University of Nebraska at Omaha
DigitalCommons@UNO

Student Work

5-1-2005

Solving nonlinear optimization problems that have a nondifferentiable objective function by using a pseudo gradient search.

Marie Louise Spilde

Follow this and additional works at: https://digitalcommons.unomaha.edu/studentwork

Please take our feedback survey at: https://unomaha.az1.qualtrics.com/jfe/form/SV_8cchtFmpDyGfBLE

Recommended Citation

Spilde, Marie Louise, "Solving nonlinear optimization problems that have a nondifferentiable objective function by using a pseudo gradient search." (2005). *Student Work*. 3562.
https://digitalcommons.unomaha.edu/studentwork/3562

This Thesis is brought to you for free and open access by DigitalCommons@UNO. It has been accepted for inclusion in Student Work by an authorized administrator of DigitalCommons@UNO. For more information, please contact unodigitalcommons@unomaha.edu.

SOLVING NONLINEAR OPTIMIZATION PROBLEMS THAT HAVE A NONDIFFERENTIABLE OBJECTIVE FUNCTION BY USING A PSEUDO GRADIENT SEARCH

A Thesis

Presented to the

Department of Mathematics

and the

Faculty of the Graduate College of the

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

Master of Arts

at the University of Nebraska at Omaha

by

Marie Louise Spilde

May 2005

UMI Number: EP74760

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI
Dissertation Publishing

UMI EP74760

Published by ProQuest LLC (2015). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code

ProQuest

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

THESIS ACCEPTANCE

Acceptance for the faculty of the Graduate College,
University of Nebraska, in partial fulfillment of the
requirements for the degree Master of Arts,
University of Nebraska at Omaha.

Committee

Dr. Steven From Steve From

Dr. Zhenyuan Wang

Dr. Haifeng Guo

Chairperson

Date 4/18/05

SOLVING NONLINEAR OPTIMIZATION PROBLEMS THAT HAVE A NONDIFFERENTIABLE OBJECTIVE FUNCTION BY USING A PSEUDO GRADIENT SEARCH

Marie Louise Spilde, MA

University of Nebraska, 2005

Advisor: Dr. Steven From

The gradient search fails in an optimization problem where the objective function is not differentiable--such as nonlinear multiregressions based on generalized Choquet integrals. In cases such as this, we may replace the gradient search with a pseudo gradient search to determine the optimal search direction. The pseudo gradient can be obtained algorithmically from a data set containing the objective attribute and relevant arguments of the objective function. The algorithm for the pseudo gradient search is based on a neural network model which uses statistical techniques such as root mean square error to determine the optimal search direction and the optimal step length. Similar to the gradient search, the pseudo gradient search has a fast convergence rate, but a disadvantage is how easily it falls in a local extrema. Hence, choosing a suitable initial point for the pseudo gradient search is rather important. A genetic algorithm is used as an initialization method because it has the advantage of being a global search, but it is only allowed to run for a limited number of iterations due to its slow

convergence rate. The pseudo gradient search may be widely applied in nonlinear multiregression, classification, and decision making.

ACKNOWLEDGEMENTS

For me, the process of getting a Master of Arts degree has been akin to mountain climbing. I stood at the base of "Mount UNO" two and a half years ago with only two thoughts in my mind. My first thought was that a Bachelor of Science degree would not guarantee me financial stability. For me, the best bang for my buck would come in the form of a Master's degree, and that was my primary reason in continuing my education. My second thought was that if I ever quit my journey to the top of the mountain (where my diploma awaits), the fall to the earth below would be accentuated by the sound of a loud *THUD* of embarrassment.

Shortly after I began my ascent, a thick fog rolled in. I could no longer see the top of the mountain because my day-to-day life had become overwhelming. The only thing I could do was keep climbing one step at a time and work on blind faith that the top of the mountain was still there.

It was.

It is my pleasure to thank the many people who have made this thesis possible. Dr. Zhenyuan Wang and Dr. Haifeng Guo for their direction and support on this work and Dr. Steve From for being a wonderful advisor with a door that is always open. I am grateful to the teaching assistants in room 243A and the roommates of 2135 S. 46th Avenue for their moral support. I also wish to thank my parents, Gerald and Sharon Spilde for rescuing me during a variety of mini-crisis such as when the battery in the van died, as well as Kristopher Setnes and the entire Setnes family for the laughs and much needed breaks from school.

INTRODUCTION

At this point in time, no algorithm exists that will solve every nonlinear optimization problem. One approach to solving a nonlinear optimization problem is to apply a gradient search procedure. As an example of how the gradient search procedure works, consider the simplest case which is when the nonlinear function is concave, differentiable, and the solution space is two dimensional (see figure 1). Concave functions have the property that given any two points on the function, the line segment connecting the two points will lie entirely below (or on) the function. The gradient search applied to this simple case amounts to taking the derivative of the function and then evaluating the derivative at a particular value of x. If the derivative at x is positive, then the solution, x^*, must lie to the right of x. If the derivative at x is negative, then the solution lies to the left of x. Once a left and right bound are identified, the value of the derivative at the midpoint between the two bounds can be used in further tests. The search ends when the value of the derivative at x equals zero or when the distance between the left and right bounds is smaller than a given tolerance. When multiple variables are involved, there are countless possible directions

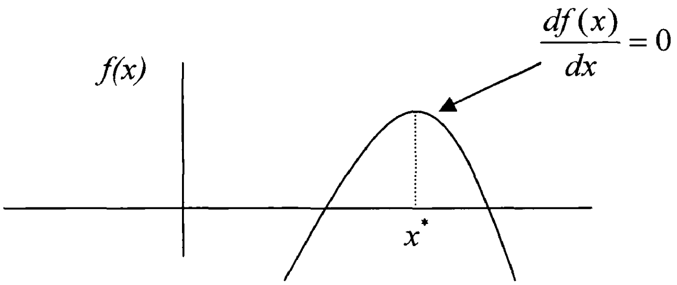


Figure 1. Example of a concave function that has an optimal solution at x*.

in which to move and partial derivatives are used for choosing the best search direction (see figure 2) [7].

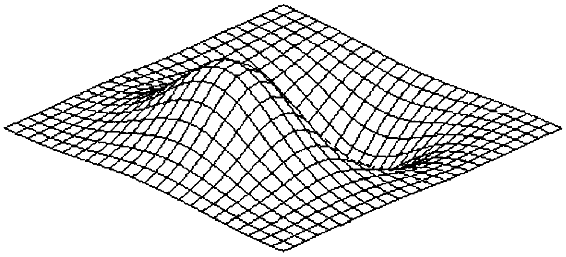


Figure 2. Example of a function that has multiple search directions at any point.

In the event that the objective function is not differentiable, such as nonlinear multiregressions based on Choquet integrals, the traditional gradient search procedure cannot be applied. A new optimization technique must be determined [14].

A reasonable replacement for the gradient search is a pseudo gradient search. The parameters in the pseudo gradient search are expressed as elements of a change vector $(\Delta x_1, \Delta x_2, ..., \Delta x_n)$, where n represents the number of dimensions in the search space. Differences instead of differentials are applied at each dimension, which avoids the problem of nondifferentiability altogether.

Iteration is necessary to progress through a series of trial solutions to reach the optimal solution. The quick convergence rate that accompanies a machine learning method such as a neural network makes it well suited to carry out the pseudo gradient search; however a neural network suffers from the problem of potentially converging to a local solution, rather than a global solution [12]. A genetic algorithm is another viable approach to the problem. It would provide a global solution, but at a cost of an undesirably slow convergence rate. This work will exploit the advantages of both the genetic algorithm and the neural network approaches. A genetic algorithm will be used to converge towards the global solution, but for a limited number of generations. Then a pseudo gradient search algorithm based on a neural network model will be initialized with the results from the genetic algorithm in order to quickly fine-tune the results and converge to the presumed global solution.

SIGNED FUZZY MEASURES AND CHOQUET INTEGRALS

Let *X* be the factor space which is composed of *n* predictive attributes $X = \{x_1, x_2, ..., x_n\}$. A signed fuzzy measure is a set function μ defined on (*X*, *𝒫(X)*) satisfying $\mu(\varnothing) = 0$, where *𝒫(X)* represents the power set of *X*. Usually, we can assume that $\mu(X) = 1$ and the measure is referred to as a regular signed fuzzy measure.

The data set for the problem takes the following form:

x_1	x_2	$\cdots$	x_n	y
f_{11}	f_{12}	$\cdots$	f_{1n}	y_1
f_{21}	f_{22}	$\cdots$	f_{2n}	y_2
$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
f_{l1}	f_{l2}	$\cdots$	f_{ln}	y_l

where row $f_{j1} \quad f_{j2} \quad \cdots \quad f_{jn} \quad y_j$ is the j^{th} observation of predictive attributes $x_1, x_2, ..., x_n$ and y_j is the corresponding value of the objective attribute. Each observation can be regarded as a function f_j, thus $f : X \to (-\infty, \infty)$ where $f_{ji} = f_j(x_i)$. The number of observations in the data set, *l*, is recommended to be greater than or equal to 5×2^n to safeguard against having too few observations to cover all of the μ values and to obtain suitable regression effects [13].

Non-linear multiregression is expressed as $y = c + \int (a + bf) d\mu + N(0, \sigma^2)$, where *c* is a constant, *a* and *b* are real-valued functions defined on *X*, *f* is an observation of $x_1, x_2, ..., x_n$, and $N(0, \sigma^2)$ is a normally distributed random perturbation with expectation 0 and variance σ^2 [14, 15]. The integral is taken to be the generalized Choquet integral which is defined as $\int f d\mu = \int_{-\infty}^{0} [\mu(F_\alpha) - \mu(X)] d\alpha + \int_{0}^{\infty} \mu(F_\alpha) d\alpha$ where not both terms on the right side are infinite and the set $F_\alpha = \{x \mid f(x) \geq \alpha, x \in X\}$ for any $\alpha \in (-\infty, \infty)$ [13]. Functions *a* and *b* are both expressed as vectors with the constraints that $a_i \geq 0$ for $i = 1, 2, ..., n$ with $\min_{1 \leq i \leq n} a_i = 0$, and $-1 \leq b_i \leq 1$ for $i = 1, 2, ..., n$ with $\max_{1 \leq i \leq n} |b_i| = 1$. For convenience, we take $\mu(X) \geq 0$.

GENETIC ALGORITHM

Overview

Genetic algorithms are designed to search for, call attention to, and propagate good solutions to a problem. A genetic algorithm flows in the manner presented in figure 3. Each individual in the population is a solution to the problem and is represented by one chromosome. Each chromosome is composed of genes which are represented as binary numbers. For the case of nonlinear multiregression, the dimensions of the *a* and *b* functions form the genes (see figure 4).

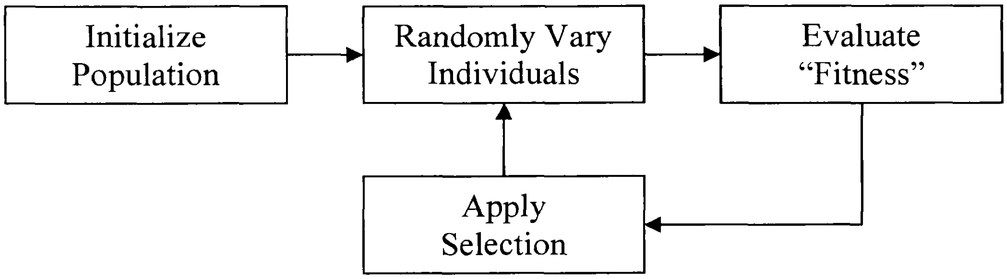


Figure 3. Flow of a genetic algorithm [4].

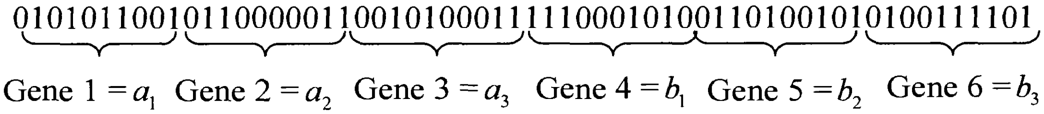


Figure 4. Example of what one chromosome looks like when there are three attributes in the data set and each gene is composed of 10 bits. One chromosome represents one individual in the population.

A hallmark of genetic algorithms is the mating step where two individuals in the current population are used to produce new offspring. During processing, the bits in the chromosomes are swapped, reversed, and changed to form new individuals in hopes of creating an offspring that is more valuable than its parents.

> *Consider a herd of antelope and suppose one of them has long feet, allowing it to escape a predator easier than could the other members of the herd, while another has exceptionally good sight, allowing it to find food easier than the others. Now when the two antelope mate, a crossover takes place and the chromosomes of their progeny will consist of pieces of the chromosomes of each parent. This in turn will lead to inheritance of parental traits by their progeny, and chances are that at least one of them will inherit both the long feet from one of the parents and the good sight from the other. Clearly, that young antelope will be superior to either of its parents.* [3]

The genetic operators that cause the genetic variation are three-bit mutation, two-point crossover, and two-point realignment. The genetic operator of three-bit mutation accepts one chromosome as input and returns one chromosome as output. Three random, unique integers between one and the total number of bits in the chromosome are generated. The values of the bits at the three positions indicated by the random numbers are then flipped causing the mutation (see figure 5). Two-point crossover accepts two chromosomes as input and returns two chromosomes as output. Two unique random numbers between one and the total number of bits in the chromosome are generated. These random numbers are used to split the chromosomes into three sections. The middle portions of the chromosomes are then swapped causing the crossover (see figure 6). Two-point realignment accepts one chromosome as input and returns one chromosome as output. The chromosome is split in the same way as the two-point crossover. Next, the three portions of the chromosome are rearranged and possibly reversed (see figure 7). The three portions of the chromosome can be rearranged in six ways and the decision about whether or not to reverse each of the three portions results

```
Original: 010101100101100000110010100011111000101001101001010100111101
     New: 010101100100100000010010100011111000101001101001000100111101
```

Figure 5. Example of three-bit mutation. The bits are flipped in positions 12, 19, and 50.

```
CH1:   010101100101100000110010100011111000101001101001010100111101
CH2:   101100101101010110001000011010101000101010010101011111101101
CH1:   010101100101100110001000011010101000101010010001010100111101
CH2:   101100101101010000110010100011111000101001101101011111101101
```

Figure 6. Example of two-point crossover. The original two chromosomes are split at positions 15 and 45. The middle positions are then swapped to produce two new chromosomes.

in 2^3 more variations. Together, a total of $6 \times 8 = 48$ different placements occur for each three portion split of the chromosome.

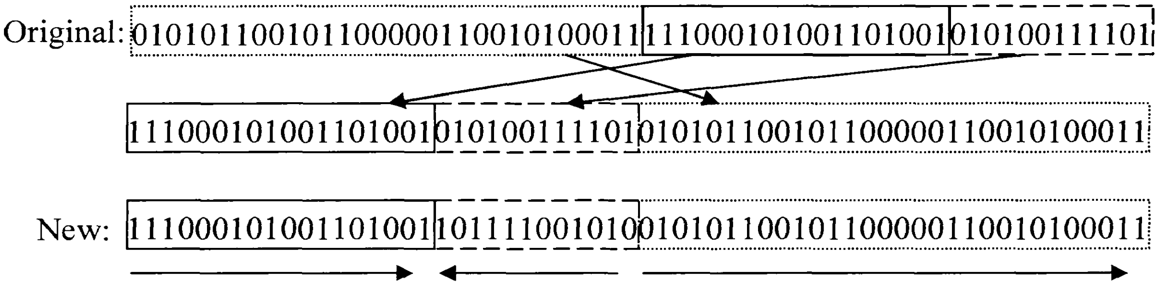


Figure 7. Example of two-point realignment. The chromosome is split at positions 30 and 48. The three pieces are then randomly swapped. Finally, the three pieces are randomly reversed.

With respect to multiregressions, an algebraic approach to finding $c, \mu_1, \mu_2, \ldots, \mu_{2^n-1}$ has been recently introduced [15]. The algebraic approach provides a huge savings in the running time of the algorithm because the size of the chromosome in the genetic algorithm is greatly reduced. The smaller the chromosome, the quicker the genetic algorithm will converge. This genetic algorithm is based on that work. A flowchart of the algorithm is presented in appendix A.

The algorithm

(1) For n, an integer equal to the number of attributes, express k in binary digits as $k = k_n k_{n-1} \cdots k_1$ for every $k = 0,1,2,\ldots,2^n - 1$. Example: $k = 6$ as a binary number is 110, so if $n = 4$, then $k_4 = 0,\ k_3 = 1,\ k_2 = 1$, and $k_1 = 0$.

(2) Use μ_k to denote $\mu(A)$ where $A = \bigcup_{k_i = 1} \{x_i\}$, $k = 0,1,2,\ldots,2^n - 1$. Example: $k = 6$ means $\mu_6 = \mu(\{x_2, x_3\})$. Note: $k = 0$ means $\mu_0 = \mu(\varnothing)$.

(3) Input integer *l* which equals the number of observations in the data set, and input the data set itself.

(4) Seed the random number generator using a large prime number such as 103,723. Set the value for the following parameters:

λ: The bit length of each gene which is dependent on the required precision of the results. For example, $\lambda = 10$ means that $2^{10} = 1024$ binary numbers in [0, 1) can be represented. That means the precision is $\frac{1}{1024}$ which is near 10^{-3}. The default for λ is 10.

p: The population size. It should be a large, positive, even integer. The default is 200.

α and β: The probabilities used in a random switch to control the choice of genetic operators for producing offspring from selected parents. The probability of using a three-bit mutation operator is represented by α, the probability of using a two-point crossover operator is represented by β, and $1-\alpha-\beta$ is the probability of using a two-point realignment operator [15]. They should satisfy the conditions that $\alpha \geq 0$, $\beta \geq 0$, and $\alpha+\beta \leq 1$. The default values for α and β are 0.2 and 0.5 respectively. A random switch uses the values for α and β to split the interval [0, 1) into three pieces (see figure 8). Depending on a random number generated on [0, 1), an appropriate genetic operator is selected. For example, if the random number generator produces a value of 0.66, then the random switch in figure 8 will select the two-point crossover as the genetic

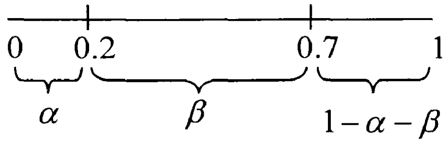


Figure 8. Example of a random switch.

operator since 0.66 falls in β 's range.

ε and δ : Small positive numbers used in stopping conditions. Their defaults are 10^{-10} and 10^{-6} respectively.

w: An integer to limit the number of successive generations that have not made significant progress. The default is 5.

(5) Calculate $\hat{\sigma}_y^2 = \frac{1}{l}\sum_{j=1}^{l}(y_j - \bar{y})^2$, where $\bar{y} = \frac{1}{l}\sum_{j=1}^{l} y_j$. Construct vector **q** $=(q_1, q_2, ..., q_l)$ as follows: $q_j = y_j$ where $j = 1,2,...,l$. Note: Index *j* will always be associated with the objective attribute and will span from 1 to *l*.

