0 \le x_j \le 1,$$

where the parameters, a_{ij}, p_{ij}, and c_i are given in Table 5 and Table 6. For $n = 3$ the global minimum is equal to -3.86 and is reached at the point $(0.114, 0.556, 0.852)$. For $n = 6$, the minimum is -3.32 and is reached at $(0.201, 0.150, 0.477, 0.275, 0.311, 0.657)$.

Shubert

Equation 4: The Shubert Function

$$f(x_1, x_2) = \left[\sum_{i=1}^{5} i \times \cos[(i+1)x_1 + i]\right] \times \left[\sum_{i=1}^{5} i \times \cos[(i+1)x_2 + i]\right],\ -10 \le x_i \le 10$$

and has 760 local minima in this region, 18 of which are global with $f(x_1, x_2) = -186.7309$.

Table 5 Parameters for the Three Dimensional Hartman Function

i	a(i,1)	a(i,2)	a(i,3)	c(i)	p(i,1)	p(i,2)	p(i,3)
1	3.0	10	30	1.0	0.3689	0.1170	0.2673
2	0.1	10	35	1.2	0.4699	0.4387	0.7470
3	3.0	10	30	3.0	0.1091	0.8732	0.5547
4	0.1	10	35	3.2	0.03815	0.5743	0.8828

Table 6 Parameters for the Six Dimensional Hartman Function

i	a(i,1)	a(i,2)	a(i,3)	a(i,4)	a(i,5)	a(i,6)	c(i)
1	10	3	17	3.5	1.7	8	1
2	0.05	10	17	0.1	8	14	1.2
3	3	3.5	1.7	10	17	8	3
4	17	8	0.05	10	0.1	14	3.2

i	p(i,1)	p(i,2)	p(i,3)	p(i,4)	p(i,5)	p(i,6)
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.2329	0.4315	0.8307	0.3736	0.1004	0.9991
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

Appendix B: Source Code for Pivot Optimization

A concrete example of the use of the pivot method of optimization is presented in the form of Fortran source code. The system being solved here is the three dimensional Hartman function. This specific example was chosen because it demonstrates the utility of both the lowest energy and the nearest neighbor methods, it requires the function to be initialized prior to being used, and it runs extremely quickly. The global minimum is found nearly one hundred percent of the time with this code to within a three percent error. The stopping criteria used here are for either the best probe to be within three percent of the known minimal energy, or for three thousand iterations to elapse. Only six probes are used in this instance to show how fast this method can be with a low dimensional system. Typically fewer than one hundred function calls are required with this implementation to find the global minimum.

