DESIGN AND IMPLEMENTATION OF A SOFTWARE PACKAGE FOR TOXICITY ANALYSIS OF WATER SAMPLES ON FRESHWATER ORGANISMS WITH JAVA

By

BEI ZHU

Bachelor of Engineering

Shanghai Jiao Tong University

Shanghai, China

1991

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of the
requirements for
the degree of
MASTER OF SCIENCE
July 1997

DESIGN AND IMPLEMENTATION OF A SOFTWARE PACKAGE FOR TOXICITY ANALYSIS OF WATER SAMPLES ON FRESHWATER ORGANISMS WITH JAVA

Thesis Approved:

H. Lu

Thesis Advisor

Kathleen M. Kaplan

Jacques E. LaFrance

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my advisor, Dr. Huizhu Lu, for her encouragement and excellent guidance throughout my thesis work. Without her support, motivation and patience, it would have been difficult to complete this work.

My special thanks go to Dr. Kathleen Kaplan and Dr. Jacques LaFrance for serving on my graduate committee. Their valuable suggestions and support were very helpful throughout the study.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
Motivation	1
Major Approaches	1
Organization	2
II. LITERATURE REVIEW	3
Statistics Analysis	3
Introduction to the Programming Language	4
Reasons for Choosing Java	6
III. ANALYSIS ALGORITHMS	8
Test Flow	8
Algorithms of Hypothesis Tests	14
Shapiro-Wilk's Test [1]	14
Bartlett's Test [1]	15
Dunnett's Test [1]	16
Bonferroni's T-Test [1]	18
Steel's Many-One Rank Test [1]	18
Wilconxon Rank Sum Test [1]	19
Fisher's Exact Test [1]	20
Algorithm of Point Estimation	21
Linear Interpolation Method [1]	22
Data Transformation	23
Arc Sine Square Root Transformation [1]	23
Minimum Significant Difference [1]	24
IV. DESIGN AND IMPLEMENTATION	26
Structure Design	26
Implementation	28
The Model Module	29
The View Module	35

The Controller Module 41

V. CONCLUSIONS 42

BIBLIOGRAPHY 43

APPENDIX 45

LIST OF TABLES

Table Page

I. Format for Contingency Table 20

II. A Description of Base Class 30

III. A Description of ShapiroWilk Class 31

IV. A Description of Bartlett Class 31

V. A Description of Bonferroni Class 32

VI. A Description of Dunnett Class 33

VII. A Description of Steel Class 33

VIII. A Description of Wilcoxon Class 34

IX. A Description of Fisher Class 35

X. A Description of Interpolation Class 35

LIST OF FIGURES

Figure Page

1. Flow Chart for Statistical Analysis of Fathead Minnow Embryo-Larval Data [1] 9

2. Flow Chart for Statistical Analysis of Ceriodaphnia Reproduction Data [1] 10

3. Flow Chart of Statistical Analysis of Fathead Minnow Larval Survival Data [1] 11

4. Flow Chart of Statistical Analysis of Fathead Minnow Larval Growth Data [1] 12

5. Flow Chart for Statistical Analysis of Algal Growth Response Data [1] 13

6. Flow Chart for Statistical Analysis of Ceriodaphnia Survival Data [1] 14

7. Connections between UI and Statistics 27

8. HTML Code for the Project 28

9. The Input Window 36

10. The Input Window with Data 38

11. The Report Window 39

12. The Chart Window 40

13. Choice Fields of Test Type 47

14. Choice Fields of Analysis Type 47

15. Load an Existing Data File 48

16. A Report for Fathead Minnow Larval Growth Test..49

17. A Chart for Fathead Minnow Larval Growth Test Result..50

Chapter 1

INTRODUCTION

Motivation

It is a national goal that the discharge of toxic pollutants in toxic amounts must be prohibited [1]. Extensive effluent toxicity tests should be conducted to achieve this goal. Toxicity tests are used to measure effluent toxicity and to estimate the safe concentration of toxic effluents in receiving water [1]. There exists a set of standard methods to measure toxicity of pollutants developed by the Environment Protection Agency (EPA). Thus, it is useful to provide user-friendly analysis tools to automate this process and to help researchers to obtain results accurately and efficiently.

Major Approaches

The objective of this thesis is to design and implement a software package with Java to automate toxicity analysis. The implementation will include a user-friendly graphics interface and a set of methods for toxicity analysis of water samples on freshwater organisms. This enables average users who do not have much knowledge on statistics or computer science to take full advantages of the tools. With features of Java and today's Internet and Intranet technologies, the analysis tools can be posted on a network. Users who have a Java-capable Web browser installed can do the analysis without having the tools installed locally. This feature makes the analysis software more portable, and also greatly reduces the maintenance cost. These advantages will help the EPA to carry out its policy much easier than it has been done in the past.

Organization

Chapter 3 provides flow charts of six analysis methods, together with a detailed description of each test as documented by the EPA. Chapter 4 explains the design and implementation of the program. It includes design of the overall program structure and classes that implement the tests. Chapter 5 summarizes the thesis, including the major advantage of the project. The following chapter will discuss toxicity analysis methods and Java programming language.

Chapter 2

LITERATURE REVIEW

Statistics Analysis

According to *Short-Term Methods for Estimating the Chronic Toxicity of Effluents and Receiving Waters to Freshwater Organisms* [1] published by the EPA, hypothesis tests and point estimates are used to analyze toxicity test data. These tests are used to determine the highest "safe" or "no-effect concentration" of test data. The following arguments are used to determine it.

1. No-Observed-Effect-Concentration (NOEC): NOEC is defined as the highest concentration of toxicant, to which organisms are exposed in a full life-cycle or partial life-cycle test. NOEC causes no observable adverse effects on the test organisms [1].

2. Lowest-Observed-Effect-Concentration (LOEC): LOEC is defined as the lowest concentration of toxicant, to which organisms are exposed in a life-cycle or partial life-cycle test, which causes adverse effects on the test organisms [1].

3. Effective Concentration (EC): EC is a point estimate of the toxicant concentration that would cause an observable adverse affect on a quantal, "all or nothing," response in a given percent of the test organisms. For example, EC10 is the point estimate that would cause adverse effect in 10% of the organisms.

4. Lethal Concentration (LC): LC is a point estimate of concentration would cause an adverse effect in a given percent of test population. The adverse effect is death. For example, LC20 is the point estimate that would cause death in 20% of the organisms.

5. Inhibition Concentration (IC): IC is a point estimate of the toxicant concentration that would cause a given percent reduction in a non-quantal biological measurement.

NOEC and LOEC are determined by hypothesis tests: Dunnett's test, Bonferroni's T-test, Steel's Many-One Rank test, and Wilcoxon Rank Sum test. LC, EC, and IC are determined by point estimation techniques: Probit Analysis or Linear Interpolation Method.

It is required to determine if a hypothesis test is a parametric or non-parametric one. If data are normally distributed and variances are homogeneous, a parametric test is performed. Otherwise, a non-parametric test is performed. A parametric test is preferred because the analysis is performed with the observed data and not a rank of the data, thus increasing its credibility [1].

Point estimation of probit analysis or linear interpolation is used to predict the toxicant concentrations at a predefined mortality or inhibited percentage [1].

Introduction to the Programming Language

The Internet, a global network of computers that communicate using a common protocol, consists of millions of hosts in the world. It provides users immediate information and communication facilities. It takes an important part in many businesses in the world. The Web consists of Web pages, which are located on Internet sites. The Web has been proved to be good for the purpose of distributing information to widely distributed users.

With additional graphics, image maps and forms, Web pages may become interactive, and Java is an efficient language and tool to achieve this goal. A brief introduction to Java is given below.

Java is an object-oriented programming language developed by Sun Microsystems, Inc. Java is designed to be portable, i.e., Java executable files can be moved easily from one computer system to another without recompiling. Platform independence is one of the most significant advantages that Java has over other programming languages. It makes Java portable. More important, the advantage enables Java program to work over a network with a Java-capable browser. Java is a high-level programming language similar to C and C++, but it adds a few more features that C and C++ does not have, such as garbage collection and multithreading. Java also supports most of the requirements by programmers. That makes it suitable for about any application programming task [6]. Compared to C and C++, it eliminates most of the complex parts in them. For example, there are no pointers in Java, etc. Thus, it is easier to write, and also easier to debug [6]. In addition, Java virtual machine has built-in restrictions to prevent most traditional ways of causing damage to the client systems. Java is an object-oriented language. This helps to design programs in terms of objects. Each of them has a specific role in the program and all of them can talk to each other in a predefined way, which means it allows abstract data type to be easily created and used. It has capabilities of creating flexible, modular and reusable code. Java also has classes to support user interface functions.

Java programs have two groups: *applications* and *applets*. Java applications are general programs written in Java. Java applets are special kind of Java programs that can be downloaded from the Web server, and executed by a Java-capable Web browser on a client machine. This makes Java applets to have the advantage of structures a browser supports, such as graphics context, event-handling, existing window and surrounding user interface.

We are truly in an information society. Computers are popular more than ever. With increasing number of computers, the total cost of ownership is becoming more sensitive for large organizations. One way to solve this problem is to construct a network with one or more powerful centralized servers and relatively weak clients. Since major maintenance work is done on the server, the cost of the ownership of the majority clients is low. Java fits into this model very well. A Java applet is installed and maintained on a server. While other computer systems connected to the network need to execute such a Java program, they download the program over the network and execute it locally with a Java-capable browser. This also helps to unload computing tasks from the servers.

However, Java has its own disadvantages. Compared to C and C++, its execution speed is relatively slow because object code (bytecode) generated by a Java compiler is intermediate code for the Java virtual machine, not for the real target machine. Yet, there are also several solutions for it, such as just-in-time compiler, which converts Java bytecode into the native machine code as it loads on a client machine.

Reasons for Choosing Java

After analyzing the requirements from the EPA, the author chooses Java as the programming language for the implementation for the following reasons:

1. Java is an object-oriented language. The objected-oriented approach attempts to manage the complexity inherent in real-world problem by abstracting out knowledge,

and encapsulating it within objects. It is much natural to organize and maintain than traditional procedural languages.

2. Java includes a library called Abstract Windowing Toolkit (AWT), which provides a set of building blocks for user interfaces. It provides a common base across all platforms that support Java. It enables programmers to write one version of user interface that appears identically on all different client platforms.

3. Java is designed based on a generic virtual machine. Its compiled code is stored as bytecode of the virtual machine. Therefore, its object code is not tied to any particular hardware. Thus, Java object code is platform independent. With this feature and the technology of networking, it is possible to use Java as a means to deliver programs in their executable forms, namely through Web to pass Java bytecode to a browser that is capable of executing the program.

Chapter 3

ANALYSIS ALGORITHMS

Test Flow

The EPA has suggested the following six different experiments to analysis toxicity.

1. Fathead Minnow Embryo-Larval Survival and Teratogenecity test;

2. Ceriodaphnia Reproduction test;

3. Fathead Minnow Larval Survival test;

4. Fathead Minnow Larval Growth test;

5. Algal Growth test; and

6. Ceriodaphnia Survival test.

Their flow charts are presented in Figure 1 to Figure 6. Algorithms for tests in Figure 1 to Figure 6 follow. They are based on the EPA documentation, *Short-Term Methods for Estimating the Chronic Toxicity of Effluents and Receiving Waters to Freshwater Organisms* [1].

