Software tool for reliability estimation

Manish Jhunjhunwala
West Virginia University

Software Tool for Reliability Estimation

Manish Jhunjhunwala

Thesis submitted to the College of Engineering and Mineral Resources at West Virginia University in partial fulfillment of the requirements for the degree of

Master of Science
in
Industrial Engineering

Morgantown, West Virginia

2001

Keywords: Hardware Reliability, Software Reliability, System Reliability, Reliability Tool

ABSTRACT
Software Tool for Reliability Estimation

Manish Jhunjhunwala

Reliability engineering is now an essential part of product development. In the area of electronics, reliability needs to be checked for each critical component before the product is shipped. Electronic products can range from computers to pace makers.

Over the years, several tools have been developed to estimate hardware/software reliability. Typically, such tools are either for hardware or software. This thesis presents a Software Tool for Reliability Estimation (STORE), which can be used for hardware components, software components, and for systems having both hardware and software components.

The objective of this research was to study several hardware-software reliability models, existing algorithms to compute system reliability and to model state dependent systems. A comprehensive software tool to estimate reliability of hardware and software component that are state dependent and independent was developed. This software tool was implemented in Visual Basic for a Windows 98 operating system.

Acknowledgment

I would like to express my sincere thanks and gratitude to Dr. Rashpal Ahluwalia, my committee chair for his continuous support throughout this research. He was always available to answer my questions and explicate problems that I encountered. I also appreciate the time and effort of Dr. Majid Jaraiedi and Mr. Kenneth McGill for their helpful comments and suggestions.

I am thankful to the Industrial and Management Systems Engineering Department for its continued financial support which enabled me to complete my graduate work. Finally, I would like to thank my parents, my brother and all my friends for their support all through my studies.

Table of Contents

List of Notations

B	Bartlett's test statistic
b_0, b_1, b_2..	Parameters for software models
C_i	ith Cut set
cor	Correlation Coefficient
E[X]	Expected value of random variable X
F(t)	Probability Density Function
i,j	Indices indicating sequential number of failure events
m_t	Number of failures experienced by time t
m	Mean
M	Mann's Test Statistic
$M(\tau)$	Number of failures experienced by time τ (random variable)
$P[E_1]$	Probability of event E_1
$P[E_1\|E_2]$	Probability of event E_1 given event E_2 occurred
Q_s	Unreliability of a system
R_s	Reliability of a system
R(t)	Reliability function
σ^2	Variance
SSE	Sum of Squares of Errors
SSx	Sum of Squares of variable X
SSy	Sum of Squares of variable Y
t_e	A specific deterministic time denoting the end of failure data.
t_i	Time of ith failure
T_{med}	Shape parameter for lognormal distribution
T_i	ith Tieset
TS	T Statistic
MLE	Maximum Likelihood Estimator
h(t)	Hazard rate
α	Confidence level; significance of a model or probability of rejecting a correct model.
β	Shape parameter for Weibull distribution (hardware reliability)

λ	Parameter for exponential distribution used in hardware reliability or the hazard rate
$\lambda(\tau)$	Failure intensity function or $d\mu/dt$
μ	Mean
$\mu(\tau)$	Mean value function or expected number of failures experienced by time t or $E[M(\tau)]$
σ	Standard deviation
τ	(Realization of) cumulative execution time
γ	Location parameter for Weibull distribution
η	Scale parameter for Weibull distribution

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Background

Since the first electronic digital computer was invented almost fifty years ago, we have become dependent on computers in our daily lives. The computer revolution has created the fastest technological advancement that the world has ever seen. Today, the computer hardware and software applications are pervasive in all aspects of our society. The newest cameras, VCRs, and automobile could not be controlled and operated without computers. Computers are also embedded in wristwatches, telephones, home appliances, buildings, and aircraft. Science and technology have been demanding high performance hardware and high quality software for making improvements and breakthroughs. We can look at virtually any industry, automotive, avionics, oil, telecommunications, banking, semiconductor, pharmaceutics, etc., and see that they rely heavily on computers for their functioning [24].

The size and complexity of computer intensive systems have grown dramatically, and this trend will certainly continue in the future. Contemporary examples of highly complex hardware/software systems can be found in projects undertaken by NASA, Department of Defense (DoD), the Federal Aviation Administration (FAA), the telecommunications industry and a variety of other private industries.

The demand for complex hardware/software systems has increased more rapidly than the ability to design, implement, test and maintain them. When dependency on computers increases, the possibility of crises from computer failures also increases. The impact of these failures ranges from inconvenience (e.g., malfunctions of home appliances), economic damage (e.g., interruptions of banking systems), to loss of life (e.g., failures of flight systems or medical software).

As the system becomes more complex, data collection becomes tougher. That is, most of the experience with the models is during the testing phase of actual projects, during which researchers have little or no control over the data with which they work. The complexity of real world failure data may obscure properties of software reliability models that might be revealed by executing the models on a simpler data sets [22].

As per Nikora [22] most appropriate software reliability model is a process that continues throughout the testing phase. Even if the characteristics of the testing process are well-known for a particular development effort and whose assumptions best match these characteristics, there is no guarantee that the model will be most appropriate model. That is, reliance on a single measure to choose the most appropriate model can lead to making an incorrect choice.

The word reliability is used in everyday life in more qualitative terms than quantitative terms. It is often said that a particular equipment, motor, or automobile is reliable or unreliable, implying that it is dependable or not dependable, respectively. If a component is dependable, the possibility that it can ever fail cannot be ruled out.

Reliability [1] is defined as the probability that a component or system will perform its intended function for a given time period, when used under stated operating conditions. It is the probability of non-failure over time. According to the definition, it takes failure data to measure reliability. The failure data can be fitted to either constant failure rate distribution, such as exponential or time dependent failure distributions like Normal, Lognormal or Weibull distribution.

Maintainability [1] is defined as the probability that a failed component or a system will be restored or repaired to a specified condition within a period of time when maintenance is performed in accordance with prescribed procedures. Maintainability, is the probability of repair in a given time. Usually, when maintainability is computed, time is defined as the clock time. It may or may not include such measures as waiting time for maintenance personnel and parts, travel time, and administrative time. Often, maintainability refers to the inherent repair time, which includes only the hands-on repair of the failed unit and not any administrative or resource delay times.

Availability [1] is defined as the probability that a component or system is performing its required function at a given point in time when used under stated operating conditions. Availability may also be interpreted as percent time a component or a system is operating. It differs from reliability in that availability is the probability that the component is currently in a non-failure state even though it may have previously failed and been restored to its normal operating conditions. Therefore system availability can never be less than system reliability. Availability can be the preferred measure when a system or component can be restored since it accounts for both reliability and maintainability.

1.2 Reliability Engineering – Present Status

Over the period of time, several techniques have been developed to systematically model and analyze real systems. These techniques are constantly being enhanced to improve the modeling process and to reduce the time required for analysis of that model. A well-modeled system may be too complex to be analyzed within a reasonable amount of time. On the other hand, an oversimplified model may not be able to depict the characteristics of a real system. A global solution to these problems is to reach a compromise between the model complexity and the model execution and analysis time.

Recently, a study was undertaken at the Jet Propulsion Laboratory (JPL) to determine the applicability of current software reliability measurement techniques, to JPL's development efforts. One of the findings was that there is no known method for identifying the model most applicable to a development effort prior to test. This can be mitigated, however by executing several models at once and using statistical methods to identify the model that is most likely to produce accurate results. From the study it was also found that more accurate predictions can be produced by combining the results of several models in a linear fashion [23].

1.3 Problem Statement

Reliability is an important parameter of any system. Various techniques are used to compute system reliability. These techniques can be classified into two categories. The first category consists of techniques that find an approximate reliability solution, whereas the second category consists of techniques that find the exact solution. Some of the commonly used techniques that find the exact network reliability are the conditional probability technique, the cut set technique and the tie set technique. For a large complex system the computations times for such techniques can be extensive. The approximate reliability solutions that make use of the simplifying assumptions are much faster than the exact solutions. The penalty paid for this fast computation is a certain degree of inaccuracy in the solution. Another limitation of the approximate solution is that it does not produce the symbolic expression for terminal reliability. Determining symbolic expression for reliability is important to reevaluate the system reliability if the reliability of an individual link has changed, or to improve the system reliability under a given cost constraint.

Parekh[24] implemented various hardware and software models in Visual Basic. A major limitation of his system is that it does not compute system tiesets and cutsets and does not model state dependent systems. The focus of this work is to implement an algorithm to compute system tiesets and cutsets and to develop a procedure for analyzing state dependent systems.

1.5 Objectives

The objective of this research were as follows:

- Study existing statistical models to estimate reliability of Hardware-Software components.
- Study existing algorithm to compute reliability of complex systems.
- Implement an algorithm to compute tiesets and cutsets of complex systems.
- Compute reliability (with known or unknown parameters) of state dependent and state independent systems.

1.6 Methodology

To achieve the objectives of this research, the following methodology was carried out:

- Literature survey was done in the area of reliability and reliability estimation tool. Existing reliability tool as in [24] was enhanced to compute reliability of a system having software/hardware or both software and hardware components.
- The work was then documented explaining STORE and its application. Various examples were run on STORE and documented.
- STORE was made in Microsoft Visual Basic 6 to make it a user-friendly tool. This tool was integrated with hardware, software and system reliability.

Chapter 2: Literature Review

This chapter reviews development of both hardware reliability and software reliability models, the purpose is to indicate which concepts have been tried successfully or not successfully and which have been modified and adapted. There are several distinct themes in historical development of this subject.

- The creation of various models relating reliability to failures times experienced.
- A concern with how to estimate model parameters.
- An interest in comparison of models, which led to the development of comparison criteria.
- The classification of models.
- An increasing concern with collecting failure data.

2.1 Reliability

Dimitri [4] describes the objective of reliability as follows:

1. Determine if the performance of components, equipment, and systems with or without corrective and preventive maintenance, and with known operating procedures, is within specifications for the desired function period, and if not, whether it is the result of a malfunction or of a failure which requires corrective action.
2. Determine the pattern of recurring failures, the cause of failures, and underlying times-to-failure distribution.
3. Determine the failure rate, the mean life, and the reliability of components, equipment, and systems and their associated confidence limits at the desired levels.
4. Based on the results obtained, provide guidelines as to whether corrective actions should be taken and what these should be.
5. Provide re-evaluation of the reliability-wise performance of the units after the corrective actions are taken to assure that these actions were the correct ones and as effective as intended.
6. Determine the growth in the mean life and/or the reliability of units during their

research, engineering, and development phase, and whether the growth rate is sufficient enough to meet the mean life and/or the reliability need to be demonstrated.

7. Provide a means to statistically and scientifically determine, with chosen risks, whether a re-design has indeed improved the failure rate, mean life, or reliability of components or equipment with the desired confidence, and if not, what corrective actions need to be taken.
8. Provide a statistical and scientific means to determine which one of the two manufacturers of a component or equipment, or which design, should be preferred from the failure rate, mean life, or reliability point of view, all other factors being practically and economically the same, with the desired risk level.

Flehinger [10] has analyzed the reliability of complex systems in which components are used intermittently and which are maintained in operating condition by component replacement. The idea that a failed component causes system failure only when it is called into use is expressed mathematically. Based on component failure distributions and usage properties, the system reliability and expected time to system failure are derived as functions of system age for two different maintenance policies. With both policies, a component is replaced whenever it causes system failure. In the first case, this is the only maintenance, while in the second case, system checkout is conducted at fixed intervals and all components that have failed without causing system failure are replaced. The two policies are compared, and for the second, the dependence of system reliability on the maintenance interval in determined.

Raze [26] describes reliability prediction technique for mechanical equipment that gives a designer the ability to estimate the effects of varying design and operational constraints on a mechanical component's failure rate. The modeling process reduces the component down to its constituent parts and evaluates the effect of parametric changes on each part. This information is then used to develop both part and component failure rates. In addition, a validation-testing program was used to verify both the modeling technique and accuracy of the database information used.

2.2 Maintainability

Goktas [11] developed an integrated reliability and maintainability model to study system design and operation problems. This model helps pick a cost efficient system design within appropriate system design and operating criteria. The following operating policies are used as availability improvement tools:

1. The use of backups at any stage in any number.
2. The use of buffers at any stage with any capacity.
3. The use of preventive maintenance at either component or the system level.
4. Corrective maintenance with any crew size.
5. Combination of the above listed choices.

Jardine [16] described quantitative procedures in the area of equipment maintenance. These procedures are useful in improving the performance of the maintenance function. The specific problem areas examined cover decisions relating to maintenance, replacement and reliability of industrial equipment. Additionally, decisions relating to the facilities required for maintenance, such as manpower requirements and equipment are also considered.

Boukas and Yang [3] present a solution to the problem of controlling the production rate and maintenance rate for a failure prone manufacturing system with one machine and one part so as to minimize the discounted inventory and maintenance cost. In the case where the cost is not dependent on maintenance activity, the optimal policy for maintenance is that the machine should be kept as good as new through maintenance. In the case where cost includes maintenance and if change of the machine age increases the cost significantly, maintenance is done such that the machine age is kept as constant. If the change of the machine age does not affect the cost significantly, no maintenance needs to be done. The optimal policy for production rate is given by a critical surface.

Ushakov [14] presents a series of optimal maintenance models. In the first model, there are repairs only after system failures; the failures are recognized immediately. For this model there is no optimization problem. In the second model, planned maintenance and repairs after system failures restores the system completely; the failures are recognized immediately. In the third model, only planned maintenance is performed. If a failure is found during the procedure, the system is repaired. In the fourth model, repair of a unit in a series system is performed after each failure. In the fifth model, repairs of a unit in a series system are performed after each failure, and the entire system has periodic planned maintenance.

Schouten and Vanneste [27] present optimal group maintenance policies for a set of identical machines subject to stochastic failures. The model presented can be of use to support the maintenance and replacement decisions for the system that are composed of a large number of identical components. In particular, this model focuses on the compromise between individual component replacements and complete system replacements. As such, it might be used to balance technological improvements of a system against the investment cost of such a system. The model contains a major decision variable K, the number of doubtful components that triggers a system replacement. Two classes of policies are considered: A-policy replaces the whole system when a single component fails (or reaches its preventive maintenance age), while the number of doubtful components is greater than or equal to K. The B-policy prescribes a system replacement at the first epoch at which an individual component fails after the first moment at which the number of doubtful components has reached level K. The validation of this model is performed by simulation. As a by-product, this validation reveals that simulation itself is of little use to support the decision process. It took very long simulation runs to obtain confidence intervals of an acceptable width for the average costs and the expected time between system replacements.

2.3 Availability

Dagpunar and Jack [5] established methods for assessing the availability of a system under minimal repair with constant repair times. A new policy and an existing policy for this type are discussed. Each involves replacement at the first failure after time T, with T representing total operating time in the existing model and total elapsed time (i.e., operating time + repair time) in the new model. Optimal values of T are found for both policies over a wide range of parameter values. These results indicate that the new and the administratively easier policy produce only marginally smaller optimal availability values than the existing policy.

Chapter 3: Hardware Reliability Models

3.1 Introduction

Reliability Engineering, as a separate engineering discipline, originated in the United States during the 1950s. The increasing complexity of military electronic systems was generating failure rates, which resulted in generally reduced availability and increased cost. Solid state electronics technology offered long-term hope, but conversely miniaturization was to lead to proportionately greater complexity, which offset the reliability improvements expected.

The rapid pace of electronic device technology meant that the developers of new military systems were making increasing use of large numbers of new component types, involving new manufacturing processes, with the inevitable consequences of low reliability. The users of such equipment were also finding that the problems of diagnosing and repairing the new complex system were seriously affecting its availability for use, and the cost of spares, training and other logistics support were becoming excessive.

Meanwhile the revolution in electronic device technology continued, led by integrated micro-circuitry. Increased emphasis was now placed on improving the quality of devices fitted to production equipment. Screening techniques, in which devices are temperature cycled, vibrated, centrifuged, operated at electrical overstress and otherwise abused, were introduced in place of the traditional sampling techniques. With component populations on even a single printed circuit board becoming so large, sampling no longer provided sufficient protection against the production of defective equipment.

The concept of life cycle costs (LCC), or whole life costs, was introduced. In the United Kingdom, Defense Standard 00-40, the Management of Reliability and Maintainability was issued in 1981. The British Standards Institution has issued BS 5760-Guide on Reliability of systems, equipment and components.

In this chapter a number of terms related to reliability, such as reliability function, expected life, hazard rate, will be defined and one important reliability function will be described. These concepts apply to software and hardware reliability in general.

3.2 Hardware Reliability

Let T be the time representing the life of a system, and t as time less than T, then the probability that the system will fail by time t is given by:

$$F(t) \quad = \quad P[t \le T] = \int_0^t f(t)dt$$

Here, *f(t)* represents the probability (or failure) density function and *F(t)* the cumulative distribution function. The ***reliability function***, i.e., the probability that the system survives until time t, is defined as

$$R(t) \quad = \quad P[T > t] = 1 - F(t) = \int_t^\infty f(t)dt$$

In other words, R(t) represents the probability that the system will not fail by time t assuming it is fault-free at time 0.

The ***expected life E[T]***, or ***Mean Time to Failure (MTTF)*** is simply the mean or the expected value of the failure density function:

$$E[T] \quad = \quad \int_0^\infty t.f(t)dt.$$

It can also be shown that

$E[T] \quad = \quad \int_0^\infty R(t)dt.$ as integration of f(t)dt is equal to R(t).

Mean Time to Repair (MTTR) is the time during which repair or replacement is occurring. Mean Time Between Failures (MTBF) is the sum of MTTF and MTTR.

The ***failure rate*** is the probability that a failure per unit time occurs in an interval such as [t, t+Δt], knowing that a failure has not occurred before t1:

$$\frac{P[t \le T < t + \Delta t \mid T > t]}{t + \Delta t - t} = \frac{P[t \le T < t + \Delta t]}{(\Delta t)P[T > t]} = \frac{F(t + \Delta t) - F(t)}{(\Delta t)R(t)}$$

The ***hazard rate***, on the other hand, is defined as the limit of the failure rate as the interval approaches to zero:

$$h(t) \quad = \quad \lim_{\Delta t \to 0} \frac{F(t + \Delta t) - F(t)}{\Delta t \; R(t)} = \frac{f(t)}{R(t)}$$

So, there is a difference between the hazard rate and failure rate; the hazard rate is an instantaneous rate of failure at time t for a system of age t. The hazard rate changes over the life cycle of a physical system, typically it decreases, remains constant, and then increases with time giving a “bathtub curve” as shown in Figure 3.1. The bathtub curve is the most common model utilized for hardware components. To illustrate how this function is obtained, consider a population of identical components from which we take a large sample N and place it in operation at time T=0. The sample will experience a high failure rate at the beginning of the operation time due to weak or substandard components, manufacturing imperfections, design effects and installation defects. This period of decreasing failure rate is referred to as “Burn-in” region. At the end of burn-in region, the failure rate will eventually reach a constant value. During the constant failure-rate region, the failures do not follow a predictable pattern but occur at random due to the changes in the applied load. This region is called as “useful life” region. The third and final region of the failure-rate curve is the “wear-out” region. The beginning of wear out region is noticed when the failure rate start to increase significantly more than the useful life value, and the failures are no longer attributed to randomness but are due to the age and wear of the components [9].

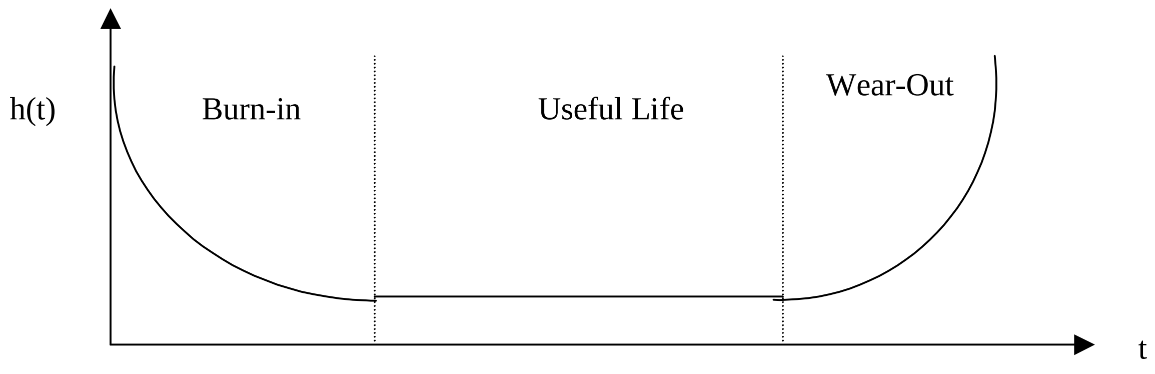


Figure 3.1: The Bathtub Curve

3.3 Types of Distributions

Failure data enable one to fit a model for a failure distribution. Some of the common distributions for hardware components are described below.

3.3.1 Normal Distribution

The Normal distribution is often used as the lifetime distribution, even though it allows negative values (-∞ to +∞) with positive probability. The normal probability density function (PDF) can be used to represent the distribution of the times to failure when the failure mode is wear-out failure mode. The parameters of this model are mean (μ) and standard deviation (σ). The normal PDF is given by:

$$f(t) \quad = \quad \frac{1}{\sqrt{2\Pi\sigma^2}} e^{-\frac{(t-m)^2}{2\sigma^2}} \qquad \begin{pmatrix} -\infty \le t \le \infty \\ -\infty \le m \le \infty \\ 0 \le \sigma^2 \le \infty \end{pmatrix}$$

3.3.2 Exponential Distribution

One of the most common failure distributions in reliability engineering is the exponential or constant failure rate (CFR) model. Failures due to completely random or chance events follow this distribution. This distribution dominates during the useful life of a system or component. It is also one of the easiest distributions to analyze statistically. The parameter of this distribution is failure rate (λ). The probability density function for the exponential distribution is:

$$f(t) \quad = \quad \lambda e^{-\lambda t} \qquad \begin{pmatrix} 0 \le t \le \infty \\ 0 \le \lambda \le \infty \end{pmatrix}$$

3.3.3 Weibull Distribution

Weibull is the most commonly used probability distribution function in reliability. Weibull Distribution can model a wide variety of failure situations. The family of Weibull distributions can be written to include both increasing and decreasing failure rate. It is characterized by a hazard rate function of the form $\lambda(t) \quad = \quad at^b$, which is a power function. The function λ(t) is increasing for a>0, b>0 and is decreasing for a>0, b<0. The parameters of this distribution are shape parameter (β), location parameter (γ) and scale parameter (η). The probability density function for the Weibull distribution is:

$$f(t) = \frac{\beta}{\eta}\left(\frac{t-\gamma}{\eta}\right)^{\beta-1} e^{\left(\frac{t-\gamma}{\eta}\right)^{\beta}} \qquad (\theta > 0, \beta > 0, t \geq 0)$$

3.3.4 Log-Normal Distribution

The positive random variable t has a Log-Normal distribution when logarithm of time is normally distributed. The density of the Log-Normal distribution has important property that is practically zeroed at the origin, increases to a maximum, and then decreases relatively quickly. The Log-Normal function is therefore suitable for modeling repair times. It is also used as a function for failure free operating time components in accelerated reliability testing as well as in cases where a large number of statistically independent random variables are combined together in multiplicative fashion. The parameters of this distribution are mean (t_{med}) and standard deviation (σ). The probability density function is:

$$f(t) = \frac{1}{\sqrt{2\Pi}\sigma t} e^{\left(\frac{-1}{2\sigma^2}\ln\left(\frac{t}{t_{med}}\right)^2\right)} \qquad (t \geq 0)$$

3.4 Data Analysis

Failure data can be complete or censored. Here we will discuss singly censored data.

3.4.1 Singly Censored Data

A common problem in generating reliability data is censoring. Censoring occurs when data is incomplete because components are removed from consideration prior to their failure or because the test is completed prior to all components failing. That is, all components have the same test time, and the test is concluded before all components have failed. Components may be removed, for example, when they fail because of other failure modes than the one being measured. The singly censored data is further classified into ***Type I*** Censoring and ***Type II*** Censoring. In Type I censoring the testing is terminated after a fixed length of time has elapsed. Whereas in Type II censoring testing is terminated after a fixed number of failures occur.

Under each distribution, least squares (L-S) estimates, correlation coefficient, T-statistic, Maximum Likelihood Estimators (MLE) to determine reliability parameters, and goodness of fit to test whether data fits in a particular distribution or not. Brief discussions of these are as follows:

3.4.2 Least Square Estimates

The first order linear model is given by:

$$Y = \beta_0 + \beta_1 X_i + \varepsilon_i$$

Where β_0 and β_1 are the intercept and slope respectively and ε_i is the error term. The predicted value of Y is:

$$Y' = b_0 + b_1 X$$

Where b_0 and b_1 are predicted parameters. If Y' is the predicted value of Y for a given value of X, then the residual is given by Y- Y'. The method of least squares chooses the regression line $Y' = b_0 + b_1 X$ that minimizes the sum of squares of residual $\sum(Y - Y')^2$ for all sample points.

3.4.3 Correlation

The primary purpose of linear correlation analysis is to measure the strength of a linear relationship between two variables. It is computed (for a sample of n measurements on X and Y) as follows:

$$cor = \frac{SS_{xy}}{\sqrt{SS_x SS_y}} = \frac{Cov(x,y)}{\sigma_x . \sigma_y}$$

Where,

$$SS_x = \sum x^2 - \frac{\left(\sum x\right)^2}{n}$$ (Sum of Squares of x)

$$SS_y = \sum y^2 - \frac{\left(\sum y\right)^2}{n}$$ (Sum of Squares of y)

$$SS_{xy} = \sum xy - \frac{\left(\sum x\right)\left(\sum y\right)}{n}$$ (Sum of Squares of xy)

3.4.4 T-Statistic

The T-Statistic is used to test for the existence of slope. The hypothesis is

H_0: $\beta_1 = 0$

H_A: $\beta_1 \neq 0$

Where β_1 is the slope.

The test is given by:

$$TS = \frac{b_1}{\left(\frac{Std.\,Error}{\sqrt{SS_x}}\right)}$$ (b_1 – Slope of regression line)

Where,

$$Std.\,Error = \sqrt{\frac{SS_y - b_1(SS_{xy})}{df}}$$ (df – degrees of freedom)

Once the TS is calculated, the hypothesis is either accepted or rejected depending on the significance level [1].

3.4.5 Maximum Likelihood Estimators (MLE)

The method of maximum likelihood is a general method of finding estimators. Suppose we are sampling failure times whose probability function of the failure times is given by f(t; θ_1,……., θ_k). Given independent values t_1,…….,t_k, the joint probability function of the failure times is given by:

$$h(t_1,.........., t_n) = \prod_{i=1}^{n} f(t_i;\theta_1,........,\theta_k)$$

When this joint probability function is viewed as a function of θ, with the values given, it is called the likelihood function L (θ). To find the MLE for any parameter with complete data, the maximum of the following likelihood function with respect to unknown parameters θ_1,…….,θ_k is found:

$$L(\theta_1,.........,\theta_k) = \prod_{i=1}^{n} f(t_i;\theta_1,............,\theta_k)$$

With singly censored data (Type I or Type II), the likelihood function is modified to be

$$L(\theta_1,.........,\theta_k) = \prod_{i=1}^{r} f(t_i;\theta_1,............,\theta_k)[R(tt)]^{n-r}$$

where r is the number of failures and n is the number at risk. The factor $[R(tt)]^{n-r}$ is the probability that the n-r censored units do not fail prior to the termination of the test.

When multiple censored data are present, the likelihood function must be modified to reflect the fact that at the censored times no failures occurred. This can be accomplished by defining the likelihood function in the following manner:

$$L(\theta) \;=\; \prod_{i \in F} f(t_i;\theta) \prod_{i \in C} R(t_i^{+};\theta)$$

where θ is the unknown parameter, F is the set of indices for the failure times, and C is the set of indices for the censored times and t^+ is a censored time. For any types of data, the objective is to find the values of the estimators of $\theta_1,\ldots\ldots,\theta_k$ that render the likelihood function as large as possible for given values of $t_1,\ldots\ldots,t_n$.

3.4.6 Goodness-Of-Fit Tests

Goodness-Of-Fit tests compare a null hypothesis with an alternative hypothesis:

H_0: The failure times came from the specified distribution.

H_A: The failure times did not come from the specified distribution.

The test consists of computing a statistic based on the sample of failure times. This statistic is then compared with a critical value. The critical value depends on the level of significance of the test and the sample size. For this research, three goodness of fit tests will be used. These three tests are designed for specific distributions. For instance, Kolmogorov-Smirnov test is designed for normal and lognormal distributions, Bartlett's test is designed for exponential distribution, and Mann's test is designed for the Weibull distribution. The tests are described in Appendix I, II and III. The general goodness of fit test can be performed by chi- square goodness-of-fit test. This can be shown as:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

where k = number of classes, O_i = observed number of failures and E_i = expected number of failures.

Chapter 4: Software Reliability

4.1 Introduction

Measurement is a vital element in the practice of engineering, with reliability being one of the most important measurements. An understanding of the failure process is central to any effort to model and comprehend reliability.

Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. It is one of the key attributes of software quality, a multidimensional property including other factors like functionality, usability, performance, serviceability, capability, instability, maintainability and documentation [12]. Software reliability is the key factor in software quality since it quantifies software failures, which can make a powerful system inoperative or even deadly, making it an essential ingredient in customer satisfaction.

Software reliability represents a user- (or customer-) oriented view of software quality. It relates directly to operation rather than design of the program, and hence it is dynamic rather than static. Therefore it becomes apparent that the distinction between terms “failure” and “fault” is important. For this reason software reliability is interested in failures occurring and not faults in program. Reliability measures are much more useful than fault measures. Software reliability may be expected to vary during the software development period.

A **Failure** occurs when the user perceives that the program ceases to deliver the expected service. In other words, failure is something dynamic, i.e., occurring at execution time. It is not a bug or fault. It is more general. For example, excessive response time may be considered as failure if it does not meet the specifications.

A **Fault** is uncovered when either a failure of the program occurs or an internal error (e.g., an incorrect state) is detected within the program. The cause of failure or internal error is said to be a fault. It is also referred to as a bug. A fault may result from an error made by the programmer. Errors occur because of (a) lack of communication between the people involved in a project or between different times for the same person; (b) incomplete knowledge of the application area, the design methodology, and the programming language; (c) incomplete analysis of the possible conditions that can occur at a given point in the program; and (d) transcription errors.

A **random process** is represented as a set of random variables, each corresponding to a point in time. In reliability studies there are two characteristics of a random process: (a) the probability distribution of the random variables, e.g., Poisson; and (b) the variation of the process with time. A random process whose probability distribution varies with time is called non-homogeneous. Two functions can be defined for the time variation of a random process: (a) the mean value function, μ, as the average cumulative failures associated with each time point; and (b) the failure intensity function, λ, as the rate of change of mean value function or the number of failures per unit time. Note that the failure intensity function is the derivative of the mean value function.

The first study on software reliability appears to have been conducted by Hudson[13]. He reviewed software development as a birth and death process. Where Fault generation being a birth and fault correction being a death. He generally confined his work to pure death process, for reasons of mathematical tractability. The number of faults detected follows binomial distribution whose mean value function of time has a Weibull distribution. Data from the system test phase of one program was presented. Reasonable agreement between model and data was obtained if the system test phase is spilt into three overlapping sub-phases and separate fits made for each.

The next major steps were made by Jelinski and Moranda[15] and Shooman[28]. Both assumed hazard rate for failures to be piecewise constant and proportional to the number of faults remaining. The hazard rate changes at each fault correction by constant amount but is constant between corrections. Shooman[28] postulated that hazard rate was proportional to fault density per instruction, the number of unique instructions executed per unit time, and a bulk constant. The bulk constant represented the proportion of faults that cause failure.

Schick and Wolverton[29] proposed another early model. The hazard rate assumed was proportional to the product of the number of faults remaining and the time. Hence, the size of the changes in hazard rate increase with time. Wagoner[34] suggested a model in which the hazard rate was proportional to the number of faults remaining and power of time. This power could be varied to fit the data. In 1978 Schich and Wolverton[30] proposed modified model in which hazard rate is a parabolic function instead of linear function of time. Schneidewind[31] initially approached software reliability modeling from empirical viewpoint. He recommended the investigation of different reliability functions and selection of the distribution that best fit the particular project in question. In a later paper, Schneidewind[32] viewed fault detection per time interval as a Non-Homogeneous Poison Process (NHPP) with an exponential mean value function. He also suggested that the time lag between failure detection and failure correction be determined from actual data and used to correct the time scale in forecast.

Moranda[18] proposed two variants of the Jelinski – Moranda model. In "geometric de-eutrophication process", the hazard rate decreases in steps that form a geometric progression. The second, called the Geometric Poisson Model (GPM), has a hazard rate that also decreases in geometric progression. However, the decrements occur at fixed intervals rather that at each failure correction.

A Bayesian approach to software reliability measurement was taken by Littlewood and Verall[17]. Almost all published models assume that failures occur randomly during the operation of program. However, while most postulate simply that the value of hazard rate is a function of number of faults remaining, Littlewood and Verrall modeled it as a random variable. One of the parameters of the distribution of this random variable is assumed to vary with the number of failures experienced. It thus characterizes reliability change. Littlewood and Verrall proposed various functional forms for the description of this variation. The values of the parameters of each functional form that produce best fit for that form is determined. Then functional forms are compared and the best fitting form is selected.

It is seen that much of the early history of software reliability modeling involved looking at different possible models. In the late 70's and 80's, efforts began to focus on comparing software reliability models, with the objective of selecting the best ones. The initial efforts at comparison suffered from lack of good failure data and lack of agreement on the criteria to be used in making comparisons. Examination of the basic concepts underlying software reliability modeling and development of classification scheme has helped to clarify and organize comparisons and to suggest possible new models. This work led to the development of Musa-Okumoto[20] Logarithmic-Poisson execution time model, which combines simplicity, with high predictive validity. The Logarithmic Poisson model is based on NHPP with an intensity function that decreases exponentially with expected failures experienced.

In late 90's efforts began to focus on tools available for reliability estimation and to handle failure data. Yuan [35] presents a study which discussed notion of discretetime software reliability modeling and distinguishing of types of software run reliability data.

4.2 Software Reliability Estimation

Software reliability estimation is the application of statistical inference procedures to failure data taken from software testing and operation to determine software reliability. There are four major components in the software reliability estimation process, namely

i. Reliability objective

ii. Operational profile.

iii. Reliability modeling and measurement.

iv. Validation.

At first quality of software is quantitatively defined from customers' point of view by defining failures and failure severity, by determining a reliability objective and specifying balance among key quality objectives (e.g., reliability delivery cost). Second customer usage is quantified by developing an operational profile. Operational profile is set of disjoint alternatives of system operation and their associated probabilities of occurrence. The construction of operational profile encourages testers to select test cases according to the systems operational usage, which contributes to more estimation of software reliability in the field [1]. The entire framework of software reliability estimation is shown in Figure 4.1.

