Title

Genetic Algorithms and Their Use in Geophysical Problems

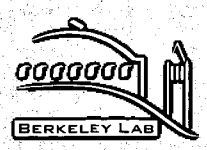
Author

Parker, Paul B

Publication Date

1999-04-01

LBNL-43148



ERNEST ORLANDO LAWRENCE
BERKELEY NATIONAL LABORATORY

Genetic Algorithms and Their Use in Geophysical Problems

Paul B. Parker

Earth Sciences Division

April 1999

Ph.D. Thesis

DISCLAIMER

LBNL-43148

Genetic Algorithms and their use in Geophysical Problems

by

Paul Bradley Parker

B.S. (California State University, Fresno), 1993

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Geophysics

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA at BERKELEY

Committee in Charge:

Professor Lane R. Johnson, Chair
Professor Thomas V. McEvilly
Professor James W. Rector

Spring 1999

This work was supported by the Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Abstract

Genetic Algorithms and their use in Geophysical Problems

by

Paul Bradley Parker

Doctor of Philosophy in Geophysics

University of California at Berkeley

Professor Lane R. Johnson, Chair

Genetic algorithms (GAs), global optimization methods that mimic Darwinian evolution are well suited to the nonlinear inverse problems of geophysics. A standard genetic algorithm selects the best or "fittest" models from a "population" and then applies operators such as crossover and mutation in order to combine the most successful characteristics of each model and produce fitter models. More sophisticated operators have been developed, but the standard GA usually provides a robust and efficient search. Although the choice of parameter settings such as crossover and mutation rate may depend largely on the type of problem being solved, numerous results show that certain parameter settings produce optimal performance for a wide range of problems and difficulties. In particular, a low (about half of the inverse of the population size) mutation rate is crucial for optimal results, but the choice of crossover method and rate do not seem to affect performance appreciably. Optimal efficiency is usually achieved with smaller (

< 50) populations. Lastly, tournament selection appears to be the best choice of selection methods due to its simplicity and its autoscaling properties. However, if a proportional selection method is used such as roulette wheel selection, fitness scaling is a necessity, and a high scaling factor (> 2.0) should be used for the best performance.

Three case studies are presented in which genetic algorithms are used to invert for crustal parameters. The first is an inversion for basement depth at Yucca mountain using gravity data, the second an inversion for velocity structure in the crust of the south island of New Zealand using receiver functions derived from teleseismic events, and the third is a similar receiver function inversion for crustal velocities beneath the Mendocino Triple Junction region of Northern California. The inversions demonstrate that genetic algorithms are effective in solving problems with reasonably large numbers of free parameters and with computationally expensive objective function calculations.

More sophisticated techniques are presented for special problems. Niching and island model algorithms are introduced as methods to find multiple, distinct solutions to the nonunique problems that are typically seen in geophysics. Finally, hybrid algorithms are investigated as a way to improve the efficiency of the standard genetic algorithm.

Table of Contents

List of Figures

List of Tables

Acknowledgments

My most sincere thanks goes to my advisor, Lane Johnson. Lane is a remarkable combination of scientific acumen, integrity, modesty, friendliness and wit that make him universally respected by all who know him. I was incredibly fortunate to step into Lane's office in 1993 while visiting the Berkeley geophysics department. I will always be indebted to him for his endless patience and guidance.

I would also like to thank Tom McEvilly, a wonderful man with a brilliant curiosity and perpetually affable demeanor, for his guidance and advice. Tom and Lane designed CCS (the Center for Computational Seismology) as a place where students, postdocs, and professors could interact freely. I am truly grateful to have been a small part of CCS for nearly six years.

Many other Berkeley professors and faculty enriched my education through courses and personal communication. Specifically, I would like to thank Walter Alvarez, Mark Bukowinski, Don DePaolo, Doug Dreger, Raymond Jeanloz, Jamie Rector, Barbara Romanowicz, Bob Urhammer and Chi Wang for their contributions.

In my first few years at CCS, I benefited from the constant tutoring of many more experienced colleagues, in particular Roland Gritto, Don Vasco, Tom Daly, Ann Kirkpatrick, Valeri Korneev, Bob Nadeau, John Peterson, Art Romero and Ken Williams. I am also thankful to have had the benefit of working with some of the best and brightest students in the field of Earth Sciences and I owe a debt of

gratitude to Dawnika Blatter, Susan Hubbard, Bruno Kaelin, Abby Kavner, Chaincy Kuo, Charles Megnin, Gordon Moore, Michael Pasyanos, Patty Seifert, Christiane Stidham and many others for all the valuable scientific knowledge that I accumulated through diffusion.

I would not have been able to get through graduate school without the help of many good people in the Geology department, including (but not limited to) Janice Elliot, Ann Goolsby, Mei Griebenow, Sharon Hurd, Eleanor Ka, and I would especially like to thank Hank Houck for being a true friend and going to bat for me many times.

I would also like to thank the physics faculty at Fresno State University, and especially John Dews, who passed away this year. John received his Ph.D. from Berkeley and his encouragement convinced me to apply here. His love of teaching physics and the kindness and respect he showed to all will be greatly missed.

I also want to thank my New Zealand colleagues, especially Helen Anderson and Donna Eberhart-Phillips, as much for their friendship and hospitality as their scientific collaboration. In addition, I would be remiss not to thank the U.S. National Science Foundation, the New Zealand Foundation for Research Science and Technology and the University of Otago for funding the SAPSE experiment, which provided a wonderful and exciting project to work on for several years and an important part of this dissertation.

Finally, I am very appreciative of those who helped review this dissertation in its final stages, sometimes more than once! Their efforts greatly improved this

work. Thanks to my dissertation committee and to Roland Gritto for his tireless help.

I dedicate this thesis to my Parents, who gave me the freedom and encouragement to succeed in life, and to my wife Anette, who gave me constant support and kept me well grounded throughout the entire time this dissertation was being written.

Chapter 1

Introduction

Most geophysical inverse problems are nonlinear and tend to have multimodal objective functions (Sen and Stoffa, 1992). Such problems cannot be reliably optimized using either conventional linear inversion techniques or quasi-Newton methods because of the propensity of these routines to go to the nearest (local) optimum point which is not necessarily the global optimum. It is apparent that the solution of such problems requires a more sophisticated approach. The exponential increase in computer speed in the last few decades has made it possible to take another approach to solving these problems, one based more on exploration than on exploitation. A good example of this approach is *simulated annealing* (Aarts and Horst, 1989; Press et al., 1992) in which the search is directed by the local gradient of the objective function in a probabilistic sense, so that there is always a possibility that the algorithm can "escape" from a local optimum in order to find the global one.

A recent *Time* magazine cover story entitled "The Killers All Around" (Lemonick, 1994) described how new antibiotic resistant bacteria are reversing human victories in the war against infectious disease. By the late 1940's antibiotics were in wide public use and scientists had discovered so many new and efficient types of drugs that many predicted an eventual end to all infectious disease. What went wrong? Too many antibiotics prescribed inappropriately and used incorrectly had eventually allowed the unintended engineering of a "super

race" of bacteria that thwart our best conscious efforts at their destruction. Now many scientists are wondering if and when a new plague may arrive. It appears that no matter how hard we try, and though dramatic battles are sometimes won, evolution slowly but surely finds a way to get the upper hand.

In the spirit of "if you can't beat 'em, join 'em", new computer based techniques for solving numerical optimization problems were developed in the 1960's by mimicking natural selection. These global optimization techniques, called *Evolutionary Algorithms*, are well suited for solving the difficult nonlinear problems of geophysics, and have been successfully applied to many types of problems (Sen and Stoffa, 1992; Stoffa and Sen, 1991; Shibutani et al., 1996).

Aside from their ability to locate the global optimum in the presence of many local optima, another major advantage to evolutionary computation is that it is easily "parallelized" to run on multiple processors at once. In one "generation" as many as several hundred objective function evaluations (a comparison between the actual data and synthetic data which was created using a model) must be made, but each is made independently of the results of any other, so parallelizing the algorithm can theoretically increase efficiency by several orders of magnitude (assuming that many processors are available). This is an important advantage, as multiprocessor computing is fast becoming the most inexpensive way to achieve maximum computing power.

In the following, a heuristic introduction to genetic algorithms is presented in Chapter 2 along with some of the essential theory and a derivation of the schema theorem. Some alternative operators are introduced at the end of the chapter. In

Chapter 3 a suite of numerical tests are performed in order to determine which factors most contribute to genetic algorithm performance. In particular optimal crossover and mutation rates and selection schemes are estimated using a set of multimodal test functions and the results are tested with standard geophysical inverse problems. Chapters 4-6 present case histories. In Chapter 4 a gravity data set is inverted with a genetic algorithm to obtain the crustal basement structure at Yucca Mountain, Nevada. In Chapters 5 and 6 deep crustal structure is investigated for the South Island of New Zealand and the Mendocino region of Northern California, respectively, by inverting receiver functions with a genetic algorithm. Finally, Chapter 7 describes specialized techniques to address such problems as loss of diversity and excessively slow convergence.

Chapter 2

Genetic algorithm background and overview

2.1 Introduction

A Genetic algorithm (GA) is a computational technique which uses principles of Darwinian natural selection in order to solve a wide variety of problems. Typically, a randomly generated initial population of binary strings (each string representing a set of model parameters) competes through an objective function evaluation for reproduction slots in the next generation. Then crossover and mutation operators are applied to the fittest strings (the strings representing the models which produce the smallest misfit between synthetic and actual data) in order to "mix" the characteristics of the best models and find an acceptable set of model parameters. The early development of the genetic algorithm is credited to John Holland (1975) and his students at the University of Michigan. The early goals of Holland's work were oriented more towards modeling natural systems rather than problem solving, and his approach has brought a new understanding of natural selection from an artificial standpoint.

Genetic Algorithms are part of a larger family of algorithms called Evolutionary Algorithms, which also include Evolutionary Programming (Fogel, 1962; Bäck, 1996; De Groot-Hedlin and Vernon, 1999) and Evolutionary Strategies. The major differences between the GA and its related techniques are:

1) GAs use a binary representation instead of real valued representation

2) GAs use crossover based recombination

3) GAs do not use self adaptation

The binary representation allows the implementation of binary operators such as simple crossover, bitwise mutation, and inversion. Because there is no self adaptation, GA's more closely follow Darwinian evolution while other evolutionary algorithms follow something akin to Lamarck's earlier evolution theory. For a complete discussion of all three methods see Bäck (1996).

Genetic algorithms can be used to solve a wide range of problems, but they excel in solving complex nonlinear problems. Three methods are commonly used to solve such problems: Quasi Newton (calculus based) methods which exploit gradient information, grid search methods which explore the solution space exhaustively, and stochastic search methods. Like simulated annealing algorithms (Aarts and Horst, 1989; Kirkpatrick, 1998), genetic algorithms fall under the classification of stochastic or "randomized" search, although this does not imply a directionless search. Genetic algorithms are highly exploitative, making use of a vast amount of parallel information in order to direct their search more efficiently. However, genetic algorithms are not "greedy" like calculus based methods, so they are capable of finding the global optimum in the presence of local optima, noise, and discontinuities. This is one of the major advantages of genetic search methods for geophysical problems. There is always a tradeoff between efficiency and exploration in solving nonlinear inverse problems, but genetic algorithms are a compromise that is acceptable for a wide range of problems. Other methods may prove more efficient for specific problems.

In the next section a model of a standard genetic algorithm will be presented with an explanation of the basic operators necessary for a robust and efficient search.

2.2 A standard genetic algorithm model

Solving problems with a standard genetic algorithm involves six basic steps:

1. Coding the problem and constructing the objective function
2. Generating an initial population of random models
3. Evaluating the fitness of each model
4. Subjecting the models to a selection process
5. Crossover and mutation
6. Overwriting the old generation

A flow chart representation of these steps is shown in Figure 2.1. The relevant terms will be defined in the following sections.

2.2.1 Coding the problem and constructing the objective function

These are the two steps that tend to be problem dependent. Coding the problem refers to constraining the model parameters and encoding them into binary strings (the use of non binary codings will be discussed in section 2.3.1). This immediately requires that the model parameters be finite, but leaves few other restrictions on their domain. The choice of the number of bits to be used in the string is largely a matter of desired accuracy. Each l bits gives 2^l possible

parameter values. As an example, assume it is necessary to search as a model space the interval [0,10.0] for the optimal value of a certain parameter x. If the desired accuracy of the solution is 99%, it would require at least 7 bits ($2^7 = 128$ possible values for x). The binary string 0000000 would then be mapped to 0.0, 1111111 to 1.0 and all the points between would be linearly interpolated. Ironically, discrete model spaces present the biggest coding problems. Suppose it is desired to code a parameter that does not have an even power of two discretizations. If a power of two is used that gives more discretizations than the model space has evaluating these models becomes a problem. Problems like these are most efficiently solved by incorporating extra constraints into the objective function.

Population sizing, while one of the most important factors in determining the convergence of a GA (Harik et al., 1996), is also one of the least understood. Naturally, a large population will produce better results per generation, but the real goal is to minimize the number of objective function evaluations necessary to obtain satisfactory results. Several equations have been derived that base the population size on the signal and noise characteristics of the problem (DeJong, 1975) and the variance in fitness (Goldberg and Rudnick, 1993), but they give population estimates that tend to be too high to be practical for most applications. Because the optimal population size depends greatly on the type of problem, some experimentation is necessary. According to Goldberg (1989), a good range to use for most complex problems is 50-200.

Construction of the objective function is typically the most subjective part of solving any inverse problem, as it is the part that usually requires a significant value judgment. Sometimes the implementation is quite straightforward, such as minimizing a misfit or maximizing a profit function, but often there are other constraints that must be considered which strongly determine the nature of the solution that will be obtained. In addition to the objective function, GA's use a fitness function, which is related to the objective function but has several properties that facilitate the performance of the algorithm.

2.2.2 Generating an initial population of random models

This is usually done in the binary representation. For each model a string of randomly generated 1's and 0's are mapped to a set of floating point parameters. This can easily be implemented by calling a random number generator that gives a real number from zero to one and using a case statement to assign a binary value based on the value obtained. After this the strings can be decoded into a floating point array in preparation for evaluation by the objective function.

2.2.3 Evaluating the fitness of each model

Genetic algorithms use a fitness function which is related to the objective function but has two key differences:

1. Whereas the objective function may produce positive or negative values, the fitness function must be a positive function which rewards smaller misfits with larger fitness values.

2. The fitness function must be scaled as the GA run progresses in order to keep selection pressure as constant as possible.

The objective function represents the optimization goal of the problem, but the fitness function is a mapping of the objective function that allows the algorithm to achieve optimal results by adjusting the amount of competition in the model population. Although the objective function may be either minimized or maximized, it is convenient to define fitness as a quantity to be maximized for genetic algorithms because most selection schemes are stochastic, selecting members to fill the next generation in proportion to their fitness. For example, if the object of the problem is to minimize an error function, a good choice for a fitness function might be:

$$f_i(x) = \varepsilon_{\max} - \varepsilon_i \tag{2.1}$$

where ε_i is the error between the observed and estimated data for each model and $\varepsilon_{\max}$ is the maximum error for the current generation. Note that this choice of fitness function gives

$$f_{\min}(x) = 0.0 \tag{2.2}$$

If a selection scheme based on ranking is used, the objective function may be used as the fitness function. More will be said about the fitness function and scaling in section 2.4.

2.2.4 Subjecting the models to a selection process

There are many different ways to select the best models, but one of the most effective (and certainly the easiest to implement on a computer) is *tournament selection.* Members are taken randomly from the population in groups of two or more at a time, and the fittest of each of these subgroups is selected to enter the next population. The larger the subgroup the greater the "selection pressure" or competitiveness but also the likelihood of premature convergence, so subgroups of 2 or 3 are usually the norm.

Although simple this method tends to give excellent results. The major advantage over other techniques is that it is based on fitness rank, while most techniques are based largely on fitness proportionality. These proportional fitness methods are all scale dependent, which means that the fitness must be scaled or "stretched" in order for the algorithm to proceed efficiently. Tournament selection is scale invariant, so fitness scaling is not necessary. Fitness scaling and several other selection methods will be described in greater detail in section 2.4.

2.2.5 Crossover and mutation

After the selection process it is desirable to combine the properties of the best individuals in various ways in order to find better models. This is accomplished through the crossover operator, which is analogous to chromosomal crossover in biological systems. First, two strings are chosen for mating and a random locus (a point between two bit positions) is found by generating a random integer between 1 and the length of the string minus 1. Second, "offspring 1" gets the bit values of

parent 1 for the bits from 1 to the crossover locus, and "offspring 2" gets the bit values of parent 2 for the bits from the crossover locus to the last bit. This is done for every set of mates or "parents" to produce a new generation. Figure 2.2 shows an example of this type of crossover (called "single point crossover") between two strings of 8 bits length. Several other types of crossover operators are commonly used, such as two point crossover and uniform crossover. These will be discussed in a later section.

The action of "crossing over" strings can lead to a subtle pitfall: it is possible to eventually have the same bit value at one or more alleles (positions) for the entire population, after which the bit value can never be changed through crossover. Thus, certain model "characteristics" can be lost forever during a program execution (these characteristics may or may not be desirable, but the idea is that information is lost and the number of possible solutions that can be obtained is limited). However, the mutation operator is a simple way to avoid this situation. This operator loops through the binary array corresponding to the new generation and changes the bit value if a randomly generated number is less than a predetermined value (usually called the mutation probability, it is typically on the order of 1/(population size)). There are two ways to implement this: either the bit is flipped automatically when the mutation operator is called, or the old value is thrown out and a new bit is randomly generated. The only difference between these two techniques is in the rate of mutation, the former being twice as frequent as the latter. The mutation operator is also analogous to mutation in natural systems. According to traditional GA theory the primary purpose of mutation is

to insure that no information is permanently lost in a run. In order to keep the algorithm working efficiently it is best used sparingly. Figure 2.3 illustrates the mechanics of mutation on a string.

2.2.6 Overwriting the old generation

In this step the old (parent) array is overwritten with the new (child) array and a generation is completed. The new population is usually checked to see if the best model from the previous generation has been reproduced and if not, it is copied into a random slot. This is called "elitism".

2.2.7 Genotype versus phenotype

It is important to differentiate between genotypic (binary based) and phenotypic (floating point based) operators. As in biology, genotype refers to the representations of the characteristics and phenotype to the physical characteristics themselves. Crossover and mutation are examples of genotypic operators and selection (as described above) is an example of a phenotypic operator. As discussed in section 2.1, GA's are the only type of evolutionary algorithm which uses a binary coding of the models parameters. This makes GAs intrinsically more complicated than the other methods, although operations such as crossover and mutation tend to be much simpler to implement in the binary representation than in the real number representation.

2.2.8 How well does this algorithm perform?

It may be asked why this type of algorithm would work more efficiently than a simple Monte Carlo routine. Do the selection process, crossover and mutation operators really improve the results of the search? A simple example illustrates how single point crossover can lead to improvement in each successive generation. Assume that the simple function

$$f(x) = x \tag{2.3}$$

is to be minimized on the interval [0,1]. If the problem is coded with 4 bits, the string {0,0,0,0} can be defined as 0.0 and {1,1,1,1} as 1.0 and the values of the strings in between can be interpolated, for example a string such as {0,1,1,0} would decode as follows:

$$0 \cdot 10^3 + 1 \cdot 10^2 + 1 \cdot 10^1 + 0 \cdot 10^0 = \frac{110}{1111} = 0.0990099 \tag{2.4}$$

If the fitness is defined as $1-(f_{obj}(x))$ (where $f_{obj}(x)$ is the objective function value) this string would evaluate fairly high. In fact, any string with a "0" in the 1,000's place would evaluate quite well, even {0,1,1,1}. It is quite easy to imagine that any reasonable selection scheme will heavily favor these strings over the ones with "1"'s in the 1,000's place. Therefore, the vast majority of strings that are selected and crossed over will have a "0" in the 1,000's place, so this characteristic will seldom be disturbed in recombination. However, because these models evaluated better than the average, many of them are also likely to have "0"s in the 100's, 10's, and 1's place, and these models are more likely to be selected into succeeding generations.

The best improvement in any stochastic search is usually seen at the beginning, and as the search continues the improvement diminishes, giving a convergence in the form of *1/n*, where *n* is the number of function evaluations. For some types of problems the performance of a genetic algorithm may be similar to that of a Monte Carlo approach early in the run, but the genetic algorithm is quickly able to estimate the solution while the Monte Carlo algorithm is simply a random walk, occasionally finding a better solution only by chance. Figure 2.4 shows the results of the search for the minimal value of the function

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = \sqrt{x_1} + \sqrt{x_2} + \sqrt{x_3} + \sqrt{x_4} + \sqrt{x_5} + \sqrt{x_6} \qquad (2.5)$$

on the interval $\{x_1, x_2, x_3, x_4, x_5, x_6\} \in [0.0, 1.0]$. Each string has 16 bits, so the search space has $(2^{16})^6 \approx 7.92 \times 10^{28}$ possible values of f. The Figure shows the results for both the standard genetic algorithm as described above and a Monte Carlo algorithm. After 147 generations the GA has reached the solution

$$(x_1, x_2, x_3, x_4, x_5, x_6) = (0.0, 0.0, 0.0, 0.0, 0.0, 0.0) \qquad (2.6)$$

while the Monte Carlo routine is still far from the solution with an objective function value of 0.97. In fact, it would take the Monte Carlo search somewhere on the order of 10^{26} generations to find the exact solution! Although this simple example is somewhat unfair to the Monte Carlo technique, it illustrates the point that the GA is capable of exploiting information in order to arrive at a solution much faster than a random search.

2.3 Genetic Algorithm theory

It is intuitive to see that genetic algorithms work by combining "traits" or "characteristics" to produce parameter sets that we have defined as being "fit". But what are these "traits"? Goldberg (1989) defines *schema* as a similarity template which describes a subset of strings with similarities at certain string positions. As an example, the strings {10101101} and {10001000} both have in common the bits in positions 1,2,4,5 and 7, so they share the similarity template ("schema") of {10*01*0*}, where the character "*" represents either a 1 or a 0 (a wildcard).

Each string of length m has 2^m schemata because each position in the string can have its own binary value or a wildcard. An upper bound on the number of schemata in a given population can be found by counting the number of schemata in one of the strings and multiplying that by the number of strings in the population (typically this is a gross overestimate of the actual number of schemata because of loss of diversity, especially late in a genetic algorithm run). Consider a population of 100 strings of length 8 bits where a maximum of $100(2^8) = 25{,}600$ schemata exist. Of course, the vast majority of schemata in a given problem are of little or no use in finding an acceptable solution, so it is more useful to know how many schemata a genetic algorithm can process and evaluate in each generation.

Following Goldberg (1989), assume a group of n binary strings each of length l. Looking only at the schemata which stay intact through recombination with a

probability p_s and therefore whose loss[1] after recombination is $p_l < 1 - p_s$ allows consideration only of schemata of length $l_s < p_l(l-1)+1$. The number of schemata of length l_s or shorter on a string of length l is then $2^{(l_s-1)} \cdot (l - l_s + 1)$. Multiplying this by the population n will give an overestimate of the total number of schemata because there are certainly duplicate schemata in the population (especially the low order schemata). A better estimate is to assume $n = 2^{l_s/2}$, so that one or less of schemata of order $l_s/2$ is expected. Counting only the higher order schemata gives a minimum number of schemata that are processed:

$$n_s \geq n(l - l_s + 1)2^{l_s-2} \tag{2.7}$$

If the population size is then restricted to $2^{l_s/2}$, the above equation becomes

$$n_s = \frac{(l - l_s + 1)}{4} n^3 = Cn^3 \tag{2.8}$$

where C is a constant. This shows that the number of schemata processed by a genetic algorithm is on the order of the population size cubed, or n^3. This result, that a genetic algorithm can process an extremely large number of schemata through a relatively small number of strings was first derived by Holland (1975), who gave it the name *implicit parallelism.* Note that this derivation does not take into account mutation, but if a low mutation rate is used the result will be the same.

---

[1] In this sense loss means that the schemata is divided in the recombination operation, disrupting the information it represented.

2.3.1 Why binary alphabets?

Every parameter coding discussed to this point has been in binary. It may be advantageous to code in another cardinality for certain problems, but there is a good reason binary codings are the standard in GA work. The binary alphabet gives the largest number of schemata per bit of information (Goldberg, 1989). This can be seen as follows: let l_2 be the length of the binary string and l_n be the length of the alphabet of cardinality n. To have the same amount of information, 2^{l_2} must be equal to n^{l_n}. As n increases, l_n decreases rapidly. According to traditional GA theory, processing large numbers of schemata is what makes a GA powerful, so it is important that the maximum number of them are available for the algorithm to process.

2.4 Other operators and techniques

2.4.1 Other selection schemes

The choice of a selection scheme has subtle implications that strongly affect the performance of the algorithm; using a scheme that is too competitive can lead to premature convergence while using one that is too forgiving is inefficient. Selection schemes fall into two general categories: *stochastic* (based on probability) and *deterministic* (based on rank). The major difference between the two categories is that the stochastic methods introduce selection noise, which although seemingly inefficient is crucial in some amount to avoid premature convergence.

It should be noted that although numerous sophisticated selection schemes have been proposed, many times the best results can be achieved by using simple tournament selection, as previously described. Although this method is based on rank it introduces noise in the initial random choice of 2 or more tournament competitors. Selection pressure in tournament selection can be increased by increasing the number of competitors in the tournament and it can be decreased by increasing the amount of noise in the selection process (see "stochastic tournament selection" below).

2.4.1.1 Roulette Wheel Selection

In Roulette Wheel Selection each model is given a slice of a "roulette wheel" that is proportional to its fitness. The roulette wheel is then spun as many times as necessary to create a subpopulation with the genetic material necessary to construct a statistically fitter new generation. To implement this scheme a random number is generated between 0.0 and the sum of all the fitness values. A loop is then used to determine which model's fitness this number corresponds to ("spinning the roulette wheel"), and this model is given a slot in the next generation. While this method is simple and intuitively appealing, there is a hidden pitfall based on scale. At the beginning of a genetic algorithm run, the variance in the population's fitness is large, and many potentially good models may be overlooked due to the strength of a few very fit models which may result in a rapid loss of diversity and early convergence. Near the end of the run the opposite occurs: Every model has nearly the same fitness with very little variance,

so the algorithm is reduced to a less efficient random walk. To avoid these difficulties some form of fitness scaling is necessary. Linear scaling (pivoting the fitness values about the average) is an effective method. The scaled fitness value is given by

$$f_s = Af + B \tag{2.9}$$

where f is the original fitness value, A the fitness slope and B is the y-intercept:

$$A = (C_s - 1.0) \cdot \left(\frac{f_{avg}}{f_{\max} - f_{avg}} \right) \tag{2.10}$$

$$B = \frac{f_{\min} - f_{avg}}{f_{\max} - f_{avg}} \tag{2.11}$$

The value of the scaling factor C_s depends on the problem but typically ranges between 1.3 (1.0 is equivalent to no scaling) and 2.5.

2.4.1.2 Rank only selection

In this scheme the truncated ratio of a model's fitness to the average fitness of the entire population determines how many offspring (if any) the model will have in the next generation. For example, if the ratio $f_i / f_{avg} = 2.9$ for an individual, the individual will produce 2 offspring in the next generation. After this is done for all the models with fitness greater than 1.0 the remaining empty slots are filled with the models having the highest fitness less than 1.0, and this can include the mantissas for the models with fitness greater than 1.0 also. The selection pressure is very high when this method is used, and although efficient its use is not

recommended for complex problems where early convergence may find a local optimum rather than a global one.

2.4.1.3 Stochastic tournament selection

Stochastic tournament selection is a variant on tournament selection in which extra noise is added to the selection process in order to slow the convergence. Instead of selecting the fittest of two or more individuals, a roulette wheel type tournament is held between the subgroup. For example, if two individuals with fitness of 4.2 and 8.4 are randomly chosen from the population, the latter has a 2/3 probability of being selected and the former probability of 1/3 (rather than automatically selecting the latter) due to the ratio of the fitness values for each individual.

Because improvements tend to diminish late in a GA run, it is often desirable to increase the competitiveness at this stage. One way to do this is to use stochastic tournament selection and scale the fitness appropriately as the run progresses. Power function scaling is probably most appropriate for this selection process, starting with an exponent of 1.0 and increasing a small amount per generation so that towards the end of the run standard tournament selection is effectively being used because small advantages are magnified by the large exponent.

2.4.1.4 Stochastic remainder selection

Stochastic remainder selection is widely used in the GA community. The major reason for its popularity is that it elegantly combines stochastic and deterministic selection in one method.

As with rank only selection, the truncated ratio of a model's fitness to the average fitness of the population automatically results in that number of offspring in the next generation, but here the empty slots are filled by roulette wheel spins using the mantissas of the fitness values. A quick and efficient way to perform this type of selection is called *Stochastic Universal selection.* A subroutine for this is given in appendix A.

2.4.2 Other crossover operators

Although crossover is necessary for optimal performance of the algorithm, too much of it may be disruptive, causing a loss of valuable information. For this reason it is suggested that not all strings be crossed over, so that some of the models in a generation are fully reproduced into the next. Typically a "crossover probability" of 60-80% is used, meaning that for each pair of strings there is a 60-80% chance that crossover will occur.

It is commonly believed that single point crossover is the best choice for a crossover operator because it is the least disruptive in terms of schemata. However, for many problems better results are obtained using more disruptive operators. The most reasonable explanation for this is that there are possibly more important factors than schema disruption in GA performance.

2.4.2.1 Two point crossover

In two point crossover two points are found at random locations ("loci") along the binary string and the bits between the two points are exchanged between the two mates (see Figure 2.5). Two point crossover is slightly more disruptive than single point, but the mechanics are very similar otherwise.

2.4.2.2 Uniform crossover

In uniform crossover (Syswerda, 1989) a new string is generated one bit at a time by randomly taking a bit from either of the parent strings (see Figure 2.6). Uniform crossover at first sight seems to contradict standard GA theory. The probability of disruption for a schema of length l under uniform crossover is $1-(1/2)^{l-1}$, while for 1 point crossover it is only $1/l$-1. Uniform crossover has also been shown to be more disruptive than 2 point crossover for schemata of order 3 in every possible case (Spears and DeJong, 1991).

However, uniform crossover appears to give better results than less disruptive methods in many cases. The reason for this has to do with the tradeoff between efficiency and exploration: disruption from crossover and mutation is clearly undesirable, but it is the only way to create the information diversity necessary to explore large regions of the model space. Spears and DeJong conjecture that uniform crossover may help overcome the limited information capacity of small populations and break up the tendency towards homogeneity.

For a total of n_p different bits between two strings single point crossover allows $2(n_p-1)$ possible combinations, while uniform recombination allows $2^{n_p}-2$. It is

easy to see that uniform crossover can produce a more heterogeneous population. A subroutine for uniform crossover is given in appendix A.

2.4.3 Variable mutation rates

At the beginning of a run mutation is usually not disruptive because the information in the early generations is still mostly random, whereas near the end of the run the information is significantly more ordered and randomly reversing a bit is more likely to produce a weaker model than a stronger one. This line of reasoning supports the idea of a variable mutation rate. Mühlenbein (1992) found that decreasing the mutation rate with each successive generation gave optimal results. However, the results were only marginally better because the GA spends the vast majority of its time finding the optimal values of the last few bits, and this process is not expedited by varying the mutation rate.

Another possibility is to vary the mutation rate for each member of the population within a single generation according to fitness. Naturally, the models with lower fitness should be subjected to a higher mutation rate than those of higher fitness. This scheme is used in Evolutionary Programming (Fogel, 1962; Bäck, 1996; De Groot-Hedlin and Vernon, 1999) in the following manner: the floating point vectors that represent each model are tested with the objective function and are mutated by a Gaussian perturbation distribution with standard deviation proportional to the square root of the model's fitness. This can also be implemented in a GA with the net result being faster convergence, although possibly at the expense of a wider search of the model space.

2.4.4 Elitism

As mentioned previously, elitism is a way to insure that the best model or models in each generation are reproduced into the next, and as a result that the minimum objective function value of the best model in each succeeding generation never increases from one generation to the next. Elitism of degree 2 or more (copying the best 2 models into the next generation) speeds up convergence significantly, which is useful if computation time is limited but it can also cause the run to converge prematurely. Figure 2.7 shows the convergence of receiver function inversions (receiver functions will be explained in detail in chapter 5) with and without elitism. The results are much better throughout the early part of the run using elitism but only slightly better at the end.

Elitism is fairly simple to implement. The binary parameters for the best model or models are saved during the fitness evaluations, and after crossover and mutation, the population is checked to see if the best model from the previous generation was replicated by looping through the entire child array. If not, the model/models are put into a random slot in the binary child array.

2.4.5 Multiobjective optimization

Geophysical inverse problems often require the optimization of two or more criteria at the same time (e.g., adding a *regularization* criterion for smoothness). This type of problem, known as *multiobjective optimization*, is easy to implement in a GA. The objective function generally takes the form

$$f_{obj} = \alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3 + ... + \alpha_n f_n \qquad (2.12)$$

where f_i represents each function to be optimized and the constant parameters α_i represent the weighting factors assigned to each function. For practical purposes, it is best if

$$\alpha_1 + \alpha_2 + \alpha_3 + \ldots + \alpha_n = 1.0 \qquad (2.13)$$

The choice for the values of α_i depend on the relative importance of the corresponding functional factor and its average magnitude and are usually determined by trial and error.

2.5 Conclusions

Genetic algorithms are derivative free global search methods based on Darwinian natural selection. Due to their robust nature and efficiency they are well suited for solving the complex nonlinear inverse problems of geophysics. Inversion codes using GAs make use of information from many parallel objective function evaluations and are therefore easily adapted to take advantage of parallel computer architecture. GAs search a predefined model space without bias, hence avoiding the problem of starting model dependence. Many sophisticated techniques have been successfully adapted to GAs from biological systems in order to solve specialized problems, although using the basic operators produces a robust and efficient search in most cases.

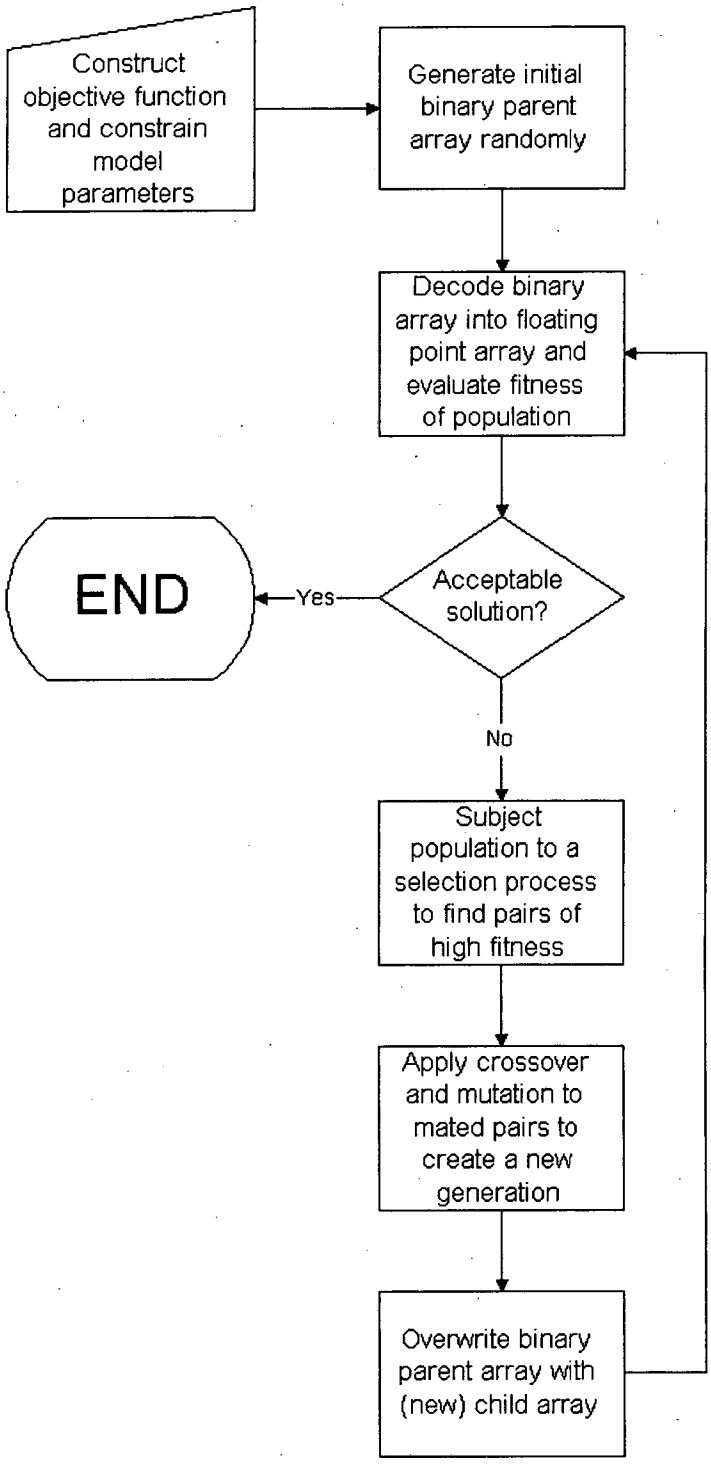


Figure 2.1. Flowchart for a simple genetic algorithm.

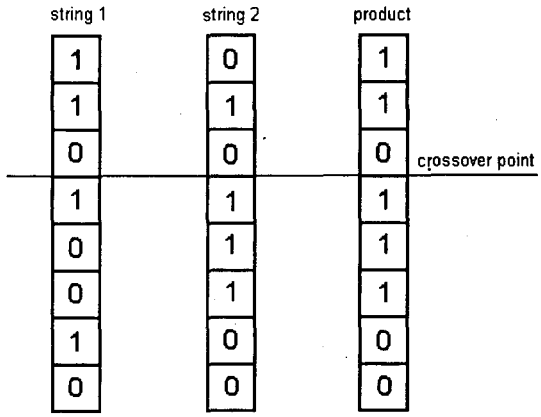


Figure 2.2. Single point crossover. The product has the upper three bits from string 1 and the lower four from string 2. The crossover point is chosen randomly.

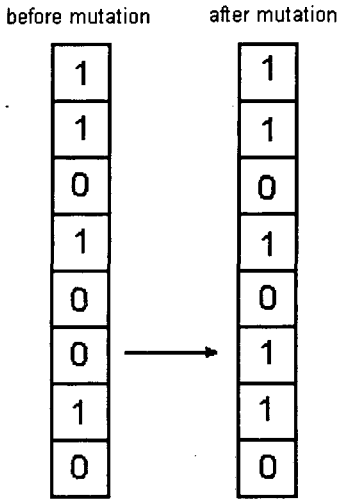


Figure 2.3. The effect of a random mutation on a string. The bit in position number 8 (from the top) is flipped from a '0' to a '1'.

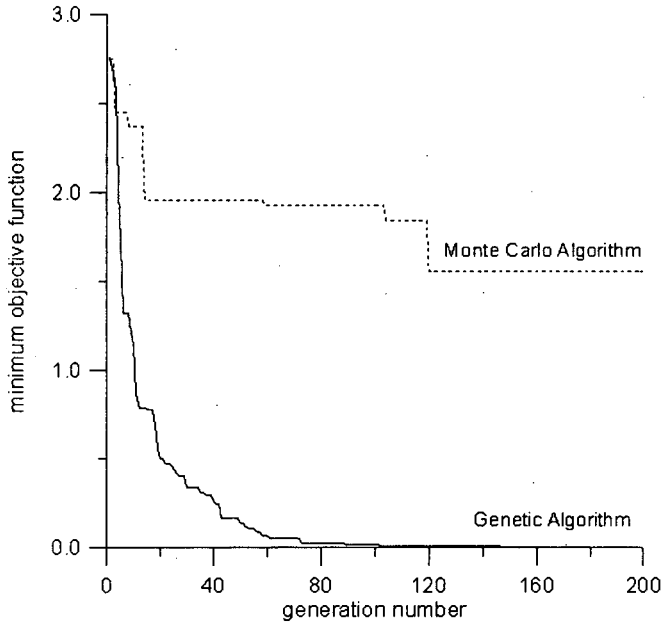


Figure 2.4. Convergence for the minimization of the function $f(x_1, x_2, x_3, x_4, x_5, x_6) = \sqrt{x_1} + \sqrt{x_2} + \sqrt{x_3} + \sqrt{x_4} + \sqrt{x_5} + \sqrt{x_6}$ on [0.0, 1.0] using both a genetic algorithm and a monte carlo routine.