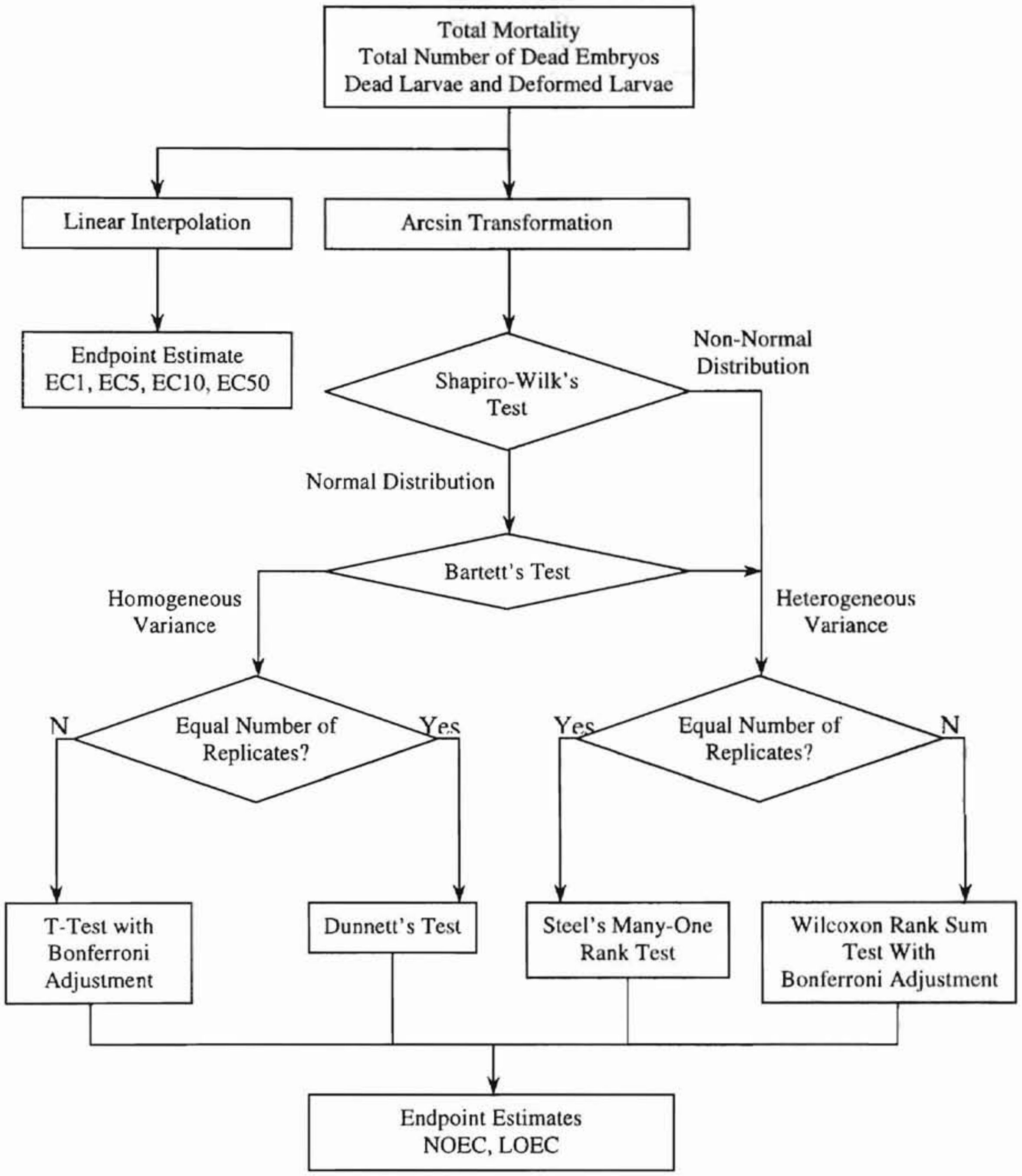


Figure 1: Flow Chart for Statistical Analysis of Fathead Minnow Embryo-Larval Data [1]

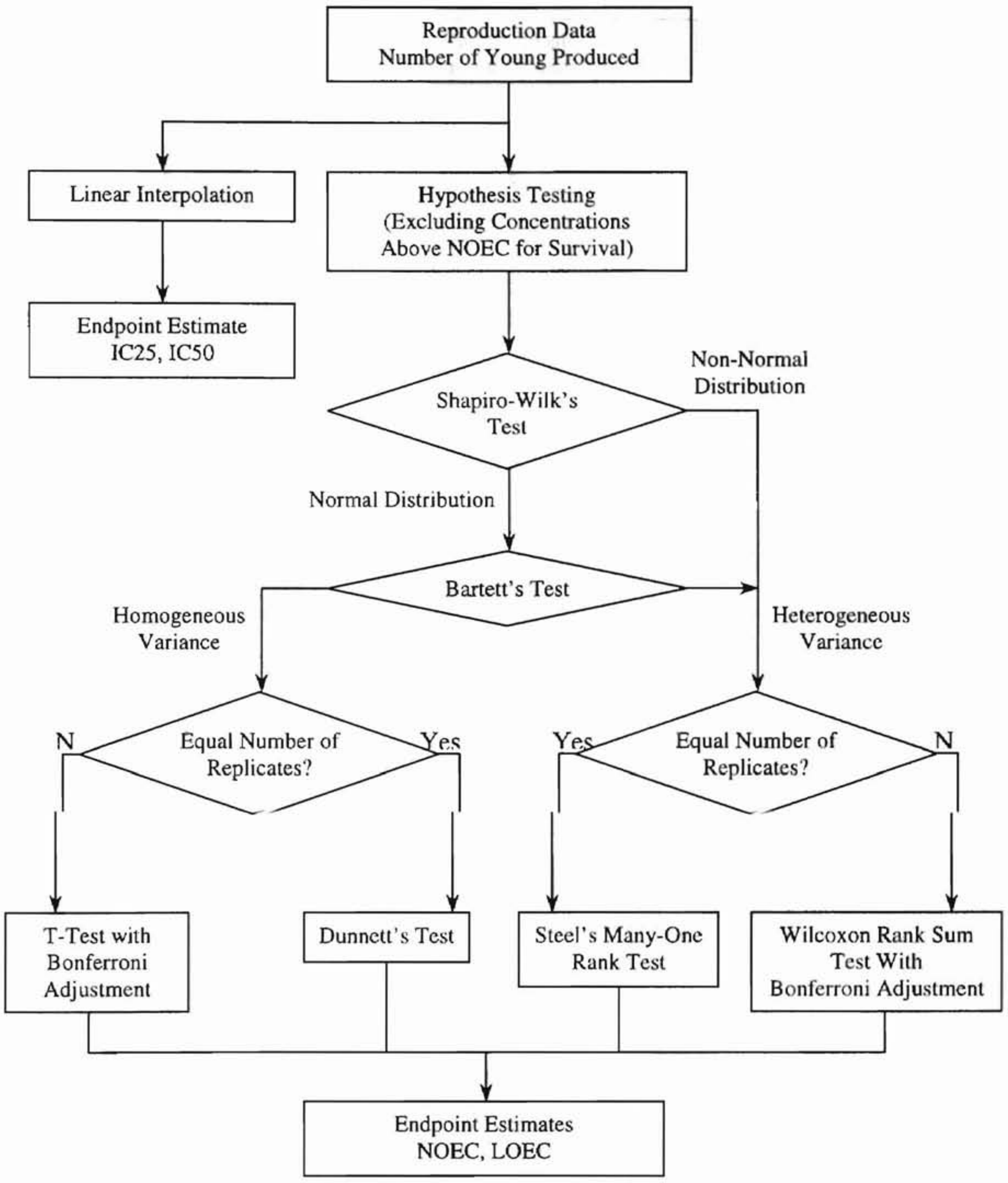


Figure 2: Flow Chart for Statistical Analysis of Ceriodaphnia Reproduction Data [1]

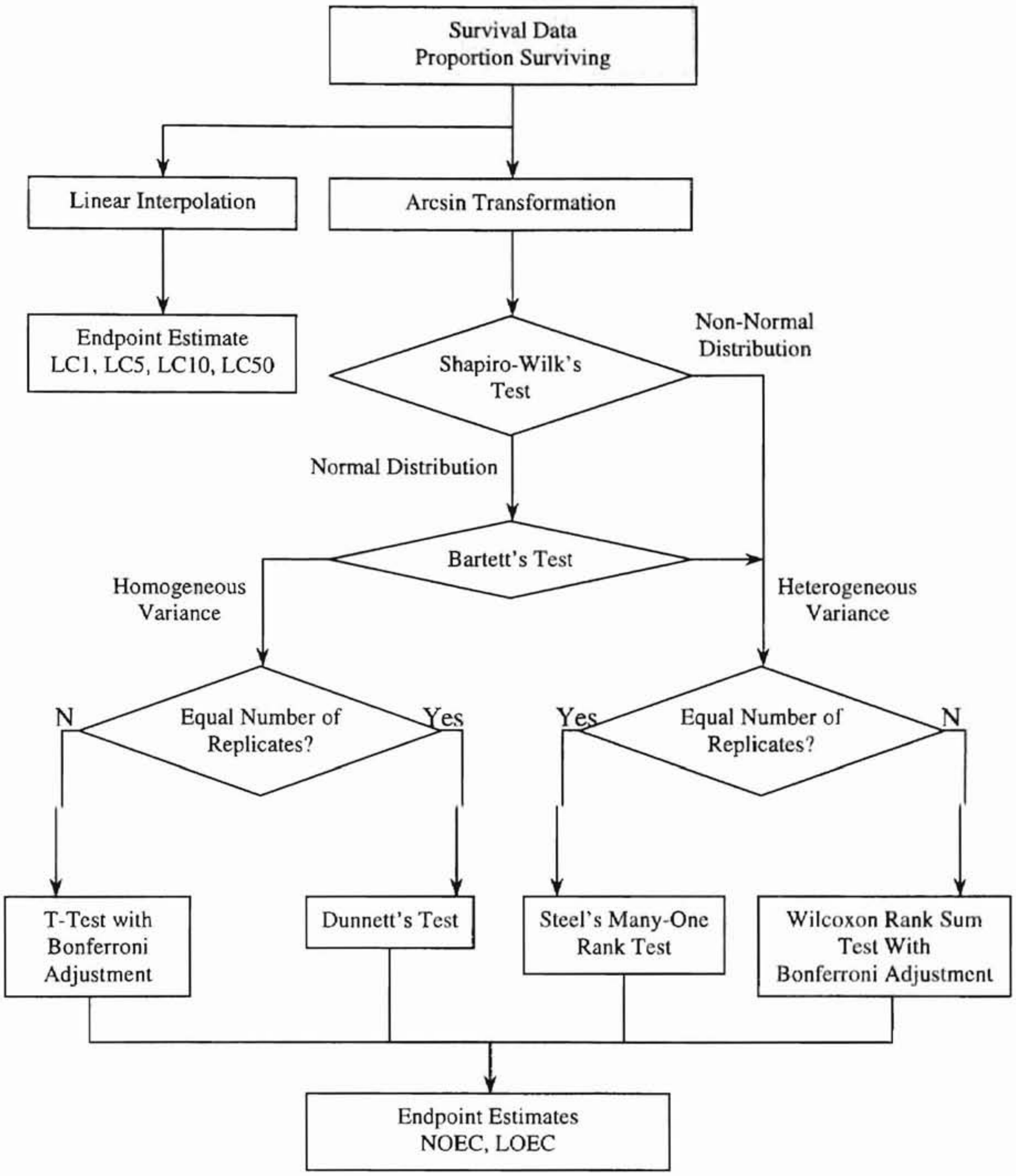


Figure 3: Flow Chart of Statistical Analysis of Fathead Minnow Larval Survival Data [1]