To create the files needed to run this program, one should just use the "make" command. This will compile all of the Fortran files into object code and link that into executable "run", which can then perform the optimization.

```
c***************************************************
c23456789
c       file cool.f
c
c       The function cool is used to reduce the distance of
c       moves from a chosen pivot. Three cooling schemes are
c       given here. The first one is the scheme used by
c       Tsallis etc. in generalized simulated annealing. The
c       second is simple decay, and the third is no cooling
```

```
c       at all.
c
      real function cool(tempin,q,fac,ngen,temp)

      integer ngen
      real tempin, q, fac, fac1,fac2

      fac1 = (2**(q-1.0))-1.0
      fac2 = ((1.0+real(ngen))**(q-1.0))-1.0

      cool = tempin * (fac1/fac2)

c      cool = temp * fac

c      cool = temp

      return

      end
c*****************************************************

c*****************************************************
c23456789
c       file dist.f
c
c       The function dist is used to compute a distance between two
c       probes.  The entire array of probes is passed to the function,
c       as are which two probes to compute this distance.
c
      real function dist(psi, first, sec)

      parameter (NMAX=3,nprobes=6)
      integer first, sec, i
      real psi(nprobes,NMAX), temp

      temp = 0.0

      do i = 1, NMAX
        temp = temp + ((psi(sec,i) - psi(first,i)) *
     2       (psi(sec,i) - psi(first,i)))
      enddo
      temp = sqrt(temp)

      dist = temp

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c    file func.f
c
c    The function func computes the energy of a probe and returns it
c    to whatever routine called it.  This is set up for the
c    three dimensional Hartman function.  To change it to the six
c    dimensional Hartman function, one merely has to change NMAX to
c    six here.  No other changes to this function would be needed,
c    but one would have to properly initialize a, c, and p elsewhere.
c    Also, NMAX in all other subroutines must be set to six.
c
      real function func(x)

      integer NMAX,i,icount
      parameter (NMAX = 3)
      real x(NMAX),a(4,NMAX),c(4),p(4,NMAX),temp

      common /hart3/ a,c,p
      common /count/ icount

      func = 0.0

      icount = icount + 1

      do i = 1,4
        temp = 0.0
        do j = 1, NMAX
          temp = temp + (a(i,j)*((x(j)-p(i,j))**2))
        enddo
        func = func + (c(i) * exp(-1.0*temp))
      enddo
      func = -1.0 * func

      return

      end

c*************************************************

c*************************************************
c23456789
c
c     file initbounds.f
c
c     The subroutine initbounds just initialize the minimum and
c     maximum values a probe is allowed to reach.  Many forms of
c     optimization confine the search area to a given region of
c     phase space, and this is the place to set that up.  A
c     quantum mechanical system would not use a function like this.
c     Instead, it would normalize the wavefunction to make sure
c     the probability density was equal to one.
c     Also, this sort of routine would not be used for an unbounded
c     optimization.
c
      subroutine initbounds()

      parameter (NMAX = 3)

      real xmin(NMAX),xmax(NMAX)

      common /bounds/ xmin,xmax

      do i = 1, NMAX
        xmin(i) = 0.0
      enddo

      do i = 1, NMAX
        xmax(i) = 1.0
      enddo

      return

      end
```

```
c*****************************************************

c*****************************************************
c23456789
c
c    file inith3.f
c
c    The subroutine inith3.f initializes the arrays a. c. and p
c    for the three dimensional Hartman potential and places it
c    into the common block for use in the potential.  NMAX is
c    a parameter set equal to three. to make it simpler to
c    use this as the basis for initializing the H6 potential.
c    Unfortunately a do loop cannot be used for this initialization.
c    as there is no easily discernible pattern to these constants.
c    save for a(i.2).
c
      subroutine inith3()

      parameter (NMAX=3)
      real a(4.NMAX).c(4).p(4.NMAX).besten

      common /hart3/ a.c.p
      common /minim/ besten

      a(1,1) = 3.0
      a(2.1) = 0.1
      a(3.1) = 3.0
      a(4.1) = 0.1

      do i = 1,4
        a(i.2) = 10.0
      enddo

      a(1.3) = 30.0
      a(2.3) = 35.0
      a(3.3) = 30.0
      a(4.3) = 35.0

      c(1) = 1.0
      c(2) = 1.2
      c(3) = 3.0
      c(4) = 3.2
      p(1,1) = 0.3689
      p(2,1) = 0.4699
      p(3,1) = 0.1091
      p(4,1) = 0.03815

      p(1,2) = 0.1170
      p(2,2) = 0.4387
      p(3,2) = 0.8732
      p(4,2) = 0.5743

      p(1,3) = 0.2673
      p(2,3) = 0.7470
      p(3,3) = 0.5547
      p(4,3) = 0.8828

      besten = -3.86

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c        file initprobe.f
c
c        The subroutine initprobe generates one probe according
c        to a desired scheme.  Two are given here.  The first
c        is merely random within the given boundaries, and the
c        second initializes all probes with the known solution.
c        This is handy to make sure the energy is being computed
c        properly.
c
         subroutine initprobe(p)

      integer NMAX,i,j
      parameter (NMAX=3)
      real p(NMAX),range,xmin(NMAX),xmax(NMAX)

      common /rnd1/ idum
      common /bounds/ xmin,xmax

      do i = 1,NMAX
        range = xmax(i) - xmin(i)
        p(i) = xmin(i) + (ran1(idum) * range)
      enddo

c      p(1) = 0.114
c      p(2) = 0.556
c      p(3) = 0.852

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c        file initset.f
c
c        The subroutine initset initializes the set of probes
c        with their values for the first cycle of the optimization,
c        along with the initial energies for those probes.
c