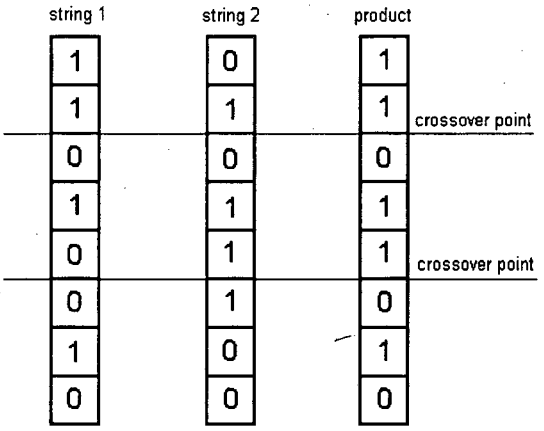


Figure 2.5. Two point crossover. The product inherits bits 1,2,6,7 and 8 from string 1 and bits 3,4 and 5 from string 2.

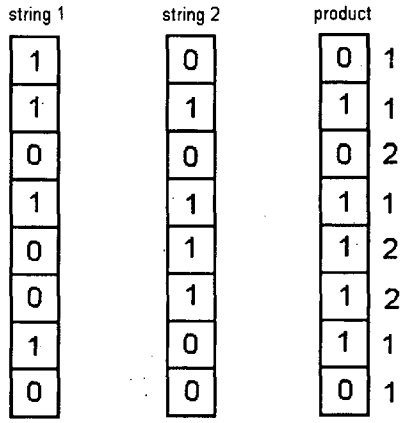


Figure 2.6. Uniform Crossover. Each bit in the product has an equal chance of being inherited from string 1 or string 2. Here the product inherited bits 1,2,4,7 and 8 from string 1 and bits 3,5 and 6 from string 2.

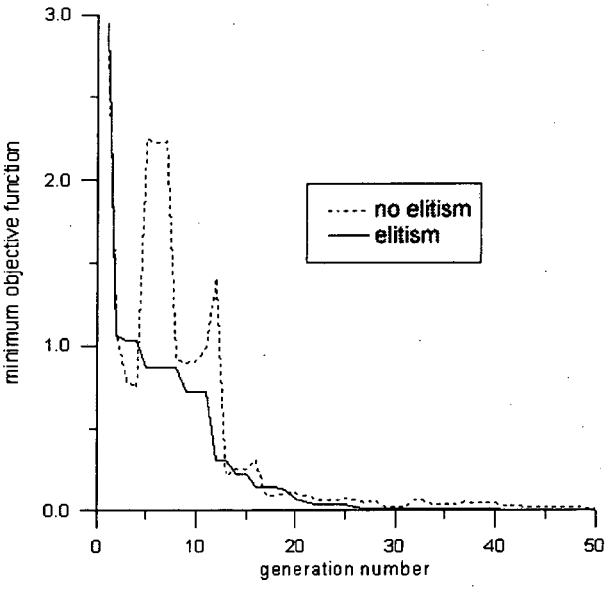


Figure 2.7. Convergence of receiver function inversion problem with and without elitism. Using elitism gives the characteristic monotonic convergence.

Chapter 3

Numerical tests of standard genetic algorithm operators

3.1 Introduction

The Genetic Algorithm (GA) (Holland, 1975; Goldberg, 1989) has proven to be an effective global optimization technique for a wide variety of problems. GA's occupy a middle ground between traditional quasi-Newton (calculus based) methods and global enumerative/random schemes. Although they are not as efficient as calculus based methods, they are capable of finding global optima in multimodal, discontinuous, or noisy objective functions. Because they do not directly use gradient information, GA's do not suffer from the problems inherent in calculating numerical derivatives. Along with simulated annealing (Aarts and Horst, 1989; Kirkpatrick, 1998), GA's are categorized as a random search method, although this does not imply that they are inefficient. GA's process and exploit vast amounts of information in parallel to arrive at a global solution.

Significant debate exists as to what makes GA's effective. Intuitively and in traditional theory GA's work by combining the traits of successful models to produce an optimal model. Goldberg (1989) argues that GA's work by processing large numbers of schemata in parallel through the operation of crossover (a schema is a similarity template which describes a subset of strings with similarities at certain positions). The N^3 argument (Fitzpatrick and Greffenstette, 1988) claims that a GA with population size N processes on the order of N^3 schemata in one generation. This implies that GA's are highly exploitative and

non-random. In such a scheme the crossover operator is performing the major part of the work and the mutation operator is used only sparingly to insure that no information is permanently lost in the crossover process. In fact Holland (1975) described mutation as a "background operator" that should serve only this purpose. All of this would suggest that a high crossover rate should be used in conjunction with a low mutation rate. However, recent research has suggested that mutation may be a more important factor in the performance of a GA than crossover. Mühlenbein (1992) demonstrates that for some simple problems an algorithm that uses mutation but no crossover performs quite well, and the field of evolutionary programming (Fogel, 1962) is based entirely on mutation without crossover.

The two major questions that will be investigated in this chapter are:

1) Which operators are the most important for genetic algorithm performance?

2) What are the optimal search parameters for these operators?

The approach is based on empirical trials using a complicated multimodal objective function.

3.2 Testing Methodology

A suite of tests are performed using the multimodal test function as shown in Figure 3.1. The model space for each parameter is [-5.0, 5.0] and 16 bit strings are used (except for test number 2, in which string length is varied) giving a precision of four decimal places per parameter. The tests are chosen to find relationships between selection pressure, optimal crossover and mutation rates and

population size, string length, and problem difficulty. In all of the tests, performance is defined as

$$performance = \frac{obj(b_0) - obj(b_C)}{obj(b_0)} \tag{3.1}$$

where b_0 is the best fitting model of the first generation, b_C is the best fitting model of the current generation, and $obj(\ )$ is the objective function. Defined in this manner an exact solution (in which the objective function is 0.0) is assigned a performance of 1.0 and no improvement over the initial population gets a 0.0. For reliability, each performance realization is averaged over 10 runs, each run with a different random number seed. Each run consists of 100 generations with a population size of 100 (except runs for test number 3, in which population size varies).

The tests consist of:

1) Performance vs. mutation rate and problem difficulty
2) Performance vs. mutation rate and string length
3) Performance vs. mutation rate and population size
4) Performance vs. crossover rate and problem difficulty
5) Performance vs. selection pressure and problem difficulty
6) Performance comparison of several selection routines

3.2.1 Problem Difficulty

In several of the tests the parameters of GA operators are varied along with problem difficulty in order to see if a relationship can be derived. The term

problem difficulty is defined by Goldberg (1993) as a set of quasi-separable factors:

- Isolation
- Misleadingness
- Noise
- Multimodality
- Crosstalk

Isolation and misleadingness combine to produce what Goldberg calls *deception*, in which the dominant gradient leads to a suboptimal point. Test problems using deception have been utilized in many recent papers, however, multimodality is used here as the primary factor of difficulty because it can be increased exponentially by adding to the dimensionality of the problem and therefore difficulty can be uniformly varied over a wide range.

3.3 Test Results

3.3.1 Test 1

The sensitivity of convergence to changes in the mutation rate is explored by estimating the global minimum of the multimodal test function shown in Figure 3.1. The problem difficulty is increased by expanding the number of dimensions from 10 to 30. Thus the search space is increased from 2^{160} to 2^{480} (10^{48} to 10^{144}) points. Tournament selection and single point crossover with a crossover

probability of 0.7 is used. An exact solution for the 10 parameter problem takes 93 generations using the above parameters and a mutation rate of 0.006.

The performance is measured while mutation rate is varied from 0.0 to 0.1 in increments of 0.001 and the problem difficulty is varied from 10 to 30 dimensions. Single point crossover with a crossover rate of 0.7 and tournament selection are used in this experiment and kept constant throughout the entire run. The results are displayed in Figures 3.2-3.5, revealing a surprising dependence of performance on mutation rate for rates below 0.1 (beyond this point the performance becomes asymptotic at just above 0.2 and the algorithm performs as a random search) and extreme sensitivity in the range of 0.0-0.04. This sensitivity is seen most clearly in Figure 3.4, in which the results of the previous figures are averaged. It is apparent from this curve that the optimal mutation rate does not vary significantly over the range of 10-30 dimensions. The peak performance corresponds to a mutation rate of 0.004, which is about half of the inverse of the population size (test 3 compares performance to mutation rate and population size). Figure 3.5 shows the mutation rate which produced peak performance for each number of dimensions. The slope of -0.0000377 is statistically insignificant, and it is clear from this figure that the optimal mutation rate is constant for this range of problem difficulty.

3.3.2 Test 2

In this test performance is measured while mutation rate is varied from 0.0 to 0.1 in increments of 0.001 and the string length (in the binary representation) is

varied from 10 to 30 bits. As in test 1, single point crossover with 0.7 crossover rate and tournament selection are used in this experiment and kept constant throughout the entire run. The test function to be minimized is that of Figure 3.1 using 20 dimensions. The results are seen in Figures 3.6-3.7. Again, the extreme dependence of performance on mutation is seen as in test 1, but there is apparently no relationship between optimal mutation rate and string length. This does not preclude a relationship for string lengths of less than 10, but below that number the solution accuracy is less than two decimal places, and the model space becomes unacceptably small for this type of analysis.

3.3.3 Test 3

The performance is tested while the mutation rate is varied from 0.01 to 0.1 in increments of 0.001 and population varies from 10 to 200. The number of generations is changed accordingly so that each run has the same number of objective function evaluations. The test function is that of Figure 3.1 using 10 dimensions. Single point crossover with 0.7 crossover rate and tournament selection are also used in this case and kept constant throughout the entire run.

The plots in Figures 3.8-3.9 show that this particular problem is most efficiently solved with a small population and a high number of generations. It is interesting to note that performance rises very steeply as the population becomes less than 50 (this is most easily seen in Figure 3.9), while the performance for populations smaller than 20 is acceptably high for all mutation rates less than 0.1. This is in stark contrast to the previous results in which performance was

unacceptably low below about 0.05. This suggests that higher mutation rates are beneficial for smaller populations which tend to have limited information content.

Note that the mutation rate is not examined below 0.01 in this experiment, which is why there is no rapid fall off in performance at that end as seen in the previous figures.

3.3.4 Test 4

Here performance is measured while the crossover rate is varied from 0 to 100% in increments of 1% and the problem difficulty is varied from 10 to 30 dimensions. The crossover operator is single point, the mutation rate is 0.01 and the selection operator is tournament selection. The results shown in Figures 3.10-3.11 are surprising: a moderate increase in performance (about 25%) is apparent if crossover is used, but the performance is virtually constant for all crossover rates above 1%. Figure 3.12 shows the rate of convergence for a much more complex problem, an inversion of gravity data to obtain crustal parameters. In this case 400 parameters are being optimized using a computationally intensive objective function. The dotted line is the convergence for the case of no crossover and 0.5% mutation rate, the solid line for 70% crossover and 0.5% mutation rate, and the dashed line is the case for 70% crossover and 10% mutation rate. For the two lower curves the misfit of the final solution is decreased by about 25% using crossover, which is consistent with the above results. However, the algorithm without crossover performs vastly better than the one with a high mutation rate.

This demonstrates that performance is much more sensitive to the choice of mutation rate than it is to the choice of crossover rate.

Thus far single point crossover has been the only crossover operator used. The newer method of *uniform crossover* (Syswerda, 1989) has been found to be more efficient for many types of problems. Although it is far more disruptive than single point crossover it is believed that uniform crossover may help to overcome the information deficiencies associated with relatively small populations (Whitley, 1996). In Figure 3.13 the performance of uniform crossover is subtracted from that of single point crossover for crossover rates from 0 to 100%. For reliability, each point represents the averaged results of 10 runs. While there is little difference in the results, the gentle but consistent slope indicates that a lower crossover rate may be more effective when using uniform crossover and a higher rate more effective when using single point crossover. This is consistent with the notion that greater disruption may be necessary to overcome information deficiencies.

3.3.5 Test 5

This test demonstrates the effectiveness of fitness scaling when using roulette wheel selection. Performance is measured as the linear scaling factor is increased from 1.0 (no scaling) to 4.0 in increments of 0.25 and the difficulty is increased from 10 to 30 dimensions. Single point crossover is used with a 70% crossover rate and a mutation rate of 0.01. In contrast to what is suggested by Goldberg (1989), Figures 3.14-3.15 imply that the best results are obtained when the scaling

factor is greater than 2.0, although performance appears to be fairly constant for values above 2.0. Something else that should be noted here is that using roulette wheel selection with no fitness scaling at all gives mediocre results at best, and for greater problem difficulty gives results only slightly better (~25%) than a random search.

3.3.6 Test 6

The last test compares the performance of roulette wheel selection and stochastic remainder selection to tournament selection. The crossover operator is again single point with 70% crossover probability while the mutation rate is 0.01. 10 runs using roulette wheel selection and stochastic remainder selection are performed for 100 different scaling factor increments between 1.0 and 4.0. Figure 3.16 shows the performance for the multimodal test function of Figure 3.1 in 10, 20, and 30 dimensions. Tournament selection without scaling is performed in parallel for comparison with performances averaged over 10 runs. From these plots it appears that tournament selection is vastly superior to the other two methods for all choices of scaling factors in both average performance and consistency. Stochastic remainder selection tends to outperform roulette wheel selection on average, but there is also considerably more variance in performance. The results are similar for each number of dimensions.

Table 3.1 gives the statistics for the plots in Figure 3.16, while table 3.2 presents the statistics for scaling factors in the range of 2.0-4.0 only. As was noted from the results of test 5, if either roulette wheel selection or stochastic

remainder selection are used, they should be used in conjunction with a scaling factor of 2.0 or larger.

	10 dimensions		20 dimensions		30 dimensions	
	mean $\bar{x}$	stdev σ	mean $\bar{x}$	stdev σ	mean $\bar{x}$	stdev σ
roulette wheel selection	0.8421	0.1163	0.6158	0.0864	0.5060	0.0842
stochastic remainder selection	0.8663	0.1212	0.6236	0.0959	0.5185	0.0832
tournament selection	0.9804	0.0023	0.7542	0.0127	0.6240	0.0086

Table 3.1. Mean performance and standard deviations of trial results for three different selection routines using a scaling factor range of 1.0-4.0.

	10 dimensions		20 dimensions		30 dimensions	
	mean $\bar{x}$	stdev σ	mean $\bar{x}$	stdev σ	mean $\bar{x}$	stdev σ
roulette wheel selection	0.8951	0.0162	0.6565	0.0110	0.5503	0.0120
stochastic remainder selection	0.9196	0.0333	0.6665	0.0346	0.5603	0.0285
tournament selection	0.9806	0.0021	0.7541	0.0128	0.6242	0.0080

Table 3.2. Mean performance and standard deviations of trial results for three different selection routines using a scaling factor range of 2.0-4.0.

3.4 Conclusions

DeJong's (1975) empirical studies using 5 different test problems suggested that for optimal GA performance the mutation rate should be in inverse proportion to the population size. Using both empirical and theoretical arguments, Shaffer et al. (1989) and Hesser and Männer (1991) confirmed this result. However, Mühlenbein (1992) shows that this is not true in all cases. The tests performed here (Figures 3.8-3.9) show a moderate but significant inverse relationship between optimal mutation rate and problem size, also suggesting that a slightly larger mutation rate may be able to compensate for the information deficiencies associated with a small initial population.

Although the choice of genetic algorithm operators and parameters is to some extent problem dependent, the results from these trials show that a particular choice of operators and parameter settings consistently produces optimal results for a reasonably broad range of problem difficulties. In particular a low mutation rate (about half of the inverse of the population size) is crucial for optimal results, but the choice of crossover method and rate seem to be much less important. If computation time is an important factor, optimal efficiency may be achieved with smaller populations (< 50).

Finally, tournament selection appears to be the best choice of selection methods due to its simplicity and its autoscaling properties, but if a proportional selection method is used such as roulette wheel selection, fitness scaling is essential to achieve acceptable performance and the scaling factor should be high for optimal performance (> 2.0).

It must be noted that in the present work modality is the only factor of difficulty that is varied, and more work should be done to determine how this choice of operators and parameters varies if deception is also varied.

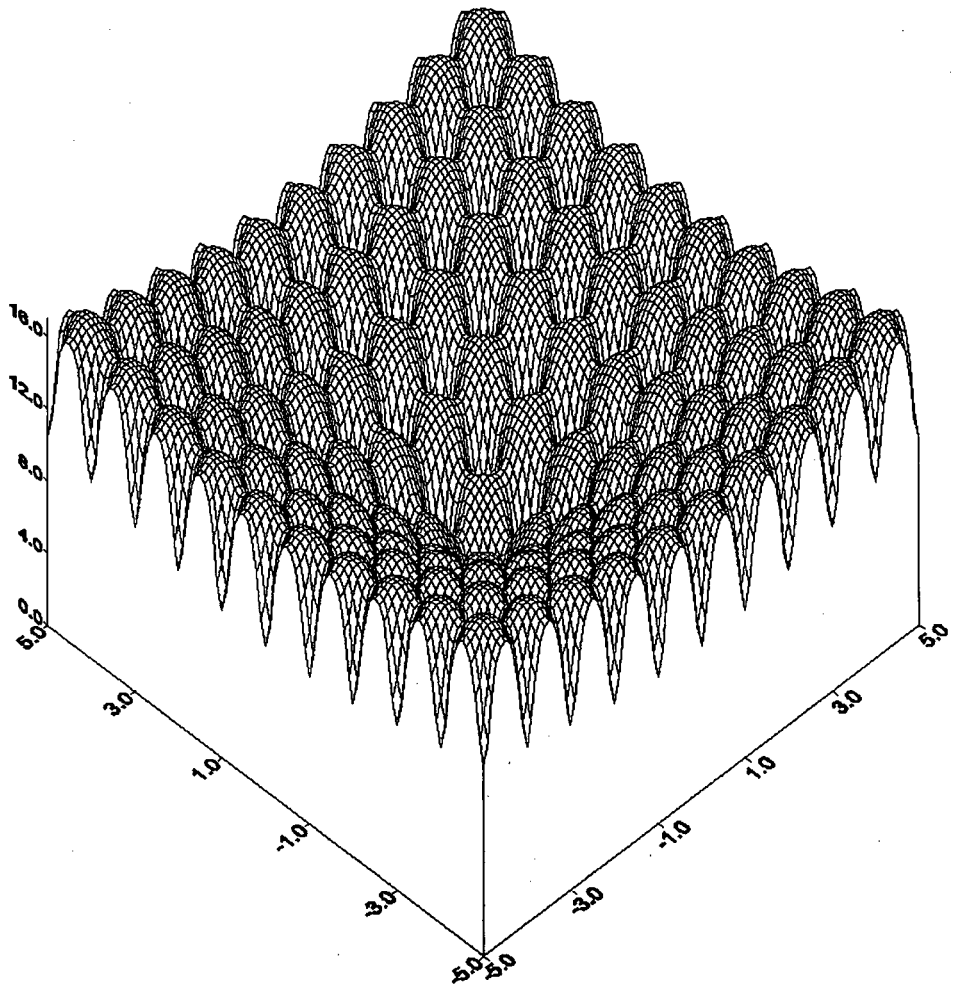


Figure 3.1. Ackley's multimodal function used in the numerical experiments (recreated from Bäck, 1996).

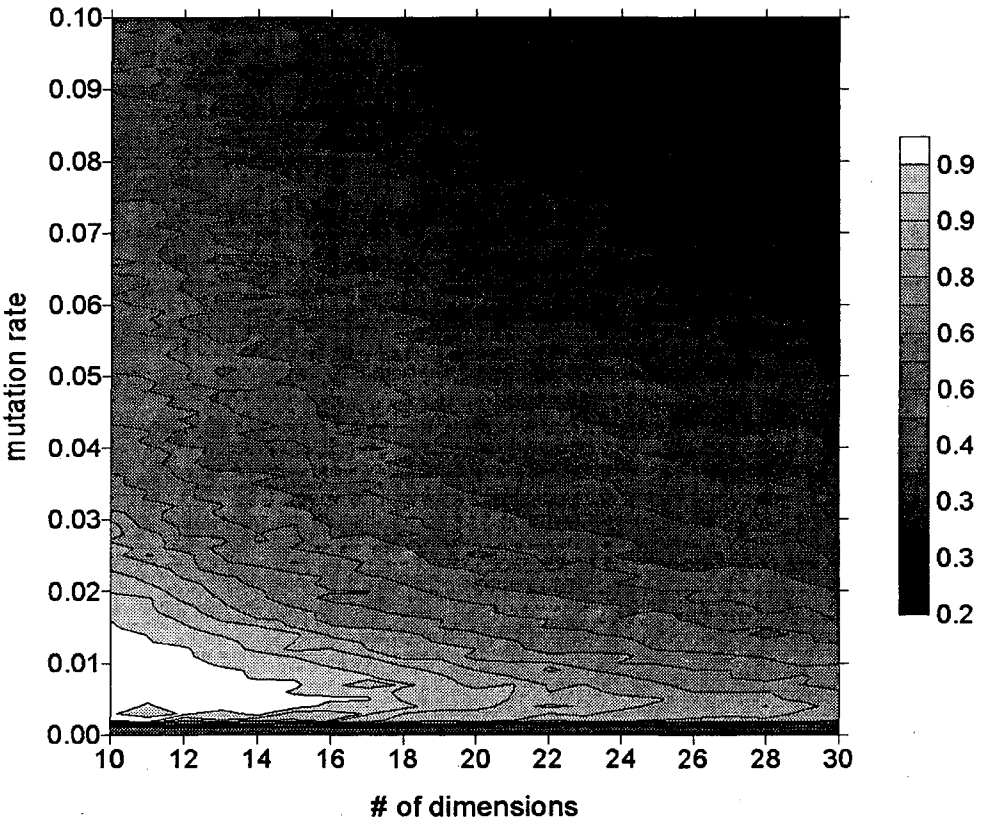


Figure 3.2. Contour plot of performance vs. mutation rate and problem difficulty.

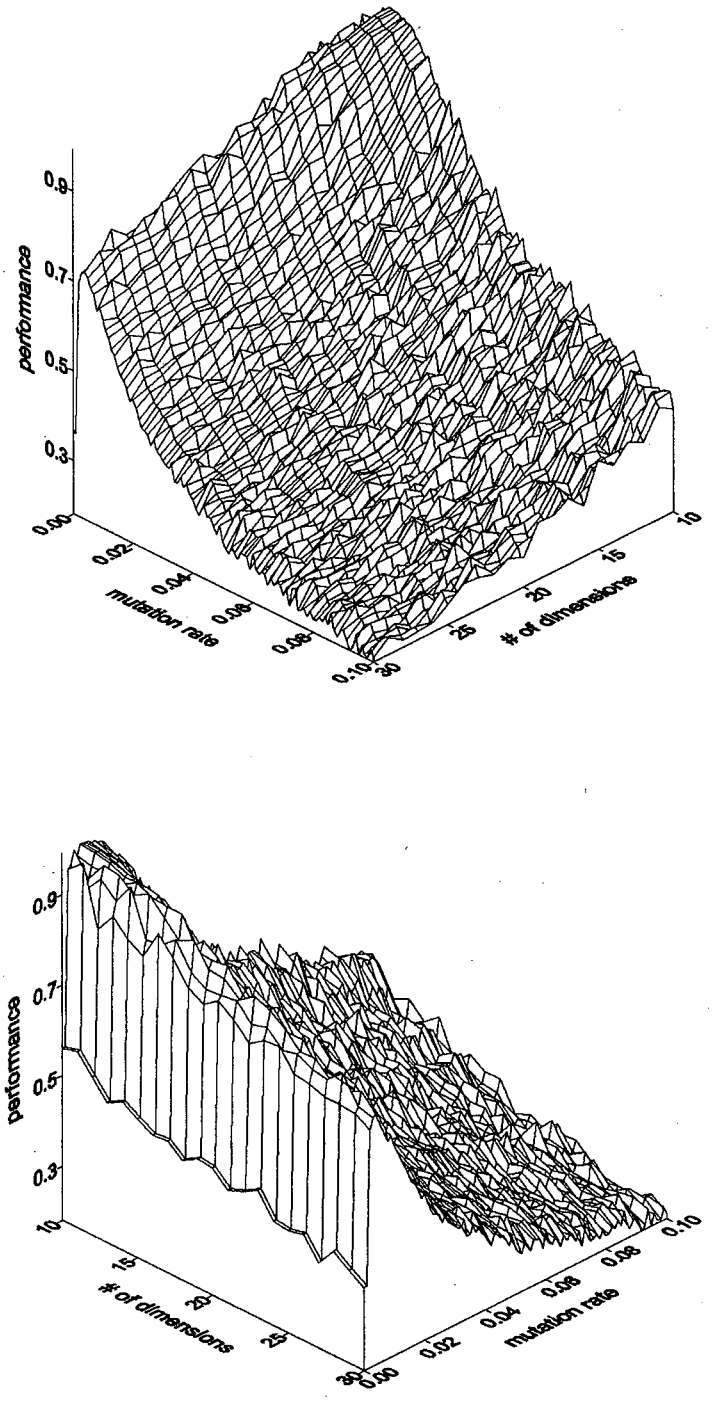


Figure 3.3. Surface plots (rotated at different angles) of performance vs. mutation rate and problem difficulty.

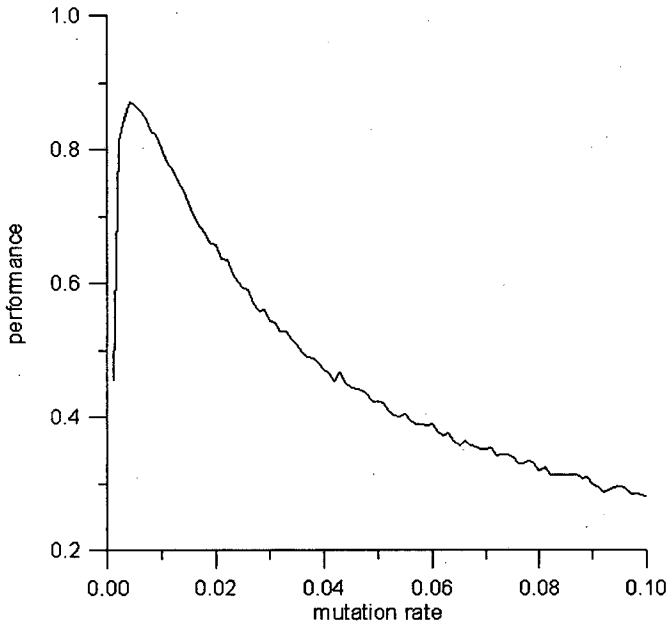


Figure 3.4. Averaged results from figures 2-3 showing peak performance at P_{mutate}= 0.004.

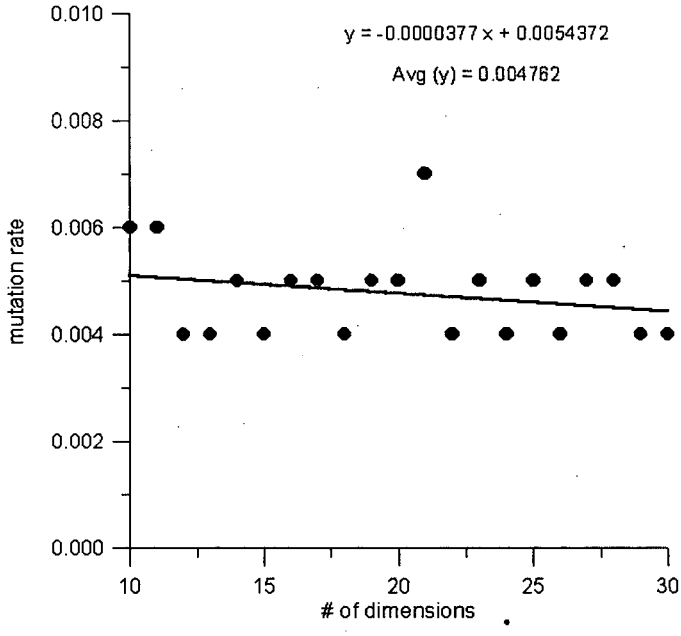


Figure 3.5. Highest performance for each number of dimensions.

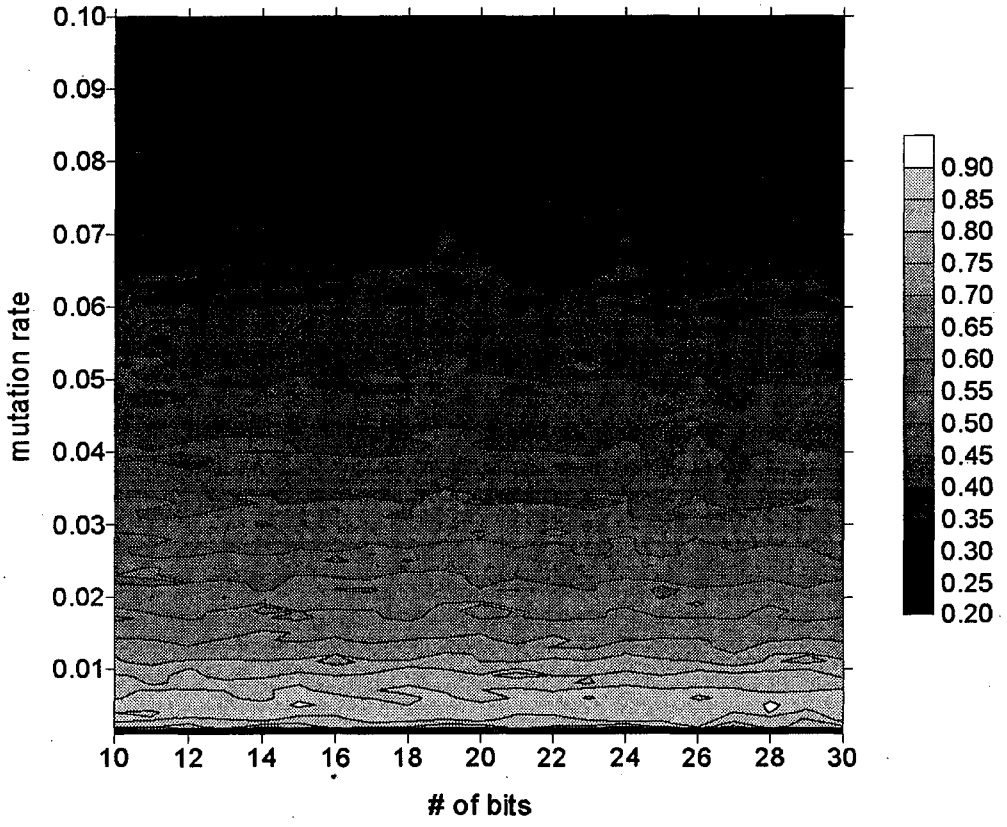


Figure 3.6. Contour plot of performance vs. mutation rate and number of bits (string length).

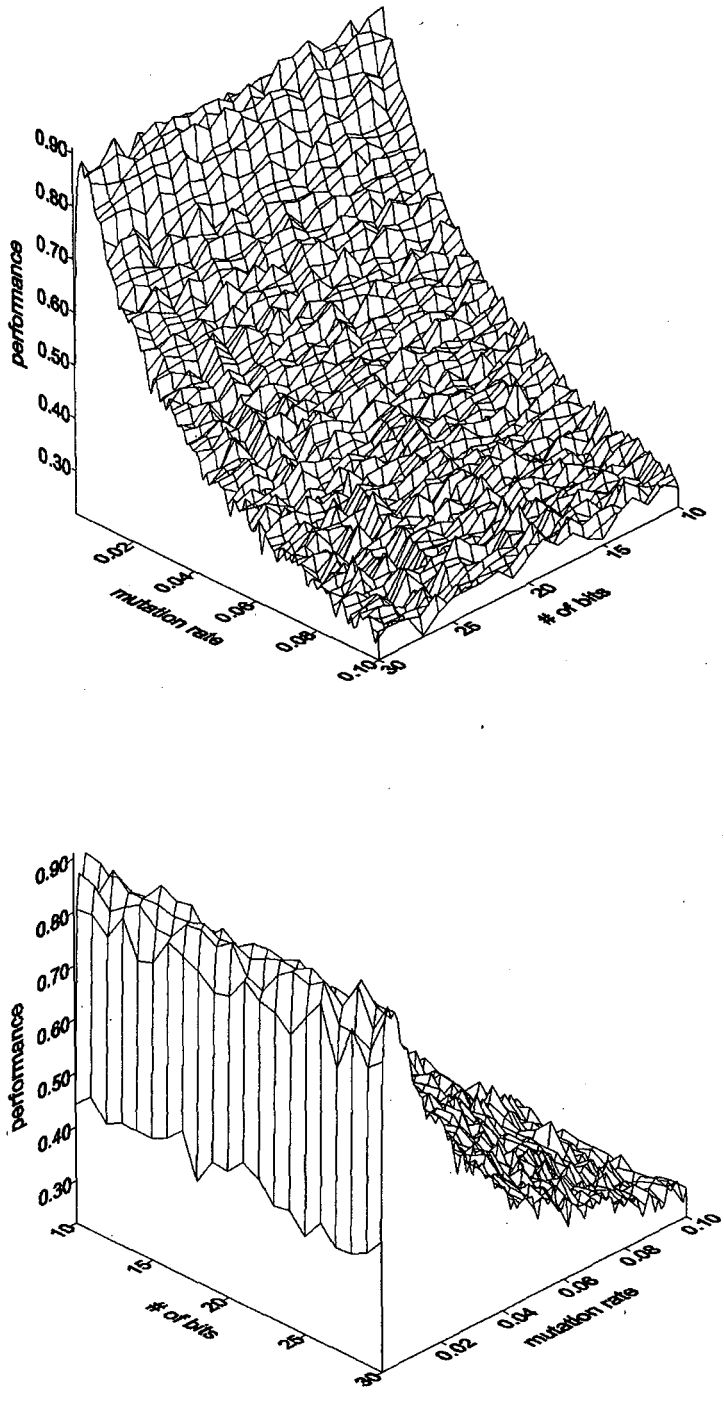


Figure 3.7. Surface plots of performance vs. mutation rate and number of bits (string length).

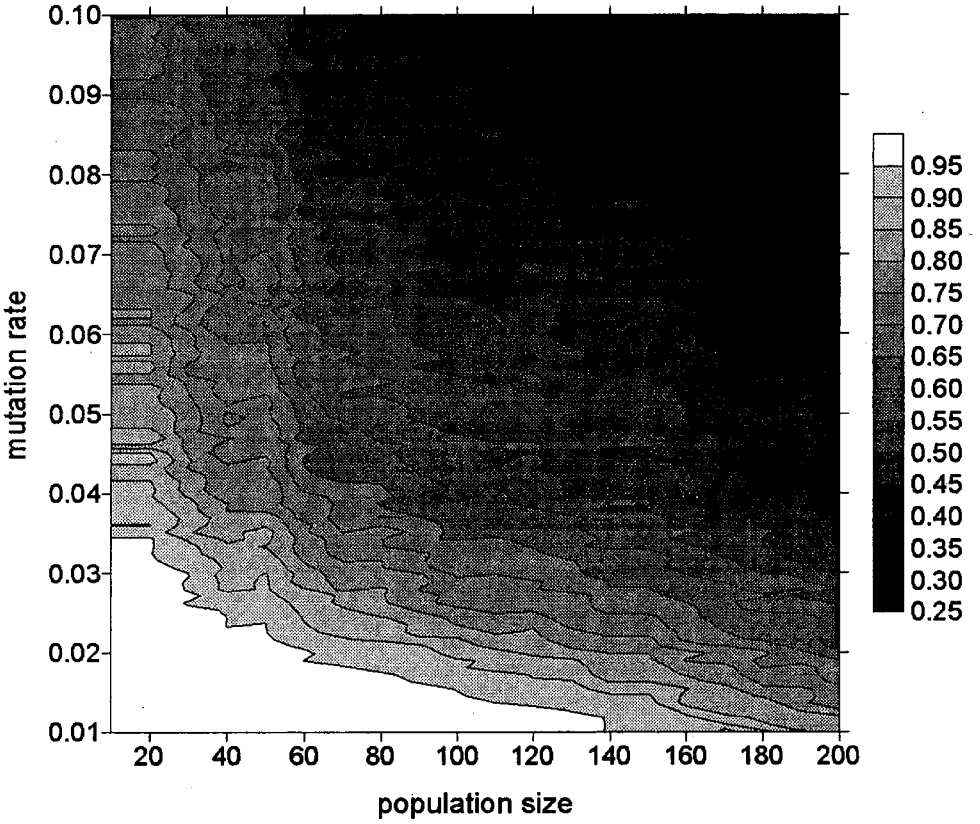


Figure 3.8. Contour plot of performance vs. mutation rate and population size.

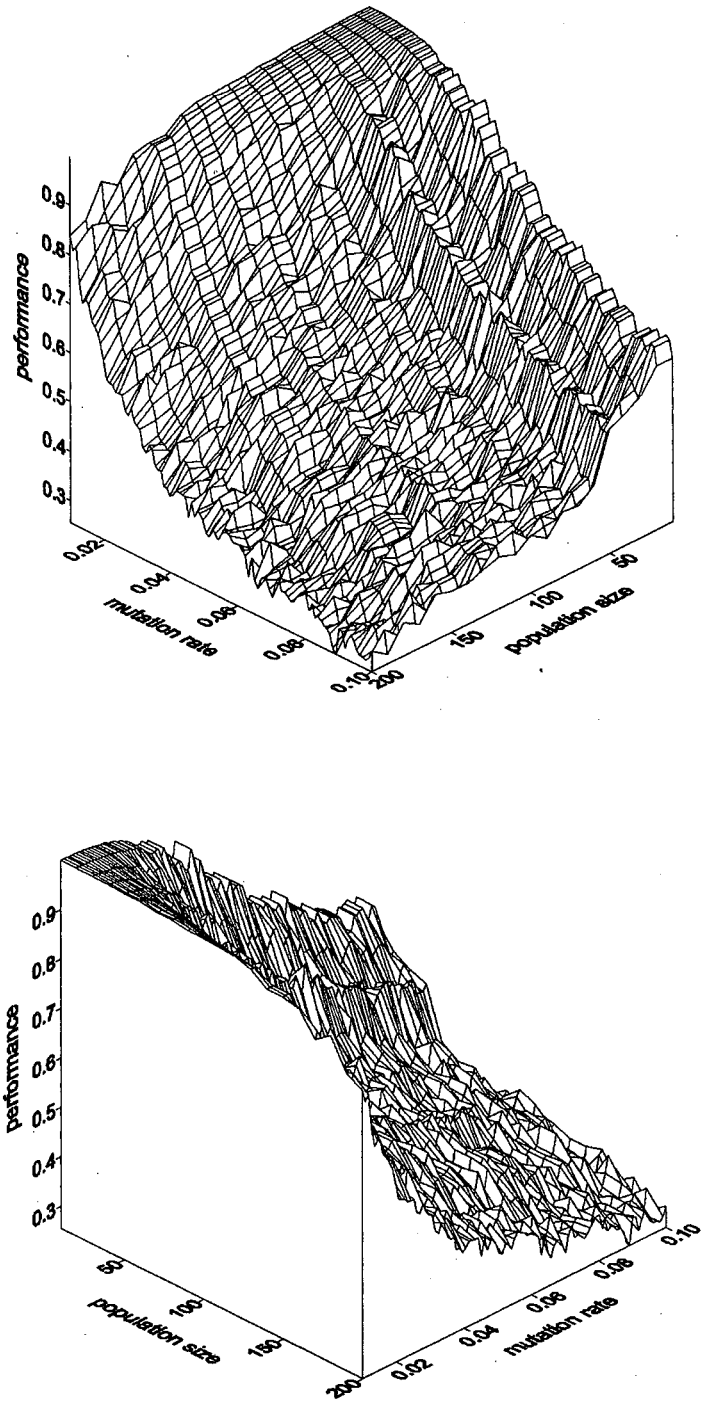


Figure 3.9. Surface plots of performance vs. mutation rate and population size.