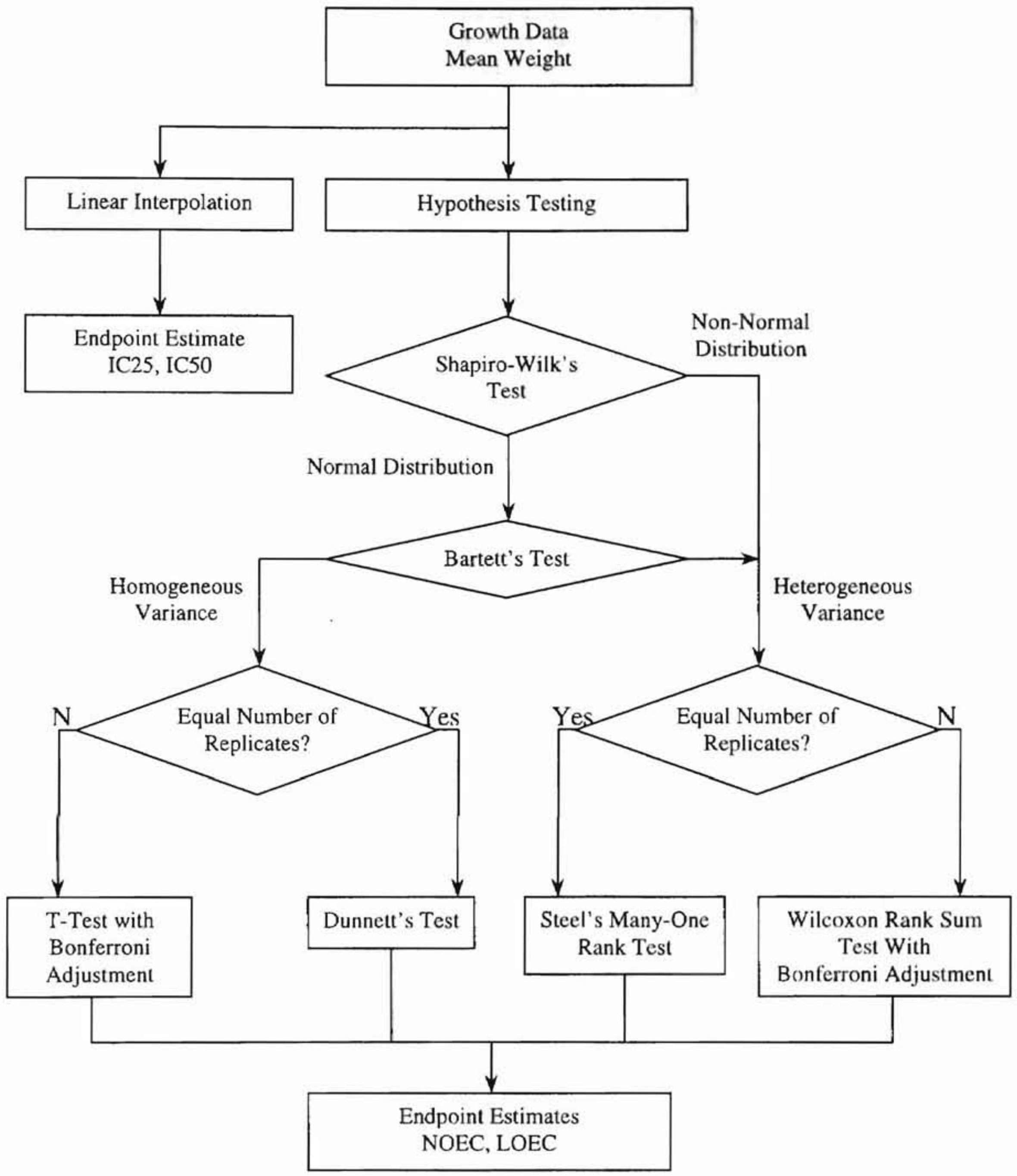


Figure 4: Flow Chart of Statistical Analysis of Fathead Minnow Larval Growth Data [1]

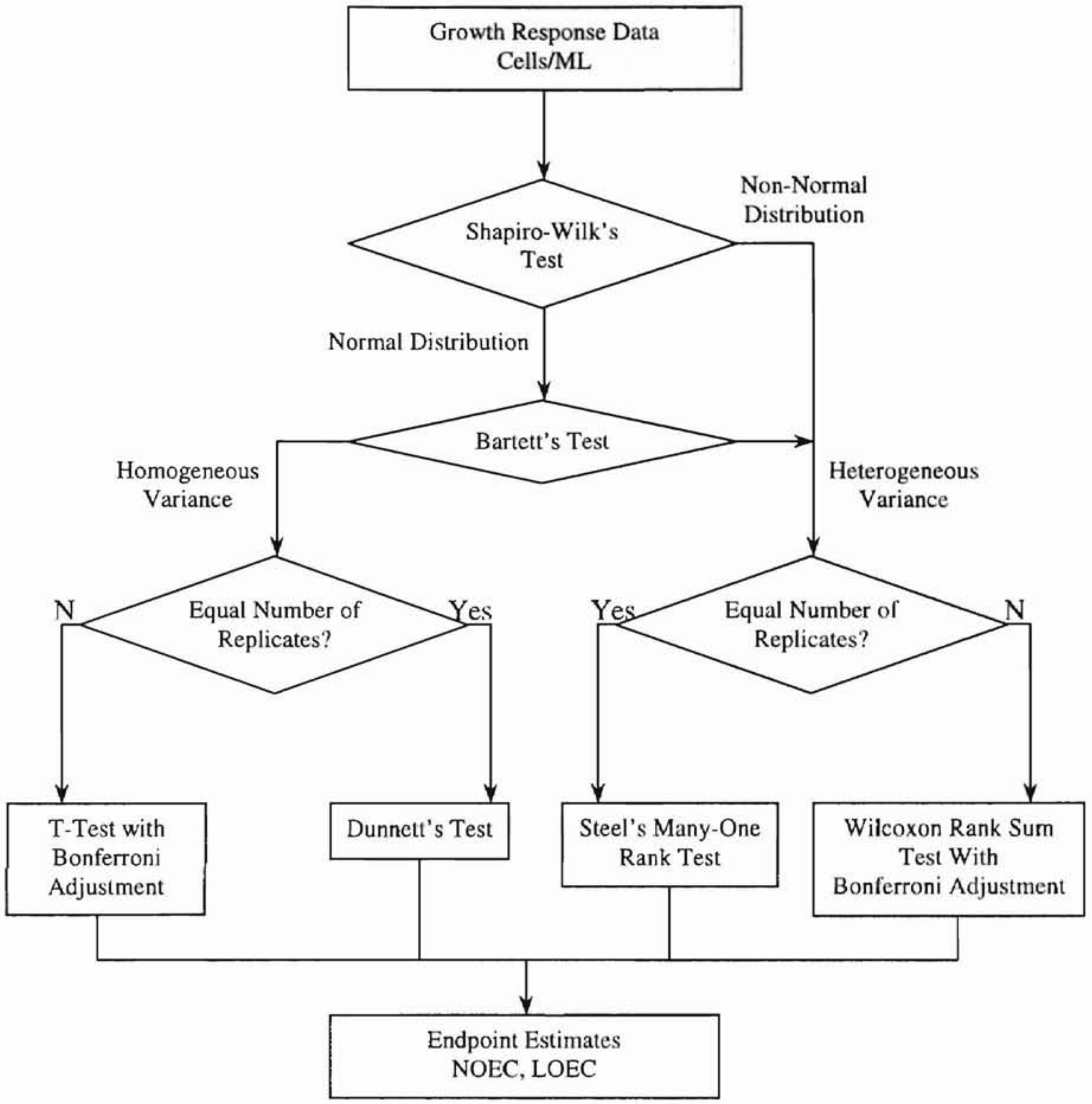


Figure 5: Flow Chart for Statistical Analysis of Algal Growth Response Data [1]

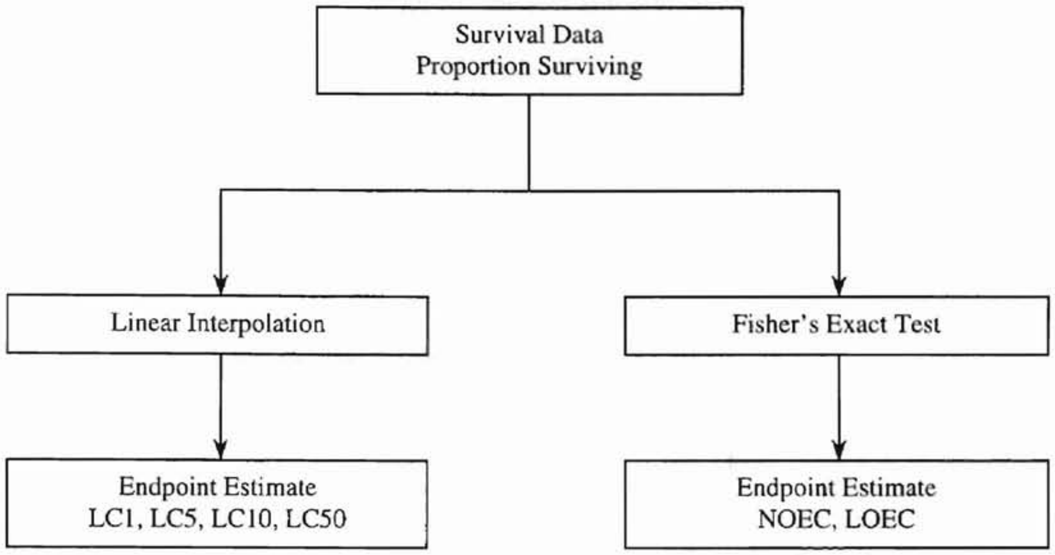


Figure 6: Flow Chart for Statistical Analysis of Ceriodaphnia Survival Data [1]

Algorithms of Hypothesis Tests

Shapiro-Wilk's test, Bartett's test, Dunnet's test, Bonferroni's T-test, Steel's Many-One Rank test and Wilcoxon Rank Sum test with Bonferroni Adjustment are used in Figure 1 to Figure 5, and they are hypothesis tests. The algorithms for those tests are depicted below according to the EPA document [1].

Shapiro-Wilk's Test [1]

Shapiro-Wilk's test is used to check whether the data are normally distributed or not. The algorithm of the test is given below.

Step 1: For the total n observations, calculate centered observations, X_i, $i=\{1,2, ..., n\}$, by subtracting the mean of all observations within a concentration from all observations, respectively.

Step 2: Calculate the overall mean of centered observations, Y.

Step 3: Calculate the denominator, *D*, for the test statistic:

$$D = \sum_{i=1}^{n} (X_i - Y)^2$$, where *n* is the total number of the observations.

Step 4: Order the centered observations in ascending order and denote them as X^1, X^2, ... , X^i, ..., X^n, where X^i is the i^{th} ordered observation.

Step 5: Get coefficients, C_i , $i = \{1, 2, ..., k\}$, from the table, where *k* is approximately equal to *n/2*.

Step 6: Compute the test statistics, *W*, as follows:

$$W = \frac{1}{D}[\sum_{i=1}^{k} C_i \cdot (X^{n-i+1} - X^i)]^2$$, where *k* is approximately *n/2*.

Step 7: Find the critical value at significance level 0.01 or 0.05 and total observations, *n*, in the table given by the EPA. If the computed *W* is greater than or equal to the critical value, then the data are normally distributed. Otherwise, the data are not normally distributed. (Notes: The calculated *W* must be greater than zero and less than or equal to one. This test is recommended for a sample size of 50 or less.)

Bartlett's Test [1]

Bartlett's test is used to determine the homogeneity of variance, and is based on the assumption that the data are normally distributed. The algorithm of the test is depicted below.

Notation:

n_i: the number of replicates for concentration *i*.

p: the number of concentration including the control.

S_i: variance of group *i*.

Step 1: Calculate $sv = \sum_{i=1}^{p}(n_i - 1)$, $dsv = \sum_{i=1}^{p}\frac{1}{n_i - 1}$, $S^2 = \frac{\sum_{i=1}^{p} S_i^2 \cdot (n_i - 1)}{sv}$,

$cosv = \sum_{i=1}^{p}(n_i - 1)\cdot \ln S_i^2$, $C = 1 + \frac{1}{3(p-1)}\cdot(dsv - \frac{1}{sv})$, $B = \frac{sv \cdot \ln S^2 - cosv}{C}$.