      subroutine initset(probes,energy)

      integer nprobes,NMAX,i,j
      parameter (nprobes=6,NMAX=3)
      real probes(nprobes,NMAX),p(NMAX),energy(nprobes)

      do i = 1, nprobes
        call initprobe(p)
        energy(i) = func(p)
        do j = 1,NMAX
          probes(i,j) = p(j)
        enddo
      enddo

      return

      end
```

```
c*************************************************
c*************************************************
c23456789
c
c       file initxi.f
c
c       The subroutine initxi just fills the array xi with a
c       set of values.  Xi is used by powell, and represents a
c       set of vectors to move along in its optimization.  The values
c       it gets filled with here don't really matter, as powell
c       changes them repeatedly, but I don't like using arrays
c       without some sort of starting value.
c
      subroutine initxi(xi)

      integer NMAX,i,j
      parameter(NMAX=3)
      real xi(NMAX,NMAX)

      do i = 1,NMAX
        do j = 1,NMAX
c         xi(i,j) = ran1(idum)
          xi(i,j) = 1.0
        enddo
        xi(i,i) = 1.0
      enddo

      return

      end

c*************************************************
c*************************************************
c23456789
c
c       file lepivot.f
c
c       The subroutine lepivot is the heart of the lowest energy
c       pivot method.  It takes an initialized set of probes along
c       with their energies and tries to find the minimum energy
```

```
c       possible.  It only runs for a number of cycles equal to
c       the parameter ncyc, and does not use any other stopping
c       criteria.  One could add more if so desired.
c       First it sorts the probes by their energies, chooses
c       pivots based on those energies, and then relocates the
c       worst probes around the pivots selected.  Every ncool
c       cycles it cools the system by calling the function cool.
c       Nfrac is the fraction of the total number of probes to move.
c
      subroutine lepivot(probes,energy)

      integer nprobes,NMAX,i,ncyc,ncool
      real q,fac,scale,tempin,temp
      parameter (nprobes=6,NMAX=3, ncyc = 3000,
     2    ncool = 10, q = 2.7, fac = .95, scale=1.0/(NMAX**0.5),
     3    tempin = 1.0,frac=(1.0/3.0),nmove = nprobes*frac)
      integer indx(nprobes),piv(nmove),iconv
      real probes(nprobes,NMAX),energy(nprobes)

      common /conv/ iconv
      common /minim/ besten

      iconv = 0
      temp = 1.0

      do i = 1, ncyc
c         print *,'i=',i
        call indexx(nprobes,energy,indx)

        if (abs((besten-energy(indx(1)))/besten) .le. 0.03) then
          iconv = 1
          print *,'success'
          goto 10
        endif

        if (mod(i,ncool).eq.0) then
c           print *,'cycle',i,'temp =',temp,'en=',energy(indx(1))
          temp = cool(tempin,q,fac,(i/ncool)+1,temp)
        endif
        call pickpivle(probes, piv, energy,indx)
        call newproble(probes, piv, energy, q, temp,scale,indx)
      enddo
```

```
 10   continue

      return

      end

c*************************************************

c*************************************************
c23456789
c
c       file linear.f
c
c       The subroutine linear is the final touch to the
c       optimization.  After the pivot method has done the work of
c       finding the basin containing the global minimum. the powell
c       function is used to move each probe separately in a downhill
c       fashion to find the bottom.
c
      subroutine linear(probes.xi.n.np.ftol.its.energy)

      integer nprobes.NMAX.i.j
      parameter (nprobes=6.NMAX=3)
      real probes(nprobes.NMAX).xi(NMAX.NMAX).energy(nprobes).p(NMAX).
     2     fret
      integer its(nprobes).iter

      do i = 1. nprobes
        do j = 1,NMAX
          p(j) = probes(i.j)
        enddo
        call powell(p.xi.n.np.ftol.iter.fret)
        its(i) = iter
        energy(i) = fret
      enddo

      return

      end

c*************************************************

c*************************************************
```

```
c23456789
c
c       file main.f
c
c       This is the main portion of the program "run".  It calls
c       all needed initializations, and then proceeds to call
c       either the lowest energy pivot method or the nearest neighbor
c       pivot method.  It then uses a linear descent at the end to
c       finish matters, and writes to a file the best probe.  For
c       averaging purposes, one can change nrun to make this process
c       repeat many times to ensure that a "good" run was not an
c       anomaly.
c
      integer iter,n,np,NMAX,idum,ndiv
      real rsize,dx,shift,rk,rmass,dm
      parameter (NMAX = 3, nprobes = 6)
      real fret,ftol,p(NMAX),xi(NMAX,NMAX),rcount,
     2    probes(nprobes,NMAX),energy(nprobes),rsum
      integer its(nprobes),indx(nprobes),k,nrun,icount,ncnt,iconv

      common /rnd1/ idum
      common /count/ icount
      common /conv/ iconv

c       ftol = sqrt(1.0/(real(NMAX)))
      ftol = .01
      n = NMAX
      np = NMAX
      idum = -1
      nrun = 100
      rsum = 0.0
      ncnt = 0
      rcount = 0.0
      nconv = 0
      do i = 1,nprobes
        its(i) = 0
      enddo

      do k = 1,nrun

        icount = 0

      call initxi(xi)
      call inith3()
      call initbounds()

      call initset(probes,energy)

c      call linear(probes,xi,n,np,ftol,its,energy)

c      call nnpivot(probes,energy)

      call lepivot(probes,energy)

c      call linear(probes,xi,n,np,ftol,its,energy)


      call indexx(nprobes,energy,indx)
      print *,'fret =',energy(indx(1)),'iter =',its(1),'run: ',k

c      do i = 1,nprobes
c        print *,i,energy(indx(i))
c      enddo
c      print *,'got here, k=',k
c      open(10,file='p.dat',status='unknown')
c      do i = 1,NMAX
c        write (10,*)i,probes(indx(1),i)
c      enddo
c      print *,'wrote best probe'
c      close(10)

      if (iconv .eq. 1) then
        nconv = nconv + 1
        ncnt = ncnt + icount
