12-2002

Data Mining with Newton's Method.

James Dale Cloyd
East Tennessee State University

Data Mining with Newton's Method

A thesis
presented to
the faculty of the Department of Computer and Information Sciences
East Tennessee State University

In partial fulfillment
of the requirements for the degree
Masters of Science in Computer Science

---

by
James Dale Cloyd
December 2002

Donald Sanderson, Chair
Laurie Moffit
Kellie Price

Keywords: Genetic algorithms, evolutionary operations, neural networks, simplex evop, maximum likelihood estimation, non-linear regression, robust regression, knowledge discovery in databases.

ABSTRACT

Data Mining with Newton's Method

by

James D. Cloyd

Capable and well-organized data mining algorithms are essential and fundamental to helpful, useful, and successful knowledge discovery in databases. We discuss several data mining algorithms including genetic algorithms (GAs). In addition, we propose a modified multivariate Newton's method (NM) approach to data mining of technical data. Several strategies are employed to stabilize Newton's method to pathological function behavior. NM is compared to GAs and to the simplex evolutionary operation algorithm (EVOP). We find that GAs, NM, and EVOP all perform efficiently for well-behaved global optimization functions with NM providing an exponential improvement in convergence rate. For local optimization problems, we find that GAs and EVOP do not provide the desired convergence rate, accuracy, or precision compared to NM for technical data. We find that GAs are favored for their simplicity while NM would be favored for its performance.

ACKNOWLEDGMENTS

The student is grateful to Dr. Terry Countermine and the faculty and staff of the Computer and Information Sciences Department of East Tennessee State University who took the time to schedule and teach evening classes for folks who work full time and would otherwise be unable to attend day classes. The financial support of Eastman Chemical Company through its employee tuition refund program is gratefully acknowledged. The support and assistance of the chair members, Drs. Laurie Moffit and Kellie Price, are greatly appreciated. The guidance, leadership, support, and assistance of the thesis chair, Dr. Donald Sanderson, are gratefully acknowledged. The student would like to acknowledge the late Professor Jerry Sayers whose teaching skills, scientific viewpoints, and leadership were very influential and he will be missed both as a friend and a teacher.

CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

Data mining and Knowledge Discovery in Databases have become commercially important techniques and active areas of research in recent years. Business applications of data mining software are commonplace and are commodities in many cases. However, data mining of technical data is still a relatively disorganized discipline compared to business applications of data mining. For example, the application of neural networks trained by genetic algorithms to a business' market basket analysis procedures would not be unusual. The use of informatics, a field that is similar to On-line Analytical Processing (OLAP), in biology and chemistry is increasing, however. There is an increasing need for data mining algorithms with scientific precision.

In this work, we survey the algorithms of data mining and propose several new algorithms for data mining. Specifically, we show how Newton's method, especially local Newton's method, could be applied to data mining applications for technical data – the method may also find uses in specialized business applications as well, *i.e.*, non-marketing applications. We also discuss genetic algorithms (GA), the fixed simplex evolutionary operation (EVOP), and the variable length simplex EVOP. GA and EVOP are evolutionary algorithms. GAs use a stochastic process and EVOPs use a deterministic process.

In the next chapter, a literature survey of data mining is given. In the following chapters, we develop an algorithm based on Newton's method as a data mining algorithm for applications involving technical data. Chapter 3 (Newton's Method) is a literature survey of Newton's method (NM), explains quadratic convergence, gives the NM convergence criteria, and illustrates convergence criteria with examples from chaos theory. Chapter 4 (Modeling and

Newton's Method) explains how Newton's method fits into modeling theory and describes the local Newton's method, global Newton's method, non-linear regression, and robust non-linear regression.

Chapter 5 (Matrix Algebra) explains the methods necessary to implement Newton's method for higher dimensional problems. The derivation of NM from the method of maximum likelihood estimation is given. And, the variance-covariance matrix for NM is derived such that statistical analysis of NM results can be obtained. Chapter 6 (Results and Discussion) gives the comparison of using Newton's method, the simplex EVOP methods, and genetic algorithms on some model problems in terms of precision, accuracy, and convergence rate. Chapter 7 (Comparison of Algorithms) compares these algorithms in terms of computational steps required, the storage space required, and the complexity of the algorithms.

CHAPTER 2

DATA MINING LITERATURE SURVEY

Computer scientists often refer to Moore's law, which states that computer processing speed doubles about every 18 months. It is less well known that computer storage capacity doubles about every nine months (Goebel and Gruenwald 1999). Like an ideal gas, computer databases expand to fill available storage capacity. The resulting large amounts of data in databases represent an untapped resource. Like a gold mine, these data could be extracted into information. That information could then be converted to valuable knowledge with data mining techniques.

It is difficult to convey the vast amount of unused data stored in very large databases at companies, universities, government facilities, and other institutions throughout the world and its current rate of increase. The Library of Congress is estimated to contain 3 petabytes (3000 terebytes) of information (Lesk 1997). Lesk estimates that about 160 terebytes of information are produced each year worldwide. And, he estimates there will be over 100,000 terebytes of disk space sold. It could soon be the case that computer data storage will exceed human capability to use that data storage and the data it contains. A process for converting large amounts of data to knowledge will become invaluable. A process called Knowledge Discovery in Databases (KDD) has evolved over the past ten to fifteen years for this purpose. Data mining algorithms are included in the KDD process.

A typical database user retrieves data from databases using an interface to standard technology such as SQL. A data mining system takes this process a step further, allowing users to discover new knowledge from the data (Adriaans and Zantinge 1996, 855). Data mining, from a computer scientist's point of view, is an interdisciplinary field. Data handling techniques such as neural networks, genetic algorithms, regression, statistical analysis, machine learning, and

cluster analysis are prevalent in the literature on data mining. Many researchers state that data mining is not yet a well-ordered discipline. The major opportunities for improvement in data mining technology are scalability and compatibility with database systems, as well as the usability and accuracy of data mining techniques. We will also discuss the issue of moving the data from secondary storage to main memory – data access will probably become the rate-limiting step for data mining of large databases.

Data Mining and Knowledge Discovery

Most authors have different definitions for data mining and knowledge discovery. Goebel and Gruenwald (G&G) define knowledge discovery in databases (KDD) as "the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data" and data mining as "the extraction of patterns or models from observed data." (Goebel and Gruenwald 1999) Berzal *et al.* define KDD as "the non-trivial extraction of potentially useful information from a large volume of data where the information is implicit (although previously unknown)." G&G's model of KDD, paraphrased below, shows data mining as one step in the overall KDD process:

1. Identify and develop an understanding of the application domain.
2. Select the data set to be studied.
3. Select complimentary data sets. Integrate the data sets.
4. Code the data. Clean the data of duplicates and errors. Transform the data.
5. Develop models and build hypotheses.
6. Select appropriate data mining algorithms.
7. Interpret results. View results using appropriate visualization tools.
8. Test results in terms of simple proportions and complex predictions.
9. Manage the discovered knowledge.

Although data mining is only a part of the KDD process, data mining techniques provide the algorithms that fuel the KDD process. The KDD process shown above is a never-ending process. Data mining is the essence of the KDD process. If data mining is being discussed, it is understood that the process of KDD is being used. In this work, we will focus on data mining algorithms.

Adriaans and Zantinge (A&Z) (Adriaans and Zantinge 1996, 5) emphasize that the KDD community reserves the term data mining for the discovery stage of the KDD process. Their definition of KDD is as follows: "... the non-trivial extraction of implicit, previously unknown and potentially useful knowledge from data." Similarly, Berzal *et al.* define data mining as "a generic term which covers research results, techniques and tools used to extract useful information from large databases." Also, A&Z point out that KDD draws on techniques from the fields of expert systems, machine learning, statistics, visualization, and database technology.

Comaford addresses some misconceptions about data mining (Comaford 1997). In Comaford's view, data mining is not the same thing as data warehousing or data analysis. Data mining is a dynamic process that enables a more intelligent use of a data warehouse than data analysis. Data mining builds models that can be used to make predictions without additional SQL queries. Data mining techniques apply to both small and very large data sets. Instead of considering just the size of the data set, one must include appropriate width, depth, and volume as three important requirements. Effective data mining requires many attributes for the database records (width), a large number of records that are instances of the database entities (depth) and many entities determined by the database design (volume). Data mining is most appropriate for customer-oriented applications instead of for general business applications. Data mining does not necessarily require artificial intelligence (AI). If a data mining algorithm uses AI, it should be invisible to the user. That is, Comaford does not see data mining as a general business tool except for customer-oriented applications. For commercial data mining applications, this assessment of data mining may be true. This assessment underscores the need for data mining applications for technical data.

A&Z take a different viewpoint than Comaford in regard to width, depth, and volume. According to Comaford, join operations eliminate the need for a volume definition by collapsing a database's attributes of interest into a set of related records. A&Z, on the other hand, consider data mining as an exploration of a multidimensional space of data. Consider a database with one entity and with a million records. If the database has one attribute, it has only one dimension. Suppose this dimension is scaled from 0 to 100 with a resolution of one part per hundred. For one million records there are on average 10,000 records per unit of space or per unit length in the one-dimensional case. For two attributes and two dimensions, there are on average 100 records

per unit area. For three attributes, there is on average only one record per unit volume. To put this number in perspective, consider that the vacuum of space contains about one to two atoms per cubic inch (Elert 1987). Thus, the data mining space of a three attribute database with one million records is an extremely low density space. Furthermore, if the database has ten attributes, then the density of records is 10^{-14} records per unit hypervolume. The point of this analogy is that hyperspace becomes relatively empty as the number of attributes increase above three even for very large databases. The density of records in hyperspace is thus a consideration in choosing a data mining technique.

Examples of Data Mining Applications

A few data mining applications are presented in this section. These applications are taken from a wide range of knowledge domains, including mortgage prepayment behavior, customer profiling, pilot bid behavior, and database analysis.

Goodarzi *et al.* describe a sample data mining application that predicts the mortgage prepayment behavior of homeowners (Goodarzi *et al.* n.d.). Mortgage prepayment typically reduces the earnings stream of an institution, in part by forcing the institution to re-invest the prepayment at a lower interest rate. Thus, the institution's return on investment suffers due to unexpected prepayments. Goodarzi *et al.*'s technique used data from a database supplied through an exclusive agreement with McDASH Analytics. Data mining software called MineSet™ was used in a collaboration with Risk Monitors, Inc. and Silicon Graphics, Inc. Attributes such as the present value ratio of the old loan and the new loan, treasury bond interest, and loan principle were modeled with a simple naïve-Bayes model. Goodarzi *et al.* concluded that the simple model resulted in efficient data cleaning and paved the way for more complicated models in the future.

Adrians and Zantinge (A&Z) describe three data mining applications based on projects at Syllogic: bank customer profiling, predicting bid behavior of pilots, and discovering foreign key relationships. The objective of bank customer profiling was to distinguish the characteristics of customers who buy many bank products from those who use only one or two bank products. The information obtained from data mining would later be used as a basis for marketing campaigns. The bank database was combined with a demographic database to provide attributes external to

the bank database. A neural network technique was used to obtain about 20 clusters of customers. Association rules and decision trees provided further analysis of these clusters. Useful patterns were claimed in terms of client psychology, bank policies, and marketing effectiveness.

A&Z described the use of data mining techniques to predict the bid behavior of pilots of KLM airlines. KLM needed a model to predict when pilots were likely to make a bid in the form of a transfer request to job openings. The objective was to use the knowledge to avoid either a surplus or shortage of pilots. A&Z found that operations research methods that were being used could not handle qualitative data effectively. A&Z instead used the pilots' historic career descriptions together with genetic algorithms. Data from pilot bids from 1986 to 1991 gave a model that was more than 80% successful. The success rate was later improved to over 90% after some fine-tuning. KLM claimed a pay-back time of less than one year for this data mining system.

The last example from A&Z concerns the reverse engineering of databases. Two uses of data mining are discussed. Data mining in a single table involves techniques such as cluster analysis and techniques that predict sub-sets of attributes from other attributes within the same table. On the other hand, discovery of the structure of the database as a whole involves techniques that span more than one table. The structure of the database includes things such as foreign key relationships and inclusion dependencies. Discovery of a database's structure may be necessary because the constraints are not given in the tables themselves but in the software programs that operate on them and the software may no longer be available. A&Z claim a polynomial time algorithm was developed to discover foreign key relationships instead of the brute force exponential method.

Data Mining Techniques

Data mining techniques include a wide range of choices from many disciplines. These choices include techniques such as support vector machines, correlation, linear regression, non-linear regression, genetic algorithms, neural networks, and decision trees. The choice of a data mining technique is contingent upon the nature of the problem to be solved and the size of the database. In the case of database size, some authors recommend that data mining techniques

should scale no higher than n(log n) where n is the number of records used as input for the algorithm (Adriaans and Zantinge 1996, 57). Optimistically, one could envision a combined procedure such as a scalable search algorithm followed by a more complex algorithm that operates efficiently upon a reduced size data set.

The techniques of simplex evolutionary operations (EVOP), genetic algorithms, Newton's method, support vector machines, association rule mining, and neural networks will be described below along with some example applications. Simplex EVOP is discussed as an optimization method that is similar to genetic algorithms but that uses deterministic search strategies. Newton's method is discussed as an optimization method that uses a special gradient descent search strategy. Finally, a summary of other commonly used data mining algorithms is given.

Evolutionary Operations (EVOP)

Engineers developed EVOP techniques in the middle of the twentieth century in order to more rapidly approach and attain optimum process conditions (Walters *et al.* 1991, 41). EVOP techniques have something in common with genetic algorithms used in data mining. In fact, Box used the analogy of lobster evolution to illustrate the EVOP method (Walters *et al.* 1991, 36). The use of simplex EVOP and data mining in pharmaceutical formulations has been recently reported (Levent 2001). It is hypothesized that the simplex EVOP described by Walters *et al.* has much in common with binary search algorithms that are very efficient in finding specified records in databases. The simplex EVOP algorithm is described in two parts: the basic simplex algorithm and the variable-size simplex algorithm.

The Basic Simplex Algorithm. The basic simplex algorithm of Walters *et al.* is a simple method to find a target value in multi-dimensional data spaces. The algorithm begins with a k dimensional vector space of factors. A k dimensional vector is formed from specific attribute values of the database such that $\mathbf{V}_j = [x_{j1}, x_{j2}, \ldots x_{jn}]$. $\mathbf{V}_j$ is the ith vector. The term x_{ij} is the database value for the j^{th} dimension of the j^{th} record $\mathbf{V}_j$. To begin, k+1 vectors are selected in the k dimensional space. Selection of the actual locations of the k+1 vectors is a matter to be

discussed later. For example, three vectors whose heads form the vertices of a triangle are typically used in a two dimensional space.

The database is used to compute a value of a fitness function, $F_j = F(\mathbf{V}_j)$, for each of the k+1 vectors. On the first iteration, the vectors are ranked according to the worst, **W**, the best, **B**, and the next to the worst values, **N** (Walters *et al.* 1991, 59). The centroid, **P**, of the hyperface formed by the exclusion of **W** is computed from the remaining k vectors. Next, the reflection vertex, **R**, is calculated as $\mathbf{R} = \mathbf{P} + (\mathbf{P} - \mathbf{W})$. A new simplex surface is created with k+1 vectors that now include the **R** vector and with the **W** vector omitted. On the next and subsequent iterations, the **N** vector becomes the **W** vector, even if **N** is not the worst case, and the process is repeated. The simplex EVOP crawls towards its objective in hyperspace. After it reaches its objective, the simplex EVOP may circle the objective. In order to determine the location of the objective, either other methods may be used or the variable size simplex algorithm discussed in the next section may be used.

The basic simplex EVOP may be illustrated with an example objective function F of two variables such that $F(x,y) = 10\exp(-(x-30)^2/30^2)\exp(-(y-45)^2/50^2)$. The function F has its maximum value at the point (x,y) = (30,45). Pick the points of the initial simplex to be (20,5), (15,5), and (10,5). The application of the algorithm described above is illustrated in Figure 1 where the xy plane is shown along with the points in the simplex.

The algorithm reaches the maximum value of F after seven generations as shown in Figure 2. The maximum of the function F shown above may be determined by inspection. However, if F had been a transcendental function or a numerical function then finding its maximum would become much more difficult. To find the maximum of F with calculus, one computes the partial derivatives of F with respect to x and y and then solves for the zeros of the derivative. If the function's derivative does not exist or is difficult to compute, then the simplex EVOP begins to look very attractive as a technique to find the maximum of a function.

The Variable Size Simplex Algorithm. The fixed step simplex was designed for large-scale production processes. In such processes, the product must be within specifications and the simplex size is necessarily small. For a database search, a simplex with a large step size may be used to quickly cover the hyperspace of the database. However, a uniform simplex step size may

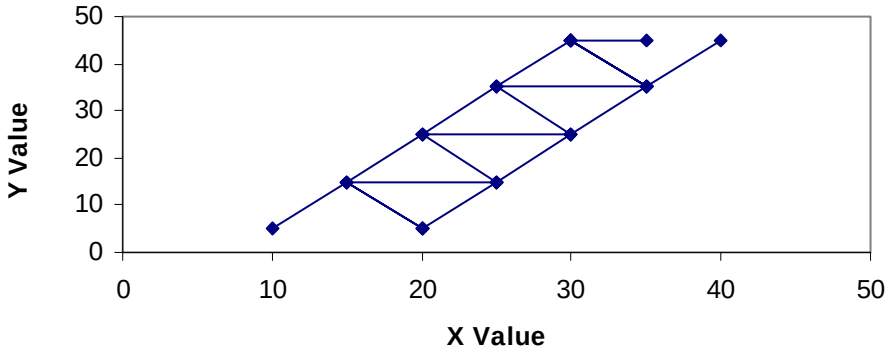

Figure 1. Two Dimensional Factor Space for a Fixed Length Simplex Optimization of Function Given in the Text.

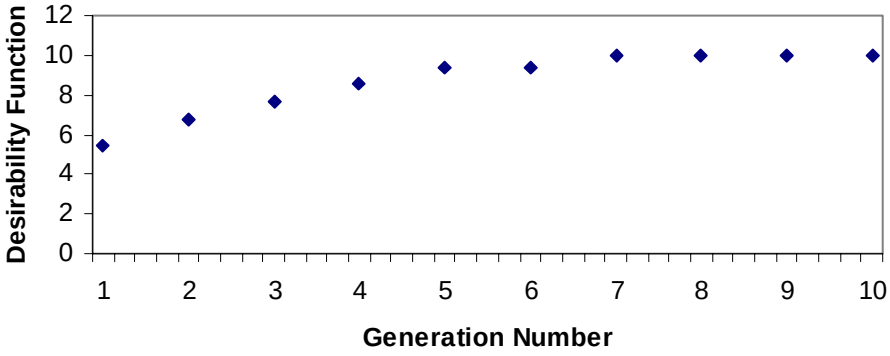

Figure 2. Function Value for Basic Simplex EVOP Example Given in the Text.

be too large or too small to be efficient over the course of an entire computation. Walters *et al.* credit Nelder and Mead with modifying the original simplex algorithm of Spendley, Hext, and Himsworth to allow the simplex to expand and contract as necessary to quickly find the objective with precision. (Press *et al.* 1988) explain this procedure as the amoeba.

The variable sized simplex either expands, contracts or stays the same size. If the simplex stays the same size, the fixed simplex algorithm described in the previous section is applied. If the simplex expands, it always expands on the side of the reflection plane opposite to **W**. The expansion vertex is represented as **E**. The Nelder and Mead algorithm defines $\mathbf{E} = \mathbf{P} + 2(\mathbf{P} - \mathbf{W})$. If contraction occurs, there are two possibilities. First, a contraction, $\mathbf{C_R} = \mathbf{P} + \frac{1}{2}(\mathbf{P} - \mathbf{W})$, can occur on the side of the reflection plane opposite to W. Second, a contraction, $\mathbf{C_W} = \mathbf{P} - \frac{1}{2}(\mathbf{P}-\mathbf{W})$, can occur on the same side as **W**. The rules for selecting which of the variable simplex options to use are given by Walters *et al.* (Walters *et al*. 1991, 77). These rules are converted to a decision in Figure 3. If the reflection, **R**, is better than **B**, then an expansion to **E** occurs unless **E** is not better than **B**. If **R** is worst than **N** and **W**, then contraction with $\mathbf{C_W}$ occurs. Otherwise, if **R** is better than or equal to **W**, then contraction to $\mathbf{C_R}$ occurs.

As mentioned previously, it is hypothesized that the simplex algorithm with the variable size modification is analogous to a binary search algorithm on an ordered list. The binary search algorithm is $\theta(\lg n)$ (see Baase and van Gelder for a definition of the θ and lg notation.) where n is the number of elements in the list. The variable size simplex expands or contracts by a factor of ½ if the appropriate selection rules are satisfied. This variable size strategy is analogous to the binary search algorithm. However, the analogy may break down if the initial simplex does not include the objective. This hypothesis will be examined in more detail later. Now, the process of selecting the initial simplex points is discussed.

Initial Simplex Points. The choice of the initial points for the simplex depends on the type of problem being solved. For a problem in which the function evaluations have real consequences, such as a manufacturing process, a fixed length simplex with small steps works best. For other problems such as process research or data mining, a variable length simplex algorithm that uses a large enough hypervolume to include the objective works best. Another consideration concerns the specific values of the simplex. Walters *et al.* state that the submatrix

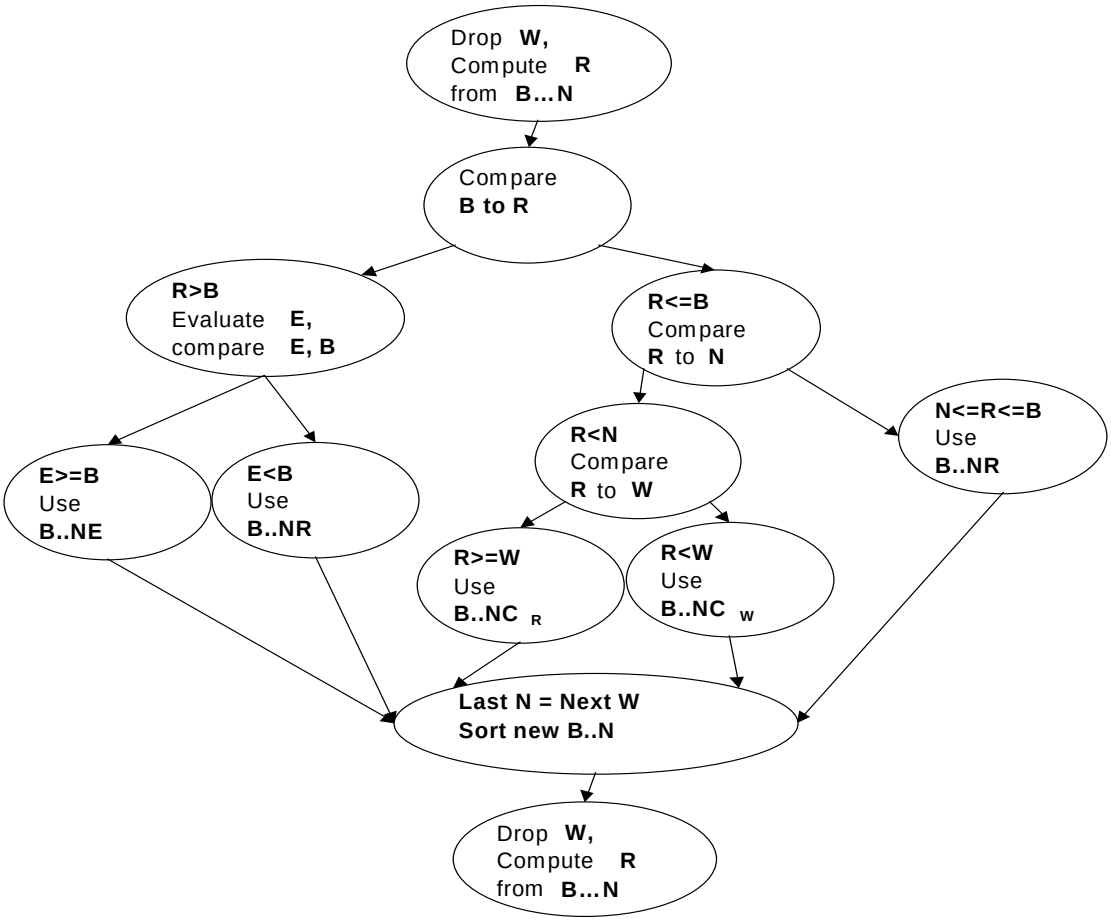

Figure 3. Decision Tree for a Variable-Size Simplex (Walters *et al*. 1991, p. 77).

formed by the simplex should have a non-zero determinant. The submatrix is interpreted here to be equivalent to the covariance matrix of the x values. The problems associated with a zero determinant for the simplex are probably as severe as for a singular data matrix in linear regression. Aside from the obvious error of using the same column twice, multi-collinearity of the factors could create a zero determinant. A singular value decomposition analysis (SVD), to be discussed later, to evaluate the starting point of the simplex algorithm is a diagnostic tool that may benefit the simplex algorithm. We will use the SVD algorithm with other optimization algorithms in later chapters.

Summary of Simplex EVOP Algorithm. The simplex EVOP algorithm follows a survival of the fittest strategy. Offspring are selected from the more successful vectors in the population. After a given number of iterations, or generations, an optimal solution should be obtained. This survival of the fittest strategy is analogous to that of the genetic algorithms. However, as shown in the next section, genetic algorithms use stochastic selection rules instead of the deterministic selection rules used by the simplex EVOP.

Genetic Algorithms

Genetic algorithms (GAs) provide a means to handle objective functions that are not well-behaved: *e.g.*, functions that are discontinuous or non-differentiable (Hodgson 2001, 413). GAs define a problem's solution space in terms of individuals or chromosomes. Each individual results in a value of a fitness function. For example, unsuccessful individuals result in a value of zero for the fitness function. Successful individuals result in a maximum value for the fitness function. Some individuals result in intermediate values of the fitness function. Individuals with higher fitness have a higher probability of being selected for mating and individuals having low fitness are killed off with a low probability of mating. The genetic processes of crossover and mutation are applied to the individuals in the mating population and a new generation is created.