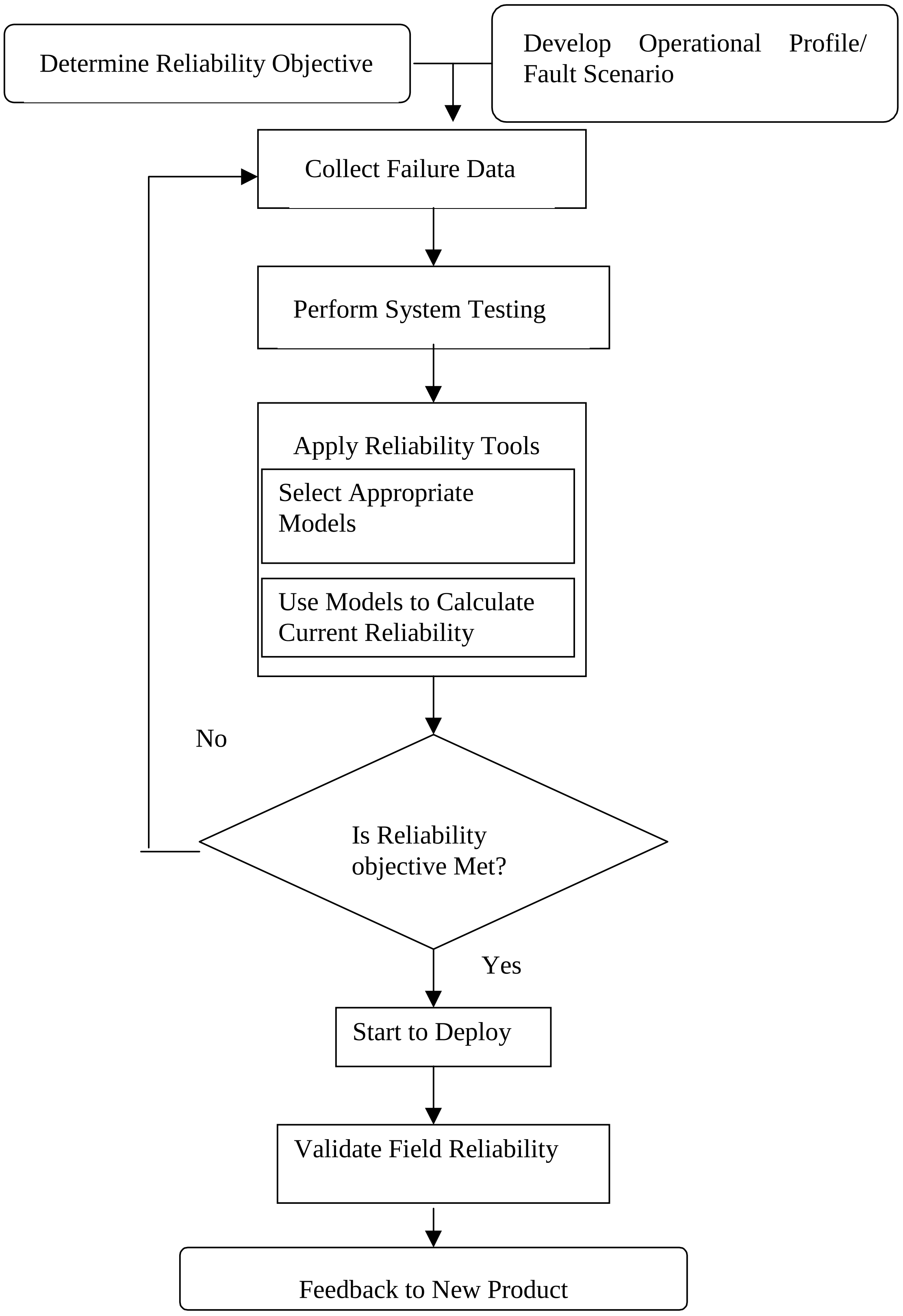

Determine Reliability Objective
Develop Operational Profile/ Fault Scenario
Collect Failure Data
Perform System Testing
Apply Reliability Tools
Select Appropriate Models
Use Models to Calculate Current Reliability
No
Is Reliability objective Met?
Yes
Start to Deploy
Validate Field Reliability
Feedback to New Product

Figure 4.1: Software Reliability Measurement Procedure Overview

Reliability quantities have usually been defined with respect to time, although it is possible to define them with respect to other variables. Time may be considered as:

a) Execution time (τ), i.e., CPU time;

b) Calendar time; and

c) Clock time, i.e., the sum of times passed from program start to program end, without counting shut shown periods.

Execution time is considered superior to calendar time [19].

There are four general ways of characterizing failure occurrences in time:

$$\left.\begin{array}{ll}\text{1.} & \text{Time of failure} \\ \text{2.} & \text{Time interval between failures (incremental)}\end{array}\right\} \text{Time-based}$$

$$\left.\begin{array}{ll}\text{3.} & \text{Cumulative failures experienced up to given time.} \\ \text{4.} & \text{Failures experienced in a time interval.}\end{array}\right\} \text{Failure-based}$$

The above four quantities are in fact *random variables*, because, a) the locations of faults within a program are unknown; and b) the conditions of program execution are generally unpredictable

When there are no changes in the software, i.e., no debugging and software corrections take place, then failure rate (λ) is constant and a homogeneous random process takes place. On the contrary, when software corrections occur, a non-homogeneous process as described before takes place. Figure 4.2 illustrates the mean value and related failure intensity functions of such a process. These graphs are typical in the sense that the mean number of failures experienced increases with time in such a manner that the failure intensity decreases.

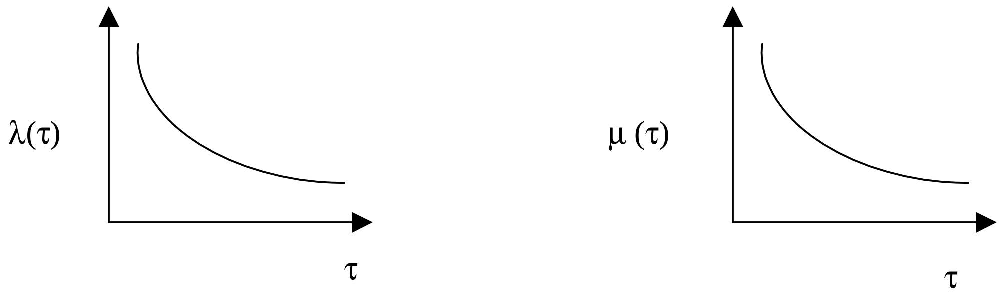


Figure 4.2: Failure Intensity Curve and Mean Value Function for a Non-Homogeneous Process

This behavior is affected by two factors: (a) the number of faults in the software; and (b) the execution environment.

Let M(τ) be a random process representing the number of failures experienced by time τ, then the mean value function is defined as

$$\mu(\tau) = E[M(\tau)],$$

i.e., expected number of failures at time τ. The failure intensity function of the M(τ) processes is the instantaneous rate of change of the expected number of failures with respect to time, or

$$\lambda(\tau) = \frac{d\mu(\tau)}{d\tau}$$

The "time" used here may be any one of the above-mentioned three times, but execution time is generally preferred in order to be compatible with hardware reliability.

Principal factors affecting software reliability are fault introduction, fault removal, and the environment [19]. Fault introduction depends on the characteristics of the code developed, i.e., created or modified – such as its size – and of the development process – such as software engineering technology and tools used and level of experience of programmers. Fault removal depends on time, operational profile, and the quality of repair activity. Environment is determined by the operational profile, which is the set of run types that a program can execute along with the probabilities with which they will occur. It is generally established by enumerating the possible input states and their probabilities of occurrence or by specifying the sequence of program modules executed.

As faults are removed, as in the test phase, failure intensity tends to decrease and reliability tends to increase. When faults are introduced during operation or test, as in cases when new features or design changes are being introduced into the system or when faults predominate repairs during debugging, there tends to be a step increase in failure intensity and a step decrease in reliability. If a system is stable, and in a program that has been released, there are no changes in code, both failure intensity and reliability tend to be constant.

The term Mean Time To Failure (MTTF), which means the average value of next failure interval, is not used as extensively in software reliability as in hardware reliability, since in many cases it is undefined. Instead, failure intensity, which is the inverse of MTTF, is preferred.

4.3 Finite Failure Category Models

The finite failure category models have fixed number of faults. The fault concept is often useful for developing finite category models because of the physical reality of defects in program. The models are classified according to how the failure quantity distribution is specified. Poisson and binomial are the most important model in this group. Binomial type models have deterministic number of faults and failures and one fault is removed for each failure. Poisson type models have random number of failures and the number of faults removed for each failure is a random variable. The models under this category are:

1. Schick-Wolverton model: Here the failure distribution type is Weibull distribution.
2. Basic Musa model: Here the failure distribution is of exponential type.
3. S-Shaped model: In this model failure distribution follows Gamma distribution.

4.3.1 Schick-Wolverton Model

In this model failure distribution follows Weibull distribution. This model belongs to finite failures category and is of the binomial type. This model has greater flexibility for the failure modeling because of the shape and scale parameters that define them. Basic assumptions are as follows:

- There are a fixed number of faults in the software.
- The time to failure of fault b0 (location parameter) is distributed as a Weibull distribution with parameter b1 and b2 (shape and scale parameter). The density function is $f(\tau) = b0 * b1 * \tau^{b2-1} * e^{-b1*\tau^{b2}}$ with b1, b2 > 0. For Schick Wolverton model value of b2 is 2.
- The number of faults (f_1,f_2,…., f_n) detected in each of the respective intervals [(t_0=0, t_1), (t_1, t_2)… (t_{i-1}, t_i)… (t_{n-1}, t_n)] are independent for any finite collection of times.

The data requirements to implement this fault count model are:

- The fault counts in each of the testing intervals i.e., the f_I, and the completion time of each period that the software is under observation i.e., t_i's.

The failure intensity and mean value function is given by:

$$\lambda(\tau) = b0 * b1 * \tau^{b2-1} * e^{-b1*\tau^{b2}} \qquad \mu(\tau) = b0 * (1 - e^{-b1*\tau^{b2}})$$

4.3.2 Basic Musa Model

The Basic Musa model has the widest distribution among the software reliability models and was developed by John Musa of AT&T Bell Laboratories [19]. Musa has been a leading contributor in this field and has been a major proponent of using models to aid in determining the reliability of software. This model was one of the first to use the actual execution time of the software component on a computer for the modeling process. The basic assumptions and the data requirements for the model are as follows:

1. The cumulative number of failures by time τ, $\mu(\tau)$, follows a Poisson process with mean value function $\mu(\tau) = b0 * (1-e^{-b1*\tau})$, where, b0, b1>0 mean value function is such that the expected number of failure occurrences for any time period is proportional to the expected number of undetected faults at that time. Since $\lim_{\tau\to\infty} \mu(\tau) \;=\; \lim_{\tau\to\infty} b0*(1-e^{-b1*\tau})$ is a finite failure model. The parameter b0 is the total number of faults that would be detected in that limit.
2. The execution times between failures are piecewise exponentially distributed, i.e., the hazard rate for a single fault is constant. This is why this model belongs to the exponential class.
3. The quantities of the resources (number of fault-identification, correction personnel and computer times) that are available are constant over a segment for which the software is observed.
4. Fault-identification personnel are fully utilized and computer utilization is constant.
5. Fault-correction personnel utilization is established by the limitation of fault queue length for any fault-correction person. Fault queue is determined by assuming that fault correction is a Poisson process and that servers are randomly assigned in time.

Assumptions 3 through 5 are needed only if the second component of the basic execution model linking execution time and calendar time is desired.

Data requirements to implement this fault count model is either the actual times that the software failed, $t_1, t_2, \ldots, t_n$ or the elapsed time between failures $x_1, x_2, \ldots, x_n$, where $x_i = t_i - t_{i-1}$.

The failure intensity and the mean value function are given as:

$$\lambda(\tau) \;=\; b0*b1*e^{-b1*\tau} \qquad\qquad \mu(\tau) \;=\; b0*(1-e^{-b1*\tau})$$

4.3.3 S-Shaped Model

The S-Shaped reliability growth model falls under the gamma distribution class. Here the per-fault failure distribution is gamma. The number of failures per time period is a Poisson type. It is finite model, i.e., $\lim_{\tau \to \infty} \mu(\tau) < \infty$. It is patterned, as the mean value function is often a characteristic S-shaped. The software error detection process can be described as an S-shaped growth curve to reflect the initial learning curve at the beginning, as test team members become familiar with the software, followed by growth and then leveling off as the residual faults become more difficult to uncover. The basic assumptions and the data requirement for the model is given as:

1. The cumulative number of failures by time τ, $\mu(\tau)$, follows a Poisson process with mean value function $\mu(\tau) = b0 * (1-((1+b1*\tau)*e^{-b1*\tau}))$, where, b0, b1>0. This is a bounded, non-decreasing function of time with limit $\lim_{\tau \to \infty} \mu(\tau) = b0$, which is less than infinity, i.e., it is a finite failure model.
2. The time between failures of the (i-1)th and the ith failure depends on the time to failure of the (i-1)th.
3. When a failure occurs, the fault, which caused it, is immediately removed and no other faults are introduced.

Data requirements to implement this model are:

- The failure times, t_i's, of the software system, or
- The number of faults detected, f, in each period of observation of the software along with the associated lengths l_i of those periods, i = 1,… n.

If data of type I are available, the data of second type can be constructed by first forming a partition of the time period over which the software is observed and then counting up the number of faults that fall in each respective period of partition. As a consequence, this model can be used for either the time between failure data or the number of faults per time period.

The failure intensity and mean value function is given by:

$$\lambda(\tau) = b0 * b1^2 * \tau * e^{-b1*\tau} \qquad \mu(\tau) = b0 * (1-((1+b1*\tau)*e^{-b1*\tau}))$$

4.4 Infinite Category Model

For infinite failures category models the number of failures in infinite time is unbounded. That is for these models $\lim_{\tau \to \infty} \mu(\tau) = \infty$ for the mean value function of the process. This means that software will never be completely fault free. This could be by additional faults being introduced in the software through the error correction process. This category includes following three models:

1. Duane's Model: This model assumes power distribution for the failure data.
2. Logarithmic Poisson Model: This model assumes geometric distribution for the failure data.
3. Inverse Linear Model: This model assumes inverse linear distribution for the failure data.

4.4.1 Duane's Model

At General Electric, Duane noticed that if the cumulative failure rate versus the cumulative testing time was plotted on Log-Log paper, it tended to follow a straight line [6]. This process is a non-homogeneous Poisson process in which the failure intensity function has the same form as the hazard rate for a Weibull distribution. This model is sometimes referred to as the power model since the mean value function for the cumulative number of failures by time τ is taken as a power of τ, that is, $\mu(\tau) = b0*\tau^{b1}$ for some, b0>0 and b1>0. This model is an infinite failure model as $\lim_{\tau \to \infty} \mu(\tau) = \infty$. The basic assumptions and data requirements are:

1. The cumulative number of failures by time t, M(t), follows a Poisson process with mean value function, $\mu(\tau) = b0*\tau^{b1}$ for some, b0>0 and b1>0.

The data requirements to implement this fault count model are:

- Either the actual times that the software failed, $t_1, t_2, \ldots, t_n$ or the elapsed time between failures $x_1, x_2, \ldots, x_n$, where $x_i = t_i - t_{i-1}$ and $t_0 = 0$.

The failure intensity and mean value functions are:

$$\lambda(\tau) = b0 * b1 * \tau^{b1-1} \qquad \mu(\tau) = b0 * \tau^{b1}$$

4.4.2 Logarithmic Poisson Model

The logarithmic Poisson model proposed by Musa and Okumoto [20] is another model that has been extensively applied. It is also a NHPP with an intensity function that decreases exponentially as failures occur. The exponential rate of decrease reflects the view that the earlier discovered failures have a greater impact on reducing the failure intensity function than those encountered later. It is called logarithmic because the expected number of failures over time is a logarithmic function. The assumptions and data requirements are:

1. The failure intensity decreases exponentially with the expected number of failures experienced, that is, $\lambda(\tau) = \frac{b0*b1}{1+b1*\tau}$.
2. The cumulative number of failures by time τ, $\mu(\tau)$ follows a Poisson process.

Because of assumption 1, it follows that $\mu(\tau) = b0*\ln(1+b1*\tau)$ and therefore it is clear that this is an infinite failure model.

Data requirements are:

- Either the actual times that the software failed, $t_1, t_2, \ldots, t_n$ or the elapsed time between failures $x_1, x_2, \ldots, x_n$, where $x_i = t_i - t_{i-1}$.

The failure intensity and mean value function are as:

$$\lambda(\tau) = \frac{b0*b1}{1+b1*\tau} \qquad \mu(\tau) = b0*\ln(1+b1*\tau)$$

4.4.3 Inverse Linear Model

This model is a special case of Littlewood Verrall Model [17]. This is also a type of infinite failure model as $\lim_{\tau\to\infty}\mu(\tau) = \infty$. Data requirements for the model is:

Either the actual times that the software failed, $t_1, t_2, \ldots, t_n$ or the elapsed time between failures $x_1, x_2, \ldots, x_n$, where $x_i = t_i - t_{i-1}$.

The failure intensity and mean value is given by:

$$\lambda(\tau) = b0*\frac{1}{2*\sqrt{b1+\tau}} \qquad \mu(\tau) = b0*(\sqrt{b1+\tau}-\sqrt{b1}$$

Chapter 5: System Reliability

5.1 Introduction

To determine the reliability of a large system, it needs to be subdivided into smaller subsystems and components whose individual reliability factors are known or can be easily determined. Depending on the manner in which these subsystems and components are connected to constitute the given system, the combination rules of probability can be applied to obtain system reliability. From the point of view of interconnection of the subsystems, a system may be classified as series, parallel, series-parallel, or a complex system.

Finding the exact reliability for series and parallel networks is quite straightforward and is described briefly in next two sections. A series-parallel network consists of distinct series and parallel components within the given system. For such a system the reliability analysis is performed in steps as described below.

A complex system is one, which cannot be completely decomposed into independent sections of series and/or parallel sub-systems. Reliability analysis for such systems is non-trivial. As a result, many algorithms resort to simplifying assumptions that produce approximate solutions [7].

5.2 Series System

Consider a simple system consisting of n software or hardware components connected in series as shown in the Figure 5.1.

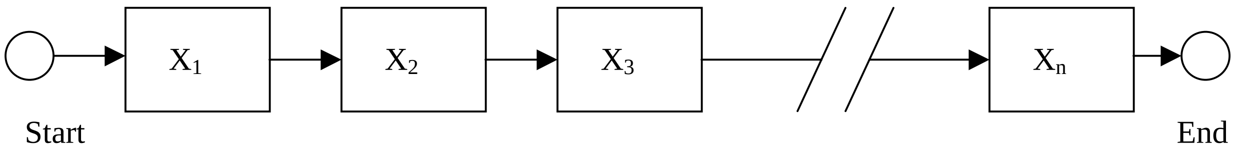


Figure 5.1: Series System

Let $P(X_i)$ represent the probability of successful operation of component X_i. For this series system the system reliability is given by:

$R_s = P(X_1 * X_2 * \ldots X_n)$

$P(X_1) * P(X_2 \mid X_1) * P(X_3 \mid (X_1 * X_2)) \ldots P(X_n \mid (X_1 * X_2 * \ldots X_{n-1}))$

Where $P(X_2 \mid X_1)$ means the probability of successful operation of component X_2

under the condition that component X_1 operates successfully. Likewise, P (X_n | X_1 and X_2 and … X_{n-1})) means the probability of successful operation of component X_n under the condition that all remaining units are working successfully. If the successful operations of all these units are independent, the above expression becomes:

$$R_s = \prod_{i=1}^{n} P(X_i)$$

5.3 Parallel Systems

Consider another simple system consisting of n components connected in parallel as shown in Figure 5.2.

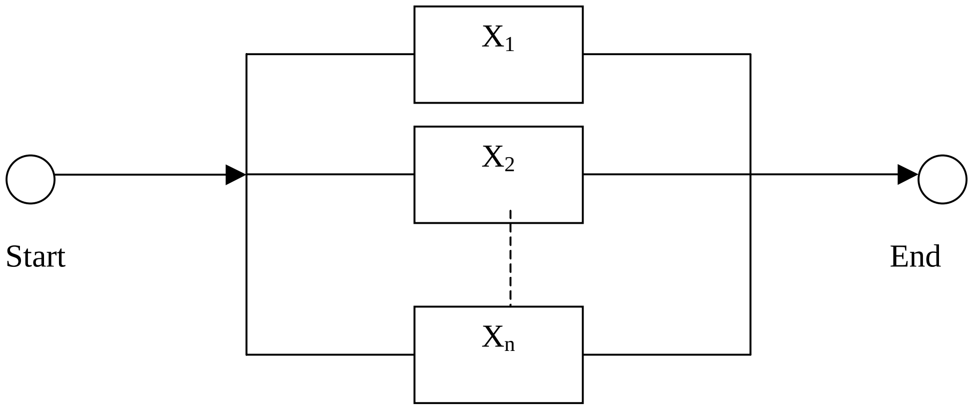


Figure 5.2: Parallel System

In parallel systems, as opposed to series system, several single path perform the same operation. The satisfactory performance of any of these paths is sufficient to ensure the successful operation of the system. Let P (Xi) represent the probability of successful operation of component Xi. Similarly let P (Xi') represent the probability of failure of component Xi. For this parallel system to fail, all of the n components must fail. Hence the unreliability of the system is given by:

Q_s = P (X'_1 * X'_2 * … X'_n)

= P(X'_1) * P(X'_2 | X'_1) * P(X'_3 | (X'_1 * X'_2))…P (X'_n | (X'_1 * X'_2 * … X'_{n-1}))

Where P(X'_2 | X'_1) means the probability of failure of unit X_2 under the condition that unit X_1 also fails. Likewise, P (X'_n | (X'_1 * X'_2 * … X'_{n-1})) means the probability of failure of unit X_n under the condition that all remaining units have failed. If the failures of all these units are independent, then

$$Q_s = \prod_{i=1}^{n} P(X'_i)$$

The system reliability, R_s, is given by

$$R_s = 1 - \prod_{i=1}^{n} P(X'_i)$$

5.4 Series-Parallel System

An example of a series-parallel system is shown in Figure 5.3.

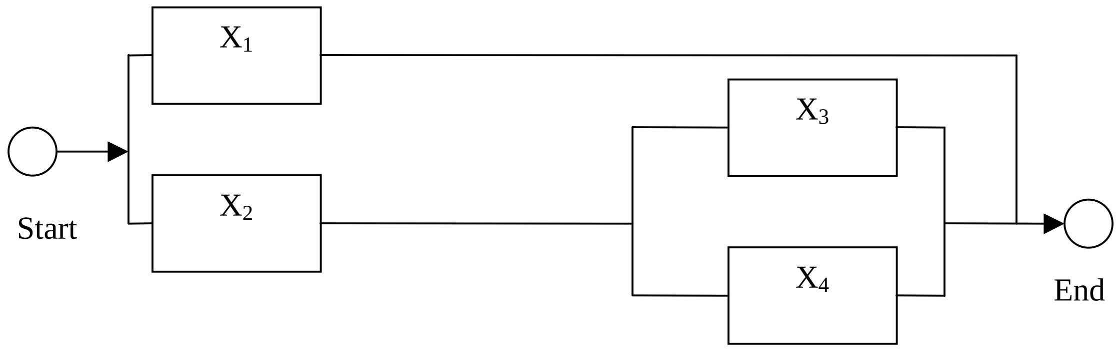


Figure 5.3: Series-Parallel System

Reliability analysis of such system is performed in steps. In each step the independent series and parallel structures are identified and solved separately. As a result of each step, the size of the system reduces until it becomes a simple series or parallel system. In the system shown in Figure 5.3, components X_3 and X_4 are in parallel and thus form a subsystem identified as subsystem1 and shown in Figure 5.4.

Using the expression derived for parallel systems, and assuming unit failures as independent events, the reliability of subsystem 1 is found as:

Rsubsys1 = 1-[1-P(X_3)]*[1-P(X_4)]

In the next step, note that unit X_2 is in series with the subsystem 1. Let the unit X_2 and subsystem 1 form a bigger subsystem, which is identified as subsystem 2 and shown in Figure 5.5. Again, assuming the failure of each unit as an independent event, the reliability of subsystem 2 is given by:

Rsubsys2 = $P(X_2)*[1-\{1-P(X_3)\}*\{1-P(X_4))]$

Finally X_1 can be combined in parallel with subsystem 2. The reliability of the overall system is:

$R_s = 1- P(X_1)*\{\text{Probability of Failure of Subsystem 2}\}$

$R_s = 1-\{1-P(X_1)\}*[1-P(X_2)*\{1-(1-P(X_3)\}*\{1-P(X_4)\}]$

Assuming that all units are identical, i.e. $P(X_1) = P(X_2) = P(X_3) = P(X_4) = p$ and $P(X'_1) = P(X'_2) = P(X'_3) = P(X'_4) = 1-p$,

$R_s = p+2p_2+3p_3+p_4$

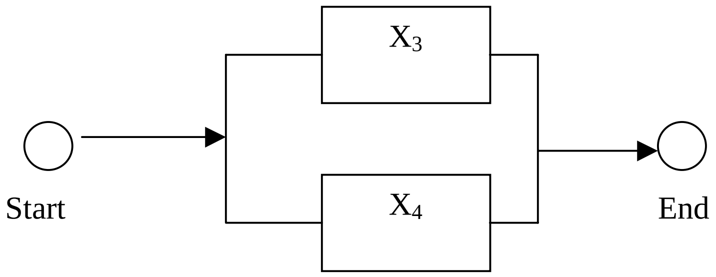


Figure 5.4: Subsystem 1

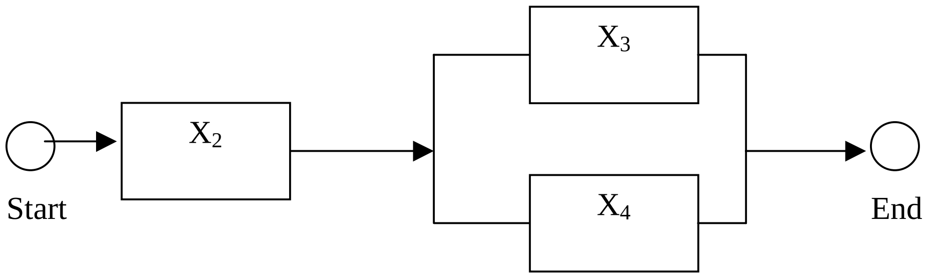


Figure 5.5: Subsystem 2

5.5 k-out-of-n System

The k-out-of-n system has a total of n components connected in parallel, and at least k components must operate for the system to function. Obviously $k \leq n$. if $k = 1$, complete redundancy occurs, and if $k = n$ then we have an n component series system. The reliability for such a system can be obtained from the binomial probability distribution.

$$R_s = \sum_{j=k}^{n} \left(\frac{n!}{j!(n-j)!} \right) * R^j * (1-R)^{n-j}$$

where R is the component reliability.

5.6 State Dependent Systems

A fundamental computation in reliability engineering is the determination of system reliability from knowledge of component reliabilities and their system configuration. On the basis of the critical assumption of independent failures among components the basic equations of system reliability can be derived. However, when component failures are in some way dependent, more powerful methods, such as Markov analysis is needed. For example, when one of the components connected in parallel fails, the failure rates of the surviving components are affected [9].

5.6.1 Markov Analysis

Markov analysis looks at a system as being in one of several states. One possible state, for example, is that in which all the components of the system are operating. Another possible state is that in which one component has failed but the other components continue to operate. The fundamental assumption in a Markov process is that the probability that a system will undergo a transition from one state to another depends only on the current state of the system and not on any previous state of the system. In other words, transition probability is not dependent on the past (state) history of the system. This is equivalent to the memorylessness of the exponential distribution, and it is therefore not surprising that exponential times to failure satisfy this Markovian property. The state transition probability is expressed as an instantaneous (failure) rate. Assuming the process is also stationary (i.e., the transition probabilities do not change over time), the transition rates will be constant. Again, this is equivalent to assuming exponential failure times.

The methodology is explained by an example. In applying Markov analysis, we assume that each of the n components will be in one of the two states – operating or failed. The system state is then defined to be one of the 2^n possible combinations of operating and failed components. For a two-component system we define the following four system states:

Table 5.1: State Table for Two Components

State	**Component X_1**	**Component X_2**
1	X_1 (Operating)	X_2 (Operating)
2	X_1' (Failed)	X_2 (Operating)
3	X_1 (Operating)	X_2' (Failed)
4	X_1' (Failed)	X_2' (Failed)

If the two components are in parallel (redundant), only state 4 results in a system failure. On the other hand, if the two components are in series, then states 2, 3 and 4 would constitute a failure state. The objective is to find the probability of the system being in each state as a function of time. We denote the probability of being in state 1 at time t as $P_i(t)$. Then for a two-component series system,

$R_s(t) = P_1(t)$

And for two-component parallel system,

$R_p(t) = P_1(t) + P_2(t) + P_3(t)$

Where $R_s(t)$ and $R_p(t)$ is reliability of series and parallel system at time t.

Observe that the system must be in one of the four sates at any given time. Therefore,

$P_1(t) + P_2(t) + P_3(t) + P_4(t) = 1$ (5.1)

What remains is to find $P_i(t)$, i = 1,2,3,4.

If we assume that individual components have constant failure rates λ_i, we can represent the two-component system using the Reliability Transition Diagram (RTD) as shown in Figure 5.6. The nodes in the figure represent the four system states, and the branches show the transition rate (λ_i) from one node to another.

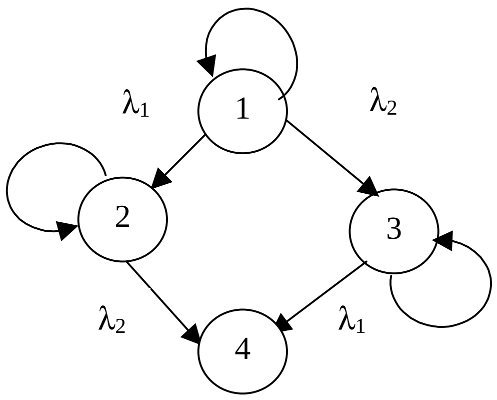


Figure 5.6: Reliability Transition Diagram

From the RTD, we can derive the following equation:

$P_1(t+\Delta t) = P_1(t) - \lambda_1 \Delta t P_1(t) - \lambda_2 \Delta t P_1(t)$ (5.2)

This equation states that the probability of the system being in state 1 at time t + Δt is equal to the probability of it being in state 1 at time t minus the probability of it being in state 1 at time t times the probability of transitioning ($\lambda_i\Delta t$) to either state 2 or 3. Observe that $\lambda_1\Delta t$ is the conditional probability of a transition to state 2 occurring during the time Δt given that the system is currently in state 1. Therefore, $\lambda_1\Delta t P_1(t)$ is the joint probability of the system being in state 1 at time t and making a transition to state 2 during time Δt. A similar argument holds for $\lambda_2\Delta t P_1(t)$ for transition to state 3.

A second equation is obtained from state 2.

$P_2(t+\Delta t) = P_2(t) + \lambda_1 \Delta t P_1(t) - \lambda_2 \Delta t P_2(t)$ (5.3)

is the probability of the system being in state 2 at time t + Δt and is equal to the probability of being in state 2 at time t plus the probability of being in state 1 at time t and making a transition ($\lambda_1\Delta t$) to state 2 in time Δt minus the probability of being in state 2 at time t and making a transition to state 4 ($\lambda_2\Delta t$) in time Δt. Similarly for state 3,

$P_3(t+\Delta t) = P_3(t) + \lambda_2 \Delta t P_1(t) - \lambda_1 \Delta t P_3(t)$ (5.4)

And for state 4,

$P_4(t+\Delta t) = P_4(t) + \lambda_2 \Delta t P_2(t) + \lambda_1 \Delta t P_3(t)$ (5.5)

Rewriting Equation 5.2,

$$\frac{P_1(t+\Delta t) - P_1(t)}{\Delta t} = -(\lambda_1 + \lambda_2)P_1(t)$$

Then

$$\lim_{\Delta t \to 0} \frac{P_1(t+\Delta t) - P_1(t)}{\Delta t} \quad = \quad \frac{dP_1(t)}{dt} \quad = \quad -(\lambda_1 + \lambda_2)P_1(t) \qquad (5.6)$$

In a similar fashion, Eqs. (5.3) and (5.4) lead to the following differential equations:

$$\frac{dP_2(t)}{dt} = \lambda_1 P_1(t) - \lambda_2 P_2(t) \qquad (5.7)$$

$$\frac{dP_3(t)}{dt} = \lambda_2 P_1(t) - \lambda_1 P_3(t) \qquad (5.8)$$

Equations (5.6), (5.7), and (5.8) along with equation (5.1) can be solved simultaneously. Their solution is

$$P_1(t) \quad = \quad e^{-(\lambda_1+\lambda_2)t} \qquad (5.9)$$

$$P_2(t) \quad = \quad e^{-\lambda_2 t} - e^{-(\lambda_1+\lambda_2)t} \qquad (5.10)$$

$$P_3(t) \quad = \quad e^{-\lambda_1 t} - e^{-(\lambda_1+\lambda_2)t} \qquad (5.11)$$

$$P_4(t) \quad = \quad 1 - P_1(t) - P_2(t) - P_3(t) \qquad (5.12)$$

Then for a series system we have

$$R_s(t) \quad = \quad P_1(t) \quad = \quad e^{-(\lambda_1+\lambda_2)t}$$

and for parallel system,

$$R_p(t) \quad = \quad P_1(t) + P_2(t) + P_3(t)$$

$$R_p(t) \quad = \quad e^{-\lambda_1 t} + e^{-\lambda\ t} - e^{-(\lambda_1+\lambda_2)t}$$

5.6.1.1 Load-Sharing System

A rather straightforward application of Markov Analysis is to a load-sharing system. There are two components in parallel and there is a dependency between them. If one component fails, the failure rate of the other component increases as a result of the additional load placed on it. Because of this dependency, the reliability block diagram techniques cannot be applied. Instead Markov analysis is used to determine the system reliability. For a two-component system the Table 5.1 shows the various states.

The RTD is shown in Figure 5.7, where λ_1+ and λ_2+ represent the increased failure rates of components 1 and 2, respectively, as a result of increased load.

