1999

Development of decision support system for reliability

Mitesh Bhupendra Parekh
West Virginia University

Development of Decision Support System for Reliability.

Mitesh B. Parekh

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 and Management Systems Engineering

Dr. R. Ahluwalia Ph.D. Chair
Dr. Majid Jaraiedi, Ph.D.
Dr. Alan McKendall , Ph.D.

Department of Industrial and Management Systems Engineering

Morgantown, West Virginia
1999

Keywords: Failure Distributions, Statistical Tests, Reliability-Software, Hardware & System, Visual Basic.

ABSTRACT

Development of Decision Support System for Reliability.

Mitesh B. Parekh

Reliability analysis is now an essential part of any product development. In the field of Electrical Engineering, reliability models are developed and analyzed for every component before it comes to market. The complexity of the product can range from a simple transistor to an integrated circuit chip, from a simple communication link to a complex network, and from a small generator to a large powerhouse.

Over the period, several techniques have been developed to systematically model real systems and then analyze them. These techniques are constantly being enhanced to improve the modeling process and to reduce the time required for the 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 model execution and analysis time.

The objectives of this research were to study several hardware-software reliability models, and existing algorithms to compute system reliability and to select an approach that can be efficiently implemented on personal computer. A comprehensive software tool that uses both hardware and software reliability models and computes exact reliability of a system was developed using Visual Basic.

ACKNOWLEDGEMENT

The author wishes to thank his parents for their unyielding support, sacrifices, and encouragement throughout his studies. The author would like to express his gratitude and appreciation to his research advisor, Dr. Rashpal Ahluwalia, for his invaluable guidance and moral support during research and to his committee members, Dr Majid Jareidi and Dr. Alan McKenddall for their helpful comments and suggestions. The author is also thankful to Industrial and Management Systems Engineering department of West Virginia University for their continued financial support which was a great help in his pursuit for completion of M.S. studies. Finally author extends his thanks to all friends and colleagues for their support and encouragement.

Table of Contents

List of Figures

List of Tables

List Of Symbols

B	Bartlett's test statistic
b0,b1,b2..	parameters for software models
Ci	i^{th} Cutset
E[X]	expected value of random variable X
f(t)	probability density function
F(t)	cumulative failure probability function
i, j	indices indicating sequential number of a failure event
m_e	number of failures experienced by time t_e
m_t	number of failures experienced by time t
M	Mann's test Statistic
M(τ)	number of failures experienced by time t (random variable)
$P[E_1]$	probability of event E_1
$P[E_1\|E_2]$	probability of event E_1 conditioned on event E_2
Q_{sys}	Unreliability of the system
R_{sys}	reliability or probability of failure-free operation
s	shape parameter for lognormal distribution
SSE	sum of squared errors
t	set of times (t_1, …, t_e)
t_e	a specific deterministic time denoting the end of failure data
t_i	time of i^{th} failure
t_{med}	shape parameter for lognormal distribution
T	cumulative time
Ti	i^{th} Tieset
TET	end of test time (used in Mathcad for verifying software models)
TNF	total number of failures (software reliability)
z(t)	(realization of) hazard rate
α	confidence level; significance of a model or probability of rejecting a correct model
β	shape parameter for weibull distribution (hardware reliability)
Θ	expected life or mean time to failure (MTTF)
λ	parameter of the exponential distribution used in hardware reliability or the constant hazard rate
λ(τ)	failure intensity function or dμ/dt
μ	mean for the data
μ(τ)	mean value function or expected number of failures experienced by time t or E[M(t)]
∏	product
∑	summation
θ	scale parameter for weibull distribution (hardware reliability)
σ	standard deviation
τ	(realization of) cumulative execution time

∞	a large value
$[t_1, t_2]$	time interval between t_1 and t_2

Chapter 1

Introduction to Reliability.

1.1 Background of Reliability

Since the first electronic digital computer was invented almost fifty years ago, human beings have become dependant on computers in their daily lives. The computer revolution has created the fastest technological advancement that the world has ever seen. Today, computer hardware and software permeates 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 and see that, they rely heavily on computers for their functioning capabilities.

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. For instance, NASA's Space Shuttles flies with approximately 500,000 lines of software code on board and 3.5 million lines of code in ground control and processing. After being scaled down from its original plan, the International Space Station Alpha is still projected to have millions of lines of software to operate innumerable hardware pieces for its navigation, communication, and experimentation. In the telecommunications industry, operations for telephone carriers are supported by hundreds of software systems, with hundreds of millions of lines of source

code. In avionics industry, almost all-new payload instruments contain their own microprocessor system with extensive embedded software. A massive amount of hardware and software also exists in the FAA's Advanced Automation System, the new generation of air traffic control system. Today's offices and homes computers cannot function without an operating system (e.g., Windows) ranging from 1 to 5 million lines of source code and many other shrink-wrapped software packages of similar size provide our daily use of these computers in variety of applications.

The demand for complex hardware/software systems has increased more rapidly than the ability to design, implement, test and maintain them. When requirements for the dependencies on computers increase, 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).

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. It merely means that the probability for failure of this component is low or very low. Conversely, if a component is not dependable, it does not mean that it will not work. Thus, the everyday use of the word Reliability - the quality of being reliable - is purely qualitative.

However, in engineering such qualitative statements are considered vague, because there is no information about the probability that a component will work or fail. Moreover, it is not known for how long a component that is currently working will continue to work without failure. This brings the aspect of availability into picture. Thus, for engineering

applications reliability must be expressed in quantitative terms. This requires developing a reliability model, which will help to predict the effect of time on system reliability.

1.2 Reliability engineering- Present Status

Reliability analysis is now an essential part of any product development. In the field of Electrical Engineering, reliability models are developed and analyzed for every component before it comes to market. The complexity of the product can range from a simple transistor to an integrated circuit chip, from a simple communication link to a complex network, and from a small generator to a large powerhouse.

Over the period, several techniques have been developed to systematically model real systems and then analyze them. These techniques are constantly being enhanced to improve the modeling process and to reduce the time required for the 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 model execution and analysis time.

1.3 Reliability in Communication Networks

A typical communication network consists of transmission lines, repeaters, routers, bridges, and gateways of known performance ratings. A message may have to pass through one or more of these elements. Such a network can be represented by a connected graph consisting of links and nodes. Links represent the transmission lines and nodes represent nodal equipment like repeaters, routers, etc. Failures of some of the nodes or links may affect the performance of the network in terms of delivering the message to its destination. Hence,

the reliability for a message to get through is sometimes improved by adding redundant links and/or nodes to the network

Some of the important parameters of a communication network are cost, throughput, delay, and reliability [1]. In this thesis, techniques are developed to determine only the reliability parameter of the network. The term Reliability in this context means the probability that the network will perform satisfactorily in delivering messages across the network in a time period of intent.

1.4 Statement of the Problem

As mentioned in Section 1.3, reliability is an important parameter of any system. Various techniques are used to compute the system reliability. These techniques can be classified into two categories. The first category consists of techniques that find an approximate reliability solution [2-5], whereas the second category consists of techniques that find the exact reliability solution [6-9]. Some of the commonly used techniques that find the exact network reliability are the conditional probability technique, the cutset technique and the tieset technique. A brief description of these techniques along with their advantages and disadvantages is presented in Chapter 3. A major problem with the above techniques is the complexity of computation. For a large complex system the computations times can be extensive to be executed. Symbolic manipulation is the main bottleneck of the algorithms that determine the exact system reliability. 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.

1.5 Objectives of this Research

The objectives of this research are to:

- Study several hardware-software reliability models, and existing algorithms to compute system reliability and to select an approach that can be efficiently implemented on personal computer.
- Develop a comprehensive software tool that uses both hardware and software reliability models and computes exact reliability of a system.
- Analyze several practical interconnection networks and examples in the literature to validate and demonstrate effectiveness of software tool..

1.6 Thesis Outline

This thesis is divided into six chapters. Chapter 1 provides some fundamental concepts in reliability, statement of the problem, and organization of this thesis. Chapter 2 deals with literature review in the field of reliability. Chapter 3 deals with the concept of hardware, software and system reliability. In addition, Chapter 3 discusses the algorithm, which is used to calculate the reliability of complex systems. Chapter 4 deals the design and development of software. Chapter 5 covers the implementation and validation of the software. The network examples are from published literature. Finally, Chapter 6 provides insight to the conclusion of the research and what future research can be carried out.

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 number of themes in historical development.

- The creation of various models relating reliability to time failures experienced and other variables.
- 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 better data.

2.1 Examples of Major Disasters

Failures are much more significant in both their economic and safety effects. For example, in 1946 the entire fleet of Lockheed Constellation aircraft was grounded following a crash killing four of the five-crew members. The crash was attributed to a faulty design in an electrical conduit that caused the fuselage to burn. In 1979 the left engine of a DC-10 broke away from the aircraft during takeoff, killing 271 people. Poor maintenance procedures and a bad design led to the crash. Engine removal procedures introduced unacceptable stresses on the pylons. The Ford Pinto, introduced in 1971, was recalled by Ford in 1978 for modifications to the fuel tank to reduce fuel leakage and fires resulting from rear-end collisions. Numerous reported deaths, lawsuits, and the negative publicity eventually

contributed to Ford discontinuing production of the Pinto. Firestone's steel-belted radials, introduced in 1972, failed at an abnormal rate as a result of the outer tread breaking apart from the main body of the tire. Because of the excessive number of failures, Firestone was forced to recall 7.5 million tires. On November 8, 1940, the Tacoma Narrows Bridge, five months old, collapsed into Puget Sound from vibrations caused by high winds. Metal fatigue induced by several months of oscillations led to the failure. The Manus River Bridge (Greenwich, Connecticut) collapsed in 1983, killing three people and injuring three. While there is disagreement on the cause of the disaster, blame has been placed on the original design, on corrosion that caused undetected displacement of the pin and hanger suspension assembly, on poor maintenance, and on inadequate inspections. The Hartford (Connecticut) Civic Center Coliseum roof collapsed in 1978 from structural failure due to the weight of the snow and ice accumulated on the roof. A major shortcoming in the roof frame system was the lack of redundancy of members to carry loads when other individual members failed. An inadequate safety margin may also have contributed. The Three Mile Island disaster in 1979, which resulted in a partial meltdown of a nuclear reactor, was a result of both mechanical and human error. When a backup cooling system was down for routine maintenance, air cut off the flow of cooling water to the reactor. Warning lights were hidden by maintenance tags. An emergency relief valve failed to close, causing additional water to be lost from the cooling system. Operators were either reading gauges that were not working properly or taking the wrong actions on the basis of those that were operating. The 1986 explosion of the space shuttle Challenger was a result of the failure of the rubber O-rings that were used to seal the four sections of the booster rockets. The below freezing temperatures before the launch contributed to the failure by making the rubber brittle. [10]

Software failures were highlighted in several major programs. In NASA's Voyager project, the Uranus encounter was in jeopardy because of late software deliveries and reduced

capability in deep space network. Several space shuttle missions have been delayed due to hardware/software interaction problem. In one DoD project, software problems caused the first flight of the AFTI/F-6 jet fighter to be delayed over a year and none of the advanced modes originally planned could be used.

Critical software failures have affected numerous civil and scientific applications. The ozone hole over Antarctica would have received attention sooner from scientific community if a data analysis program had not suppressed the anomalous data because it was "out of range". Software glitches in an automated baggage handling system forced the Denver International Airport to sit empty more than a year after airplanes were to fill its gates and runways [11]

Many software systems and packages are distributed and installed in identical or similar copies, all of which are vulnerable to same software failure. This is why even the most powerful software companies like Microsoft are fearful of the killer bugs, which can wipe out all the profits if a call back is required on the tens of millions of copies of the product [12]. To end this, many software companies see a major share of project development costs identified with the design, implementation, and assurance of reliable software, and they recognize a tremendous need for systematic approaches using software reliability engineering techniques. Clearly, developing the required techniques for reliability engineering is a major challenge to computer engineers, software engineers, industrial engineers, and engineers of various disciplines for now and decade to come [13].

2.2 Development of Hardware Reliability

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 costs.

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 gathering pace of electronic device technology meant that the developers of new military systems were making increasing use of large numbers of new components 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 equipment were seriously affecting its availability for use, and the costs of spares, training and other logistics support were becoming excessive. Against this background the US DoD and the electronics industry jointly set up the Advisory Group on Reliability of Electronic Equipment (AGREE) in 1952. The AGREE report concluded that, to break out of the spiral of increasing development and ownership costs due to low reliability, disciplines must be laid down as integral activities in the development cycle for electronic equipment. The report laid particular stress on the need for new equipment to be tested for several thousand hours in high stress cyclical environments including high and low temperatures, vibration and switching, in order to discover the majority of weak areas in a design at an early enough stage to enable them to be corrected before production commenced. Until that time, environmental tests of tens of duration had been considered adequate to prove the suitability of a design. The report also recommended that formal demonstrations of reliability, in terms of statistical confidence that a specified Mean Time Between Failure (MTBF) had been exceeded, be instituted as a condition for acceptance of equipment by the procuring agency. A large part of the report was devoted to providing detailed test plans for various levels of statistical confidence and environmental conditions.

The AGREE report was accepted by the DoD, and AGREE testing quickly became a standard procedure. Companies that invested in the expensive environmental test equipment necessary soon found that they could attain levels of reliability far higher than by traditional

methods. It was evident that designers, particularly those working at the fringes of advanced technology, could not be expected to produce highly reliable equipment without it being subjected to a test regime that would show up weaknesses. Complex systems and the components used in them included too many variables and interactions for the human designer to cope with infallibly, and even the most careful design reviews and disciplines could not provide sufficient protection. Consequently it was necessary to make the product speak for itself, by causing it to fail, and then to eliminate the weaknesses that caused the failures. The DoD reissued the AGREE report on testing as US Military Standard (MIL-STD) 781, Reliability Qualification and Production Approval Tests.

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 temperatures cycled, vibrated, centrifuged, operated at electrical overstress and otherwise abused, were introduced in place of the traditional sampling techniques. With component populations on even single printed circuit boards becoming so large, sampling no longer provided sufficient protection against the production of defective equipment. These techniques were formalized in military standards covering the full range of electronic components. Components produced to these standards were called 'Hi-rel' components.

Engineering reliability effort in the United States developed quickly, and the AGREE and reliability program concepts were adopted by NASA and many other major suppliers and purchasers of high technology equipment. In 1965 the D0D issued MIL-STD-785- Reliability Programs for Systems and Equipment. This document made mandatory the integration of a program of reliability engineering activities with the traditional engineering activities of design, development and production, as it was by then realized that such an integrated program was the only way to ensure that potential reliability problems would be

detected and eliminated at the earliest, and therefore the cheapest, stage in the development cycle. Much written work appeared on the cost-benefit of higher reliability, to show that effort and resources expended during early development and during production testing, plus the imposition of demonstrations of specified levels of reliability to MIL-STD-781, led to reductions in service costs which more than repaid the reliability program expenditure.

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's and Components.

Specifications and test systems for electronic components, based upon the US Military Standards, have been developed in the United Kingdom and in continental Europe. Electronic component standards including test and quality aspects are being harmonized internationally through the International Electrotechnical Commission (IEC) [14].

2.3 Software Reliability Development

The first study on software reliability appears to have been conducted by Hudson [15]. He reviewed software development as a birth and death process. Fault generation was a birth and fault correction was 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 is obtained if the system test phase is split into three overlapping sub-phases and separate fits made for each.

The next major steps were made by Jelinski and Moranda [16] and Shooman [17]. Both assumed hazard rate for failures that was 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 [17] 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 causes failure.

Schick and Wolverton [18] 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 [19] 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 Schick and Wolverton [20] proposed modified model in which hazard rate is a parabolic function instead of linear function of time.

Schneidewind [21] 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 later paper Schneidewind [22] viewed fault detection per time interval as a Non-Homogenous Poisson 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 [23] has also 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 which also decreases in geometric progression. However, the decrements occur at fixed intervals rather than at each failure correction.

Shortly after some early work, Musa [24] presented an Execution Time Model of software reliability. This theory built on earlier contributions but also broke new ground in

several ways. He postulated that execution time, the actual processor time utilized in executing the program, was the best practical measure of failure, inducing stress, that was being placed on the program. Hence he concluded that reliability theory should be based on execution time rather than calendar time. Musa [24] also had observed that when rates were taken with respect to execution time, the fault correction rate was generally proportional to the fault detection or hazard rate. This observation made consideration of debugging personnel profiles unnecessary. The concepts were tested in real time on four development projects with excellent result. Thus modeling approach became universal and much easier to apply. Hecht [25] has independently verified the simplification resulting from looking at software reliability as a function of execution time rather than calendar time.

A calendar time component was developed for the model that related execution time to calendar time, allowing execution time predictions to be converted into dates. The calendar time component is based on the fact that available resources limit the amount of execution that is practical each time.

A Bayesian approach to software reliability measurement was taken by Littlewood and Verrall [26]. 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 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.

Keiller [27] investigated model similar to Littlewood-Verrall general model. It characterizes the randomness of the hazard rate with the same distribution. However, it uses a different parameter of that distribution to express reliability change.

The differential fault model proposed by Littlewood may be viewed as a variant of general Littelwood- Verrall model [28]. It is similar in viewing the hazard rate as a random variable and using in Bayesian inference. Reliability growth is modeled through two mechanisms. One is number of faults remaining. The second is the variation of the per fault hazard rate with time. Littlewood considers that uncertainties in reliability growth probably result more from uncertainties in the relative frequencies of execution of different input states than uncertainties in fault correction.

Goel and Okumoto [29] developed a modification of Jelinski- Moranda model for case of imperfect debugging. It is based on a view of debugging as a markov process, with appropriate transition probabilities between states. Several useful quantities can be derived analytically, with mathematics remaining tractable. Kremer [30] developed this idea further, including the possibility of introducing new faults due to repair activity. In Goel and Okumoto [31] reasoning from assumptions similar to those of Jelinski-Moranda, described failure detection as NHPP with an exponentially decaying rate function. The cumulative number of failures detected and the distribution of remaining failures are both found to be Poisson. A simple modification of NHPP model was investigated by Yamada, Ohba, and Osaki [32] where the cumulative number of failures detected is described as a S-shaped curve.

Crow [33] proposed a model for reliability estimation of hardware systems during development testing. It is a NHPP process with a failure intensity function that is a power function in time. It can be applied to software with certain ranges of parameter values.

It is seen that much of 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 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 [34] 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.

Measurement is vital element in the practice of engineering. In addition, with respect to the major disasters cited in this chapter, reliability is one of the most important measurements. An understanding of the failure process is central to any effort to model and comprehend reliability.

Chapter 3

Concepts of Hardware, Software & System Reliability.

The *reliability* of a system is defined as the probability that it will adequately perform its intended function—without failure—for a specified interval of time under stated environmental conditions, which may be defined as the user requirements. Roughly, it can be said that, reliability is inversely proportional to the rate at which failures occur [35].

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

3.1 Hardware Reliability

If T is a random variable representing the failure time of a system, then the probability that the system will fail by time t, i.e., the *failure probability*, is

$$F(t) = P[T \leq t] = \int_0^t f(x)\, dx. \qquad (3.1)$$

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

$$R(t) = P[T > t] = 1 - F(t) = \int_t^\infty f(x)\, dx \qquad (3.2)$$

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

The *expected life E[T]*, or *mean time to failure (MTTF)*—also denoted by Θ—is simply the mean or the expected value of the failure density function:

$$E[T] = \int_0^\infty t \cdot f(t)dt . \tag{3.3}$$

It can be shown also that

$$E[T] = \int_0^\infty R(t)dt . \tag{3.4}$$

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 $[t_1, t_2]$, knowing that a failure has not occurred before t_1:

$$\frac{P[t_1 \le T < t_2 \mid T > t_1]}{t_2 - t_1} = \frac{P[t_1 \le T < t_2]}{(t_2 - t_1)\, P[T > t_1]}$$

$$= \frac{F(t_2) - F(t_1)}{(t_2 - t_1)\, R(t_1)}. \tag{3.5}$$

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

$$z(t) = \lim_{\Delta t \to 0} \frac{F(t + \Delta t) - F(t)}{\Delta t \;\; R(t)} = \frac{f(t)}{R(t)}. \tag{3.6}$$

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."

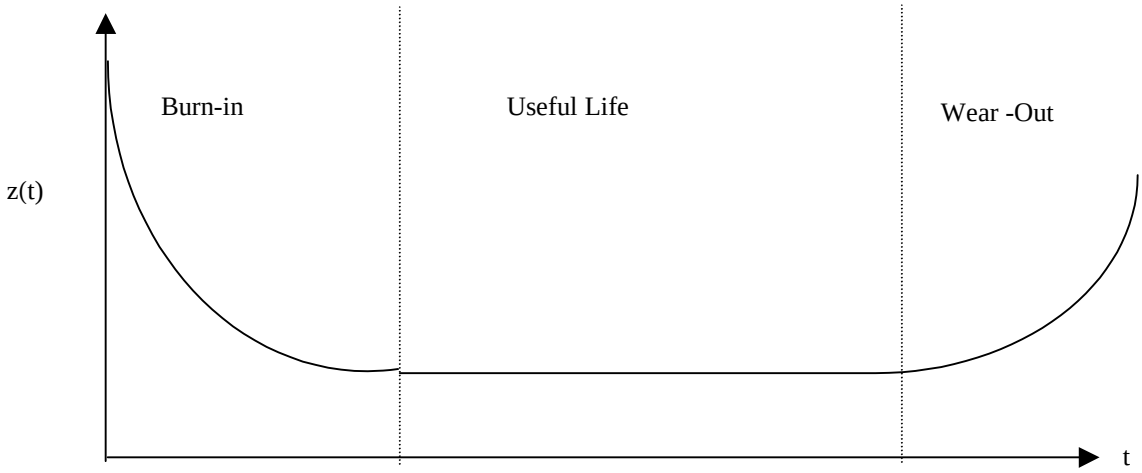


Figure 3.1 The Bathtub-Curve

3.1.1 Exponential Distribution:

For an exponential distribution, the failure rate is constant and equal to λ. This is an important property characteristic of the exponential distribution and does not appear with any other continuous distribution. If a component or a system has a failure free operating time that is exponentially distributed, its behavior in the future will not depend on how long it has already been operating.