In the simplex EVOP described above, the k dimensional vectors, $\mathbf{V}_j$, represent individuals with values of the fitness function given as F_j. However, the simplex EVOP uses deterministic rules to determine the mating population and its offspring. GAs use stochastic selection rules assigning a non-zero probability that any individual will be in the mating

population. This probability is a function of that individual's fitness. The stochastic selection process ensures that there is more coverage of the range of the variables then is the case for the simplex EVOP.

The mating population produces offspring using two basic types of operators: crossover and mutation. Crossover takes two individuals and produces two new individuals. Several possibilities exist for crossover methods. Several kinds of mutation operators can be used as well. A mutation changes an individual into a new individual by a random change to its chromosome.

GAs belong to the class of evolutionary computational methods. However, GAs have the following distinguishing characteristics (Mitchell 1996):

1. A population of chromosomes is randomly generated to cover the search space. For example, if the chromosomes are encoded as five bit strings, the chromosomes P1 = {00101} and P2 = {11000} would represent two members of the population.
2. A fitness function, F(x), is used to determine the desirability of each member of the population, x.
3. New offspring are produced from two parents by a process called crossover. For example, if P1 and P2 above crossover at position two, then the children would have C1 = {00000} and C2 = {11101} as their chromosomes.
4. Random mutation of new offspring occurs. For example, if C1 above mutates at position 5, then it would have C1' = {00001} as its chromosome.

To summarize, a GA is an evolutionary computational algorithm that uses random searches, a fitness function, crossover, and mutation to explore the search space of solutions to the problem. There are many variants to the algorithm. The following is a simple version of a genetic algorithm:

1. Random Generation. Generate a population of n randomly selected L-bit chromosomes.
2. Fitness. Calculate the fitness function, F(x), for each member of the population.
3. Selection. Two individuals are selected as parent pairs based on their fitness and a probability function, Ps = Ps(F(x)). And, the same individual may be selected for breeding more than once.

4. Crossover. With probability, Pc, a parent pair undergoes crossover at a randomly chosen point in the two chromosomes.
5. Mutation. With probability, Pm, a random bit of each offspring is flipped.
6. Continue steps 3, 4, and 5 until n new offspring are created.
7. Replace the old population with the offspring population.
8. Go back to step 2.

The preceding algorithm uses a binary bit string to represent a chromosome. It is also possible to use GAs with chromosomes made from multivariate data (Hodgson 2001) (Michalewicz 1999). Hodgson initiates the GA by selecting the initial population from a uniform distribution with each variable within prescribed bounds. The number of individuals in the population remains fixed after each generation. A specific set of crossover and mutation processes is applied to the selected mating population. The generation process continues until a termination criterion is reached. The termination criterion is usually a maximum number of generations. Finally, the results are evaluated for the optimal value(s) of the fitness function.

GAs are powerful in their ability to model both qualitative and quantitative search spaces. And, due to random mutations GAs typically do not lock in to a local maximum of the fitness function as could be the case for deterministic EVOP's. GA applications include the determination of optimum fitness and the determination of optimum composition for new product development. It is possible to use GAs in conjunction with neural network algorithms discussed below. Next, a technique that is able to find the optimum of a function but using the function's derivatives is discussed.

Newton's Method

The genetic algorithm technique is an optimization strategy that mimics biological evolution. Individuals are selected at random. If enough individuals are selected to cover the response surface, then these individuals will successfully breed and mutate to the location of the optimum. Newton's method is an optimization strategy that is a specialization of the class of optimization strategies called gradient descent methods. A gradient descent method chooses an improved value of a cost function c(x) by equation (1).

$$x_n = x_L - R(\partial c/\partial x)|_{xL} \quad (1)$$

In equation (1) c(x) is a cost function and R is the learning rate (Baldi 1998). In a genetic algorithm, the learning rate is controlled by a stochastic survival of the fittest strategy similar to random direction descent methods. In Newton's method, the learning rate is determined by the Hessian matrix of the cost function as will be discussed later. To learn the values of the parameters of a function, the cost function is not necessarily needed explicitly. For example to solve an equation of more than one variable, it will be shown that the recursion relation in Equation (2) is equivalent to the Newton technique.

$$\mathbf{x}_n = \mathbf{x}_L = (\mathbf{J'J})^{-1}\mathbf{J'}[\mathbf{F}(\mathbf{x}_L;\mathbf{a}) - \mathbf{F}_{obs}] \quad (2)$$

Where $\mathbf{x_L} \equiv$ the last vector of parameters, $\mathbf{J} \equiv$ the Jacobian matrix of derivatives, $F(\mathbf{x}_L;\mathbf{a}) \equiv$ the value of the function at $\mathbf{x}_L$, $\mathbf{a}_j$ is a vector of independent attributes from the database record j, and $\mathbf{F}_{obs}$ are the values of F from the database.

The Jacobian matrix is defined by equation (3).

$$\mathbf{J} \equiv \frac{\partial (F_1, F_2, ..., F_{NO})}{\partial (x_1, x_2, ..., x_n)} \quad (3)$$

Where **J** is an m by n matrix of partial derivatives of F and m is the number of records. NO is the number of functions in the system that depend on **x**.

Newton's Method is capable of the precision and accuracy desired for mining of technical data as illustrated in later chapters. Also, it will be shown that the standard errors of the individual attributes, assuming normally distributed residuals, are as given in equation (4).

$$S_{xi} = T_{(m-n)} S[(J^T J)^{-1}]_{ii}^{1/2}. \quad (4)$$

where S_{xi} is the standard error for the ith parameter, $T_{(m-n)}$ is the T value for (m-n) degrees of freedom, m is the number of records, n is the number of parameters, and S is the standard error of prediction.

Finally, it will be shown that the confidence interval for the predicted values after the Nth iteration are given by equation (5).

$$F_j = F_j\left(X_{N}\right) \pm S\sqrt{T^2 (J_j (J^T J)^{-1} J_j{}^T)} \qquad (5)$$

The statistics given in equations (4) and (5) are analogous to similar equations for a least squares linear regression (Deming and Morgan 1987). However, in the present case, F_j is a function of both the attribute values from the database and of the model parameters, e.g., $F_j = F(\mathbf{x};\mathbf{a}_j)$. Without the attribute values, there is rarely a way to solve equation (2) for more than one model parameter.

In cases where the function is not available to compute **J**, it is still possible to estimate **J** from the last two guesses. In fact, this technique of estimating **J** is similar to the secant method (Cont and de Boor 1980) or the truncated Newton's method (O'Leary 2000, 8). The secant method is computationally less expensive then Newton's method but its rate of convergence is smaller than the Newton's method. Convergence behavior of Newton's method is quadratic. This important feature of the method is discussed and illustrated in a later chapter. However, alternative strategies such as the simplex EVOP or genetic algorithm discussed above may be necessary if the computational overhead of Newton's method is too great. The software for such a combined simplex EVOP and a global Newton's method is given in (Press *et al.* 1988). However, Press *et al.* state that the corresponding higher dimensional local Newton's method represented by equation (2) is usually not solvable.

It is shown in a later chapter that correlation, regression, and Newton's method may be viewed as special cases of the gradient descent strategy. Thus, correlation and regression techniques, simplex and genetic algorithms, and the non-linear regression techniques could all conceivably be used together in an integrated approach to data mining of technical data.

Support Vector Machine

Techniques such as correlation and regression scale as N^3 due to the need for matrix multiplication. Some workers claim that the technique of support vector machines has better scalability. (Bennett and Campbell 2000). In one variant of the support vector method, attribute values for a database record map into a vector, **x**. A discrimination vector, **w**, is used to classify

the data. Classification occurs by the use of the function, F(**x**) = Sign(**w** dot **x** – b). The value of b determines the kind of discrimination to be achieved.

Association Rule Mining

Association rule mining is used quite frequently in business applications of data mining. Association rule mining techniques offer algorithms designed to discover correlation in qualitative data. For example, suppose a sales manager wants to learn about customer buying patterns. Association rule mining might find that 95% of all people who buy products x, y, and z also buy product w. Association rule algorithms attempt to discover such rules from binary attributes in a large database. In the example above, the association rule is denoted as an implication x, y, z => w. Consider a database that contains n binary attributes. There are n(n-1)/2 association rules of the type x=>y. In general, there are C(n, k) association rules that contain (k-1) attributes associated with a given attribute. Because C(n, k) sums to 2^n, then 2^n – (n+1) association rules are possible in a database of n attributes. The term (n+1) is subtracted to eliminate the null set and the set of singletons that correspond to the diagonal of the correlation matrix.

Define the implication X=>Y such that X and Y contain no common attributes. Then, the exponential number of possible association rules is reduced as follows (Berzal *et al.* 2001). Define the support, S, of the implication as the percentage of tuples in the database that contain X∪Y. Define the confidence, C, as the percentage of tuples in the database that contain X and Y (Berzal *et al*. 48). Next, define the minimum confidence, C_m, and minimum support, S_m. The association rule mining process then consists of two basic steps. Firstly, find all k combinations of attributes that have $C > C_m$ for k = 2 to n. Secondly, if X∪Y and X pass the first rule, then the rule X=>Y holds if $S(X \cup Y)/S(X) > C_m$. It can be shown that $S(X=>Y) \geq S_m$.

For example, consider a database with two binary attributes {A,B} that consists of the following tuples: {0,0}, {0,1}, {1,1}, and 7 {1,0}'s. Set $S_m = 50\%$. Then, there are only surviving tuples with A=1 (8 tuples) and B=0 (8 tuples). For k = 2, there is only one surviving tuple with A=1 and B = 0 (7 tuples). The confidence of the implication A=1 => B=0 is C = 7/8.

Berzal *et al.* introduce a new tree based association rule finding algorithm, TBAR. They also describe and compare existing algorithms such as the Apriori algorithm (Agrawal and Shim

1996) and the direct hashing and pruning algorithm, DHP (Park *et al*. 1995). Adriaans and Zantinge also reference the Apriori algorithm in the interface to their data mining software; however, they do not explicitly discuss their algorithm for association rule mining. TBAR and Apriori were implemented by Berzal *et al.* (2001, 56) using Java DataBase Connectivity (JDBC) and the Java standard Call-Level Interface (CLI) . Berzal *et al.* also cite other implementation alternatives (Sarawagi and Agrawal1998).

Neural Networks

Neural network algorithms (NNs) come from the field of artificial intelligence. NNs relate to the technique that uses a systolic array of function boxes (Kaskali and Margaritas 1996). Smith and Gupta (2000, 1024) state that neural networks have become the foundation of most commercial data mining products. NNs are similar to linear regression techniques in that a prediction model is produced. However, Smith and Gupta state that NNs are more powerful in their ability to model non-linear behavior and to require no assumptions about the underlying data. Many kinds of NNs exist including multilayer feedforward neural networks (MFNN), Hopfield neural networks, self-organizing neural networks, adaptive resonance networks, radical basis networks, modular networks, neocognitrons, and brain-state-in-a-box (Smith and Gupta 2000). A description of the MFNN technique follows.

An MFNN consists of an N dimensional input vector **x** consisting of (N-1) variables from the database and with the remaining variable equal to –1. The neural network processes the input vector and produces an NO dimensional output vector **y**. The neural network contains two or more layers of neurons. Layers between the input values and the last layer are called hidden layers. The neurons process the output from the previous layer of neurons and sends its output to the input of the next layer of neurons.

Let η be the number of layers of neurons. And, let ν_k be the number of neurons for layer k. Assume that NO = ν_η. The input signals are propagated through the network according to the following rules. For the first layer, define a matrix $\boldsymbol{\omega}$ such that the following equation (6) holds.

$$\omega_{jk} = \sum_{i=1}^{N} W_{kji} x_i$$

where $k = 1$ and,

ω_{jk} is the output of node j of layer k,

W_{kji} is the weighting for input i of node j of layer k. (6)

The input for layers $k \geq 2$ is given by equation (7).

$$Y_{jk} = f(\omega_{jk-1}) = \frac{1}{1 + \exp(-\lambda \omega_{jk-1})}$$

where $k \geq 2$ and,

Y_{jk} is the j^{th} input to level k,

$Y_{\nu_k, k} \equiv -1$,

λ is a gradient parameter. (7)

The output for layers $k \geq 2$ is given by equation (8).

$$\omega_{jk} = \sum_{i=1}^{\nu_k + 1} W_{kji} Y_{ik}$$

where $k \geq 2$ and,

ν_k is the number of neurons for level k. (8)

The output of the algorithm is equal to the output of level η. An example of a neural network with two inputs, two outputs, and a hidden layer of three nodes is given in Figure 4.

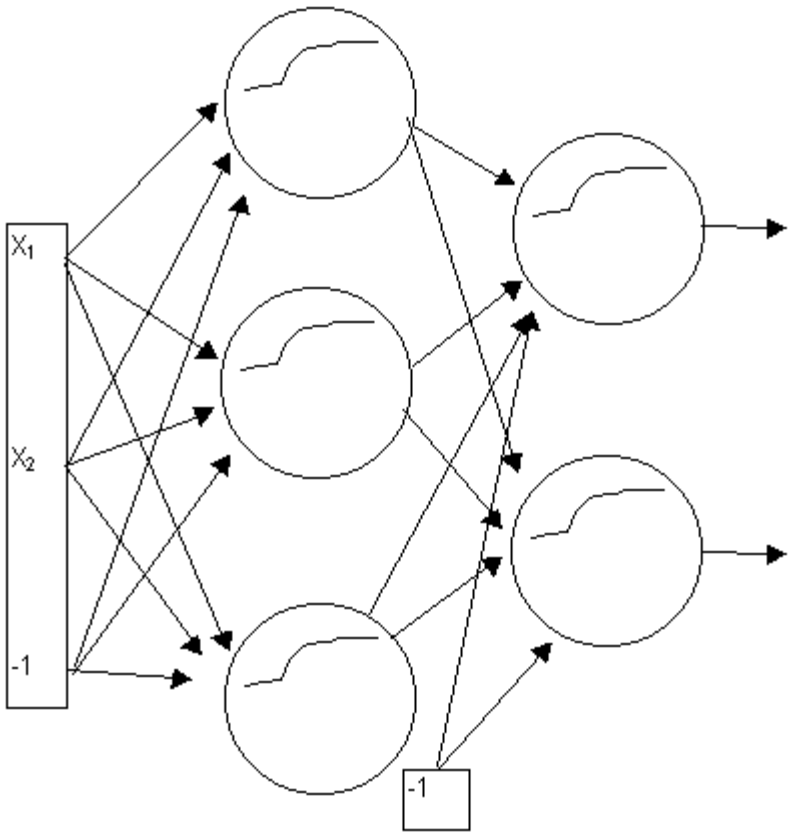

Figure 4. Neural Network Diagram with Two Inputs, Two Outputs, and One Hidden Layer with Three Neurons.

The neural network is diagrammed in terms of the equations (6), (7), and (8) in Figures 5 and 6. Evidently, the number of independent weights, W_{jki}, is $T_W = N\nu_1 + (\nu_1+1)\nu_2 + (\nu_2+1)\nu_3 + \ldots + (\nu_{\eta-1}+1)\nu_\eta$. In the case of Figure 4, the value of $T_W = 3*3 + 4*2 = 15$. That is, a total of fifteen constants are needed to train the neural network of Figure 4.

In their neural network example for data mining, A&Z use nine binary inputs: age<30, 30<age<50, age>50, house, car, area 1, area 2, area 3, area 4. The neural network has five binary outputs: car magazine, house magazine, sports magazine, music magazine, and comic magazine. Finally, the neural network has one hidden layer with four neurons. The A&Z example then has a value of $T_W = 10*4 + 5*5 = 65$. To determine 65 regression coefficients requires a careful sampling design of hundreds of values to establish statistical significance of the model and its predictions. A non-linear model is even more difficult. Methods of assessing statistical significance of neural network models could conceivably be done with equations (3) to (5) for technical data. For qualitative data, such as marketing data, GAs are probably more effective as a means to train NNs.

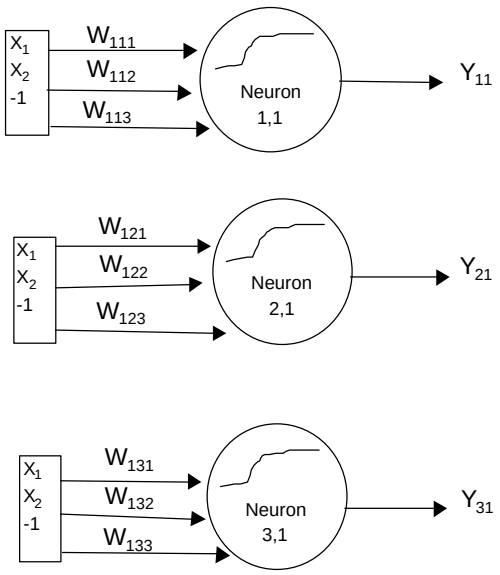

Figure 5. Illustration of Mathematical Notation from the Text for Layer One of Neural Network in Figure 4.

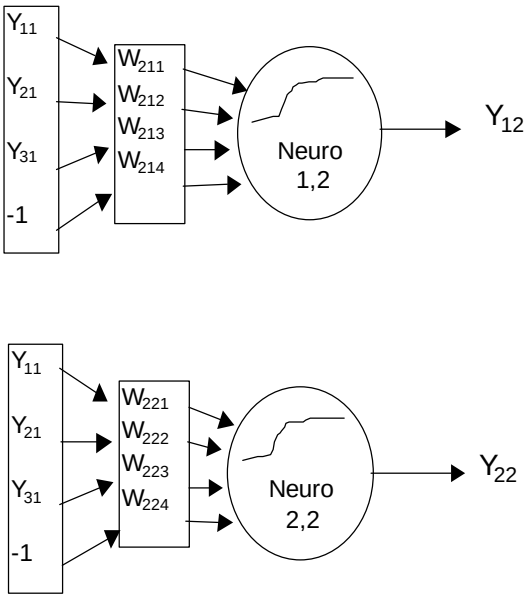

Figure 6. Illustration of Mathematical Notation for Layer Two of the Neural Network in Figure 4.

Other Techniques Used in Data Mining

Knowledge discovery in databases (KDD) is a process for defining a problem, extracting non-trivial knowledge about the problem from a database, interpreting the results, and managing the found knowledge. Data mining is the part of KDD that involves techniques from many disciplines to facilitate the KDD process of knowledge discovery. The techniques of correlation and regression, simplex EVOP, genetic algorithms, Newton-Raphson method, support vector machines, association rule mining, and neural networks are considered to be within the context of data mining. Time and space do not permit a complete coverage or even a satisfactory description of a sufficiently useful list of techniques. Actually, one might better define data mining as the art of selecting a manageably few and appropriate techniques from operations research, machine learning, artificial intelligence, computer science, econometrics, etc., that will solve the KDD problem at hand. Some other major data mining techniques are given below.

Some techniques are not strictly data mining techniques because they do not produce non-trivial knowledge. However, these techniques are useful in KDD and data mining nevertheless. Query tools belong to this category, especially the queries for counts, averages, and standard deviations. A&Z refer to the use of query tools to produce naïve predictions that are later used to test the minimum validity of a non-trivial data mining technique. Statistical techniques such as analysis of variance, cross-correlation studies, and time series analysis represent other standard statistical techniques that are useful to KDD. Also, visualization tools such as bar charts, scatter diagrams, and cluster analysis are useful in KDD. Online analytical processing (OLAP) tools store data from the database in special format that allow the user to answer specific multi-dimensional questions. OLAP tools are not considered data mining tools since data mining tools are expected to work directly on data stored in a database (Adrians and Zantinge 1996, 56). Reportedly, the k-nearest neighbor method seeks to determine the behavior of an individual by examining the behavior of its neighbors. The k-nearest neighbor method has poor scalability. It is debatable whether the k-nearest neighbor method is a learning method or a search method.

Decision trees are a scalable data mining algorithm that predict the most probable behavior of an individual based on a classification and decomposition of its attributes. Decision trees have an advantage over neural networks in that the decision tree is easily used and

interpreted by humans. In contrast, a neural network may give excellent predictions but it is difficult to understand why and how the prediction works. Such an understanding is usually a requirement for technical data mining. Reportedly, decision trees do not perform well on multivariate data spaces, however.

Latent Semantic Analysis is a machine learning technique that could potentially be used in data mining (Gotoh and Renals 1997). Documents are read and a frequency matrix of words and their occurrence frequency in sentences and phrases is constructed. Singular Value Decomposition (SVD) is then used to find all the correlations between the various words and sentences. It is then possible to answer questions in the language of the documents. We will cover SVD in later chapters.

Datasets for Experimentation

Any research in data mining requires large databases to test methods and hypotheses. Some databases available publicly are given below.

Berzal *et al.* cite several datasets for experimentation in data mining:

- UCI Machine Learning Database Repository (UCIR) at http://www.ics.uci.edu/~mlearn/MLRRepository.html
    1. Golf: contains weather conditions.
    2. Soybean database: prepared for soybean disease diagnosis.
    3. Mushroom database: collection of edible, poisonous, unknown, and not recommended mushrooms.
    4. ADULT: replaces CENSUS database in UCIR.
- Census database: data extracted from the census bureau at http://www.census.gov/ftp/pub/DES/www/welcome.html and http://www.bls.census.gov/cps/cpsmain.htm.

Many other types of databases may be found at these websites.

Data Mining Literature Survey Summary

This concludes a literature survey of data mining. Data mining is the use of learning algorithms to extract non-trivial knowledge from data. And, data mining is used in conjunction with the process of knowledge discovery in databases. The literature search resulted in a wide

range of data mining algorithms: genetic algorithms, neural networks, decision trees, and association networks – to name just a few. Other candidate techniques for data mining of technical data were proposed: simplex evolutionary operations, Newton-Raphson iteration, and the secant method. The simplex EVOP methods, genetic algorithm techniques, Newton's method techniques, and neural network techniques are all variations on the gradient descent optimization strategy with different values for the learning rate. An integrated approach using these techniques in data mining would utilize the strengths of each optimization strategy according to the specific problem.

CHAPTER 3

NEWTON'S METHOD

The derivation of non-trivial knowledge from a data base according to strategies such as the gradient descent method discussed above generally leads to the solution of an equation of the form $f(x) = 0$. Many interesting problems in mathematics, engineering, science, and economics also amount to solving an equation of the form $f(x) = 0$ for x. This chapter discusses Newton's method for solving $f(x) = 0$. As will be shown, Newton's method is one of the most important numerical techniques for solving $f(x)=0$ due to its quadratic rate of convergence.

A truncated Taylor series method can be used to derive Newton's method: $f(\xi)=0 \wedge f(\xi) = f(x) + f'(x)(\xi-x) +O(\xi-x)^2 \Rightarrow \xi \approx x - f(x)/f'(x)$. So, for any guess for x that fits the criteria discussed below an improved guess may be obtained. Furthermore, matrix methods can easily extend Newton's method to functions of more than one variable. Finally, it is straightforward to then use Newton's method to solve multivariate unconstrained optimization problems that result in a system of n non-linear equations in n unknowns. These computations require matrix methods that are discussed in a later chapter.

Newton's method is often referred to as the Newton-Raphson method. Dunham (1994) credits Newton as having discovered the method in the 1660s. Later modifications were made by Joseph Raphson in 1690 and by Thomas Simpson in 1740. The exact sequence is difficult to determine since Newton did not initially publish his putative discovery of calculus – Newton referred to this as the theory of fluxions – until Leibniz claimed invention of calculus and introduced it to the world in 1684. Ypma (1995) gives an excellent development of the method in historical terms that is summarized below. In fact, previous iteration equations similar to Newton's method existed prior to 1660 with the derivative replaced by finite difference approximations, e.g., the secant method discussed below. It is probable that the classical Greek and Babylonian mathematicians used such iterative techniques (Gleick 1987) and also (Ypma

1995, 534). In the 12th century, Sharaf al-Din al-Tusi used a technique that is algebraically equivalent to Newton's method. And, Al-Kashi used this method to solve $x^p - N = 0$ to find roots of N in the 15th century (Ypma 1995, 539). The French algebraist, Francois Viete, published *De numerosa potestatum*, in 1600, a work that concerned numerical solution of nonlinear equations. Raphson expressed Newton's method in terms of calculus – actually using fluxions instead of Leibniz's notation. And, Simpson finally extended the method to multivariate unconstrained optimization problems.

Our objective is to learn more about the convergence behavior of Newton's method for the purpose of applying it to techniques requiring machine learning such as data mining. Along the way, we include other non-linear equation solving algorithms for comparison. A rigorous theory from the literature is summarized and presented for functions whose range and domain are real valued numbers. However, for multivariate functions and for functions in the complex plane, the situation becomes more complicated. We will analyze Newton's method on the real line, the complex plane and for multivariate real values. Finally, we will end with some examples using a few simple functions that illustrate either convergence or divergence of Newton's method and with suggestions for future research.