Step 2: Find the critical value with significance level 0.01 or 0.05 and p-1 degrees of freedom in the table given by the EPA. If B is less than the critical value, then the homogeneity of variance is satisfied. Namely, the variances are equal.

Dunnett's Test [1]

Dunnett's test is used to determine whether the mean for i^{th} concentration is different from the mean for control. Namely, it is used to compare each concentration mean with the control mean to decide if any of the concentrations differs from the control. This test can detect a significant reduction in mean weight if there is any. The test requires that the data are normally distributed and variances of the data obtained from each toxicant concentration and the control are equal [1]. The number of replicates for each concentration is also required to be equal. Otherwise, Bonferroni's T-test is used as an alternative. From results of Dunnett's test, the NOEC and the LOEC for growth can be determined.

The algorithm of Dunnett's test is presented below according to the EPA document [1].

Notation

n_i: the number of replicates for concentration i.

T_i: the total of the replicate measurements for concentration i.

G: the grand total of all sample observations.

N: the total sample size.

Y_{ij}: the jth observation for concentration i.

k: the number of groups including control.

N: the number of observations.

SST: total sum of squares.

SSB: between sum of squares.

SSW: within sum of squares.

Step 1: Calculate $T_i = \sum_{j=1}^{n_i} Y_{ij}$, $G = \sum_{i=1}^{k} T_i$, $N = \sum_{i=1}^{k} n_i$, $SST = \sum_{i=1}^{k}\sum_{j=1}^{n_i} Y_{ij}^2 - \frac{G^2}{N}$,

$SSB = \sum_{i=1}^{k} \frac{T_i^2}{n_i} - \frac{G^2}{N}$, $SSW = SST - SSB$, $Sw = \sqrt{\frac{SSW}{N-k}}$.

Step 2: Calculate statistic $t_i = \frac{Y_1 - Y_i}{S_w \sqrt{\frac{1}{n_1} - \frac{1}{n_i}}}$ for each concentration i and the control, where

Y_1 is the control mean, Y_i is the mean for the concentration i, S_w is defined as in Step 1, n_l is the number of replicates in the control, n_i is the number of replicates for concentration i.

Step 3: Get the critical value with significance level 0.01 or 0.05 and N-k degrees of freedom in the table. For every t_i, if t_i is greater than the critical value, then the group i (i.e., concentration i) is significantly different from the control. Namely, the concentration i has significantly lower growth than the control.

Bonferroni's T-Test [1]

Bonferroni's T-test is used as an alternative to Dunnett's test when the number of replicates is not the same for all concentrations. Like Dunnett's test, Bonferroni's T-test is based on the assumptions of (i) the data is normally distributed, (ii) homogeneity of variance. The result of Bonferroni's T-test is also used to determine the NOEC and LOEC. The algorithm of Bonferroni's T-test is presented below. The notation used in this algorithm is the same as in Dunnett's test.

Step 1: Calculate $T_i = \sum_{j=1}^{n_i} Y_{ij}$, $G = \sum_{i=1}^{k} T_i$, $N = \sum_{i=1}^{k} n_i$.

Step 2: Calculate $SST = \sum_{i=1}^{k}\sum_{j=1}^{n_i} Y_{ij}^2 - \frac{G^2}{N}$, $SSB = \sum_{i=1}^{k}\frac{T_i^2}{n_i} - \frac{G^2}{N}$, $SSW = SST - SSB$.

Step 3: Calculate $S_w = \sqrt{\frac{SSW}{N-k}}$.

Step 4: Calculate t_i, the statistic for each concentration and control: $t_i = \frac{Y_1 - Y_i}{S_w\sqrt{\frac{1}{n_1}+\frac{1}{n_i}}}$.

Step 5: Find the critical value, with N-k degrees of freedom in the Bonferroni's T table. If t_i is greater than the critical value, then group i is significantly different from the control. Namely, the mean of concentration i is significantly less than the control mean.

Steel's Many-One Rank Test [1]

If the data is not normally distributed and/or variances are not equal, then Dunnett's test and Bonferroni's T-test can not be used. In those cases, Steel's Many-One Rank test can be carried out if the number of replicates for each concentration are the same. Otherwise, Wilcoxon Rank Sum test is used.

Steel's Many-One Rank test is a multiple comparison method for comparing several treatments with a control. The data are ranked, and the analysis is performed on the ranks rather than on data themselves. It is necessary to have at least four replicates per toxicant concentration to use Steel's test. This is a non-parametric test and therefore the assumptions of normality and homogeneity does not need to be met. The sensitivity of the test can not be stated in terms of the minimum difference between treatment means and the control mean.

The algorithm of the Steel's Many-One Rank test is described as follows [1].

Step 1: Combine the data and arrange the observations in order of size from the smallest to largest *b*. Assign ranks to the ordered observations.

Step 2: Calculate the sum of the ranks, R_i, at each concentration and the control.

Step 3: For each R_i, if R_i is less than or equal to the critical rank sum in the table of Steel's Many-One Rank Sum test, then the group *i* is significantly different from the control. Test results are used to determine the NOEC and LOEC.

Wilconxon Rank Sum Test [1]

Wilcoxon's Rank Sum Test is used as an alternative to Steel's Many-One Rank Test when the numbers of replicates are not the same at each concentration. The control is used to set an upper bound of alpha on the overall error rate, in contrast to Steel's Many-One Rank test. Thus, Steel's test is a more powerful test.

The Algorithm is depicted as follows.

Step 1: Combine the data and arrange the observations in the order of size from the smallest to largest. Assign ranks to the ordered observations.

Step 2: Calculate the sum of the ranks, R_i, at each concentration and the control.

Step 3: For each rank R_i, find the critical rank sum at the significance level 0.05 or 0.01 from the table of Wilcoxon Rank Sum test. If R_i is less than or equal to the critical rank sum, then the group (concentration) i is significantly from the control. The NOEC and LOEC can be determined from the test result.

Fisher's Exact Test [1]

Fisher's Exact test is a statistical method based on the hypergeometric probability distribution that can be used to test if the proportion of successes is the same in two Bernoulli populations [1]. The test is for Ceriodaphnia Survival data as shown in Figure 6. To perform this test, each replicate value must be between 0 and 15.

	# Successes	# Failures	# Observations
Row 1	A	A – a	A
Row 2	B	B – b	B
Total	a + b	[(A + B) - a - b]	A + B

Table 1: Format for Contingency Table

Arrange Contingency Table (Table 1) so that the total number of observations for row one is greater than or equal to the total number for row two ($A \geq B$). Categorize a success such that the proportion of successes for row one is greater than or equal to the proportion of successes for row two ($a/A \geq b/B$). Then find the critical value with A, B, a, and the significance level 0.05 or 0.01 from the table. If b is less than or equal to the critical value, then the group is significantly different from the control.

Algorithm of Point Estimation

The LC, EC, or IC is derived from a mathematical model that assumes a continuous dose-response relationship. This is the reason why any LC, EC, or IC value is an estimate of some amount of adverse effect. Thus, to use a point estimate such as LC, EC, or IC to determine a "safe" concentration would require the specification by biologists or toxicologist of what level of adverse effect would be deemed acceptable or "safe". Point estimation techniques have the advantage of providing a point estimate of the toxicant concentration causing a given amount of adverse (inhibiting) effect.

The linear interpolation method is used for the point estimate of the effluent or other toxicant concentration that causes a given percent reduction (e.g., 25%, 50%, etc.) in the reproduction or growth of the test organisms (Inhibition Concentration, or IC). The linear interpolation method assumes that the responses are monotonically non-increasing, where the mean response for each higher concentration is less than or equal to the mean response for the previous concentration. If the data are not monotonically non-increasing, they are adjusted by smoothing (averaging). The IC is estimated by linear interpolation between two concentrations whose responses bracket the response of interest, the p percent reduction from the control. To obtain the estimate, determine the concentrations C_j and C_{j+1} which bracket the response M_1 (1-p/100), where M_1 is the smoothed control mean response and p is the percent reduction in response relative to the control response. The algorithm of linear interpolation method used for the point estimate is presented below according to the EPA document [1].

Linear Interpolation Method [1]

Step 1: Calculate the smoothed mean by averaging adjacent means as follows. Let y_i be the control mean. If y_{i+1} is less than or equal to y_i , then y_{i+1} is used. Otherwise, the average y_i and y_{i+1} is used as the new mean for group i and i+1. If y_{i+2} is greater than y_i , the average of y_i , y_{i+1} and y_{i+2} is used as the new mean for group i, i+1 and i+2. This continues till all means are in decreasing order.

Step 2: Calculte ICp as follows: $ICp = C_j + (M_1 \cdot (1 - \frac{p}{100}) - M_j) \cdot \frac{C_{j+1} - C_j}{M_{j+1} - M_j}$, where

IC_p: the percent reduction,

C_j: the 1st concentration whose observed mean response is greater than $M_1 \cdot (1 - \frac{p}{100})$,

C_{j+1}: the 1st concentration whose observed mean response is less than $M_1 \cdot (1 - \frac{p}{100})$,

M_1: the smoothed mean for control,

M_j: the smoothed mean for concentration j,

M_{j+1}: the smoothed mean for concentration j+1.

Data Transformation

When the assumptions of normality and homogeneity of variance are not met, input data may be transformed by an Arc Sine Root Transformation. Then, the data may be analyzed by parametric procedures, rather than a non-parametric technique such as Steel's Many-one Rank Test or Wilcoxon's Rank Sum Test. After the data have been transformed, Shapiro-Wilk's and Bartlett's tests should be performed on the transformed observations to determine whether the assumptions of normality and/or homogeneity of variance are met [1].

Arc Sine Square Root Transformation [1]

Arc Sine Square Root Transformation is used in Fathead Minnow Larval Survival analysis as shown in Figure 3 for hypothesis testing, which deal with survival proportion. When the proportion is 0 or 1, the Arc Sine Square Root Transformation ($arc\ sine\ \sqrt{P_i}$) is commonly used to stabilize the variance and satisfy the normality requirements, where P_i is the expected proportion (response/no response or live/dead) for the treatment. Following are detail explanation and examples of the Arc Sine Square Root Transformation, which is based on the EPA document [1].

Step 1: Calculate the response proportion (*RP*) at each effluent concentration, where

RP = (number of dead or "affected" organisms) / (number exposed).

For example, if 8 of 20 animals in a given treatment die, $RP = 8/20 = 0.4$.

Step 2: Transform each *RP* to *arc sine*.

(1) If $RP = 0.4$, $Angle = arc\ sine\ \sqrt{0.4} = arc\ sine\ 0.6325 = 0.6847$ radians.

(2) If $RP = 0$, special modification on procedures as follows:

Angle (in radians) = *arc sine* $\sqrt{1/(4N)}$, where N = Number of animals/treatment. Assume 20 animals are used. Then, *Angle* = *arc sine* $\sqrt{1/80}$ = *arc sine* 0.1118 = 0.112 radians.

(3) If $RP = 1$, a special modification on the *Angle* is made as follows:

Angle = 1.5708 – (radians for $RP = 0$). Using previous data, *Angle* = 1.5708 – 0.112 = 1.4588 radians.

Minimum Significant Difference [1]

The minimum significant difference (*MSD*) is used to determine the sensitivity of a test [1]. The *MSD* is calculated after either Dunnett's or T-test with Bonferroni Adjustment is used.

1. Calculate the *MSD* for each group $MSD = cS_w\sqrt{1/N + 1/N_i}$, where c is the critical value for either Dunnett's or T-test with Bonferroni Adjustment, S_w is the square root of within mean square, N is the number of observations in control, and N_i is the number of observations in group i.

2. If the data has been transformed, then the untransformed MSD_u is calculated as the following example given in the EPA document [1].