Model Form:

A continuous positive random variable *t* has an exponential distribution if cumulative failure distribution is F (*t*) = 1 – exp (–λ*t*).

The probability density function is

$f(t) = \lambda \exp(-\lambda t), \quad \lambda > 0,$

Reliability is given by,

$$R(t) = e^{-\lambda \cdot t} \tag{3.7}$$

3.1.2 Normal Distribution

A commonly used distribution function in theory and practice is the normal distribution. A normally distributed variable assumes values from - ∞ to + ∞. The density of normal distribution is symmetric with respect to the mean of distribution.

Model Form

A continuous positive random variable *t* has a normal distribution if probability density function.

$$f(t) = \frac{1}{\sqrt{2\pi}\sigma} e^{\left(\frac{-1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right)} \tag{3.8}$$

3.1.3 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 it is practically zeroed at the origin, increases to maximum, and then decreases relatively quickly. The Log-Normal function is therefore suitable for modeling repair times. It is also used as a distribution function for the 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 a multiplicative fashion.

Model Form

$$f(t) = \frac{1}{\sqrt{2\pi}st} e^{\left(\frac{-1}{2s^2}\ln\left(\frac{t}{t_{med}}\right)^2\right)} \tag{3.9}$$

3.1.4 Weibull Distribution

It is one of the most useful distributions in hardware reliability. It can be used for both increasing and decreasing failure rate. The Weibull distribution with $\beta > 1$, often occurs

in applications as a distribution of the failure free operating times of components which are subject to wearout and/or fatigue.

Model Form

A continuous positive random variable *t* has a Weibull distribution if, probability density function is

$$f(t) = \frac{\beta}{\theta}\left(\frac{t}{\theta}\right)^{\beta-1}.e^{-\left(\frac{t}{\theta}\right)^{\beta}} \qquad (3.10)$$

3.2 Software Reliability

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 that static. For this reason software reliability is interested in failures occurring and not faults in a program. Reliability measures are much more useful than fault measures. Software reliability may be expected to vary during the software development period. It becomes apparent that the distinction between the terms “failure” and “fault” is an important one.

Failure means a function of the software that does not meet user requirement [36]. It is an external behavior of the system deviating from that is required by its specifications. 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 a failure if it does not meet the specifications.

On the other hand, a *fault*, or bug, is a defect in a program, that when executed under particular conditions will result in a failure. A fault can be a source of more than one failure. By definition, there cannot be multiple faults causing a single failure.

A fault may result from an *error* made by the programmer. Errors occur because of (*a*) incomplete communication between the people involved in a project or between different

times for the same person; (*b*) defective 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.

The probability, or relative frequency of times, that a given program will work as intended by the user, i.e., without failures, in a specified environment and for a specified duration can be termed as *software reliability*. The aim of a software engineer is to increase this probability and make it as close to one as possible. To do this he or she must measure the reliability of the software. A commonly used approach for measuring software reliability is by using an analytical model whose parameters are generally estimated from available data on software failures.

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 in three different ways [36]:

(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 shutdown periods.

Execution time is considered superior to calendar time [36].

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

(A) Time of failure
(B) Time interval between failures (incremental) } time-based

(C) Cumulative failures experienced up to a given time
(D) Failures experienced in a time interval } failure-based

All these 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. Of course, this does not mean that they are completely unpredictable.

A *random process* is 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 *nonhomogeneous*. 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.

When there are no changes in the software, i.e., no debugging and software corrections take place, then λ is constant and a *homogeneous random process* takes place. On the contrary, when software corrections occur, a nonhomogeneous process as described above takes place. Figure 3.2 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.

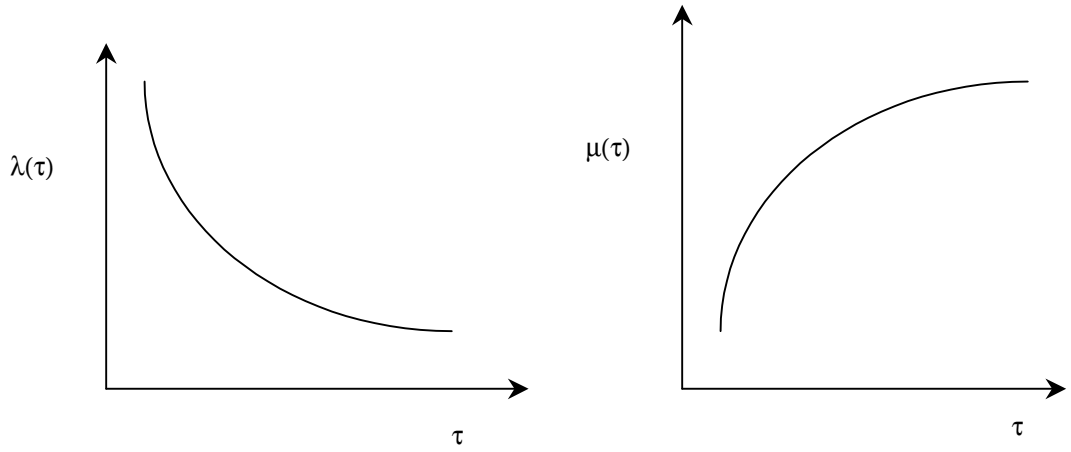


Figure 3.2: Failure Intensity Curve and Mean Value Function.

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

Let $M(t)$ be a random process representing the number of failures experienced by time t. Then the mean value function is defined as

$$\mu(\tau) = E[M(\tau)], \tag{3.11}$$

i.e., the expected number of failures at time t. The failure intensity function of the $M(\tau)$ process is the instantaneous rate of change of the expected number of failures with respect to time, or

$$\lambda(\tau) = \frac{d\mu(\tau)}{d\tau}. \tag{3.12}$$

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 environment [36]. *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 test phase, failure intensity tends to decrease and reliability 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, as in a program that has been released and 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 so extensively in software reliability as in hardware reliability, since in many cases it is undefined. Instead, *failure intensity*, which is roughly the inverse of MTTF, is preferred.

3.2.1 Finite Failure Category Models:

The finite failure category models have a 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: This is an example of where failure distribution exhibits Weilbull 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.

3.2.1.1 Schick-Wolverton Model:

In this model per fault failure distribution is traditional Weibull distributions. This model belongs to finite failures category and is the binomial type. This model has greater flexibility given for the failure modeling because of the shape and scale parameters that define them. The basic assumptions and the data requirements are as follows:

- There are a fixed number of faults in the software at the beginning of the time in which the software is observed.
- The time to failure of fault b0 is distributed as a Weibull distribution with parameter b1 and b2. 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 (the shape parameter) 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-l}, 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., *ti'*s.

The failure intensity and mean value function is given by as follows:

$$\lambda(\tau) = b0 * b1 * b2 * \tau^{b2-1} * e^{-b1*\tau^{b2}} \qquad \mu(\tau) = b0 * (1 - e^{-b1*\tau^{b2}}) \qquad (3.13)$$

3.2.1.2 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 [36]. 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 firsts to use the actual execution time of the software component on a computer for the modeling process. The assumptions and the data requirements for the model are as follows:

1. The cumulative number of failures by time τ, μ(τ), 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

$\lim_{\tau\to\infty} \mu(\tau) = \lim_{\tau\to\infty} b0 * (1 - e^{-b1*\tau})$ it 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 the failure 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.

The data requirements to implement this fault count model are:

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

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

$$\lambda(\tau) = b0 * b1 * e^{-b1*\tau} \qquad \mu(\tau) = b0 * (1 - e^{-b1*\tau}) \qquad (3.14)$$

3.2.1.3 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, however, is a Poisson type. It is a finite failure 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 $\mu(\tau) = b0 * (1 - ((1 + b1 * \tau) * e^{-b1*\tau}))$ for 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 that is, it is a finite failure model.
2. The time between failures of the *(i-1)* st and the i^{th} failure depends on the time to failure of the *(i-1)*st
3. When a failure occurs, the fault, which caused it, is immediately removed and no other faults are introduced.

The 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, $i = 1, \ldots n$.

If data of type 1 are available, the data of the 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 the partition. As a consequence, this model can be used for either the time-between-failures data or the number of faults per time period.

The failure intensity and mean value function is given by:

$$\lambda(\tau) = b0 * b1^2 * \tau * e^{-b1*\tau} \qquad \mu(\tau) = b0 * (1 - ((1 + b1 * \tau) * e^{-b1*\tau})) \tag{3.15}$$

3.2.2 Infinite category model:

For infinite failures category models the number of failures in infinite time is unbounded. That is for these models $\lim_{t \to \infty} \mu(\tau) = \infty$ 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

3.2.2.1 Duane's Model:

While 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 [37]. This process is a NHPP 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 t is taken as a power of t, that is, $\mu(\tau) = b0 * \tau^{b1}$ for some, b0 > 0 and b1 > 0. This model is an infinite failure model as $\lim_{t \to \infty} \mu(\tau) = \infty$. 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 t0 = 0.

The failure intensity and mean value function is given as:

$$\lambda(\tau) = b0 * b1 * \tau^{b1-1} \qquad \mu(\tau) = b0 * \tau^{b1} \tag{3.16}$$

3.2.2.2 Logarithmic Poisson Model:

The logarithmic Poisson proposed by Musa and Okumoto [34] 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 the data requirements are given below:

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:

$$\lambda(\tau) = \frac{b0 * b1}{1 + b1 * \tau} \qquad \mu(\tau) = b0 * \ln(1 + b1 * \tau) \tag{3.17}$$

3.2.2.3 Inverse Linear Model:

This model is as special case of Littlewood Verrall model [26]. 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, . . . 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 is given by:

$$\lambda(\tau) = b0 * \frac{1}{2 * \sqrt{b1 + \tau}} \qquad \mu(\tau) = b0 * (\sqrt{b1 + \tau} - \sqrt{b1}) \qquad (3.18)$$

3.3 System Reliability

To determine the reliability of a large system, it needs to be subdivided into smaller subsystems and elements whose individual reliability factors are known or can be easily determined. Depending on the manner in which these subsystems and elements 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 a 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 Sections 3.3.1 and 3.3.2. A series-parallel network consists of distinct series and parallel sections within the given network. For such a network the reliability analysis is performed in steps as described in Section 3.3.3.

A complex network is one, which cannot be completely decomposed into independent sections of series and/or parallel sub-networks. Reliability analysis for such systems is non-trivial. As a result, many algorithms resort to simplifying assumptions that produce approximate solutions [2]-[5].

3.3.1 Series Systems

Consider a simple system consisting of n units connected in series as shown in the Figure 3.3 below.

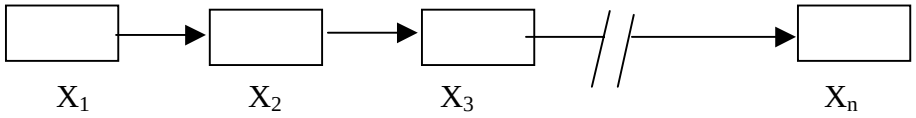


Figure 3.3 Series System

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

R_{sys} = P (X_1 and X_2 and . . . X_n)

P (X_1) x P (X_2 | X_1) x P (X_3 | (X_1 and X_2))...P (X_n | (X_1 and X_2…X_{n-1})) (3.19)

Where P (X_2 | X_1) means the probability of successful operation of unit X_2 under the condition that unit X_1 operates successfully. Likewise, P (X_n | (X_1 and X_2…X_{n-1})) means the probability of successful operation of unit 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:

$$R_{sys} = \prod_{i=1}^{i=n} P(Xi) \qquad (3.20)$$

3.3.2 Parallel Systems

Consider another simple system consisting of n units connected in parallel as shown in the Figure 3.4 below.

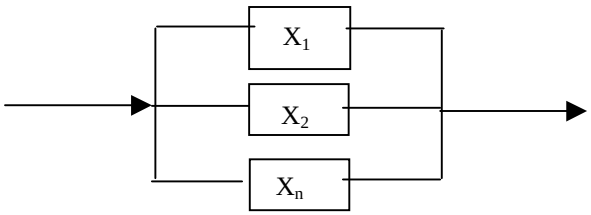


Figure 3.4. A Parallel System

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

Q_{sys} = P (X'_1 and X'_2 and X'n)

= P (X'_1) x P(X'_2| X'_1) x P(X'_3 | (X'_1 and X'_2))...P (X_n | (X_1 …X'_{n-1})) (3.21)

Where P (X'_2| X'_1) means the probability of failure of unit X2 under the condition that unit X_1 also fails. Likewise, P (X'_n | (X'_1 and X'_2 and … 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_{sys} = P (X'_1) x P (X'_2) x P (X'_3)... P (X'_n)

$$Q_{sys} = \prod_{i=1}^{i=n} P(X'i)$$

The system reliability, R_{sys}, is given by

$$R_{sys} = 1 - \prod_{i=1}^{i=n} P(X'i) \qquad (3.22)$$

3.3.3 Series-Parallel Systems

An example of a series-parallel system shown in Figure 3.5.

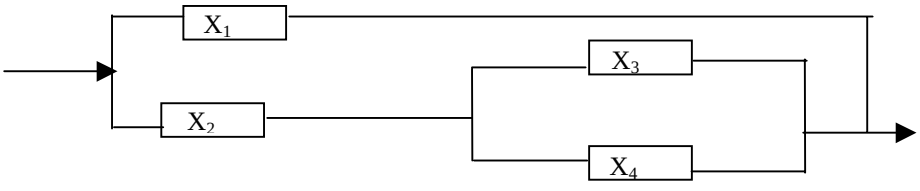
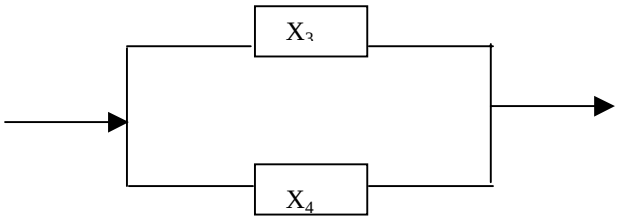
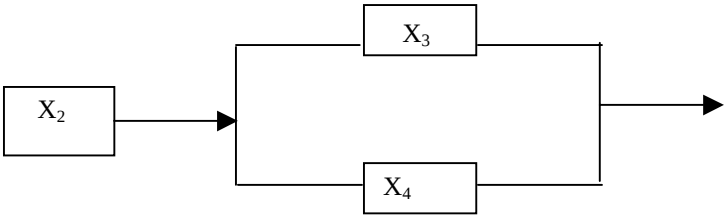


Figure 3.5 Series-Parallel System

Reliability analysis of such systems 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 of Figure 3.5a, units X_3 and X_4 are in parallel and thus form a subsystem identified as subsystem1 and shown in Figure 3.5b.



(a) Figure 3.5a. Subsystem-1



(b) Figure 3.5b. Subsystem-2

Using the expression 3.13 derived for parallel systems, and assuming unit failures as independent events, the reliability of subsystem1 is found as:

$R_{subsys\text{-}l} = 1 - [1- P (X_3)] \times [1- P (X_4)]$

In the next step, note that unit X_2 is in series with the subsystem-l. Let the unit X_2 and subsystem-l form a bigger subsystem, which is identified as subsystem2 and shown in Figure 3.5b. Again, assuming the failure of each unit as an independent event, the reliability of subsystem-2 is given by:

$R_{subsys\text{-}2} = P(X_2) \times [1 - \{1-P (X_3)\} \times \{1- P (X_4))]$

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

$R_{sys} = 1- P(X_1) \times \{$Probability of Failure of subsystem-2$\}$

$R_{sys} = 1- \{1- P (X_1)\} \times [1- P (X_2) \times \{1- (1- P (X_3)) \times (1- P (X_4)))]$

Assuming that all units are identical, i.e. $P (X_1) = P (X_2) = P (X_3) = P (X_4) = p$ and

$P (X'_1) = P (X'_2) = P (X'_3) = P (X'_4) = (1- p)$,

$R_{sys} = p + 2p^2 -3p^3 + p^4$

3.3.4 Complex Systems

A complex system cannot be categorized as a series system or a parallel system or a combination of series and parallel systems. The analysis of such systems is nontrivial and requires special algorithms. Figure 3.6 shows a complex system.

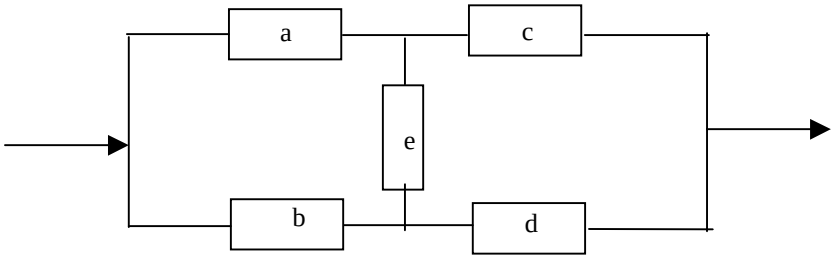


Figure 3.6. A Complex System

The solution of this system is not straightforward. Generally the algorithms to find complex system reliability are classified into two categories. The first category consists of algorithms, which produce the exact solution, and the second category consists of the algorithms, which produce an approximate solution.

The first type of algorithms is complex in nature and involves manipulation with mathematical expression [6]-[9]. In addition to being complex, they are computation-intensive, which leads to two problems in implementing these algorithms. The first problem is concerned with software development that needs some special techniques to overcome the complexity characteristic of the problem. These techniques especially involve structured programming, manipulation of lists, and graceful termination of the program under certain error conditions. The second problem is the limited workspace within the computer system and is overcome by efficiently utilizing the system memory. This is accomplished by

dynamic allocation of memory when needed and immediate release of memory when finished.

The second type of algorithms produces approximate solution of the reliability analysis [2]-[5]. These algorithms are not computation-intensive and their implementation is relatively easy. These algorithms compromise accuracy for a relatively simplified reliability analysis approach.

In the following sections, however, we will focus on some of the existing techniques for finding only the exact system reliability for a complex system. A brief description of these techniques will serve as the background for the research undertaken.

3.3.4.1 Existing Techniques for Complex System Reliability Solution

The conditional probability, cutset, and the tieset techniques give exact reliability solution and are described in this section

3.3.4.1.1 Conditional Probability Technique

As mentioned earlier in this chapter, complex systems cannot be reduced to simple series or parallel models, or combinations of these and therefore their reliability cannot be calculated easily. The concept of conditional probability makes it possible to decompose the complex system into simple series or parallel systems. Solution of these simple models is rather straightforward, as explained above.

The Conditional probability P (A | B) is defined as the probability of occurrence of an event A provided that event B has taken place. If events A and B are independent then

P (A | B) = P(A) and likewise P(B | A) = P(B).

Conditional Probability approach consists of four main steps as outlined below:

- Choose a keystone element (say element i)
- Obtain the system model when keystone element i works

- Obtain the system model when keystone element i fail.
- Calculate the system reliability using the theorem of total probability

R_{sys} = (Probability that a system works | i works) x P (X_i) + (Probability that a system works | i fails) x P(X'_i)

The following example illustrates this approach. Consider the complex structure shown in the figure.

Step 1

Element e is chosen as a keystone element. Choose a keystone element such that the system will reduce to simple series – parallel structure when keystone element is assumed to work or fail. Sometimes this reduction is not possible in the first iteration. In that case more keystone element is chosen, one in each step until simple series – parallel structure is not obtained.

Step 2

Assuming the element e is working the model obtain looks like as shown in Figure 3.6a

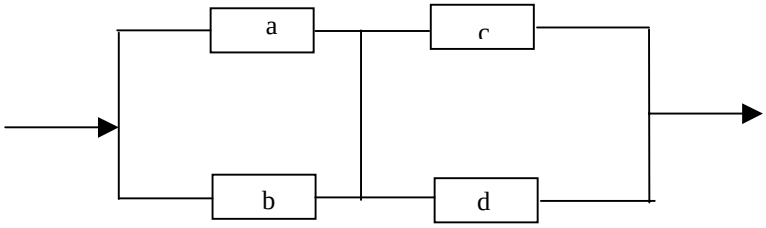


Figure 3.6a Modified System when Keystone Element Works

Step 3

Assuming the element e is not working the model obtained is shown in Figure 3.6b.

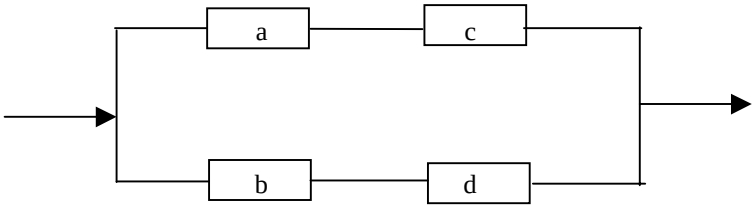


Figure 3.6b Modified System when Keystone Element Fails

Step 4

Let p1 denote the probability that system works when element e is working. Also let Pa, Pb, Pc, and Pd be the probability that the elements a, b, c, and d work, respectively and these events are mutually independent.

Reliability for the system of Figure 3.6a is then obtained as follows.

P1 = (Reliability of a and b in parallel) x (Reliability of c and d in parallel)

= [1- (l-Pa) x (l-Pb)] x [1- (l-Pc) X (l-Pd)]

= [1-(1-Pa-Pb+Pa.Pb)] x [1-(1-Pc-Pd+Pc.Pd)]

=[Pa + Pb - Pa .Pb] x [Pc + Pd - Pc .Pd]

Let P2 denote the probability that the system works when element e is not working.