(6) Randomly create the initial population of *p* chromosomes. Each chromosome has 2*n* genes, denoted by $g_1, g_2, ..., g_n, g_{n+1}, g_{n+2}, ..., g_{2n}$; the first *n* genes represent vector *a*, the second *n* genes represent vector *b*. Each gene, g_i , represents a rational number in $[0,1)$. In binary, the numerator of g_i has the form $t_\lambda t_{\lambda-1} \cdots t_d \cdots t_1$, where $t_d \in \{0,1\}$, and $d = \lambda, \lambda-1, ..., 1$. The denominator is a one followed by λ zeros. For example: if $\lambda = 10$, $g_1 = \frac{0101011110}{10000000000}$. Converting g_1 to base 10 results in $g_1 = \frac{350}{1024}$.

(7) Variable *GC* is initialized to p and is used to count how many chromosomes have been generated. Variable *WT* is a counter that is initialized to zero; it increases when no significant progress has been made from one generation to the next generation. (Once progress is made, *WT* is reset back to zero.) *SE* is a variable that represents the saved error and it is initialized to $\hat{\sigma}_y^2$.

(8) Decode each chromosome to produce the elements of vectors a and b using the following formulae:

$$a_i = \frac{g_i - m(g)}{(1-g_i)(1-m(g))}, \text{ where } m(g) = \min_{1\le k\le n} g_k$$
$$b_i = \frac{2g_{n+i} - 1}{M(g)}, \text{ where } M(g) = \max_{1\le k\le n} |2g_{n+k} - 1|$$

where $i = 1, 2, ..., n$. Note: Index i will always be associated with the predictive attributes and will span from 1 to n.

(9) For each chromosome in the population, construct a matrix **Z** with dimensions $l \times 2^n$ as follows:

$$z_{j0} = 1,$$
$$z_{jk} = \begin{cases} \min(a_i + b_i f_{ji}) - \max(a_i - b_i f_{ji}), & \text{if it is} > 0 \text{ or if } k = 2^n - 1 \\ & 0, \text{otherwise} \end{cases}$$

where $j = 1, 2, ..., l$ and $k = 0, 1, ..., 2^n - 1$. If the columns of **Z** are not linearly independent, this chromosome should be rejected. A new chromosome should be randomly generated to replace the rejected chromosome. The new chromosome should be decoded using step 8 and it should havc thc property that the newly generated **Z** matrix has linearly independent columns. *GC* should be updated to include the number of rejected chromosomes.

(10) For each matrix **Z**, apply the QR decomposition theorem to find the least squares solution of the system of linear equations **Zv** = **q**, where the elements of **v** represent the unknown variables $c, \mu_1, \mu_2, ..., \mu_{2^n-1}$ [2]. There may be instances where *a* and *b* cause matrix **Z** to contain one or more columns which are all zero (i.e. matrix **R** in the QR decomposition is singular). When this happens, any columns which contain all zeros should be removed from matrix **Z** and μ's values that correspond to the removed columns can be arbitrarily set. The remaining values of μ are still determined from the least squares solution of the **Z** matrix with the columns of zeros removed.

(11) For each chromosome in the population, calculate the regression residual error $\hat{\sigma}^2$ which is defined as follows:

$$\begin{aligned}\hat{\sigma}^2 &= \frac{1}{l}\sum_{j=1}^{l}[y_j - c - \int(a+bf_j)d\mu]^2 \\ &= \frac{1}{l}\sum_{j=1}^{l}(y_j - c - \sum_{k=1}^{2^n-1} z_{jk}\mu_k)^2.\end{aligned}$$

The residual error of the r^{th} chromosome in the population is denoted by $\hat{\sigma}_r^2$.

(12) Let $m(\hat{\sigma}^2) = \min_{1\le r\le p} \hat{\sigma}_r^2$ and set $R = \{r \mid \hat{\sigma}_r^2 = m(\hat{\sigma}^2)\}$. Erase the last history record associated with *R*, if any, and save $m(\hat{\sigma}^2)$, *a*, *b*, *c*, and μ of all $r \in R$ in the current population. Display *GC, WT,* and $m(\hat{\sigma}^2)$.

(13) If $m(\hat{\sigma}^2) < \varepsilon\hat{\sigma}_y^2$, then go to step 22. The desired precision has been reached. Otherwise, take the next step.

(14) If $SE - m(\hat{\sigma}^2) < \delta\hat{\sigma}_y^2$, then $WT + 1 \Rightarrow WT$ and take the next step (i.e. no significant progress was made); otherwise, $0 \Rightarrow WT$ and go to step 16.

(15) If $WT > w$, then go to step 22, (i.e. no significant progress has been made in the past w generations); otherwise, take the next step.

(16) The relative goodness of the r^{th} chromosome in the current population is defined by $G_r = \dfrac{m(\hat{\sigma}^2)}{\hat{\sigma}_r^2}$, $r = 1,2,...,p$. $G_r \in (0,1]$ for all $r = 1,2,...,p$. Note: The smaller the error in the r^{th} chromosome, the larger G_r will be.

(17) The probability distribution of the r^{th} chromosome in the current population is defined by $p_r = \dfrac{G_r}{\sum_{r=1}^{p} G_r}$, $r = 1,2,...,p$. Note: This definition allows chromosomes with smaller error to have a larger probability of being chosen as a parent.

(18) Use the probability distribution $\{p_r \mid r = 1,2,...,p\}$ and a random switch to select two different chromosomes from the population to use as parents. Use α, β and a random switch to select a genetic operator to produce two new chromosomes as offspring.

(19) Repeat step 18 for $\dfrac{p}{2}$ total times to get a new generation of p chromosomes. $GC + p \Rightarrow GC$. Save $m(\hat{\sigma}^2)$ in SE.

(20) For each new chromosome, repeat steps 8-11 to determine *a*, *b*, *c*, μ and $\hat{\sigma}_r^2$.

(21) Use the magnitude of $\hat{\sigma}_r^2$ (the smaller the better) to select the *p* best chromosomes from both generations to form the new population. Then go to step 12.

(22) For all $r \in R$, check the sign of μ_{2^n-1}. In cases where $\mu_{2^n-1} < 0$, perform the following sub steps: replace *c* by $c + \mu_{2^{n-1}} \cdot \max\limits_{1 \le i \le n} a_i$, replace a_i by $\max\limits_{1 \le i \le n} a_i - a_i$, switch the sign of vector *b*, and switch the sign of all μ's values. (These sub steps are performed to ensure uniqueness in the solution.) If $WT > w$, proceed to the pseudo gradient search algorithm. Otherwise, display *s*, *p*, λ, *α*, *β*, ε, *δ*, and *w*. After deleting any duplicates from the history record, display *a*, *b*, *c*, and μ of the r^{th} chromosome for all $r \in R$. Stop.

Singular value decomposition versus QR decomposition

In step ten, there are three approaches to performing the matrix least squares computation: normal equations, singular value decomposition, and QR decomposition. The normal equations approach is the fastest but it also has the highest error and was dismissed to keep the results as accurate as possible [2]. The singular value decomposition is attractive because it is more stable than the QR decomposition on matrices that are not of full rank.

> ***Theorem (Singular Value Decomposition)*** – Let **A** be an arbitrary *m*-by-*n* matrix with $m \ge n$. Then we can write $\mathbf{A} = \mathbf{UDV}^{\mathrm{T}}$, where **U** is *m*-by-*n* and satisfies $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$, **V** is *n*-by-*n* and satisfies $\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}$, and **D** is a $n \times n$ diagonal matrix.
> ***Theorem (QR Decomposition)*** - Let **A** be *m*-by-*n* with $m \ge n$. Suppose that **A** has full column rank. Then there exists a unique *m*-by-*n* orthogonal matrix **Q** and a

unique n-by-n upper triangular matrix **R** with positive diagonals $r_{ii} > 0$ such that **A**=**QR**.

Definition – A matrix **B** is orthogonal if it has the property that $\mathbf{B}^{\mathrm{T}}\mathbf{B} = \mathbf{I}$, where **I** represents the identity matrix. [2]

Examining the performance of the singular value decomposition and the QR decomposition reveals that the singular value decomposition uses one additional expensive matrix multiplication in the final result (see figure 9). The QR decomposition was ultimately chosen because it involves fewer matrix multiplication operations.

Singular Value Decomposition	QR Decomposition
$\mathbf{A}x = b$	$\mathbf{A}x = b$
$\mathbf{UDV}^{\mathrm{T}}x = b$	$\mathbf{QR}x = b$
$\mathbf{U}^{\mathrm{T}}\mathbf{UDV}^{\mathrm{T}}x = \mathbf{U}^{\mathrm{T}}b$	$\mathbf{Q}^{\mathrm{T}}\mathbf{QR}x = \mathbf{Q}^{\mathrm{T}}b$
$\mathbf{IDV}^{\mathrm{T}}x = \mathbf{U}^{\mathrm{T}}b$	$\mathbf{IR}x = \mathbf{Q}^{\mathrm{T}}b$
$\mathbf{D}^{-1}\mathbf{DV}^{\mathrm{T}}x = \mathbf{D}^{-1}\mathbf{U}^{\mathrm{T}}b$	$\mathbf{R}^{-1}\mathbf{R}x = \mathbf{R}^{-1}\mathbf{Q}^{\mathrm{T}}b$
$\mathbf{IV}^{\mathrm{T}}x = \mathbf{D}^{-1}\mathbf{U}^{\mathrm{T}}b$	$x = \mathbf{R}^{-1}\mathbf{Q}^{\mathrm{T}}b$
$\mathbf{VV}^{\mathrm{T}}x = \mathbf{VD}^{-1}\mathbf{U}^{\mathrm{T}}b$	
$x = \mathbf{VD}^{-1}\mathbf{U}^{\mathrm{T}}b$	

Figure 9. Comparing the performance of the singular value decomposition and QR decomposition on an arbitrary matrix equation.

Efficient implementation of the genetic operations

The genetic operations performed in step eighteen can be implemented in more than one way. A genetic operation can be performed once, twice, or $2n$ times per chromosome (see figure 10). The goal is to find an implementation that gives a suitable amount of genetic variation yet it must not give so much variation that it mimics a pure random search. To test which implementation provides the most efficient results, three simulations were conducted. The even natural number n was used to represent how

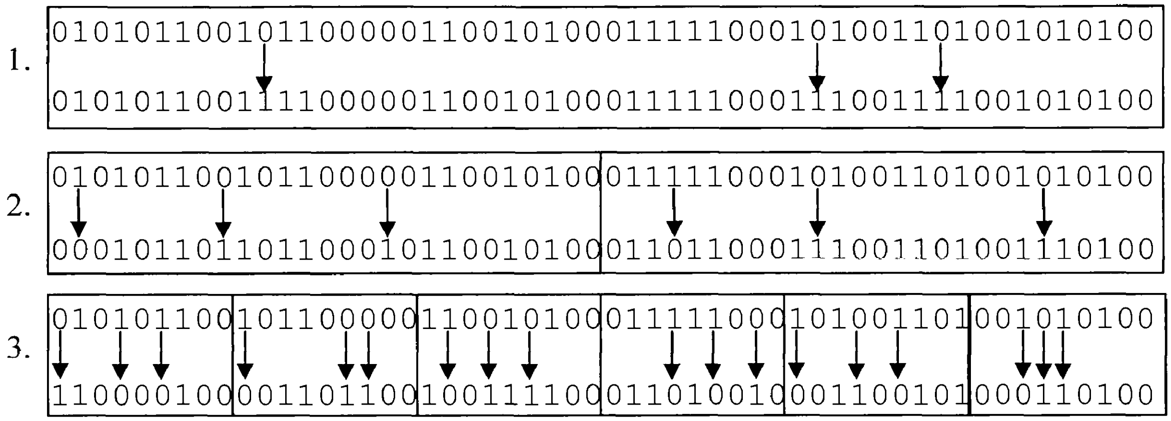


Figure 10. 1. Performing a three-bit mutation once on the chromosome as a whole. 2. Performing a three-bit mutation twice. The first mutation is performed on the genes that make up the *a* vector. The second mutation is performed on the genes that make up the *b* vector. 3. Performing the three-bit mutation once on each gene.

many random numbers on the interval [0, 1023] were generated in a simulation; it also represents the number of genes in one chromosome. The goal of the program was to use a genetic algorithm to reclaim the original *n* random numbers by having each gene represent one of the numbers. Two hundred chromosomes, each containing 10*n* bits, were randomly generated as an initial population for the genetic algorithm. Three tests per simulation were conducted to examine the average sum of squares error and the average number of generated chromosomes. The first test performed genetic operations on the chromosome as a whole. The second test performed genetic operations twice on the chromosome (once on the first half representing the *a* vector, once on the second half representing the *b* vector), and the final test performed *n* genetic operations (once per gene). In all simulations, implementation of the genetic operators at the chromosome level provided the smallest amount of error (see table 1). Too much genetic variation is caused by performing a genetic operation multiple times in one mating instance. As an added bonus, coding the genetic operations at the chromosome level is much more straightforward, which makes the code easier to follow. A drawback of chromosome-level implementation is that the number of generated chromosomes was also higher than in the gene level and vector level cases, but the decreased error was worth the extra effort. A flowchart depicting one iteration of the simulation is provided in appendix B.

Simulation Details		**Test 1 Chromosome level**		**Test 2 Twice per chromosome**		**Test 3 Gene level**	
n	# trials	Avg. error	Avg. GC*	Avg. error	Avg. GC	Avg. error	Avg. GC
6	1000	**153.082**	15870.8	166.909	13445.0	557.419	13418.6
6	500	**102.702**	15743.6	190.334	13671.6	499.534	13365.2
8	500	**267.16**	21013.2	361.392	17016.8	1417.26	17391.2

Table 1. Simulation results. Testing which implementation of the genetic operators is most efficient. *GC represents the number of generated chromosomes.

PSEUDO GRADIENT SEARCH

Neural network motivation

Neural networks are powerful modeling tools capable of learning complex nonlinear functions that were previously unknown in data sets [1]. In this work, neural networks will be used as a basis for the design of a pseudo gradient search algorithm.

Neural networks are based on a rudimentary biological model of the human brain. Millions of interconnected units, called neurons, send and receive signals between one another in an attempt to process sensory input (see figure 11). The input mechanism of a neuron is a web of arms collectively referred to as dendrites. The output mechanism of a neuron is a single long branch referred to as the axon. The end

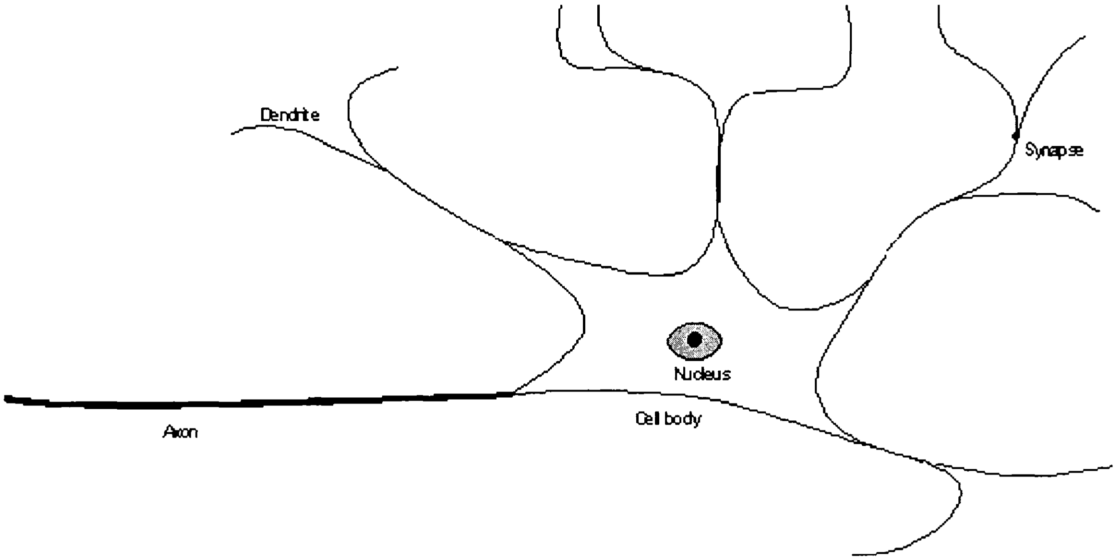


Figure 11. Illustration of a neuron [5].

of each axon splits into many branches, and each branch contains a synapse at its end. A neuron connects to dendrites of other neurons through the synapse. The role of the synapse is to convert the axon's signal into an electrochemical signal that travels through the dendrites and reaches the cell body where it is processed. If the sum total of the input signals exceeds a threshold value of the cell body, the cell will "fire" a signal through its axon. The signal will reach all the neurons attached to this neuron (hence the name "network") and the process begins again. Learning occurs by either exciting or inhibiting the flow of the signal through the synapse [8].

Scientists have not yet uncovered all the secrets of how biological neurons work, so artificial neurons are produced using the most essential features of biological neurons. An artificial neural network unit is comprised of an input layer, an output layer, and zero or more hidden layers. The links between the layers are memory units called connection weights. A signal traveling along a link can either be strengthened or weakened by the value of the weight on a link. A hidden unit will sum together the values of the weighted input into the unit and then pass the sum through an activation function, which is commonly the S-shaped sigmoid function $y = \frac{1}{1+e^{-x}}$ depicted in figure 12. If the weights and the input values are closely matched, the output from the activation function will be close to 1. If the weights and input values oppose one another, the activation function will be close to 0. Intermediate agreements will take on appropriate values [1].

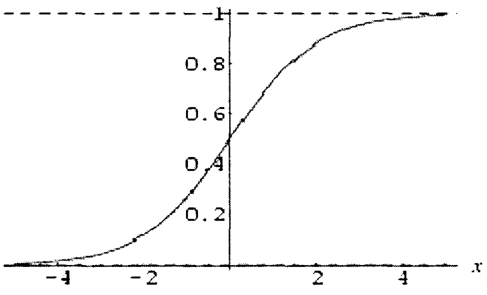


Figure 12. Illustration of the sigmoid function [6].

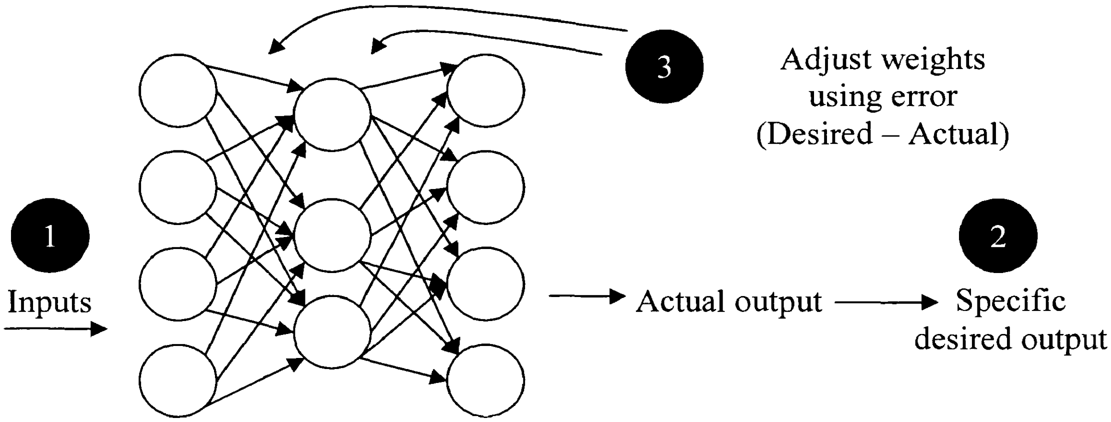


Figure 13. Supervised learning paradigm [1].