An example: if the transformed control mean *ControlMean* = 0.714 and *MSD* in transformed unit = 0.087.

(a) Calculate untransformed units *UMSD*.

$UMSD = [sine\ (0.714)]^2$.

(b) Calculate untransformed difference *DMSD*.

$DMSD = [sine\ (0.714\text{-}0.087)]^2 = 0.344$.

(c) Calculate untransformed MSD_u.

$MSD_u = UMSD - DMSD = 0.429 - 0.344 = 0.085$.

Chapter 4

DESIGN AND IMPLEMENTATION

Structure Design

The thesis presents an object-oriented approach using Java for the toxicity analysis of water samples on freshwater organisms. Object-oriented design decomposes a program into objects. Each object knows how to perform its own operations and remembers its own information. Meanwhile, objects have a private side. The private side of an object is not a concern of other objects. With this, objects are free to change their private sides without affecting other objects. If the software has been designed with rigorous consistency, interfaces can be extended and entities can be added. Programmers can add new entities that response to old requests in ways appropriate to the new system of which they are now a part. If the interfaces between entities have been rigorously controlled, new portions of the system can be created to use the same interface, but to do different things with them [9]. Thus, object-oriented design can be easily reused, refined, tested, maintained, and extended.

The whole project is divided into three independent parts: model (statistical analysis), controller and view (graphical user interface) (Figure 7). According to the heuristics implied by Jacobson's Objectory methodology, the policy information should not be placed inside of classes involved in the policy decision because it renders them unreusable by binding them to the domain that set the policy [9]. To realize this heuristics, there should be a special type of class called controller. It only contains behavior. It gets data from outside the class and is used to decouple classes from their policy. On the other hand, controller classes do render their host classes more reusable, because those classes are mindless. In our case, the design is as Figure 7.

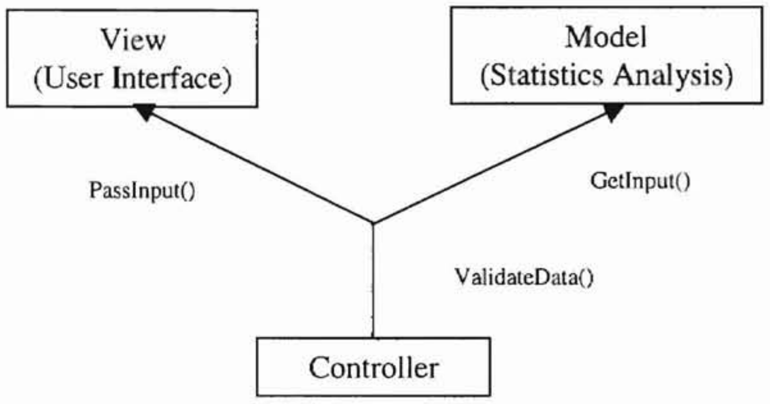


Figure 7: Connections between UI and Statistics

A controller handles data between view (user interface) and model (statistics analysis). For example, it checks the data getting from view, converts them into its correct type (usually changing from string to float in this case). This type of design is as the artificial separation of data and behavior in a bi-directionally related package. A controller is clearly a useful facility. If changes are made to the statistical analysis part, only the functions relating to it have to be changed, and leave the user interface part unchanged. Uses interfaces display the internals of a model, allow a user to update those internals, and put the internals back into the model.

The program is implemented in such a way that the same bytecode can be used as either an applet or an application. An applet is a class derived from the Applet class. It can only be run with a browser. An applet shares the same frame (window) with the browser that launches it. Java virtual machine first calls init() method in the applet, then start(). A Java application is executed using a different way. It uses a static method main() as its entry point. It is not given a frame automatically. We need to simulate an environment as what a browser provides in order to let an applet run under an application environment. To do so, we need to create a frame in main(). Then create an instance of the applet and put it on the frame just created. The main() should also call init() and start() of the applet.

Implementation

To run an applet, a Web page should be created. It tells the browser where and how to load the program. The HTML file in the package is created for this purpose (Figure 8). While running the application, we use Java loader to load the program.

```
<html>
<head>
<title>Toy</title>
</head>
<body>
<hr>
<applet code=Toy.class id=Toy width=600 height=600>
</applet>
<hr>
</body>
</html>
```

Figure 8: HTML Code for the Project

Base on the structure design and the documents of the EPA, we create twelve classes for the model module, five classes for the view module, and some other classes. They are presented in detail.

Toy: it is the entry point of the project.

AppFrame: When we run the project as an application, this is the frame object that is created in main(). It is not used when we run the project as an applet.

The Model Module

The following twelve classes belongs to the model module. Except for BadDataException, TestData, ModuleClass and Fisher, the rest eight classes, derived from BaseClass, specify different tests

1. BadDataException: this class defines an exception type for the project.

2. TestData: this class holds all test data. It also validates the test data when an instance of the class is created.

3. ModuleClass: it is the class to judge which single test should use during the whole test procedure. It controls the flow of the test.

4. BaseClass: it handles the basic functions of the statistical test. For example, GetInput(…) is for getting input data; DataTrans(…) is for handling mathematical transformation of test data before they are used for testing. All methods in BaseClass can be inherited by its child classes. Table 2 is the description of BaseClass.

Methods	Attributes
BaseClass(int);	int lastRep;
int GetInput(float[], int, int, float, float[]);	int lastCct;
void DataTrans(int, int);	int curRep;
int NumOfRep();	int curCct;
int CurRep();	int totalObs;
int CurCct();	float p_value;
int LastRep();	float mean[];
int LastCct();	float variance[];
float Mean(int);	float sumSquarerPerCct[];
float Variance(int);	float squareSumPerCct[];
float SumSquare(int);	float cctArray[];
float SquareSum(int);	int missingDataFlagOn[];
int TotalNums();	float concctionLevel[];
int ConLevel(float[]);	float inputData[][];
void SetCurRep(int);	
void SetCurCct(int);	
int ComputeGrandTotalObs();	
float ComputeMean(int, int);	
float ComputeVariance(int, int);	
float ComputeSumSquare(int, int);	
float ComputeSquareSum(int, int);	

Table 2: A Description of Base Class

5. ShapiroWilk: it checks for the normality of the test data. Two methods ExistW(…) and TestW(…) are used to check the normality. ExistW(…) gets the *W* value from the existing table while TestW(…) calculates the *W* value from the input data. If ExistW(…) is less than or equal to TestW(…), the test data are normal. Otherwise it is abnormal. Table 3 is the description of ShapiroWilk class.

Methods	Attributes

ShapiroWilkClass(int, int);
int Shaptest();
float GetD();
void ShapQuick(float[], int);
void ShapQuicksort(float[], int, int);
void AscendingOrder();
void GetCoeff();
float TestW();
float ExistW();
	float orderedCenteredData[];
	float coeffs[];
	float centeredData[][];

Table 3: A Description of ShapiroWilk Class

6. Bartlett: it determines whether variances are equal or not for the test data. Its class structure is similar to ShapiroWilk class. Two methods, ExistB(…) and TestB(…), are used to judge the equality. ExistB(…) gets the *B* value from the existing table while TestB(…) calculates the *B* value from the input data. If ExistB(…) is greater than or equal to TestB(…), the test data have equal variance. Otherwise it is unequal. Table 4 is the description of Bartlett class.

Methods	Attributes

Bartlett(int);
void SetV();
int SumV();
float SetC();
float TestB();
float ExistB();
int Bartest();
	int sumOfV;
	int v[];

Table 4: A Description of Bartlett Class

7. Bonferroni: it is the class used as a parametric test to calculate the values of NOEC and LOEC when the test data have an equal number of replicates. Several methods such as CalSST(…), CalSSW(…), etc., calculate the values of SST, SSW, etc. according to the document of the EPA. Then we have the values of NOEC and LOEC. Table 5 is the description of Bonferroni class.

Methods	Attributes
BonferroniClass(int);	float totalSamples;
void SetSamp();	int numOfObsPerCon[];
float CalSSB();	float NOEC[];
float CalSSt();	float LOEC[];
float CalSSW();	float MSD[];
float CalMSB();	
float CalMSW();	
float TestT(int);	
float ExistT();	
void Bonitest(float[], float[]);	

Table 5: A Description of Bonferroni Class

8. Dunnett: The class structure is similar to Bonferroni class. It is used when the number of replicates is equal for parametric testing. Table 6 is the description of Dunnett class.

9. Steel: this class is used when it is a non-parametric test with equal number of replicates. It is also for calculating values of NOEC and LOEC. The class structure is similar to Bonferroni class. Table 7 is the description of Steel class.

Methods	Attributes
DunnettClass(int);	float totalSamples;
void SetSamp();	float MSW;
float GetSamp();	float meanControl;
void ObsPerCon();	int numOfObsPerCon[];
float CalSSB();	float NOEC[];
float CalSST();	float LOEC[];
float CalSSW();	float MSD[];
float CalMSB();	
float CalMSW();	
float TestT(int);	
float ExistT();	
void Dunntest(float[], float[], int, int, float[], float[]);	

Table 6: A Description of Dunnett Class

Methods	Attributes
SteelClass(int, int);	float group[];
void SArrayofRank();	float rankedArray[];
void SGroupNum(int, int);	float NOEC[];
void StQuicksort(int, int);	float LOEC[];
void StQuick(int);	
void SRankNum(int);	
void Stest(float[], float[]);	
float TestSt(int);	
float ExistSt();	

Table 7: A Description of Steel Class

10. Wilcoxon: this class is used when it is a non-parametric testing and the test data have an unequal number of replicates. Values of NOEC and LOEC can be determined from the results. Table 8 is the description of Wilcoxon.

Methods	Attributes
WilcoxonClass(int, int);	float group[];
void WarrayofRank();	float rankedArray[];
void WgroupNum(int, int[]);	float NOEC[];
void WilQuicksort(int, int);	float LOEC[];
void WilQuick(int);	float MSD[];
void WrankNum(int);	float MSD[];
void Wtest(float[], float[]);	
float WrankSum(int);	
int WrepPerCct(int);	
float ExistW(int, int);	

Table 8: A Description of Wilcoxon Class

11. Fisher: This is the only test class that is not derived from BaseClass. This test is to determine if each concentration group mean differs from the control group mean, and further determine NOEC and LOEC. The method GetB(…) gets the *b* value in the statistics table. The method, TestB(…) returns the *b* value calculated from the test data. By comparing two *b*s, we get the value of NOEC and LOEC. Table 9 is the description of Fisher class.

12. Interpolation: it is a point estimation class. The concentration means must be non-increasing to perform the test. If montoninity is not met, the means can be smoothed. Methods CalSmoothedMean(…), CalBaseMean(…) and CalICp(…) etc. are employeed to calculate the value of IC, EC and LC. Table 10 is the description of Interpolation class.

Methods	Attributes
FisherClass(int);	Int numOfCct;
void GetInput(float, float[], int[]);	Float significantLevel;
int GetB(int, int, int);	Int deadData[];
String Ftest(float, float[], int[], int);	Int aliveData[];
String TestB(int, int, int[], int[], int[], int[]);	Int totalForDeadAlive[];
	float effluenCCt[];

Table 9: A Description of Fisher Class

Methods	Attributes
InterpolationClass(int);	float smoothedMean[];
void CalSmoothedMean();	float percentile;
float CalBaseMean(float, int[], float[], float[]);	Int neededIndex[];
int ConcentNum(float[], float[], float);	float lowLimit[];
float CalICp(float);	float upLimit[];
float Itest(float, float);	

Table 10: A Description of Interpolation Class

The View Module

The following five classes belong to the view module.