P2 = (Reliability of a and c in series) in parallel with (Reliability of b and d in series)

P2 = 1- (1 –Pa.Pc) x (1 –Pb.Pd)

= l -(l-Pa.Pc-Pb.Pd + Pa.Pb.Pc.Pd)

= Pa.Pc + Pb.Pd – Pa.Pb.Pc.Pd

According to the conditional probability approach, the system reliability is given by:

R_{sys} = (Probability that system works | e works) x Pe + (Probability that system works | e fails) x (1- Pe)

R_{sys} = (P1) x Pe + (P2) x (1- Pe)

(Pa + Pb – Pa.Pb) x (Pc + Pd – Pc.Pd) x Pe + (Pa.Pc+ Pb.Pd – Pa.Pb.Pc.Pd) x (1- Pe)

Assuming that all units are identical, i e Pa = Pb = Pc = Pd = Pe = p

$R_{sys} = (2p- p^2)^2 * p + (2p^2 -p^4) * (1-p)$

$R_{sys} = 2p^2+2p^3 -5p^4+2p^5$

3.3.4.1.2 Cutset Technique

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

1. Find minimal paths.
2. Find cutsets using the minimal paths found in step 1.
3. Determine the system reliability using cutsets found in step 2.

In this section, techniques to determine minimal cutsets from a set of minimal paths are described. The exact system reliability is then determined using these cutsets.

Determination of Cutsets from given Minimal Paths

Incidence matrix and Boolean algebra techniques are used to find the cutsets of a network. A brief description of boolean algebra technique is given under this section

Boolean Algebra Technique:

The Boolean Algebra technique, which is used to find the cutsets of a network, also assumes that all the minimal paths of the network are known. This technique is simple and can be easily illustrated through an example. The technique involves some manipulation of Boolean expressions and therefore is computation-intensive. For the same system discussed in the previous section, the minimal paths are

Tl =ac, T2=bd, T3 = ade, and T4=bce.

The system will fail if one or more components in each path fail. Hence for T1, either a or c or both must fail to cause the path to fail. This is expressed as (a + c). Since each path must fail to cause the system failure, it imposes a Boolean AND condition to find the cutsets. Hence the expression (a+c) x (b+d) x (a+d+e)x (b+c+e) will give all possible cutsets. To get the minimal cutsets, Boolean simplification has to be performed on this expression. The product of the first three terms is

(a+c) x (b+d)x(a+d+e)

= ab+abe+abd+abc+abd+ad+ad+ade + acd + cd + cde + bce

=ab+ad+cd+bce.

Multiplying with the fourth term

(a+c)x(b+d)x(a+d+e)x(b+c+e)

=(ab+ad+cd+bce) x (b+c+e)

= ab + abe + abc + abd + ade + acd + bcd + cde + cd + bce + bce + bce

=ab + cd + ade + bce.

Therefore ab, cd, ade, and bce are the minimal cutsets for the given network.

Reliability Computation using Cutsets

A system fails if all the components of any of its cutsets fail. This concept is represented by a reliability block diagram using cutsets as shown in Figure 3.7.

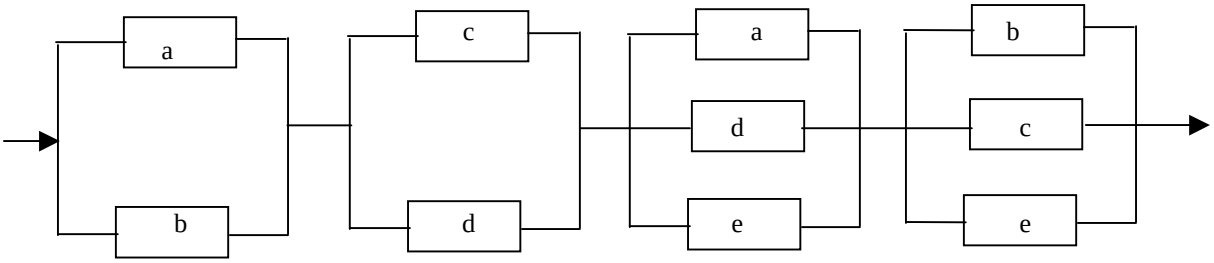


Figure 3.7. Cutset Reliability Block Diagram

Let P (Ci) be the probability of occurrence of cutset i. It is the same as the probability of failure of all elements in cutset i.

Q_{sys} = P (C1 ∪ C2 ∪ C3 ∪ C4)

= [P (Cl) + P (C2) + P (C3) + P (C4)]

=[P (C1∩ C2) + P (C1∩ C3) + P (C1∩ C4) + P (C2 ∩C3) + P (C2 ∩ C4) + P (C3 ∩C4)] [P (C1 ∩ C2 ∩ C3) + P (C1 ∩ C2 ∩ C4) + P (C1 ∩ C3 ∩ C4) +P (C2 ∩ C3 ∩ C4)] - [P (C1 ∩ C2 ∩ C3 ∩ C4)].

Assuming that Qa, Qb, Qc, Qd, and Qe are the probabilities for failure of element a, b, c, d, and e respectively, terms in the above expression are computed as follows.

P (C1) = Qa.Qb

P (C2) = Qc.Qd

P (C3) = Qa.Qd.Qe

P (C4) = Qb.Qc.Qe

P (C1∩ C2) = Qa.Qb.Qc.Qd

P (C1∩ C3) = Qa.Qb.Qd.Qe

P (Cl ∩ C4) = Qa.Qb.Qc.Qe

P (C2 ∩ C3) = Qa.Qc.Qd.Qe

P (C2 ∩ C4) = Qb.Qc.Qd.Qe

P (C3 ∩ C4) = Qa.Qb.Qc.Qd.Qe

P (Cl ∩ C2 ∩ C3)= Qa.Qb.Qc.Qd.Qe

P (C1∩C2 ∩ C4)= Qa.Qb.Qc.Qd.Qe

P (C1∩ C3 ∩ C4)= Qa.Qb.Qc.Qd.Qe

P (C2 ∩ C3 ∩ C4)= Qa.Qb.Qc.Qd.QP

P (C1 ∩ C2 ∩ C3 ∩ C4) =Qa.Qb.Qc.Qd.Qe

If all units are identical and Qa = Qb = Qc = Qd = Qe = Q = (1-p),

$Q = (2Q^2 + 2Q^3) - (5Q^4 + Q^5) + (4Q^5)(Q^5)$

$= 2Q^2+2Q^3 -5Q^4+2Q^5$

$= 2(1-p)^2+2(1-p)^3-5(1-p)^4+2(1-p)^5$

$= 1-2p^2 -2p^3+5p^4-2p^5$

$R_{sys} = (1-Q_{sys}) = 1-(1-2p^2-2p^3+5p^4 -2p^5)$

$= 2p^2+2p^3 -5p^4+2p^5$

3.3.4.1.3 Tieset Technique

The term tieset is defined as a minimal set of working components that cause the system to work. Finding the exact system reliability using the tieset technique consists of the following two steps:

1. Find the tiesets for a given complex network.
2. Determine the system reliability using the tiesets found in step 1.

Determination of Tiesets

The Node Removal and Matrix Multiplication techniques are used to determine the tiesets of a network. Brief descriptions of these techniques are discussed below.

Node Removal Technique

Before introducing this technique, the connection matrix is defined as follows. It is a matrix, which shows the interconnection between all nodes of the system. The order of this square matrix is equal to the number of nodes of the system. An entry in position (i,j) where i not equal to j represents the element that allows signals to pass from node i to node j. An entry of zero means either node i and j are not physically connected or, if they are connected, information cannot flow from node i to node j. All principal diagonal elements of this matrix are 1.To illustrate the construction of the connection matrix, the system of Figure 3.5 is considered again. The signal flow directions are marked and the nodes are numbered as shown in Figure 3.8. Note that the elements a, b, c, and d are unidirectional and can allow information to flow in one direction only. On the other hand, element e is bi-directional. The connection matrix representing this circuit is shown in Figure 3.9.

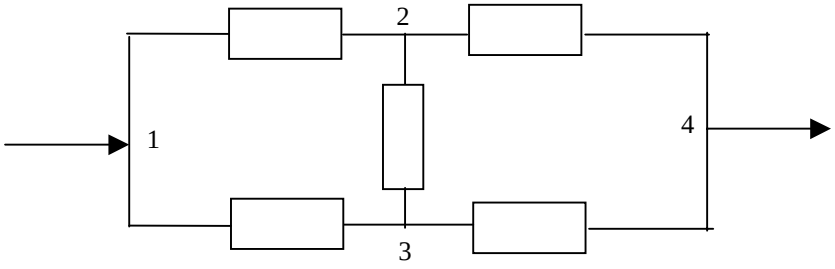


Figure 3.8 A Complex Network

To Node

From Node	1	2	3	4
1	1	a	b	0
2	0	1	e	c
3	0	e	1	d
4	0	0	0	1

Figure 3.9. A Connection Matrix

In this technique all nodes that are not input or output nodes are removed from the connection matrix. As a result, the connection matrix is reduced to a 2x2 matrix whose row numbers represent the source nodes and column numbers represents the destination nodes. The entry (i,j) that corresponds to the source node i and destination node j represents all tiesets from node i to node j. The process of removing a node, say node k, from the connection matrix involves removal of k^{th} row and k^{th} column from the connection matrix. All elements in the reduced matrix are modified such that

$N_{i,j}(new) = N_{i,j(}old) + N_{i,k}(old) \times N_{k,j}(old)$ where $i \neq k, j \neq k$

For example, to determine the tiesets from node 1 to 4, node 2 and node 3 are removed from the connection matrix.

Elements after removal of Node 2

$N_{1,1}(new) = N_{l,l}(old) + N_{1,2}(old) \times N_{2\,l}(old) = 1 + a.0 = 1$

$N_{l,3}(new) = N_{l,3}(old) + N_{l,2}(old) \times N_{2,3}(old) = b + a.e$

$N_{1,4}(new) = a.c$

$N_{3,1}(new) = 0$

$N_{3,4}(new) = d + e.c$

$N_{4,1}(new) = 0$

$N_{4,3}$(new) = 0

$N_{4,4}$(new)= b + a.e

The modified connection matrix is,

To Node

From Node		1	3	4
	1	1	b + a.e	a.c
	3	0	1	d+ c.e
	4	0	0	1

Elements after removal of node 3

The resulting connection matrix after removal of node 3 is

To Node

From Node		1	4
	1	1	a.c +b.d + b.c.e +a.d.e + a.c.e
	4	0	1

$N_{1,4}$ is reduced using Boolean simplifications. Hence,

$N_{1,4}$ = ac + bd+ bce + ade which represents all tiesets from node 1 to node 4.

Reliability Computation using Tiesets

From the definition of the tieset it is clear that the system will work only if all the components in at least one of the tiesets work. The relationship between all the tiesets is expressed by the reliability block diagram shown in Figure 3.10. For the stem shown in Figure 28, the tiesets for node 1 to node 4 are T1 = ac, T2 = bd, T3 = ade, and T4 = bce.

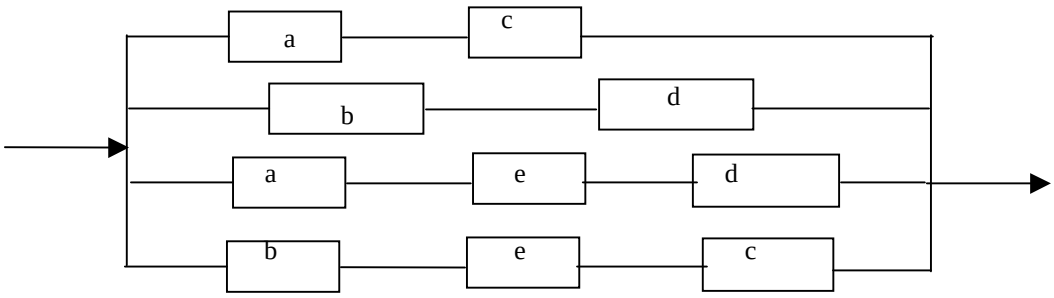


Figure 3.10 Tieset Reliability Block Diagram

P (Ti) be the probability of occurrence of tieset i. It is the same as the probability t all elements in tieset i work.

R_{sys} = P (T1 ∪ T2 ∪ T3 ∪ T4)

= [P (Tl) + P (T2) + P (T3) + P (T4)]

=[P (T1 ∩ T2) + P (T1 ∩ T3) + P (TI ∩ T4) + P (T2 ∩ T3) + P (T2 ∩ T4) +

P (T3 ∩ T4)] + [P (T1 ∩ T2 ∩ T3) + P (T1 ∩ T2 ∩ T4) + P (T1 ∩ T3 ∩ T4) +

P (T2 ∩ T3 ∩ T4)] - [P (T1 ∩ T2 ∩ T3 ∩ T4)].

Let Ra, Rb, Rc, Rd, and Re denote the probabilities of success of element a, b, c, d, and e, respectively. The terms in the above expression are computed as follows.

P (T1) = Ra.Rb

P (T2) = Rc.Rd

P (T3) = Ra.Rd.Re

P (T4) = Rb.Rc.Re

P (T1∩ T2) = Ra.Rb.Rc.Rd

P (T1∩T3) = Ra.Rb.Rd.Re

P (Tl ∩ T4) = Ra.Rb.Rc.Re

P (T2 ∩ T3) = Ra.Rc.Rd.Re

P (T2 ∩ T4) = Rb.Rc.Rd.Re

P (T3 ∩ T4) = Ra.Rb.Rc.Rd.Re

P (Tl ∩ T2 ∩ T3)= Ra.Rb.Rc.Rd.Re

P (T1∩ T2 ∩ T4)= Ra.Rb.Rc.Rd.Re

P (T1∩ T3 ∩ T4)= Ra.Rb.Rc.Rd.Re

P (T2 ∩ T3 ∩ T4)= Ra.Rb.Rc.Rd.Re

P (T1 ∩ T2 ∩ T3 ∩ T4) =Ra.Rb.Rc.Rd.Re

If all units are identical and Ra = Rb = Rc = Rd = Re = p, then

$R_{sys} = (2p^2 + 2p^3) – (5p^4 + p5) + (4p^5) – (p^5)$

$= (2p^2 + 2p^3) – (5p^4) + (2p^5)$

3.4 Computer Algorithm for Calculating Reliability

As described, the above discussion, it is seen that estimating reliability of a complex system is not a straightforward task and hence to deal with this kind of problems, a computer algorithm is described in this section. The simplicity of the algorithm is that it uses sequence of prediction equations, which provides increasingly closer bounds on the system reliability.

This algorithm will use the tieset and cutset determined from one of the techniques discussed above. Let $T_{i,\,i}$ *= 1,2…n* be the number of tiesets. Let $C_{j,\,j}$ *= 1,2 …m* be number of cutset. The preceding statements for system reliability R_{sys} can be expressed as follows:

R_{sys} =Pr {$T_1 \bullet T_2 \bullet \ldots \bullet T_n$} = Pr {at least one tieset is good}

Expressed in terms of cutsets

R_{sys} = Pr {$C_1 \cup C_2 \cup \ldots \cup C_n$}

= Pr {all cut sets are good, viz. contain at least one element of the set which is operative}

Equivalently, the unreliability is expressed as

$1 - R_{sys} = \Pr\{\overline{T_1} \cdot \overline{T_2} \ldots \overline{T_n}\} = \Pr\{\text{all tiesets have failure}\}$ or

$1 - R_{sys} = \Pr\{\overline{C_1} + \overline{C_2} + \ldots + \overline{C_m}\} = \Pr\{\text{at least one cut occurs}\}$

Where $\overline{T_n}$ and $\overline{C_m}$ are the complements of the events T_n and $C_{m,}$ respectively.

From the above exact reliability expressions bounds can be obtained by using the basic probabilistic inequalities

$$R_{sys} = \Pr\{T_1 + T_2 + \ldots + T_n\} \leq \sum \Pr\{T_i\}$$

$$R_{sys} = \Pr\{T_1 + T_2 + \ldots + T_n\} \geq \sum \Pr\{T_a\} - \sum_{i<j} \Pr\{T_i \cdot T_j\} \quad 1 \leq i, j \leq n$$

Thus an upper bound R_{U1} and a lower bound R_{L1} to the reliability are;

$$R_{U1} = \sum \Pr\{T_i\} \tag{3.23}$$

$$R_{L1} = \sum \Pr\{T_i\} - \sum_{i<j} \Pr\{T_i \cdot T_j\} \tag{3.24}$$

In the same manner another upper bound is obtained as;

$$R_{U2} = \sum \Pr\{T_i\} - \sum_{i<j} \Pr\{T_i \cdot T_j\} + \sum_{i<j<k} \Pr\{T_i \cdot T_j \cdot T_k\} \; 1 \leq i, j, k \leq n$$

Similarly inequalities can be applied to cutsets to obtain another lower limit;

$R_{sys} \leq \sum \Pr\{\overline{C_j}\}$ thus,

$R_{L2} = 1 - \sum \Pr\{\overline{C_j}\}$ and by using two terms we get another upper limit;

$$R_{U3} = 1 - \sum \Pr\{\overline{C_j}\} + \sum_{i<j} \overline{C_i} \cdot \overline{C_j} \; 1 \leq i, j \leq m$$

Thus by using the equations of tieset and cutset series of upper bounds and lower bounds are generated which when converge provides the actual system reliabty.

Chapter 4

Design and Development of Software.

The software described in this chapter was developed to facilitate the calculation of reliability of a product or a system using the theories and models described in Chapter 3. Apart from calculating the reliability of a product or a system, it also provides useful statistical information, which helps user to select the best model for a particular set of a failure data.

The software is an integration of three different modules, which are used for estimation of hardware reliability, software reliability and system reliability. Apart from the three different parts, there is also reliability calculator, which helps in determining hardware reliability using the inputs from user.

4.1 Software Development Tools

The software is implemented using Microsoft Visual Basic®5. As Microsoft's premium programming language, Visual Basic®5 is the easiest object-oriented programming approach for application development. It is a sophisticated front end, which has some powerful windows features. The other end is the backend, which the user does not interact with, but this is the part of the software that interacts with the front end. Thus, the front end has the unique ability of interacting with both the user and the backend of software. The Jet Database Engine was used to establish such a connection. Visual Basic®5 program has several outstanding features that have made it an efficient tool for application development. It essentially consists of Forms or Screens, which are any normal screen. In Visual Basic®5, the programmer builds the forms. These forms are similar to the screens displayed by Windows 95, 98/NT, such as log-on dialog box shown at the start up. The only difference is

that the programmer chooses how these dialog boxes appear, when they appear, and what they do. Control Buttons or Toolbox Controls are a set of controls that are used on the custom designed forms. Table 4.1 provides a set of toolbox controls that have been used on the forms to arrive at the desired results in this software. Control Buttons are located in the toolbox and are dragged with the help of the mouse and placed on the screen or form. The positioning of these control buttons are often specified by the software designer.

Table 4.1: Toolbox Controls Used in this Software

Control	**Use/Description**
Text Box	Displays and allows data entry.
Label	Displays plain text on forms, such as captions for other controls.
Command Button.	Initiates an action.
Frame	Group controls that are related in same way; also displays group of option buttons.
MS -Flex Grid	In the form of columns and rows, used to display the input data.
MS- Chart Control	Used to display graphs.
Option Button	Implemented in groups, lets the user make a choice for a small number of options

A number of other controls are also used. The controls are described individually in the forms, where they are used. The Code Window, which is screened for the user, is the driving force behind the software. Another property of Visual Basic®5 that was used in this project is the Grid Control. Grid Controls are the same as any other Visual Basic®5 control objects, except that they may have been given additional properties, events, and methods that allow the user to display the data from the respective data file. The Grid makes it easy to create data-aware input and display data that the user can use to perform the estimation of

reliability. Another feature of this software is that it uses Chart control, which when connected with the columns of the Grid control x-y plots.

4.2 Software Description

Various forms constitute the building block of the software. These forms and associated code allows the user to input the data and estimate the reliability. The forms have a .frm extension. The toolbox controls as described in Table 4.1 are used as required to construct these links. The forms are described as they appear in the program. The code is shown in Appendix F.

- **Menu Form.**

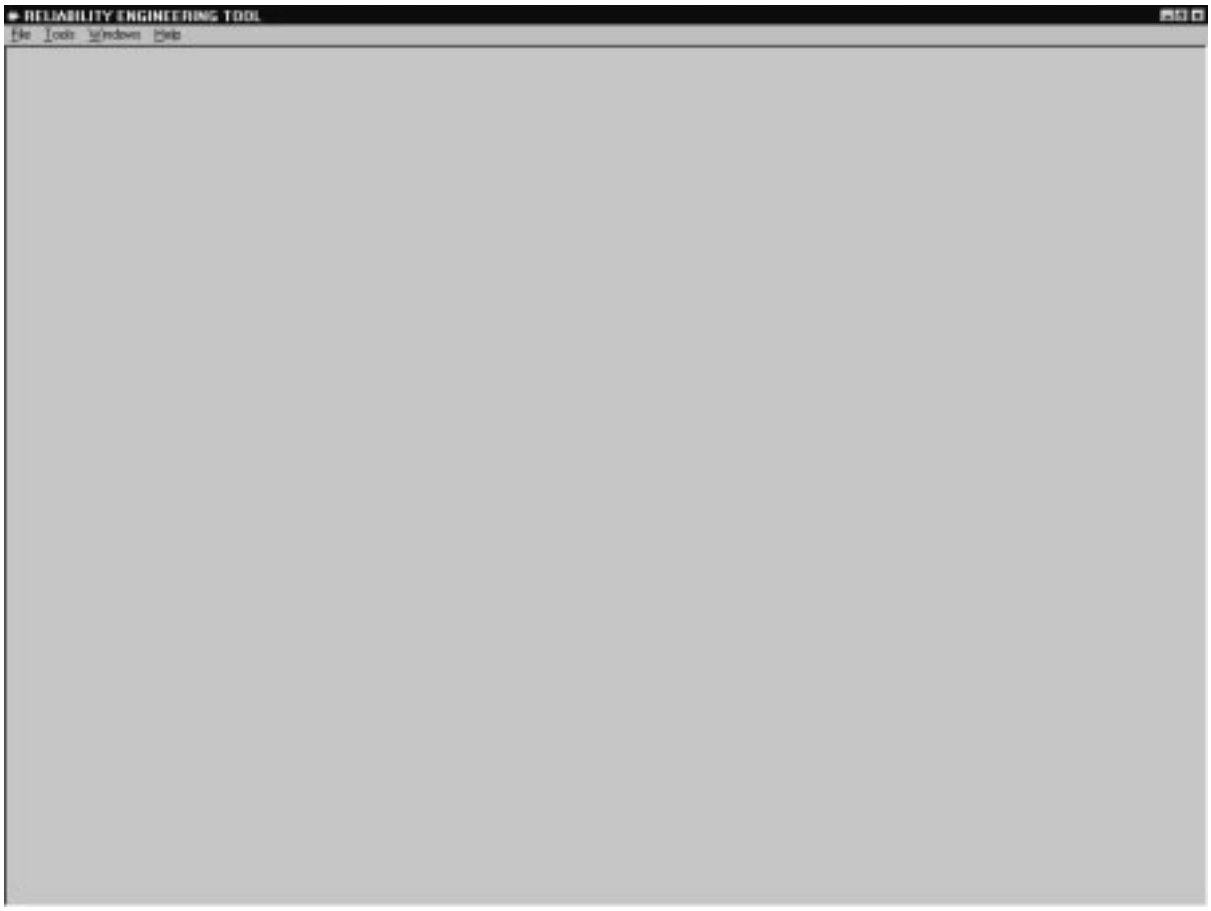


Figure 4.1 Menu Screen

When the software is started the user sees the above Menu form on the screen of computer. Like any other Windows application, this form also has menus for controlling

complete software. Under the menu “File”, there are various submenus like “Open”, “Print”, “Close” and “Exit”. Clicking the “Open” submenu allows selecting which part of software user wants to execute. “Print” submenu prints the specified form. “Close” submenu will close a particular form and “Exit” submenu terminates the software. Apart from the submenus described above, the other important submenu is “Data Editor” which will be discussed as separate form. Under the Menu “Tools” is Reliability Calculator. Since this software was designed similar to other typical software available in the market, it allows the user to open more than one form at the same time and hence there arises need to arrange the forms in a particular fashion. Under “Windows” Menu, user can select the manner in which the forms need to be arranged (Cascade, Tile or as icons). Other important feature of the software is the provision of “Help” Menu that allows the user to learn about the feature and the usage of any particular form.