      endif

      rsum = rsum + energy(indx(1))
      enddo
      rsum = rsum/float(nrun)
      rcount = float(ncnt)/float(nconv)
      print *,'avg energy = ',rsum
      print *,nconv,'runs converged, average of',rcount,'calls/good run'

      end
```

```
c*****************************************************

#*****************************************************
#
#       file makefile
#
#       This file describes how to make the executables.  It assumes the
#       Fortran compiler is called f77, that -O2 is the flag for optimizing
#       code, and that all files exist in a directory called Test (merely the
#       name of the directory used in developing the code.)
#       It is recommended that GNU make be used.  Minor problems
#       have been found with other make utilities.

F77 = f77
OPT = -O2
OBJ = brent.o \
        cool.o \
        dist.o \
        fldim.o \
        func.o \
        gammln.o \
        gasdev.o \
        indexx.o \
        initbounds.o \
        inith3.o \
        initprobe.o \
        initset.o \
        initxi.o \
        lepivot.o \
        linear.o \
        linmin.o \
        mnbrak.o \
        newproble.o \
        newprobnn.o \
        nnpivot.o \
        pickpivle.o \
        pickpivnn.o \
        powell.o \
        qdist.o \
        ran1.o

all:    run
```

```
run:    main.o ${OBJ}
        ${F77} ${OPT} -o run main.o ${OBJ}

brent.o:        brent.f
        ${F77} ${OPT} -c brent.f

cool.o: cool.f
        ${F77} ${OPT} -c cool.f

dist.o: dist.f
        ${F77} ${OPT} -c dist.f

fldim.o:        fldim.f
        ${F77} ${OPT} -c fldim.f

func.o: func.f
        ${F77} ${OPT} -c func.f

gammln.o:       gammln.f
        ${F77} ${OPT} -c gammln.f

gasdev.o:       gasdev.f
        ${F77} ${OPT} -c gasdev.f

indexx.o:       indexx.f
        ${F77} ${OPT} -c indexx.f

initbounds.o:   initbounds.f
        ${F77} ${OPT} -c initbounds.f

inith3.o:       inith3.f
        ${F77} ${OPT} -c inith3.f

initprobe.o:    initprobe.f
        ${F77} ${OPT} -c initprobe.f

initset.o:      initset.f
        ${F77} ${OPT} -c initset.f

initxi.o:       initxi.f
        ${F77} ${OPT} -c initxi.f

lepivot.o:      lepivot.f
```

```
        ${F77} ${OPT} -c lepivot.f

linear.o:      linear.f
        ${F77} ${OPT} -c linear.f

linmin.o:      linmin.f
        ${F77} ${OPT} -c linmin.f

main.o:        main.f
        ${F77} ${OPT} -c main.f

mnbrak.o:      mnbrak.f
        ${F77} ${OPT} -c mnbrak.f

newproble.o:   newproble.f
        ${F77} ${OPT} -c newproble.f

newprobnn.o:   newprobnn.f
        ${F77} ${OPT} -c newprobnn.f

nnpivot.o:     nnpivot.f
        ${F77} ${OPT} -c nnpivot.f

pickpivle.o:   pickpivle.f
        ${F77} ${OPT} -c pickpivle.f

pickpivnn.o:   pickpivnn.f
        ${F77} ${OPT} -c pickpivnn.f

powell.o:      powell.f
        ${F77} ${OPT} -c powell.f

qdist.o:qdist.f
        ${F77} ${OPT} -c qdist.f

ran1.o: ran1.f
        ${F77} ${OPT} -c ran1.f

clean:
        rm run ; rm *.o ; rm *~ ; rm core ; rm *.dat

dist:
        cd .. ; tar cvf Test.tar Test ; gzip -9 Test.tar
```

```
#*****************************************************

c*****************************************************
c23456789
c
c       file newproble.f
c
c       The subroutine newproble is the section of the lowest
c       energy pivot method that selects new probes.
c
      subroutine newproble(psi,piv, energy, q, temp, scale,indx)

      parameter (NMAX=3,nprobes=6,frac=(1.0/3.0),nmove=nprobes*frac,
     2    nkeep=nprobes-nmove)
      integer piv(nmove),idum,i,j,indx(nprobes)
      real psi(nprobes,NMAX),energy(nprobes),q,temp,scale,
     2    p(NMAX),rwhat,entemp,rtp,xmin(NMAX),xmax(NMAX)

      common /rnd1/ idum
      common /bounds/ xmin,xmax

      rwhat = 0.0
      do i = 1, nmove
        rwhat = ran1(idum)
        do j = 1, NMAX
          p(j) = psi(piv(i),j)
        enddo

        do j = 1,NMAX
          scale = xmax(j) - xmin(j)
10        continue
          rtp = p(j) +
     2        (scale * qdist(q,temp,idum))
c    2        (scale * temp * gasdev(idum))
          if ((rtp .lt. xmin(j)) .or. (rtp .gt. xmax(j))) then
            goto 10
          else
            p(j) = rtp
          endif
        enddo

        entemp = func(p)
```

```
c       if (entemp.lt.energy(indx(nkeep+i))) then
         energy(indx(nkeep+i)) = entemp
         do j = 1,NMAX
           psi(indx(nkeep+i),j) = p(j)
         enddo
c       endif

      enddo

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c       file newprobnn.f
c
c       The subroutine newprobnn is the section of the nearest
c       neighbor pivot method that selects new probes.
c
      subroutine newprobnn(psi,piv, energy, q, temp, scale)

      parameter (NMAX=3,nprobes=6)
      integer piv(nprobes/2,2),idum,i,j
      real psi(nprobes,NMAX),energy(nprobes),q,temp,scale,
     2    p(NMAX),rwhat,entemp,rtp,xmin(NMAX),xmax(NMAX)

      common /rnd1/ idum
      common /bounds/ xmin,xmax

      rwhat = 0.0
      do i = 1, nprobes/2
        rwhat = ran1(idum)
        do j = 1, NMAX
          p(j) = psi(piv(i,1),j)
        enddo

        do j = 1,NMAX
          scale = xmax(j) - xmin(j)
```

```
 10        continue
          rtp = p(j) +
     2        (scale * qdist(q,temp,idum))
c    2        (scale * temp * gasdev(idum))
          if ((rtp .lt. xmin(j)) .or. (rtp .gt. xmax(j))) then
            goto 10
          else
            p(j) = rtp
          endif
        enddo

        entemp = func(p)

        if (entemp.lt.energy(piv(i,2))) then
          energy(piv(i,2)) = entemp
          do j = 1, NMAX
            psi(piv(i,2),j) = p(j)
          enddo
        endif

      enddo

      return

      end

c****************************************************

c****************************************************
c23456789
c
c       file nnpivot.f
c
c       The subroutine nnpivot is the heart of the nearest neighbor
c       pivot method.  It runs for a number of cycles equal to the
