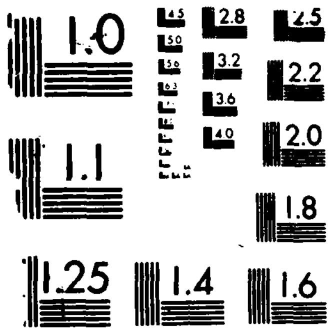
AD-A182 686 RELIABILITY ANALYSIS BASED ON EXPONENTIAL FAILURE DISTRIBUTIONS AND DEVEL (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI K N COLE 1/3

UNCLASSIFIED 19 DEC 86 AFIT/DS/ENG/86-06 F/G 12/3 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

AIR INSTITUTE OF TECHNOLOGY

RELIABILITY ANALYSIS BASED ON EXPONENTIAL FAILURE DISTRIBUTIONS AND DEVELOPMENT OF A SUPPORTING COMPUTER ENVIRONMENT

DISSERTATION

Kenneth N. Cole
Captain, USAF

AFIT/DS/ENG/86-06

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/DS/ENG/86-06

RELIABILITY ANALYSIS BASED ON EXPONENTIAL FAILURE DISTRIBUTIONS AND DEVELOPMENT OF A SUPPORTING COMPUTER ENVIRONMENT

DISSERTATION

Kenneth N. Cole
Captain, USAF

AFIT/DS/ENG/86-06

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A182 6[illegible]

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/DS/ENG/86-06	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL *(If applicable)* AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS *(City, State, and ZIP Code)* Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433	7b. ADDRESS *(City, State, and ZIP Code)*
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO

11. TITLE *(Include Security Classification)*
Reliability Analysis Based on Exponential Failure Distributions and Development of a Supporting Computer Environment.

12. PERSONAL AUTHOR(S)
Kenneth N. Cole, B.S., B.S.E.E., M.S.E.E., Capt., USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
PhD Dissertation	FROM ______ TO ______	1986 December 19	238

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	Reliability, Failure Analysis, Statistical Tests, Statistical Analysis
12	01		

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This investigation defines a generalized likelihood ratio test criteria for exponential failure distributions that is applicable to any number of sample with varying sample sizes. The formulas have been implemented in C-language computer programs as practical algorithms for reliability analysis.

In addition, a software system has been developed that allows simple construction of programs that will generate tables of test criteria percenta points or automatically compute the test value for multiple data files and report the result of the test. The system includes a screen oriented editor working with a line compiler for generating the test programs. The software is designed to operate on the UNIX, (a trade mark of Bell Labs) operating system.

[illegible]

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Panna B. Nagarsenker	22b. TELEPHONE *(Include Area Code)* (513) 255-7210 22c. OFFICE SYMBOL AFIT/ENC

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

AFIT/DS/ENG/86-06

RELIABILITY ANALYSIS BASED ON EXPONENTIAL FAILURE DISTRIBUTIONS AND DEVELOPMENT OF A SUPPORTING COMPUTER ENVIRONMENT

DISSERTATION

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Kenneth N. Cole, B.S., B.S.E.E., M.S.E.E.

Captain, USAF

December 1986

Approved for public release; distribution unlimited

AFIT/DS/ENG/86-06

RELIABILITY ANALYSIS BASED ON EXPONENTIAL FAILURE DISTRIBUTIONS AND DEVELOPMENT OF A SUPPORTING COMPUTER ENVIRONMENT

Kenneth N. Cole, B.S., B.S.E.E., M.S.E.E.
Captain, USAF

Approved:

Panna B Nagarsenker	2 Dec 1986
[illegible]	1 Dec 1986
[illegible]	2 Dec 1986
Henry B. Potoczny	2 Dec. 1986
[illegible]	1 Dec 1986

Accepted:

[illegible] 2 Dec 1986
Dean, School of Engineering

Preface

This dissertation presents the development of a method for reliability analysis based on exponential failure distributions. It has been said, if you can excuse the humor, that "we can learn from our mistakes." The current emphasis on reliability and maintainability is a recognition that we must prepare to learn from the failures before we start to design. Reliability is a study of the past that determines the future. I truly hope that this work will make that task easier.

I owe a great debt to my advisors, Dr. Panna B. Nagarsenker and Dr. B. N. Nagarsenker, for all their efforts on my behalf. They knew where I was going long before I even started on the right path. Their firm guidance was much appreciated.

I must, also, gratefully acknowledge the contribution made by Lt. Col. Walt Seward. His suggestions helped to make great improvements in the final draft.

Most of all, this work is for Suzan. She believed in me so strongly, she carried me past all the rough places. My love is hers, always.

Kenneth N. Cole

Table of Contents

Table of Contents (continued)

List of Figures

List of Tables

List of Symbols

$B_r(.)$	:Bernoulli Polynomial of degree r and order 0.
$\beta(p,q)$	:Beta Function of p and q.
C.D.F.	:Cumulative Density Function.
E(X)	:Expected value of a random variable X.
EXP(x)	:Exponential function with MTBF = x.
h(t)	:Hazard function of t.
H_0	:Null hypothesis that all samples have the same mean.
$I_x(p,q)$	:Incomplete Beta function defined as $I_x(p,q) = \int_0^x x^{p-1} \frac{(1-x)^{q-1}}{\beta(p,q)}$ where $\beta(.)$ is the Beta function.
i.i.d.	:Independent and Identically Distributed.
	:Percent of total failures that occur in the ith sample.
L	:Likelihood Ratio Test criteria λ to the power p/R $L = \lambda^{p/R}$
M.G.F.	:Moment Generating Function - the mathematical expectation of a random variable X such that $M(t) = E(e^{tx}) = \int e^{tx} f(x)\,dx$ if it exists.

List of Symbols (continued)

M.T.B.F.	:Mean Time Between Failures
p	:Number of data samples being compared.
p.d.f.	:Probability Density Function.
r_i	:Number of failures in the ith sample.
R	:Total number of failures in all samples.
R(t)	:Reliability function of t.
T_i	:Total time on test for items studied in ith sample.
$\hat{\Theta}_i$	:Maximum likelihood estimator of mean life in ith sample.
$\tilde{\Theta}$	:Maximum likelihood estimator of mean life of entire population.
λ	:Likelihood ratio test criteria.
δ	:Adjustment factor (see Appendix for computation).
Θ	:Mean Time Between Failures (MTBF) or mean time before failure is equal to Mean Time Till Failure (MTTF) for a single parameter exponential failure distribution.
$\equiv$	:Equivalence symbol - denotes the equivalence of two statements.
$\sim$	:Is distributed

List of Symbols (continued)

$\exp(x)$	:Exponential function of x, e^x.
$\chi^2(x)$	:Chi Squared distribution with parameter x.
$G(a,b)$	:Gamma distribution with parameters a and b.
$\Gamma(x)$	:Gamma function of x.
$P(.)$	:Probability that the event will occur.

AFIT/DS/ENG/86-06

Abstract

One of the most frequently used methods for studying reliability is the comparison of failure data. The analysis of failure data can be made in a wide variety of ways, but the formulas used are usually restricted to special cases of the number of samples, the number of failures, and the expected underlying distribution of failures.

This investigation defines a generalized likelihood ratio test criteria for exponential failure distributions that is applicable to any number of samples with varying sample sizes. The formulas have been implemented in C-language programs as practical algorithms for reliability analysis.

In addition, a software system has been developed that allows simple construction of programs that will generate tables of test criteria percentage points or automatically compute the test value for multiple data files and report the result of the test. The system includes a screen oriented editor working with a line compiler for generating the test programs. The software is designed to operate on the UNIX (a trademark of Bell Labs) operating system.

Chapter I
Introduction

The management and maintenance of complex systems is an area of great importance to the Air Force, as well as to private industry. The achievement of reliable and maintainable equipment does not happen by accident. In the Air Force, it is mandated that these characteristics must be considered throughout the entire equipment life cycle, including the planning, designing, developing, producing and supporting efforts (Ref. MIL-STD-785A). A key concept in this task is the analysis of statistical data to determine the reliability and maintainability of those systems.

The formulas used to determine these characteristics can be complex and, often, change significantly for each set of data to be examined (Ref. MIL-STD-785A, MIL-STD-781C). New developments in numerical analysis are providing mathematical methods that allow computation of the necessary statistics across a useful range of data distributions.

Problem Statement

The goal of this research effort was to derive exact and asymtotic censored distributions in a computable form and to explore the development of a computer environment to apply statistical analysis to reliability problems.

Background

The general concepts of reliability and maintainability are well established. Reliability is used to describe how "failure free" a system is likely to be over a given time interval. In other words, reliability is the probability that a system will not fail over a specific number of operations or over a specified period of time. Maintainability is the likelihood that the system can be kept in a defined condition using prescribed maintenance methods. When we have designed and built a reliable system, we would like to keep it that way (Ref. AFR 80-5, Epstein 1953:437). Quantative evaluation of these characteristics has come with the advent of more complex systems. The result has been the requirement that minimally acceptable standards be established and met by new systems in all phases of their life cycles (Ref. AFR 80-5, MIL-STD-785A). Engineers and managers must be familiar with the tools necessary to evaluate these characteristics.

These tools may be computer programs that can evaluate sets of data, mathematical formulas for the statistics that apply, or simply, rule-of-thumb judgements for making the decisions that are necessary. Each of these tools has advantages as well as problems.

Typically, rule-of-thumb solutions require expertise on the part of the user for good results. While this type of evaluation can be performed quickly and should not be ignored when applied by someone experienced in the field, there is a strong dependence on the user to understand the applicability of the rule and the implications of the result. This expertise is not always available to the system manager.

Correct application of formulas requires less expertise in the field of application but, in many cases, more in mathematics. Selection of the proper formula for a particular case, and its calculation and interpretation, may require a statistician. The statistic that is used often depends, first, upon the type of decision that is needed and, second, upon the amount or type of data available (Ref. AFM 80-5, Hogg and Craig 1978, Lawless 1982:29-31). For example, graphical examination or "goodness of fit" tests can be used to determine if the data could be from an assumed underlying density. The Chi-Square test would then be selected if the sample size is 30 or greater, and the hypothesized density is discrete. A decision based on calculations, without an assurance of correctness, is less than desireable.

Computer programs can provide the user with answers quickly and accurately. There is, again, the problem of applying the program where it is not designed to be used or writing a program which properly implements the mathematical solution. It is possible, however, to have a program that is properly designed and documented to allow the user to apply modifications or parameters that would adjust the program for the application required.

Reliability and maintainability characteristics are determined by the liklihood of an event. For reliability, it is the likelihood of a system failure. Maintainability relates to the likelihood that the system is in a certain condition. Usually, these characteristics are determined from a relatively small sample and projected for the future of a large number units. This projection can successfully be made if the expected frequency of the event can be determined from the data available.

In other words, if the distribution of occurances of an event is known, then the characteristics of that distribution can be assumed for the event and the probabilities of future events can be determined (Ref. Trivedi 1982:2-3).

In many of these situations, the problem becomes testing whether there are significant differences among the underlying distributions of a number of samples. The result could determine that the known characteristics of one group may be assumed for another, or that multiple samples may be considered as a whole. There are many examples of this type.

Example 1: A particular aircraft part is being built by two different manufacturers. Is there a significant difference in the reliability or maintainability of the two parts?

Example 2: A part is being maintained at several different equipment depots around the country. Is there a significant difference in the maintenance results at each depot?

Example 3: The failure times of several electronic computers are recorded. How close do the computations of the failure rate for each computer need to be before we can consider them to have the same reliability?

Example 4: A new design is to be tested. The results will be compared with data collected on the old model. Is there a significant difference in reliability between the two systems?

The frequency situations where we wish to compare the characteristics leads to the decision that a generalized method for comparing the underlying distributions should be one of the first problems to be studied.

As described in Chapter 2, the exponential failure distribution is, perhaps, the most often used distribution for lifetime modeling. Its use can be justified for most complex systems with long lifetimes and wherever the failure rate can be demonstrated to be constant over a period of time (Ref. Lawless 1982). This research examines the comparison of samples from exponential failure distributions.

A partial solution to the comparison of exponential distributions was developed by Epstein in 1953 for the restricted case of only two samples with the same number of failures in each sample. The goal of this dissertation is to extend this analysis to the general case for more than two samples with unequal numbers of failures in each sample. The mathematical formulas for these calculations are developed in the next chapters.

In addition, a computer environment is developed which demonstrates the application of these procedures to reliability problems. It should be noted that computer solutions for special cases (e.g. fixed number of samples, data sets of equal sample size, etc.) have been implemented using standard programming languages (Ref. Amell 1984, Lawton 1984). The environment developed in this dissertation is intended to provide a higher level of support when applying a general solution which can be parameterized to the special cases required.

Approach

This chapter has presented an introduction to this research.

The details of this work are provided in the following chapters.

The second chapter presents the mathematical background and development of the formulas needed for using the likelihood ratio criteria in the most general case. The next chapter gives the development of the statistic for testing for equality of the means of several exponential populations. The fourth and fifth chapters show the derivation of the exact and asymtotic forms of the cumulative distribution function (C.D.F.) of the test statistic.

The sixth chapter describes the application of these results to problems similar to the examples given in the first chapter. Then, the seventh chapter discusses a computer programming environment that would support this application. The final chapter contains the conclusions and recommendations for future work.

Chapter II

Background

To effectively analyze the reliability of Air Force systems, or any systems, the measurements and concepts of reliability must be precisely defined. In many cases, we wish to study the characteristics of a large number of nearly identical systems. Where actual operational conditions and complex systems are concerned, no two systems can be exactly alike. The failures of such systems cannot be expected to occur at exactly the same moment. But, if a large number of nearly identical systems are examined, when operating under similar conditions, then their failures can be discussed in probabilistic terms. The analysis of system reliability must be, therefore, based on concepts from probability theory.

These fundamental concepts of probability and reliability are presented in this chapter. Discussion begins with the applicability of the exponential probability distribution to the study of complex systems. Then, we focus on the definition of the exponential distribution and those related distributions that are necessary for our later presentations. The concepts of 'reliability', 'failure rate', and 'mean time between failures' are defined and their characteristics are examined for the case where the underlying distribution of failures is exponential. System lifetime models are, also, discussed for the exponential case. Finally, we define Type II censoring and show how the probability density function is modified when testing is suspended after a certain number of failures has occurred.

The concepts and definitions that are presented in this chapter can be found in many text books on probability or reliability theory (Ref. Lawless, 1982; Kapur, 1977; Trivedi, 1982). They are presented here in a concise form. In addition, several theorems, not generally found in these texts, are given in detail to provide the basis for the work presented in the following chapters.

Exponential Distribution

Most models for component and system life and reliability are based on the exponential distribution, which plays much the same role in reliability theory that the normal distribution plays in sampling theory. It has been pointed out by Drenick (Ref. 1960: 680-690) that under some (reasonably) general conditions, the distribution of the time between failures tends to the exponential as the complexity and the time of operation increase. Early work by Sukhatme (Ref. 1937) and then by Epstein and Sobel (Ref. 1953, 1954, 1955) and Epstein (Ref. 1954, 1960) provided several results and popularized the exponential distribution for lifetime modeling, particularly for industrial life testing. Many others have contributed statistical methods for applying the exponential distribution. The bibliographies of Mendenhall (Ref. 1958), Govindarajulu (Ref. 1964) and Johnson and Kotz (Ref. 1970: Chapter 18) give some idea of the large number of papers in this area prior to 1970 (Ref. Lawless 1982).

In addition, the exponential failure distribution is also important because it can be derived from the constant hazard function, or failure rate, as will be discussed in a later section. When considering complex systems with long (several years and thousands of hours of operation) lifetimes, the unstable periods of system installation and wearout can be set aside. The long period of a relatively fixed inventory maintained by routine repair procedures can be examined as a population with a stable, constant failure rate (see Figure 2.1).

This situation is typical of many systems in the Air Force inventory. Complex weapon systems like aircraft and motorized vehicles are, ideally, stable populations that are expected to have long useful lifetimes. Supply system inventory items that are repairable systems in their own right can also be modeled in this way. System managers need to monitor changes in the failure rate to properly evaluate the value of the maintenance programs being used.

Thus, study of the exponential failure distribution has been selected as the focus of this research. Mathematically, a random variable X is said to have an exponential distribution if it has a probability distribution function (p.d.f.) of

$$f(x) = \frac{1}{\theta} \exp\left(\frac{-x}{\theta}\right) \tag{2.1}$$

for $x > 0$. We can write " $X \sim EXP(\theta)$ " to mean that X is distributed exponentially with the p.d.f. of Eq. (2.1).

In our case, the random variable is the "time to failure" of an individual system. The term 'time' will refer to the unit of measure for the system, although the actual measurement may be in distance, operations, running time, flights, or any other measurable unit that is significant to a particular system. Generally, the lowercase letter 't' will be used to represent this 'time' in the formulas used in this paper.

Chi-Squared and Gamma Distributions

Having identified the exponential distribution as the assumed distribution of failure times for the samples in this study, there are two other distributions that are important in our work; the Chi-Squared and Gamma distributions. They are both directly related to the exponential distribution. These distributions will appear when we look at the total test time for type II censored samples, later in this chapter.

A random variable, X, is said to have a Chi-Squared distribution if the p.d.f. of X is

$$f(x) = \frac{\exp\left(\frac{-x}{2}\right) x^{\frac{n}{2}-1}}{2^{\frac{n}{2}} \Gamma\left(\frac{n}{2}\right)} \qquad (2.2)$$

for $x > 0$. We will write " $X \sim \chi^2(n)$ " where we mean to use the Chi-Squared distribution with the p.d.f. of Eq. (2.2).

A random variable, X, has a Gamma distribution with parameters a and b if the p.d.f. of X has the form

$$f(x) = \frac{\exp\left(\frac{-x}{b}\right) x^{a-1}}{b^{a} \Gamma(a)} \qquad (2.3)$$

for $x > 0$ and we will write " $X \sim G(a,b)$ " to denote this.

From an examination of the defining p.d.f.'s of the exponential, Chi-Squared, and Gamma distributions, we can deduce the following equivalences:

$$X \sim \chi^2(n) \equiv X \sim G(n/2, 2) \tag{2.4}$$

$$X \sim G(a,2) \equiv X \sim \chi^2(2a) \tag{2.5}$$

$$X \sim EXP(\theta) \equiv X \sim G(1,\theta) \tag{2.6}$$

and we must also make note of the relations expressed in the following theorems.

<u>Theorem 2.1:</u> Let $X \sim G(a,b)$, then $cX \sim G(a,cb)$, where c > 0.

<u>Proof:</u> Let y = cx. Then dy/dx = c and x = y/c.
The p.d.f. of Y is:

$$f(y) = f(x)\left|\frac{dx}{dy}\right| = \frac{x^{a-1}}{b^a\Gamma(a)}\exp\left(\frac{-x}{b}\right)\cdot\frac{1}{c} \tag{2.7}$$

So,

$$f(y) = \frac{(y/c)^{a-1}}{b^a\Gamma(a)}\exp\left(\frac{-y}{cb}\right)\cdot\frac{1}{c} = \frac{y^{a-1}\exp\left(\frac{-y}{cb}\right)}{(cb)^a\Gamma(a)} \tag{2.8}$$

Therefore, $cX \sim G(a,cb)$.

<u>Theorem 2.2:</u> (Ref. Barr and Zehna 1983: 258) Let $X_1 \dots X_n$ be independent random varaibles with Moment Generating Functions (M.G.F.) $M_{X_i}(t)$ for i = 1,2,...n.

Then the M.G.F. of $a_1X_1 + a_2X_2 + \dots + a_nX_n$ is

$$M(t) = M_{X_1}(a_1t) \cdot M_{X_2}(a_2t) \dots M_{X_n}(a_nt) \qquad (2.9)$$

where $M_{X_i}(t)$ is the M.G.F. of X_i.

<u>Theorem 2.3:</u> Let $X_1 \dots X_n$ be independent random variables such that $X_i \sim G(a_i, b)$. Then $\sum_{i=1}^{n} X_i \sim G(a,b)$, where $a = \sum_{i=1}^{n} a_i$.

<u>Proof:</u> By Theorem 2.2,

$$M_{X_1+X_2+\dots+X_n}(t) = \prod_{i=1}^{n} M_{X_i}(t)$$

$$= (1-bt)^{-a_1} \cdot (1-bt)^{-a_2} \dots (1-bt)^{-a_n}$$

$$= (1-bt)^{-(\sum_{i=1}^{n} a_i)} \qquad (2.10)$$

Therefore, $\sum_{i=1}^{n} X_i \sim G(a,b)$ where $a = \sum_{i=1}^{n} a_i$.

<u>Theorem 2.4:</u> Let $X_1 \dots X_n$ be identical and independently distributed (i.i.d.) $EXP(\Theta)$. Then $\sum_{i=1}^{n} X_i \sim G(n,\Theta)$.

<u>Proof:</u> $X_i \sim EXP(\Theta)$ implies $X_i \sim G(1,\Theta)$ by Eq. (2.6). Then, by Theorem 2.3, for a sum of independent Gamma distributions

$$\sum_{i=1}^{n} X_i \sim G\left(\sum_{i=1}^{n} 1, \theta\right) . \quad (2.11)$$

Therefore $\sum_{i=1}^{n} X_i \sim G(n, \theta)$.

<u>Theorem 2.5:</u> If $X \sim G(a,b)$, then $\frac{2X}{b} \sim \chi^2(2a)$.

<u>Proof:</u> $X \sim G(a,b)$ implies $\frac{2X}{b} \sim G(a, \frac{2b}{b}) \equiv G(a,2)$, by Theorem 2.1 and this is equivalent to $\chi^2(2a)$ by Eq. (2.5). Therefore, $\frac{2X}{b} \sim \chi^2(2a)$.

<u>Theorem 2.6:</u> Let $X_1 \ldots X_n$ be i.i.d. $EXP(\theta)$.
Then $\frac{2T}{\theta} \sim \chi^2(2n)$, where $T = \sum_{i=1}^{n} X_i$.

<u>Proof:</u> Since $X_1 \ldots X_n$ are i.i.d. $EXP(\theta)$, $T \sim G(n,\theta)$ by Theorem 2.4. Then by Theorem 2.5, $2T/\theta$ must be a Chi-Squared distribution with 2n degrees of freedom. That is

$$\frac{2T}{\theta} \sim \chi^2(2n) . \quad (2.12)$$

Now, we have defined the underlying failure distribution for the samples and the related distributions that we will need for later work. The following sections present the fundamental concepts of system reliability with specific reference to the the exponential failure distribution.

Mean Time Between Failures

The 'mean time between failures' (MTBF) is the expected time during which the system will perform satisfactorily. It is assumed that the system being examined is being renewed through maintenance and repairs. This is typical of complex military hardware systems during its useful life. However, this does not include systems that may be categorized as 'one shot' situations such as ammunition or solid fuel rocket engines. The discussion in this research is limited to maintainable systems that are repaired or replaced to support a constant inventory. The MTBF is defined as

$$MTBF = E(X) = \int_{\phi}^{\infty} t\, f(t)\, dt \qquad (2.13)$$

where f(t) is the probability density function defining the distribution of failures. The MTBF for the exponential density function can be written as

$$MTBF = \int_{\phi}^{\infty} t/_{\Theta} \exp(^{-t}/_{\Theta})\, dt = \Theta \qquad (2.14)$$

So, the MTBF is equivalent to the defining parameter of the exponential failure distribution function, theta, from Eq. (2.1). The failure distribution function is, also, directly related to the reliability function defined in the next section.

Reliability Function

'Reliability' can be defined as the probability that the system will meet its requirements under given conditions for a specified unit of cycles. These cycles may be measured in time, completed operations or distance where applicable to the system being studied.

It is obvious that the purpose intended for the system actually determines the type of reliability measure that is most meaningful. The degree of function necessary for the system to perform its mission can vary among systems and among the different tasks performed by a single system. Each operation of a system may have a different reliability. Also, the reliability of a system may vary at different points in its life, whether that life is measured in time, operations or distance.

In the following discussions, the term 'failure' is used to mean the system is not able to perform as required. It must be noted that this failure, in the case of a large or complex system, refers only to a particular operation under study. There may, or may not, be other operations which are dependent upon the success of this one and their respective reliabilities will, therefore, be effected.

The reliability function is the probability that the system will not fail before a certain time. Mathematically, this can be written as

$$R(t) = P(x > t) \tag{2.15}$$

where X is a random variable denoting the time of failure. If X has a density function f(t), then

(2.16)

If the time to failure is described by the exponential density of Eq. (2.1) then the reliability function can be written as

$$R(t) = \int_t^{\infty} \frac{1}{\theta} \exp(-y/\theta)\, dy = \exp(-t/\theta) \qquad (2.17)$$

for $t > 0$. So, we can see that the reliability function for a system having an exponential failure density is dependent only upon the MTBF of that density. Thus, any comparison we can make between the sample's MTBFs is equivalent to a comparison of the corresponding reliabilities of the systems represented by the samples.

At the beginning of this chapter we stated that the exponential failure distribution could be assumed for a system demonstrating a constant failure rate. More precisely stated, the hazard function for the system must be constant. The terms 'failure rate', 'hazard function' and their relation to the system 'lifetime model' are discussed in the next sections.

Failure Rate

The likelihood of failure occuring within a certain period of time $[t_1, t_2]$ can be expressed as the difference of the reliabilities at each time. Thus, the integral of the failure density, over a specific period of time, can be expressed as either

$$\int_{t_1}^{t_2} f(y)\, dy = \int_{t_1}^{\infty} f(y)\, dy - \int_{t_2}^{\infty} f(y)\, dy \qquad (2.18)$$

or

$$\int_{t_1}^{t_2} f(y)\,dy = R(t_1) - R(t_2) \tag{2.19}$$

The rate at which failures occur during a time period is defined as the likelihood that a failure per unit time occurs in the interval, assuming that a failure has not occured before the beginning of the interval. This 'failure rate' is

$$H_{t_1,t_2}(t) = \frac{R(t_2)-R(t_1)}{(t_2-t_1)\,R(t_1)} \tag{2.20}$$

This failure rate is shown as a function of time. As stated before, the unit might be time, operations, or distance, depending the nature of the system under study.

Hazard Function

The 'hazard function' is the instantaneous failure rate. The hazard function indicates the change in the failure rate at any moment in the life of the system. This is often more significant than the failure rate at that moment, because it indicates whether the system reliability is improving or decreasing. For example, the stabilization of the instantaneous failure rate could indicate the end of the system installation phase or break-in period. Likewise, an increase in the hazard function value could indicate improper maintenance procedures or the beginning of system wearout.

Equations may be derived, showing the relationships among the three functions; the failure distribution f(t), the reliability function R(t), and the hazard function h(t). These equations show that any two of these functions can be obtained from the third.

$$h(t) = \frac{f(t)}{R(t)} \tag{2.21}$$

$$f(t) = h(t)\exp\left[-\int_{\emptyset}^{t} h(y)\,dy\right] \tag{2.22}$$

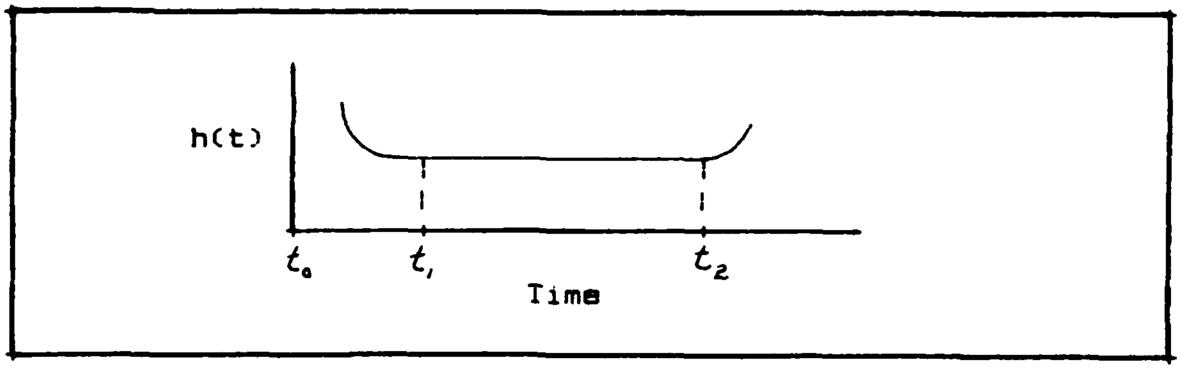
$$R(t) = \exp\left[-\int_{\emptyset}^{t} h(y)\,dy\right] \tag{2.23}$$

When the failure density function is exponential, the hazard function becomes h(t) = 1/MTBF, where MTBF = , from Eq. (2.14).

Lifetime Model

The 'lifetime model' is a statement of how the hazard function is expected to vary over the life of the population. Figure 2.1 shows a typical case with three areas of variance. From $t_\emptyset$ to t_1 represents the failures due to material or manufacturing defects. The second phase, from t_1 to t_2 represents the random failures associated with unusual or extreme conditions. Finally, the remainder of the curve, beyond t_2, shows an increase in failures due to wearout of the equipment.

This study is concerned with the period of time where the failures can only be attributed to unusual conditions. Between t_1 and t_2, the hazard function is constant. If the hazard



(Ref. Kapur and Lamberson 1977:22)

Figure 2.1 Hazard Function Over System Lifetime

function is given by

$$h(t) = 1/\theta \tag{2.24}$$

where θ is a constant, the functions for failure density and reliability can be found. Eq. (2.22) can be used to find the reliability function.

$$R(t) = \exp\left[-\int_{\emptyset}^{t} h(y)\,dy\right] = \exp(-t/\theta) \tag{2.25}$$

and Eq. (2.22) can be used to find the p.d.f. for failures.

$$f(t) = h(t)\exp\left[-\int_{\emptyset}^{t} h(y)\,dy\right] = \frac{1}{\theta}\exp\left(\frac{-t}{\theta}\right) \tag{2.26}$$

Thus, the assumption of a constant hazard function implies an exponential density function for system failures.

That completes our discussion of the exponential failure distribution and how it relates to the cases we wish to study. We now turn our attention to the data collected for the samples to be compared.

Type II Censoring

Actual data of the failures in a population of systems often comes with restrictions that make the analysis more difficult. It is rare that the failure time is known for every member of the population, or, even, for every member of the sample. 'Censoring' occurs when the exact failure times are known only for a portion of the population. Right censoring is the case when, for a portion of the population, it is known only that they did not fail before a certain time. Right censoring is very common in failure, or lifetime, analysis.

One form of right censoring is called Type II censoring. A Type II censored sample is one in which only the r smallest failure times are recorded in a random sample of n items. The value of r is determined before the test is begun. All n items are tested simultaneously, but the test is terminated when the first r items have failed. The probability density function of the samples with known failure times can be computed by

$$\frac{n!}{(n-r)!} f(t_1) \cdot f(t_2) \ldots f(t_r) \cdot R(t_r) \qquad (2.27)$$

where f(t) is the p.d.f for each member of the sample that failed at time t_i and $R(t_r)$ is the reliability function for the remaining (n - r) members of the population.

If the members of the sample have an exponential failure distribution, Eq. (2.1), with the reliability function of Eq.

(2.25), Eq. (2.27) becomes

$$\frac{n!}{(n-r)!} \exp\left[-\frac{1}{\theta}\left(\sum_{i=1}^{r} t_i + (n-r)t_r\right)\right] \tag{2.28}$$

which is the joint probability density function of all the members of the sample.

Now, we can state a theorem about the total testing time for all the members of a type II censored sample.

Theorem 2.7: Let T be the total time on test defined as

$$T = \sum_{i=1}^{r} t_i + (n-r)t_r \tag{2.29}$$

where the t_i's are distributed exponentially with p.d.f.'s of Eq. (2.1) for i = 1...r. Then T is a random variable whose distribution is $G(r,\theta)$.

Proof: Let $t_1, t_2, \dots t_r$ be the first r ordered observations of a random sample of size n from an exponential distribution described by Eq. (2.1). Then we can show that the quantities W_i for i = 1,2...r, defined by

$$W_1 = nt_1$$
$$W_i = (n-i+1)(t_i - t_{i-1}) \tag{2.30}$$

for i = 2...r, are i.i.d., also with p.d.f. (2.1).

Let $T = \sum_{i=1}^{r} W_i$. Then the Jacobian is given by

$$\left|\frac{(W_1 \ldots W_r)}{(t_1 \ldots t_r)}\right| = \frac{n!}{(n-r)!} \tag{2.31}$$

So,

$$f(\omega_1 \ldots \omega_r) = f(t_1 \ldots t_r)\frac{n!}{(n-r)!} \tag{2.32}$$

$$= \frac{1}{\Theta^r} \exp\left(-\sum_{i=1}^{r} \frac{\omega_i}{\Theta}\right) \tag{2.33}$$

$$= \prod_{i=1}^{r} \frac{\exp(-\omega_i/\Theta)}{\Theta} \tag{2.34}$$

Therefore, the W_i's are i.i.d. $EXP(\Theta)$, which implies that their sum, $T \sim G(r,\Theta)$.

In this chapter we have described the exponential failure distribution and discussed its importance in the study of complex systems or systems demonstrating a constant hazard function. We have defined the reliability function and demonstrated how a comparison of system reliabilities may be accomplished by a comparison of the underlying failure distribution functions. In limiting our study to the exponential case, we can directly relate a comparison of the MTBF for a system to its reliability.

We have also defined the type II censored sample and shown, by using relationships between several related distributions, that the total time on test, for such a sample, is a random variable with a Gamma distribution defined by the number of failures, r, and the MTBF of those failures.

We are now ready to define the likelihood ratio test for comparing the MTBF's of several type II censored samples from populations with exponential failure distributions. This test statistic is presented in the next chapter.

Chapter III

The Test Statistic and Its Moments

As we suggested in the first chapter, there are many cases where we must determine if different samples could have been taken from the same population. Generally, this is done by comparing the values of specific parameters of the population's distribution (e.g. mean, variance, or both). The value of the parameter is estimated from the data for each sample and then compared to an estimate of the same parameter for the entire population. Of course, to make these estimates comparable, we must assume that all the sample populations have the same underlying distribution. We are assuming an exponential failure distribution for all sample populations.

In this chapter, we define the test statistic for the null hypothesis that the samples have equal means. Then we derive the moments of the statistic. In the next chapters, we will develop the density functions that can be used to compute significance points of the test statistic.

Likelihood Ratio Test

One method commonly used for testing composite hypotheses is the Likelihood Ratio Test (LRT). As the name suggests, this is a ratio of the two likelihood functions. We can define this more precisely as follows.

First, we define the likelihood function of the sample. Let $t_1 \ldots t_n$ be the failure times of a sample of size n. Then the likelihood function that the set of failures, t, has occurred given the MTBF for the underlying distribution, $L(t;\Theta)$, is defined by

$$L(t;\Theta) = \prod_{i=1}^{n} f(t_i;\Theta), \tag{3.1}$$

where $f(t_i;\Theta)$ is the p.d.f. of t_i for i = 1,2,...,n.

Now, let D be a parametric space of dimension p. Suppose that $H_\emptyset$ is the null hypothesis that a collection of parameters $\Theta = (\Theta_1, \Theta_2, \ldots \Theta_p)$ are allowed to take values in a region within D.

$$H_\emptyset : \Theta \in D_\emptyset \tag{3.2}$$

when $D_\emptyset \in D$. Under the alternative hypothesis, H_1, the parameters are allowed to take values is some other region D_1, such that

$$H_1 : \Theta \in D_1 \tag{3.3}$$

where $D = D_\emptyset \cup D_1$.

As we allow the parameters to vary over the region $D_\emptyset$ there is some point, or set of points, where the likelihood function takes its maximum value for the particular sample X, $\max_{\Theta \in D_\emptyset} L(X;\Theta)$. If the parameters are allowed to range over the entire space D, then

$$\max_{\Theta \in D} L(X;\Theta) \geq \max_{\Theta \in D_\emptyset} L(X;\Theta) \tag{3.4}$$

The ratio L(X) where

$$L(x) = \frac{\max_{\Theta \in D_\emptyset} L(X;\Theta)}{\max_{\Theta \in D} L(X;\Theta)} \qquad (3.5)$$

is called the likelihood ratio criterion for testing $H_\emptyset$ against H_1. Our test of $H_\emptyset$ versus H_1 will be to reject $H_\emptyset$ when

$$L(x) < C \qquad (3.6)$$

where the constant C is chosen such that $P[L(x) < C \mid H_\emptyset] = \alpha$ where α is the desired level of significance (Ref. Blake 1979:309). For example, if the level of significance is 0.05, then there is a five percent probability that the tested hypothesis will be rejected when it is actually true. In other words, a five percent chance of mistakenly saying the MTBF's of the samples are not the same.