When a neural network is trained to model a data set, a learning technique called supervision can be applied (see figure 13). The input is sent through the network and the output is examined. If the actual output does not match the desired output from the data set, the weights in the neural network are updated and the input is sent through again. The algorithm used to implement supervised learning in this manner is called back propagation. This process is illustrated through an example in figure 14. In the first iteration, the neural network outputs a value that is greater than the desired value. The weights are adjusted and in the second iteration, the actual neural network approximation is closer to the desired result from the data set. Once again the weights are adjusted and the third iteration has a value even closer to the desired result. Because the input values are being pushed forward through the network, the connection type is referred to as feed-forward. Diagrams depicting feed-forward connections use arrows pointing from the left to the right.

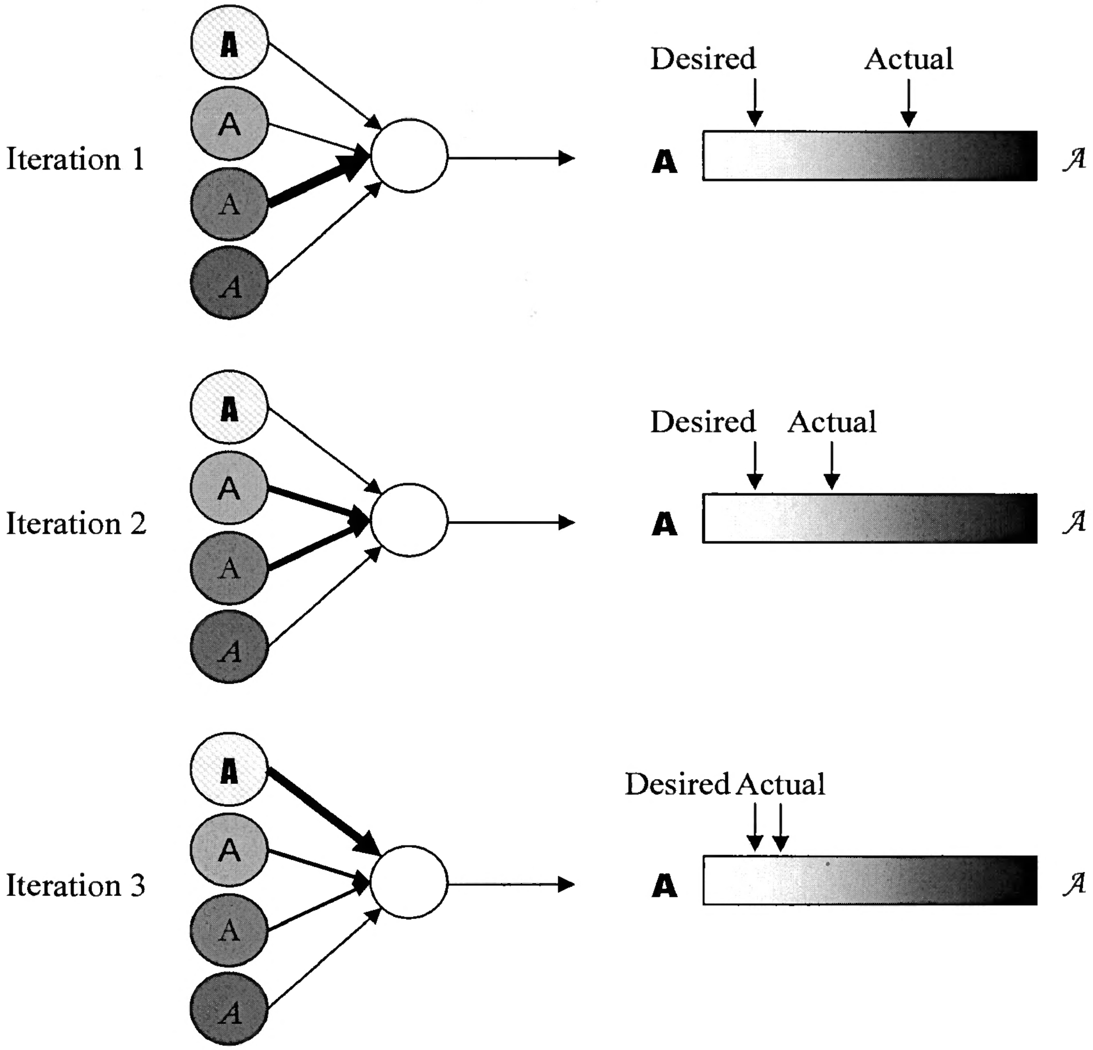


Figure 14. Illustration of a neural network that learns to distinguish differences in the capital letter A.

Why a neural network algorithm cannot be applied

There are two reasons why a neural network algorithm cannot be directly applied to learn the values of the *a* and *b* vectors in nonlinear multiregressions based on generalized Choquet integrals. Consider the neural network design in figure 15. The

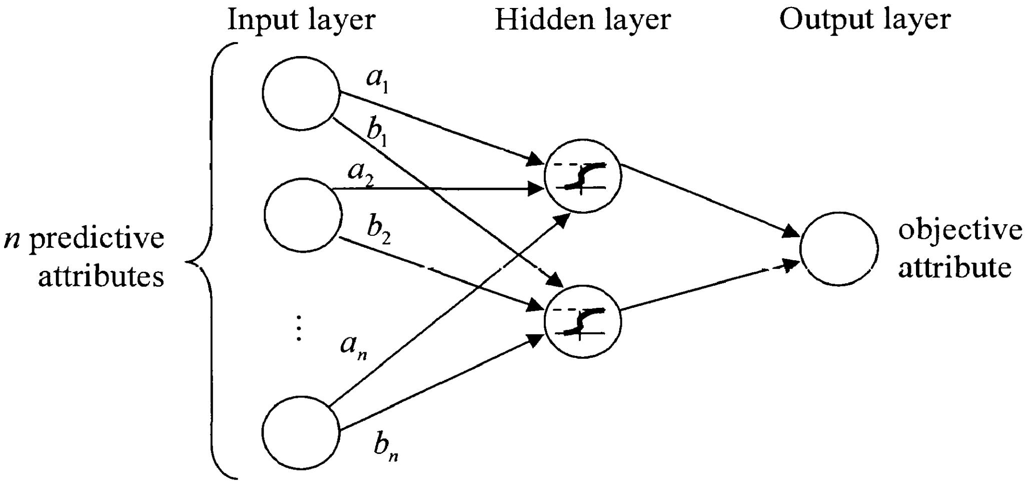


Figure 15. A neural network design to model multiregression.

predictive attributes are used as the input to the network, and the actual output to be modeled by the network is the objective attribute. Since the *a* and *b* vectors are what must be changed, the weights connecting the input layer to the hidden layer are taken to be the elements of the *a* and *b* vectors. The hidden layer is composed of two nodes using the sigmoid function as an activation function. The first problem is that this set-up does not use the formula for multiregression, so the resulting weights will have nothing to do with the *a* and *b* vectors used in the multiregression formula. It would be as if the weights were randomly initialized at the start of the neural network. The second problem arises if the formula for multiregression is used in place of the sigmoid function. The multiregression formula based on generalized Choquet integrals is not differentiable and the gradient descent formula used to adjust the weights in the back propagation algorithm cannot be used [9].

A different approach: the pseudo gradient search

Neural network ideas are still useful in designing an alternative algorithm. The predictive attributes still represent the input, the objective attribute is still the output, the connections are still feed-forward, and the *a* and *b* vectors are updated in each iteration so the overall error is reduced. The new concept is to apply differences instead of differentials to compose a change vector that will be used to update the *a* and *b* vectors.

To begin, the root mean square error, *e*, is calculated using *a*, *b*, and the observations from the data set. The *a* and *b* vectors are examined separately starting with the *a* vector. Beginning with the first dimension of vector *a*, a step of length δ is taken in the positive direction (see figure 16). Next, the multiregression formula is calculated for each observation of the data set using this new value of a_1. The root mean square error formula is then used to calculate the error, e_{δ^+}. Next, a step of length δ is taken in the negative direction from the original a_1. Once again, the multiregression formula is calculated for each observation of the data set and the root mean square error formula is used to find e_{δ^-}. Next, the change vector (*CV*) will be updated to indicate which step (if any) reduced the error the most. The change vector has length 2*n*. The first *n* elements of *CV* represent the error change of vector *a*. The second *n* elements represent the error change of vector *b*. In the event that both steps increased the error or the error did not change at all (i.e. $e_{\delta^+} \geq e$ and $e_{\delta^-} \geq e$) a step length of 0 is adopted as the first component of change vector, *CV*. Otherwise, one of

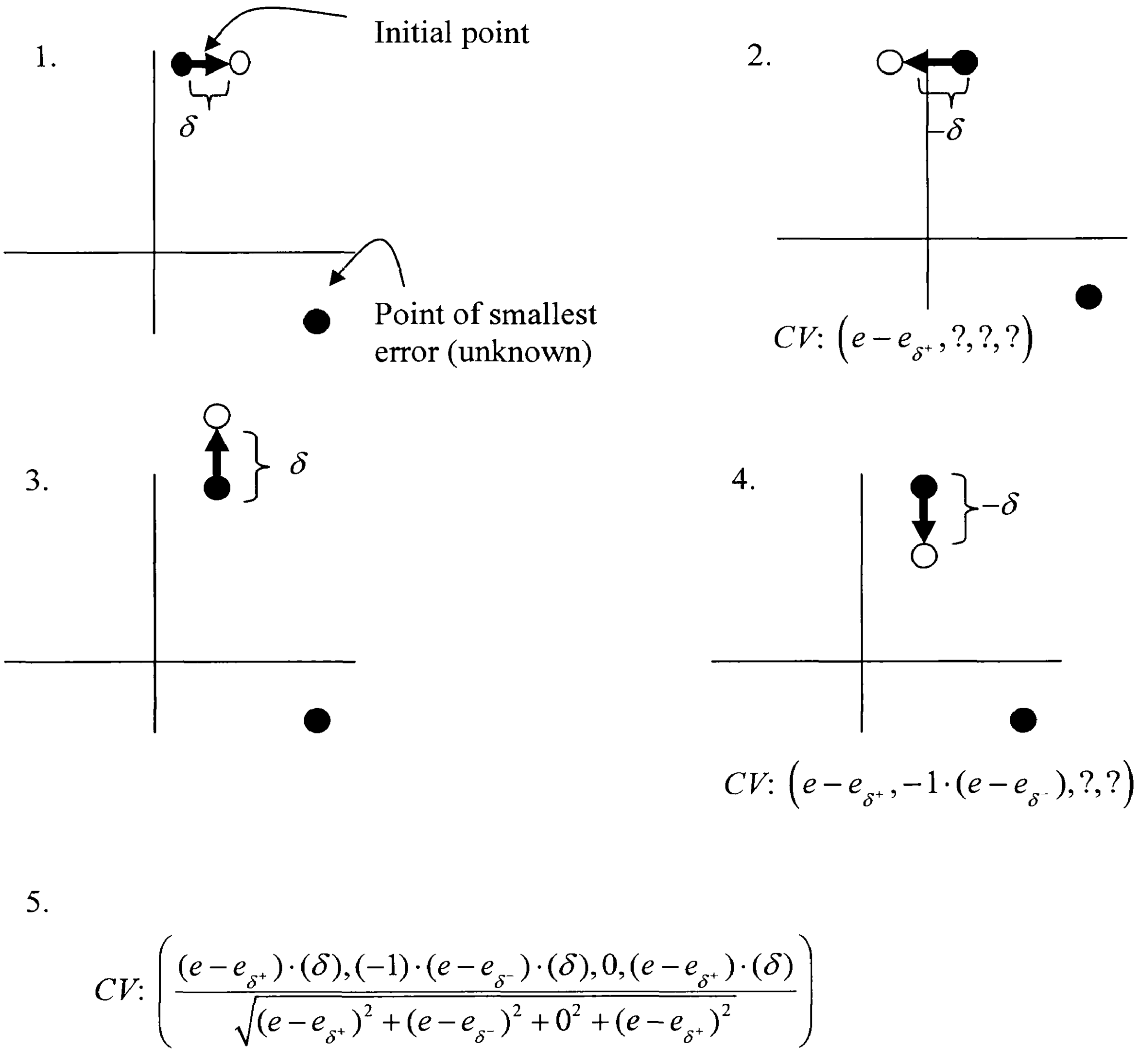

1.
Initial point
δ
Point of smallest error (unknown)
2.
$-\delta$
$CV\colon \left(e - e_{\delta^+}, ?, ?, ?\right)$
3.
δ
4.
$-\delta$
$CV\colon \left(e - e_{\delta^+}, -1 \cdot (e - e_{\delta^-}), ?, ?\right)$
5.
$CV\colon \left(\frac{(e - e_{\delta^+}) \cdot (\delta), (-1) \cdot (e - e_{\delta^-}) \cdot (\delta), 0, (e - e_{\delta^+}) \cdot (\delta)}{\sqrt{(e - e_{\delta^+})^2 + (e - e_{\delta^-})^2 + 0^2 + (e - e_{\delta^+})^2}}\right)$

Figure 16. Sequence of steps used in the pseudo gradient search to compose a change vector when two attributes are present. 1. A step of length δ in the positive direction is taken for the first dimension of vector *a*. 2. A step of length δ in the negative direction is taken for the first dimension. The positive step resulted in a larger error reduction, so the first component of the change vector will be $e - e_{\delta^+}$. 3. A step in the positive direction is taken for the second dimension. 4. A step in the negative direction is taken for the second dimension. The negative direction resulted in a smaller error, so the second component of the change vector will be $-1 \cdot (e - e_{\delta^-})$. 5. Next vector *b* is processed and the composed change vector is normalized. The normalized vector is multiplied by δ .

the steps decreased the error. If the step in the positive direction decreased the error more than the step in the negative direction, then $e - e_{\delta^+}$ will be adopted as the first component of the change vector. If the step in the negative direction decreased the error more than the step in the positive direction, then $-1(e - e_{\delta^-})$ will be adopted as the first component of the change vector. The factor $e - e_{\delta^+}$ ($e - e_{\delta^-}$ respectively) is used to scale future steps in this dimension. If the error is greatly reduced in this dimension, then $e - e_{\delta^+}$ ($e - e_{\delta^-}$ respectively) will be large and it will encourage even larger future steps in this dimension. Next, the variable a_1 is reset to its original value and then the next dimension, a_2, is examined. Once all dimensions of vector *a* have been considered, then vector *b* takes its turn. When vector *b* is done processing, the change vector is normalized by dividing each element of *CV* by $\sqrt{CV_1^2 + CV_2^2 + \cdots + CV_{2n}^2}$. Finally, each element of *CV* is multiplied by δ.

Once the final change vector has been determined, phase two begins. In this stage, the elongation of the change vector is addressed. To begin, the magnitude of the change vector is doubled and the respective portions of the change vector are added to the *a* and *b* vectors (see figure 17). The multiregression formula is calculated for each observation of the data set and the root mean square error is determined. If the error decreases compared to the previous error (the original error in this case), then once again the magnitude of the change vector is doubled, the multiregression formula is calculated, and the root mean square error (RMSE) is calculated. This process continues until the RMSE grows from one iteration to the next iteration. Then the

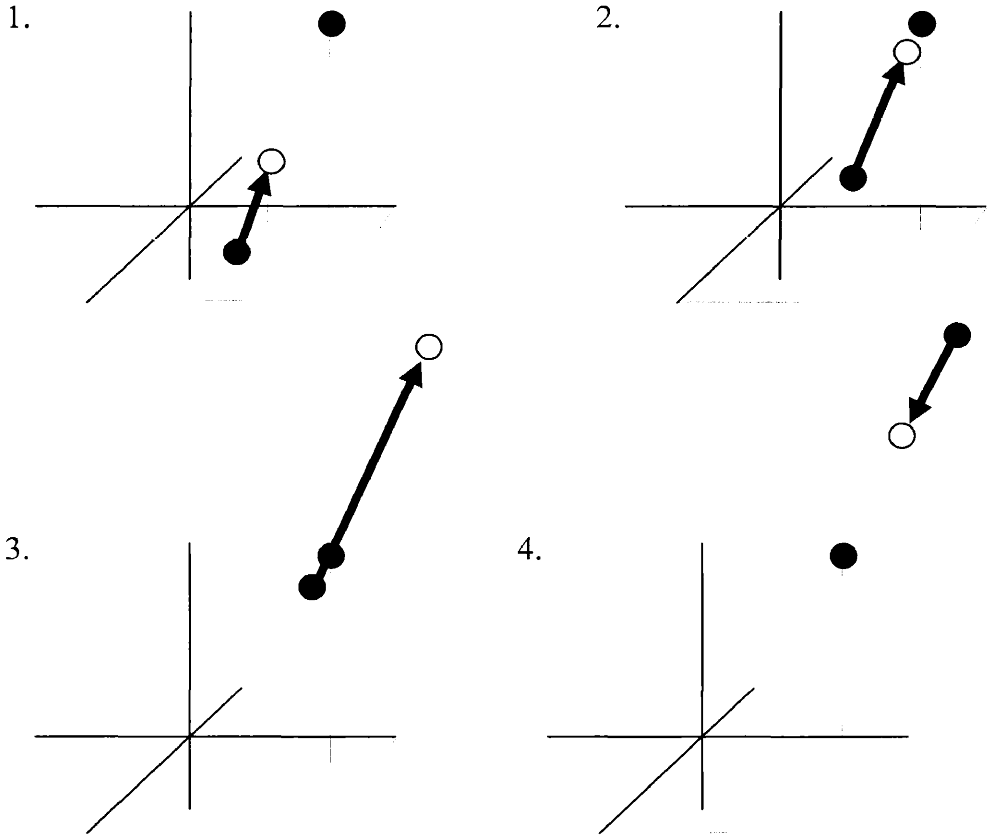


Figure 17. Illustration of how the change vector is elongated. Note: The elongation happens in parallel for both vectors *a* and *b* but only one vector is shown here. 1. The magnitude of the change vector is doubled. 2. The magnitude of the change vector is doubled again. 3. The magnitude of the change vector is doubled, but this time the RMSE grows instead of decreases. 4. The direction of the change vector is reversed and the magnitude is halved. The change vector will continue to be halved in this direction until the RMSE grows between iterations or until the absolute value of each dimension in the change vector is less than δ.

direction of the change vector is reversed and the magnitude of the change vector is cut in half. The change vector is iteratively halved in this reversed direction until the root mean square error grows between iterations, or the absolute value of each dimension of the change vector is less than delta on each of its components. A flowchart of the algorithm is presented in appendix C.