c       parameter ncyc, and cools every ncool cycles.  During each
c       cycle it pairs every probe with its nearest neighbor, and the
c       lower of the two is chosen to be the pivot.  Details of this
c       can be found in pickpivotnn.f.  Once pivots are chosen, each
c       member of the pair with the higher energy is relocated.
c       Energies are then computed for each new probe.
c
      subroutine nnpivot(probes,energy)
      integer nprobes,NMAX,i,ncyc,ncool
      real q,fac,scale,tempin,temp
      parameter (nprobes=6,NMAX=3, ncyc = 3000,
     2   ncool = 10, q = 2.7, fac = .95, scale=1.0/(NMAX**0.5),
     3   tempin = 1.0)
      integer indx(nprobes),piv(nprobes/2,2),iconv
      real probes(nprobes,NMAX),energy(nprobes),besten

      common /conv/ iconv
      common /minim/ besten

      iconv = 0
      temp = 1.0
      temp = 1.0

      do i = 1, ncyc
c        print *,'i=',i
        call indexx(nprobes,energy,indx)

        if (abs((besten-energy(indx(1)))/besten) .le. 0.03) then
          iconv = 1
          print *,'success'
          goto 10
        endif

        if (mod(i,ncool).eq.0) then
          call indexx(nprobes,energy,indx)
c           print *,'cycle',i,'temp =',temp,'en=',energy(indx(1))
          temp = cool(tempin,q,fac,(i/ncool)+1,temp)
        endif
        call pickpivnn(probes, piv, energy)
        call newprobnn(probes, piv, energy, q, temp,scale)
      enddo

 10   continue

      return

      end

c*************************************************
```

```
c************************************************************
c23456789
c
c       file pickpivle.f
c
c       The subroutine pickpivle picks a set of pivots for
c       the probes to be moved.  This implementation uses
c       a normalized Boltzmann probability for this choice.
c       Thus, the best probes have the highest chance to be
c       picked, and probes that are not very good still have
c       a non-zero chance to be chosen.  There are clearly
c       many ways to alter this, such as including a temperature
c       in this (here temperature is implicitly equal to 1.0
c       at all times), or even using a wholly different
c       distribution function.
c
      subroutine pickpivle(psi, piv, energy,indx)

      parameter (NMAX=3,nprobes=6,frac=(1.0/3.0),nmove=nprobes*frac,
     2    nkeep=nprobes-nmove)
      integer piv(nmove),mark,i,j,pnt,count,indx(nprobes),idum
      real psi(nprobes,NMAX),energy(nprobes),temp, lo,rnorm, temp2

      common /rnd1/ idum

      temp = 0.0
      lo = 0.0
      rnorm = 0.0
      count = 1

      do i = 1, nmove
        piv(i) = 0
      enddo

      do i = 1, nkeep
        rnorm = rnorm + exp(-1.0*energy(indx(i)))
      enddo

      do i = 1, nmove
        temp = ran1(idum)
        temp2 = 0.0
        mark = 0
        do j = 1, nkeep
```

```
          temp2 = temp2 + (exp(-1.0*energy(indx(j)))/rnorm)
          if (temp.le.temp2) then
            mark = j
            goto 10
          endif
        enddo
 10     continue
        piv(i) = indx(mark)
      enddo

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c       file pickpivnn.f
c
c       The subroutine pickpivnn is used by the nearest neighbor
c       pivot method.  It pairs off all probe with their nearest
c       neighbor. starting with the first probe and removing it
c       and its nearest neighbor from further consideration. and
c       repeats until all probes are paired.  It then chooses the
c       probe with lower energy in each pair to be the pivot.  The
c       array piv is used to keep track of the pivots.
c
      subroutine pickpivnn(psi. piv. energy)

      parameter (NMAX=3,nprobes=6)
      integer piv(nprobes/2,2),mark(nprobes),i,j,pnt,count
      real psi(nprobes,NMAX),energy(nprobes),temp. lo

      temp = 0.0
      lo = 0.0
      count = 1

      do i = 1. nprobes/2
        piv(i,1) = 0
        piv(i,2) = 0
      enddo
      do i = 1, nprobes
        mark(i) = 0
      enddo

      do i = 1, nprobes -1
c         count = count + 1
        if (mark(i).ne.0) goto 10
        lo = 10000.0
        pnt = 0
        do j = i + 1, nprobes
          if (mark(j).ne.0) goto 20
          temp = dist(psi,i,j)
          if (temp.lt.lo) then
            lo = temp
            pnt = j
          endif
 20        continue
        enddo
        mark(i) = 1
        mark(pnt) = 1
        if (energy(i).lt.energy(pnt)) then
          piv(count,1) = i
          piv(count,2) = pnt
        else
          piv(count,1) = pnt
          piv(count,2) = i
        endif
        count = count + 1
 10     continue
      enddo

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c        file qdist.f
c
```

```
c       The function qdist returns a pseudorandom number
c       biased according to the Tsallis distribution, given
c       the value for q, the current "temperature", and the
c       random number seed.  This uses gammln and gasdev from
c       Numerical Recipes.  This code is nearly verbatim from
c       Tsallis and Stariolo's paper on Generalized Simulated
c       Annealing, but is included here as fair use under copyright
c       law.  I do not claim creation of this code, nor do I
c       warrant that it is perfect.  It has been tested and does
c       work, however.
c
      real function qdist(q,temp,idum)

      integer idum
      real q,temp,pi,fac1,fac2,fac3,fac4,fac5,fac6,sigmax,x,y,den

      pi = 3.14159265
      fac1 = exp(log(temp)/(q-1.0))
      fac2 = exp((4.0-q)*log(q-1.0))
      fac3 = exp((2.0-q)*log(2.0)/(q-1.0))
      fac4 = (sqrt(pi)*fac1*fac2)/(fac3*(3.0-q))
      fac5 = (1.0/(q-1.0))-0.5
      fac6 = pi*(1.0-fac5)/sin(pi*(1.0-fac5))/exp(gammln(2.0-fac5))
      sigmax = exp(-(q-1.0)*log(fac6/fac4)/(3.0-q))
      x = sigmax*gasdev(idum)
      y = gasdev(idum)
      den = exp((q-1.0)*log(abs(y))/(3.0-q))
      qdist = x/den

      return

      end

c******************************************************
```