Our objective is to obtain the cumulative distribution function of the likelihood ratio, in a computational form, so that the value of C can be found for any level of significance, number of samples or number of failures. The first step is to define, precisely, the likelihood ratio criteria for multiple samples.

Derivation of the LRT for Testing p Exponential Distributions

Suppose that there are p exponential distributions as shown in Figure 3.1. Each has an exponential failure distribution function as defined by Eq. (2.1). Now, select a sample of size n_i from each population. The samples are not, necessarily, of

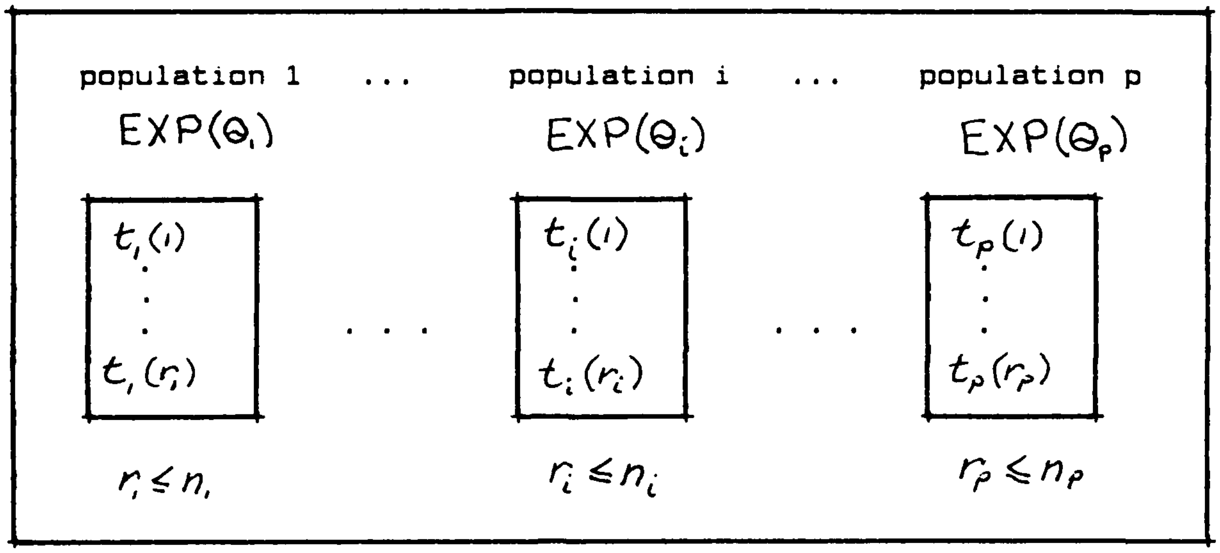


Figure 3.1. Samples From Several Exponential Populations

the same size. Failure data is collected for each sample as described for type II censored samples. Let

$$t_i(1) \leq t_i(2) \leq t_i(3) \leq \ldots \leq t_i(r_i) \tag{3.7}$$

be the failure times of the first r_i items in the i-th sample. Let the total test time for each sample be

$$T_i = \sum_{j=1}^{r_i} t_i(j) + (n_i - r_i)\,t_i(r_i) \tag{3.8}$$

for i = 1 ... p. Then $T_i \sim G(r_i, \Theta_i)$ by Theorem 2.7. Now, we shall state a theorem defining the likelihood ratio criterion for testing the hypothesis that all the populations have the same MTBF against the general alternative. Then, we will find the moments of the test criteria to prepare for the derivation of the cumulative distribution function.

<u>Theorem 3.1:</u> The likelihood ratio criterion for testing

$$H_\emptyset : \Theta_1 = \Theta_2 = \dots = \Theta_p \tag{3.9}$$

for p exponential distributions, against the general alternative is given by

$$\lambda = \frac{\prod_{i=1}^{p} \left(T_i / r_i\right)^{r_i}}{\left(\sum_{i=1}^{p} T_i / R\right)^{R}} \tag{3.10}$$

where $R = \sum_{i=1}^{p} r_i$ and the T_i's are defined by Eq. (3.8).

<u>Proof:</u> We know that the joint p.d.f. of $t_i(1) \le t_i(2) \le \dots \le t_i(r_i)$ is given by

$$f(T_i ; \Theta_i) = \frac{n_i!}{(n_i - r_i)!} \left(\frac{1}{\Theta_i}\right)^{r_i} \exp\left(-T_i / \Theta_i\right) \tag{3.11}$$

for the i-th sample. So, the combined likelihood is

$$L(X ; \Theta_1, \Theta_2 \dots \Theta_p) = K \prod_{i=1}^{p} \left[\left(\frac{1}{\Theta_i}\right)^{r_i} \exp\left(T_i / \Theta_i\right)\right] \tag{3.12}$$

where

$$K = \prod_{i=1}^{p} \frac{n_i!}{(n_i - r_i)!}$$

Under the null hypothesis $H_\emptyset$, we have $\Theta_i = \Theta$ for all i, where Θ is unknown. So, the likelihood function under $H_\emptyset$ is given by

$$L(x;\Theta) = K\prod_{i=1}^{p}\left[\left(\frac{1}{\Theta}\right)^{r_i}\exp\left(T_i/\Theta\right)\right] \qquad (3.13)$$

We then estimate Θ by maximizing the likelihood.

Let $R=\sum_{i=1}^{p} r_i$. Then

$$L(x;\Theta) = K\left(\frac{1}{\Theta}\right)^{R}\exp\left(\sum_{i=1}^{p} T_i/\Theta\right) \qquad (3.14)$$

By taking the natural log

$$\ln L(x;\Theta) = \ln K - R\ln\Theta - \sum_{i=1}^{p} T_i/\Theta \qquad (3.15)$$

Taking the derivative

$$\frac{\partial \ln L(x;\Theta)}{\partial\Theta} = \emptyset - R/\Theta - \sum_{i=1}^{p} T_i/\Theta \qquad (3.16)$$

and setting this equal to zero the maximum for $L(x;\Theta)$ is found at

$$\tilde{\Theta} = \sum_{i=1}^{p} T_i/R \qquad (3.17)$$

Under the unrestricted model, we know that the maximum likelihood estimator of Θ is

$$\hat{\Theta}_i = T_i/r_i \qquad (3.18)$$

for the i-th sample (Ref. Trivedi 1982:483).

So, if we substitute our value for $\tilde{\Theta}$ from Eq. (3.17) into Eq. (3.13) we get

$$\max_{\Theta \in D_\phi} L(X;\Theta) = K \frac{1}{\left(\sum_{i=1}^{p} T_i/R\right)^R} \exp\left[-\sum_{i=1}^{p} T_i \Big/ \left(\sum_{i=1}^{p} T_i/R\right)\right] \tag{3.19}$$

and

$$\max_{\Theta \in D_\phi} L(X;\Theta) = K \frac{R}{\left(\sum_{i=1}^{p} T_i\right)^R} \exp(-R) \tag{3.20}$$

where K is defined by Eq. (3.13). For the unrestricted case, we substitute for $\hat{\Theta}_i$ from Eq. (3.18) into Eq. (3.12) giving

$$\max_{\Theta \in D} L(X;\Theta_1 \ldots \Theta_p) = K \prod_{i=1}^{p} \frac{\exp\left[-T_i/(T_i/r_i)\right]}{(T_i/r_i)^{r_i}} \tag{3.21}$$

$$= K \prod_{i=1}^{p} \left(\frac{r_i}{T_i}\right)^{r_i} \exp(-R) \tag{3.22}$$

and, again, K is defined by Eq. (3.13). So, the likelihood ratio from Eq. (3.20) and Eq. (3.22) is given by

$$\lambda = \frac{\max_{\Theta \in D_\phi} L(X;\Theta)}{\max_{\Theta \in D} L(X;\Theta_1 \ldots \Theta_p)} = \frac{R^R \prod_{i=1}^{p} (T_i)^{r_i}}{\left(\sum_{i=1}^{p} T_i\right)^R \prod_{i=1}^{p} (r_i)^{r_i}} \tag{3.23}$$

or

$$\lambda = \prod_{i=1}^{p} \frac{\hat{\Theta}_i^{r_i}}{\tilde{\Theta}^R} \tag{3.24}$$

when $\hat{\Theta}_i = T_i/r_i$ and $\tilde{\Theta} = \sum_{i=1}^{p} T_i/R$.

Derivation of the h-th moment of the Statistic

With the test statistic defined by Theorem 3.1, we now wish to find its moments which will be used in the next chapter to obtain the density function and the cumulative distribution of the test statistic. We begin by stating a theorem.

Theorem 3.2: The h-th moment of the statistic defined by Theorem 3.1 when $H_\emptyset$ is true is

$$E(\lambda^h) = \frac{R^R\Gamma(R)}{\Gamma(R(h+1))}\prod_{i=1}^{p}\frac{r_i^{-hr_i}\Gamma(r_i(h+1))}{\Gamma(r_i)} \qquad (3.25)$$

Proof: The h-th moment of the statistic in Eq. (3.10) is

$$E(\lambda^h) = \frac{R^{Rh}}{\prod_{i=1}^{p} r_i^{hr_i}}\int\cdots\int_{T_i>\emptyset}\prod_{i=1}^{p}\frac{T_i^{hr_i}P(T_i)}{\sum_{i=1}^{p}T_i}\,dT_1\ldots dT_p \qquad (3.26)$$

$$= \frac{d^r}{d\sigma^r}\frac{R^{Rh}}{\prod_{i=1}^{p} r_i^{hr_i}}E\left[\prod_{i=1}^{p}T_i^{hr_i}\exp\left(\sigma\sum_{i=1}^{p}T_i\right)\right]\Bigg|_{\substack{\sigma=\emptyset\\ r=-Rh}} \qquad (3.27)$$

where $P(T_i)$ is the p.d.f. of T_i. For justification of the operation in Eq. (3.27), see Stekloff (Ref. 1914) and Wilks (Ref. 1967). Now, define

$$\Phi(\sigma) = E\left[\prod_{i=1}^{p}T_i^{hr_i}\exp\left(\sigma\sum_{i=1}^{p}T_i\right)\right] \qquad (3.28)$$

or

$$\Phi(\sigma) = \underset{T_i > \emptyset}{\int \cdots \int} \prod_{i=1}^{p} T_i^{hr_i} P(T_i) \exp\left(\sigma \sum_{i=1}^{p} T_i\right) dT_1 \ldots dT_p \qquad (3.29)$$

Recall from Eq. (3.8) that $T_i \sim G(r_i, \theta_i)$. So, Eq. (2.3) can be used as the p.d.f. of the test times and Eq. (3.29) takes the form

$$\Phi(\sigma) = \underset{T_i > \emptyset}{\int \cdots \int} \prod_{i=1}^{p} T_i^{hr_i} \exp\left(\sigma \sum_{i=1}^{p} T_i\right) \prod_{i=1}^{p} \frac{T_i^{r_i - 1} \exp\left(\frac{-T_i}{\theta_i}\right)}{\Gamma(r_i)\, \theta_i} dT_1 \ldots dT_p \qquad (3.30)$$

Then

$$\Phi(\sigma) = \frac{1}{\prod_{i=1}^{p} \Gamma(r_i)\, \theta_i^{r_i}} \underset{T_i > \emptyset}{\int \cdots \int} \prod_{i=1}^{p} \left[T_i^{hr_i + r_i - 1} \exp\left(\sigma T_i - T_i/\theta_i\right) dT_i \right] \qquad (3.31)$$

$$= \frac{1}{\prod_{i=1}^{p} \Gamma(r_i)\, \theta_i^{r_i}} \prod_{i=1}^{p} \frac{\Gamma(hr_i + r_i)}{\left(1/\theta_i - \sigma\right)^{hr_i + r_i}} \qquad (3.32)$$

Which simplifies to

$$\Phi(\sigma) = \prod_{i=1}^{p} \frac{\theta_i^{hr_i}}{\Gamma(r_i)} \prod_{i=1}^{p} \frac{\Gamma(r_i(h+1))}{(1 - \sigma\theta_i)^{hr_i + r_i}} \qquad (3.33)$$

Under the null hypothesis we have $\theta_i = \theta$ for all i = 1,2...p. So, Eq. (3.33) can be simplified to

$$\Phi(\sigma) = \frac{\theta^{Rh}}{\prod_{i=1}^{p} \Gamma(r_i)} \prod_{i=1}^{p} \frac{\Gamma(r_i(h+1))}{(1 - \sigma\theta)^{Rh + R}} \qquad (3.34)$$

$$\Phi(\sigma) = \frac{\Theta^{Rh}}{\prod_{i=1}^{p} \Gamma(r_i)} \prod_{i=1}^{p} \Gamma(r_i(h+1))(1-\sigma\Theta)^{-s} \tag{3.35}$$

where $s = Rh+R$ and $R = \sum_{i=1}^{p} r_i$.

Now, we pause to consider the derivatives of $(1-\sigma\Theta)^{-s}$.

$$\begin{aligned} d/d\sigma\,(1-\sigma\Theta)^{-s} &= -s(1-\sigma\Theta)^{-s-1}(-\Theta) \\ &= s\Theta(1-\sigma\Theta)^{-(s+1)} \end{aligned} \tag{3.36}$$

$$\begin{aligned} d^2/d\sigma^2\,(1-\sigma\Theta)^{-s} &= s\Theta(-s-1)(1-\sigma\Theta)^{-s-2}(-\Theta) \\ &= s(s+1)(1-\sigma\Theta)^{-(s+2)}\Theta^2 \end{aligned} \tag{3.37}$$

$$\vdots$$

$$d^r/d\sigma^r\,(1-\sigma\Theta)^{-s} = s(s+1)\ldots(s+r-1)\Theta^r(1-\sigma\Theta)^{-(s-r)} \tag{3.38}$$

So,

$$d^r/d\sigma^r\,(1-\sigma\Theta)^{-s}\Bigg|_{\substack{\sigma=\emptyset \\ r=-Rh}} = \frac{\Theta^{-Rh}\Gamma(R)}{\Gamma(Rh+R)} \tag{3.39}$$

Using Eqs.(3.39) and (3.35) in Eq. (3.27) we have

$$E(\lambda^h) = \frac{R^{Rh}}{\prod_{i=1}^{p} r_i^{hr_i}} \prod_{i=1}^{p} \frac{\Gamma(r_i(h+1))}{\Gamma(r_i)} \frac{\Gamma(R)}{\Gamma(R(h+1))} \tag{3.40}$$

So,

$$E(\lambda^h) = \frac{R^{Rh}\Gamma(R)}{\Gamma(Rh+R)} \prod_{i=1}^{p} \frac{r_i^{hr_i}\Gamma(r_ih+r_i)}{\Gamma(r_i)} \tag{3.41}$$

is the h-th moment of the statistic given by Eq. (3.10). This proves the theorem.

In the following chapters we will use the moments of the statistic to derive the probability density function and then the exact and asymtotic forms for the C.D.F. of the test statistic.

Chapter IV

Exact Distribution of the Statistic

In the previous chapter, we defined the test statistic for comparing the MTBF's of several populations, with exponential distributions, by a ratio of likelihoods. In order to apply this statistic it is necessary to know the probability distribution for its values and have a cumulative distribution function from which we can compute the expected value of the statistic for a given probability.

In this chapter, we obtain the cumulative distribution function of the test statistic in a computational form. The inverse Mellin Transform is used to obtain the probability distribution from the moment function that was derived in the last chapter. Then the asymtotic expansion of the Gamma function is used to obtain a computable form of the cumulative distribution function of the statistic.

Lemmas

The following results are needed for the derivations presented later in this chapter.

Lemma 4.1: (Ref. Anderson 1958:204) The following expansion for the natural log of the Gamma function holds:

$$\ln\Gamma(x+h) = \ln(2\pi)^{1/2} + (x+h-1/2)\ln x - x - \sum_{r=1}^{m} \frac{(-1)B_{r+1}(h)}{r(r+1)x^r} + R_{m+1}(x) \qquad (4.1)$$

where $R_m(x)$ is the remainder, such that $|R_m(x)| \leq C/|x^m|$ for some constant C independent of x. $B_r(h)$ is the Bernoulli Polynomial of degree r and order unity defined by

$$\frac{t \exp(ht)}{\exp(t) - 1} = \sum_{r=\emptyset}^{\infty} \frac{t^r}{r!} B_r(h) \qquad (4.2)$$

The first three polynomials are

$$B_1(h) = h - 1/2$$
$$B_2(h) = h^2 - h + 1/6$$
$$B_3(h) = h^3 - (3/2)h^2 + (1/2)h$$

<u>Lemma 4.2:</u> (Ref. Kalinin and Shalaevskii 1971) Let the asymtotic series $\sum_{j=1}^{\infty} a_j x^j$ converge to the function g(x) in the neighborhood of x = Ø (or be its asymtotic expansion when x = Ø). We then have

$$\exp(g(x)) = 1 + \sum_{j=1}^{\infty} B_j x^j \qquad (4.3)$$

where the coefficients B_j satisfy the recurrence relation

$$B_j = \frac{1}{j} \sum_{k=1}^{j} k a_k B_{j-k} \qquad j > 1$$

and $B_\emptyset = 1$.

Lemma 4.3: (Ref. Norlund 1916, Nair 1940) Let $\Phi(t) = E(X^t)$ be the moment function of a random variable X with density function f(x). If $\Phi(t) = O(t^{-k})$ with the real part of t tending to infinity, then $\Phi(t)$ has the following exact representation as a factorial series:

$$\Phi(t) = \sum_{n=\emptyset}^{\infty} R_n \frac{\Gamma(t+a)}{\Gamma(t+a+k+n)} \tag{4.4}$$

where a is a constant.

Derivation of the Probability Functions

Now, we are ready to state our goal and derive the C.D.F. of the statistic.

Theorem 4.1: Let $L = \lambda^{p/R}$ Then the cumulative distribution function of L is given by

$$F(x) = K \sum_{i=\emptyset}^{\infty} R_i I_x(m+a, v+i) \frac{\Gamma(m+a)}{\Gamma(m+a+v+i)} \tag{4.5}$$

where

$$K = \left(\frac{2\pi}{p}\right)^v \prod_{i=1}^{p} k_i^{Rk_i - 1/2} \frac{\Gamma(R)}{\prod_{i=1}^{p} \Gamma(Rk_i)}, \quad \delta = \frac{\left(\sum_{i=1}^{p} k_i^{-1}\right) - 1}{6(p-1)}$$

$$m = \frac{(R-\delta)}{p},$$

also

$$v = \frac{p-1}{2}, \quad a = \frac{1-v}{2}, \quad k_i = r_i/R, \quad R = \sum_{i=1}^{p} r_i .$$

Proof: Replacing h by ph/R in Eq. (3.26) it follows that

$$E(L^h) = \frac{R^{ph}\Gamma(R)}{\Gamma(R+ph)} \prod_{i=1}^{p} \frac{r_i^{-r_i ph/R}\Gamma(r_i + (r_i/R) - ph)}{\Gamma(r_i)} \qquad (4.6)$$

Let $r_i/R = k_i$, so that $\sum_{i=1}^{p} k_i = 1$.

Then we have, from Eq. (4.6),

$$E(L^h) = \frac{\Gamma(R)}{\prod_{i=1}^{p}\Gamma(Rk_i)} \prod_{i=1}^{p} \frac{[k_i^{-phk_i}\Gamma(R+ph)k_i]}{\Gamma(R+ph)} \qquad (4.7)$$

Using the inverse Mellin Transform, the density function of L is given by

$$f(x) = \frac{\Gamma(R)}{\prod_{i=1}^{p}\Gamma(Rk_i)} \cdot \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} x^{-h-1} \frac{\prod_{i=1}^{p}[k_i^{-phk_i}\Gamma(R+ph)k_i]\,dh}{\Gamma(R+ph)} \qquad (4.8)$$

Define $pm = R - \delta$, where δ is a convergence factor chosen later. So, we have

$$f(x) = \frac{\Gamma(R)}{2\pi i \prod_{i=1}^{p}\Gamma(Rk_i)} \int_{c-i\infty}^{c+i\infty} x^{-h-1} \frac{\prod_{i=1}^{p}[k_i^{-phk_i}\Gamma(pm+\delta+ph)k_i]\,dh}{\Gamma(pm+\delta+ph)} \qquad (4.9)$$

Now, define $t = m + h$, so that $h = t - m$, and from Eq. (4.9) we have

$$f(x) = \frac{\Gamma(R)}{2\pi i \prod_{i=1}^{p} \Gamma(Rk_i)} \int_{c-i\infty}^{c+i\infty} x^{-t+m-1} \frac{\prod_{i=1}^{p} \left[k_i^{-pk_i(t-m)} \Gamma(pt+\delta)k_i \right] dt}{\Gamma(pt+\delta)} \quad (4.10)$$

or

$$f(x) = \frac{\Gamma(R)}{\prod_{i=1}^{p} \Gamma(Rk_i)} \; \frac{x^{m-1} \prod_{i=1}^{p} k_i^{pmk_i}}{2\pi i} \int_{c-i\infty}^{c+i\infty} x^{-t} \Phi(t)\, dt \quad (4.11)$$

where

$$\Phi(t) = \frac{\prod_{i=1}^{p} \left[k_i^{-ptk_i} \Gamma(pt+\delta)k_i \right]}{\Gamma(pt+\delta)}$$

Using Lemmas 4.1 and 4.2 we have

$$\Phi(t) = \left[(2\pi)^{\nu} p^{-\nu} \prod_{i=1}^{p} k_i^{\delta k_i - \frac{1}{2}} \right] t^{-\nu} \left[1 + \frac{q_1}{t} + \frac{q_2}{t^2} \dots \right] \quad (4.12)$$

where $\nu = \frac{(p-1)}{2}$.

and the q_r coefficients are determined by

$$q_r = \frac{1}{r} \sum_{k=1}^{r} k A_k q_{r-k}$$

with $q_0 = 1$.

and the A_r coefficients are given by

$$A_r = \frac{(-1)^r}{r(r+1)p^r} \; \frac{\left[B_{r+1}(\delta) - \sum_{i=1}^{p} B_{r+1}(\delta k_i) \right]}{k_i^r}$$

for $r \geq 1$.

Then Eq. (4.12) shows that

$$\frac{\bar{\Phi}(t)}{\left[(2\pi)^{v} p^{-v} \prod_{i=1}^{p} k_i^{\delta k_i - 1/2}\right]} = O(t^{-v}) \qquad (4.13)$$

with the real part of t tending to infinity. Therefore, by Lemma 4.3, $\bar{\Phi}(t)$ can be expressed as a factoral series:

$$\bar{\Phi}(t) = \left[(2\pi)^{v} p^{-v} \prod_{i=1}^{p} k_i^{\delta k_i - 1/2}\right] \sum_{i=0}^{\infty} R_i \frac{\Gamma(t+a)}{\Gamma(t+a+v+i)} \qquad (4.14)$$

where a is a convergence factor chosen such that $R_1 = \emptyset$, giving $a = (1-v)/2$ (see Appendix A). Also,

$$\sum_{j=\emptyset}^{k} R_{k-j} D_{k-j,j} = q_k$$

with $R_\emptyset = 1$ and

$$D_{i,r} = \sum_{k=1}^{r} k C_{i,k} D_{i,r-k}$$

with $D_{i,\emptyset} = 1$ and

$$C_{i,r} = (-1)^{r-1} \frac{B_{r+1}(a) - B_{r+1}(a+v+i)}{r(r+1)} .$$

Using Eq. (4.19) in Eq. (4.13), we integrate the series term by term (since a factoral series is uniformly convergent in a half plane (Ref. Doetsch 1971)), giving

$$f(x) = K \sum_{i=\emptyset}^{\infty} R_i \frac{x^{m+a+i}(1-x)^{v+i+1}}{\Gamma(v+i)} \tag{4.15}$$

where

$$K = \left[\frac{2\pi}{p}\right]^{v} \prod_{i=1}^{p} k_i^{Rk_i - \frac{1}{2}} \frac{\Gamma(R)}{\prod_{i=1}^{p} \Gamma(Rk_i)}$$

Now, choose δ such that $A_1 = \emptyset$ in Eq. (4.10).

This gives us

$$\delta = \left[\left(\sum_{i=1}^{p} k_i^{-1}\right) - 1\right] / 6(p-1) \tag{4.16}$$

(see Appendix A). The C.D.F. can now be obtained from Eq. (4.11) by integrating.

$$F(x) = K \sum_{i=1}^{\infty} R_i I_x(m+a, v+i) \frac{\Gamma(m+a)}{\Gamma(m+a+v+i)} \tag{4.17}$$

where K is defined by Eq. (4.15) and $I_x(\cdot)$ is the Incomplete Beta function with parameters p and q, defined by

$$I_x(p,q) = \int_{\emptyset}^{x} x^{p-1} \frac{(1-x)^{q-1}}{\beta(p,q)}$$

where $\beta(\cdot)$ is the Beta function.

We now have a computable form of the C.D.F. for the statistic L. This will allow us to directly determine the probability that the value of the statistic is less than or equal to a given value when the tested hypothesis, H_ϕ, is true. In the next chapter, we will derive an asymtotic form of this distribution that will be useful in computations where the numbers of samples and failures are high.

Chapter V

Asymtotic Expansion of the Exact Distribution

In Chapter 3, we defined a test statistic for comparing multiple samples from populations with exponential failure distributions. The distribution function, derived in Chapter 4, allows us to directly compute the probability that the statistic is less than or equal to a given value when the MTBF's of the populations are, in fact, equal. In cases where the number of samples or failures are large, it is useful to have an alternate method of calculating the cumulative probability. Ideally, this method would provide a more efficient calculation method and asytotically approach the exact value of the probability for larger parameter values.

In this chapter we obtain the asymtotic expansion of the test statistic, $L = \lambda^{P/R}$. This asymtotic form of the distribution is valid for moderately large sample sizes (total number of failures above 100).

Lemmas

As in the derivation of the exact distribution, the definition of a lemma will help to simplify the derivation of the asymtotic expansion. The major step in the derivation of the asymtotic form of the C.D.F. is the expansion of a ratio of Gamma functions. Lemmas 4.1 and 4.2 can combined to form the following lemma.

Lemma 5.1: (Ref. Nagarsenker and Nagarsenker 1984:359) It follows immediately from Lemmas 4.1 and 4.2 that the following expansion for the ratio of two Gamma functions holds:

$$\frac{\Gamma(ms+a)}{\Gamma(ms+a+v+k)} = (ms)^{-(v+k)}\left[1+\frac{C_{k,1}}{Sm}+\frac{C_{k,2}}{(Sm)^2}\cdots\right] \qquad (5.1)$$

where the coefficients $C_{k,r}$ satisfy the following recurrence relation:

$$C_{k,r} = \frac{1}{r}\sum_{j=1}^{r} j A_{k,j} C_{k,r-j}$$

$$C_{k,\emptyset} = 1$$

and

$$A_{k,r} = \frac{(-1)^r}{r(r+1)}\left[B_{r+1}(a+v+k) - B_{r+1}(a)\right] .$$

Derivation of the Asymtotic Expansion

In proving the following theorem we expand Gamma functions found in the distribution Eq. (4.5).

Theorem 5.1: The asymtotic expansion of the distribution of L, in terms of m increasing, is given by

$$F(x) = I_x(m+a,v) + \sum_{r=1}^{k}\frac{S_r}{m^r} + O(m^{-k-1}) \qquad (5.2)$$

where

$$\delta = \left[\left(\sum_{i=1}^{p} k_i^{-1}\right) - 1\right] / 6(p-1) \quad , \quad m = \frac{R-\delta}{p} \quad , \quad v = \frac{p-1}{2} \quad , \quad a = \frac{1-v}{2} \; ,$$

as were stated for the exact form of the C.D.F. in Theorem 4.1. In addition,

$$S_r = \sum_{j=1}^{r} R_j I_x(m+a, v+i) W_{j,r-j}$$

and

$$W_{i,r} = \sum_{k=\emptyset}^{r} T_k C_{i,r-k}$$

$$T_r = -A_r$$

where A_r is defined in Eq. (4.12) and $C_{i,r}$ is defined by Eq. (4.14).

<u>Proof:</u> Using Lemma 4.1 on the definition of K in Eq. (4.5) we obtain

$$K = m^v \left[1 + \frac{T_2}{m^2} + \frac{T_3}{m^3} \cdots \right] \tag{5.3}$$

when δ is chosen so that $T_1 = \emptyset$. This results in the same value for δ (Eq. (5.2)) that was necessary to make $A_1 = \emptyset$ in the previous chapter, Eq. (4.16) (see Appendix A). Also, the T_r coefficients are related to the A_r coefficients of Eq. (4.12) by

$$T_r = -A_r \; ; \quad r \geq 2 . \tag{5.4}$$

We can now use Lemma 5.1 in Eq. (5.3) to obtain

$$F(x) = m^{v}\left[1+\frac{T_1}{m}+\frac{T_2}{m^2}\cdots\right]\sum_{i=\emptyset}^{\infty} R_i I_x(m+a, v+i)\left[1+\frac{C_{i,1}}{m}+\cdots\right] \tag{5.5}$$

which can easily be expressed in the form given in Eq. (5.3).

The following corollary to Theorem 5.1 can be used to further simplify the calculations when only two samples are considered and the number of failures is the same in each sample.

Corollary 5.1: If $p=2$ and $k_1=k_2=\frac{1}{2}$ then the asymtotic expansion of the distribution becomes

$$F(x) = I_x(m+a, v) \tag{5.6}$$

and is precisely the same as the exact distribution of L.

Proof: First, when $p=2$ Theorems 4.1 and 5.1 give us

$$v = \frac{p-1}{2} = \frac{1}{2} \tag{5.7}$$

$$a = \frac{1-v}{2} = \frac{1}{4} \tag{5.8}$$

and $k_1 = k_2$ means that

$$\delta = \left[\left(\frac{1}{k_1}+\frac{1}{k_2}\right)-1\right]/6(2-1) = \frac{1}{2} \tag{5.9}$$

and

$$m = \frac{R-\delta}{p} = \frac{R}{2}-\frac{1}{4} \tag{5.10}$$

In Theorem 5.1 (Eq. (5.2)), it can be shown that the S_r coefficients all become zero for these values. In Theorem 4.1 (Eq.(4.5)), the values of R_i become zero for i > 0 and both Eqs. (4.5) and (5.2) then become

$$F(x) = I_x(m+a, v) \tag{5.11}$$

Thus, the corollary holds for two samples with equal numbers of failures.

Now, we have a test statistic (Theorem 3.1) and two methods of calculating the cumulative probabilities for its value (Theorems 4.1 and 5.1). Although these equations are more complex that I would like to try on a slide rule, it would be straight forward to apply these theorems to any number of samples containing failure data, compute the value of the statistic, $\lambda^{P/R}$, and determine the probability that the statistic was less than or equal to that value when the samples came from populations with the same failure distributions.

In the next chapters, we will discuss the practical application of these results to the types of problems described in Chapter 1 and the development of supporting computer software to provide solutions for the most general cases.

Chapter VI

Application of Results

In Chapter 1, several examples were given, of problems which could be solved by analysis of failure data. In this chapter a method for solving these problems is discussed. An approach for the analysis is defined in the first section. Then, the application of the theorems from Chapters 3, 4, and 5 are discussed. The data and assumptions are examined and, finally, a sample problem is presented to demonstrate the application of the mathematical results.

Approach to Solution

All the example problems from Chapter 1 may be solved in the same way. The approach is, simply, to determine if samples taken from each group could have come from the same distribution by examining the expected MTBF of each sample.

In Chapter 2 it was shown that it is reasonable to assume the underlying distribution of failure times is exponential. This exponential distribution can be defined by a single parameter, the MTBF (mean-time-between-failures).

Relationships were also presented among the characteristics of the exponential distribution, showing the reliability and hazard functions are determined by the MTBF. Then, all of the example problems can be reduced to a comparison of MTBF's for samples of failure times from each situation under study. The difference in MTBF's can be translated to a difference in failure rate, reliability, or maintainability, by application of the appropriate formula from Chapter 2.

The test criteria defined by Theorem 3.1 is only capable of indicating whether a significant difference exists among the samples tested. It cannot indicate the amount of difference or the particular sample that varies from the others. It is important to understand this limitation of the analysis.

Application of this work to Air Force systems will provide a new tool for managers to analyze failure data of complex systems. Procedures are well documented for testing and verifying reliability data for Air Force systems (Ref. MIL-STD-781C). In other words, a sample of failure data may be used to esitmate the MTBF for the system, or verify the projected MTBF given by the system designers. However, there is no existing procedure to directly compare several samples of failure data. The theorems presented in this work can be used to compare samples directly and indicate the significance of the difference in the MTBF's of any number of samples.

Application of Theorems

There are three theorems that may be applied in this analysis. Theorem 3.1 defines the test criteria and must be applied to determine the value of L(X) for the samples being considered. Computation of the criteria requires three pieces of information for each sample; the number of items under test (n), the number of items that failed (r), and the failure times of those items (t[k], k = 1...r). In the most general case, these values will vary for each sample.

The other two necessary theorems may be applied through tables or computed directly. Theorem 4.1 and Theorem 5.1 define the cumulative distribution function of the test criteria in exact and asymtotic forms, respectively. The cumulative distribution function gives the probability that the test criteria will be less than or equal to the given value. These theorems may be used to generate tables for the simplest cases (as provided in Appendicies B and C), where the samples all contain the same number of failures.

The theorems may also be applied to specific cases where tables are not usually available. The test criteria is applicable to problems where several samples may contain different number of failures from test populations of different sizes. Percentage point values for such specific cases cannot generally be found in tables.

In any case, a level of significance must be selected to determine the decision point for the test criteria. As stated in Chapter 3, the hypothesis that the samples came from the same distribuiton will be rejected when $L(X) < C$, where C is chosen so the probability that $L(X) < C$, when the hypothesis is true, is equal to the level of significance. Thus, if the level of significance is small, the chance of mistakenly saying the distributions are not equal is also small.

Assumptions and Data

There are both assumptions and data necessary for making these calculations. It is always important to be certain that the assumptions are valid and the data is appropriate before the results of the analysis can be considered useful.

The only assumption made during the development of the theorems was that the failure times are distributed exponentially. This is a reasonable assumption based on works referenced in Chapter 2.

Data used in the calculations is of two types. First, there is the collected failure times of the items under study. The other data is the subjective data from which a decision point is established for the calculations.

The theorems were developed for 'type II censored samples' as described at the end of Chapter 2. This means that the 'sample' data necessary includes only the number of items under test (n), the number of items that failed before the test was stopped (r), and the amount of time each 'failed' item had been operating when the failure occurred (t[k] for k = 1...r). This data is required for each population to be considered in the analysis. Figure 3.1 shows a picture of this information for a variable number of populations (p). Note that the size of the populations and their respective samples are not required to be equal. It is also not necessary for the test durations to be the same.

As discussed in the previous section, the level of significance needs to be defined to determine the value of the statistic at which the hypothesis should be rejected. This is a subjective determination which may have a direct effect on the reported result. In other words, if the statistic indicates that the hypothesis should be rejected (the samples did not come from the same distribution) and there is still a 0.1 probability that the samples came from the same distribution, should it be reported that they are the same or not? Management factors should determine the allowable error.

Some assistance in making this judgement can be obtained in knowing the likelihood of the test criteria value being less than or equal to the computed value when, in fact, the sample populations have the same failure distribution. This is obtained by direct calculation of the C.D.F. using Theorems 4.1 or 5.1 with the sample and failure count data for the particular situation.

When this technique is used with the normal calculation and comparison of the test statistic, additional information about the validity of the comparison is provided. For example, if the test criteria indicates the hypothesis (that the samples have the same MTBF) should be rejected and the C.D.F. value is large then the test should be considered less reliable than when the C.D.F. is smaller. Also, if the test criteria indicates that the samples have the same MTBF but the C.D.F. is small, the test may, again, be misleading. It is important to consider both values as providing important information for the analysis of the data.

An Example Problem

To demonstrate the steps necessary in applying this analysis, a sample problem is presented in the following paragraphs.

Suppose two manufactures are developing a new navigation

Able Company's NC74 Navigation Computer	Baker Company's B357 Navigation Computer
17 Test Units	17 Test Units
4 Failures during test	4 Failures during test
Failure Times (Hours)	Failure Times (Hours)
0.7	1.5
1.6	2.3
3.0	6.7
5.1	8.5

Figure 6.1. Failure Data for A-7D Navigation Computers

computer for use in the A-7D aircraft. Each manufacturer has provided a number of production units to be tested by the Air Force. The number of units under test and the failure data is provided in Figure 6.1. The problem is that each manufacturer claims their units are more reliable that their competitors. It must be decided if there is a significant difference in their reliabilities. Since there are only two samples taken from populations of equal sizes and each test is stopped after the same number of failures have occurred, then, the test ratio can be simplified to:

$$L(x) = \frac{4T_1T_2}{(T_1+T_2)^2} \tag{6.1}$$

where T_1 and T_2 are the sums of the test times defined by Eq. (3.8).