Non-Linear Equation Iteration Algorithms

Algorithms for Newton's method, the secant method, *regula falsi*, and the bisection method are presented in Conte and de Boor (1980), Ypma (1995), and numerous other references. A short dicussion of these techniques based on the references above but using a slightly different notation and derivation are given below. Consider a function, f(x), that is to be solved for ξ such that $f(\xi) = 0$. If we expand f(x) in a Taylor series about x_0, we obtain equation (9):

$$f(x) = f(x_0) + f'(x_0)(x - x_0) \quad \text{(Taylor Series Expansion)} \tag{9}$$

If $x \approx \xi$ such that $f(x) \approx 0$, then equation (9) may be solved for x as shown in equation (10):

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{10}$$

Equation (10) is usually written in its iteration form as in equation (11):

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \text{ (Newton's Method)} \tag{11}$$

Equation (11) is the equation for the Newton-Raphson method for solving the equation $f(x) = 0$ for $x = \xi$. If we approximate the derivative in equation (11) by $\Delta y/\Delta x$, $y = f(x)$, we obtain one form of the secant method as given in equation (12):

$$x_{n+1} = x_n - \frac{f(x_n)}{\Delta y_n / \Delta x_n} \quad \text{(Secant Method Form I)} \tag{12}$$

In equation (12), $\Delta y_n/\Delta x_n \equiv [f(x_n) - f(x_{n-1})]/(x_n - x_{n-1})$. Equation (12) is easily rearranged to equation (13):

$$x_{n+1} = \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})} \quad \text{(Secant Method Form II)} \tag{13}$$

In equation (13), if $f(x_n)$ and $f(x_{n-1})$ have opposite signs such that $f(x_n)f(x_{n-1}) < 0$, then equation (13) amounts to a weighted average of the last two iterations. In case $f(x_n)f(x_{n-1}) < 0$ – and $f'(x)$ is of constant sign on the interval, *etc.*- the weighted average is used in the *regula falsi* algorithm as indicated in equation (14):

(Regula falsi or false - position algorithm)

Given : $f(x_n)f(x_{n-1}) < 0$

$$\text{Let } w = \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})} \tag{14}$$

$$f(x_{n-1})f(w) \le 0 \Rightarrow (x_n \leftarrow x_{n-1}) \wedge (x_{n+1} \leftarrow w)$$

$$\text{else } \Rightarrow (x_{n+1} \leftarrow x_n) \wedge (x_n \leftarrow w)$$

In case the denominator for either Newton's method, the secant method or *regula falsi* are zero, the algorithm halts. In case the algorithm halts, a simple arithmetic average instead of the weighted average shown for the *regula falsi* method may be used. We then obtain the bisection method in equation (15):

(Bisection Method)

Given : $f(x_n)f(x_{n-1}) < 0$

$$\text{Let } m = \frac{x_{n-1} + x_n}{2} \tag{15}$$

$$f(x_{n-1})f(m) \le 0 \Rightarrow (x_n \leftarrow x_{n-1}) \wedge (x_{n+1} \leftarrow m)$$

$$\text{else } \Rightarrow (x_{n+1} \leftarrow x_n) \wedge (x_n \leftarrow m)$$

Newton's Method in Higher Dimensions

Newton's method applies equally well for x in $\mathbf{R}$, x in $\mathbf{R}^n$, and for x in $\mathbf{C}$. For x in $\mathbf{C}$, equation (11) becomes two independent equations for the real and imaginary parts of x. However, for x in $\mathbf{R}^n$, the situation is more complicated since we now have one equation with n unknowns. For unconstrained optimization problems, the gradient of the function to be maximized or minimized will result in a system of n equations in n unknowns that may be solved by Newton's method. To determine which extremum is determined, one uses the Hessian matrix, ($\nabla^2 F$).

As will be shown by example below, the problem of multivariate Newton's method is easily solved if there is a parameter, ϕ, that may be varied independently of x. In physical science problems, such parameters arise quite naturally as concentration, pressure, temperature, *etc*. In the case there is such a parameter, equation (5) becomes the system of equations as shown in equation (16):

$$\begin{bmatrix} f(\mathbf{x};\phi_1) \\ f(\mathbf{x};\phi_2) \\ \dots \\ f(\mathbf{x};\phi_n) \end{bmatrix} = \begin{bmatrix} f(\mathbf{x}_0;\phi_1) \\ f(\mathbf{x}_0;\phi_2) \\ \dots \\ f(\mathbf{x}_0;\phi_n) \end{bmatrix} + \begin{bmatrix} f_1(\mathbf{x}_0;\phi_1) & f_2(\mathbf{x}_0;\phi_1)\dots \\ f_1(\mathbf{x}_0;\phi_2) & f_2(\mathbf{x}_0;\phi_2)\dots \\ \dots & \\ f_1(\mathbf{x}_0;\phi_n) & f_2(\mathbf{x}_0;\phi_n)\dots \end{bmatrix} [\mathbf{x} - \mathbf{x}_0] \tag{16}$$

In equation (16), **x** is an n dimensional vector and f has a derivative for each dimension of **x**, *e.g*., $f_1(\mathbf{x})$, $f_2(\mathbf{x})$, *etc*. Now let us rewrite equation (16) in order to define the function vector, **F**, and the Jacobian matrix, **J**, in equation (17):

$$\mathbf{F}(\mathbf{x}) = \mathbf{F}(\mathbf{x}_0) + \mathbf{J}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) \quad \text{(Multivariate Taylor's Series)} \tag{17}$$

We now set $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ and in a manner similar to before we solve for the iteration function. By applying Kramer's rule for systems of n equations in n unknowns, we obtain Newton's method for multivariate functions in equation (18):

$$\mathbf{x}_{n+1} = \mathbf{x}_n - [\mathbf{J}(\mathbf{x}_n)]^{-1}\mathbf{F}(\mathbf{x}_n) \quad \text{(Multivariate Newton's Method)} \tag{18}$$

Convergence Behavior of the Newton-Raphson Algorithm

In this section, we discuss the convergence behavior of Newton's method. The development is based on that of Conte and de Boor (1980). However, whereas they use a more rigorous approach - mean value theorems for derivatives, Rolle's theorem, *etc*. – our development is based on truncation of the appropriate Taylor series expansion.

Fixed Point Iteration

Define the iteration equation in Equations (10) and (11) as g(x) as is customary. Then, we have the following relationships for Newton's method in terms of the fixed point iteration function, g(x), in equation (19):

$$g(x) = x - \frac{f(x)}{f'(x)}$$
$$x_{n+1} = g(x_n) \tag{19}$$

If ξ is a solution of f(x)=0, then the equation (20) holds:

$$\xi = \lim_{n\to\infty}(x_{n+1}) = \lim_{n\to\infty}(g(x_n)) = g\left(\lim_{n\to\infty} x_n\right) = g(\xi) \tag{20}$$

The value ξ is then called a fixed point of g(x) – ξ is also called an attractor. For example, if g(x) = x/2 + 1/x, then $\sqrt{2}/2 + 1/\sqrt{2} = \sqrt{2}/2 + \sqrt{2}/2 = \sqrt{2}$ and $\xi = \sqrt{2}$ is a fixed point for this iteration function. Furthermore, suppose g(x) is differentiable and there exists a K such that $0 < K < 1$ and $g'(x) < K$ on some interval. Define the iteration error as $e_n = \xi - x_n$. Using the fixed point iteration function, we can write $e_n = g(\xi) - g(x_{n-1})$. But, the Taylor series expansion of $g(\xi)$ about x_{n-1} is as follows:

$g(\xi) \approx g(x_{n-1}) + g'(x_{n-1})(\xi - x_{n-1}) \Rightarrow g(\xi) - g(x_{n-1}) = g'(x_{n-1})e_{n-1}$. We then have the following recursion relation for the error in the fixed point iteration method in equation (21):

$$e_n \approx g'(x_{n-1})e_{n-1} \le Ke_{n-1} \tag{21}$$

If we apply the above equation for e_n recursively, we obtain equation (22):

$$|e_n| \le K|e_{n-1}| \le K^2|e_{n-2}| \le K^3|e_{n-3}| \ldots \le K^n|e_0| \tag{22}$$

Since $K^n \to 0$ as $n \to \infty$, it follows that $e_n \to 0$ and that $x_n \to \xi$.

Quadratic Convergence of Newton's Method

Now, let us consider the Taylor series expansion of $g(x_n)$ about ξ in equation (23):

$$g(x_n) \approx g(\xi) + g'(\xi)(x_n - \xi) + \frac{1}{2} g''(\xi)(x_n - \xi)^2 \tag{23}$$

It turns out that for the Newton iteration method, $g'(\xi) = 0$. A proof is given in equation (24):

$$\begin{aligned} g(x) &= x - \frac{f(x)}{f'(x)} \\ g'(x) &= 1 - \left[\frac{f'(x)}{f'(x)} - \frac{f(x) f''(x)}{[f'(x)]^2} \right] = 1 - \left[1 - \frac{f(x) f''(x)}{[f'(x)]^2} \right] \\ g'(x) &= \frac{f(x) f''(x)}{[f'(x)]^2} \Rightarrow g'(\xi) = 0 \text{ since } f(\xi) = 0 \end{aligned} \tag{24}$$

Similarly, it can be shown that $g''(\xi) = f''(\xi)/[f'(\xi)]^5$.

Thus, if we solve this expansion for $g(\xi) - g(x_n) = \xi - x_{n+1} = e_{n+1}$, we obtain equation (25):

$$|e_{n+1}| \approx \left| \frac{1}{2} g''(\xi) \right| e_n^2 \tag{25}$$

For example, if $f(x) = x^2 - 2$ and $g(x) = x/2 + 1/x$, then $g'(x) = ½ - 1/x^2$. And, we have $g'(\sqrt{2}) = ½ - ½ = 0$. Also, $f'(x) = 2x$ and $f''(x) = 2$. Thus $g''(\xi) = 2/2^{15/2} = 2^{-13/2} = 0.0110\ldots$. This development and example illustrate what is meant when it is stated that Newton's method converges quadratically. On the other hand, it can be shown that the secant method converges at the 1.618…th power instead of the 2nd power. (See Conte and de Boor for a proof).

Number of Iterations Required

Consider the bisection method from equation (15). Every iteration reduces the interval for the solution by a factor of ½. If the initial interval is $M = |x_1 - x_2|$, then the interval after n iterations is given as: $|x_n - x_{n-1}| = M/2^n$. Or, the number of iterations required for the bisection method is $n = \lg(M/\varepsilon)$ where ε is the desired error level. Every iteration improves the answer by one bit which amounts to a linear rate of convergence for the bisection method. The bisection method error recursion relation is given as $\varepsilon_n = \varepsilon_{n-1}/2$. As shown above, the error recursion relation for Newton's method is given as $e_n = g''(\xi)(e_{n-1})^2/2$.

We have shown that $e_{n+1} = G(e_n)^2$ where $G = g''(\xi)/2$. If we apply this equation recursively, we obtain the following equation (26):

$$e_{n+1} = \left(Ge_0\right)^{2^n} \Big/ G \tag{26}$$

If we solve this equation for n, we obtain equation (27)

$$n = \lg\left[\left(\frac{\ln e_{n+1} + \ln G}{\ln e_0 + \ln G}\right)\right] \tag{27}$$

In contrast, the value of n for the bisection method given above is $n = \lg(e_0/e_{n+1})$. Thus, Newton's method provides an exponential speed up compared to the bisection method.

Conditions for Convergence of Newton's Method

The conditions for convergence of Newton's method are given below (Conte and de Boor 1987):

1. f(x) is twice differentiable on [a,b]
2. $f(a)f(b) < 0$
3. $f'(x) \neq 0$ on [a,b]
4. $f''(x) \geq 0$ or $f''(x) \leq 0$ on [a,b]

5. $|f(a)|/|f'(a)| < |b-a|$ and $|f(b)|/|f'(b)| < |b-a|$

Condition 1 is basically a requirement that f(x) and f'(x) be continuous. And, if f''(x) does not exist, then the quadratic convergence formula given above is not valid. Condition 2 indicates that there is at least one zero of f(x) on [a,b]. Condition 3 states that there are no maxima or minima on [a,b] – and hence not more than one zero - and also ensures that the iteration formula is valid. Condition 4 states that there are no critical points on [a,b]. Condition 5 ensures that all iterations are bounded by [a,b].

Example Iteration Functions

The convergence behavior of Newton's method is illustrated below using the Mandlebrot Set; a modified Mandelbrot, set f(x) = exp(-1/x); the square root of two, $f(x) = x^2-2$; Euler's equation, f(x) = exp(x) + 1; the fourth roots of 1, $x^4-1 = 0$ (Gleick 1987); the Verhulst population growth equation (Addison 1997); and the Krieger-Dougherty equation (Goodwin and Hughes 2000).

Mandlebrot Set

The Mandelbrot set is the set of all complex numbers, c, for which the iteration z^2+c is Bounded – equation (28).

$$z_{n+1} = z_n^2 + c \quad \text{(Mandlebrot Set)} \tag{28}$$

This iteration function is well-known to result in fractal surfaces in the complex plane.

Modified Mandelbrot Set

Consider the function in equation (29):

$$\begin{aligned} f(z) &= z_\infty \exp\left(-1/z\right) \\ f'(z) &= -\frac{z_\infty}{z^2}\exp\left(-1/z\right) \\ g(z) &= z - \frac{z_\infty \exp\left(-1/z\right)}{-\frac{z_\infty}{z^2}\exp\left(-1/z\right)} = z^2 + z \end{aligned} \tag{29}$$

The resulting iteration function is a variant of the Mandlebrot set iteration function. This function is discontinuous at the origin and violates one requirement for convergence of Newton's method (see above).

The iteration function, however, is simple and is very similar to that of the Mandlebrot set – equation (30).

$$z_{n+1} = z_n^2 + z_n \quad \text{(Modified Mandlebrot Set)} \tag{30}$$

Square Root of 2

The function $f(z) = z^2 - 2$ has the derivative $f'(z) = 2z$. The Newton's method iteration equation is given in equation (31).

$$z_{n+1} = z_n - \frac{z_n^2 - 2}{2z_n} = \frac{z_n}{2} + \frac{1}{z_n} \quad \text{(Square Root of 2)} \tag{31}$$

Euler's Formula: $\exp(\pi i) + 1 = 0$

The equation $f(z) = \exp(z)+1$ has the series of roots, $\pm(2n-1)\pi I$, $i = \sqrt{-1}$. The derivative is $f'(z) = \exp(z)$. The Newton's method iteration equation is given in equation (32).

$$z_{n+1} = z_n - \frac{e^{z_n} + 1}{e^{z_n}} = z_n - 1 - e^{-z_n} \quad \text{(Roots are odd multiples of } \pi \text{ i.)} \tag{32}$$

Fourth Roots of One

The equation $f(z) = z^4 - 1$ has the roots $\pm 1, \pm i$. The derivative is $f'(z) = 4z^3$ and the iteration function is given in equation (33):

$$z_{n+1} = \frac{3z_n}{4} + \frac{1}{4z_n^3} \quad \text{(Iteration function for fourth roots of one)} \tag{33}$$

As is the case for the square root of two function, this function has a zero derivative at the origin. Furthermore, it has a zero second derivative at the origin. Although the conditions for

convergence stated above are violated, convergence is obtained in interesting ways as described below.

Verhulst Population Growth Model

The Verhulst population growth model predicts a population's growth behavior.

$$z_{n+1} = z_n + a(1 - z_n)z_n \quad (\text{VerhulstPopulationGrowthModel}) \tag{34}$$

This iteration function in equation (34) is similar to the modified Mandlebrot function in that it is a polynomial of degree two in z.

Krieger-Dougherty Equation

The Krieger-Dougherty equation (35) predicts the viscosity of a colloidal dispersion

$$\eta_r = \left(1 - \phi/\phi_m\right)^{-[\eta]\phi_m} \quad (\text{Krieger Dougherty Equation})$$
$$\eta_r = \text{dispersion relative viscosity}$$
$$\phi = \text{volume fraction of particles} \tag{35}$$
$$\phi_m = \text{maximum packing fraction of particles}$$
$$[\eta] = \text{intrinsic viscosity of particles}$$

(Goodwin and Huges 2000).

The Jacobian of the Krieger-Dougherty Equation is given below in equation (36).

$$\mathbf{J}^T = \begin{bmatrix} \partial\eta_r/\partial[\eta] \\ \partial\eta_r/\partial\phi_m \end{bmatrix} = \begin{bmatrix} -\phi_m \ln\left(1 - \phi/\phi_m\right)\eta_r \\ -[\eta]\eta_r\left[\ln\left(1 - \phi/\phi_m\right) + \dfrac{\phi/\phi_m}{1 - \phi/\phi_m}\right] \end{bmatrix} \tag{36}$$

Convergence Behavior for x in R: Quadratic Convergence

If in fact the Greeks used iteration equations similar to Newton's method, they would have used fractions since the decimal system was not yet invented. Consider the iteration

function for $f(x) = x^2 - 2 = 0$: $g(x) = x/2 + 1/x$. If we start with an initial value of 2, we obtain the sequence: 2, 3/2, 17/12, 577/408, 665857/470832. In modern day decimal notation, this sequence is 2, 1.5, 1.416…, 1.41421…, 1.414213562375 which is to be compared to $\sqrt{2}$ = 1.414213562373… . Thus, after only three iterations, the ancient Greeks should have calculated the square root of 2 accurate to the equivalent of 12 significant digits. Figure 7 shows the iteration sequence for this example for the starting value of 0.5. In Figure 7, the iteration sequence is 0.5, 2.25, 1.5694…, 1.421890363815, 1.414234285940, 1.414213562525, 1.414213562373. Figure 7 is the local Newton's Method analog to optimization algorithms such as illustrated in Figure 1.

The convergence behavior for Newton's method is compared to that of the bisection and secant methods in Figure 8. The relative error reported in Figure 8 is the absolute relative error of the method to calculate the square root of 2. The order of the rates of convergence for the three methods, i.e., Newton > Secant >> Bisection, is consistent with the theoretical predictions given in the previous sections.

Newton's Method in the Complex Plane

A simple program to generate a pixel map was written to test convergence of iterated function sequences in the complex plane based on usual methods for showing fractal images (Gleick 1987). If $|z_n| > M$ in R where $z_n = g(z_{n-1})$, then a color, C_n (n = 1 to 15), was assigned to the pixel corresponding to z_0. If M was exceeded after one iteration, then the pixel color was set to C_1. For two iterations the color was C_2 and so on. In other words, if the iterated function sequence does not diverge after 15 iterations, the corresponding starting point on the complex plane is colored black. (An additional feature was added that colors the black region as shades of red according to the sign of the imaginary part of ξ.) The "Display Solutions" window shows both the real and imaginary parts of the last 15 solutions from random points in the view window.

Convergence intervals were determined by varying the value of M prior to the start of the iteration loop. For example, for $f(x) = x^2-2$, a value of M = 1 resulted in no black regions. Regions of black appear, however, if we set M = 1.5 which usually indicates convergence has

occurred in those regions to a square root of two. Now, the convergence behavior of the various functions given above will be discussed.

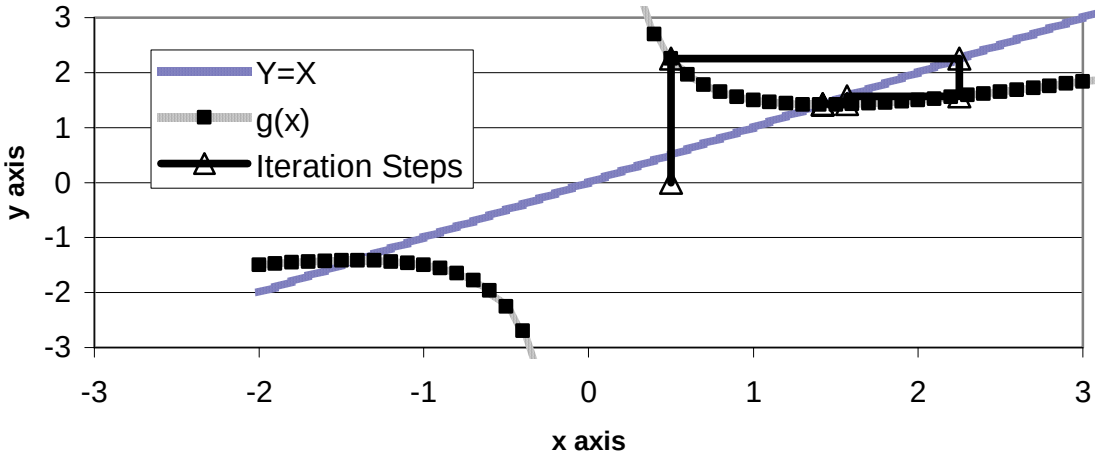

Figure 7. Graph of Newton's Method for $f(x) = x^2 - 2$ and $g(x) = x/2 + 1/x$ starting at 0.5. Dashed line: Y = X. Square: g(x). Triangle: Iteration Steps

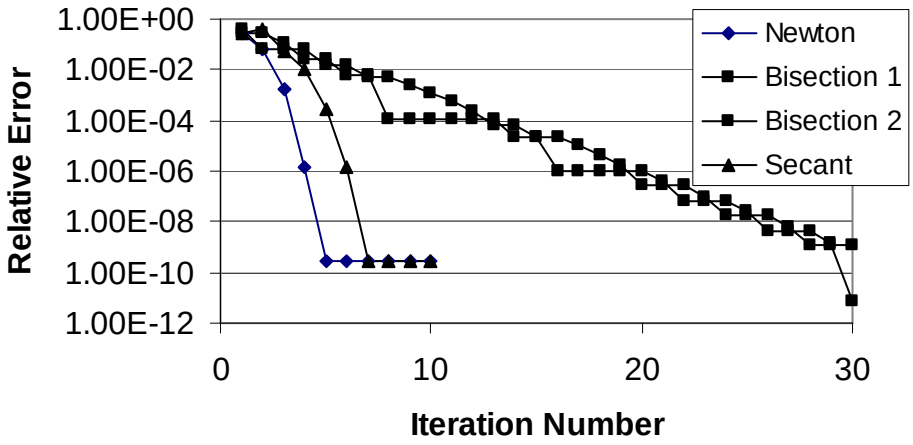

Figure 8. Comparison of Iteration Errors for Various Algorithms for f(x) = x^2-2. Diamond: Local Newton's Method. Square: Upper (Bisection 1) and Lower (Bisection (2) Values of Bisection Method. Triangle: Secant Method.

Mandlebrot Set

The Mandlebrot set is well-known to represent a non-linear model whose boundaries between bounded and unbounded initial guesses consist of a fractal object (Gleick 1987). The color map generated for the Mandlebrot set is shown in Figure 9. The pattern is as expected from similar maps in the literature (Gleick 1987). A black region is visible at M = 0.1 and grows in size until M = 2. Above M = 2, no further growth in size of the black region occurs. It turns out that the black region in this case does not represent convergence but rather lack of divergence. Multiple values are obtained for the "solution" in the black region – a truly chaotic situation. The well-known Mandlebrot set is an iterated function sequence like Newton's method. The boundary between divergence and lack of divergence is a fractal object.

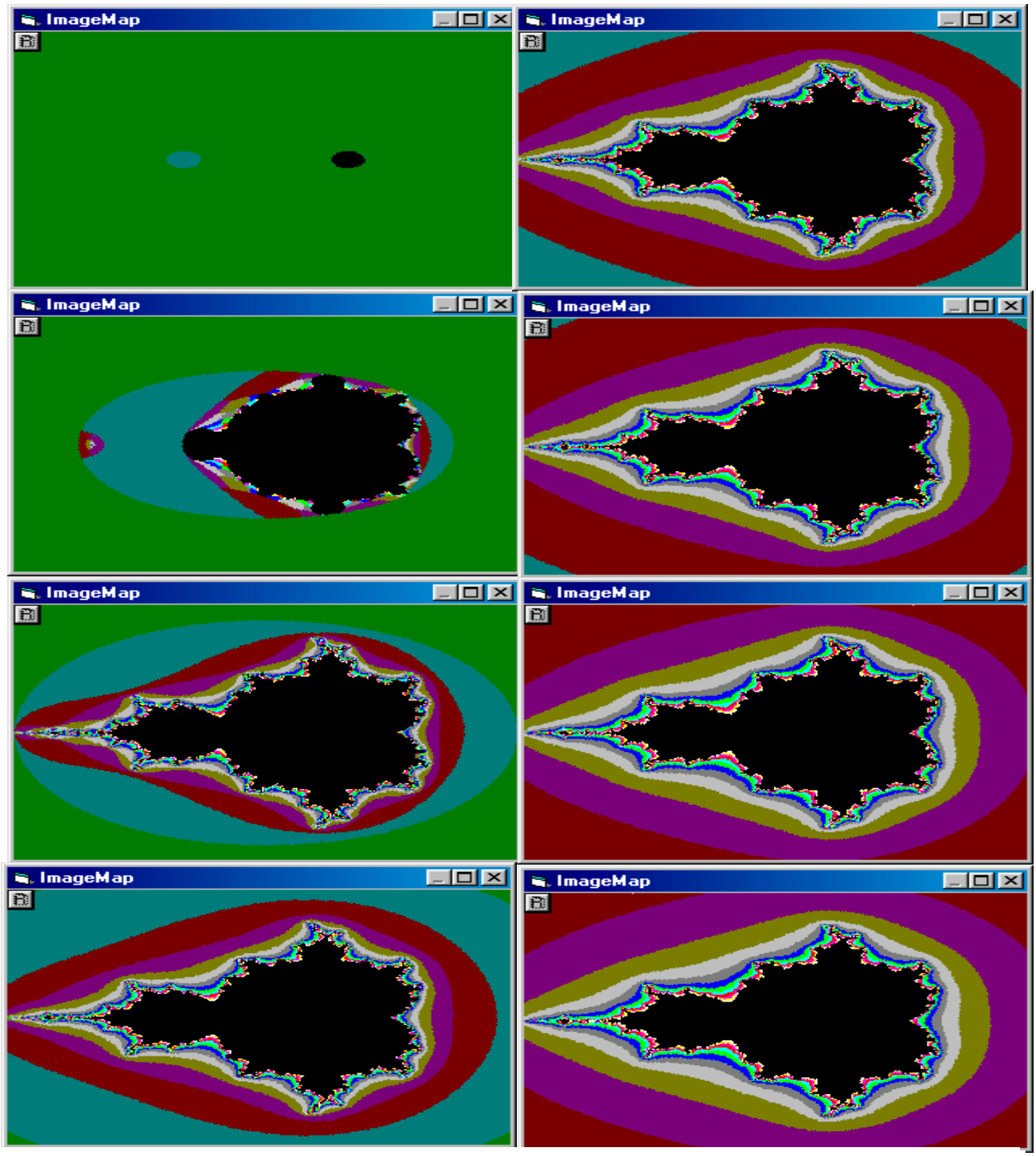

Figure 9. Color Map for Mandlebrot Set Iteration Function. Black Region is Region of No Divergence for Max Value. Top Left Down: Max Value = 0.1, 1, 2, 4. Top Right Down: Max Value = 8, 16, 32, 64. Horizontal Axes: -1.0 to 0.5. Vertical Axes: +/- 0.75i.