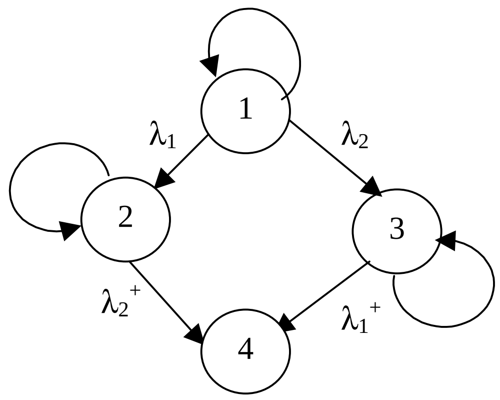


Figure 5.7: Reliability Transition Diagram for Load Sharing System

The resulting differential equations are:

$$\frac{dp_1(t)}{dt} = -(\lambda_1 + \lambda_2)\, p1(t) \qquad (5.13)$$

$$\frac{dp_2(t)}{dt} = \lambda_1 p_1(t) - \lambda_2{}^{+} p_2(t) \qquad (5.14)$$

$$\frac{dp_3(t)}{dt} = \lambda_2 p_1(t) - \lambda_1{}^{+} p_3(t) \qquad (5.15)$$

The solutions to these equations are as follows:

$$P_1(t) \quad = \quad e^{-(\lambda_1+\lambda_2)t} \qquad (5.16)$$

$$P_2(t) \quad = \quad \frac{\lambda_1}{\lambda_1 + \lambda_2 - \lambda_2{}^{+}}[e^{-\lambda_2{}^{+}t} - e^{-(\lambda_1+\lambda_2)t}] \qquad (5.17)$$

$$P_3(t) \quad = \quad \frac{\lambda_2}{\lambda_1 + \lambda_2 - \lambda_1{}^{+}}[e^{-\lambda_1{}^{+}t} - e^{-(\lambda_1+\lambda_2)t}] \qquad (5.18)$$

and

$R_s(t) = P_1(t) + P_2(t) + P_3(t)$

If we let $\lambda_1 = \lambda_2 = \lambda$ and $\lambda_1{}^{+} = \lambda_2{}^{+} = \lambda^{+}$, then

$$R(t) \quad = \quad e^{-2\lambda t} + \frac{2\lambda}{2\lambda - \lambda^{+}}[e^{-\lambda^{+}t} - e^{-2\lambda t}] \qquad (5.19)$$

and

$$MTTF \quad = \quad \int_0^\infty R(t)dt \quad = \quad \frac{1}{2\lambda} + \frac{2\lambda}{2\lambda - \lambda^{+}}\left[\frac{1}{\lambda^{+}} - \frac{1}{2\lambda}\right] \qquad (5.20)$$

5.6.1.2 Standby Systems

Standby systems are an important area of study within reliability. Depending on the probability of failure occurring when switching to a standby unit, these systems are much more reliable than the active redundant system. The two-component standby system differs form the active redundant system discussed earlier in that the standby unit will have no failures or a reduced failure rate while in its standby mode. Once active, the backup unit may experience the same failure rate as the online (primary) system (if they are identical) or may have a different failure rate. The dependency arises because the failure rate of the standby unit depends on the state of the primary unit.

The states of the system are shown in Figure 5.8. The differential equations for this system is as shown

$$\frac{dp_1(t)}{dt} = -(\lambda_1 + \lambda_2) p_1^-(t) \tag{5.21}$$

$$\frac{dp_2(t)}{dt} = \lambda_1 p_1(t) - \lambda_2 \ p_2(t) \tag{5.22}$$

$$\frac{dp_3(t)}{dt} = \lambda_2^- p_1(t) - \lambda_1 \ p_3(t) \tag{5.23}$$

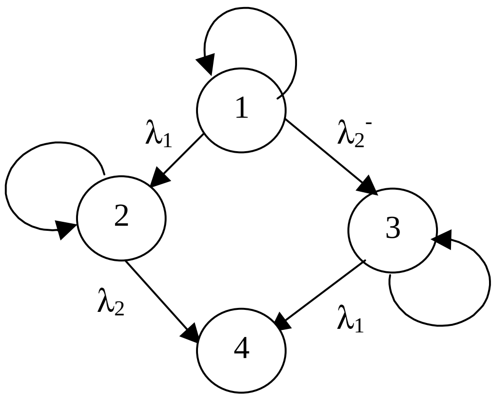


Figure 5.8: Reliability Transition Diagram for Standby System

The solutions to above differential equations are as follows:

$$P_1(t) \quad = \quad e^{-(\lambda_1+\lambda_2^-)t} \tag{5.24}$$

$$P_2(t) \quad = \quad \frac{\lambda_1}{\lambda_1 + \lambda_2^- - \lambda_2}[e^{-\lambda_2 t} - e^{-(\lambda_1+\lambda_2^-)t}] \tag{5.25}$$

$$P_3(t) \quad = \quad [e^{-\lambda_1 t} - e^{-(\lambda_1+\lambda_2^-)t}] \tag{5.26}$$

and

R(t) = $P_1(t) + P_2(t) + P_3(t)$

$$R(t) \quad = \quad e^{-\lambda_1 t} + \frac{\lambda_1}{\lambda_1 + \lambda_2^{-} - \lambda_2}[e^{-\lambda_2 t} - e^{-(\lambda_1 + \lambda_2^{-})t}] \qquad (5.27)$$

and

$$MTTF \quad = \quad \frac{1}{\lambda_1} + \frac{\lambda_1}{\lambda_2(\lambda_1 + \lambda_2^{-})} \qquad (5.28)$$

If there are no failures of the standby unit, set $\lambda_2^- = 0$ in Eqs. (5.27) and (5.28). If $\lambda_1 = \lambda_2 = \lambda$ and $\lambda_2^- = \lambda^-$, then Eqs. (5.27) and (5.28) simplify to

$$R(t) \quad = \quad e^{-\lambda t} + \frac{\lambda}{\lambda^-}[e^{-\lambda t} - e^{-(\lambda + \lambda^-)t}] \qquad (5.29)$$

$$MTTF \quad = \quad \frac{1}{\lambda} + \frac{1}{\lambda + \lambda^-} \qquad (5.30)$$

5.6.1.3 Standby Systems with Switching Failures

It is not uncommon in a standby system to have some probability p of there being an on-demand failure of a switching device designed to place the standby system on-line. The transition diagram is as shown in figure 5.9.

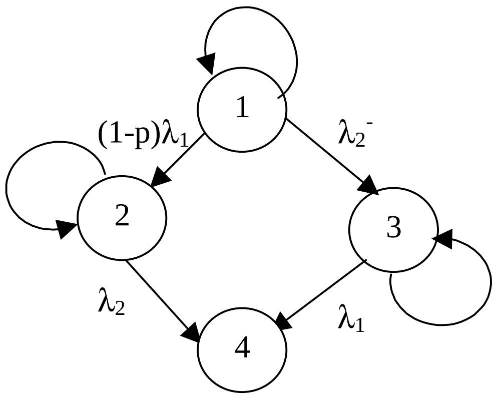


Figure 5.9: Reliability Transition Diagram for Standby Systems with Switching Failures

The reliability function is given by

$$R(t) \quad = \quad e^{-\lambda_1 t} + \frac{(1-p)\lambda_1}{\lambda_1 + \lambda_2^{-} - \lambda_2}[e^{-\lambda_2 t} - e^{-(\lambda_1 + \lambda_2^{-})t}] \qquad (5.31)$$

5.6.1.4 Three-Component Standby System

Consider a system with one active component and two standby components. For simplicity assume that no component fail while in standby and that all three components have the same constant failure rate when on-line. The following states are defined as follows:

State	**Component 1**	**Component 2**	**Component 3**
1	On-line	Standby	Standby
2	Failed	On-line	Standby
3	Failed	Failed	On-line
4	Failed	Failed	Failed

This leads to following differential equations:

$$\frac{dp_1(t)}{dt} = -\lambda P_1(t) \quad (5.32)$$

$$\frac{dp_2(t)}{dt} = \lambda P_1(t) - \lambda P_2(t) \quad (5.33)$$

$$\frac{dp_3(t)}{dt} = \lambda P_2(t) - \lambda P_3(t) \quad (5.34)$$

with initial conditions $P_1(0) = 1$, $P_2(0) = 0$, and $P_3(0) = 0$. The solution is as follows:

$$P_1(t) \quad = \quad e^{-\lambda t} \quad (5.35)$$

$$P_2(t) \quad = \quad \lambda t e^{-\lambda t} \quad (5.36)$$

$$P_3(t) \quad = \quad \frac{\lambda^2 t^2}{2} e^{-\lambda t} \quad (5.37)$$

Since the system is functioning while in any of the first three states,

$$R(t) \quad = \quad e^{-\lambda t}\left[1 + \lambda t + \frac{\lambda^2 t^2}{2}\right] \quad (5.38)$$

and,

$$\text{MTTF} = 3 / \lambda \quad (5.39)$$

5.6.1.5 Degraded Systems

Some systems may continue to operate in a degraded mode following certain types of failures. The system may continue to perform its function but not at a specified operating level. For example, a computer system may not be able to access all of its direct access storage devices, a copying machine may not be able to automatically feed originals and may thereby require manual operation, or a multi-engine aircraft may experience a problem in one of its engines [8]. Whether the degraded mode is considered a failure or not must be determined as part of the reliability specification. However, if it is important to distinguish the degraded state from that of a complete failure, then Markov analysis can be utilized if constant failure rates are assumed.

Defining the states of a system as fully operational (state 1), degraded (state 2), and failed (state 3), the RTD is as shown in Figure 5.10.

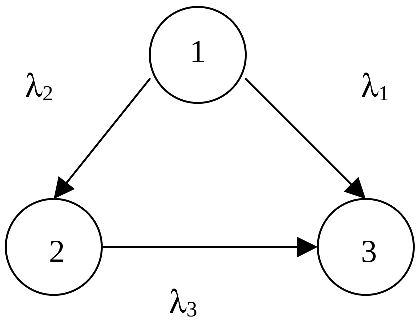


Figure 5.10: Reliability Transition Diagram for Degraded Systems

The differential equations are

$$\frac{dP_1(t)}{dt} = -(\lambda_1 + \lambda_2)P_1(t) \tag{5.40}$$

$$\frac{dP_2(t)}{dt} = \lambda_2 P_1(t) - \lambda_3 P_2(t) \tag{5.41}$$

The solution to Eq. (5.40) and (5.41) is given by

$$P_1(t) \quad = \quad e^{-(\lambda_1+\lambda_2)t} \tag{5.42}$$

$$P_2(t) \quad = \quad \frac{\lambda_2}{\lambda_1 + \lambda_2 - \lambda_3}[e^{-\lambda_3 t} - e^{-(\lambda_1+\lambda_2)t}] \tag{5.43}$$

Finally, P3(t) = 1 – P1(t) – P2(t)

And,

$$MTTF = \frac{1}{\lambda_1 + \lambda_2} + \frac{\lambda_2}{\lambda_1 + \lambda_2 - \lambda_3}\left[\frac{1}{\lambda_3} - \frac{1}{\lambda_1 + \lambda_2}\right]$$

5.7 State Independent System Reliability – Tie Set and Cut Set Algorithm

The term cut set means a minimal set of system components which when fail, cause the system to fail. This technique consists of the following three distinct steps [21].

1. Find minimal paths or tie sets.
2. Find cut sets using the minimal paths found in step 1.
3. Determine the system reliability using cut sets found in step 2.

5.7.1 Tie Set or Minimal Paths Algorithm

A tie set is a minimal set of components from start to end node. Failing of all tiesets causes the system to fail, or if components in any of the tie set works, the system works. A minimal path tree shows the link of all components from the end to the start node. The minimal paths can be obtained by tracing network end to start node.

Assumptions:

1. A network represents the actual system.
2. Two or more parallel branches connected between any pair of nodes are merged to form a single-branch.
3. User identifies previous components for each component in the system.

Algorithm:

1. Place end node on top of the tree (level 1).
2. Check for previous matrix for predecessor of end node. Put them at level 2.
3. For each component at a given level which is not start node, do the following:
    - 3.1 Obtain predecessor for a given component.
    - 3.2 For predecessor connected to given component check the following:
        - 3.2.1 Is the component connected the same way somewhere on the minimal path traced out so far from end to present component? If yes, go to step 3.2 for next component (repetition).

3.2.2 If not place them in level 3 and go to step 3.

If start node, then go to step 4 – backward procedure.

4. Go back on one level and check for other predecessor. Go to step 3. Repeat until all predecessors are taken into account at all levels.

After obtaining all paths of network one can eliminate redundant paths to form minimal paths. This can be done by comparing the current component at the current level with the components from the previous level in the corresponding branch.

5.7.2 Determination of Cut sets from given Minimal Paths

Determination of cut sets is important for evaluation of system reliability. Evaluation of the combination of component failures that discontinue the minimal paths can obtain minimal cut sets. Boolean logic is used to obtain cut sets from the matrix containing the minimal paths.

Assumptions:

1. Minimal paths are represented as binary matrix T_{MxN} , where M is the number of minimal paths and N is the number of components. T[i,j] = 1, if component j is in cut set i, 0 otherwise. T_j is jth column vector of T_{MxN} matrix.

Algorithm:

First order cut sets:

For all jth column vector of T_{MxN} matrix do:

1. If $T_{ji} = 1$ for all i = 1,2….M, the corresponding element j is a single element cut. No single-elements cuts otherwise.

Second order cut sets:

For all $j \neq c$ (column vectors) of T_{MxN} matrix:

2. If $T_{ji} + T_{ci} = 1$ for all i = 1,2….M, then the elements j and c form a second order cut set. Here, '+' indicates the logic sum or union. Eliminate non-minimal cut set as follows:
   2.1 After obtaining a cut set, check it against all cuts of order one.
      2.1.1 If the cut set contains a smaller order cut set, eliminate it.
      2.1.2 Proceed to next cut set (step 2) until all cut sets are identified.
   3. Repeat the procedure for all possible cuts of higher order 1,2…..N, where N is the number of elements in the system..

5.7.3 Example

Let there be a system that has 5 components, X_1 through X_5. Connection as shown in Figure 5.11. Start and end are pseudo nodes.

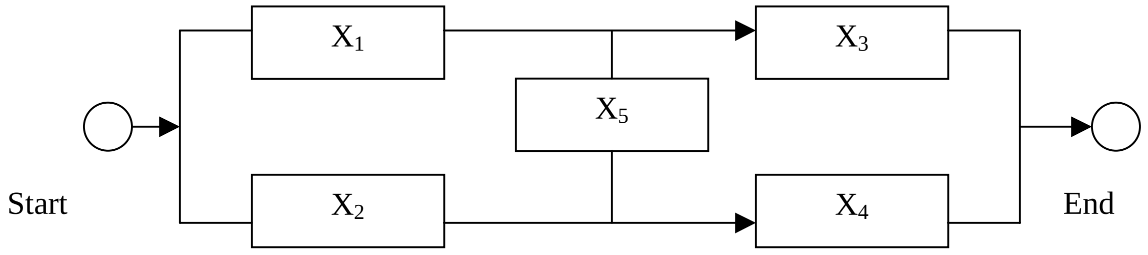


Figure 5.11: Complex System

The previous component (PC) matrix of this system is given as:

$$PC = \begin{array}{c|ccccc} & \mathbf{X_1} & \mathbf{X_2} & \mathbf{X_3} & \mathbf{X_4} & \mathbf{X_5} \\ \hline \mathbf{X_1} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{X_2} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{X_3} & 1 & 0 & 0 & 0 & 1 \\ \mathbf{X_4} & 0 & 1 & 0 & 0 & 1 \\ \mathbf{X_5} & 1 & 1 & 0 & 0 & 0 \end{array}$$

Rows (i) and columns (j) of the PC matrix correspond to component number. PC_{ij} = 1 if component j is previous to component i. To form the minimal path tree the End (E) node is placed at the level 1. All nodes that are predecessors to the End node are placed at the level 2. Process is repeated until no previous nodes exist. The tree for network is shown in Figure 5.12.

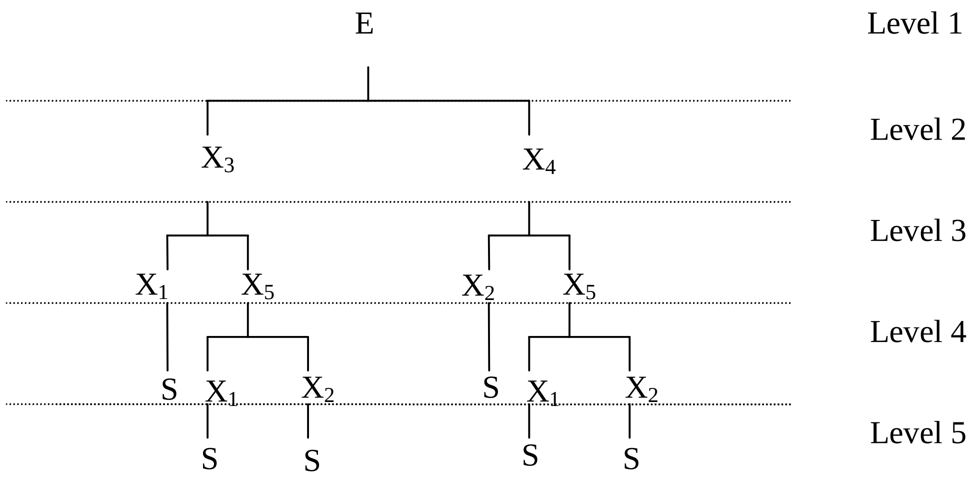


Figure 5.12: Tree Structure of the Complex System

The minimal path (MP) matrix is generated by the above procedure. From MP matrix the End (E) and the Start (S) nodes can be eliminated resulting in the tie set, T.

$$MP = \begin{bmatrix} E & X_3 & X_1 & S & \\ E & X_3 & X_5 & X_2 & S \\ E & X_4 & X_2 & S & \\ E & X_4 & X_5 & X_1 & S \\ E & X_4 & X_5 & X_2 & S \\ E & X_3 & X_5 & X_1 & S \end{bmatrix} \qquad T = \begin{bmatrix} X_3 & X_1 & \\ X_3 & X_5 & X_2 \\ X_4 & X_2 & \\ X_4 & X_5 & X_1 \\ X_4 & X_5 & X_2 \\ X_3 & X_5 & X_1 \end{bmatrix}$$

5.7.3 Algorithm for Calculating System Reliability

This section describes a computer algorithm to determine the reliability of a complex system. The algorithm is based on [21]. The algorithm uses a sequence of prediction equations, which provides increasingly closer bounds on the system reliability. It is based on the knowledge of tie set and cut set. Let T_m is the mth tie set, M (number of tiesets) in number and C_k is kth cut set, K (number of cutsets) in number. Then system reliability is given by $R_s = P\,\{T_1{*}T_2.....{*}T_M\}$ = P (at least one tie set is good).

Expressed in terms of cut sets, $R_s = P\,\{C_1 + C_2 + ...+ C_k\}$ = P (cut sets are good, i.e. contain at least one element of the set which is operative). Equivalently, the system failure is expressed as:

$$1 - R_s = P\{T'_1{*}T'_2{*}.......{*}T'_M\} = P\{\text{all tie sets have a failure}\} \text{ or}$$

$$1 - R_s = P\{C'_1 + C'_2 + ... + C'_K\} = P\{\text{all tie sets have a failure}\} \text{ or}$$

Where T'_M and C'_K are the compliments of the events T_M and C_K respectively. From the above reliability expression bounds can be obtained by using the basic probabilistic inequalities.

$$R_s = P\{T_1 + T_2 + ... + T_M\} \le \sum_{i=1}^{M} P\{T_i\}$$

$$R_s = P\{T_1 + T_2 + ... + T_M\} \ge \sum_{i=1}^{M} P\{T_i\} - \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} P\{T_i * T_j\}, \qquad 1 \le i, j \le M$$

The upper and lower bounds R_{U1} and R_{L1} to the reliability are given by:

$$R_{U1} = \sum_{i=1}^{M} P\{T_i\}$$

$$R_{L1} = \sum_{i=1}^{M} P\{T_i\} - \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} P\{T_i * T_j\}, \qquad 1 \le i, j \le M$$

In the same manner, another upper bound is obtained

$$R_{U2} = \sum_{i=1}^{M} P\{T_i\} - \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} P\{T_i * T_j\} + \sum_{i=1}^{M-2} \sum_{j=i+1}^{M-1} \sum_{k=j+1}^{M} P\{T * T_j * T_k\}, \qquad 1 \le i, \quad j,k \le M$$

Similarly inequalities can be applied to cut sets to obtain another upper and lower bounds.

$$R_{sys} \ge 1 - \sum_{i=1}^{K} P\{C'_i\}, thus \qquad R_{L2} = 1 - \sum_{i=1}^{K} P\{C'_{i\}}$$

$$R_{U3} = 1 - \sum_{i=1}^{K} P\{C'_i\} + \sum_{k<m} P\{C'_K * C_M\}, \qquad 1 \le K, \quad M \le K$$

Thus by using the equations of reliability based on tie set and cut set series of upper and lower bounds are generated which when converge provides the actual system reliability. The above equations were for three tiesets. If a system has more tiesets, similar equations are obtained. The only difference is that as the tiesets increase, the combination is among more tiesets till the combination of total number of tiesets, with +/- signs placed alternatively among them. This when converges, gives the actual reliability of the system.

Chapter 6: Software Tool for Reliability Estimation (STORE)

A Software Tool for Reliability Estimation (STORE) is described in this chapter. It was developed to facilitate the calculation of reliability of a component or a system using the different hardware/software models and system reliability algorithm discussed in chapters 3, 4 and 5. Apart from calculating the reliability of a component or a system, it also provides useful statistical analysis, which helps a user to select the model for given data. STORE is capable of the following,

- Study existing statistical models to estimate reliability of Hardware-Software components.
- Study existing algorithm to compute reliability of complex systems.
- Implement an algorithm to compute tiesets and cutsets of complex systems.
- Compute reliability (with known or unknown parameters) of state dependent and state independent systems.

6.1 STORE Description

STORE is a Windows based reliability estimation tool. It is implemented using Microsoft Visual Basic 6 [33]. As a Microsoft's premium programming language, Visual Basic 6 is the easiest object-oriented programming approach for application development. It has a sophisticated front end, with powerful windows features and the backend, where the logic is developed that interacts with the front end. The Jet Database Engine was used to establish such connection[33]. STORE essentially consists of forms or screens. These forms are similar to those displayed by Windows 95/98/NT. There are control buttons or toolbox controls, which are used on the custom designed forms. These control buttons generally build the interface between the program and the user. The major tool used in this program consists of flexigrid, which allows user to open data in tabular form from a data file. A file open menu, which allows the user to select data files from the directory in their PC or disk and open the data in the program.

STORE is intended for practicing reliability engineers, who are interested in calculating and analyzing reliability of hardware and software components and systems. It allows a user to analyze hardware, software and system reliability under one environment. STORE also has capability to calculate reliability and Mean Time To Failure (MTTF) for state dependent systems, which most of the commercial reliability tools do not support.

6.2 STORE Menu Structure

Various forms constitute the building block of STORE. These forms and associated code allow the user to input data and estimate the reliability. The forms are discussed here as they appear in the program and the code is shown in Appendix V. Figure 6.1 shows the menu structure of STORE.

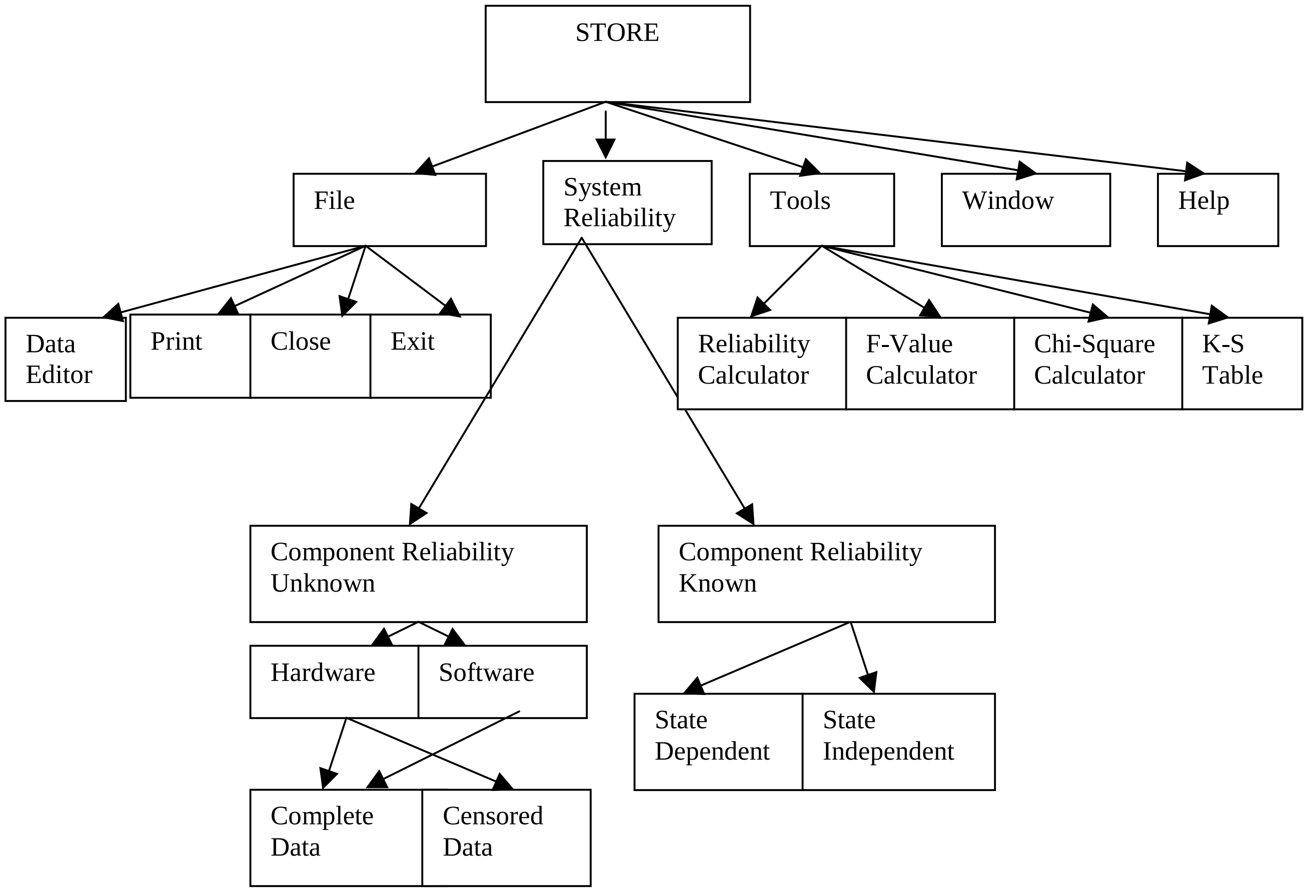


Figure 6.1: Menu Structure

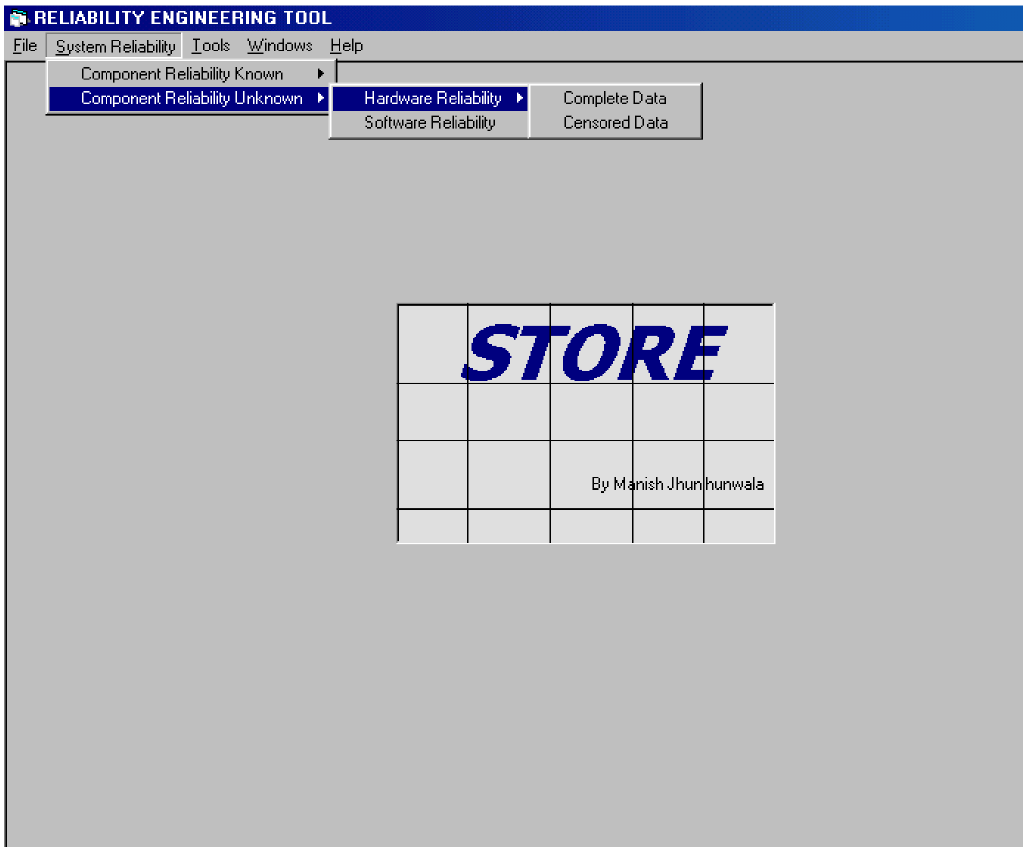


Figure 6.2: Opening Screen

When the software is started the screen as shown in Figure 6.2 is displayed. The options on this screen are File, System Reliability, Tools, Windows and Help.

Under the File menu, there is Data Editor, which allows the user to create data file or to edit an existing data file. The “Print” option allows user to print the results obtained. The “Close” option closes the applicatiuon and “Exit” terminates the program.

Under system reliability, there are two submenus, one for system reliability when the component reliability is known and other when component reliability is unknown. When component reliability is known, the user can work upon both state dependent and state independent systems. If the component reliability is unknown, then user can compute component reliability using Hardware or Software Reliability menu. The hardware reliability can be calculated for both complete and censored data, whereas software is only for complete data.

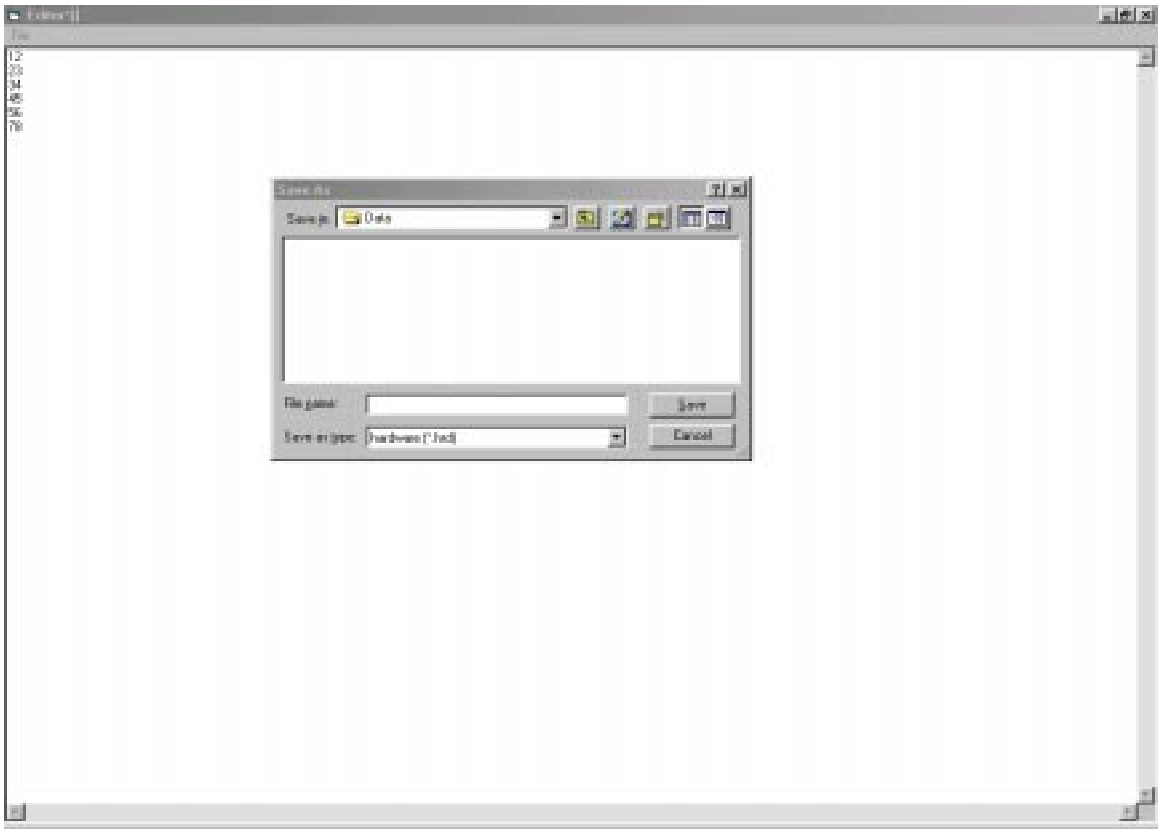
Under the Tools menu, there is F-value calculator, Reliability calculator, Kolmogorv-Smirnov Table and Chi-square calculator. These tools assist the user to select the “best” model. For example, if the user needs to check the distribution for different number of risk, this can be done using the tools instead of going through complete program.

The “Windows” menu helps the user arrange the window frames in cascade or tile form. Also this menu lists all the open forms of the program.

The Help menu provides the basic help explaining terms used in reliability, notations and nomenclature used in program and how to use the program.

6.2.1 Data Editor

The Data Editor is a text editor for this particular software. The user can enter the failure data for hardware or software item and can save the file with “hrd” for hardware or “sft” for software extension. Also user can enter system block in this editor and save it using “sys” for system as extension. The different extensions allows the user to differentiate between the different types of data files. The data files stored through the data editor are then recalled in the respective application for further evaluation. The sample screen of the data editor is shown in Figure 6.3.