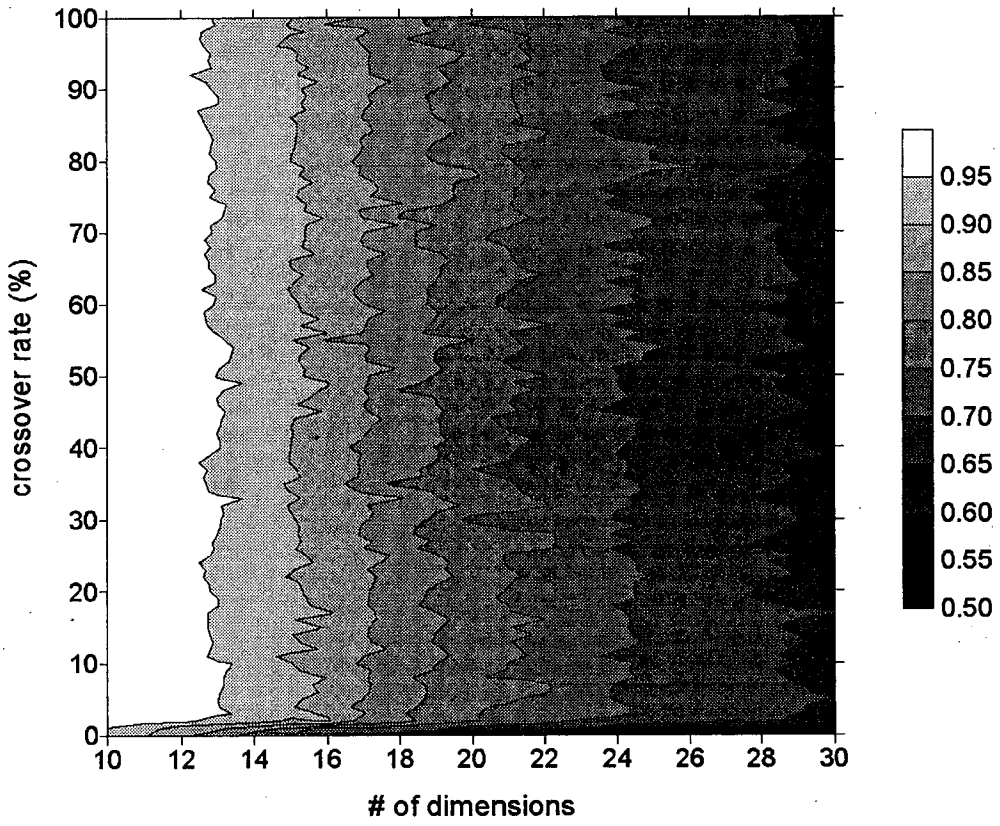


Figure 3.10. Contour plot of performance vs. crossover rate and problem difficulty.

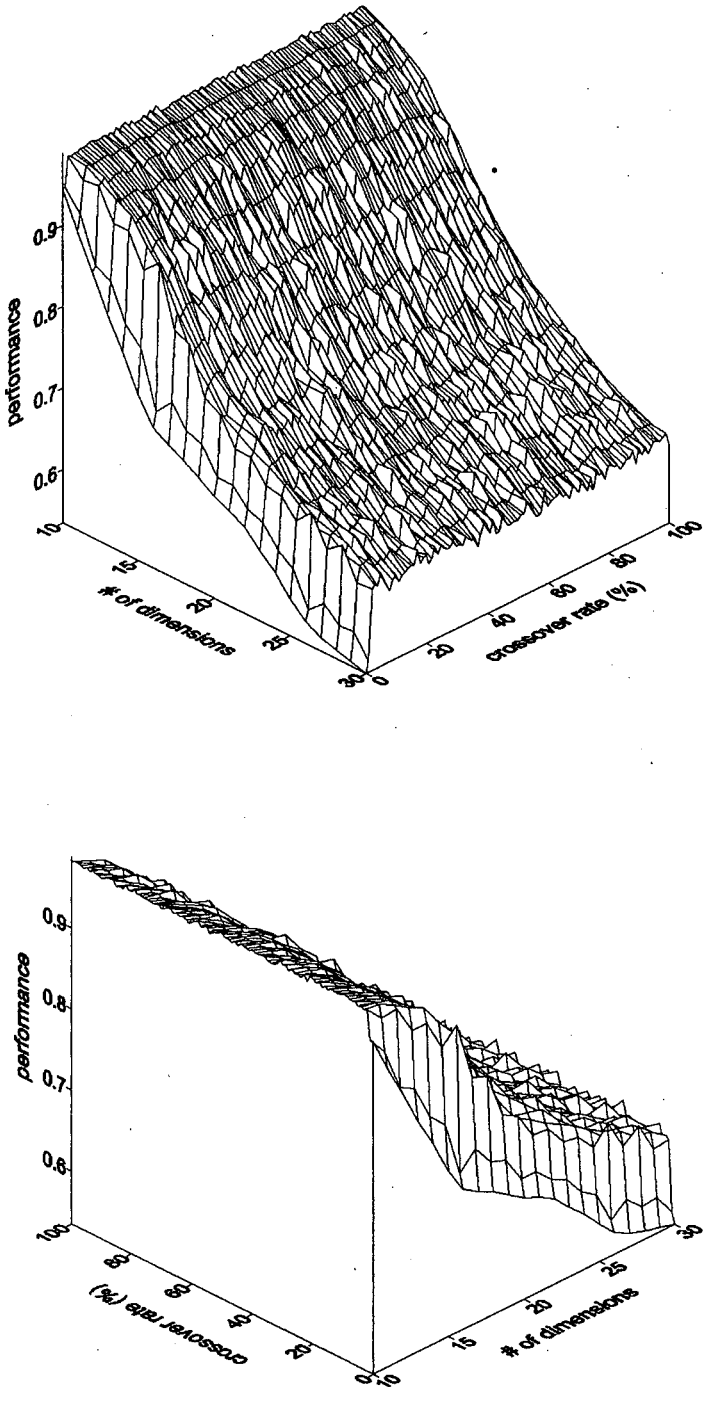


Figure 3.11. Surface plots of performance vs. crossover rate and problem difficulty.

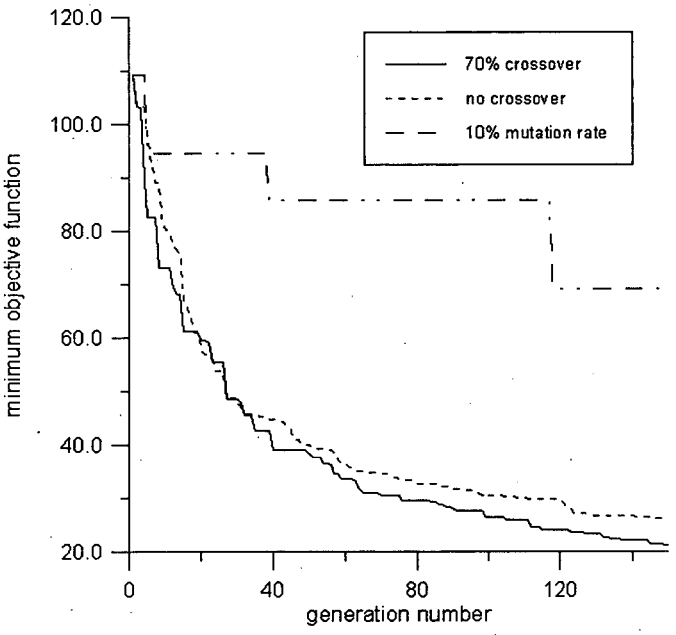


Figure 3.12. Convergence of gravity problem for three different cases, demonstrating the relative sensitivity of performance to crossover and mutation rates. The runs with 70% and no crossover both have a mutation rate of 0.005.

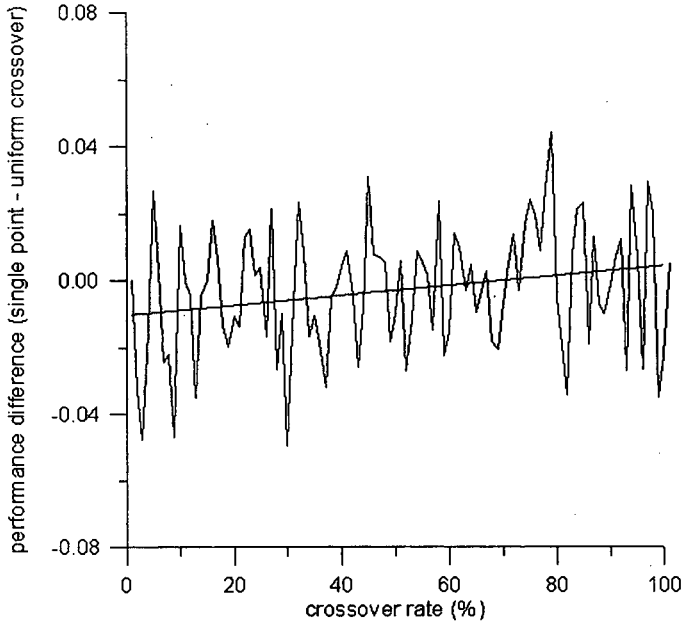


Figure 3.13. Performance difference for single point crossover - uniform crossover.

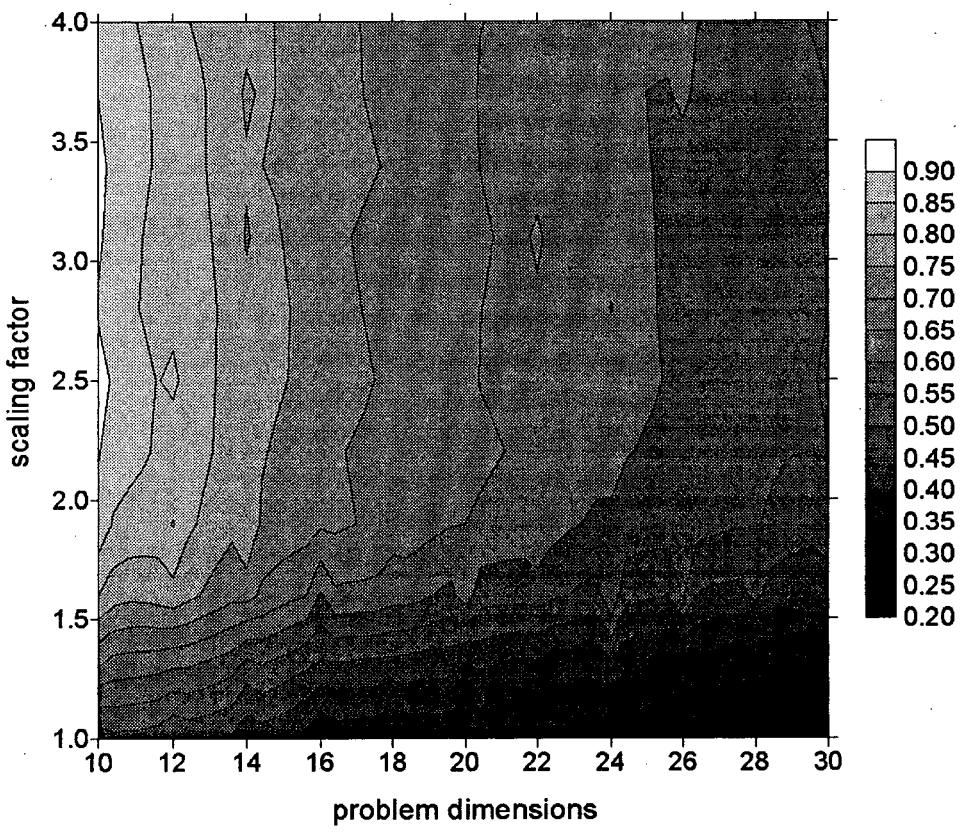


Figure 3.14. Contour plot of performance vs. fitness scaling factor and problem difficulty.

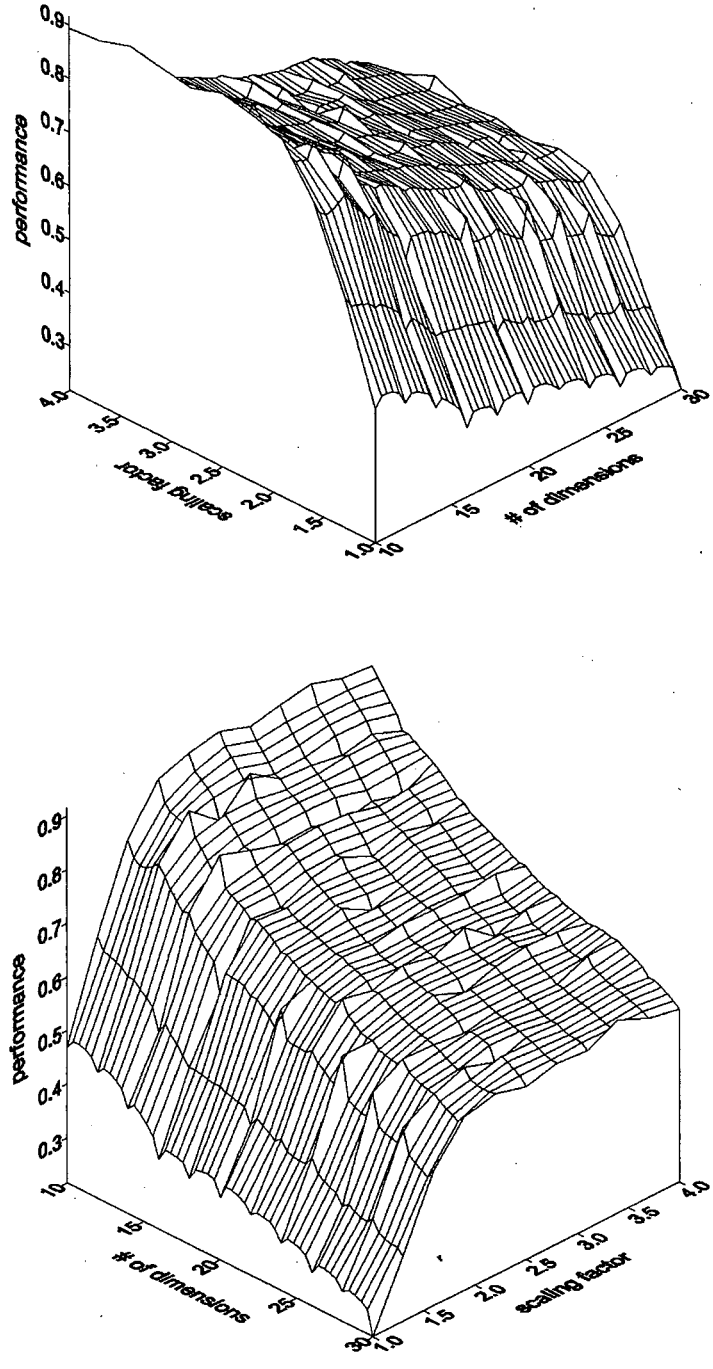


Figure 3.15. Surface plots of performance vs. fitness scaling factor and problem difficulty.

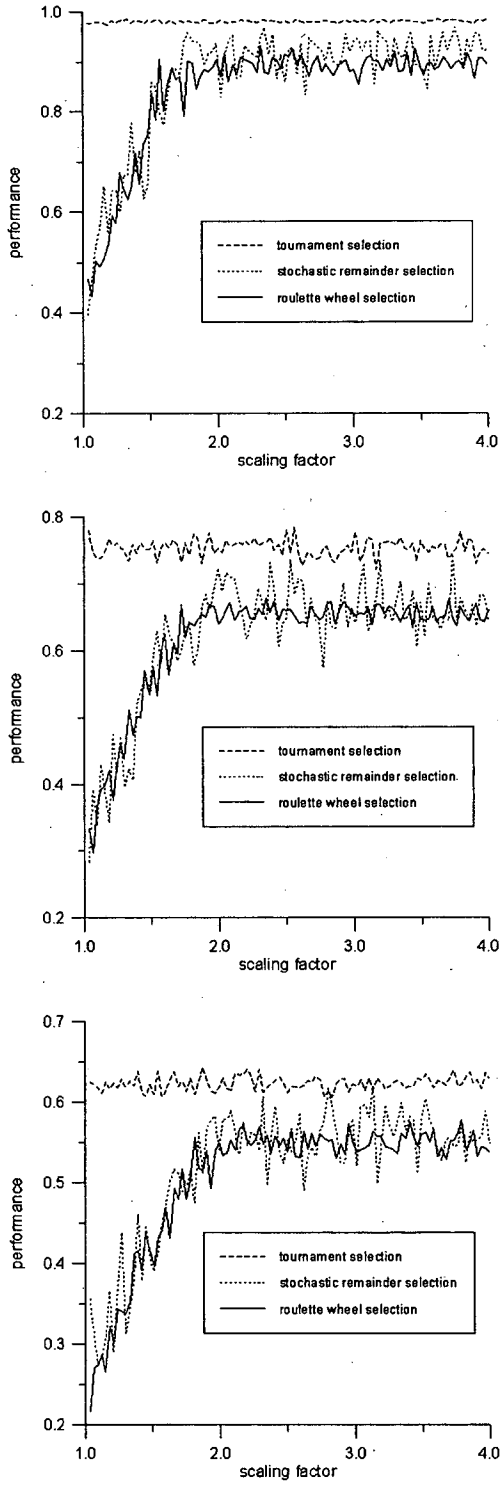


Figure 3.16. Performance of scaling factors from 1.0-4.0 using 3 different selection methods for 10 dimensions (top), 20 dimensions (middle) and 30 dimensions (bottom).

Chapter 4

An example problem from gravimetry: inversion for crustal basement depth

4.1 Introduction

The difficulties involved with the inversion of gravity data are typical of many geophysical problems: large data set of surface measurements, a computationally costly objective function evaluation, many parameters to be solved for, and nonuniqueness due to an inherent ambiguity between depth and the magnitudes of physical measurements. In the case of gravity the nonuniqueness derives from a depth/density tradeoff, but it can be minimized or even removed by constraining density as a function of depth (using borehole data or geologic information) at one or more points within the region of study. In this chapter a genetic algorithm will be used to estimate the crustal basement depth below Yucca Mountain in Nevada assuming that the density profile is known.

In many inverse problems, the quality of the solution is limited by the available computation time. The two independent factors primarily responsible for the potential time required are the number of parameters to be estimated in the problem and the evaluation of the objective function. In order to solve an inverse problem with any degree of accuracy and efficiency, the choice of solution method should take these factors into account. The following problem involves the optimization of 400 parameters and requires approximately 1.2 seconds to evaluate the objective function on a 366 MHz Sun Enterprise 4000 computer so

that 150 generations of 150 models take approximately 7.4 hours. Calculating the inverse of a (non sparse) 400 X 400 matrix is computationally expensive, as is any method in which computation time increases with the square of the number of parameters. Although GA's are not the most efficient optimization method, the computation time required to obtain an acceptable solution is not dependent on the square of the number of parameters, so it is a reasonable method to use for this problem. A greater problem is the objective function evaluation time as GA's use an extremely large number of objective function evaluations. An objective function evaluation that requires 1 second is significant if the problem needs to be solved in about a day. An objective function evaluation time greater than a few seconds would not allow a population size worthy of the complexity of this type of problem and may not be suitable for a GA, although as computers become faster methods that require a high number of objective function evaluations become more feasible.

4.2 Problem background

Raw gravity data can be reduced to derive Bouguer gravity anomalies which can be inverted to estimate crustal parameters. The acceleration of gravity in the z direction (the direction along which gravity is measured) in rectangular coordinates is given by

$$g_z = \frac{\partial U}{\partial z} = -\gamma \iiint \frac{z}{r^3} \rho(z)\, dxdydz \qquad (4.1)$$

where U is the gravitational potential and γ is the gravitational constant (Telford, et al., 1976). There is inherent nonuniqueness involved in inverting this equation due to a depth/density tradeoff which can be seen in the integrand of the above equation, but if one of these parameters is constrained by some other means (e.g. density profiles from borehole or well logging data) this nonuniqueness can be minimized or effectively removed if the data coverage is good enough.

4.3 Gravity data and data reduction

Figure 4.1 shows 127 station points along 17 linear gravity profiles collected in 1994 in the vicinity of Yucca Mountain, Nevada, from 36.8° to 36.9° North latitude and from 116.5° to 116.4° West longitude. This region, approximately 11 by 9 km in size, is directly above the site of the proposed underground nuclear waste repository. The gravity measurements were made with LaCoste-Romberg model G gravimeters. A drift correction factor was calculated after measuring gravity with both meters at roughly ¼ of all the stations. These crossing points were also used to assure that the accuracy of the measurements was within 0.1 mgal throughout the survey.

To estimate a solution for the basement depth from these data, it is necessary to know the density as a function of depth for the region. This information can be obtained by studying surface rocks and drill cores, and by taking measurements in boreholes. Using the aforementioned techniques Snyder and Carr (1984) found that the density varies with depth within the tuff section below Yucca Mountain according to the relation

$$\rho(z) = 1.95 + 0.26z \; (g / cm^3) \tag{4.2}$$

where z is the depth and $\rho(z)$ is the density. The density of the corresponding paleozoic basement rocks was found to be 2.66 g/cm^3. The basement density is assumed to be constant with depth for the purposes of the inversion.

The Bouger gravity anomalies were calculated using the following formula (modified from Telford, et al., 1976):

$$\delta g_B = g_{obs} + dg_L + dg_{FA} - dg_B + dg_T + dg_t + dg_d \tag{4.3}$$

where g_{obs} is the gravity measured at the station, dg_L the latitude correction, dg_{FA} the free air correction, dg_B the Bouger correction, dg_T the terrain correction, dg_t the tidal correction, and dg_d the drift correction.

4.4 Problem parameterization and Inversion method

In the interest of simplifying the inversion, long wavelength Bouger anomalies are interpreted as changes in depth to the paleozoic basement. The region shown in Figure 4.1 was discretized by dividing it into 400 rectangular cells in which density varied with depth according to Equation (4.2) and the depth to basement was a free parameter to be estimated. Each of the 400 basement depth parameters was represented in the GA with a binary string of 8 bits in length, giving a numerical precision of 1 part in 256. The search space was created using simplified initial model proposed by Johnson et al. (1995) which fit the data reasonably well. The minimum value of each parameter was

$$z_{\min_i} = z_{0_i} - 0.3z_{0_i} - 1.5 \;\; [\text{km}] \tag{4.4}$$

and its maximum value was

$$z_{\max_i} = z_{0_i} + 0.3z_{0_i} + 1.5 \quad [\text{km}] \tag{4.5}$$

where z_{0_i} is the parameter value from Johnson's model.

The forward calculations for the inversion were made following the method of Johnson and Litehiser (1972) for calculating the gravitational field of three dimensional bodies of arbitrary shape within a spherical earth. The objective function is given by the summed squares of the gravity residuals:

$$f_{obj} = \sum_{i=1}^{n} (d_i^{obs} - d_i^{est})^2 \tag{4.6}$$

4.5 Inversion results

The GA inversions consisted of 150 generations with initial populations of 150 models. Figure 4.2 shows the convergence for 3 different GA runs and one Monte Carlo run. The three GA runs represent three different selection methods: roulette wheel selection, stochastic universal selection and tournament selection. Tournament selection produced the best fit, with an average misfit for each station of 0.055 mgal, which is about half of the measurement precision, and about one-third of the misfit of the initial model by Johnson. No smoothness constraint or regularization of any type were applied to the objective function, and the inversion appears somewhat unstable in regions with sparse data coverage. However, the same major features can be identified in all of the inversion results.

Figure 4.3 is a contour plot of the estimate for the basement depth for the run using tournament selection. A surface plot of the basement topography for the

same model is shown in Figure 4.4. The most prominent feature is a rise of basement depth from west to east just south of 36.85 degrees latitude, which may be indicative of a tilted, uplifted block beneath Yucca Mountain.

4.6 Conclusions

The inversion for depth of a density contrast, although nonunique, can be constrained with the added information of borehole or other geologic information to produce a quasi-unique solution (it can only be truly unique with infinite surface data density). Despite the relatively large number of model parameters and the computational cost of the objective function evaluation, the inverse problem can be readily solved using a GA. As found in chapter 3, tournament selection combined with a very low mutation rate produces the most efficient search.

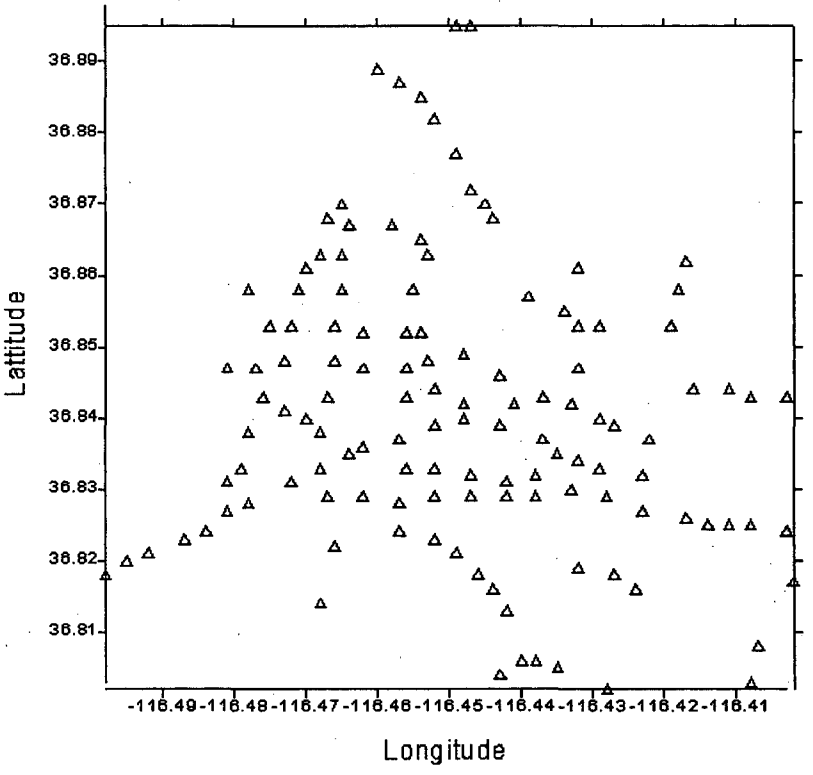


Figure 4.1. Gravity stations used in the experiment.

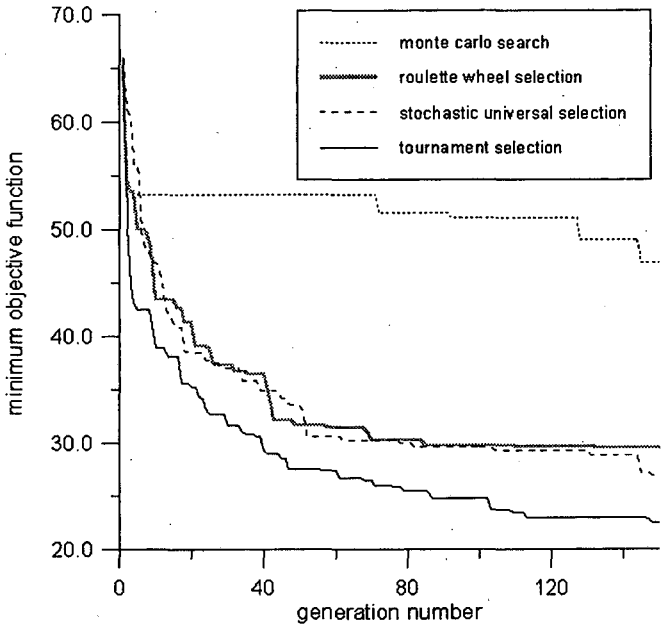


Figure 4.2. Convergence for gravity problem using a genetic algorithm with 3 different selection schemes and a Monte Carlo search.

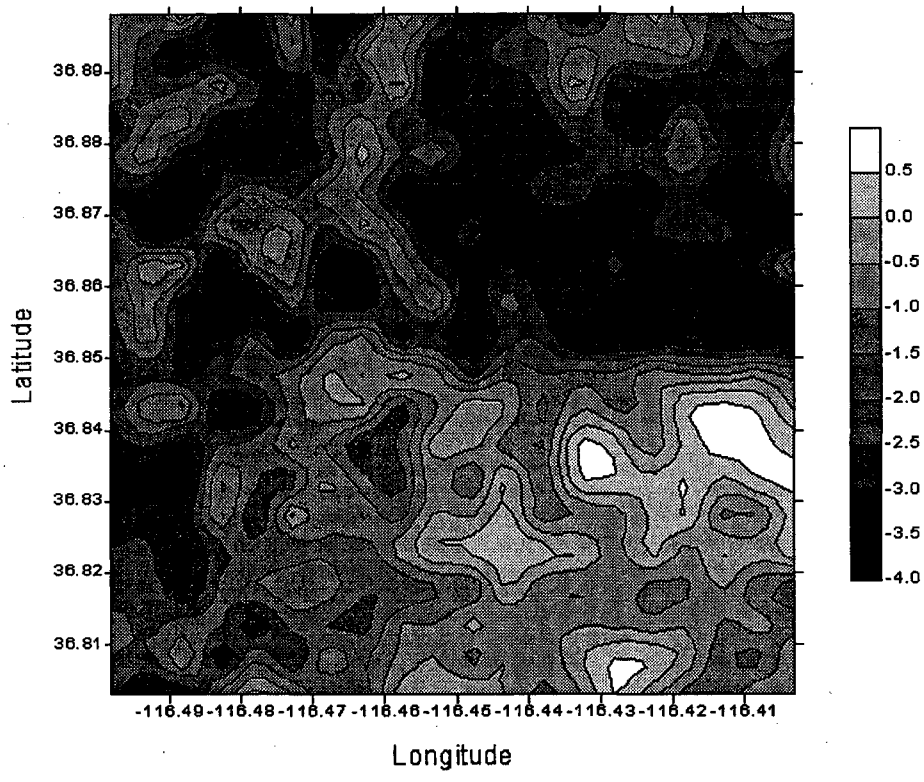


Figure 4.3. Contour plot of the estimated basement depth from inversion of gravity data from the stations in Figure 4.1.

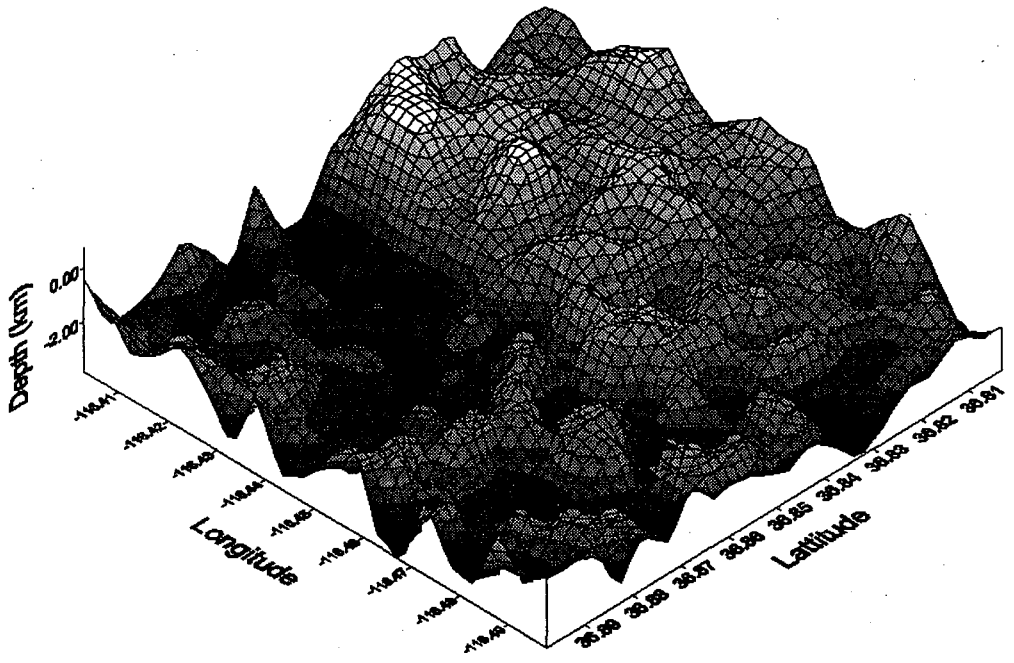


Figure 4.4. Surface plot of estimated basement topography from inversion of gravity data from the stations in Figure 4.1. Note that the angle of vision is rotated relative to Figure 4.3 in order to make the features visible.

Chapter 5

An example from seismology: New Zealand receiver function inversion

5.1 Introduction

Receiver functions (Langston, 1979) provide a way to model the velocity structure below a seismographic station using a single three component recording. The inversion of receiver functions involves estimating velocity, density, and quality factor ("Q") for various depths. Typically, density and Q are related to the velocity using empirical relationships in order to reduce the degrees of freedom, but the problem is inherently nonunique due to the depth/velocity tradeoff. The objective function evaluation consists of calculating a synthetic seismogram for a layered structure, deconvolving the vertical component from the radial to generate the synthetic receiver function, and calculating the sum of the squares of the residuals between the empirical and synthetic receiver functions. The problem is moderate in terms of the objective function calculation (about 0.5 seconds on a 366 MHz Sun Enterprise 4000) and in the number of free parameters (usually about 10-40, depending on desired resolution). Genetic algorithms are an efficient and reliable method for solving this type of nonlinear problem. In this chapter a genetic algorithm is used to invert for the crustal structure of the South Island of New Zealand by fitting empirical receiver functions.

5.2 New Zealand Tectonic and Geologic Background

Oblique convergence of the Pacific and Australian plates manifests itself as approximately 39 mm of strike slip displacement and 12 mm of convergence annually (Walcott, 1997). Since the Oligocene about 450 km of strike slip motion, 90 km of Pacific plate subduction and 25 km of uplift caused by the deformation of the overlying Australian plate have transpired. The major transform feature of the South Island, the Alpine Fault, roughly connects the region of the Pacific plate subduction at the Hikurangi trough in the Northeast with the region of continental collision at the Puysegur trench at the Southwest part of the Island. The Chatham rise to the east of the North-central part of the South Island is a region where the Pacific plate becomes near continental in thickness, thickening to about 27 km compared to an average of about 15 km (Reyners and Cowan, 1993). There is some question whether this thickened crust is subducted completely or if a new plate contact is formed in the subducted layer. Recent work by Eberhart-Philips and Reyners using earthquake hypocenters show the subducting plate dipping to the Northwest in this region, roughly parallel to the strike of the Alpine Fault. The slab dips almost vertically at about 100 km depth, probably due to a change in the plate density between the shallow and deeper sections of the plate (Eberhart-Philips and Reyners, 1997).

An interesting result of the tectonic regime can be seen at the surface as the Alpine Fault is approached from the East. Outcrops of increasingly higher grade of metamorphosed upturned schists can be seen which were accreted from as deep

as 25 km, exposing a slice of almost an entire crustal section perpendicular to the Fault.

5.3 SAPSE Project Background

The Southern Alps Passive Seismic Experiment (SAPSE) is a cooperative project that was founded during a 1993 NSF science workshop in New Zealand. The original objective of the workshop was to discuss plans for two wide angle reflection/refraction profiles across the South Island. This project, SIGHT (South Island Geophysical Transect), focused on the middle of the South Island, where the plate boundary manifests itself as a strike slip fault with mountainous uplift along its southeast side. It became obvious at the workshop that earthquake data would be an important complement to the active source SIGHT experiment.

The goal of the project is to gain a broad view of the seismicity and the 3-D inhomogeneity in the lithosphere of the South Island in order to understand how deformation from the obliquely convergent plate boundary is accommodated along the Alpine Fault in the Southern Alps. This can be achieved through the completion of six specific objectives:

1.) Microearthquake hypocenter location and source characteristics with acceptable precision to characterize the state of stress and seismicity along the Alpine Fault.

2.) Determination of three dimensional P and S wave velocity structure in the crust through travel time residual tomography and receiver function inversion.

3.) Determination of crustal thickness and the degree of isostatic compensation in the Alpine range through receiver function inversion.

4.) Analysis of the variation of crustal shear wave velocities through surface wave tomography.

5.) Exploration of the anisotropy in the lower crust and upper mantle through a combination of surface wave tomography and a study of variations in teleseismic S wave polarization.

6.) Investigation of regional scale lithospheric anomalies in areas such as the Chatham rise, Lord Howe rise, and Macquarie ridge regions.

The second and third objectives are addressed in this paper. The approach used to model the crust three dimensionally involves 1-D receiver function inversions under each station in order to estimate the 3-D structure.

The SAPSE array consisted of twenty-six broadband, fourteen 1 Hz stations and seventeen permanent New Zealand 1 Hz stations (see Table 5.1). All of the stations were in operation from November 1995 to April 1996. The spatial distribution of the stations was slightly weighted toward the central Alpine Fault in order to better understand the 2-D transect results as they relate to the broad scale 3-D structure of the crust and upper mantle. The sensors at each broadband site were coupled to solid rock with cement. The Reftek recorders, batteries, power board, and removable hard disk for each station were contained in steel

cages to protect them from New Zealand's alpine parrot, the Kea. Figure 5.1 shows the South Island and the source lines for the onshore/offshore experiment, along with the broadband stations that were used in this paper for receiver function analysis.

Stn. code	Stn. name	Latitude	Longitude	Elevation (m)
ABUA	Abut Head	-43.1467	170.4625	97
ARPA	Arthur's Pass	-42.9749	171.5787	710
BERA	Berwen	-44.5294	169.8838	505
BLBA	Blackbirch	-41.7139	173.8774	253
CHTA	Chatham Island	-43.7712	176.5808	178
CLAA	Clarks Junction	-45.7902	170.0437	423
CLIA	Clinton	-46.2916	169.3135	352
DENA	Denniston	-41.7449	171.8053	669
DOTA	Doubtful Sound	-45.5299	167.2723	255
EWZA	Erewhon	-43.5100	170.8526	625
GLAA	Gillespies Bch.	-43.4219	169.8476	25
GLEA	Glenorchy	-44.8729	168.4081	460
HOKA	Hokitika	-42.7411	171.0915	240
JACA	Jackson Head	-43.9688	168.6094	50
KAHA	Kahutara	-42.4176	173.5415	72
LAMA	Lake Moeraki	-43.7122	169.4642	93
LATA	Lake Taylor	-42.7811	172.2690	640
LUDA	Lauder	-45.0335	169.6871	382
LUMA	Lumsden	-45.7288	168.4494	290
MAKA	Makaroa	-44.2504	169.2229	335
MAYA	Mayfield	-43.7454	171.3694	530
MTCA	Mt. Cook	-43.7342	170.0913	859
MTJA	Mt. John	-43.9856	170.4649	1042
QRZA	Quartz Range	-40.8253	172.5299	294
SHEA	Sheffield	-43.3914	171.8801	450
TIMA	Timaru	-44.3825	171.0789	243
TOPA	Tophouse	-41.7627	172.9053	754

Table 5.1. SAPSE broadband stations.

5.4 Receiver Functions

Teleseismic P wave coda contains information about the crustal structure below the recording site in the form of P-S conversions at each interface below the surface. The effects of the source can be effectively removed by using a source equalization procedure as prescribed by Langston (1979), which consists of deconvolving the vertical component from the radial.

The three components of response at a station due to a teleseismic P-wave are

$$X_V(t) = I(t) \otimes S(t) \otimes E_V(t) \tag{5.1}$$

$$X_R(t) = I(t) \otimes S(t) \otimes E_R(t) \tag{5.2}$$

$$X_T(t) = I(t) \otimes S(t) \otimes E_T(t) \tag{5.3}$$

where $I(t)$ is the impulse response of the instrument, $S(t)$ is the source function, $E(t)$ is the Earth's response, V, R, and T represent the vertical, radial and transverse components, respectively, and the symbol $\otimes$ signifies convolution.

Langston (1979) shows that for the vertical component the Earth's response is approximately a delta function:

$$E_V(t) \approx \delta(t) \tag{5.4}$$

and therefore the vertical component of response is approximately equal to the instrument response convolved with the source function, which are exactly the factors that have to be removed in order to isolate the Earth's response $E(t)$. The radial and tangential receiver functions are defined as

$$f_R(t) = X_R(t) \otimes X_V^{-1}(t) \tag{5.5}$$

$$f_T(t) = X_T(t) \otimes X_V^{-1}(t) \tag{5.6}$$

or in the frequency domain,

$$f_R(f) = \frac{X_R(f)}{X_V(f)} \tag{5.7}$$

$$f_T(f) = \frac{X_T(f)}{X_V(f)} \tag{5.8}$$

Because the procedure requires a deconvolution, there is a high degree of sensitivity to small values of $X_V(f)$. Typically a frequency band is selected where $X_V(f)$ does not fall below a certain level, but the presence of noise can still cause division by zero problems in the above quotient, adding much extraneous information to the receiver function. Following Clayton and Wiggins (1976) and Owens (1984), a *water level* (similar to a optimal filter) can be used in the operation, so that the deconvolution is of the form

$$\frac{X_R(f)X_V^*(f)}{X_V(f)X_V^*(f)+\sigma^2} \tag{5.9}$$

here σ is a constant (usually between 0.001 and 0.0001) multiplied by the maximum spectral amplitude, and * refers to the complex conjugate. In addition, a Gaussian filter can be applied in the frequency domain to smooth the receiver function. In this case the deconvolution takes the form

$$\frac{X_R(f)X_V{}^*(f)}{X_V(f)X_V{}^*(f)+\sigma^2} F_G(f) \tag{5.10}$$

where

$$F_G(f) = \exp\left(-\frac{f^2}{\alpha^2}\right) \tag{5.11}$$

and α, the width of the Gaussian function, is usually between 1.0 and 5.0, representing a 1 Hz and 5 Hz low pass, respectively. Figure 5.2 shows Gaussian functions for a range of different α values.