Square Root of 2

On the other hand, Newton's method for $x^2-2 = 0$ converges as expected except along the imaginary axis for M > 1.414…. The color map for $x^2-2 = 0$ is shown in Figure 10. In contrast to the Mandlebrot set, the black region in Figure 10 grows without bound as M is increased and the values always converge to a square root of two. It is surmised that the pattern generated along the imaginary axis has a fractal dimension in a similar fashion to a Cantor set (Addison 1997). It is worth noting that the solutions are obviously real but that crafting the problem to include starting points in the complex plane results in a larger number of potential starting points that may be used. The solutions displayed in Figure 10 are the square roots of two for random starting points in the complex plane for the last fifteen starting points. The first column is the real part of the root and the second column is the imaginary part of the root.

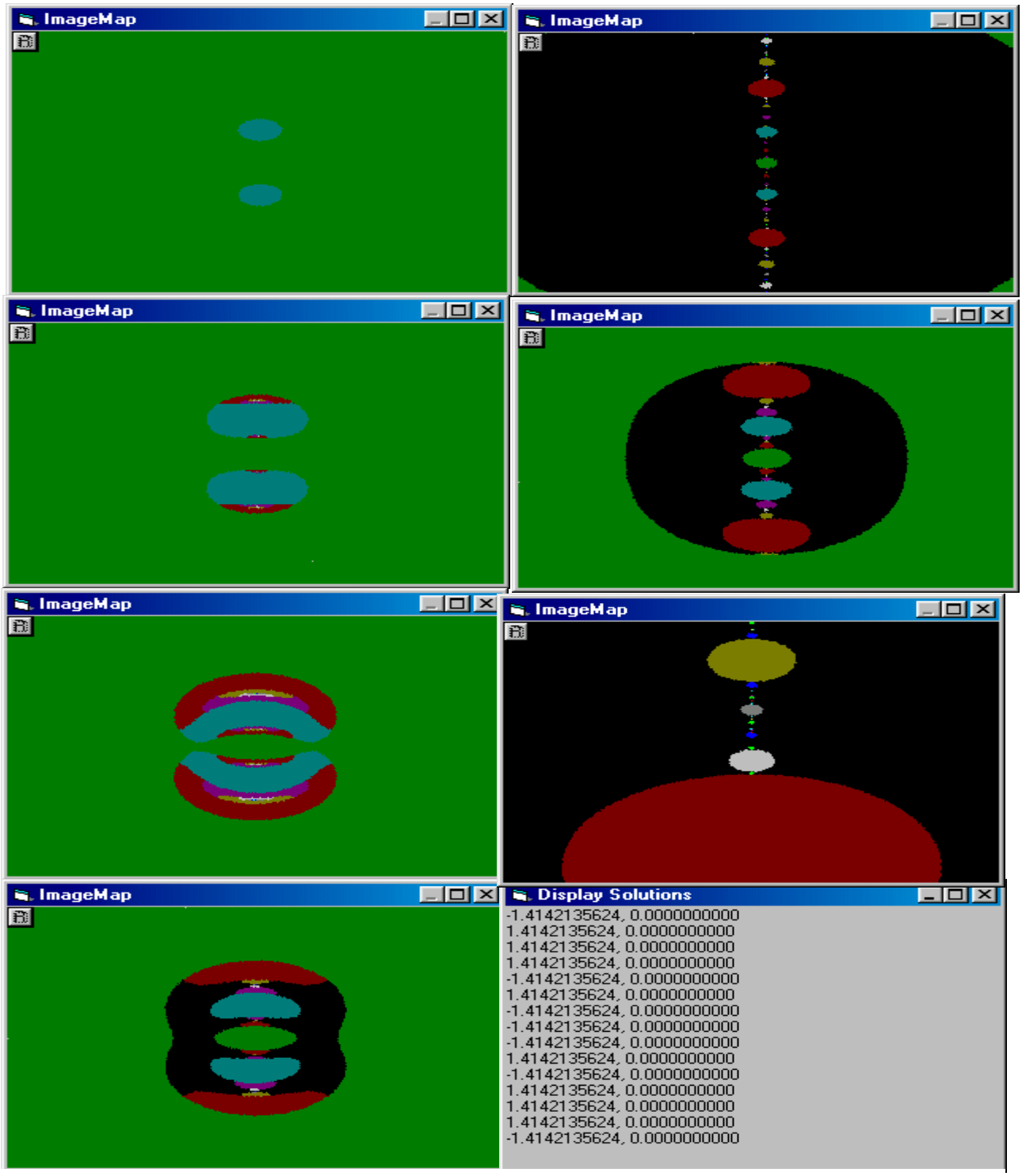

Figure 10. Color Map for $f(x) = x^2-2$ Iteration Function. Black Region is Region of Convergence to $+/-2^{1/2}$ for Max Value. Top Left Down: Max Value = 0.5, 1, 1.4, 1.5. Top Right Down: Max Value = 2, 4, Zoom in 4. Bottom Right: Printout of Convergence Values. Horizontal Axes: +/- 3. Vertical Axes: +/- 3i.

Euler's Equation

The color map for Euler's equation is shown in Figure 11. The solutions are $\pm n\pi i$, n = 1, 3, 5, … to ∞. The infinite number of solutions are restricted by restricting the value of M. If $3\pi > M > \pi$, then we will see $\pm\pi i$. If $5\pi > M > \pi$, then we will see $\pm\pi i$, $\pm 3\pi i$ and so on. Black regions are visible for $M > \pi$. Correct values of π were obtained; however, sometimes roots along the negative imaginary axis were obtained from starting points in the positive half of the complex plane (not shown). More evidence of fractal surfaces between the black convergence region and the region of divergence provide a fascinating view of the convergence behavior of Euler's equation. The major point is that Newton's method has a relatively large set of initial guesses that may be used for Euler's equation. And, it is necessary to restrict the number of iterations and to bound the updated values (with M). When a bad initial guess is picked, it usually diverges very quickly. Also, a good guess may be very close to a bad guess. However, if the iteration function is not well-behaved, as for the Mandlebrot set, then it is necessary to check other starting points in the neighborhood of a given successful point to see if convergence to the same root is achieved.

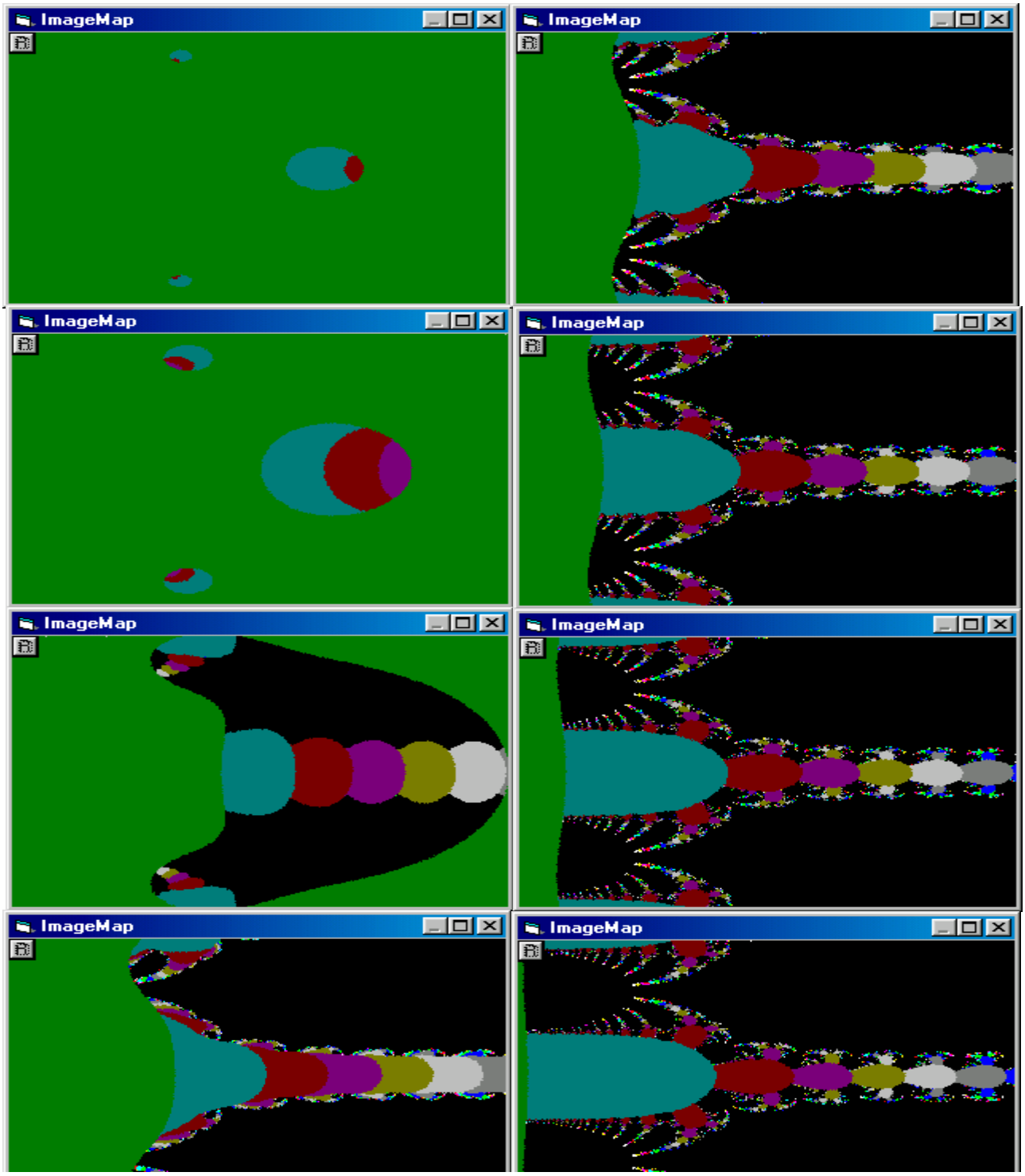

Figure 11. Color Map for Euler's Equation Iteration Function. Black Region is Region of Convergence to +/- πi for Max Value. Top Left Down: Max Value = 1, 2, 4, 8. Top Right Down: Max Value = 16, 32, 64, 128. Horizontal Axes: +/- 2.5. Vertical Axes: +- 2.5i.

One Fourth Roots of One

The color map for $x^4-1 = 0$ is given in Figure 12. A similar analysis was originally presented in (Gleick 1987) except that we use the M value to determine when lack of divergence begins. Four large black convergence regions (that increase in size with M) are indicated. Instead of one line along the imaginary axis with a fractal dimension – observed for the square root of 2 – we now have two lines going off at 45 degrees to the imaginary axis – as might be expected – but with much more complexity in their divergence behavior. It is worth noting that for functions with solutions, increasing M increases the window of convergence. For the Mandlebrot set function, increasing the value of M does not have the same benefit. This behavior of non-convergence is illustrated better with the Modified Mandlebrot Set – next section.

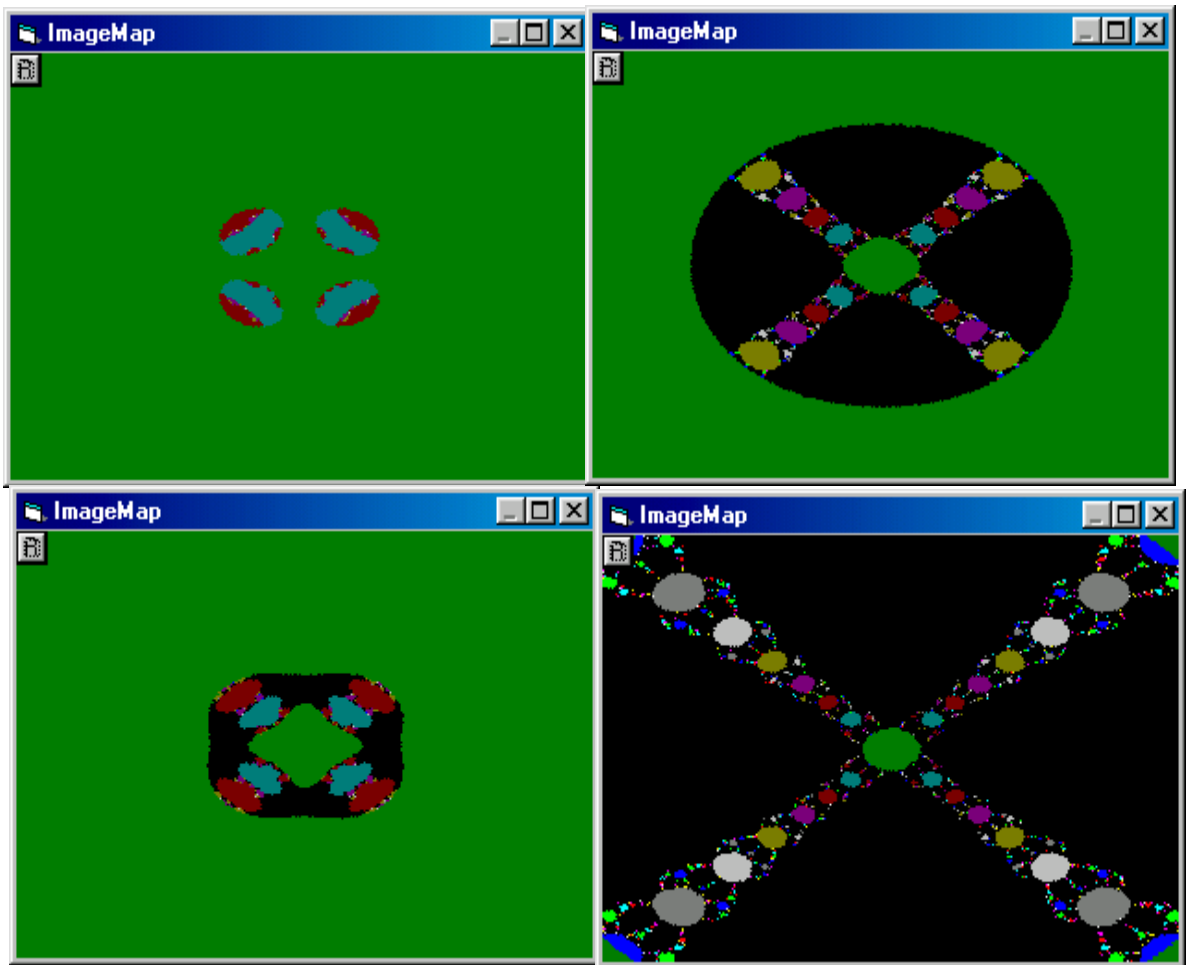

Figure 12. Color Map for x^4-1 Iteration Function. Black Region is Region of Convergence to +/-1, +/-i for Max Value. Top Left Down: Max Value = 0.9, 1.1. Top Right Down: Max Value = 2, 4. Horizontal Axes: +/- 2. Vertical Axes: +/- 2i.

Modified Mandlebrot Function

The color map for the Modified Mandlebrot set is given in Figure 13. Its behavior is analogous to the Mandlebrot set but with more symmetry. And, the black regions again contain a multitude of solutions in a chaotic manner. The Verhulst population growth equation also behaved in a manner similar to the Mandlebrot set (figure not shown). As is well-known, this population growth equation predicts that population growth is chaotic and sensitive to initial conditions. That population growth and the chaos illustrated by the Mandlebrot set are connected is an intriguing concept. Figure 13 ilustrates behavior of a function that does not satisfy the requirements for convergence of Newton's method. After a black region appears, it grows with M up to a certain value of M and then quits growing with M. And, the solutions are bounded random numbers in the region of no divergence (black region). The solutions alternate (chaotically) between 0 and –1, for which the function, equation (29), is undefined. So, when using Newton's method one cannot use lack of divergence to conclude that a solution exists. Also, according to the Euler's equation example, one cannot use convergence to a solution to rule out that other roots may be found with different starting points.

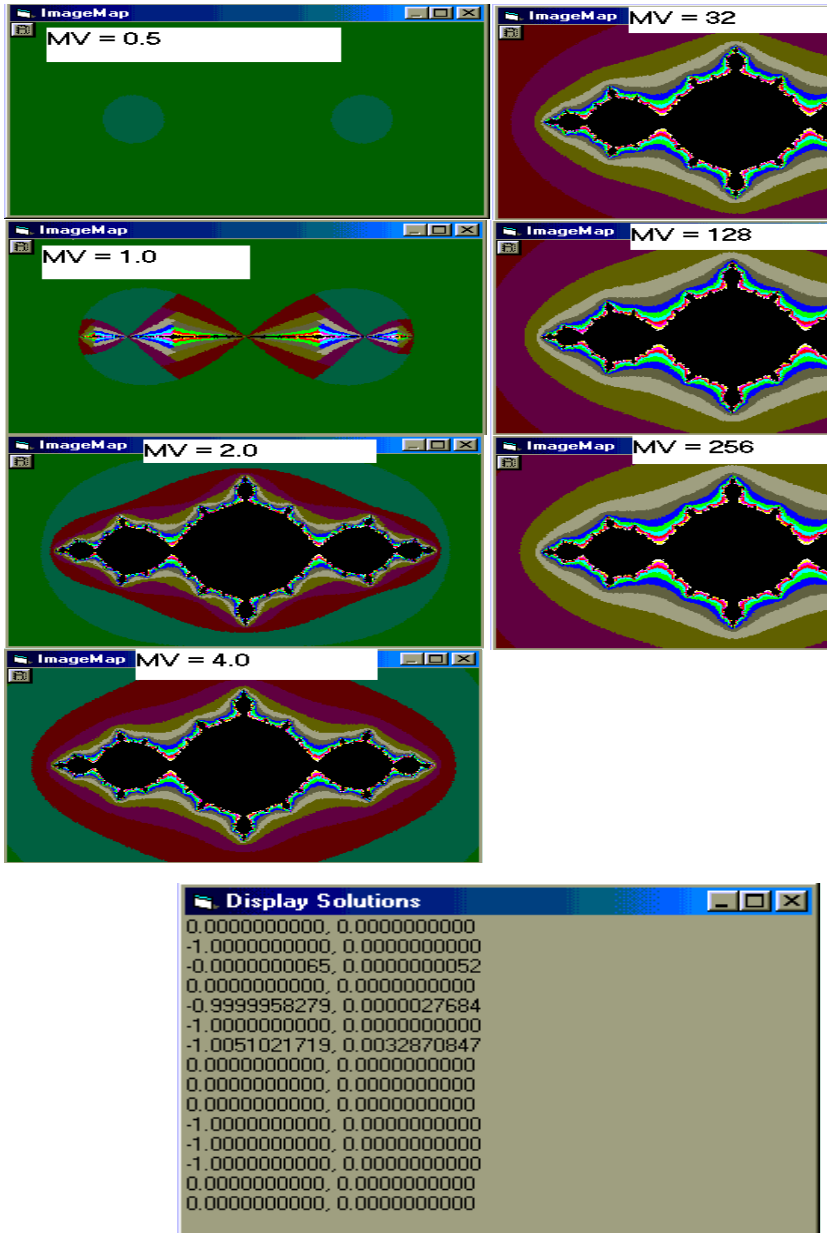

Figure 13. Color Map for Modified Mandlebrot Set Iteration Function. Black Region is Region of No Divergence. Top Left Down: Max Value = 0.5, 1.0, 2.0, 4.0. Top Right Down: Max Value = 32, 128, 256. Horizontal axes: +/- 2. Vertical axes: +/- i. Bottom Center: Values in Black Region after 15 Iterations.

Krieger-Dougherty Equation

We now make a transition from the sublime to the practical applications of Newton's method. The convergence behavior for the Krieger-Dougherty equation, an equation for modeling colloidal dispersions such as milk, tea, beer, industrial coatings formulas, *etc*., with the multivariate Newton's method is shown in Figure 14. The color map for Newton's method is shown in Figure 15. The multivariate data used are given later in Chapter 6. In the present case, the Jacobian matrix was computed analytically. The black region for this case is relatively small but examination of the solutions indicates convergence to the proper values ($[\eta] = 2.5$ and $\varphi_M = 0.63$. See equation (35)).

An interesting experiment would be a test of the convergence behavior with this method that allows the intrinsic viscosity, maximum packing fraction, and relative viscosity to take on complex imaginary values. For example, if a guess of $\varphi_M < \varphi$ is picked, then $(1-\varphi/\varphi_M) < 0$ and the Krieger-Dougherty equation cannot be evaluated – like taking a log of a negative number. If we allow complex numbers, logs of negative numbers are alright. The relative viscosity

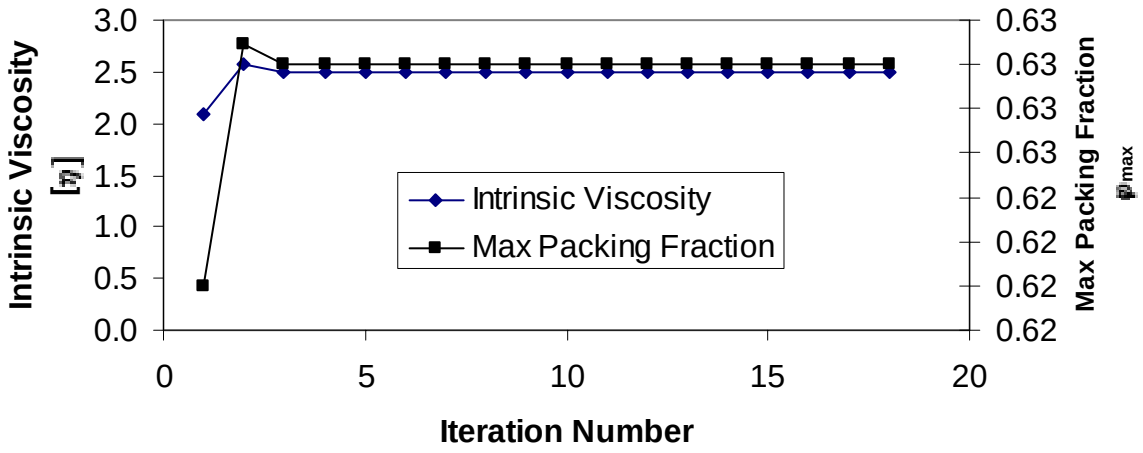

Figure 14. Convergence Behavior of 2D Newton's Method with the Krieger-Dougherty Equation. Squares: Max Packing Fraction. Diamonds: Intrinsic Viscosity.

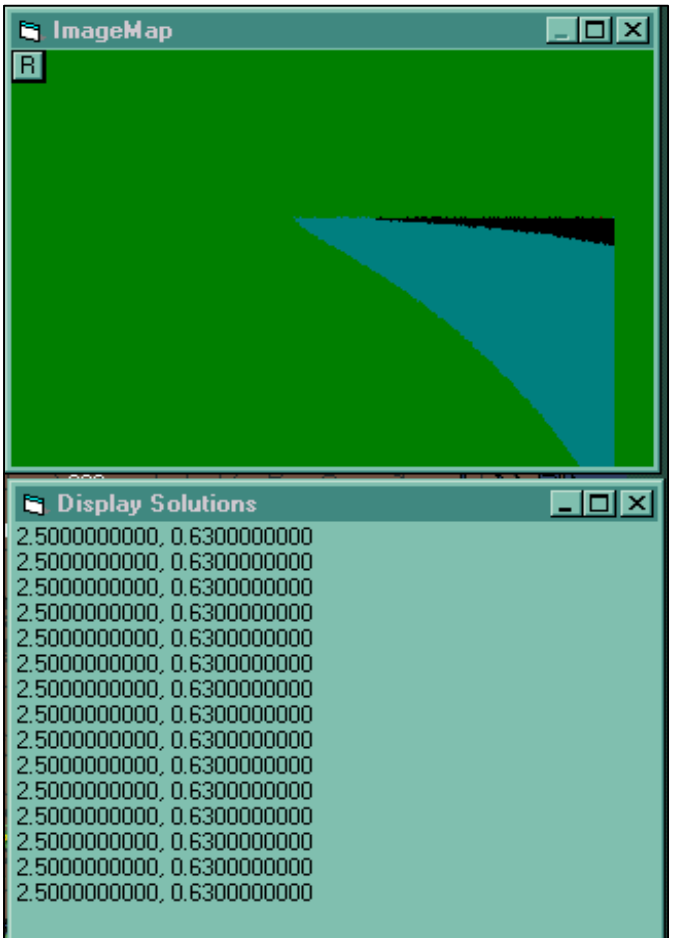

Figure 15. Behavior of Krieger-Dougherty Equation Iteration Function. Black Region is Region of Convergence to $[\eta] = 2.5$, $\varphi_{max} = 0.63$ for Max Value = 100. Top: Color Map. Bottom: Convergence Values for Random Starting Points. Horizontal axes: -1.5 to 4.5. Vertical axes: 0.3 to 1.8.

becomes a complex number: $\eta_r = \exp(-i\pi[\eta]\varphi_M)(\varphi/\varphi_M - 1)^{-[\eta]\varphi M}$; $(1-\varphi/\varphi_M) < 0$ – where we have used Euler's equation to set $\exp(i\pi) = -1$. However, adapting the local Newton's method to a data mining algorithm *and* testing the effect on convergence behavior of using complex numbers for multivariate Newton's method would be beyond the scope of this work.

Comparison of Convergence Behaviors

The square root of two function, $g_{R2}(x)$, converges to the proper value over a very large region of the complex plane. The modified Mandlebrot function, $g_{MM}(x)$, does not diverge – yields many possible values - over a small region of the complex plane. The attractor of $g_{R2}(x)$ is $\sqrt{2}$ as expected and the attractor of $g_{MM}(x)$ is zero. However, $f_{MM}(x)$ is discontinous at 0 which violates the convergence criteria. Furthermore, we have $g'_{R2}(x) = ½ - 1/x^2$ and $g'_{MM}(x) = 2x+1$. Thus, the root two function has quadratic convergence, $g'_{R2}(\xi) = 0$, whereas $|g'_{MM}(x)|$ is always greater than one or equal to one and, as shown above, cannot converge.