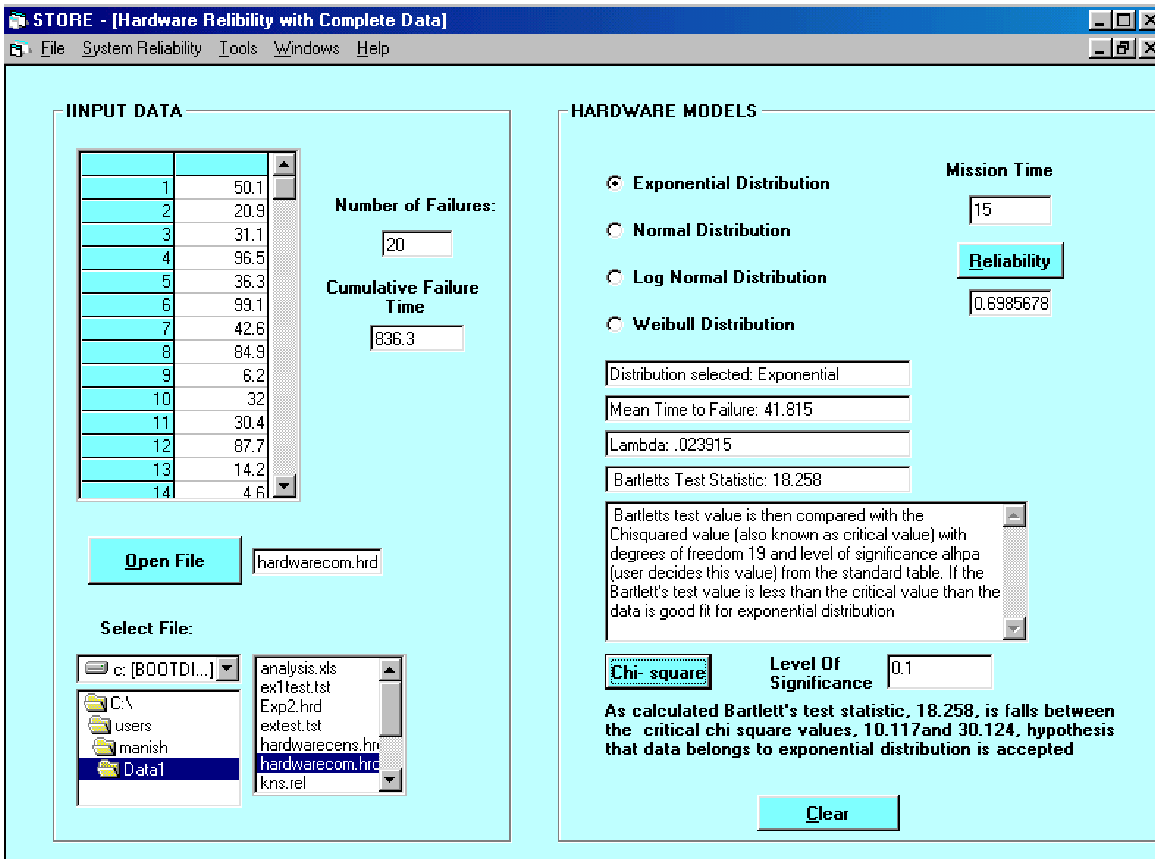


```
12
23
34
45
56
78
```

Figure 6.3: Data Editor Screen

6.2.2 Hardware Reliability – Complete Data

This screen opens when the user clicks the “Complete data” under the Hardware reliability option of “Open” submenu. This screen helps the user in specifying information about the failure data. The file control provision allows the user to select the data file for the particular hardware product. When the selected file is open, user can see the data in the data grid. Data grid is only for displaying the data, the user cannot edit the data through data grid. If the user wants to edit the data, the data file has to be opened in the “Data Editor”. User can then select the appropriate distribution. This selection is possible through the “option button”. One distribution at a time can be selected. On clicking “Hardware Reliability” button, program calculates the parameters for that particular distribution and also the reliability of the product for a specified mission time. The output of the program is displayed in the object “label”. User can experiment with different distributions and compare the output. Also user can print the screen by selecting the print option under “File” menu. “Clear” button allows the user to erase all the previously calculated information and start afresh with a data file. The sample screen of hardware reliability is shown in Figure 6.4.



1	50.1
2	20.9
3	31.1
4	96.5
5	36.3
6	99.1
7	42.6
8	84.9
9	6.2
10	32
11	30.4
12	87.7
13	14.2
14	4.6

Number of Failures: 20

Cumulative Failure Time: 836.3

hardwarecom.hrd

[✓ Exponential Distribution]
Normal Distribution
Log Normal Distribution
Weibull Distribution

Mission Time: 15

Reliability: 0.6985678

Distribution selected: Exponential

Mean Time to Failure: 41.815

Lambda: .023915

Bartletts Test Statistic: 18.258

Bartletts test value is then compared with the Chisquared value (also known as critical value) with degrees of freedom 19 and level of significance alhpa (user decides this value) from the standard table. If the Bartlett's test value is less than the critical value than the data is good fit for exponential distribution

Level Of Significance: 0.1

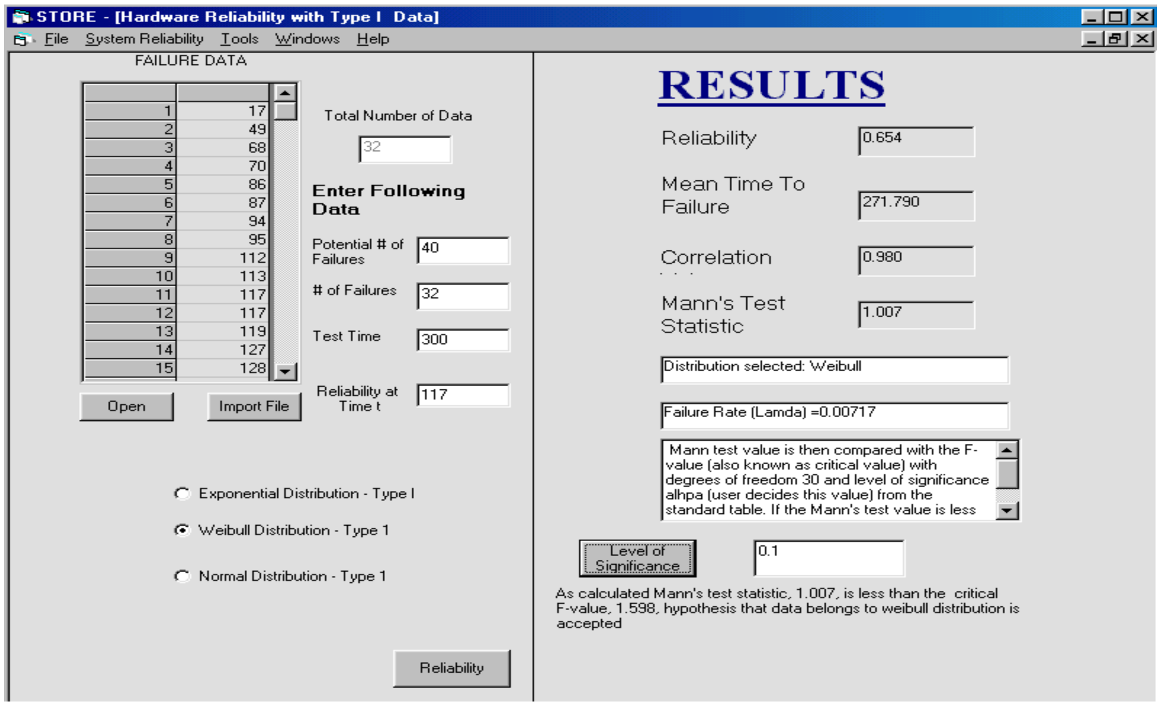
As calculated Bartlett's test statistic, 18.258, is falls between the critical chi square values, 10.117and 30.124, hypothesis that data belongs to exponential distribution is accepted

Figure 6.4: Hardware Reliability – Complete data screen

6.2.3 Hardware Reliability – Censored Data

This screen opens when user user clicks censored in hardware reliability option under “Open” submenu. This screen helps the user to open the failure data in the grid on the upper right hand side of the screen. Then the user can compute reliability, do statistical analysis for the censored data of Type I.

The data is displayed in this screen from a file, which can be opened by clicking the “Open” button on the screen and then clicking “Extract”. In this screen, the user can select from three different distributions. This selection is possible through the option button shown on the screen. User can select only one option at a time. The user is also required to input potential number of failures, test time and time t. The program calculates number of failures. Once the required information is supplied on the screen, just click the “Reliability” button. This action will calculate reliability, corelation, mean time to failure and test statistic for the item for a specified test time. The output is displayed in the text box, explaining the results. The goodness of fit test is performed by specifying the level of significane in the text box. After specifying a level of significance, the program computes Goodness of Fit test and displays the output in the text box. The sample screen of hardware reliability for censored data is shown in Figure 6.5.



FAILURE DATA

1	17
2	49
3	68
4	70
5	86
6	87
7	94
8	95
9	112
10	113
11	117
12	117
13	119
14	127
15	128

Total Number of Data: 32

Enter Following Data

Potential # of Failures: 40

# of Failures: 32

Test Time: 300

Reliability at Time t: 117

Exponential Distribution - Type I [✓ Weibull Distribution - Type 1] Normal Distribution - Type 1

RESULTS

Reliability: 0.654

Mean Time To Failure: 271.790

Correlation: 0.980

Mann's Test Statistic: 1.007

Distribution selected: Weibull

Failure Rate (Lamda) =0.00717

Mann test value is then compared with the F-value (also known as critical value) with degrees of freedom 30 and level of significance alhpa (user decides this value) from the standard table. If the Mann's test value is less

Level of Significance: 0.1

As calculated Mann's test statistic, 1.007, is less than the critical F-value, 1.598, hypothesis that data belongs to weibull distribution is accepted

Figure 6.5: Hardware Reliability – Censored data screen

6.2.4 Software Reliability

Like hardware reliability, this screen is designed to analyze the software failure data. There is provision for opening the failure data file and selecting the different distributions. By clicking on the "Failure Intensity" button, the program calculates the failure intensity for that particular software failure data. There is also provision for calculating the failure intensity for the user specified time. This helps the user to know how long the software product has to be tested. This screen also displays the graphical behavior of the fitted distribution. This is achieved with the special control "MS-Chart Control". For this the two columns of the data grid are designated as X-axis and Y-axis, and then the graph is plotted between them. The sample screen for software reliability is shown in Figure 6.6.

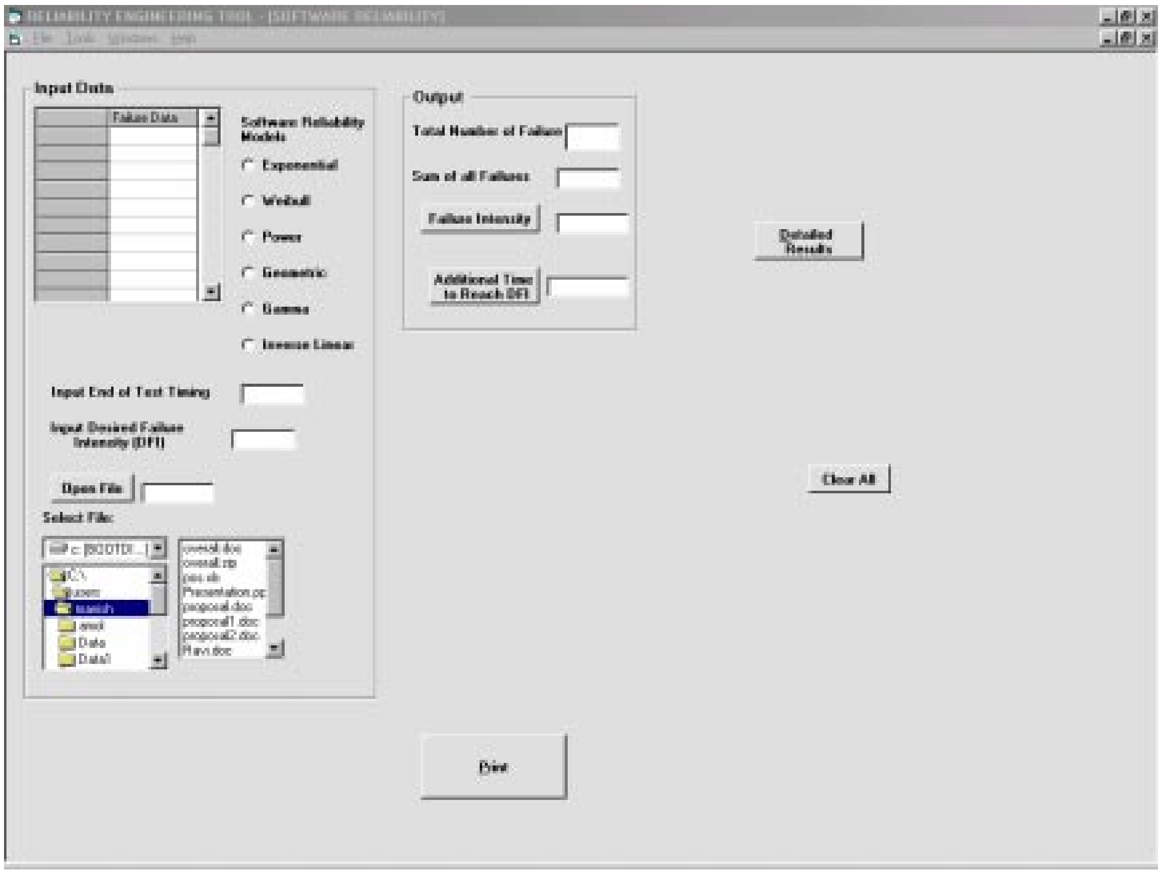
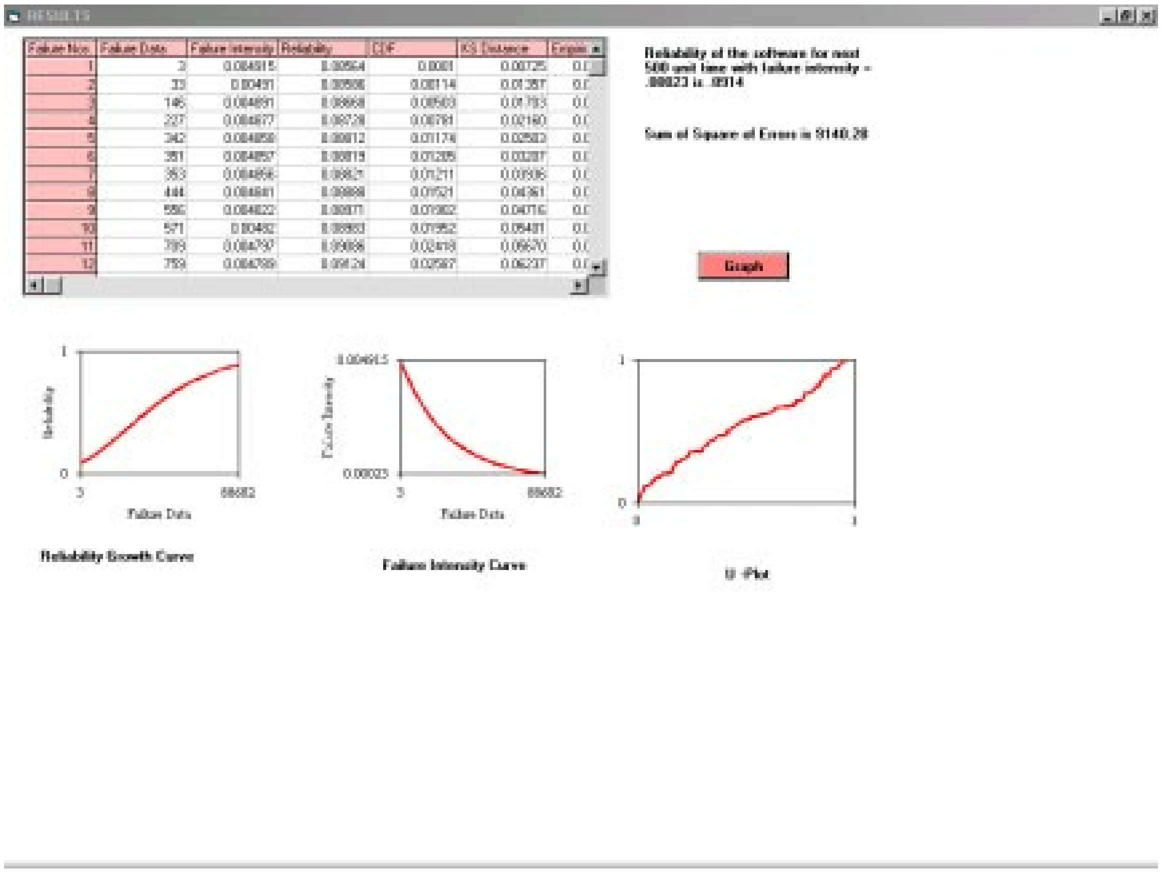


Figure 6.6: Software Reliability Screen

6.2.5 Result Screen

This screen is associated with the software reliability screen. It cannot be called from the menu bar. Once the software reliability screen is executed to analyze the failure data, the results screen can be called by clicking “Detailed Results” button. On this screen all of the necessary information related to the analyzed data is displayed. Data grid is used to display the failure intensity and the reliability of the software at the end of a particular test time. Also important statistical information pertaining to the data is displayed. By clicking “Graph” the user is able to see various graphs related with the analyzed data as shown in Figure 6.7.



Failure Nos	Failure Data	Failure Intensity	Reliability	CDF	KS Distance	Empiri
1	3	0.004915	0.98564	0.0001	0.00725	0.(
2	33	0.00491	0.98586	0.00114	0.01387	0.(
3	146	0.004891	0.98668	0.00503	0.01793	0.(
4	227	0.004877	0.98728	0.00781	0.02160	0.(
5	342	0.004858	0.98812	0.01174	0.02583	0.(
6	351	0.004857	0.98819	0.01205	0.03287	0.(
7	353	0.004856	0.98821	0.01211	0.03936	0.(
8	444	0.004841	0.98888	0.01521	0.04361	0.(
9	556	0.004822	0.98971	0.01902	0.04716	0.(
10	571	0.00482	0.98983	0.01952	0.05481	0.(
11	709	0.004797	0.99086	0.02418	0.05670	0.(
12	759	0.004789	0.99124	0.02587	0.06237	0.(

Reliability of the software for next 500 unit time with failure intensity = .00023 is .9914

Sum of Square of Errors is 9140.28

Figure 6.7: Result Screen for Software Reliability

6.2.6 System Reliability – State Dependent

This screen is designed to calculate reliability of two component state depenedent systems. This feature is not available in any commercial reliability tools. This screen is different from the other STORE screens. In this screen there are five tabs that allow user to open desired screen depending upon the state of the system.

Figure 6.8 shows a sample form for such calculations. Different conditions that are used are as follows:

1. Series and Parallel System

2. Load Sharing System
3. Standby Systems
4. Standby systems with switching failures
5. Three component system.

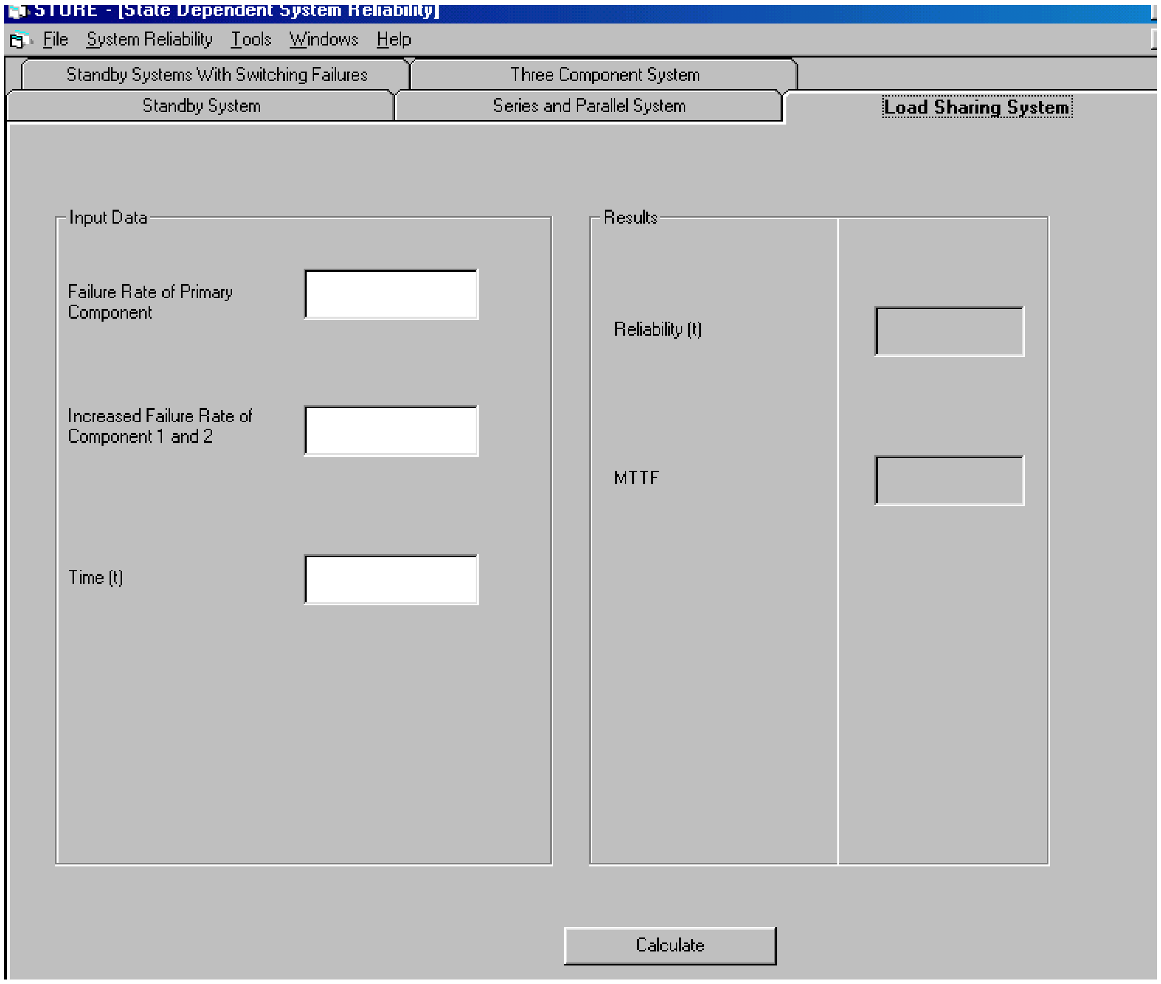
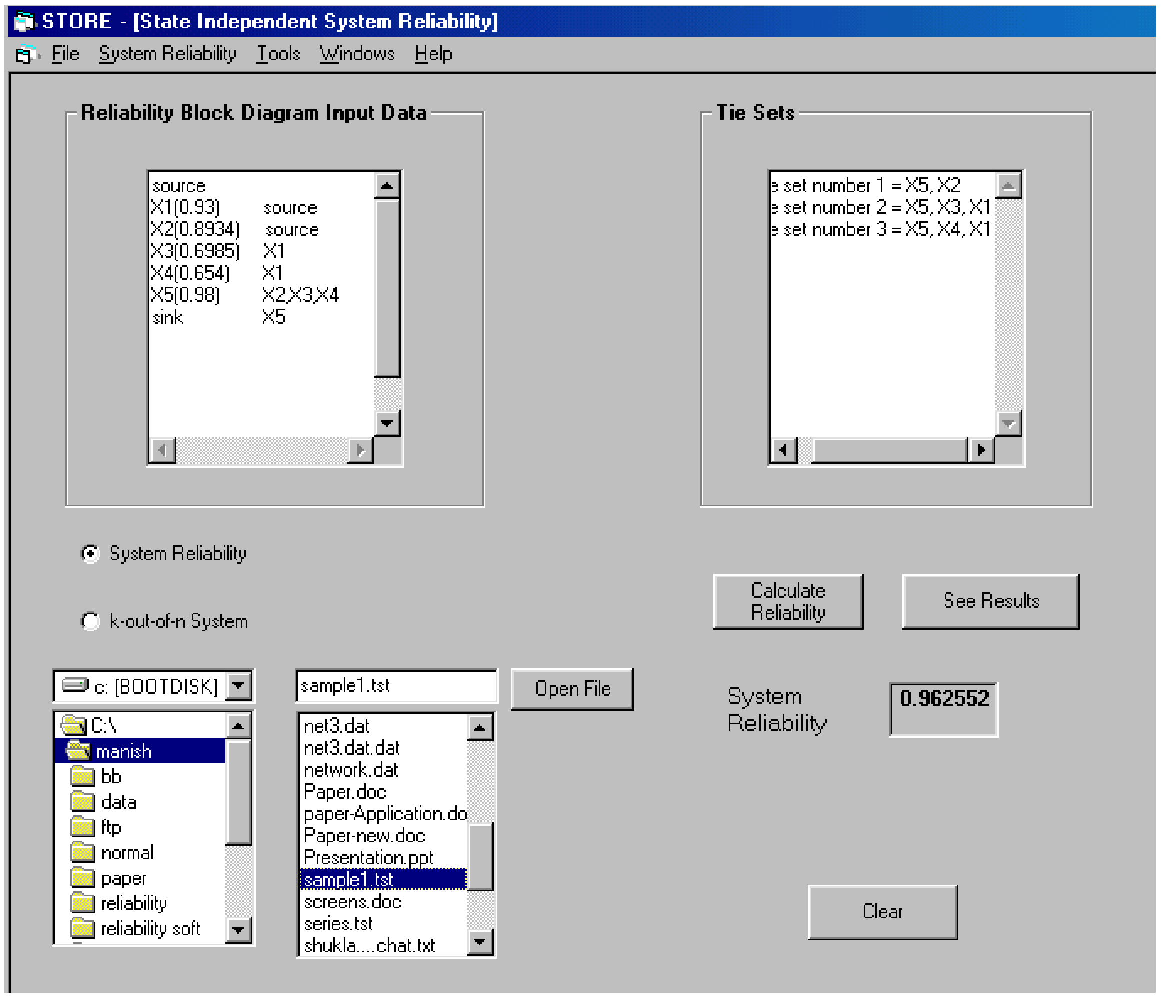

STURE - [State Dependent System Reliability]
File System Reliability Tools Windows Help
Standby Systems With Switching Failures
Three Component System
Standby System
Series and Parallel System
Load Sharing System
Input Data
Failure Rate of Primary Component
Increased Failure Rate of Component 1 and 2
Time (t)
Results
Reliability (t)
MTTF
Calculate

Figure 6.8: State Dependent System Reliability Screen

In this form the user is asked to provide the failure rate and time in state 1, on the basis of which the program calculates system reliability and MTTF.

6.2.7 System Reliability

This screen was designed to calculate system reliability from the given reliability block diagram. On this screen as shown in Figure 6.9, user is required to open the data file having the description of the reliability block diagrams along with their reliability values. The file format is explained in Appendix IV. The data file is then displayed on the screen. Once the file is open, the user presses calculate button, which activates the program and the software calculates the tie sets and system reliability. The results are displayed on the screen on pressing "See Results" button. There is a delay (due to computation) to calculate reliability for the system having more than 15 tie sets.



```
source
X1(0.93)      source
X2(0.8934)    source
X3(0.6985)    X1
X4(0.654)     X1
X5(0.98)      X2,X3,X4
sink          X5
```

```
e set number 1 = X5, X2
e set number 2 = X5, X3, X1
e set number 3 = X5, X4, X1
```

sample1.tst

System Reliability 0.962552

Figure 6.9: System Reliability Screen

6.2.8 Reliability Calculator

This screen was designed to calculate the hardware reliability of a component. It can be found under the menu "Tools". This feature is helpful to the user, if user knows the parameters. Here the user can input the parameter for a particular distribution. STORE then calculates the reliability for different mission times using the same failure data. Four different distributions are available on this calculator as shown in Figure 6.10.

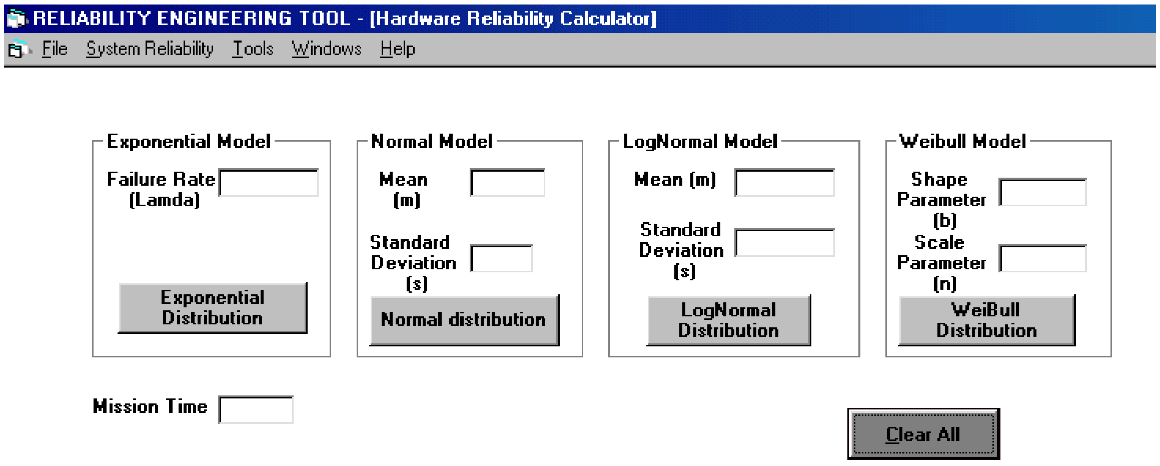


Figure 6.10: Reliability Calculator

Chapter 7: STORE Application

This chapter presents the application of STORE to Series system, Parallel system, Series-Parallel system, k-out-of-n systems and complex systems with hardware components, software components and hardware-software components. STORE is also applied to State dependent systems, i.e. Load Sharing system, Standby system, Standby systems with switching failures and degraded systems.

7.1 Example 1: Series System

Consider a series system that consists of three components and the reliability values of components X_1, X_2 and X_3 are 0.9, 0.8 and 0.75 respectively. Estimate reliability of the system [9]. The estimated system reliability value calculated by STORE is 0.54 as shown in Figure 7.2, which is the same as in [9].

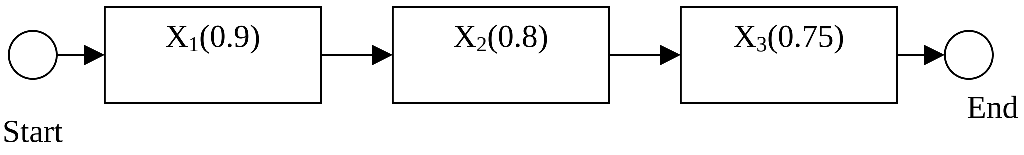
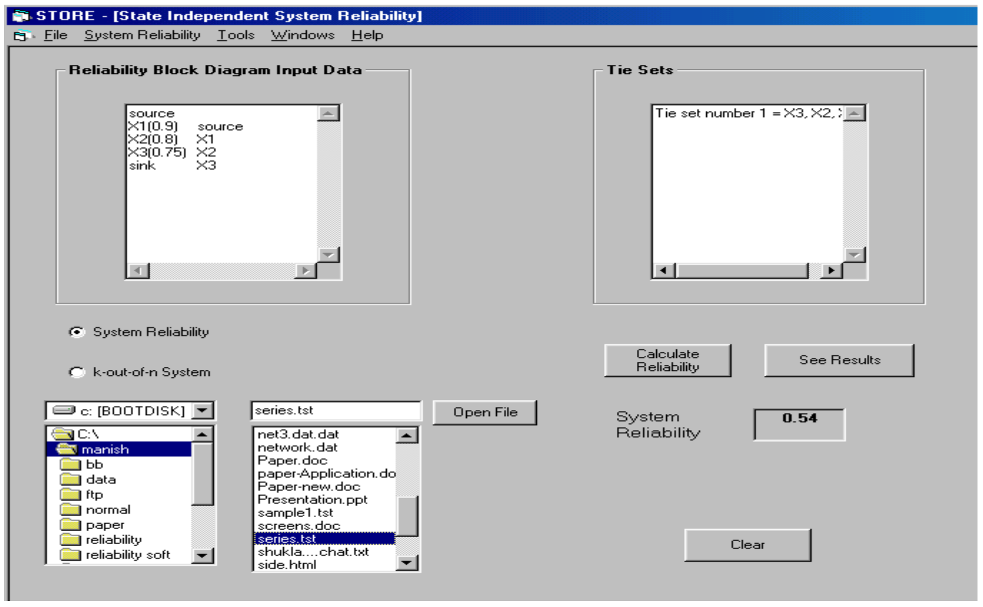


Figure 7.1: RBD of Series System



STORE - [State Independent System Reliability]
File System Reliability Tools Windows Help
Reliability Block Diagram Input Data
Tie Sets
System Reliability
k-out-of-n System
c: [BOOTDISK]
Open File
Calculate Reliability
See Results
Clear

```
source
X1(0.9)    source
X2(0.8)    X1
X3(0.75)   X2
sink       X3
```

Tie set number 1 = X3, X2, X

System Reliability: 0.54

series.tst

```
C:\
manish
bb
data
ftp
normal
paper
reliability
reliability soft
```

```
net3.dat.dat
network.dat
Paper.doc
paper-Application.do
Paper-new.doc
Presentation.ppt
sample1.tst
screens.doc
series.tst
shukla....chat.txt
side.html
```

Figure 7.2: Sample STORE Screen for System Reliability of Series System

7.2 Example 2: Parallel System

Consider a parallel system that consists of three components and the reliability values of that components X_1, X_2 and X_3 are 0.9, 0.8 and 0.75 respectively. Estimate reliability of the system [9]. The estimated system reliability value calculated by STORE is 0.995 as shown in Figure 7.4, which is the same as in [9].