The algorithm

(1) Retrieve the a, b, c, and μ values of one chromosome from set R in the genetic algorithm. Also retrieve l, δ, n, f_j and y_j for $j = 1,2,...,l$. Construct vector $\mathbf{q} = (q_1, q_2, ..., q_l)$ as follows: $q_j = y_j$ where $j = 1,2,...,l$.

(2) $a_i \Rightarrow \hat{a}_i$ and $b_i \Rightarrow \hat{b}_i$ for all $i = 1,2,...,n$. The $\hat{a}$ and $\hat{b}$ vectors represent copies of a and b. The copies are manipulated in the algorithm rather than vectors a and b. At the end of a complete iteration, the a and b vectors are set to $\hat{a}$ and $\hat{b}$ which have been determined to have the least error thus far.

(3) Calculate $\hat{y}_j = c + \int(\hat{a} + \hat{b}f_j)d\mu$ for all $j = 1,2,...,l$. The current estimate of the objective attribute for the j^{th} observation in the data set is represented by $\hat{y}_j$.

(4) Calculate the initial error $e_0 = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(y_j - \hat{y}_j)^2}$.

(5) $e_0 \Rightarrow e$. The root mean square error calculated using the current values of $\hat{a}$ and $\hat{b}$ in the multiregression formula is represented by e.

(6) $0 \Rightarrow CV_h$ for $h = 1,2,...,2n$. The first n elements of vector CV store the change in each dimension of vector a. The second n elements of vector CV store the change in each dimension of vector b.

(7) For all $i = 1,2,...,n$,

   a. $a_p \Rightarrow \hat{a}_p$ for $p = 1,2,...n$. In this for loop, each dimension of the $\hat{a}$ vector should be considered independent of changes made in previous dimensions. This step resets the $\hat{a}$ vector back to a before determining the change direction for dimension i.

b. Take a step in the positive direction by adding δ to $\hat{a}_i$.

c. Construct matrix **Z** with dimensions $l \times 2^n$ as follows:

$$z_{j0} = 1,$$
$$z_{jk} = \begin{cases} \min(\hat{a}_i + \hat{b}_i f_{ji}) - \max(\hat{a}_i - \hat{b}_i f_{ji}), & \text{if it is} > 0 \text{ or if } k = 2^n - 1 \\ 0, \text{otherwise} \end{cases}$$

where $j = 1, 2, \ldots, l$ and $k = 0, 1, \ldots, 2^n - 1$.

d. Apply the QR decomposition theorem to find the matrix least squares solution of the system of linear equations **Zv** = **q**, where the elements of **v** represent the unknown variables $c, \mu_1, \mu_2, \ldots, \mu_{2^n-1}$ [2]. There may be instances where $\hat{a}$ and $\hat{b}$ cause matrix **Z** to contain one or more columns which are all zero (i.e. matrix **R** in the QR decomposition is singular). When this happens, any columns which contain all zeros should be removed from matrix **Z** and μ's values that correspond to the removed columns can be arbitrarily set. The remaining μ values are still determined from the matrix least squares solution of the **Z** matrix with the columns of zeros removed.

e. Calculate $\hat{y}_j$ for all $j = 1, 2, \ldots, l$ using the formula in step 3.

f. Calculate the error, e_{δ^+}, using the formula in step 4.

g. Take a step in the negative direction by subtracting 2δ from $\hat{a}_i$. Note: Ultimately, δ will be subtracted from a_i.

h. Repeat steps 7c – 7e. Then calculate the error, e_{δ^-}, using the formula in step 4.

i. If $e_{\delta^+} \geq e$ and $e_{\delta^-} \geq e$ then $0 \Rightarrow CV_i$. In this case, the step in the positive direction and the step in the negative direction caused the error to increase rather than decrease. No step should be taken so this dimension of the change vector is set to zero. Otherwise, if $e_{\delta^+} < e_{\delta^-}$ then $e - e_{\delta^+} \Rightarrow CV_i$, else $(-1)\left(e - e_{\delta^-}\right) \Rightarrow CV_i$.

(8) $a_i \Rightarrow \hat{a}_i$ for $i = 1, 2, ..., n$. Vector $\hat{a}$ is reset back to a so when the $\hat{b}$ vector is processed next it will not be influenced by any $\hat{a}$ adjustments that were made when trying to determine the optimal step direction from step 7.

(9) For all $i = 1, 2, ..., n$,

a. $b_p \Rightarrow \hat{b}_p$ for $p = 1, 2, ..., n$. Each dimension should be considered independent of changes made in previous dimensions. Vector $\hat{b}$ is reset back to b before determining the change direction for dimension i.

b. Take a step in the positive direction by adding δ to $\hat{b}_i$.

c. Determine the new c and μ values by performing steps 7c and 7d.

d. Calculate $\hat{y}_j$ for all $j = 1, 2, ..., l$ using the formula in step 3.

e. Calculate the error, e_{δ^+}, using the formula in step 4.

f. Take a step in the negative direction by subtracting 2δ from $\hat{b}_i$. Note: Ultimately, this will subtract δ from b_i.

g. Repeat steps 10c and 10d. Then calculate the error, e_{δ^-}, using the formula in step 4.

h. If $e_{\delta^+} \geq e$ and $e_{\delta^-} \geq e$ then $0 \Rightarrow CV_{i+n}$. The step in the positive direction and the step in the negative direction increased the error. The step length should be set to zero. Otherwise, if $e_{\delta^+} < e_{\delta^-}$ then $e - e_{\delta^+} \Rightarrow CV_{i+n}$, else $(-1)\cdot(e - e_{\delta-}) \Rightarrow CV_{i+n}$.

(10) $b_i \Rightarrow \hat{b}_i$ for $i = 1, 2, \ldots, n$. Vector $\hat{b}$ is reset to b.

(11) $\dfrac{\delta CV_h}{\sqrt{CV_1^2 + CV_2^2 + \cdots + CV_{2n}^2}} \Rightarrow CV_h$ for $h = 1, 2, \ldots, 2n$. The change vector must be normalized.

(12) Now the process of doubling the change vectors begins. The variable *PE* represents the previous error and variable *LE* represents the latest error. Initialize *PE* and *LE* as follows: $e \Rightarrow PE$ and $e \Rightarrow LE$.

(13) Update *PE*. $LE \Rightarrow PE$.

(14) For all $h = 1, 2, \ldots, 2n$,

a. Double the magnitude of the change vector in this dimension. $2 \cdot CV_h \Rightarrow CV_h$.

b. Incorporate the doubling from the previous step. If $h \leq n$, then

$\hat{a}_h + CV_h \Rightarrow \hat{a}_h$ else $\hat{b}_{h-n} + CV_h \Rightarrow \hat{b}_{h-n}$.

c. If $h \leq n$, then if $\hat{a}_h < 0$, $0 \Rightarrow \hat{a}_h$. A multiregression constraint is that the values of the a vector must be nonnegative.

d. If $h > n$

i. If $\hat{b}_{h-n} > 1$, then $\hat{b}_{h-n} = 1$.

ii. If $\hat{b}_{h-n} < -1$, then $\hat{b}_{h-n} = -1$.

(15) Determine the new c and μ values by performing steps 7c and 7d.

(16) Calculate $\hat{y}_j$ for all $j = 1,2,...,l$ using the formula in step 3.

(17) Calculate LE using the formula in step 4.

(18) If $LE < PE$, go back to step 13. Doubling the magnitude of vector CV again may lessen the error even further.) Otherwise go to the next step.

(19) The doubling performed in steps 13-17 went too far. We need to reverse the direction of the change vector and return half as far in each step. Reverse the direction of the change vector by setting $-CV_h \Rightarrow CV_h$ for all $h = 1,2,...,2n$.

(20) $LE \Rightarrow PE$.

(21) For all $h = 1,2,...,2n$,

a. $\frac{1}{2} \cdot CV_h \Rightarrow CV_h$

b. Incorporate the halving from the previous step. If $h \leq n$, then

$\hat{a}_h + CV_h \Rightarrow \hat{a}_h$ else $\hat{b}_{h-n} + CV_h \Rightarrow \hat{b}_{h-n}$.

c. If $h \leq n$, then if $\hat{a}_h < 0$, $0 \Rightarrow \hat{a}_h$. A multiregression constraint for a is that its values must be nonnegative.

d. A multiregression constraint for b is that its values fall in the range [-1, 1]. If $h > n$,

i. If $\hat{b}_{h-n} > 1$, then $\hat{b}_{h-n} = 1$.

ii. If $\hat{b}_{h-n} < -1$, then $\hat{b}_{h-n} = -1$.

(22) Determine the new c and μ values by performing steps 7c and 7d.

(23) Calculate $\hat{y}_j$ for all $j = 1, 2, ..., l$ using the formula in step 3.

(24) Calculate LE using the formula in step 4.

(25) Let $H = \max_{1 \leq h \leq 2n}\{|CV_h|\}$. If $LE < PE$ and $H > \delta$ then go back to step 20, otherwise go to the next step.

(26) Reverse the direction of the change vector again because the halving went too far. $-CV_h \Rightarrow CV_h$ for all $h = 1, 2, ..., 2n$.

(27) If $H > \delta$ then go back to step 20, otherwise go to the next step.

(28) Determine the new c and μ values by performing steps 7c and 7d.

(29) Calculate $\hat{y}_j$ for all $j = 1, 2, ..., l$ using the formula in step 3.

(30) Calculate e using the formula in step 4.

(31) Display the initial error, e_0.

(32) Let $M(\Delta ab) = \max_{1 < i < n}\{|a_i - \hat{a}_i|, |b_i - \hat{b}_i|\}$. Ultimately, $M(\Delta ab)$ will be checked to see if the largest change in any dimension of the change vectors was greater than δ.

(33) If $e_0 > 0$, $e > 0$ and $M(\Delta ab) \geq \delta$ then continue to the next step. Otherwise, proceed to step 39.

(34) If $e > e_0$, continue with the next step. Otherwise, proceed to step 36.

(35) The change from a to $\hat{a}$ and b to $\hat{b}$ needs to be iteratively reduced until the error is smaller than the initial error.

a. $a_i - \hat{a}_i \Rightarrow \tilde{a}_i$, $b_i - \hat{b}_i \Rightarrow \tilde{b}_i$ for all $i = 1, 2, \ldots, n$.

b. For all $i = 1, 2, \ldots, n$, if $|\tilde{a}_i| > \delta$ then $\frac{\tilde{a}_i}{2} \Rightarrow \tilde{a}_i$ and $\hat{a}_i + \tilde{a}_i \Rightarrow \hat{a}_i$.

c. For all $i = 1, 2, \ldots, n$, if $|\tilde{b}_i| > \delta$ then $\frac{\tilde{b}_i}{2} \Rightarrow \tilde{b}_i$ and $\hat{b}_i + \tilde{b}_i \Rightarrow \hat{b}_i$.

d. Determine the new c and μ values by performing steps 7c and 7d.

e. Calculate $\hat{y}_j$ for all $j = 1, 2, \ldots, l$ using the formula in step 3.

f. Calculate e by using the formula in step 4.

g. Let $M = \max_{1 \leq i \leq n}\{|\tilde{a}_i|, |\tilde{b}_i|\}$. If $e > e_0$ and $M > \delta$, go to step 35b, otherwise continue with the next step.

(36) New values for vectors a and b have been found which reduce the error from the previous iteration. Update a and b to retain the new values. $\hat{a}_i \Rightarrow a_i$, $\hat{b}_i \Rightarrow b_i$ for all $i = 1, 2, \ldots, n$.

(37) $e \Rightarrow e_0$.

(38) Go to step 6.

(39) Output a, b, c and μ. Stop.

Implementation considerations

When the vectors are examined to determine the optimal step direction, a positive step and a negative step are taken for each of the $2n$ elements resulting in a complexity of $O(n)$. Suppose $n = 3$. These twelve vectors are examined: $(a_1 + \delta, a_2, a_3)$, $(a_1 - \delta, a_2, a_3)$, $(a_1, a_2 + \delta, a_3)$, $(a_1, a_2 - \delta, a_3)$, $(a_1, a_2, a_3 + \delta)$, $(a_1, a_2, a_3 - \delta)$, $(b_1 + \delta, b_2, b_3)$, $(b_1 - \delta, b_2, b_3)$, $(b_1, b_2 + \delta, b_3)$, $(b_1, b_2 - \delta, b_3)$, $(b_1, b_2, b_3 + \delta)$, and $(b_1, b_2, b_3 - \delta)$. Next, a $2n$-dimensional change vector is composed based on which of the previous twelve vectors decreased the error the most. Suppose the positive step in the first, fourth, and sixth dimensions and the negative step in the second and fifth dimensions decreased the error while neither of the steps in the third dimension decreased the error. The composed change vector is

$$\left(\frac{\left(e - e_{\delta^+}\right)\cdot(+\delta), \left(e - e_{\delta^-}\right)\cdot(-\delta), 0, \left(e - e_{\delta^+}\right)\cdot(+\delta), \left(e - e_{\delta^-}\right)\cdot(-\delta), \left(e - e_{\delta^+}\right)\cdot(+\delta)}{\sqrt{\left(e - e_{\delta^+}\right)^2 + \left(e - e_{\delta^-}\right)^2 + 0^2 + \left(e - e_{\delta^+}\right)^2 + \left(e - e_{\delta^-}\right)^2 + \left(e - e_{\delta^+}\right)^2}} \right).$$

For brevity the notation $(CV_1, CV_2, CV_3, CV_4, CV_5, CV_6)$ will be used. An assumption is then made that $\left(a_1 + CV_1, a_2 + CV_2, a_3 + CV_3, b_1 + CV_4, b_2 + CV_5, b_3 + CV_6,\right)$ will lower the error even further. Once the change vector is composed, it may be the case that the error actually grows rather than decreases. A way to avoid this problem is to examine every possible change vector that could be composed. This raises the complexity to $O(3^{2n})$ because steps of length 0, δ, and $-\delta$ must be taken in each of the $2n$ dimensions.

Does the extra processing time required by examining all change vectors result in improved results?

The answer is no. Two programs were created with the primary difference being how the change vectors were determined. One program took positive and negative steps in each dimension while the second program examined all possible combinations of steps. In all cases, the accuracy of the results was comparable (see table 2). There was little difference in processing speed with three attributes, but for files with more than three attributes the program that examined all possible change vectors was noticeably slower.

Type of data set	**Number of attributes**	**Final RMSE** $O(n)$	**Final RMSE** $O(3^{2n})$	**Processing time (min)** $O(n)$	**Processing time (min)** $O(3^{2n})$
Artificial	3	1.28×10^{-7}	1.10×10^{-7}	0.1403	0.2678
Artificial	4	3.6×10^{-7}	3.9×10^{-7}	1.45	5.623
Artificial	4	1.00×10^{-7}	1.047×10^{-7}	1.4357	4.906
Real	3	1.4237	1.4299	0.0625	0.06125
Real	3	45.8887	49.32	0.0606	0.0619
Real	8	1.95122	*	1870	*

Table 2. Comparing the accuracy and running times of two programs. There is no benefit to examining all possible change vectors. * Following 3660 minutes of processing, the program was terminated.

An improvement over the iterative search

The pseudo gradient search represents an improvement over the iterative search presented in a previous paper by myself. In the iterative search, the doubling and halving steps are performed one dimension at a time and then a final change vector is composed once the halving steps have completed (see figure 18). The pseudo gradient search performs the doubling steps only once per iteration and includes a pseudo gradient to amplify steps in the direction where the error decreases the most. In the iterative search, the doubling steps are performed 2*n* times, once for each dimension of two vectors. The pseudo gradient search is clearly an improvement over the iterative search.

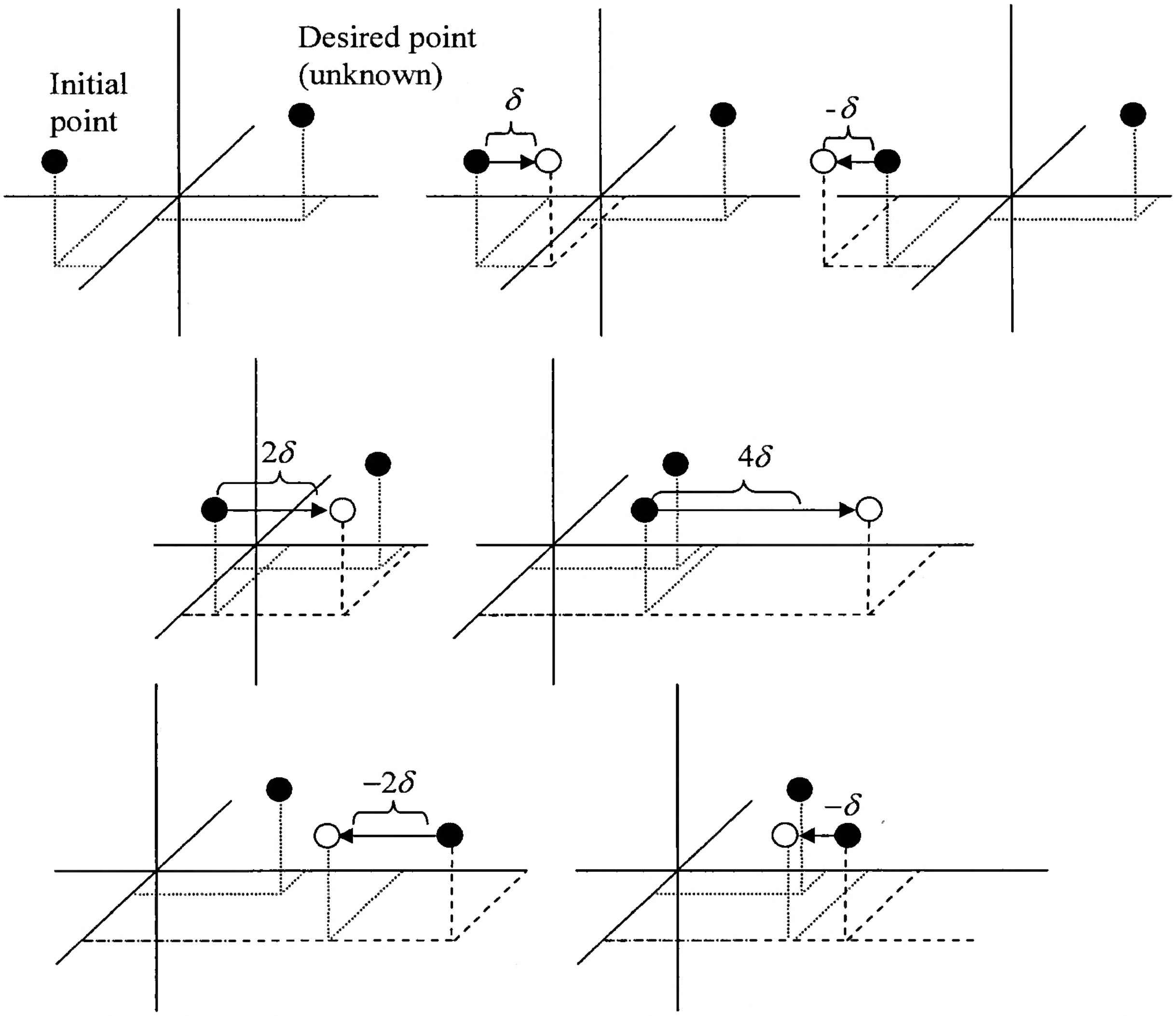


Figure 18. The iterative search algorithm examines one dimension at a time and performs all doubling and halving steps on the current dimension. Once the halving steps are complete, the dimension of the change vector is determined. In this example, the first dimension of the change vector is $\delta + 2\delta + 4\delta - 2\delta - \delta = 4\delta$.