1. InputWindow: The functionality of InputWindow is to receive input (Figure 9). InputWindow contains the following fields:

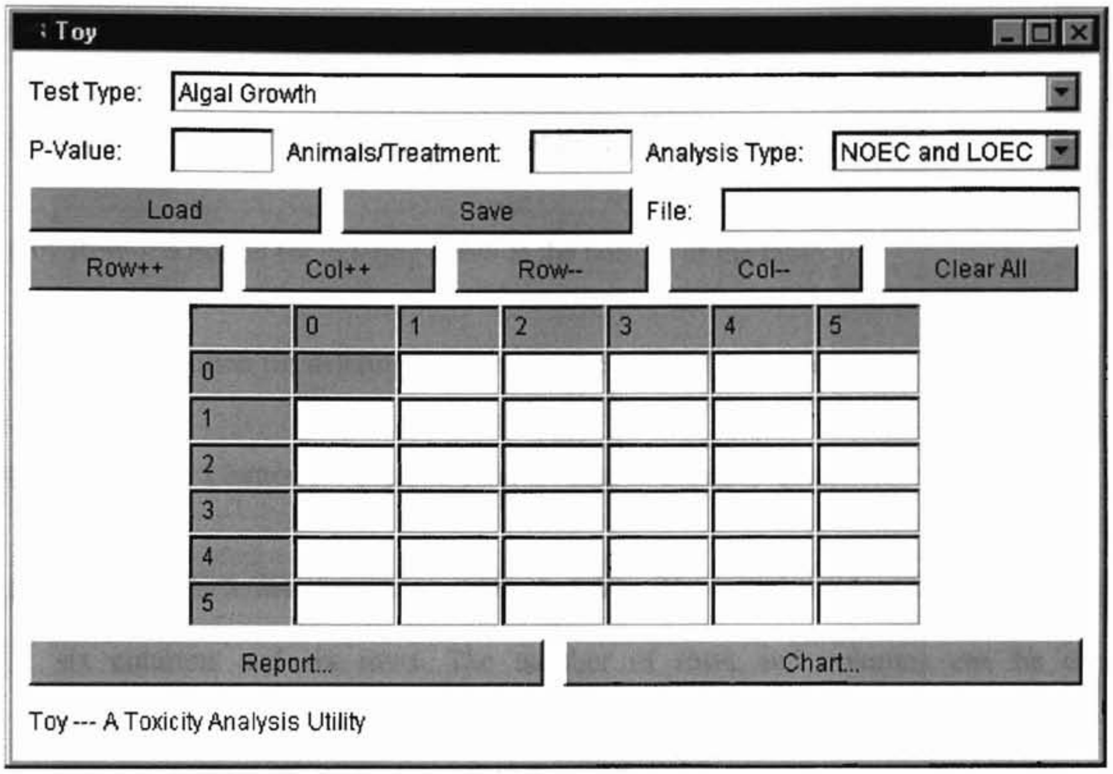


Figure 9: The Input Window

(1) Test Type: a choice field that contains a list of test types for users to choose.

(2) P-Value: a text field in which users enter the specific P-value.

(3) Animals/Treatment: a text field in which users enter the number of animals per treatment.

(4) Analysis Type: a choice field for users to specify a specific analysis type.

(5) File: a text field for users to enter the file name to be loaded from or saved to.

(6) Load: a button for loading a data file whose name is specified in the File field.

(7) Save: a button for saving data into a data file whose name is specified in the File field.

(8) Row++: a button for adding a row at the bottom of the table.

(9) Col++: a button for adding a column at the right of the table.

(10) Row--: a button for deleting a row at the bottom of the table.

(11) Col--: a button for deleting a column at the right of the table.

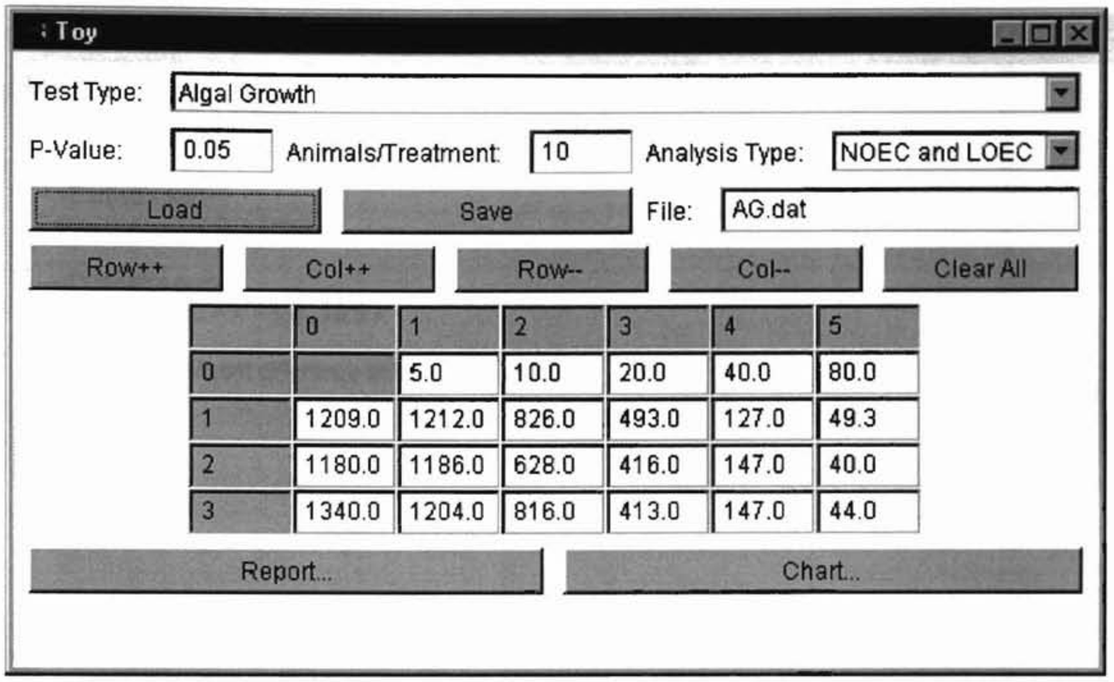
(12) Clear All: a button for users to clear all data in the data table.

(13) Data Table: a table for users to enter test data. The initial window contains a table with six columns and six rows. The number of rows and columns can be changed dynamically by users.

(14) Report: a button for users to output the test result to a report.

(15) Chart: a button for users to get the test result in a chart.

Figure 10 shows a sample of the InputWindow with test data loaded.


Toy

Test Type: Algal Growth

P-Value: 0.05 Animals/Treatment: 10 Analysis Type: NOEC and LOEC

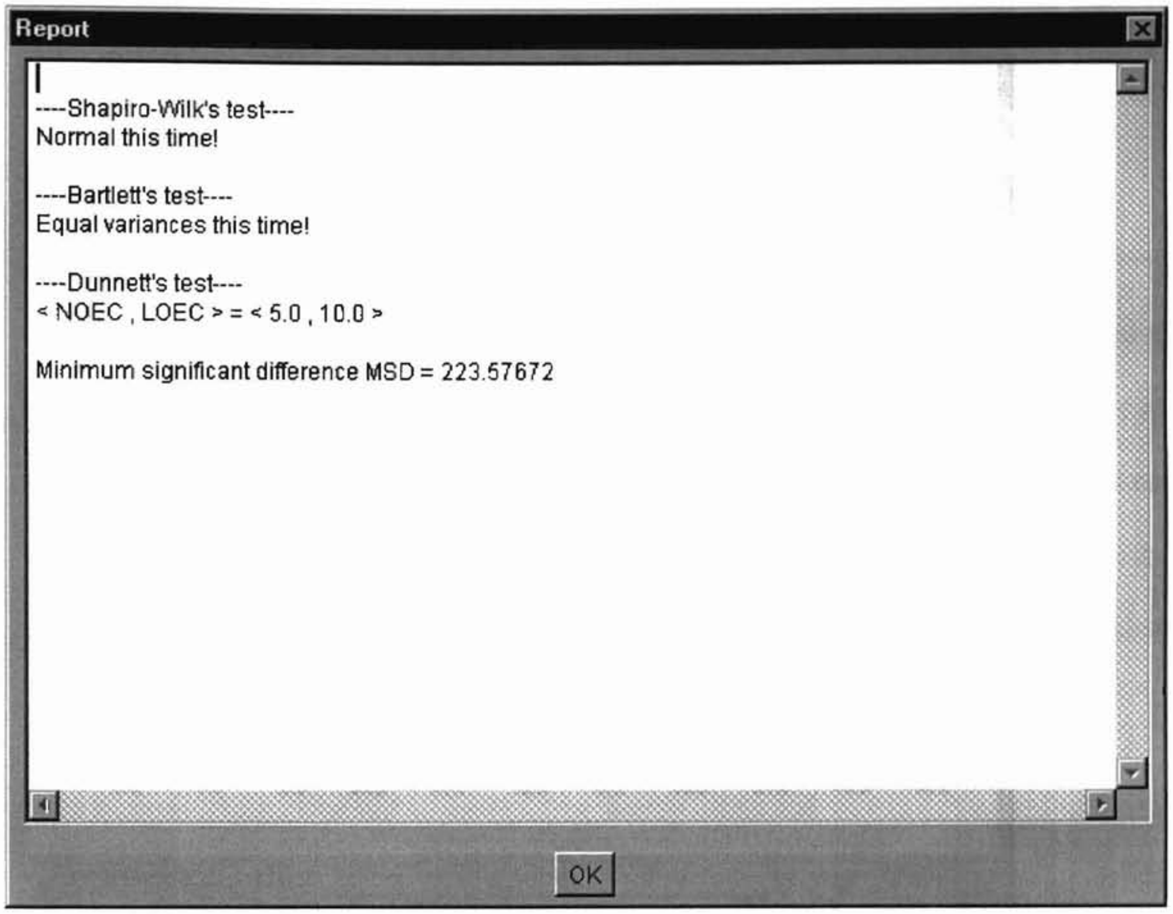
File: AG.dat

	0	1	2	3	4	5
0		5.0	10.0	20.0	40.0	80.0
1	1209.0	1212.0	826.0	493.0	127.0	49.3
2	1180.0	1186.0	628.0	416.0	147.0	40.0
3	1340.0	1204.0	816.0	413.0	147.0	44.0

Figure 10: The Input Window with Data

2. OutputWindow: OutputWindow provides a skeleton of a output window. It is used with other components, such as TextArea and Chart, to construct a complete output window.