The level of significance is chosen to be 0.1 for this example. Calculation of the statistic for the data of Figure 6.1

using Eq. (6.1) gives L(X) = 0.934432. Table B.1 in the appendix shows that the decision point for these values is 0.698207, so the hypothesis must be accepted at this level of significance. In other words, the statistic indicates that the failure data may have come from populations with the same exponential distributions so, neither manufacturer's units are significantly better than the other.

If, as we mentioned earlier, the cumulative probability for the statistic is calculated. The likelihood that the statistic is less than or equal to 0.934432 when the samples come from populations with the same failure distribution is found to be 0.475, or 47.5%. This large value reinforces our decision that the samples may have come from populations with the same failure distribution. In fact, we could have compared the data using a level of significance as high as 0.475 and the statistic, still, would indicated that the hypothesis should be accepted.

This procedure for analysis of sample data could be implemented in a computer program that would only require the operator to provide the data, in stored files, and the level of significance for the analysis. Such a program would only require an implementation of the theorems presented and the steps described for this simple example. However, to provide computer support for the most general problem requires that we address some additional problems.

General Problem Considerations

In the most general application of this work, it would be useful to provide a computer system that supports all necessary uses of the theorems. Not only should should we provide a program which would compare the MTBF's of any number of samples, of any size, but, we should also provide a method for generating tables of percentage points, as given in Appendix B, for the exact calculations, or Appendix C, for the asymtotic form of the function.

In a program designed to compute either table values or a comparison of sample data, there are several things that must be established for the program, and known by the user, for the program to provide a useful solution. These qualifications include the method of caluclation (exact or asymtotic), the accuracy of the calculation, and the maximum number of samples that may be compared (if any).

The accuracy of the results will depend on the number of terms used to compute the probability using the Cumulative Distribution Function of the statistic. Examination of the forms for the mathematical solution of the C.D.F. indicates that, in both the exact and asymtotic functions, there is a need to establish a limit for the number of terms that will be included when computing the probability. It is generally not practical to compute an infinite number of values in a computer program. So, in all the computer programs provided, the programmer or system operator establish the number of terms to be used and, therefore, the accuracy of the calculations.

Obviously, it would be possible to provide a program that simply provides a solution to a given problem at a stated accuracy. However, a computer system that supports a general application of this analysis would be more useful because the operator could tailor the system to his current needs and then re-adjust the parameters to satisfy his future needs. Ideally, this would not require as much work or expertise as the original programming effort to design and implement the system. The next chapter discusses the implementation of a software system to provide this type of computer support.

Chapter VII

Programming Environments to Support Mathematical Tools

The development of a mathematical solution is generally not an end in itself. To be practical, the solution must be made into a usable algorithm for solving a problem. Obviously, the algorithm should be tested and verified or proven to be correct. Then the solution may be used in the study of other problems, or to provide answers to specific questions within the solution space of the original problem. All of these uses (testing, continued development, and generation of data) may be provided for if the algorithm is created within an appropriate environment which supports these operations.

Providing a single environment for all of the algorithm's users would increase the communication among the participants in the problem's solution. Let's classify the users of the algorithm into two groups. 'Managers' have the questions which must be solved by 'mathematicians' and the answer must be stated in a form the manager can comprehend.

The manager is the originator of the problem which the algorithm attempts to solve. He defines the problem and determines the boundaries of the problem space and the solution space. The manager is also the end-user of the algorithm which solves the problem. Thus, the problem is originally stated by the manager and the solution must be usable and understandable to him. If the problem is not solved in an environment known to the manager, it must be translated into that form before it is useful. This is a step we wish to avoid.

The mathematician is the technician and must create an algorithm for solving the problem. He must understand the manager's problem in the form it is presented or translate it to a domain where he can operate. His working environment must provide for the application of standard mathematical techinques with predictable results. He must be able to create algorithms that not only provide a solution for the manager, but also may be used as components of other problems, not yet defined.

With the current trends towards standardization within military computer systems, it is becomming more likely that the managers and mathematicians described here will have access to compatable computer systems. A common environment which supports the needs of both users could increase productivity by increasing the communication, and thus the understanding, between these two groups. Such an environment should be a living entity that provides, for the mathematician, a place for new tools to be created, modified and tested, and for the manager, a place to use those tools and provide feedback to their designer.

In general, the support provided by programming environments is limited to text editors, languages, and file manipulation routines that perform specific transformations (Ref. Barstow 1984:199-200, Taylor and Standish 1985:302). General purpose languages (e.g. Fortran, Pascal, C, etc.) contain only minimal support for mathematics. The basic functions of addition, multiplication, division, subtraction and, sometimes, exponentiation are available. More complex functions must be provided by libraries of subroutines. Some of these libraries are extensive (e.g. IMSL) and contain routines that can be applied in many cases. However, the most recent solutions to mathematical problems are often not available in such libraries. It may also be found that the language of choice may not have access to the libraries that are needed. Computer support for reliability and maintainability studies is hampered by all these problems.

Environments and program libraries (such as, S, IMSL, and BMDP) that provide statistical routines are, generally, limited to fundamental statistical functions (e.g. t-tests, analysis of variance and covariance, generation of histograms, etc.). These functions may be combined to perform more complex work, but, only BMDP provides a survival analysis function (Ref. Dixon et al. 1983:576, Francis 1981).

A system supporting statistical analysis should provide the features of a good operating system with enhancements to assist in the special needs of the users. These enhancements are the tools which will be developed and used in the environment. Other features should include user-freindliness, use of common or popular structures and organization, and the ability to grow and change to meet new requirements. These features have made the UNIX operating system one of the most popular in use today (Ref. Kernigan and Mashey 1981:25)

Environment Definition

We will now define the goals for a Statistical Analysis Environment (SAE). This environment is not intended to have all the desirable features of such a system. It is only provided to demonstrate how an existing operating system may be enhanced to provide useful tools for the design, development and use of statistical algorithms.

The SAE should be designed to run in an established operating system. An operating system that supports mathematical and text manipulations is a necessary foundation for a successful environment. The SAE should be an enhancement to the operating system; not a restriction. The user should be able to select from any of the tools offered by the operating system, as well as those provided in the SAE.

The tools provided in the SAE should include an interpreter or compiler of some type, which would allow the user to write programs to analyze data files. A stored program system is necessary to allow algorithms to be used repeatedly and modified for varying needs or improvements. Mathematical theorems, such as those developed in the previous chapters, may be applied in many different ways to assist in solving problems in a large domain. Providing a fixed programmed solution to a particular problem is not efficient use of the effort required to produce the mathematical theorem.

A syntatical editor should be incorporated in the system to assist the user in writing valid programs. Such a specialized editor is typical of interactive programming environments like Smalltalk, MENTOR, Interlisp and others (Ref. Winograd 1973:14-16).

Useful information should be available to the user at all times. Information about the proper use of available commands, current status of the system or editor, and information about errors committed in programming. 'Help' functions can provide much of this information from stored text files (e.g. 'man' in UNIX and 'help' in VMS).

The SAE system should be designed to be extensible in several ways. The environment should allow the use of any new programs that are provided for the resident operating system. This would allow new tools to be created independently to perform data handling, pre- and post-processing of data or results, generation of information displays, real-time processing of failure data or new methods of analysis.

Modifications should be allowed in the language to incorporate new mathematical methods or improve those already in use. The 'help' information data should be able to be modified to incorporate corrections, improvements, and user tailored information to personalize the system to the programmer's needs.

In fact, modifications should be allowed to all parts of the environment program. This would allow the user a choice when new tools or features are needed. The SAE should be a living entity that changes to meet the needs of its users.

An Example Environment

In the course of developing the software to apply the mathematical theorems of chapters 3, 4 and 5, a minimal SAE system has been created. To demonstrate the application of this environment the following paragraphs describe the use of the system to analyze failure data in two seperate cases.

File: d2	File: d3	File: d4
10 items tested	10 items tested	10 items tested
Failures at	Failures at	Failures at
305.19	75.54	24.52
71.68	17.17	5.06
29.73	6.68	1.56
41.71	9.68	2.56
38.11	8.78	2.56
59.45	14.11	4.04
96.36	23.34	7.11
32.30	7.32	1.77
32.28	7.32	1.77
24.62	5.41	1.14

Figure 7.1. IMSL Data From Populations With Different MTBF's

The first case is the comparison of failure data that has been generated from known exponential distributions, using an 'IMSL' library function. Data was generated for MTBF's of 2, 3, and 4 (see Figure 7.1) and is contained in files named 'd2', 'd3' and 'd4', respectively. We know what the results of the comparison should be; that the samples did not come from populations with the same failure distribution. However, we will perform the analysis to demonstrate how the system would indicate this result.

Before we compare this data, we must use the environment to create a program that will perform the analysis. There are three or four parameters that must be assigned in the definition of the program, as we mentioned at the end of Chapter 6. The method of calculation, exact or asymtotic, must be specified. The accuracy of the result must be specified in the terms to be used in the calculations. Then the maximum number of samples to be compared, needs to be stated. The last two parameters are necessary to define the amount of storage needed by the program, internally.

There is one more parameter that may be specified when the program is defined, the level of significance. This is the probability that the statistic will indicate that the samples do not come from populations with the same failure distribution, when in fact, they do. It may be convenient to have a program that always uses the same level of significance, such as, the case where similar comparisons will be done several times on different sets of data. The more general case will require that the level of significance be entered by the user when the program is run. The same data may, then, be tested at different levels of significance by the same program.

These parameters are defined in the 'SAE' programming language. The 'definition program' containing these commands is then compiled, which creates the 'test program' which will evaluate sample data in the way just specified. The program can be used as often as necessary to compare any sets of data to which the parameters of the program apply. The definition program may also be modified at any time and used to create another test program for evaluating sample data.

An example showing the use of the environment to analyze the data of Figure 7.1 is shown in Figure 7.2. In the case shown, the 'definition program' was previously created using a simple text editor. The SAE system contains an editor which assists the programmer by providing immediate compilation of the edited text

```
SAE > cat test.src
-- Test program
environment (maxterms = 7, method = asymtotic);
"test7" = lrt (samples = 10);

SAE > mv test.src test7.src

SAE > c test7.src

SAE Compiler, Version 0.2

Compiling test7.src:

1  -- Test program
2  environment (maxterms = 15, method = exact);
3  "test15e" = lrt (samples = 10);

Summary:  3 lines   0 error

SAE > test15e d1 d2 d3

Likelihood Ratio Test Program   Version 2.0

Exact Method for calculating test criteria
     Maximum number of terms used:  15
     10 Samples may be given
     Level of significance will be requested later

Summary:
Test criteria from data files = 0.203618
Enter desired level of significance: 0.1

Probability that LRT value is less than or equal to 0.203618
   when the sample populations have the same failure rate = 0.000

Reference point @ 0.100 Significance = 0.790246
Therefore:
... Samples CANNOT be assumed to have come from populations
    with the same failure distributions

SAE >
```

Figure 7.2. Example Solution Using the SAE System

by the system compiler. The use of the SAE editor would make it unnecessary to compile the 'definition program' after editing. More details are given in Users Manual (Appendix E).

In the example shown in Figure 7.2, the exact method was used with a maximum of 15 terms in the calculations. The maximum value of 10 samples is does not restrict the program to only be used with exactly 10 samples but, allows the same test program to analyze up to that number of sample files. The final parameter, level of significance, was not defined for the program, which means that the test program will prompt the user to provide a value each time it is run. The test program, 'test15e', may be used as often as necessary, where these parameters are adequate for the analysis.

When 'test15e' is executed, the parameters have been determined are displayed as a program header. In this case, the maximum number of terms, the maximum number of samples, and the method of calculation, are all provided for the user. The test criteria is calculated and, because it was not specified when the program was defined, the level of significance is requested from the user.

At this point, the 'p-value' or 'probability value' is computed for the test criteria value computed from the data given. This is the probability that the test criteria value is less than or equal to the given value when the populations do, in fact, have the same failure distribution. In other words, it is the probability that the normal evaluation of the test criteria would IMPROPERLY indicate that the samples were NOT from populations with the same failure rates. If this value is less than one tenth of a percent, then only zero will be given.

This 'p-value' is the switching point for the level of significance. If the level of significance requested is less than or equal to the 'p-value' computed, then, the test program will indicate that the samples 'cannot be assumed to have come from populations with the same failure distributions.' On the other hand, if the level of significance is higher than the 'p-value' the test will indicate that the samples 'may have come from populations with the same failure distribution.' The user has the freedom to select the value he desires and is given complete knowledge about how the selection of the level of significance will affect the results of the test.

When the level of significance was determined by the programmer and cannot be changed for the test program being used, the 'p-value' is, still, computed to show the relative value of the decision made by the program. Thus, the user is always given the information needed to properly interpret the program results.

In this way this environment has assisted the programmer to use the method of analysis in the most general applications. The programmer is restricted to the proper procedure for the analysis but not limited in the data he may analyze or the significance of the results. It is the user's responsibility to only apply the analysis where it is appropriate; to samples whose expected failure distribution is exponential, where the data fits the definition of a type II censored sample.

The operation of the SAE system is described in more detail in Appendix E. This system is not intended to be a complete environment, but only to demonstrate that an environment could be useful in the application of complex mathematical solutions for data analysis. Recommendations for expansion of this concept and further development of the theoretical work are presented in the next chapter.

Chapter VIII

Summary and Recommendations

Comparing the reliabilities of several complex systems can often be reduced to a comparison of their underlying failure distributions. Where these failure distributions are exponential, the comparison further reduces to examination of the mean-time-between-failures (MTBF) for the systems. In 1953, Epstein provided a procedure using a likelihood ratio test statistic for comparing the MTBF of two exponential distributions based on sample failure data. The technique was limited to comparing only two samples containing exactly the same number of failures.

This dissertation presents the derivation of a likelihood ratio test criteria that is applicable to multiple samples, containing almost any number of failures. The significant contribution is contained in the derivation of the h-th moment of the statistic (Theorem 3.2) and the subsequent derivation of the exact (Theorem 4.1) and asymtotic (Theorem 5.1) forms of the cumulative distribution function.

In addition, the application if this work is discussed extensively and a computer software environment has been created which allows a programmer to easily create programs which implement the test criteria in a practical manner. There are a number of directions that might be taken to extend the work done here

Statistical Package Implementation

An obvious suggestion concerning the implementation of the test criteria would be to extend the environment to include an existing library of programs (e.g. IMSL, BMDP, etc.) and allow the user to incorporate these functions into programs similar to 'table' and 'lrt'. It might be of more practical value, however, to implement these test criteria in a form that could be included in an existing library or data base environment. As previously mentioned in the text, survival analysis functions are generally lacking in the statistical packages available now.

Automatic Programming Environments

The SAE provides a small level of automatic programming to the user. The environment allows the user to define some important characteristics of the program in a simple language. This system limits the user's access to the details of the analysis technique. However, it assists the user in applying the procedure correctly by reducing his/her elements of concern to those that directly affect the results of the test: the accuracy of the result, the number of samples that may be compared, the level of significance for the comparison, and the method of calculation (the choice of exact or asymtotic C.D.F.).

Once these parameters are identified, the SAE system compiler uses the UNIX 'make' facility to construct the program as requested. The new program may then be used repeatedly to analyze different sets of data in exactly the same way. If the program is destroyed or no longer of use, a new copy may be created easily with the same or different characteristics.

Application of new techniques from artificial intelligence work may help to develop future systems with expanded capabilities in this area. Automatic programming techniques are currently being used in other areas to increase the speed and accuracy with which programs are created. The complexity of the task involved in programming will demand that the types of programs created becomes more specialized as the power of the automated techniques increase.

Improve the Processing Speed

A problem with the implementation of these formulas is that they are dependent on the creation of several sets of series coefficients. This is a time consuming task. More than half of the execution time (68% average, based on 60 computations of 2 to 5 samples with from 3 to 100 failures per sample) is spent on the creation of the coefficient values. Of course, the number of terms requested for accuracy of the computations will affect this timing. But, for lower number of failures in each sample, more terms are necessary for accuracy of the result. Consideration should be given to finding a more efficient implementation of the formulas than has been done here.

Parallel processing systems offer a definite possibility to improving the speed of the calculations. The two dimensional coefficients 'C' could be computed independently from the 'q' and 'A' coefficients in the current software implementation (these coefficients are named 'Cjr', 'Q', and 'Ar', respectively, in the software listings given in Appendix F). Concurrent processing of these two distinct paths in the computation of the coefficients would significantly reduce the processing time. Implementation of these computations as concurrent 'tasks' on a multi-tasking single-processor system might also improve the speed of the computations, however, this would be more dependent upon the operating system's task control parameters.

Parallel processing may also be applied to the system level of the SAE by making the editing and interpreting concurrent tasks. The SAE system editor currently calls the compiler to evaluate a part of the text each time a modification is made to any line in the editor. A compiler running concurrently with the editor might improve characteristics of that tool. Usually, the SAE programs are very small (3 or 4 lines) and, so, there is not a lot of time to be gained in this area.

Application to Other Failure Distributions

While all of the suggestions just made relate to the computer programming aspects of this effort, that does not diminish the usefulness of the mathematical work. The techniques used to derive the h-th moment of the test criteria have been available to mathematicians for a long time (Ref. Stekloff 1914, Wilks 1967). Their application is this work should create some interest to find other cases where they may be applied.

In particular, since this work considered the single parameter exponential distribution, the first step would be to apply a similar approach to the case of the two parameter exponential failure distribution. Dr. B. N. Nagarsenker, my advisor in this work, is currently exploring this problem. If he is successful, further work should include the examination of distributions that are closely related to the exponential.

Bibliography

Aho, Alfred V. and Jeffrey D. Ullman. The Theory of Parsing, Translation, and Compiling, Volume II: Compiling. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1973.

Barr, Donald R. and Peter W. Zehna. Probability: Modeling Uncertainty. Reading, Massachusetts: Addison-Wesley Publishing Co., 1983.

Barstow, David R. "A Display-Oriented Editor for INTERLISP," Interactive Programming Environments, edited by David R. Barstow, Howard E. Shrobe, and Erik Sandewall, New York: McGraw-Hill Book Co., 1984.

Barrett, William A. and John D. Couch. Compiler Construction: Theory and Practice. Chicago, Illinois: Science Research Associates, Inc., 1979.

Blake, Ian F. An Introduction to Applied Probability. New York: John Wiley & Sons, Inc., 1979.

Buxton, John N., and Larry E. Druffel. "Rationale for STONEMAN," Fourth International Computer Software and Applications Conference, 66-72, Chicago, Illinois: IEEE, 1980.

Department of the Air Force. Reliability and Maintainability Program for Systems, Subsystems, Equipment, and Munitions, AFR 80-5/AFSC/ASD Sup 1, Washington: HQ USAF,.

Department of Defense. Requirements for Reliability Program (for Subsystems and Equipment), MIL-STD-785B, Washington: Government Printing Office, 15 September 1980.

-----. Reliability Tests Exponential Distribution, MIL-STD-781C, Washington: Government Printing Office, 21 October 1977.

Dixon, W. J. et. al. ed. BMDP Statistical Software, Berkeley, California: University of California Press, 1983.

Drenick, R. F. "The Failure Law of Complex Equipment," Society for Industrial and Applied Mathematics, Vol 8, 680-690, 1960.

Epstein, B. "Statistical Life Test Acceptance Procedures," Technometrics, (2) 435-446, 1960.

-----. "Truncated Life Tests in the Exponential Case," Ann. Math. Stat., (25) 555-564, 1954.

Epstein, B. and M. Sobel. "Sequential Life Tests in the Exponential Case," Ann. Math. Stat., (26) 82-93, 1955.

-----. "Some Theorems Relevant to Life Testing from an Exponential Distribution," Ann. Math. Stat., (25) 373-381, 1954.

-----. "Life Testing," J. Am. Stat. Assoc., (48) 486-502, 1953.

Francis, Ivor. Statistical Software: A Comparative Review, New York, N.Y.: Elsevier North Holland, Inc., 1981.

Govindarajulu, Z. "A Suplement to Mendenhall's Bibliography on Life Testing and Related Topics," J. Am. Stat. Assoc., (59) 1231-1291, 1964.

Hogg, Robert V. and Allen T. Craig. Introduction to Mathematical Statistics (Fourth Edition). New York: Macmillan Publishing Co., Inc., 1978.

Johnson, N. L. and S. Kotz. Continuous Univariate Distributions, Vols. 1 and 2. Boston, Massachusetts: Houghton Mifflin, 1970.

Kapur, K. C. and L. R. Lamberson. Reliability in Engineering Design. New York: John Wiley & Sons, Inc., 1977.

Kernigan, Brian W., and John R. Mashey. "The UNIX Programming Environment," Computer, 14:4 25-34 (April 1981).

Lawless, J. F. Statistical Models and Methods for Lifetime Data. New York: John Wiley & Sons, Inc., 1982.

Lawton, David J. Asymtotic Expansions of the Distribution of Test Statistics Associated with Several Two Parameter Exponential Distributions. MS Thesis, MATH 84D-1, Engineering School, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1984.

Mendenhall, W. "A Bibliography on Life Testing and Related Topics," Biometrika, (45) 521-543, 1958.

Nagarsenker, B. N. "On a Test of Equality of Several Exponential Survival Distributions," Biometrika, 67 (2): 475-478, Great Britain (January 1980).

Nagarsenker, B. N. and P. B. Nagarsenker. "Distribution of LRT for Testing the Equality of Several 2-Parameter Exponential Distributions," IEEE Transactions on Reliability, 34 (1): 65-68 (April 1985).

-----. "On a Test of Equality of Two-Parameter Exponential Distributions," Statistics & Probability Letters, 2: 357-361, North Holland: Elsevier Science Publishers B. V. (December 1984).

Nagarsenker, B. N. and J. Suniaga. "Distributions of a Class of Statistics Useful in Multivariate Analysis," Journal of the American Statistical Association, 78 (382):472-475 (June 1983).

Stekloff, W. "Quelques Applications Nouvelles de la Theorie de Fermeture au Probleme de Representation Approchee des Functions et au Probleme des Moments," Memories de l'Academie Imperiale des Sciences de St. Petersbourg, Vol XXXII, No. 4, 1914.

Subrahmaniam, K. "On the Asymtotic Distributions of Some Statistics Used for Testing Equality of Sums," The Annals of Statistics, 3 (4): 916-925 (July 1975).

Sukhatme, P.V. "Tests of Significance for Samples of the X2 Population with Insulation Aging Tests," Ann. Eugen., (8) 52-56, 1937.

Trivedi, Kishor S. Probability and Statistics with Reliability, Queuing, and Computer Science Applications. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1982.

Wilks, Samuel S. "On the Distributions of Statistics in Samples from a Normal Population of Two Variables with Matched Sampling of One Variable," S.S. Wilks: Collected Papers; Contributions to Mathematical Statistics, T. W. Anderson, ed., New York: John Wiley & Sons, Inc., 1967.

Winograd, Terry. "Beyond Programming Languages," Communications of the ACM, 22:7 391-401 (July 1979).

-----. "Breaking the Complexity Barrier (Again)," Proceedings of the ACM SIGPLAN-SIGIR Interface Meeting on Programming Languages - Information Retrieval, Gaithersburg, Maryland: ACM, 1973.

Appendix A

Calculation of Adjustment Factors

The purpose of this appendix is to show the calculation of the convergence factors defined in Chapters 4 and 5. These factors are delta (δ), which is defined by

$$m\rho = R - \delta \qquad \text{(A.1)}$$

when making a substitution in Eq. (4.8) for R and α, which is a constant necessary to apply Lemma 4.3 to Eq. (4.12). The values of these factors are calculated in the following sections.

Calculation of Delta

Delta is an adjustment factor defined, in Chapter 4, to set

$$A_1 = \emptyset \qquad \text{(A.2)}$$

where

$$A_r = \frac{(-1)^r}{r(r+1)p^r}\left[B_{r+1}(\delta) - \sum_{i=1}^{p}\frac{B_{r+1}(\delta k_i)}{k_i^{\,r}}\right] \qquad \text{(A.3)}$$

from Eq. (4.12) where p is the number of samples and k_i is the percentage of failures contained in the i-th sample (Eq. (4.5). B_r is the Bernoulli Polynomial of degree r and order unity. The second polynomial is

$$B_2(h) = h^2 - h + 1/6 \qquad \text{(A.4)}$$

Now, A_1 can be written

$$A_1 = \frac{-1}{2p}\left[B_2(\delta) - \sum_{i=1}^{p} \frac{B_2(\delta k_i)}{k_i}\right] \tag{A.5}$$

Setting the left-hand side to zero and expanding the Bernoulli Polynomials to get

$$\emptyset = \frac{-1}{2p}\left[(\delta^2 - \delta + 1/6) - \sum_{i=1}^{p} \frac{(\delta^2 k_i^2 - \delta k_i + 1/6)}{k_i}\right] \tag{A.6}$$

Computing the summation gives

$$\emptyset = \frac{-1}{2p}\left[(\delta^2 - \delta + 1/6) - \delta^2 + p\delta - \frac{1}{6(\sum_{i=1}^{p} k_i)}\right] \tag{A.7}$$

and, then combining terms

$$\emptyset = \frac{-1}{2p}\left\{\delta(p-1) - 1/6\left[\left(\sum_{i=1}^{p} k_i^{-1}\right) - 1\right]\right\} \tag{A.8}$$

Since, p must have an integer value greater than 2, delta is defined by

$$\delta = \left[\left(\sum_{i=1}^{p} k_i^{-1}\right) - 1\right] \Big/ 6(p-1) \tag{A.9}$$

as stated in Chapter 4.

In Chapter 5, the value of delta depends on T_r . These coefficients have a relationship to the A_r coefficients of Chapter 4. In fact,

$$T_r = -A_r \tag{A.10}$$

as stated in Eq. (5.4). It is obvious that the same value of delta, as derived above, will satisfy the requirement that

$$T_1 = \emptyset \tag{A.11}$$

as stated in Chapter 5. This value of delta will now be used to calculate the required value for a, the other convergence factor used in Chapters 4 and 5.

Calculation of a

The value of a is chosen such that the coefficient R_1 is zero. The R_i coefficients are defined in Eq. (4.14) by

$$\sum_{j=\emptyset}^{k} R_{k-j} D_{k-j,j} = q_k \tag{A.12}$$

where $R_\emptyset = 1$ and

$$D_{i,r} = \sum_{k=1}^{r} k C_{i,k} D_{i,r-k} \tag{A.13}$$

where $D_{i,\emptyset} = 1$ and

$$C_{i,r} = \frac{(-1)^{r-1}}{r(r+1)} \left[B_{r+1}(a) - B_{r+1}(a+v+i) \right] \tag{A.14}$$

Also, from Eq. (4.12)

$$q_r = \frac{1}{r} \sum_{k=1}^{r} k A_k q_{r-k} \tag{A.15}$$

and

$$v = \frac{(p-1)}{2} \quad . \tag{A.16}$$

The value of R_1 can be found from

$$R_1 D_{1,\emptyset} + R_\emptyset D_{\emptyset,1} = q_1 = A_1 \tag{A.17}$$

where $A_1 = \emptyset$, from the discussion of delta in this appendix. So, if $R_1 = \emptyset$ is desired, then

$$\emptyset = D_{\emptyset,1} = C_{\emptyset,1} D_{1,\emptyset} \tag{A.18}$$

which means

$$\emptyset = C_{\emptyset,1} = \frac{1}{2}\left[B_2(a) - B_2(a+v)\right] \tag{A.19}$$

Expanding the Bernoulli Polynomials gives

$$\emptyset = \frac{1}{2}\left[a^2 - a + \frac{1}{6} - (a+v)^2 + (a+v) - \frac{1}{6}\right] \tag{A.20}$$

which reduces to

$$\emptyset = v - 2av - v^2 \tag{A.21}$$

or $a = \dfrac{(1-v)}{2}$ as stated in Chapter 4.

Thus, the values of the convergence factors δ and a have been calculated to meet the requirements specified in Chapters 4 and 5.

Appendix B

Tables of Exact Computation of Statistic

This appendix contains percentage point tables generated from application of the exact form of the cumulative distribution function (C.D.F.) of the statistic. The tables were generated using a *maximum of* 20 terms. Similar tables, containing the same information can be generated by defining a 'table' program for the SAE environment with the following commands:

```
environment (maxterms = 20, method = exact);
"table" = table (
     samples = (2,6,1),
     failures = (3,100,5),
     alpha = (0.1, 0.05, 0.025, 0.01)
     );
```

A more detailed explaination can be found in Appendix E.

The tables provide percentage points of the statistic L which is function of the statistic, lambda, given in Theorem 3.1. The necessary equivalences are:

$L = \lambda^{p/R}$

p = number of samples

r = number of failures per sample

λ = likelihood ratio defined by Theorem 3.1

R = total number of failed items (= pr)

The first five tables are for situations where the samples have an equal number of failures. The last two tables are provided to demonstrate the use of these theorems for cases where the number of failures is not the same for each sample.

Corresponding tables generated using the asymtotic expansion of the exact C.D.F. are provided in Appendix C.

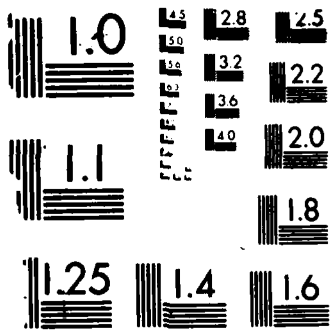
Table B-1

Percentage Points of $L = \lambda^{P/R}$
for
Two Samples with Equal Number of Failures

Failures	Level of Significance				
r	0.1	0.05	0.025	0.01	0.005
3	0.613751	0.500527	0.405047	0.303874	0.243556
4	0.698207	0.600707	0.513780	0.415398	0.352607
5	0.752728	0.668244	0.590433	0.498895	0.438088
6	0.790694	0.716537	0.646769	0.562583	0.505173
7	0.818608	0.752684	0.689725	0.612381	0.558654
8	0.839978	0.780716	0.723487	0.652237	0.602059
9	0.856858	0.803074	0.750687	0.684790	0.637889
10	0.870525	0.821313	0.773052	0.711846	0.667915
11	0.881814	0.836472	0.791757	0.734671	0.693414
12	0.891295	0.849267	0.807629	0.754175	0.715323
13	0.899370	0.860209	0.821261	0.771027	0.734340
14	0.906329	0.869674	0.833096	0.785731	0.750997
15	0.912390	0.877941	0.843466	0.798669	0.765703
16	0.917714	0.885223	0.852626	0.810141	0.778780
17	0.922429	0.891686	0.860775	0.820382	0.790483
18	0.926633	0.897461	0.868072	0.829577	0.801015
19	0.930405	0.902652	0.874644	0.837880	0.810543
20	0.933808	0.907343	0.880593	0.845414	0.819204
25	0.946814	0.925338	0.903500	0.874576	0.852867
30	0.955550	0.937483	0.919039	0.894493	0.875981
35	0.961821	0.946230	0.930269	0.908956	0.892827
40	0.966542	0.952831	0.938765	0.919935	0.905647
45	0.970224	0.957988	0.945416	0.928552	[illegible]
50	0.973176	0.962129	0.950764	0.935495	0.923868
55	0.975595	0.965527	0.955158	0.941208	[illegible]
60	0.977614	0.968366	0.958832	0.945993	[illegible]
65	0.979325	0.970772	0.961949	0.950057	[illegible]
70	0.980793	0.972839	0.964628	0.953552	[illegible]
75	0.982066	0.974632	0.966954	0.956591	[illegible]
80	0.983181	0.976204	0.968994	0.959256	[illegible]
85	0.984165	0.977592	0.970796	[illegible]	[illegible]
90	0.985040	0.978827	0.972400	[illegible]	[illegible]
95	0.985824	0.979933	0.973837	[illegible]	[illegible]
100	0.986530	0.980929	[illegible]	[illegible]	[illegible]

AD-A182 686 RELIABILITY ANALYSIS BASED ON EXPONENTIAL FAILURE DISTRIBUTIONS AND DEVEL (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI K N COLE 2/3

UNCLASSIFIED 19 DEC 86 AFIT/DS/ENG/86-06 F/G 12/3 NL



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

Table B-2

Percentage Points of $L = \lambda^{P/R}$
for
Three Samples with Equal Number of Failures

Failures r	Level of Significance 0.1	0.05	0.025	0.01	0.005
4	0.544845	0.454030	0.378431	0.297548	0.248114
5	0.618279	0.535089	0.463141	0.382721	0.331341
6	0.671736	0.595989	0.528815	0.451535	0.400699
7	0.712246	0.643136	0.580754	0.507508	0.458322
8	0.743947	0.680606	0.622674	0.553611	0.506519
9	0.769405	0.711050	0.657134	0.592098	0.547224
10	0.790286	0.736252	0.685921	0.624640	0.581956
11	0.807717	0.757443	0.710306	0.652475	0.611883
12	0.822483	0.775505	0.731215	0.676533	0.637906
13	0.835150	0.791077	0.749334	0.697520	0.660723
14	0.846135	0.804639	0.765182	0.715981	0.680878
15	0.855751	0.816555	0.779157	0.732340	0.698805
16	0.864238	0.827105	0.791571	0.746932	0.714847
17	0.871784	0.836512	0.802670	0.760028	0.729284
18	0.878536	0.844951	0.812651	0.771843	0.742342
19	0.884614	0.852563	0.821675	0.782557	0.754208
20	0.890113	0.859465	0.829873	0.792314	0.765038
25	0.911265	0.886130	0.861689	0.830411	0.807509
30	0.925593	0.904298	0.883494	0.856726	0.837017
35	0.935938	0.917471	0.899367	0.875983	0.858699
40	0.943759	0.927457	0.911437	0.890683	0.875299
45	0.949879	0.935289	0.920923	0.902271	0.888412
50	0.954797	0.941594	0.928574	0.911639	0.899033
55	0.958837	0.946781	0.934876	0.919369	0.907809
60	0.962214	0.951121	0.940157	0.925856	0.915183
65	0.965079	0.954807	0.944645	0.931378	0.921465
70	0.967540	0.957977	0.948508	0.936134	0.926881
75	0.969677	0.960731	0.951866	0.940274	0.931599
80	0.971550	0.963146	0.954814	0.943910	0.935745
85	0.973205	0.965281	0.957421	0.947129	0.939417
90	0.974679	0.967183	0.959744	0.949999	0.942692
95	0.975998	0.968887	0.961827	0.952573	0.945632
100	0.977187	0.970422	0.963704	0.954895	0.948285

Table B-3

Percentage Points of $L = \lambda^{P/R}$
for
Four Samples with Equal Number of Failures

Failures r	Level of Significance 0.1	0.05	0.025	0.01	0.005
5	0.521563	0.443317	0.378008	0.307225	0.263112
6	0.583391	0.509907	0.446843	0.376314	0.330959
7	0.631452	0.562920	0.502938	0.434348	0.389247
8	0.669762	0.605914	0.549209	0.483282	0.439209
9	0.700961	0.641390	0.587881	0.524868	0.482201
10	0.726835	0.671114	0.620610	0.560526	0.519426
11	0.748625	0.696353	0.648627	0.591373	0.551883
12	0.767219	0.718038	0.672859	0.618284	0.580382
13	0.783268	0.736860	0.694009	0.641944	0.605574
14	0.797257	0.753347	0.712623	0.662894	0.627984
15	0.809558	0.767903	0.729124	0.681565	0.648035
16	0.820457	0.780847	0.743849	0.698304	0.666072
17	0.830180	0.792431	0.757068	0.713390	0.682378
18	0.838907	0.802857	0.768998	0.727055	0.697187
19	0.846783	0.812290	0.779819	0.739487	0.710692
20	0.853927	0.820865	0.789676	0.750846	0.723057
25	0.881555	0.854197	0.828184	0.795506	0.771910
30	0.900403	0.877088	0.854802	0.826638	0.806181
35	0.914081	0.893774	0.874292	0.849565	0.831529
40	0.924457	0.906475	0.889174	0.867147	0.851029
45	0.932598	0.916465	0.900909	0.881054	0.866491
50	0.939156	0.924528	0.910399	0.892329	0.879049
55	0.944551	0.931172	0.918231	0.901654	0.889451
60	0.949067	0.936741	0.924804	0.909493	0.898207
65	0.952903	0.941477	0.930400	0.916176	0.905679
70	0.956202	0.945553	0.935221	0.921940	0.912130
75	0.959070	0.949098	0.939417	0.926963	0.917755
80	0.961584	0.952211	0.943103	0.931378	0.922704
85	0.963808	0.954964	0.946367	0.935291	0.927091
90	0.965788	0.957418	0.949276	0.938781	0.931008
95	0.967563	0.959618	0.951886	0.941915	0.934525
100	0.969163	0.961602	0.954241	0.944743	0.937701