Time and space do not permit a discussion of the other cases shown above. However, it is clear that the convergence criteria above have merit but that additional analysis is required to address convergence behavior in the complex plane and for the multivariate Newton's method.

CHAPTER 4

MODELING AND NEWTON'S METHOD

Newton's method is applicable to two broad cases of problems. In one case, we have f(x) = 0 and the algorithm used is typically called the local Newton's method that is discussed in the previous chapter. In the other case, we have to minimize f(x) (min f(x)) and the algorithm is called the global Newton's method.

Local Newton's Method

Consider a vector space $x \in R^n$ and a function $f:R^n \rightarrow R \Leftrightarrow f(x)$. The objective of the local Newton's method is to find a $\xi \in R^n$ such that $f(\xi) = 0$. Using the Taylor series method, the

$$\nabla f(x_L)\, \Delta x = -f(x_L) \tag{37}$$

iterative equations are derived from equation (37):

The above expression generates the normal equations for the local Newton's method. Several examples are given below:

Local Newton's Method for Square Root of Two:

Consider the square root of two function discussed in Chapter 3.

$f(x) = x^2-2 = 0.$

$\nabla f(x) = f'(x) = 2x.$

$2x\, \Delta x = -(x^2-2)$

$x_N = x_L - (x_L^2 - 2)/(2x_L) = (2x_L^2 - x_L^2 +2)/(2x_L)$

$x_N = x_L/2 + 1/x_L$

Starting with a value of 1, this iteration generates the sequence: 1.5, 1.4166, 1.4142.

Local Newton's Method Example for a Two Parameter Function:

Consider the Kreiger-Dougherty equation. In this case, we have $f([\eta], \phi_m; \phi) = 0$. And the vector $x' = ([\eta], \phi_m)$. The normal equations, $\nabla f(x_L)\,\Delta x = -f(x_L)$, generate the following matrix equation (38):

$$\begin{bmatrix} f_{[\eta]}(x_L;\varphi_1) & f_{\varphi_m}(x_L;\varphi_1) \\ f_{[\eta]}(x_L;\varphi_2) & f_{\varphi_m}(x_L;\varphi_2) \\ f_{[\eta]}(x_L;\varphi_3) & f_{\varphi_m}(x_L;\varphi_3) \\ \dots & \\ f_{[\eta]}(x_L;\varphi_m) & f_{\varphi_m}(x_L;\varphi_m) \end{bmatrix} \begin{bmatrix} \Delta[\eta] \\ \Delta\varphi_m \end{bmatrix} = - \begin{bmatrix} f(x_L;\varphi_1) \\ f(x_L;\varphi_2) \\ f(x_L;\varphi_3) \\ \dots \\ f(x_L;\varphi_m) \end{bmatrix} \tag{38}$$

It is convenient to define the Jacobian matrix, **J**, and to write the equation above in a more compact form in equation (39):

$$\mathbf{J}_L\Delta\mathbf{x} = -\mathbf{F}(\mathbf{x}_L) \tag{39}$$

The dimensions of J are m rows (observations) and n columns (number of parameters). In general, J is non-square and the above iterative equation can be solved algebraically as in equation (40):

$$\mathbf{x}_N = \mathbf{x}_L - (\mathbf{J}_L'\mathbf{J}_L)^{-1}\mathbf{J}_L'\mathbf{F}(\mathbf{x}_L) \tag{40}$$

However, the process of duplicating the algebraic matrix inverse and matrix multiplications shown above in a computer program is not always recommended due to the possibility of loss of precision and the possibility that **J** is singular. The SVD technique, discussed in a later chapter, is one of the recommended techniques to solve equation (40).

Global Newton's Method

Now consider $f:\mathbf{R}^n \rightarrow \mathbf{R} \Leftrightarrow f(\mathbf{x})$. To find **x** that minimizes $f(\mathbf{x})$, we want to solve the equation $\nabla f(\mathbf{x}) = \mathbf{0}$. The standard practice is to write $f(\mathbf{x})$ in the Taylor series expansion and to solve for **x** (O'Leary 2000, 2). The algorithm used here is called the global Newton's method. However, it simply applies the local Newton's method to the function $\nabla f(\mathbf{x}) = \mathbf{0}$. There are additional complications that will be explained. For example, the normal equations for the global Newton's method are given in equation (41):

$$\Delta\mathbf{x}'\nabla^2 f(\mathbf{x}_L) = -\nabla f(\mathbf{x}_L) \quad (41)$$

This expression is rather awkward. With a little re-arrangement, it can be cast into a standard normal form. It is customary to define the Hessian matrix, $\mathbf{H} = \nabla^2 f(\mathbf{x})$. And, the Jacobian matrix is defined as $\mathbf{J} = \nabla f(\mathbf{x})$. Using these definitions and taking the transpose of the above expression, we obtain the normal equations for the global Newton's method:
$\mathbf{H}'(\mathbf{x}_L)\Delta\mathbf{x} = -\mathbf{J}'(\mathbf{x}_L)$. When there is no possibility of confusion, we will just write these normal equations as equation (42):

$$\mathbf{H}'_L\Delta\mathbf{x} = -\mathbf{J}'_L \quad (42)$$

It is worth noting that if **H** is too expensive to compute or store, then the following approximation in equation (43) is recommended:

$$\mathbf{H}_L \approx (\mathbf{J}(\mathbf{x}_L+h) - \mathbf{J}(\mathbf{x}_L)) / h \quad (43)$$

Newton's method with the approximate Hessian matrix as shown above is called the Truncated Newton's Method (O'Leary 2000, 8).

The dimensions of the Hessian matrix are n (number of parameters) x n. As a square matrix, it is possible to algebraically solve the normal equations for $\Delta\mathbf{x}$. As before, a direct solution is not recommended due to the possibility of loss or precision due to machine round-off error. The LU decomposition or the Choleski decomposition are recommended instead.

Maximum Likelihood Estimation

There are quite a few problems that may be solved with either the local or global Newton's method. Now let us consider a special case of the global Newton's method. Suppose we have a model $\mu(X(\mathbf{x};\mathbf{a}_j))$ of some observable quantity where $\mathbf{x} \in \mathbf{R}^n$, X is a function set, and $\mathbf{a}_j$ is a database record. The ordered pairs, $\{X(\mathbf{x};\mathbf{a}_j), M_j\}$, correspond to a sequence of measurements, M_j, corresponding to independent variables x_i. If the model is a true representation of the system, then $\mu(X(\mathbf{x};\mathbf{a}_j)) = M_j$ for all j.

The confidence in the model, μ, is represented as a conditional probability: P(μ|D) read as the probability of the model, μ, given the data D and is also called the updated belief. The Bayesian probability equation then gives equation (44):

$$P(\mu|D) = P(\mu)\, P(D|\mu) / P(D) \tag{44}$$

The probability P(μ) is called the prior and is the confidence level in the model prior to having any data. The probability P(D|μ) is the likelihood and is the probability that the data are correct given the model. Since P(μ) is usually fixed and P(D), the evidence, is independent of the model parameters, it is usual to assume that P(μ|D) can be maximized by maximizing the likelihood (Baldi 1998, 50).

Due to random errors, the difference $M_j - \mu_j$ will follow some sort of probability distribution. Suppose that $\rho(M_j, \mu(X(\mathbf{x};\mathbf{a}_j)))$ is the negative logarithm of that probability density. The likelihood of the data set, $D = \{\mathbf{x}_0;\mathbf{a}_j, M_j\}$, and the model, $\mu(X(\mathbf{x};\mathbf{a}_j)$, is then given as equation (45):

$$P(D\,|\,\mu) \propto \prod^{m} \exp\left[-\rho\left(M_j, \mu\left[X\left(\mathbf{x}\ ;\mathbf{a}_j\right)\right]\right)\right]\Delta M \tag{45}$$

We would now like to maximize P by appropriate selection of the parameter vector **x**. These results are a modification of the method of local estimates from Press *et al.* (1988, 700). But, maximizing P is the same as minimizing the negative logarithm of P as in equation (46):

$$-\ln P \propto \sum_{i=1}^{m} \rho\left(M_j, \mu\left\{X(\mathbf{x};\mathbf{a}_j\right\}\right) \equiv f(\mathbf{x}) \tag{46}$$

If we now solve the minimization problem for f(**x**) by the global Newton's method technique, we obtain the set of parameters, **x**, that will maximize the probability that the data and the model are correct, i.e., that will maximize the likelihood. We will now illustrate with a few examples.

Case 1: Linear Least Squares

Suppose that the M_j are normally distributed, $X(\mathbf{x}) = \mathbf{x}$, and $\mu(\mathbf{x};\mathbf{a}_j) = \langle\mathbf{a}_j,\mathbf{x}\rangle$ where $\langle\mathbf{a}_j,\mathbf{x}\rangle$ is the dot product of $\mathbf{a}_j$ and **x**. Then, we have equation (47):

$$\rho = \frac{1}{2}\left(\frac{M_j - \mu(\mathbf{x}; \mathbf{a}_j)}{\sigma} \right)^2 \quad (47)$$

The minimization problem then becomes equation (48):

$$f(\mathbf{x}) = \sum_{j=1}^{m} \left(M_j - \langle \mathbf{a}_j, \mathbf{x} \rangle \right)^2 \quad (48)$$

Since σ is constant, it is not required in the function f(**x**). After some reflection, it is easy to see that f(**x**) = $\boldsymbol{\delta}'\boldsymbol{\delta}$ where $\boldsymbol{\delta}$ is a vector and $\boldsymbol{\delta}_j = M_j - \langle \mathbf{a}_j, \mathbf{x} \rangle$. In matrix form, we have $\boldsymbol{\delta} = \mathbf{M} - \mathbf{ax}$. Now f(**x**) may be minimized by matrix differentiation. For example:

$\partial f(\mathbf{x})/\partial \mathbf{x}' = (\partial/\partial \mathbf{x}')\, \boldsymbol{\delta}'\boldsymbol{\delta} = (\partial/\partial \mathbf{x}')[(\mathbf{M}'-\mathbf{x}'\mathbf{a}')(\mathbf{M}-\mathbf{ax})]$

$= (\partial/\partial \mathbf{x}')(\mathbf{M}'\mathbf{M}-\mathbf{x}'\mathbf{a}'\mathbf{M}-\mathbf{M}'\mathbf{ax}+\mathbf{x}'\mathbf{a}'\mathbf{ax}) = -\mathbf{a}'\mathbf{M}+\mathbf{a}'\mathbf{ax}$.

So, either **a**' = 0 or equation (49) follows:

$$-\mathbf{M}+\mathbf{ax} = \mathbf{0} \Leftrightarrow \mathbf{ax} = \mathbf{M} \quad (49)$$

But, the equations represented by Equation (49) are just the normal equations for the familiar linear least squares regression method where the **x**(nx1) vector contains the coefficients, **a**(mxn) are the database records, and **M**(mx1) represents the response values. These normal equations are well-known to solve the minimization of |**M**-**ax**| (O'Leary 2000, 3). And, this example provides a good illustration of why the variance of the dependent variable is required to be constant and normal in that case. Numerous illustrations of this normality constraint exist (O'Leary 2000, 8). An important point to remember is that, in many cases, a few points- usually termed as outliers- exist that violate the normality assumption required by linear least squares algorithms and easily result in the wrong answer. The techniques of robust regression discussed below are designed to prevent such wrong answers from occurring due to outliers but are computationally expensive.

Case 2: Linear Model with Uncertainty in Independent Variables

Now suppose we have the same situation as case 1 but that the residuals of the independent and dependent variables are not necessarily normally distributed. This development is based on Press *et al.* (1988, 666-671). Equation (50):

$$\sigma_j^2 = \sigma_{Mj}^2 + \sum_{i=1}^{n} x_i^2 \sigma_{ij}^2 \tag{50}$$

And, we obtain equation (51):

$$f(\mathbf{x}) = \sum_{j=1}^{m} \frac{\left(M_j - \langle \mathbf{a}_j, \mathbf{x} \rangle\right)^2}{\sigma_j^2} \tag{51}$$

However, f(**x**) in this case cannot be differentiated as before due to the presence of the parameters, σ_j and x_i, in the denominators of the terms of the sum.

We define $\boldsymbol{\delta}_{\text{linear}}$ to be the residual vector from case 1. And, we define $\boldsymbol{\delta}_{\text{non-linear}}$ in equation (52):

$$\delta_{\text{non-linear}} = W^{-1}\delta_{\text{linear}} \ni W_{ij} = \left\{ \begin{matrix} 0; \forall\ i \neq j \\ \sigma_j; \forall\ i = j \end{matrix} \right\} (i, j = 1 \text{ to } m) \tag{52}$$

Then, it follows that we have equation (53) for f(**x**):

$$f(\mathbf{x}) = \boldsymbol{\delta}'_{non-linear}\boldsymbol{\delta}_{non-linear} = \boldsymbol{\delta}'_{linear}\mathbf{W}^{-2}\boldsymbol{\delta}_{linear} \tag{53}$$

The Jacobian of f(**x**) is given by:

$$[J(\mathbf{x})]_j = [\nabla f(\mathbf{x})]_j = \sum_{k=1}^{m} \frac{2\delta_{non-lin,k}}{\sigma_k^2}\left[\sigma_k \frac{\partial \delta_{linear,k}}{\partial x_j} - \delta_{linear,k}\frac{\partial \sigma_k}{\partial x_j}\right] \tag{54}$$

The Hessian matrix of f(**x**) is given by equation (55):

$$\begin{aligned}[H(\mathbf{x})]_{ij} = [\nabla^2 f(\mathbf{x})]_{ij} = &\sum_{k=1}^{m} \frac{2}{\sigma_k^4}\left[\sigma_k \frac{\partial \delta_{linear,k}}{\partial x_j} - \delta_{linear,k}\frac{\partial \sigma_k}{\partial x_j}\right]\left[\sigma_k \frac{\partial \delta_{linear,k}}{\partial x_i} - \delta_{linear,k}\frac{\partial \sigma_k}{\partial x_i}\right] + \\ &\sum_{k=1}^{m} \frac{2\delta_{non-lin,k}}{\sigma_k^2}\left[\sigma_k \frac{\partial^2 \delta_{linear,k}}{\partial x_j \partial x_i} + \frac{\partial \sigma_k}{\partial x_i}\frac{\partial \delta_{linear,k}}{\partial x_j} - \frac{\partial \delta_{linear,k}}{\partial x_i}\frac{\partial \sigma_k}{\partial x_j} - \delta_{linear,k}\frac{\partial^2 \sigma_k}{\partial x_j \partial x_i}\right] - \\ &\sum_{k=1}^{m} \frac{4\delta_{non-lin,k}}{\sigma_k^3}\left[\sigma_k \frac{\partial \delta_{linear,k}}{\partial x_j} - \delta_{linear,k}\frac{\partial \sigma_k}{\partial x_j}\right]\left(\frac{\partial \sigma_k}{\partial x_i}\right)\end{aligned} \tag{55}$$

The normal equations are given as equation (56):

$$\mathbf{H}'_L \Delta\mathbf{x} = -\mathbf{J}'_L \tag{56}$$

The above expressions for f(**x**), **J**(**x**), and **H**(**x**) are more general then they appear. Notice that no assumptions have been made about the functional form of the model function or of the variance terms. In many cases, many of the terms will (hopefully) drop out. For example, the second derivatives of the standard deviation, σ_k, will usually be zero. And, the values for δ_k are approximately zero when the current value of the **x** vector is close to the solution. In many cases, the form of the variance terms will be dictated by the type of problem. For example, if the data are taken from mean values, a constant value of σ is used and the expressions are greatly simplified. If we are dealing with count data, then a Poisson distribution may be appropriate in which case the variances are equal to the means. If we are dealing with frequency data, then it is reasonable to set the variances to p(1-p) where p is the percentage of occurrence. In other cases, plots of variance versus mean values could be used to determine the form of the variance. In some instances, the noise is intentionally added to a database (Tendrick and Matloff 1994).

Case 3: Linear Model with Non-Normal Residuals

Consider again the weighted residuals in equation (57) as shown in Case 2:

$$f(\mathbf{x}) = \delta'_{non\text{-}linear}\delta_{non\text{-}linear} = \delta'_{linear}W^{-2}\delta_{linear} \tag{57}$$

If the variances for the independent variables are zero, then the problem again becomes a linear least squares problem with the **a** matrix replaced by $\mathbf{W}^{-1}\mathbf{a}$. The normal equations are as before with this substitution shown in equation (58):

$$\mathbf{a}'\mathbf{W}^{-2}\mathbf{a}\mathbf{x} = \mathbf{a}'\mathbf{W}^{-1}\mathbf{M} \tag{58}$$

Furthermore, this case can be transformed into the augmented system of equation (59):

$$\begin{bmatrix} W & \mathrm{a} \\ a' & 0 \end{bmatrix}\begin{bmatrix} \delta_{linear} \\ x \end{bmatrix} = \begin{bmatrix} M \\ 0 \end{bmatrix} \tag{59}$$

This case is discussed at length by O'Leary (2000, 6).

Case 4: Non-linear Robust Regression

If the W matrix is replaced by some function of the size of the residual, then the effect of outliers that may distort the solution is reduced. This technique is sometimes called robust regression. Furthermore, suppose that the general probability function is such that we have $\rho = \rho(\delta_i/\sigma_i)$. The deviation, $\delta_i = M_i - \mu(\mathbf{x};\mathbf{a}_j)$, uses the same definition as above. For convenience, define $\rho_i = |\rho(\delta_i/\sigma_i)|^{1/2}$. Define $\partial\delta_i/\partial a_j = \delta_{ij}$. Define $\psi(z) = \partial\rho(z)/\partial(z)$, $\psi_i \equiv \partial\rho(z)/\partial z\ z = (\delta_i/\sigma_i)$. Define $\Delta_{ij} = \partial\delta_i/\partial x_j$. Furthermore, define $\delta_{ikj} = \partial\delta_{ij}/\partial x_k$. Finally, define the matrices Λ_k such that $[\Lambda_k]_{ij} = \Lambda_{ikj} = \partial\Delta_{ij}/\partial x_k$. Then, the function to be minimized is:

$F(\mathbf{x}) = \boldsymbol{\rho}'\boldsymbol{\rho}$. ($\boldsymbol{\rho}$ is an mx1 vector of weighted generalized model residuals.)

The Jacobian of $F(\mathbf{x})$ is given in equation (60):

$$\mathbf{J(a)} = \boldsymbol{\psi}'\mathbf{W}^{-1}\boldsymbol{\Delta} \tag{60}$$

($\boldsymbol{\psi}$ is an mx1 vector and $\boldsymbol{\Delta}$ is an mxn matrix.)

The Hessian matrix of $F(\mathbf{x})$ is given in equation (61):

$$\mathbf{H} = \nabla\left(\boldsymbol{\psi}'\mathbf{W}^{-1}\boldsymbol{\Delta}\right) + \begin{bmatrix} \left(\boldsymbol{\psi}'\mathbf{W}^{-1}\right)\boldsymbol{\Lambda}_1 \\ \left(\boldsymbol{\psi}'\mathbf{W}^{-1}\right)\boldsymbol{\Lambda}_2 \\ \cdots \\ \left(\boldsymbol{\psi}'\mathbf{W}^{-1}\right)\boldsymbol{\Lambda}_n \end{bmatrix} \tag{61}$$

The normal equations are given by:

$\mathbf{H}'_L \Delta\mathbf{x} = J'_L$. For the case of normally distributed residuals, $\rho(z) = (1/2)z^2$, we have $\psi(z) = z$. Then, $\nabla\psi = \mathbf{W}^{-1}\Delta$. And, the normal equations become equation (62):

$$\Delta'\mathbf{W}^{-2}\Delta\,\Delta\mathbf{x} = \Delta'\mathbf{W}^{-2}\delta' \tag{62}$$

Now, if the residuals are not normally distributed and we are close to the solution, then the Hessian can be approximated by its first term and we have equation (63):

$$\Delta'\mathbf{W}^{-1}(\nabla\psi)\,\Delta\mathbf{x} = \Delta'\mathbf{W}^{-1}\psi \tag{63}$$

Suggested weighting functions are (Press *et al*. 1988, 701-702):

1. Normal. $\rho(z) = (1/2)z^2$, $\psi(z) = z$.
2. Double Exponential: $\rho(z) = |z|$, $\psi(z) = \text{sgn}(z)$.
3. Cauchy or Lorentzian: $\rho(z) = \log(1+z^2/2)$, $\psi(z) = z/(1+z^2/2)$.

Since it is often the case that the weighting functions are not well behaved, use of other optimization techniques is recommended in addition to the normal equations shown above. The simplex EVOP technique is one such alternative method for solving the robust regression minimization problem although it does not offer the quadratic convergence rate of Newton's method (Press *et al.* 1988, 702).

Converting Local to Global Newton's Method

Occasions may arise in the solution of $f(\mathbf{x}) = 0$ when the derivatives are too costly to compute or $f(\mathbf{x})$ is discontinuous. In such cases, a global optimization method for F(x) such as given in equation (64) might be used:

$$F(x) = \int_a^x f(s)ds \tag{64}$$

We will show in the next chapter that this integral may be replaced by the sum, $\delta'\delta$, and we will show how to calculate the updated belief probability for the multivariate Newton's method.

Summary of Modeling with Newton's Method

The following is a summary of the various algorithms discussed in this chapter. Each algorithm is a variation on the gradient descent method in the form $R^{-1}\Delta x = \nabla C(x)$ where R is the learning rate and C(x) is a cost function.

1. Equation (37) $\nabla f(x_L)\,\Delta x = -f(x_L)$. Local Newton's method. Functions of one variable. Solving f(x) = 0.
2. Equation (39) $\mathbf{J}_L\Delta\mathbf{x} = -F(\mathbf{x}_L)$. Local Newton's method normal equations. Functions of more than one variable. Requires an independent variable in addition to the parameters to be determined.
3. Equation (42) $\mathbf{H'}_L\Delta\mathbf{x} = \mathbf{-J'}_L$. Global Newton's method normal equations. Minimize a function f(**x**).
4. Equation (49) $\mathbf{a'ax} = \mathbf{a'M}$. Global Newton's method. Maximum Likelihood Estimation. Normally distributed residuals. Reduces to Linear Least Squares Regression.
5. Equation (56) $\mathbf{H'}_L\,\Delta\mathbf{x} = \mathbf{-J'}_L$. Linear model with uncertainty in independent variables.
6. Equation (58) $\mathbf{a'W^{-2}ax} = \mathbf{a'W^{-1}M}$. Linear model with non-normal residuals.
7. Equation (62) $\boldsymbol{\Delta}'\mathbf{W}^{-2}\boldsymbol{\Delta}\,\Delta\mathbf{x} = \boldsymbol{\Delta}'\mathbf{W}^{-2}\boldsymbol{\delta}'$. Non-linear robust regression. Normally distributed residuals.
8. Equation (63) $\boldsymbol{\Delta}'\mathbf{W}^{-1}(\nabla\boldsymbol{\psi})\,\Delta\mathbf{x} = \boldsymbol{\Delta}'\mathbf{W}^{-1}\boldsymbol{\psi}$. Non-linear robust regression. Non-normally distributed residuals.

CHAPTER 5

MATRIX ALGEBRA

All the Newton's method equations in the previous section are of the form $\mathbf{Ax} = \mathbf{b}$. **A** is a rectangular matrix (mxn), **x** (nx1) is a parameter vector, and **b** (mx1) is a constant vector. Solving $\mathbf{Ax} = \mathbf{b}$ directly involves left multiplication by **A**', an $O(mn^2 + mn)$ operation. Then, $(\mathbf{A'A})^{-1}$ is calculated - $O(n^3)$. Finally, **x** is found by left multiplication by $(\mathbf{A'A})^{-1}$, an $O(n^3)$ operation. The total number of operations are $O(n^3+mn^2+mn)$. The error accumulation in the **x** vector amounts to a factor of m(n+1) times the machine precision per factor assuming no error in the calculation of the inverse. For these reasons, the direct method is not recommended (Press *et al*. 1988, 34). The improvement of matrix computations beyond the above direct method is an important area for mathematical and computer research (Nash 1990, 19).

Developments in linear algebra have contributed a great deal to progress in solving optimization problems on computers. The relationship between optimization algorithms and matrix methods in linear algebra has been called symbiotic (O'Leary 2000, 1). Many literature citations exist and there are many textbooks and references on the subject including Stewart (1973), Strang (1976), Golub and Van Loan (1983), Kahan, Molar and Nash (1989), and Bjorck(1996). We will give brief descriptions of Gauss elimination, Gauss-Jordan elimination, LU Decomposition, Cholesky Factorization, QR Factorization, and Singular Value Decomposition that are some of the main techniques used to efficiently solve matrix equations of the form $\mathbf{Ax} = \mathbf{b}$ and, thus, to solve optimization problems such as equations using algorithms in the previous chapter. However, complete coverage of the subject is beyond the scope of this work. There are also specialized techniques available that involve choices related to the nature of the problem and the sparsity of the **A** matrix.

Gauss Elimination

Solution of **Ax = b** by Gaussian elimination is conversion of **A** to a triangular matrix by elementary operations followed by determination of **x** by back substitution (Harris and Stocker 1998, 441). These terms are be defined for later use.