- **Data Editor**

Like any other editor available in the market, Data Editor is a text editor for this particular software. User can enter the failure data of hardware or software product and can save the file with an extension “hrd” or “sft” respectively. Since the data editor has an interface with the system (desktop) user can anytime recall the data file and make changes to the file. The extension “hrd” and “sft” allows the user to distinguish between hardware product data file and software failure data file. As the software is also meant for calculating the system reliability, user can also input the matrix for tieset, cutset and component reliability and save these files with extension “tst”, “cst” and “rel” respectively. The data file stored through the data editor is then recalled in the respective application for further evaluation of the data.

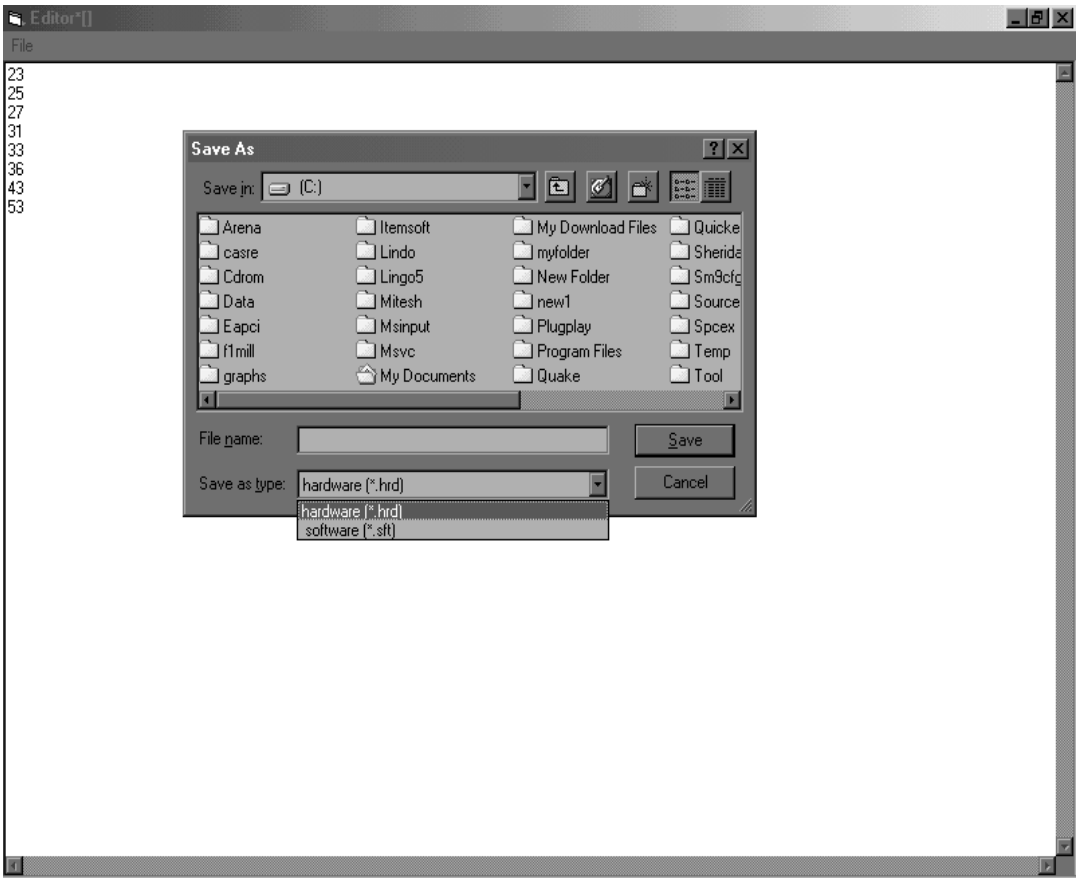


Figure 4.2 Data Editor Screen

- **Hardware Reliability Screen**

This screen opens when the user clicks the hardware reliability option under “open” submenu. This screen helps the user in specifying information about the failure data. On the screen there is a file control provision. This 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, user cannot edit the data through a data grid. If user wants to edit the data, the data file has to be opened in the "Data Editor". User can then select the distribution. This selection is possible through the object “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 then play with data for different distribution and compare the output. Also user can print the screen by selecting the print option under "File" menu. "Clear All" button allows the user to erase all the previously calculated information and start afresh with new data file.

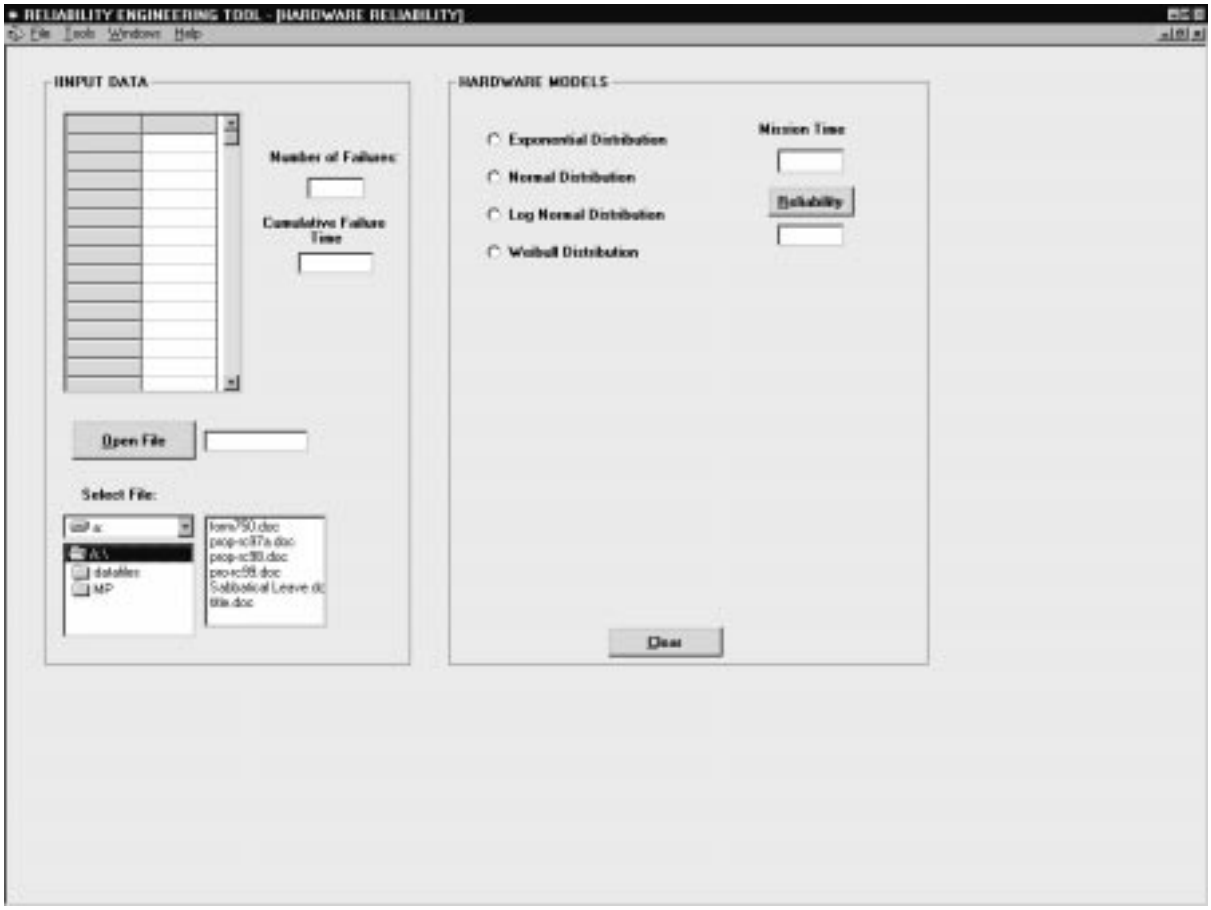
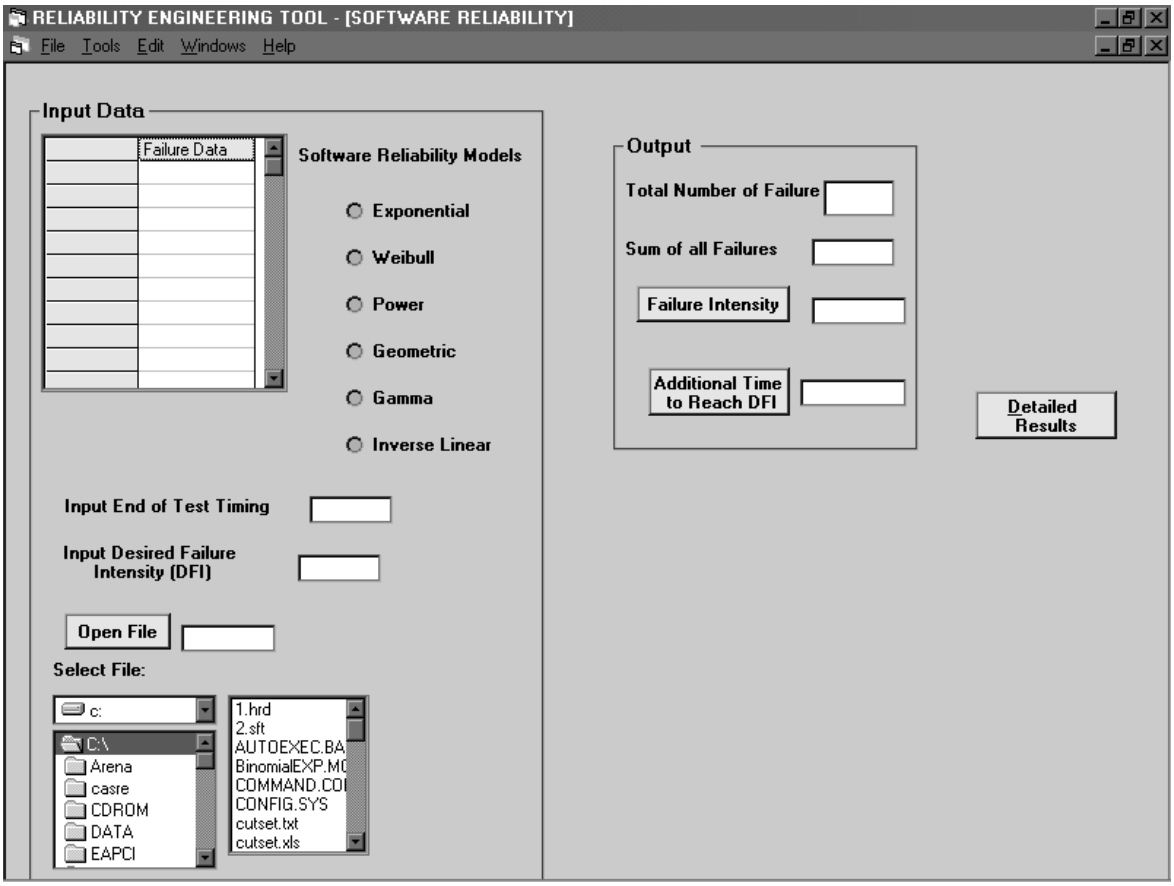


Figure 4.3 Hardware Reliability Screen

- **Software Reliability**

Like hardware reliability this screen is designed to analyze the software failure data. The software reliability screen looks similar to the hardware reliability screen. This is done to maintain homogeneity and to increase the user friendliness of the software. Here also 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 future failure

intensity. 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.

Figure 4.4 Software Reliability Screen

- **Result Screen**

This Screen is associated with the software reliability screen. It cannot be called from the menu bar provided. Once the software reliability screen is executed to analyze the failure data, then result screen can be called by clicking the "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 some important statistical information pertaining to the data is displayed. On clicking the "Graph" button user is able to see various graphs related with the analyzed data.

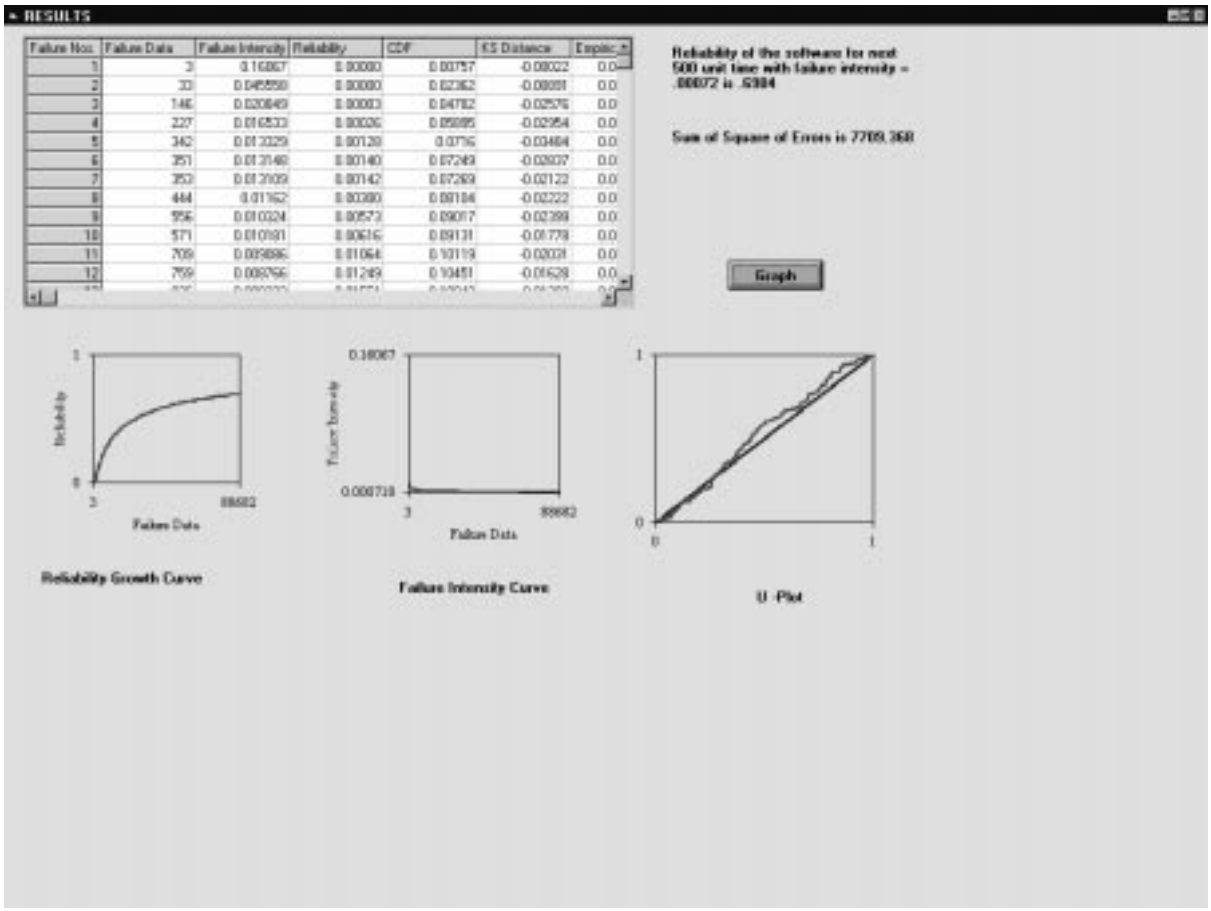


Figure 4.5 Result Screen

- **System Reliability Screen**

This screen is designed to calculate the reliability of a complex system. The setup of this screen is similar to hardware reliability or software reliability screen. When user selects system reliability from the Menu, this screen shows up. There are three separate file controls in this screen, which enables the user to select the respective tieset file, cutset file and component reliability file. Once the files are selected, with the click of the command button "system reliability", program calculates the system reliability for that set of inputs by using the algorithm in chapter 3. The text boxes are used to display the intermediate iterations as well as final reliability of the system. There is also an option to display the final system reliability from two- decimal place to five - decimal places.

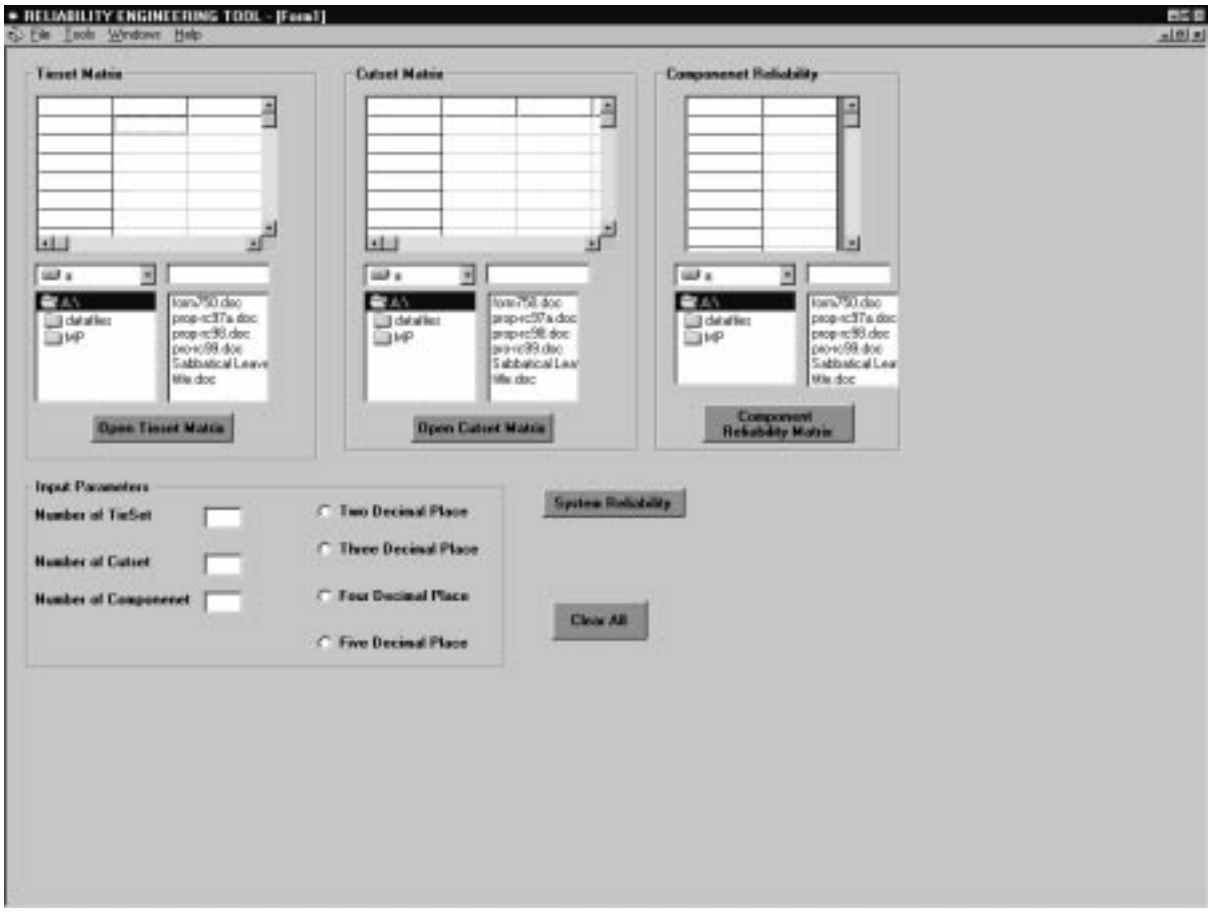


Figure 4.6 System Reliability Screen

- **Reliability Calculator Screen**

This screen was designed to calculate the hardware reliability of a product. It can be found under the menu "Tools". Here a user can input the parameter for a particular distribution. There is no provision for failure data input and hence this screen can only be used when user is aware of parameters of a particular distribution. It becomes easier for the user if reliability has to be calculated for different mission time using the same failure data. Four different distributions are given in the screen.