Table B-4

Percentage Points of L = $\lambda^{P/R}$
for
Five Samples with Equal Number of Failures

Failures r	Level of Significance 0.1	0.05	0.025	0.01	0.005
6	0.511852	0.441912	0.383279	0.319062	0.278468
7	0.564710	0.498157	0.441165	0.377245	0.335876
8	0.607562	0.544621	0.489861	0.427346	0.386169
9	0.642911	0.583499	0.531173	0.470611	0.430174
10	0.672523	0.616434	0.566548	0.508178	0.468781
11	0.697665	0.644648	0.597120	0.541009	0.502802
12	0.719264	0.669066	0.623768	0.569892	0.532936
13	0.738010	0.690391	0.647180	0.595464	0.559770
14	0.754428	0.709167	0.667898	0.618242	0.583789
15	0.768924	0.725819	0.686353	0.638647	0.605395
16	0.781813	0.740683	0.702891	0.657021	0.624922
17	0.793348	0.754032	0.717791	0.673647	0.642647
18	0.803730	0.766082	0.731282	0.688759	0.658803
19	0.813123	0.777015	0.743553	0.702550	0.673585
20	0.821661	0.786976	0.754760	0.715184	0.687157
25	0.854848	0.825911	0.798802	0.765178	0.741138
30	0.877638	0.852842	0.829480	0.800317	0.779336
35	0.894251	0.872570	0.852061	0.826342	0.807755
40	0.906895	0.887640	0.869370	0.846381	0.829711
45	0.916840	0.899526	0.883059	0.862282	0.847177
50	0.924867	0.909140	0.894154	0.875205	0.861401
55	0.931481	0.917076	0.903327	0.885914	0.873207
60	0.937026	0.923737	0.911039	0.894932	0.883162
65	0.941740	0.929409	0.917612	0.902631	0.891669
70	0.945798	0.934295	0.923281	0.909278	0.899022
75	0.949328	0.938549	0.928220	0.915077	0.905442
80	0.952426	0.942286	0.932562	0.920180	0.911095
85	0.955167	0.945595	0.936409	0.924704	0.916110
90	0.957610	0.948544	0.939841	0.928743	0.920590
95	0.959800	0.951191	0.942922	0.932371	0.924616
100	0.961775	0.953579	0.945702	0.935648	0.928254

Table B-5

Percentage Points of $L = \lambda^{P/R}$
for
Six Samples with Equal Number of Failures

Failures r	Level of Significance				
	0.1	0.05	0.025	0.01	0.005
7	0.507662	0.443757	0.389965	0.330565	0.292616
8	0.553651	0.492349	0.439869	0.380822	0.342395
9	0.592052	0.533547	0.482802	0.424873	0.386631
10	0.624529	0.568809	0.519973	0.463575	0.425918
11	0.652316	0.599270	0.552382	0.497720	0.460882
12	0.676338	0.625814	0.580839	0.527994	0.492105
13	0.697299	0.649129	0.605993	0.554974	0.520098
14	0.715740	0.669758	0.628370	0.579140	0.545301
15	0.732085	0.688129	0.648392	0.600892	0.568085
16	0.746668	0.704590	0.666403	0.620561	0.588766
17	0.759756	0.719418	0.682686	0.638424	0.607611
18	0.771568	0.732843	0.697475	0.654713	0.624846
19	0.782279	0.745053	0.710962	0.669621	0.640661
20	0.792035	0.756203	0.723310	0.683313	0.655221
25	0.830139	0.800013	0.772102	0.737816	0.713493
30	0.856470	0.830521	0.806334	0.776421	0.755058
35	0.875745	0.852973	0.831655	0.805162	0.786153
40	0.890462	0.870182	0.851135	0.827379	0.810273
45	0.902065	0.883790	0.866582	0.845060	0.829520
50	0.911446	0.894818	0.879129	0.859463	0.845231
55	0.919189	0.903936	0.889522	0.871420	0.858297
60	0.925687	0.911601	0.898271	0.881504	0.869331
65	0.931219	0.918134	0.905737	0.890124	0.878774
70	0.935984	0.923768	0.912183	0.897576	0.886946
75	0.940132	0.928677	0.917804	0.904082	0.894086
80	0.943775	0.932992	0.922749	0.909811	0.900380
85	0.947000	0.936815	0.927133	0.914895	0.905967
90	0.949876	0.940226	0.931047	0.919437	0.910962
95	0.952456	0.943287	0.934562	0.923519	0.915453
100	0.954783	0.946050	0.937736	0.927208	0.919513

Table B-6

Percentage Points of L = $\lambda^{P/R}$
for
Two Samples with Unequal Number of Failures
Level of Significance is 0.5

Percentage of Failures per Sample		Total Number of Failures = R			
		10	20	60	100
0.50	0.50	0.66824365	0.82131333	0.93748256	0.96212923
0.45	0.55	0.66807749	0.82126096	0.93747601	0.96213015
0.40	0.60	0.66755956	0.82109744	0.93745503	0.96212238
0.35	0.65	0.66662578	0.82080144	0.93741700	0.96210830
0.30	0.70	0.66514658	0.82032925	0.93735617	0.96208578
0.25	0.75	0.66287789	0.81959629	0.93726135	0.96205065
0.20	0.80	0.65934591	0.81843068	0.93710946	0.96199434
0.15	0.85	0.65355281	0.81643861	0.93684615	0.96189658
0.10	0.90	0.64306243	0.81251018	0.93630917	0.96169657
0.05	0.95	0.61368872	0.80204031	0.93471112	0.96109370

Table B-7

Percentage Points of $L = \lambda^{P/R}$
for
Three Samples with Unequal Number of Failures
Level of Significance is 0.5

Percentage of Failures per Sample			Total Number of Failures = R 10	20	60	100
0.33	0.33	0.34	0.38374316	0.62865090	0.85946431	0.91350162
0.30	0.30	0.41	0.38321532	0.62842843	0.85942966	0.91348832
0.26	0.26	0.47	0.38187782	0.62786298	0.85934144	0.91345444
0.23	0.23	0.54	0.37969493	0.62693256	0.85919570	0.91339844
0.20	0.20	0.60	0.37647907	0.62554158	0.85897628	0.91331407
0.17	0.17	0.67	0.37184652	0.62348715	0.85864841	0.91318784
0.13	0.13	0.74	0.36509887	0.62035853	0.85813902	0.91299131
0.10	0.10	0.80	0.35502593	0.61525808	0.85727662	0.91265730
0.07	0.07	0.87	0.33940541	0.60589024	0.85555703	0.91198531

Appendix C

Tables of Asymtotic Expansion of Statistic

This appendix contains percentage point tables generated from application of the asymtotic expansion of the cumulative distribution function (C.D.F.) of the statistic. These tables were generated using a maximum of 7 terms. Similar tables containing this data can be generated by defining a 'table' program for the SAE environment with the following commands:

```
environment (maxterms = 7, method = asymtotic);
"table" = table (
     samples = (2,6,1),
     failures = (3,100,5),
     alpha = (0.1, 0.05, 0.025, 0.01)
     );
```

A more detailed explaination can be found in Appendix E.

The tables provide percentage points of the statistic L which is function of the statistic, lambda, given in Theorem 3.1. The necessary equivalences are:

$L = \lambda^{p/R}$

p = number of samples

r = number of failures per sample

λ = likelihood ratio defined by Theorem 3.1

R = total number of failed items (= pr)

The five tables are for situations where the samples have an equal number of failures. Corresponding tables generated by the exact C.D.F. are provided in Appendix B.

Table C-1

Percentage Points of $L = \lambda^{p/R}$
for
Two Samples with Equal Number of Failures

Failures r	Level of Significance 0.1	0.05	0.025	0.01	0.005
4	0.698207	0.600706	0.513779	0.415398	0.352607
5	0.752728	0.668244	0.590433	0.498895	0.438088
6	0.790694	0.716537	0.646769	0.562583	0.505173
7	0.818608	0.752684	0.689725	0.612381	0.558654
8	0.839978	0.780716	0.723487	0.652237	0.602059
9	0.856858	0.803074	0.750687	0.684790	0.637889
10	0.870525	0.821313	0.773052	0.711846	0.667915
11	0.881814	0.836472	0.791757	0.734671	0.693414
12	0.891295	0.849267	0.807629	0.754175	0.715323
13	0.899370	0.860209	0.821261	0.771027	0.734340
14	0.906329	0.869674	0.833096	0.785731	0.750997
15	0.912389	0.877941	0.843466	0.798669	0.765703
16	0.917714	0.885223	0.852626	0.810141	0.778780
17	0.922429	0.891686	0.860775	0.820381	0.790483
18	0.926633	0.897461	0.868072	0.829577	0.801015
19	0.930405	0.902652	0.874644	0.837880	0.810543
20	0.933808	0.907343	0.880593	0.845414	0.819204
25	0.946814	0.925338	0.903500	0.874576	0.852867
30	0.955550	0.937483	0.919039	0.894493	0.875981
35	0.961821	0.946230	0.930269	0.908956	0.892827
40	0.966542	0.952831	0.938765	0.919935	0.905647
45	0.970224	0.957988	0.945416	0.928552	0.915730
50	0.973176	0.962129	0.950764	0.935495	0.923868
55	0.975595	0.965527	0.955158	0.941208	0.930572
60	0.977614	0.968366	0.958832	0.945993	0.936192
65	0.979325	0.970772	0.961949	0.950057	0.940971
70	0.980793	0.972839	0.964628	0.953552	0.945084
75	0.982066	0.974632	0.966954	0.956591	0.948661
80	0.983181	0.976204	0.968994	0.959256	0.951800
85	0.984165	0.977592	0.970796	0.961613	0.954578
90	0.985040	0.978827	0.972400	0.963712	0.957054
95	0.985824	0.979933	0.973837	0.965594	0.959273
100	0.986530	0.980929	0.975132	0.967290	0.961275

Table C-2

Percentage Points of $L = \lambda^{P/R}$
for
Three Samples with Equal Number of Failures

Failures r	Level of Significance 0.1	0.05	0.025	0.01	0.005
4	0.543618	0.452491	0.376639	0.295521	0.245982
5	0.617586	0.534185	0.462046	0.381413	0.329905
6	0.671310	0.595418	0.528105	0.450657	0.399710
7	0.711966	0.642754	0.580271	0.506895	0.457619
8	0.743753	0.680338	0.622330	0.553168	0.506004
9	0.769265	0.710856	0.656881	0.591769	0.546836
10	0.790182	0.736106	0.685730	0.624388	0.581658
11	0.807638	0.757331	0.710159	0.652278	0.611649
12	0.822421	0.775417	0.731099	0.676377	0.637719
13	0.835101	0.791007	0.749241	0.697394	0.660571
14	0.846095	0.804582	0.765106	0.715878	0.680753
15	0.855718	0.816508	0.779094	0.732254	0.698701
16	0.864211	0.827066	0.791518	0.746861	0.714760
17	0.871761	0.836479	0.802626	0.759967	0.729210
18	0.878517	0.844923	0.812614	0.771792	0.742279
19	0.884597	0.852539	0.821643	0.782512	0.754154
20	0.890099	0.859444	0.829845	0.792276	0.764990
25	0.911258	0.886119	0.861674	0.830391	0.807483
30	0.925588	0.904292	0.883486	0.856714	0.837002
35	0.935936	0.917466	0.899362	0.875976	0.858690
40	0.943757	0.927454	0.911433	0.890678	0.875292
45	0.949877	0.935287	0.920920	0.902267	0.888408
50	0.954796	0.941593	0.928572	0.911636	0.899030
55	0.958836	0.946780	0.934875	0.919367	0.907807
60	0.962213	0.951121	0.940156	0.925855	0.915181
65	0.965078	0.954807	0.944644	0.931376	0.921463
70	0.967540	0.957976	0.948507	0.936133	0.926880
75	0.969677	0.960730	0.951866	0.940273	0.931598
80	0.971550	0.963145	0.954813	0.943910	0.935744
85	0.973205	0.965281	0.957421	0.947129	0.939416
90	0.974678	0.967182	0.959744	0.949998	0.942692
95	0.975998	0.968886	0.961826	0.952572	0.945631
100	0.977187	0.970422	0.963705	0.954896	0.948287

Table C-3

Percentage Points of $L = \lambda^{e/R}$
for
Four Samples with Equal Number of Failures

Failures r	Level of Significance 0.1	0.05	0.025	0.01	0.005
4	0.435576	0.353429	0.287789	0.220114	0.180040
5	0.519104	0.440349	0.374628	0.303447	0.259138
6	0.581831	0.507968	0.444570	0.373675	0.328100
7	0.630404	0.561591	0.501349	0.432454	0.387154
8	0.669025	0.604967	0.548059	0.481885	0.437642
9	0.700425	0.640692	0.587024	0.523811	0.481002
10	0.726432	0.670585	0.619955	0.559708	0.518490
11	0.748315	0.695943	0.648116	0.590729	0.551140
12	0.766976	0.717714	0.672452	0.617768	0.579783
13	0.783073	0.736600	0.693681	0.641524	0.605085
14	0.797099	0.753134	0.712353	0.662548	0.627579
15	0.809428	0.767727	0.728900	0.681277	0.647696
16	0.820348	0.780700	0.743662	0.698061	0.665785
17	0.830089	0.792307	0.756909	0.713184	0.682134
18	0.838829	0.802751	0.768863	0.726878	0.696978
19	0.846717	0.812200	0.779702	0.739335	0.710511
20	0.853869	0.820787	0.789575	0.750713	0.722899
25	0.881525	0.854155	0.828130	0.795434	0.771824
30	0.900386	0.877063	0.854770	0.826596	0.806130
35	0.914069	0.893758	0.874271	0.849538	0.831496
40	0.924449	0.906464	0.889160	0.867128	0.851006
45	0.932592	0.916457	0.900899	0.881041	0.866474
50	0.939152	0.924522	0.910391	0.892319	0.879037
55	0.944548	0.931167	0.918225	0.901646	0.889442
60	0.949065	0.936738	0.924800	0.909487	0.898200
65	0.952902	0.941474	0.930396	0.916171	0.905673
70	0.956201	0.945551	0.935218	0.921936	0.912125
75	0.959068	0.949097	0.939415	0.926959	0.917751
80	0.961583	0.952209	0.943101	0.931376	0.922701
85	0.963807	0.954963	0.946365	0.935289	0.927089
90	0.965788	0.957417	0.949275	0.938780	0.931005
95	0.967563	0.959617	0.951885	0.941913	0.934523
100	0.969163	0.961601	0.954241	0.944744	0.937704

Table C-4

Percentage Points of L = $\lambda^{P/R}$
for
Five Samples with Equal Number of Failures

Failures	Level of Significance				
r	0.1	0.05	0.025	0.01	0.005
4	0.351917	0.278926	0.222434	0.165922	0.133342
5	0.440177	0.367031	0.307601	0.244799	0.206520
6	0.508438	0.437878	0.378741	0.314022	0.273162
7	0.562373	0.495334	0.437919	0.373534	0.331883
8	0.605897	0.542576	0.487471	0.424558	0.383121
9	0.641685	0.581974	0.529368	0.468471	0.427808
10	0.671595	0.615267	0.565155	0.506505	0.466913
11	0.696946	0.643737	0.596023	0.539678	0.501304
12	0.718695	0.668342	0.622889	0.568817	0.531719
13	0.737553	0.689806	0.646466	0.594584	0.558769
14	0.754056	0.708687	0.667310	0.617513	0.582955
15	0.768616	0.725420	0.685863	0.638036	0.604694
16	0.781556	0.740349	0.702478	0.656505	0.624327
17	0.793131	0.753749	0.717440	0.673207	0.642139
18	0.803545	0.765841	0.730982	0.688380	0.658365
19	0.812964	0.776807	0.743294	0.702222	0.673205
20	0.821524	0.786796	0.754536	0.714899	0.686825
25	0.854775	0.825814	0.798681	0.765022	0.740956
30	0.877595	0.852785	0.829408	0.800223	0.779225
35	0.894223	0.872533	0.852014	0.826281	0.807683
40	0.906876	0.887615	0.869339	0.846339	0.829661
45	0.916827	0.899508	0.883036	0.862253	0.847142
50	0.924857	0.909127	0.894137	0.875183	0.861375
55	0.931474	0.917066	0.903315	0.885898	0.873187
60	0.937020	0.923730	0.911029	0.894919	0.883146
65	0.941736	0.929402	0.917604	0.902620	0.891657
70	0.945795	0.934290	0.923274	0.909270	0.899013
75	0.949325	0.938545	0.928215	0.915070	0.905434
80	0.952423	0.942283	0.932558	0.920174	0.911088
85	0.955165	0.945592	0.936406	0.924699	0.916105
90	0.957608	0.948542	0.939838	0.928739	0.920586
95	0.959798	0.951189	0.942919	0.932368	0.924612
100	0.961774	0.953578	0.945701	0.935649	0.928258

Table C-5

Percentage Points of L = $\lambda^{P/R}$
for
Six Samples with Equal Number of Failures

Failures r	Level of Significance 0.1	0.05	0.025	0.01	0.005
4	0.284224	0.220055	0.171781	0.124820	0.098436
5	0.374293	0.307027	0.253589	0.198315	0.165258
6	0.445891	0.379186	0.324356	0.265428	0.228802
7	0.503523	0.438937	0.384585	0.324612	0.286343
8	0.550663	0.488808	0.435848	0.376272	0.337520
9	0.589829	0.530877	0.479730	0.421337	0.382794
10	0.622833	0.566749	0.517578	0.460782	0.422857
11	0.650993	0.597649	0.550482	0.495480	0.458407
12	0.675287	0.624517	0.579307	0.526173	0.490079
13	0.696450	0.648076	0.604742	0.553474	0.518421
14	0.715045	0.668890	0.627335	0.577891	0.543897
15	0.731509	0.687407	0.647526	0.599842	0.566900
16	0.746185	0.703982	0.665672	0.619670	0.587757
17	0.759348	0.718902	0.682063	0.637662	0.606745
18	0.771219	0.732401	0.696939	0.654055	0.624097
19	0.781978	0.744672	0.710499	0.669050	0.640010
20	0.791775	0.755872	0.722907	0.682815	0.654652
25	0.830001	0.799834	0.771883	0.737542	0.713177
30	0.856388	0.830415	0.806202	0.776255	0.754865
35	0.875692	0.852905	0.831570	0.805054	0.786027
40	0.890426	0.870135	0.851077	0.827305	0.810186
45	0.902039	0.883756	0.866541	0.845007	0.829457
50	0.911428	0.894793	0.879099	0.859423	0.845185
55	0.919175	0.903918	0.889499	0.871390	0.858261
60	0.925676	0.911587	0.898253	0.881481	0.869304
65	0.931210	0.918122	0.905722	0.890105	0.878752
70	0.935977	0.923759	0.912171	0.897561	0.886928
75	0.940126	0.928669	0.917794	0.904070	0.894072
80	0.943770	0.932986	0.922741	0.909801	0.900368
85	0.946996	0.936810	0.927127	0.914887	0.905957
90	0.949873	0.940221	0.931041	0.919430	0.910954
95	0.952453	0.943283	0.934557	0.923513	0.915446
100	0.954781	0.946048	0.937734	0.927207	0.919517

Appendix D

Language Syntax

This appendix contains a brief description of the language syntax that is supported by both the editor and compiler of the Statistical Analysis Environment (SAE). The first section contains a description of the language statements and their various forms. Then tables of the reserved words and language tokens are given and, finally, the language syntax is defined in a set of eighteen syntax rules.

Language syntax discussion:

Environment definition statement:

```
ENVIRONMENT;
ENVIRONMENT (<environment_arguments>);
```

If no argument is given, then defaults are used for all elements of the environment.

Multiple arguments are separated by commas.

Environment_arguments:

MAXTERMS = n

Set the maximum number of terms to be used in calculating the values of the CDF. This is not necessarily the number of terms that will be used ... that number is determined at runtime, however, it cannot be greater than 'n'.

The default is MAXTERMS = 10.

METHOD = EXACT | ASYMTOTIC

Include the program modules necessary for using the Exact or Asymtotic form for the CDF of the statistic.

The default is EXACT.

Program definition statement:

```
"<program_name>" = {TABLE | LRT } [(<argument_list>)];
```

This defines the program to be created. The <program_name> is the file name of the executable program created. This file name is surrounded by double quotes ("). The <argument_list> defines the variables for the specific type of program to be created.

The TABLE generation program will create a table of percentage point values for the ranges given in the <argument_list>.

The LRT program will accept data file names at run-time, then compute the test criteria and the percentage point necessary to determine whether the samples could have come from the same distribution.

Argument list:

The following arguments may appear in the <argument_list> for a TABLE program:

SAMPLES = <integer> ! RANGE (<start>, <stop>, [<increment>])

The number of samples may be given as a single integer or a range of values. A table will be generated for each number of samples in the range.
The default is SAMPLES = 2.
The minimum number of samples is 2.
The default increment is 1.

FAILURES = <integer> ! RANGE (<start>, <stop>, [<increment>])

The number of failures occurring in each sample may, also, be given as a single value or a range. A line will be generated in each table for each number of failures in the range.
The default is FAILURES = 3.
The minimum number of failures in each sample is SAMPLES+1.
The default increment is 1.

ALPHA = <fraction> ! (<fraction>,<fraction>,<fraction>,<fraction>)

The level of significance is represented by the value alpha. Alpha is usually in the range from 0.15 to 0.005, however, any values greater than zero and less than one (0 < alpha < 1) may be used. From one to four values may be given for alpha, separated by commas. A column will be generated in each table for each value of alpha.
The default is ALPHA = 0.10.

The following values may be included in the definition of an LRT program:

SAMPLES = <integer>

For the LRT program, the maximum number of samples must be defined. This only means that any number of samples, up to the maximum, can be used with the program at run-time.
The default is SAMPLES = 2.

ALPHA = <fraction>

The level of significance is represented by the value of alpha. If no value is specified then the user will be prompted to provide a value at run-time. The value may be in the range 0 < alpha < 1.

Comments:

```
-- <comment>
```

Comments begin with two dashes ('--') and are terminated by the end of the line. The comments may appear anywhere on the line, but all text to the right of the dashes is assumed to be comment.

Table D-1

Language Reserved Words

Complete word	Abbreviation
ALPHA	AL
ASYMTOTIC	AS
ENVIRONMENT	EN
EXACT	EX
FAILURES	FA
LRT	LR
MAXTERMS	MA
METHOD	ME
RANGE	RA
SAMPLES	SA
TABLE	TA

Table D-2

Language Tokens

Symbol	Description
EOL	End of line
SIMICOL	Semicolon
EQUALS	Equal sign
LPAREN	Left parenthesis
RPAREN	Right parenthesis
COMMA	Comma
INTEGER	Integer value
FRACTION	Fraction value (real)
FILENAME	File name variable
RESVD	Reserved words

Syntax Rules Notation:

```
     ::=    definition
     <.>    non terminal element
     '.'    terminal element
     [.]    optional element (zero or one occurance)
     {.}    repeated element (one or more occurances)
    [{.}]   optional repeated element (zero or more occurances)
     E      all characters except the new line character
     |      or
     (.)    precedence determination
     <EOL>  end of line
```

Note: White space is ignored between elements.
       Terminal elements are surrounded by single quotes.

Syntax Rules:

```
1.   <compilation_unit> ::=  <command>

2.   <command> ::= <environment_cmd> ';'
                 | <program_cmd> ';'
                 | <comment_stmt>

3.   <environment_cmd> ::= 'ENVIRONMENT' ['(' {<env_list>} ')']

4.   <env_list> ::= <env_arg> | <env_list> ',' <env_arg>

5.   <env_arg> ::= METHOD '=' ('EXACT' | 'ASYMTOTIC')
                 | 'MAXTERMS' '=' <int>

6.   <program_cmd> ::= '"'<filename>'"' '=' <program_def>

7.   <program_def> ::= 'TABLE' ['(' <table_list> ')' ]
                 | 'LRT' ['(' <lrt_list> ')' ]

8.   <table_list> ::= <table_arg> | <table_arg> ',' <table_list>

9.   <lrt_list> ::= <lrt_arg> | <lrt_arg> ',' <lrt_list>

10.   <table_arg> ::= 'SAMPLES' '=' (<int> | <range_def>)
                 | 'FAILURES' '=' (<int> | <range_def>)
                 | 'ALPHA' '=' <alpha_list>

11.   <lrt_arg> ::= 'SAMPLES' '=' <int>
                 | 'ALPHA' '=' <fraction>

12.   <alpha_list> ::= <fraction> | <fraction> ',' <alpha_list>

13.   <range_def> ::= 'RANGE' '(' <int> ',' <int> [',' <int>] ')'
```

```
14.  <filename> ::= '{any non-whitespace printing chars except '"'}'

15.  <int> ::= '1..9' [{<digit>}] | '0'

16.  <fraction> ::= ['0'] '.' [{<int>}]

17.  <digit> ::= '0..9'

18.  <comment_stmt> ::= '--' ['{E}'] <EOL>
```

Appendix E

SAE Operating Manual

Introduction

The Statistical Analysis Environment (SAE) is a programming environment created specifically to assist in performing reliability analysis. The current capabilities of the system provide for the analysis of failure data by determination of the likelihood that several samples came from distributions with the same mean-time-between-failures (MTBF). The following sections describe the operation of the SAE.

SAE Startup

To begin operation of the SAE system, the current directory must contain the executable files 'sae' and 'sae.shell'. The execution of the 'sae' program will print a header line and begin running the shell program. Complete operation of the SAE system's 'compiler' and 'help' functions will require additional files and a specific directory structure (for details, see the following sections). The directory structure and files are shown in Figure E.1.

The system is designed to be run on the UNIX (trademark of Bell Labs) operating system and was developed on the Berkley Version 4.2 UNIX. Assuming the standard percent prompt from the UNIX C-shell program, the SAE system operation begins like this:

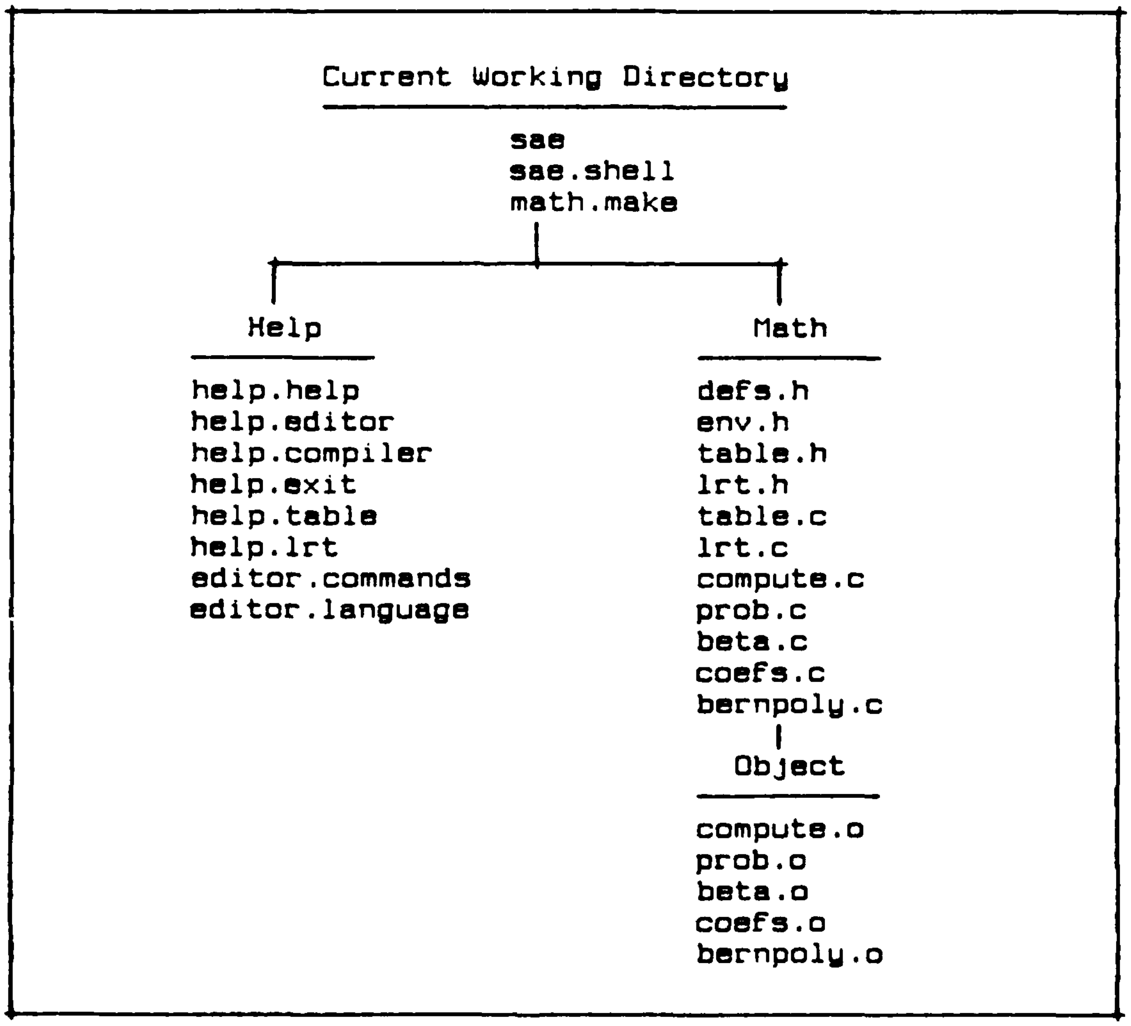


Figure E.1. SAE Directory Structure and Files

```
% sae

Welcome to the Statistical Analysis Environment

SAE >
```

When the prompt 'SAE >' appears the system is ready to accept commands. In general, any valid UNIX command may be given. However, there are some restrictions in calling programs which conflict with SAE program names. These anomolies are discussed in the following sections.

SAE Commands

There are only six SAE commands. These are, simply:

exit or quit	- return to UNIX,
editor	- run the SAE editor,
compile	- run the SAE interpreter,
table	- execute a program named 'table',
lrt	- execute a program named 'lrt',
help or ?	- run the SAE help system.

These commands may be given in any abbreviated form that does not conflict with another SAE command. For example, the editor may be called 'ed', 'edit', 'edi', or 'editor' but not 'e' because that could not be descriminated from the exit command.

Standard UNIX commands may be given in the normal manner, except in cases where their names would conflict with the name of an SAE command. Then the UNIX command may be entered with a back slash preceeding it. For example, '\ed' is the command necessary to run the UNIX editor instead of the SAE editor.

There one other important restriction on the execution of UNIX commands. The current working directory may not be changed. The execution of any non-SAE command is done by starting a copy of the UNIX C-shell program and executing the UNIX command within that shell. Any changes to the working directory are not exported to the SAE shell. This is necessary because the locations of the 'Math' and 'Help' directories are important to the proper operation of the SAE editor, compiler and help functions.

SAE Editor

The SAE editor is a small screen-oriented text editor for creating SAE language programs. The editor may be entered with a specific file to be edited. Example commands are:

```
SAE > ed                    <- no file to edit
SAE > editor textfile       <- 'textfile' is edited
SAE > edit Help/help.me     <- 'help.me' in the Help
                               directory is edited
```

The editor is limited to 15 lines of text to edit. A longer file will be truncated when it is read into the editor. However, only the edited version will be changed. The original file will not be modified.

A backup file is always created when the edited file is saved. The backup file is given the complete name of the edited file, with the extension '.bak' appended to the end.

Commands within the editor are generally given by single characters. The list of available commands will appear at the bottom of the screen by pressing the 'H' key for help. A list of reserved words and language statement formats will appear in the same place in response to the 'L' key. Also, a display of the editor's status may be seen by using the 'S' key. Pressing the same key again will clear the bottom of the screen of any of these displays. These and other editor commands are listed in Table E.1. Many of these are similar to the commands of the UNIX editor 'vi', although not as extensive.

There are four secondary editor commands (see Table E.2). These commands are specifically for reading and writing

Table E.1

Editor Commands

Command	Description
i	begin insert mode
a	begin append mode
o	insert a new line and begin insert mode
<ESC>	end insert or append modes
r	replace a character
dd	delete a line
x	delete a character
v	erase and redraw the screen
ZZ	save the editor contents and exit to SAE
C	turn on/off the compiler
H	turn on/off the help display
S	turn on/off the status display
L	turn on/off the language help display
:	execute secondary editor command

information to and from the editor. The write command (':w <filename>') will write the entire contents of the editor to the file named. If no file is given in the 'write' command, the file named previously (when the editor was started) will be used. The current contents of the file are moved to a backup version.

The exit command (':e <filename>') performs exactly as the write command and then exits to the SAE shell. The quit command (':q') leaves the editor without saving the contents.

The last secondary command for the editor is the read command (':r <filename>'). This will read the contents of the given file into the editor at the line following the current cursor position.

The compiler can be used with the editor to advise the user when proper commands have been created. Upon entering the editor, the compiler is turned off. To start the compiler, press

Table E.2

Secondary Editor Commands

Command	Description
w	write edited text to a file
r	read text from a file to editor
q	quit the editor without saving contents
e	save the editor contents and exit to SAE

the 'C'. On the right side of the screen, a single character will give the status of each line of text. There are three possible values; a period '.' indicates a valid language statement, a dash '-' indicates a partial statement with no errors, and the letter 'E' indicates an error. If the status display is turned on (press 'S') the error message will appear at the bottom of the screen for the line at the current cursor position. If the 'C' is pressed again, the compiler will be turned off and the right side of the screen will be blanked. The compiler error messages and status characters are summarized in Table E.3.

If the compiler is on at the time the editor contents are written to a file (with the ':e', ':w' or 'ZZ' commands) then the program will be executed as if the file were compiled outside the editor. If the compiler is turned off, the file will only be stored as text.

SAE Compiler

Outside the SAE editor, the SAE compiler may be run on any

Table E.3

Compiler Error Messages

```
Not compiled
VALID
VALID - Incomplete line
ERROR - Non-specific
ERROR - Invalid character
ERROR - Expected filename or ENVIRONMENT
ERROR - Expected semicolon or left paren
ERROR - Expected comma or right paren
ERROR - Expected semicolon at end of command
ERROR - Expected argument (see language help)
ERROR - Expected equal sign
ERROR - Expected integer
ERROR - Expected TABLE or LRT
ERROR - Expected fraction
ERROR - Expected integer or RANGE
ERROR - Expected left paren
ERROR - Expected right paren
ERROR - Expected comma
ERROR - Invalid integer or fraction
ERROR - Invalid file name
ERROR - Invalid word in source
```

text file by giving the command "compile <filename>". The file will be interpreted exactly the same as if it were in the editor. Valid statements will be executed and invalid statements will generate error messages (see Table E.3). The contents of the source file will be written back to the standard output device with error messages following the line in which they occurred. The file output will be followed by a summary stating the number of lines read and the number of errors that were found. If there are no errors, the statements will be executed.

The result of execution will be a program. There are two types of programs that may be generated by the system. First, a 'table' program can be used to generate a table of percentage point values for applying the statistic developed in this dissertation. This table may be used to evaluate a number of samples to determine if they may have come from the same distribution. The table values are only for cases where there are exactly the same number of failures recorded for each sample. A single percentage point will be computed by the program for each specific case of three values; the number of samples, the number of failures in each sample, and the level of significance (alpha).

The second type of program (lrt) will accept faliure date and compute the specific percentage point needed. Then the program will make a statement on whether the samples may have come from the same distribution, based on the given level of significance. This program will operate interactively, accepting failure data from disk-files and prompting the user for information that is not provided.

Specific information on the execution of the 'lrt' and 'table' programs is provided in the following sections. Here we are concerned with the creation of the programs by the compiler. The compiler statements provide for the input of value defaults and ranges that determine the operating features of the resulting programs. There are only three types of statements that are acceptable to the compiler; comment statements, environment statements and program statements. The general forms for these statements are given in Table E.4.

Comment statements may appear at any place in the program. They begin with two consecutive dashes '--' and are terminated by the end of the line. Comment lines are ignored by the compiler

Table E.4

SAE Compiler Statements

```
-- A comment <CR>

        Comments are terminated by the end of the
        line and may appear at any point in the line.

environment (maxterms = n, method = {exact|asymt});

        Environment statement may appear only once
        in the program.  Parentheses are necessary
        if either argument is present.  'n' must
        be a positive integer.
        Defaults:  maxterms = 10, method = exact.


"file-name" = table (samples = {n | range(n1,n2,n3)},
                     failures = {n | range(n1,n2,n3)},
                     alpha = r1, r2, r3, r4);

        Table program definition may contain three
        arguments.  'n' must be a positive integer.
        'r' must be a positive fraction less than
        one.  'n1' is the starting value.  'n2' is
        the ending value.  'n3' is the increment.
        Defaults:  samples = 2, failures = 5,
                   alpha = 0.10, n3 = 1.


"file-name" = lrt (samples = n, alpha = r);

        Lrt program definition may contain two
        arguments.  'n' must be a positive integer.
        'r' must be a positive fraction less than
        one.
        Defaults:  samples = 2, alpha = 0.10.
```

and are not passed on to the resulting program in any way.