If Analysis Type is NOEC and LOEC, the report window contains a trace of all tests and their results as shown in Figure 11. The values of NOEC, LOEC and MSD (Minimum Significance Difference) are included at the end of the report. If the Analysis Type is EC, IC and LC, report window would display the interpolation values of the test data.



```
----Shapiro-Wilk's test----
Normal this time!

----Bartlett's test----
Equal variances this time!

----Dunnett's test----
< NOEC , LOEC > = < 5.0 , 10.0 >

Minimum significant difference MSD = 223.57672
```

Figure 11: The Report Window

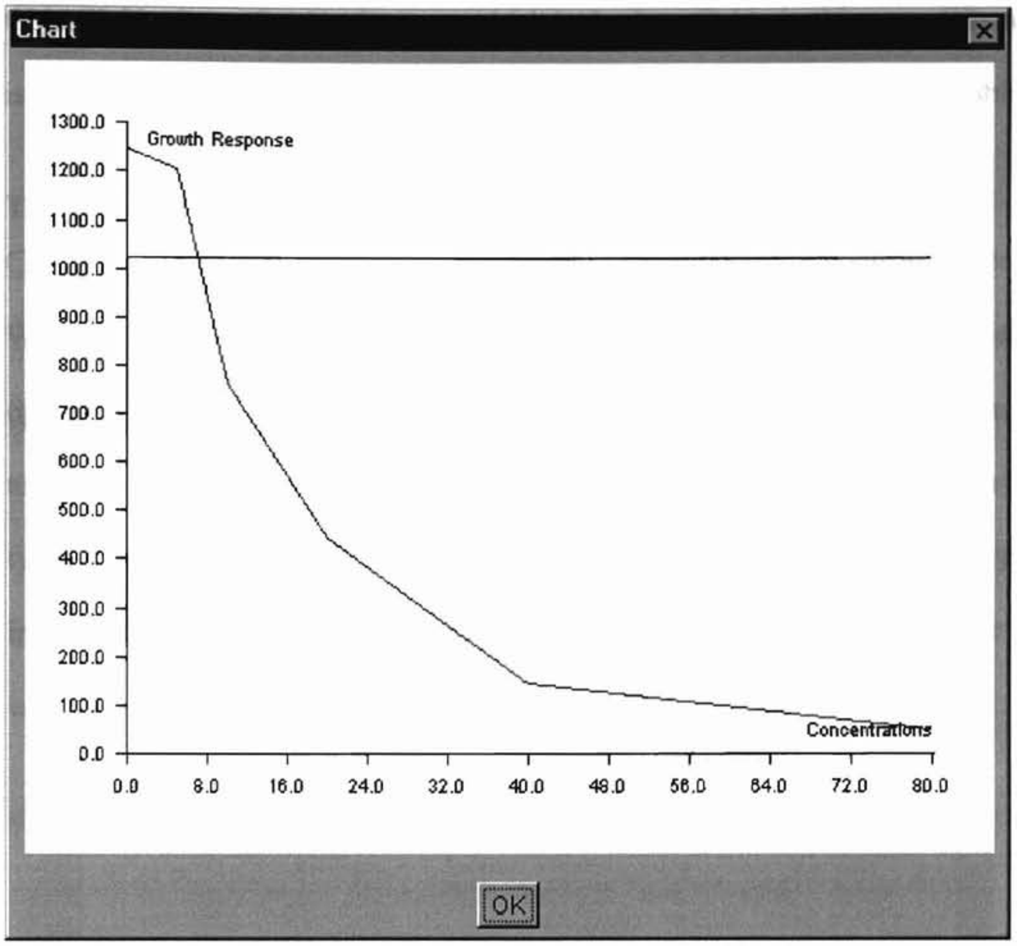


Figure 12: The Chart Window

3. Chart: it displays the critical value and mean replicates by two curves (Figure 12).

4. InputTable: it is a class used to implement an input table. It contains an ObjectTable. ObjectTable contains a vector. Objects can be dynamically inserted into or deleted from a vector. InputTable specializes ObjectTable to be a table of text fields. It is also responsible for displaying the table.

InputTable and ObjectTable uses an observer and observable synchronization model. ObjectTable is an object that can be observed. Whenever a change is made to the

ObjectTable, it notifies its observer, which is the InputTable in this case. When InputTable is notified, it updates the screen according to the current state of the ObjectTable.

The Controller Module

Controller is a coordinator between model and view modules. It controls the program flow. When the view module receives input, the controller is the one to contact models for further operations. For example, if the view wants to save a data file, it should pass input data together with a file name, to the controller. The controller converts input data to its correct type, and invokes the model module to validate the input data. If there is any error in the input data, controller will generate an exception and return it back to the view. The view will display an error message.

Chapter 4

CONCLUSIONS

In this thesis, the design and implementation of a software package for toxicity analysis with Java are presented. The analysis methods are based on documents provided by the United States Environmental Protection Agency (EPA).

The software package uses an object-oriented design. It is easier to be extended and maintained. The graphical user interface is user-friendly and provides a means for users to interact with the program. The software package handles a set of statistical tests.

Since the whole package is written in Java, it can be posted on a Web server. Anyone who has Java compatible browser installed can download it from the server and run it locally. This approach reduces local maintenance cost. On the other hand, the package can also be installed locally and run on a Java virtual machine.

BIBLIOGRAPHY

1. United States Environmental Protection Agency. *Short-Term Methods for Estimating the Chronic Toxicity of Effluents and Receiving Waters to Freshwater Organisms*, second ed. United States Environmental Protection Agency, 1989.

2. Sun Microsystems, Inc. *The Java Language Specification*, version 1.0 beta. Sun Microsystems, Inc., 1995.

3. Gosling, J. and McGilton, H. *The Java Language Environment: A White Paper*. Sun Microsystems, Inc., 1995.

4. Sun Microsystems, Inc. *The Java Virtual Machine Specification*, release 1.0 beta. Sun Microsystems, Inc., 1995.

5. Lemay, L. and Perkins, C. L. *Teach Yourself Java in 21 Days*, Sams.net Publishing, 1996.

6. Wirfs-Brock, R., Wilkerson, B., and Wiener L. *Designing Object-Oriented Software*. Prentice-Hall, 1990.

7. Lemay, L. *Teach Yourself Web Publishing with HTML 3.0 in a Week*, second ed. Sams.net Publishing, 1996.

8. Kehoe, B. *Zen and the Art of the Internet, a Beginner's Guide to the Internet*, fourth ed. Prentice-Hall, 1996.

9. Jacobson, Ivar, Maria Ericsson, Agneta Jacobson. *The Object Advantage:Business Process Re-Engineering with Object Technology*, Addison-Wesley, 1995.

10. Arthur J. Riel *Object-Oriented Design Heuristics* Addison Wesley 1995.

APPENDIX

User's Manual

The software package implements a set of toxicity analysis tests documented in *Short-Term Methods for Estimating the Chronic Toxicity of Effluents and Receiving Waters on Freshwater Organisms* by the Environment Protection Agency (EPA). Any platforms with Java capability are able to use this tool.

Installation

The program can be run as an application or an applet. Client-based installation is used when the tool is run as an application. Server-based installation is used when the tool is run as an applet.

When do a client-based installation, copy the whole software package into the local storage of a client machine. For server-based installation, a Web server is needed to serve all clients. A link to the Java program should be implemented on a Web page. When client browser loads the page, it loads the program automatically. The advantage of server-based installation is that no matter how many clients run the tool, only one installation on the server is needed.

Usage

Starting the program

1. Server-based

Download the program from the server and run it locally. To run the program in this way, users should have Java compatible browser installed. Then type the correct URL address to start the program.

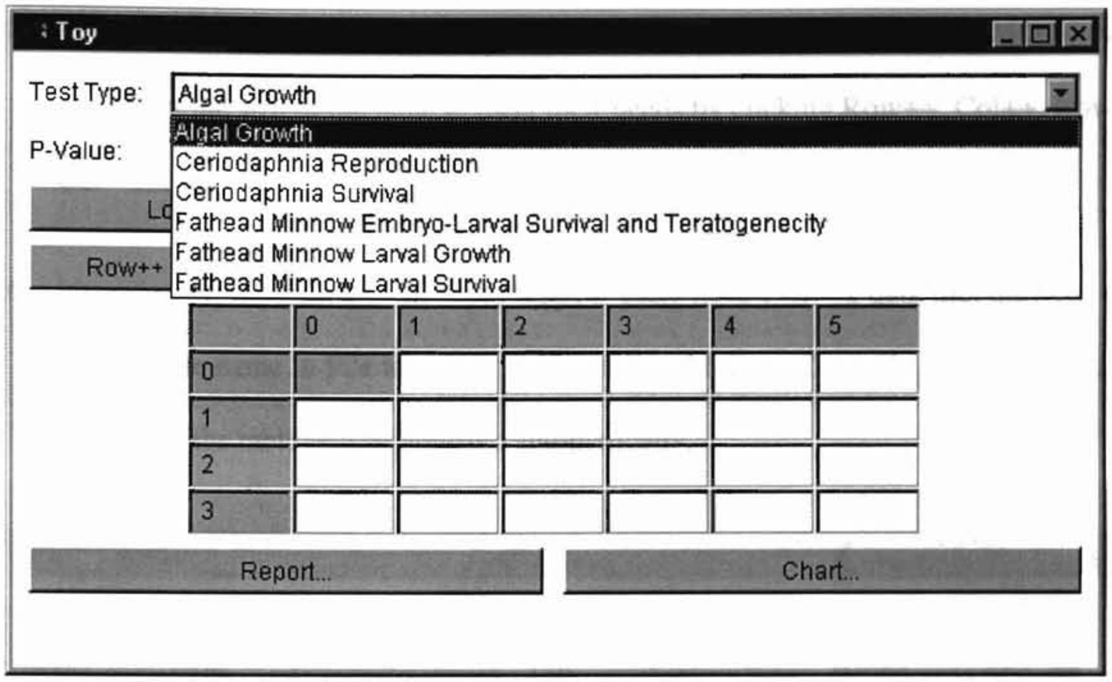
2. Client-based

Install the program (tool) locally and run it by a Java loader. Namely, using a Java loader to load the tool to run on a Java virtual machine.

After the software starts (Start up window is as Figure 9), the following steps can be applied to both server-based and client-based cases.

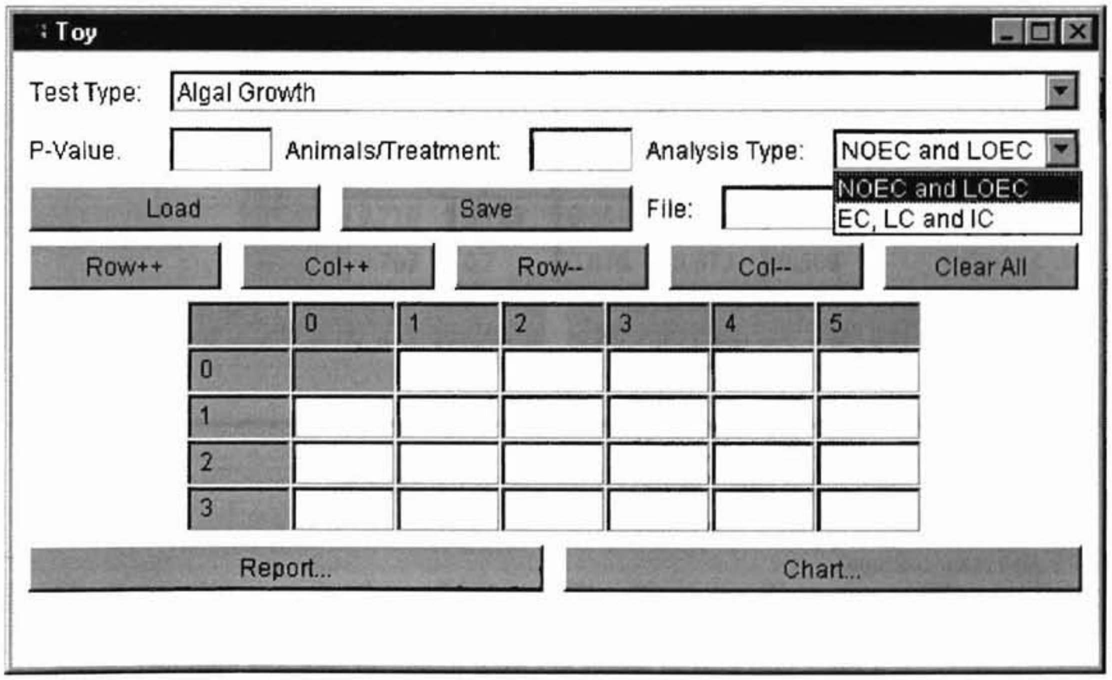
Entering Data

1. Choose Test Type, users can choose the test type by clicking the choice field droplist button (Figure 13).
2. Enter p-value in P-Value text field.
3. Enter animals per treatment in Animals/Treatment text field.
4. Select Analysis Type. Users can choose the analysis type by clicking the choice field droplist button (Figure 14).