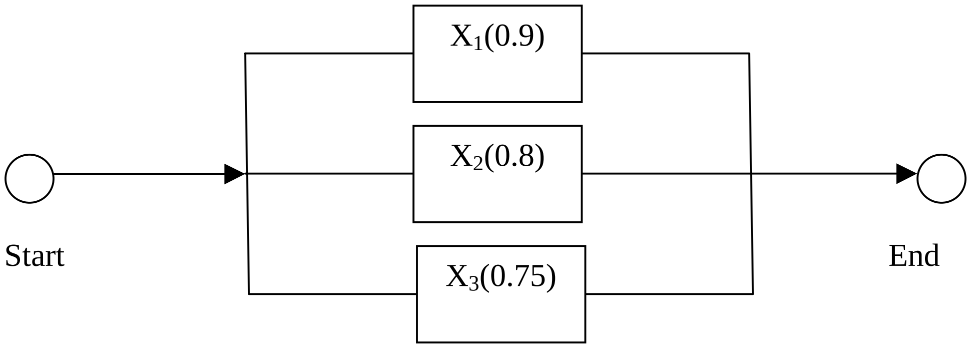
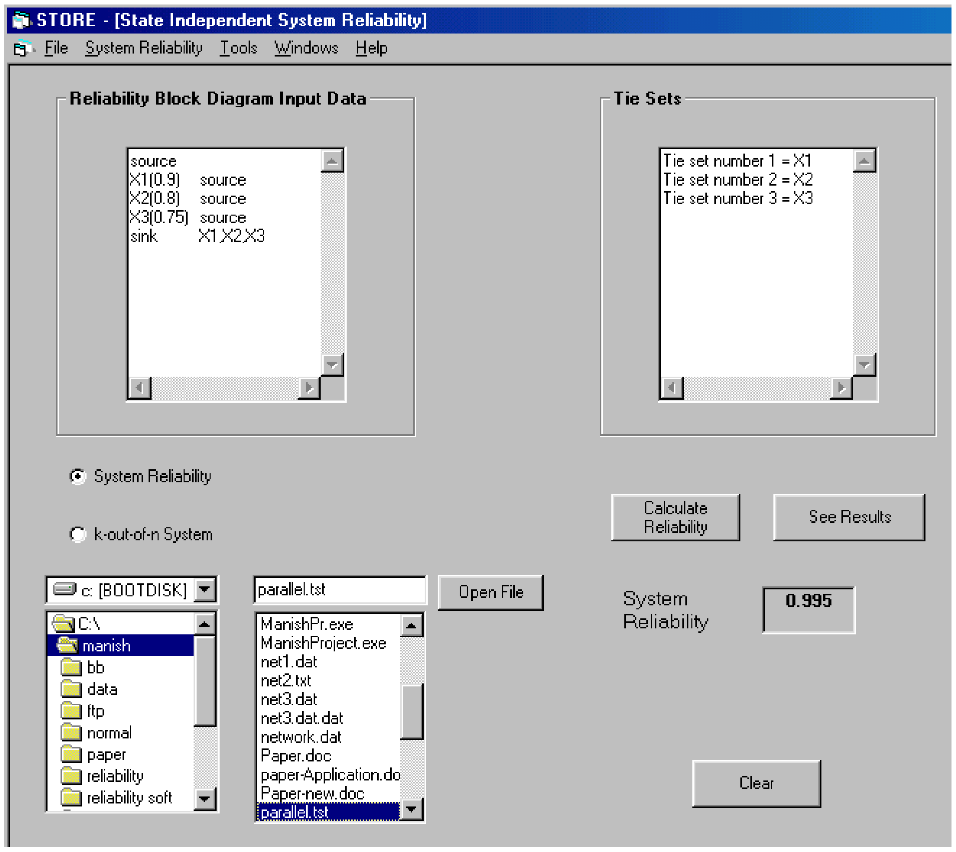


Figure 7.3: RBD for Parallel System



```
source
X1(0.9)    source
X2(0.8)    source
X3(0.75)   source
sink       X1X2X3
```

```
Tie set number 1 = X1
Tie set number 2 = X2
Tie set number 3 = X3
```

[✓ System Reliability] k-out-of-n System

File: parallel.tst

System Reliability: 0.995

Figure 7.4: Sample STORE Screen for System Reliability of Parallel System

7.3 Example 3: k-out-of-n Systems

Consider a telecommunication system that consists of four different parallel channels. A system is considered operational if any three channels are operational. Determine reliability of the system if reliability value of each channel is 0.85 [9]. The estimated system reliability value calculated by STORE is 0.89048 as shown in Figure 7.6, which is same as in [9].

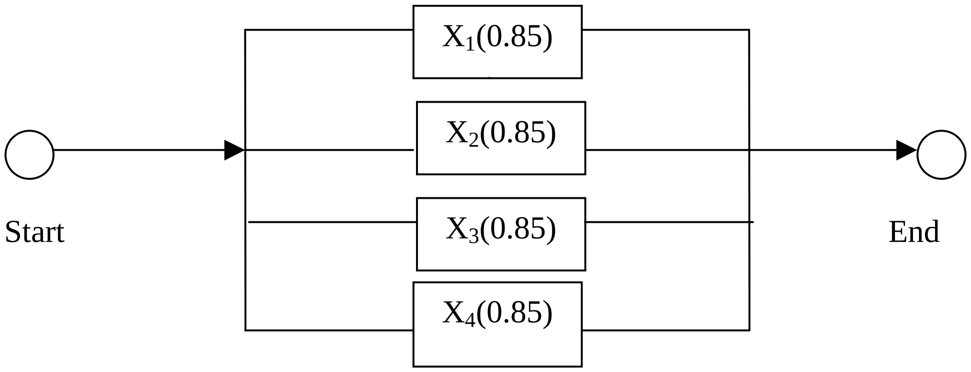
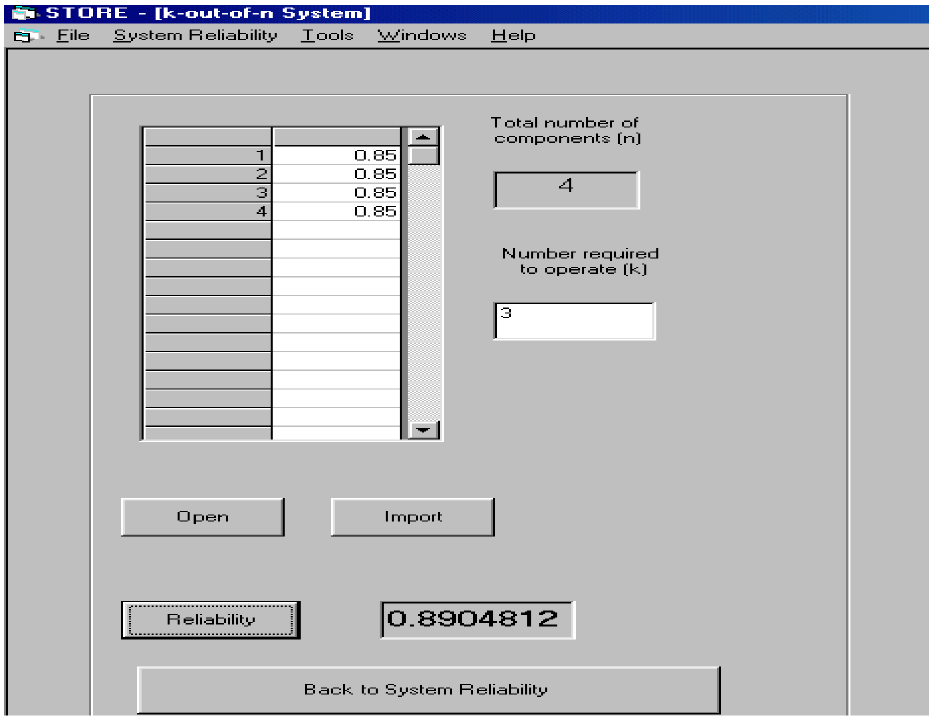


Figure 7.5: RBD for k-out-of-n System



1	0.85
2	0.85
3	0.85
4	0.85

Total number of components (n)

4

Number required to operate (k)

3

0.8904812

Figure 7.6: Sample STORE Screen for System Reliability of k-out-of-n System

7.4 Example 4: Reliability Block Diagram

Consider a software-hardware combined system as shown in Figure 7.7.

Assuming that component X_2 is software component and rest are hardware components. In order to find reliability of the system, we need to first determine reliability of each component. If the user knows the reliability values of all the components, then STORE can calculates system reliability using tieset algorithm for state independent systems. But if the user does not know reliability of any particular component, then the software asks for the type of the component, i.e. hardware or software. Then depending on the type of the component, the user can use STORE to calculate component reliability based on the failure data. STORE allows the user to select different distributions and determine the best fit.

For example, in our reliability block diagram, the user does not know the reliability of components X_2, X_3 and X_4.

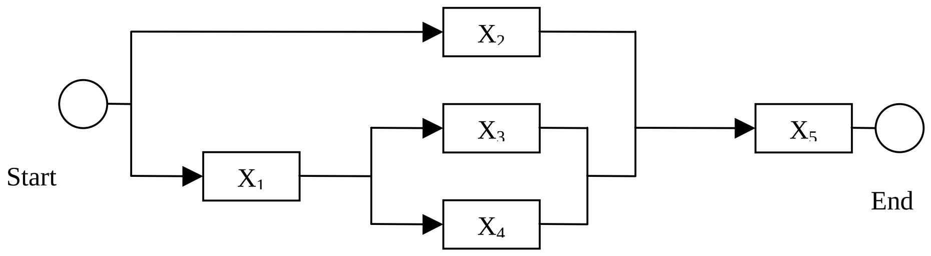


Figure 7.7: Reliability Block Diagram for a Hardware – Software System

7.4.1 Reliability Calculation for Software Component

Let us consider that the data in Table 7.1 is the failure data for a particular software component. The different models can be applied to this data. The data was obtained from [16]. It shows software failures experienced at t_1, $t_2 \ldots t_m$, for total of number of failures (TNF), time is recorded in CPU seconds. The time at end of observation is given by TET, which in this example does not correspond with the last failure, that is, time at end of test (TET), which occurred at time 91208.

Table 7.1: Software Failure Data

3	33	146	227	342	351	353	444	556	571	709	759
836	860	968	1056	1726	1846	1872	1986	2311	2366	2608	2676
3098	3278	3288	4434	5034	5049	5085	5089	5089	5097	5324	5389
5565	5623	6080	6380	6477	6740	7192	7447	7644	7837	7843	7922
8738	10089	10237	10258	10491	10625	10982	11175	11411	11442	11811	12559
12559	12791	13121	13486	14708	15251	15261	15277	15806	16185	16229	16358
17168	17458	17758	18287	18568	18728	19556	20567	21012	21308	23063	24127
25910	26770	27753	28460	28493	29361	30085	32408	35338	36799	37642	37654
37915	39715	40508	42015	42045	42188	42296	42296	45406	46653	47596	48296

49171	49416	50145	52042	52489	52875	53321	53443	54433	55381	56463	56485
56560	57042	62551	62651	62661	63732	64103	64893	71043	74364	75409	76057
81542	82702	84566	88682								

Table 7.2:Software Reliability Models

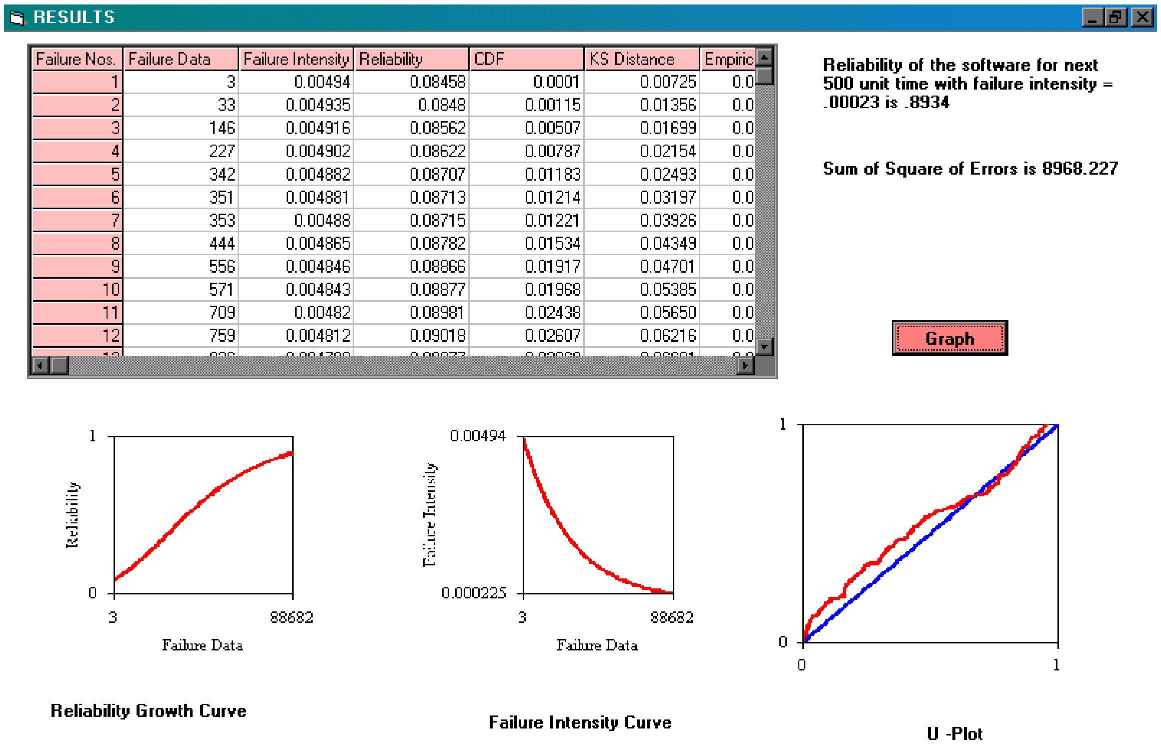
Model Name	Failure Intensity Function	Mean value function
Weibull Distribution	$\lambda(\tau) = b0 * b1 * b2 * \tau^{b2-1} * e^{-b1*\tau^{b2}}$	$\mu(\tau) = b0 * (1 - e^{-b1*\tau^{b2}})$
Exponential Distribution	$\lambda(\tau) = b0 * b1 * e^{-b1*\tau}$	$\mu(\tau) = b0 * (1 - e^{-b1*\tau})$
Gamma Distribution	$\lambda(\tau) = b0 * b1^2 * \tau * e^{-b1*\tau}$	$\mu(\tau) = b0 * (1 - ((1 + b1 * \tau) * e^{-b1*\tau}))$
Power Distribution	$\lambda(\tau) = b0 * b1 * \tau^{b1-1}$	$\mu(\tau) = b0 * \tau^{b1}$
Geometric Distribution	$\lambda(\tau) = \frac{b0 * b1}{1 + b1 * \tau}$	$\mu(\tau) = b0 * \ln(1 + b1 * \tau)$
Inverse Linear Distribution	$\lambda(\tau) = b0 * \frac{1}{2 * \sqrt{b1 + \tau}}$	$\mu(\tau) = b0 * (\sqrt{b1 + \tau} - \sqrt{b1})$

For software reliability estimation, a U-plot is often used. The graph represents a line of unit slope. Any serious departure of the plot from this line is indicative of non-uniformity of the model to that data. The result obtained from all six software models are summarized in Table 7.3.

Table 7.3: Software Failure Results

Model Name	Failure Intensity Failure/CPU Secs	Reliability for next 500 CPU secs	Sum of Squares of Errors
Weibull Distribution	0.00002	0.9881	97565.6
Exponential Distribution	0.00023	0.8934	8968.227
Gamma Distribution	0.00007	0.9668	41992.43
Power Distribution	0.00072	0.6984	7709.368
Geometric Distribution	0.00051	0.7761	2467.949
Inverse Linear Distribution	0.00076	0.6843	11457.86

A sample STORE screen for the exponential model is shown in Figure 7.8. The figure shows a portion of the raw data along with reliability values, the growth curve, failure intensity plot, and the U plot. The reliability value calculated for X_2 from the result screen is 0.8934



Failure Nos.	Failure Data	Failure Intensity	Reliability	CDF	KS Distance	Empiric
1	3	0.00494	0.08458	0.0001	0.00725	0.0
2	33	0.004935	0.0848	0.00115	0.01356	0.0
3	146	0.004916	0.08562	0.00507	0.01699	0.0
4	227	0.004902	0.08622	0.00787	0.02154	0.0
5	342	0.004882	0.08707	0.01183	0.02493	0.0
6	351	0.004881	0.08713	0.01214	0.03197	0.0
7	353	0.00488	0.08715	0.01221	0.03926	0.0
8	444	0.004865	0.08782	0.01534	0.04349	0.0
9	556	0.004846	0.08866	0.01917	0.04701	0.0
10	571	0.004843	0.08877	0.01968	0.05385	0.0
11	709	0.00482	0.08981	0.02438	0.05650	0.0
12	759	0.004812	0.09018	0.02607	0.06216	0.0

Reliability of the software for next 500 unit time with failure intensity = .00023 is .8934

Sum of Square of Errors is 8968.227

Figure 7.8: Sample STORE Screen for the Exponential Model (Software)

.

7.4.2 Calculation of Hardware Reliability – Complete Data

Figure 7.9 shows the screen of hardware reliability calculation form. In this form, the data can be read from a file, which can be opened using the directory and file menus on lower left side. Once the data is read, it automatically displays number of failures and cumulative failure time. After opening the file, the user is required to select one of the four distributions given and enter mission time. Once this is done, then by clicking the "Reliability" button, the program calculates the reliability and displays the value in the text box. Along with this MTTF, failure intensity and test statistics are also computed.

Consider the following failure data for hardware component X_3, obtained from [4]. It represents time to failure (in hours) of a certain component. Let us consider this as complete data.

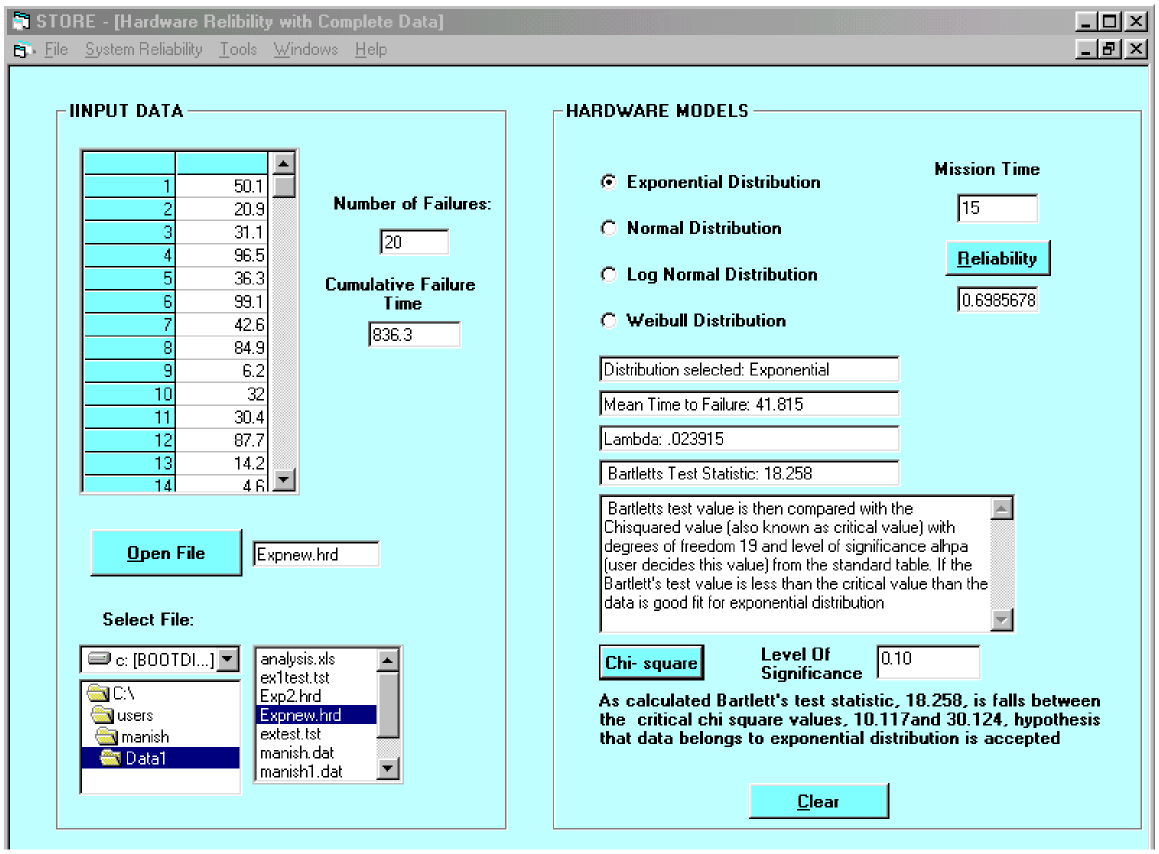
Table 7.4: Data for Hardware Reliability

50.1	20.9	31.1	96.5	36.3	99.1	42.6	84.9	6.2	32.0
30.4	87.7	14.2	4.6	2.5	1.8	11.5	84.6	88.6	10.7

The four hardware models were applied to the above data. The results obtained from each model are summarized in Table 7.5. Based on the test statistics the exponential model is most appropriate for the given data set. STORE determines the appropriate test statistics and the Chi square value.

Table 7.5: Results of Hardware Reliability Models

Model	Parameters	Reliability for a 15 hour. mission.	Test Statistic	Hypothesis
Exponential	b0 (λ) = 0.023915	0.6985	Bartlett's Test: 18.258	Accepted
Normal	b0(m) =41.85 b1(s)=35.22	0.7770	KS test value: 0.1875	Not accepted
LogNormal	b0(m)=1.217 b1(s)=24.4571	0.6563	KS test value:1.0002	Not accepted
Weibull	b0(γ) =0.924 b1(β) =44.1003	0.6911	Mann's Test: 0.874	Not accepted


STORE - [Hardware Relibility with Complete Data]
File System Reliability Tools Windows Help
IINPUT DATA
HARDWARE MODELS
Open File
Select File:
Reliability
Chi- square
Clear

1	50.1
2	20.9
3	31.1
4	96.5
5	36.3
6	99.1
7	42.6
8	84.9
9	6.2
10	32
11	30.4
12	87.7
13	14.2
14	4.6

Number of Failures: 20

Cumulative Failure Time: 836.3

Expnew.hrd

c: [BOOTDI...]

C:\
users
manish
Data1

analysis.xls
ex1test.tst
Exp2.hrd
Expnew.hrd
extest.tst
manish.dat
manish1.dat

(●) Exponential Distribution
( ) Normal Distribution
( ) Log Normal Distribution
( ) Weibull Distribution

Mission Time: 15

0.6985678

Distribution selected: Exponential

Mean Time to Failure: 41.815

Lambda: .023915

Bartletts Test Statistic: 18.258

Bartletts test value is then compared with the Chisquared value (also known as critical value) with degrees of freedom 19 and level of significance alhpa (user decides this value) from the standard table. If the Bartlett's test value is less than the critical value than the data is good fit for exponential distribution

Level Of Significance: 0.10

As calculated Bartlett's test statistic, 18.258, is falls between the critical chi square values, 10.117and 30.124, hypothesis that data belongs to exponential distribution is accepted

Figure 7.9: Sample STORE Screen for the Exponential Model (Hardware)

We find that only exponential model is accepted. Hence we consider the reliability of the component X_3 as 0.6985.

7.1.3 Calculation of Hardware Reliability – Censored Data

Let us assume that for component X_4, we have censored data, with Type I censoring. That is there are total 40 observations, but we take observation till 32 components fail as shown in Table 7.5. The failure data was obtained from [1].

Table 7.5: Data for Hardware Reliability – Censored Data

117	279	128	148	156	17	236	94	182	172	112	117	119	194	297	49
70	155	113	252	86	175	174	250	68	87	290	203	95	263	183	127

The three hardware models were applied to the above data. The results obtained from each model are summarized in Table 7.6. Based on the test statistics the weibull model is most appropriate for the given data set. STORE determines the appropriate test statistics, the Chi square value and the F-values.

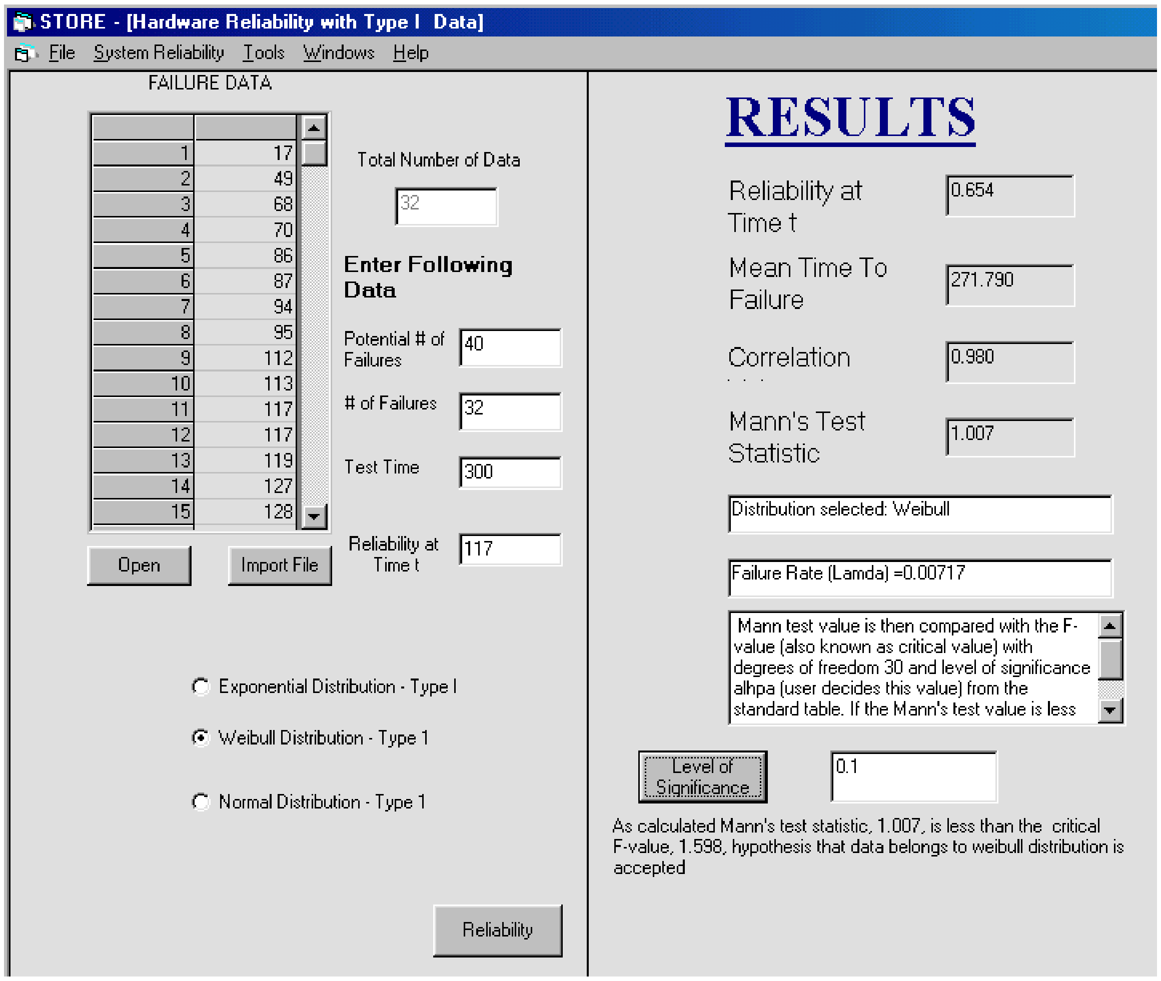
Table 7.6: Results of Hardware Reliability Models

Model	Parameters	Reliability for a 15 hour mission.	Test Statistic	Hypothesis
Exponential	b1 = 0.00418	0.603	Bartlett's Test: 7.516	Not Accepted

Normal	b0 = −1.863
	b1 = 0.009953
		0.9631	KS test value:
			0.1195
				Not accepted

Weibull	b0 = −10.273
	b1 = 1.977
		0.654	Mann's Test: 1.007	Accepted

The sample screen using Weibull distribution is shown in the Figure 7.10. It shows that Weibull distribution is accepted and the reliability of the component X_4 is 0.654.


STORE - [Hardware Reliability with Type I Data]
File System Reliability Tools Windows Help
FAILURE DATA
Open
Import File
Reliability
RESULTS
Level of Significance

1	17
2	49
3	68
4	70
5	86
6	87
7	94
8	95
9	112
10	113
11	117
12	117
13	119
14	127
15	128

Total Number of Data: 32

Enter Following Data

Potential # of Failures: 40

# of Failures: 32

Test Time: 300

Reliability at Time t: 117

( ) Exponential Distribution - Type I
(●) Weibull Distribution - Type 1
( ) Normal Distribution - Type 1

Reliability at Time t: 0.654

Mean Time To Failure: 271.790

Correlation: 0.980

Mann's Test Statistic: 1.007

Distribution selected: Weibull

Failure Rate (Lamda) =0.00717

Mann test value is then compared with the F-value (also known as critical value) with degrees of freedom 30 and level of significance alhpa (user decides this value) from the standard table. If the Mann's test value is less

Level of Significance: 0.1

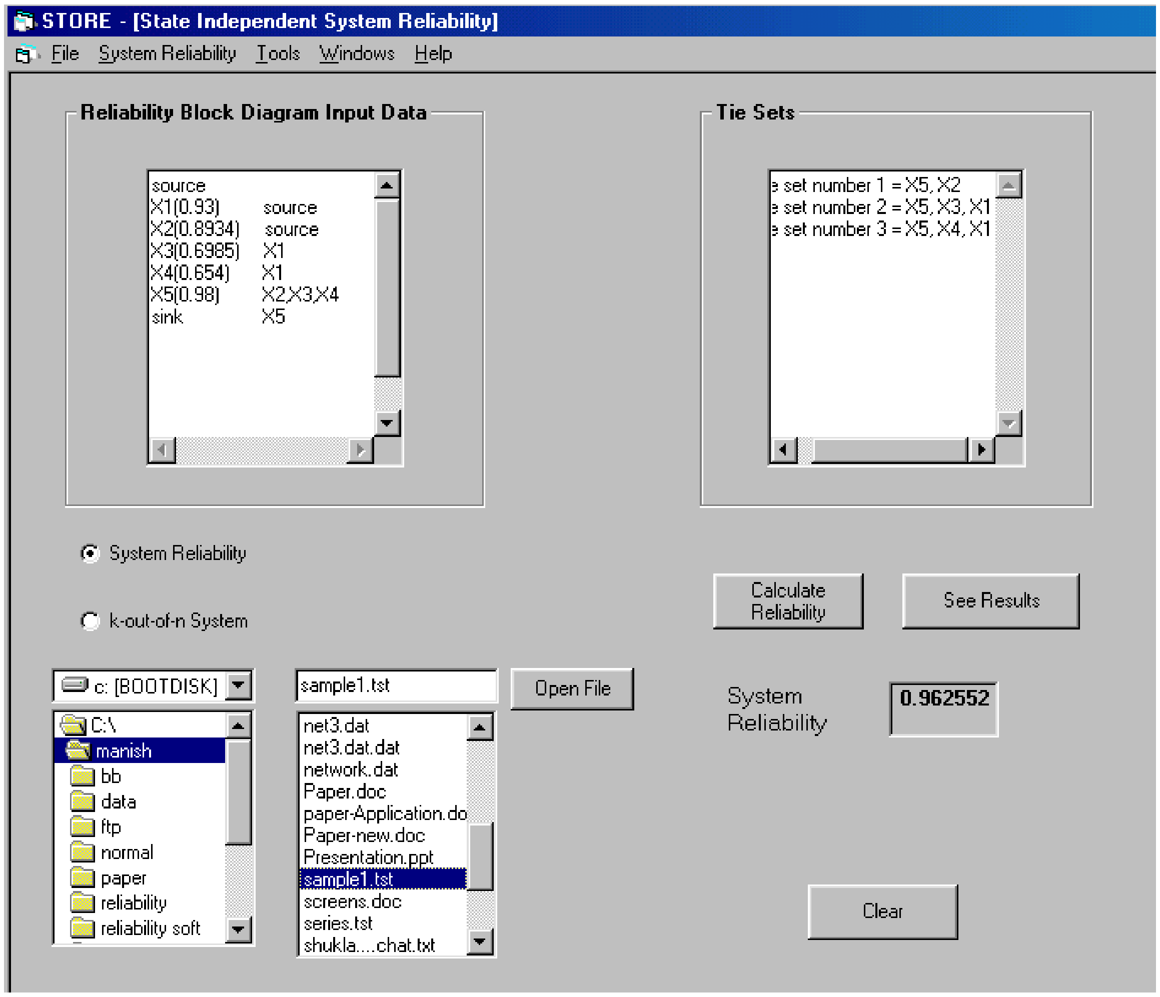
As calculated Mann's test statistic, 1.007, is less than the critical F-value, 1.598, hypothesis that data belongs to weibull distribution is accepted

Figure 7.10: Hardware Reliability – Censored Data Screen

7.4.4 Calculation of System Reliability

Once the user knows the reliability values for all the components. STORE can calculate the system reliability using the tie set algorithm. Now we know that reliabilities for the components are as follows: X_1 = 0.93, X_2 = 0.8934, X_3 = 0.6985, X_4 = 0.654 and X_5 = 0.98.

The tie sets and the system reliability is displayed in the system. The above reliability block diagram was taken from [21]. The tie sets obtained by STORE were the same as in [21]. Since we assumed different components as hardware and software and took failure data from other references, system reliability is not the same as in [21]. The tie sets obtained are: $T_1 = X_5, X_2$; $T_2 = X_5, X_3, X_1$; $T_3 = X_5, X_4, X_1$ and the system reliability as 0.97805. The system reliability calculation screen is shown in Figure 7.11.



```
source
X1(0.93)      source
X2(0.8934)    source
X3(0.6985)    X1
X4(0.654)     X1
X5(0.98)      X2X3X4
sink          X5
```

```
set number 1 = X5, X2
set number 2 = X5, X3, X1
set number 3 = X5, X4, X1
```

sample1.tst

```
net3.dat
net3.dat.dat
network.dat
Paper.doc
paper-Application.do
Paper-new.doc
Presentation.ppt
sample1.tst
screens.doc
series.tst
shukla....chat.txt
```

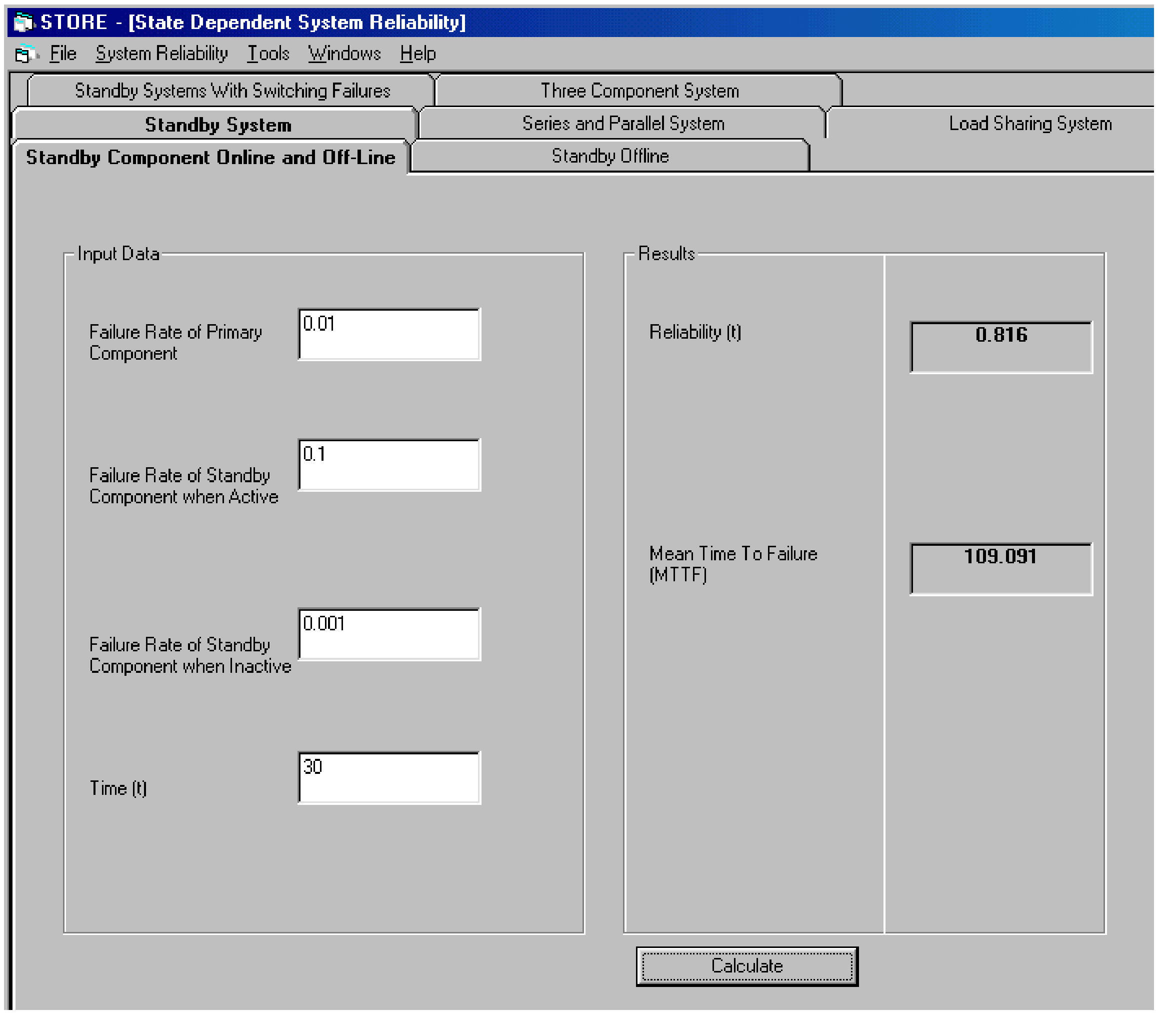
System Reliability 0.962552

Figure 7.11: System Reliability Screen

7.5 Example 5: State Dependent Systems

Consider the following problem from example 6.2 of [4] for Standby system: An active generator has a failure rate (failures per day) of 0.01. An older standby generator has a failure rate of 0.001 while in standby and a failure rate of 0.10 when on-line. Determine the system reliability for a planned 30-day use and compute the system MTTF.