- Triangular matrix. A matrix whose elements are zero on one side of its diagonal.
  - Upper triangular matrix. Non-zero elements of the triangular matrix are on or above the diagonal.
  - Lower triangular matrix. Non-zero elements of the triangular matrix are on or below the diagonal.
- Elementary operations. Operations on the matrix equation **Ax=b** that do not change the answer. There are three types of elementary operations on the matrix equation **Ax = b**.
  - Multiply a row of **A** and corresponding row of **b** by a scalar factor.
  - Add or subtract a multiple of a row of **A** and the corresponding row of **b** to another row of A and corresponding row of b. Mathematically, this multiplication is represented by equation (65):

$$
\begin{aligned}
&(1 - s_{rc}E^{rc})Ax = (1 - s_{rc}E^{rc})b \equiv E_s^{rc}Ax = E_s^{rc}b \\
&\text{where,} \\
&s_{rc} \text{ is a scalar multiplier for the operation } E_s^{rc}, \\
&E_{ij}^{rc} = \left\{ \begin{matrix} 1 \text{ if } i = r \wedge j = c \\ 0 \text{ if } i \neq r \vee j \neq c \end{matrix} \right\}, \\
&E_s^{rc} \equiv (1 - s_{rc}E^{rc})
\end{aligned}
\tag{65}
$$

  - Row Pivoting. Interchanging two rows of **A** and the corresponding rows of **b**. Row pivoting amounts to multiplying both sides of **Ax=b** by a matrix $\mathbf{R}^{rs}$ to give $\mathbf{R}^{rs}\mathbf{Ax} = \mathbf{R}^{rs}\mathbf{b}$ that interchanges row r and row s. The row pivot matrix, $\mathbf{R}^{rs}$, is given by equation (66):

$$
R_{ij}^{rs} = \left\{ \begin{matrix} 1 \text{ if } i = r \wedge j = s \\ 1 \text{ if } i = s \wedge j = r \\ 0 \text{ otherwise} \end{matrix} \right\} \tag{66}
$$

- Column Pivoting. Interchanging two columns of **A** and the corresponding rows of **x**. Column pivoting may be represented by multiplication by a matrix C^{cs} to obtain: $AC^{cs}[C^{cs}]^{-1}x = b$ which represents the interchange of columns c and s in **A** and rows c and s of **x**. Note that since the operation is a row pivot on **x**, then $[C^{cs}]^{-1} = R^{cs}$.

- Back substitution. The process of solving **Ax=b** when **A** is in triangular form. For example, if **A** is in upper triangular form, then $x_n = b_n/A_{nn}$. But, since **A** is upper triangular, we have $A_{n-1,n-1}x_{n-1} + A_{n-1,n}x_n = b_{n-1} \Rightarrow x_{n-1} = (b_{n-1} - A_{n-1,nxn}x_n)/A_{n-1,n-1}$. This pattern is repeated until we have solved for x_1.

With the above definitions, it is now possible to illustrate the Gaussian elimination method. If, in $\mathbf{E}^{rc}{}_s$, we require that $s_{rc} = A_{rc}/A_{cc}$, then the formation of an upper triangular matrix is represented in equation (67):

$$\left[\prod_{j=1}^{n}\left\{\prod_{i=j+1}^{m}\mathbf{E}_s^{ij}\right\}\right]\mathbf{Ax} = \left[\prod_{j=1}^{n}\left\{\prod_{i=j+1}^{m}\mathbf{E}_s^{ij}\right\}\right]\mathbf{b} \tag{67}$$

It is important to note that the scalar multiplier is computed based on the current modification of the **A** matrix. The above procedure can fail through attempted division by a zero valued diagonal element even if the **A** matrix is non-singular. A zero valued diagonal element may result by chance due to the combination of previous operations. Thus, row pivoting is absolutely required for stability of the method as in equation (68).

$$\left[\prod_{j=1}^{n}\left\{\prod_{i=j+1}^{m}\mathbf{E}_s^{ij}\right\}\mathbf{R}^{jMaxj}\right]\mathbf{Ax} = \left[\prod_{j=1}^{n}\left\{\prod_{i=j+1}^{m}\mathbf{E}_s^{ij}\right\}\mathbf{R}^{jMaxj}\right]\mathbf{b}$$

where Maxj is the row in column j below the diagonal with the largest value. (68)

It is further helpful to pivot the largest element in a column to the diagonal in order to help reduce round-off error. Other types of pivoting, including column pivoting, are possible. There is great deal of debate about pivoting methods in the literature (Press *et al*. 1988, 38-39).

In practice, E and R are sparse matrices and it is more efficient to perform the elimination directly instead of with matrix operations. Also, it is not necessary to keep the lower triangle of **A** up to date. The algorithm is usually performed in place and the final A contains the upper

triangular values and garbage in the lower part. Several algorithms for Gaussian elimination are available in the literature (Nash 1990, 76; Harris and Stocker 1998, 441-445; Press *et al*. 1988, 41-43).

Gauss-Jordan Elimination

The Gaussian elimination technique solves the equation **Ax=b** by converting **A** to a triangular matrix. Now, suppose we have a new **b** vector. Then, **A** will have to be reconstructed and the process repeated. With Gauss-Jordan elimination, **A** is converted to the unit matrix, the inverse of **A** is calculated and the new **b** vectors may be converted to new **x** vectors by multiplication by $\mathbf{A}^{-1}$. The process is an extension of Gaussian elimination in which the upper triangular matrix formed is further processed by elementary operations to the unit matrix. Solving the augmented matrix equation (69) with the Gauss-Jordan algorithm results in solutions for $x_1, x_2, \ldots, x_n$, and $\mathbf{A}^{-1}$ (Press *et al*. 1988, 37).

$$\mathbf{A}[\mathbf{x_1} \sqcup \mathbf{x_2} \sqcup \ldots \sqcup \mathbf{x_n} \sqcup \mathbf{A}^{-1}] = [\mathbf{b_1} \sqcup \mathbf{b_2} \sqcup \ldots \sqcup \mathbf{b_n} \sqcup \mathbf{1}] \quad (69)$$

However, the additional storage required to perform the algorithm and the round-off error that results from multiplication of new **b** vectors by $\mathbf{A}^{-1}$ make the Gauss-Jordan method less attractive than other algorithms discussed below. A program for the Gauss-Jordan algorithm may be found in Press *et al.* (1988, 39-40).

LU Decomposition

If no pivoting is required for its Gaussian elimination, then matrix **A** may be written as a product of a lower triangular matrix, **L**, and an upper triangular matrix, **U** (Harris and Stocker 1998, 445). The matrix equation **Ax = b** becomes: **Ax = b** => **LUx = b**. If we solve for **Ly = b** by back substitution, then **x** is obtained by back substitution, also, since **Ux = y**. Thus, factoring **A** into **LU** allows a solution by two back substitution steps. If new **b** vectors become available, then the backsubstitution steps are repeated on those new **b** vectors.

The LU decomposition of **A** is not unique. For example, in Doolittle's decomposition, **U** is equal to the coefficients from Gaussian elimination, the diagonal elements of **L** are set equal to **1**, and the remaining elements of **L** are found from the definition of matrix multiplication. In Crout's decomposition, the diagonal elements of **U** are set equal to **1**, the first column of **L** is set equal to the first column of **A**, and the elements of **L** and **U** are determined from left to right and

top to bottom in order by the definition of matrix multiplication. LU decomposition is the preferred method when dealing with sparse matrices. Reordering of rows in the case of sparse matrices is recommended in order to prevent fill-in (O'Leary 2000, 4). For non-sparse matrices, QR factorization or SVD are usually preferred. Algorithms for LU decomposition are given by Harris and Stocker(1998, 445) and Press *et al.* (1988, 43-48).

Cholesky Factorization

When **A** is a symmetric positive definite matrix, the LU factorization technique of Cholesky may be used to solve **Ax = b**. Matrix **U** is chosen such that **U = L'**. The value L_{11} is set to $(A_{11})^{1/2}$. And the remaining values are determined according to the rules of matrix multiplication. For example:

$$
\begin{aligned}
L_{j1} &= A_{j1} \Big/ U_{11} \\
L_{jj} &= \left(A_{jj} - \sum_{j-2}^{n} \sum_{s=1}^{j-1} L_{js}^2 \right)^{1/2} \\
L_{jk} &= \left(\frac{1}{L_{kk}} \right) \left(A_{jk} - \sum_{s=1}^{j-1} \sum_{j=k+1}^{n} L_{js} L_{ks} \right) \quad (k \geq 2)
\end{aligned}
\tag{70}
$$

The symmetry of the problem results in fewer total operations than for the regular LU factorization. The Choleski algorithm is given by Harris and Stocker (1998, 445-448).

QR Factorization

QR factorization again decomposes the matrix with an upper triangular matrix on the right, called the **R** matrix instead of the **U** matrix. The matrix on the left is called the **Q** matrix and is orthogonal with **QQ' = 1** and **Q'Q = 1** (Nash 1990, 26). (The Q matrix is not called the L matrix for the following reason. There is a technique called QL factorization in which the right hand matrix is lower triangular or "Left" triangular.) For the problem of **Ax = b**, we have **QRx = b**. We can set **Rx = y**. Since, then we have **Qy = b**, **Q'Q = 1** results in **y = Q'b**. Then, **x** is solved from **R** and **y** by back substitution.

Suppose **R** is given from Gaussian elimination, then direct solution for **Q** may be obtained from **R'Q' = A'** using any of the techniques discussed above. However, this technique

would be very inefficient. Typically, the **A** matrix is first converted to tridiagonal form by Householder transformations. Then, the factorization becomes very efficient (Press et al. 1988, 375-397). The detailed development and the subsequent follow-on to the SVD algorithm are, however, beyond the scope of this work.

Singular Value Decomposition(SVD)

The QR procedure will have problems if **A** is a singular matrix. The SVD is able to self diagnose a singular **A** matrix. Suppose the matrix **R** is decomposed into the product **WV**' where **W** is a diagonal matrix and **V** is an orthogonal matrix such that $\mathbf{V'V = VV' = 1}$. In that case, it turns out that certain columns of **Q** become arbitrary and the truncated form of **Q** may be represented as **U** where $\mathbf{U'U = 1}$ but **UU'** is no longer necessarily the unit matrix due to the truncation of **Q**. Then the resulting product $\mathbf{A = UWV'}$ is the singular value decomposition of **A** (Nash 1990, 26-28).

The inverse of **A** in SVD is given as $\mathbf{A^{-1} = VW^{-1}U'}$. And, if **A** is singular there will be one or more $W_{kk} = 0$. For the equation, $\mathbf{Ax = b}$, set $1/W_{kk} = 0$ if $W_{kk} = 0$ and the solution $x = \mathbf{VW^{-1}U'b}$ will give the solution vector **x** with the smallest length (Press *et al*. 1988, 61). In this way, solutions are obtained with SVD even though **A** is singular.

Newton's Method with SVD

We are now prepared to synthesize the concepts of Newton's method from Chapter 3, maximum likelihood estimation from Chapter 4, and singular value decomposition (SVD) from this chapter together to explain the Newton's Method algorithm to be used in later sections. The normal equations for linear regression are given as:

$$\mathbf{Ax = b} \tag{71}$$

The algebraic solution is given as:

$$\mathbf{x} = (\mathbf{A'A})^{-1}\mathbf{A'b} \tag{72}$$

According to equation (49), equation (72) is the best least squares minimization of the sum of squared errors (sse):

$$\boldsymbol{\delta'\delta} = (\mathbf{Ax-b})'(\mathbf{Ax-b}). \tag{73}$$

if the deviations, δ_i, are normally distributed. The standard error of prediction, s, is

given as:

$$s = (\boldsymbol{\delta}'\boldsymbol{\delta}/m)^{1/2} \tag{74}$$

And, m is the number of rows of $\boldsymbol{\delta}$. The standard deviation of coefficient $\mathbf{x}_i$ from linear regression theory is:

$$s_i = Ts[(\mathbf{A}'\mathbf{A})^{-1}]_{ii} \tag{75}$$

The standard error of the prediction from linear regression theory is:

$$s_{j,pred} = Ts(\mathbf{A}_j(\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}_j')^{1/2} \tag{76}$$

T is the student T value and is a function of the desired confidence level and the number of degrees of freedom (m-n) for the problem.

As mentioned earlier, the algebraic solution with a computer is not recommended due to round off error that can occur in most computers. Furthermore, if **A** is singular, then the derivation fails. However, if SVD is used instead to solve the problem, then the round off error is minimized and the problem is still solvable even if A is singular. If **A** is solved by SVD such that:

$$\mathbf{A} = \mathbf{U}\mathbf{W}\mathbf{V}', \tag{77}$$

then **U** is a row orthonormal eigenvector matrix, **W** is a diagonal eigenvalue matrix, and, **V** is an orthonormal eigenvector matrix. The expression for **x** then becomes:

$$\mathbf{x} = \mathbf{V}\mathbf{W}^{-1}\mathbf{U}'\mathbf{b}. \tag{78}$$

Since **W** is a diagonal matrix, its inverse is trivial, there is not an error problem, and the **x** vector is easily found. The calculation of **x** may be further simplified by the process of factor compression. In factor compression, the insignificant eigenvectors in **W** are dropped and the dimension of the inner matrix calculations is reduced.

The variance-covariance matrix is given as:

$$(\mathbf{A}'\mathbf{A}) = \mathbf{V}\mathbf{W}^2\mathbf{V}' \tag{79}$$

and its inverse is given as:

$$(\mathbf{A}'\mathbf{A})^{-1} = \mathbf{V}\mathbf{W}^{-2}\mathbf{V}' \tag{80}$$

and, this inverse is used to calculate the coefficient standard deviations in equation (75). The corresponding term for the standard error of prediction is:

$$\mathbf{A}(\mathbf{A}'\mathbf{A})^{-1}\mathbf{A}' = \mathbf{U}\mathbf{U}' \tag{81}$$

which gives a much simpler expression for the standard error of prediction in equation (76). The sse of the system simplifies to:

$\boldsymbol{\delta}'\boldsymbol{\delta} = \mathbf{b}'\mathbf{b} - \mathbf{b}'\mathbf{U}\mathbf{U}'\mathbf{b}$ (sse = sst – ssr). (82)

Therefore, SVD stabilizes the solution of the linear regression normal equations and eliminates problems that occur for a singular **A** matrix.

If we now make the substitutions

$\mathbf{A} = \mathbf{J}(\mathbf{X}_L;\mathbf{a})$ (83)

$\mathbf{x} = (\mathbf{X}_N - \mathbf{X}_L)$ (84)

$\mathbf{b} = -(\mathbf{F}(\mathbf{X}_L;\mathbf{a}) - \mathbf{F}_{obs})$ (85)

into equation (71), we obtain

$\mathbf{J}(\mathbf{X}_L;\mathbf{a})\ (\mathbf{X}_N - \mathbf{X}_L) = -\ (\mathbf{F}(\mathbf{X}_L;\mathbf{a}) - \mathbf{F}_{obs})$ (86)

Equation (86) is just the over-determined analog of the multivariate Newton's method of equation (18) and of the local Newton's method in equation (39).

If equation (86) is solved algebraically, we obtain:

$(\mathbf{X}_N - \mathbf{X}_L) = -(\mathbf{J}'\mathbf{J})^{-1}\mathbf{J}'(\mathbf{F}(\mathbf{X}_L;\mathbf{a}) - \mathbf{F}_{obs})$ (87)

which is equivalent to equation (1) with $R = (\mathbf{J}'\mathbf{J})^{-1}$ and with $\mathbf{J}'(\mathbf{F}(\mathbf{X}_L;\mathbf{a}) - \mathbf{F}_{obs})$ equivalent to the gradient of a certain cost function, e.g., sse. Furthermore, equation (87) is equivalent to equation (2) that is the multivariate local Newton's method.

Therefore, if we solve $\mathbf{J}(\mathbf{X}_L)$ by SVD:

$\mathbf{J}(\mathbf{X}_L) = \mathbf{U}\mathbf{W}\mathbf{V}'$ (88)

then Newton's method can be solved by SVD as shown in equations (77) to (82) with the substitutions in equations (83), (84), and (85) and the learning rate $R = \mathbf{V}\mathbf{W}^{-2}\mathbf{V}'$. See equation (1). Furthermore, if the residuals, $\boldsymbol{\delta}$, are normally distributed, then the standard deviations of the coefficients and the standard error of prediction can be determined from equation (75) and (76) as claimed in equations (4) and (5). If the residuals are not normally distributed, then the robust regression methods shown in Chapter 4 may be used. If the conditions of Chapter 2 are met, equation (87) can be iterated to obtain quadratic convergence and exponential speed-up over algorithms such as simplex EVOP. If the algorithm fails to converge in a given number of iterations, then new initial values are picked as illustrated in Chapter 2. Also, if the algorithm does not converge after a large number of iterations, then it is probable that the conditions for

convergence in Chapter 2 are not met like, for example, the Modified Mandlebrot Set Function in Chapter 2. Publicly available software found in LAPACK, LINPACK, IMSL, or NAG can be used to perform the SVD as well as other matrix operations (Berry *et al*. 1993) and (Press *et al*. 1988, 35).

Local Newton's Method by Maximum Likelihood Estimates

Now that we have shown how multivariate Newton's method is solved using techniques of linear algebra, we will conclude with the calculation of the probability of the maximum likelihood of Newton's Method for data mining of technical data. The matrix **a** is an m x np matrix constructed from m database records. There are a corresponding number of m observations, $\mathbf{F}_{obs}$, from the database – there may be more than one column of $\mathbf{F}_{obs}$ but we assume only one column. The model consists of a model function, $F(\mathbf{x}_0;\mathbf{a}_j)$, where the 1xn $\mathbf{x}_0$ vector is the initial guess of the parameters to be determined and $\mathbf{a}_j$ is row j of **a**. The mxn Jacobian, $\mathbf{J}(\mathbf{x}_0;\mathbf{a})$, is the matrix of derivatives of the model function: $[\mathbf{J}(\mathbf{x}_0;\mathbf{a})]_{ij} = \partial F(\mathbf{x}_0;\mathbf{a}_j)/\partial \mathbf{x}_i$. We define the following shorthand notation: $\mathbf{J}_S \equiv \mathbf{J}(\mathbf{x}_S;\mathbf{a})$, $\mathbf{F}_S \equiv \mathbf{F}(\mathbf{x}_S;\mathbf{a})$, where S = 0, 1, …, L(last), N(next).

In terms of the equations (44) to (46), we have

- the prior probability, $P(\mu) = P(F)$, the confidence in the model prior to having data
- the evidence, $P(D) = P(\mathbf{x}_0;\mathbf{a};\mathbf{F}_{obs})$, the confidence in the data mine
- the likelihood, $P(D|\mu) = P(\mathbf{x}_0;\mathbf{a};\mathbf{F}_{obs} \mid \mathbf{F}(\mathbf{x}_0;\mathbf{a}))$, the probability the data are correct given the model
- the updated belief probability, $P(\mathbf{F}(\mathbf{x}_0;\mathbf{a})| \mathbf{x}_0;\mathbf{a};\mathbf{F}_{obs})$, the probability the model is correct given the data – this is the probability we would like to maximize.

It is customary to assume that the prior and the evidence are constant and maximize the likelihood, $P(\mathbf{x}_0;\mathbf{a};\mathbf{F}_{obs} \mid \mathbf{F}(\mathbf{x}_0;\mathbf{a}))$. Now suppose that the residuals, $\boldsymbol{\delta}_j = \mathbf{F}_{obs,j} - \mathbf{F}_{L,j}$, are normally distributed such that $-\log(P) = \rho_j = \tfrac{1}{2}(\boldsymbol{\delta}_j/\sigma)^2$ – the case for non-normal residuals was described in Chapter 4. The likelihood is given as:

$$P(\mathbf{a};\mathbf{x}_0;\mathbf{F}_{obs} \mid \mathbf{F}_N) \propto \prod_{j=1}^{m} \exp\left[-\rho\left(F_{obs,j}, F(\mathbf{x}_N;\mathbf{a}_j)\right)\right]\Delta F \tag{89}$$

To maximize P, we minimize –log(P):

$$-\log(\mathrm{P}) \propto \sum_{j=1}^{m} \delta_j^2 = \boldsymbol{\delta}'\boldsymbol{\delta} \equiv \mathrm{sse} \tag{90}$$

Since $\boldsymbol{\delta} = (\mathbf{F}_{obs} - \mathbf{F})$, then $\boldsymbol{\delta}'\boldsymbol{\delta} = (\mathbf{F}'_{obs} - \mathbf{F}')(\mathbf{F}_{obs} - \mathbf{F})$. To minimize $\boldsymbol{\delta}'\boldsymbol{\delta}$, we differentiate equation (90) with respect to $\mathbf{x}'$. Only non-constant transpose terms survive and we obtain equation (91):

$$\partial(\boldsymbol{\delta}'\boldsymbol{\delta})/\partial\mathbf{x}' = (\partial/\partial\mathbf{x}')(-\mathbf{F}'\mathbf{F}_{obs} + \mathbf{F}'\mathbf{F}) \tag{91}$$

Now expand $\mathbf{F}'$ into a power series:

$$\mathbf{F}(\mathbf{x}_N;\mathbf{a})' \approx \mathbf{F}'_L + \Delta\mathbf{x}'\mathbf{J}'_L \tag{92}$$

Since, $\partial\mathbf{F}'_N/\partial\mathbf{x}' = \mathbf{J}'_N$, we obtain equation (93):

$$\partial(\boldsymbol{\delta}'\boldsymbol{\delta})/\partial\mathbf{x}' \approx -\mathbf{J}'_L\mathbf{F}_{obs} + \mathbf{J}'_L(\mathbf{F}_L + \mathbf{J}_L\Delta\mathbf{x}) \tag{93}$$

If$(\mathbf{J}'_L\mathbf{J}_L) \neq 0$, then we obtain:

$$-\mathbf{F}_{obs} + \mathbf{F}_L + \mathbf{J}_L\Delta\mathbf{x} = 0 \Rightarrow \mathbf{J}_L\Delta\mathbf{x} = -(\mathbf{F}_L - \mathbf{F}_{obs}) \tag{94}$$

Equation (94) is the iteration equation for the local Newton's method. The final answer is given as $\boldsymbol{\xi} \approx \mathbf{x}_0 + \Sigma\Delta\mathbf{x}$, where the sum is over all iterations. In terms of data mining, the Newton's method algorithm is:

Data Mine: **a** and $\mathbf{F}_{obs}$ + Prior Knowledge: $\mathbf{F}(\mathbf{x}_0;\mathbf{a}) = (\mathbf{J}) \Rightarrow$ Non-trivial Knowledge: $\mathbf{x} \approx \mathbf{x}_0 + \Sigma\Delta\mathbf{x}$

So, **J** represents an operator that converts prior knowledge and the data mine into non-trivial knowledge. Future predictions, $F(\boldsymbol{\xi};\mathbf{a}_j)$, are made without additional database queries.

Now, let us expand $\boldsymbol{\delta}'\boldsymbol{\delta}$ in a power series about $\mathbf{x} = \boldsymbol{\xi}$.

$$\boldsymbol{\delta}'\boldsymbol{\delta} = \boldsymbol{\delta}'\boldsymbol{\delta}|_{\mathbf{x}=\boldsymbol{\xi}} + \left|\frac{\partial \boldsymbol{\delta}'\boldsymbol{\delta}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\boldsymbol{\xi}} (\mathbf{x}-\boldsymbol{\xi}) + \frac{1}{2}(\mathbf{x}-\boldsymbol{\xi})' \left|\frac{\partial^2 \boldsymbol{\delta}'\boldsymbol{\delta}}{\partial \mathbf{x}\partial \mathbf{x}'}\right|_{\mathbf{x}=\boldsymbol{\xi}} (\mathbf{x}-\boldsymbol{\xi}) \tag{95}$$

But, since $\mathbf{x} = \boldsymbol{\xi}$ is a minimum, the first derivative term in equation (95) is zero. Since $\boldsymbol{\delta}'\boldsymbol{\delta}$ gives –log(P) where P is the likelihood, then the updated belief probability becomes:

$$\text{Updated Belief} \propto \exp(-\boldsymbol{\delta}'\boldsymbol{\delta}) \propto \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\xi})'\mathbf{J}'\mathbf{J}(\mathbf{x}-\boldsymbol{\xi})\right) \tag{96}$$

since $\partial^2(\boldsymbol{\delta}'\boldsymbol{\delta})/\partial\mathbf{x}\partial\mathbf{x}' = \mathbf{J}'\mathbf{J}$. Equation (96) establishes $\mathbf{J}'\mathbf{J}$ as the variance-covariance matrix for the system of equations represented in equation (94) and gives the updated belief probability as a multivariate normal distribution. The variance-covariance matrix, $\mathbf{J}'\mathbf{J}$, plays the role of the Hessian matrix for the local optimization. Since the diagonal elements of $\mathbf{J}'\mathbf{J}$ are always positive, then we are almost always guaranteed that we are searching for the minimum of $\boldsymbol{\delta}'\boldsymbol{\delta}$ when using the local Newton's method. However, it may still be wise to check $\mathbf{J}'\mathbf{J}$ for other second order maxima conditions.

CHAPTER 6

RESULTS AND DISCUSSION

Now that we have a statistically valid and exponentially fast Newton's Method algorithm, we will compare it to other data mining algorithms. The genetic algorithm(GA) was compared to the simplex EVOP algorithms and to the Newton's Method algorithm. Global optmization results are shown for the simplex EVOP algorithms in Figure 16 and for the GA and Global Newton's Method in Figure 17. Local optimization results are shown for the GA in Figure 18 and for the Local Newton's Method in Figure 19. A two dimensional Gaussian function was maximized to compare the algorithms for a global maximization problem. A local optimization problem that consisted of finding the roots, or parameters, of the transcendental Krieger-Dougherty equation discussed in Chapter 3 was used. For global optimizations, the Gaussian function described in Chapter 2, Figure 1, was used:

$F(x,y) = F_{max}\exp(-[(x-x_m)/s_x]^2)\exp(-[(y-y_m)/s_y]^2)$

where $F_{max} = 10$, $x_m = 30$, $y_m = 45$, $s_x = 30$, and $s_y = 50$. The initial conditions for each of the global algorithms were:

- Basic Simplex EVOP: (x,y) = (10, 10), (12, 12), and (10, 12).
- Variable Simplex EVOP: (x,y) = (10, 10), (60, 60), (10, 60).
- Genetic Algorithm: $(x_{max}, y_{max}) = (100, 100)$ and $(x_{min}, y_{min}) = (10, 10)$.
- Global Newton's Method: $(x_0, y_0) = (10, 10)$.