Appendix C: Source Code for the Quantum Harmonic Oscillator

Source code for the various files necessary to implement the solution of the Schroedinger equation for the harmonic oscillator follow. Copyright law does not permit reprinting of those functions from Numerical Recipes in Fortran, and any required functions not found here may be found in Numerical Recipes in Fortran or in Appendix A. Small modifications to the code found there may be required, specifically NMAX in several locations must be increased to the value found in main.f, and one must insert a call to the subroutine "norm" immediately prior to any call to the function "func".

To create the files needed to run this program, one should just use the "make" command. This will compile all of the Fortran files into object code and link that into executables. Two executables will be created, "run" and "test". "Run" is the actual program used in this thesis, and "test" was created to ensure that all initializations were being performed properly. Both will output files with the extension "dat" upon completion, and these files may be imported easily into a graphing utility such as "xmgr".

```
c***************************************************
c23456789
c       file func.f
c
c       The function func computes the energy of a given
c       probe.  It uses arrays for position and momentum
c       operators to do this, and a function called self
c       for any self-interaction energies.  Details on the
```

```
c       op function can be found in the file op.f.
c
      real function func(p)

      integer NMAX,i
      parameter (NMAX = 64)
      real p(NMAX),t1(NMAX),pos(NMAX),rmom(NMAX)

      common /pot/ pos,rmom

      func = 0.0

      do i = 1, NMAX
        t1(i) = p(i)
      enddo

      call four1(t1,NMAX/2,-1)
      call norm(p)
      call norm(t1)

      func = func + op(p,pos)
c       print *,func
      func = func + op(t1,rmom)
c       print *,func
      func = func + self(p)
c       print *,func

        if (func .eq. 0.0) then
          print *,'ok'
          open(35,file='hmm.dat',status='unknown')
          do j = 1,NMAX
            write(35,*)j,p(j),t1(j),pos(j),rmom(j)
          enddo
          print *,'stopped, energy is',func
          close(35)
          stop
        endif

      return

      end

c**************************************************
```

```
c*****************************************************
c23456789
c       file initpos.f
c
c       This subroutine takes the initial information on the
c       starting location (shift), the size of discretization of
c       space (dx), and the spring constant (rk), and initializes
c       the position operator pos.  Pos is an array used to
c       compute the potential energy of the system.
c
      subroutine initpos(dx, shift, rk)

      integer NMAX, i
      parameter (NMAX=64)
      real pos(NMAX), dx, shift, rk, temp, p1,rmom(NMAX)

      common /pot/ pos,rmom

      p1 = shift

      do i = 1, NMAX - 1, 2
        temp = p1 * p1
        pos(i) = (rk / 2.0) * temp
        pos(i+1) = pos(i)
        p1 = p1 + dx
      enddo

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c       file initprobe.f
c
c       The subroutine initprobe generates one probe according
c       to a desired scheme.  Three are given here.  The first
c       generates a normalized Gaussian probe very nearly the actual
c       solution of the system, for use in testing the energy
```

```
c       calculation.  The second generates a probe with starting
c       values ranging from -1 to 1, which is then normalized.
c       and the third method creates a probe with values from 0
c       to 1, which is also normalized.
c
      subroutine initprobe(p)

      integer NMAX,i,j
      parameter (NMAX=64,rsz=5.0)
      real p(NMAX)

      common /rnd1/ idum

      do i = 1,NMAX
c        p(i) = exp(-0.5*(real(((NMAX/2)-i)*rsz)/real(NMAX))**2)
        p(i) = (2.0 * ran1(idum)) - 1.0
c        p(i) = ran1(idum)
      enddo

      call norm(p)

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c       file initrmom.f
c
c       The subroutine initrmom initializes the momentum operator
c       rmom based on the discretization of momentum space (dm) and
c       the mass of the particle (rmass).  Rmom is an array used by
c       the function op.  More details can be found in the file op.f.
c
      subroutine initrmom(dm, rmass)

      integer NMAX, i
      parameter (NMAX=64)
      real rmom(NMAX), dm, rmass, temp, rm1, twopi,pos(NMAX)
      common /pot/ pos,rmom

      twopi = acos(-1.0) * 2.0

      rm1= dm

      rmom(1) = 0.0
      rmom(2) = 0.0

      do i = 3, (NMAX/2)-1,2
        temp = rm1* rm1
        rmom(i) = temp * ((twopi * twopi) / (2.0 * rmass))
        rmom(i+1) = rmom(i)
        rmom((NMAX-i)+2) = rmom(i)
        rmom((NMAX-i)+3) = rmom(i)
        rm1= rm1+ dm
      enddo

      temp = rm1 * rm1
      rmom((NMAX/2)+1) = temp * ((twopi * twopi) / (2.0 * rmass))
      rmom((NMAX/2)+2) = rmom((NMAX/2)+1)

      return

      end

c****************************************************

c****************************************************
c23456789
c
c      file lepivot.f
c
c      The subroutine lepivot is the heart of the lowest energy
c      pivot method.  It takes an initialized set of probes along
c      with their energies and tries to find the minimum energy
c      possible.  It only runs for a number of cycles equal to
c      the parameter ncyc, and does not use any other stopping
c      criteria.  One could add more if so desired.
c      First it sorts the probes by their energies, chooses
c      pivots based on those energies, and then relocates the
c      worst probes around the pivots selected.  Every ncool
c      cycles it cools the system by calling the function cool.
```

```
c       Nfrac is the fraction of the total number of probes to move.
c
      subroutine lepivot(probes,energy)

      integer nprobes,NMAX,i,ncyc,ncool
      real q,fac,scale,tempin,temp
      parameter (nprobes=100,NMAX=64, ncyc = 3000,
     2    ncool = 25, q = 2.5, fac = .95, scale=1.0/(NMAX**0.5),
     3    tempin = 1.0,frac=(1.0/3.0),nmove = nprobes*frac)
      integer indx(nprobes),piv(nmove)
      real probes(nprobes,NMAX),energy(nprobes)

      temp = 1.0

      do i = 1, ncyc
c        print *,'i=',i
        call indexx(nprobes,energy,indx)
        if (mod(i,ncool).eq.0) then
c          print *,'cycle',i,'temp =',temp,'en=',energy(indx(1))
          temp = cool(tempin,q,fac,(i/ncool)+1,temp)
        endif
c        print *,'got here 1'
        call pickpivle(probes, piv, energy,indx)
c        print *,'got here 2'
        call newproble(probes, piv, energy, q, temp,scale,indx)
c        print *,'got here 3'
      enddo

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c       file main.f
c
c       This is the main portion of the program "run".  It calls
c       all needed initializations, and then proceeds to call
c       either the lowest energy pivot method or the nearest neighbor
c       pivot method.  It then uses a linear descent at the end to
```

```
c        finish matters, and writes to a file the best probe.  For
c        averaging purposes, one can change nrun to make this process
c        repeat many times to ensure that a "good" run was not an
c        anomaly.
c        Credit must be given to Stephen Belair for coming up with the
c        idea to initialize the position and momentum operators only
c        once as a time saving mechanism.  Prior to this suggestion, the
c        values were computed on demand, which can be computationally
c        expensive.
c        Parameters used include NMAX - the number of values in one
c        probe, ndiv - the number of divisions of space (half NMAX, as
c        wave functions are allowed to have both real and imaginary