This problem can be solved using STORE. The solution is shown in Figure 7.12.


STORE - [State Dependent System Reliability]
File System Reliability Tools Windows Help
Standby Systems With Switching Failures
Three Component System
Standby System
Series and Parallel System
Load Sharing System
Standby Component Online and Off-Line
Standby Offline
Calculate

Input Data

Failure Rate of Primary Component	0.01
Failure Rate of Standby Component when Active	0.1
Failure Rate of Standby Component when Inactive	0.001
Time (t)	30

Results

Reliability (t)	0.816
Mean Time To Failure (MTTF)	109.091

Figure 7.12: Sample STORE Screen Showing State Dependent System Reliability for Standby system.

The user is required to input failure rate of active component, standby component online and offline, with time in state 1. The tool then calculates the system reliability and MTTF as 0.816 and 109.091 respectively, which is same as in [8].

STORE uses the computational formulae given in [8] for state dependent systems for system reliability and MTTF calculation. STORE is capable of calculating system reliability and MTTF for Markov Analysis, Load Sharing System, Standby Systems, Standby Systems with Switching Failures and Three Component Systems. The results using STORE for all these types is shown in Table 7.7, which is same as examples 6.1, 6.2, 6.5 and 6.6 of [8].

Table 7.7: Reliability and MTTF for State Dependent Systems

Type	Reliability	MTTF
Standby Systems	0.816	109.091
Standby Systems with Switching Failures	0.808	108.18
Load Sharing	0.931	60
Three Component	0.744	857.143

Chapter 8: Conclusions and Future Work

8.1 Conclusion

The aim of this research was to develop a software tool to estimate hardware, software and system reliability. In hardware reliability, problems with both complete and censored data were considered. And in system reliability, two component state dependent system reliability was implemented. The author did not come across any literature where such an integrated tool was presented or described. The software was implemented in Visual Basic. It provides a user-friendly environment, where small sets of easily comprehend input factors are required. Using this software, data entry, data update and retrieval can be performed in a short period of time.

The following models were implemented under hardware reliability for censored data were:

i. Exponential
ii. Normal
iii. Weibull

The models for estimating hardware reliability for complete data were:

i. Exponential
ii. Normal
iii. Weibull
iv. Log Normal

The models for estimating software reliability were:

i. Exponential
ii. Weibull
iii. Gamma
iv. Power
v. Logarithmic
vi. Inverse Linear

Apart from the above models several goodness of fit tests, maximal likelihood tests and other statistical tests were incorporated to aid user in deciding which is the best model for that particular data set. In addition to software and hardware reliability calculations, system reliability estimation was incorporated for both state dependent and independent systems.

In state dependent systems, following types of systems were implemented:

i. Markov Analysis

ii. Load Sharing System

iii. Standby Systems

iv. Standby Systems with switching failures

v. Three-component systems

For state independent system reliability estimation, tie set algorithm was used as described in [21] for given block diagram.

8.2 Future Work

The software developed was able to perform calculations for hardware reliability, software reliability, state dependent system reliability and state independent system reliability. However as the respective areas of application being very vast, the software can be enhanced in the following areas:

1. Developing a Graphical User Interface which will allow the user to
    - Build a reliability block diagram to show the network of components.
    - Dynamically add, move or delete components in a system.
    - Analyze the calculated system reliability by rearranging the network to enhance the reliability and decide on adding parallel or standby components.
    - Design a system with predefined reliability.
2. Develop a more efficient algorithm for tie set calculation.
3. Incorporate State Dependent Systems for n components from the existing two-component system using Markov analysis.
4. Implement other models for Software Reliability.
5. Implement Software Reliability models for Censored data
6. Provide a comprehensive Help function..

References

1 Ahluwalia R S, "Quality and Reliability Engineering", Lecture Notes, West Virginia University, 1998.

2 Ahluwalia R S, Parekh M, “Software Reliability Estimation using Exponential & Gamma Models”, Manufacturing Tech. Beyond 2000, Bangalore, India, 1999.

3 Boukas, Yang, “Cost effective maintenance management, Productivity improvement and Downtime reduction”, 1983 Noyes Publication.

4 Dimitri Kececioglu, “Reliability and Life Testing Handbook, Vol.1”, 1993, PTR Prentice-Hall, Inc.

5 Dungarpur J S and Jack N, “ Optimizing System Availability under Minimal Repair with Non-negligible Repair and Replacement”, 1993, Journal of Operational Research Society, Vol.44, No. 11, pp. 1097-1103.

6 Duane J T., “Learning Curve Approach to Reliability Monitoring”, IEEE Transactions on Aerospace Vol. 2 pp. 563-566, 1964.

7 DeMercado J, Spyratos N, and Bowen B A, “A Method for Calculation of Network Reliability,” IEEE Trans. Reliability, Vol. R-25 pp. 71-77, June 1976.

8 Ebeling C E, “ An Introduction to Reliability and Maintainability Engineering”, McGraw Hill, 1997.

9 Elsayed A. E, “Reliability Engineering”, Addison Wesley Longman, June 1996.

10 Flehinger B J, “System Reliability as a Function of System Age: Effects of Intermittent Component Usage and Periodic Maintenance”, 1960, Operations Research, 1960, Vol. 8, No. 1, pp. 30-44.

11 Gotkas, Yavuz, “Integrated Reliability, Availability, Maintainability cost modelling for the manufacturing design system”, PhD. Dissertation, West Virginia University, 1995.

12 Grady, R.B., and Casewell, D.L., “Software Metrics: Establishing A Company-Wide Program”, Prentice-Hall, Englewood Cliffs, New Jersy, 1987.

13 Hudson G R., “Program Errors as Birth and Death Process”, System Development Corporation, Report SP –3011, Santa-Monica, CA, 1967.

14 Igor A. Ushakov, and Robert A. Harrison, “Handbook of Reliability Engineering”, 1994, John Wiley and Sons.

15 Jelinski Z, and Moranda P B., “Software Reliability Research”, Statistical Computer

Performance Evaluation, Academic, New York, pp. 465-484, 1972.

16 Jardine A K S, “Maintenance, Replacement and Reliability”, 1973, John Wiley and Sons.

17 Littlewood B and Verrall J L., “A Bayesian Reliability Growth Model for Computer Software”, Journal Royal Statistical Society –Series C, 22(3), pp.332-346, 1973.

18 Moranda P B, “Predictions of Software Reliability During Debugging”, Proceedings Annual Reliability and Maintainability Symposium Washington DC, pp. 327-332, 1975.

19 Musa J D, “Validity of Execution-Time Theory of Software Reliability”, IEEE Transactions on Reliability, Vol. R-28, pp. 181-191, August 1979.

20 Musa J D, A Iannino, K Okumoto, “Software Reliability: Measurement, Prediction, Application”, McGraw- Hill, New- York, 1987.

21 Nelson A C, Batts J R, and Beadles R L, “A Computer Program Approximating System Reliability,” IEEE Trans. Reliability, Vol. R-19 pp. 6165, May 1970.

22 Nikora A P, "An Experiment in Determining Software Reliability Applicability", Toulouse, France, International Symposium on Software Reliability Engineering, October 1995.

23 Nikora A P, "A JPL Software Reliability study and a Windows-Based Software Reliability Tool", Claremount, California, USA, November 1993.

24 Parekh, M, “Development of Decision Support System for Reliability”, MS Thesis, West Virginia University, 1999.

25 Patrick D T O'Connor; “Practical Reliability Engineering”, John Wiley & Sons, 1991.

26 Raze J, Nelson J J, Simrad D J, Bradley M, “Reliability Models for Mechanical Equipment”, 1987 Proceedings Annual Reliability and Maintainability Symposium, pp. 130-133.

27 Schoutten F A and Vanneste S G, “ Simple Control Policies for a Multicomponent Maintenance Systems”, 1993, Operations Research Journal, Vol.41, No.6, pp. 1125-1136.

28 Shooman, M L; “Probabilistic Models for Software Reliability Prediction”, Statistical Computer Performance Evaluation, Academic, New York, pp. 485-502, 1972.

29 Schick G J, and Wolverton R W., “Assessment of Software Reliability”, Proceedings Operations Research, Physica-Verlag, Wurzburg-Wein, pp. 395-422.

30 Schick G J, and Wolverton R W., “An Analysis of Competing Software Reliability Models”, IEEE Transactions on Software Engineering, SE–(4), pp. 104-120, 1978.

31 Schneidewind N F., “An Approach to Software Reliability Prediction and Quality Control”, Fall joint Computer Conference, AFPIS conference Proceedings, 41 AFPIS press, Montvale, NJ, pp.837-847, 1972.

32 Schneidewind N F., “Analysis of Error Processes in Computer Software”, Proceedings 1975 International Conference on Reliable Software Los Angeles, pp. 337-346, 1975.

33 Schneider, D I, “An Introduction to Programming Using Visual Basic 6.0”, Prentice Hall, Fourth Edition, 1999.

34 Wagoner, W L; “The Final Report on Software Reliability Measurement Study”, Aerospace Corporation, Report TOR -0074(4112)-1.

35 Yuan, Kai, Cai, “Towards a conceptual framework of software run reliability modeling”, Information Sciences, July 2000, Vol. 126 Issue 1-4, p137, 27p.

Appendix I: Bartlett's Test for Exponential Distribution

A specific test for fitting exponential distribution is Bartlett's test [1]. The hypotheses are:

Ho: Failure times are exponential

Ha: Failure times are not exponential

The test statistic is

$$B \quad = \quad \frac{2.n.\left(\ln\left((1/n).\left(\sum_{i=1}^{n} t_i\right)\right) - (1/n).\sum_{i=1}^{n} \ln(t_i)\right)}{1+(n+1)/(6.n)}$$

Where t_i = time of failure of ith unit.

n = number of failures

the test statistic B under the null hypothesis has a chi-square distribution with n-1 degrees of freedom. In this test if,

$\chi^2(1-\alpha/2, n-1) < B < \chi^2(\alpha/2, n-1)$

then null hypothesis is accepted; otherwise the alternative hypothesis is accepted.

Appendix II: Kolmogorov-Smirnov Test

A goodness of fit test for Normal and Log normal distribution is Kolmogorov-Smirnov test [1]. The hypotheses for the test are:

Ho: The failure time are normal (lognormal)

Ha: The failure times are not normal (lognormal)

The test statistic is

D_n = max $\{D_1,D_2\}$

$$where \quad D_1 = \left(\phi\left(\frac{t_i - \bar{t}}{s}\right) - \frac{i-1}{n}\right) \quad and \quad D_2 = \left(\frac{1}{n} - \phi\left(\frac{t_i - \bar{t}}{s}\right)\right)$$

Where

$$\bar{t} = \sum_{i=1}^{n} \frac{t_i}{n} \quad and \quad s^2 = \frac{\sum_{i=1}^{n} (t_i - \bar{t})^2}{n-1}$$

If $D_n < D_{crit}$, then accept Ho otherwise accept Ha. The values of D_{crit} may be found in the standard Kolmogorov-Smirnov table.

Appendix III: Mann's Test for Weibull Distribution

A specific test for weibull failure distribution is a test developed by Mann, Schafer, and Singapurwalla [1]. The hypotheses are:

Ho: The failure times are of Weibull

Ha: The failure times are not of Weibull.

The test statistic is:

$$M = \frac{k_1 \sum_{i=k1+1}^{n-1} (\ln(t_{i+1}) - \ln(t_i))/M_i}{k_2 \sum_{i=1}^{k1} (\ln(t_{i+1}) - \ln(t_i))/M_i}$$

Where $k_1 = [n/2]$, $k_2 = [(n-1)/2]$,

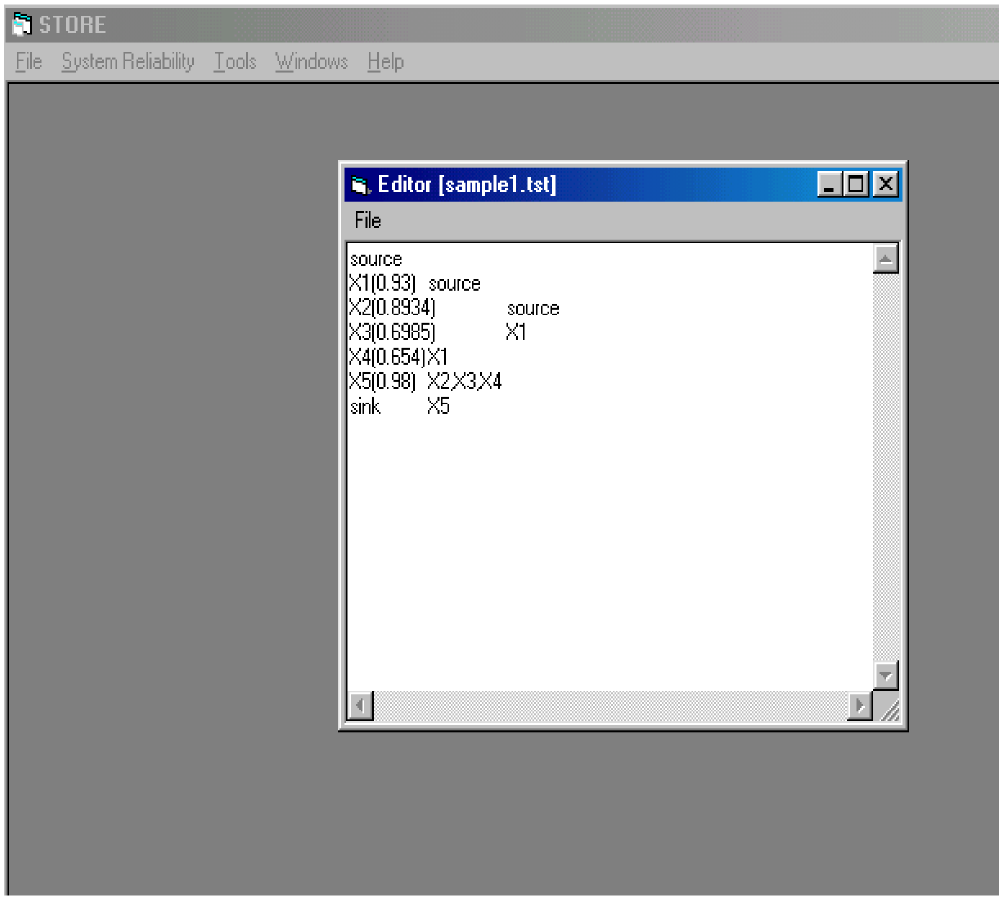
$M_i = Z_i + 1 - Z_i$

$$Z_i = \ln\left[-\ln\left(1 - \frac{1-0.5}{n+0.25} \right) \right]$$

And [x] is less than number *x*. M_i is an approximation. If $M > F_{crit}$, then Ha is accepted. Values for F_{crit} may be obtained from tables of F-distribution if one lets the number of degrees of freedom for numerator to be $2k_2$ and the number of degrees of freedom for the denominator be $2k_1$.

Appendix IV: Reliability Block Diagram Input

In STORE, a RBD of the system is inputted by stating component, its reliability and predecessor. Where source is the predecessor of X1 and X1 is of X3. The value in parentheses is the reliability value of the component. The format is as shown in figure below.



```
source
X1(0.93)  source
X2(0.8934)          source
X3(0.6985)          X1
X4(0.654)X1
X5(0.98)  X2,X3,X4
sink      X5
```

Appendix V: Source Code

<u>Opening Menu</u>

```
Option Explicit
' Cd drive variable
Private Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal
lpstrCommand As String, ByVal lpstrReturnString As String, ByVal uReturnLength As Long,
ByVal hwndCallback As Long) As Long
'editor variables
Private FileName As String  ' The full file name.
Private FileTitle As String ' The file name without path.
Private DataModified As Boolean
Private Sub mnuarrangeicons_Click()
frmret.Arrange vbArrangeIcons
End Sub

Private Sub mnucascade_Click()
frmret.Arrange vbCascade
End Sub

Private Sub mnucdclose_Click()
 mciSendString "Set CDAudio Door Closed Wait", _
        0&, 0&, 0&
End Sub

Private Sub mnucdopen_Click()
   mciSendString "Set CDAudio Door Open Wait", _
        0&, 0&, 0&
End Sub

Private Sub mnucensored_Click()
ExpModel.Show
End Sub

Private Sub mnuchi_Click()
frmchisquare.Show
End Sub

Private Sub mnucomplete_Click()
frmhardware1.Show
End Sub

Private Sub mnuexit_Click()
Beep
End
End Sub

Private Sub mnufisher_Click()
frmfisher.Show
End Sub
Private Sub mnuForm1_Click()
frmhelp1.Show
End Sub

Private Sub mnuks_Click()
frmks.Show
End Sub
```

```
Private Sub mnunewfile_Click()
editorform.Show
End Sub

Private Sub mnupdetail_Click()
On Error GoTo 10
frmresult.PrintForm
MsgBox "Done !", 64, vbExclamation
Exit Sub
10
MsgBox "There was problem printing to yout printer", vbExclamation

End Sub

Private Sub mnuphardware1_Click()

On Error GoTo 1

frmhardware1.PrintForm
MsgBox "Done !", 64, vbExclamation
Exit Sub
1
MsgBox "There was problem printing to yout printer", vbExclamation

End Sub
Private Sub mnucens_Click()

On Error GoTo 1

ExpModel.PrintForm
MsgBox "Done !", 64, vbExclamation
Exit Sub
1
MsgBox "There was problem printing to yout printer", vbExclamation
End Sub

Private Sub mnupsumm_Click()
On Error GoTo 10
software.PrintForm
MsgBox "Done !", 64, vbExclamation
Exit Sub
10
MsgBox "There was problem printing to yout printer", vbExclamation
End Sub

Private Sub mnupsystem_Click()
On Error GoTo 10
System.PrintForm
MsgBox "Done !", 64, vbExclamation
Exit Sub
10
MsgBox "There was problem printing to yout printer", vbExclamation
End Sub

Private Sub mnuptieset_Click()
On Error GoTo 11
Tieset.PrintForm
MsgBox "Done !", 64, vbExclamation
```

```
        Exit Sub
11
MsgBox "There was problem printing to yout printer", vbExclamation
End Sub

Private Sub mnurcal_Click()
frmhardware.Show
End Sub

Private Sub mnusoftware_Click()
software.Show
End Sub

Private Sub mnusysinde_Click()
Tieset.Show
End Sub

Private Sub mnusysstate_Click(index As Integer)
System.Show
End Sub

Private Sub mnutile_Click()
frmret.Arrange vbTileVertical
End Sub

Private Sub Picture1_LostFocus()
Picture1.Visible = False
End Sub
```

<u>Data Editor</u>

```
Option Explicit
Private FileName As String  ' The full file name.
Private FileTitle As String ' The file name without path.

Private DataModified As Boolean

' Return True if the data is safe.
Private Function DataSafe() As Boolean
‘ No problem if the data is unmodified.
If Not DataModified Then
DataSafe = True
Exit Function
End If

‘ See if the user wants to save changes.
Select Case MsgBox("The data has been modified. Do you want to save the changes?", _
vbYesNoCancel)
Case vbYes
' Save the data. Procedure SaveData
' will reset DataModified.
mnuesaveas_Click
DataSafe = Not DataModified

Case vbNo
 ' Discard the changes to the data.
DataSafe = True

Case vbNo
```

```
        ' Cancel.
DataSafe = False
End Select
End Function

' Load data from the file.
'
' @@@ Modify to load data of the correct format.
Private Sub LoadData(ftitle As String, fname As String)
Dim fnum As Integer

' Open the file.
fnum = FreeFile
Open fname For Input As fnum

' Read all the bytes in the file into the TextBox.
EditorText.Text = Input(LOF(fnum), fnum)

' Close the file.
Close fnum

' Save the file name and title.
FileTitle = ftitle
FileName = fname

' Make sure the caption gets updated.
DataModified = True
SetDataChanged False
End Sub
' Save data into the file.
' @@@ Modify to save data in the correct format.
Private Sub SaveData(ftitle As String, fname As String)
Dim fnum As Integer

' Open the file.
fnum = FreeFile
Open fname For Output As fnum

' Write text from the TextBox into the file.
Print #fnum, EditorText.Text

‘ Close the file.
Close fnum

‘ Save the file name and title.
FileTitle = ftitle
FileName = fname

' Make sure the caption gets updated.
DataModified = True
SetDataChanged False
End Sub
' Set DataModified. Display an asterisk in the
' form's Caption next to the file name if
' appropriate.
Private Sub SetDataChanged(changed As Boolean)
' Don't bother if it's already been done.
If DataModified = changed Then Exit Sub
```

```
        DataModified = changed
If changed Then
Caption = "Editor*[" & FileTitle & "]"
Else
Caption = "Editor [" & FileTitle & "]"
End If
End Sub

' Set the file dialog's path for the next time.
Private Sub SetDialogPath()
Dim file_path As String

' Remove characters from the right until the
' path ends in \.
file_path = filedialog.FileName
Do While Right$(file_path, 1) <> "\"
file_path = Left$(file_path, Len(file_path) - 1)
Loop

filedialog.InitDir = file_path

‘ Save the directory in the registry.
' @@@ Change the application and section names.
SaveSetting "SimpleEditor", "Directories", _
"SaveDir", filedialog.InitDir
End Sub

' Mark the data as modified.
' @@@ Call DataChanged whenever the user
' @@@ changes the data.
Private Sub EditorText_Change()
SetDataChanged True
End Sub

Private Sub Form_Load()
Dim wid As Single
Dim hgt As Single

' Get the last directory the program accessed.
' If there is no entry, use the App.Path.
' @@@ Change the application and section names.
filedialog.InitDir = GetSetting( _
"SimpleEditor", "Directories", _
"SaveDir", App.path)
End Sub

' Make sure the data is safe to unload.
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
Cancel = Not DataSafe
End Sub
Private Sub Form_Resize()
EditorText.Move 0, 0, ScaleWidth, ScaleHeight
End Sub

' Unload the form.
Private Sub mnueexit_Click()
' Note: The QueryUnload event handler checks
' that the data is safe.
Unload Me
```

```
        End Sub

' Start a new file.
Private Sub mnuenew_Click()
' Make sure the existing data is safe.
If Not DataSafe Then Exit Sub
' @@@ Do whatever is necessary to start a
' @@@ new document.
EditorText.Text = ""

' Save the file name and title.
FileTitle = ""
FileName = ""

' Make sure the caption gets updated.
DataModified = True
SetDataChanged False
End Sub

' Open a file.
Private Sub mnueopen_Click()
' Make sure the existing data is safe.
If Not DataSafe Then Exit Sub

' Start in the application directory.
If filedialog.InitDir = "" Then _
filedialog.InitDir = App.path

' @@@ Set desired flags.
filedialog.Flags = cdlOFNFileMustExist + _
cdlOFNHideReadOnly + _
cdlOFNLongNames

' @@@ Set desired filters.
filedialog.Filter = "hardware (*.hrd)|*.hrd |"
filedialog.Filter = filedialog.Filter + "software (*.sft)|*.sft |"
filedialog.FilterIndex = 0

' Let the user select the file to open.
On Error Resume Next
filedialog.ShowOpen
If Err.Number = cdlCancel Then
' The user canceled.
Exit Sub
ElseIf Err.Number <> 0 Then
MsgBox "Error" & Str$(Err.Number) & " selecting file." & _
vbCrLf & Err.Description
Exit Sub
End If
On Error GoTo 0

' Set the dialog path for next time.
SetDialogPath

' Load the data.
LoadData filedialog.FileTitle, filedialog.FileName
End Sub
```

```
        ' Save using the current file name.
Private Sub mnuesave_Click()
' If there's no file name, treat as Save As.
If FileTitle = "" Then
mnuesaveas_Click
Exit Sub
End If

' Save the data using the current file name.
SaveData FileTitle, FileName
End Sub

' Save using a new file name.
Private Sub mnuesaveas_Click()
' Start in the application directory.
If filedialog.InitDir = "" Then _
filedialog.InitDir = App.path

' @@@ Set desired flags.
filedialog.Flags = cdlOFNPathMustExist + _
cdlOFNHideReadOnly + _
cdlOFNLongNames

' @@@ Set desired filters.
filedialog.Filter = "hardware (*.hrd)|*.hrd | "
filedialog.Filter = filedialog.Filter + "software (*.sft)|*.sft |"
filedialog.Filter = filedialog.Filter + "Tieset Matrix (*.tst)|*.tst|"
filedialog.Filter = filedialog.Filter + "Cutset Matrix (*.cst)|*.cst|"
filedialog.Filter = filedialog.Filter + "Reliability Matrix (*.rel)|*.rel|"
filedialog.FilterIndex = 1

' Let the user select the file to open.
On Error Resume Next
filedialog.ShowSave
If Err.Number = cdlCancel Then
‘ The user canceled.
Exit Sub
ElseIf Err.Number <> 0 Then
MsgBox "Error" & Str$(Err.Number) & " selecting file." & _
vbCrLf & Err.Description
Exit Sub
End If
On Error GoTo 0

' Set the dialog path for next time.
SetDialogPath

' Load the data.
SaveData filedialog.FileTitle, filedialog.FileName
End Sub
```

State Independent System Reliability

```
Dim hConsole As Long
Private Sub cmdopen_Click()
End Sub
```

```
        Private Sub cmdclear_Click()
RBDText.Text = ""
TieTxt.Text = ""
relTxt.Text = ""
txtfilename2.Text = ""
End Sub

Private Sub Command1_Click()
If Text1.Text = "" Then
MsgBox "You must Open the Data File", vbExclamation
Exit Sub
End If
Shell Text1.Text, vbHide
End Sub

Private Sub Command2_Click()
On Error GoTo err_handle1
Inp2 = path + "Manish.dat"
Inp3 = path + "Manish1.dat"
Dim fnum As Integer
' Open the tieset file.
fnum = FreeFile
Open Inp2 For Input As fnum
' Read all the bytes in the file into the TextBox.
TieTxt.Text = Input(LOF(fnum), fnum)
' Close the file.
 Close fnum
' Open the Reliability file.
Open Inp3 For Input As 2
relTxt.Text = Input(LOF(2), 2)
Close 2
err_handle1:
If Err = 53 Then
MsgBox "Reliability not Calculated!"
End If
End Sub

Private Sub fopen_Click()
Dim data(100) As String, i As Integer
If txtfilename2.Text = "" Then
MsgBox "You must Select a Data File!"
Exit Sub
End If
Dim Inp1 As String
pathname = dirdirectoryh2.path
If Right(dirdirectoryh2.path, 1) <> "\" Then
path = dirdirectoryh2 + "\"
Inp1 = path + txtfilename2.Text
Text1.Text = "c:\manish\reliability\manish " + Inp1
Else
Inp1 = pathname + txtfilename2.Text
End If
Dim fnum As Integer
' Open the file.
fnum = FreeFile
Open Inp1 For Input As fnum
' Read all the bytes in the file into the
' TextBox.
RBDText.Text = Input(LOF(fnum), fnum)
```

```
        ' Close the file.
Close fnum
End Sub

Private Sub dirdirectoryh2_Change()
'Here we change the files according to the directory
filfilesh2.path = dirdirectoryh2.path
End Sub
Private Sub drvdriveh2_Change()
'The next statement checks the error in the path
On Error GoTo driveerror
'change the path of the new drive
dirdirectoryh2.path = drvdriveh2.Drive
Exit Sub
driveerror:
' Here we restore the original drive
MsgBox "Device is not ready!", vbCritical, "Error"
drvdriveh2.Drive = dirdirectoryh2.path
End Sub

Private Sub filfilesh2_Click()
txtfilename2.Text = filfilesh2.FileName
End Sub

Private Sub Form_Load()
If AllocConsole() Then
hConsole = GetStdHandle(STD_OUTPUT_HANDLE)
If hConsole = 0 Then MsgBox "Couldn't allocate STDOUT"
Else
MsgBox "Couldn't allocate console"
End If
frmret.mnuptieset.Enabled = True
End Sub

Private Sub Form_Unload(Cancel As Integer)
CloseHandle hConsole
FreeConsole
End Sub

Private Sub Option2_Click()
knsys.Show
End Sub
```

State Dependent System Reliability

```
Private Sub Cal_Click()
'Stand by sytem with failure rate for online and offline
rel_SB = (Exp(-Val(txtFR_SB(2).Text) * Val(Time_t(0).Text))) + (Val(txtFR_SB(2).Text) /
(Val(txtFR_SB(2).Text) + Val(trans4(0).Text) - Val(trans5(0).Text))) * (Exp(-Val(trans5(0).Text) *
Val(Time_t(0).Text)) - Exp(-(Val(txtFR_SB(2).Text) + Val(trans4(0).Text)) * Val(Time_t(0).Text)))
rel3.Caption = Format(Str(rel_SB), "0.000")
mttf_SB = (1 / Val(txtFR_SB(2).Text)) + (Val(txtFR_SB(2).Text) / (Val(trans5(0).Text) *
(Val(txtFR_SB(2).Text) + Val(trans4(0).Text))))
mttf2.Caption = Format(Str(mttf_SB), "0.000")
End Sub

Private Sub Cal1_Click()
' Standby Systems with reduced failure rate
Dim mttf_SB As Double, rel_SB As Double
```

```
    rel_SB = (Exp(-Val(txtFR_SB(1).Text) * Val(Time_t(1).Text))) + (Val(txtFR_SB(1).Text) /
Val(trans4(1).Text)) * (Exp(-Val(txtFR_SB(1).Text) * Val(Time_t(1).Text)) - Exp(-
(Val(txtFR_SB(1).Text) + Val(trans4(1).Text)) * Val(Time_t(1).Text)))
mttf_SB = (1 / Val(txtFR_SB(1).Text)) + (1 / (Val(txtFR_SB(1).Text) + Val(trans4(1).Text)))
rel4.Caption = Format(Str(rel_SB), "0.000")
mttf3.Caption = Format(Str(mttf_SB), "0.000")
End Sub

Private Sub Cal2_Click()
a = -((Val(trans1(0).Text) + Val(trans2(0).Text)) * Val(Time_t(2).Text))
Relser = Exp(a)
rel.Caption = Format(Str(Relser), "0.000")
b = Exp(-Val(trans1(0).Text) * Val(Time_t(2).Text)) + Exp(-Val(trans2(0).Text) *
Val(Time_t(2).Text)) - Exp(-Val((trans1(0).Text) + Val(trans2(0).Text)) * Val(Time_t(2).Text))
relpar.Caption = Format(Str(b), "0.000")
End Sub

Private Sub Cal3_Click()
' Load sharing System
Dim mttf_LS As Double, rel_LS As Double
rel_LS = Exp(-2 * Val(trans1(1).Text) * Val(Time_t(3).Text)) + (2 * Val(trans1(1).Text)) / (2 *
Val(trans1(1).Text) - Val(Trans3.Text)) * (Exp(-Val(Trans3.Text) * Val(Time_t(3).Text)) - Exp(-2 *
Val(trans1(1).Text) * Val(Time_t(3).Text)))
rel5.Caption = Format(Str(rel_LS), "0.000")
mttf_LS = (1 / (2 * Val(trans1(1).Text))) + (2 * Val(trans1(1).Text)) / (2 * Val(trans1(1).Text) -
Val(Trans3.Text)) * ((1 / Val(Trans3.Text)) - (1 / (2 * Val(trans1(1).Text))))
mttf4.Caption = Format(Str(mttf_LS), "0.000")
End Sub

Private Sub Command2_Click()
'Standby system with switching failure
rel_SB = (Exp(-Val(txtFR_SB(0).Text) * Val(Time_t(4).Text))) + (((1 - Val(txtProb.Text)) *
Val(txtFR_SB(0).Text)) / (Val(txtFR_SB(0).Text) + Val(trans4(2).Text) - Val(trans2(1).Text))) *
(Exp(-Val(trans2(1).Text) * Val(Time_t(4).Text)) - Exp(-(Val(txtFR_SB(0).Text) +
Val(trans4(2).Text)) * Val(Time_t(4).Text)))
rel2.Caption = Format(Str(rel_SB), "0.000")
End Sub

Private Sub Command5_Click()
'Three component stsandby system
rel_SB = Exp(-Val(trans1(2).Text) * Val(Time_t(5).Text)) * (1 + (Val(trans1(2).Text) *
Val(Time_t(5).Text)) + ((Val(trans1(2).Text)) ^ 2 * (Val(Time_t(5).Text)) ^ 2) / 2)
mttf_SB = 3 / Val(trans1(2).Text)
rel1.Caption = Format(Str(rel_SB), "0.000")
mttf1.Caption = Format(Str(mttf_SB), "0.000")
End Sub
Private Sub Form_Load()
frmret.mnupsystem.Enabled = True
End Sub
```