Environment statements begin with the reserved word 'environment' and define two important features of the 'lrt' and 'table' program. The first feature is the method used to compute the CDF of the test criteria, either exactly or asymtotically. The mathematical implications will not be discussed here.

However, there are some simple considerations. If there are only two samples to be compared, the exact method is calculated slightly faster, but, there is no significant difference in the values produced by either method. If there are more than two samples and the number of failures in each sample is large (greater than 50), the asymtotic method is more efficient.

The second feature determined by the environment statement is the maximum number of terms used in the calculations for either method. For only two samples, this number is not used in the calculations. However, for more than two samples the number of terms necessary for the same accuracy is inversely proportional to the number of failures in each sample. That is, more terms are necessary when fewer failures have occurred in each sample. More than twenty terms are necessary for the most general use (e.g. generating tables with a wide range of values). Ten terms are sufficient if the number of failures remains above about 40 in each sample. In general, the asymtotic method requires fewer terms, usually about half, for the same accuracy as the exact method.

The program statements are used to create 'table' and 'lrt' programs. The statement is an assignment of a program description to a file name (see Table E.4). The program is identified by the reserved words 'table' and 'lrt' and the definition is completed by the specification of the argument values; samples (the number of samples to be compared), failures (the number of failures in each sample) and alpha (the level of significance for the percentage point).

Table E.5

SAE Compiler Reserved Words

Reserved Word	Abbreviation
ALPHA	AL
ASYMTOTIC	AS
ENVIRONMENT	EN
EXACT	EX
FAILURES	FA
LRT	LR
MAXTERMS	MA
METHOD	ME
RANGE	RA
SAMPLES	SA
TABLE	TA

The 'failures' argument is not allowed in the definition of an 'lrt' program because the number of failures is taken directly from the data files that are read during program execution. Also, there can only be one value assigned to 'samples' and 'alpha' (the level of significance).

The value of 'samples' in the 'lrt' program is not very important. It is only the maximum number of samples that may be processed by the program. For example, if the maximum is set to 10, any number of samples (from 2 to 10) may be compared by the program. The value of 'alpha' is, also, not absolutely necessary to the statement. If 'alpha' is not set in the 'lrt' program definition, the program will prompt the user for a value during operation. More details of the 'lrt' program's operation are given in a later section.

In the 'table' program definition, each of the arguments may be assigned more than one value. The 'range' can be specified for the 'samples' and 'failures' arguments by specifing the values of initial-value, maximum-value, and incremental-value, as shown in Table E.4. The level of significance ('alpha') may have up to four values (separated by commas). A table will be generated by the program for each distinct value of 'samples'. There will be a column in each table for each value of 'alpha' and a row for each value of 'failures'.

It is important to remember that the environment may be defined only once in every program. However, a valid SAE program may contain many 'table' and 'lrt' definition statements.

Abbreviations may be used for any of the reserved words listed in Table E.5. The compiler does not descriminate between capital and small letters in reserved words. So, 'Exact' would be accepted as well as 'eXaCt' for the same word. This is not true for file names given within the required quotes. Any printing characters, except blanks, may legally appear in a file name. It may be helpful to read the next two sections on the operation of the 'table' and 'lrt' programs before beginning to use the compiler.

SAE Table

The SAE table command will execute a program named 'table'. In general, this is not very useful, because most table generation programs created by using the SAE will not be named 'table'. The command is included more to provide easy access to a 'help' information file on the use of the table programs created by the compiler and their default argument values.

Table generation programs will execute with no user inputs and generate the tables specified in the 'table' definition statement used to create the program. The table will appear at the standard UNIX output which may be re-directed using standard C-shell features (e.g. '>' and '>>').

SAE lrt

The SAE lrt command is, mostly, just for reference to an appropriate 'help' file. Like the table command, a file named 'lrt' will be executed. However, unlike the table programs, user input is usually necessary. There are three important features of 'lrt' programs for the user to know. First, if the level of significance was set when the program was created, then it cannot be changed while the program is running. If it was omitted in the definition statement, the program will prompt the user for a value during each execution. There is no default value for 'alpha'.

Second, the names of the data files must be provided to the program. There must be one data file for each sample. The names may be given in the calling statement for the program or the program will prompt the user. For example, 'lrt data1 data2', 'lrt data3 data2 data1' and 'lrt' are all valid program calls. Only in the last case will the program prompt the user for data files.

The third important point is the organization of the data files. Each data file is expected to contain a list of failure times for the sample. These may appear in any acceptable UNIX format for real numbers (e.g. 1.2, 0.5, 1.3E2, etc.). There is, however, one more value that is needed; the sample size. That is, the number of items under test in the sample, not the number of failures (which can be calculated from the size of the data file). The sample size is expected to be an integer value given as the first entry in the data file. If this is not true for the data files to be used, the program can prompt the user. The file name must be prefixed with a dash ('-') to tell the program that the sample size is not included in the data file. For example, the command 'lrt -data1 data2 -data3' would indicate that only the second data file contains the sample size within the file. The user would be prompted to supply the sample sizes for the first and third files when they are read.

The 'lrt' program outputs the computed value for the test criteria, the percentage point at the given level of significance, and a statement indicating the decision indicated by the values computed. As with the 'table' program, these values are sent to the UNIX standard output device.

SAE Help

The SAE help command is provided to allow the user easy access to an organized system of information files. The command is given as 'help <argument>' where <argument> is the name of the subject disired. The question mark (?) may be used in place of the command name 'help'. Examples of help commands are:

```
SAE > he exit              <- exit information
SAE > ?                    <- general help information
SAE > ? editor.commands    <- list 'Help/editor.commands'
SAE > help c               <- compiler information
```

The arguments may be SAE command names or specific file names in the Help directory. SAE command names may be given in the same manner as would be accepted by the SAE shell parser. For example, 'h ed' would get information about the SAE editor and 'h c' would provide the compiler's description. If the argument is not a valid SAE command, the help program searches for a file with the name 'Help/xxx' where 'xxx' was the argument given in the command, as in 'help xxx'.

All help-information files are expected to be found in a directory named 'Help' in the current working directory. Files are then named in two parts. First, the name of the program that is intended to read them is given. This is followed by a period and some indication of the specific information contained in the file. For example, a file with information about the SAE editor for use with the system help command would be named 'help.editor'. So, a file with editor command information for use by the editor would be named 'editor.cmds'.

If there are no arguments given in the help command, then the first lines of all files referenced by the UNIX path name 'Help/help.*' are displayed. These files are intended to be used for describing SAE commands and the first line contains the name of the command and an indication of its usage. If more files are added to the Help directory, they should follow this format.

Expansion of the SAE System

The SAE system is designed to be extensible is several ways. This allows the user a choice when new tools or features are needed.

The shell of the environment allows any UNIX programs to be executed, even command lists. This allows the user to create programs outside the SAE environment using any tools that are available within the UNIX system.

Programs may also be made an explicit part of the SAE by modifying the SAE-shell programs to properly handle the program name according to the SAE conventions. This will not restrict the use of the program to the SAE, but only make calling the program easier (only enough of the program name to make it unique need be entered for SAE commands).

The final method for adding a program to the SAE involves modification of the programs that are linked together and loaded as the SAE, such as the editor and the shell, itself. This will increase the speed of execution but, obviously, increase the size of the SAE program. For small functions that may be used often, this proceedure should be considered. Another drawback to this method is that program cannot be used when the SAE is not loaded.

Limitations of the SAE Environment

There are only a few limitations within the SAE environment. The most important is that the current working directory cannot be changed. This was done to make it easier to find the help-information files which are stored in a sub-directory. This means that each directory in which the SAE is used can have it's own help-information, tailored to the needs of the user. Other methods could have been used but this was simple and does not seriously restrict the user while giving the ability to modify help files.

Another limitation of the SAE is also one of its strengths. That is, the ability of the programmers to modify almost everything about the environment can lead to destruction of the system. The environment's integrity is dependent upon the users adhering to good programming practices and obeying common sense regarding the protection of valuable programs and information.

SAE Programs

The next appendix of this dissertation contains listings of the programs that make up the SAE. Copies of these programs may also be obtained from the author or by contacting the AFIT School of Engineering, Wright-Patterson Air Force Base, Dayton, Ohio, 45433.

Sample Operation of the SAE

The remaining pages of this appendix contain an example of the use of the SAE system. As mentioned in the first section 'SAE Startup', the first command to the UNIX system is 'sae' to begin operation of the SAE shell program. Comments to explain the commands being used are boxed to keep them seperate from the normal output. All user commands are underlined to make the system responses obvious.

It should also be mentioned that this example does not contain a demonstration of the SAE editor. It was not practical to show the editor displays in this type of listing. Instead, the SAE language program was created beforehand, using another UNIX system editor. Operation of the editor is explained in detail in other sections of this appendix and the displays are self-explanitory during operation.

```
Script started on Tue Nov 18 11:42:11 1986
% sae                                   +------------------+
                                        | starting the SAE |
                                        +------------------+
Welcome to the Statistical Analysis Environment - Version 1.2

SAE > h                                      +-------------------------+
SAE Help command, Version 2.0                | SAE help command called |
                                             | with no argument        |
Command         Description                  +-------------------------+
-------         -----------

compile         SAE Language Compiler..  'compile [file]'

editor          SAE Syntatical Editor..   'edit [file]'

exit            SAE Exit command.. 'exit' or 'quit'

help            Help information is displayed..  'help [cmd]'

table           'table' is a compiler program statement

SAE > ? h                                    +-----------------------+
SAE Help command, Version 2.0                | information on 'help' |
                                             +-----------------------+
help            Help information is displayed..  'help [cmd]'
-------------------------------------------------------------
The 'help' command selectively displays the information in e
'Help' subdirectory of the current working directory.

With no specified 'command', the first line of each help-information
file is displayed.  This should give a list of commands and simple
descriptions.

If a 'command' name is given, then the entire file named 'help.command'
is displayed.  For example 'help help' will display this file.

Any unique abbrevation of the command name may be given.
The back-slash '\' may be used to escape from this shell's parser and
execute a UNIX command with the same name (eg. the editor 'ex' can
be run by using '\ex').


SAE > help compi                             +--------------------------+
SAE Help command, Version 2.0                | information on 'compile' |
                                             +--------------------------+
compile         SAE Language Compiler..  'compile [file]'
-----------------------------------------------------------
Run the compiler on a source file.
Compiler will echo the lines of the file and indicate any errors
     after each line.
A summary of the number of lines compiled and errors found will be
     printed at  the end of the file listing.
The commands in the file will be executed if there are no errors.
```

```
+---------------------------------------------------------+
| UNIX commands may be called...                          |
| e.g.  'man', 'ls', any aliases or script commands, etc. |
+---------------------------------------------------------+

SAE > ls -R
Functions Help      Math      Source    b720.1         b720.2
b720.3    d1        d2        d3        data1          data2
direct    math.make sae       sae.shell table10.src    test15e
test15e.src

Help:
editor.commands     editor.reserved          help.compiler
help.editor         help.editor.commands     help.exit
help.help           help.lrt                 help.table

Math:
Object    Output    asymt.100 bernpoly.c     beta.c
coefs.c   compute.c defs.h    env.h          env.sav
lrt.c     lrt.h     makefile  prob.c         table.c
table.h

Math/Object:
bernpoly.o     beta.o    coefs.o   compute.o prob.o

Source:
Object         Version        cp.compile.c   cp.err.c
cp.token.c     ed.cmd.c       ed.disp.c      ed.functions.c
ed.help.c      ed.misc.c      ed.table.c     makefile
sae.compile.c  sae.edit.c     sae.h          sae.help.c
sae.main.c     sae.parser.c   sae.shell.c    status.h
tokens.h       window.h

Source/Object:
cp.compile.o   cp.err.o       cp.token.o     ed.cmd.o
ed.disp.o      ed.functions.o ed.help.o      ed.misc.o
ed.motion.o    ed.table.o     sae.compile.o  sae.edit.o
sae.help.o     sae.parser.o

Source/Version:
sae.shell


+---------------------------------------------------------+
| an SAE program has been created containing a comment    |
| and two statements defining a program for comparing     |
| up to ten samples                                       |
+---------------------------------------------------------+

SAE > cat test.src
-- Test program
environment (maxterms = 15, method = exact);
"test15e" = lrt (samples = 10);

SAE > mv test.src test15e.src
SAE > c test15e.src                 +-----------------------------+
                                    ! compile the SAE program to  !
SAE Compiler, Version 0.2           ! create the executable UNIX  !
                                    ! program 'test15e'           !
Compiling test15e.src:              +-----------------------------+

1  -- Test program
2  environment (maxterms = 15, method = exact);
3  "test15e" = lrt (samples = 10);

Summary:  3 lines   0 error

SAE > cat data1 data2               +-------------------------+
17                                  ! data files from example !
0.7                                 ! given in Chapter 6      !
1.6                                 +-------------------------+
3.0
5.1

17
1.5
2.3
6.7
8.5
                                    +------------------------------+
SAE > test15e data1 data2           ! run the program on the data !
                                    +------------------------------+
Likelihood Ratio Test Program   Version 2.0

Exact Method for calculating test criteria
    Maximum number of terms used: 15
    10 Samples may be given
    Level of significance will be requested later

Summary:
Test criteria from data files = 0.934432
Enter desired level of significance: 0.1

Probability that LRT value is less than or equal to 0.934432
   when the sample populations have the same failure rate = 0.475

Reference point @ 0.100 Significance = 0.698804
Therefore:
 .. Samples MAY have come from populations with the same
    failure distributions.

+-----------------------------------------------------------+
! Results indicate that at that level of significance the  !
! samples may have the same MTBF's.  Also, notice the      !
! probability value of 0.475.  This is the level of        !
! significance at which the test would have switched its   !
! indicated result.  Sometimes this is called the          !
! 'p-value' for the statistic and this data.               !
+-----------------------------------------------------------+
+--------------------------------------------------------+
! Now, we will test a more realistic set of data.   The  !
! data files named 'b720' contain failure data from      !
! Boeing 720 airconditioning systems compiled during a   !
! study by Proschan in 1963.  Since, all three samples   !
! are from the same population of systems, we would      !
! expect that they have the same failure distribution.   !
!                                                        !
! Note that there are different numbers of failures and  !
! sample sizes in these data files.                      !
+--------------------------------------------------------+

SAE > cat b720.*
27   sample size

Failure times
1 4 11 16 18 18 24 31 39 46 51 54 63 68

23 sample size

Failure times
7 9 14 18 18 22 31 34 34 36 37 57 58 62 65

29 sample size

Failure times
10 14 20 23 24 25 26 29 44 44 49 56 60 61


SAE > test15e

Likelihood Ratio Test Program   Version 2.0

Exact Method for calculating test criteria
     Maximum number of terms used: 15
     10 Samples may be given
     Level of significance will be requested later
Enter file name of first sample: b720.1
Enter file name of next sample: b720.2
Enter file name of next sample: b720.3
Enter file name of next sample:

Summary:
Test criteria from data files = 0.958349
Enter desired level of significance: 0.5

Probability that LRT value is less than or equal to 0.958349
   when the sample populations have the same failure rate = 0.549

Reference point @ 0.500 Significance = 0.95207
Therefore:
 .. Samples MAY have come from populations with the same
    failure distributions.
```

```
SAE > test15e b720.1 b720.2 b720.3

Likelihood Ratio Test Program   Version 2.0

Exact Method for calculating test criteria
     Maximum number of terms used: 15
     10 Samples may be given
     Level of significance will be requested later

Summary:
Test criteria from data files = 0.958349
Enter desired level of significance: 0.6

Probability that LRT value is less than or equal to 0.958349
   when the sample populations have the same failure rate = 0.549

Reference point @ 0.600 Significance = 0.964449
Therefore:
 .. Samples CANNOT be assumed to have come from populations
    with the same failure distributions

+----------------------------------------------------------+
! The test was run twice, at different significance        !
! levels, only to indicate that the results can be forced  !
! to accept or reject the hypothesis by level selected.    !
+----------------------------------------------------------+

+----------------------------------------------------------+
! The last portion of this demonstration is the use of     !
! the table generating feature of the system.   For        !
! simple cases, where the sample size and number of        !
! failures are equal, tables can be useful for applying    !
! the test where the computer program cannot be run.       !
+----------------------------------------------------------+

SAE > cat table10.src
-- Test program
en (maxterms = 10, method = asymtotic);
"table10" = ta
        (samples = range(2,4,1),
        failures = range(3,10,1),
        alpha = (0.1, 0.05, 0.025));

SAE > table10

SAE Table Generation Program    Version 2.0

Asymtotic Method is used in calculating the test criteria
     10 Terms used
```

```
Percentage points of L = lambda to the power of (p/R)
     p = 2 samples
     R = 2 x (number of failures in each sample)

            Level of Significance
Failures    0.100         0.050         0.025
per sample

    3       0.615765      0.504265      0.411661
    4       0.699068      0.602382      0.516920
    5       0.753169      0.669122      0.592130
    6       0.790948      0.717051      0.647779
    7       0.818767      0.753009      0.690372
    8       0.840085      0.780935      0.723924
    9       0.856933      0.803227      0.750995
   10       0.870579      0.821425      0.773278


Percentage points of L = lambda to the power of (p/R)
     p = 3 samples
     R = 3 x (number of failures in each sample)

            Level of Significance
Failures    0.100         0.050         0.025
per sample

    4       0.545448      0.455703      0.382178
    5       0.618568      0.535975      0.465266
    6       0.671892      0.596504      0.530110
    7       0.712338      0.643458      0.581593
    8       0.744004      0.680819      0.623244
    9       0.769443      0.711198      0.657537
   10       0.790312      0.736358      0.686216


Percentage points of L = lambda to the power of (p/R)
     p = 4 samples
     R = 4 x (number of failures in each sample)

            Level of Significance
Failures    0.100         0.050         0.025
per sample

    5       0.520677      0.443047      0.379236
    6       0.582797      0.509675      0.447580
    7       0.631036      0.562729      0.503401
    8       0.669460      0.605760      0.549511
    9       0.700735      0.641265      0.588086
   10       0.726662      0.671012      0.620753

SAE > exit
```

Appendix F

Program Listings

This appendix contains the program listings for the SAE system. The first pages contain a summary of the program files and their contents. Next, Figures F.1 and F.2 show the relationships among the SAE files and the Math files respectively. Finally, the program listings are presented.

SAE Programs List

File Name	Contents/Modules
makefile	UNIX makefile for the SAE programs
sae.h	definitions common to all SAE programs
status.h	definitions of compiler status values
tokens.h	definitions of language reserved words and tokens
window.h	definitions of direct screen manipulation functions
sae.main.c	main (sae)
sae.shell.c	main (sae.shell)
sae.help.c	help
sae.edit.c	edit
sae.compile.c	compile
	cpline
cp.token.c	token
	ckword
	readnumber
	readfilename
cp.compile.c	cparse
	setoutputon
	setoutputoff
cp.err.c	errtext
sae.parser.c	parser
	parseck
ed.disp.c	initdisplay
	readydisplay
	refreshdisp
	switchdisplay
	ckdisplay
	updatedisplay
	switchcflag
ed.help.c	inithelp
	switchhelp
	switchlang
	readhelp
	refreshhelp
	ckhelp
ed.table.c	cklocy
	cklocx
	length
	tabnewline
	tabdeleteline
	lastline
	inittable
	writetext
	readtext
	tableread
	tablewrite
	compileline
	compileall
	clearcompile

ed.cmd.c	cmdcmd
	cmdinit
	cmdprint
ed.misc.c	motion
	getkey
	motioninit
	mvcursor
	ederror
ed.functions.c	edappend
	edinsert
	edinsertln
	eddelline
	eddelchar
	edreplacechar
	edwrite
	edread
	edredraw
	edinit
	edexit
	edcommand

Math Programs List (Asymtotic and Exact Computations)

File Name	Contents/Modules
math.make	UNIX makefile for the math functions
defs.h	definitions common to all programs
env.h	definitions modified by 'environment' command
table.h	definitions modified for 'table' program
lrt.h	definitions modified for 'lrt' program
lrt.c	main (lrt)
table.c	main (table, asymt, exact)
compute.c	readfile
	computelrt
	computepct
prob.c	newton
	prob
beta.c	betap
	ibeta
	beta
	loggam
bernpoly.c	bernpoly
	b
	com
coefs.c	modR
	modS
	createCjr
	Queue