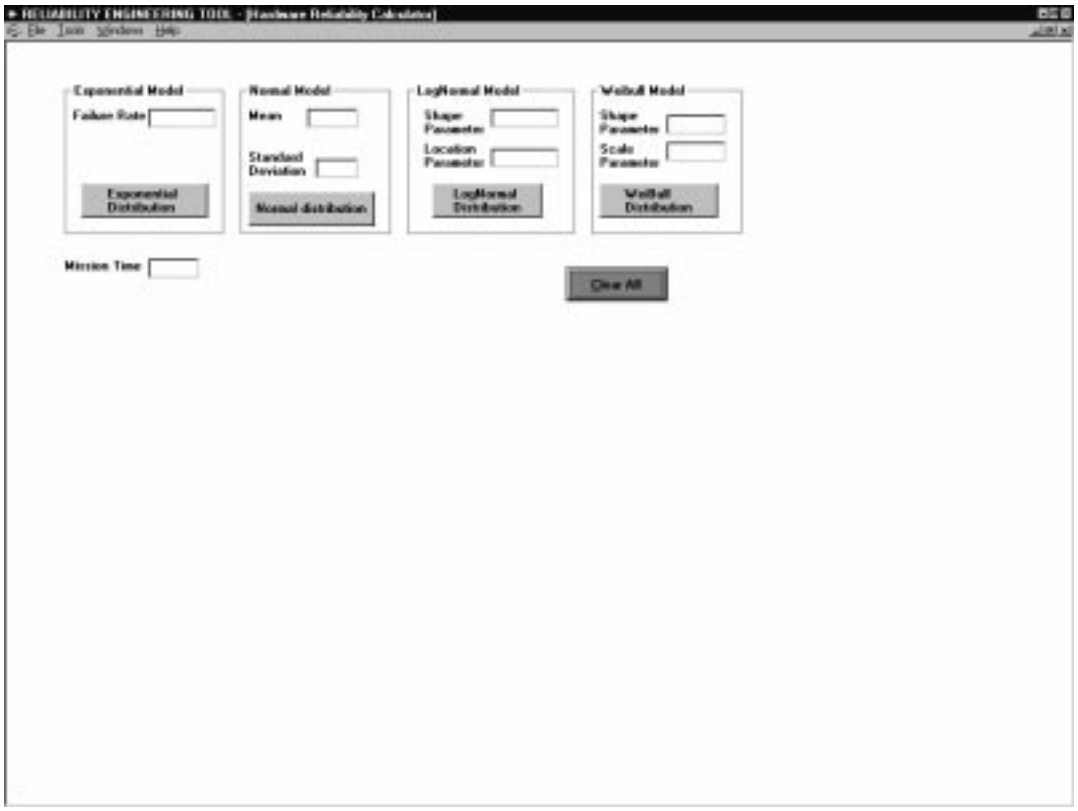


Figure 4.7 Reliability Calculator Screen

- **Input Boxes and Message boxes.**

Input box is a special feature of Visual Basic which allows the user to input data. Input of data can also be possible through the use of text boxes but advantage of using input boxes is user can never miss to enter the critical values, as returning the input box as "blank" will not initiate the code or the program. In this way by using the features of Visual Basic, it is made sure that user always provides the critical information needed for calculation. Figure 4.8 shows the use of input box for entering the critical data used for estimating reliability. Like input box, message box is another special feature of Visual Basic, which allows the software to display any critical message during run–time to the user. Message box can be used to give warnings like user has entered wrong data values or it can be used to give some useful information. Figure 4.9 shows how the message box has been incorporated in the software to make the software more robust.

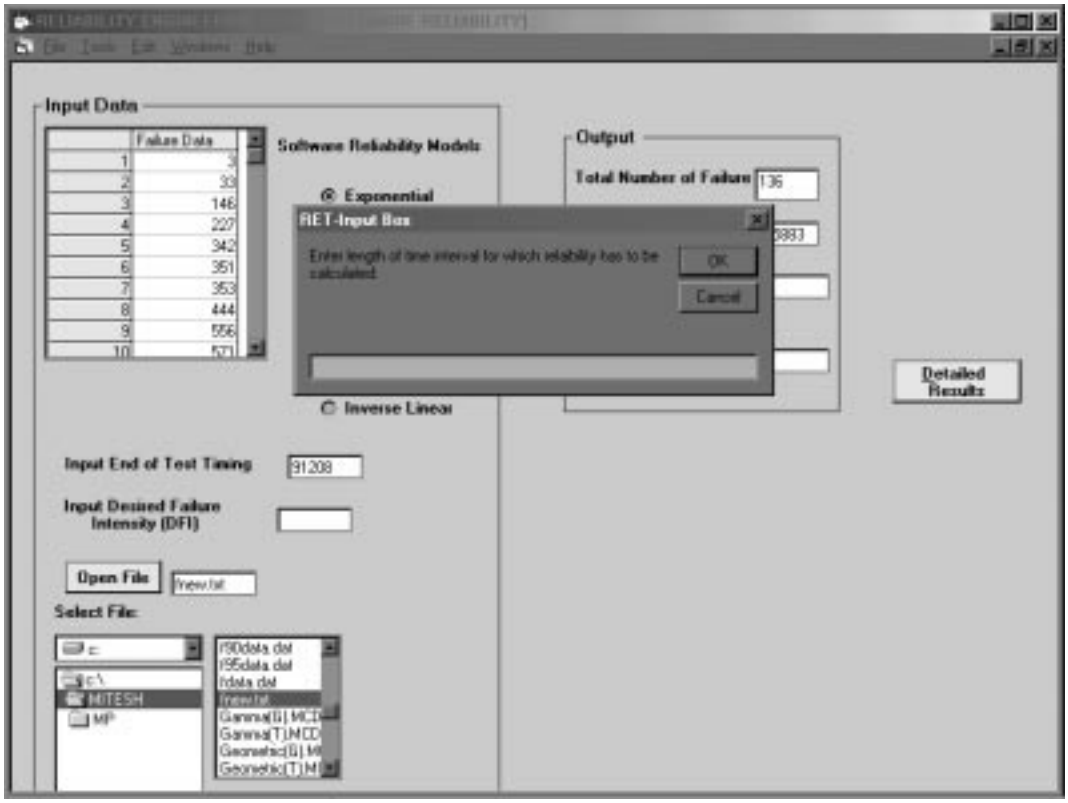


Figure 4.8 Use of Input Box

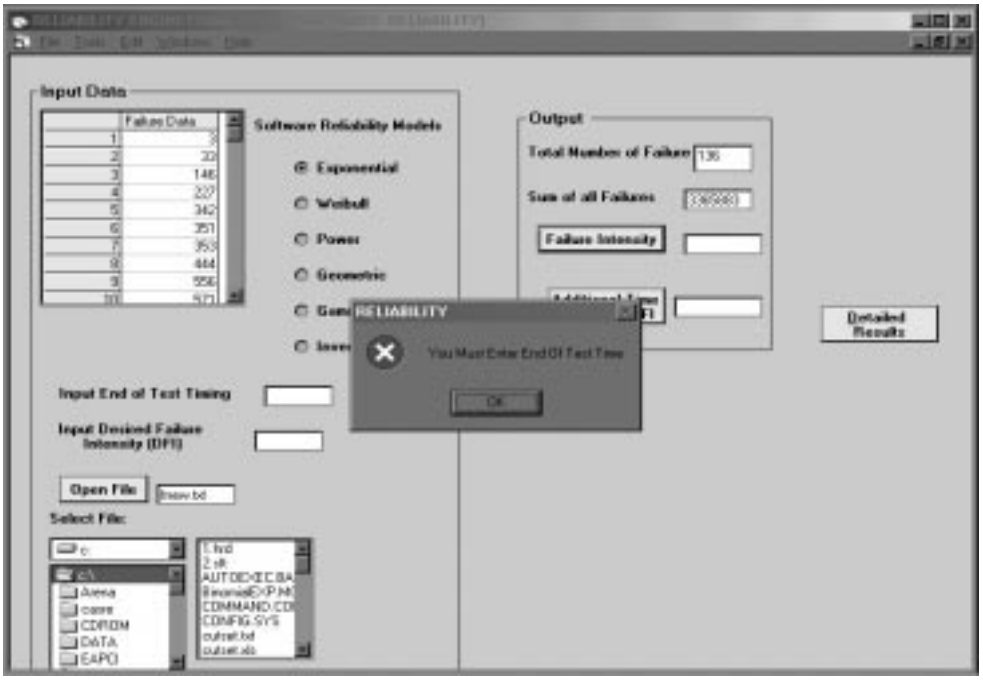


Figure 4.9 Use of Message Box

Thus using the flexibility provided by the visual basic and using the object-oriented feature of the Visual Basic software, it was made possible to design an application to estimate the reliability for not only hardware or software products but also for complex systems.

Chapter 5

Implementation and Validation of Software.

This chapter presents the application of the software tool described in Chapter 4. For hardware reliability module, test cases for different distribution from the literature were run. For software reliability same set of failure data was used to run for different models to see which model is the "best". Also for software reliability, the models were implemented in MATHCAD software package for verification. And for the system reliability, different systems varying from simple series–parallel, to complex systems, having more than 10 components were studied.

5.1 Hardware Reliability Validation.

In this section examples for different hardware distribution will be described and the results obtained will be compared with the ones existing in the literature.

Exponential Model

The following are complete data representing the number of minutes required to perform corrective maintenance on a jet engine.

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

Estimator for the exponential distribution is given by

$$\lambda = \frac{n}{T}$$ Where n = number of failures and T = cumulative test time

$\lambda = 0.02389$ And $MTTF = 41.85Hr$.

Using the expression for reliability $R = e^{-\lambda \cdot t}$ where t = mission time for which reliability has to be calculated. For t = 15 hrs, and plugging the value of $\lambda = 0.02389$, we get the reliability as 0.698. For exponential a specific statistic test, Bartlett's test, was used to check weather the failure data fits the distribution. The complete theory of Bracelet's test is shown in Appendix A. The test statistic is given by

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

Using the values of *n and* t_i we get, *B = 18.258*. As the test statistic under the null hypothesis has a chi-squared distribution with *n-1* degrees of freedom, the failure times will follow exponential distribution if;

$$\chi^2_{(1-\alpha/2,n-1)} < B < \chi^2_{(\alpha/2,n-1)}.$$

Assuming $\alpha = 0.10$ and using the standard Chi Squared table, values on lower and upper bounds were found to be 10.1 and 30.1 respectively. As the value of *B* falls within the range, the distribution is proved to be exponential. Doing this example by hand involves lot

of tedious calculation, but when the same set of failure data is run in the software, it gives the accurate results as shown in Figure 5.1.

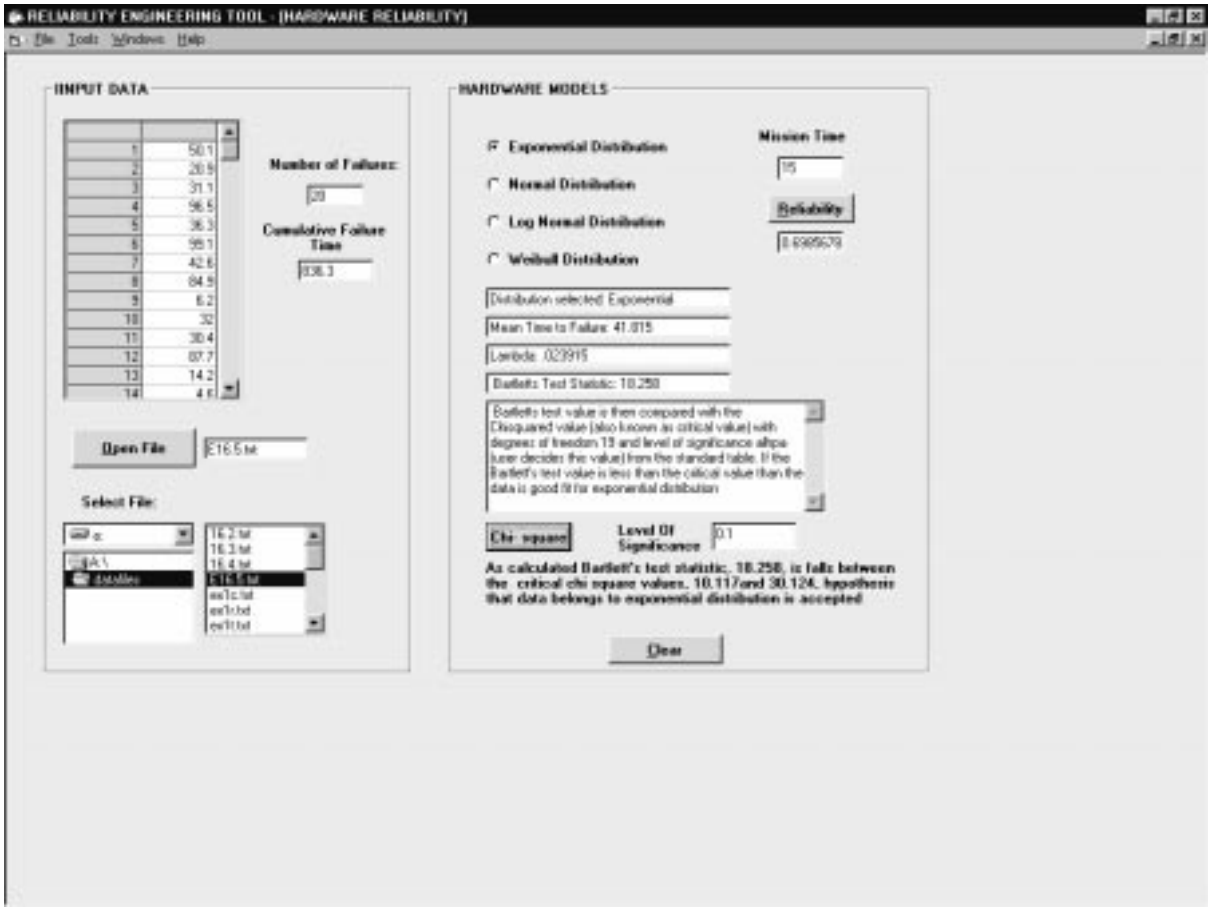


Figure5.1 Demonstration of Hardware Exponential Model

Log-Normal Model

The time to failure of hose assemblies, due to structural fatigue and chemical breakdown, is believed to have a log-normal distribution. The following 25 failure times were obtained from environmental stress testing.

240.5	511.8	1083.4	821.3	1725.4	629.4
326.9	964.8	1677.8	282.3	652.3	639.2
1847.8	670.8	338.8	818.1	1407.5	4991.0
452.0	464.9	734.9	220.2	1078.1	1077.3
1773.0					

Estimator of lognormal distribution are given by,

$$t_{med} = e^{\mu} \text{ Where } \mu = \sum_{i=1}^{n} \frac{\ln(t_i)}{n}$$

$$\text{And } s = \sqrt{\frac{\sum_{i=1}^{n} (\ln(t_i) - \mu)^2}{n}}$$

t_{med}=765.426 and s=0.725. Using the expression of hazard rate function reliability is then calculated as, $R(t) = 1 - \phi\left(\frac{1}{s} \cdot \ln\left(\frac{t}{t_{med}}\right)\right)$, where t is the mission time for which the reliability has to be calculated. For t = 333, we get reliability as 0.8744

For lognormal distribution general test statistic, Kolmogorov-Smirnov test, is used to see weather the data fits the distribution. The complete procedure for Kolmogorov-Smirnov test is shown in Appendix B. The test statistic is given by;

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

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

The failure data will follow Log-Normal distribution when

$D_n < D_{crit}$, where D $_{crit}$ is a value found from the standard Kolmogorov -Smirnov Table. Value of α is assumed by user.

For this particular data max D1 and max D2 are 0.122 and 0.1288. And hence D_n for this data is 0.076. Assuming $\alpha = 0.10$ and using the standard Kolmogorov-Smirnov table, Kolmogorov-Smirnov value was found out to be 0.165. As the calculated value is less than the critical value the set of failure data fits Log-Normal distribution. As seen from the Figure 5.2, same values are achieved.

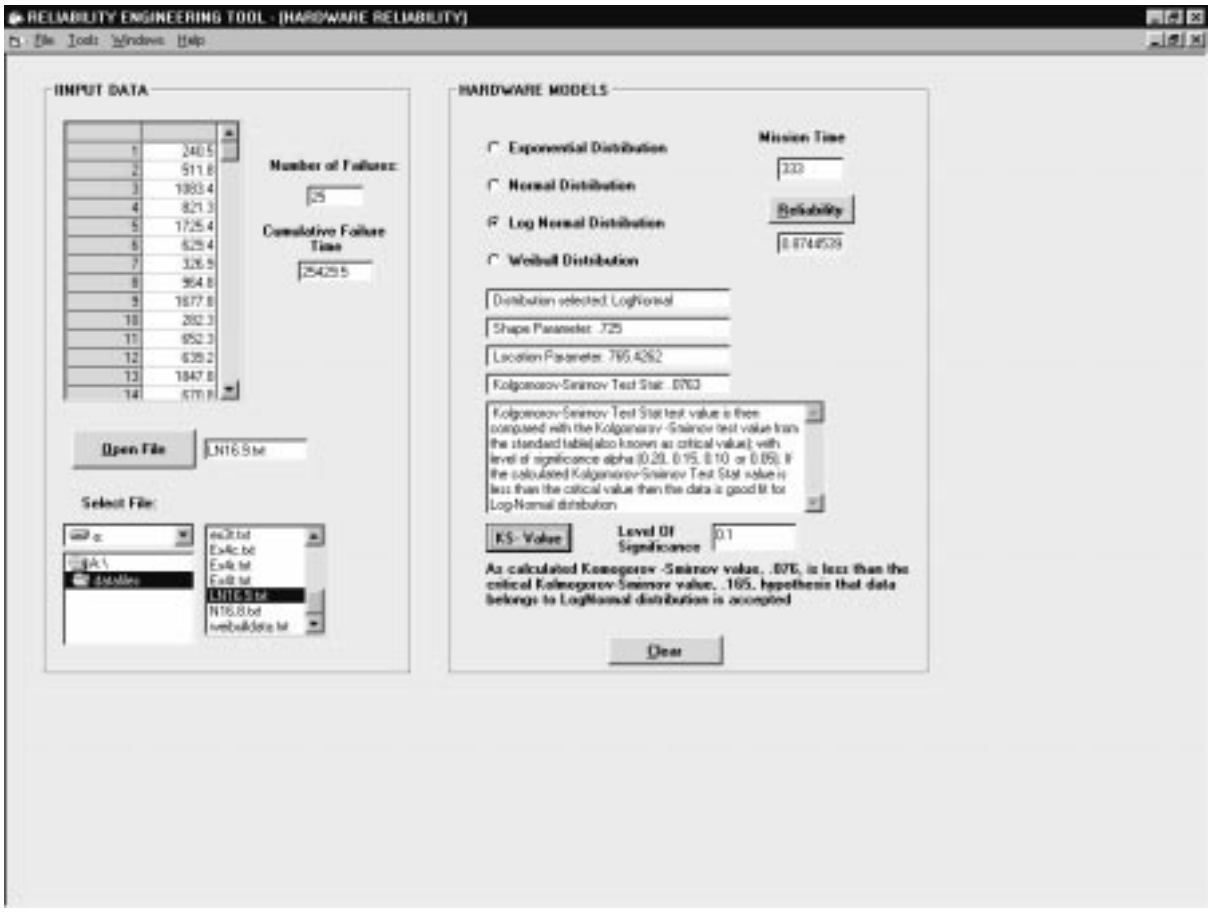


Figure5.2 Demonstration of Hardware Log-Normal model

Normal Model

The following 15 observations represent a sample of repair times, in hours, of a complex piece of machinery.

61.6 70.0 78.4 75.3 83.5 72.3 65.1 77.1 83.2 63.4 72.7 72.5

84.3 73.0 65.5.

Estimators of normal distribution are given as;

$\mu = T/n$ Where, *T* is total test time and *n* number of failure data.

And $\sigma^2 = (n-1)\cdot s^2/n$ where, *s* is population variance and standard deviation is σ.

μ =73.20346 and σ =7.041221 Using the expression of hazard rate function reliability is then calculated as, $R(t) = 1 - \phi\left(\frac{t-\mu}{\sigma}\right)$, where *t* is the mission time for which the reliability has to be calculated. Assuming t = 65, we get reliability as 0.8691

For Normal distribution general test statistic, Kolmogorov-Smirnov test, is used to see weather the data fits the distribution. The complete procedure for Kolmogorov-Smirnov test is shown in Appendix B. The test statistic is given by;

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

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

The failure data will follow Log-Normal distribution when

$D_n < D_{crit}$, where D_{crit} is a value found from the standard Kolmogorov -Smirnov Table. Value of α is assumed by user.

For this particular data max D1 and max D2 are 0.119 and 0.122. And hence the D_n for this data set is 0.122. Assuming $\alpha = 0.10$ and using the standard Kolmogorov-Smirnov table, Kolmogorov-Smirnov value was found out to be 0.201. As the calculated value is less

than the critical value the set of failure data fits Normal distribution.. As seen from the Figure 5.3, same values are achieved.