Hardware Reliability Form

```
Option Explicit
Dim File_Name As String

Private Sub cmdImport_Click()
On Error GoTo err_handle
Dim i As Single, pathname As String, path As String
Dim Inp1 As String, noofdata As Double, n As Integer
```

```
        Dim j As Integer, k As Integer, l As Integer
Open File_Name For Input As #1
'calculating number of enteries
n = 0
Do While Not EOF(1)
Input #1, noofdata
n = n + 1
Loop
Close #1
Open File_Name For Input As #1
i = 0
Do While Not EOF(1)
i = i + 1
Input #1, failure(i)
Loop
Close #1
txtnumberoffailures1.Text = i
noOffailures.Text = i
For i = 1 To n
gridexpo.Col = 0
gridexpo.Row = i
gridexpo.Text = i
gridexpo.Col = 1
gridexpo.Row = i
gridexpo.Text = Val(failure(i))
Next i
err_handle:
If Err = 75 Then
MsgBox "Please Select a File to Open!"
End If
End Sub

Private Sub cmdopen2_Click()
With commondc
.Filter = "Text Files(*.txt)|*.txt|" + "Hardware Files(*.hrd)|*.hrd|" + "Software Files(*.sft)|*.sft|" + "All
Files(*.*)|*.*|"
.ShowOpen
File_Name = .FileName
End With
End Sub

Private Sub end_Click()
End
End Sub

Private Sub Command1_Click()
Dim p1 As Double

If txtalpha.Text = "" Then
MsgBox "Input the level of significance", vbCritical
Exit Sub
End If
p1 = Val(txtalpha.Text)
If IsNumeric(txtalpha.Text) Then
p1 = Val(txtalpha.Text)
If p1 < 0 Or p1 > 1 Then
MsgBox " invalid number"
Exit Sub
End If
```

```
        Else
MsgBox "Invalid number"
Exit Sub
End If
If expT1 = True Then
Dim cal_p As Double, p(2) As Double
Dim dof As Integer
Dim x(2) As Double
Dim k As Double, t As Double, a As Double
Dim v As Double, dv As Double
For i = 1 To 2
p(1) = p1 / 2
p(2) = 1 - p1 / 2
n = Val(noOffailures.Text)
dof = n - 1
v = 0.5
dv = 0.5
x(i) = 0
Do While (dv > 0.000001)
x(i) = 1 / v - 1
dv = dv / 2
cal_p = Exp(-0.5 * x(i))
If dof Mod 2 > 0 Then
cal_p = cal_p * Sqr(2 * x(i) / 3.14)
End If
k = dof
Do While (k > 2 Or k = 2)
cal_p = cal_p * (x(i) / k)
k = k - 2
Loop
t = cal_p
a = dof
Do While (t > 0.000001 * p(i))
a = a + 2
t = t * (x(i) / a)
cal_p = cal_p + t
Loop
If (1 - cal_p) > 1 - p(i) Then
v = v - dv
Else
v = v + dv
End If
Loop
Next i
Dim b As Double
b = Val(txtbart.Text)
If x(1) < b And b < x(2) Then
lbldes.Caption = "As calculated Bartlett's test statistic," + Str(Format(b, "#.000")) + ", is falls
between the  critical chi square values," + Str(Format(x(1), "0.000")) + "and" + Str(Format(x(2),
"0.000")) + ", hypothesis that data belongs to exponential distribution is accepted"
Else
lbldes.Caption = "As calculated Bartlett's test statistic," + Str(Format(b, "0.000")) + ", does not
falls between the  critical chi square values," + Str(Format(x(1), "0.000")) + "and" +
Str(Format(x(2), "0.000")) + ", hypothesis that data belongs to exponential distribution is  not
accepted"
End If
Exit Sub
End If
If weib = True Then
```

```
        Dim n1 As Double, n2 As Double, k1 As Double, k2 As Double, f As Double
Dim ff As Double
n = Val(noOffailures.Text)
k1 = Int(n / 2)
k2 = Int((n - 1) / 2)
p1 = Val(txtalpha.Text)
n1 = 2 * k1
n2 = 2 * k2
ff = afishf(p1, n1, n2)
If ff < mann Then
lbldes.Caption = "As critical F-value," + Str(Format(afishf(p1, n1, n2), "0.000")) + ", is less than
the calculated Mann's test statistic," + Str(Format(mann, "#.000")) + ", hypothesis that data
belongs to weibull distribution is not accepted"
Else
lbldes.Caption = "As calculated Mann's test statistic," + Str(Format(mann, "#.000")) + ", is less
than the  critical F-value," + Str(Format(afishf(p1, n1, n2), "0.000")) + ", hypothesis that data
belongs to weibull distribution is accepted"
End If
End If
If NormOpt = True Then
Dim ks2 As Double
n = Val(noOffailures.Text)
p1 = Val(txtalpha.Text)
If (p1 = 0.2 Or p1 = 0.15 Or p1 = 0.1 Or p1 = 0.05 Or p1 = 0.01) And (n >= 4 Or n <= 20 Or n = 25
Or n = 30) Then
If p1 = 0.2 Then
If n >= 4 And n <= 20 Then
frmks.grdks.Col = 1
frmks.grdks.Row = n - 3
ks2 = Val(frmks.grdks.Text)
Else
If n = 25 Then
frmks.grdks.Col = 1
frmks.grdks.Row = 18
ks2 = Val(frmks.grdks.Text)
End If
If n = 30 Then
frmks.grdks.Col = 1
frmks.grdks.Row = 18
ks2 = Val(frmks.grdks.Text)
End If
If n > 30 Then
ks2 = 0.736 / Sqr(n)
End If
End If
End If
If p1 = 0.15 Then
If n >= 4 And n <= 20 Then
frmks.grdks.Col = 2
frmks.grdks.Row = n - 3
ks2 = Val(frmks.grdks.Text)
Else
If n = 25 Then
frmks.grdks.Col = 2
frmks.grdks.Row = 18
ks2 = Val(frmks.grdks.Text)
End If
If n = 30 Then
frmks.grdks.Col = 2
```

```
        frmks.grdks.Row = 18
ks2 = Val(frmks.grdks.Text)
End If
If n > 30 Then
ks2 = 0.768 / Sqr(n)
End If
End If
End If
If p1 = 0.1 Then
If n >= 4 And n <= 20 Then
frmks.grdks.Col = 3
frmks.grdks.Row = n - 3
ks2 = Val(frmks.grdks.Text)
Else
If n = 25 Then
frmks.grdks.Col = 3
frmks.grdks.Row = 18
ks2 = Val(frmks.grdks.Text)
End If
If n = 30 Then
frmks.grdks.Col = 3
frmks.grdks.Row = 18
ks2 = Val(frmks.grdks.Text)
End If
If n > 30 Then
ks2 = 0.805 / Sqr(n)
End If
End If
End If
If p1 = 0.05 Then
If n >= 4 And n <= 20 Then
frmks.grdks.Col = 4
frmks.grdks.Row = n - 3
ks2 = Val(frmks.grdks.Text)
Else
If n = 25 Then
frmks.grdks.Col = 4
frmks.grdks.Row = 18
ks2 = Val(frmks.grdks.Text)
End If
If n = 30 Then
frmks.grdks.Col = 4
frmks.grdks.Row = 18
ks2 = Val(frmks.grdks.Text)
End If
If n > 30 Then
ks2 = 0.886 / Sqr(n)
End If
End If
End If
If p1 = 0.01 Then
If n >= 4 And n <= 20 Then
frmks.grdks.Col = 5
frmks.grdks.Row = n - 3
ks2 = Val(frmks.grdks.Text)
Else
If n = 25 Then
frmks.grdks.Col = 5
frmks.grdks.Row = 18
```

```
        ks2 = Val(frmks.grdks.Text)
End If
If n = 30 Then
frmks.grdks.Col = 5
frmks.grdks.Row = 18
ks2 = Val(frmks.grdks.Text)
End If
If n > 30 Then
ks2 = 1.031 / Sqr(n)
End If
End If
End If
Else
MsgBox "Input alpha value from 0.2,0.15,0.10,0.05,0.01 or data not available for n =
21,22,23,24,26,27,28,29"
Exit Sub
End If
If NormOpt = True Then
Dim k_s1 As Double
k_s1 = Val(txtbart.Text)
If k_s1 < ks2 Then
lbldes.Caption = "As calculated Komogorov -Smirnov value," + Str(Format(k_s1, "0.000")) + ", is
less than the critical Kolmogorov-Smirnov value," + Str(Format(ks2, "#.000")) + ", hypothesis that
data belongs to Normal distribution is accepted"
Else
lbldes.Caption = "As critical Komogorov -Smirnov value," + Str(Format(ks2, "0.000")) + ", is less
than the calculated Kolmogorov-Smirnov value ," + Str(Format(k_s1, "#.000")) + ", hypothesis that
data belongs to Normal distribution is  not accepted"
End If
End If
End If
End Sub

Function fishf(f As Double, n1 As Double, n2 As Double)
Dim v As Double, dv As Double, k1 As Double, k2 As Double
Dim x As Double
Dim th As Double, a As Double, sth As Double, cth As Double, b As Double, b1 As Double
Dim c As Double, pi As Double, pid2 As Double, k As Double
pi = 3.14159265358979
pid2 = pi / 2
k1 = Int(n / 2)
k2 = Int((n - 1) / 2)
n1 = 2 * k1
n2 = 2 * k2
x = n2 / (n1 * f + n2)
If n1 Mod 2 = 0 Then
fishf = statcom(1 - x, n2, n1 + n2 - 4, n2 - 2) * (x ^ (n2 / 2))
Exit Function
End If
If n2 Mod 2 = 0 Then
fishf = 1 - statcom(x, n1, n1 + n2 - 4, n1 - 2) * ((1 - x) ^ (n1 / 2))
Exit Function
End If
th = Atn(Sqr(n1 * f / n2))
a = th / pid2
sth = Sin(th)
cth = Cos(th)
If n2 > 1 Then
a = a + (sth * cth * (statcom(cth * cth, 2, n2 - 3, -1)) / pid2)
```

```
        End If
If n1 = 1 Then
fishf = 1 - a
End If
c = 4 * statcom(sth * sth, n2 + 1, n1 + n2 - 4, n2 - 2) * sth * ((cth ^ n2) / pi)
If n2 = 1 Then
fishf = 1 - a + c / 2
End If
k = 2
Do While (k <= (n2 - 1) / 2)
c = c * k / (k - 0.5)
k = k + 1
Loop
fishf = 1 - a + c
End Function
Function statcom(q, i, j, b)
Dim zz As Double, z As Double, k As Double
zz = 1
z = zz
k = i
Do While (k <= j)
zz = zz * q * k / (k - b)
z = z + zz
k = k + 2
Loop
statcom = z '( returns z)
End Function
Function afishf(p1 As Double, n1 As Double, n2 As Double)
Dim v As Double, dv As Double, f As Double, k1 As Double, k2 As Double
p1 = Val(txtalpha.Text)
v = 0.5
dv = 0.5
f = 0
Do While (dv > 0.000001)
f = 1 / v - 1
dv = dv / 2
If (fishf(f, n1, n2)) > p1 Then
v = v - dv
Else
v = v + dv
End If
Loop
afishf = f '(returns f)
End Function

Private Sub Command2_Click()
Dim k As Integer, j As Integer
k = Val(noOffailures.Text)
For j = 1 To k
gridexpo.Col = 0
gridexpo.Row = j
gridexpo.Text = ""
Next j
For j = 1 To k
gridexpo.Col = 1
gridexpo.Row = j
gridexpo.Text = ""
Next j
Label13.Visible = False
```

```
        Text1.Visible = False
Label14.Visible = False
Label11.Visible = False
Label12.Visible = False
txtbart.Visible = False
txtmttf.Text = ""
txtcor.Text = ""
txthrel1.Text = ""
txtmtime.Text = ""
noOffailures.Text = ""
noOfrisks.Text = ""
txtnumberoffailures1.Text = ""
testtime.Text = ""
lbldes.Caption = ""
Command1.Visible = False
txtalpha.Visible = False
End Sub

Private Sub expT1_Click()
If expT1 = True Then
labexpt1.Visible = True
Label2.Visible = True
Label4.Visible = True
Label5.Visible = True
noOfrisks.Visible = True
noOffailures.Visible = True
testtime.Visible = True
End If
End Sub

Private Sub swap(x, Y)
Dim temp As Single
temp = x
x = Y
Y = temp
End Sub

Public Sub swapdata(l As Integer)
Dim failure(1 To 100) As Double
Call swap(failure(l), failure(l + 1))
End Sub

Private Sub Form_Load()
frmret.mnuphardware.Enabled = True
End Sub

Private Sub rel_Click()
'Exponential with type I data
If Val(noOffailures.Text) > (txtnumberoffailures1.Text) Then
MsgBox "You do not have enough failure data!"
Exit Sub
End If
If expT1.Value = True Then
Dim lambda As Double, sumb1 As Double, sumb2 As Double, n As Integer
Dim mttf As Double, mtime As Double, hrel As Double, i As Integer
Dim sst As Double, tottesttime As Double
Dim fj As Double, uj As Double, sumb3 As Double, sumb4 As Double
Dim sumb5 As Double, SSE As Double, b1 As Double
Dim df As Integer, ssu As Double, sstu As Double
```

```
        Dim Estderror As Double, cor As Double, TS As Double
Dim b As Double, sumb6 As Double
n = Val(noOffailures.Text)
sumb1 = 0
sumb2 = 0
sumb3 = 0
sumb4 = 0
sumb5 = 0
sumb6 = 0
df = noOffailures.Text - 2
For i = 1 To n
gridexpo.Col = 1
gridexpo.Row = i
sumb1 = sumb1 + gridexpo.Text
sumb2 = sumb2 + (gridexpo.Text) ^ 2
fj = (i - 0.3) / (noOfrisks.Text + 0.4)
uj = Log(1 / (1 - fj))
sumb3 = sumb3 + uj
sumb4 = sumb4 + uj ^ 2
sumb5 = sumb5 + gridexpo.Text * uj
sumb6 = sumb6 + Log(gridexpo.Text)
Next i
b1 = sumb5 / sumb2
sst = sumb2 - ((sumb1) ^ 2 / noOfrisks.Text)
ssu = sumb4 - (sumb3) ^ 2 / noOfrisks.Text
sstu = sumb5 - (sumb1 * sumb3) / noOfrisks.Text
SSE = ssu - b1 * sstu
Estderror = (SSE / df) ^ 0.5
cor = sstu / ((sst * ssu) ^ 0.5)
TS = b1 * (sst) ^ 0.5 / Estderror
txtcor.Text = Format(Str(cor), "0.000")
'reliability calculation
mtime = Val(txtmtime.Text)
tottesttime = sumb1 + (noOfrisks.Text - noOffailures.Text) * testtime.Text
lambda = Val(noOffailures.Text) / tottesttime
mttf = 1 / lambda
txtmttf.Text = Format(Str(mttf), "0.000")
hrel = (Exp(-lambda * mtime))
If hrel < 0 Then
txthrel1.Text = 0
Else
If hrel > 1 Then
txthrel1.Text = 1
Else
txthrel1.Text = Format(Str(hrel), "0.000")
End If
End If
'bartletts' test
b = (2 * noOffailures.Text * (Log((1 / noOffailures.Text) * sumb1) - (1 / noOffailures.Text) *
sumb6)) / (1 + (noOffailures.Text + 1) / (6 * noOffailures.Text))
If expT1.Value = True Then
Label12.Visible = False
Label15.Visible = False
txtbart.Visible = True
Label11.Visible = True
Label13.Visible = True
Label14.Visible = True
Text1.Visible = True
Label13.Caption = "Distribution selected: Exponential"
```

```
        Text1.Text = " Bartletts test value is then compared with the Chisquared value (also known
as critical value) with degrees of freedom" + Str(df) + " and level of significance alhpa (user
decides this value) from the standard table. If the Bartlett's test value is less than the critical value
than the data is good fit for exponential distribution"
txtbart.Text = Format(Str(b), "0.000")
Label14.Caption = "Failure Rate (Lambda) =" + Format(Str(lambda), "0.00000")
txtalpha.Visible = True
Command1.Visible = True
End If

'****Weibull Distribution
Else
If weib.Value = True Then
Dim sumw1 As Double, sumw2 As Double, sumw3 As Double
Dim sumw4 As Double, sumw5 As Double
Dim df1 As Integer, i1 As Integer, fj1 As Double, yj1 As Double
Dim bw1 As Double, ssx As Double, ssy As Double, ssxy As Double, sumw6 As Double
Dim TS1 As Double, cor1 As Double, Estderror1 As Double, sse1 As Double
Dim sumw7 As Double, m As Integer, aw1 As Double, eta As Double, SD As Double
Dim z1(100) As Double, m1(1000) As Double, num As Double, sdata(1000) As Double, den As
Double
Dim i3 As Integer, num1 As Double, den1 As Double, num2 As Integer, i2 As Integer
Dim data(1000000) As Double, l As Integer, min2 As Double, lamdab As Double
df1 = noOffailures.Text - 2
sumw1 = 0
sumw2 = 0
sumw3 = 0
sumw4 = 0
sumw5 = 0
sumw6 = 0
sumw7 = 0
n = Val(noOffailures.Text)
m = Val(noOfrisks)
For i1 = 1 To n
gridexpo.Col = 1
gridexpo.Row = i1
data(i1) = gridexpo.Text
sumw1 = sumw1 + Log(gridexpo.Text)
sumw2 = sumw2 + (Log(gridexpo.Text)) ^ 2
fj1 = (i1 - 0.3) / (noOffailures.Text + 0.4)
yj1 = Log(Log(1 / (1 - fj1)))
sumw3 = sumw3 + yj1
sumw4 = sumw4 + yj1 ^ 2
sumw5 = sumw5 + Log(gridexpo.Text) * yj1
sumw7 = sumw7 + gridexpo.Text * yj1
Next i1
min2 = 10000000
For m = 1 To n
For l = 1 To n
If data(l) < min2 Then
min2 = data(l)
index = l
End If
Next l
sdata(m) = min2
data(index) = 1000000
min2 = 10000000
Next m
For i2 = 1 To n
```

```
        gridexpo.Col = 1
gridexpo.Row = i2
SD = SD + (Log(gridexpo.Text) - (sumw1 / n)) ^ 2
Next i2
bw1 = (sumw5 - (n * (sumw3 / n) * (sumw1 / n))) / (SD)
aw1 = (sumw3 / n) - (bw1 * (sumw1 / n))
eta = Exp(-aw1 / bw1)
ssx = sumw2 - ((sumw1) ^ 2 / noOffailures.Text)
ssy = sumw4 - (sumw3) ^ 2 / noOffailures.Text
ssxy = sumw5 - (sumw1 * sumw3) / noOffailures.Text
sse1 = ssy - bw1 * ssxy
Estderror1 = (sse1 / df1) ^ 0.5
cor1 = ssxy / ((ssx * ssy) ^ 0.5)
TS1 = bw1 * (ssx) ^ 0.5 / Estderror1
txtcor.Text = Format(Str(cor1), "0.000")
mttf = eta * (1 + (1 / bw1))
txtmttf.Text = Format(Str(mttf), "0.000")
hrel = Exp(-((txtmtime.Text) / eta) ^ bw1)
lamdab = (bw1 / eta) * ((txtmtime.Text / eta) ^ (bw1 - 1))
If hrel < 0 Then
txthrel1.Text = 0
Else
If hrel > 1 Then
txthrel1.Text = 1
Else
txthrel1.Text = Format(Str(hrel), "0.000")
End If
End If
For i = 1 To n
z1(i) = (Log(-Log(1 - ((i - 0.5) / (n + 0.25)))))
Next i
For i1 = 1 To n
m1(i1) = z1(i1 + 1) - z1(i1)
Next i1
k1 = Int(n / 2)
k2 = Int((n - 1) / 2)
num2 = 0
For i2 = k1 To n - 1
num2 = num2 + (((Log(sdata(i2 + 1))) - (Log(sdata(i2)))) / (m1(i2)))
Next i2
den = 0
For i3 = 1 To k1
den = den + (((Log(sdata(i3 + 1))) - (Log(sdata(i3)))) / (m1(i3)))
Next i3
num1 = num2 * k1
den1 = den * k2
mann = num1 / den1
If weib.Value = True Then
Label11.Visible = False
Label15.Visible = False
txtbart.Visible = True
Label12.Visible = True
txtbart.Text = Format(Str(mann), "0.000")
End If
End If
Label13.Visible = True
Text1.Visible = True
Label14.Visible = True
Label13.Caption = "Distribution selected: Weibull"
```

```
        Text1.Text = " Mann test value is then compared with the F- value (also known as critical
value) with degrees of freedom" + Str(n - 2) + " and level of significance alhpa (user decides this
value) from the standard table. If the Mann's test value is less than the critical value than the data
is good fit for Weibull distribution"
Label14.Caption = "Failure Rate (Lamda) =" + Format(Str(lamdab), "0.00000")
txtalpha.Visible = True
Command1.Visible = True
End If

'******Normal distribution
If NormOpt.Value = True Then
'**to calculate meant time.
Dim dfnormal, no_risk As Integer
Dim sumdata, diff, diffsq, sumsqdiff, sumpro, sumtime, sumnorvalue, sumsqtime, sumsqnormal,
be1, be0, estmean, eststd As Double
Dim sst1, ssz, sstz, SSE2, se, TS2, Stdev, correlation, sumnormalvalue As Double
n = Val(noOffailures.Text)
Dim meantime, sampmean, sampstd As Double
Dim sumg, pnormvalue, diff_tm, diff_gm, ae1, ae2, ae3, p, zek, yek, phik, qek, vek, capk, wek,
sumwek, estimean, diff_tem, diff_wm, estistd, reliabili, failrate As Double
Dim norvalues(1000), fej(1000) As Double
sumdata = 0
For i = 1 To n
gridexpo.Col = 1
gridexpo.Row = i
sumdata = sumdata + gridexpo.Text
Next i
meantime = sumdata / n
'**to calculate standard deviation of the data
sumsqdiff = 0
For i = 1 To n
gridexpo.Col = 1
gridexpo.Row = i
diff = gridexpo.Text - meantime
diffsq = diff * diff
sumsqdiff = sumsqdiff + diffsq
Next i
Stdev = Sqr(sumsqdiff / (n - 1))
'** to calculate the normal values of the data
For i = 1 To n
gridexpo.Col = 1
gridexpo.Row = i
fej(i) = (i - 0.3) / (n + 0.4)
Next i
For i = 1 To n
gridexpo.Col = 1
gridexpo.Row = i
norvalues(i) = (1 / (Sqr(2 * 3.14))) * Exp(-((0.5) * (fej(i) ^ 2)))
Next i
'** To calculate the slope b1 and intercept b0
sumpro = 0
sumtime = 0
sumnorvalue = 0
sumsqtime = 0
sumsqnormal = 0
For i = 1 To n
gridexpo.Col = 1
gridexpo.Row = i
sumpro = sumpro + (gridexpo.Text * norvalues(i))
```

```
        sumtime = sumtime + gridexpo.Text
sumnorvalue = sumnorvalue + norvalues(i)
sumsqtime = sumsqtime + (gridexpo.Text ^ 2)
sumsqnormal = sumsqnormal + (norvalues(i) ^ 2)
Next i
be1 = ((n * sumpro) - (sumtime * sumnorvalue)) / ((n * sumsqtime) - ((sumtime ^ 2)))
be0 = (sumnorvalue / n) - (be1 * sumtime / n)
'** to calculate estimated mean and standard deviation
estmean = be0 / be1
eststd = 1 / be1
'** degrees of freedom
dfnormal = Val(noOffailures.Text) - 2
'** sum of squares of t
sst1 = sumsqtime - ((sumtime * sumtime) / n)
'** sum of squares of z
ssz = sumsqnormal - ((sumnorvalue * sumnorvalue) / n)
'** sum of squares of tz
sstz = sumpro - (sumtime * sumnorvalue / n)
'** sum of squares of error
SSE2 = ssz - (be1 * sstz)
'** estimated standard error
se = Sqr(SSE2 / n)
'** correlation
correlation = sstz / Sqr(sst1 * ssz)
txtcor.Text = correlation
'** test statistic of t (for slope)
TS2 = be1 / (se / Sqr(sst1))
'** to calculate the sample mean
no_risk = Val(noOfrisks.Text)
sumg = (no_risk - n) * Val(testtime.Text)
sampmean = (sumtime + (sumg)) / no_risk
'** to calculate the sample standard deviation
diff_tm = 0
For i = 1 To n
gridexpo.Col = 1
gridexpo.Row = i
diff_tm = diff_tm + ((gridexpo.Text - sampmean) ^ 2)
Next i
diff_gm = 0
For i = 1 To (no_risk - n)
diff_gm = diff_gm + ((Val(testtime.Text) - sampmean) ^ 2)
Next i
sampstd = Sqr((diff_tm + diff_gm) / (no_risk - 1))
ae1 = 0.4361836
ae2 = -0.1201676
ae3 = 0.937298
p = 0.33267
zek = (Val(testtime.Text) - sampmean) / sampstd
yek = 1 / (1 + ((p * zek)))
phik = (1 / (Sqr(2 * 3.14))) * Exp((-0.5) * (zek ^ 2))
qek = phik * ((ae1 * yek) + (ae2 * (yek ^ 2)) + (ae3 * (yek ^ 3)))
vek = phik / qek
capk = vek * (vek - zek)
wek = sampmean + (sampstd * vek)
'*** to calculate estimated mean
sumwek = 0
For i = 1 To (no_risk - n)
   sumwek = sumwek + wek
Next i
```

```
        estimean = (sumwek + sumtime) / no_risk
diff_tem = 0
For i = 1 To n
gridexpo.Col = 1
gridexpo.Row = i
diff_tem = diff_tem + ((gridexpo.Text - estimean) ^ 2)
Next i
diff_wm = 0
For i = 1 To (no_risk - n)
diff_wm = diff_wm + ((wek - estimean) ^ 2)
Next i
estistd = Sqr((diff_wm + diff_tem) / (n + ((no_risk - n) * capk)))
'** to calculate reliability
Dim mtime1 As Double, x11, sumh11, h11, hrel11, z11 As Double
x11 = -10
mtime1 = Val(txtmtime.Text)
z11 = (mtime1 - estimean) / estistd
h11 = 0.005
sumh11 = 0
Do While x11 < z11
sumh11 = sumh11 + (((1 / Sqr(2 * 3.14)) * Exp((-(x11 ^ 2) / 2))) * h11)
x11 = x11 + h11
Loop
hrel11 = 1 - sumh11
If hrel11 < 0 Then
txthrel1.Text = 0
Else
txthrel1.Text = hrel11
End If
failrate = ((1 / Sqr(2 * 3.14 * estistd ^ 2)) * Exp((-1 / 2) * ((Val(txtmtime.Text) - estimean) / estistd)
^ 2)) / hrel11
txtmttf.Text = 1 / failrate
Dim min1 As Double, failure(10000), z As Double
Dim index1 As Double, hfailure(10000) As Double
Dim newd1(100) As Double, newd2(100) As Double, l1 As Integer, m2 As Integer

For i = 1 To n
gridexpo.Col = 1
gridexpo.Row = i
failure(i) = gridexpo.Text
Next i
min1 = 100000                     ' sorts the failure data
For m2 = 1 To n
For l1 = 1 To n
If failure(l1) < min1 Then
min1 = failure(l1)
index1 = l1
End If
Next l1
hfailure(m2) = min1
failure(index1) = 100000
min1 = 100000
Next m2
Dim kssum(100) As Double, x As Double, h As Double
Dim D1(100) As Double, D2(100) As Double
For i = 1 To n
x = -4
z = (hfailure(i) - meantime) / (Stdev)
h = 0.005
```

```
        kssum(i) = 0
Do While x < z
kssum(i) = kssum(i) + (((1 / Sqr(2 * 3.14)) * Exp((-(x ^ 2) / 2))) * h)
x = x + h
Loop
D1(i) = ((kssum(i)) - ((i - 1) / n))
D2(i) = ((i / n) - (kssum(i)))
Next i
min1 = 100000                    ' sorts the failure data
For m2 = 1 To n
For l1 = 1 To n
If D1(l1) < min1 Then
min1 = D1(l1)
index1 = l1
End If
Next l1
newd1(m2) = min1
D1(index1) = 100000
min1 = 100000
Next m2
min1 = 100000                    ' sorts the failure data
For m2 = 1 To n
For l1 = 1 To n
If D2(l1) < min1 Then
min1 = D2(l1)
index1 = l1
End If
Next l1
newd2(m2) = min1
D2(index1) = 100000
min1 = 100000
Next m2
Dim k_s As Double
If newd1(n) < newd2(n) Then
k_s = newd2(n)
Else
k_s = newd1(n)
End If
Label12.Visible = False
Label11.Visible = False
txtbart.Visible = True
Label13.Visible = True
Text1.Visible = True
Label14.Visible = True
Label15.Visible = True
Label13.Caption = " Distribution selected: Normal"
txtbart.Text = Str(Format(k_s, "#.0000"))
Text1.Text = " Kolgomorov-Smirnov Test Stat test value is then compared with the Kolgomorov -
Smirnov test value from the standard table(also known as critical value); with level of significance
alpha (0.20, 0.15, 0.10  or 0.05). If the calculated Kolgomorov-Smirnov Test Stat value is less
than the critical value then the data is good fit for Normal distribution"
Label14.Caption = "Failure Rate (Lamda) =" + Format(Str(failrate), "0.00000")
End If
End Sub
```