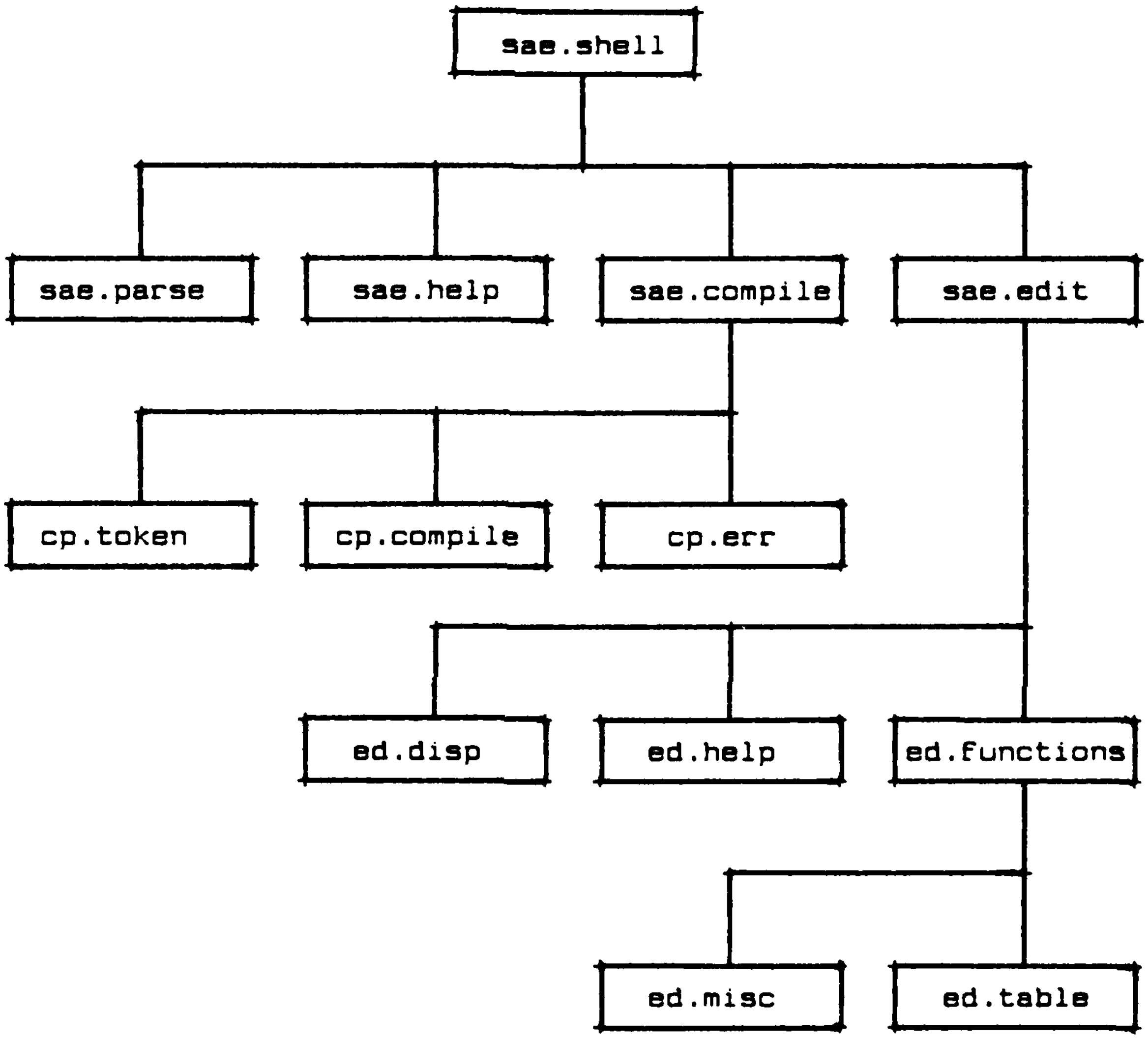


Figure F.1 SAE Basic Program Structure

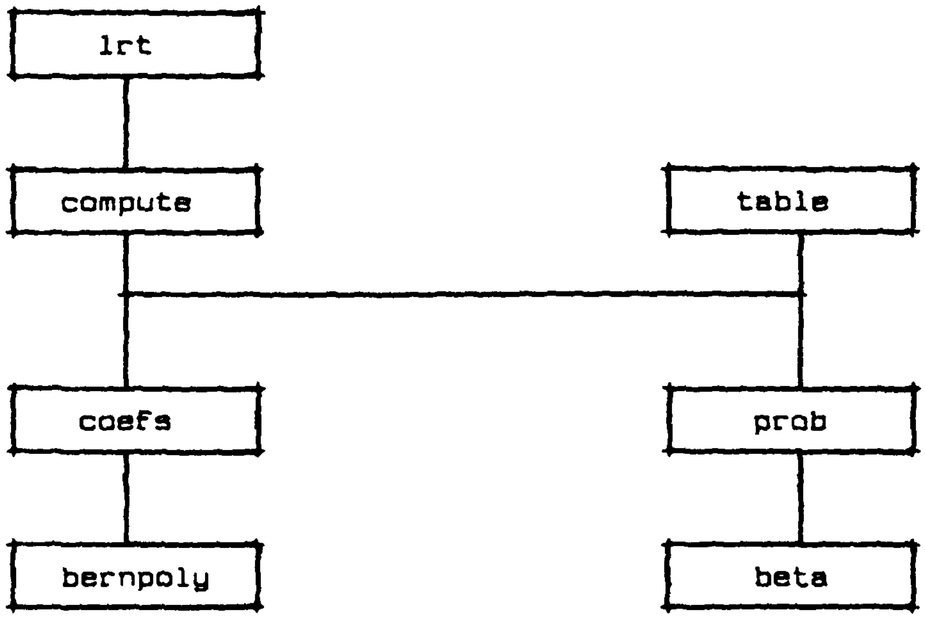


Figure F.2 Math Basic Program Structure

```
#/*******************************************************
#**                                                   **
#**         Statistical Analysis Environment          **
#**                                                   **
#**  File:  makefile                                  **
#**  Original:  20 September 1985 - K N Cole          **
#**  Current:  15 October 1986 - K N Cole             **
#**                                                   **
#**  Contains UNIX make instructions for SAE          **
#**                                                   **
#*******************************************************/
OBJECTS = Object/sae.edit.o Object/sae.help.o \
        Object/sae.parser.o Object/ed.functions.o \
        Object/ed.help.o Object/ed.misc.o \
        Object/ed.cmd.o Object/ed.disp.o \
        Object/ed.table.o Object/sae.compile.o \
        Object/cp.token.o Object/cp.compile.o Object/cp.err.o

CFLAGS = -c

sae.shell: sae.shell.c $(OBJECTS)
        cc -o sae.shell sae.shell.c $(OBJECTS) -lcurses -ltermlib
        mv ../sae.shell Version
        mv sae.shell ..

sae: sae.main.c
        cc -o sae sae.main.c -lcurses -ltermlib

Object/ed.misc.o: ed.misc.c sae.h window.h
        cc $(CFLAGS) ed.misc.c
        mv ed.misc.o Object

Object/ed.functions.o: ed.functions.c sae.h window.h
        cc $(CFLAGS) ed.functions.c
        mv ed.functions.o Object

Object/ed.table.o: ed.table.c status.h sae.h window.h
        cc $(CFLAGS) ed.table.c
        mv ed.table.o Object

Object/sae.parser.o: sae.parser.c sae.h
        cc $(CFLAGS) sae.parser.c
        mv sae.parser.o Object

Object/sae.compile.o: sae.compile.c tokens.h status.h sae.h
        cc $(CFLAGS) sae.compile.c
        mv sae.compile.o Object

Object/cp.compile.o: cp.compile.c tokens.h status.h sae.h
        cc $(CFLAGS) cp.compile.c
        mv cp.compile.o Object

Object/cp.token.o: cp.token.c tokens.h status.h sae.h
        cc $(CFLAGS) cp.token.c
        mv cp.token.o Object
```

```
Object/cp.err.o: cp.err.c tokens.h status.h sae.h
        cc $(CFLAGS) cp.err.c
        mv cp.err.o Object

Object/sae.edit.o: sae.edit.c sae.h
        cc $(CFLAGS) sae.edit.c
        mv sae.edit.o Object

Object/sae.help.o: sae.help.c sae.h
        cc $(CFLAGS) sae.help.c
        mv sae.help.o Object

Object/ed.help.o: ed.help.c sae.h
        cc $(CFLAGS) ed.help.c
        mv ed.help.o Object

Object/ed.cmd.o: ed.cmd.c
        cc $(CFLAGS) ed.cmd.c
        mv ed.cmd.o Object

Object/ed.disp.o: ed.disp.c sae.h
        cc $(CFLAGS) ed.disp.c
        mv ed.disp.o Object
```

```
/*****************************************************
**                                                  **
**          Statistical Analysis Environment        **
**                  SAE DEFINITIONS                 **
**                                                  **
**  File: sae.h                                     **
**  Original: 29 April 1986 - K N Cole              **
**  Current:  6 October 1986 - K N Cole             **
**                                                  **
*****************************************************/

#ifndef TRUE
#define TRUE            1
#endif
#ifndef FALSE
#define FALSE           0
#endif
#define boolean         int

#define EDLINES         15
#define EDCOLS          80
#define MAXLINE         EDCOLS-2
#define MAXTEMP         256
#define MAXCOMMAND      80
#define TABSIZE         8
#define ERRORCODE       999

#define NOLINE          '~'
#define UPLN            '\013'
#define DOWN            '\012'
#define LEFT            '\b'
#define RIGHT           '\014'
#define ESC             '\033'
#define BELL            '\007'
#define TAB             '\t'
#define CR              '\r'
#define NL              '\n'
#define BLANK           ' '

#define EDITOR          0       /* shell command codes */
#define HELP            1
#define TABLECMD        2
#define LRTCMD          3
#define COMPILER        4
#define RETURN          5
#define EXIT            6

                        /* COMPILER output files */
#define CPENVFILE       "Math/env.h"
#define CPTABLEFILE     "Math/table.h"
#define CPLRTFILE       "Math/lrt.h"

                        /* SYSTEM HELP command definitions */
#define COMPILERHELP    "compiler"
#define EDITORHELP      "editor"
```

```
#define TABLEHELP       "table"
#define LRTHELP         "lrt"
#define EXITHELP        "exit"
#define HELPHELP        "help"

                        /* EDITOR HELP program definitions */
#define EDHELPFILE      "Help/editor.commands"
#define EDLANGFILE      "Help/editor.reserved"
#define HELPLINES       7       /* number of lines */
#define HELPCOLS        70      /* number of columns */
#define HELPINDT        5       /* beginning column */
#define HELPSTART       16      /* beginning line */

                        /* DISPLAY program definitions */
#define DISPLINES       7       /* number of lines */
#define DISPCOLS        70      /* number of columns */
#define DISPINDT        5       /* beginning column */
#define DISPSTART       16      /* beginning line */

                        /* COMMAND program definitions */
#define CMDLINE         15      /* line number of command line */

                        /* COMPILER program definitions */
#define MAXSYMBOLS      20      /* maximum size of symbol storage */
#define MAXWORD         80      /* maximum size of filename */
#define DEFMXTERMS      10
#define DEFMETHOD       1       /* 1 EXACT  0 ASYMTOTIC */
#define DEFSAMPLES      2
#define DEFALPHA        0.1
#define DEFINC          1
```

```
/*****************************************************
**                                                 **
**          Statistical Analysis Environment       **
**                       MAIN                      **
**   File:  sae.main.c                             **
**   Original: 14 April 1986 - K N Cole            **
**                                                 **
**   Contains Modules:                             **
**      main - MAIN program routine                **
**                                                 **
**   Calls Modules                                 **
**      sae.shell - shell program (sae.shell.c)    **
**                                                 **
*****************************************************/

#include <stdio.h>
#include <curses.h>

main()
{
int status, proc_id;

/* Display Introduction */
printf("\nWelcome to the Statistical Analysis
                                   Environment - Version 1.2\n");

/* Save current screen characteristics */
savetty();

/* Start SHELL program */
if ((proc_id = fork()) == 0)
  execl("/bin/csh", "csh", "-c", "sae.shell", NULL);

while (wait(&status) != proc_id);    /* wait for shell to terminate */
if (status) resetty();            /* reset terminal characteristics */

/* Exit when SHELL program terminates */
exit();
}
```

```
/*******************************************************
**                                                    **
**          Statistical Analysis Environment          **
**                        SHELL                       **
**   File:  sae.shell.c                               **
**   Original:  7 April 1986 - K N Cole               **
**   Current:  6 October 1986 - K N Cole              **
**                                                    **
*******************************************************/

#include <stdio.h>
#include "sae.h"
char argstore[MAXCOMMAND];       /* command line storage */

main()
/* Main program loop... prompt - read - execute
*/
{
int procid, status;
char *argpointer;

while (TRUE) {
  printf("\nSAE > ");                          /* prompt */
  argpointer = argstore;
  fgets(argpointer, MAXCOMMAND, stdin); /* read command line */
  switch (parser(&argpointer)) {         /* read and execute */
    case EDITOR:          /* run EDITOR */
      edit(argpointer); break;
    case HELP:            /* display COMMAND EXPLAINATIONS */
      help(argpointer); break;
    case TABLECMD:        /* run TABLE program */
      system("table"); break;
    case COMPILER:        /* COMPILE program */
      compile(argpointer); break;
    case EXIT:            /* exit to operating system */
      putc('\n',stdout); exit(); break;
    case RETURN:          /* ignore simple returns */
      break;
    default:              /* try to execute UNIX system command */
      argpointer = argstore;
      while (*argpointer && (*argpointer != NL)) argpointer++;
      *argpointer = NULL;
      procid = fork();
      if (procid == 0)
        execl("/bin/csh", "csh", "-c", argstore, NULL);
      else if (procid == -1)
        printf("\n*** Cannot Execute Command '%s'\n", argstore);
      else {
        while (wait(&status) != procid);
        if (status)
          printf("\n*** Error in Executing Command '%s'\n",argstore);
        }
      break;
} } }
```

```
/*****************************************************
**                                                  **
**         Statistical Analysis Environment         **
**                     PARSER                       **
**                                                  **
**  File:  sae.parser.c                             **
**  Original:  7 April 1986 - K N Cole              **
**  Current:  6 October 1986 - K N Cole             **
**                                                  **
**  Contains Modules:                               **
**      parser                                      **
**      parseck                                     **
**                                                  **
*****************************************************/

#include <stdio.h>
#include <ctype.h>
#include "sae.h"

/***************************************************/

int
parser(arg)
char **arg;
/* parser - parses the input for valid command names.  Returns boolean
   indication for valid command inputs.
*/
{
int value;
char *argpointer;

argpointer = *arg;
switch (*argpointer) {
  case 'c':
    value = parseck(argpointer,"ompiler", COMPILER); break;
                                                /* compiler */
  case 'e':
    switch (*(++argpointer)) {
      case 'x':
        value = parseck(argpointer,"it", EXIT); break;
                                                /* exit */
      case 'd':
        value = parseck(argpointer,"itor", EDITOR); break;
                                                /* edit */
      default:
        value = ERRORCODE; break;
      }
    break;
  case 'q':
    value = parseck(argpointer,"uit", EXIT); break;
                                                /* exit */
  case 't':
    value = parseck(argpointer,"able", TABLECMD); break;
                                                /* table */
  case 'l':
    value = parseck(argpointer,"rt", LRTCMD); break;
                                                  /* lrt */
  case 'h':
    value = parseck(argpointer,"elp", HELP); break;
                                                  /* help */
  case '?':
    value = HELP; break;
  case '\n':
  case '\r':
    value = RETURN; break;              /* only a return */
  case '\\':
    value = ERRORCODE; break;

  default:
    value = ERRORCODE; break;           /* error */
  }

/* move pointer to beginning of arguments ... after command word */
while (*argpointer && !isspace(*argpointer)) argpointer++;
while (*argpointer && isspace(*argpointer)) argpointer++;
*arg = argpointer;

return(value);
}

/*******************************************************/

int
parseck(argpointer,cp,n)
char *argpointer, *cp;
int n;
/* parseck - checks the remainder of the command word after the
     parser has given found the first letter to be valid.
*/
{
int value;

value = ERRORCODE;
while (*(++argpointer) == *cp) cp++;
if (isspace(*argpointer)) value = n;
return(value);
}
```

```
/*******************************************************
**                                                   **
**           Statistical Analysis Environment        **
**                 HELP COMMAND                      **
**                                                   **
**   File: sae.help.c                                **
**   Original:  7 April 1986 - K N Cole              **
**   Current:  20 October 1986 - K N Cole            **
**                                                   **
**   Contains Module:                                **
**      help - display HELP information              **
**                                                   **
*******************************************************/
#include <stdio.h>
#include <ctype.h>
#include "sae.h"

help(arg)
/* help - SAE system help command
   displays selected information from the Help directory
*/
char *arg;
{
char *c[80], *s, *ss;
int code;

printf("SAE Help command, Version 2.0\n\n");

if (*arg) {      /* append the proper file name to Help/help */
  s = arg;
  strcpy(c,"cat Help/help.");
  code = parser(&s);
  switch (code) {
    case COMPILER:       strcat(c,COMPILERHELP); break;
    case EDITOR:         strcat(c,EDITORHELP); break;
    case LRTCMD:         strcat(c,LRTHELP); break;
    case TABLECMD:       strcat(c,TABLEHELP); break;
    case EXIT:           strcat(c,EXITHELP); break;
    case HELP:           strcat(c,HELPHELP); break;
    default:
      ss = s;   /* otherwise just print the file given */
      while (*ss && !isspace(*ss)) ss++;
      if (*ss) *ss++ = NULL;
      strcpy(c,"cat Help/");
      strcat(c,s);
    }
  system(c);
  }
else {
  /* print the first line of each file named 'help.xxx' */
  printf("Command\t\tDescription\n-------\t\t-----------\n");
  system("head -1 Help/help.* | grep -v \"=>\" | cat -s");
  }
}
```

```
/*****************************************************
**                                                 **
**          Statistical Analysis Environment       **
**             EDITOR WINDOW COMMANDS              **
**                                                 **
**  File: window.h                                 **
**  Original: 29 April 1986 - K N Cole             **
**  Current:  8 September 1986 - K N Cole          **
**                                                 **
*****************************************************/

#define ewmove(y,x)      VOID(wmove(editor,y,x))
#define ewclrtoeol()     VOID(wclrtoeol(editor))
#define ewgetch()        VOID(wgetch(editor))
#define ewdeleteln()     VOID(wdeleteln(editor))
#define ewdelch()        VOID(wdelch(editor))
#define ewtouch()        VOID(touchwin(editor))
#define ewclear()        VOID(wclear(editor))
#define ewrefresh()      VOID(wrefresh(editor))
#define ewaddch(y,x,ch)
               VOID(wmove(editor,y,x)==ERR?ERR:waddch(editor,ch))
#define ewinsch(y,x,ch)
               VOID(wmove(editor,y,x)==ERR?ERR:winsch(editor,ch))
#define ewinsertln(y)
               VOID(wmove(editor,y,0)==ERR?ERR:winsertln(editor))
#define ewgetyx(y,x)     y = editor->_cury, x = editor->_curx
#define ewchar(y,x)      (editor->_y[y][x] & 0177)
#define ewcursor()
               VOID(wmove(editor,editor->_cury,editor->_curx))
```

```
/*********************************************************
**                                                      **
**          Statistical Analysis Environment            **
**                       EDITOR                         **
**                                                      **
**  File: sae.edit.c                                    **
**  Original:  7 April 1986 - K N Cole                  **
**  Current:  6 October 1986 - K N Cole                 **
**                                                      **
**  Contains Module:                                    **
**     edit - EDITOR main program                       **
**                                                      **
*********************************************************/
#include <stdio.h>
#include <ctype.h>
#include "sae.h"

extern void edappend(), edcommand(), edinsert(), eddelline(),
            edinsertln(), edreplacechar(), edredraw(),
            eddelchar(), edwrite(), edinit(), switchcflag(),
            switchhelp(), switchlang(), switchdisplay();
extern boolean cmdcmd();

boolean cflag;

edit(arg)
char *arg;
{
boolean done;
char *editfile;

/* Display Editor Signon Message and initialize windows */
printf("\nSAE Editor, Version 1.3\n\n");

/* Prepare the filename and initialize the editor */
editfile = arg;
cflag = FALSE;
if (*arg) {
  while (*arg && !isspace(*arg)) arg++;
  *arg = NULL;
  }

edinit(editfile);

/* Main program loop */
done = FALSE;
while (!done) {
  switch (edcommand()) {
    case 'a':                   /* append mode */
      edappend(cflag); break;
    case 'i':                   /* insert mode */
      edinsert(cflag); break;
    case 'd':                   /* delete line */
      eddelline(cflag); break;
    case 'C':                   /* toggle automatic compiling */
```

```
        switchcflag(&cflag); break;
      case 'H':                      /* toggle help display */
        switchhelp(); break;
      case 'L':                      /* toggle language display */
        switchlang(); break;
      case 'o':                      /* open a new line */
        edinsertln(); edinsert(cflag); break;
      case 'r':                      /* replace character */
        edreplacechar(cflag); break;
      case 'S':                      /* toggle statistics display */
        switchdisplay(); break;
      case 'v':                      /* redraw screen */
        edredraw(); break;
      case 'x':                      /* delete character */
        eddelchar(cflag); break;
      case 'Z':                      /* exit editor */
        edwrite(editfile); done = TRUE; break;
      case ':':                      /* process line commands */
        done = cmdcmd(); break;
      default:                       /* invalid command */
        break;
      }
    }
  edexit();
  }
```

```
/*******************************************************
**                                                   **
**          Statistical Analysis Environment         **
**                 EDITOR FUNCTIONS                  **
**                                                   **
**   File: ed.functions.c                            **
**   Original:  7 April 1986 - K N Cole              **
**   Current:  6 October 1986 - K N Cole             **
**                                                   **
**   Contains Modules:                               **
**      edinit                                       **
**      edexit                                       **
**      edappend                                     **
**      edinsert                                     **
**      edinsertln                                   **
**      eddelline                                    **
**      eddelchar                                    **
**      edreplacechar                                **
**      edwrite                                      **
**      edread                                       **
**      edredraw                                     **
**      edcommand                                    **
**                                                   **
*******************************************************/

#include <stdio.h>
#include <ctype.h>
#include <curses.h>
#include "sae.h"
#include "window.h"

extern void edread(), cmdinit(), initdisplay(), inithelp(),
        inittable(), motioninit(), cmdprint(), tableread(),
        tablewrite(), mvcursor(), tabnewline(), writetext(),
        clearcompile();
extern char *readtext(), motion(), getkey();
extern int length(), compileall();
extern boolean cflagtrue();

WINDOW *editor;
char templine[MAXTEMP];
```

```
/*************************************************************/

void
edinit(arg)
char *arg;
/* edinit - initializes the edit window as required by the 'curses'
   package.
*/
{
                        /* Initialize terminal window */
if (initscr() == ERR)
  printf("*** Error during screen initialization ***\n\n");
else {
  raw(); noecho(); nl();

                        /* initialize editor windows */
  if (!(editor = newwin(EDLINES,EDCOLS,0,0)))
    printf("*** Error during editor window initialization *** \n\n");
  else {
    ewclear();
    cmdinit(arg);
    initdisplay();
    inithelp();
    inittable();

    if (*arg) edread(arg,0);     /* read file into editor */
    else cmdprint("No Edit File");

    motioninit();        /* set original position of editor */
    }
  }
}

/*************************************************************/

void
edexit(cflag)
boolean cflag;
/* edexit - prepares edit window to exit the editor
*/
{
                /* clear bottom of screen and exit to shell */
move(0,16);
clrtobot();
refresh();
endwin();
}
```

```
/*********************************************************/

void
edappend(cflag)
boolean cflag;
/* edappend - append text to the current edit cursor location
*/
{
int ey,ex;
extern void edinsert();

ewgetyx(ey,ex);
if (length(ey)) mvcursor(ey,ex+1);
edinsert(cflag);
}

/*********************************************************/

void
edinsert(cflag)
boolean cflag;
/* edinsert - append text before the current edit cursor location
*/
{
char c;
char *s, *ss;
int ey,ex,tx,cx;

ewgetyx(ey,ex);
ss = readtext(ey);                /* pointer to current text line */
if (ss == NULL) {
  writetext(ey, NULL);
  ss = readtext(ey);
  }
strcpy(templine, ss);             /* copy current line to templine */
ss = ss + ex;                     /* pointer to point of insertion */
s = templine + ex;                /* pointer to point of insertion */
tx = 0;                           /* length set to current position */
while ((c = ewgetch()) != ESC) {
  switch (c) {
    case CR:
    case NL:
                *s++ = NL; *s = NULL;          /* terminate line */
                tabnewline(ey+1, ss);          /* new line */
                writetext(ey++, templine);     /* finished line */
                tx = 0; ex = 0;                /* reset pointers */
                ss = readtext(ey);
                strcpy(templine, ss);
                s = templine;
                if (cflag) compileline(ey-1);
                mvcursor(ey,ex);
                break;
    case TAB:
                if ((length(ey)+tx+TABSIZE) < MAXTEMP) {
                  for (cx = 1; cx <= TABSIZE; cx++) {
                    if (ex < MAXLINE) ewinsch(ey,ex,BLANK);
                    ex++; *s++ = BLANK;
                    }
                  tx = tx + TABSIZE;
                  ewmove(ey,ex);
                  }
                else ederror(1);
                break;
      case LEFT:
                if (tx > 0) {
                  ex--; ewmove(ey,ex);
                  ewdelch();
                  tx--; s--;
                  }
                else ederror(1);
                break;
      default:
                if ((length(ey)+tx+1) < MAXTEMP) {
                  if (ex < MAXLINE) ewinsch(ey,ex,c);
                  tx++; ex++; *s++ = c; ewmove(ey,ex);
                  }
                else ederror(1);
                break;
      }
     mvcursor(ey,ex);
    }
  *s = NULL;
  strcat(templine, ss);
  writetext(ey, templine);
  if (cflag && tx) compileline(ey);
  mvcursor(ey,ex + (tx ? -1 : 0));
  }
```

```
/*********************************************************/

void
edinsertln()
/* edinsertln - insert a line at the current editor cursor position.
*/
{
int ey, ex;

ewgetyx(ey, ex);
tabnewline(++ey, NULL);
mvcursor(ey,0);
}

/*********************************************************/

void
eddelline(cflag)
boolean cflag;
/* eddelline - delete the line at the current editor position.
*/
{
int ey, ex;

ewgetyx(ey, ex);
ewdeleteln();
tabdeleteline(ey);
if (cflag) compileall();
mvcursor(ey,0);
}

/*********************************************************/

void
eddelchar(cflag)
boolean cflag;
/* eddelchar - delete a character at the current editor
     cursor position.
*/
{
int ey,ex;
char *s, *st;

ewgetyx(ey,ex);
if (length(ey)) {
  ewdelch();
  s = (char *) ((int) readtext(ey) + ex)
  if (*s != NL) while (*s) {*s = *(s+1); s++;}
  else ederror(1);
  if (cflag) compileline(ey);
  mvcursor(ey,ex);
  }
else ederror(1);
}
```

```
/**************************************************************/

void
edreplacechar(cflag)
boolean cflag;
/* edreplacechar - replaces the character at the current
     editor cursor location.
*/
{
int ey,ex;
char *s, c;

ewgetyx(ey,ex);
s = readtext(ey);
if (*(s + ex) != NL) {
  ewaddch(ey,ex,c = ewgetch());
  *(s + ex) = c;
  }
else ederror(1);
if (cflag) compileline(ey);
mvcursor(ey,ex);
}
```

```
/*************************************************************/

void
edwrite(outfile)
char *outfile;
/* edwrite - opens the fle named by (outfile) and writes
   the contents of the edit buffer to the file.
   If the 'cflag' is true, the file is compiled.
*/
{
FILE *fp;
                        /* prepare command to make backup copy */
strcpy(templine,"mv ");
strcat(templine,outfile);
strcat(templine," ");
strcat(templine,outfile);
strcat(templine,".bak");
system(templine);       /* make backup file */

if (fp = fopen(outfile, "w")) {
  sprintf(templine,"Writing %s\r", outfile);
  tablewrite(fp);
  fclose(fp);
  if (cflagtrue())
    if (compileall() == 0) {
      setoutputon();
      compileall();
      setoutputoff();
      }
  }
else
  sprintf(templine,"Cannot open %s for output", outfile);
cmdprint(templine);
}
```

```
/************************************************************/

void
edread(infile,ey)
char *infile;
int ey;
/* edread - opens the file named by (infile) and reads
   the contents into the editor.
*/
{
FILE *fp;

if (fp = fopen(infile, "r"))
  {
  sprintf(templine, "Reading %s\r",infile);
  cmdprint(templine);
  tableread(fp,ey);
  fclose(fp);
  }
else sprintf(templine, "Cannot open %s", infile);
cmdprint(templine);
}

/************************************************************/

void
edredraw()
/* edredraw - redraws the edit window by forcing the screen to
   be cleared and then redrawn. *** NOT the same as 'ewrefresh()'.
*/
{
clearok(curscr,TRUE);
ewtouch();
ewrefresh();
}
```

```
/**********************************************************/

char
edcommand()
/* edcommand - returns a single character command code
*/
{
char c;

while (!(c = motion()));
switch (c) {
  case 'd':                         /* delete mode */
    switch (getkey()) {
      case 'd':                     /* delete line */
        c = 'd'; break;
      case 'c':                     /* delete character */
      case BLANK:
        c = 'x'; break;
      default:
        c = NULL; break;
      }
    break;

  case 'Z':                         /* save and exit */
    if (getkey() != 'Z') c = NULL;
    break;

  default:  break;                  /* single character command */
  }
return(c);
}
```

```
/*******************************************************
**                                                   **
**          Statistical Analysis Environment         **
**                      EDITOR                       **
**                                                   **
**   File: ed.cmd.c                                  **
**   Original:  7 April 1986 - K N Cole              **
**   Current:  28 April 1986 - K N Cole              **
**                                                   **
**   Contains Modules:                               **
**      cmdinit                                      **
**      cmdclear                                     **
**      cmdprint                                     **
**      cmdcmd                                       **
**                                                   **
*******************************************************/
#include <stdio.h>
#include <curses.h>
#include "sae.h"
#include "window.h"

extern WINDOW *editor;

WINDOW *command;
char sourcefile[MAXCOMMAND];

/**********************************************************/

void
cmdinit(arg)
char *arg;
/* cmdinit - initialize the command line window.
   Copy input filename to storage.
*/
{
if (arg) strncpy(sourcefile, arg, MAXCOMMAND-1);
else sourcefile[0] = NULL;

command = newwin(1,80,CMDLINE,0);
wclear(command);
}

/**********************************************************/

void
cmdclear()
/* cmdclear - clear the command line
*/
{
wmove(command,0,0);
wclrtoeol(command);
}

/**********************************************************
```

```
void
cmdprint(s)
char *s;
{
int ey,ex;

ewgetyx(ey,ex);
wmove(command,0,0);
wclrtoeol(command);
waddstr(command,s);
wrefresh(command);
mvcursor(ey,ex);
}
```

```
/*************************************************************/

boolean
cmdcmd()
/* cmdcmd - read and process commands.
*/
{
boolean exit;
char cmd, filename[MAXCOMMAND], *c;
int ey,ex;

exit = FALSE;
cmdprint(": ");
cmd = wgetch(command);
switch (cmd) {
  case 'q':                /* QUIT command */
    cmdprint(": quit");
    exit = TRUE;
    break;
  case 'r':                /* READ command */
    ewgetyx(ey,ex);
    cmdprint(": read    input file: ");
    c = filename;
    echo();
    while ((*c = wgetch(command)) != CR) c++; *c = NULL;
    noecho();
    if (filename[0]) edread(ey, filename);
    else cmdprint("Read what?");
    break;
  case 'e':                /* EXIT command */
    cmdprint(": exit    output file: ");
    exit = TRUE;
  case 'w':                /* WRITE command */
    if (!exit) cmdprint(": write   output file: ");
    c = filename;
    echo();
    while ((*c = wgetch(command)) != CR) c++; *c = NULL;
    noecho();
    if (filename[0])  edwrite(filename);
    else if (sourcefile[0]) edwrite(sourcefile);
    else cmdprint("Write what?");
    break;
  default:                 /* DEFAULT */
    cmdprint("What?");
    break;
  }
return(exit);
}
```

```
/*****************************************************************
**                                                              **
**            Statistical Analysis Environment                  **
**                       EDITOR HELP                            **
**                                                              **
**   File: ed.help.c                                            **
**   Original:  7 April 1986 - K N Cole                         **
**   Current:  24 April 1986 - K N Cole                         **
**                                                              **
**   Contains Modules:                                          **
**          inithelp                                            **
**          readhelp                                            **
**          switchhelp                                          **
**          switchlang                                          **
**          refreshhelp                                         **
**          ckhelp                                              **
**                                                              **
*****************************************************************/
#include <stdio.h>
#include <curses.h>
#include "sae.h"

extern void edredraw();
                        /* global window and flags */
WINDOW *helpwindow;
boolean helpon, langon, helpready, langready;

/*****************************************************************/

void
inithelp()
/* inithelp - initialize the help window and flags
*/
{
helpwindow = newwin(HELPLINES,HELPCOLS,HELPSTART,HELPINDT);
helpon = FALSE;
langon = FALSE;
helpready = FALSE;
langready = FALSE;
}
```

```
/*****************************************************/

void
readhelp()
/* readhelp - reads the help or language files into the window
*/
{
FILE *hp;
char templn[HELPCOLS];
int hy;

if (helpon) {
  if (!(hp = fopen(EDHELPFILE, "r")))
    cmdprint("*** ERROR ***  Editor Help File NOT Found");
  }
else if (langon)
  if (!(hp = fopen(EDLANGFILE, "r")))
    cmdprint("*** ERROR ***  Editor Language File NOT Found");
if (hp) {
  for (hy = 0; hy < HELPLINES; hy++) {
    fgets(templn,HELPCOLS-1,hp);
    wmove(helpwindow,hy,0);
    wprintw(helpwindow,templn);
    }
  fclose(hp);
  }
}

/*****************************************************/

void
switchhelp()
/* switchhelp - change the on/off state of the help display
*/
{
if (helpon) {
  helpon = FALSE;
  edredraw();
  }
else {
  helpon = TRUE;
  langon = FALSE;
  if (!helpready) {
    readhelp();
    helpready = TRUE;
    langready = FALSE;
    }
  if (ckdisplay()) switchdisplay();
  touchwin(helpwindow);
  wrefresh(helpwindow);
  }
}
```

```
/*****************************************************/

void
switchlang()
/* switchlang - turns OFF the helpdisplay and ON the language display
*/
{
if (langon) {
  langon = FALSE;
  edredraw();
  }
else {
  langon = TRUE;
  helpon = FALSE;
  if (!langready) {
    readhelp();
    langready = TRUE;
    helpready = FALSE;
    }
  if (ckdisplay()) switchdisplay();
  touchwin(helpwindow);
  wrefresh(helpwindow);
  }
}

/*****************************************************/

void
refreshhelp()
/* refreshhelp - refresh the help window if necessary
*/
{
if (helpon || langon) {
  touchwin(helpwindow);
  wrefresh(helpwindow);
  }
}

/*****************************************************/

boolean
ckhelp()
/* ckhelp - show the condition of the flag 'helpon'
*/
{
return(helpon);
}
```

```
/*******************************************************
**                                                    **
**          Statistical Analysis Environment          **
**                     EDITOR DISPLAY                 **
**                                                    **
**   File: ed.disp.c                                  **
**   Original:  7 April 1986 - K N Cole               **
**   Current:  24 October 1986 - K N Cole             **
**                                                    **
**   Contains Modules:                                **
**         initdisplay                                **
**         switchdisplay                              **
**         refreshdisp                                **
**         updatedisplay                              **
**         ckdisplay                                  **
**         switchcflag                                **
**         cflagtrue                                  **
**                                                    **
*******************************************************/
#include <stdio.h>
#include <curses.h>
#include "sae.h"

extern void clearcompile(), switchhelp(),
            setoutputoff();
extern boolean ckhelp();
extern int length(), lastline(), statcode(), compileall();
extern char *errtext();

WINDOW *display;
bool displayon, cpon;

/******************************************************/

void
initdisplay()
{
display = newwin(DISPLINES,DISPCOLS,DISPSTART,DISPINDT);
displayon = FALSE;
cpon = FALSE;
}
```

```
/*********************************************************/

void
switchdisplay()
{
if (displayon) {
  displayon = FALSE;
  wclear(display);
  wrefresh(display);
  }
else {
  displayon = TRUE;
  if (ckhelp()) switchhelp();
  touchwin(display);
  wrefresh(display);
  }
}

/*********************************************************/

void
refreshdisp()
/* refreshdisp - refresh the display window... if 'displayon'
*/
{
if (displayon) {
  touchwin(display);
  wrefresh(display);
  }
}

/*********************************************************/

void
updatedisplay(y,x)
int y,x;
{
wmove(display,1,2);
wprintw(display,"Line %2d   Column %2d  ", y, x);
wmove(display,1,50);
if (cpon) wprintw(display,"Compiler ON ");
else wprintw(display,"Compiler OFF");
wmove(display,3,2);
wprintw(display,"Line length: %2d     Last line: %2d",
                                        length(y-1), lastline());
wmove(display,5,2);
if (cpon) wprintw(display,"Compiler Message: %s",
                                        errtext(statcode(y-1)));
else wclrtoeol(display);
refreshdisp();
}
```

```
/*******************************************************/

boolean
ckdisplay()
/* ckdisplay - return the current value of the 'displayon' flag
*/
{
return(displayon);
}

/*******************************************************/

void
switchcflag(cflag)
int *cflag;
/* switchcflag - change the mode of editor operation
   to (or from) using the compiler.
*/
{

if (*cflag) {
  *cflag = FALSE;
  cpon = FALSE;
  wmove(display,1,50);
  wprintw(display,"Compiler OFF");
  clearcompile();
  }
else {
  *cflag = TRUE;
  cpon = TRUE;
  setoutputoff();
  wmove(display,1,50);
  wprintw(display,"Compiler ON ");
  compileall();
  }
}

/**********************************************************/

boolean
cflagtrue()
/* cflagtrue - returns value of 'cpon' flag
*/
{
return(cpon);
}
```

```
/*******************************************************
**                                                   **
**          Statistical Analysis Environment         **
**                       EDITOR                      **
**                                                   **
**  File: ed.misc.c                                  **
**  Original:  7 April 1986 - K N Cole               **
**  Current:  24 October 1986 - K N Cole             **
**                                                   **
**  Contains Modules:                                **
**     mvcursor                                      **
**     ederror                                       **
**     getkey                                        **
**     motion                                        **
**     motioninit                                    **
**                                                   **
*******************************************************/
#include <stdio.h>
#include <curses.h>
#include "sae.h"
#include "window.h"

extern void cmdprint(), cmdclear();

extern WINDOW *editor;

int xpos;                  /* horizontal position storage */

/***********************************************************/

void
mvcursor(ey,ex)
int ey,ex;
/* mvcursor - moves the cursor by the specified location in the
   edit window.
*/
{
updatedisplay(ey+1,ex+1);
ewmove(ey,ex);
ewtouch();
ewrefresh();
}
```

```
/***********************************************************/

void
ederror(n)
int n;
/* ederror - editor error routine.
*/
{
int ey,ex;

ewgetyx(ey,ex);
switch (n) {
  case 1:
    cmdprint("\007");           /* just ring bell */
    break;

  default:
    cmdprint("Editor Error... Incorrect Command.\007");
    break;
  }
mvcursor(ey,ex);
}
```

```
/************************************************************/

char
getkey()
/* getkey - accepts command strings from the edit window and
   translates the "termcap" strings for arrow keys.
*/
{
char c;
char input[5], cap[5], *temp;
int leng;

if (((c = ewgetch()) < ' ') || (c > '~')) {
  if (c == ESC) {
    input[0] = c;
    tgetstr("kd",cap);
    leng = strlen(cap) - 1;
    temp = &input[1];
    while (leng > 0 ) {*temp++ = ewgetch(); leng--;}
    *temp = NULL;
    if (strcmp(cap,input) == 0) c = DOWN;
    else {
      tgetstr("ku",cap);
      if (strcmp(cap,input) == 0) c = UPLN;
      else {
        tgetstr("kd",cap);
        if (strcmp(cap,input) == 0) c = LEFT;
        else {
          tgetstr("kr",cap);
          if (strcmp(cap,input) == 0) c = RIGHT;
          else c = NULL;
          }
        }
      }
    }
  }
return(c);
}
```

```
/*************************************************************/

char
motion()
/* motion - moves the cursor in the edit window in response to
   valid motion commands.
*/
{
bool ismotion, error;
char c;
int ey,ex;

ismotion = TRUE; error = FALSE;
ewgetyx(ey,ex);
cmdclear();
        /* assume we are in a valid position */
        /* check new position only */
mvcursor(ey,ex);
switch (c = getkey()) {
  case UPLN:     if (!(error = !cklocy(ey-1))) {
                   ey--;
                   ex = xpos;
                   if (ex < length(ey)) ex = length(ey);
                   }
                 break;
  case DOWN:     if (!(error = !cklocy(ey+1))) {
                   ey++;
                   ex = xpos;
                   if (ex < length(ey)) ex = length(ey);
                   }
                 break;
  case BLANK:
  case RIGHT:
  case TAB:      if (!(error = !cklocx(ey,ex+1)))
                 xpos = ++ex;
                 break;
  case LEFT:     if (!(error = !cklocx(ey,ex-1)))
                 xpos = --ex;
                 break;
  case CR:       if (!(error = !cklocy(ey+1)))
                 ey++;
                 xpos = ex = 0;
                 break;
  default:       ismotion = FALSE; xpos = ex; break;
  }

if (error) ederror(1);
if (ismotion || error) c = NULL;
else xpos = ex;
mvcursor(ey,ex);
return(c);
}
```

```
/***********************************************************/

void
motioninit()
/* motioninit - initializes the editor motion position at the
   time of entry to the editor.
*/
{
xpos = 0;
mvcursor(0,0);
}
```

```
/*********************************************************
**                                                     **
**          Statistical Analysis Environment           **
**                       EDITOR                        **
**                                                     **
**   File: ed.motion.c                                 **
**   Original:  7 April 1986 - K N Cole                **
**   Current:  15 September 1986 - K N Cole            **
**                                                     **
**   Contains Modules:                                 **
**      motion                                         **
**      getkey                                         **
**      motioninit                                     **
**      mvcursor                                       **
**                                                     **
*********************************************************/

#include <stdio.h>
#include <curses.h>
#include "sae.h"
#include "window.h"

extern WINDOW *editor;

int xpos;                /* horizontal position storage */

/************************************************************/

void
mvcursor(ey,ex)
int ey,ex;
/* mvcursor - moves the cursor by the specified location in the
   edit window.
*/
{
updatedisplay(ey+1,ex+1);
ewmove(ey,ex);
ewtouch();
ewrefresh();
}
```

```
/**************************************************************/

char
getkey()
/* getkey - accepts command strings from the edit window and
   translates the "termcap" strings for arrow keys.
*/
{
char c;
char input[5], cap[5], *temp;
int leng;

if (((c = ewgetch()) < ' ') || (c > '~')) {
  if (c == ESC) {
    input[0] = c;
    tgetstr("kd",cap);
    leng = strlen(cap) - 1;
    temp = &input[1];
    while (leng > 0 ) {*temp++ = ewgetch(); leng--;}
    *temp = NULL;
    if (strcmp(cap,input) == 0) c = DOWN;
    else {
      tgetstr("ku",cap);
      if (strcmp(cap,input) == 0) c = UPLN;
      else {
        tgetstr("kd",cap);
        if (strcmp(cap,input) == 0) c = LEFT;
        else {
          tgetstr("kr",cap);
          if (strcmp(cap,input) == 0) c = RIGHT;
          else c = NULL;
          }
        }
      }
    }
  }
return(c);
}
```

```
/*************************************************************/

char
motion()
/* motion - moves the cursor in the edit window in response to
   valid motion commands.
*/
{
bool ismotion, error;
char c;
int ey,ex;

ismotion = TRUE; error = FALSE;
ewgetyx(ey,ex);
        /* assume we are in a valid position */
        /* check new position only */
mvcursor(ey,ex);
switch (c = getkey()) {
  case UPLN:     if (!(error = !cklocy(ey-1))) {
                   ey--;
                   ex = xpos;
                   if (ex < length(ey)) ex = length(ey);
                   }
                 break;
  case DOWN:     if (!(error = !cklocy(ey+1))) {
                   ey++;
                   ex = xpos;
                   if (ex < length(ey)) ex = length(ey);
                   }
                 break;
  case BLANK:
  case RIGHT:
  case TAB:      if (!(error = !cklocx(ey,ex+1)))
                 xpos = ++ex;
                 break;
  case LEFT:     if (!(error = !cklocx(ey,ex-1)))
                 xpos = --ex;
                 break;
  case CR:       if (!(error = !cklocy(ey+1)))
                 ey++;
                 xpos = ex = 0;
                 break;
  default:       ismotion = FALSE; xpos = ex; break;
  }

if (error) ederror(1);
if (ismotion || error) c = NULL;
else xpos = ex;
mvcursor(ey,ex);
return(c);
}
```

```
/**************************************************************/

void
motioninit()
/* motioninit - initializes the editor motion position at the
   time of entry to the editor.
*/
{
xpos = 0;
mvcursor(0,0);
}
```

```
/***************************************************************
**                                                          **
**           Statistical Analysis Environment               **
**                        EDITOR                            **
**                                                          **
**   File: ed.table.c                                       **
**   Original: 29 April 1986 - K N Cole                     **
**   Current: 17 September 1986 - K N Cole                  **
**                                                          **
**   Contains Modules:                                      **
**      cklocy                                              **
**      cklocx                                              **
**      length                                              **
**      tabnewline                                          **
**      tabdeleteline                                       **
**      lastline                                            **
**      inittable                                           **
**      writetext                                           **
**      readtext                                            **
**      tableread                                           **
**      tablewrite                                          **
**      clearcompile                                        **
**      compileall                                          **
**      compileline                                         **
**      statcode                                            **
**                                                          **
***************************************************************/

#include <stdio.h>
#include <curses.h>
#include "sae.h"
#include "status.h"
#include "window.h"

extern WINDOW *editor;
extern void cpreset(), mvcursor(), ederror();
extern int cpline();

typedef struct lineref
  {
  char *text;              /* pointer to text of the line */
  int length;              /* length of line */
  int status;              /* compiler status of the line */
  int cstate;              /* state of the compiler after checking */
  } linetype;

linetype tablestore[EDLINES];
linetype *edtable[EDLINES];
int lastln, errct;
char temptext[MAXLINE+1];
```

```
/*****************************************************************/

boolean
cklocy(ey)
int ey;
/* cklocy - verifies the given line location (ey) in the
   edit window.
*/
{
boolean valid;

valid = ((ey >= 0) && (ey < lastln));
return(valid);
}

/*****************************************************************/

boolean
cklocx(ey,ex)
int ey,ex;
/* cklocx - verifies the given column location (ex) on the
   line (ey) in the edit window.
*/
{
boolean valid;

valid = cklocy(ey);
if (valid)
  valid = ((ex >= 0) && ((ex+1) <= edtable[ey]->length));
return(valid);
}

/*****************************************************************/

int
length(ey)
int ey;
/* length - returns the current length of line 'ey'.
*/
{
return(edtable[ey]->length);
}
```

```
/***********************************************************/

void
tabnewline(ey,s)
int ey;
char *s;
/* tabnewline - creates a new line in the table at the screen
     location given by (ey).  The text is taken from the string
     pointed to by (s).
*/
{
int ct;
linetype *py;
extern void writetext();

if ((lastln < EDLINES) && (ey <= lastln)) {
  if (ey < lastln) {
    py = edtable[lastln];
    for (ct = lastln; ct > ey; ct--)
      edtable[ct] = edtable[ct-1];
    edtable[ey] = py;
    ewinsertln(ey);
    }
  writetext(ey,s);
  lastln = lastln + 1;
  }
}

/***********************************************************/

void
tabdeleteline(ey)
int ey;
/* tabdeleteline - deletes the line at location (ey) from
   the file.
*/
{
int ct;
linetype *py;

if ((ey >= 0) && (ey < lastln)) {
  py = edtable[ey];
  for (ct = ey; ct < lastln-1; ct++) {
    edtable[ct] = edtable[ct+1];
    }
  edtable[lastln-1] = py;

  cfree(py->text);
  py->text = NULL;
  py->cstate = 0;
  py->length = 0;
  py->status = NONE;
  lastln = lastln-1;
  }
}
```

```
/***********************************************************/

int
lastline()
{
return(lastln);
}

/***********************************************************/

void
inittable()
/* inittable - initializes the indicators of the amount of text
   in the window.
*/
{
int ey;

lastln = 0;
for (ey = 0; ey < EDLINES; ey++) {
  ewaddch(ey,0,NOLINE);
  edtable[ey] = &tablestore[ey];
  edtable[ey]->text = NULL;
  edtable[ey]->cstate = 0;
  edtable[ey]->length = 0;
  edtable[ey]->status = NONE;
  }
}
```

```
/**********************************************************/

void
writetext(ey,s)
int ey;
char *s;
/* writetext - replaces the current line (ey) text with the new
   string (s).
*/
{
linetype *py;
boolean space;

if ((ey >= 0) && (ey <= lastln)) {
  py = edtable[ey];
  if (py->text) space = TRUE;
  else {
    if (py->text = (char *) calloc(MAXLINE,sizeof(char)))
       space = TRUE;
    else space = FALSE;
    }
  ewmove(ey,0);
  ewclrtoeol();
  if (space) {
    if (s) {
      strncpy(py->text,s,MAXLINE);
      py->length = strlen(py->text);
      wprintw(editor, py->text);
      }
    else {
      py->text[0] = NULL;
      py->length = 0;
      }
    }
  else printf("*** Unable to allocate memory for 'text' ***");
  }
}

/**********************************************************/

char *
readtext(ey)
int ey;
/* readtext - returns a pointer to the text referenced by
   the screen position line (ey).
*/
{
char *s;

if ((ey >= 0) && (ey < lastln)) s = edtable[ey]->text;
else s = NULL;

return(s);
}
```

```
/*********************************************************/

void
tableread(fp, ey)
FILE *fp;
int ey;
/* tableread - read file into the editor starting at
   screen line (ey).
*/
{
if ((ey >= 0) && (ey <= lastln)) {
  while (fgets(temptext, MAXLINE, fp) && (lastln < EDLINES))
    tabnewline(ey++, temptext);
  }
}

/*********************************************************/

void
tablewrite(fp)
FILE *fp;
/* tablewrite - writes the contents of the edit table out
   to the file referenced by (fp).
*/
{
int ey;

for (ey = 0; ey < lastln; ey++) {
  fputs(edtable[ey]->text, fp);
  }
}

/*********************************************************/

void
clearcompile()
/* clearcompile - clears the compiler status characters
     from the screen.
*/
{
int ey;

if (lastln)
  for (ey = 0; ey < lastln; ey++) {
    ewaddch(ey,STATCOL,BLANK);
    edtable[ey]->status = NONE;
    }
}
```

```
/************************************************************/

int
compileall()
/* compileall - compile all the lines currently in the editor
*/
{
int ey, stat, state, sy, sx;

ewgetyx(sy,sx);             /* save cursor location */
errct = 0; state = 0;
cpreset();
if (lastln)
  for (ey = 0; ey < lastln; ey++) {
    stat = cpline(edtable[ey]->text, &state);
    if (stat != VALID) errct++;
    edtable[ey]->status = stat;
    edtable[ey]->cstate = state;
    switch (stat) {
      case VALID:         ewaddch(ey,STATCOL,VALIDSTAT); break;
      case PARTVLD:       ewaddch(ey,STATCOL,PARTSTAT); break;
      case UNCKD:
      case NONE:          ewaddch(ey,STATCOL,BLANK); break;
      default:            ewaddch(ey,STATCOL,ERRSTAT);
    } }
mvcursor(sy,sx);            /* reset cursor */
return(errct);
}

/************************************************************/

void
compileline(ey)
int ey;
{
int ty, stat, state;

if (ey > 0) {
  ty = ey-1;
  while ((ty >= 0) && (edtable[ty]->status != VALID)) ty--;
  ty = ty +1;
  }
else ty = ey;

for (; ty <= ey; ty++) {
  stat = cpline(edtable[ty]->text, &state);
  edtable[ty]->status = stat;
  edtable[ty]->cstate = state;
  switch (stat) {
    case VALID:           ewaddch(ty,STATCOL,VALIDSTAT); break;
    case PARTVLD:         ewaddch(ty,STATCOL,PARTSTAT); break;
    case UNCKD:
    case NONE:            ewaddch(ty,STATCOL,BLANK); break;
    default:              ewaddch(ty,STATCOL,ERRSTAT);
} } }
```