The global optimization objective was to find the maximum value of the function.

For local optimizations, the Krieger-Dougherty equation was used (real numbers only):

$\eta_r = (1-\varphi/\varphi_M)^{-[\eta]\varphi M}$

where $\varphi_M = 0.63$, $[\eta] = 2.5$ – the "unknowns", $\{\varphi_i\} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$, and $\{\eta_{ri}\} = \{$ 1.313, 1.825, 2.769, 4.889, 12.01, 120.9$\}$

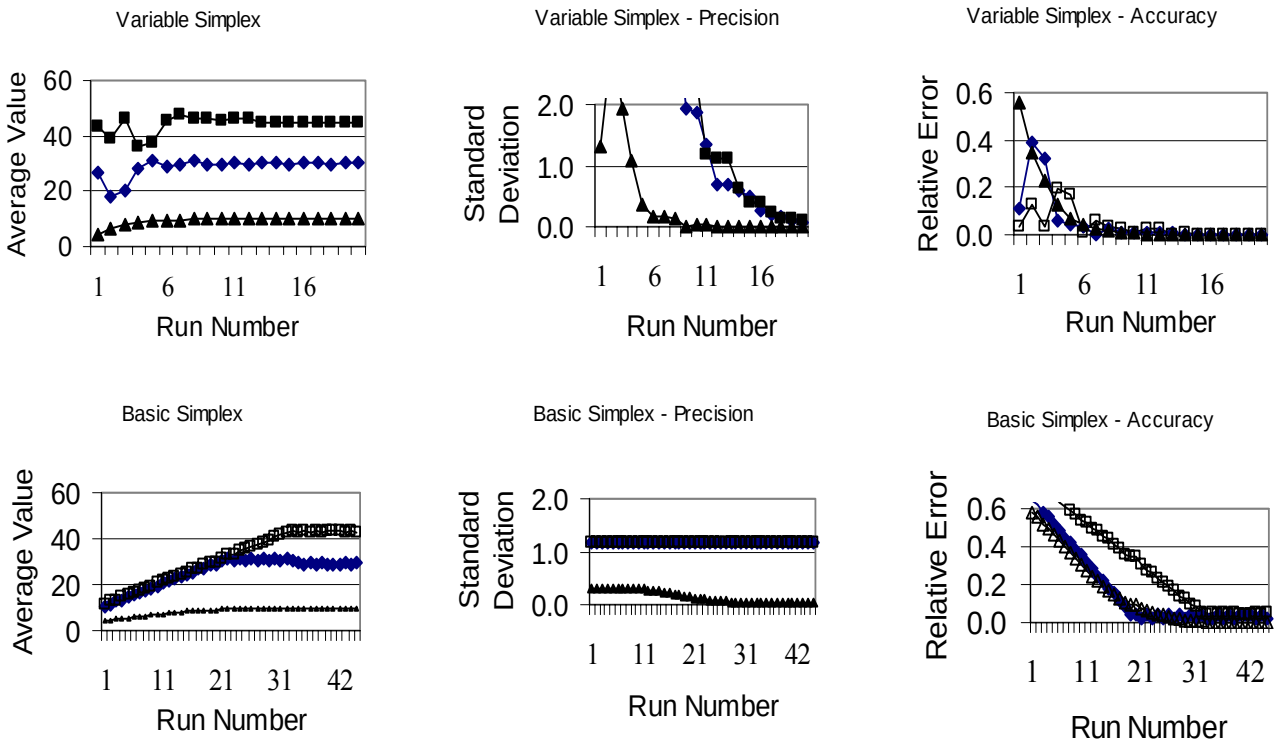

Figure 16. Convergence Behavior, Precision and Accuracy for the Basic and Variable Simplex Methods for Maximizing the Gaussian Function. Top Left to Right: Variable Simplex Values, Precision, and Accuracy. Bottom Left to Right: Basic Simplex Values, Precision, and Accuracy. Diamons: X Values. Squares: Y Values. Triangles: Function Values.

According to Figure 15, Newton's method has a small region of convergence for the Krieger-Dougherty equation. The initial conditions for each of the algorithms were:

- Basic Simplex EVOP: $([\eta], \varphi_M) = (2.1, 0.62)$, (2.2, 0.62), and (2.1, 0.615).
- Variable Simplex EVOP: $([\eta], \varphi_M) = (1.0, 0.61)$, (5.0, 0.61), and (5.0, 0.70).
- Genetic Algorithm: $([\eta]_{max}, \varphi_{M,max}) = (8.0, 0.80)$ and $([\eta]_{min}, \varphi_{M,min}) = (0.5, 0.61)$.
- Local Newton's Method: $([\eta]_{max}, \varphi_{M,max}) = (8.0, 0.80)$ and $([\eta]_{min}, \varphi_{M,min}) = (0.5, 0.61)$.

The local optimization objective was to minimize the function $\delta'\delta$ = sse as described in the previous chapter. The objective was transformed to a maximization problem for the simplex

EVOP methods and the genetic algorithm by using the logistic function: $1/[1+\exp(-\lambda(-\log<sse>))]$ with $\lambda = 0.2$. Although not required for the Local Newton's Method, the $\boldsymbol{\delta}'\boldsymbol{\delta}$ was computed to monitor progress and to decide when switching to other algorithms would be necessary. However, the backtracking strategy, the algorithm switching strategy, and the factor compression step were not called for with this set of experimental data.

For global optimizations, the vectors, $\mathbf{V}_j$, are read from the database. The fitness function may be calculated from the $\mathbf{V}_j$ or may be read from the database, also. However, if the algorithm calls for a value of $\mathbf{V}_j$ that has no corresponding fitness value in the database, then this method presents a problem, a data gap. Thus, a global strategy could be started and subsequently be halted because of data gaps in the database for either the $\mathbf{V}_N$ vector or the response, F_N. In Chapter 2, width, depth, and density of database records were discussed. It was shown that for a large number of attributes, the database density is very low. Obviously low database density could lead to data gaps and the scalability of global optimizations for data mining is in doubt. However, the data gap problem is not a major concern for local optimizations discussed later. The local optimization method we propose is overdetermined and missing records reduce the degrees of freedom of the solution but do not necessarily halt the algorithm. A support percentage equal to the count of $(\mathbf{a}_j,\mathbf{F}_j)$ tuples divided by the theoretical maximum number is proposed that is similar to the support percentage that was given in Chapter 2. To evaluate the algorithm itself, we assume that all vectors and fitness functions called for by the algorithm are available from the database.

The constant simplex EVOP given in Chapter 2 was implemented using the following algorithm:

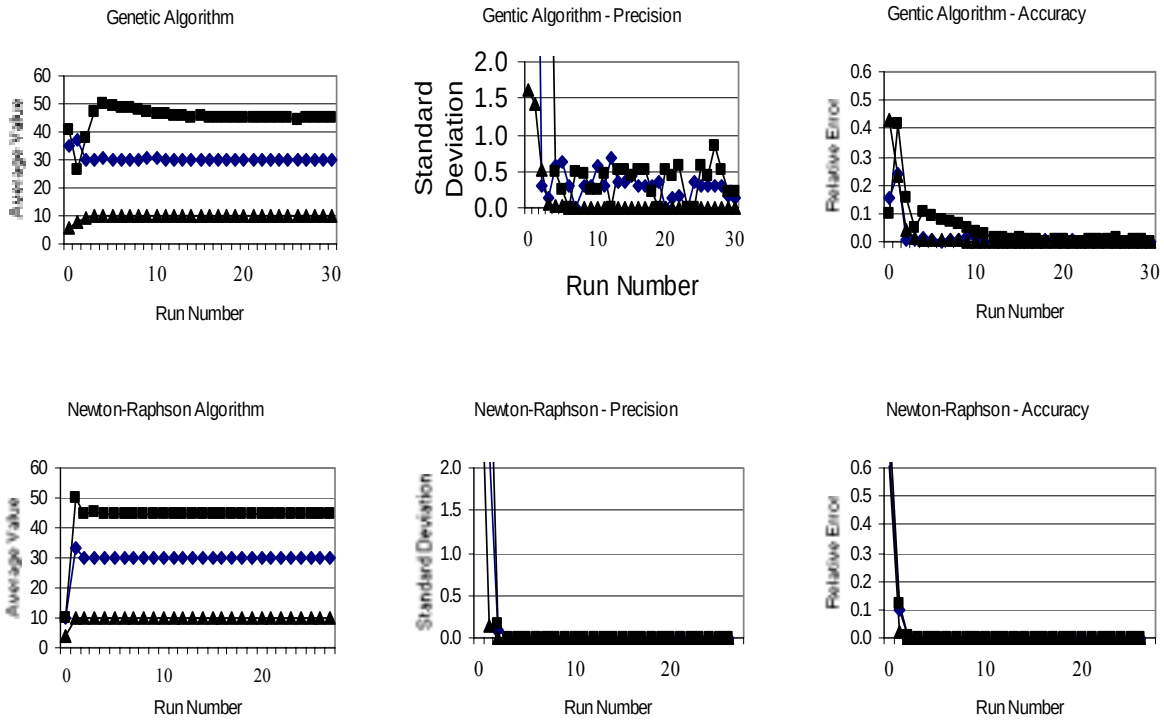

Figure 17. Convergence Behavior, Precision, and Accuracy for the Genetic Algorithm and Newton's Method for Maximizing the Gaussian Function. Top Left to Right: Genetic Algorithm Values, Precision, and Accuracy. Bottom Left to Right: Newton's Method Values, Precision, and Accuracy. Diamonds: X Values. Squares: Y Values. Triangles: Function Values.

Constant Simplex EVOP Algorithm

The constant simplex EVOP algorithm consisted of the following steps:

1. Define the initial simplex: k+1 vectors $\mathbf{V}_j$..
2. Find the **N**, **W**, and **B** vectors.
3. Calculate the centroid, **P,** excluding W**.**.
4. Calculate the reflection **R** from **P** and **W.**.
5. Replace **W** with **R**.

6. Find the **N**, **W**, and **B** vectors.
7. Let the **N** vector become **W**.
8. Check for completion.
9. Go back to 3 until done.

The Constant Simplex EVOP precision was obtained from the standard deviation of the current simplex. For example, the initial simplex for global optimization was (x,y) = (10, 10), (12, 12), and (10, 12). This initial simplex gives (x_{ave}, y_{ave}) = (10.7, 11.3), (x_{sd}, y_{sd}) = (1.2, 1.2), and (x_{acc}, y_{acc}) = (64.4%, 77.3%). The standard deviations of the simplex, (x_{sd}, y_{sd}), were always based on three coordinates. The accuracy measure, (x_{acc}, y_{acc}), is the percent difference between the average values, (x_{ave}, y_{ave}), and the final answer, (30, 45). The average, standard deviation, and accuracy of the response value were obtained in a similar manner. The Variable Simplex EVOP was performed as follows.

Variable Simplex EVOP

The variable simplex EVOP described in Chapter 2 was implemented using the following algorithm:

1. Define the initial simplex: k+1 vectors $\mathbf{V}_j$.
2. Find the **N**, **W**, and **B** vectors.
3. Calculate the centroid, **P**, excluding **W**.
4. Calculate the reflection R from P and W.
5. Select **B… NE, B…NR, B…NC_R, or B…NC_W** according to Figure 3.
6. Find the **N**, **W**, and **B** vectors.
7. Let the **N** vector become **W**.
8. Check for completion.
9. Go back to 3 until done.

The Genetic Algorithm was performed as follows.

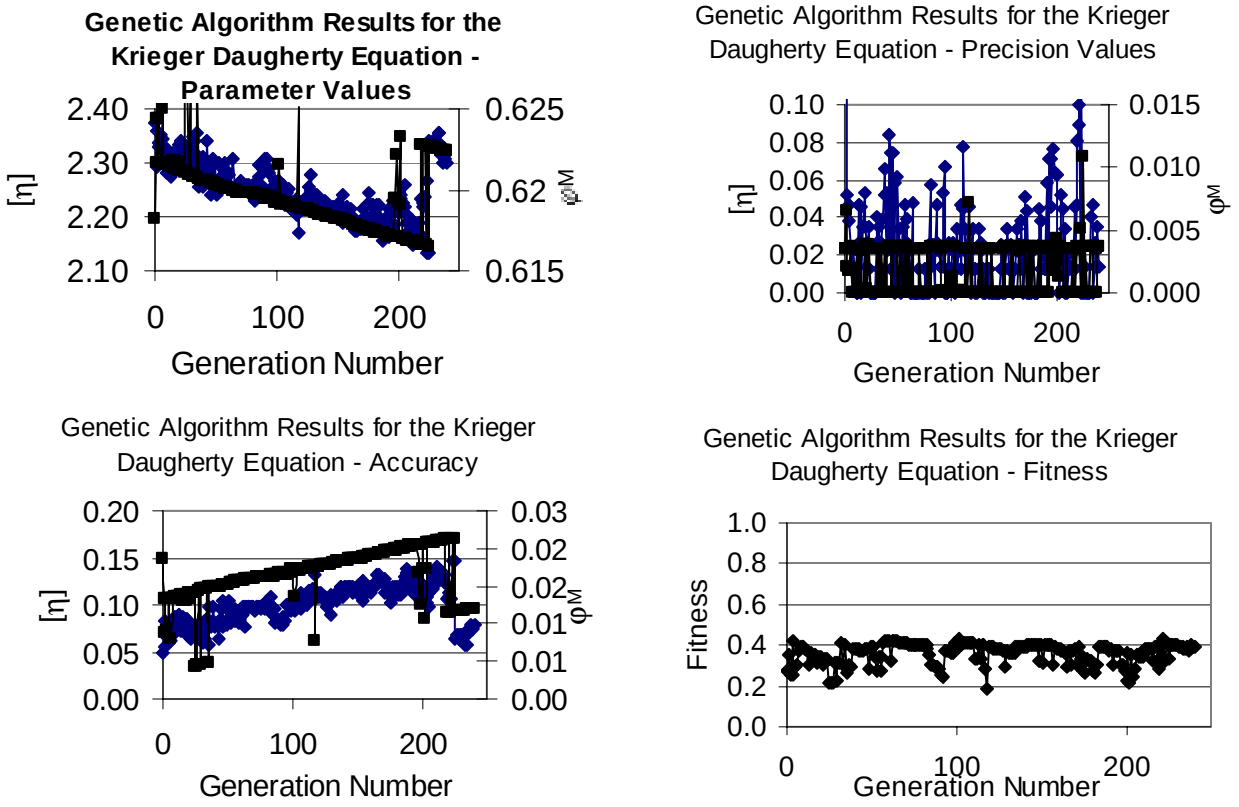

Figure 18. Genetic Algorithm Behavior for Finding Krieger-Dougherty Equation Parameters. Top Left: Parameter Values. Top Right: Precision. Bottom Left: Accuracy. Bottom Right: Fitness. Diamonds: Intrinsic Viscosity, [η]. Squares: Max Packing Fraction, φ_M.

Genetic Algorithm

A floating point GA was implemented according to the following algorithm (Michalewicz 1999):

1. Random Generation. Generate a population of n randomly selected vectors, $\mathbf{V}_j$.
2. Fitness. Calculate the fitness function, $F_j(\mathbf{V}_j)$, for each member of the population.
3. Sort the population according to F_j ($\mathbf{V}_j$).
4. Selection. Two individuals are selected as parent pairs based on their fitness and a probability function, $Ps_j = Ps_j(F_j(\mathbf{V}_j))$. And, the same individual may be selected for breeding more than once.
5. Crossover. Execute the parent pair breeding strategy.
6. Mutation. With probability, Pm, select a $\mathbf{V}_j$ and execute a major mutation strategy.
7. Minor mutation. With a probability of 0.9, execute a minor mutation strategy.
8. Continue steps 3, 4, 5, and 6 until n new offspring are created.
9. Replace the old population with the offspring population.
10. Go back to step 2.

The GA parameters for the global optimization were: population size = 10, geometric probability distribution for parent selection with Qbest = 0.10, probability of mutation = 0.05, probability of minor mutations = 0.90, arithmetic crossover, and uniform mutation.

For the global optimization, an example initial population is:

x	y	F
97.2	80.5	0.0
87.2	70.5	0.2
77.2	60.5	0.8
67.2	50.5	2.1
63.9	47.2	2.8
20.5	93.8	3.5
10.5	83.8	3.6

- 53.8 37.2 5.2
- 30.5 13.9 6.8
- 43.8 27.2 7.1

The resulting population after five generations was:

- x y F
- 29.6 38.3 9.8
- 30.8 51.0 9.9
- 30.8 50.0 9.9
- 30.8 50.0 9.9
- 29.6 50.0 9.9
- 30.2 50.0 9.9
- 31.1 49.5 9.9
- 29.6 49.0 9.9
- 30.2 49.0 9.9
- 30.2 49.0 9.9

The parameters for local optimization using the genetic algorithm were the same as for the global optimization except that the population size was increased to 20. For example, the first and 50th generation for the local optimization was:

- 0 Generations 50 Generations
- [η] φ_M Fitness [η] φ_M Fitness
- 7.9 0.76 0.02 2.3 0.61 0.15
- 7.2 0.74 0.03 2.3 0.64 0.16
- 7.0 0.74 0.03 2.2 0.61 0.17
- 6.2 0.72 0.04 2.2 0.61 0.17
- 6.1 0.71 0.04 2.3 0.63 0.17
- 5.3 0.70 0.05 2.2 0.63 0.18
- 5.2 0.69 0.05 2.3 0.63 0.19
- 5.1 0.69 0.06 2.2 0.63 0.19
- 4.4 0.67 0.07 2.2 0.63 0.19
- 4.2 0.67 0.08 2.2 0.63 0.19

- 3.5 0.65 0.10 2.3 0.63 0.20
- 3.4 0.65 0.11 2.3 0.63 0.20
- 3.2 0.64 0.12 2.3 0.63 0.21
- 0.5 0.77 0.13 2.3 0.62 0.22
- 0.7 0.77 0.13 2.3 0.63 0.23
- 1.4 0.79 0.13 2.3 0.63 0.24
- 1.5 0.79 0.13 2.3 0.62 0.24
- 2.5 0.63 0.20 2.4 0.63 0.27
- 2.4 0.62 0.23 2.3 0.62 0.36
- 2.2 0.62 0.28 2.3 0.62 0.39

Minor mutations were necessary in both the global and local genetic algorithm approaches due to the problem that the population would tend to all $\mathbf{V}_j$ being exactly equal with no minor mutations.

The global Newton's method was performed as follows.

Global Newton's Method

1. Input the initial guess, $\mathbf{X}_L$, of the parameter vector.
2. Calculate the gradient of $F(\mathbf{X}_L) = \nabla F(\mathbf{X}_L)$.
3. Calculate the Hessian matrix $\mathbf{H}(\mathbf{X}_L) = \nabla^2 F(\mathbf{X}_L)$.
4. Check diagonal elements of $\mathbf{H}$ for proper search direction and critical points.
5. Calculate $\mathbf{X}_N = \mathbf{X}_L - \mathbf{H}^{-1}\nabla F(\mathbf{X}_L)$.
6. Let $\mathbf{X}_L = \mathbf{X}_N$.
7. Repeat steps 2, 3, 4, 5, and 6 until done.

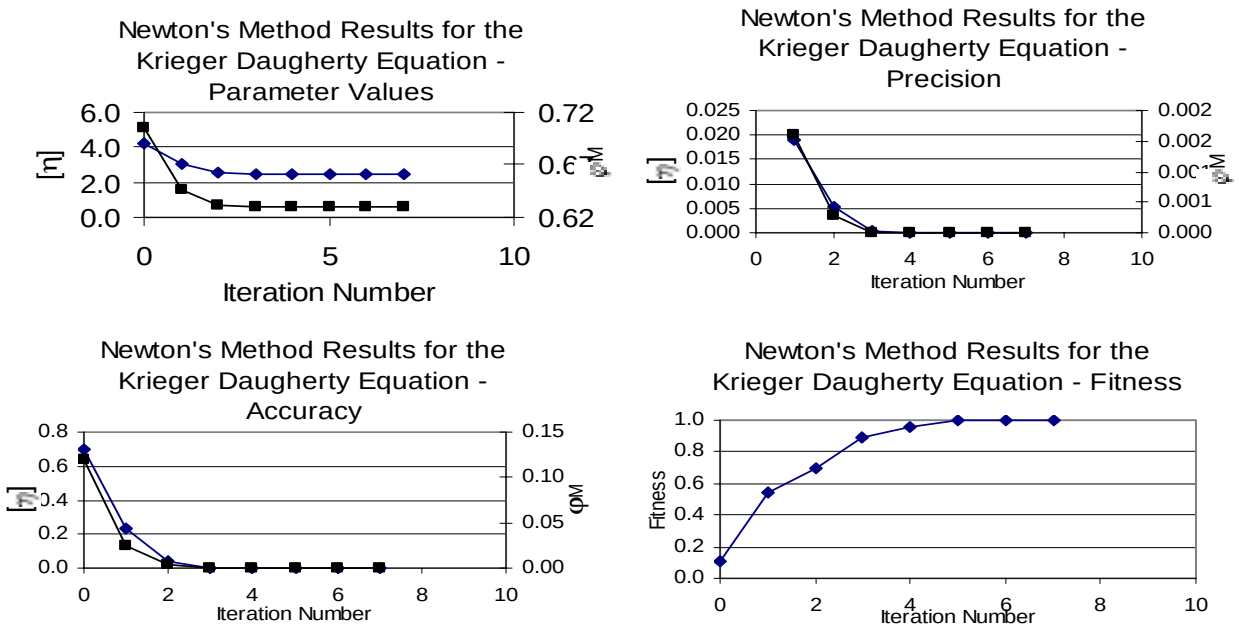

Figure 19. Newton's Method Behavior for Finding Krieger-Dougherty Equation Parameters Using Singular Value Decomposition. Top Left: Values from Equation (94), (88), (78). Top Right: Precision from Equation (75). Bottom Left: Accuracy. Bottom Right: $1/(1+\exp(\lambda\ln(\boldsymbol{\delta}'\boldsymbol{\delta})))$ Diamonds: $[\eta]$. Squares: φ_M.

Step 4 was necessary because the initial starting points for the global search test are outside of the critical points for the Gaussian function. Newton's method finds the extrema that could either be minima or maxima. The search direction is found from the Hessian matrix. Otherwise, the algorithm searches for minima from the starting points selected which are at $(x,y) = (-\infty,-\infty)$. The average values, standard deviations, and accuracies were calculated as described for the basic simplex EVOP except that the last three iterations of the global Newton's method were used.

The local Newton's method was performed as described in the preceding chapter.

Local Newton's Method

The local Newton's method was performed as follows using SVD and a backtracking strategy:

1. Input the initial guess, $\mathbf{X}_L$, of the n dimensional parameter vector.
2. Read the m values of p dimensional $\boldsymbol{\varphi}_i$ and the r dimensional $\mathbf{F}_{obs,j}$ from the file or database.
3. Evaluate the $F_j(\mathbf{X}_L)$.
4. Compute the Jacobian $\mathbf{J}(\mathbf{X}_L)$.
5. Solve the normal equations $\mathbf{J}(\mathbf{X}_L)\ (\mathbf{X}_N-\mathbf{X}_L) = -\ (\mathbf{F}(\mathbf{X}_L) - \mathbf{F}_{obs})$ for $\mathbf{X}_N$.
6. Execute backtracking strategy if $\mathbf{X}_N$ violates constraints.
7. Let $\mathbf{X}_L = \mathbf{X}_N$.
8. Repeat steps 3, 4, 5, 6, and 7 until done.

In the current case, we have p = r = 1. There was no need to calculate averages and standard deviations as done for the other algorithms. The Local Newton's Method proposed in the preceding chapters and implemented supplies the standard deviations of the parameter estimates automatically with the use of equation (75).

Global Optimization Function Results

All four algorithms found the maximum value of the Gaussian function. The results for the simplex EVOP algorithms are shown in Figure 16. The results for the genetic algorithm and the global Newton's method are shown in Figure 17. The variable simplex EVOP (VSE) converges to the correct response after about 7 generations (See Accuracy in Figure 16.). Acceptable accuracy for the VSE is obtained after about five generations. Acceptable precision is achieved after about 20 iterations for the VSE. The basic simplex method(BSM) reaches the desired accuracy after about 35 generations since each step is of fixed length. The precision of the x and y values is constant as expected. However, acceptable response precision is achieved after about 30 iterations. According to Chapter 2, if we require an accuracy of 0.01%, then the equivalent binary search strategy would take about Ceiling[lg(1000)] = 14 iterations. The

theoretical number of iterations required for Newton's method would be between 4 and 5 (assuming $[\eta] = 2.5$, $G = 0.5$, $e_0 = 0.5$) according to equation (27).

The GA (Figure 17) reaches an accurate result in about 6 generations. However, precision is not achieved through 30 generations. The Newton's method algorithm reached accurate and precise results after about 3 iterations. Thus, in terms of generations required (iterations), precision, and accuracy, the Global Newton's Method outperformed all algorithms studied. However, the Global Newton's Method is subject to the data gap problems of database completeness like other global algorithms. Furthermore, the Global Newton's Method requires a higher density of points in the database as well in order to calculate the higher order derivatives to construct the Hessian matrix. It may be possible to circumvent this problem of Hessian matrix updates by only computing the Hessian matrix once or every few generations – or by estimation. This update problem is not a serious problem for the Local Newton's Method since the higher order derivatives are calculated - either analytically, numerically, or by the secant method – from the user supplied function.

Local Optimization Function

To test the algorithms, the Krieger-Daugherty equation was solved for intrinsic viscosity, $[\eta]$, and maximum volume fraction, φ_M. The local Newton's method solves for these two parameters directly. The problem was globalized to use the genetic algorithm and simplex methods as described above. The results for the genetic algorithm are shown in Figure 18.