EXAMPLES

The program has been coded in Java and it runs on a laptop computer. Four examples are presented. The first example involves artificial data while the next three examples involve real data.

Example 1

An artificial data set with three attributes was created using the multiregression formula. The following parameters were used in the computer program and the results are listed in table 3: $\lambda = 14$, $p = 200$, $\varepsilon = 10^{-10}$, $\delta = 10^{-6}$, and $w = 5$. All of the

Variable	Actual Values	Following Genetic Algorithm	Following Pseudo gradient search
a_1	0	0	0.03198
a_2	0.8	0.85032	0.83196
a_3	0.4	0.44368	0.43197
b_1	0.5	0.51983	0.49999
b_2	0.8	0.79546	0.79997
b_3	1.0	1.0	0.99997
c	4.0	3.96362	3.96801
$\mu(\{\})$	0	0	0
$\mu(\{x_1\})$	0.1	0.28884	0.10000
$\mu(\{x_2\})$	0.2	0.50037	0.20000
$\mu(\{x_1, x_2\})$	0.4	0.69185	0.40001
$\mu(\{x_3\})$	0.3	0.59706	0.30000
$\mu(\{x_1, x_3\})$	0.5	0.79014	0.50001
$\mu(\{x_2, x_3\})$	0.7	0.89590	0.70002
$\mu(\{x_1, x_2, x_3\})$	1.0	0.99173	1.00003

Table 3. Results from using the algorithm on an artificial data set. The actual parameters used to create the data are listed in the second column. The fourth column is the final results from the computer program.

variables were reasonably reproduced using the algorithm. Vector *a* and variable *c* had values that were furthest from the actual values used to create the data set. The total running time of the program is 19.8 seconds. The complete data set is listed in appendix D.

Example 2

An appropriate multiregression problem is to estimate the gross domestic product of a country by using the worth of a country's fixed assets, the total number of people in the labor force, and the number of people who pursued higher education. Nineteen observations during the last century were taken as the input. The attribute of fixed assets was represented in billion dollars, and the labor force and higher education attributes were each represented as millions of people. The complete data set is given in appendix E.

The following parameters were used in the computer program: $\lambda = 14$, $p = 200$, $\varepsilon = 10^{-10}$, $\delta = 10^{-6}$, and $w = 5$. Seven thousand two hundred chromosomes were generated in the genetic algorithm portion of the program. Comparing the error at the end of the first generation and the error in the final generation or the genetic algorithm resulted in an error reduction of 72%. The pseudo gradient search portion of the program resulted in an additional 7% improvement. The genetic algorithm took 10.6 seconds to process. The pseudo gradient search took an additional 3 minutes and 21 seconds.

Next, the objective attributes in the data set were compared against the estimated objective attributes which were calculated using the values in table four (see table 5). Though the data size is not sufficiently large, we can see that the method provided in this paper is efficient for solving nonlinear multiregression problems based on the generalized Choquet integral. A valid criticism of this example is that the algorithm may have over fitted the data set because of its small size. A way to ensure

Table 4. Results from using the algorithm on the real data set given in appendix E.

Variable	Following Genetic Algorithm	Following Pseudo gradient search
a_1	1.22892	1.22956
a_2	0	0.00007
a_3	909.08932	907.84772
b_1	0.61832	0.62105
b_2	0.38195	0.38250
b_3	1.0	1.00000
c	-111809.741	-114342.666
$\mu(\{\ \})$	0	0
$\mu(\{x_1\})$	1.46917	1.59508
$\mu(\{x_2\})$	-18.21820	-17.49393
$\mu(\{x_1, x_2\})$	3.22435	1.59780
$\mu(\{x_3\})$	124.92529	127.77463
$\mu(\{x_1, x_3\})$	0	0
$\mu(\{x_2, x_3\})$	117.25948	120.10339
$\mu(\{x_1, x_2, x_3\})$	119.12952	121.87264

Table 5. Comparing the true objective attributes from the data set against the objective attributes produced from the computer program.

Objective Attribute from Data Set (billion dollars)	Estimated Objective Attribute (billion dollars)
322.07	332.34
427.50	424.20
500.06	484.03
603.71	600.33
765.76	728.15
843.72	869.74
897.99	918.81
1081.75	1074.39
1365.06	1357.39
1909.49	2006.52
2666.86	2560.63
3524.79	3545.19
4146.06	4222.11
4638.24	4544.64
4987.50	4985.40
5364.89	5392.50
6036.34	6033.08
6748.15	6765.85

the multiregression formula produced by the algorithm is correct is to use a portion of the data set as a training set and keep a portion of the data set as a validation set. This technique was done in the next example.

Example 3

The statistics division of the United Nations keeps track of several databases of information. The UN allows unrestricted access to certain databases which it has compiled into readable formats referred to as yearbooks. One of the yearbooks is the *Demographic Yearbook 2001* which contains demographic statistics such as population trends, birth rates, mortality rates, marriage rates, and divorce rates for over 230 countries in the world.

One of the tables in the yearbook is titled "Vital statistics summary and expectation of life at birth: 1997-2001". The table presents the following vital statistics for the many countries listed: number of live births, crude birth rate, number of deaths, crude death rate, rate of natural increase, number of infant deaths, infant mortality rate, life expectancy by gender, and fertility rate. (The definition of some of these terms is given in appendix F along with the data set that was used.) The table has missing data; for example, Saudi Arabia has data for the years 1999 and 2000, but not for 1997, 1998, or 2001. As another example, Argentina has data for 1997-2001, but it is missing the values for life expectancy. Due to the number of missing entries, the table needed to be preprocessed. To avoid the problem of confounding variables, only one year from the range 1997-2001 was processed. The most data was available for the year 1997, so all other years were removed from the table. The attributes of crude birth rate, infant death rate, and the rate of natural increase were chosen as the predictive attributes and the life expectancy of males at birth was chosen as the objective attribute. Any country that did not have complete data for each of those attributes was removed. Forty-two countries remained. Eighty percent of the countries (34) were used as a training set and twenty percent of the countries (8) were used as a validation set. Each country was numbered from one to forty-two and a computer program randomly generated eight distinct numbers in that range. The eight countries were then used as the validation set. The results of running the training set through the process are present in tables 6 and 7 of appendix F.

The results are extremely encouraging. The farthest that any of the estimated objective attributes is off from any of the true objective attributes is 3.21 years. The average difference is 1.17 years. Next, the validation set was used to see how well the results generalize. As table eight demonstrates, the validation set also had excellent results. The maximum change from the estimated objective attribute is 4.22 years and the average change was 1.42 years.

Objective Attribute from Data Set (years)	**Estimated Objective Attribute (years)**
62.74	62.70
75.45	76.39
74.86	74.09
75.31	71.36
74.29	73.95
77.19	76.27
73.6	73.35
58.0	62.22

Table 8. Comparing the estimated objective attribute of the validation set against the true objective attribute from the data set.

Example 4

All three of the previous examples were run on small data sets with three attributes. This example involves a large data set with eight attributes.

The abalone is a single shelled, ocean-dwelling creature. Scientists who wish to know the age of an abalone, must cut through its shell, stain the shell, view the stained shell under a microscope, and count the number of visible rings [10]. This task is time consuming. Having a mathematical way to accurately estimate the age of an abalone would be useful.

Scientists in the Marine Resources Division of Tasmania Australia shared a 4,177 instance database with the University of California-Irvine. The database has unrestricted access and can be viewed at the website listed in the references section of this document. The database contains the following attributes: sex (male, female, infant), length (mm), diameter (mm), height (mm), whole weight (grams), shucked weight (grams), viscera weight (grams), shell weight (grams), and rings (integer). Missing entries have been removed from the data set already. The only preprocessing that was done to the data set was the nominal attribute of sex was given numerical values by representing infant as 1, female as 2 and male as 3.

The entire data set was initially given as input to the program. When the first iteration of the genetic algorithm took over a half hour to complete, the computer program was terminated and the size of the data set given to the program was shortened by separating the data into a training set that contained 3,342 observations (80%), and a validation set which had 835 observations (20%). The following parameters were used in the computer program: $\lambda = 14$, $p = 200$, $\varepsilon = 10^{-10}$, $\delta = 10^{-6}$, and $w = 6$. The program took an undesirable 36 hours and 47 minutes to process the training set on a computer with a 1.70 GHz processor and 760 MB of RAM. Of the total processing time, 36 hours and 24 minutes was devoted to the genetic algorithm and 23 minutes was used by the pseudo gradient search. The final calculation of the root mean square error on the training set was 1.968941.

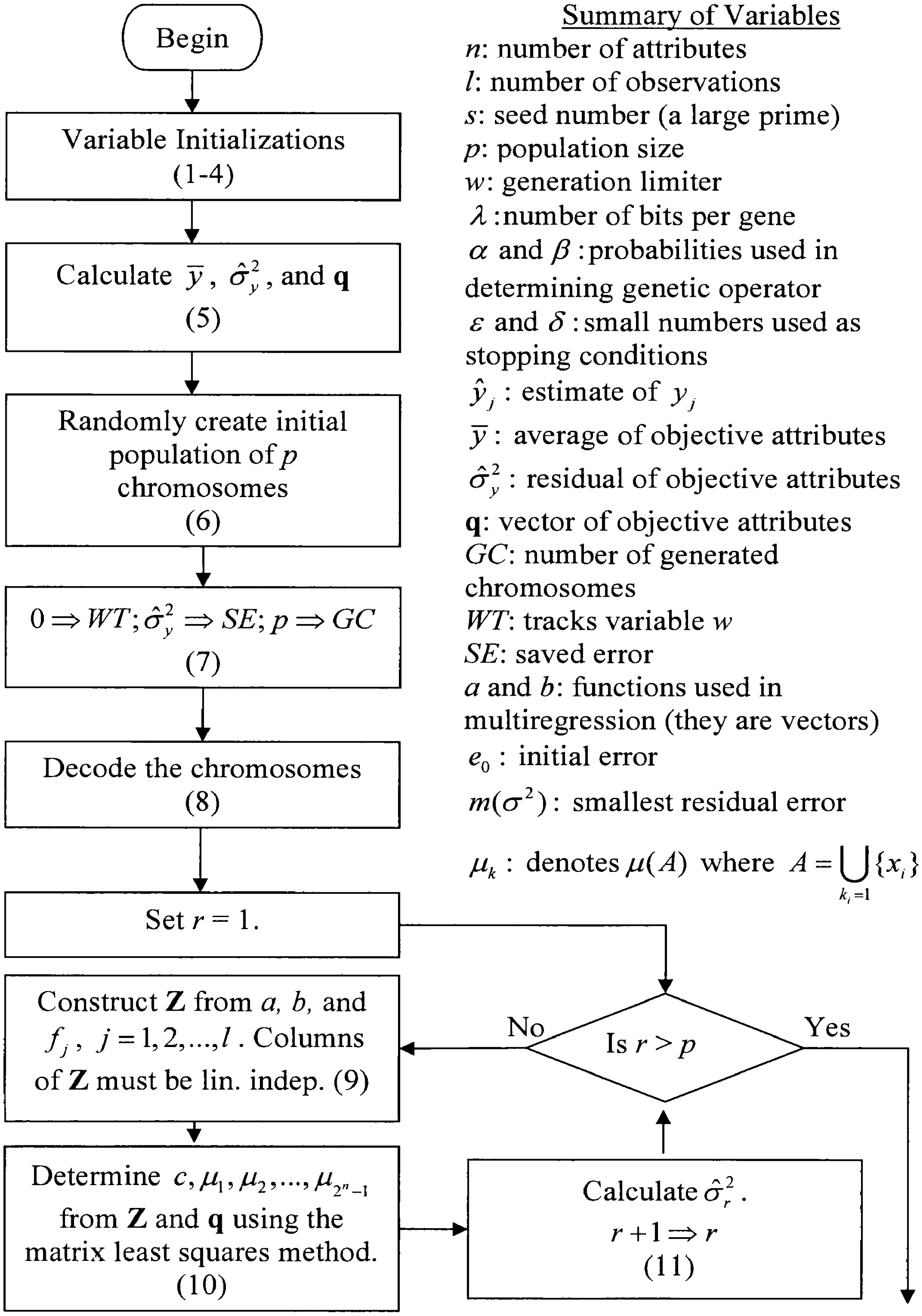
Next, the results of the program were used to test the validation set. The 168th validation set observation resulted in a ring estimate of -31.15527245. Careful examination of the parameters involved in the calculation for this observation revealed that $\mu_3 = -3234.905507$. Presumably, the training set did not contain enough observations for μ_3 to achieve a more accurate result because μ_3 was only used once during the entire time the validation set was processed. Removing this observation from the validation set yields a root mean square error of 2.31693. The results are reasonable because the root mean square error of the training set is close to the value of the root mean square error from the validation set, however a final determination of the usefulness of the results would need to be made by an expert in the field.

CONCLUSION

Though the complete algorithm shows great promise, much work still must be done to reduce the complexity. As was demonstrated by the fourth example, using this algorithm to generate the multiregression parameters from a data set with more than five attributes will require some waiting given the computing power available at the time of this writing. It also must be stressed that this algorithm cannot be guaranteed to locate the optimal solution to a problem. This algorithm can only provide a useful prediction of what an optimal solution may be and often this estimate is sufficient.

How can the algorithm be improved? My first suggestion would be to make the genetic portion of the algorithm adaptive. The most significant bits in the genes would be altered in the initial stages to cause wider genetic variation. In subsequent generations, bits of less significance would be altered. My second suggestion is to consider small implementation changes in the pseudo gradient search. For example, once the root mean square error grows between iterations in the doubling stage of the change vector, it may be more beneficial to perform *another* pseudo gradient search from the current position rather than perform all of the halving steps. My final suggestion is to have the algorithm automatically determine which attributes are necessary for processing a large data set. When I preprocessed the United Nations data set in example 3, it was pure luck that I chose three attributes that produced such excellent results. In fact, choosing a different set of three attributes may produce a result that is even better than what I found, but since there are 35 ways to choose a set of 3 attributes from a full sct of 7 attributes, the amount of time it would take to prepare 35 different input files would take days due to the incompleteness of the data.

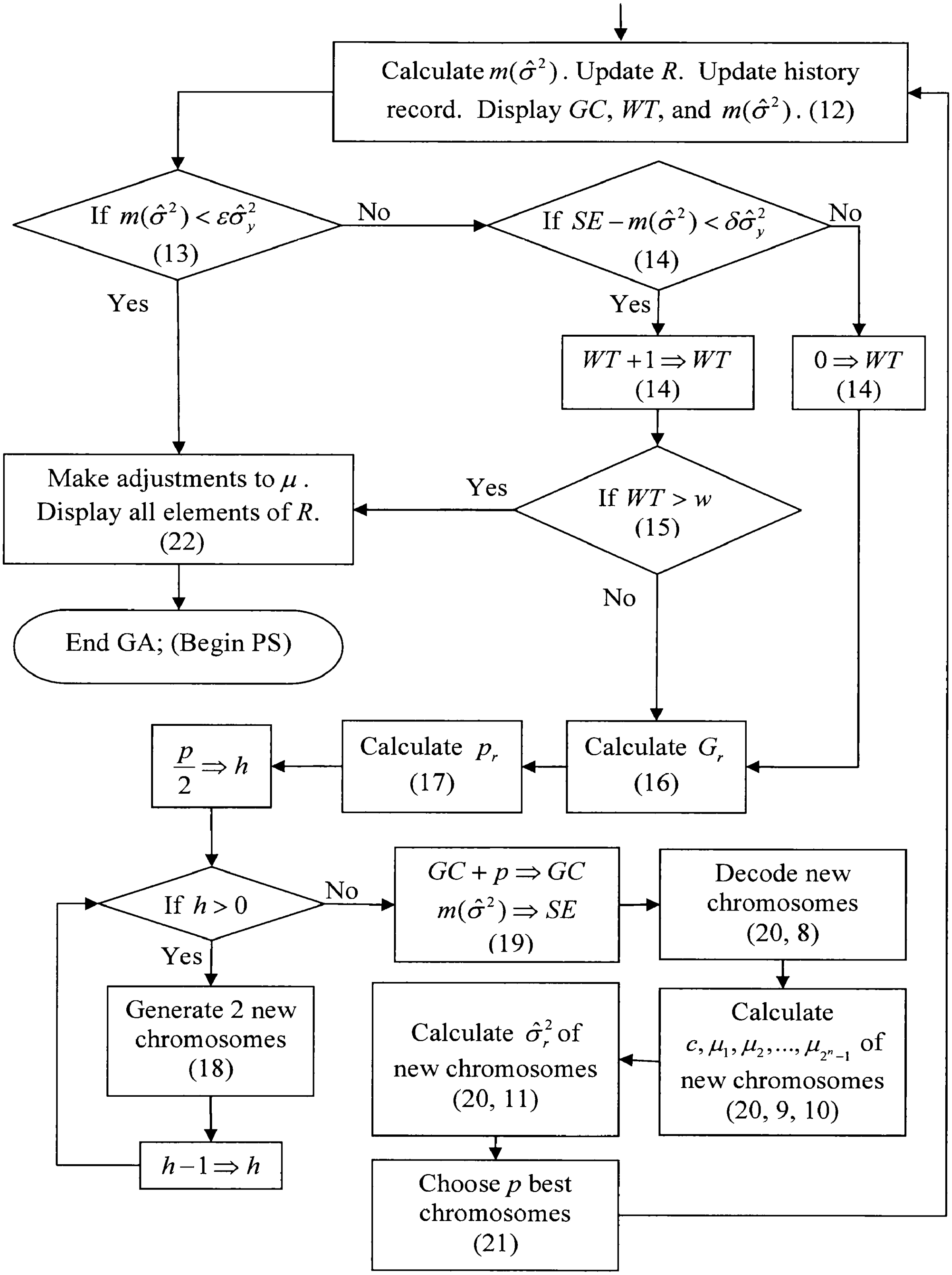
APPENDIX A


Begin
Variable Initializations
(1-4)
Calculate $\bar{y}$, $\hat{\sigma}_y^2$, and $\mathbf{q}$
(5)
Randomly create initial population of p chromosomes
(6)
$0 \Rightarrow WT; \hat{\sigma}_y^2 \Rightarrow SE; p \Rightarrow GC$
(7)
Decode the chromosomes
(8)
Set $r = 1$.
Is $r > p$
No
Yes
Construct $\mathbf{Z}$ from a, b, and f_j, $j = 1, 2, ..., l$. Columns of $\mathbf{Z}$ must be lin. indep. (9)
Determine $c, \mu_1, \mu_2, ..., \mu_{2^n-1}$ from $\mathbf{Z}$ and $\mathbf{q}$ using the matrix least squares method.
(10)
Calculate $\hat{\sigma}_r^2$.
$r+1 \Rightarrow r$
(11)