Because of the removal of source effects and instrument response, receiver functions offer direct insight into the structure below the instrument. In some cases crustal thickness can be inferred to a reasonable degree of accuracy by simply looking at the receiver function, and Moho dip can be estimated from a series of functions from a group of stations. Figure 5.3 shows the receiver function for a single layer over a half space with an impedance contrast at the interface. In real media with many layers and attenuation the P-S phases are overwhelmingly dominant, a quality which makes them aesthetically appealing because each P-S phase corresponds to a conversion at an interface. The relationship between t_1 and z_1 in the Figure is simply

$$z_1 = \frac{t_1}{\left(\sqrt{\frac{1}{V_s^2} - p^2} - \sqrt{\frac{1}{V_p^2} - p^2}\right)} \tag{5.12}$$

where p is the ray parameter. Note that in this illustration the impedance contrast is positive with depth ($V_2 > V_1$). If the contrast were negative, the amplitude peak representing the P-S conversion (PS on the Figure) would be negative.

5.5 Inversion method

In the following synthetic seismograms are calculated with the reflectivity method (Kennett, 1983) and processed in the same manner that the data are to

generate the synthetic receiver functions. A Poisson relationship is assumed between V_P and V_S ($V_P / V_S = 1/\sqrt{3}$), and the density in the crust is approximated using the formula

$$\rho = 0.32V_P + 0.77 \tag{5.13}$$

(Berteussen, 1977, Ammon et al., 1990). A genetic algorithm is used to find the best fitting models.

5.5.1 Starting models

One of the subjective problems in modeling receiver functions is finding a starting model or, as is the case with genetic algorithms, to find a range of model space in which to search. One common approach is to use a number of thin layers of fixed thickness such that the actual medium can be reasonably approximated (the velocities of the thin layers take on constant values for regions of constant velocity, and will increase or decrease slightly where the actual layer thicknesses don't match the fixed layers). While this method provides very good fits to the data, there are some disadvantages. Because of the time/depth nonuniqueness, the models produced by this technique can be quite unstable (alternating high/low velocity layers), especially when fitting noisy data. Ammon et al. (1990) suggests a minimum roughness criteria which can be incorporated directly into the objective function. Another problem in modeling with a large number of layers lies in the computation time of the forward problem, although this is usually only

a problem when using a "randomized" inversion method like a GA, in which there is a vast number of function evaluations.

Another approach to the problem is to use the receiver function that is being fit to find an approximate starting model. This can be done with a bootstrapping technique. An exact inversion of the receiver function is not possible because of its simultaneous nonlinear dependence on both time and amplitude (Shibutani et al., 1996). However, if some a priori velocity information is available it is possible to estimate a rough starting model. Because each positive peak on a receiver function corresponds to an increase in impedance (at least in the first few seconds where multiples do not dominate), one can invert the travel times to obtain a velocity model using the following formula relating depth to time on a receiver function:

$$z = \frac{\Delta t}{\left(\sqrt{\frac{1}{V_S^{\,2}} - p^2} - \sqrt{\frac{1}{V_P^{\,2}} - p^2}\right)} \tag{5.14}$$

where z is the depth below the station, and Δt is the time beyond $t = 0.0$ on the receiver function. Naturally, a model must be assumed to determine the layer depths. An initial crustal and final mantle velocity estimated from the refraction data are assumed and velocity steps are interpolated in between. Because the inversion is inherently nonunique it is beneficial to make use of any *a priori* information to exclude unrealistic models. The model space searched by the algorithm consists of this starting model $\pm 30\%$ (in both velocity and layer depths).

The starting model generated with this method is far from perfect, mainly because it is derived from a receiver function that has already been filtered (due to processing and also the effects of the Earth) but it must be understood that it is only a starting model space in which to search.

5.5.2 Nonuniqueness

The travel time difference between a P to S conversion from a depth z and a direct arrival can be expressed as

$$\Delta t_{P->S} = z\left(\sqrt{\frac{1}{V_S^{\,2}} - p^2} - \sqrt{\frac{1}{V_P^{\,2}} - p^2}\right) \tag{5.15}$$

Taking the total differential of the above expression gives

$$d\left(\Delta t_{P->S}\right) = -\frac{z}{V_S^{\,2}} dV_S + \frac{z}{V_P^{\,2}} dV_P + \left(\frac{1}{V_S} - \frac{1}{V_P}\right) dz \tag{5.16}$$

For a fixed time lag $d\left(\Delta t_{P->S}\right) = 0$, and if we assume a Poisson solid we get

$$-1.732\frac{dV_S}{V_S} + \frac{dV_P}{V_P} + 0.732\frac{dz}{z} = 0 \tag{5.17}$$

This equation demonstrates the inherent time-depth nonuniqueness of the receiver function. The solutions to this equation for a fixed travel time are spread out in a plane. Even if a relationship between V_S and V_P is assumed there still exist an infinite number of solutions that fall along a line. Unique solutions can only be obtained if a relationship between V_S and V_P and either the velocity structure (for either V_S or V_P) or the layer thicknesses are known. This information is seldom

available, but one can minimize the range of possible nonunique solutions by incorporating some sort of a priori knowledge into an inversion.

5.6 Synthetic Tests

The inversion method is first tested with synthetic data, by attempting to invert for the velocities of layers with fixed thickness for three cases:

1. Noise free synthetic data (the performance under ideal conditions)
2. Synthetic data with 10 and 20% noise.
3. Synthetic data with 20% noise and an additional constraint in the objective function that penalizes models with layers that decrease in velocity with depth, i.e. the objective function is given by

$$f = f_o * \sum_{i=1}^{n} \frac{\alpha_{i+1}}{\alpha_i} \tag{5.18}$$

where f_o is the objective function without the additional constraint, α_i the P wave velocity for each layer, and n the number of layers.

The model used to construct the synthetic seismograms in these tests is the same in all cases, a simple six layer case in which velocity increases gradually with depth:

depth (km)	velocity (km/s)
0.0	4.0
10.0	4.5
20.0	5.5
30.0	6.0
40.0	6.5
50.0	7.5

The model space searched by the algorithm consists of this initial velocity model $\pm 20\%$. The number of bits per parameter is 16, so the number of different possible models is $(2^{16})^6 \approx 7.9228 \times 10^{28}$, which is unnecessarily high resolution for this problem but does not affect the computation time of the genetic algorithm (in this particular problem the bottleneck is the objective function evaluation, any computation time used by the genetic algorithm is insignificant). White uncorrelated noise is added in the time domain to the original traces with an rms amplitude equal to 10 and 20% of the peak amplitude of the trace. Figure 5.4 shows the correlation coefficients between a noise free receiver function and receiver functions with 0-30% noise added. The correlation is still above 70% for the case of 30% noise. All of the runs consist of 100 generations of 150 models. The results, summarized in Table 5.2, show that the inversion is quite robust even in the presence of noise. The results for case (3) are surprising: adding the additional constraint produces a model misfit that is roughly 1/3 of the case with no additional constraint. This implies that adding a priori information to the inversion is crucial in the presence of strong noise. Figure 5.5 shows the fit for this case. Note that many of the peaks in the synthetic "data" are direct results of the added noise.

For a more realistic inversion, the layer thicknesses can also be left unknown so that the algorithm must search a much larger model space and find both depth and velocity. This is also a way to estimate the likelihood of obtaining nonunique solutions. Figure 5.6 shows six different, independent fits of the receiver function

after 300 generations. For these inversions the algorithm searched a space deviating from the original model by 20% in velocity and 20% in layer thicknesses. Note that V_s is still tied to V_p, but there is no other a priori constraint such as velocity increase with depth. The receiver function fits are fairly good, as might be expected, but it is surprising that the model fits are also fairly closely grouped (see Figure 5.6). In Figure 5.7 the data misfit for these runs is plotted against the model misfit, showing a linear trend. This indicates that the nonunique solutions may be constrained to a small region of the model space.

	0% noise	10% noise	20% noise	20% with constraint
minimum objective function	0.0018	1.2197	4.3663	4.3916
model misfit (%)	0.1569	1.1243	8.1308	2.4314

Table 5.2. Minimum objective function and percent model misfit for 0, 10 and 20% noise, and 20% noise with the constraint that velocity not decrease with depth.

5.7 Data

The broadband stations consisted of matched three component instruments which recorded data at 20 samples per second for the early part of the experiment and 50 samples per second for the latter part. With some exceptions, the collected data are found to have overwhelming long period noise, especially those collected during the winter. A combination of low attenuation in the crust and a high degree of microseismic noise were the major contributors to this effect (no station

on the South Island was more than 200 kilometers from the ocean, most were less than 100). The stations BERA, CLAA, EWZA, LAMA, LATA, QRZA, and SHEA produced records with acceptable signal quality for this analysis and they cover most of the interesting tectonic regions of the South Island (see Figure 5.1).

date	time	latitude	longitude	depth (km)	m_b	location
96/01/07	13:14:29.2	-6.963	155.872	33	5.5	Solomon Islands
96/01/10	22:36:02.6	-6.147	133.671	33	5.2	Aru Islands
96/01/11	03:51:35.1	-8.427	158.708	96	6.6	Solomon Islands
96/01/12	02:17:34.1	-23.191	170.775	33	5.6	Loyalty Islands
96/03/17	14:48:56.3	-14.686	167.247	164	5.8	Vanuatu Islands

Table 5.3. Locations and magnitudes for the 5 events used in the inversion.

The 5 events used to generate and stack receiver functions occurred in the Melanesia region, all within a 40 degree azimuthal swath and a distance of 26-39 degrees (see Table 5.3). The events were also chosen based on signal to noise ratio (these events occurred during the summer in New Zealand, when microseismic noise is low). Each trace is picked one second before the onset of the P phase and contains a total of 20 seconds of record. The traces are tapered with a cosine taper for the first 4% and the last 50% of the time series and are padded with zeros to obtain 1024 points. The receiver functions for each station-event are filtered in the frequency domain with a Gaussian filter of width $\alpha = 4.0$, while a water level with $\sigma = 0.001$ is used to remove low amplitude, high

frequency noise. The receiver functions are subsequently stacked at each station to improve the signal to noise ratio. The final stacked receiver functions are shown in Figure 5.8.

5.8 Inversion Results

The number of layers in each starting model was determined using the one step inversion technique described in section 5.5.1 and varied from 7-15, depending on the complexity of the empirical receiver function. Figure 5.9 shows the results of an initial experiment to determine an optimal population size for the genetic algorithm. The convergence in fitting the receiver function for station CLAA is shown for 5 different runs, each utilizing 10,000 objective function evaluations but different population sizes. Clearly, the run with a population of 100 gave the best results and therefore all of the inversions were made with a population size of 100 and 150 generations.

The final models show an apparent correlation between relief and crustal thickness with the thickest crust (40 km) found under the station EWZA (Erewhon), which is in the heart of the Southern Alps. The stations in the Eastern foothills of the Alps, such as BERA (Berwen) and SHEA (Sheffield) have crustal thicknesses of 33 and 37 km, respectively. The receiver function fits and the corresponding models for each station are seen in Figures 5.10 - 5.16. Note that the units for amplitude of the receiver function are 1/time. This is because the physical units of time cancel out in the deconvolution, but a unit of frequency is gained in the transformation back into the time domain.

Figure 5.17 shows a preliminary model from Stern et al. (1997) which is based on the onshore/offshore refraction data. The crust thickens significantly as the Alpine Fault is approached, due to both the angle of the subducting slab itself and the compressional uplift. Very strong reflections are observed at the interface between the amphibolite and the old oceanic crust. In fact, these reflections tend to be more prominent than those from the interface between the old oceanic crust and the mantle, suggesting at least a comparable impedance contrast. This hypothesis is also supported by the receiver function results, as the velocity increase for the amphibolite/old oceanic crust tends to be larger than that for the Moho interface at many of the stations such as BERA (Berwen) and LATA (Lake Taylor).

Little is known about the crustal structure near the Alpine Fault Zone at the Western edge of the island, but the receiver function for station LAMA (Lake Moraki) suggests that there is significant scattering in this region. The Moho depth could not be determined from the refraction study, and it is difficult to derive from the receiver function due to scattering, but it is most likely more shallow than other regions in the South Island, with a depth between 20 and 25 km. The most obvious impedance contrast on the model in Figure 5.13, at about 19 km depth, is attributed to the largest peak in the receiver function at about 5 seconds. Amplitude peaks that are likely due to Moho conversions tend to arrive later in the other receiver functions, between 6-7 seconds. Delayed phases observed on the Mt. Cook refraction line suggest a low velocity zone in the region of the Alpine Fault (Stern et al., 1997). It is possible that the large impedance

contrast is caused by the interface between a low velocity layer and the old oceanic crust in Figure 5.17, and that the Moho conversion is the amplitude peak arriving at about 6.5 seconds, in which case the crust could be as thick as 28 km.

Several anomalies in the models appear to be artifacts of poorly fit data, for example the first major peak after the direct P arrival on the receiver function for the station LATA (Lake Taylor, Figure 5.14) is overestimated, producing a step in the velocity profile at about 3 km depth. Also, the algorithm did not fit some of the extreme amplitudes of some of the receiver functions, but many of these features are clearly artifacts of the deconvolution process and their fit would produce unrealistic models.

Figure 5.18 shows the crustal thickness for each station plotted as a function of station elevation. A linear trend with a slope of about 4-6 is indicative of Airy type isostatic compensation (Fowler, 1990). In this case the thickness increases with elevation, but the slope is 35 and there is significant scatter. Referring to Figure 5.17, it can be seen that as the Alpine Fault is approached from the east, the crust thickens substantially due to the angle of the subducting Pacific plate. The trend in Figure 5.18 is much more likely to be attributed to this effect than to any isostatic compensation. In Figure 5.19, the crustal thickness is plotted as a function of the perpendicular distance from the Fault (moving towards the East only). The fit for this case is much better than that of Figure 5.18, indicating a much stronger correlation. If Airy type isostatic compensation is neglected, the slope for the least squares fit of these points, -0.66, is an estimate of the slope of

the Moho as the Alpine Fault is approached and it agrees reasonably well with the interpretations of Stern et al. (1997).

5.9 Conclusions

Receiver functions afford direct insight on shear wave velocity discontinuities beneath a seismic station using a single three component station event. By deconvolving the vertical component seismogram from the radial, source and receiver effects are removed leaving only information from P-S conversions. Using a standard genetic algorithm, receiver functions can be inverted to estimate crustal structure below a station using only a single station event. The solutions are nonunique due to a depth/velocity tradeoff which is similar in nature to what is seen in the gravity problem in chapter 4, but if a modest number of model parameters are being inverted for the nonuniqueness appears to be constrained to a small region of the model space.

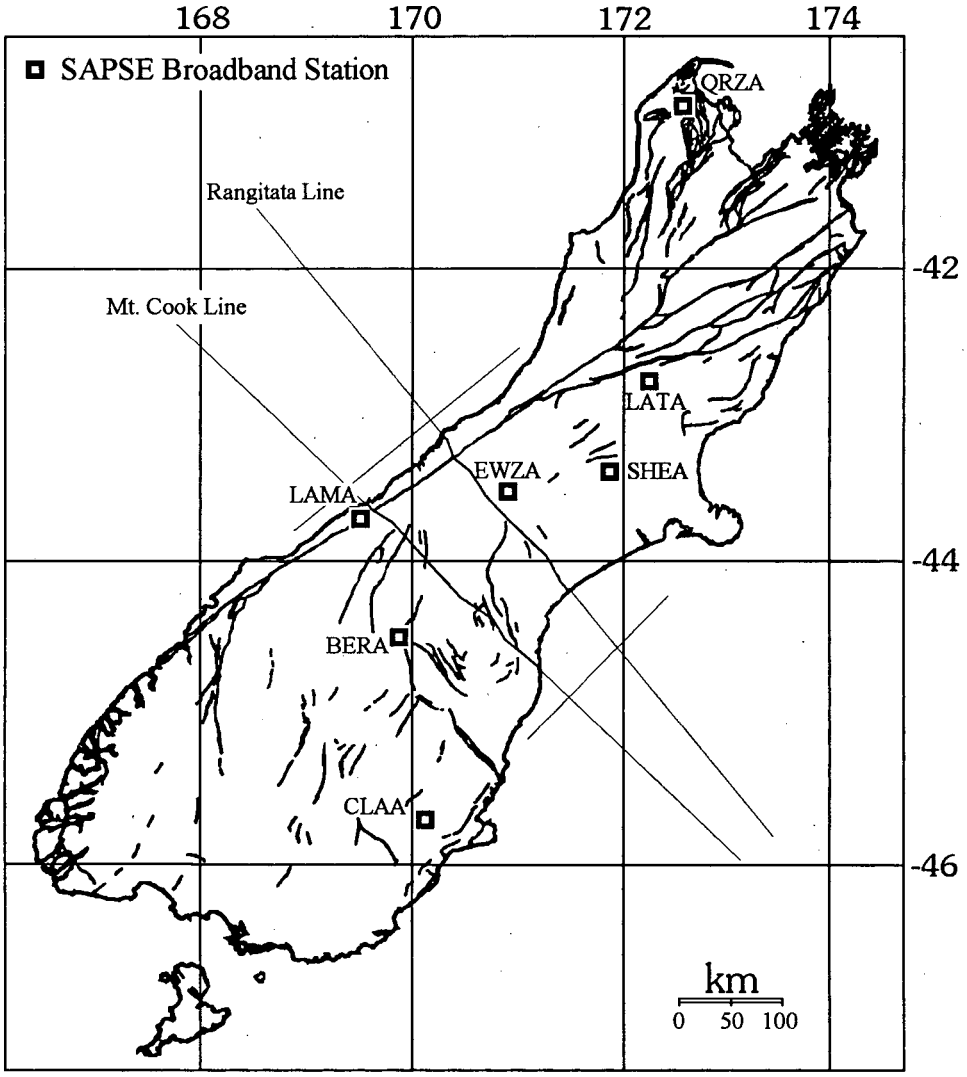


Figure 5.1. The South Island of New Zealand, showing the onshore/offshore refraction lines and the seven broadband stations used in this study.

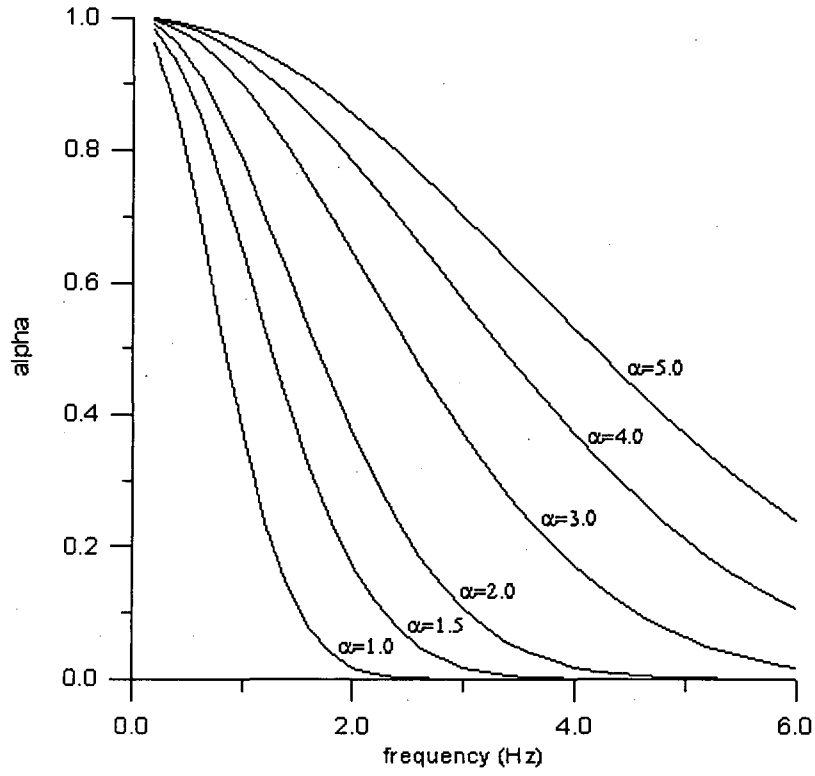


Figure 5.2. Gaussian filter coefficients for six different values of α.

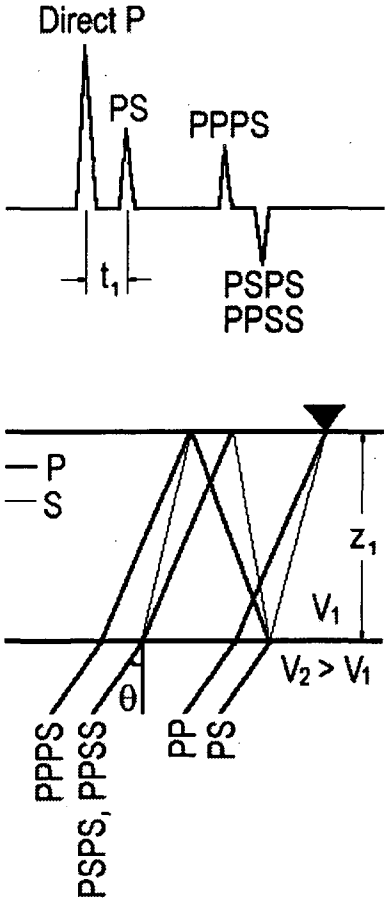


Figure 5.3. Significant direct arrivals and multiples seen on a receiver function.

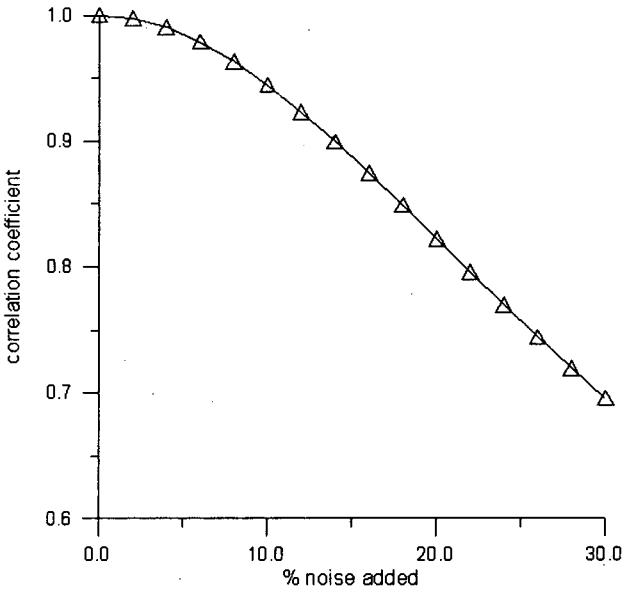


Figure 5.4. Correlation coefficients comparing noise free receiver functions with receiver functions contaminated with up to 30% noise.

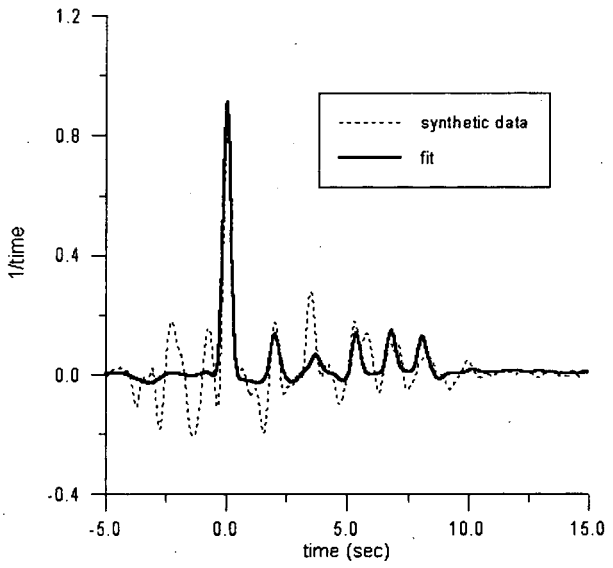


Figure 5.5. Fit of synthetic receiver function with 20% noise added to the original "data" seismograms, and with the additional constraint penalizing models having decreasing velocity with depth.

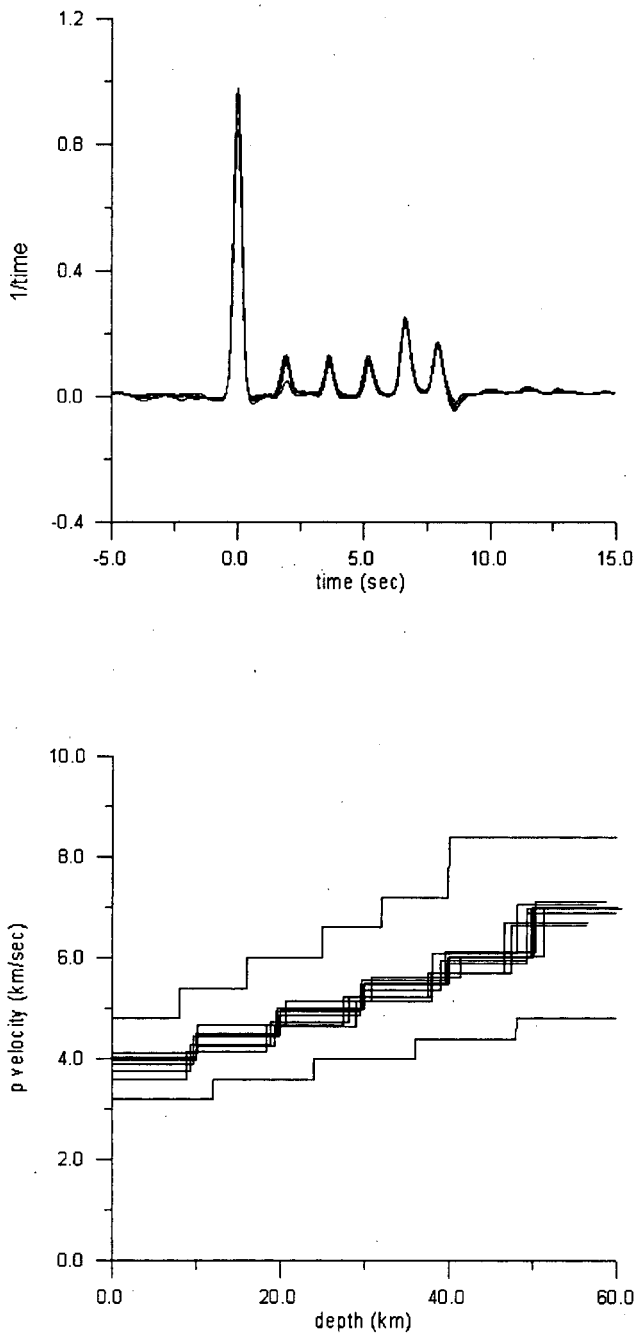


Figure 5.6. 10 independent receiver function fits inverted for layer velocity and thickness (above), and the 10 models produced by the fits (below).

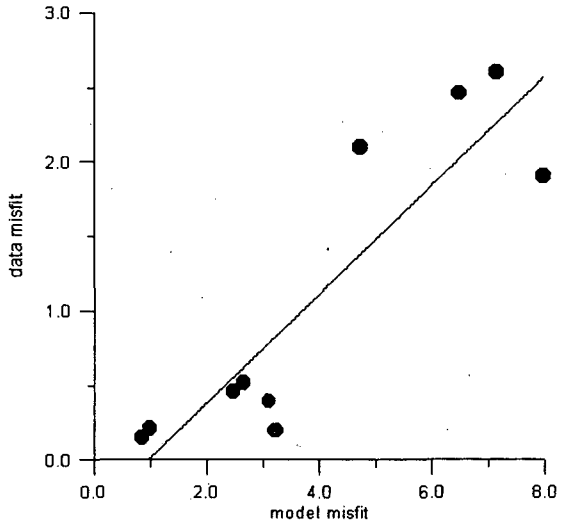


Figure 5.7. Data misfit plotted as a function of model misfit for the inversions in Figure 5.6. The trend shows that the two are roughly proportionate, indicating that the nonunique solutions may be constrained to a small region in the model space.

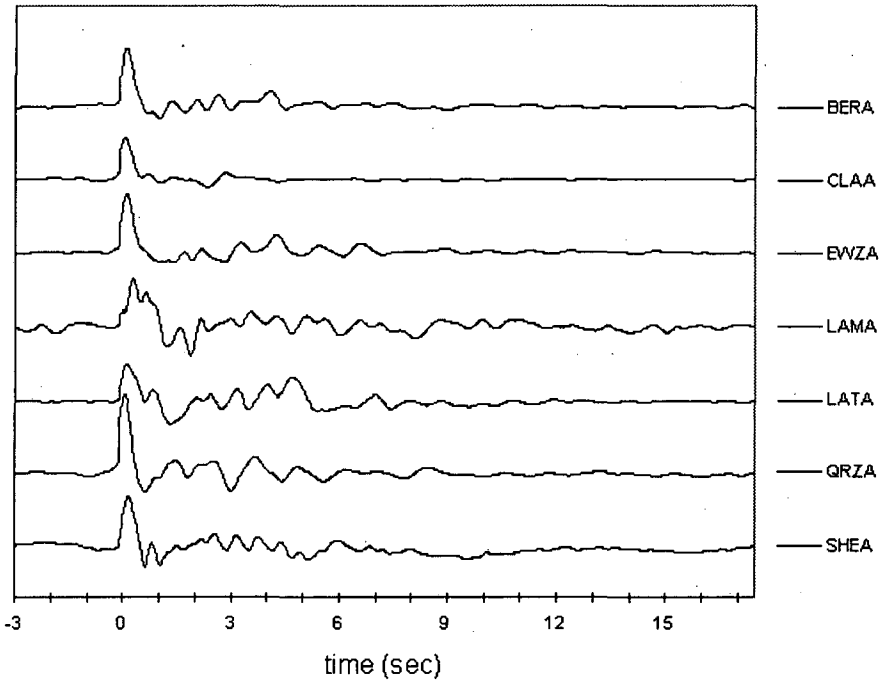


Figure 5.8. Stacked receiver functions for the seven broadband stations on the South Island.

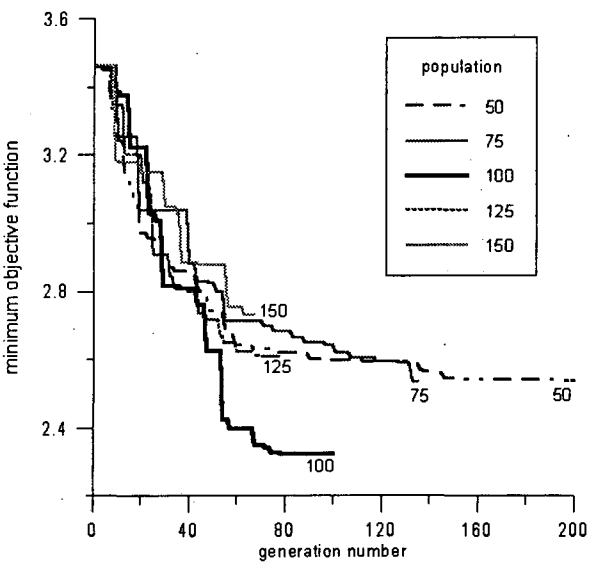


Figure 5.9. Convergence of five different runs for the New Zealand receiver function inversion, each with 10,000 objective function evaluations but varying populations.

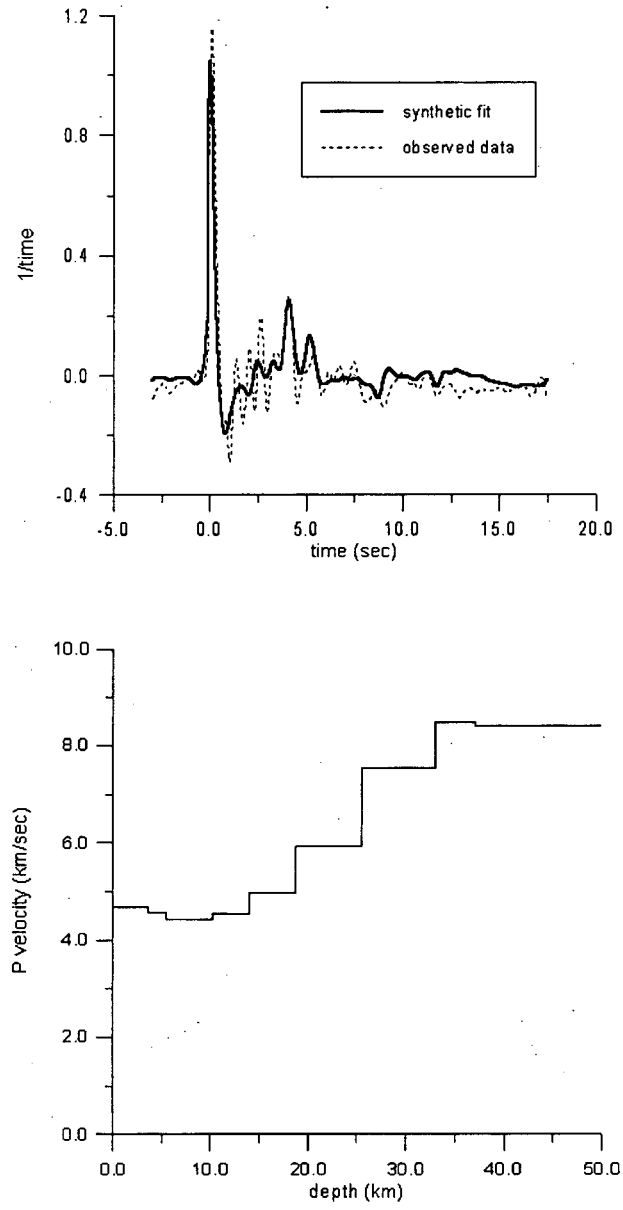


Figure 5.10. Receiver function fit for station BERA (Berwen) (above) and crustal model (below).

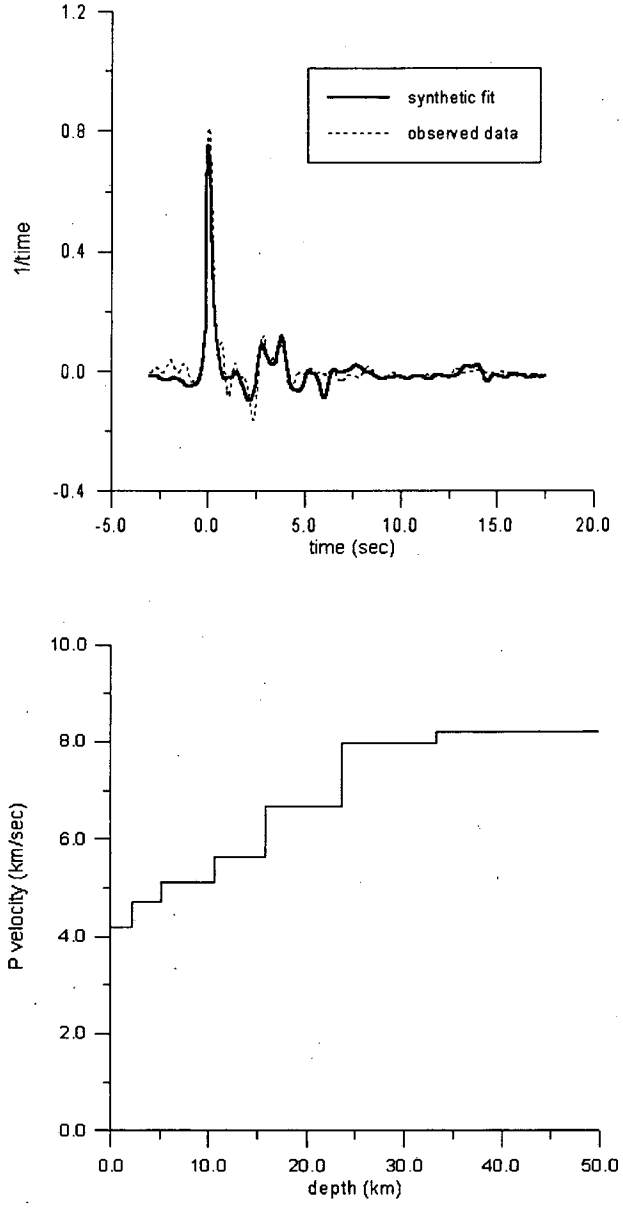


Figure 5.11. Receiver function fit for station CLAA (Clarks Junction) (above) and crustal model (below).

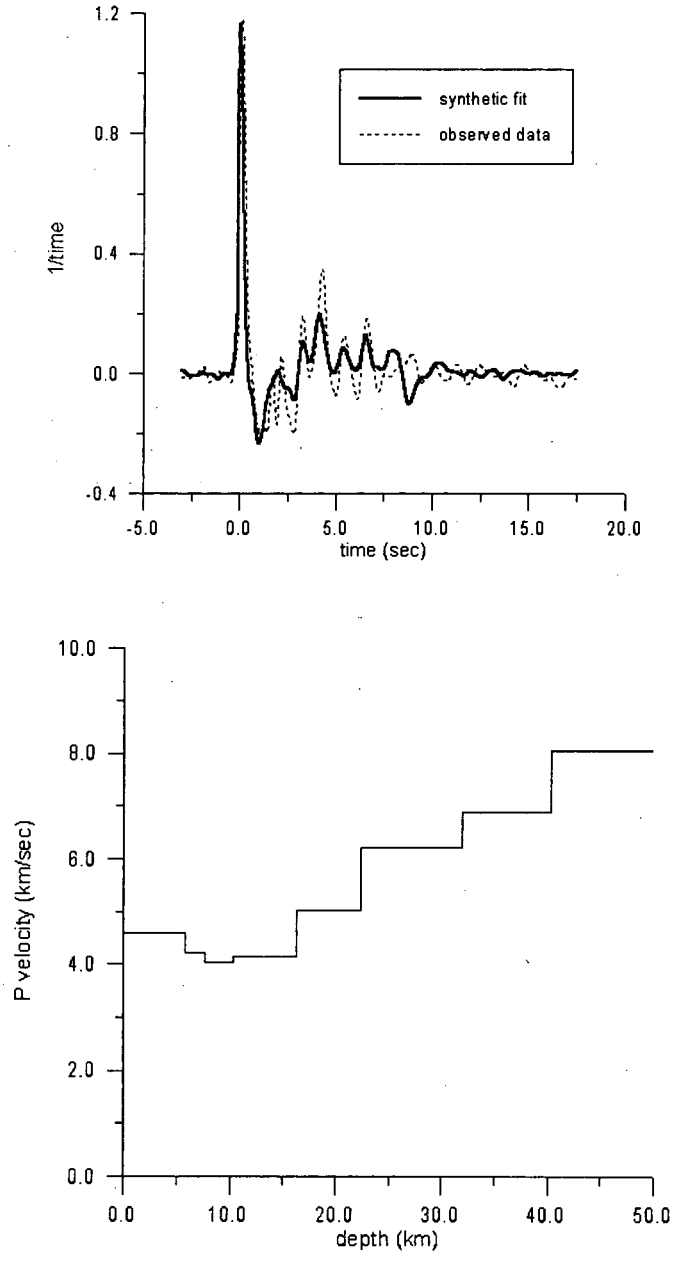


Figure 5.12. Receiver function fit for station EWZA (Erewhon) (above) and crustal model (below).

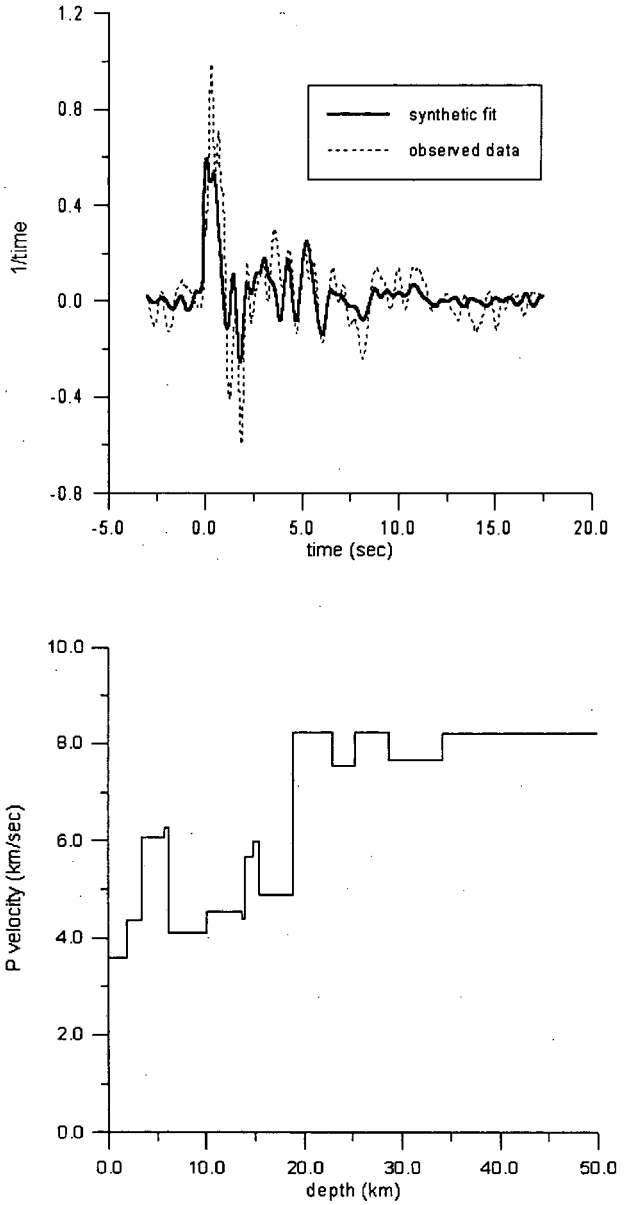


Figure 5.13. Receiver function fit for station LAMA (Lake Moeraki) (above) and crustal model (below).