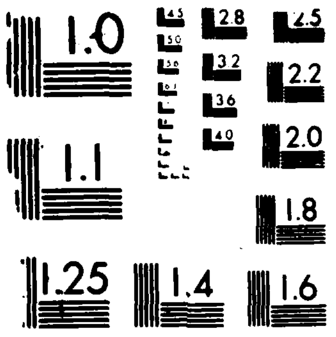
```
/***********************************************************/

int
statcode(ey)
int ey;
/* statcode - returns the status code for the line (ey).
*/
{
int stat;

if ((ey >= 0) && (ey < lastln)) stat = edtable[ey]->status;
else {
  stat = NONE;
  ederror(1);
  }
return(stat);
}
```

```
/*****************************************************
**                                                  **
**          Statistical Analysis Environment        **
**                   PARSER STATUS                  **
**                                                  **
**   File: status.h                                 **
**   Original: 29 April 1986 - K N Cole             **
**   Current:  15 September 1986 - K N Cole         **
**                                                  **
*****************************************************/

#define NONE         0       /* no status                         */
#define UNCKD        1       /* unchecked line                    */
#define VALID        2       /* valid line                        */
#define PARTVLD      3       /* valid incomplete line             */
#define ERROR        4       /* non-specific error                */
#define INVLDCHAR    5       /* invalid character                 */
#define EXFE         6       /* expected filename or ENVIRONMENT*/
#define EXSL         7       /* expected semicolon or left paren*/
#define EXCR         8       /* expected comma or right paren     */
#define EXS          9       /* expected semicolon                */
#define EXA          10      /* expected argument                 */
#define EXE          11      /* expected equal sign               */
#define EXI          12      /* expected integer                  */
#define EXTL         13      /* expected TABLE or LPT             */
#define EXF          14      /* expected fraction                 */
#define EXEA         15      /* expected EXACT or ASYMTOTIC       */
#define EXIR         16      /* expected integer or RANGE         */
#define EXL          17      /* expected left paren               */
#define EXR          18      /* expected right paren              */
#define EXC          19      /* expected comma                    */
#define INVLDNUM     20      /* invalid number                    */
#define INVLDFN      21      /* invalid file name                 */
#define INVLDWD      22      /* invalid word in text              */

#define MAXERROR     23      /* size of error tabble              */

#define STATCOL      78      /* column to display status          */

#define VALIDSTAT    '.'     /* indicates a valid statement       */
#define PARTSTAT     '-'     /* indicates a partial statement     */
#define ERRSTAT      'E'     /* indicates an error                */
```

AD-A182 606 RELIABILITY ANALYSIS BASED ON EXPONENTIAL FAILURE DISTRIBUTIONS AND DEVEL (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI K N COLE 3/3