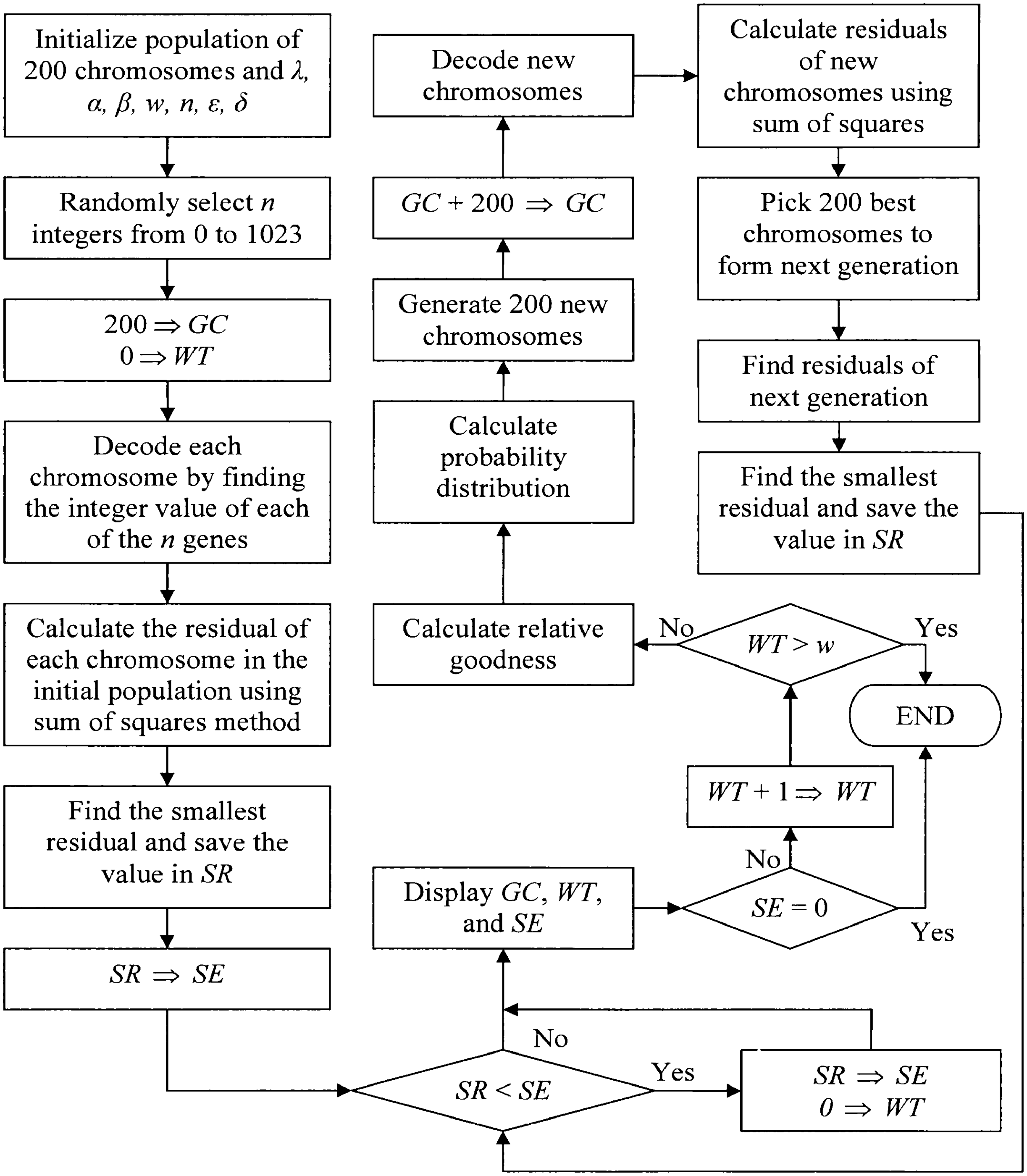
Summary of Variables

n: number of attributes
l: number of observations
s: seed number (a large prime)
p: population size
w: generation limiter
λ :number of bits per gene
α and β :probabilities used in determining genetic operator
ε and δ : small numbers used as stopping conditions
$\hat{y}_j$: estimate of y_j
$\bar{y}$: average of objective attributes
$\hat{\sigma}_y^2$: residual of objective attributes
$\mathbf{q}$: vector of objective attributes
GC: number of generated chromosomes
WT: tracks variable w
SE: saved error
a and b: functions used in multiregression (they are vectors)
e_0 : initial error
$m(\sigma^2)$: smallest residual error

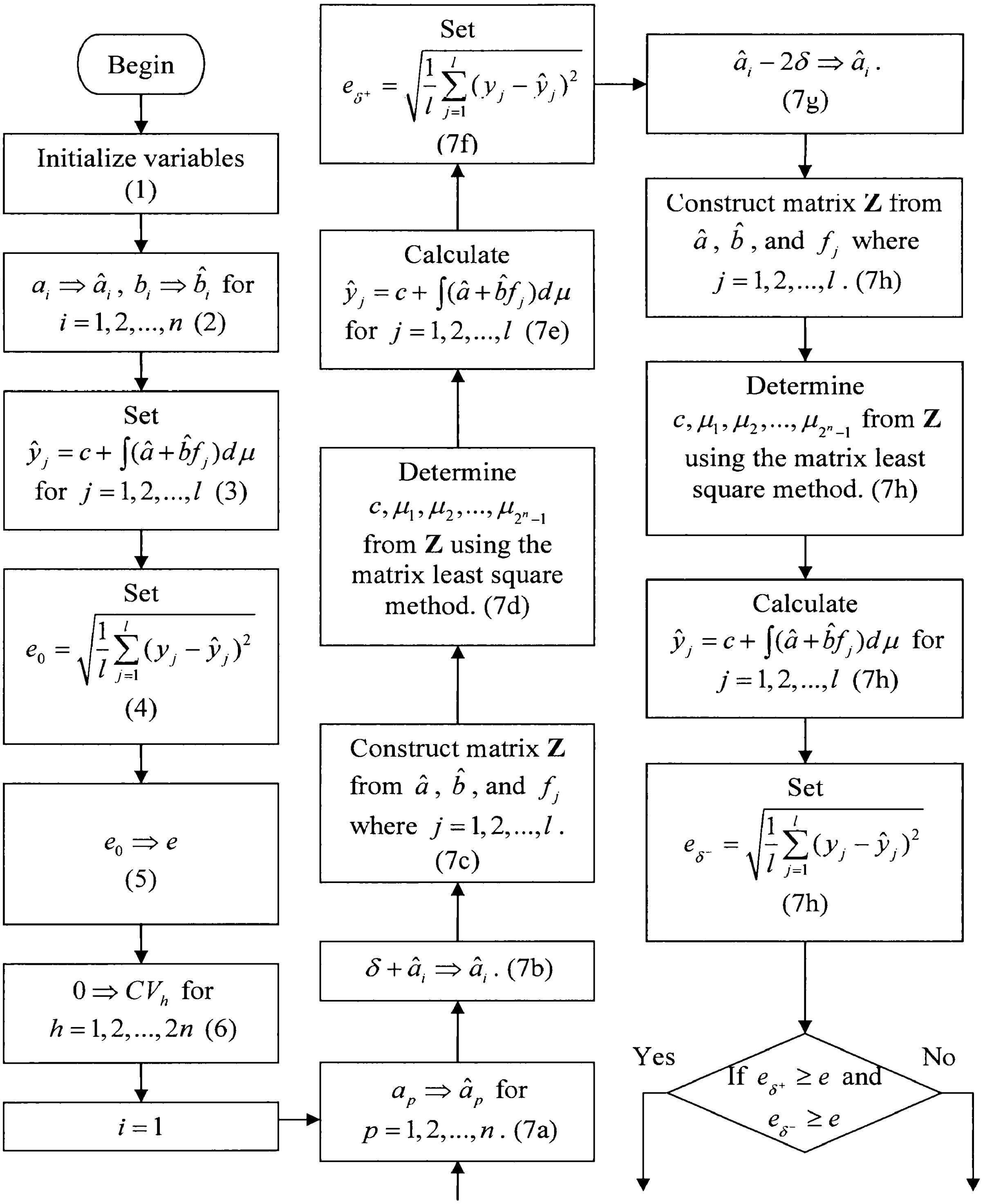
μ_k : denotes $\mu(A)$ where $A = \bigcup_{k_i=1} \{x_i\}$

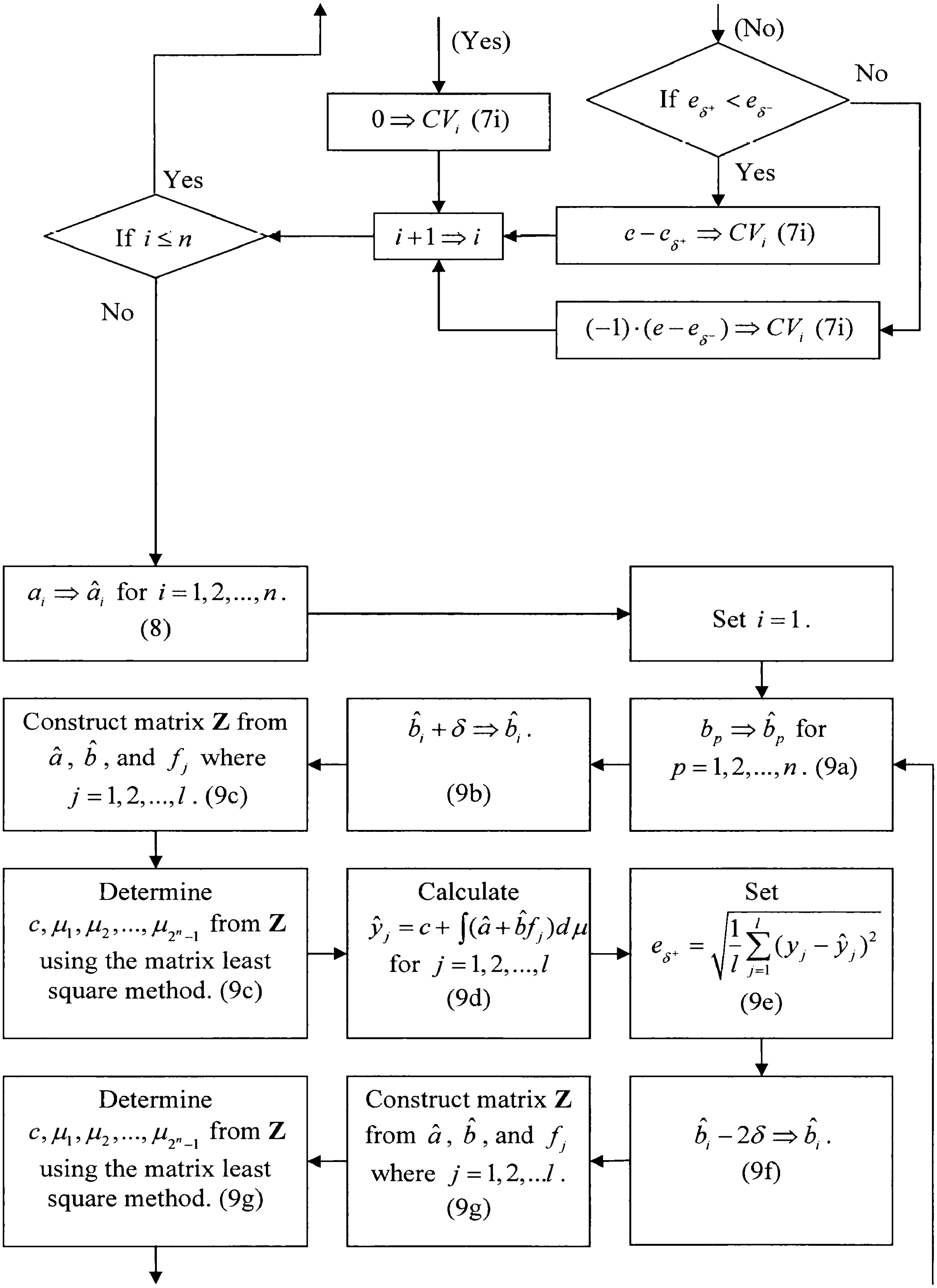

Calculate $m(\hat{\sigma}^2)$. Update R. Update history record. Display GC, WT, and $m(\hat{\sigma}^2)$. (12)
If $m(\hat{\sigma}^2) < \varepsilon\hat{\sigma}_y^2$ (13)
No
If $SE - m(\hat{\sigma}^2) < \delta\hat{\sigma}_y^2$ (14)
No
Yes
Yes
$WT + 1 \Rightarrow WT$ (14)
$0 \Rightarrow WT$ (14)
Make adjustments to μ. Display all elements of R. (22)
Yes
If $WT > w$ (15)
End GA; (Begin PS)
No
$\frac{p}{2} \Rightarrow h$
Calculate p_r (17)
Calculate G_r (16)
If $h > 0$
No
$GC + p \Rightarrow GC$ $m(\hat{\sigma}^2) \Rightarrow SE$ (19)
Decode new chromosomes (20, 8)
Yes
Generate 2 new chromosomes (18)
Calculate $\hat{\sigma}_r^2$ of new chromosomes (20, 11)
Calculate $c, \mu_1, \mu_2, \ldots, \mu_{2^n-1}$ of new chromosomes (20, 9, 10)
$h - 1 \Rightarrow h$
Choose p best chromosomes (21)