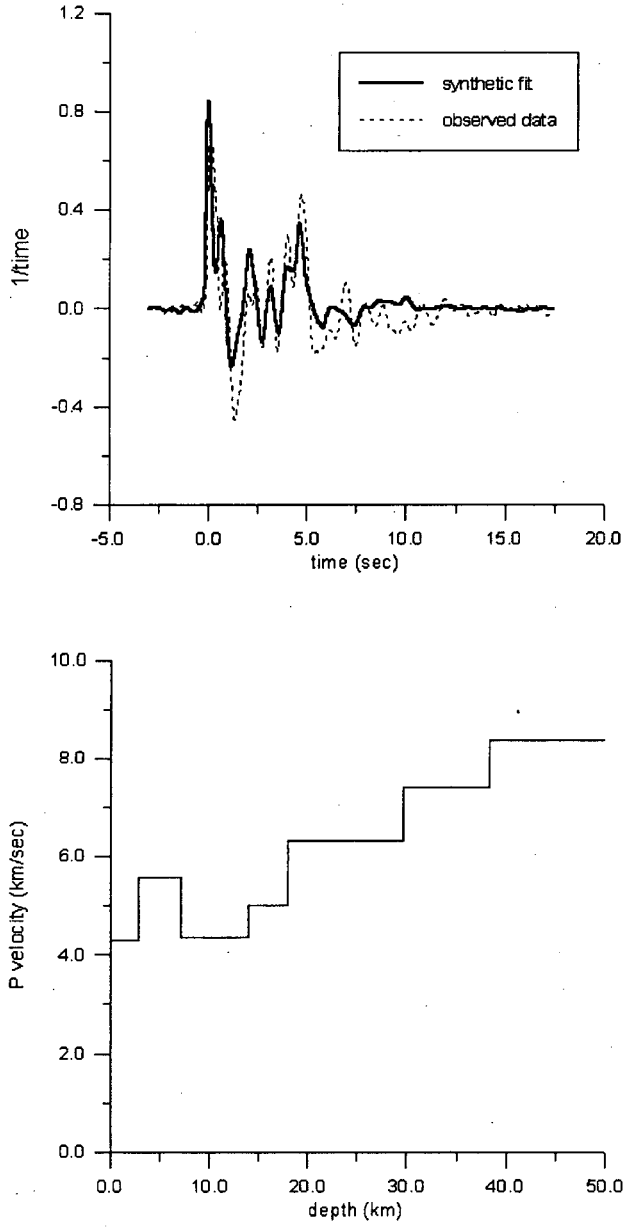


Figure 5.14. Receiver function fit for station LATA (Lake Taylor) (above) and crustal model (below).

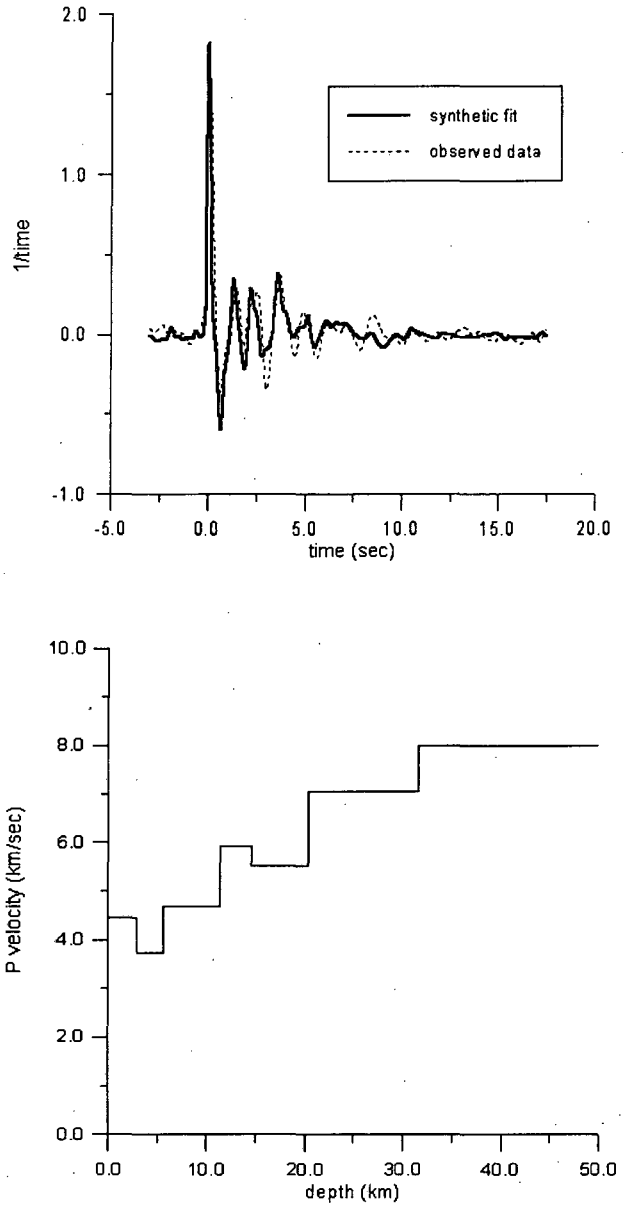


Figure 5.15. Receiver function fit for station QRZA (Quartz Range) (above) and crustal model (below).

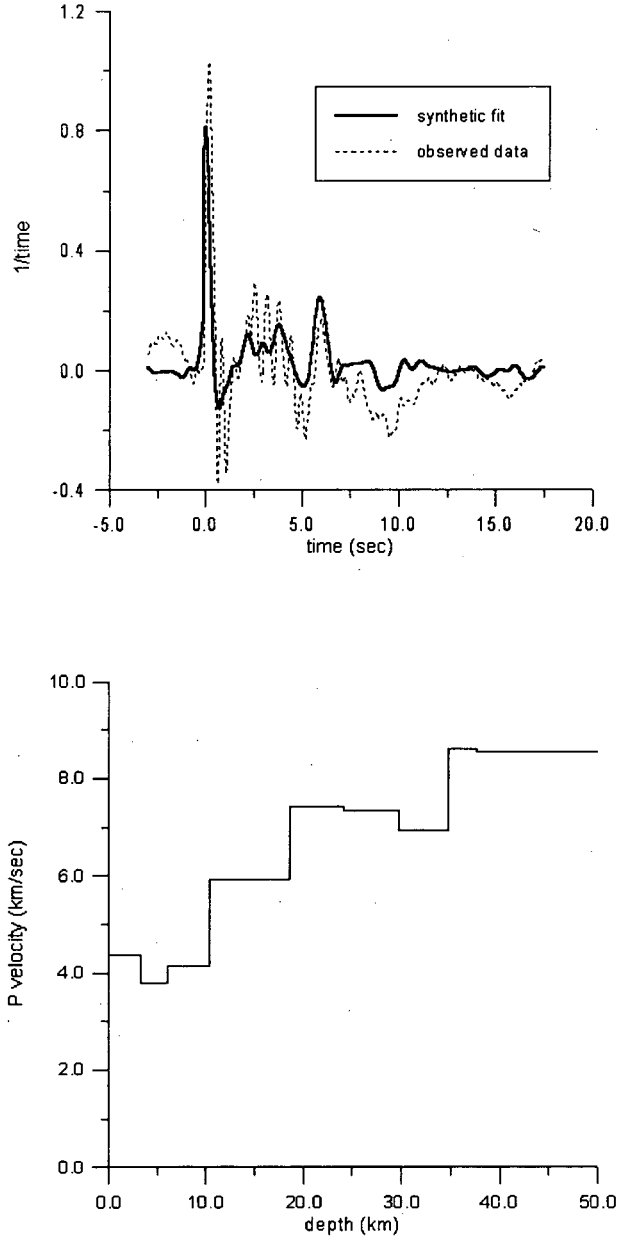


Figure 5.16. Receiver function fit for station SHEA (Sheffield) (above) and crustal model (below).

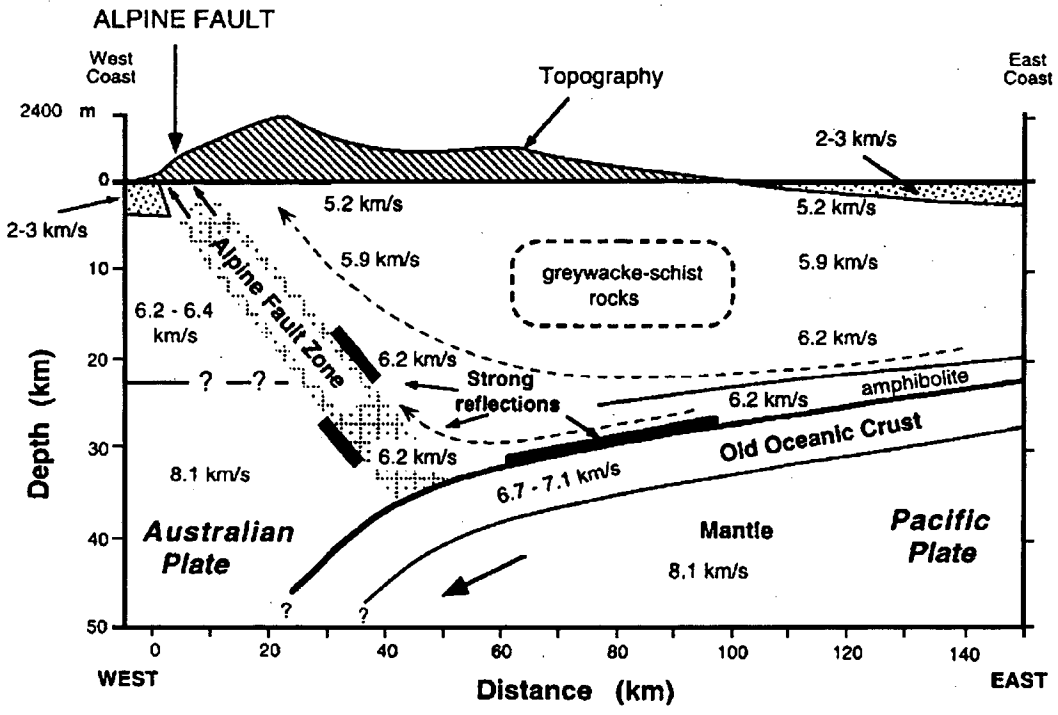


Figure 5.17. Preliminary crustal model derived from Mt. Cook onshore/offshore refraction line (modified with permission from Stern et al., 1997).

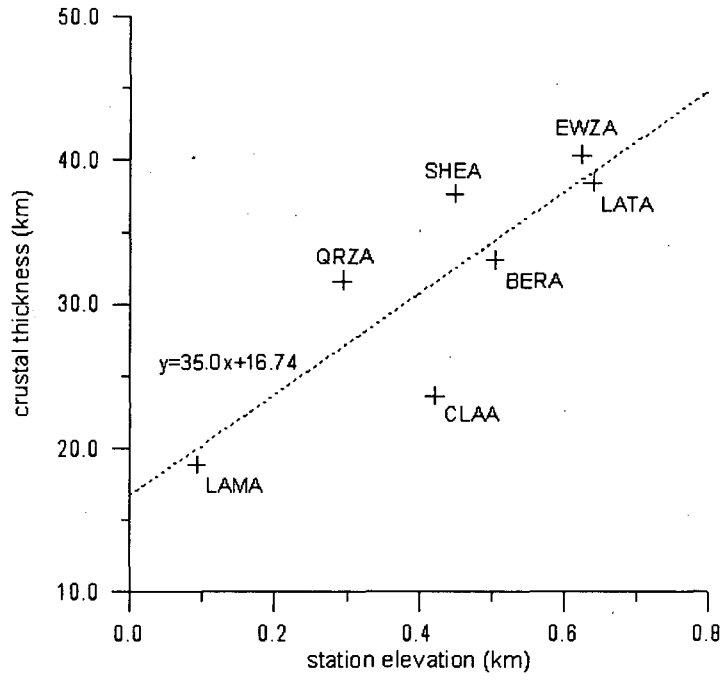


Figure 5.18. Crustal thickness for the 7 station models plotted against station elevation.

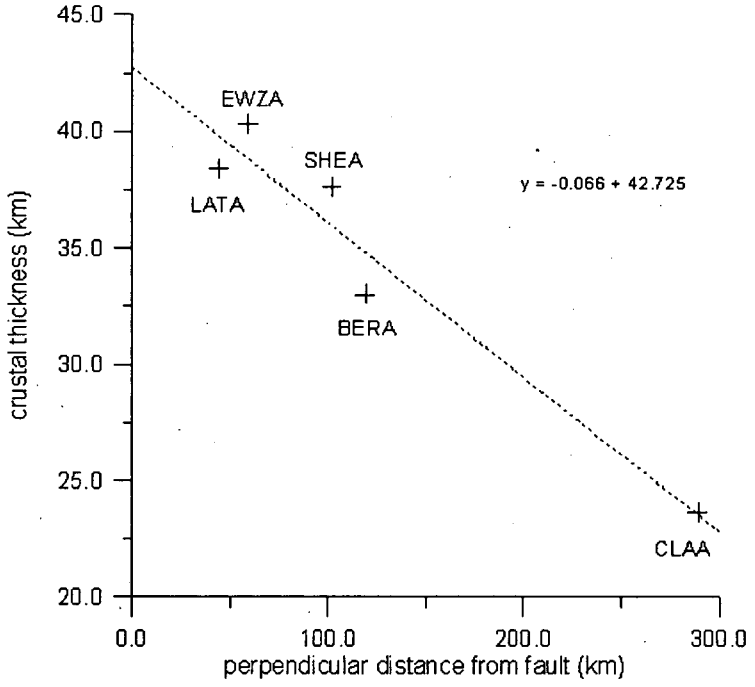


Figure 5.19. Crustal thickness plotted against the perpendicular distance from the fault, moving towards the East.

Chapter 6

An example from seismology: Mendocino Triple Junction receiver function inversion

6.1 Introduction

In this chapter a genetic algorithm is used to invert receiver functions to estimate the crustal structure of the Mendocino Triple Junction (MTJ) region in Northern California. Unlike the previous examples, this one utilizes multiobjective optimization by incorporating regularization into the objective function. Whereas in the previous chapter a single step inversion is used to obtain the starting model, in this case the starting model consists of a medium of many thin layers (~20-30). In order to best fit the data, the velocities of each layer will either increase or decrease (in the case of an amplitude peak on the receiver function) or take on constant values (in the case of constant amplitude). In practice this approach tends to be somewhat unstable because of the high number of degrees of freedom, typically generating models with alternating fluctuations between high and low velocity, so a minimum roughness criteria is added to the objective function. As with any multiobjective optimization, it must be stressed that adding additional constraints causes a tradeoff between the fit of the data and the smoothness of the model.

6.2 Mendocino Triple Junction tectonic and geologic background

The complex plate geometry of the Mendocino Triple Junction (MTJ) is the focal point of Northern California tectonics. Bounded by strike slip faulting to the West and South and subduction to the North, the junction itself has been moving to the North at a rate of approximately 5 cm/yr for the last 5.5 million years (Atwater, 1970). This northward migration (and the oblique convergence of the Juan de Fuca/Gorda Plate) are responsible for many of the significant tectonic events in North America, including (1) volcanism in the Northern Coast Ranges, (2) a broad zone of faulting and deformation in the Coast Ranges, and (3) extinction of arc volcanism to the North of the MTJ (Benz et al., 1992).

The North American Plate in the region of the MTJ is an accretionary complex of Mesozoic to Cenozoic origin (the Franciscan, manifesting itself in the Coast Ranges and Klamath Mountains) which is overlain by the Eel River Basin, a sedimentary forearc basin of Cenozoic origin (Beaudoin and Magee, 1994). Velocities in these crustal units are believed to be relatively uniform, in the range of 5.5-5.8 km/s in the West and 6.0-6.5 km/s in the East (Beaudoin and Magee, 1994; Benz et al., 1992). The subducting Gorda Plate is believed to vary in thickness from 7 km at the southern end of the MTJ to 10 km in the North, decreasing in velocity from 6.7 km/s to 6.2 km/s at the same time (Beaudoin and Magee, 1994).

Figure 6.1 is a schematic cross section of the standard model for the tectonic interaction in Northern California as proposed by Benz et al. (1992). The cross section is assumed to be valid for the region between 40 and 41 degrees latitude. The dip associated with the subducting Gorda Plate is a matter of much contention. The most reliable estimates come from east-west cross sections of seismicity, which define a Benioff zone that dips 10^{O} to the east at the coast and up to 25^{O} below the southern Cascades. Estimates of the velocities V_1, V_2 and V_3 are not as reliable as the estimates of Benioff zone dip. V_1 is likely the least uncertain, typically in the range of 6.3 to 6.5 km/s. V_2 is found to be as low as 6.7 km/s (Beaudoin and Magee, 1994) and as high as 8.0 km/s (Benz et al., 1992), and the underlying mantle velocity V_3 is usually estimated to be from 8.0 to 8.2 km/s.

Focal mechanisms and seismicity patterns imply a change in the orientation of stress on the Gorda Plate from compression in the North-South direction to down slab extensional beyond 236^{O} east (Dicke, 1998). An obvious consideration with a model in which $V_1 < V_2 < V_3$ is the lack of a driving force for this type of subduction. If $V_2 < V_3$, the Gorda Plate should also be less dense than the underlying mantle material and therefore more buoyant. It is possible that compressional forces associated with spreading from the Gorda Ridge are driving the subduction. In this case there should either be a thickening of the Gorda Plate as it is being subducted or some type of imbrication. Benz et al. suggest that the slab is imbricated in the MTJ region resulting in accreted slab fragments under the North American Plate, but Verdonck and Zandt (1994) find the plate to be intact.

6.3 Data and Processing

Receiver functions are calculated for 8 teleseismic events recorded at five Northern California Broadband stations in the Berkeley Digital Seismic Network (BDSN) (see Table 6.1). The stations and their spatial relationship to the MTJ can be seen in Figure 6.2. The teleseismic data used in this investigation consist of 7 events of magnitude 7.0 or greater between -15 and -28 degrees latitude and -173 and -179 degrees longitude, and the Bolivian M_0 = 8.2 Event of 1994 (see Table 6.2). The 7 South Pacific events have travel paths approximately perpendicular to the zone of convergence in the MTJ region, while the travel path of the Bolivian event is roughly parallel.

Station code	Station name	Latitude	Longitude	Elevation (m)
ARC	Arcata	40.877	-124.075	60
HOPS	Hopland	38.994	-123.072	299
MIN	Mineral	40.345	-121.605	1495
ORV	Oroville	39.556	-121.500	360
WDC	Whiskeytown	40.580	-122.540	300
YBH	Yreka	41.732	-122.710	1110

Table 6.1. Berkeley Digital Seismic Network (BDSN) broadband stations used in this study.

The data, sampled at 0.05 s, is filtered to remove microseismic noise below 0.15 Hz, cosine tapered and 25% zero padded before being deconvolved with a Gaussian filter coefficient of α = 2.0 to produce both radial and tangential receiver functions for each station-event. The receiver functions for each South Pacific event and the stacked functions are shown Figures 6.3-6.8, while those for

the Bolivian event are presented in Figure 6.9. Note that the final 5 seconds of each time series is "wrapped around" to the beginning purposely to give a clearer picture of the first arrival. Because the latter part of the time series was zero padded before deconvolution, the rms amplitudes for the first 5 seconds can give some insight into the amount of noise present.

date	latitude	longitude	depth(km)	M_0	location
06/09/94	-13.841	-67.553	631.3	8.2	Bolivia
03/09/94	-18.039	-178.413	562.5	7.6	South Pacific
04/07/95	-15.199	-173.529	21.2	8.0	South Pacific
07/03/95	-29.211	-177.589	35.3	7.2	South Pacific
08/05/96	-20.690	-178.310	550.2	7.4	South Pacific
09/20/97	-28.683	-177.624	30.0	7.2	South Pacific
10/14/97	-22.101	-176.772	167.3	7.7	South Pacific
03/29/98	-17.576	-179.061	536.6	7.2	South Pacific

Table 6.2. Locations and magnitudes for the 8 events used in the inversion.

There is strong similarity between the stacked and the individual traces in Figures 6.3-6.8. The correlation coefficients ranged from 0.611 to 0.949, with an average of 0.805, indicating a good correlation.

6.4 Estimation of Moho Depth by direct interpretation of receiver functions

As a first approximation of Moho depth below each station, a simple inversion is performed using equation 5.14 and assuming a constant crustal velocity of 6.5 km/s to find depths for the major amplitude peaks on the stacked receiver functions. On the receiver function for the station at Arcata (Figure 6.3), two

amplitude peaks are visible in the first 5 seconds, corresponding to depths of 21.9 km and 32.0 km, respectively. It is likely that the crust is thinnest in this region (Verdonck and Zandt, 1994; Benz et al., 1992), so the first amplitude peak is most likely caused by a P-S conversion at the Moho at 21 km depth. The receiver function determined from the station at Hopland (Figure 6.4) shows amplitude peaks corresponding to 26.4 and 37.1 km depth, and because this is still in the coast ranges 26.4 km is the most likely Moho depth. The receiver function for the station at Mineral (Figure 6.5) has a long, flat amplitude peak that begins at 31.1 km and begins to recede at 46.6 km. Sustained positive amplitude on a receiver function is associated with a gradual increase in velocity, so it is possible that there is a slower transition to Mantle velocities in this region. The receiver function determined for Oroville (Figure 6.6) has significant amplitude peaks corresponding to 31.4 and 39.7 km. Because it is in the Sierra foothills, is likely to overlie a fairly thick crust, but here the choice is not as obvious as with the previous ones. The receiver function calculated for Whiskeytown (Figure 6.7) has two obvious amplitude peaks which correspond to 34.2 and 48.0 km in depth. Lastly, the receiver function determined from the station at Yreka (Figure 6.8) has major amplitude peaks which correspond to depths of 32.6 km and 41.4 km, although the peak at 41.4 km is the largest of the two, implying that the conversion at 41.4 km has the largest impedance contrast.

6.5 Inversion method

Following Ammon et al. (1990) the approach that is used here is to model with many thin layers of fixed thickness. As previously discussed, modeling with too many thin layers can produce unstable solutions, which can be seen in Figure 6.10. The fit is very good, but the model has many alternating high and low velocity layers that may not be realistic. Because the problem is inherently nonunique it is desirable to minimize the number of degrees of freedom in order to find the simplest model that fits the data. To accomplish this, a minimum roughness constraint is added to the objective function, with the model roughness being calculated using an $\|\mathbf{L}_1\|$ norm:

$$X_r = \sum_{i=1}^{n} \left[\left(\alpha_{i+1} - \alpha_i \right)^2 \right]^{1/2} \tag{6.1}$$

where α_i represents the P wave velocity for each layer. The roughness parameter is then multiplied by a normalized weighting factor which is found through trial and error. A value of 0.4 is used as a weighting factor for the minimum roughness constraint and 0.6 for the receiver function fit for the inversions described in this chapter.

6.6 Inversion Results

The inversion results are shown in Figures 6.11-6.21. Figures 6.11-6.16 are the receiver function fits and models for the stacked South Pacific events and Figures 6.17-6.21 those for the Bolivian event. The amplitude peak

corresponding to the direct arrival on many of the South Pacific receiver functions cannot be fit adequately. Because the height of this amplitude peak is a function of the angle of incidence of the seismic wavefront, it is likely that dipping layers are producing an angle of incidence that is not as steep as the one that is used in the modeling. The travel path of the Bolivian event is roughly parallel to the zone of convergence, so this effect is not apparent for that event. Therefore, interpretations of the inversion results will be more heavily weighted towards the Bolivian data.

Inversions results for the coastal station of Arcata (Figures 6.11, 6.17) imply a gradual increase to upper mantle velocities occurring from 18 to 24 km depth. No consistent low velocity zone can be seen here. It would be interesting to look at receiver functions for a station at the same longitude as Arcata but south of the MTJ to see if there is a low velocity zone where Benz et al. (1992) suggest the existence of a slab window. Inversions results for the stations in the Southern Cascade Range are more interesting. The results derived for the station at Yreka (near the Oregon border, Figures 6.12, 6.18) show a steep initial velocity indicating a high velocity at about 5-8 km depth, followed by gradually increasing velocity to upper mantle values at 36-38 km depth. Inversion results for the station at Whiskeytown (Figures 6.13, 6.19) show a relatively constant velocity profile down to about 44 km in depth, where the velocity jumps to nearly 8 km/s. Another increase of approximately 0.75 km/s occurs on the Bolivian inversion (Figure 6.19) at 44 km depth. It is possible that this is the bottom of the Gorda slab, but it is probably more indicative of a gradual rise in velocity through the

upper mantle with depth. Inversion results for the station at Mineral (near Mount Lassen, Figures 6.14, 6.20) display a relatively constant velocity structure until 34-36 km depth, where there is a large increase to near mantle velocities, and a lesser increase at 40 km. The results for the station at Oroville (Figures 6.15, 6.21) suggest a crustal thickness of 44-46 km, which is consistent with the simple one step inversion result described above. There are no data for the Hopland station during the Bolivian event, but the South Pacific inversions suggest a crustal thickness of about 32 km (Figure 6.16). However, the inversion appears to be somewhat unstable, alternating between high and low velocities in an attempt to fit the high amplitudes of the peaks, which may be a result of a dipping layer geometry of the region.

In Figure 6.22, the results from section 6.4 are plotted in two dimensions for the purpose of comparison with existing subduction models of the MTJ region. Estimated depths derived from amplitude peaks on the stacked South Pacific receiver functions which are larger than 50% of the direct arrival peak and fall between 2.3 and 5.8 seconds (corresponding to roughly 20-50 km depth if a constant velocity of 6.3 km/s is assumed for the region) are plotted as a function of one dimensional distance from the MTJ. The results are consistent with the schematic in Figure 6.1 for the case where $V_1 < V_2 < V_3$. Assuming the lowest points correspond to conversions originating from the bottom of the Gorda Plate, the dip on the Gorda Plate can be estimated. Between the stations ARC and HOPS the dip is approximately 3.5^{O}, between HOPS and YBH it is approximately 8.0^{O}, and between YBH and WDC approximately 22.3^{O}. This agrees fairly

closely with the estimated dip of the Benioff zone in the region between YBH and WDC but is considerably lower than that for the dip of the Benioff zone near the coast. The two higher points representing conversions for the stations Oroville and Mineral have approximately the same depth at 31 km. If these are Moho conversions, the lower points could be conversions from slab fragments as suggested by Benz et al. (1992).

An alternative interpretation for the apparent thickening of the crust is simply the isostatic compensation for the Cascade Range. If this hypothesis is correct then the two lower points in Figure 6.22 for Oroville and Mineral may be more consistent with Moho depth and the two higher points could be the result of mid-crustal boundaries.

6.7 Conclusions

Empirical receiver functions contaminated by noise and/or affected by dipping layers can lead to unrealistic or unstable models with extreme velocity variance. In this case, multiobjective optimization can be used to add a minimum roughness constraint to the objective function. Some amount of experimentation is necessary to find a suitable tradeoff between data fit and model smoothness.

In addition to carrying out formal inversion procedures, direct interpretations of receiver functions can provide a great deal of information. Assuming a constant velocity in the crust and then calculating the depths which correspond to significant amplitude peaks on the receiver function can give a rough estimate of the depth and dip of the Moho and other discontinuities.

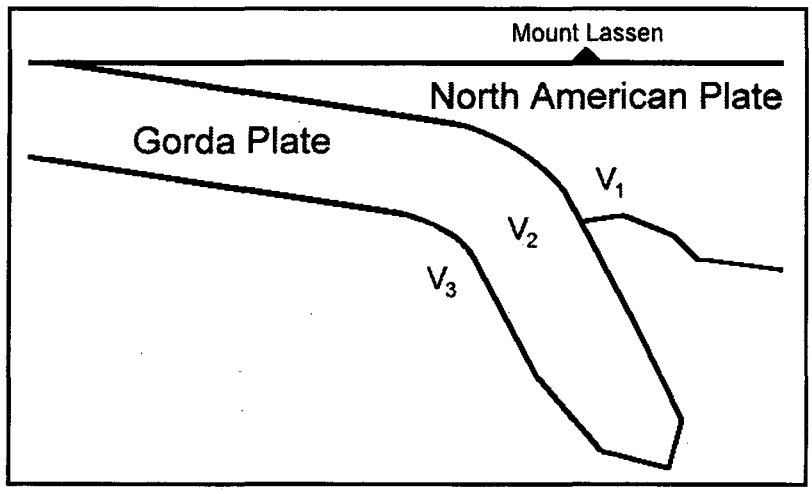


Figure 6.1. Schematic east-west cross section of standard model for the tectonic interaction in Northern California as proposed by Benz et al. (1992) (not to scale).

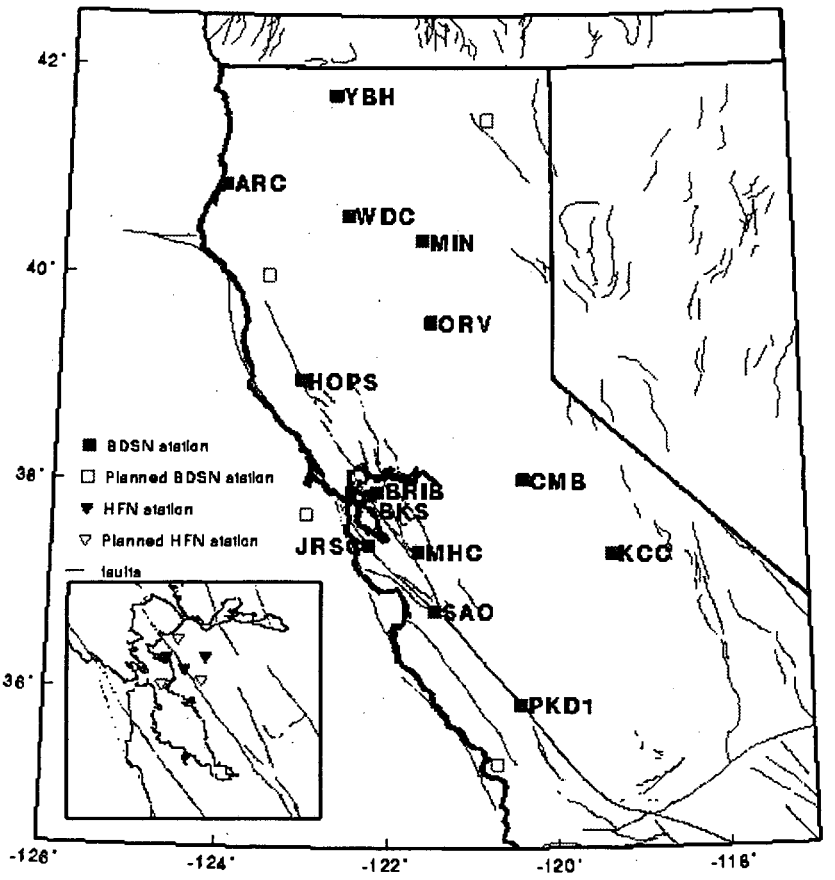


Figure 6.2. Berkeley Digital Seismic Network (BDSN) stations. YBH, ARC, WDC, MIN and HOPS were used in this study.

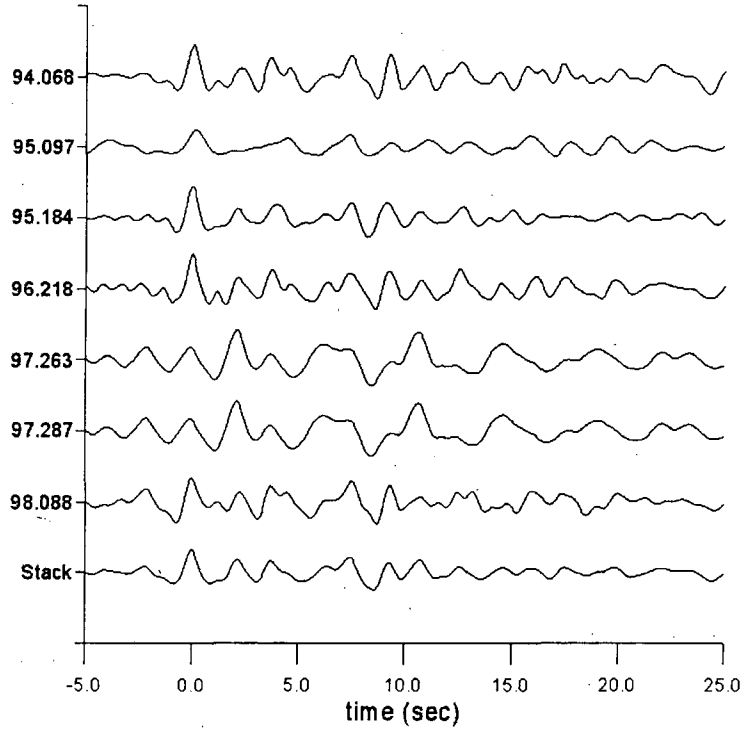


Figure 6.3. Radial receiver functions for station ARC (Arcata) from the 7 South Pacific events, with receiver function stack at bottom.

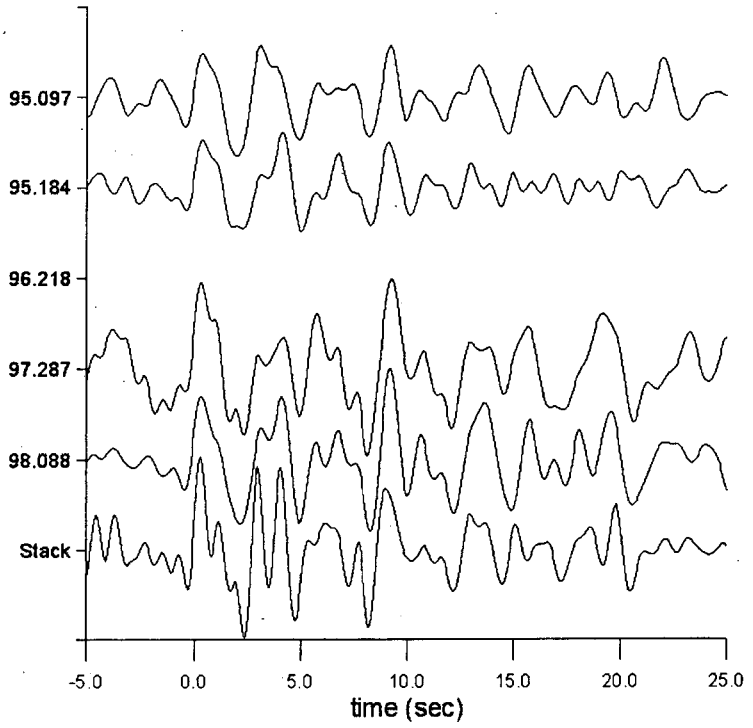


Figure 6.4. Radial receiver functions for station HOPS (Hopland) from the 4 South Pacific events, with receiver function stack at bottom.

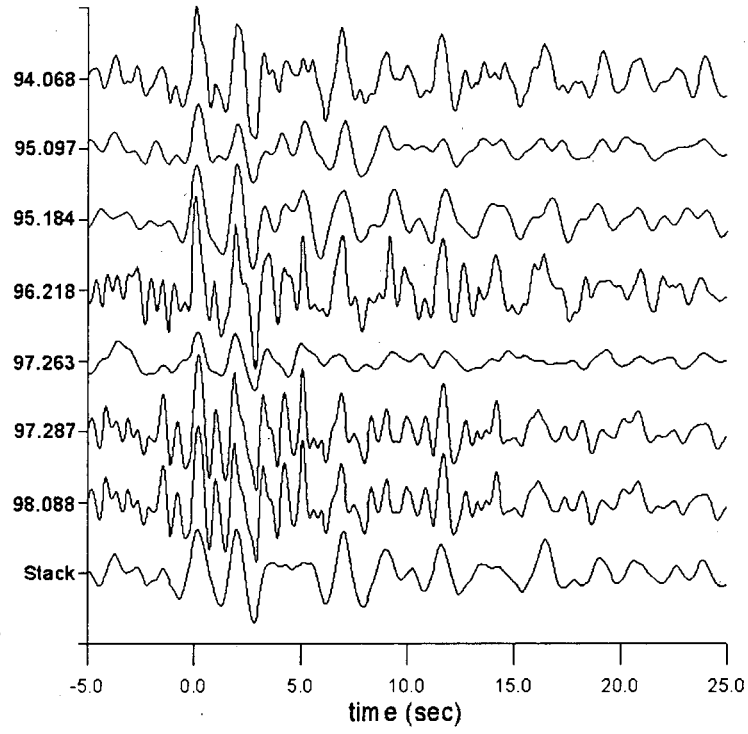


Figure 6.5. Radial receiver functions for station MIN (Mineral) from the 7 South Pacific events, with receiver function stack at bottom.

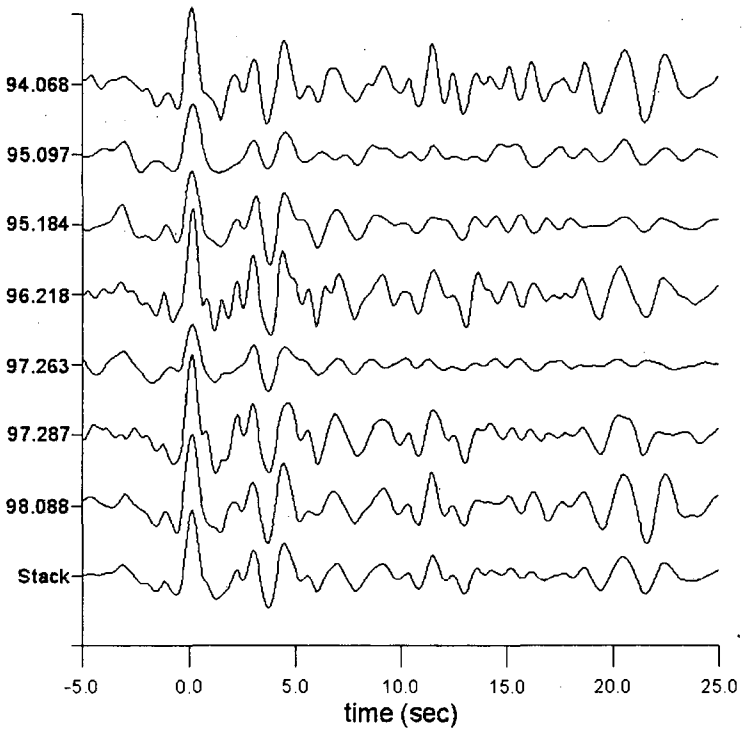


Figure 6.6. Radial receiver functions for station ORV (Oroville) from the 7 South Pacific events, with receiver function stack at bottom.

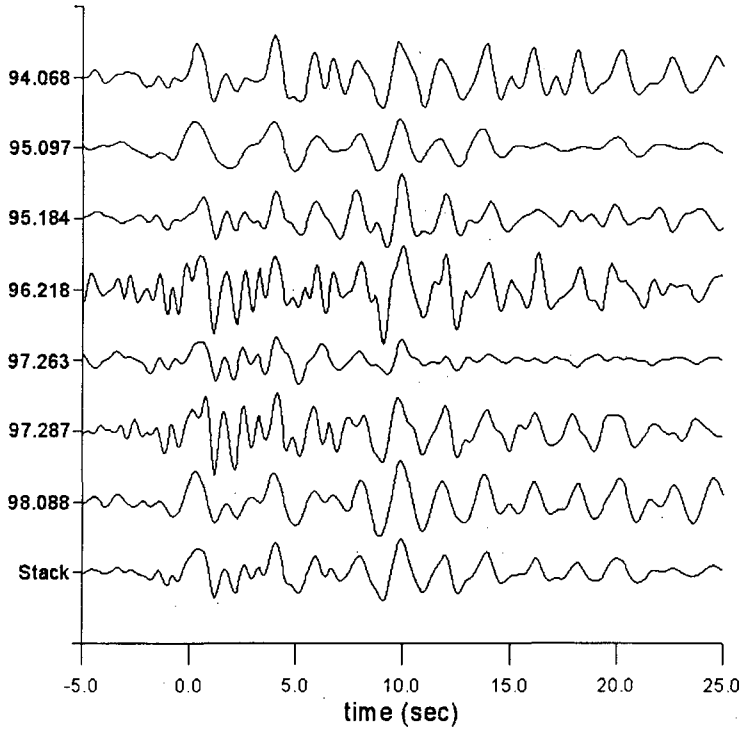


Figure 6.7. Radial receiver functions for station WDC (Whiskeytown) from the 7 South Pacific events, with receiver function stack at bottom.