c        parts), rsize - the width of the space, dx - the
c        width of the smallest unit of space, shift - the start of
c        space, rk - the spring constant for the harmonic oscillator,
c        rmass - the mass of the object oscillating, dm - the size of
c        discretization of momentum space, and nprobes - the number of
c        probes searching space.
c
      integer iter,n,np,NMAX,idum,ndiv
      real rsize,dx,shift,rk,rmass,dm
      parameter (NMAX = 64,ndiv = NMAX/2,rsize = 5.0, dx = rsize/ndiv,
     2  shift=(-0.5*rsize),rk = 1.0, rmass = 1.0, dm = 1.0/rsize,
     3  nprobes = 100)
      real fret,ftol,p(NMAX),xi(NMAX,NMAX),pos(NMAX),rmom(NMAX),
     2  probes(nprobes,NMAX),energy(nprobes),rsum
      integer its(nprobes),indx(nprobes),k,nrun

c     common /pot/ pos,rmom
      common /rnd1/ idum

c     ftol = sqrt(1.0/(real(NMAX)))
      ftol = .01
      n = NMAX
      np = NMAX
      idum = -1
      nrun = 100
      rsum = 0.0

      do k = 1,nrun

      call initxi(xi)
      call initpos(dx,shift,rk)
      call initrmom(dm,rmass)

      call initset(probes,energy)

c      call linear(probes,xi,n,np,ftol,its,energy)

      call nnpivot(probes,energy)

c      call lepivot(probes,energy)

      call linear(probes,xi,n,np,ftol,its,energy)


      call indexx(nprobes,energy,indx)
      print *,'fret =',energy(indx(1)),'iter =',its(1),'run ',k

c      do i = 1,nprobes
c        print *,i,energy(indx(i))
c      enddo

      open(10,file='p.dat',status='unknown')
      do i = 1,NMAX -1, 2
        write (10,*)(i+1)/2,((probes(indx(1),i)*probes(indx(1),i)) +
     2      (probes(indx(1),i+1)*probes(indx(1),i+1)))
      enddo

      close(10)

      rsum = rsum + energy(indx(1))
      enddo
      rsum = rsum/float(nrun)
      print *,'avg energy = ',rsum

      end

c*****************************************************

#*****************************************************
#
#     file makefile
#
#     This file describes how to make the executables.  It assumes the
#       Fortran compiler is called f77. that -O2 is the flag for optimizing
#       code. and that all files exist in a directory called New3 (merely the
#       name of the directory used in developing the code.)
#       It is recommended that GNU make be used.  Minor problems
#       have been found with other make utilities.

F77 = f77
OPT = -O2
OBJ = brent.o \
        cool.o \
        dist.o \
        fldim.o \
        four1.o \
        func.o \
        gammln.o \
        gasdev.o \
        indexx.o \
        initpos.o \
        initprobe.o \
        initrmom.o \
        initset.o \
        initxi.o \
        lepivot.o \
        linear.o \
        linmin.o \
        mnbrak.o \
        newproble.o \
        newprobnn.o \
        nnpivot.o \
        norm.o \
        op.o \
        pickpivle.o \
        pickpivnn.o \
        powell.o \
        qdist.o \
        ran1.o \
        self.o

all:    run test

run:    main.o ${OBJ}
        ${F77} ${OPT} -o run main.o ${OBJ}
```

```
brent.o:        brent.f
        ${F77} ${OPT} -c brent.f

cool.o: cool.f
        ${F77} ${OPT} -c cool.f

dist.o: dist.f
        ${F77} ${OPT} -c dist.f

fldim.o:        fldim.f
        ${F77} ${OPT} -c fldim.f

four1.o:        four1.f
        ${F77} ${OPT} -c four1.f

func.o: func.f
        ${F77} ${OPT} -c func.f

gammln.o:       gammln.f
        ${F77} ${OPT} -c gammln.f

gasdev.o:       gasdev.f
        ${F77} ${OPT} -c gasdev.f

indexx.o:       indexx.f
        ${F77} ${OPT} -c indexx.f

initpos.o:      initpos.f
        ${F77} ${OPT} -c initpos.f

initprobe.o:    initprobe.f
        ${F77} ${OPT} -c initprobe.f

initrmom.o:     initrmom.f
        ${F77} ${OPT} -c initrmom.f

initset.o:      initset.f
        ${F77} ${OPT} -c initset.f

initxi.o:       initxi.f
        ${F77} ${OPT} -c initxi.f

lepivot.o:      lepivot.f
```

```
        ${F77} ${OPT} -c lepivot.f

linear.o:       linear.f
        ${F77} ${OPT} -c linear.f

linmin.o:       linmin.f
        ${F77} ${OPT} -c linmin.f

main.o:         main.f
        ${F77} ${OPT} -c main.f

mnbrak.o:       mnbrak.f
        ${F77} ${OPT} -c mnbrak.f

newproble.o:  newproble.f
        ${F77} ${OPT} -c newproble.f

newprobnn.o:  newprobnn.f
        ${F77} ${OPT} -c newprobnn.f

nnpivot.o:      nnpivot.f
        ${F77} ${OPT} -c nnpivot.f

norm.o:         norm.f
        ${F77} ${OPT} -c norm.f

op.o:   op.f
        ${F77} ${OPT} -c op.f

pickpivle.o:    pickpivle.f
        ${F77} ${OPT} -c pickpivle.f

pickpivnn.o:    pickpivnn.f
        ${F77} ${OPT} -c pickpivnn.f

powell.o:       powell.f
        ${F77} ${OPT} -c powell.f

qdist.o:qdist.f
        ${F77} ${OPT} -c qdist.f

ran1.o: ran1.f
        ${F77} ${OPT} -c ran1.f
```

```
self.o: self.f
        ${F77} ${OPT} -c self.f

test:   test.o ${OBJ}
        ${F77} ${OPT} -o test test.o ${OBJ}

test.o: test.f
        ${F77} ${OPT} -c test.f

clean:
        rm run ; rm *.o ; rm *~ ; rm core ; rm *.dat ; rm test

dist:
        cd .. ; tar cvf New3.tar New3 ; gzip -9 New3.tar

#****************************************************

c****************************************************
c23456789
c
c       file newproble.f
c
c       The subroutine newproble is the section of the lowest
c       energy pivot method that selects new probes.  It cycles
c       over the number of probes to be moved, and half the time
c       it makes changes in real space, half the time it makes changes
c       in momentum space.  The latter is done by first taking the
c       inverse Fourier transform of the pivot probe, varying that,
c       and then returning the new probe to real space by a Fourier
c       transform.  After the new probe is constructed, it is
c       normalized and its energy is computed.
c
      subroutine newproble(psi,piv, energy, q, temp, scale,indx)

      parameter (NMAX=64,nprobes=100,frac=(1.0/3.0),nmove=nprobes*frac,
     2    nkeep=nprobes-nmove)
      integer piv(nmove),idum,i,j,indx(nprobes)
      real psi(nprobes,NMAX),energy(nprobes),q,temp,scale,
     2    p(NMAX),rwhat,entemp

      common /rnd1/ idum
```

```
        rwhat = 0.0
        do i = 1, nmove
          rwhat = ran1(idum)
          do j = 1, NMAX
            p(j) = psi(piv(i),j)
          enddo

          if (rwhat.le.(0.5)) then
            call four1(p,NMAX/2,-1)
          endif

          do j = 1,NMAX
            p(j) = p(j) +

c           At present we are using a Gaussian distribution.  The
c           Tsallis distribution may be used in its place by uncommenting
c           the next line of code an commenting out the one immediately
c           after it.

c     2            (scale * qdist(q,temp,idum))
      2            (scale * temp * gasdev(idum))
          enddo

          if (rwhat.le.(0.5)) then
            call four1(p,NMAX/2,1)
          endif

          call norm(p)

          entemp = func(p)

c          if (entemp.lt.energy(indx(nkeep+i))) then
            energy(indx(nkeep+i)) = entemp
            do j = 1,NMAX
              psi(indx(nkeep+i),j) = p(j)
            enddo
c          endif

        enddo

        return

        end
```

```
c****************************************************
c****************************************************
c23456789
c
c       file newprobnn.f
c
c       The subroutine newprobnn is the section of the nearest
c       neighbor pivot method that selects new probes.  It cycles
c       over the number of probes to be moved, and half the time
c       it makes changes in real space, half the time it makes changes
c       in momentum space.  The latter is done by first taking the
c       inverse Fourier transform of the pivot probe, varying that,
c       and then returning the new probe to real space by a Fourier
c       transform.  After the new probe is constructed, it is