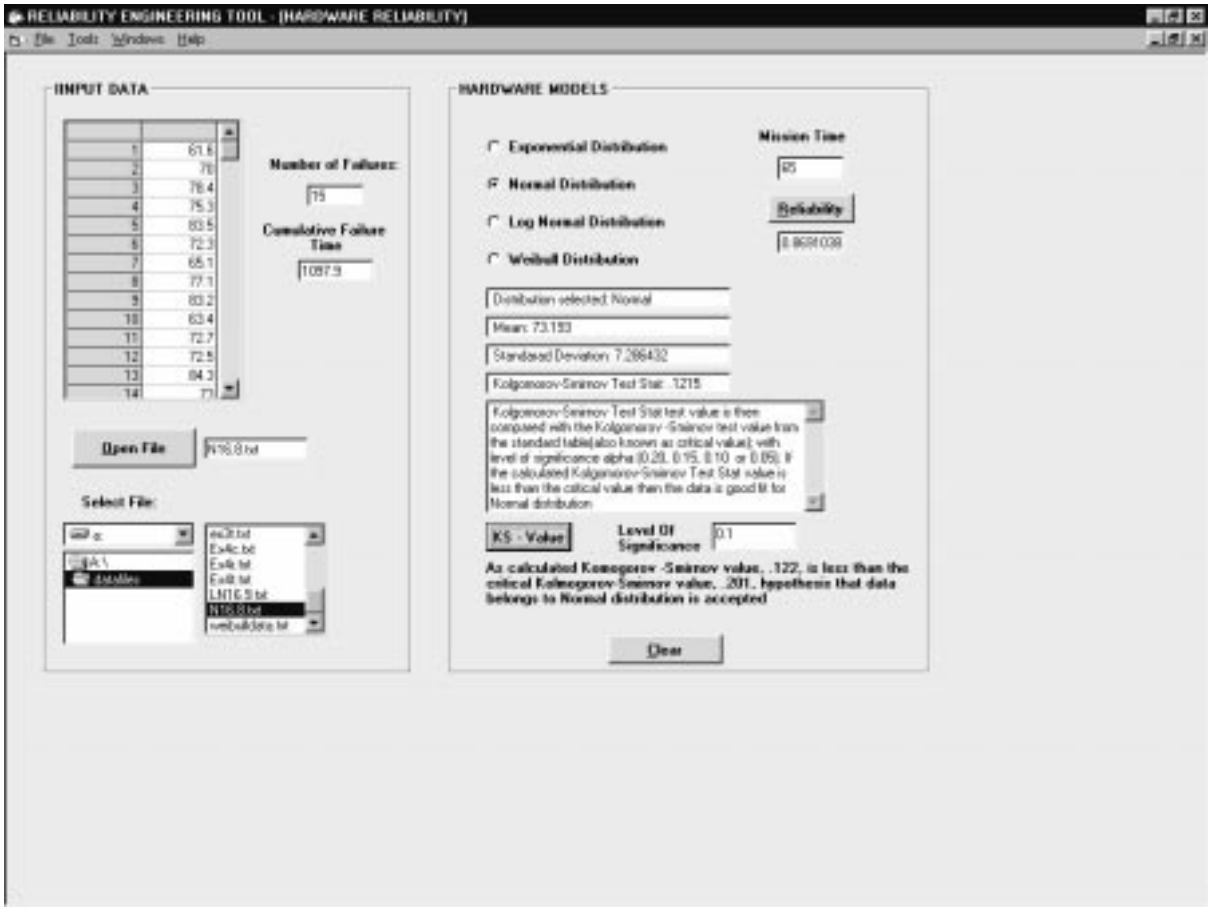


Figure5.3 Demonstration of Hardware Normal Model

Weibull Model

The following failure items were obtained from the testing 15 units until each had failed

112.2 139.8 156.8 113.6 75.5 88.5 73.9 95.5 218 25.1 403.1 150.3 164.5 138.5 151.9.

Estimator for Weibull distribution is given as,

$$\frac{\sum_{i=1}^{n} t_i^{\beta} \cdot \ln(t_i)}{\sum_{i=1}^{n} t_i^{\beta}} - \frac{1}{\beta} - \frac{1}{n} \cdot \sum_{i=1}^{n} \ln(t_i) = 0.$$

The above equation is solved using Newton-Raphson Method (Appendix D)

$$\text{And } \theta = \left(\frac{1}{n} \cdot \sum_{i=1}^{n} t_i^{\beta} \right)^{\frac{1}{\beta}}$$

β= 1.8027 and θ =161.4. . Using the expression of hazard rate function reliability is then calculated as, $R(t) = e^{-\left(\frac{t}{\theta}\right)^{\beta}}$ where *t* is the mission time for which the reliability has to be calculated. Assuming t = 100, we get reliability as 0.66

For Weibull a specific statistic test, Mann's test, is used to check weather the failure data fits the distribution. The complete theory of Mann's test is shown in Appendix C. The test statistic is given by

$$M = \frac{k1 \cdot \sum_{i=k1+1}^{n-1} \left(\ln(t_{i+1}) - \ln(t_i)\right) / M_i}{k2 \cdot \sum_{i=1}^{k1} \left(\ln(t_{i+1}) - \ln(t_i)\right) / M_i}$$

Solving for this particular set of data the value of *M* comes out to be 1.473. The failure data will follows Weibull distribution when

$M < F_{(\alpha,v1,v2)}$ where F-critical may be obtained from the F-distribution table.

For this particular data ν1,ν2 are 14. Assuming $\alpha = 0.05$ and using the standard F-distribution table, F value was found out to be 2.484. As the calculated value is less than the critical value the set of failure data fits Weibull distribution. As seen from the Figure 5.4, same values are achieved.

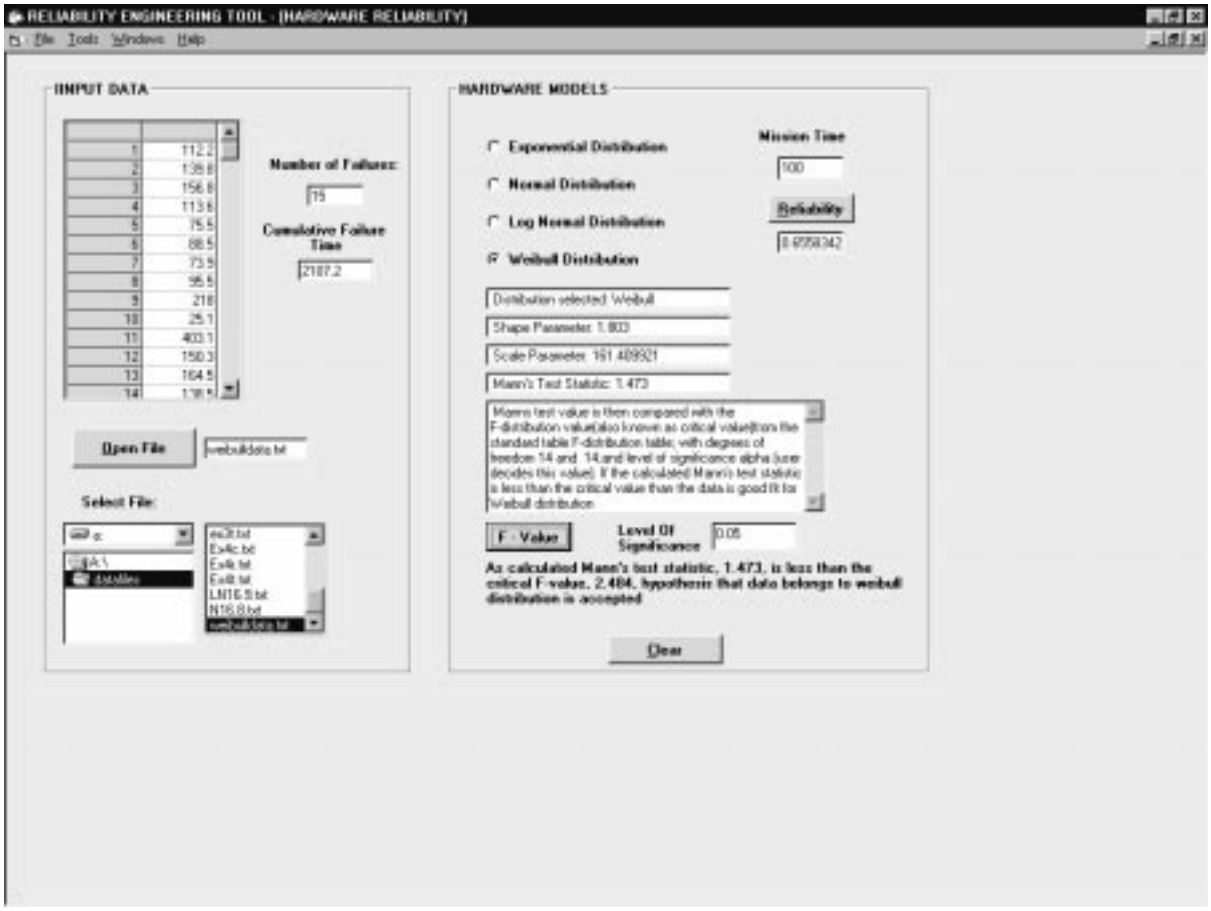


Figure5.4 Demonstration of Hardware Weibull Model

5.2 Software Reliability Validation.

To test the software reliability module, existing data [36], is used. The data recorded in CPU seconds, is time between failures, converted to cumulative failure times, with failures being experienced at times t_1, $t_2 \ldots t_{me}$, for total of *me* failures. The total time of observation or testing is denoted by t_e, which in the example given below does not correspond with last failure. Since the literature does show the results for all the models used in the developed software, all the models were run in MATHCAD and the results are then compared with the one obtained by running the software tool. The failure data is listed as below;

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.

End of testing occurs at 91208.

Exponential Distribution

Estimation of Parameters:

$$\frac{TNF}{b1} - \frac{TNF * TET}{e^{b1*TET} - 1} - \sum_{j=1}^{TNF} TTF_j = 0 \qquad b1 = 3.841 * 10^{-5}$$

$$b0 = \frac{TNF}{1 - e^{-b1*TET}} \qquad b0 = 141.932 \text{ faults}$$

Calculation of Failure Intensity:

$$\lambda(\tau) = b0 * b1 * e^{-b1*\tau} \qquad \lambda(88682) = 2.23 * 10^{-4} \text{ failures/CPU Secs}$$

Cal*culation of Reliability*

Let the mission time be 500 and reliability is calculated after the last failure has occurred.

$R(\tau) = e^{-(\lambda(\tau)\cdot 500)}$ R =0.893

Calculation of Square of Errors

$SSE = \sum_{i=1}^{N} (o_i - e_i)^2$ Where *o* is observed number failure at the time *i* and *e* is the expected number of failure at the time *i*. SSE = 8968.33

Apart from the giving some statistical information, U-Plot is also graphed by the software. U-plot is one mode of predictive analysis. The graph representing the U-plot is line of a unit slope. Any serious departure of the plot from this line is indicative of non-uniformity of the model to that data. The complete explanation of U-plot is given in Appendix E. The result from the software is shown in Figure 5.5.

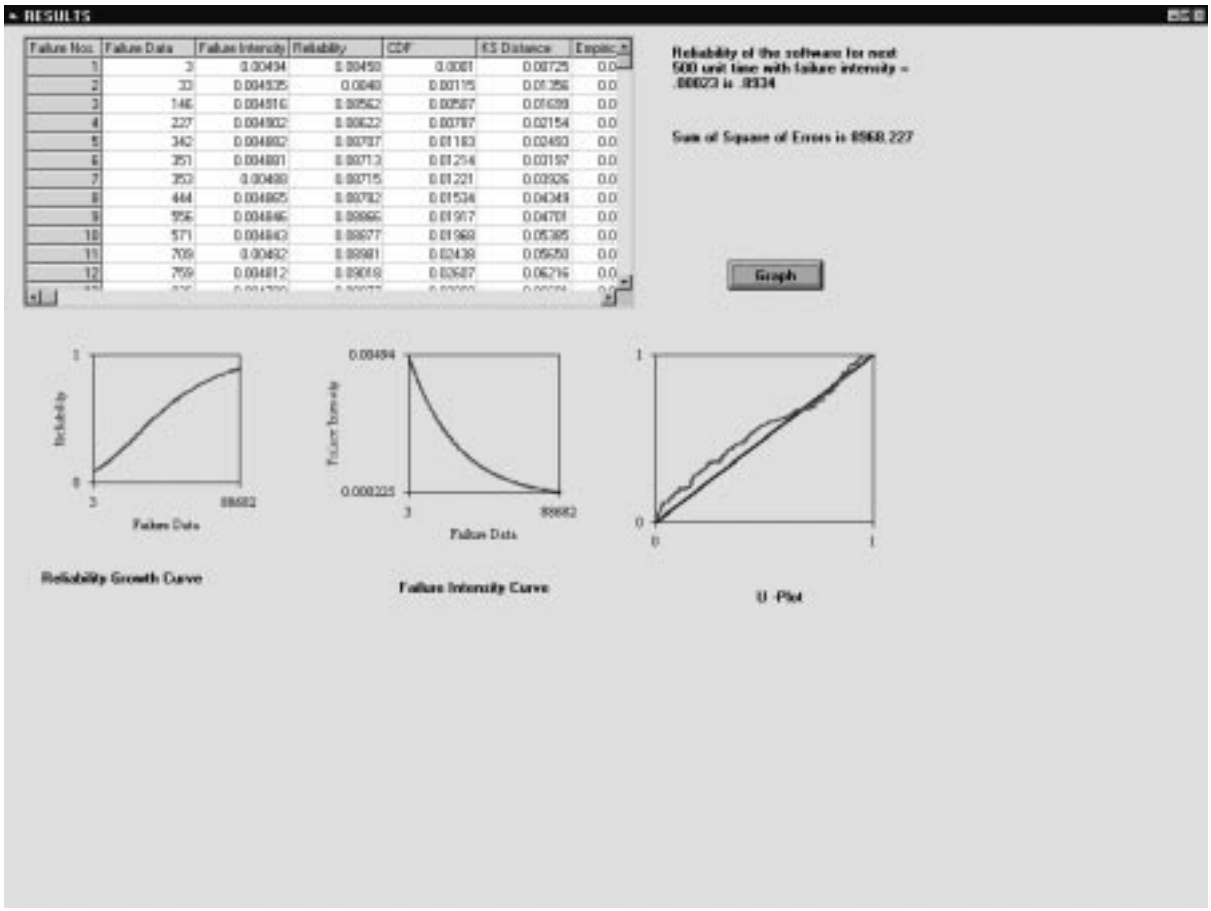


Figure 5.5 Demonstration of Software Exponential Distribution

Weibull Distribution

Estimation of Parameters:

$$b0 = \frac{TNF}{1 - e^{b1 * TET^{b2}}}$$ b0=136 faults

$$b1 = \frac{TNF}{-(b0 - TNF) * TET^{b2} + \sum_{j=1}^{TNF} TTF_j{}^{b2}}$$ $b1 = 8.553 * 10^{-10}$

Calculation of Failure Intensity and Expected Number of Failure:

$$\lambda(\tau) = b0 * b1 * b2 * \tau^{b2-1} * e^{-b1 * \tau^{b2}}$$ $\lambda(88682) = .00002$ failures/CPU Secs

Cal*culation of Reliability*

Let the mission time be 500 and reliability is calculated after the last failure has occurred.

$$R(\tau) = e^{-(\lambda(\tau) \cdot 500)}$$ R =0.988

Calculation of Square of Errors

$SSE = \sum_{i=1}^{N} (o_i - e_i)^2$ Where *o* is observed number failure at the time *i* and *e* is the expected number of failure at the time *i*. SSE = 97656.6

The result from the software is shown in Figure 5.6.

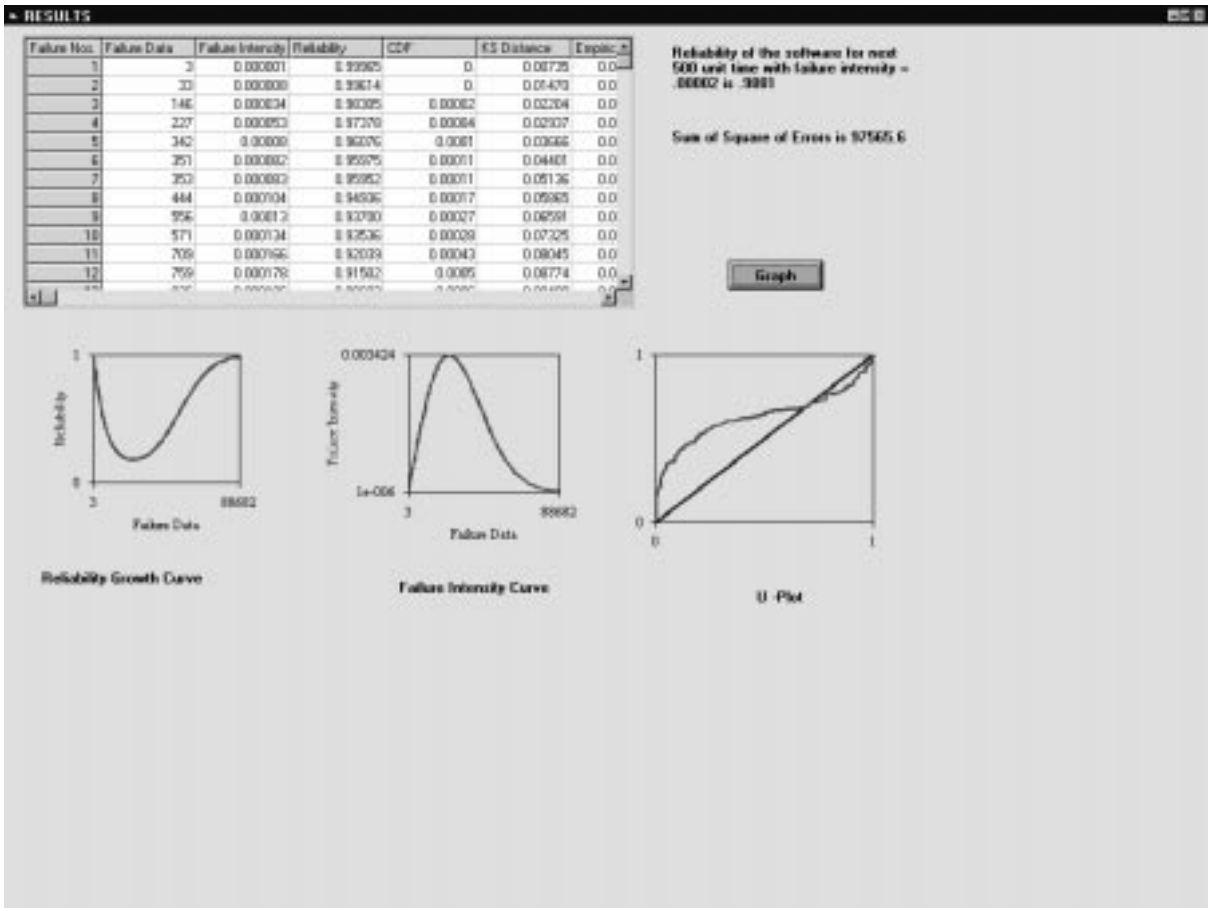


Figure 5.6 Demonstration of Software Weibull Distribution

Gamma Distribution

Estimation of Parameters:

$$\frac{2*TNF}{b1} - \frac{TNF*TET*b1*TET}{e^{b1*TET}-1-b1*TET} - \sum_{j=1}^{TNF} TTF_j = 0 \qquad b1 = 7.927*10^{-5}$$

$$b0 = \frac{TNF}{1-(1+b1*TET)*e^{-b1*TET}} \qquad b0 = 136.82 \text{ faults}$$

Calculation of Failure Intensity:

$$\lambda(\tau) = b0*b1^2*\tau*e^{-b1*\tau} \qquad \lambda(88682) = 0.00007 \text{ failures/CPU Secs}$$

Cal*culation of Reliability*

Let the mission time be 500 and reliability is calculated after the last failure has occurred.

$$R(\tau) = e^{-(\lambda(\tau)\cdot 500)} \text{ R =0.967}$$

Calculation of Square of Errors

$SSE = \sum_{i=1}^{N}(o_i - e_i)^2$ Where o is observed number failure at the time i and e is the expected number of failure at the time i SSE = 41992.3.

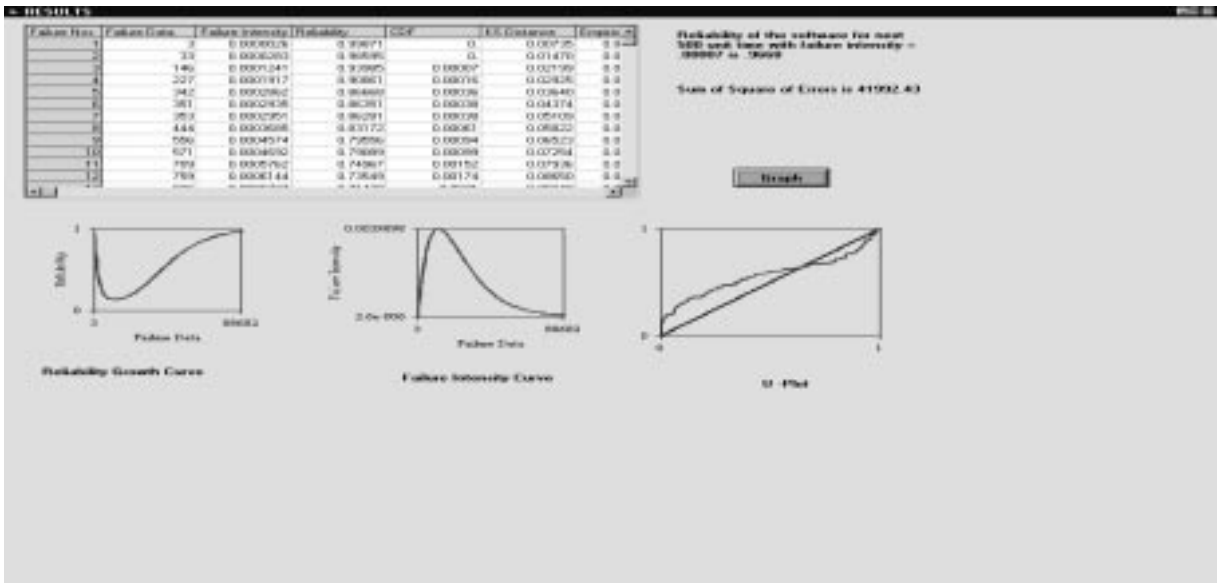


Figure 5.7 Demonstration of Software Gamma Distribution

Power Distribution

Estimation of Parameters:

$$\frac{TNF}{b1} + \sum_{j=1}^{TNF} \ln(TTF_j) - TNF * \ln(TET) = 0 \qquad \text{b1=0.474}$$

$$b0 = \frac{TNF}{TET^{b1}} \qquad \text{b0=0.603 failures}$$

Calculation of Failure Intensity:

$$\lambda(\tau) = b0 * b1 * \tau^{b1-1} \qquad \lambda(88682) = 0.00072 \text{ failures/CPU Secs}$$

Cal*culation of Reliability*

Let the mission time be 500 and reliability is calculated after the last failure has occurred.

$$R(\tau) = e^{-(\lambda(\tau)\cdot 500)} \quad \text{R =0.6984}$$

Calculation of Square of Errors

$SSE = \sum_{i=1}^{N} (o_i - e_i)^2$ Where *o* is observed number failure at the time *i* and *e* is the expected number of failure at the time *i*. SSE = 7709.368

The result from the software is shown in Figure 5.8.