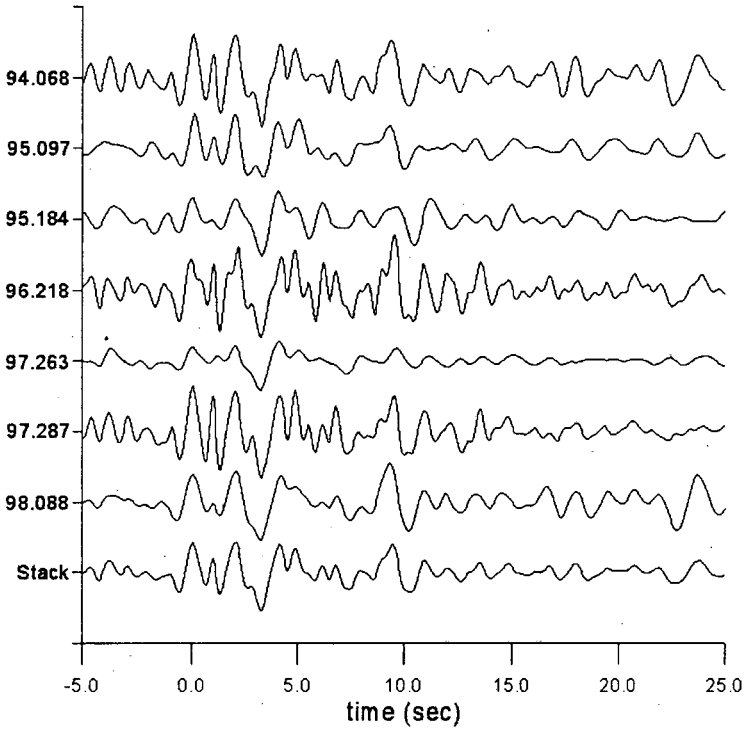


Figure 6.8. Radial receiver functions for station YBH (Yreka) from the 7 South Pacific events, with receiver function stack at bottom.

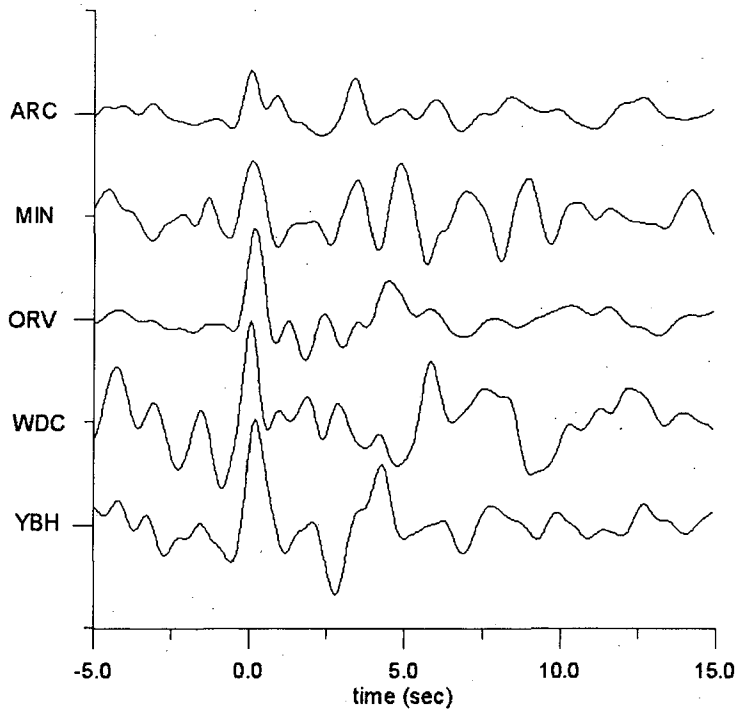


Figure 6.9. Radial receiver functions for 5 of the broadband BDSN stations from the Bolivian event.

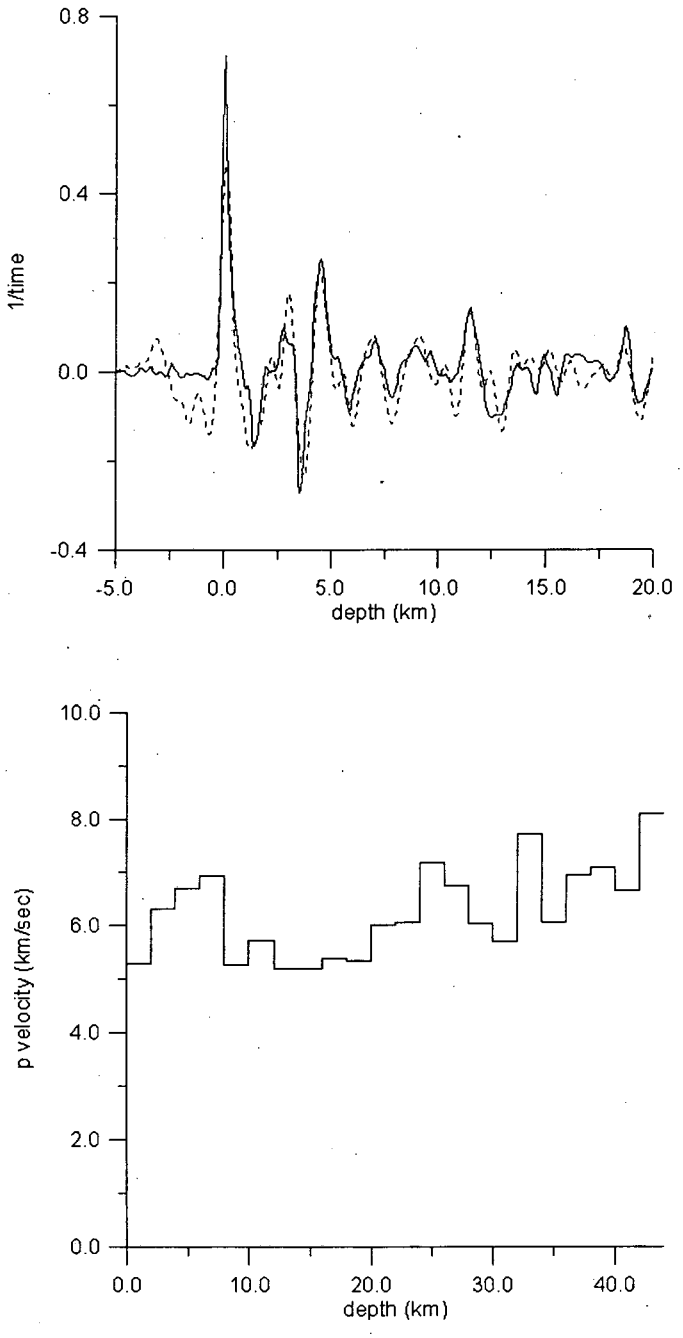


Figure 6.10. (above) Empirical receiver function for the stacked South Pacific events and fit for station Oroville and (below) model based on fit. No minimum roughness criteria is used in the objective function here.

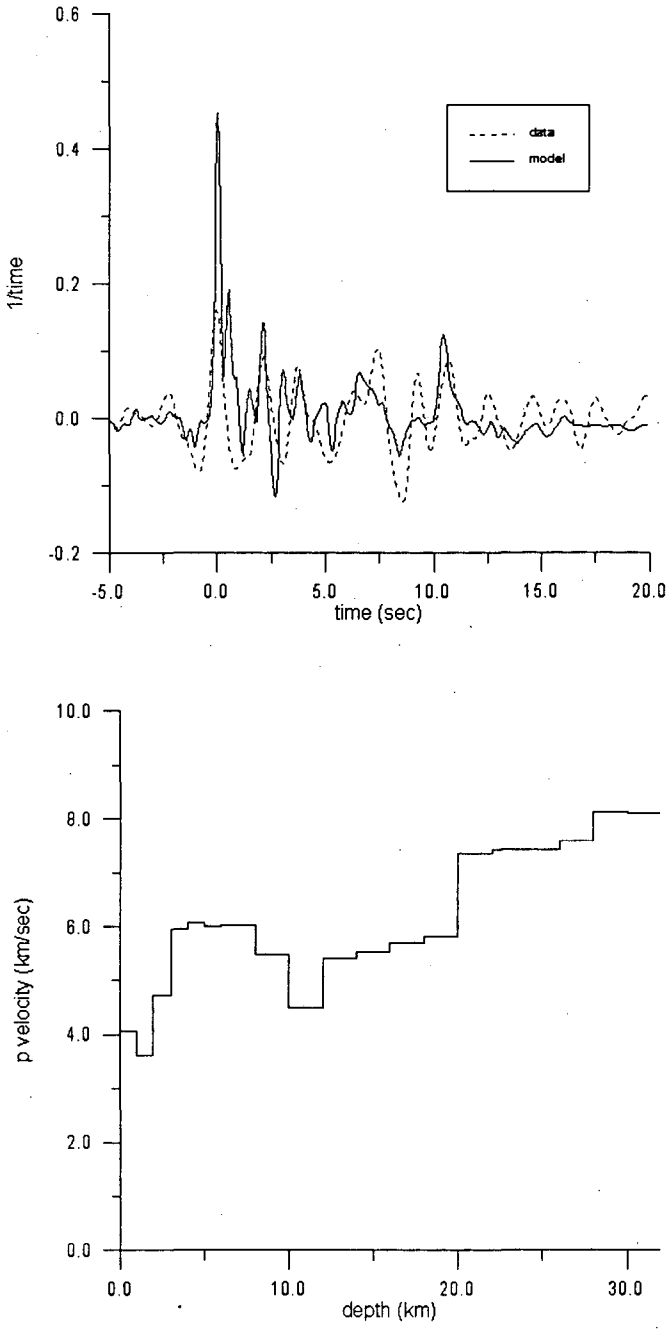


Figure 6.11. (above) Empirical receiver function for the stacked South Pacific events and fit for station Arcata and (below) model based on fit.

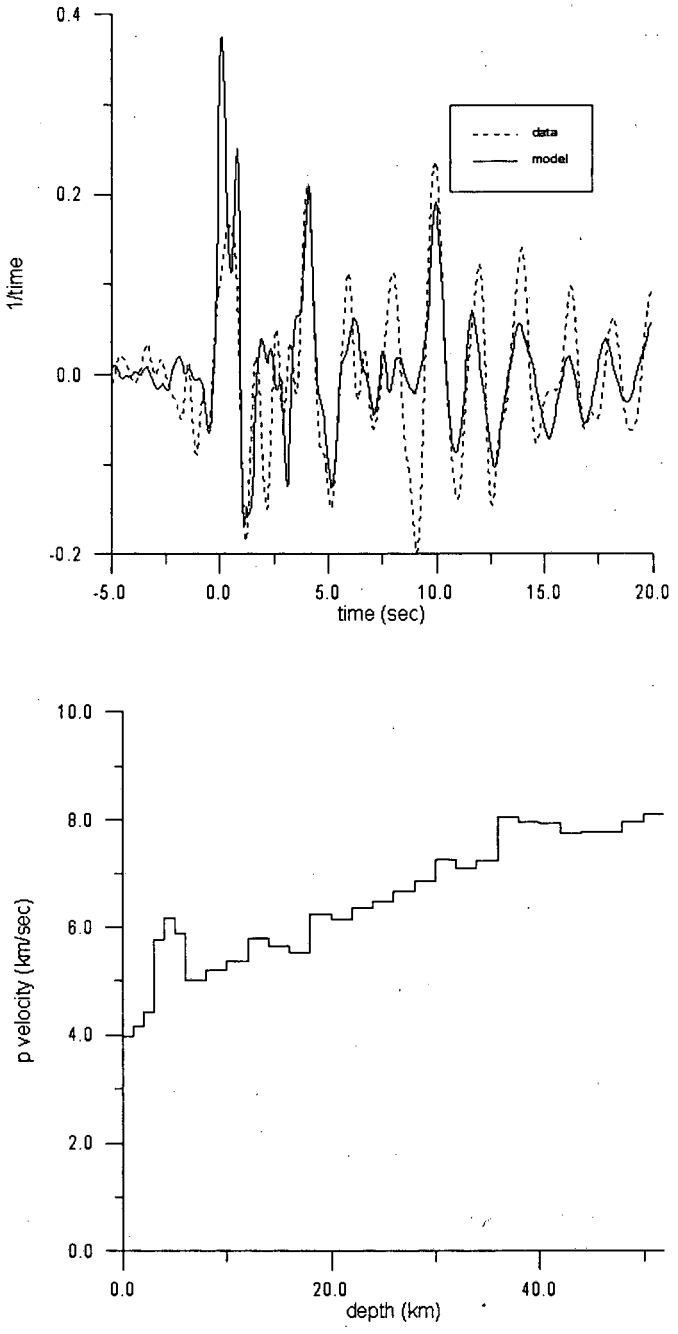


Figure 6.12. (above) Empirical receiver function for the stacked South Pacific events and fit for station Yreka and (below) model based on fit.

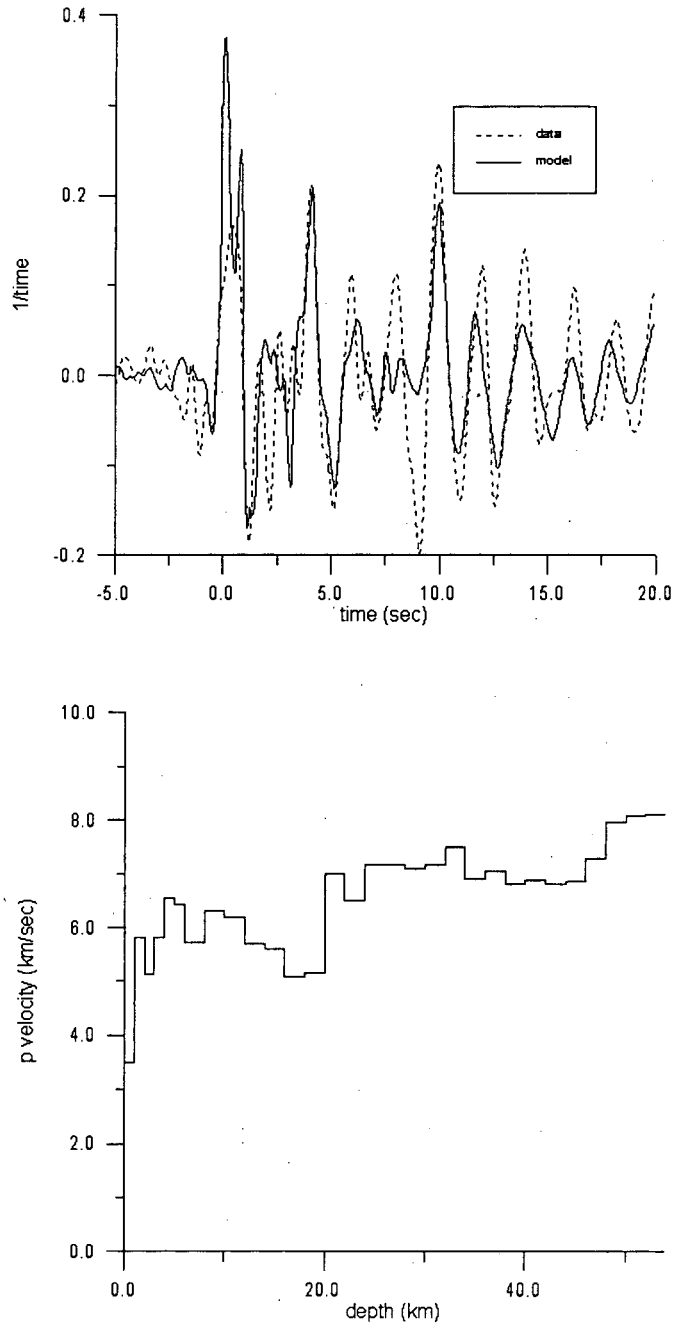


Figure 6.13. (above) Empirical receiver function for the stacked South Pacific events and fit for station Whiskeytown and (below) model based on fit.

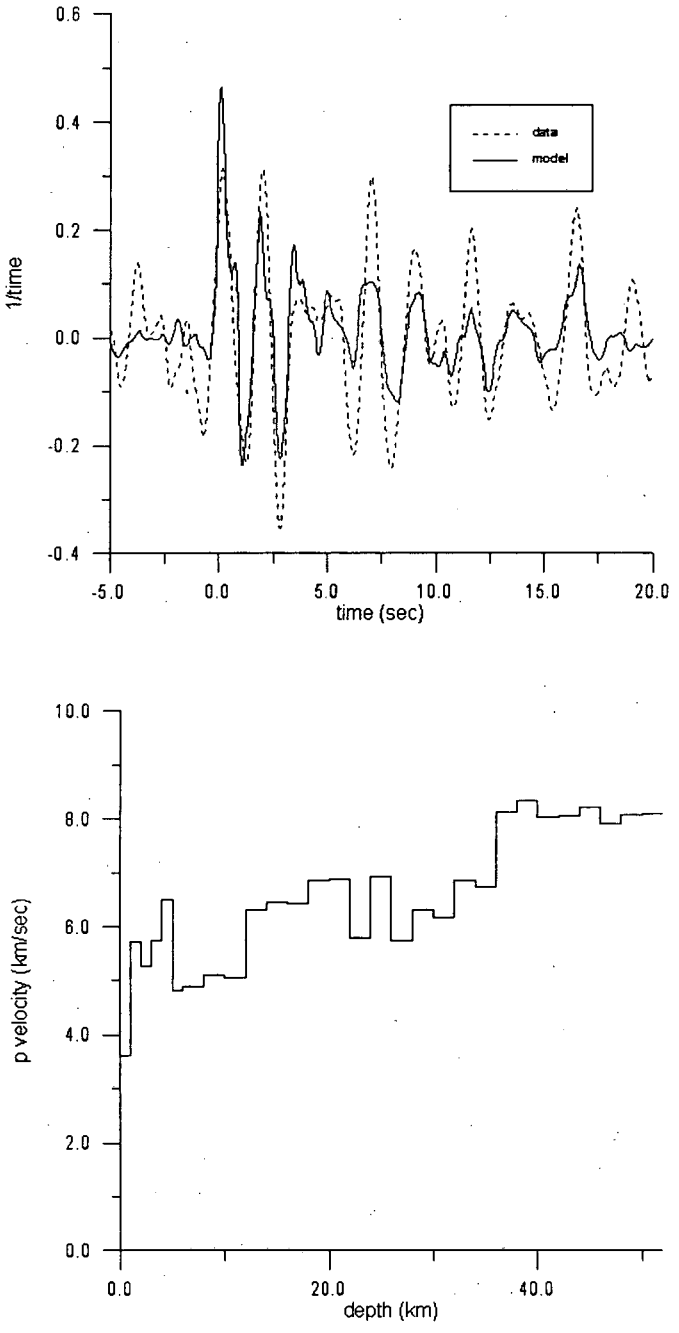


Figure 6.14. (above) Empirical receiver function for the stacked South Pacific events and fit for station Mineral and (below) model based on fit.

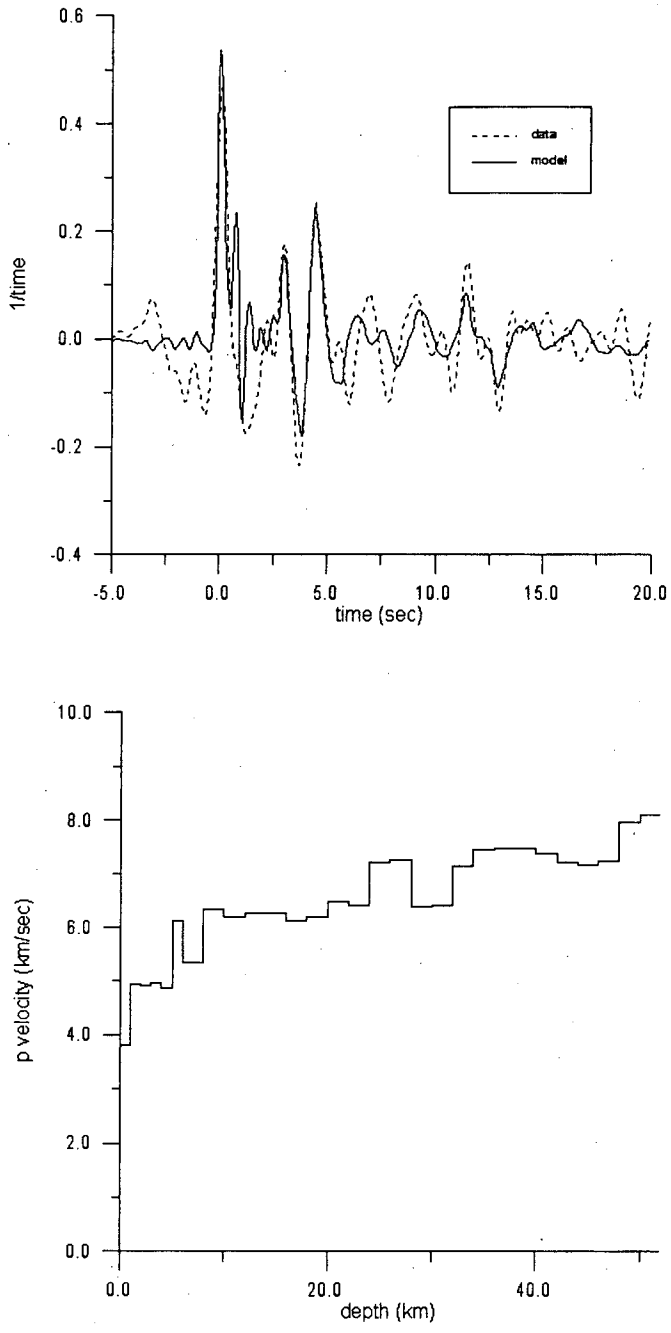


Figure 6.15. (above) Empirical receiver function for the stacked South Pacific events and fit for station Oroville and (below) model based on fit.

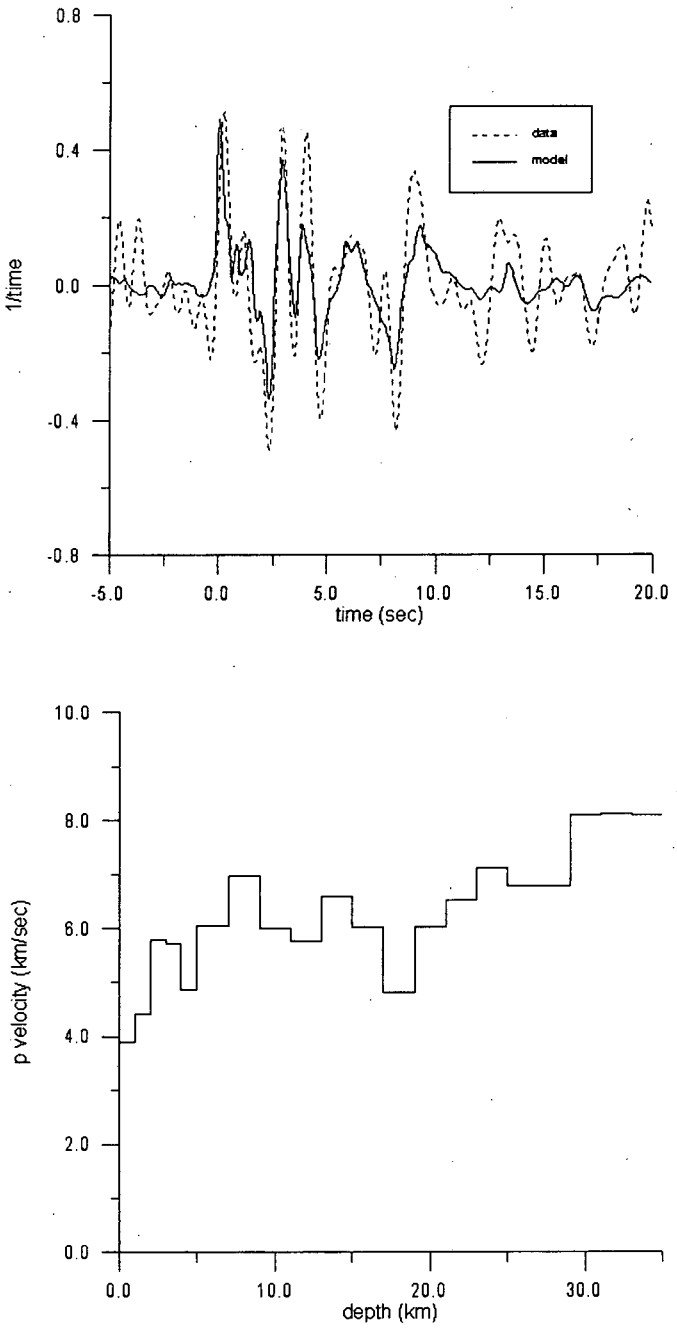


Figure 6.16. (above) Empirical receiver function for the stacked South Pacific events and fit for station Hopland and (below) model based on fit.

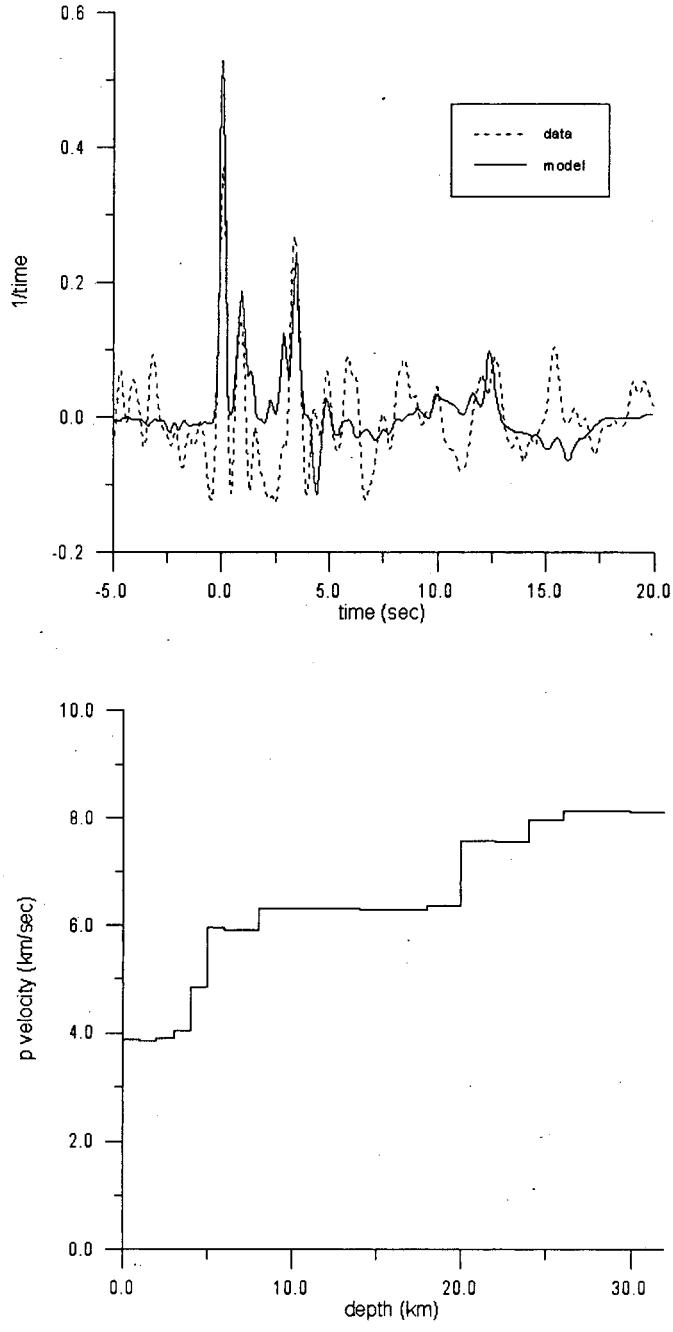


Figure 6.17. (above) Empirical receiver function for the Bolivian events and fit for station Arcata and (below) model based on fit.

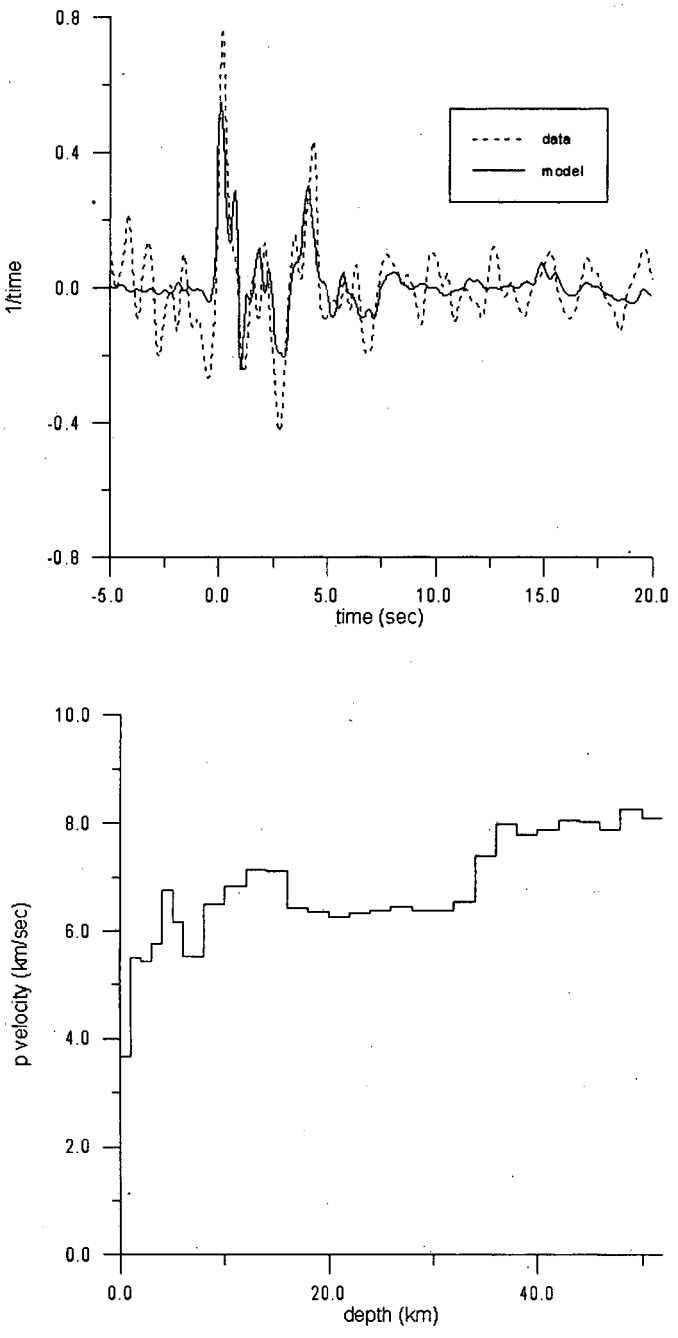


Figure 6.18. (above) Empirical receiver function for the Bolivian events and fit for station Yreka and (below) model based on fit.

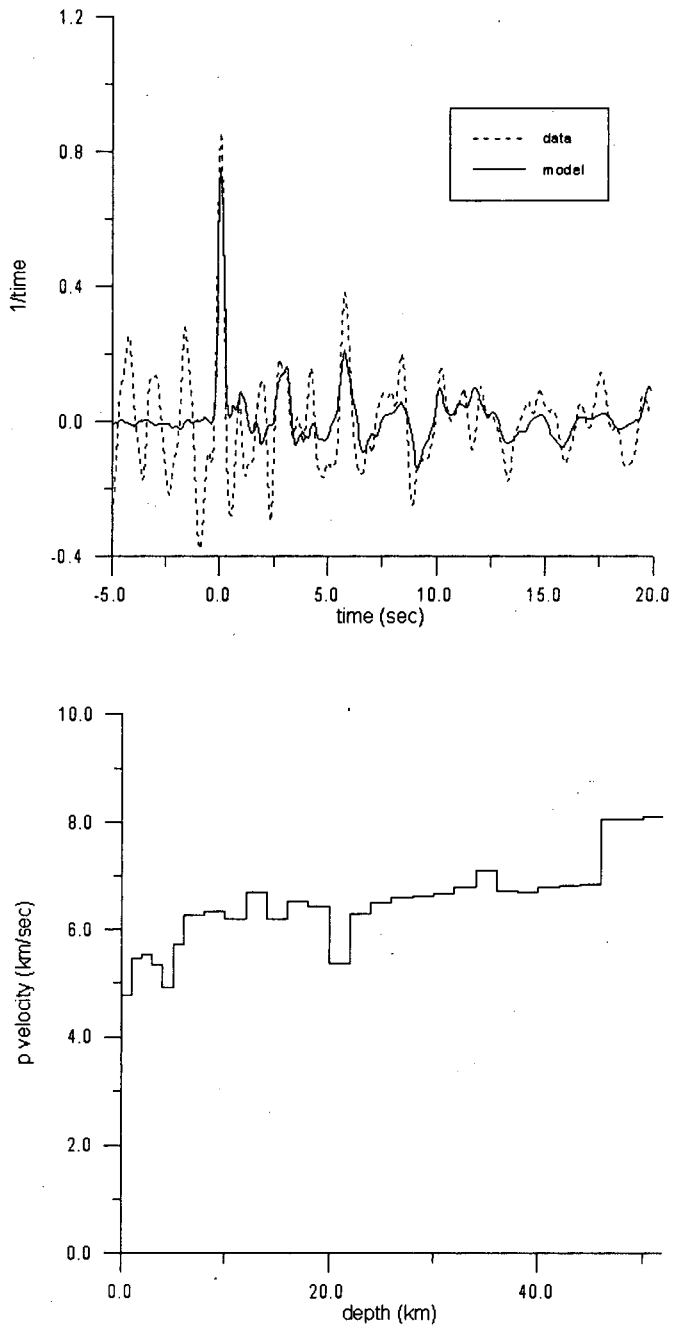


Figure 6.19. (above) Empirical receiver function for the Bolivian events and fit for station Whiskeytown and (below) model based on fit.

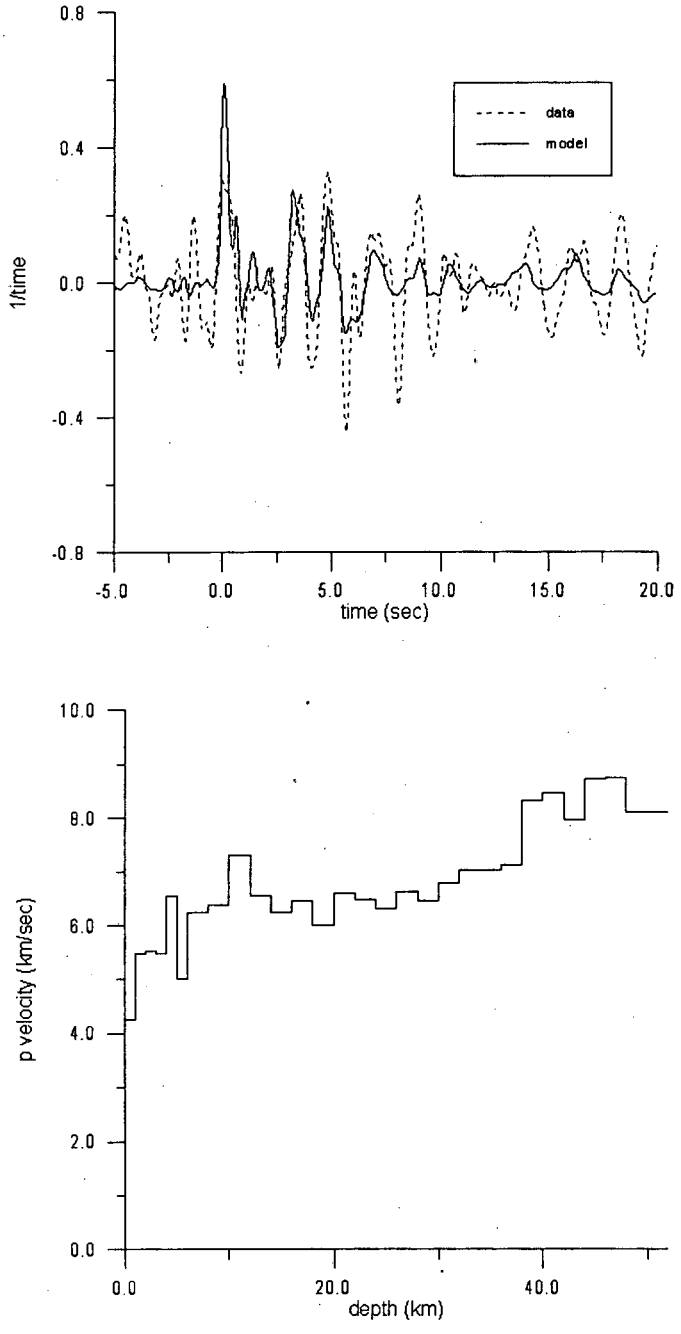


Figure 6.20. (above) Empirical receiver function for the Bolivian events and fit for station Mineral and (below) model based on fit.

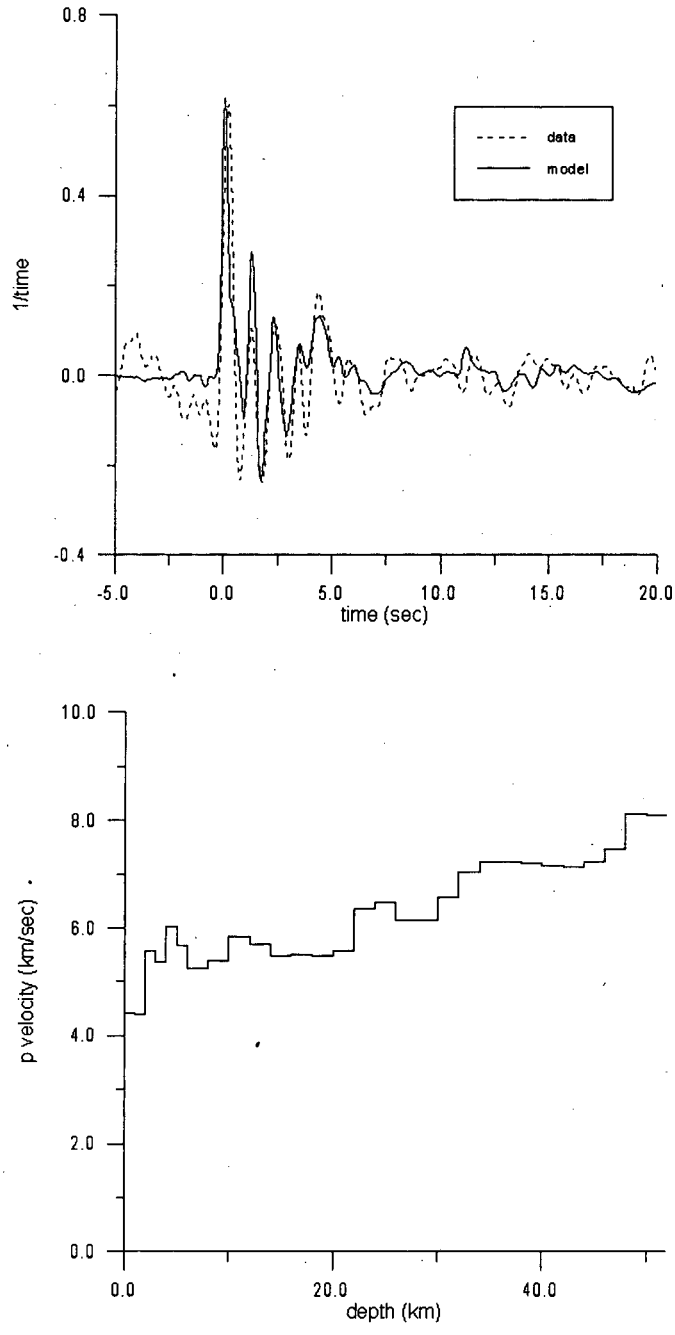


Figure 6.21. (above) Empirical receiver function for the Bolivian events and fit for station Oroville and (below) model based on fit.

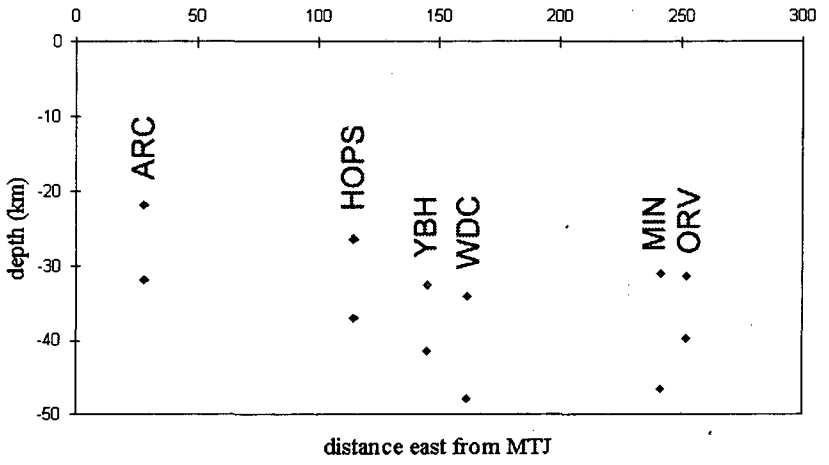


Figure 6.22. Estimated depths of conversion derived from major amplitude peaks on the stacked South Pacific receiver functions.