c       normalized and its energy is computed.
c
      subroutine newprobnn(psi,piv, energy, q, temp, scale)

      parameter (NMAX=64,nprobes=100)
      integer piv(nprobes/2,2),idum,i,j
      real psi(nprobes,NMAX),energy(nprobes),q,temp,scale,
     2    p(NMAX),rwhat,entemp

      common /rnd1/ idum

      rwhat = 0.0
      do i = 1, nprobes/2
        rwhat = ran1(idum)
        do j = 1, NMAX
          p(j) = psi(piv(i,1),j)
        enddo

        if (rwhat.le.(0.5)) then
          call four1(p,NMAX/2,-1)
        endif

        do j = 1,NMAX
          p(j) = p(j) +

c       At present the Tsallis distribution is being used.  The
c       Gaussian distribution may be used by commenting out the
```

```
c       next line of code and uncommenting the one immediately
c       following it.

   2          (scale * qdist(q,temp,idum))
c   2         (scale * temp * gasdev(idum))
      enddo

      if (rwhat.le.(0.5)) then
        call four1(p,NMAX/2,1)
      endif

      call norm(p)

      entemp = func(p)

      if (entemp.lt.energy(piv(i,2))) then
        energy(piv(i,2)) = entemp
        do j = 1, NMAX
          psi(piv(i,2),j) = p(j)
        enddo
      endif

    enddo

    return

    end

c*****************************************************
c*****************************************************
c23456789
c
c       file nnpivot.f
c
c       The subroutine nnpivot is the heart of the nearest neighbor
c       pivot method.  It runs for a number of cycles equal to the
c       parameter ncyc, and cools every ncool cycles.  During each
c       cycle it pairs every probe with its nearest neighbor, and the
c       lower of the two is chosen to be the pivot.  Details of this
c       can be found in pickpivotnn.f.  Once pivots are chosen, each
c       member of the pair with the higher energy is relocated.
c       Energies are then computed for each new probe.
```

```
c
      subroutine nnpivot(probes,energy)

      integer nprobes,NMAX,i,ncyc,ncool
      real q,fac,scale,tempin,temp
      parameter (nprobes=100,NMAX=64, ncyc = 3000,
     2    ncool = 25, q = 2.5, fac = .95, scale=1.0/(NMAX**0.5),
     3    tempin = 1.0)
      integer indx(nprobes),piv(nprobes/2,2)
      real probes(nprobes,NMAX),energy(nprobes)

      temp = 1.0

      do i = 1, ncyc

        if (mod(i,ncool).eq.0) then
          call indexx(nprobes,energy,indx)
c           print *,'cycle',i,'temp =',temp,'en=',energy(indx(1))
          temp = cool(tempin,q,fac,(i/ncool)+1,temp)
        endif

        call pickpivnn(probes, piv, energy)

        call newprobnn(probes, piv, energy, q, temp,scale)

      enddo

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c       file norm.f
c
c       The subroutine is an extremely simple procedure for
c       normalizing a probe by summing the squares of every
c       member of the probe, taking the square root, and then
c       replacing every member by its original value divided by
c       the square root of the sum of the squares.  Pythagoras
```

```
c       is still useful today.
c
      subroutine norm(p)

      integer NMAX,i
      parameter (NMAX = 64)
      real p(NMAX),temp

      temp = 0.0

      do i = 1, NMAX
        temp = temp + (p(i) * p(i))
      enddo

      temp = sqrt(temp)

      do i = 1, NMAX
        p(i) = p(i) / temp
      enddo

      return

      end

c***************************************************

c***************************************************
c23456789
c
c       file op.f
c
c       The function op returns the result of operating on a
c       wavefunction by an arbitrary operator.  We are actually
c       interested in the result of <p|o|p>, so we are doing this
c       by multiplying the value of the operator at each point in
c       space by the square of the corresponding value of the
c       wavefunction.  If the momentum operator is being used, one
c       must pass the inverse Fourier transform of the wavefunction,
c       which is the representation in momentum space.  An explicit
c       example of how this works can be found in func.f.
c
      real function op(p,oper)
      integer NMAX,i
      parameter (NMAX = 64)
      real p(NMAX),oper(NMAX)

      op = 0.0

      do i = 1, NMAX
        op = op + (oper(i) * (p(i) * p(i)))
      enddo

      return

      end

c*****************************************************

c*****************************************************
c23456789
c
c        file self.f
c
c        The function self is a dummy function that returns
c        zero.  It is included as a place marker for future
c        development.  It represents the part of an energy
c        function that is not well described by the local
c        operator method used in op.f.  An example of this
c        would be two electrons interacting.
c
      real function self(p)

      integer NMAX
      parameter (NMAX=64)
      real p(NMAX)

      self = 0.0

      return

      end

c*****************************************************

c*****************************************************
```

```
c23456789
c
c       file test.f
c
c       This is the starting point for the test program.
c       Its only purpose is to show that the potential and
c       momentum operators initialize properly. and that a
c       wavefunction with a known energy is computed to have
c       that energy.  In our case we can start with the known
c       solution to the harmonic oscillator and show that this
c       does indeed give a value of almost exactly 0.5.  A small
c       variance from 0.5 may occur. as rounding errors can
c       happen.
c
      integer iter.n.np.NMAX.idum.ndiv
      real rsize.dx.shift.rk.rmass.dm.energy
      parameter (NMAX = 64.ndiv = NMAX/2.rsize = 5.0. dx = rsize/ndiv.
     2    shift=(-0.5*rsize).rk = 1.0. rmass = 1.0. dm = 1.0/rsize)
      real fret.ftol.p(NMAX).xi(NMAX.NMAX).pos(NMAX).rmom(NMAX)

      common /pot/ pos.rmom
      common /rnd1/ idum

      ftol = 0.0000000001
      n = NMAX
      np = NMAX
      idum = -1
      print *,'got here'
      call initpos(dx.shift.rk)
      print *,'got here 2'
      call initrmom(dm.rmass)
      print *.'got here 3'

      call initprobe(p.xi)
      print *.'got here 4'
      energy = func(p)
      print *.'got here 5'
      print *.'energy=',energy
      open(10.file='pos.dat'.status='unknown')
      open(20.file='mom.dat',status='unknown')
      open(30.file='pt.dat'.status='unknown')
      do i = 1,NMAX
        write(10,*)i,pos(i)
        write(20,*)i,rmom(i)
        write(30,*)i,p(i)
      enddo
      close(10)
      close(20)
      close(30)

      end
c*****************************************************
```

VITA

VITA

Aaron Fletcher Stanton was born in Muncie Indiana on January 31, 1969. He graduated in May of 1993 with a B.A. in Physics from DePauw University in Greencastle Indiana, where he was honored with membership in Sigma Pi Sigma. In May of 1998 he completed the Applied Management Principles course in the Krannert School of Management at Purdue University. Aaron is in the process of completing his Ph.D. in Chemistry at Purdue University, having been a graduate student for the past five years.

Aaron has been a teaching assistant for Freshman Chemistry over the course of two summers and is for the current semester as well. Most of the time he has been a teaching assistant for his advisor, Professor Sabre Kais, in two graduate classes on Quantum Chemistry.

Aaron's research interests include global optimization of functions, quantum chemistry, chaos theory, molecular nanotechnology, wavelets, and computer programming. He has published three papers and given four public lectures.

Aaron is currently living in Lebanon, Indiana, and has been married for almost six years. He is in the process of seeking employment as a researcher.