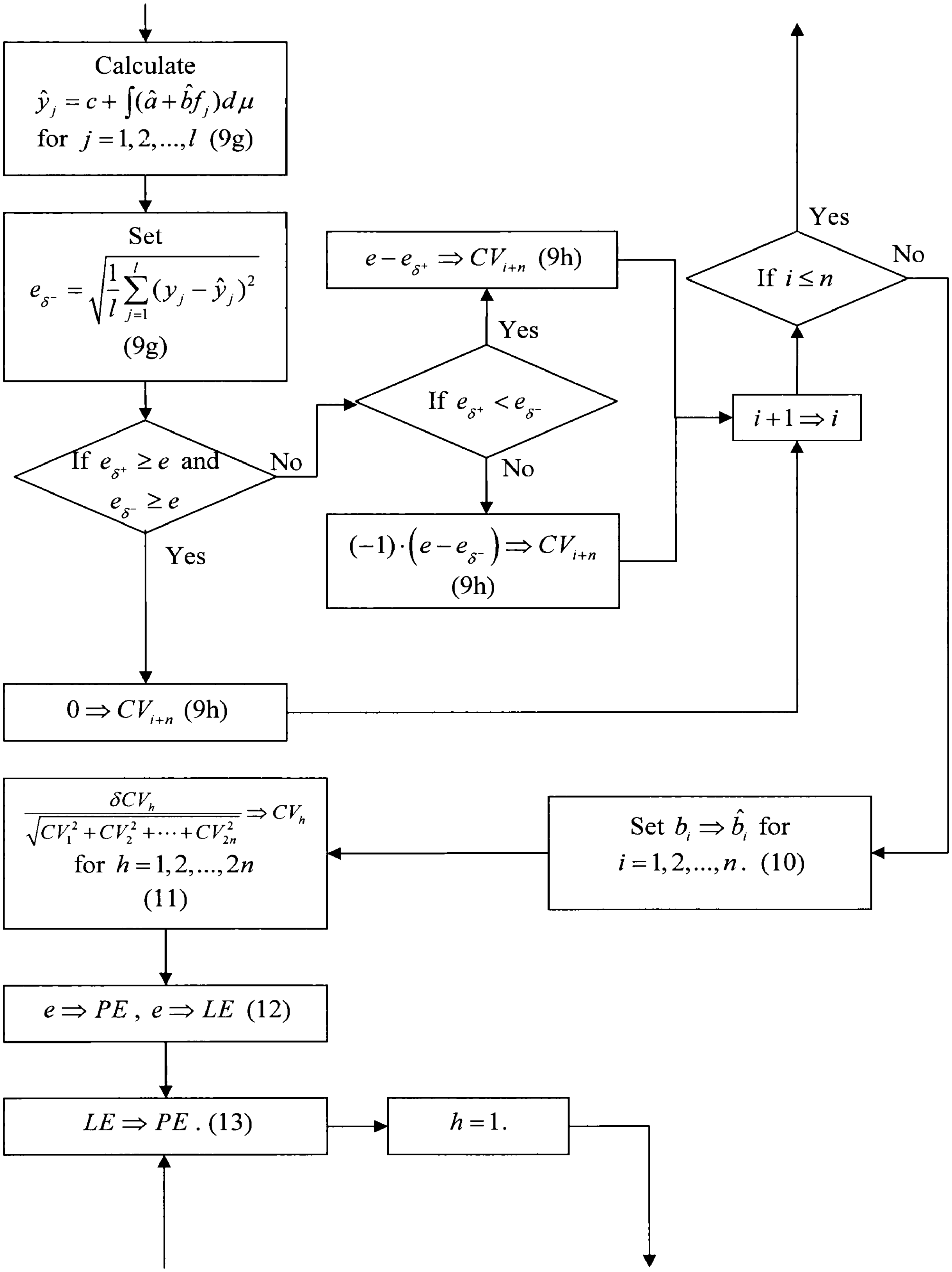
APPENDIX B

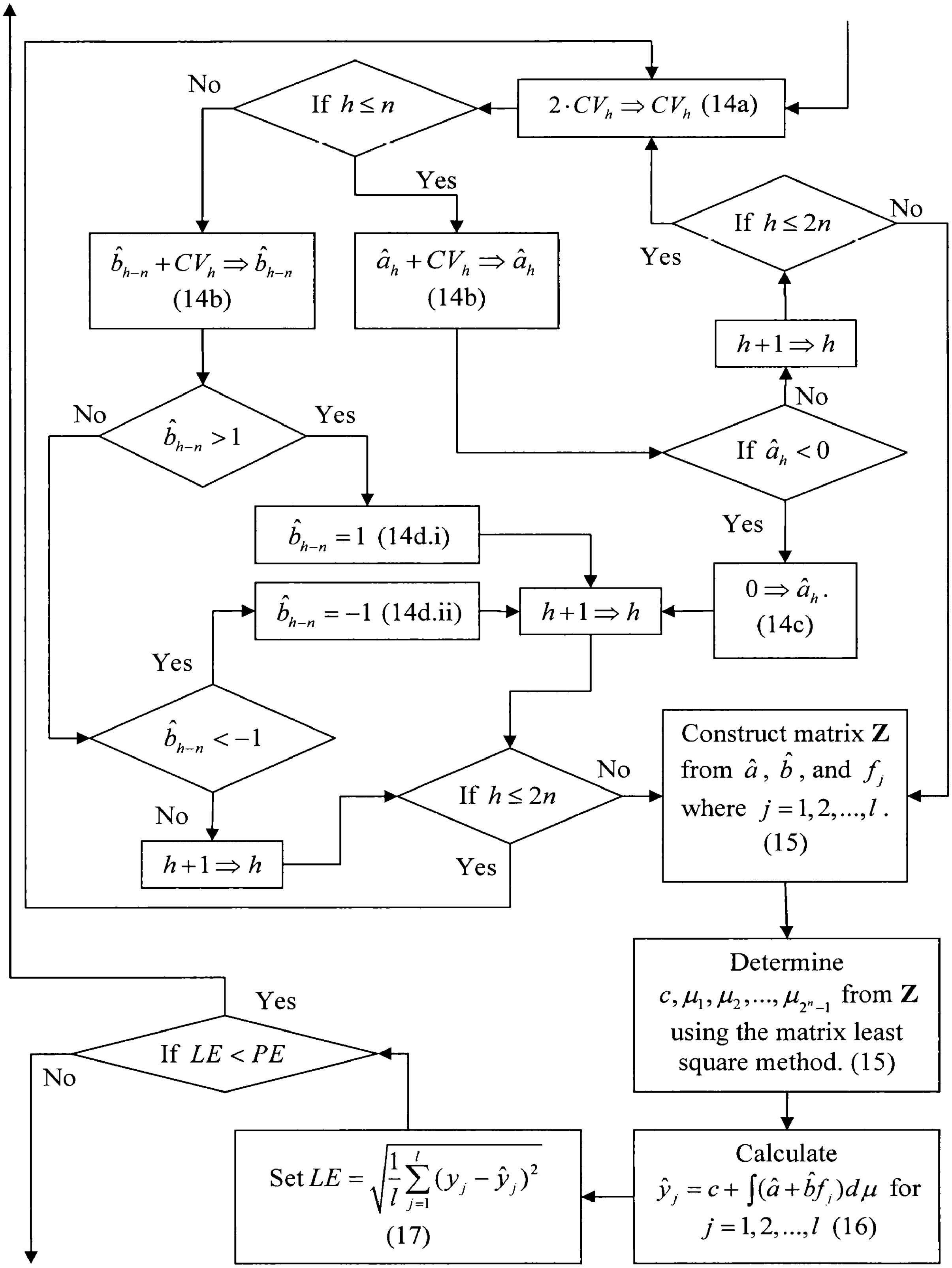

Initialize population of 200 chromosomes and λ, α, β, w, n, ε, δ
Randomly select n integers from 0 to 1023
200 ⇒ GC
0 ⇒ WT
Decode each chromosome by finding the integer value of each of the n genes
Calculate the residual of each chromosome in the initial population using sum of squares method
Find the smallest residual and save the value in SR
SR ⇒ SE
SR < SE
No
Yes
SR ⇒ SE
0 ⇒ WT
Display GC, WT, and SE
SE = 0
No
Yes
WT + 1 ⇒ WT
WT > w
No
Yes
END
Calculate relative goodness
Calculate probability distribution
Generate 200 new chromosomes
GC + 200 ⇒ GC
Decode new chromosomes
Calculate residuals of new chromosomes using sum of squares
Pick 200 best chromosomes to form next generation
Find residuals of next generation
Find the smallest residual and save the value in SR

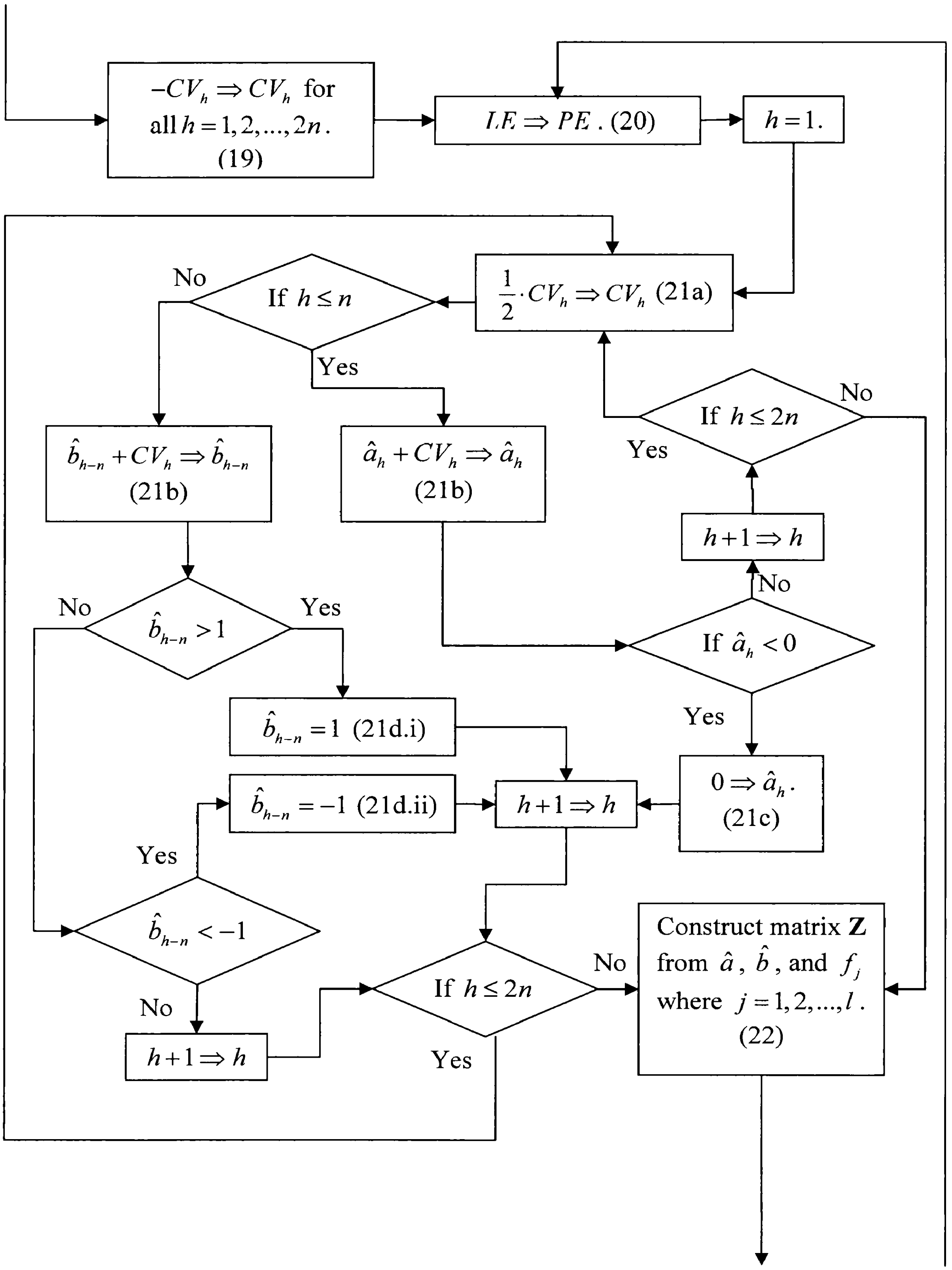
APPENDIX C

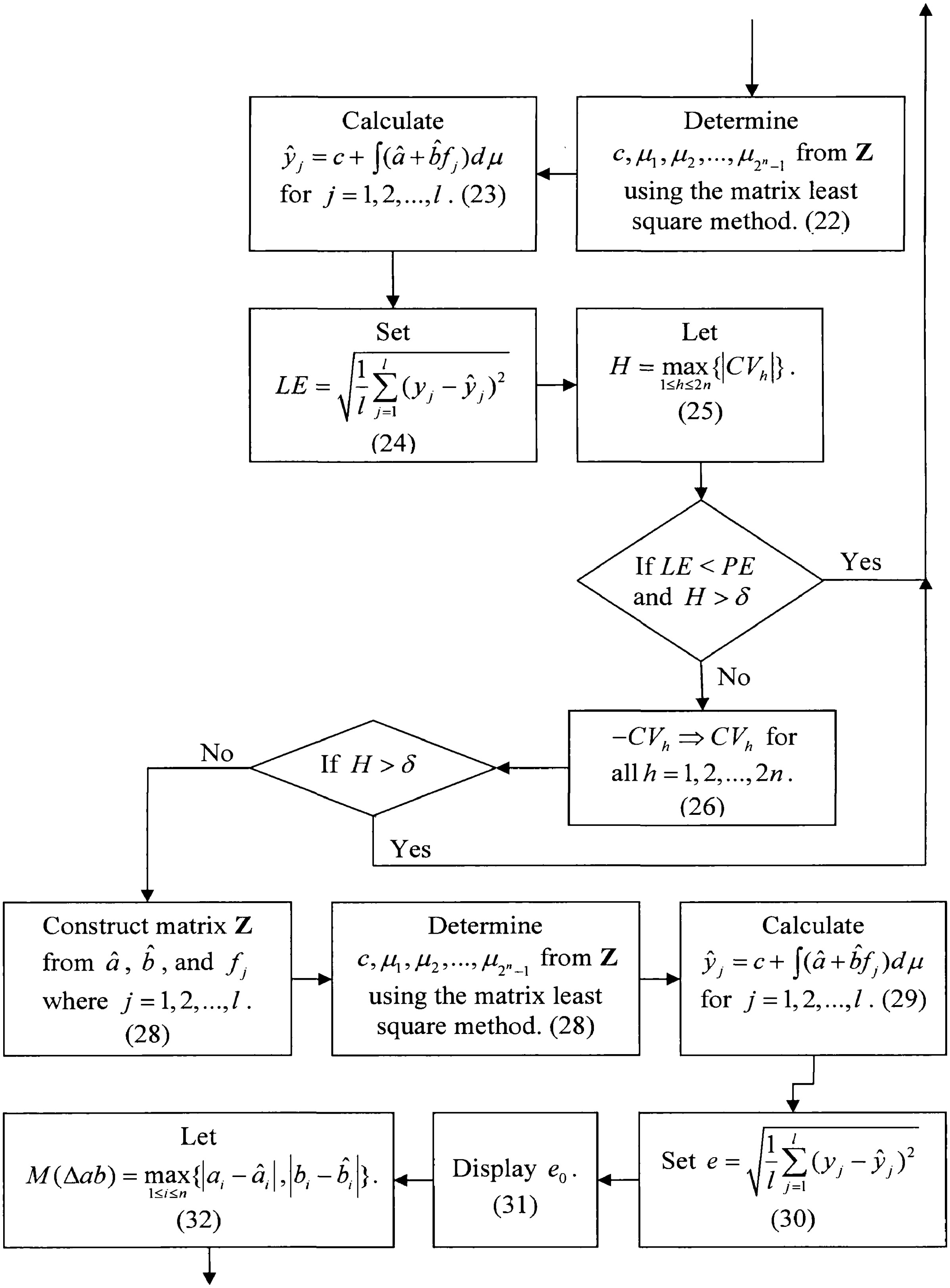

Begin
Initialize variables (1)
$a_i \Rightarrow \hat{a}_i$, $b_i \Rightarrow \hat{b}_i$ for $i = 1,2,\ldots,n$ (2)
Set $\hat{y}_j = c + \int(\hat{a} + \hat{b}f_j)d\mu$ for $j = 1,2,\ldots,l$ (3)
Set $e_0 = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(y_j - \hat{y}_j)^2}$ (4)
$e_0 \Rightarrow e$ (5)
$0 \Rightarrow CV_h$ for $h = 1,2,\ldots,2n$ (6)
$i = 1$
$a_p \Rightarrow \hat{a}_p$ for $p = 1,2,\ldots,n$. (7a)
$\delta + \hat{a}_i \Rightarrow \hat{a}_i$. (7b)
Construct matrix **Z** from $\hat{a}$, $\hat{b}$, and f_j where $j = 1,2,\ldots,l$. (7c)
Determine $c, \mu_1, \mu_2, \ldots, \mu_{2^n-1}$ from **Z** using the matrix least square method. (7d)
Calculate $\hat{y}_j = c + \int(\hat{a} + \hat{b}f_j)d\mu$ for $j = 1,2,\ldots,l$ (7e)
Set $e_{\delta^+} = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(y_j - \hat{y}_j)^2}$ (7f)
$\hat{a}_i - 2\delta \Rightarrow \hat{a}_i$. (7g)
Construct matrix **Z** from $\hat{a}$, $\hat{b}$, and f_j where $j = 1,2,\ldots,l$. (7h)
Determine $c, \mu_1, \mu_2, \ldots, \mu_{2^n-1}$ from **Z** using the matrix least square method. (7h)
Calculate $\hat{y}_j = c + \int(\hat{a} + \hat{b}f_j)d\mu$ for $j = 1,2,\ldots,l$ (7h)
Set $e_{\delta^-} = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(y_j - \hat{y}_j)^2}$ (7h)
If $e_{\delta^+} \geq e$ and $e_{\delta^-} \geq e$
Yes
No

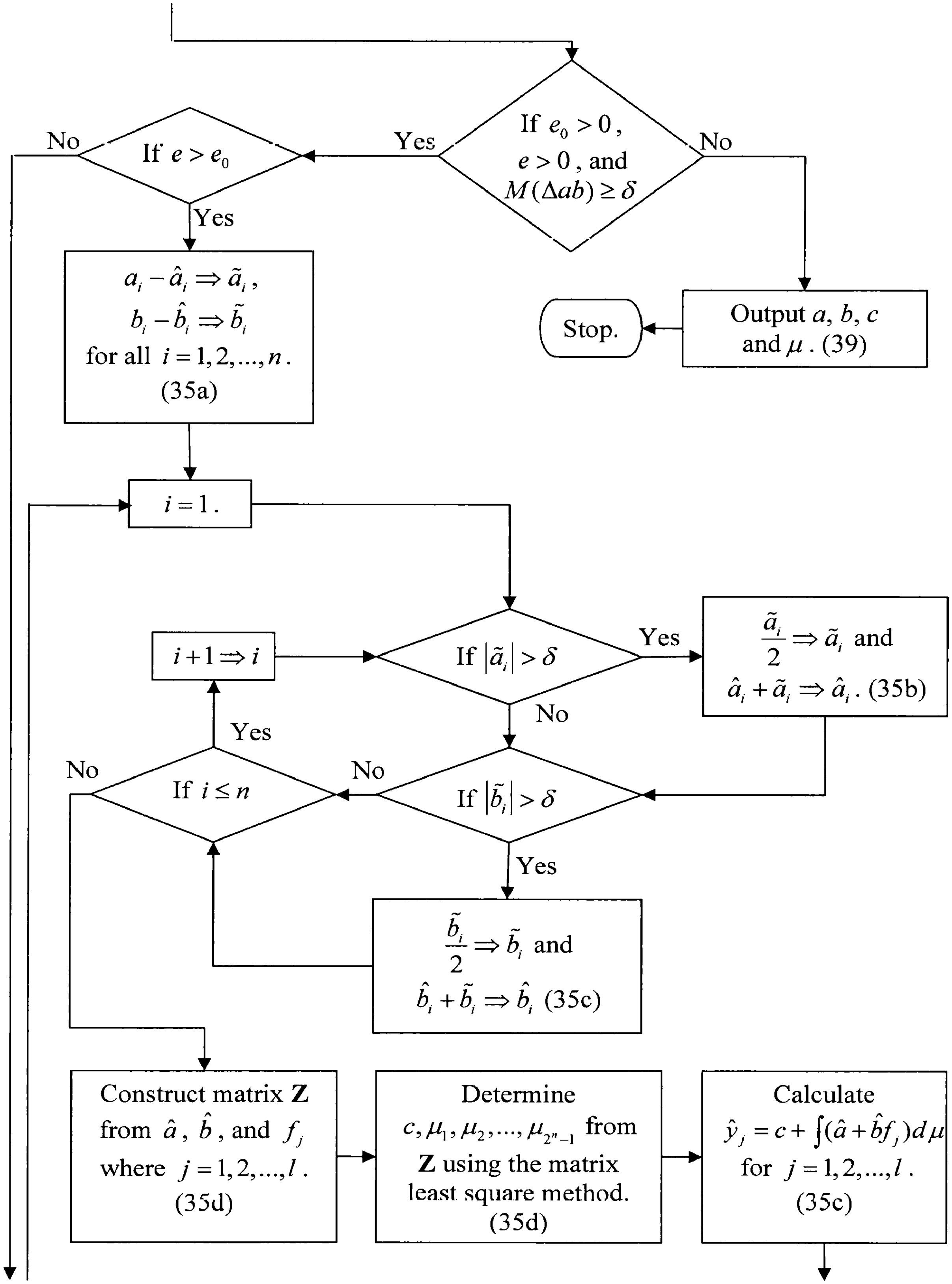

(Yes)
(No)
No
If $e_{\delta^+} < e_{\delta^-}$
$0 \Rightarrow CV_i$ (7i)
Yes
Yes
If $i \le n$
$i+1 \Rightarrow i$
$c - e_{\delta^+} \Rightarrow CV_i$ (7i)
No
$(-1)\cdot(e - e_{\delta^-}) \Rightarrow CV_i$ (7i)
$a_i \Rightarrow \hat{a}_i$ for $i = 1,2,...,n$. (8)
Set $i = 1$.
Construct matrix **Z** from $\hat{a}$, $\hat{b}$, and f_j where $j = 1,2,...,l$. (9c)
$\hat{b}_i + \delta \Rightarrow \hat{b}_i$. (9b)
$b_p \Rightarrow \hat{b}_p$ for $p = 1,2,...,n$. (9a)
Determine $c, \mu_1, \mu_2, ..., \mu_{2^n-1}$ from **Z** using the matrix least square method. (9c)
Calculate $\hat{y}_j = c + \int(\hat{a} + \hat{b}f_j)d\mu$ for $j = 1,2,...,l$ (9d)
Set $e_{\delta^+} = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(y_j - \hat{y}_j)^2}$ (9e)
Determine $c, \mu_1, \mu_2, ..., \mu_{2^n-1}$ from **Z** using the matrix least square method. (9g)
Construct matrix **Z** from $\hat{a}$, $\hat{b}$, and f_j where $j = 1,2,...l$. (9g)
$\hat{b}_i - 2\delta \Rightarrow \hat{b}_i$. (9f)

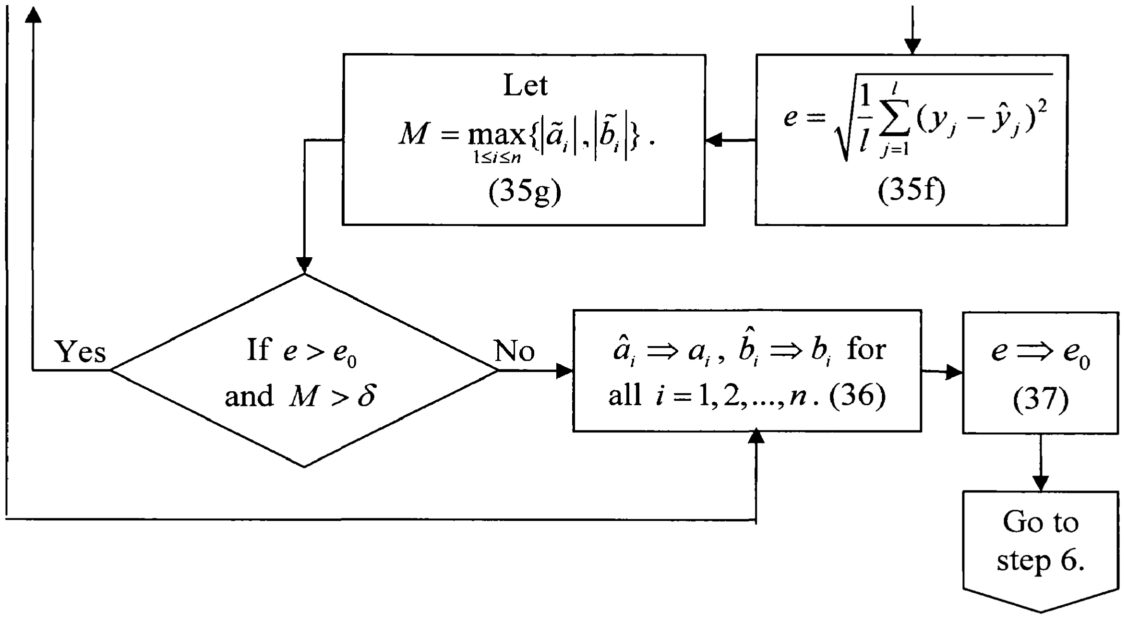

Calculate $\hat{y}_j = c + \int(\hat{a} + \hat{b}f_j)d\mu$ for $j = 1, 2, \ldots, l$ (9g)
Set $e_{\delta^-} = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(y_j - \hat{y}_j)^2}$ (9g)
If $e_{\delta^+} \geq e$ and $e_{\delta^-} \geq e$
No
Yes
If $e_{\delta^+} < e_{\delta^-}$
Yes
No
$e - e_{\delta^+} \Rightarrow CV_{i+n}$ (9h)
$(-1)\cdot\left(e - e_{\delta^-}\right) \Rightarrow CV_{i+n}$ (9h)
$0 \Rightarrow CV_{i+n}$ (9h)
$i + 1 \Rightarrow i$
If $i \leq n$
Yes
No
Set $b_i \Rightarrow \hat{b}_i$ for $i = 1, 2, \ldots, n$. (10)
$\frac{\delta CV_h}{\sqrt{CV_1^2 + CV_2^2 + \cdots + CV_{2n}^2}} \Rightarrow CV_h$ for $h = 1, 2, \ldots, 2n$ (11)
$e \Rightarrow PE$, $e \Rightarrow LE$ (12)
$LE \Rightarrow PE$. (13)
$h = 1$.