UNCLASSIFIED 19 DEC 86 AFIT/DS/ENG/86-06 F/G 12/3 NL

END
8-87
DTIC



MICROCOPY RESOLUTION TEST CHART

NA[illegible] [illegible] STANDARDS [illegible] A

```
/*****************************************************
**                                                 **
**          Statistical Analysis Environment       **
**               TOKEN  DEFINITIONS                **
**                                                 **
**   File: tokens.h                                **
**   Original: 29 April 1986 - K N Cole            **
**   Current:  6 October 1986 - K N Cole           **
**                                                 **
*****************************************************/

                                        /* Tokens */
#define MAXTOKEN        9

#define EOL             0       /* end of line or comment */
#define FRACTION        1       /* fraction value (real) */
#define INTEGER         2       /* integer value */
#define FILENAME        3       /* program file name */
#define COMMA           4       /* , */
#define SEMICOL         5       /* ; */
#define EQUALS          6       /* = */
#define LPAREN          7       /* ( */
#define RPAREN          8       /* ) */
#define RESVD           9       /* reserved word */


                                /* Reserved Words       */
#define MAXRESVD        10

#define ALPHA           0
#define ASYMTOTIC       1
#define ENVIRONMENT     2
#define EXACT           3
#define FAILURES        4
#define LPT             5
#define MAXTERMS        6
#define METHOD          7
#define SAMPLES         8
#define TABLE           9
#define RANGE           10

#define LANGSIZE        2 * (MAXRESVD+1)
```

```
/**********************************************************
**                                                      **
**           Statistical Analysis Environment           **
**                   SAE COMPILER                       **
**                                                      **
**   File:  sae.compile.c                               **
**   Original:  7 April 1986 - K N Cole                 **
**   Current:  9 October 1986 - K N Cole                **
**                                                      **
**   Contains Modules:                                  **
**      cpline                                          **
**      compile                                         **
**                                                      **
**********************************************************/
#include <stdio.h>
#include <ctype.h>
#include "sae.h"
#include "status.h"
#include "tokens.h"

extern void cpreset(), setoutputon(), tokenreset();
extern int cparse(), token();
extern char *errtext();

char sline[MAXLINE];

/**********************************************************/

int
cpline(s,state)
char *s;
int *state;
/* cpline - Compile a line (s).
   Returns a status indicator from the list in (status.h).
   Modifies the state variable to show the new state of the compiler.
*/
{
unsigned status;
int t, v;
char *sptr;

status = VALID;
sptr = s;
while ((status == VALID) && (t = token(&sptr,&v))) {
  if (t < 0) status = -t;
  else status = cparse(state, t, v);
  }
if ((status == VALID) && *state) status = PARTVLD;
else if (status != VALID) *state = 0;

return(status);
}
```

```
/****************************************************************************/

void
compile(arg)
char *arg;
{
FILE *fp;
int errct, lnct, state;
unsigned ecode;
char *c;

printf("SAE Compiler, Version 0.2\n\n");

errct = 0;
lnct = 0;
state = 0;

/* prepare the filename */
c = arg;
while (*c && !isspace(*c)) c++;
*c = NULL;                            /* place a null after the filename */

if (*arg) {
  if (fp = fopen(arg, "r")) {              /* open the file */
    printf("\nCompiling %s:\n",arg);
    cpreset(); tokenreset();
    while (fgets(sline, MAXLINE, fp)) {
                                /* compile each line as it is read */
      lnct++;
      ecode = cpline(sline,&state);
      if ((ecode == VALID) || (ecode == PARTVLD)) {
        printf("%d  %s",lnct, sline);
        }
      else {
        printf("%d  %s  ** %s\n", lnct, sline, errtext(ecode));
        errct++;
        }
      }
    printf("\nSummary:  %d lines   %d error\n", lnct, errct);
    if ((errct == 0) && (state == 0)) {
      rewind(fp);
      cpreset(); tokenreset();
      setoutputon();
      state = 0;                           /* re-compile with output */
      while (fgets(sline, MAXLINE, fp))
        ecode = cpline(sline,&state);
      }
    fclose(fp);
    }
  else  printf("Compiler:  Cannot open input file.\n");
  }
else printf("Compiler:  No input file.\n");
}
```

```
/*******************************************************
**                                                    **
**          Statistical Analysis Environment          **
**                   SAE COMPILER                     **
**                                                    **
**  File:  cp.compile.c                               **
**  Original:  7 April 1986 - K N Cole                **
**  Current:  9 October 1986 - K N Cole               **
**                                                    **
**  Contains Modules:                                 **
**     cparse                                         **
**     setoutputoff                                   **
**     setoutputon                                    **
**     cpreset                                        **
**                                                    **
*******************************************************/

#include <stdio.h>
#include <ctype.h>
#include "sae.h"
#include "status.h"
#include "tokens.h"

extern int intsymbol();
extern float floatsymbol();
extern char *wordsymbol();

                        /* globals for info storage */
boolean setdefenv = FALSE,
        outputon = FALSE,
        envflag, sflag;
char *filename, cmdline[80];
int stmttype;

        /* environment values */
int mxterms, method;

        /* table and lrt values */
int alphact;
int tsamples, tmxsamples, tincsamples,
    tfail, tmxfail, tincfail;
float talpha[3];
```

```
/*******************************************************/

int
cparse(state, token, v)
int *state, token, v;
/* cparse - determine the next state of the compiler given
   the input (state) and the next (token).
   Returns a status value (see status.h).
*/
{
int status, i;
FILE *fp;

status = VALID;
switch (*state) {
  case 0:  switch (token) {
             case RESVD:  if (v == ENVIRONMENT) {
                                if (envflag) status = EXTL;
                                else {
                                  stmttype = 0;
                                  *state = 1;
                                  }
                                }
                          else status = EXFE; break;
             case FILENAME:  *state = 3;
                 if (!setdefenv) {
                   mxterms = DEFMXTERMS; method = DEFMETHOD;
                   }
                 filename = wordsymbol(v);
                 break;
             default:  status = EXFE;
             } break;
  case 1:  switch(token) {
             case LPAREN:  *state = 2; break;
             case SEMICOL:  *state = 0;
                 setdefenv = TRUE;
                 mxterms = DEFMXTERMS; method = DEFMETHOD;
                 envflag = TRUE;
                 if (outputon) {
                   if (fp = fopen(CPENVFILE, "w")) {
                     fprintf(fp,"#define MAXTERMS\t%d\n",mxterms);
                     if (method == 1)
                              fprintf(fp,"#define EXACT\t\t1\n");
                     fclose(fp);
                     }
                   }
                 break;
             default:  status = EXSL;
             } break;
  case 2:  if (token == RESVD)
             switch (v) {
               case METHOD:  *state = 5; break;
               case MAXTERMS:  *state = 4; break;
               default:  status = EXA;
               }
```

```
                else status = EXA;
                break;
        case 3:
        case 4:
        case 5: if (token == EQUALS) *state = *state + 3;
                else status = EXE;
                break;
        case 6: if (token == RESVD)
                  switch (v) {
                    case TABLE:  *state = 9; stmttype = 1;
                      tsamples = DEFSAMPLES;
                      tmxsamples = tsamples;
                      tincsamples = 1;
                      tfail = tsamples + 1;
                      tmxfail = tfail;
                      tincfail = 1;
                      talpha[0] = DEFALPHA;
                      alphact = 1;
                      break;
                    case LRT:  *state = 10; stmttype = 2;
                      tsamples = DEFSAMPLES;
                      alphact = 0;
                      break;
                    default:  status = EXTL;
                    }
                else status = EXTL;
                break;
        case 7: if (token == INTEGER) {
                  mxterms = intsymbol(v);
                  *state = 11;
                  }
                else status = EXI;
                break;
        case 8: if (token == RESVD) {
                  *state = 11;
                  if (v == EXACT) method = 1;
                  else if (v == ASYMTOTIC) method = 0;
                  else status = EXEA;
                  }
                else status = EXEA;
                break;
        case 9: switch(token) {
                  case LPAREN:  *state = 12; break;
                  case SEMICOL:  *state = 0;
                      if (outputon) {
                        if (!envflag) {
                          envflag = TRUE;
                          system("cp Math/env.sav Math/env.h");
                          }
                        if (fp = fopen(CPTABLEFILE, "w")) {
                          fprintf(fp,"#define PINIT\t%d\n",tsamples);
                          fprintf(fp,"#define PMAX\t%d\n",tsamples);
                          fprintf(fp,"#define PINC\t1\n");
                          fprintf(fp,"#define RINIT\t%d\n",tfail);
                          fprintf(fp,"#define RMAX\t%d\n",tfail);
                    fprintf(fp,"#define RINC\t1\n");
                    fprintf(fp,"#define ALPHA1\t%4.3f\n",talpha[0]);
                    fprintf(fp,"#define MAXALPHA\t1\n");
                    fclose(fp);
                    }
                  strcpy(cmdline,"make -f math.make -s table;
                                                mv Math/table ");
                  strcat(cmdline,filename);
                  if (fork() == 0) {
                       system(cmdline);
                       exit();
                       }
                  }
                break;
            default:  status = EXSL;
            } break;
case 10:   if (token == LPAREN) *state = 13;
           else if (token == SEMICOL) {
             *state = 0;
             if (outputon) {
               if (!envflag) {
                 envflag = TRUE;
                 system("cp Math/env.sav Math/env.h");
                 }
               if (fp = fopen(CPLRTFILE, "w")) {
                 fprintf(fp,"#define MAXSAMPLES\t%d\n",tsamples);
                 fclose(fp);
                 }
               strcpy(cmdline,"make -f math.make -s lrt;
                                                mv Math/lrt ");
               strcat(cmdline,filename);
               if (fork() == 0) {
                       system(cmdline);
                       exit();
                       }
               }
             }
           else status = EXSL;
           break;
case 11:   if (token == COMMA) *state = 2;
           else if (token == RPAREN) *state = 26;
           else status = EXCR;
           break;
case 12:   if (token == RESVD)
             switch (v) {
               case FAILURES:  sflag = FALSE; *state = 14; break;
               case SAMPLES:  sflag = TRUE; *state = 14; break;
               case ALPHA:  *state = 15; break;
               default:  status = EXA;
               }
           else status = EXA;
           break;
case 13:   if (token == RESVD)
             switch (v) {
               case SAMPLES:  *state = 16; break;
```

```
                        case ALPHA:  *state = 17; break;
                        default:  status = EXA;
                        }
                    else status = EXA;
                    break;
          case 14:
          case 15:
          case 16:
          case 17:  if (token == EQUALS) *state = *state + 4;
                    else status = EXE;
                    break;
          case 18:  if (token == INTEGER) {
                      if (sflag) {
                        tsamples = intsymbol(v);
                        tmxsamples = tsamples;
                        tincsamples = 1;
                        }
                      else {
                        tfail = intsymbol(v);
                        tmxfail = tfail;
                        tincfail = 1;
                        }
                      *state = 22;
                      }
                    else if ((token == RESVD) && (v == RANGE)) *state = 23;
                    else status = EXIR;
                    break;
          case 19:  if (token == FRACTION) {
                      talpha[0] = floatsymbol(v);
                      alphact = 1;
                      *state = 22;
                      }
                    else if (token == LPAREN) *state = 24;
                    else status = EXF;
                    break;
          case 20:  if (token == INTEGER) {
                      tsamples = intsymbol(v);
                      *state = 25;
                      }
                    else status = EXI;
                    break;
          case 21:  if (token == FRACTION) {
                      alphact = 1;
                      talpha[0] = floatsymbol(v);
                      *state = 25;
                      }
                    else status = EXF;
                    break;
          case 22:  if (token == COMMA) *state = 12;
                    else if (token == RPAREN) *state = 26;
                    else status = EXCR;
                    break;
          case 23:  if (token == LPAREN) *state = 27;
                    else status = EXL;
                    break;
```

```
case 24:  if (token == FRACTION) {
            talpha[0] = floatsymbol(v);
            alphact = 1;
            *state = 28;
            }
          else status = EXF;
          break;
case 25:  if (token == COMMA) *state = 13;
          else if (token == RPAREN) *state = 26;
          else status = EXCR;
          break;
case 26:  if (token == SEMICOL) {
            *state = 0;
            if (stmttype == 0) {      /* environment */
                envflag = TRUE;
                if (outputon) {
                  if (fp = fopen(CPENVFILE, "w")) {
                      fprintf(fp,"#define NUM\t%d\n",mxterms);
                      if (method == 1)
                        fprintf(fp,"#define EXACT\t1\n");
                      fclose(fp);
                      }
                    }
                }
            else if (stmttype == 1) {          /* table program */
                if (outputon) {
                  if (!envflag) {
                    envflag = TRUE;
                    system("cp Math/env.sav Math/env.h");
                    }
                  if (fp = fopen(CPTABLEFILE, "w")) {
                    fprintf(fp,"#define PINIT\t%d\n",tsamples);
                    fprintf(fp,"#define PMAX\t%d\n",tmxsamples);
                    fprintf(fp,"#define PINC\t%d\n",tincsamples);
                    fprintf(fp,"#define RINIT\t%d\n",tfail);
                    fprintf(fp,"#define RMAX\t%d\n",tmxfail);
                    fprintf(fp,"#define RINC\t%d\n",tincfail);
                    for (i = 0; i < alphact; i++)
                      fprintf(fp,"#define ALPHA%1d\t%4.3f\n",
                                                 i+1,talpha[i]);
                    fprintf(fp,"#define MAXALPHA\t%d\n",alphact);
                    fclose(fp);
                    }
                  strcpy(cmdline,"make -f math.make -s table;
                                                 mv Math/table ");
                  strcat(cmdline,filename);
                  if (fork() == 0) {
                      system(cmdline);
                      exit();
                      }
                  }
                }
            else if (stmttype == 2) {          /* lrt program */
              if (outputon) {
                if (!envflag) {
                    envflag = TRUE;
                    system("cp Math/env.sav Math/env.h");
                    }
                  if (fp = fopen(CPLRTFILE, "w")) {
                    fprintf(fp,"#define MAXSAMPLES\t%d\n",tsamples);
                    if (alphact == 1)
                      fprintf(fp,"#define ALPHA\t%4.3f\n",talpha[0]);
                    fclose(fp);
                    }
                  strcpy(cmdline,"make -f math.make -s lrt;
                                                  mv Math/lrt ");
                  strcat(cmdline,filename);
                  if (fork() == 0) {
                        system(cmdline);
                        exit();
                        }
                  }
                }
              else status = ERROR;
              }
            else status = EXS;
            break;
case 27:    if (token == INTEGER) {
              if (sflag) {
                tsamples = intsymbol(v);
                tmxsamples = tsamples;
                tincsamples = 1;
                }
              else {
                tfail = intsymbol(v);
                tmxfail = tfail;
                tincfail = 1;
                }
              *state = 29;
              }
            else status = EXI;
            break;
case 28:    if (token == COMMA) *state = 31;
            else if (token == RPAREN) *state = 22;
            else status = EXCR;
            break;
case 29:    if (token == COMMA) *state = 30;
            else status = EXC;
            break;
case 30:    if (token == INTEGER) {
              if (sflag) tmxsamples = intsymbol(v);
              else tmxfail = intsymbol(v);
              *state = 32;
              }
            else status = EXI;
            break;
case 31:    if (token == FRACTION) {
              talpha[1] = floatsymbol(v);
              alphact = 2;
              *state = 33;
```

```
                }
              else status = EXF;
              break;
    case 32:  if (token == COMMA) *state = 34;
              else if (token == RPAREN) {
                if (sflag) tincsamples = DEFINC;
                else tincfail = DEFINC;
                *state = 22;
                }
              else status = EXCR;
              break;
    case 33:  if (token == COMMA) *state = 35;
              else if (token == RPAREN) *state = 22;
              else status = EXCR;
              break;
    case 34:  if (token == INTEGER) {
                if (sflag) tincsamples = intsymbol(v);
                else tincfail = intsymbol(v);
                *state = 36;
                }
              else status = EXI;
              break;
    case 35:  if (token == FRACTION) {
                talpha[2] = floatsymbol(v);
                alphact = 3;
                *state = 37;
                }
              else status = EXF;
              break;
    case 36:
    case 39:  if (token == RPAREN) *state = 22;
              else status = EXR;
              break;

    case 37:  if (token == COMMA) *state = 38;
              else if (token == RPAREN) *state = 22;
              else status = EXCR;
              break;
    case 38:  if (token == FRACTION) {
                talpha[3] = floatsymbol(v);
                alphact = 4;
                *state = 39;
                }
              else status = EXF;
              break;
    default:  status = ERROR;
    }
  return(status);
  }
```

```
/*************************************************************/

void
setoutputon()
/*   setoutputon  - enables  the  parser  to  generate  '.h'  and
     'makefile' commands.
     Opposite of 'setoutputoff()'.
*/
{
outputon = TRUE;
}

/*************************************************************/

void
setoutputoff()
/*   setoutputoff - disables the parser from generation of output.
     Opposite of 'setoutputon()'.
*/
{
outputon = FALSE;
}

/*************************************************************/

void
cpreset()
/* cpreset - resets initializing values for compiler.
*/
{
envflag = FALSE;
outputon = FALSE;
}
```

```
/*******************************************************
**                                                   **
**         Statistical Analysis Environment          **
**                  SAE COMPILER                     **
**                                                   **
**  File:  cp.token.c                                **
**  Original:  7 April 1986 - K N Cole               **
**  Current:  15 October 1986 - K N Cole             **
**                                                   **
**  Contains Modules:                                **
**     readword                                      **
**     readnumber                                    **
**     readfilename                                  **
**     token                                         **
**     tokenreset                                    **
**                                                   **
*******************************************************/

#include <stdio.h>
#include <ctype.h>
#include "sae.h"
#include "status.h"
#include "tokens.h"

struct resvdword {
  char *word;
  int value;
  }
langtable[LANGSIZE] = {
        "AL", ALPHA,
        "ALPHA", ALPHA,
        "AS", ASYMTOTIC,
        "ASYMTOTIC", ASYMTOTIC,
        "EN", ENVIRONMENT,
        "ENVIRONMENT", ENVIRONMENT,
        "EX", EXACT,
        "EXACT", EXACT,
        "FA", FAILURES,
        "FAILURES", FAILURES,
        "LR", LRT,
        "LRT", LRT,
        "MA", MAXTERMS,
        "MAXTERMS", MAXTERMS,
        "ME", METHOD,
        "METHOD", METHOD,
        "SA", SAMPLES,
        "SAMPLES", SAMPLES,
        "TA", TABLE,
        "TABLE", TABLE,
        "RA", RANGE,
        "RANGE", RANGE
        };
```

```
typedef struct symref {
  int token;
  int ivalue;
  float rvalue;
  char word[MAXWORD];
  } symtype;

symtype symtable[MAXSYMBOLS];
int nextsymbol;
```

```
/*****************************************************/

char
topper(c)
char c;
{
char cc;
if ((c >= 'a') && (c <= 'z')) cc = c - 32;
else cc = c;
return(cc);
}

/**********************************************************/

int
readword(sptr,v)
char **sptr;
int *v;
/* readword - reads a reserved word from the line pointed to
   by (sptr).  Returns a token and sets (v) to the value of
   the reserved word.
*/
{
int n, t;
char *w, *s;

for (n = 0; n < LANGSIZE; n++) {
  w = langtable[n].word;
  s = *sptr;
  while (*w && (topper(*s) == *w)) {s++; w++;}
  if (*w || isalpha(*s)) t = -INVLDWD;
  else {
    t = RESVD;
    *v = langtable[n].value;
    while (isalpha(**sptr)) (*sptr)++;
    break;
    }
  }
return(t);
}
```

```
/*****************************************************************/

int
readfilename(sptr, ns)
char **sptr;
int ns;
/* readfilename - reads a file name from the line pointed to
   by (sptr).
*/
{
char *c,*w;
int t,ct;

w = ++(*sptr);
c = symtable[ns].word;
ct = 0;
while (!isspace(*w) && ((*w != '"') && isprint(*w))) {
  if (++ct == MAXWORD) break;
  *c++ = *w++;
  }
*c = NULL;
if (*w == '"') {
  t = FILENAME;
  *sptr = ++w;
  }
else t = -INVLDFN;

return(t);
}
```

```
/*****************************************************************/

int
toint(c)
char c;
{
int v;
if ((c >= '0') && (c <= '9')) v = c -'0';
else v = NULL;
return(v);
}

int
readnumber(sptr, ns)
char **sptr;
int ns;
{
char *c;
int t, iv;
float rv, dv;

t = -INVLDNUM;
c = *sptr; iv = 0; rv = 0.0; dv = 0.1;
while (*c && isdigit(*c)) {
  iv = (10 * iv) + toint(*c);
  c++;
  }
if (iv > 0) t = INTEGER;
if (*c == '.') {
  c++;
  while (*c && isdigit(*c)) {
    rv = rv + (toint(*c) * dv);
    dv = dv / 10;
    c++;
    }
  rv = (float) iv + rv;
  if (rv > 0.0) t = FRACTION;
  }
if (*c && !isspace(*c))
  switch (*c) {
    case '(':
    case ')':
    case ';':
    case ',':  break;
    default:  t = -INVLDNUM;
    }
if (t != -INVLDNUM) {
  symtable[ns].rvalue = rv;
  symtable[ns].ivalue = iv;
  *sptr = c;
  }
return(t);
}
```

```
/*********************************************************************/

int
token(sptr,v)
char **sptr;
int *v;
/* token - reads the string pointed to by (sptr) and
   returns tokens based on the language (tokens.h).
   The value (*v) is set to a reference to the specific
   reserved word or an index to the symbol table (symtable[]).
*/
{
int tt, n;

*v = 0; tt = 0;
while (**sptr && isspace(**sptr)) (*sptr)++;      /* skip spaces */

if (nextsymbol == MAXSYMBOLS) nextsymbol = 0;

if (**sptr == '"') {                            /* read filename */
  if ((tt = readfilename(sptr, nextsymbol)) == FILENAME)
         *v = nextsymbol++;
  }
else if (isdigit(**sptr) || (**sptr == '.')) {  /* read number */
  if ((tt = readnumber(sptr, nextsymbol)) != -INVLDNUM)
         *v = nextsymbol++;
  }
else if (isalpha(**sptr)) {                     /* read reserved word */
  tt = readword(sptr, v);
  }
else switch (*(*sptr)++) {
  case NULL:     tt = EOL; break;
  case '-':      if (**sptr == '-') tt = EOL;
                 else tt = -INVLDCHAR;
                 break;
  case ';':      tt = SEMICOL; break;
  case '(':      tt = LPAREN; break;
  case ')':      tt = RPAREN; break;
  case ',':      tt = COMMA; break;
  case '=':      tt = EQUALS; break;
  default:       tt = -INVLDCHAR; break;
  }
return(tt);
}
```

```
/***********************************************************/

int
intsymbol(v)
int v;
{
int vv;

if ((v >= 0) && (v <= nextsymbol)) vv = symtable[v].ivalue;
else vv = 0;

return(vv);
}

/***********************************************************/

float
floatsymbol(v)
int v;
{
float vv;

if ((v >= 0) && (v <= nextsymbol)) vv = symtable[v].rvalue;
else vv = 0.0;

return(vv);
}

/***********************************************************/

char *
wordsymbol(v)
int v;
{
char *vv;

if ((v >= 0) && (v <= nextsymbol)) vv = symtable[v].word;
else vv = NULL;

return(vv);
}

/***********************************************************/

void
tokenreset()
/* tokenreset - initialize the symbol table pointer to top
*/
{
nextsymbol = 0;
}
```

```
/*******************************************************
**                                                    **
**          Statistical Analysis Environment          **
**                    SAE COMPILER                    **
**                                                    **
**  File:  cp.err.c                                   **
**  Original:  7 April 1986 - K N Cole                **
**  Current:  15 October 1986 - K N Cole              **
**                                                    **
**  Contains Modules:                                 **
**     errtext                                        **
**                                                    **
*******************************************************/
#include <stdio.h>
#include "status.h"

                                /* status and error information */
char *errtable[] = {                                        /* status code */
   " ",                                                     /* NONE        */
   "Not compiled",                                          /* UNCKD       */
   "VALID",                                                 /* VALID       */
   "VALID - Incomplete line",                               /* PARTVLD     */
   "ERROR - Non-specific",                                  /* ERROR       */
   "ERROR - Invalid character",                             /* INVLDCHAR */
   "ERROR - Expected filename or ENVIRONMENT",              /* EXFE        */
   "ERROR - Expected semicolon or left paren",              /* EXSL        */
   "ERROR - Expected comma or right paren",                 /* EXCR        */
   "ERROR - Expected semicolon at end of command",          /* EXS         */
   "ERROR - Expected argument (see language help)",         /* EXA         */
   "ERROR - Expected equal sign",                           /* EXE         */
   "ERROR - Expected integer",                              /* EXI         */
   "ERROR - Expected TABLE or LPT",                         /* EXTL        */
   "ERROR - Expected fraction",                             /* EXF         */
   "ERROR - Expected EXACT or ASYMTOTIC",                   /* EXEA        */
   "ERROR - Expected integer or RANGE",                     /* EXIR        */
   "ERROR - Expected left paren",                           /* EXL         */
   "ERROR - Expected right paren",                          /* EXR         */
   "ERROR - Expected comma",                                /* EXC         */
   "ERROR - Invalid integer or fraction",                   /* INVLDNUM    */
   "ERROR - Invalid file name",                             /* INVLDFN     */
   "ERROR - Invalid word in source"                         /* INVLDWD     */
};
```

```
/**********************************************************/

char *
errtext(ecode)
int ecode;
/* errtext - returns a pointer to the error explaination
   string when given a valid compiler error code.
*/
{
char *s;

s = NULL;
if ((ecode >= 0) && (ecode < MAXERROR)) s = errtable[ecode];

return(s);
}
```

```
# ****************************************
# **                                    **
# **  File:  math.make                  **
# **                                    **
# **  Current:  25 October 1986         **
# **                  Capt. K N Cole    **
# **                                    **
# **  Makefile for SAE math functions.  **
# **                                    **
# ****************************************

OBJECTS = Math/Object/bernpoly.o Math/Object/coefs.o \
        Math/Object/prob.o Math/Object/beta.o

CFLAGS = -c -O

#       Source files for modules

table: $(OBJECTS) Math/table.c Math/defs.h Math/env.h Math/table.h
        cc -O -o Math/table Math/table.c $(OBJECTS) -lm
        rm table.o

lrt: $(OBJECTS) Math/lrt.c Math/Object/compute.o Math/defs.h \
          Math/env.h Math/lrt.h
        cc -O -o Math/lrt Math/lrt.c Math/Object/compute.o \
          $(OBJECTS) -lm
        rm lrt.o

Math/Object/prob.o: Math/prob.c Math/defs.h
        cc $(CFLAGS) Math/prob.c
        mv prob.o Math/Object

Math/Object/compute.o: Math/compute.c Math/defs.h Math/lrt.h
        cc $(CFLAGS) Math/compute.c
        mv compute.o Math/Object

Math/Object/bernpoly.o: Math/bernpoly.c Math/defs.h
        cc $(CFLAGS) Math/bernpoly.c
        mv bernpoly.o Math/Object

Math/Object/coefs.o: Math/coefs.c Math/defs.h Math/env.h
        cc $(CFLAGS) Math/coefs.c
        mv coefs.o Math/Object

Math/Object/beta.o: Math/beta.c Math/defs.h
        cc $(CFLAGS) Math/beta.c
        mv beta.o Math/Object
```

```
#
#  Makefile v 3.0
#
#  Modified v 3.0  2 Oct 86
#  Create LRT program and simplify Table program
#
#  Modified v 2.0  25 Sep 86
#  Combine Asymt and Exact directories
#
#  Original v 1.0  26 Apr 85
#  Create executable program from original source code
#
# ******************************************
# **                                      **
# **  by Capt. Kenneth N. Cole            **
# **     DS-86                            **
# **                                      **
# **  for Dr. P. Nagarsenker              **
# **      AFIT/ENC                        **
# **                                      **
# ******************************************
OBJECTS = Object/bernpoly.o Object/coefs.o \
        Object/prob.o Object/beta.o

CFLAGS = -c -O

#       Source files for modules

table: $(OBJECTS) table.c defs.h  table.h
        cc -O -o table table.c $(OBJECTS) -lm
        rm table.o

lrt: $(OBJECTS) lrt.c Object/compute.o defs.h lrt.h
        cc -O -o lrt lrt.c Object/compute.o $(OBJECTS) -lm
        rm lrt.o

Object/prob.o: prob.c defs.h
        cc $(CFLAGS) prob.c
        mv prob.o Object

Object/compute.o: compute.c defs.h  lrt.h
        cc $(CFLAGS) compute.c
        mv compute.o Object

Object/bernpoly.o: bernpoly.c defs.h
        cc $(CFLAGS) bernpoly.c
        mv bernpoly.o Object

Object/coefs.o: coefs.c defs.h env.h
        cc $(CFLAGS) coefs.c
        mv coefs.o Object

Object/beta.o: beta.c defs.h
        cc $(CFLAGS) beta.c
        mv beta.o Object
```

```
/*****************************************************
**                                                 **
** File Name: defs.h                               **
**                                                 **
** Function: This file contains the definitions    **
**           necessary for the table and percent   **
**           programs.                             **
**                                                 **
** THIS FILE IS INCLUDED IN EACH PROGRAM CODING.   **
**                                                 **
** History: 9 May 86 - Original by                 **
**                         Capt. Kenneth N. Cole   **
**                                                 **
*****************************************************/

                                  /* general info */
#define boolean          int
#define TRUE             1
#define FALSE            0

#define LIMVAL           0.0000001
#define NEWTONLIMIT      26       /* attempts to reach limit */

#define MAXANSWER        80       /* string storage size */

/*****************************************************/




/*****************************************************
**                                                 **
** File Name: env.h      (default file)            **
**                                                 **
** Function: This file contains the definitions    **
**           necessary for the table and lrt       **
**           program environment definition.       **
**                                                 **
** THIS FILE IS INCLUDED IN EACH PROGRAM CODING.   **
**                                                 **
** History: 9 May 86 - Original by                 **
**                         Capt. Kenneth N. Cole   **
**                                                 **
*****************************************************/

#define EXACT            1
                                  /* size of arrays */
#define NUM              10

/*****************************************************/
```

```
/*********************************************************
**                                                     **
** File Name: table.h                                  **
**                                                     **
** Function: This file contains the definitions        **
**           necessary for the table program.          **
**                                                     **
** History: 9 May 86 - Original by                     **
**                         Capt. Kenneth N. Cole       **
**                                                     **
*********************************************************/

                                /* TABLE program info */
                                /* range of failures */
#define RINIT           4       /* start: at least 4 */
#define RINC1           1
#define RINC2           5
#define RSHIFT          20
#define RSET            20
#define RMAX            100

                                /* range of number of samples */
#define PINIT           2       /* must be at least 2 */
#define PMAX            6

#define ALPHA1          0.1
#define ALPHA2          0.05
#define ALPHA3          0.025
#define ALPHA4          0.01

#ifdef ALPHA4
#define MAXALPHA        4
#else
#ifdef ALPHA3
#define MAXALPHA        3
#else
#ifdef ALPHA2
#define MAXALPHA        2
#else
#define MAXALPHA        1
#endif
#endif
#endif

/*********************************************************/
```

```
/*************************************************************
**                                                         **
** File Name: lrt.h                                        **
**                                                         **
** Function: This file contains the definitions            **
**           necessary for the lrt program.                **
**                                                         **
** History: 9 May 86 - Original by                         **
**                          Capt. Kenneth N. Cole          **
**                                                         **
*************************************************************/

                                        /* LRT program info */
#define MINFAIL          2              /* minimum failures per sample */
#define MAXANSWER        80             /* length of user response */
#define MAXSAMPLES       10             /* default number of data samples */
#define ALPHA            0.01           /* default level of significance */

/*************************************************************/
```

```
/*********************************************************
**                                                     **
** File Name: bernpoly.c                               **
**                                                     **
** Contains Modules: com, b, bernpoly                  **
**                                                     **
** Current: 26 Sep 86                                  **
**                                                     **
*********************************************************/
#include <math.h>
#include "defs.h"

extern int arraysize;
extern long float barray[];

/*********************************************************
**                                                     **
** Module Name: com                                    **
** Function: Calculates the number of possible         **
**           combinations                              **
** Inputs: n, i                                        **
** Outputs: com (long float)                           **
** Calling Modules: b, bernpoly                        **
**                                                     **
** History: 8 May 85 - Capt. K.N. Cole                 **
**          translated from pascal                     **
**          (Original by Capt. Mark Amell)             **
**                                                     **
*********************************************************/

long float
com(n,i)
int n,i;
{
long float prod, fn, fi, fj;

fn = n;
fi = i;
fj = fn - fi;
prod = 1.0;
while (fi > 0.0)
  {
  prod = prod * (fn / fi);
  fn = fn - 1.0;
  fi = fi - 1.0;
  }
while (fj > 0.0)
  {
  prod = prod * (fn / fj);
  fn = fn - 1.0;
  fj = fj - 1.0;
  }
return(prod);
}
```

```
/*******************************************************
**                                                   **
** Module Name: b                                    **
** Function: Calculates the Bernoulli numbers        **
**           and stores them in array (barray)       **
**                                                   **
** Inputs: none                                      **
** Outputs: none                                     **
** Calling Modules: bernpoly                         **
**                                                   **
** History: 8 May 85 - Capt. K.N. Cole               **
**          translated from pascal                   **
**          (Original by Capt. Mark Amell)           **
**                                                   **
*******************************************************/

void
b()
{
long float sum;
int i,j;

barray[0] = 1.0;
for (j = 1; j <= (arraysize+1); j++) {
  sum = 0.0;
  for (i = 0; i <= (j-1); i++)
      sum = sum + com(j+1,i) * barray[i];
  barray[j] = (-1.0 * sum)/(j + 1);
  }
for (j = 1; j <= (arraysize+1); j++) {
  if ((barray[j] < 0.0000001) && (barray[j] > -0.0000001))
      barray[j] = 0.0;
  }
}
```

```
/*************************************************************
**                                                          **
** Module Name: bernpoly                                    **
** Function: Evaluates Bernoulli polynomials                **
**                                                          **
** Inputs: n, x                                             **
** Outputs: bernpoly (long float)                           **
** Calling Modules: ArCoef, createAjr                       **
**                                                          **
** History: 8 May 85 - Capt. K.N. Cole                      **
**          translated from pascal                          **
**          (Original by Capt. Mark Amell)                  **
**                                                          **
*************************************************************/

boolean bflag = FALSE;

long float
bernpoly (n, x)
int n;
long float x;
{
int i;
long float sum, power;

sum = 0.0;
power = 1.0;

if (!bflag) {
  bflag = TRUE;
  b();                       /* initialize barray */
  }
for (i = n; i >= 0; i--) {
  sum = sum + com(n,i) * barray[i] * power;
  power = power * x;
  }
return (sum);
}
```

```
/*********************************************************
**                                                     **
** File Name: beta.c                                   **
**                                                     **
** Contains Modules:  beta, betap, ibeta, loggam       **
**                                                     **
** Current: 29 Sep 86                                  **
**                                                     **
*********************************************************/
#include <math.h>
#include "defs.h"

/*********************************************************
**                                                     **
** Module Name: loggam                                 **
** Function: Computes the natural log of the Gamma     **
**           function of n                             **
** Inputs: dx                                          **
** Outputs: loggam (long float)                        **
** Calling Modules: beta, prob                         **
**                                                     **
** History: 1 May 85 - Capt. K.N. Cole                 **
**          translated from pascal program             **
**          (Original by Capt. Mark Amell)             **
**                                                     **
**********************************************************/


long float
loggam (dx)
  long float dx;
  {
  long float rdo,dy,dterm,de,da,db,domeg,dlggm;
  long float ds,dz,dw,dv,du,dt,dr,dq,dp;

  rdo = 0.0;
  dy = dx;
  dterm = 1.0;
  de = 1.0;
  domeg = 1.0e25;
  da = 0.9999999999;
  db = 1.0000000001;
  dlggm = domeg;
  if (dx >= rdo) {
      dlggm = rdo;
      if ((dx <= da) || (dx >= db)) {
          if ((dx <= (da+de)) || (dx >= (db+de))) {
              while ((dy-18.00) <= 0.0) {
                  dterm = dterm * dy;
                  dy = dy + de;
                  }
              ds = de / (dy * dy);
              dz = (long float) 0.0054102564102564l0;
              dw = (long float) -0.0019175269175269l8;
              dv = (long float) 0.000841750841750841 8;
              du = (long float) -0.000595238095238095 2;
```

```
                dt = (long float) 0.00079365079365079365;
                dr = (long float) -0.0027777777777777778;
                dq = (long float) 0.0833333333333333333;
                dp = (long float) 0.918938533204672;
                dlggm = ((dy - 0.5) * log(dy)) + dp - dy - log(dterm);
                dlggm = dlggm + ((((((dz * ds + dw) * ds + dv) *
                      ds + du) * ds + dt) * ds + dr) * ds + dq) / dy;
                }
            }
        }
    return(dlggm);
    }


/*********************************************************************
**                                                                 **
** Module Name: beta                                               **
** Function: Calculates the beta function of (p,q)                 **
**                                                                 **
** Inputs: p, q                                                    **
** Outputs: beta (long float)                                      **
** Modules Called: loggam                                          **
** Calling Modules: ibeta                                          **
**                                                                 **
** History: 1 May 85 - Capt. K.N. Cole                             **
**          translated from pascal                                 **
**          (Original by Capt. Mark Amell)                         **
**                                                                 **
*********************************************************************/

long float
beta (p, q)
long float p,q;
{
long float temp;

temp = exp(loggam(p) + loggam(q) - loggam(p+q));
return (temp);
}
```

```
/**************************************************************
**                                                          **
** Module Name: ibeta                                       **
** Function: Calculates the Incomplete beta function        **
**                                                          **
** Inputs: p, q, x                                          **
** Outputs: ibeta (long float)                              **
** Modules Called: beta                                     **
** Calling Modules: prob, betap                             **
**                                                          **
** History: 1 May 85 - Capt. K.N. Cole                      **
**          translated from pascal                          **
**          (Original by Capt. Mark Amell)                  **
**                                                          **
**************************************************************/

long float
ibeta (p, q, x)
  long float p, q, x;
  {
  long float  bint,pp,qq,dp,dq,xx,rxx,t,d,b,s;
  int ick,iq,ip,iqm,i;

  bint = 0.0;
  if ((fabs(x) <= 1.0e-17) || (x == 1.0)) bint = x;
     else {
        pp = p;
        qq = q;
        ick = 0;
        xx = x;
        iq = (int) floor(q);
        dq = (long float) iq;
        ip = (int) floor(p);
        dp = (long float) ip;
        if ((dq == q) || (dp == p)) {
          if (dp == p) {
            iq = ip;
            pp = q;
            qq = p;
            ick = 1;
            xx = 1.0 - x;
            }
          t = (long float) 1.0;
          rxx = xx / (1.0 - xx);
          bint = t;
          iqm = iq - 1;
          if (iqm != 0) {
            for (i = 1; i <= iqm; i++) {
              d = (long float) i;
              t = t * rxx * (qq-d) / (pp+d);
              bint = bint + t;
              if ((log10(rxx) + log10(t)) > 32)
                {
                bint = 1.0e36;
                break;
          }
        }
      }
    }
  else {
    if (xx > (pp/(pp+qq))) {
      pp = q;
      qq = p;
      xx = 1.0 - x;
      ick = 1;
      }
    bint = (long float) 1.0;
    t = (long float) 1.0;
    iq = (int) (qq + (1.0-xx) * (pp+qq));
    s = (long float) iq;
    if (iq == 0) {
      t = 1.0 - xx;
      bint = t;
      }
    else {
      if (iq != 1) {
        rxx = xx / (1.0 - xx);
        iqm = iq - 1;
        for (i = 1; i <= iqm; i++) {
          d = (long float) i;
          t = t * rxx * (qq-d) / (pp+d);
          bint = bint + t;
          }
        }
      t = t * xx * (qq-s) / (pp+s);
      bint = bint + t;
      }
    i = 1;
    while ((i < 101)
         && (fabs(t/bint) > 0.0000000000000000000000001)) {
      d = (long float) i;
      t = t * (pp + qq + d - 1.0) / (pp + s + d) * xx;
      bint = bint + t;
      i = i + 1;
      }
    }
  b = beta(p,q);
  bint = exp(pp * log(xx)) * exp((qq-1.0)
              * log(1.0-xx)) / (b*pp) * bint;
  if (ick == 1) {
    bint = 1.0 - bint;
    }
  }
  return (bint);
}
```

```
/*****************************************************************
**                                                              **
** Module Name: betap                                           **
** Function: Calculates the inverse beta function               **
**                                                              **
** Inputs: alpha, p, q                                          **
** Outputs: betap (long float)                                  **
** Modules Called: ibeta                                        **
** Calling Modules: newton                                      **
**                                                              **
** History: 1 May 85 - Capt. K.N. Cole                          **
**          translated from pascal                              **
**          (Original by Capt. Mark Amell)                      **
**                                                              **
*****************************************************************/

long float
betap (alpha, p, q)
  long float alpha, p, q;
  {
  int jj,jend,j,i;
  long float dp,dl,dif,dlx,dux,dmp,decr,dmpu,
      dm,dn,du,dfd,dabf,dfune,
      espl,esp2,esp3,esp4;
  boolean flag;
  long float darg[5], dfun[5];

  espl = 1.0e-180;
  esp2 = 1.0e-13;
  esp3 = 1.0e-11;
  esp4 = 1.0e-10;
  dp = alpha;
  dm = p;
  flag = TRUE;
  dn = q;
  du = 1.0;
  if (((dp * (du - dp)) < 0.0)||((dm < 0.0)||(dn < 0.0)))
    dmp = 0.0;
  else if ((dp * (du-dp)) == 0.0)
    dmp = alpha;
  else if (dm == 1.0)
    dmp = du - exp ((du/dn) * log(du - dp));
  else if (dn == 1.0)
    dmp = exp((du/dm) * log(dp));
  else {
    flag = FALSE;
    dl = 0.0;
    dif = 1.0/3.0;
    dlx = -dp;
    dux = du-dp;
    jj = 0;
    dmpu = 0.0;
    jend = 3;
    while ((jj < 25) && (!flag)) {
      if (jj == 25) jend = 3;
```

```
jj = jj + 1;
j = 1;
while ((j <= jend) && (!flag)) {
  dmp = (du+dl)/2.0;
  i = 1;
  if (((du-dl) < esp1) || (((du-dl) < (esp2*dp))
         && (dl > esp2)))
    flag = TRUE;
  else
    while ((i < 3) && (!flag)) {
      darg[i] = dl + (du-dl) * dif * i;
      dfun[i] = ibeta(dm,dn,darg[i]) - dp;
      if (dfun[i] == 0) dmp = darg[i];
      if (dfun[i] == 0)
        flag = TRUE;
      else if ((dfun[i] < 0.0) && (i == 2)) {
        dl = darg[2];
        dlx = dfun[2];
        }
      else if (dfun[i] > 0.0) {
        du = darg[i];
        dux = dfun[i];
        if (i == 2) {
          dl = darg[1];
          dlx = dfun[1];
          }
        else
          i = 2;
        }
        ++i;
      }
      ++j;
    }
    if (!flag) {
      jend = 2;
      dmp = (du+dl)/2.0;
      dfd = dux-dlx;
      if ((dfd < esp3) && (dfd < (esp4*dp)))
        flag = TRUE;
      }
    if (!flag) {
      decr = dux * (du-dl) / dfd;
      dmp = du - decr;
      if (((dmp-dl) < esp1)
              || (((dmp-dl) < esp2) && (dl > esp2)))
        flag = TRUE;
      if (!flag) {
        dfun[3] = ibeta(dm, dn, dmp) - dp;
        dabf = fabs(dfun[3]);
        dfune = dfun[3];
        if (((dabf < esp3) && (dabf < (esp4 * dp)))
              || ((dmp < esp1) || (((du-dmpu) < esp2)
              && (du > 0.999999999999)) || (dfun[3] == 0.0)))
          flag = TRUE;
        }
```

```
            if (!flag) {
              if (dfun[3] < 0.0) {
                if (decr < (0.9 * (du - dl))) {
                  dl = dmp;
                  dlx = dfune;
                  }
                else {
                  dmpu = dmp;
                  dmp = 5.0 * (dmp - dl) + dl;
                  dfune = ibeta(dm, dn, dmp) - dp;
                  if (dfune == 0.0)
                    flag = TRUE;
                  if (!flag) {
                    if (dfune < 0.0) {
                      dl = dmp;
                      dlx = dfune;
                    }
                  else {
                    du = dmp;
                    dux = dfune;
                    dl = dmpu;
                    dlx = dfun[3];
              } } } }
            else {
              if (decr >= (0.1 * (du - dl))) {
                du = dmp;
                dux = dfune;
                }
              else {
                dmpu = dmp;
                dmp = du - 5.0 * decr;
                dfune = ibeta(dm, dn, dmp) - dp;
                if (dfune == 0.0)
                  flag = TRUE;
                if (!flag) {
                  if (dfune < 0.0) {
                    du = dmpu;
                    dux = dfun[3];
                    dl = dmp;
                    dlx = dfune;
                    }
                  else {
                    du = dmp;
                    dux = dfune;
        } } } } } } } }
    return(dmp);
  }
```

```
/*******************************************************************
**                                                               **
** File Name: coefs.c                                            **
**                                                               **
** Contains Modules:  Queue, ArCoef, createCjr,                  **
**                     createAjr, createWjr,                     **
**                     modR, modS                                **
**                                                               **
** Current: 30 Sep 86                                            **
**                                                               **
*******************************************************************/
#include <math.h>
#include "defs.h"
#include "env.h"

extern long float bernpoly(), ibeta();
extern long float ka[];

long float R[NUM+1], S[NUM+1];
long float Ar[NUM+1], Q[NUM+1], Tr[NUM+1];
long float Ajr[NUM+1][NUM+1];
long float Cjr[NUM+1][NUM+1];
long float Wjr[NUM+1][NUM+1];
int arraysize = NUM;

long float barray[NUM+2];

#ifdef EXACT
boolean exact = TRUE;
#else
boolean exact = FALSE;
#endif
```

```
/*******************************************************
**                                                   **
** Module Name: ArCoef                               **
** Function: Calculates the Ar coefficients for      **
**             module Queue                          **
** Inputs: p, delta                                  **
** Outputs: none                                     **
** Global Tables Used: ka[]                          **
** Global Tables Changed: Ar[]                       **
** Calling Modules: Queue                            **
**                                                   **
** History:  Dec 84 - Capt K N Cole                  **
**           5 Nov 85 - Modified for new formula     **
**                                                   **
*******************************************************/

void
ArCoef(p,delta)
long float p,delta;
{
long float factor, partial;
int i, k;

Ar[0] = 1.0;
factor = 1.0;
for(k = 1; k <= arraysize; k++) {
  factor = factor * -1.0 / p;
  for (partial = 0.0, i = 0; i < p; i++)
    partial = partial + bernpoly(k+1, delta * ka[i])
                                  / exp(log(ka[i]) * k);
  Ar[k] = (factor/(long float)(k *(k+1)))
                            * (bernpoly(k+1,delta) - partial);
  }
}
```

```
/**********************************************************
**                                                      **
** Module Name: Queue                                   **
** Function: Calculates the Q coefficients for          **
**           module modR                                **
** Inputs: p, l, delta                                  **
** Outputs:                                             **
** Global Tables Used: Ar[]                             **
** Global Tables Changed: Q[]                           **
** Calling Modules: modR                                **
**                                                      **
** History: Dec 84 - Original by Capt. Mark F. Amell    **
**          1 May 85 - Headers Added - K. Cole          **
**          8 May 85 - Converted to C-language (KNC)    **
**          19 Aug 85 - No changes for Exact model      **
**                                                      **
**********************************************************/

void
Queue (p, l, delta)
long float p, l, delta;
{
int i, k;
long float sum;

ArCoef(p,delta);
Q[0] = 1.0;
for (i = 1; i <= arraysize; i++) {
  for (sum = 0.0, k = 1; k <= i; k++)
    sum = sum + k * Ar[k] * Q[i-k];
  Q[i] = (1.0/(long float) i) * sum;
  }
}
```

```
/************************************************************
**                                                        **
** Module Name: createAjr                                 **
** Function: Build the Ajr coefficients                   **
**                                                        **
** Inputs: a, v                                           **
** Outputs: none                                          **
** Global Tables Changed: Ajr[]                           **
** Calling Modules: createCjr                             **
**                                                        **
** History: Dec 84 - Original by Capt. Mark F. Amell      **
**          8 May 85 - Headers Added - K. Cole            **
**          19 Aaug 85 - No Changes for Exact model       **
**                                                        **
************************************************************/

void
createAjr(a,v)
long float a,v;
{
int l,m,rt;
long float sign, temp;

for (m = 0; m <= arraysize; ++m) Ajr[m][0] = 1.0;

for (sign = 1.0, l = 1; l <= arraysize; ++l) {
  rt = l + 1;
  temp = bernpoly(rt,arraysize,a);
  for (m = 0; m <= arraysize; ++m)
    Ajr[m][l] = (sign/(l * (l+1.0)))
                                    * (temp-bernpoly(rt,arraysize,a+v+m));
  sign = sign * (-1.0);
  }
}
```

```
/**********************************************************
**                                                      **
** Module Name: createCjr                               **
** Function: Creates the Cjr coefficients for modR      **
**                                                      **
** Inputs: a, v                                         **
** Outputs:                                             **
** Global Tables Used: Ajr[]                            **
** Global Tables Changed: Cjr[]                         **
** Calling Modules: modR                                **
**                                                      **
** History: Dec 84 - Original by Capt. Mark F. Amell    **
**          8 May 85 - Headers Added - K. Cole          **
**          19 Aug 85 - No Changes for Exact model      **
**                                                      **
**********************************************************/

void
createCjr(a,v)
long float a,v;
{
int l,m,k;
long float sum;

createAjr(a,v);
for (m = 0; m <= arraysize; m++)
  Cjr[m][0] = 1.0;
for (l = 1; l <= arraysize; l++) {
  for (m = 0; m <= arraysize; m++) {
    for (sum = 0.0, k = 1; k <= l; k++)
      sum = sum + k * Ajr[m][k] * Cjr[m][l-k];
    Cjr[m][l] = sum / (long float) l;
    }
  }
}
```

```
/*****************************************************
**                                                  **
** Module Name: modR                                **
** Function: Creates the R coefficients for the     **
**           function prob                          **
** Inputs: a, v, p, t, delta                        **
** Outputs:                                         **
** Global Tables Used: Q[], Cjr[]                   **
** Global Tables Changed: R[]                       **
** Calling Modules: main                            **
**                                                  **
** History: Dec 84 - Original by Capt. Mark F. Amell **
**          1 May 85 - Headers Added - K. Cole      **
**          8 May 85 - Converted to C-language (KNC **
**          19 Aug 85 - No Changes for Exact model  **
**                                                  **
*****************************************************/

void
modR (a, v, p, t, delta)
long float a, v, p, t, delta;
{
long float sum;
int i,k;

createCjr(a,v);
Queue(p,t,delta);
R[0] = 1.0;
for (i = 1; i <= arraysize; i++) {
  for (sum = 0.0, k = 1; k <= i; k++)
    sum = sum + R[i-k] * Cjr[i-k][k];
  R[i] = (Q[i] - sum) / Cjr[i][0];
  }
}
```

```
/*****************************************************
**                                                  **
** Module Name: createWjr                           **
** Function: Creates the Wjr coefficients for modR  **
**                                                  **
** Inputs: none                                     **
** Outputs: none                                    **
** Global Tables Used: Cjr[], Ar[]                  **
** Global Tables Changed: Wjr[], Tr[]               **
** Calling Modules: modS                            **
**                                                  **
** History: 20 Oct 85 - Original by Capt. K.N. Cole **
**                                                  **
*****************************************************/

void
createWjr()
{
int i,k,m;
long float sum;

Tr[0] = 1.0;

for (i = 1; i <= arraysize; i++) {
  for (sum = 0.0, k = 1; k <= i; k++)
    sum = sum + (long float) k * (-Ar[k]) * Tr[i-k];
  Tr[i] = (1.0/(long float) i) * sum;
  }

for (i = 0; i <= arraysize; i++)  Wjr[i][0] = 1.0;

for (i = 1; i <= arraysize; i++)
  for (m = 0; m <= arraysize; m++) {
    for (sum = 0.0, k = 0; k <= i; k++)
      sum = sum + (Tr[k] * Cjr[m][i-k]);
    Wjr[m][i] = sum;
    }
}
```

```
/*************************************************************
**                                                          **
** Module Name: modS                                        **
** Function: Creates the S coefficients for the             **
**              function prob                               **
** Inputs: a, v, p, t, x, m                                 **
** Outputs: none                                            **
** Global Tables Used: R[], Wjr[]                           **
** Global Tables Changed: S[]                               **
** Calling Modules: newton                                  **
**                                                          **
** History: 20 Oct 85 - Original by Capt. K.N. Cole         **
**                                                          **
*************************************************************/

boolean Wjrflag = FALSE;

void
modS (a, v, x, m)
long float a, v, x, m;
{
long float sum;
int i,k;

if (!Wjrflag) {             /* create Wjr coefficients */
  createWjr();
  Wjrflag = TRUE;
  }
S[0] = 1.0;                 /* create S coefficients */
for (i = 1; i <= arraysize; i++) {
  for (sum = 0.0, k = 1; k <= i; k++)
    sum = sum + (R[k] * ibeta(m+a, v+k, x) * Wjr[k][i-k]);
  S[i] = sum;
  }
}
```

```
/*******************************************************
**                                                   **
** File Name: prob.c                                 **
**                                                   **
** Contains Modules: prob, newton                    **
**                                                   **
** Current:  30 Sep 86                               **
**                                                   **
*******************************************************/
#include <stdio.h>
#include <math.h>
#include "defs.h"

extern long float loggam(), ibeta(), betap();
extern void modS();
extern long float R[], ka[], S[];
extern boolean exact;
extern int arraysize;
```

```
/*************************************************************
**                                                          **
** Module Name: prob                                        **
** Function: Computes the distribution function of x        **
**                                                          **
** Inputs: delta, a, v, p, m, x, limit                      **
** Outputs: prob (long float)                               **
** Global Tables Used: R[], S[], ka[]                       **
** Calling Modules: newton                                  **
**                                                          **
** History: 1 May 85 - Capt. K. N. Cole                     **
**                                                          **
*************************************************************/

long float
prob (delta, a, v, p, t, m, x, limit)
long float delta, a, v, p, t, m, x;
int limit;
{
int i,k;
long float tsum, t2, t3, K1;

if (exact) {                  /* exact method */
  tsum = 0.0;
  t3 = (p * m) + delta;  /* R */

  for (k = 0, t2 = 0.0; k < (int) p; k++) {
    tsum = tsum + ((t3 * ka[k] - 0.5) * log(ka[k]));
    t2 = t2 + loggam(t3 * ka[k]);
    }

  K1 = exp(loggam(t3) - t2 + tsum + (v * log((2.0 * 3.14159)/p)));

  for (i = 0, t2 = 0.0; i <= limit; i++) {
    tsum = exp(loggam(m+a) - loggam(m+a+v+i));
    t2 = t2 + R[i] * tsum * ibeta(m+a, v+i, x);
    }

  tsum = K1 * t2;
  }
else {                        /* Asymtotic Method */
  modS(a,v,t,x,m);
  tsum = ibeta(m+a, v, x);
  for (i = 1; i <= limit; i++)
    tsum = tsum + (S[i] / exp(i * log(m)));
  }

return(tsum);
}
```

```
/**********************************************************
**                                                      **
** Module Name: newton                                  **
** Function: Calculates the percentage point using      **
**           Newton's approximation method              **
** Inputs: delta, a, v, t, p, m, alpha, R               **
** Outputs: newton (long float)                         **
** Calling Modules: table, computepct                   **
**                                                      **
** History: 1 May 85 - Capt. K. N. Cole                 **
**                                                      **
**********************************************************/

long float
newton(delta, a, v, t, p, m, alpha, RR)
long float delta, a, v, t, p, m, alpha;
int RR;
  {
  long float z[NEWTONLIMIT+2], pa[NEWTONLIMIT+2];
  long float sp, x, sign;
  int k, done, limit;

  z[1] = betap(alpha,RR-t,v);
  if (p <= 3.0)
    z[2] = z[1] + 0.05;
  else
    if (z[1] > 0.05) z[2] = z[1] - 0.05;
    else z[2] = z[1]/2.0;
  x = z[1];

  limit = arraysize-1;
  sp = prob(delta,a,v,p,t,m,x,limit);
  pa[1] = sp;

  done = FALSE;
  for (k = 2; (k <= NEWTONLIMIT) && !done; k++)
      {
       if (z[k] > 1.0) z[k] = z[k-1] + ((1 - z[k-1]) / 2.0);
       x = z[k];

       sp = prob(delta,a,v,p,t,m,x,limit);

       pa[k] = sp;
       if (fabs(sp-alpha) < 0.0000001) done = TRUE;
       else
          z[k+1] = z[k] - (z[k]-z[k-1]) * (sp-alpha) / (sp-pa[k-1]);

       if (z[k+1] < 0.0) z[k+1] = z[k] / 2.0;
       if ((sp > 1.0) || (sp < 0.0)) {
          printf("\nNewton.c:  Incorrect Prob Value %f %f \n",x,sp);
          done = TRUE;
          }
       }
  return(z[k-1]);
  }
```

```
/*********************************************************
**                                                      **
** File Name: table.c                                   **
**                                                      **
** Module Name: main                                    **
** Function: This is the main program for the           **
**          percentage point table generation           **
**                                                      **
** History: 8 May 85 - Original by                      **
**                              Capt. Kenneth N. Cole   **
**          25 Oct 86 - Modified for SAE system         **
**                                                      **
**********************************************************/
#include <stdio.h>
#include "defs.h"
#include "table.h"

extern long float R[], S[];
extern boolean exact;
extern int arraysize;
extern void modR();
extern long float newton(), prob();

long float ka[PMAX+1];

long float alist[MAXALPHA] = { ALPHA1,
#ifdef ALPHA2
                                ALPHA2,
#endif
#ifdef ALPHA3
                                ALPHA3,
#endif
#ifdef ALPHA4
                                ALPHA4
#endif
                                };

/**********************************************************/

main ()
{
long float p, m, v, a, t, RR;
long float temp, prb, sum, delta;
int i, k, r;

                /* print header to identify program */
printf("\nSAE Table Generation Program    Version 2.0\n");
if (exact) {
  printf("\nExact Method is used in calculating
                            the test criteria\n");
  printf("     Maximum number of terms used is %d\n",arraysize);
  }
else {
  printf("\nAsymtotic Method is used in calculating
                           the test criteria\n");
```

```
    printf("     %d Terms used\n",arraysize);
    }

for (p = PINIT; p <= PMAX; p = p + PINC)
                              /* loop on number of samples */
    {
    v = (p - 1.0) / (long float) 2.0;
    a = (1.0 - v)/ (long float) 2.0;
    t = (p + 1.0) / (6.0 * p);
    for (i = 0; i < p; i++) ka[i] = 1.0/p;

    for (i = 0, sum = 0; i < p; i++) sum = sum + 1.0/ka[i];
    delta = (sum - 1.0) / ((p - 1.0) * 6.0);

    modR(a,v,p,t,delta);          /* initialize coefficients */

    printf("\n\n");               /* print table header */
    printf("Percentage points of L = lambda to the power (p/R)\n");
    printf("     p = %g samples \n", p);
    printf("     R = %g x
                     (number of failures in each sample)\n\n",p);
    printf("          Level of Significance\n");
    printf("Failures    %4.3f",ALPHA1);
#ifdef ALPHA2
    printf("        %4.3f",ALPHA2);
#endif
#ifdef ALPHA3
    printf("        %4.3f",ALPHA3);
#endif
#ifdef ALPHA4
    printf("        %4.3f",ALPHA4);
#endif
    printf("\nper sample\n");

    for (r = RINIT; r <= RMAX; r = r + RINC) {

      if (r < (p+1)) r = (int) (p + 1);    /* minimum value r = p+1 */
      printf("\n  %4d", r);

      m = ((p * (long float) r) - delta) / p;

      for (k = 0; k < MAXALPHA; k++) {  /* loop through alpha values */
        prb = newton(delta,a,v,t,p,m,alist[k],(p*(long float) r));
        printf("      %8.6f", prb);
        }
      }
    printf("\n\n");
    }
}
```

```
/*******************************************************
**                                                    **
** File Name: compute.c                               **
**                                                    **
** Module Names: readfile()                           **
**               computelrt()                         **
**               computepct()                         **
**                                                    **
** Function: Computes the LRT Criteria and percentage **
**           point from the info stored.              **
** Inputs:                                            **
**    readfile(fn)                                    **
**    computepct(alpha)                               **
** Outputs:                                           **
**    (long float) computelrt()                       **
**    (long float) computepct()                       **
** Calling Modules:                                   **
**    main (lrt.c)                                    **
**                                                    **
** History: 9 Sep 85 - Original by                    **
**                          Capt. Kenneth N. Cole     **
**                                                    **
*******************************************************/
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include "defs.h"
#include "lrt.h"

extern void modR();
extern long float newton(), prob();

int sampcount = 0;               /* globals */
int fail = 0;
int sampfail[MAXSAMPLES];
long float sampsum[MAXSAMPLES], sampsize[MAXSAMPLES];
long float ka[MAXSAMPLES+1];
```

```
/*************************************************************
**                                                          **
** Module Name: readfile                                    **
** Function: read data files for the LRT program            **
**                                                          **
** History: 8 May 85 - Original by                          **
**                              Capt. Kenneth N. Cole       **
**          25 Oct 86 - Modified for SAE system             **
**                                                          **
*************************************************************/

int
readfile(fn)
char *fn;
/* readfile - reads the data file named (fn) and
   returns (TRUE) if all the information is provided
*/
{
FILE *fp, *fopen();
boolean error, getn;
char *c, s[MAXANSWER];
int n, r;
long float t, tt, maxt;

if (sampcount == MAXSAMPLES) error = TRUE;
else {
  error = FALSE; getn = TRUE;
  maxt = 0.0;
  c = fn;

  while (isspace(*c)) c++;
  if (*c == '-') {
    printf("Enter Sample Size: ");
    fgets(s,MAXANSWER,fp);
    sscanf(s, " %d", &n);
    c++;
    getn = FALSE;
    }

  if (fp = fopen(c, "r")) {
    if (getn) fscanf(fp, " %d ", &n);
    r = 0; tt = 0.0;
    while (!feof(fp)) {
      t = 0.0;
      fscanf(fp, "%lf ", &t);
      if (t != 0.0) {
        if (t > maxt) maxt = t;
        tt = tt + t;
        r++;
        }
      else
        while (!feof(fp) && (fgetc(fp) != '\n'));
      }
    fclose(fp);
    if (n < r) error = TRUE;
```

```
        if ((tt == 0) || (r == 0)) error = TRUE;
        }
      else {
        printf("Readfile: can't open data file -> %s\n", c);
        error = TRUE;
        }

      if (!error) {
        tt = tt + ((n - r) * maxt);
        sampsum[sampcount] = tt;
        sampsize[sampcount] = n;
        sampfail[sampcount] = r;
        sampcount++;
        }
      }
    return(!error);
    }
```

```
/*************************************************************
**                                                          **
** Module Name: computelrt                                  **
** Function: compute the LRT value for the data read        **
**                                                          **
** History: 8 May 85 - Original by                          **
**                            Capt. Kenneth N. Cole         **
**           25 Oct 86 - Modified for SAE system            **
**                                                          **
*************************************************************/

long float
computelrt ()
/* computelrt - Compute the test criteria from the data
   contained in the global variables (sampcount, sampsum,
   sampsize, and sampfail).
*/
{
long float numer, denom, lrt, templ, temp2, sum;
int i, j;

if (sampcount >= 2) {
  sum = 0.0; fail = 0;          /* total the failures */
  for (i = 0; i < sampcount; i++) {
    fail = fail + sampfail[i];
    sum = sum + sampsum[i];
    }

  numer = 0.0;                  /* compute the numerator */
  for (i = 0; i < sampcount; i++) {
    templ = log(sampsum[i]/(long float) sampfail[i]);
    for (temp2 = 0.0, j = 1; j <= sampfail[i]; j++)
        temp2 = temp2 + templ;
    numer = numer + temp2;
    }
  numer = numer * ((long float) sampcount/(long float) fail);

  denom = 0.0;                  /* compute the denominator */
  for (i = 0; i < sampcount; i++) {
        denom = denom + log(sum / (long float) fail);
        }

  lrt = numer - denom;
  lrt = exp(lrt);
  }
return(lrt);
}
```

```
/*************************************************************
**                                                         **
** Module Name: computepct                                 **
** Function: compute the percentage point needed           **
**                                                         **
** History: 8 May 85 - Original by                         **
**                             Capt. Kenneth N. Cole       **
**           25 Oct 86 - Modified for SAE system           **
**                                                         **
*************************************************************/

void
computepct(alpha,lrt,pct,prb,limit)
long float alpha,lrt;
long float *pct, *prb;
int limit;
/* computepct - Compute the percentage point necessary for
   evaluating the LRT criteria at the values given
   by the global variables (sampcount, sampsum, sampsize
   sampfail and fail).
   Also, compute the probability that the value of the criteria
   is less than or equal to the LRT value given.
   Values are stored in the locations pointed to by
   pct -> percentage point for alpha value given, and
   prb -> probability value.
*/
{
long float m, v, a, t, delta, p, temp;
int i;

p = (long float) sampcount;
v = (p - 1.0) / (long float) 2.0;
a = (1.0 - v)/ (long float) 2.0;
t = (p + 1.0) / (6.0 * p);

for (i = 0; i < sampcount; i++)
  ka[i] = (long float) sampfail[i] / (long float) fail;

for (i = 0, temp = 0; i < sampcount; i++) temp = temp + 1.0/ka[i];
delta = (temp - 1.0) / ((p - 1.0) * 6.0);

modR(a,v,p,t,delta);            /* initialize coefficients */

m = ((long float) fail - delta) / p;
*prb = prob(delta,a,v,p,t,m,lrt,limit);
*pct = newton(delta,a,v,t,p,m,alpha,fail);
}
```

```
/***********************************************************
**                                                       **
** File Name: lrt.c                                      **
**                                                       **
** Module Name: main                                     **
** Function: Computes the LRT Criteria from the input    **
**           data provided.                              **
** Modules Called: computelrt, computepct, readfile      **
**                                                       **
** History: 9 Sep 85 - Original by                       **
**                             Capt. Kenneth N. Cole     **
**                                                       **
***********************************************************/
#include <stdio.h>
#include <ctype.h>
#include "defs.h"
#include "env.h"
#include "lrt.h"

extern long float computelrt();
extern void computepct();
extern int readfile();

main(argc, argv)
int argc;
char *argv[];
{
int p, sc;
char fn[MAXANSWER], s[MAXANSWER], *c;
long float C, prb, alpha, lrt;

                        /* print header to identify program */
printf("\nLikelihood Ratio Test Program   Version 2.0\n");
#ifdef EXACT
printf("\nExact Method for calculating test criteria\n");
printf("     Maximum number of terms used: %d\n",NUM);
#else
printf("\nAsymtotic Method for calculating test criteria\n");
printf("     %d Terms used\n",NUM);
#endif
printf("     %d Samples may be given\n",MAXSAMPLES);
#ifdef ALPHA
printf("     %4.3f Level of significance\n",ALPHA);
#else
printf("     Level of significance will be requested later\n");
#endif
p = 0;                  /* initialize sample counter */
if (argc > 1) {         /* sample filenames given in call */
  while (--argc > 0)
    if (readfile(*++argv)) p++;
  }
else {                  /* no arguments ... prompt for inputs */
  printf("Enter file name of first sample: ");
  gets(s);
  while (*s) {
      if (sscanf(s, "%s", fn))
        if (readfile(fn)) p++;
      printf("Enter file name of next sample: ");
      gets(s);
      }
    }

  if (p > 1) {
    printf("\nSummary:\n");
    lrt = computelrt();
    printf("Test criteria from data files = %g\n", lrt);
#ifdef ALPHA
    alpha = ALPHA;
#else
    printf("Enter desired level of significance: ");
    scanf("%lf", &alpha);
#endif
    computepct(alpha,lrt,&C,&prb,NUM);
    printf("\nProbability that LRT value is less than
                                        or equal to %g\n",lrt);
    printf("   when the sample populations have the
                                        same failure rate ");
    printf("= %4.3f \n\n", prb);
    printf("Reference point @ %4.3f Significance = %g\n", alpha, C);
    printf("Therefore:\n");
    if (lrt < C) {
      printf("... Samples CANNOT be assumed to have come
                                        from populations\n");
      printf("    with the same failure distributions\n");
      }
    else {
      printf("... Samples MAY have come from populations
                                        with the same\n");
      printf("    failure distributions.\n");
      }
    }
  else
    printf("\nLRT: can't compute LRT from less than TWO samples\n\n");
  }
```

Vita

Capt. Kenneth N. Cole received Bachelor of Science degrees in Business Management from the University of Maryland, in 1967, and Electrical Engineering from the University of Florida, in 1980. He has also received the M.S.E.E degree from the Air Force Institute of Technology (AFIT), in 1983.

From 1980 to 1982 Captain Cole was a Development Systems Engineer for the Air Force Aeronautical Systems Division at Wright-Patterson Air Force Base, Dayton, Ohio. He began full-time studies at AFIT in June 1982. He has an assignment to the Air Force Weapons Lab, Kirtland Air Force Base, Albuquerque, New Mexico, beginning in January, 1987.

Capt. Cole is a member of the Tau Beta Pi and Eta Kappa Nu engineering honor societies and of the Association for Computing Machinery and the IEEE Computer Society.

Permanent Address: 245 Aspinwall Road
Troy, Michigan 48098