Algal Growth
Ceriodaphnia Reproduction
Ceriodaphnia Survival
Fathead Minnow Embryo-Larval Survival and Teratogenecity
Fathead Minnow Larval Growth
Fathead Minnow Larval Survival

Figure 13: Choice Fields of Test Type

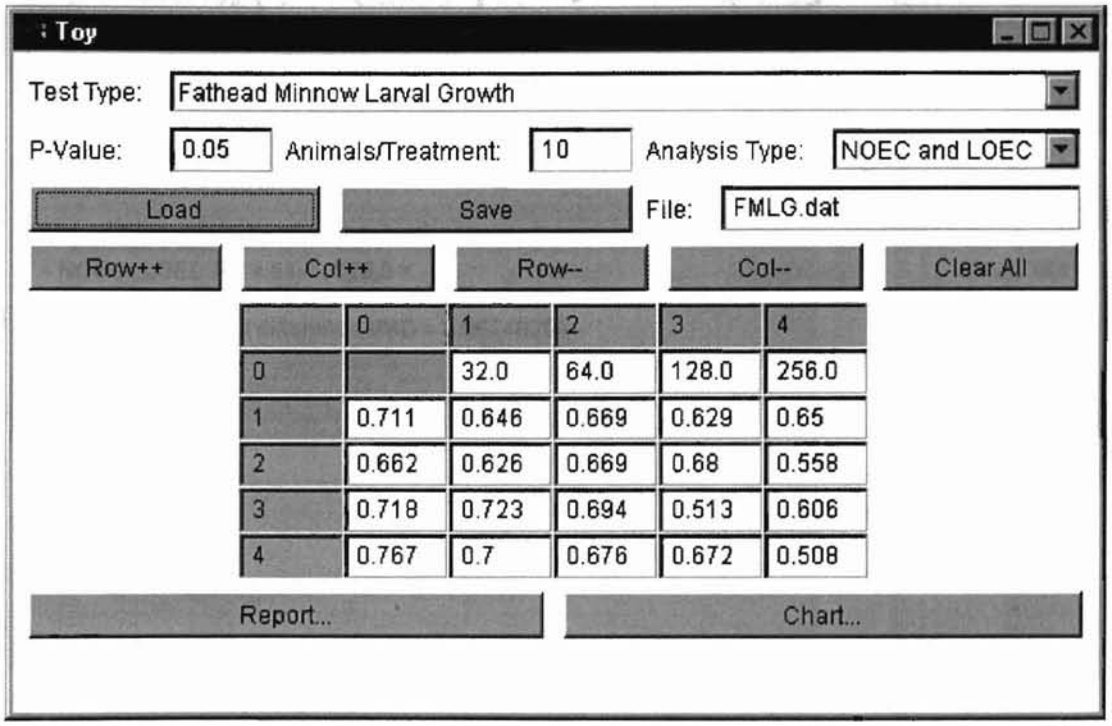


NOEC and LOEC
EC, LC and IC

Figure 14: Choice Fields of Analysis Type

5. Enter test data in the input table. The initial size of the table is six by six (6×6). Users can adjust the size of the table to meet their needs by clicking Row++, Col++, Row-- or Col-- button.

6. Instead of typing data into the input table, if users have existing data file, they can enter the data file name in File text field and click Load button. Data file will be loaded, and the size of the table will be adjusted automatically.



Test Type: Fathead Minnow Larval Growth

P-Value: 0.05 Animals/Treatment: 10 Analysis Type: NOEC and LOEC

File: FMLG.dat

	0	1	2	3	4
0		32.0	64.0	128.0	256.0
1	0.711	0.646	0.669	0.629	0.65
2	0.662	0.626	0.669	0.68	0.558
3	0.718	0.723	0.694	0.513	0.606
4	0.767	0.7	0.676	0.672	0.508

Figure 15: Load an Existing Data File

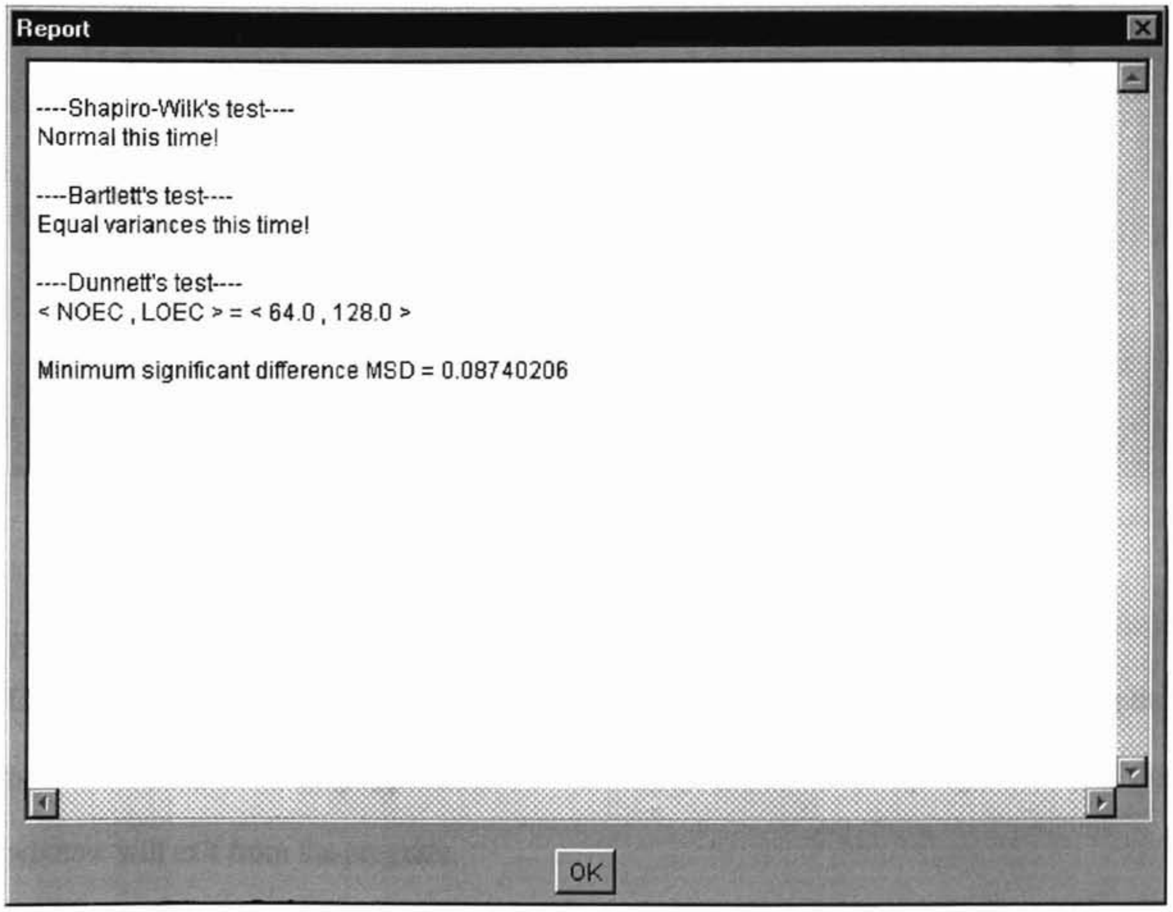
7. Users can also save the data table into a file. (Type the file name in File text field, click Save. The data file will be saved with the name in File text field.)

8. If users want to clear data in the input table, click Clear ALL button. Data in the input table will be erased.

9. Users can get the test report by clicking Report button. If the test data is not proper, error message will be displayed.

10. If users want to display test results in a chart, Chart button is the correct one to click.

Figure 15 is the window after inputting all fields for Fathead Minnow Larval Growth test, Figure 16 is its corresponding report, and Figure 17 is its corresponding chart.



```
----Shapiro-Wilk's test----
Normal this time!

----Bartlett's test----
Equal variances this time!

----Dunnett's test----
< NOEC , LOEC > = < 64.0 , 128.0 >

Minimum significant difference MSD = 0.08740206
```

Figure 16: A Report for Fathead Minnow Larval Growth Test

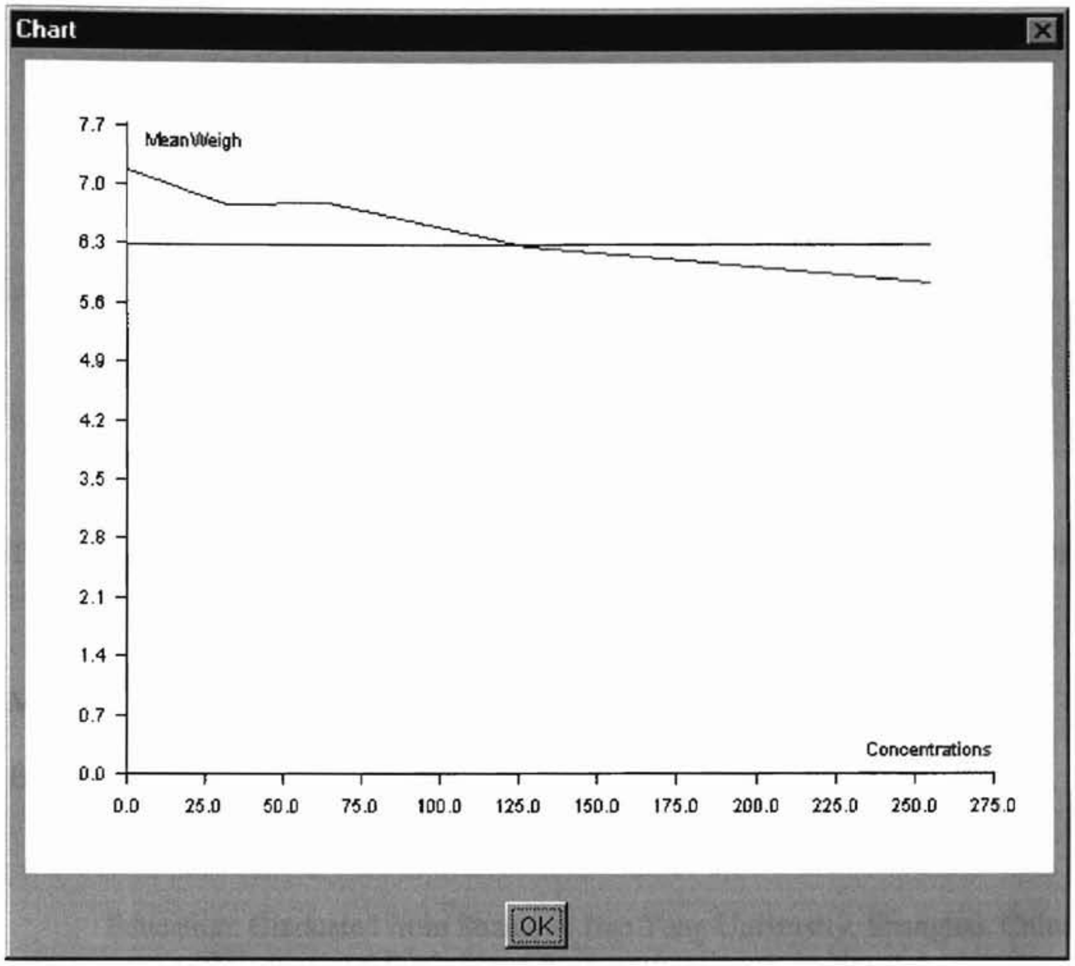


Figure 17: A Chart for Fathead Minnow Larval Growth Test Result

Exiting from the program

Users using browser to run this software package can exit from the program by closing the browser. For users running application version locally, click × at the up-right corner of the window will exit from the program.

2

VITA

Bei Zhu

Candidate for the Degree of

Master of Science

Thesis: DESIGN AND IMPLEMENTATION OF A SOFTWARE PACKAGE FOR TOXICITY ANALYSIS OF WATER SAMPLES ON FRESHWATER ORGANISMS WITH JAVA

Major Field: Computer Science

Biographical:

Personal Data: Born in Shanghai, China, in 1968.

Education: Graduated from Shanghai Jiao Tong University, Shanghai, China in 1991; received Bachelor of Engineering degree in Naval Architecture. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in July 1997.

Experience: University Computer Center Lab assistant, Oklahoma State University, January 1995 to May 1995; Software design engineer/test, Volt Computer Services, November 1996 to present.