The accuracy of the GA is about 5% for $[\eta]$ and about 1% for φ_M after about 220 generations. However, the fitness function is only about 0.4. The GA with these parameters was unable to obtain accurate and precise results. The results for the local Newton's method algorithm are shown in Figure 19. The Newton's method algorithm obtained an accurate and precise result after about 3 iterations. And, the fitness function achieved its maximum value of 1.0 after about 5 iterations.

The poor performance of the genetic algorithm was surprising. Perhaps the poor performance was due to the pathological qualities of the sse function of the Krieger-Dougherty equation. In fact, this function has several local maxima and a very narrow peak width at the optimum. The sse function is shown in Figure 20. Figure 20 does indicate that there are several

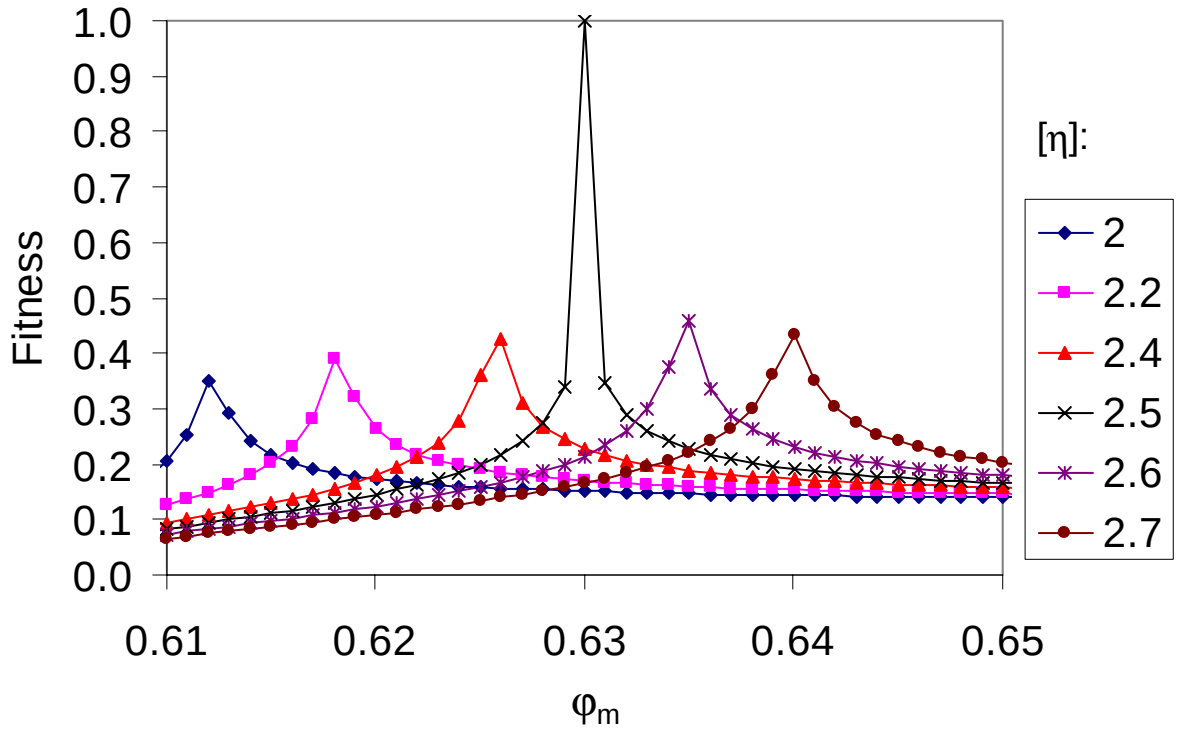

Figure 20. Global Fitness Function for Finding the Parameters of the Krieger-Dougherty Equation for $\{\varphi_j\}$ and $\{\eta_{r,j}\}$ Given in the Text. Fitness is $1/[1+\exp\{-\lambda(-\log<sse>)\}]$. Diamonds $[\eta] = 2.0$; Squares $[\eta] = 2.2$; Triangles $[\eta] = 2.4$; X's $[\eta] = 2.5$; Circle/X's $[\eta] = 2.6$; Triangles $[\eta] = 2.4$; Circles $[\eta] = 2.7$.

maxima for the fitness function. Thus, it may not be surprising that the GA found a sub-optimum maximum. The same problem occurred with the simplex EVOP algorithms. The local Newton's method, however, was not fooled by the numerous maxima in the fitness function for the local optimization problem. Surprisingly, addition of a small amount of random noise to the data resulted in improved parameter results for the GA. However, the final fitness function in that case was still quite low compared to the Local Newton's Method algorithm. These results are summarized for all four algorithms - with and without noise addition - in Table 1. In Table 1, the starting points, Generation 0, are shown for comparison purposes. In Table 1, the Newton's method achieves machine precision in about 5 iterations and there is no point in performing further iterations.

Table 1. Comparison of Various Data Mining Methods for Local Optimization of the Krieger-Dougherty Equation

Algorithm	Gener-ations	[η] Value	[η] S.D.	[η] %Accu-racy	ϕ_m Value	ϕ_m S.D.	ϕ_m %Accu-racy	Fit. F.
Basic Simplex No Noise	0	2.067	0.058	17.3%	0.612	0.003	2.9%	0.24
Basic Simplex No Noise	650	2.133	0.058	14.7%	0.618	0.003	1.9%	0.28
Basic Simplex with Noise	0	2.067	0.058	17.3%	0.612	0.003	2.9%	0.28
Basic Simplex with Noise	650	2.133	0.058	14.7%	0.618	0.003	1.9%	0.28
Variable Simplex No Noise	0	3.667	2.309	46.7%	0.640	0.052	1.6%	0.07
Variable Simplex No Noise	150	2.250	0.750	10.0%	0.675	0.003	7.1%	0.15
Variable Simplex Noise	0	3.667	2.309	46.7%	0.640	0.052	1.6%	0.07
Variable Simplex Noise	150	2.250	0.750	10.0%	0.675	0.003	7.1%	0.15
GA No Noise	0	2.376	0.136	5.0%	0.618	0.003	1.9%	0.28
GA No Noise	240	2.301	0.013	8.0%	0.622	0.004	1.2%	0.39
GA with Noise	0	2.230	0.369	10.8%	0.690	0.006	9.6%	0.14
GA with Noise	350	2.517	0.029	0.7%	0.632	0.000	0.3%	0.38
NM No Noise	0	4.250	5.303	70.0%	0.705	0.134	11.9%	0.11
NM No Noise	5	2.500	0.000	0.0%	0.630	0.000	0.0%	1.00
NM with Noise	0	4.250	5.303	70.0%	0.705	0.134	11.9%	0.11
NM with Noise	4	2.513	0.032	0.5%	0.631	0.001	0.1%	0.70

CHAPTER 7

COMPARISON OF ALGORITHMS

We now compare the stochastic GA and the deterministic Newton's method algorithms. Criteria for comparison are accuracy, precision, speed (cpu cycles), storage requirements (main memory requirements and disk access requirements), and complexity (degree of difficulty). The space, cpu, and disk accesses for the GA are shown in Table 2. In Table 2, m is the number of database records (rows of the data matrix and the response matrix), n is the number of parameters to be determined, p is the number of database attributes (columns of the data matrix), r is the number of response functions (number of columns of the response matrix), and pop is the number of individuals in the population. Table 2 step 2 assumes that sorting is done with an efficient algorithm such as merge-sort.

Table 2. Analysis of Time and Space Requirements for the Genetic Algorithm.				
Step	**Comments**	**Space**	**CPU**	**Disk**
1. Random Generation. Generate a population of pop randomly selected vectors, Vi.	Read the Data matrix(mxp) and the response matrix(mxr). Set the max and min values for Vi (2n), generating requires pop vectors (pop*n).	mp+mr+2n+ pop*n	mp+mr+ 2n+ pop*n	mp+mr
2a. Fitness. Calculate the fitness function, Fi(Vi), for each member of the population.	Fitness function is usually the sum of squared errors.		= pop*mnp	
2b. Sort the population according to Fi(Vi).	Assume efficient sorting algorithm		pop* log(pop)	
3. Selection. Two individuals are selected as parent pairs based on their fitness and a probability function, Psi = Psi(Fi(Vi)). And, the same individual may be selected for breeding more than once.	Select two members from the population at random.		2	
4. Crossover. Execute the parent pair breeding strategy.	Changes the values of the parents.		pop*2n	
5. Mutation. With probability, Pm, select a Vi and execute a major mutation strategy.			pop*n	
6. Minor mutation. With a probability of 0.9, execute a minor mutation strategy.			pop*n	
7. Continue steps 3, 4, 5, and 6 until np new offspring are created.				
8. Go back to step 2.				
Totals		mp+mr+2n+ pop*n	mp+mr+ 2n+pop*[5n+mnp+ ln(pop)]	m(p+r)

Table 3 shows the time and space requirements for the local Newton's method. In Table 3, it was assumed that the Jacobian matrix is estimated numerically (truncated Newton's method).

Table 3. Analysis of Time and Space Requirements for the Local Newton's Method Algorithm.				
Step	**Comments**	**Space**	**CPU-Time**	**Disk Reads**
1. Input the initial guess, $\mathbf{X}_L$, of the n dimensional parameter vector.	Requires Xmax, Xmin, XL, XN, DelX	5n	5n	
2. Read the m values of p dimensonal φ_i and the r dimensional $F_{obs,i}$ from the file or database.	Reads the data matrix and the response matrix from the database.	mp+ mr	+mp+mr	mp+mr
3. Evaluate the Fi($\mathbf{X}_L$).	Depends on the cost of computing the function.	+mpr	+mpr	
4. Compute the Jacobian J($\mathbf{X}_L$).	Depends on if the derivative is supplied or calculated numerically.	2mr	+rmp +dm	
5. Solve the normal equations $\mathbf{J}(\mathbf{X}_L)\ (\mathbf{X}_N\text{-}\mathbf{X}_L) = -\ (\mathbf{F}(\mathbf{X}_L) - \mathbf{F}_{obs})$ for $\mathbf{X}_N$.	Requires updated response matrix and linear regression.		+mrp+ m^2n + nmr + n^3 +n	
6. Execute backtracking strategy if $\mathbf{X}_N$ violates constraints.	Could be performed multiple times of XN is out of bounds.		n	
7. Let $\mathbf{X}_L = \mathbf{X}_N$.			n	
8. Repeat steps 3, 4, 5, 6, and 7 until done.				
Totals per iteration.		5n+mn+3 mr	$=n^3+n(m^2$ +2m+mr +8) + 2mr(1+p)	= m(p+r)

For very large databases, disk access is the slow step for both algorithms since mp disk accesses take about 1000 times longer than mp cpu cycles. For small dimensionality (small n), Newton's method is faster for small data sets. For high dimensionality and large data sets, the GA is faster if convergence is obtained after a few generations. The main memory requirements for both algorithms are about the same. The calculated speeds of the algorithms are illustrated in Figures 21 and 22 where we have assumed that the required population size for the GA is 20n and have used the formulas from Tables 3 and 4. The Newton's method requires fewer CPU

steps for small data matrices. Table 4 summarizes the strengths and weakness for the two algorithms. For large data matrices, the GA requires fewer CPU steps and would become faster if the GA converges.

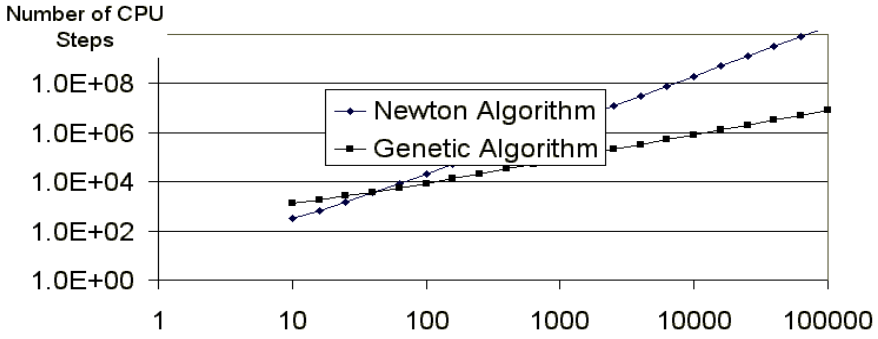

Figure 21. CPU Steps per Iteration for the Newton and Genetic Algorithms – 2D Parameter Vector.

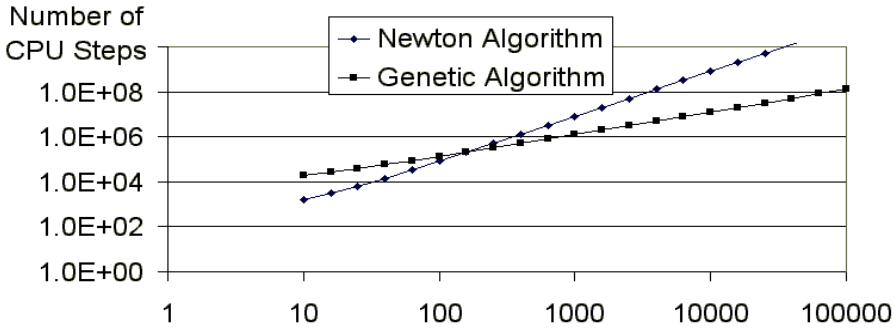

Figure 22. CPU Steps per Iteration for the Newton and Genetic Algorithms – 8D Parameter Vector.

Table 4. Comparison of the Genetic Algorithm and Local Newton's Method.

Method	Genetic Algorithm		Newton' Method	
	Global	Local	Global	Local
Accuracy	+	-	++	++
Precision	-	-	++	++
Convergence	+	-	++	++
Speed	++	++	+	+
Main Memory	-	-	-	-
Disk Accesses	-	-	-	-
Simplicity	++	+	+	-

Newton's method is considered to have the best accuracy, precision, and convergence behavior for the test functions used. The GA has the best speed per generation. However, the exponentialconvergence rate of the Newton's method could be used to counter this advantage of the GA. On the other hand, a finely tuned GA could still achieve a speed advantage over Newton's method. Both algorithms required excessive main memory and disk accesses. The GA is considered to be the least complex. Several parameters need to be tuned such as population size, Ps, Pm, crossover method, and mutation method. However, these GA parameters are intuitive. Note that the EVOP techniques are a special case of the GA for comparison purposes. The EVOP techniques always have an initial population size of n+1. The selection rules given in Chapter 2 result in only one child per generation based on n parents.

In terms of simplicity, Newton's method is rated below GA. The Newton's method algorithm is more complex since the convergence criteria in Chapter 3 must be understood – this requires an advanced knowledge of calculus. Also, application of the local Newton's method is

more complex due to the domain knowledge required to set up the response functions as outlined in the Knowledge Discovery process given in Chapter 2.

Main memory storage and disk access requirements are major weaknesses of both algorithms. However, disk access requirements are a major weakness of any data mining algorithm.

CHAPTER 8

CONCLUSION

Data mining is the extraction of non-trivial knowledge from databases using algorithms from computer science and other disciplines. Current data mining procedures have been successful with business applications such as market basket analysis. However, as data mining of technical data becomes important in such technical areas as medicine and engineering, the potential costs of errors will require the data miner to consider other algorithms in addition to the commonly used algorithms such as Genetic Algorithms and Neural Networks. Genetic Algorithms(GA) and Neural Network models(NN) provide for highly complex models but no capabilities to test statistical significance were found in the literature. And, without statistical tests, the reliability of GAs and NNs is in question. It has been shown that a local Newton's method (LNM) derived from global Newton's Method can be used as a data mining algorithm that provides tests of statistical significance of the parameter estimates and of the model predictions. It has been further shown how Newton's method may be stabilized by a combination of techniques: singular value decomposition, factor compression, backtracking strategy, switch to a global search strategy if required, and checks for second order minimization conditions.

Chapter 2 outlined the key features for a data mining algorithm from the literature: useability, accuracy, scalability, and compatibility. For LNM, useability and compatibility have been demonstrated in terms of database tuples ($\mathbf{a}_j$, $\mathbf{F}_j$), the data mine, and a multivariate function, $\mathbf{F}(\mathbf{x}_0)$, the prior knowledge. Non-trivial knowledge, $\mathbf{x}=\boldsymbol{\xi}$, is obtained by the Jacobian, $\mathbf{J}$, operating on the data mine. Then, new function values may be determined without additional database queries - a key requirement for a data mining algorithm according to Comaford. The accuracy and statistical significance of LNM results were shown to be a part of the algorithm's output. These features were not found for either the NN or for the GA. However, NN could actually be considered as a function, $F(\mathbf{x};\mathbf{a})$, rather than an algorithm where the parameters to be

determined are the W matrices given in Chapter 2. And, GAs could be used in conjunction with the LNM technique. Furthermore, accuracy was shown to improve quadratically for LNM. Scalability of LNM was superior to global methods since the local method is not as sensitive to the data gap problem. Also, LNM was found to scale-up better due to its exponential speed-up compared to the other algorithms considered. And, the use of singular value decomposition makes LNM more scalable due to the ability to use factor compression. Factor compression eliminates the problem of a singular Jacobian and reduces the computation steps required for a problem with a large dimensional x vector. The major drawback of the NM algorithm is its complexity. Specialized knowledge is required to understand how to set up the functions for local optimization and to apply both the global and local NM algorithms successfully.

REFERENCE LIST

Addison, Paul S. 1997. Fractals and Chaos. Philadelphia: Institute of Physics Publishing.

Adriaans, Peter and Zantinge, Dolf. 1996. Data Mining. New York: Addison-Wesley.

Agrawal, R., Shim, K. 1996. Developing tightly-coupled applications on IBM DB2/CS relational database system: methodology and experience. In Proceedings of the Second International Conference on Knowledge Discovery in Databases and Data Mining Held in Portland, Oregon August 1996, 287-290.

Baase Sara and van Gelder, Allen. 2000. Computer Algorithms: Introduction to Design and Analysis. 3rd ed. New York: Addison-Wesley.

Baldi, P. 1998. Bioinformatics. MIT Press: Cambridge.

Bennett K.P and Campbel, Colin. 2000. Support Vector Machines: Hype or Halleluja. SIGKD Explorations 2 , no. 2 (Dec): 1-13.

Berry, M., Do, T., O'Brien, G., Krishna, V., and Varadhan, S. 1993. SVDPACKC (version 1.0) User's Guide. Technical Report CS-93-194. University of Tennessee Department of Computer Science.

Berzal, Fernando, Cubero, Juan-Carlos, Marin Nicolas, Serrano, Jose-Maria. 2001. TBAR: An Efficient Method for Association Rule Mining in Relational Databases. Knowledge Engineering 37: 47-64.

Bjorck, Ake. 1996. Numerical Methods for Least Squares Problems. Pennsylvania: SIAM.

Comaford, Christine. 1997. Unearthing data mining methods, myths. PC Week 14, no. 1 (January 6): 65.

Conte, S.D. and de Boor, Carl. 1980. Elementary Numerical Analysis, 3rd ed. New York: McGraw-Hill Book Company.

Deming, Stanley N. and Morgan, Stephen L. 1987. Fundamentals of Experimental Design. ACS Short Courses. Washington: American Chemical Society.

Dunham, William. 1994. The Mathematical Universe. New York: John Wiley & Sons, Inc.

Elmasri, Ramez and Shamkant, B. Navathe. 2000. Fundamentals of Database Systems, 3rd ed. New York: Addison-Wesley.

Elert, Glenn, ed. 2000. The Physics Factbook. http://www.hypertextbook.com/facts/2000.

Gleick, James. 1987. Chaos. New York: Penguin Books.

Goebel, Michael and Gruenwald, Le. 1999. A Survey of Data Mining and Knowledge Discovery Tools. SIGKDD Explorations, ACM SIDKDD 1, no. 1 (June): 20-33.

Golub, G. H. and Van Loan, C. F. 1983. Matrix Computations. Baltimore: Johns Hopkins University Press.

Goodarzi, R., Kohavi, R., Harmon, R., and Senku, A. 1998. Loan Prepayment Modeling. http://citeseer.nj.nec.com/148586: American Association for Artificial Intelligence.

Goodwin, J. W. and Huges, R. W. 2000. Rheology for Chemists. Cambridge, U.K.: Royal Society of Chemists. 84.

Gotoh, Y. and Renals, S. 1997. Document Space Models Using Latent Semantic Analysis. In Eurospeech.

Harris, J. W. and Stocker, Horst. 1998. Handbook of Mathematics and Computational Science. New York: Springer-Verlag.

Hodgson, R. J. W. 2001. Genetic Algorithm Approach to the Determination of Particle Size Distributions from Static Light-Scattering Data. Journal of Colloid and Interface Science 240: 412-418.

Kahaner, D., Moler, C., and Nash, S. G. 1989. Numerical Analysis and Software. Englewood Cliffs, NJ: Prentice Hall.

Kaskalis T. and Margaritis K.G. 1996.
ANN Prototyping Using the Ptolemy Environment. Workshop on Neural Networks: From Biology to Hardware Implementations poster presentation.
Systolic Array Prototyping Using the Ptolemy Environment. Proc. Int. Conf. on Electronics, Circuits and Systems 2: 663-666.

Lesk, Michael. 1997. How Much Information is There in the World? http://www.lesk.com/mlesk/ksg97/ksg.html.

Levent, Kirilmaz. 2001. Data Mining Exercises for Controlled Release Dosage Forms. http://www.pharmaportal.com/articles/pte/Levent.pdf : Pharmaceutical Technology Europe.

Michalewicz, Zbigniew. 1999. Genetic Algorithms + Data Structures = Evolution Programs, 3rd ed. Berlin: Springer.

Mitchell, Melanie. 1996. An Introduction to Genetic Algorithms. Cambridge: The MIT Press. (http://emedia.netlibrary.com/reader/reader.asp?product_id=1337)

Nelder, J. A., and Mead, R. 1965. A simplex method for function minimization. Computer Journal 7: 308-313.

Nash, J. C. 1990. Compact Numerical Methods for Computers, 2nd ed. Bristol: Adam Hilger.

O'Leary, Dianne P. 2000. Symbiosis between Linear Algebra and Optimization. Journal of Computational and Applied Mathematics 123: 447-465.

Park, J. S., Chen, M. S., and Yu, P. S. 1995. An Effective Hash-based Algorithm for Mining Association Rules. In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data Held in San Jose, CA May 22-25, 1995, 175-186.

Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William T. 1988. Numerical Recipes in C: The Art of Scientific Computing. New York: Cambridge University Press.

Sarawagi, S., Thomas, S., and Agrawal, R. 1998. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. In SIGMOD 1998 Proceedgings of the ACM SIGMOD International Conference on Management of Data, Held in Seattle,Washington (June 2-4): 343-354.

Smith, K. A and Gupta, J. N. D. 2000. Neural Networks in Business: Techniques and Applications for the Operations Researcher. Computers & Operations Research 27: 1023-1044.

Spendley, W., Hext, G.R., and Himsworth, F. R. 1962. Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation. Technometrics 4, no. 4: 441-461.

Stewart, G. W. 1973. Introduction to Matrix Computations. New York: Academic Press.

Strang, G. 1976. Linear Algebra and Its Applications. New York: Academic Press.

Tendrick, P. and Matloff, N. 1994. A Modified Random Perturbation Method for Database Security. ACM Transactions on Database Systems 19, no. 1 (March): 47-63.

Walpole, Ronald E and Myers, Raymond H. 1978. Probability and Statistics for Engineers and Scientists, 2nd ed. New York: MacMillan Publishing Co., Inc.

Walters, Frederick H., Parker, Lloyd R., Morgan, Stephen L, and Deming, Stanley N. 1991. Sequential Simplex Optimization. Boca Raton, FL: CRC Press, Inc.

Ypma, Tjaling J. 1995. Historical Development of the Newton-Raphson Method. SIAM Review 37, no. 4 (December): 531-551.

VITA

JAMES D. CLOYD

Personal Data:	Marital Status: Married. Wife, Cherrie (BS ETSU 1989); Daughter, Melissa Cherie (BS ETSU 1997); Son, Daniel James (ETSU Class of 2003).
Education:	East Tennessee State University, Johnson City, Tennessee. Chemistry, B.S, Mathematics, B.S., 1972 (Magna Cum Laude) University of Illinois. Physical Chemistry, M.S., 1974
Professional Experience:	U. S. Navy. Nuclear Submarine Officer. Highest rank Lieutenant (O3). Qualified for Supervision of Maintenance and Operation of Naval Nuclear Reactors after completion of Naval Nuclear Power School at Idaho Falls, Idaho. Six deterrent patrols on the USS Casimir Pulaski SSBN 633. Qualified Submarines, Engineering Officer of the Watch. Staff rheologist for a major chemical company. Lab manager for Polymer Rheology and Polymer Science Lab. Colloidal and Physical Chemistry of Coatings. Currently leading a team in High Throughput Experimentation and Simulation project.
Selected Publications and Speeches:	Cloyd, J. D., Carico, K. C., and Collins, M. J. 2001. Rheological Study of Associative Thickener and Latex Particle Interactions. 28th Annual International Waterborne Symposium. New Orleans, LA. Cloyd, J. D., Carico, K., and Ewing, W. E. 1999. Medium Effects on the Enol/Keto Equilibrium of Ethyl Acetoacetate. SERMACS 99, Knoxville, Tenn. Cloyd, J. D. and Booton, J. D. 1996. Viscoelastic Behavior of Paints and Thickeners. Fourth North American Research Conference on Organic Coatings Science and Technology. Hilton Head, S.C. Seo, K.S. and Cloyd, J.D, Kinetics of Hydrolysis and Thermal Degradation of Polyester Melts. 1991. Journal of Applied Polymer Science. 42. 845-850. Cloyd, J. D., Seo, K. S., and Snow, B. D. 1991. Prediction of Extrudate Length from Complex Viscosity. 63rd Meeting of the Society of Rheology. Rochester, N. Y.
Honors and Awards:	ETSU Departmental award in Chemistry. ETSU Departmental award in Mathematics, Phi Kappa Phi, Kappa Mu Epsilon.