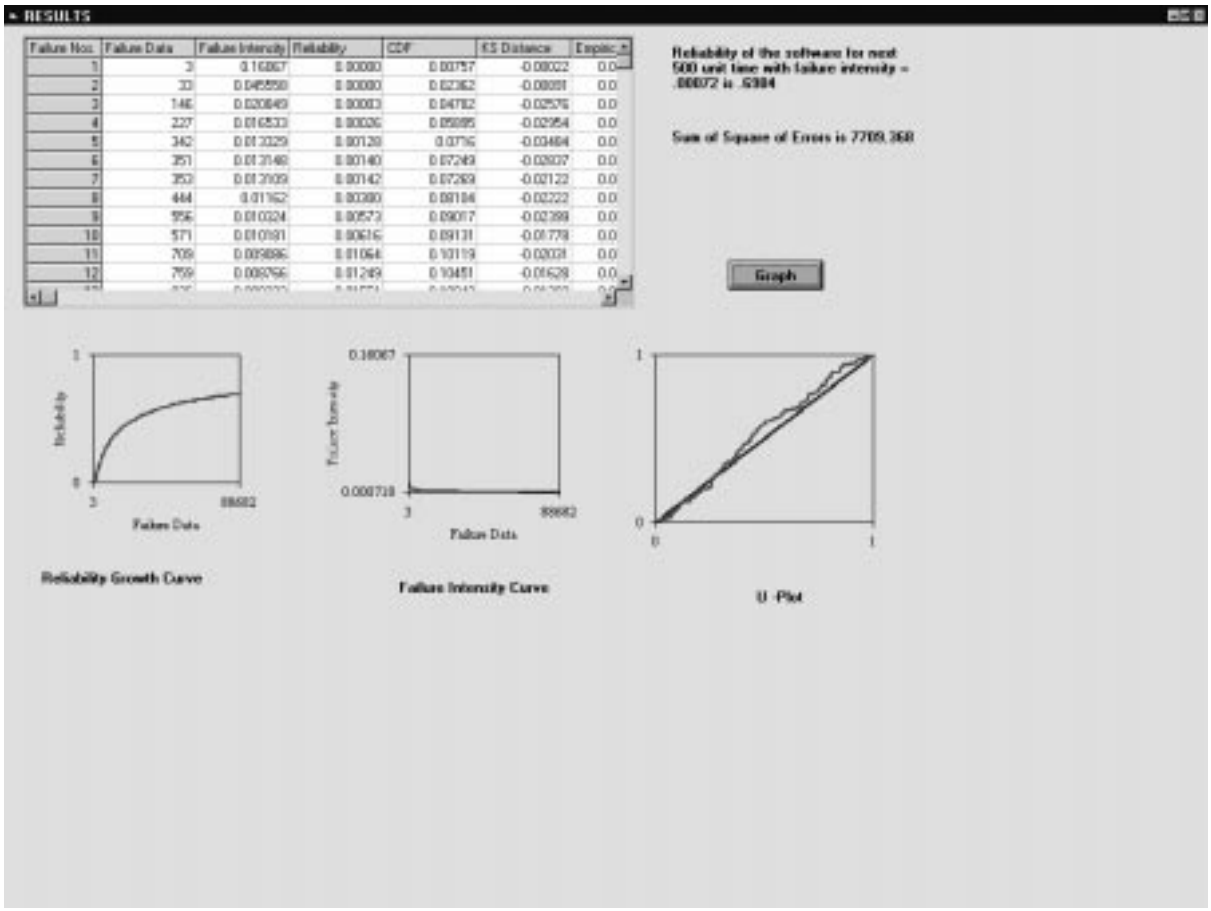


Figure 5.8 Demonstration of Software Power Distribution

Geometric Distribution

Estimation of Parameters:

$$\left(\sum_{j=1}^{TNF} \frac{1}{1+b1*TTF_j} \right) * \frac{1}{b1} - \frac{TNF*TET}{(1+b1*TET)*\ln(1+b1*TET)} = 0 \qquad b1 = 2.623*10^{-4}$$

$$b0 = \frac{TNF}{\ln(1+b1*TET)} \qquad b0 = 42.293 \text{ faults}$$

Calculation of Failure Intensity:

$$\lambda(\tau) = \frac{b0*b1}{1+b1*\tau} \qquad \lambda(88682) = 0.00051 \text{ failure/CPU Secs}$$

Cal*culation of Reliability*

Let the mission time be 500 and reliability is calculated after the last failure has occurred.

$$R(\tau) = e^{-(\lambda(\tau)\cdot 500)} \quad \text{R =0.7761}$$

Calculation of Square of Errors

$SSE = \sum_{i=1}^{N} (o_i - e_i)^2$ Where *o* is observed number failure at the time *i* and *e* is the expected number of failure at the time *i*. SSE = 2467.94

The result from the software is shown in Figure 5.9.

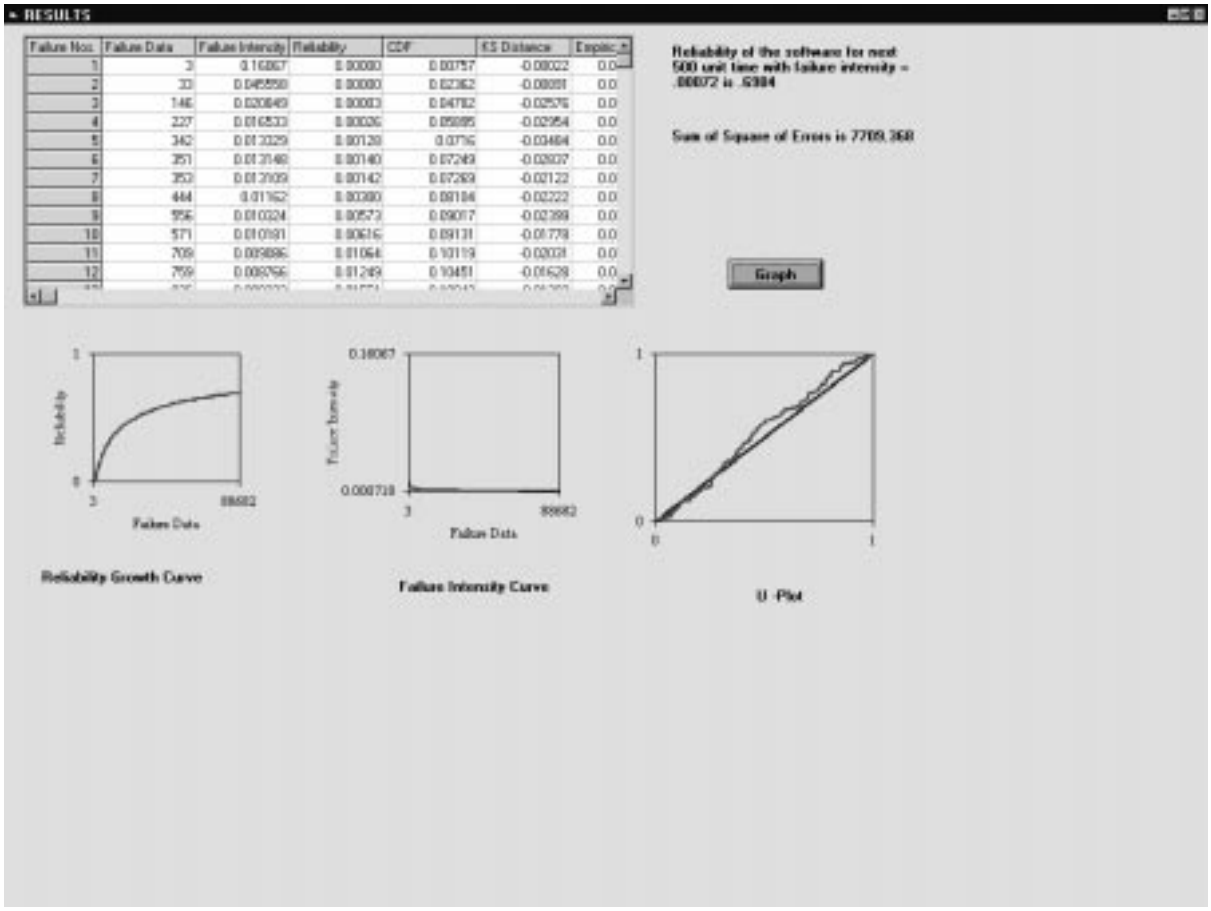


Figure 5.9 Demonstration of Software Geometric Distribution

Inverse -Linear Distribution.

Estimation of Parameters:

$$\sum_{j=1}^{TNF} \frac{-2}{3} * \left(\frac{1}{b1 + TTF_j} \right) - \frac{1}{2} * \left(TNF * \frac{\sqrt{b1 + TET} - \sqrt{b1}}{\sqrt{b1 + TET} - \sqrt{b1}} \right) = 0 \qquad b1 = 1.152$$

$$b0 = \frac{TNF}{\sqrt{b1 + TET} - \sqrt{b1}} \qquad b0 = 0.452 \text{ Faults}$$

Calculation of Failure Intensity:

$$\lambda(\tau) = b0 * \frac{1}{2 * \sqrt{b1 + \tau}} \qquad \lambda(88682) = 0.00076 \text{ failures/CPU Secs}$$

Cal*culation of Reliability*

Let the mission time be 500 and reliability is calculated after the last failure has occurred.

$$R(\tau) = e^{-(\lambda(\tau) \cdot 500)} \text{ R =0.6843}$$

Calculation of Square of Errors

$SSE = \sum_{i=1}^{N} (o_i - e_i)^2$ Where *o* is observed number failure at the time *i* and *e* is the expected number of failure at the time *i*. SSE = 11457.86

The result from the software is shown in Figure 5.10.

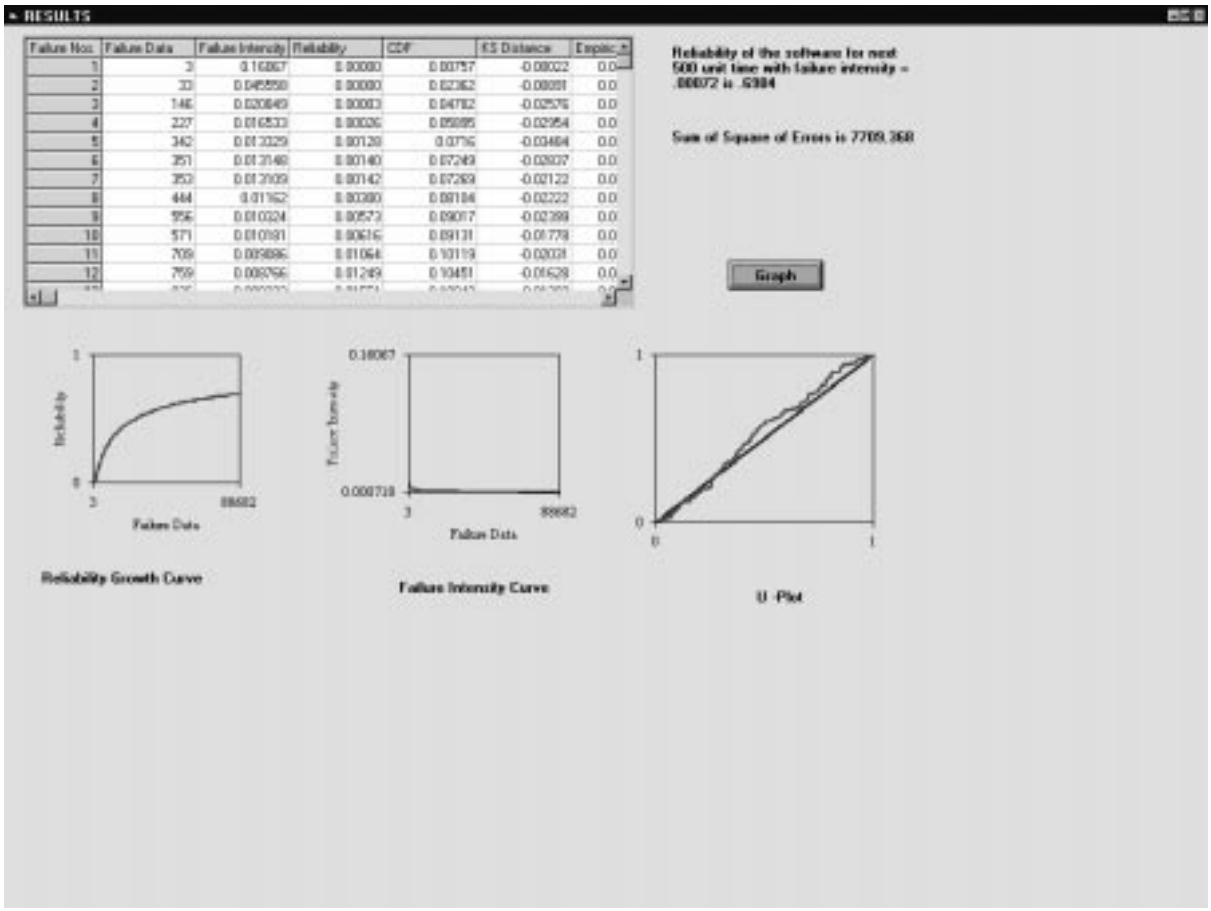


Figure 5.10 Demonstration of Software Inverse Linear Distribution

From the above runs in MATHCAD and the software developed, SSE for geometric model is the least and hence it has the better fit for the data then the remaining models. The fact that geometric model is better for the set of data has also been mentioned in the literature [34]. The analysis in the literature is based on the graphical evaluation of the models.

5.3 System Reliability Validation.

This section presents the results of various kinds of networks analyzed using the software tool developed. The results from the software tool are then compared with the existing solution. The examples analyzed are:

(a) Simple Series-Parallel network

(b) Ladder Network and

(c) A system from Nelson, Batts &Beadles. [3]

Simple Series Parallel Network

Consider the network as shown in the Figure 5.11. The probabilities of success for each of the component is given as Pr{1}=0.93, Pr{2}=0.86, Pr{3}=0.92, Pr{4}=0.95, Pr{5}=0.98.

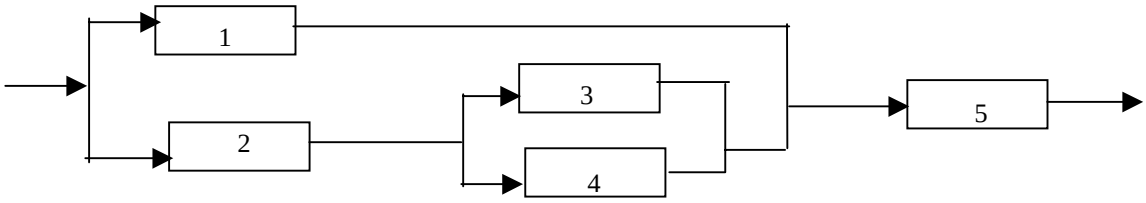


Figure 5.11 Series Parallel Network

As the network shown is simple, manual reliability calculation is shown which uses formulae used for calculating series and parallel function. An assumption is that there is statistical independence between the components.

Step1

Component 3 and component 4 are in parallel and the reliability of that block is given by $R1 = (1-(1-\Pr(3))\cdot(1-\Pr(4)))$. R1 = 0.996. Below is the diagram of system after step1

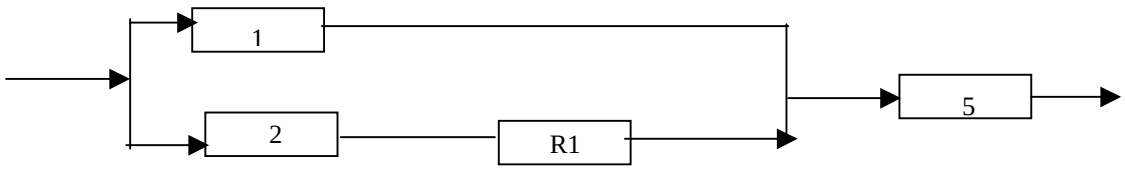


Figure 5.11-a Series Parallel Network

Step 2

From figure 511-a, component 2 and R1 are in series and the reliability for that block is,

$R2 = R1 \cdot \Pr(2)$, R2 =0.9268. Block diagram after step 2 is shown below

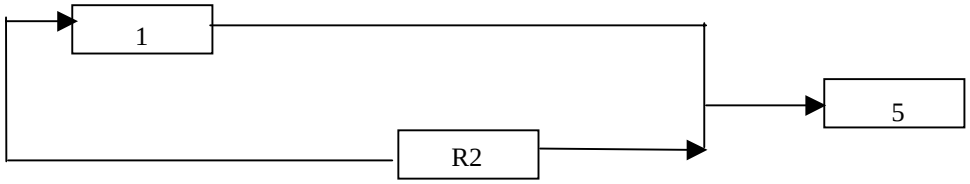


Figure 5.11-b Series Parallel Network

Step3

Component R1 and 1 are in parallel and reliability is given by,

$R3 = (1 - (1 - \Pr(1)) \cdot (1 - R2))$; R3 = 0.9896. Block diagram after step 3 is shown below

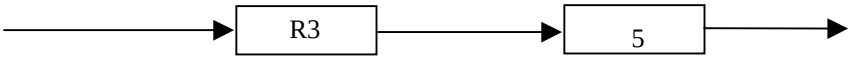


Figure 5.11-c Series Parallel Network

Step 4

Component 5 and R3 are in series thus reducing the system to one component having the reliability equivalent to the reliability of system.

$R4 = R1 \cdot \Pr(5)$; R4 = 0.96988

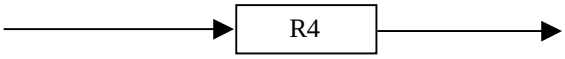


Figure 5.11-d Series Parallel Network (Final)

For calculating the reliability of the above network using the software developed, algorithm described in section 3.4 of Chapter 3 was used. The inputs for the software are tieset matrix, cutset matrix and component reliability matrix. The tieset and cutset for the network are given below,

Tiesets: T1 {2,5}, T2 {1.3,5}, and T3 {1,4,5}.

Cutsets: C1 {1,2}, C2 {2,3,4}, and C3 {5}.

The result from the software is shown in Figure 5-12.

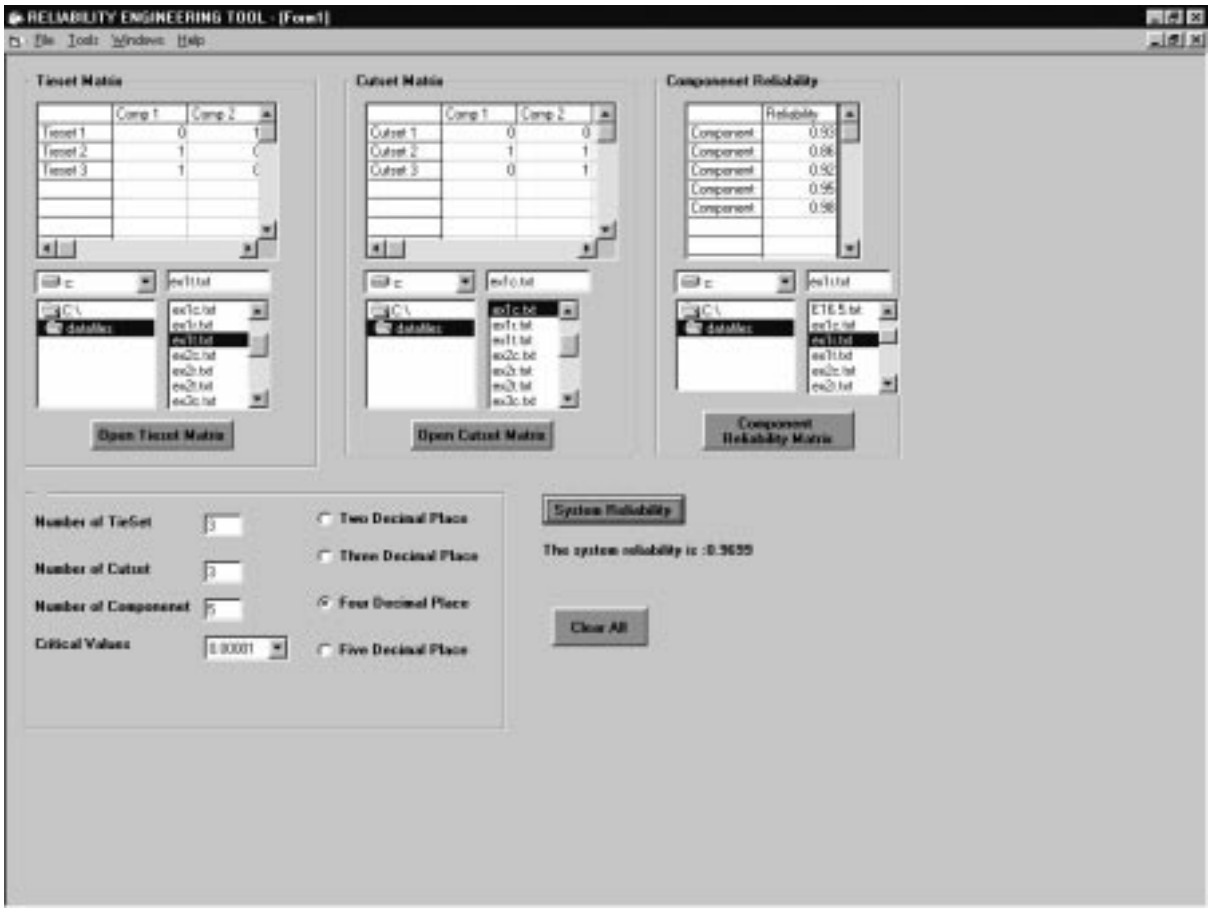
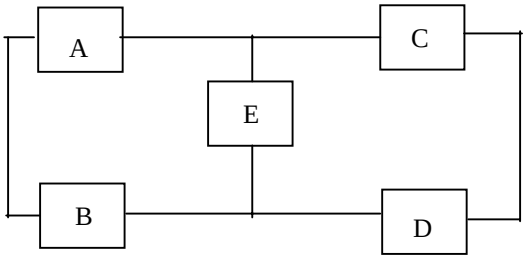


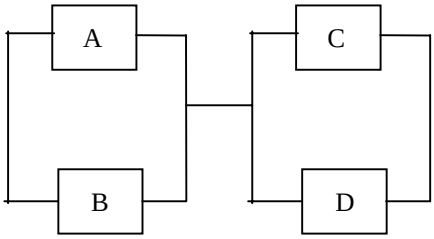
Figure 5-12: System Reliability for Example1 Using the Developed Software.