Chapter 7

More sophisticated techniques

7.1 Introduction

Although a simple GA will often produce excellent results, it is likely that more sophisticated operators may be necessary to obtain solutions to many special problems. For example, it is not possible to obtain multiple distinct solutions to a nonunique problem with a standard GA (An exception to this is to re-run the algorithm many times using a different random number seed, but there is still no assurance that the solutions will be distinct). Also, some problems may require a forward calculation that is too computationally costly to be solved by a standard GA. In this chapter several techniques are introduced to solve these and other problems.

7.2 Inversion operators

The expectation that the crossover operator will mix genetic information in a way that will produce improved models is based on one important assumption: That the schema or "building blocks" which combine advantageously to produce strings of higher fitness are in positions on the string that actually allow crossover to combine them. In the simple example of minimizing the function $f(x) = x$, the combination of minimizing a linear function and the natural coding order of descending powers of ten insures that crossover will produce better results. But

not all problems are simple linear optimizations, so the question of coding becomes an important issue.

There are two factors to consider when coding a problem. First, the number of bits to use and second, the order of the bits. The major constraint on the first factor is usually the accuracy desired, but experimenting with different length strings can sometimes produce faster rates of convergence. Up until this point, the second factor, the order of the bits in the string has not been considered. There is no reason why the bits can't be arranged in another order, for example $\{10^2, 10^0, 10^4, 10^1, 10^3\}$, as long as the original order can be restored so the string can be decoded. This can be done using a position independent coding in which the order of each bit on the binary string is kept in another string. In the example above, the order would be 3, 5, 1, 4, 2. The question is, in which order should they be arranged to get the best results from crossover? In most cases this is a problem of difficulty on the order of the optimization problem, so exchanging the bit positions or *inversion* is usually a random trial and error process.

Inversion is usually carried out in the following manner: two points are selected on a string, and the order of the bits between the two points is reversed. This is done to both the binary parameter string and the order (allele) string. When the fitness of the string is to be evaluated, its corresponding order string is first used to arrange the bits in their proper (original) order.

Goldberg (1989) points out that inversion must be combined with crossover in order to achieve a significant improvement in results. There are several ways to accomplish this, the best known being Partially Matched Crossover (PMX)

(Goldberg and Lingle, 1985). A subroutine for Partially Matched Crossover is given in Appendix A.

Note that inversion is only a remedy to improve the performance of traditional crossover methods (i.e. 1 and 2 point crossover). Inversion is superfluous if a disruptive recombination operator such as uniform crossover is used.

7.3 Preserving Diversity in the model population

Because numerous problems in geophysics are inherently nonunique many feasible solutions to a problem may exist. If one solution is significantly better than the rest, there can be reasonable confidence that a standard genetic algorithm will find that solution. But what if two or more solutions have very similar objective function values? Can a standard genetic algorithm find more than one solution in this case? The answer is almost always no. This is due to what people in the genetic algorithm community call *genetic drift* (DeJong, 1975). Because of instabilities associated with the sampling of a finite population, the standard genetic algorithm tends to converge to only one solution, with all the models competing for the same parameters.

An example of a multimodal function of this type is illustrated in Figure 7.1. This is a simple five peak multimodal function, with the peaks decreasing in height as the x coordinate increases. The equation for the function is

$$f(x) = \exp(-2\ln(2)\left(\frac{x-0.01}{0.8}\right)^2 \sin^6(5\pi x) \qquad (7.1)$$

Figure 7.2 shows the result of genetic drift. In trying to estimate all of the local maxima of the multimodal function all diversity is lost as the population competes for the solution $x = 0.1$ and hence only one solution is obtained. Examples of how to solve the problem of lost diversity are given in the next sections.

7.3.1 Niching

As is common in the development of genetic algorithms, observing the dynamics of natural systems gives insight into how to solve this type of problem. Nature does not force its populations to compete for the same resources, but allows individuals and species to take advantage of unique traits in order to survive. *Niching*, or *sharing* (Holland, 1975; Goldberg, 1989) is a method for maintaining diversity in the population that uses a similar criterion for selection. The idea is to give a selection advantage to models in the population that are the least similar to the rest of the population. A common way to do this is to sum and normalize the differences between the floating point parameters of each model and the rest of the population. In this way, each model is given a value from 0.0 to 1.0, where 1.0 means that the individual is exactly the same as all the others and 0.0 is as different as possible from the other models (and the other models are all exactly alike). The fitnesses of the population are then divided by this value so that the models that are the most diverse are given higher fitness and the ones that are most frequently represented are practically unchanged.

Sharing is implemented by calculating a *similarity metric*, or a multidimensional distance between a model and the other models in the population. This similarity metric can be calculated with the floating point model parameters (phenotype) or with the binary parameters (genotype). Goldberg and Richardson (1987) propose an elegant phenotypic method using a sharing function, in which the similarity metric for an individual is calculated by normalizing the difference between each real valued parameter and its corresponding parameters in the population:

$$s(d(x_i, x)) = \frac{1}{n}\sum_{k=1}^{n}\left[\frac{1}{m}\sum_{j=1}^{m}\left(\frac{(x_i)_j - (x_k)_j}{x_{\max} - x_{\min}}\right)^2\right]^{\frac{1}{2}} \tag{7.2}$$

where n is the population size, m is the number of floating point parameters for each model and x the parameter values. The adjusted fitness $(f'(x_i))$ is then calculated with the following formula:

$$f'(x_i) = \frac{f(x_i)}{s(d(x_i, x))} \tag{7.3}$$

Note that the denominator ranges from 0.0 for the model that is most unlike the rest of the population to 1.0 for the model which is identical to the rest of the population. Depending on the problem, it may be desired to find a broader range of solutions at the cost of a lesser fit. This can be done by raising the denominator to an integer or non integer power higher than 1. The function is called a "triangular sharing function" if the exponent has a value of 1.

Despite its aesthetic appeal, naively implementing sharing can produce somewhat disappointing results. Oei et al. (1991) observed that combining

tournament selection and sharing to select a new generation is inherently unstable because the rescaling of sharing conflicts with the autoscaling of tournament selection. This results in the eventual loss of diversity and genetic drift. Figure 7.3 demonstrates this diversity loss after 100, 200, 300, and 500 generations. Only the solution $x = 0.1$ remains.

Fortunately there is a fairly straightforward modification called *continuously updated sharing* (Oei et al., 1991) that maintains the diversity of the population even after several thousand generations. The only difference is that the sharing coefficient used to adjust the fitness of a potential mate is calculated by comparing that individual's phenotype to that of the next generation as it is being created. In Figure 7.4 the multimodal problem is solved using this modification. Note that even after 1,000 generations significant diversity is preserved because the best 3 of 5 possible solutions are estimated.

7.3.2 Parallel models

Another way to avoid the problem of genetic drift is to create a more "grainy" population. The most effective way to implement this (in terms of maintaining diversity) is to divide the population into isolated groups. This removes the problem of genetic drift altogether because there is no sharing of information between the groups, but there is a downside: the smaller each subpopulation, the less likely it will contain all the information necessary to obtain an optimal solution. The natural solution is to use larger subpopulations which requires more

computing time. This type of problem is well suited to parallel computing as the subpopulations can be divided between processors.

Island Models (Starkweather et al., 1991) are a slightly more integrated approach. As with the parallel model, each subpopulation is treated as a standard genetic algorithm computation, but every 10-20 generations the populations exchange strings in order to share in a larger pool of genetic material. The number of strings that are migrated between groups and the frequency of migrations is an effective way to control the convergence of the algorithm. By using a low migration rate, one can create stable subpopulations that search the model space with little influence from other subpopulations, producing multiple solutions for multimodal functions. A higher migration rate between subpopulations is closer to a single population model, but tends to converge more slowly because of the slower rate of information exchange, often producing better results. Figure 7.5 shows the solutions for the multimodal problem in the previous section obtained by using 10 islands and no migration for 100, 200, 300, and 1,000 generations. Note the stability of the solutions in comparison with the niching approach. Except for a few points, the positions of each solution are unchanged after 1,000 generations.

7.3.3 Niching versus Islands for preserving diversity

From the results in Figures 7.3-7.5 it appears that island models preserve population diversity better and give more stable solutions to multimodal

problems. Looking at the statistics strengthens this hypothesis. The *percent heterogeneity* of the population can be defined as

$$h(x)(\%) = \frac{100}{n}\sum_{i=1}^{n} s(d(x_i, x)) \tag{7.4}$$

where $s(d(x_i,x))$ is given in Equation 7.2. The percent heterogeneity is just the normalized sum of the sharing functions for each member of the population. Table 7.1 compares the heterogeneity after 100, 200, 300, 400 and 500 generations for standard niching, continuously updated sharing, islands with no migration, and islands with migration (20% migration probability every 20 generations).

	100	200	300	400	500
1 island (standard GA run) of 100 models	0.58%	0.0%	0.0%	0.0%	0.0%
10 islands of 10 models each, no migrations	26.71%	25.21%	26.06%	25.53%	25.89%
10 islands of 10 models each, migration (20%)	22.54%	24.97%	24.92%	25.01%	25.96%
standard sharing	13.75%	11.66%	17.5%	15.47%	15.15%
Continuously updated sharing	14.76%	13.08%	7.52%	7.86%	6.48%

Table 7.1. Heterogeneity for 100, 200, 300, 400 and 500 generations using both sharing and island models.

In addition to maintaining heterogeneity in the model population, Figures 7.4-7.5 show that the island models give more precise solutions than those obtained with standard niching techniques. This is because all of the models in an island

are converging on the same solution, whereas with niching there are many solutions within the same population group.

7.4 Hybrid Algorithms

As versatile as genetic algorithms are, it is typically much faster to use a more efficient method (one that processes gradient information more directly such as a calculus based method) for simple or unimodal problems. Similarly, it is faster to use more direct methods to find a local (or global) optimum when the starting model is within the region of the optimal point. Therefore it is reasonable to assume that after the application of a genetic algorithm which returns solutions in the neighborhood of an optimal point, time could be saved by using a more direct method to find the solution that corresponds to the optimum. *Hybrid algorithms* can combine the global search properties of a GA with the efficiency of a gradient based method. The GA can be used to search for peaks or troughs in the objective function after which a direct method can be used to find the extremum rapidly and precisely. The best way to search out the model space is to use an island model with many islands and a low migration rate. Methods such as *Levenberg-Marquardt* (Marquardt, 1963) and *conjugate gradient* (Press et al., 1992) are fast and efficient candidates for local search.

Pattern search (Hooke and Jeeves, 1961; Lewis et al., 1998; Torczon and Trosset, 1999) is a deterministic, derivative free search method that has gained much attention in the optimization community. Although not generally

considered a global inversion technique[1], pattern search tends to converge much faster than stochastic methods like GAs and simulated annealing. Pattern Search is also very simple and elegant, and like the stochastic methods it does not rely on gradient information directly. These qualities make it an excellent complementary method for a genetic algorithm hybridization. Figure 7.6 shows the average convergence of 10 runs for a receiver function inversion using both a GA and a GA/pattern search hybrid algorithm. The GA runs consist of 20 generations with a population of 100, while the hybrid runs consist of 10 generations of 100 followed by 10 iterations of 100 objective function evaluations. GA convergence is represented by the dashed line and convergence for the hybrid scheme by the solid line. The overall improvement in the solution using the hybrid approach is 26%. Use of the GA alone required an average of 53 generations to improve the solution by this amount.

A flow chart outlining the Pattern Search algorithm is found in Figure 7.7. The algorithm works by adjusting each parameter of a starting model by a step size, and if the model is improved by the adjustment the changes are saved. After the last parameter has been adjusted a pattern move is made if the parameters have been changed or the step size is decreased if they have not. The algorithm finishes when the step size becomes smaller than a user specified misfit size. The pattern search method has been successfully applied to problems with as many as 256

---

[1] Some confusion exists due to the definition of "global" in the above references. In these papers global convergence is defined as converging to the optimal point nearest the starting point.

variables (Torczon and Trosset, 1999). A FORTRAN code based on the original paper by Hooke and Jeeves (1961) is presented in appendix B.

7.5 Discussion and Conclusions

Certain types of problems are not conducive to the traditional GA approach, but relatively simple modifications can be made which will solve these problems more efficiently, or find solutions to problems that cannot be solved by more traditional GAs. In this chapter several techniques were presented to demonstrate how to avoid some shortcomings of the simple GA: inversion to insure that crossover is effective, niching and island models to retain diversity in the model population, and hybrid algorithms to speed convergence for time intensive problems.

Niching and island models are both effective approaches for maintaining diversity in the model population and thereby obtaining more than one distinct solution to a nonunique or multimodal problem. Island models are a passive approach to diversity, and as such there is no assurance that they will produce distinct solutions, but they appear to give more stable solutions than those obtained through the use of niching. It is also possible to combine these two techniques in order to obtain the best results: an island model algorithm could make use of niching to reward members of each subpopulation for being least similar to the members of the other subpopulations. No comparison would be made between the individual and its own population. This would add an incentive towards developing a distinct niche in each island.

There are many other possible modifications that can be made to a standard GA to increase effectiveness. For example, GAs can incorporate principles from other nonlinear optimization techniques. Stoffa and Sen (1991) suggest stretching the fitness function according to the amount of improvement from generation to generation using a Boltzmann-like energy distribution function, much like simulated annealing. Taking the opposite approach, Mahfoud and Goldberg (1995) present a parallel simulated annealing procedure which borrows principles from genetic algorithms.

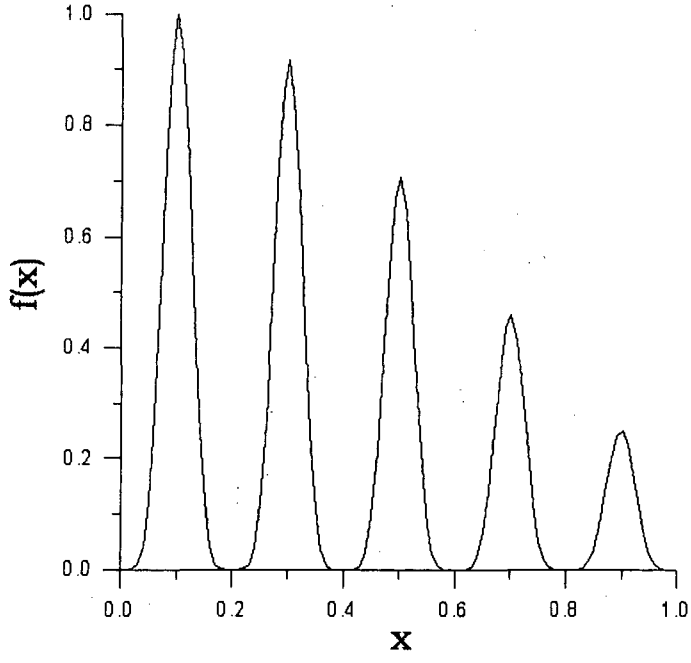


Figure 7.1. The function $f(x) = \exp(-2\ln(2)\left(\frac{x-0.01}{0.8}\right)^2 \sin^6(5\pi x)$.

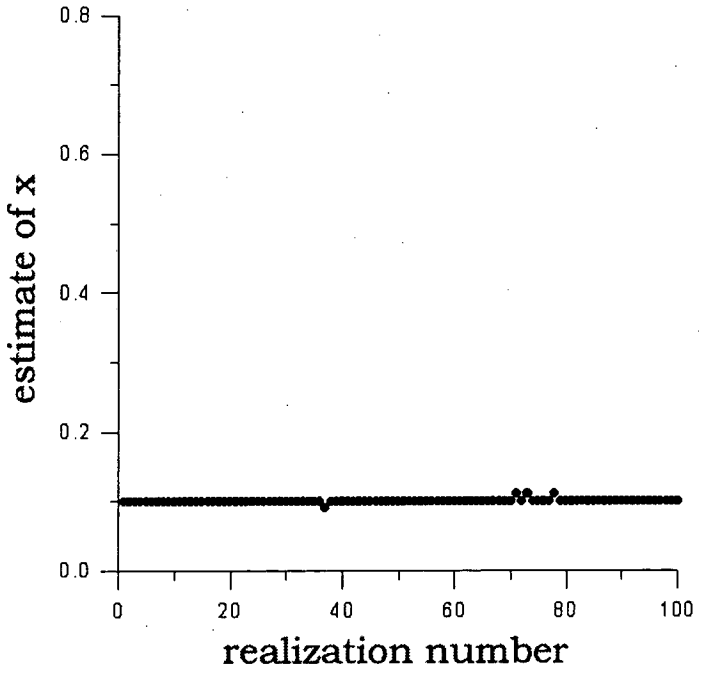


Figure 7.2. Result of genetic drift in the maximization of the multimodal function in Figure 7.1. All 100 solutions are at or very near the value x = 0.1, which is the value of the maximum peak on the function in Figure 7.1.

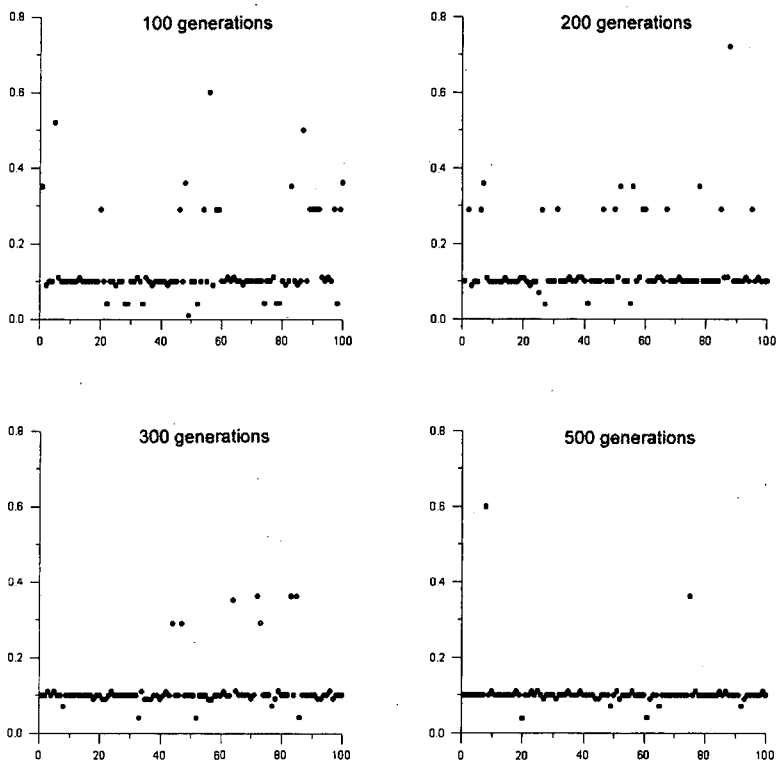


Figure 7.3. Loss of diversity attempting to find solutions to the equation plotted in Figure 7.1 using standard niching. After 500 generations all but one niche is lost. The axes are as labeled in figure 7.2.

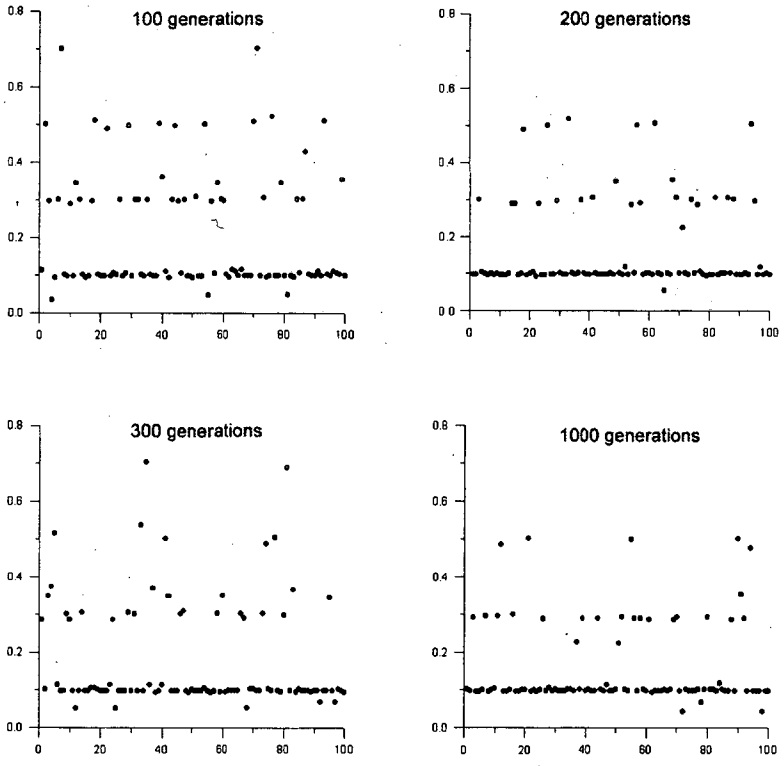


Figure 7.4. The results using continuously updated niching, showing that three niches are fairly stable even after 1,000 generations. The axes are as labeled in figure 7.2.

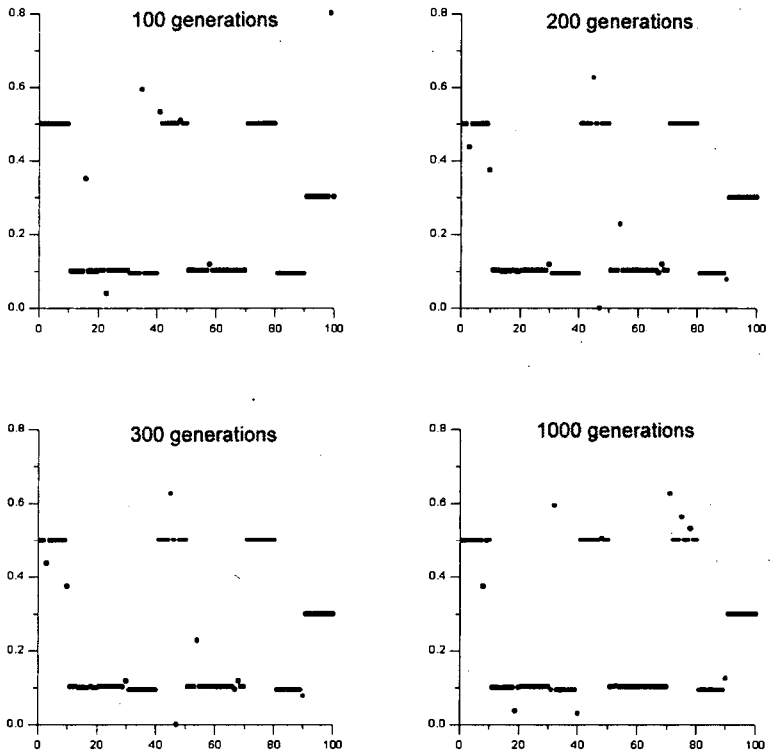


Figure 7.5. Results of the multimodal function maximization using 10 islands with no migration. The axes are as labeled in figure 7.2.

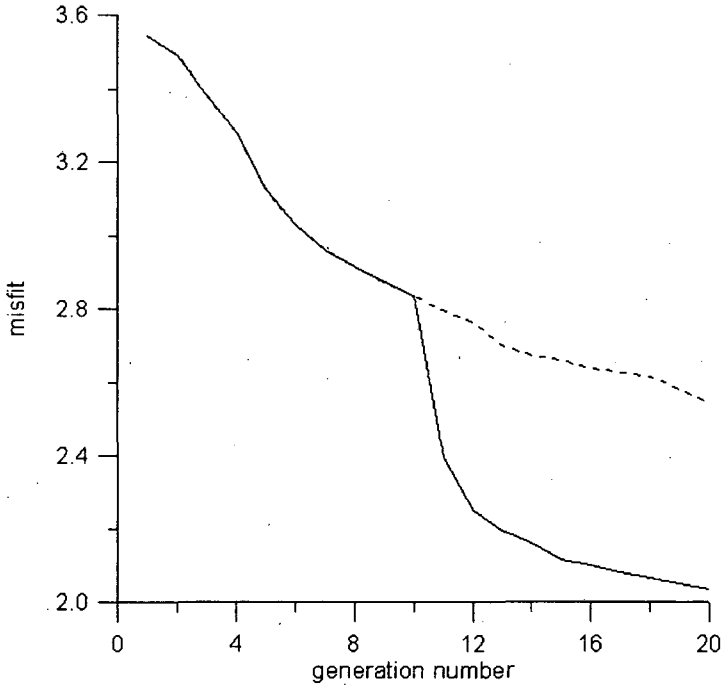


Figure 7.6. Average convergence of 10 runs for receiver function inversion using a genetic algorithm (dashed line) and a hybrid GA/pattern search (solid line). The pattern search begins after 10 generations.

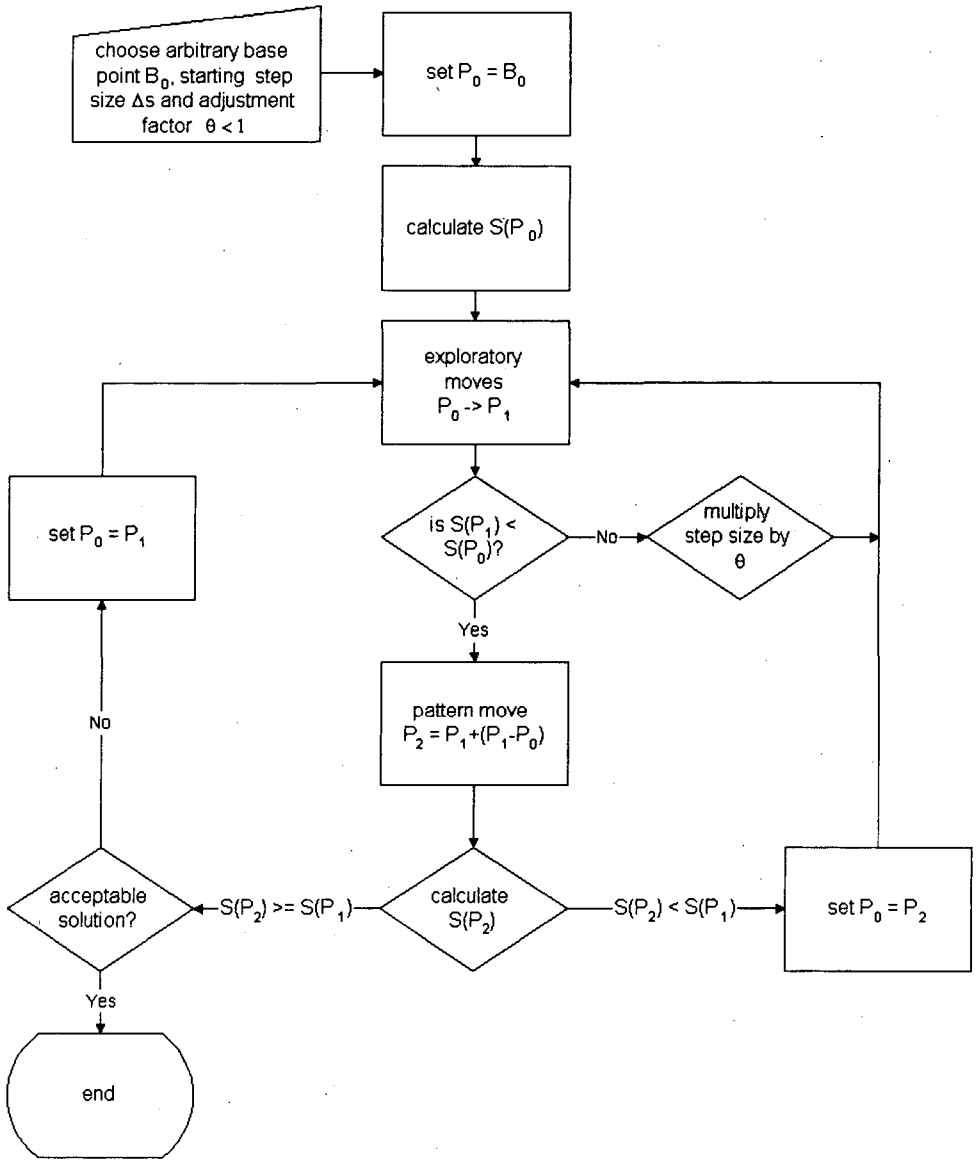


Figure 7.7. Flowchart representation of the Pattern Search algorithm used in the hybrid scheme. The algorithm is modified from the original paper by Hooke and Jeeves (1961).

Chapter 8

Conclusions

Genetic algorithms have many advantages over the methods traditionally used to solve nonlinear inverse problems in geophysics such as linear inversion techniques and quasi-Newton methods:

- Due to their stochastic nature, they are capable of finding a global optimum even in the presence of many local optima.
- The necessary objective function calculations are intrinsically parallel, so GAs can be run on parallel machines with little or no modification. This can reduce computation time by several orders of magnitude.
- They are derivative free in that they do not rely on direct calculation of the gradient. This makes them inherently stable, avoiding division by zero and other numerical difficulties associated with the calculations of derivatives.
- Rather than using a single starting point from which to search, they search a bounded model space and the search is unbiased within that space. This removes much of the subjectivity associated with starting model dependence from the inversion.

The standard genetic algorithm performs well enough for most problems, but simple modifications can allow the solution of special problems that are not solved using the traditional approach. For example, the population may be subdivided into "islands" in order to find multiple distinct solutions to a nonunique problem. Some problems in geophysics require time consuming

forward calculations, and others have very large numbers of parameters to be estimated. Such methods are not typically amenable to a stochastic search method such as a GA. However, a robust and efficient search can be made by hybridizing the GA with another more direct method such as a gradient based procedure or even a linear inversion technique.

Traditional theory holds that GAs achieve most of their performance through the action of crossing over the best parameter sets in order to combine desirable traits into new models. The purpose of mutation is believed to be only to insure that no bit information is permanently lost in the action of crossing over. Based on these assumptions, best performance should be achieved with a high crossover rate and a low mutation rate. However, numerical simulations suggest that although the mutation rate should be very low, mutation is in general a far more important operator than crossover in genetic search. In fact, an algorithm using mutation only and no crossover performs reasonably well, but one using crossover without mutation performs very poorly. The same simulations also suggest that GA performance is highly dependent on the choice of the selection method. Tournament selection appears to give better results than the other methods for all the problems discussed here.

Limited computational resources have traditionally forced geophysicists to solve nonlinear inverse problems by using linear approximations which produce models that are heavily dependent on starting models. Many problems of current interest in geophysics are inherently nonlinear in both the forward and inverse problem, and such problems are reliably solved only with a global approach.

Examples of this type of problem are fluid flow modeling and mantle convection. In combination with increasing computer speed, GAs will provide a robust and relatively efficient approach to these and other problems.

References

Ammon, C., G. E. Randall and G. Zandt, 1990, On the Nonuniqueness of Receiver Function Inversions, *J. Geophys. Res.* **95**, 15,303-15,318.

Aarts, E. and J. Horst, 1989, *Simulated Annealing and Boltzmann Machines*, New York: John Wiley and Sons.

Atwater, T., 1970, Implications of Plate Tectonics For Cenozoic Tectonic Evolution of Western North America, *Geol. Soc. Am. Bull.*, **81**, 3513-3536.

Bäck, T., 1996, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford: Oxford University Press.

Beaudoin, B., and M. Magee, 1994, Crustal Velocity Structure of the Mendocino Triple Junction, *J. Geophys. Res.* **21**, 2319-2322.

Benz, H. M., G. Zant, and D.H. Oppenheimer, 1992, Lithospheric Structure of Northern California From Teleseismic Images of the Upper Mantle, *J. Geophys. Res.*, **97**, 4791-4807.

Berteussen, K. A., 1997, Moho Depth Determinations Based on Spectral Ratio Analysis of NORSAR Long Period *P* Waves, *Phys. Earth and Planet. Inter.*, **31**, 313-326.

Clayton, R. W. and R. A. Wiggins, 1976, Source Shape Estimation and Deconvolution of Seismic Body Waves, *Geophys. J.R. Astron. Soc.*, **47**, 151-177.

De Groot-Hedlin, C. D., and F. L. Vernon, 1999, An Evolutionary Programming Method for Estimating Layered Velocity Structure, *Bull. Of the Seis. Soc. of America*, **88**, 1023-1035.

De Jong, K. A., 1975, An analysis of the behavior of a class of genetic adaptive systems. *Dissertation Abstracts International*, vol. **36**, no. **10**, 5140B (University Microfilms No. 76-9381).

Dicke, M., 1998, Seismicity and Crustal Structure at the Mendocino Triple Junction, Northern California. Master's Thesis, University of California at Berkeley.

Eberhart-Phillips, D., and M. Reyners, 1997, Continental Subduction and Three Dimensional Crustal Structure: The Northern South Island, New Zealand, *J. Geophys. Res.* **102**, 11,843-11,861.

Fitzpatrick, J. M. and J. J. Grefenstette, 1988, Genetic algorithms in noisy environments. *Machine Learning,* **3**, 101-120.

Fogel, L. J., 1962, Autonomous Automata, *Industrial Res.*, **4**, 14-19.

Fowler, C. M. R., 1990, *The Solid Earth an Introduction to Global Geophysics*, Cambridge: Cambridge University Press.

Goldberg, D. E., and J. Richardson, 1987, An investigation into the Niche and species formation in genetic function optimization, *Proceedings of the Third International Conference on Genetic Algorithms*, 42-50, San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, MA: Addison-Wesley.

Goldberg, D. E. and R. Lingle, 1985, Alleles, loci, and the traveling salesman problem. *Proceedings of an International Conference on Genetic Algorithms and their Applications*, 154-159.

Goldberg, D. E., and M. Rudnick, 1993, Genetic algorithms and the variance of fitness, *Complex Systems*, vol. **5**, no. **3**, 25-49.

Harik, G., E. Cantu-Paz, D. E. Goldberg and B. L. Miller, 1996, The gambler's ruin problem, genetic algorithms, and the sizing of populations, ILLIGAL report #96004. h*ttp://GAL4.GE.UIC.EDU/cgi-bin/orderform/orderform.cgi/*

Hesser, J. and R. Männer, 1991, Towards an optimal mutation probability for genetic algorithms, *Parallel Problem Solving from Nature*, 23-32.

Holland, J., 1975, *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

Hooke, R. and T. A. Jeeves, 1961, Direct Search solution of numerical and statistical problems, *J. of Assoc. Comput. Mach.*, **8**, 212- 229.

Johnson , L. R. and J. J. Litehiser, 1972, A method for computing the gravitational attraction of three dimensional bodies in a spherical or ellipsoidal earth, *J. Geophys. Res.* **77**, 6999-7009.

Johnson, L. R., P. B. Parker, K. H. Williams and A. E. Romero, 1995, Gravity and magnetic data on regional seismic lines, *LBNL Summary Report 0BB01-LBNL.*

Kennet, B. L. N., 1983, *Seismic Wave Propagation in a Stratified Media*, Cambridge: Cambridge University Press.

Kirkpatrick, S., C. D. Gelatt and M. P. Vechi, 1988, Optimization by simulated annealing, *Science,* **220**, 671-680.

Langston, C. A., 1979, Structure Under Mount Rainier, Washington, Inferred From Teleseismic Body Waves, *J. Geophys. Res.* **84**, 4749-4762.

Lemonick, M. D., 1994, The Killers all Around, *Time Magazine*, **vol**. 144, **no**. 11.

Lewis, R. M., V. Torczon, and M. W. Trosset, 1998, Why Pattern Search Works, *Optima* **59**, 1-7.

Majer, M. Feighner, L. Johnson, T. Daley, E. Karageorgi, K. H. Lee, K. Williams, and T. McKevilly, 1996, Surface Geophysics at Yucca Mountain and Vicinity, *Milestone OB05M*, Lawrence Berkeley National Laboratory.

Marquardt, D. W., 1963, *Journal of the Society for Industrial and Applied Mathematics*, **11**, 431-441.

Mühlenbein, H., 1992, How genetic algorithms really work: I. mutation and hillclimbing, *Parallel Problem Solving from Nature -2-*, 15-25.

Mahfoud, S. W. and D. E. Goldberg, 1995, Parallel recombinative simulated annealing: A genetic algorithm, *Parallel Computing* **21**, 1-28.

Oei, C., Goldberg, D., Chang, S., 1991, Tournament selection, niching, and the preservation of diversity, ILLIGAL report #91011.
http://GAL4.GE.UIC.EDU/cgi-bin/orderform/orderform.cgi/

Owens, T. J., G. Zandt, and S. R. Taylor, 1984, Seismic Evidence for an Ancient Rift Beneath the Cumberland Plateau, Tennessee: A Detailed Analysis of Broadband Teleseismic P Waveforms, *J. Geophys. Res.*, **89**, 7783-7795.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, 1992, *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, Cambridge: Cambridge University Press.

Reyners, M. and H. Cowan, 1993, The transition from subduction to continental collision: Crustal structure in the North Canterbury region, New Zealand, *Geophys. J. Int.*, **115**, 1124-1136.

Sen, M. and P. Stoffa, 1992, Rapid sampling of model space using genetic algorithms: examples from seismic waveform inversion. *Geophys. J. Int.*, **108**, 281-292.

Shaffer, J. D., Caruana, R. A., Eshelman, L. J., and R. Das, 1989, A study of control parameters affecting online performance of genetic algorithms for function optimization. *Proc. of the Third International Conference of Genetic Algorithms*, 51-61.

Shibutani, T. A., M. Sambridge and B. Kennet, 1996, Genetic Algorithm Inversion for Receiver Functions with Application to the Crust and Uppermost Mantle Structure Beneath Eastern Australia, *Geophysical Research Letters*, **23**, 1829-1832.

Snyder, D. B., and W. J. Carr, 1984, Interpretation of gravity data in a complex volcano-tectonic setting, southwest Nevada, *J. Geophys. Res.* **89**, 10,193-10,206.

Spears, W. and K. DeJong, 1991, An analysis of multi point crossover, *Foundations of Genetic Algorithms*, G. Rawlins, ed., Morgan-Kaufmann, 94-100.

Starkweather, T., D. Whitley, and K. Mathias, 1991, Optimization using distributed genetic algorithms, *Parallel Problem Solving in Nature*, Springer Verlag.

Stern, T. A., P. E. Wannamaker, D. Eberhardt Phillips, D. Okaya, F. J. Davey and the South Island Working Group, 1997, Mountain Building and Active Deformation Studied in New Zealand, *EOS*, vol. **78**, no. **32**.

Stoffa, P. A. and M. K. Sen, 1991, Nonlinear multparameter optimization using genetic algorithms: Inversion of plane-wave seismograms, *Geophysics*, vol. **56**, no. **11**, 1794-1810.

Syswerda, G., 1989, Uniform crossover in genetic algorithms, *Proceedings of the 3rd International Conference of Genetic Algorithms*, Morgan-Kaufmann, 2-9.

Telford, W. M., L. P. Geldart, R. E. Sheriff and D. A. Keys, 1976, *Applied Geophysics*, Cambridge: Cambridge University Press.

Torczon, V. and M. W. Trosset, 1999, From Evolutionary Operation to Parallel Direct Search: Pattern Search Algorithms for Numerical Computation, *Comp. Sci. and Stat.*, **29**, 396-401.

Verdonck, D. and G. Zandt, 1994, Three Dimensional Crustal Structure of the Mendocino Triple Junction Region From Local Earthquake Travel Times, *J. Geophys. Res.* **99**, 23843-23858.

Walcott, R. I., 1997, Models of oblique compression: late Cenozoic tectonics of the South Island of New Zealand, *Rev. Geophys.*

Whitley, D., 1996, *A Genetic Algorithm Tutorial.* Samizdat Press. *http://landau.Mines.EDU/~samizdat/*

Appendix A: An island model genetic algorithm in FORTRAN

This code is currently set to handle a maximum population size of 200, a maximum parameter size of 48, and a maximum string length of 384. The parameter statements must be changed in the main program and the subroutines in order to use higher values.

There are three possible selection schemes: Roulette Wheel selection, Tournament selection and Stochastic Universal selection. For all but tournament selection, a fitness scaling method should be used to prevent premature convergence early in a run and to promote healthy competition in a mature run.

There are two choices for crossover algorithms: single-point and uniform crossover. Fitness scaling is not available for function maximization, so a scale independent selection scheme should be used for maximization (in this case tournament selection).