$2 \cdot CV_h \Rightarrow CV_h$ (14a)
If $h \le n$
No
Yes
$\hat{b}_{h-n} + CV_h \Rightarrow \hat{b}_{h-n}$ (14b)
$\hat{a}_h + CV_h \Rightarrow \hat{a}_h$ (14b)
If $h \le 2n$
Yes
No
$h+1 \Rightarrow h$
No
If $\hat{a}_h < 0$
Yes
$0 \Rightarrow \hat{a}_h$. (14c)
No
$\hat{b}_{h-n} > 1$
Yes
$\hat{b}_{h-n} = 1$ (14d.i)
$\hat{b}_{h-n} = -1$ (14d.ii)
$h+1 \Rightarrow h$
Yes
$\hat{b}_{h-n} < -1$
No
$h+1 \Rightarrow h$
If $h \le 2n$
No
Yes
Construct matrix **Z** from $\hat{a}$, $\hat{b}$, and f_j where $j = 1,2,\ldots,l$. (15)
Determine $c, \mu_1, \mu_2, \ldots, \mu_{2^n-1}$ from **Z** using the matrix least square method. (15)
Calculate $\hat{y}_j = c + \int(\hat{a} + \hat{b}f_j)d\mu$ for $j = 1,2,\ldots,l$ (16)
Set $LE = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(y_j - \hat{y}_j)^2}$ (17)
If $LE < PE$
Yes
No


$-CV_h \Rightarrow CV_h$ for all $h = 1, 2, \ldots, 2n$. (19)
$LE \Rightarrow PE$. (20)
$h = 1$.
$\frac{1}{2} \cdot CV_h \Rightarrow CV_h$ (21a)
If $h \leq n$
No
Yes
$\hat{b}_{h-n} + CV_h \Rightarrow \hat{b}_{h-n}$ (21b)
$\hat{a}_h + CV_h \Rightarrow \hat{a}_h$ (21b)
If $h \leq 2n$
No
Yes
$h + 1 \Rightarrow h$
No
$\hat{b}_{h-n} > 1$
No
Yes
If $\hat{a}_h < 0$
Yes
$\hat{b}_{h-n} = 1$ (21d.i)
$\hat{b}_{h-n} = -1$ (21d.ii)
$h + 1 \Rightarrow h$
$0 \Rightarrow \hat{a}_h$. (21c)
Yes
$\hat{b}_{h-n} < -1$
No
$h + 1 \Rightarrow h$
If $h \leq 2n$
No
Yes
Construct matrix $\mathbf{Z}$ from $\hat{a}$, $\hat{b}$, and f_j where $j = 1, 2, \ldots, l$. (22)


Determine
$c, \mu_1, \mu_2, \ldots, \mu_{2^n-1}$ from **Z**
using the matrix least
square method. (22)
Calculate
$\hat{y}_j = c + \int(\hat{a} + \hat{b}f_j)d\mu$
for $j = 1, 2, \ldots, l$. (23)
Set
$LE = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(y_j - \hat{y}_j)^2}$
(24)
Let
$H = \max_{1 \le h \le 2n}\{|CV_h|\}$.
(25)
If $LE < PE$
and $H > \delta$
Yes
No
$-CV_h \Rightarrow CV_h$ for
all $h = 1, 2, \ldots, 2n$.
(26)
If $H > \delta$
No
Yes
Construct matrix **Z**
from $\hat{a}$, $\hat{b}$, and f_j
where $j = 1, 2, \ldots, l$.
(28)
Determine
$c, \mu_1, \mu_2, \ldots, \mu_{2^n-1}$ from **Z**
using the matrix least
square method. (28)
Calculate
$\hat{y}_j = c + \int(\hat{a} + \hat{b}f_j)d\mu$
for $j = 1, 2, \ldots, l$. (29)
Set $e = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(y_j - \hat{y}_j)^2}$
(30)
Display e_0.
(31)
Let
$M(\Delta ab) = \max_{1 \le i \le n}\{|a_i - \hat{a}_i|, |b_i - \hat{b}_i|\}$.
(32)


If $e_0 > 0$,
$e > 0$, and
$M(\Delta ab) \geq \delta$
Yes
No
If $e > e_0$
No
Yes
$a_i - \hat{a}_i \Rightarrow \tilde{a}_i$,
$b_i - \hat{b}_i \Rightarrow \tilde{b}_i$
for all $i = 1, 2, ..., n$.
(35a)
Output a, b, c
and μ. (39)
Stop.
$i = 1$.
$i + 1 \Rightarrow i$
If $|\tilde{a}_i| > \delta$
Yes
$\frac{\tilde{a}_i}{2} \Rightarrow \tilde{a}_i$ and
$\hat{a}_i + \tilde{a}_i \Rightarrow \hat{a}_i$. (35b)
No
Yes
No
If $i \leq n$
No
If $|\tilde{b}_i| > \delta$
Yes
$\frac{\tilde{b}_i}{2} \Rightarrow \tilde{b}_i$ and
$\hat{b}_i + \tilde{b}_i \Rightarrow \hat{b}_i$ (35c)
Construct matrix **Z**
from $\hat{a}$, $\hat{b}$, and f_j
where $j = 1, 2, ..., l$.
(35d)
Determine
$c, \mu_1, \mu_2, ..., \mu_{2^n-1}$ from
Z using the matrix
least square method.
(35d)
Calculate
$\hat{y}_j = c + \int(\hat{a} + \hat{b}f_j)d\mu$
for $j = 1, 2, ..., l$.
(35e)


$e = \sqrt{\frac{1}{l}\sum_{j=1}^{l}(y_j - \hat{y}_j)^2}$
(35f)
Let
$M = \max_{1 \le i \le n}\{|\tilde{a}_i|, |\tilde{b}_i|\}$.
(35g)
If $e > e_0$
and $M > \delta$
Yes
No
$\hat{a}_i \Rightarrow a_i$, $\hat{b}_i \Rightarrow b_i$ for
all $i = 1, 2, \ldots, n$. (36)
$e \Rightarrow e_0$
(37)
Go to
step 6.

APPENDIX D

The data set used in example 1. To demonstrate that the program is not affected by how the predictive attributes are scaled, x_1 was given values in the range $[0,4]$, x_2 was given values in the range $[0,2]$, and x_3 was given values in the range $[0,3]$.

1.967130247	0.847818726	1.117556228	5.341638398
3.397071739	0.904675509	2.160916075	5.8698522
3.355496211	0.029718429	0.83917255	5.075331202
3.357802115	1.552498792	0.212214538	5.111508741
0.355111368	1.591193199	1.625737093	5.480726164
1.004692662	1.488287267	1.822790292	5.613792912
3.766679791	1.260728246	0.547288833	5.299282069
3.109332393	1.352356519	1.42600321	5.755778507
1.225763457	0.720867476	0.719703966	5.019055298
3.634872427	1.501924163	2.142448089	6.108581023
1.798518125	0.695320031	2.965508713	5.821932743
0.986065744	0.803076248	0.007283637	4.631468956
2.274463743	0.27288315	2.802261103	5.697277965
1.767879374	1.531685003	0.138302028	4.904838755
2.434533428	1.727064723	1.64710727	5.825064005
1.220210451	1.620896004	0.718760875	5.161755366
2.315029399	1.819453346	2.459695602	6.107388161
3.830788286	1.779276635	2.857348562	6.441191335
2.427522383	1.835177075	0.063120322	4.974252763
3.083743655	1.322300598	0.913712604	5.468170024
2.48819079	0.384350184	1.161091884	5.270886718
1.678354182	1.951542427	2.201707614	5.976758988
1.852139934	0.027129716	1.389385184	5.132881435
2.089417784	0.190543077	1.642000137	5.29775905
2.870901293	1.973521114	0.834186215	5.503365237
2.713340329	0.222296011	0.297114437	4.847286721
0.064117686	0.122783584	0.755768918	4.715639075
0.165632927	1.917956519	0.729749695	5.056592829
1.526401653	1.410325941	0.162640979	4.875876903
0.116880831	1.815716369	1.051711295	5.193902391
2.522051326	0.410247344	2.481662969	5.680802961
0.363646191	1.5875701	2.726654523	5.820565718
1.835735538	1.205801963	0.409578031	5.022248687
1.05399075	1.473449656	1.83375699	5.619729599
2.165968506	0.213443795	0.859908003	5.07994677
0.987621531	0.589182791	2.348249055	5.481156439
2.627197922	0.672881564	2.438422352	5.780928494
3.988867154	0.717954982	0.133397865	4.931791273
0.160174615	1.358626667	2.495649765	5.647481655
0.245053377	1.934189371	1.93015202	5.671304316

APPENDIX E

The data set used in example 2.

Fixed assets (Billion dollars)	Total Labor Force (Million people)	Higher Education (Million people)	GDP (Billion dollars)
57.96	2248.91	40.76	322.07
105.45	2318.56	42.67	427.5
134.13	2386.42	44.45	500.06
191.02	2444.73	46.21	603.71
240.71	2502.73	48.21	765.76
262.99	2522.86	50.04	843.72
264.66	2554.46	51.92	897.99
324.97	2579.36	53.78	1081.75
443.8	2600.38	55.91	1365.06
782.14	2615.89	58.68	1909.49
1054.42	2640.51	61.73	2666.86
1482.62	2621.47	64.54	3524.79
1608.56	2625.06	65.20727	4146.06
1611.44	2619.66	67.30455	4638.24
1801.74	2612.54	69.40182	4987.5
1958.05	2625.17	71.49909	5364.89
2349.95	2726.09	73.59636	6036.34
2834.94	2796.65	75.69364	6748.15
3477.47	2835	77.79091	7670

APPENDIX F

The training data set from example 3.

Country	Crude birth rate	Infant death rate	Rate of natural increase	Life expectancy (males)
Maurice	17.4	20.3	10.5	66.86
Puerto Rico	17.3	11.3	9.4	71.41
Saint Lucia	22.1	17.6	15.5	70.76
Chile	17.8	10.5	12.4	72.13
Armenia	11.6	15.4	5.3	70.3
Azerbaijan	16.8	19.6	10.9	67.4
China: Hong Kong SAR	9.1	3.9	4.2	76.77
Israel	21.4	6.4	15.2	76.05
Kazakhstan	14.8	25.3	4.6	59.04
Korea	14.7	2.5	9.4	70.56
Kyrgyzstan	21.8	28.6	14.4	62.55
Malaysia	24.9	9.4	20.4	69.58
Maldives	23.9	26.7	19.4	69.2
Singapore	12.5	3.8	8.4	75
Belarus	8.8	12.6	-4.6	62.88
Czech Republic	8.8	5.9	-2.1	70.5
Denmark	12.8	5.2	1.5	73.68
Estonia	8.7	10.1	-4.1	64.78
Finland	11.5	3.9	2	73.43
France	12.4	4.7	3.4	74.77
Germany	9.9	4.9	-0.6	74.44
Hungary	9.9	9.9	-3.8	66.35
Latvia	7.7	15.3	-6	64.21
Lithuania	10.6	10.3	-0.9	65.9
Netherlands	12.3	5	3.6	75.39
Poland	10.7	10.2	0.8	68.45
Serbia and Montenegro	12.4	14.3	1.8	69.79
Slovenia	9.1	5.2	-0.4	71.05
Sweden	10.2	3.6	-0.3	76.7
Switzerland	11.4	4.8	2.5	76.5
Macedonia	14.8	15.7	6.5	70.3
United Kingdom	12.3	5.9	1.6	74.66
Australia	13.6	5.3	6.6	76.22
New Zealand	15.4	6.5	8	75.24

The validation set from example 3.

Country	Crude birth rate	Infant death rate	Rate of natural increase	Life expectancy (males)
Ukraine	8.7	14.2	-6.1	62.74
Norway	13.6	4.1	3.5	75.45
Malta	12.6	6.4	5.1	74.86
Greece	9.7	6.4	0.2	75.31
Austria	10.4	4.7	0.6	74.29
Japan	9.5	3.7	2.2	77.19
United States	14.5	7.2	5.8	73.6
Swaziland	33.4	37.7	24.3	58

The following explanation is provided with the data:

> *Crude birth rates and crude death rates presented in this table are calculated using the number of live births and the number of deaths obtained from civil registers. These civil registration data are used only if they are considered reliable (estimated completeness of 90 percent or more).*
>
> *Rate computation: The crude birth and death rates are the annual number of each of these vital events per 1000 mid-year population.*
>
> *Similarly, infant mortality rates presented in this table are calculated using the number of live births and the number of infant deaths obtained from civil registers. If, however, the registration of infant births or deaths for any given country or area is estimated to be less than 90 percent complete, the rates are not calculated.*
>
> *Infant mortality rate is the annual number of deaths of infants under one year of age per 1000 live births in the same year.*
>
> *Rates of natural increase are the difference between the crude birth rate and the crude death rate.*
>
> *The expectation-of-life values are those provided by national statistical offices.* [11]

The results of running the United Nations data through the program.

Variable	Results
a_1	0.00000
a_2	5.58085
a_3	4.31221
b_1	-0.16366
b_2	-1.00000
b_3	-0.41077
c	78.45665
$\mu(\{\ \})$	0
$\mu(\{x_1\})$	0
$\mu(\{x_2\})$	-2.85010
$\mu(\{x_1, x_2\})$	-14350934.55
$\mu(\{x_3\})$	-1.37695
$\mu(\{x_1, x_3\})$	-0.49111
$\mu(\{x_2, x_3\})$	0.02079
$\mu(\{x_1, x_2, x_3\})$	0.10117

Table 6. Values of the multiregression parameters that resulted from running the program using the United Nations data.

Objective Attribute from Data Set (years)	Estimated Objective Attribute (years)
66.86	67.22
71.41	71.94
70.76	70.97
72.13	74.04
70.3	68.02
67.4	67.95
76.77	77.12
76.05	74.97
59.04	61.30
70.56	70.78
62.55	63.88
69.58	68.23
69.2	67.52
75	75.69
62.88	64.49
70.5	70.77
73.68	73.73
64.78	66.26
73.43	75.85
74.77	75.51
74.44	73.00
66.35	66.38
64.21	62.25
65.9	67.68
75.39	75.21
68.45	68.69
69.79	66.58
71.05	72.71
76.7	74.98
76.5	74.87
70.3	68.06
74.66	72.82
76.22	76.47
75.24	75.56

Table 7. Comparing the true objective attribute against the estimate objective attribute.

REFERENCES

[1] Bigus, J. P., *Data Mining with Neural Networks*, New York: McGraw Hill, 1996.

[2] Demmel, J., *Applied Numerical Linear Algebra*, Philadelphia: SIAM, 1997.

[3] Falkenauer, E., *Genetic Algorithms and Grouping Problems.* New Jersey: John Wiley & Sons, Inc., 1998

[4] Fogel, D. B., "An Introduction to Evolutionary Computation and Some Applications." *Evolutionary Algorithms in Engineering and Computer Science* West Sussex, England: John Wiley & Sons Ltd, 1999. pp. 24.

[5] Graphic of a Neuron. http://koti.mbnet.fi/~phodju/nenet/Pics/Neuron.gif

[6] Graphic of Sigmoid Function.

http://mathworld.wolfram.com/SigmoidFunction.html

[7] Hillier, F. S., and G. J. Lieberman, *Introduction to Operations Research (Seventh Edition)*, New York: McGraw Hill, 2001.

[8] Hinton, G. E., "How Neural Networks Learn from Experience", *Scientific American*, vol. 267, pp. 144-151, 1992.

[9] Mitchell, T. M., *Machine Learning*, New York: McGraw Hill, 1997.

[10] Murphy, P. M., and D. W. Aha. UCI Repository of machine learning databases. <http://www.ics.uci.edu/~mlearn/MLRepository.html>. Irvine, CA: University of California, Dcpartmcnt of Information and Computcr Sciencc (1994).

[11] *Principles and Recommendations for a Vital Statistics System, Revision 2*, United Nations Publication, Sales No. E.01.XVII.10, United Nations, New York, 2001. <http://unstats.un.org/unsd/demographic/products/dyb/DYB2001/Table04.xls>

[12] Russell, S. J., and P. Norvig, *Artificial Intelligence a Modern Approach*, New Jersey: Prentice-Hall, 1995.

[13] Wang, Z., et. al, "Nonlinear Nonnegative Multiregressions Based on Choquet Integrals," *International Journal of Approximate Reasoning*, vol. 25, pp. 71-87, 2000.

[14] Wang, Z., and G. J. Klir, *Fuzzy Measure Theory*, New York: Plenum Press, 1992.

[15] Wang, Z., "A New Genetic Algorithm for Nonlinear Multiregressions Based on Generalized Choquet Integrals," *Proc. FUZZ-IEEE2003,* pp. 819-921, 2003.