<u>Software Reliability Form</u>

```
'ADDITIONAL TIME CALCULATION
'****************************
Private Sub cmdadditional_Click()
```

```
        Dim DFI As Double, deltat As Double
'ERROR CHECK
If txtDFI.Text = "" Then
MsgBox " Enter Desired failure Intensity ", vbExclamation
Exit Sub
End If
DFI = Val(txtDFI.Text)
If IsNumeric(txtDFI.Text) Then
DFI = Val(txtDFI.Text)
If DFI < 0 Then
MsgBox "invalid number"
Exit Sub
End If
Else
MsgBox "Invalid Number"
Exit Sub
End If
If Val(txtDFI.Text) > Val(txtFI.Text) Then
MsgBox "Desired Failure Intensity should be less than current failure intensity", vbCritical
Exit Sub
End If
'ERROR CHECK OVER

'=================
 'EXPONENTIAL MODEL
 '=================
 If optexponential.Value = True Then
 deltat = ((1 / b1new) * Log((TNF * b1new) / ((DFI) * (1 - Exp(-b1new * TET))))) - TET
 txtadditional.Text = deltat
 End If
'============
' GAMMA MODEL
'============
If optgamma.Value = True Then
deltat = ((Log((DFI) * (((Exp(b1new * TET)) - ((b1new * TET) + 1)) / (TNF * b1new * b1new * TET
* (Exp(b1new * TET)))))) * (-1 / b1new)) - TET
txtadditional.Text = deltat
End If

'============
' POWER MODEL
'============
If optpower.Value = True Then
deltat = ((((DFI) * (TET ^ b1new)) / (TNF * b1new)) ^ (1 / (b1new - 1))) - TET
txtadditional.Text = deltat
End If
'================
' GEOMETRIC MODEL
'================
If optlp.Value = True Then
deltat = (TNF / (DFI * Log(1 + (b1new * TET)))) - (1 / b1new) - TET
txtadditional.Text = deltat
End If
'====================
' INVERSE LINEAR MODEL
'====================
If optinverse.Value = True Then
deltat = ((((TNF) / ((((B1old + TET) ^ (1 / 2)) - ((B1old) ^ (1 / 2))) * 2 * DFI)) ^ 2) - (b1new)) - TET
txtadditional.Text = deltat
```

```
        End If
'====================
'WEIBULL MODEL
'====================
If optweibull.Value = True Then
MsgBox "Weibull has binomial data distribution and hence has estimated all the failures",
vbInformation
End If
End Sub

Private Sub cmdcls_Click()
Dim k As Integer
For k = 1 To 200
grddata.Col = 0
grddata.Row = k
grddata.Text = ""
Next k
For k = 1 To 200
grddata.Col = 1
grddata.Row = k
grddata.Text = ""
Next k
txtTET.Text = ""
txtTNF.Text = ""
txtfile.Text = ""
txtFI.Text = ""
txtDFI.Text = ""
txtadditional.Text = ""
txtsum = ""
optexponential.Value = False
optgamma.Value = False
optinverse.Value = False
optweibull.Value = False
optlp.Value = False
optpower.Value = False
MSChart2.Visible = False
Label3.Visible = False
End Sub

'FAILURE INTENSITY CALCULATION
'******************************
Private Sub cmdFI_Click()
Dim final As Double
If txtfile.Text = "" Then
MsgBox "You Must Select File !", vbExclamation
Exit Sub
End If
grddata.Col = 1
grddata.Row = 1
If grddata.Text = "" Then
MsgBox " Open the selected File", vbExclamation
Exit Sub
End If
TET = Val(txtTET.Text)
If txtTET.Text = "" Then
MsgBox " You Must Enter End Of Test Time ", vbCritical
Exit Sub
End If
If IsNumeric(txtTET.Text) Then
```

```
        TET = Val(txtTET.Text)
If TET <= 0 Then
MsgBox "invalid number"
Exit Sub
End If
Else
MsgBox "Invalid Number"
Exit Sub
End If
If optexponential.Value = False And optgamma.Value = False And optinverse.Value = False And
optlp.Value = False And optpower.Value = False And optweibull.Value = False Then
MsgBox "You Must Select Model!", vbExclamation
Exit Sub
End If
srtime = InputBox("Enter length of time interval for which relaibility has to be calculated:", "RET-
Input Box")
If srtime = "" Then
MsgBox "Please enter the time interval", vbExclamation
Exit Sub
End If
If Not IsNumeric(srtime) Then
MsgBox "Invalid Time !", vbExclamation
Exit Sub
End If
TNF = Val(txtTNF.Text)
TET = Val(txtTET.Text)
If Val(txtTET) < failure(TNF) Then
MsgBox "Invalid Number - Test end  time should be equal to or greater than last failure time",
vbExclamation
Exit Sub
End If


'================
'EXPONENTIAL MODEL
'================
If optexponential.Value = True Then
B1old = 0.0001
10  f = ((TNF / B1old) - ((TNF * TET) / ((Exp(B1old * TET)) - 1)) - (sum))
f1 = (-TNF / B1old ^ 2) - (TNF * (TET ^ 2) * (-1) * ((((Exp(B1old * TET)) - 1)) ^ -2) * (Exp(B1old *
TET)))
b1new = B1old - (f / f1)
If Abs(b1new - B1old) < 0.00000001 Then
b0 = TNF / (1 - (Exp(-b1new * TET)))
For l = 1 To TNF
grddata.Col = 1
grddata.Row = l
fintensity = b1new * b0 * (Exp(-b1new * grddata.Text))
FD(l) = grddata.Text
grdresult1.Col = 2
grdresult1.Row = l
grdresult1.Text = Format(fintensity, "0.######")
FI(l) = fintensity
Next l
final = b1new * b0 * (Exp(-b1new * TET))
txtFI.Text = Format(final, "0.######")
Else
B1old = b1new
GoTo 10
```

```
        End If
End If
'============
' GAMMA MODEL
'============
If optgamma.Value = True Then
B1old = 0.0001
20  f = ((2 * TNF / B1old) - ((TNF * TET * B1old * TET) / ((Exp(B1old * TET)) - 1 - (B1old * TET)))
- (sum))
    f1 = (-2 * TNF / B1old ^ 2) - ((TNF * (TET ^ 2)) * (((-B1old) * ((((Exp(B1old * TET)) - 1 - (B1old *
TET)) ^ -2) * (((Exp(B1old * TET)) * TET) + TET))) + (((Exp(B1old * TET)) - 1 - (B1old * TET)) ^ -
1)))
b1new = B1old - (f / f1)
If Abs(b1new - B1old) < 0.00000001 Then
b0 = TNF / (1 - ((1 + b1new * TET) * (Exp(-b1new * TET))))
For l = 1 To TNF
grddata.Col = 1
grddata.Row = l
fintensity = (b1new ^ 2) * b0 * (Exp(-b1new * grddata.Text)) * grddata.Text
FD(l) = grddata.Text
grdresult1.Col = 2
grdresult1.Row = l
grdresult1.Text = Format(fintensity, "0.#######")
FI(l) = fintensity
Next l
final = (b1new ^ 2) * b0 * (Exp(-b1new * TET)) * TET
txtFI.Text = Format(final, "0.######")
Else
B1old = b1new
GoTo 20
End If
End If
'===============
'GEOMETRIC MODEL
'===============
If optlp.Value = True Then
Dim term1 As Double, term2 As Double, t1 As Double, term3 As Double, term4 As Double
Dim sum1 As Double, k As Integer
B1old = 0.0001
30 For k = 1 To TNF
sum1 = 0
t1 = 0
grddata.Col = 1
grddata.Row = k
sum1 = sum1 + (1 / (1 + (B1old * grddata.Text)))
t1 = t1 + ((grddata.Text) / ((1 + (B1old * grddata.Text)) ^ 2))
Next k
term1 = sum1 / B1old
term2 = -(t1 / B1old) - ((sum1) * (1 / (B1old ^ 2)))
f = term1 - ((TET * TNF) / ((1 + (TET * B1old)) * (Log(1 + (TET * B1old)))))
term3 = (TET * TET * TNF) / ((((1 + TET * B1old)) ^ 2) * ((Log(1 + (TET * B1old))) ^ 2))
term4 = (TET * TET * TNF) / ((((1 + TET * B1old)) ^ 2) * ((Log(1 + (TET * B1old))) ^ 1))
f1 = term2 + term3 + term4
b1new = B1old - (f / f1)
If Abs(b1new - B1old) < 0.001 Then
b0 = (TNF / (Log(1 + (b1new * TET))))
For l = 1 To TNF
grddata.Col = 1
grddata.Row = l
```

```
        fintensity = (b0 * b1new) / (1 + (b1new * grddata.Text))
FD(l) = grddata.Text
grdresult1.Col = 2
grdresult1.Row = l
grdresult1.Text = Format(fintensity, "0.######")
FI(l) = fintensity
Next l
final = (b0 * b1new) / (1 + (b1new * TET))
txtFI.Text = Format(final, "0.######")
Else
B1old = b1new
GoTo 30
End If
End If
'===========
'POWER MODEL
'===========
If optpower.Value = True Then
B1old = 0.0001
40  f = (TNF / B1old) + (sumln) - (TNF * Log(TET))
f1 = (-TNF / B1old ^ 2)
b1new = B1old - (f / f1)
If Abs(b1new - B1old) < 0.00000001 Then
b0 = TNF / (TET ^ b1new)
For l = 1 To TNF
grddata.Col = 1
grddata.Row = l
fintensity = b1new * b0 * (grddata.Text ^ (b1new - 1))
FD(l) = grddata.Text
grdresult1.Col = 2
grdresult1.Row = l
grdresult1.Text = Format(fintensity, "0.######")
FI(l) = fintensity
Next l
final = b1new * b0 * (TET ^ (b1new - 1))
txtFI.Text = Format(final, "0.######")
Else
B1old = b1new
GoTo 40
End If
End If
'=============
'WEIBULL MODEL
'=============
If optweibull.Value = True Then
Dim D As Double
D = TET * TET
B1old = 1
50  f = (-TNF) + (sumsq - (D) * ((-TNF) + (TNF / (1 - (Exp(-D * B1old)))))) * B1old
f1 = (((1 / (1 - (Exp(-B1old * D)))) - (((B1old * D * (Exp(-B1old * D)))) / ((1 - (Exp(-B1old * D))) ^
2))) * (-TNF * D)) + (TNF * D) + sumsq
b1new = B1old - (f / f1)
If Abs(b1new - B1old) < 0.00001 Then
b0 = TNF / (1 - (Exp(-b1new * D)))
For l = 1 To TNF
grddata.Col = 1
grddata.Row = l
fintensity = 2 * b0 * b1new * grddata.Text * (Exp(-b1new * grddata.Text * grddata.Text))
FD(l) = grddata.Text
```

```
        grdresult1.Col = 2
grdresult1.Row = l
grdresult1.Text = Format(fintensity, "0.######")
FI(l) = Str(fintensity)
Next l
final = 2 * b0 * b1new * TET * (Exp(-b1new * D))
txtFI.Text = Format(final, "0.##############")
Else
B1old = b1new
GoTo 50
End If
End If
'===================
'INVERSE LINEAR MODEL
'===================
If optinverse.Value = True Then
Dim suminv As Double, inum As Double, iden As Double
Dim suminv1 As Double, j As Double
B1old = 1
60  suminv = 0
For i = 1 To TNF
grddata.Col = 1
grddata.Row = i
suminv = suminv + (((-2) / 3) * ((1) / (B1old + grddata.Text)))
Next i
inum = ((B1old + TET) ^ ((-1) / 2)) - ((B1old) ^ ((-1) / 2))
iden = (((B1old + TET) ^ (1 / 2)) - ((B1old) ^ (1 / 2)))
f = suminv - ((1 / 2) * (TNF) * (inum / iden))
suminv1 = 0
For j = 1 To TNF
grddata.Col = 1
grddata.Row = j
suminv1 = suminv1 + ((2 / 3) * ((1) / ((B1old + grddata.Text) ^ (2))))
Next j
f1 = suminv1 + ((-inum) * (inum / 2) * ((iden) ^ (-2))) + (((-1) / 2) * ((iden) ^ (-1)) * (((B1old + TET) ^
(-3 / 2)) - (B1old ^ (-3 / 2))))
b1new = B1old - (f / f1)
If Abs(b1new - B1old) < 1 Then
b0 = TNF / iden
For l = 1 To TNF
grddata.Col = 1
grddata.Row = l
fintensity = b0 * (1 / (2 * (b1new + grddata.Text) ^ (1 / 2)))
FD(l) = grddata.Text
grdresult1.Col = 2
grdresult1.Row = l
grdresult1.Text = Format(fintensity, "0.######")
FI(l) = fintensity
Next l
final = b0 * (1 / (2 * (b1new + TET) ^ (1 / 2)))
txtFI.Text = Format(final, "0.######")
Else
B1old = b1new
GoTo 60
End If
End If
'Graph Plotting
For p = 1 To TNF
grdresult1.Col = 2
```

```
        grdresult1.Row = p
fidata(p) = grdresult1.Text
Next p
min = 1
For m = 1 To TNF
For l = 1 To TNF
If fidata(l) < min Then
min = fidata(l)
index = l
End If
Next l
sfidata(m) = min
fidata(index) = 1
min = 1
Next m
For i = 1 To TNF
values(1, i) = FD(i)
values(2, i) = FI(i)
Next i
MSChart2.Plot.UniformAxis = False
With MSChart2.Plot.Axis(VtChAxisIdX)
.AxisScale.Type = VtChScaleTypeLinear
.AxisGrid.MinorPen.Style = VtPenStyleNull
.CategoryScale.Auto = False
.ValueScale.Maximum = Val(failure(TNF))
.ValueScale.Minimum = Val(failure(1))
.AxisTitle = "Failure Data"
End With

With MSChart2.Plot.Axis(VtChAxisIdY)
.AxisScale.Type = VtChScaleTypeLinear
.AxisGrid.MinorPen.Style = VtPenStyleNull
.CategoryScale.Auto = False
.ValueScale.Maximum = Val(sfidata(TNF))
.ValueScale.Minimum = Val(sfidata(1))
.AxisTitle = "Failure Intensity"
End With
MSChart2.chartType = VtChChartType2dXY
MSChart2.ColumnCount = 2
MSChart2.RowCount = TNF
'MSChart.ShowLegend = True
'MSChart.ColumnLabel = " Failure Intensity Curve "
For introw = 1 To TNF
MSChart2.Row = introw
MSChart2.Column = 1
MSChart2.data = values(1, introw)
Next introw
For introw = 1 To TNF
MSChart2.Column = 2
MSChart2.Row = introw
MSChart2.data = values(2, introw)
Next introw
MSChart2.Visible = True
Label3.Visible = True
End Sub

'FILE CONTROLS
'*************
Private Sub cmdopen_Click()
```

```
        If txtfile.Text = "" Then
MsgBox "You must Select a Data File!"
Exit Sub
End If
pathname = dirdirectory.path
If Right(dirdirectory.path, 1) <> "\" Then
path = dirdirectory + "\"
Inp = path + txtfile.Text
Else
Inp = pathname + txtfile.Text
End If
Open Inp For Input As #1
i = 0
Do While Not EOF(1)
i = i + 1
Input #1, failure(i)
grddata.Col = 0
grddata.Row = i
grddata.Text = i
grddata.Col = 1
grddata.Row = i
grddata.Text = Val(failure(i))
Loop
txtTNF.Text = i
Close #1
sum = 0
sumln = 0
sumsq = 0
For j = 1 To Val(txtTNF.Text)
grddata.Col = 1
grddata.Row = j
sum = sum + grddata.Text
sumln = sumln + Log(grddata.Text)
sumsq = sumsq + ((grddata.Text) * (grddata.Text))
Next j
txtsum.Text = sum
End Sub

'RESULT TABULATION
'*****************
Private Sub cmdresults_Click()
Dim b As Integer, c As Integer
Open Inp For Input As #4
i = 0
Do While Not EOF(4)
i = i + 1
Input #4, failure(i)
grdresult1.Col = 0
grdresult1.Row = i
grdresult1.Text = i
grdresult1.Col = 1
grdresult1.Row = i
grdresult1.Text = Val(failure(i))
Loop
Close #4
For c = 1 To 5
frmresult.grdresult.ColWidth(c) = 1200
Next c
frmresult.grdresult.Row = 0
```

```
        frmresult.grdresult.Col = 0
frmresult.grdresult.Text = "Failure Nos."
frmresult.grdresult.Col = 1
frmresult.grdresult.Text = "Failure Data"
frmresult.grdresult.Col = 2
frmresult.grdresult.Text = "Failure Intensity"
frmresult.grdresult.Col = 3
frmresult.grdresult.Text = "Reliability"
frmresult.grdresult.Col = 4
frmresult.grdresult.Text = "CDF"
frmresult.grdresult.Col = 5
frmresult.grdresult.Text = "KS Distance"
frmresult.grdresult.Col = 6
frmresult.grdresult.Text = "Empirical"
Open Inp For Input As #3
i = 0
Do While Not EOF(3)
i = i + 1
Input #3, failure(i)
frmresult.grdresult.Col = 0
frmresult.grdresult.Row = i
frmresult.grdresult.Text = i
frmresult.grdresult.Col = 1
frmresult.grdresult.Row = i
frmresult.grdresult.Text = Val(failure(i))
Loop
Close #3
SSE = 0
'================
'EXPONENTIAL MODEL
'================
If optexponential.Value = True Then
For l = 1 To TNF
grddata.Col = 1
grddata.Row = l
fintensity = b1new * b0 * (Exp(-b1new * grddata.Text))
SSE = SSE + (((l) - ((b0) * (1 - Exp(-b1new * grddata.Text)))) ^ 2)
srel = (Exp(-(srtime) * (fintensity)))
CDF1 = ((1 - Exp(-b1new * grddata.Text)))
eval1 = l / TNF
ks1 = eval1 - CDF1
FD(l) = grddata.Text
frmresult.grdresult.Col = 2
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(fintensity, "0.######")
FI(l) = fintensity
frmresult.grdresult.Col = 3
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(srel, "0.#####")
r(l) = srel
frmresult.grdresult.Col = 4
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(CDF1, "0.#####")
CDF(l) = CDF1
frmresult.grdresult.Col = 5
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(ks1, "0.00000")
ks(l) = ks1
frmresult.grdresult.Col = 6
```

```
        frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(eval1, "0.00000")
eval(I) = eval1
Next I
End If
'============
' GAMMA MODEL
'============
If optgamma.Value = True Then
For I = 1 To TNF
grddata.Col = 1
grddata.Row = I
fintensity = (b1new ^ 2) * b0 * (Exp(-b1new * grddata.Text)) * grddata.Text
srel = (Exp(-(srtime) * (fintensity)))
SSE = SSE + (((I) - ((b0) * ((1) - ((1 + b1new * grddata.Text) * (Exp(-b1new * grddata.Text)))))) ^
2)
CDF1 = ((1) - ((1 + b1new * grddata.Text) * (Exp(-b1new * grddata.Text))))
eval1 = I / TNF
ks1 = (I / TNF) - CDF1
FD(I) = grddata.Text
frmresult.grdresult.Col = 2
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(fintensity, "0.#######")
FI(I) = fintensity
frmresult.grdresult.Col = 3
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(srel, "0.00000")
r(I) = srel
frmresult.grdresult.Col = 4
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(CDF1, "0.#####")
CDF(I) = CDF1
frmresult.grdresult.Col = 5
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(ks1, "0.00000")
ks(I) = ks1
frmresult.grdresult.Col = 6
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(eval1, "0.00000")
eval(I) = eval1
Next I
End If
'===============
'GEOMETRIC MODEL
'===============
If optlp.Value = True Then
For I = 1 To TNF
grddata.Col = 1
grddata.Row = I
fintensity = (b0 * b1new) / (1 + (b1new * grddata.Text))
srel = (Exp(-(srtime) * (fintensity)))
SSE = SSE + (((I) - (b0 * (Log(1 + b1new * grddata.Text)))) ^ 2)
CDF1 = (Log(1 + b1new * grddata.Text) / Log(1 + b1new * failure(TNF)))
eval1 = I / TNF
ks1 = (I / TNF) - CDF1
FD(I) = grddata.Text
frmresult.grdresult.Col = 2
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(fintensity, "0.######")
```

```
        FI(l) = fintensity
frmresult.grdresult.Col = 3
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(srel, "0.00000")
r(l) = srel
frmresult.grdresult.Col = 4
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(CDF1, "0.#####")
CDF(l) = CDF1
frmresult.grdresult.Col = 5
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(ks1, "0.00000")
ks(l) = ks1
frmresult.grdresult.Col = 6
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(eval1, "0.00000")
eval(l) = eval1
Next l
End If
'===========
'POWER MODEL
'===========
If optpower.Value = True Then
For l = 1 To TNF
grddata.Col = 1
grddata.Row = l
fintensity = b1new * b0 * (grddata.Text ^ (b1new - 1))
srel = (Exp(-(srtime) * (fintensity)))
SSE = SSE + (((l) - (b0) * ((grddata.Text) ^ (b1new))) ^ 2)
CDF1 = ((grddata.Text) ^ (b1new)) / ((failure(TNF)) ^ (b1new))
eval1 = l / TNF
ks1 = (l / TNF) - CDF1
FD(l) = grddata.Text
frmresult.grdresult.Col = 2
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(fintensity, "0.######")
FI(l) = fintensity
frmresult.grdresult.Col = 3
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(srel, "0.00000")
r(l) = srel
frmresult.grdresult.Col = 4
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(CDF1, "0.#####")
CDF(l) = CDF1
frmresult.grdresult.Col = 5
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(ks1, "0.00000")
ks(l) = ks1
frmresult.grdresult.Col = 6
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(eval1, "0.00000")
eval(l) = eval1
Next l
End If
'============
'WEIBULL MODEL
'============
If optweibull.Value = True Then
```

```
        For I = 1 To TNF
grddata.Col = 1
grddata.Row = I
fintensity = 2 * b0 * b1new * grddata.Text * (Exp(-b1new * grddata.Text * grddata.Text))
srel = (Exp(-(srtime) * (fintensity)))
SSE = SSE + (((I) - ((b0) * ((1 - Exp(-b1new * ((grddata.Text) ^ 2)))))) ^ 2)
CDF1 = ((1 - Exp(-b1new * ((grddata.Text) ^ 2))))
eval1 = I / TNF
ks1 = (I / TNF) - CDF1
FD(I) = grddata.Text
frmresult.grdresult.Col = 2
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(fintensity, "0.######")
FI(I) = Str(fintensity)
frmresult.grdresult.Col = 3
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(srel, "0.00000")
r(I) = srel
frmresult.grdresult.Col = 4
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(CDF1, "0.#####")
CDF(I) = CDF1
frmresult.grdresult.Col = 5
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(ks1, "0.00000")
ks(I) = ks1
frmresult.grdresult.Col = 6
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(eval1, "0.00000")
eval(I) = eval1
Next I
End If
'===================
'INVERSE LINEAR MODEL
'===================
If optinverse.Value = True Then
For I = 1 To TNF
grddata.Col = 1
grddata.Row = I
fintensity = b0 * (1 / (2 * (b1new + grddata.Text) ^ (1 / 2)))
srel = (Exp(-(srtime) * (fintensity)))
SSE = SSE + (((I) - ((b0) * (((b1new + grddata.Text) ^ (1 / 2)) - ((b1new) ^ (1 / 2))))) ^ 2)
CDF1 = (((b1new + grddata.Text) ^ (1 / 2)) - ((b1new) ^ (1 / 2))) / (((b1new + failure(TNF)) ^ (1 /
2)) - ((b1new) ^ (1 / 2)))
eval1 = I / TNF
ks1 = (I / TNF) - CDF1
FD(I) = grddata.Text
frmresult.grdresult.Col = 2
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(fintensity, "0.######")
FI(I) = fintensity
frmresult.grdresult.Col = 3
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(srel, "0.00000")
r(I) = srel
frmresult.grdresult.Col = 4
frmresult.grdresult.Row = I
frmresult.grdresult.Text = Format(CDF1, "0.#####")
CDF(I) = CDF1
```

```
        frmresult.grdresult.Col = 5
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(ks1, "0.00000")
ks(l) = ks1
frmresult.grdresult.Col = 6
frmresult.grdresult.Row = l
frmresult.grdresult.Text = Format(eval1, "0.00000")
eval(l) = eval1
Next l
End If
frmresult.Show
End Sub

Private Sub dirdirectory_Change()
'Here we change the files according to the directory
filfiles.path = dirdirectory.path
End Sub

Private Sub drvdrive_Change()
'The next statement checks the error in the path
On Error GoTo driveerror
'change the path of the new drive
dirdirectory.path = drvdrive.Drive
Exit Sub
driveerror:
' Here we restore the original drive
MsgBox "Device is not ready !", vbCritical, "Error"
drvdrive.Drive = dirdirectory.path
End Sub

Private Sub filfiles_Click()
txtfile.Text = filfiles.FileName
End Sub

Private Sub Form_Load()
frmret.mnupsoftware.Enabled = True
grddata.ColWidth(1) = 1200
grddata.Row = 0
grddata.Col = 1
grddata.Text = "Failure Data"
MSChart2.Visible = False
Label3.Visible = False
End Sub
```

Reliability Calculator

```
Option Explicit
Dim frate As Single
Dim pi As Single
Dim x As Double, z As Double
Dim h As Single
Dim sum As Double
Dim nmean As Double, nstddev As Double
Dim lnshape As Double, lnloc As Double
Dim wscale As Single
Dim wshape As Single


Private Sub cmde_Click()
```

```
        mtime = Val(txtmtime.Text)
frate = Val(txtfailurerate.Text)
If txtfailurerate.Text = "" Then
MsgBox "Enter the parameters for Exponential distribution", vbExclamation
Exit Sub
End If
If txtmtime.Text = "" Then
MsgBox " Enter the Mission Time"
Exit Sub
End If
hrel = (Exp(-frate * mtime))
lblrel.Caption = "Reliabilty for exponetial model for mission time" + Str(mtime) + " is" + Str(hrel)
End Sub

Private Sub cmdn_click()
mtime = Val(txtmtime.Text)
nmean = Val(txtmean.Text)
nstddev = Val(txtstandarddev.Text)
If txtmtime.Text = "" Then
MsgBox " Enter the Mission Time"
Exit Sub
End If
If txtmean.Text = "" Or txtstandarddev.Text = "" Then
MsgBox "Enter the parameters for Normal distribution", vbExclamation
Exit Sub
End If
pi = 3.14
x = -20
z = (mtime - nmean) / nstddev
h = 0.005
sum = 0
Do While x < z
sum = sum + (((1 / Sqr(2 * pi)) * Exp((-(x ^ 2) / 2))) * h)
x = x + h
Loop
hrel = 1 - sum
lblrel.Caption = "Reliabilty for normal model for mission time" + Str(mtime) + " is" + Str(hrel)
End Sub
Private Sub cmdln_click()
mtime = Val(txtmtime.Text)
lnshape = Val(txtshapeparameter.Text)
lnloc = Val(txtlocation.Text)
If txtmtime.Text = "" Then
MsgBox " Enter the Mission Time"
Exit Sub
End If
If txtshapeparameter.Text = "" Or txtlocation.Text = "" Then
MsgBox "Enter the parameters for Log Normal distribution", vbExclamation
Exit Sub
End If
pi = 3.14
x = -20
z = (1 / lnshape) * Log(mtime / lnloc)
h = 0.005
sum = 0
Do While x < z
sum = sum + ((1 / (Sqr(2 * pi))) * (Exp((-x ^ 2 / 2)))) * h
x = x + h
Loop
```

```
        hrel = 1 - sum
lblrel.Caption = "Reliabilty for LogNormal model for mission time" + Str(mtime) + " is" + Str(hrel)
End Sub
Private Sub cmdw_click()
mtime = Val(txtmtime.Text)
wshape = Val(txtwshape.Text)
wscale = Val(txtwscale.Text)
If txtmtime.Text = "" Then
MsgBox " Enter the Mission Time"
Exit Sub
End If
If txtwshape.Text = "" Or txtwscale.Text = "" Then
MsgBox "Enter the parameters for Weibull distribution", vbExclamation
Exit Sub
End If
hrel = Exp(-((mtime / wscale) ^ wshape))
lblrel.Caption = "Reliabilty for Weibull model for mission time" + Str(mtime) + " is" + Str(hrel)
End Sub
Private Sub cmdclear_Click()
lblrel.Caption = ""
txtfailurerate.Text = ""
txtshapeparameter.Text = ""
txtlocation.Text = ""
txtmean.Text = ""
txtstandarddev.Text = ""
txtwshape.Text = ""
txtwscale.Text = ""
txtmtime.Text = ""
End Sub
```

F-Value Calculator

```
Option Explicit
Private Sub Command1_Click()
Dim p1 As Double, n1 As Double, n2 As Double
lblans.Caption = Format(Str(afishf(p1, n1, n2)), "0.000")
End Sub

Public Function fishf(f As Double, n1 As Double, n2 As Double)
Dim p As Double
Dim v As Double, dv As Double
Dim x As Double, Y As Double, w As Double
Dim th As Double, a As Double, sth As Double, cth As Double, b As Double, b1 As Double
Dim c As Double, pi As Double, pid2 As Double, k As Double
pi = 3.14159265358979
pid2 = pi / 2
b = n1 / 2
b1 = n2 / 2
x = n2 / (n1 * f + n2)
If n1 Mod 2 = 0 Then
fishf = statcom(1 - x, n2, n1 + n2 - 4, n2 - 2) * (x ^ (n2 / 2))
Exit Function
End If
If n2 Mod 2 = 0 Then
fishf = 1 - statcom(x, n1, n1 + n2 - 4, n1 - 2) * ((1 - x) ^ (n1 / 2))
Exit Function
End If
th = Atn(Sqr(n1 * f / n2))
a = th / pid2
```

```
        sth = Sin(th)
cth = Cos(th)
Y = cth * cth
w = sth * sth
If n2 > 1 Then
a = a + (sth * cth * (statcom(Y, 2, n2 - 3, -1)) / pid2)
End If
If n1 = 1 Then
fishf = 1 - a
Exit Function
End If
c = 4 * statcom(w, n2 + 1, n1 + n2 - 4, n2 - 2) * sth * ((cth ^ n2) / pi)
If n2 = 1 Then
fishf = 1 - a + c / 2
Exit Function
End If
k = 2
Do While (k <= (n2 - 1) / 2)
c = c * k / (k - 0.5)
k = k + 1
Loop
fishf = 1 - a + c
End Function
Public Function statcom(q, i, j, b)
Dim zz As Double, z As Double, k As Double
zz = 1
z = zz
k = i
Do While (k <= j)
zz = zz * q * k / (k - b)
z = z + zz
k = k + 2
Loop
statcom = z '( returns z)
End Function

Function afishf(p1 As Double, n1 As Double, n2 As Double)
Dim v As Double, dv As Double, f As Double
p1 = Val(Text1.Text)
n1 = Val(Text2.Text)
n2 = Val(Text3.Text)
v = 0.5
dv = 0.5
f = 0
Do While (dv > 0.000001)
f = 1 / v - 1
dv = dv / 2
If (fishf(f, n1, n2)) > 1 - p1 Then
v = v - dv
Else
v = v + dv
End If
Loop
afishf = f '(returns f)
End Function
```

Chi-square Calculator

```
Option Explicit
```

```
        Private Sub cmdprob_Click()
Dim cal_p As Double, p As Double
Dim n As Integer
Dim x As Double
Dim k As Double, t As Double, a As Double
Dim v As Double, dv As Double
v = 0.5
dv = 0.5
x = 0
If IsNumeric(txtx.Text) Then
GoTo 10
Else
MsgBox "invalid number"
End If
Exit Sub
10 p = Val(txtx.Text)
n = Val(txtdf.Text)
Do While (dv > 0.000001)
x = 1 / v - 1
dv = dv / 2
cal_p = Exp(-0.5 * x)
If n Mod 2 > 0 Then
cal_p = cal_p * Sqr(2 * x / 3.14)
End If
k = n
Do While (k > 2 Or k = 2)
cal_p = cal_p * (x / k)
k = k - 2
Loop
t = cal_p
a = n
Do While (t > 0.000001 * p)
a = a + 2
t = t * (x / a)
cal_p = cal_p + t
Loop
If (1 - cal_p) > 1 - p Then
v = v - dv
Else
v = v + dv
End If
Loop
lblp.Caption = Format(Str(x), "0.000")
End Sub
```

<u>Tieset Source Code (C++)</u>

```
// Manish.cpp : Defines the entry point for the console application.
#include "stdafx.h"
#include "Readfile.h"
#include "Rel.h"
int main(int argc, char* argv[])
{
if (argc == 2) {
Rel r(argv[1]);
r.makeTree();
r.makeMat();
r.printMat();
}
```

```
        return 0;
}

#include "stdafx.h"
#include "StringTokenizer.h"
#include "Readfile.h"
int Readfile::readLine() {
if (ifs->getline(buffer,200)) return 1;
return 0;

void Readfile::parseLine(vector<string>& tokens, char del)
{
string str(buffer);
StringTokenizer st(str, del);
while (st.isMoreTokens()) {
string& str = st.nextToken();
if (str != "") {
tokens.push_back(str);
else delete &str;
}
}
#include "stdafx.h"
#include "Readfile.h"
#include "Rel.h"
#include "StringTokenizer.h"
#define D(t) cout << t << endl;

void Rel::makeTree() {
vector<string> tokens;
vector<string> pred;
string id;
float f;
while (rf->readLine()) {
rf->parseLine(tokens, '\t');
int size = tokens.size();
for (int i=1; i<size; i++) {
//D(tokens[i]);
StringTokenizer st(tokens[i], ',');
while (st.isMoreTokens()) {
string& token = st.nextToken();
if (token != "") {
pred.push_back(token);
}
else delete &token;
}
}
parseNode(tokens[0], id, f);
tree[id] = pred;
table[id] = f;
pred.clear();
tokens.clear();
}
}

void Rel::makeMat() {
Tree::iterator titr;
titr = tree.find("sink");
if (titr == tree.end()) {
D("sink not defined");
```

```
        return;
}

row& r = (*titr).second;
row ro;
for (int i=0;i<r.size();i++) {
traverseRow(ro, r[i]);
ro.clear();
}
}

void Rel::traverseRow(row& r1, string& s) {
if (s == "source") {
InsertRow(r1);
r1.push_back(s);
return;
}
Tree::iterator titr;
titr = tree.find(s);
r1.push_back(s);
row& r = (*titr).second;
for (int i=0;i<r.size();i++) {
traverseRow(r1,r[i]);
r1.pop_back();
}
}

void Rel::InsertRow(row& ro) {
row r(ro);
rel.push_back(r);
}

void Rel::parseNode(string& str, string& id, float& f) {
f = 1.0;
int pos = str.find('(',0);
if (pos == string::npos) {
id = string(str);
return;
}
id = string(str.substr(0,pos));
//D(id);
int end = str.find(')',pos+1);
if (end != string::npos) {
if = (float)atof(str.substr(pos+1,end-1).c_str());
//D(f);
}
}

double Rel::powerSet(set<string> totRow, int pos, int size) {
if(size == 1) {
set<string> tempRow;
set<string>::iterator itr;
relTable::iterator rel_itr;
double reliability = 0.0;
for(int i=pos;i<matSize;i++) {
row& row_i = rel[i];
set_union(row_i.begin(), row_i.end(),
totRow.begin(), totRow.end(),
inserter(tempRow, tempRow.begin()));
```

```
        double prod = 1.0;
itr = tempRow.begin();
for(;itr != tempRow.end();itr++) {
rel_itr = table.find(*itr);
if (rel_itr != table.end()) {
prod *= (*rel_itr).second;
}
}
reliability += prod;
tempRow.erase(tempRow.begin(), tempRow.end());
tempRow.clear();
}
return reliability;
}
--size;
set<string> tempSet;
double rel_t = 0.0;
for(int i=pos;i<matSize;i++) {
row& row_present = rel[i];
set_union(totRow.begin(), totRow.end(),
row_present.begin(), row_present.end(),
inserter(tempSet, tempSet.begin()));
rel_t += powerSet(tempSet, i+1, size);
tempSet.clear();
}
return rel_t;
}

void Rel::printMat() {
string waitChr;
double reliability = 0.0;
double prevVal = 0.0;
ofstream ofs(filePath.c_str());
ofstream ofs1(filePath1.c_str());
matSize = rel.size();
for (int i=0;i<matSize;i++) {
ofs << "Tie set number " << (i+1) << " = ";
row& r = rel[i];
for(int y=0;y<r.size();y++) {
ofs << r[y];
if (y != r.size()-1) ofs <<", ";
}
ofs << endl;
}
for(i=0; i <matSize; i++) {
set<string> tempSet;
if (i%2 == 0) {
reliability += powerSet(tempSet, 0, i+1);
}
else reliability -= powerSet(tempSet, 0, i+1);
cout << "reliability till now = " << reliability << endl;
if(fabs(reliability-prevVal) <= covergence) break;
else prevVal = reliability;
tempSet.clear();
}
cout << "" << reliability << endl;
ofs1 << "" << reliability << endl;
}
```

```
        // stdafx.cpp : source file that includes just the standard includes
//                                  Manish.pch will be the pre-compiled header
//                           stdafx.obj will contain the pre-compiled type information
#include "stdafx.h"
// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
#include "stdafx.h"
#include "StringTokenizer.h"
string& StringTokenizer::nextToken() {
string* token = new string;
while(start != end && *start != delimiter) *token += *start++;
if (start != end) start++; //jump over delimiter
return *token;
}
#ifndef _Read_File
#define _Read_File
class Readfile {
ifstream* ifs;
const char* filename;
char buffer[200];
public:
Readfile() : filename("network.dat") {
ifs = new ifstream(filename);
}
Readfile(const char * file) : filename(file) {
ifs = new ifstream(filename);
}
~Readfile() {
if (ifs) delete ifs;
}
int readLine();
void parseLine(vector<string>& tokens, const char del);
};
#endif
#ifndef REL
#define REL
class Rel {
Readfile* rf;
typedef vector<string> row;
typedef vector<row> Matrix;
Matrix rel;
typedef map<string, row> Tree;
Tree tree;
typedef map<string, float> relTable;
relTable table;
double totRel;
double bakwasRel;
const double covergence;
int matSize;
string filePath;
string filePath1;
public:
Rel(const char* fName):totRel(0.0),bakwasRel(0.0),covergence(0.001),
matSize(0)
{
if (fName) {
rf = new Readfile(fName);
}
string f(fName);
```

```
	int pos = f.find_last_of('\\');
if ( pos != string::npos) {
filePath = string(f.substr(0,pos));
filePath += "\\manish.dat";
filePath1 += f.substr(0,pos)+"\\manish1.dat";
cout << filePath << endl;
}
}
~Rel() { if (rf) delete rf; }
void makeTree();
void makeMat();
void printMat();
private:
double powerSet(set<string>, int, int);
void parseNode(string&, string&, float&);
void InsertRow(row& ro);
void traverseRow(row& r1, string& s);
};
#endif
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#define WIN32_LEAN_AND_MEAN		// Exclude rarely-used stuff from Windows headers
#include <stdio.h>
// TODO: reference additional headers your program requires here
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <map>
#include <cmath>
#include <cstdlib>
#include <set>
#include <algorithm>
using namespace std;
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif //
!defined(AFX_STDAFX_H__C906B5AE_F478_11D4_8EBE_00104B62BC56__INCLUDED_)
#ifndef _String_Tokenizer
#define _String_Tokenizer
// StringTokenizer class for parsing strings
class StringTokenizer {
const char delimiter;
string::const_iterator start;
string::const_iterator end;
public:
StringTokenizer(const string& str, char del='\t') : delimiter(del) {
start = str.begin();
end = str.end();
}
string& nextToken();
bool isMoreTokens() { return (start != end); }
};
#endif
#ifndef _String_Tokenizer
#define _String_Tokenizer
// StringTokenizer class for parsing strings
class StringTokenizer {
        const char delimiter;
string::const_iterator start;
string::const_iterator end;
public:
StringTokenizer(const string& str, char del='\t') : delimiter(del) {
start = str.begin();
end = str.end();
}
string& nextToken();
bool isMoreTokens() { return (start != end); }
};
#endif
```