The random number generator has been left out, as most people have their own favorite subroutine for this. See for example Press et al. (1992)

Program variable definitions

fit	Array of fitnesses for each individual.
ibest	Binary array of parameters for fittest individual.
ichild	Binary array of offspring.
idum	The initial random number seed for the GA run This can be set to any negative integer, e.g. idum= -1000.

ielite	= 0 for no elitism (best individual not necessarily replicated from one generation to the next).
	= 1 for elitism to be invoked (best individual replicated into next generation); elitism is recommended.
imax	The number of generations (iterations).
iparent	Binary parameter parent array.
icross	= 0 for single-point crossover
	= 1 for uniform crossover.
irestart	= 0 for normal run
	= 1 for restart (must have restart.inp file from previous run).
istr	= 0 for no sigma truncation (a procedure to remove weak "lethals" from the population by setting the fitness of weak outlyers to 0)
	= 1 for sigma truncation.
iselect	= 0 for Tournament Selection
	= 1 for Roulette Wheel Selection
	= 2 for Stochastic Universal Selection.
min	= 0 for function Maximization
	= 1 for function Minimization.
mixint	Number of generations per migration.
nchrmax	Maximum number of chromosomes (bits) per string.
nchrome	Number of chromosomes for each individual
	Equal to nparam*nbits.
nparam	Number of input parameters.
nparmax	Maximum number of parameters allowed.
npopmax	Maximum population size allowed.
nbits	Array of the number of bits per parameter, e.g., *nbits*=2 gives 4 (=2**2) possible models per parameter.
nrestart	Number of generations between each update of restart.inp file.
parent	Floating point parameter parent array.

parmax	Upper bound values of the parameter array to be optimized from the lower bound values in the parmin array.
parmin	Lower bound values of the parameter array to be optimized.
pbest	Floating Point array of parameters for fittest individual.
pcross	the crossover probability, 0.6 or 0.7 is recommended.
pmutate	The mutation probability, usually set to 1/modpop.
smult	Multiple for Scaling. Should be between 1.0 and 2.0, for simple functions about 1.8 works well.
stmult	Multiple for sigma truncation (should be between 1.0 and 3.0).
xmixprob	Migrating probability per individual.

Subroutine Descriptions

cross	Performs single-point crossover on pairs of iparent strings to generate ichild array.
decode	Decodes a binary string into a real number.
evaluate	Evaluates the fitness of the population.
lscale	Applies a linear scaling to the fitness array.
mix	Allows individuals to migrate between islands.
mutate	Converts a 1 to a 0 or a 0 to a 1 on a binary string with a probability of pmutate.
nexgen	Writes ichild array onto iparent array, and replicates the fittest individual into the array if not already done (if ielite = 1).
output	Writes parameters for final generation to genalg.out testing niching.
restart	Reads in binary file restart.inp
rselect	Selects a mate according to the "Roulette Wheel" method, where the individual's chance of being chosen is proportional to its fitness.
shuffle	Shuffles the ichild array for two by two mating.
start	Begins the run by generating the initial binary parameter array.

strunc	Applies Sigma Truncation to fitnesses. this is useful to remove "lethals" from the population and allow a better linear scaling.
tselect	Selects the fittest of two random parents for mating.
ucross	Performs uniform crossover on pairs of iparent strings to generate ichild array.
uselect	Selects the fittest members of the population according to the Stochastic Universal method, an efficient variation of Stochastic Remainder selection (requires shuffling to randomize pairs for mating)
wrestart	writes iparent array to restart.inp file.

Input files

Sample genalg.param file:

```
6          //nparam (# of parameters)
8          //nbits (# of bits per parameter)
0          //icross (=0 for single point, 1 for uniform)
0.7        //pcross (crossover probability)
0.01       //pmutate (mutation probability per bit)
0          //iselect (=0 for tournament, 1 for RW, 2 for US)
1          //ielite (1 for elitism, 0 for no)
1          //min (1 for minimization, 0 for maximization)
2.0        //smult (scaling multiple, ~1.3-2.0)
0          //istr (1 for sigma truncation, 0 for no)
2.0        //stmult (multiple for sigma truncation)
0          //irestart (1 if run is a restart, 0 for normal)
100        //nrestart (number of generations per restart file)
-1         //idum (random number seed for initial number)
5          //mixint (number of generations between mixing)
0.1        //xmixprob (probability of each individual mixing)
```

Sample genalg.in file (for six parameters, all real numbers):

```
0.0 1.0    //xmin xmax
0.0 1.0    //xmin xmax
0.0 1.0    //xmin xmax
0.0 1.0    //xmin xmax
0.0 1.0    //xmin xmax
0.0 1.0    //xmin xmax
```

Output files

converge.out	At end of every generation the generation number, the minimum objective function, the average fitness and the maximum fitness of the population are output.
genalg.out	For each island, the number of the best model, its fitness, and its parameter values are output (floating point) at the end of the run.
restart.inp	Binary parameter values for restart file.

Main Routine

```
      PROGRAM GENALG

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION iparent(npopmax,nchrmax),ichild(npopmax,nchrmax)
      DIMENSION parent(npopmax,nparmax),fit(npopmax)
      DIMENSION ibest(nislmax,nchrmax),kbest(nislmax)
      DIMENSION parmin(nparmax),parmax(nparmax)
      COMMON / gen1 / parent,fit
      COMMON / gen2 / iparent,ichild
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen4 / pcross,pmutate
      COMMON / gen5 / iselect,smult,stmult
      COMMON / gen6 / parmin,parmax
      COMMON / gen7 / islpop,nisland,xmixprob

10    format (A20)
11    format (A24,f4.3)
12    format (A14,E10.4)
13    format (A28,I4)
14    format (' Population per Island = ',$)
15    format (' Number of Generations = ',$)
16    format (A35)
17    format (' Error: Pop. must be at least 2')
18    format (' Error: Population too large')
22    format (' Error: Pop. must be a multiple of 2')
24    format (' Number of Islands =  ',$)
28    format (' Error: Too many Islands')

      print 16, ' ================================='
      print 10, ' GenAlg Version 4.1 '
      print 10, ' Single Precision   '
      print 16, ' ================================='
      print *, ''
c-----Prompt for population size and number of generations
100   print 24
      read *, nisland
```

```
      if (nisland.gt.nislmax) then
          print 28
          go to 100
      end if
110   print 14
      read *, islpop
      modpop=islpop*nisland
      mremain=mod(islpop,2)
      if (islpop.lt.2) then
          print 17
          go to 110
      else if (modpop.gt.npopmax) then
          print 18
          go to 110
      else if (mremain.ne.0) then
          print 22
          go to 110
      end if
      print 15
      read *, imax
      print *, ''
c-----Open genalg.param and read input parameters
      open (unit=1,file='genalg.param',status='old')
      open (unit=4,file='converge.out',status='unknown')
      read (1,*) nparam
      read (1,*) nbits
      read (1,*) icross
      read (1,*) pcross
      read (1,*) pmutate
      read (1,*) iselect
      read (1,*) ielite
      read (1,*) min
      read (1,*) smult
      read (1,*) istr
      read (1,*) stmult
      read (1,*) irestart
      read (1,*) nrestart
      read (1,*) idum
      read (1,*) mixint
      read (1,*) xmixprob
      close (1)
      nchrome=nbits*nparam
c-----Initiate generation counter
      icount=1
c-----Open Genalg.in and generate the parmin and parmax arrays
      open (unit=2,file='genalg.in',status='old')
      do 200 i=1,nparam
          read(2,*) parmin(i),parmax(i)
200   continue
      close (2)
c-----If this run is a restart then read in old parameters to
c-----the ichild array
      if (irestart.eq.1) then
          call restart(modpop,nchrome)
```

```
c-----Otherwise, generate modpop random models from model space
      else
          call start(modpop,nchrome)
      end if
c-----Begin main processing loop
400   continue
c-----Decode the binary strings to real number arrays and test
c-----their fitness with the fitness function
      call evaluate(icount,min,ibest,kbest,fmax,fitavg)
      rcount=amod(float(icount),float(nrestart))
c-----Write to restart file if count/nrestart=0
      if (rcount.eq.0.0.and.icount.ge.nrestart) then
          call wrestart(iparent)
      end if
      print 13, ' Finished Generation Number ',icount
      if (icount.eq.imax) then
          call output(kbest)
          go to 800
      end if
c-----Enter population into selection, crossover and mutation
c-----First select parents from the population using either
c-----tournament selection, roulette wheel selection, or
c-----stochastic remainder selection
      if (iselect.eq.0) then
          call tselect
      else if (iselect.eq.1) then
          call rselect
      else if (iselect.eq.2) then
          call uselect
      end if
c-----Then apply either single-point or uniform crossover
      if (icross.eq.0) then
          call cross(pcross)
      else
          call ucross(pcross)
      end if
c-----Then mutation
      call mutate(iparent)
c-----If elitism is used, check to see if best individual was
c-----replicated and replicate if not.
      if (ielite.eq.1) then
          call elite(ibest)
      end if
      iremain=mod(icount,mixint)
      if (nisland.gt.1.and.iremain.eq.0) then
          call migrate(ichild)
      end if
      icount=icount+1
c-----If this is not the last generation continue loop.
      if(icount.le.imax) go to 400
800   close (4)
      print *, ''
      print 16, '==== Finished run successfully ===='
```

```
      STOP
      END
```

Starting a new run

This subroutine generates the initial binary *iparent* array randomly. It is called once at the beginning of each run if *irestart* is not equal to 1 (the run is not a restart).

```
      SUBROUTINE start(modpop,nchrome)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION ichild(npopmax,nchrmax),iparent(npopmax,nchrmax)
      COMMON / gen2 / iparent,ichild

      do 100 i=1,modpop
          do 110 j=1,nchrome
              rtemp=ran1(idum)
              iparent(i,j)=1
              if (rtemp.lt.0.5) iparent(i,j)=0
110       continue
100   continue

      RETURN
      END
```

Restarting a previous run

If irestart is equal to 1, the following subroutine reads in the binary *restart.inp* file at the beginning of the run.

```
      SUBROUTINE restart(modpop,nchrome)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION ichild(npopmax,nchrmax),iparent(npopmax,nchrmax)
      COMMON / gen2 / iparent,ichild

32    format (384(i2))

      open (unit=30,file='restart.inp',status='unknown')
      do 200 i=1,modpop
          read (30,32) (iparent(i,j),j=1,nchrome)
200   continue
      close (30)
```

```
      RETURN
      END
```

Single Point Crossover

For each pair of consecutive strings in the *iparent* array, if *rand1* is less than or equal to the crossover probability *pcross*, the following subroutine performs single-point crossover to generate two new strings. This is done until the *ichild* array is filled.

```
      SUBROUTINE cross(pcross)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION ichild(npopmax,nchrmax),iparent(npopmax,nchrmax)
      COMMON / gen2 / iparent,ichild
      COMMON / gen3 / nbits,nparam,modpop,nchrome

      do 100 i=1,modpop,2
          rand1=ran1(idum)
          if (rand1.le.pcross) then
              icross=2+int(real(nchrome-1)*ran1(idum))
              do 200 j=1,icross-1
                  iparent(i,j)=ichild(i,j)
                  iparent(i+1,j)=ichild(i+1,j)
200           continue
              do 300 j=icross,nchrome
                  iparent(i,j)=ichild(i+1,j)
                  iparent(i+1,j)=ichild(i,j)
300           continue
          else
              do 400 j=1,nchrome
                  iparent(i,j)=ichild(i,j)
                  iparent(i+1,j)=ichild(i+1,j)
400           continue
          end if
100   continue

      RETURN
      END
```

Uniform Crossover

For each pair of consecutive strings in the *iparent* array, if *rand1* is less than or equal to the crossover probability *pcross*, the following subroutine performs uniform crossover to generate two new strings. This is done until the *ichild* array is filled.

```
      SUBROUTINE ucross(pcross)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION ichild(npopmax,nchrmax),iparent(npopmax,nchrmax)
      COMMON / gen2 / iparent,ichild
      COMMON / gen3 / nbits,nparam,modpop,nchrome

      do 100 i=1,modpop,2
          rand1=ran1(idum)
          if (rand1.le.pcross) then
              do 200 j=1,nchrome
                  rand2=ran1(idum)
                  if (rand2.le.0.5) then
                      iparent(i,j)=ichild(i,j)
                      iparent(i+1,j)=ichild(i+1,j)
                  else
                      iparent(i,j)=ichild(i+1,j)
                      iparent(i+1,j)=ichild(i,j)
                  end if
200           continue
          else
              do 300 j=1,nchrome
                  iparent(i,j)=ichild(i,j)
                  iparent(i+1,j)=ichild(i+1,j)
300           continue
          end if
100   continue

      RETURN
      END
```

Mutation

The following subroutine performs mutation on the entire population of binary strings. For each bit in the array *ichild*, if *randn* is less than or equal to the mutation probability *pmutate*, the bit is flipped (note that this gives a mutation rate twice as high as if a new bit is generated).

```
      SUBROUTINE mutate(ichild)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION ichild(npopmax,nchrmax)
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen4 / pcross,pmutate

      do 100 j=1,modpop
          do 110 k=1,nchrome
             randn=ran1(idum)
             if (randn.le.pmutate) then
                 if (ichild(j,k).eq.0) then
                     ichild(j,k)=1
                 else
                     ichild(j,k)=0
                 end if
             end if
110       continue
100   continue

      RETURN
      END
```

Tournament Selection

The following subroutine selects the fittest members of the population according to the "Tournament" method, in which two members of the parent array are are drawn randomly, and the one with the higher fitness fills a slot in the intermediate (pre-crossover) *ichild* array. This is repeated until the *ichild* array is filled.

```
      SUBROUTINE tselect

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION parent(npopmax,nparmax),fit(npopmax)
      DIMENSION ichild(npopmax,nchrmax),iparent(npopmax,nchrmax)
      COMMON / gen1 / parent,fit
      COMMON / gen2 / iparent,ichild
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen7 / islpop,nisland,xmixprob

      do 100 k=1,nisland
          kdum=(k-1)*islpop
          do 200 i=1,islpop
              iran1=int(real(islpop)*ran1(idum))+1+kdum
              if (iran1.gt.islpop+kdum) iran1=islpop+kdum
210           iran2=int(real(islpop)*ran1(idum))+1+kdum
```

```
            if (iran2.gt.islpop+kdum) iran2=islpop+kdum
            if (iran2.eq.iran1) go to 210
            if (fit(iran2).gt.fit(iran1)) then
                mate=iran2
            else
                mate=iran1
            end if
            do 220 j=1,nchrome
                ichild(i+kdum,j)=iparent(mate,j)
220         continue
200     continue
100 continue

    RETURN
    END
```

Roulette Wheel Selection

The following subroutine selects the fittest members of the population according to the "Roulette Wheel" method, in which an individual's chance of being selected is proportional to its fitness. Slots in the intermediate (pre-crossover) *ichild* array are filled by repeated spins.

```
    SUBROUTINE rselect

    PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
    DIMENSION fit(npopmax),parent(npopmax,nparmax)
    DIMENSION ichild(npopmax,nchrmax),iparent(npopmax,nchrmax)
    COMMON / gen1 / parent,fit
    COMMON / gen2 / iparent,ichild
    COMMON / gen3 / nbits,nparam,modpop,nchrome
    COMMON / gen7 / islpop,nisland,xmixprob

    do 100 k=1,nisland
        kdum=(k-1)*islpop
        pfitsum=0.0
        do 110 l=1,islpop
            pfitsum=pfitsum+fit(l+kdum)
110     continue
        do 200 i=1,islpop
            partsum=0.0
            rand=ran1(idum)*pfitsum
            do 300 j=1,islpop
                partsum=partsum+fit(j+kdum)
                if (partsum.ge.rand) then
                    mate=j+kdum
```

```
                    go to 400
                end if
300             continue
400             continue
                do 500 n=1,nchrome
                    ichild(i+kdum,n)=iparent(mate,n)
500             continue
200         continue
100     continue

        RETURN
        END
```

Stochastic Universal Selection

The following subroutine selects the fittest members of a population by the Stochastic Universal method, an efficient implementation of Stochastic Remainder selection.

```
        SUBROUTINE uselect

        PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
        DIMENSION fit(npopmax),parent(npopmax,nparmax)
        DIMENSION ichild(npopmax,nchrmax),iparent(npopmax,nchrmax)
        COMMON / gen1 / parent,fit
        COMMON / gen2 / iparent,ichild
        COMMON / gen3 / nbits,nparam,modpop,nchrome
        COMMON / gen7 / islpop,nisland,xmixprob

        do 100 k=1,nisland
            kdum=(k-1)*islpop
            pfitsum=0.0
            do 110 l=1,islpop
                pfitsum=pfitsum+fit(l+kdum)
110         continue
            divf=pfitsum/real(islpop)
            rand=ran1(idum)*divf
            addsum=rand
            do 200 i=1,islpop
                partsum=0.0
                do 300 j=1,islpop
                    partsum=partsum+fit(j+kdum)
                    if (partsum.ge.addsum) then
                        do 400 n=1,nchrome
                            ichild(i+kdum,n)=iparent(j,n)
400                     continue
                        addsum=addsum+divf
                        go to 200
```

```
                end if
300           continue
200       continue
100   continue
      call shuffle(ichild)

      RETURN
      END
```

Shuffling

The following subroutine shuffles the position of each member of the *ichild* array in preparation for two by two crossover. This is required after using Stochastic Universal Selection.

```
      SUBROUTINE shuffle(ichild)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION ichild(npopmax,nchrmax),itemp(nchrmax)
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen7 / islpop,nisland,xmixprob

      do 100 k=1,nisland
          kdum=(k-1)*islpop
          do 200 i=1,islpop
              iran1=int(real(islpop)*ran1(idum))+1+kdum
              do 300 j=1,nchrome
                  itemp(j)=ichild(i+kdum,j)
300           continue
              do 400 j=1,nchrome
                  ichild(i+kdum,j)=ichild(iran1,j)
400           continue
              do 500 j=1,nchrome
                  ichild(iran1,j)=itemp(j)
500           continue
200       continue
100   continue

      RETURN
      END
```

Evaluating the population

This subroutine evaluates the fitness of the model population by decoding the binary strings into sets of real parameters, calling the fitness function *func* for each

set and then converting the objective function value into a fitness value in the following way:

if minimization is the object,

$$fit(i) = \max(obj) - obj(i)$$

if maximization is the object,

$$fit(i) = obj(i)$$

where $fit(i)$ is the fitness function value for the i^{th} member of the population, $\max(obj)$ is the maximum objective function value for the entire population, and $obj(i)$ is the objective function value for the i^{th} member of the population.

If a selection method other than tournament selection is used, linear fitness scaling is applied and if istr=1, sigma truncation is performed to remove low fitness "lethals" from the population.

```
      SUBROUTINE evaluate(icount,min,ibest,kbest,fmax,fitavg)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION parent(npopmax,nparmax),iparent(npopmax,nchrmax)
      DIMENSION fit(npopmax),objfunc(npopmax)
      DIMENSION ichild(npopmax,nchrmax)
      DIMENSION ibest(nislmax,nchrmax),kbest(nislmax)
      COMMON / gen1 / parent,fit
      COMMON / gen2 / iparent,ichild
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen5 / iselect,smult,stmult

30    format (i4,4(1x,f10.3))
31    format (i4,6(1x,f10.3))

      omax=0.0
      osum=0.0
      fmax=0.0
      fitsum=0.0
c-----Decode the binary strings to real number arrays and test
c-----their fitness with the objective function
      do 460 j=1,modpop
          call decode(j,parent,iparent)
          objfunc(j)=func(j,parent)
```

```
          osum=osum+objfunc(j)
          if (objfunc(j).gt.omax) then
              omax=objfunc(j)
              jtemp1=j
          end if
460       continue
          omin=omax
          do 470 j=1,modpop
          if (objfunc(j).lt.omin) then
              omin=objfunc(j)
              jtemp2=j
          end if
470       continue
          if (icount.eq.1) omperm=omax
c-----If the object is minimization, map the objective function
c-----into a fitness function
          if (min.eq.1) then
              jworst=jtemp1
              jbest=jtemp2
              do 480 j=1,modpop
                  fit(j)=omax-objfunc(j)
480           continue
              call findbest(fmax,ibest,fitsum,sigma,kbest)
              fitavg=fitsum/real(modpop)
              if (icount.eq.imax) then
                  call output(kbest)
              end if
              if (iselect.eq.0) go to 540
c-----If istr=1, apply sigma truncation to fitnesses
              if (istr.eq.1) then
                  call strunc(fit,fitavg,sigma)
              end if
c-----Scale fitness for minimization by linear scaling
              call lscale(fit,fitavg,fmax,fitsum,sfitsum,smult)
              sfmax=fit(jbest)
530           sfitavg=sfitsum/real(modpop)
              write (4,31) icount,omin,fitavg,
     &              omperm-omin,sfitavg,sfmax
540           if (iselect.eq.0) then
                  write (4,30) icount,omin,fitavg,fmax
              end if
c-----If maximization is the object, use objective function for
c-----fitness function
          else
              do 600 j=1,modpop
                  fit(j)=objfunc(j)
600           continue
              jbest=jtemp1
              fitavg=osum/real(modpop)
              fmax=omax
              sfitsum=osum
              write (4,*) icount,fitavg,fmax
              call findbest(fmax,ibest,fitsum,sigma,kbest)
          end if
```

```
      RETURN
      END
```

Migration between islands

If the population is divided into islands, this subroutine mixes the members according to the parameter *xmixprob*. It is called every *mixint* generations.

```
      SUBROUTINE migrate(ichild)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION ichild(npopmax,nchrmax),itemp(nchrmax)
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen7 / islpop,nisland,xmixprob

      do 100 i=1,modpop
          rand1=ran1(idum)
          if (rand1.le.xmixprob) then
              iran2=int(real(modpop)*ran1(idum))+1
              do 200 j=1,nchrome
                  itemp(j)=ichild(i,j)
200           continue
              do 300 j=1,nchrome
                  ichild(i,j)=ichild(iran2,j)
300           continue
              do 400 j=1,nchrome
                  ichild(iran1,j)=itemp(j)
400           continue
          end if
100   continue

      RETURN
      END
```

Decoding the binary strings into real numbers

The following subroutine decodes the binary parent array of dimension (*nchrome* X *modpop*) into a real number array of dimension (*nparam* X *modpop*).

```
      SUBROUTINE decode(i,parent,iparent)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION parent(npopmax,nparmax),iparent(npopmax,nchrmax)
```

```
      DIMENSION parmin(nparmax),parmax(nparmax)
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen6 / parmin,parmax

      m=0
      do 100 k=1,nparam
          iparam=0
          sum=0.0
          do 200 j=1,nbits
              iparam=iparam+iparent(i,j+m*nbits)*(2**(nbits-j))
              sum=sum+2.0**(j-1)
200       continue
          parent(i,k)=parmin(k)+real(iparam)*(parmax(k)-
     &    parmin(k))/sum
          m=m+1
100   continue

      RETURN
      END
```

Elitism

The following subroutine implements elitism if *ielite* = 1 by first checking to see if the best individual in each island was reproduced and replicating the model into a random slot in the island if not.

```
      SUBROUTINE elite(ibest)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION iparent(npopmax,nchrmax)
      DIMENSION ichild(npopmax,nchrmax)
      DIMENSION ibest(nislmax,nchrmax)
      COMMON / gen2 / iparent,ichild
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen7 / islpop,nisland,xmixprob

c-----Check to see if best parent was replicated in each
c-----population
      nelite=0
      do 100 l=1,nisland
          ldum=(l-1)*islpop
          do 200 i=1,islpop
              do 210,j=1,nchrome
                  if (iparent(i+ldum,j).eq.ibest(l,j)) nelite=1
                  if (nelite.eq.nchrome) melite=1
210           continue
200       continue
c-----If the best parent was not replicated, replicate in a random
```

```
c-----slot
          if (melite.eq.0) then
              rand=ran1(idum)
              irand=1+int(real(islpop)*rand)+ldum
              do 120 k=1,nchrome
                  iparent(irand,k)=ibest(1,k)
120           continue
          end if
100   continue

      RETURN
      END
```

Finding the best model in each island

This subroutine finds the best model in each island and copies the binary parameters of each into the array *ibest*.

```
      SUBROUTINE findbest(fmax,ibest,fitsum,sigma,kbest)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION fit(npopmax),iparent(npopmax,nchrmax)
      DIMENSION ibest(nislmax,nchrmax),kbest(nislmax)
      DIMENSION parent(npopmax,nparmax),ichild(npopmax,nchrmax)
      COMMON / gen1 / parent,fit
      COMMON / gen2 / iparent,ichild
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen5 / iselect,smult,stmult
      COMMON / gen7 / islpop,nisland,xmixprob

      var=0.0
      fitsum=0.0
      do 100 k=1,nisland
          fmax=0.0
          kdum=(k-1)*islpop
          do 200 i=1,islpop
              fitsum=fitsum+fit(i+kdum)
              var=var+fit(i+kdum)**2.0
              if (fit(i+kdum).gt.fmax) then
                  fmax=fit(i+kdum)
                  kbest(k)=i+kdum
                  do 110 j=1,nchrome
                      ibest(k,j)=iparent(i+kdum,j)
110               continue
              end if
200       continue
100   continue
      sigma=sqrt(var/real(modpop))
```

```
      RETURN
      END
```

Sigma Truncation

This subroutine applies Sigma Truncation to the fitness array. This is useful to remove low fitness "lethals" from the population and allow a better linear scaling.

```
      SUBROUTINE strunc(fit,fitavg,sigma)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION fit(npopmax)
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen5 / iselect,smult,stmult

      do 100 n=1,modpop
          fit(n)=fit(n)-(fitavg-stmult*sigma)
          if (fit(n).lt.0.0) fit(n)=0.0
100   continue

      RETURN
      END
```

Linear Scaling

This subroutine applies linear scaling to the fitness array. The fitnesses are pivoted about the average value in order to lessen the slope early in the run and to steepen it later in the run.

```
      SUBROUTINE lscale(fit,fitavg,fmax,fitsum,sfitsum)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION fit(npopmax)
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen5 / iselect,smult,stmult

      sfitsum=0.0
      if (fmax.gt.(smult*fitavg)) then
          delta=fmax-fitavg
          a=(smult-1.0)*fitavg/delta
```

```
            b=fitavg*(fmax-smult*fitavg)/delta
        else
            sfitsum=fitsum
            go to 200
        end if
        do 100 n=1,modpop
            fit(n)=a*fit(n)+b
            sfitsum=sfitsum+fit(n)
100     continue

200     RETURN
        END
```

Writing to the output file

This subroutine writes the best model parameters for each island to the file *genalg.out*.

```
        SUBROUTINE output(kbest)

        PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
        DIMENSION parent(npopmax,nparmax)
        DIMENSION fit(npopmax)
        DIMENSION kbest(nislmax)
        COMMON / gen1 / parent,fit
        COMMON / gen3 / nbits,nparam,modpop,nchrome
        COMMON / gen7 / islpop,nisland,xmixprob

17      format (i3,1x,f8.3,1x,48(1x,f8.3))

        open (unit=12,file='genalg.out',status='unknown')
        do 100 j=1,nisland
            jdum=kbest(j)
            write (12,17)
     &      jdum,fit(jdum),(parent(jdum,k),k=1,nparam)
100     continue
        close (12)

        RETURN
        END
```

Writing to the restart file

This subroutine writes the binary *iparent* array to the file *restart.inp*. It is called every *nrestart* generations.

```
      SUBROUTINE wrestart(iparent)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384,nislmax=100)
      DIMENSION iparent(npopmax,nchrmax)
      COMMON / gen3 / nbits,nparam,modpop,nchrome

20    format (384(i2))

      open (unit=30,file='restart.inp',status='unknown')
      rewind 30
      do 100 i=1,modpop
          write (30,20) (iparent(i,j),j=1,nchrome)
100   continue
      close (30)

      RETURN
      END
```

Partially Matched Crossover (PMX) inversion operator

This subroutine implements Goldberg's Partially Matched Crossover inversion operator, combining inversion and crossover into one step.

Two new arrays must be added to the main routine that contain the order of the binary parent array and the binary child array. Also, the decoding subroutine must be modified slightly (see below).

```
      SUBROUTINE order(pcross)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384)
      DIMENSION ichild(npopmax,nchrmax),iparent(npopmax,nchrmax)
      DIMENSION iplocus(npopmax,nchrmax),iclocus(npopmax,nchrmax)
      DIMENSION itemp1(nchrmax),itemp2(nchrmax)
      DIMENSION itemp3(nchrmax),itemp4(nchrmax)
      COMMON / gen2 / iparent,ichild
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen7 / iplocus,iclocus

27    format (48(1x,I2))
      do 100 j=1,modpop,2
          rand1=ran1(idum)
          if (rand1.gt.pcross) go to 100
          irand1=1+int(real(nchrome-1)*ran1(idum))
110       irand2=1+int(real(nchrome-1)*ran1(idum))
          if (irand2.eq.irand1) go to 110
```

```
      icross1=min(irand1,irand2)
      icross2=max(irand1,irand2)
      do 200 k=1,nchrome
          iplocus(j,k)=iclocus(j,k)
          iplocus(j+1,k)=iclocus(j+1,k)
          do 210 i=icross1+1,icross2
              if (iclocus(j,k).eq.iclocus(j+1,i))
     &        iplocus(j,k)=-1
              if (iclocus(j+1,k).eq.iclocus(j,i))
     &        iplocus(j+1,k)=-1
210       continue
200   continue
      m=1
      do 410 k=icross2+1,nchrome
          itemp1(m)=iplocus(j,k)
          itemp2(m)=iplocus(j+1,k)
          itemp3(m)=ichild(j,k)
          itemp4(m)=ichild(j+1,k)
          m=m+1
410   continue
      do 420 k=1,icross2
          itemp1(m)=iplocus(j,k)
          itemp2(m)=iplocus(j+1,k)
          itemp3(m)=ichild(j,k)
          itemp4(m)=ichild(j+1,k)
          m=m+1
420   continue
      do 500 k=1,nchrome+icross1-icross2
          if (itemp1(k).lt.0) then
              do 510 l=k,nchrome
                  if (itemp1(l).gt.0) then
                  itemp1(k)=itemp1(l)
                  itemp3(k)=itemp3(l)
                  itemp1(l)=-1
                  go to 500
                  end if
510               continue
          end if
500   continue
      do 550 k=1,nchrome+icross1-icross2
          if (itemp2(k).lt.0) then
              do 560 l=k,nchrome
                  if (itemp2(l).gt.0) then
                      itemp2(k)=itemp2(l)
                      itemp4(k)=itemp4(l)
                      itemp2(l)=-1
                      go to 550
                  end if
560               continue
          end if
550   continue
      m=1
      do 600 k=icross2+1,nchrome
          iplocus(j,k)=itemp1(m)
```

```
            iplocus(j+1,k)=itemp2(m)
            iparent(j,k)=itemp3(m)
            iparent(j+1,k)=itemp4(m)
            m=m+1
600       continue
          do 710 k=1,icross1
            iplocus(j,k)=itemp1(m)
            iplocus(j+1,k)=itemp2(m)
            iparent(j,k)=itemp3(m)
            iparent(j+1,k)=itemp4(m)
            m=m+1
710       continue
          do 810 k=icross1+1,icross2
            iplocus(j,k)=iclocus(j+1,k)
            iplocus(j+1,k)=iclocus(j,k)
            iparent(j,k)=ichild(j+1,k)
            iparent(j+1,k)=ichild(j,k)
810       continue
100   continue

      RETURN
      END
```

This routine decodes a binary string into a real number. It is modified slightly to handle allele information.

```
      SUBROUTINE decode(i,parent,iparent,iplocus)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384)
      DIMENSION parent(npopmax,nparmax),iparent(npopmax,nchrmax)
      DIMENSION parmin(nparmax),parmax(nparmax)
      DIMENSION iplocus(npopmax,nchrmax)
      DIMENSION itemp(nchrmax)
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen6 / parmin,parmax

      do 100 k=1,nchrome
          idum1=iplocus(i,k)
          itemp(idum1)=iparent(i,k)
100   continue
      m=0
      do 200 k=1,nparam
          iparam=0
          sum=0.0
          do 300 j=1,nbits
              iparam=iparam+itemp(j+m*nbits)*(2**(nbits-j))
              sum=sum+2.0**(j-1)
300       continue
          parent(i,k)=parmin(k)+real(iparam)*(parmax(k)-
```

```
     &                  parmin(k))/sum
          m=m+1
200   continue

      RETURN
      END
```

Niching

The following subroutine adjusts fitness according to diversity. It uses Goldberg's triangular sharing function (Goldberg, 1989) to continuously update sharing and calculate the normalized similarity from members of new generation instead of parent generation (Oei, 1991). This method solves the instability problems of the standard niching approach. A binary array must be added to the main routine which stores a "1" for each parameter that niching is desired for and a "0" for the ones that it is not. This routine must be called by the modified tournament selection routine given below, and the main program must also be modified so that the new generation slots are filled by calling the tournament selection routine repeatedly until the slots are filled.

```
      SUBROUTINE niche(ii,jpick,sumshar)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384)
      DIMENSION fit(npopmax)
      DIMENSION parent(npopmax,nparmax),child(npopmax,nparmax)
      DIMENSION parmin(nparmax),parmax(nparmax)
      DIMENSION iniche(nparmax)
      COMMON / gen1 / parent,child,fit
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen6 / parmin,parmax
      COMMON / gen7 / iniche,nichesum

      itotal=ii-1
      sumshar=0.0
      do 100 j=1,itotal
          del2=0.0
          do 200, k=1,nparam
```

```
            if (iniche(k).eq.1) then
                del2=del2+((parent(jpick,k)-child(j,k))/
     &                (parmax(k)-parmin(k)))**2.0
            end if
200       continue
          del1=sqrt((del2)/float(nichesum))
          if (del1.lt.float(nichesum)) then
              sumshar=sumshar+1-del1
          else
              sumshar=sumshar+1.0
          end if
100   continue
      sumshar=sumshar/float(itotal)

      RETURN
      END
```

This is the modified tournament selection routine that must be used with the niching routine:

```
      SUBROUTINE tselect(i,mate,fit)

      PARAMETER (npopmax=200,nparmax=48,nchrmax=384)
      DIMENSION fit(npopmax)
      DIMENSION iniche(nparmax)
      COMMON / gen3 / nbits,nparam,modpop,nchrome
      COMMON / gen7 / iniche,nichesum

      iran1=int(real(modpop)*ran1(idum))+1
      if (iran1.gt.modpop) iran1=modpop
100   iran2=int(real(modpop)*ran1(idum))+1
      if (iran2.eq.iran1) go to 100
      if (iran2.gt.modpop) iran2=modpop
      if (i.gt.2.and.nichesum.ne.0) then
          call niche(i,iran1,sumshare1)
          call niche(i,iran2,sumshare2)
          fit1=fit(iran1)/sumshare1
          fit2=fit(iran2)/sumshare2
      else
          fit1=fit(iran1)
          fit2=fit(iran2)
      end if
      if (fit2.gt.fit1) then
          mate=iran2
      else
          mate=iran1
      end if

      RETURN
      END
```

Appendix B: A pattern search algorithm in FORTRAN

This Pattern search algorithm roughly follows the method outlined in the paper by Hooke and Jeeves (1961), with the exception that the step size is decreased exponentially rather than arithmetically. This code is currently set to handle a maximum of 200 parameters. The parameter statements must be changed in the main program and the subroutines in order to use higher values.

Program variable definitions

base	The basepoint (starting model).
maxit	Maximum number of iterations.
min	= 0 for function Maximization.
	= 1 for function Minimization.
newbase	The next basepoint.
nparam	Number of input parameters.
nparmax	Maximum number of parameters allowed.
step	= Initial step size (should be on the order of 1/10 of the desired search space).
xfac	= The factor that is multiplied by the initial step size after each iteration.

Subroutine Descriptions

adjust	Adjusts the step size.
explore	Takes exploratory steps.
output	Writes output to converge.out.
pattern	Makes pattern move.

Input files

Sample pattern.in file:

```
5              //nparam
1              //min
10             //maxit
5              //maxrun
3.4  0.4  0.5   //base(1),step(1),xfac(1)
4.6  0.4  0.5   //base(2),step(2),xfac(2)
5.1  0.5  0.5   //base(3),step(3),xfac(3)
5.2  0.5  0.5   //base(4),step(4),xfac(4)
5.3  0.5  0.5   //base(5),step(5),xfac(5)
```

Main Routine

```
      PROGRAM PATTERN

      PARAMETER (nparmax=200)
      DIMENSION base(nparmax),trial(nparmax)
      DIMENSION step(nparmax),xistep(nparmax),xfac(nparmax)

13    format (A26,I6)
18    format (i6,3x,f9.4)

c-----Open input and output files
c-----Read in input file parameters
      open (unit=1,file='pattern.in',status='old')
      read (1,*) nparam
      read (1,*) min
      read (1,*) maxit
      read (1,*) maxrun
      do 100 i=1,nparam
          read(1,*) base(i),step(i),xfac(i)
          xistep(i)=step(i)
100   continue
      close (1)
c-----Initiate iteration and run counter
      icount=1
      irun=1
      open (unit=4,file='converge.out',status='unknown')
      obj0=func(base,nparam)
      obj1=obj0
      write (4,18) 0,obj1
      iflag=0
c-----Begin main processing loop
200   continue
      objt=obj1
c-----Make exploratory moves and if successful a pattern move
```

```
      call explore(base,trial,step,nparam,objt)
      if (objt.lt.obj1) then
          obj1=objt
c-----If pattern moves and trial moves are both unsuccessful
c-----adjust the step size
      else
          if (icount.lt.maxit) then
              call adjust(step,xfac,nparam)
              idum1=icount+(irun-1)*maxit
              print 13, 'Finished Iteration Number ',idum1
              write (4,18) idum1,obj1
              call flush (4)
              icount=icount+1
          else
              go to 300
          end if
          go to 200
      end if
      go to 200
300   if (irun.lt.maxrun) then
          do 400 i=1,nparam
              step(i)=xistep(i)
400       continue
          irun=irun+1
          icount=1
          go to 200
      end if
      close (4)
c-----When finished, write results to output
      call output(base,nparam)
      print *, ''
      print 16, '--- Finished run successfully ---'

      STOP
      END
```

Exploratory moves

This subroutine makes exploratory moves, perturbing each parameter one at a time and checking to see if the solution is improved. If there is overall improvement a pattern move can be made, and if not the step size is decreased.

```
      SUBROUTINE explore(base,trial,step,nparam,obj0)

      PARAMETER (nparmax=200)
      DIMENSION base(nparmax),trial(nparmax)
```

```
      DIMENSION step(nparmax)

      obj1=obj0
      do 100 i=1,nparam
          trial(i)=base(i)
100   continue
c-----Make exploratory steps for each parameter
c-----individually.
      do 200 i=1,nparam
          trial(i)=base(i)+step(i)
          obj2=func(trial,nparam)
          if (obj2.gt.obj1) then
              trial(i)=base(i)-step(i)
              obj2=func(trial,nparam)
              if (obj2.gt.obj1) then
                  trial(i)=base(i)
              else
                  obj1=obj2
              end if
          else
              obj1=obj2
          end if
200   continue
c-----If exploratory moves are successful trial point
c-----becomes new basepoint, base and trial are switched,
c-----and pattern move is made
      if (obj2.lt.obj0) then
          do 300 i=1,nparam
              dum1=trial(i)
              trial(i)=base(i)
              base(i)=dum1
300       continue
          call pattern(base,trial,nparam,obj2)
          obj0=obj2
      end if

      RETURN
      END
```

Pattern move

If exploratory moves are successful, the successful series of moves is repeated here if it improves the solution. This step makes the algorithm very efficient because a major improvement in the solution can be made with only 1 objective function evaluation.

```
      SUBROUTINE pattern(base,trial,nparam,obj0)

      PARAMETER (nparmax=200)
      DIMENSION base(nparmax),trial(nparmax),test(nparmax)

      do 100 i=1,nparam
          diff=base(i)-trial(i)
          test(i)=base(i)-diff
100   continue
      obj1=func(test,nparam)
      if (obj1.lt.obj0) then
          do 200 i=1,nparam
              base(i)=test(i)
200       continue
          obj0=obj1
      end if

      RETURN
      END
```

Decreasing the step size

If exploratory moves are unsuccessful, the step size vector is decreased by multiplying it by the factor *xfac*.

```
      SUBROUTINE adjust(step,xfac,nparam)

      PARAMETER (nparmax=200)
      DIMENSION step(nparmax),xfac(nparmax)

      do 100 i=1,nparam
          step(i)=step(i)*xfac(i)
100   continue

      RETURN
      END
```

Writing to the output file

This subroutine writes the best model parameters for each island to the file *params.out*.

```
      SUBROUTINE output(param,nparam)

      PARAMETER (nparmax=200)
      DIMENSION param(nparmax)

10    format (i5,2x,f10.5)
      open (unit=12,file='params.out',status='unknown')
      do 100 i=1,nparam
          write (12,10) i,param(i)
100   continue
      close (12)

      RETURN
      END
```

Ernest Orlando Lawrence Berkeley National Laboratory
One Cyclotron Road | Berkeley, California 94720

Prepared for the U.S. Department of Energy under Contract No. DE-AC03-76SF00098