Ladder Network

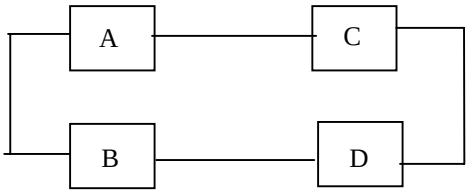
Figure 5-13 -a describes two-stage ladder network. Systems like this cannot be solved by applying simple series parallel rule, as linkages with component E will not permit definition of a subset of components that are strictly in series or in parallel. This kind of system is analyzed using decomposition method.



a: System Block diagram



b: Component E works



c: Component E Fails

Figure 5-13:Two Stage Ladder Network

Decomposition Method

Two sub-networks are created: one shown in figure 5.13-b, in which component E is assumed to be functioning and one Figure5.13-c, in which E has failed. The system reliability is given by,

$$R_s = RE \cdot R(b) + (1 - RE) \cdot R(c)$$

Where R (b) and R(c) are reliabilities of system in Figure 5-13 b and figure 5-13 c.

$$R(b) = [1 - (1 - RA) \cdot (1 - RB)][1 - (1 - RC)(1 - RD)]$$
$$R(c) = [1 - (1 - RA \cdot RB)(1 - RC \cdot RD)]$$

R(b) =0.98725

R(c) =0.978975

And R_s=(0.8)(0.9875)+(0.2)(0.978975) = 0.9858

The result of the software run is shown in Figure 5-14. The tiesest and cutset for the network are listed below:

Tieset: T1 {A,C), T2{B,D},T3{A,E,D},T4{B,E,D}

Cutset: C1 {A,B},C2{C,D},C3{A,E,D},C4{B,E,C}

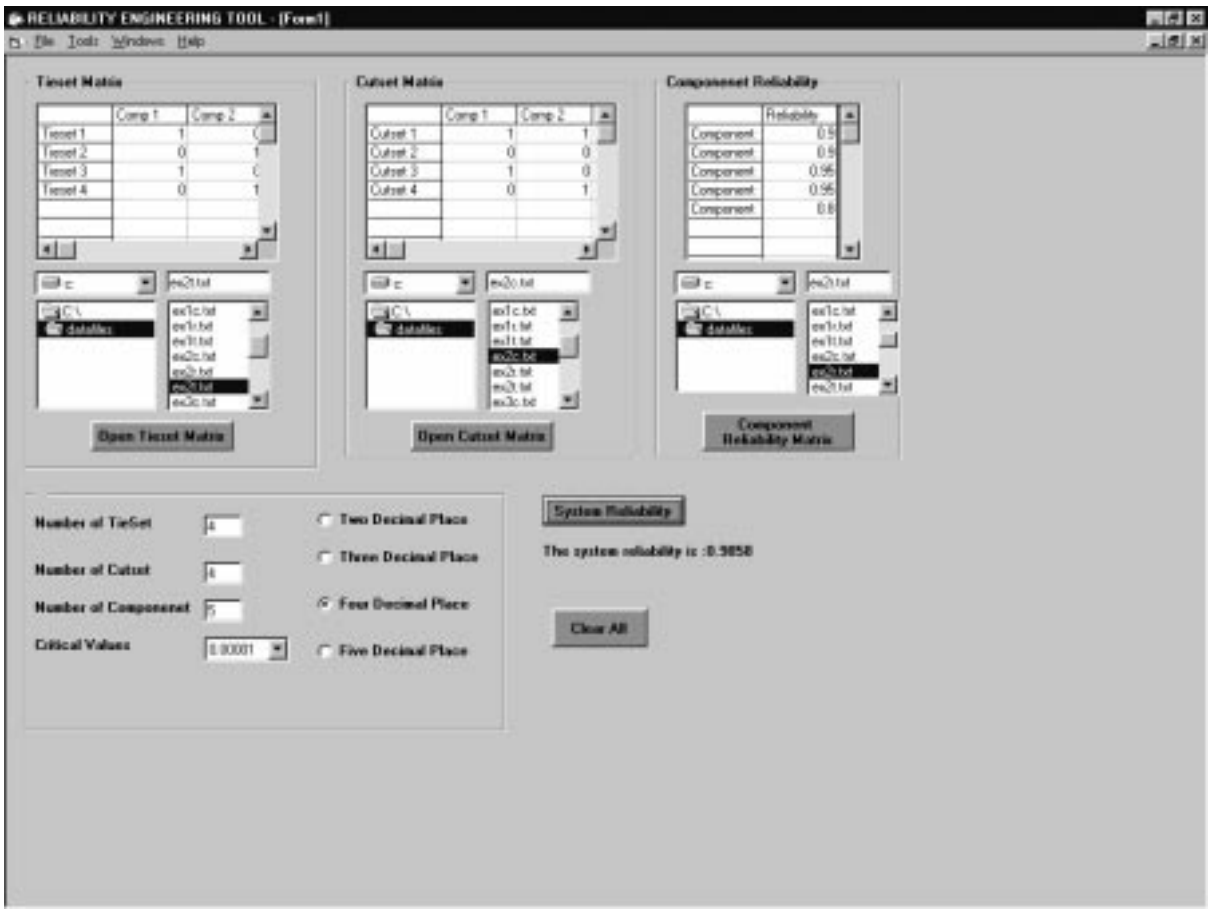


Figure 5-14: System Reliability for Example2 Using the Developed Software.

A System from Nelson, Batts &Beadles.[3]

Figure 5-15 shows a complex system. It has 16 components and arranged in series a parallel combination. Because of large number of components, computation of reliability requires use of some kind of software tool and thus this system was an apt example to verify the software developed under this research. System has 55 tiesets and 10 cutsets. All of the tieset and the cutset along with the individual probability of success of component are listed below:

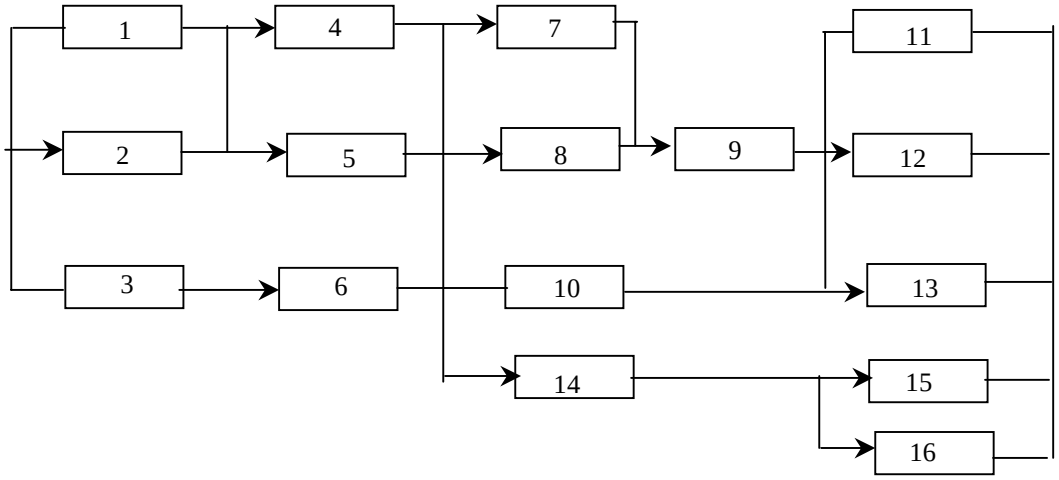


Figure 5 –15: A Complex System

Cutsets:

C1{1,2,3},C2{1,2,6},C3{3,4,5},C4{4,5,6},C5{9,10,14},C6{7,8,10,14},C7{9,10,15,16},

C8{11,12,13,14},C9{7,8,10,15,16},C10{11,12,13,15,16}.

Probability of Success:

Pr{1}=0.80, Pr{1}=0.80, Pr{2}=0.80, Pr{3}=0.90, Pr{4}=0.85, Pr{5}=0.75, Pr{6}=0.82, Pr{7}=0.82, Pr{8}=0.89, Pr{9}=0.88, Pr{10}=0.85, Pr{11}=0.85, Pr{12}=0.85, Pr{13}=0.80, Pr{14}=0.75, Pr{15}=0.70, Pr{16}=0.70.

Tiesets:

Table 5.1 Number of Tiesets for the System in 5-15

S.Nos	Tieset	S.Nos	Tieset	S.Nos	Tieset	S.Nos	Tieset
T1	{1,4,7,9,11}	T15	{1,5,8,9,11}	T29	{1,4,10,11}	T43	{1,5,14,15},
T2	{1,4,7,9,12}	T16	{1,5,8,9,12}	T30	{1,4,10,12}	T44	{1,5,14,16},
T3	{1,4,7,9,13}	T17	{1,5,8,9,13}	T31	{1,4,10,13}	T45	{3,6,7,9,11}
T4	{1,4,8,9,11}	T18	{1,5,10,11}	T32	{1,4,14,15}	T46	{3,6,7,9,12}
T5	{1,4,8,9,12}	T19	{1,5,10,12}	T33	{1,4,14,16}	T47	{3,6,7,9,13}
T6	{1,4,8,9,13}	T20	{1,5,10,13}	T34	{1,5,7,9,11}	T48	{3,6,8,9,11}
T7	{1,4,10,11}	T21	{1,5,14,15}	T35	{1,5,7,9,12}	T49	{3,6,8,9,12}
T8	{1,4,10,12}	T22	{1,5,14,16}	T36	{1,5,7,9,13}	T50	{3,6,8,9,13}
T9	{1,4,10,13}	T23	{1,4,7,9,11}	T37	{1,5,8,9,11}	T51	{3,6,10,11}
T10	{1,4,14,15}	T24	{1,4,7,9,12}	T38	{1,5,8,9,12}	T52	{3,6,10,12}
T11	{1,4,14,16}	T25	{1,4,7,9,13}	T39	{1,5,8,9,13}	T53	{3,6,10,13}
T12	{1,5,7,9,11}	T26	{1,4,8,9,11}	T40	{1,5,10,11}	T54	{3,6,14,15}
T13	{1,5,7,9,12}	T27	{1,4,8,9,12}	T41	{1,5,10,12}	T55	{3,6,14,16}
T14	{1,5,7,9,13}	T28	{1,4,8,9,13}	T42	{1,5,10,13}		

System reliability was found out to be 0.97226, which is same as that stated in the paper, by Nelson, Batts & Beadles. Figure 5-16 shows the results obtained by running the software.

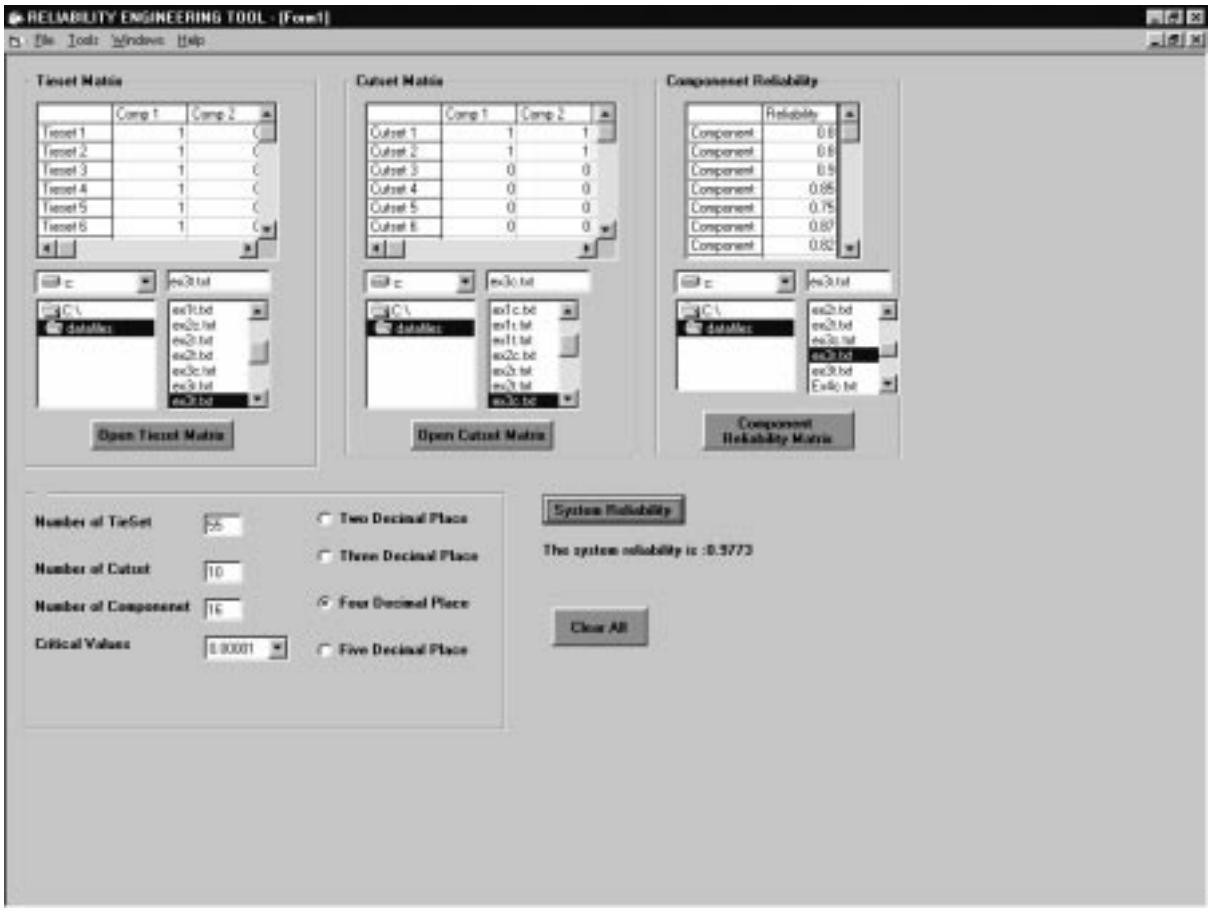


Figure 5-16: System Reliability for Example-3 Using the Developed Software.

Chapter 6

Conclusions and Future Work.

6.1 Conclusions:

The aim of this research was to develop a software tool to estimate hardware, software and system reliability. 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 set of easily comprehend input factors are required. Data entry, data update and data retrieval can be performed in a short period of time.

In most of the reliability estimation several things are needed to find out the exact reliability of products. They are listed as follows:

(a) Collecting failure and test time data.

(b) Calculating estimates of model parameters using the data.

(c) Testing the fit of a model against the collected data.

(d) Applying the model for future predictions.

It is the commonality of these and other features, which leads to the development of special purpose reliability measurement tools. The model implemented under hardware reliability were:

I. Exponential.

II. Normal.

III. Log-Normal.

IV. Weibull.

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 were incorporated in the software to aid user in deciding which is the best model for that particular data set. The software tool was applied to various data and the results like model parameters, reliability and failure intensity values, statistical test values and tiesets and cutsets probabilities were verified with hand calculations and from published literature.

6.2 Future Work.

The software developed was able to perform calculations for hardware reliability, software reliability and system reliability. However as the respective areas of application being very vast, the software has following limitations.

1. For hardware reliability the software assumes complete data. For other kinds of data set like singly censored data, multiple censored data etc., reliability cannot be estimated.
2. For system reliability the software reliability does not calculate tieset and cutset for given block diagram of system.
3. For system reliability the matrix size is limited to 100 by 100 thereby allowing software to estimate reliabilities of system having upto 100 tiesets or cutsets.

The capability of this software can be improved by interfacing it with other software and databases. The software can be extended in a number of areas, some of which are as follows:

- Reliability estimation by using other distributions and models.
- Provision for additional statistical analysis for the data.
- An algorithm which determines tiesets and cutsets for the system.
- Models for of maintainability and availability of the products/systems.

Appendix A: Bartlett's Test for Exponential Distribution.

A specific test for fitting exponential distribution is Bartlett's test. The hypotheses are

H0: Failure times are exponential

H1: Failure times are not exponential.

The test statistic is

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

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 B: Kolmogorov-Smirnov Test

A goodness of fit test for Normal and Log normal distribution is Kolgomorov-Smirnov test. The hypotheses for the test is ;

H0: The failure times are normal (lognormal)

H1: the failure times are not normal (lognormal)

The test statistic is

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

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

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

If $D_n < D_{crit}$, then accept H0 otherwise accept H1. The values of D_{crit} may be found in the standard Kolgomorov-Smirnov table.

Appendix C: Mann's Test for Weibull Distribution.

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

H0: The failure times of weibull.

H1: The failure times are not weibull.

The test statistic is:

$$M = \frac{k1 \cdot \sum_{i=k1+1}^{n-1} \left(\ln(t_{i+1}) - \ln(t_i)\right) / M_i}{k2 \cdot \sum_{i=1}^{k1} \left(\ln(t_{i+1}) - \ln(t_i)\right) / M_i}$$

where k1 = | n/2 | , k2 | (n-1)/2|,

Mi =Zi+1 –Zi

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

And |x| is the integer portion for number *x*. M_i is an approximation. If M > then F_{crit}, then H1 is accepted. Values for F_{crit}, may be obtained from tables of the F-distribution if one lets the number of degrees of freedom for numerator to be 2k2 and the number of degrees of freedom for the denominator be 2k1.

Appendix D: Newton Raphson Method for Finding Roots

As a numerical root finding procedure for systems of nonlinear simultaneous equations the Newton-Raphson procedure can be used. This is a widely used technique for solving equations where a direct algebraic solution in not possible. If a root for the equation $f(x)=0$ is sought, the formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n=0, 1, \ldots \qquad \text{(C.1)}$$

can be used to improve the first approximation x_0 to root to whatever degree of accuracy that is required.

More generally, we want to solve multiple equations for roots β_k ($k=0, 1\ldots w$), i.e., we have the equation $\mathbf{U}(\boldsymbol{\beta})=0$, where $\mathbf{U}(\boldsymbol{\beta})$ is a $(w+1)\times 1$ column vector with elements $\mathbf{U}_k(\boldsymbol{\beta})=f_k(\boldsymbol{\beta})$, $k=0, 1, \ldots, w$. Then we have the formula

$$\boldsymbol{\beta}' = \boldsymbol{\beta} - \mathbf{H}^{-1}(\boldsymbol{\beta}) \times \mathbf{U}(\boldsymbol{\beta}) \qquad \text{(C.2)}$$

where $\boldsymbol{\beta}'$ is closer to root than the previous approximation $\boldsymbol{\beta}$. The matrix $\mathbf{H}(\boldsymbol{\beta})$, also called the *Hessian matrix*, is a $(w+1)\times(w+1)$ square matrix with elements

$$\mathbf{H}_{kl}(\boldsymbol{\beta}) = \frac{\partial f_k(\boldsymbol{\beta})}{\partial \beta_l} = f_{kl}(\boldsymbol{\beta}), \quad k, l=0, 1, \ldots, w. \qquad \text{(C.3)}$$

The above "matrix" version of the original Newton-Raphson formula is applied repeatedly by replacing $\boldsymbol{\beta}$ with the newly found vector $\boldsymbol{\beta}'$, until successive estimates agree to a specified tolerance on an element by element basis.

If $w=0$, then $\boldsymbol{\beta}$ is a single-element vector and we obtain the original formula with x_n replaced by β_0, x_{n+1} by β_0', $f(x_n)$ by $f_0(\beta_0)$, and $f'(x_n)$ by $f_{00}(\beta_0)=\frac{\partial f_0(\beta_0)}{\partial \beta_0}$.

If $w=1$, then we have

$$\begin{bmatrix}\hat{\beta}_0\\ \hat{\beta}_1\end{bmatrix}=\begin{bmatrix}\beta_0\\ \beta_1\end{bmatrix}-\begin{bmatrix}f_{00} & f_{01}\\ f_{10} & f_{11}\end{bmatrix}^{-1}\times\begin{bmatrix}f_0\\ f_1\end{bmatrix}=\begin{bmatrix}\beta_0\\ \beta_1\end{bmatrix}-\frac{\begin{bmatrix}f_{11} & -f_{01}\\ -f_{10} & f_{00}\end{bmatrix}\times\begin{bmatrix}f_0\\ f_1\end{bmatrix}}{\begin{vmatrix}f_{00} & f_{01}\\ f_{10} & f_{11}\end{vmatrix}}$$

$$=\begin{bmatrix}\beta_0\\ \beta_1\end{bmatrix}-\frac{1}{\begin{vmatrix}f_{00} & f_{01}\\ f_{10} & f_{11}\end{vmatrix}}\begin{bmatrix}\begin{vmatrix}f_0 & f_{01}\\ f_1 & f_{11}\end{vmatrix}\\ \\ \begin{vmatrix}f_{00} & f_0\\ f_{10} & f_1\end{vmatrix}\end{bmatrix} \qquad \text{(C.4)}$$

where $f_{kl}(\boldsymbol{\beta})$ is shown as f_{kl} for simplicity.

In most cases w does not exceed 1 and in some cases it is 0, so we need not bother ourselves with more complex situations.

In reliability estimation we have a function to be maximized. For example, in maximum likelihood estimation we have the likelihood function $L(\boldsymbol{\beta})$. To maximize it, its derivatives are set to zero, i.e.,

$$\frac{\partial \ln L(\boldsymbol{\beta})}{\partial \beta_k}=0, \quad k=0, 1, \ldots, w. \qquad \text{(C.5)}$$

So the elements of vector $\mathbf{U}(\boldsymbol{\beta})$ are

$$\mathbf{U}_k(\boldsymbol{\beta})=f_k(\boldsymbol{\beta})=\frac{\partial \ln L(\boldsymbol{\beta})}{\partial \beta_k}=0, \quad k=0, 1, \ldots, w, \qquad \text{(C.6)}$$

and of matrix $\mathbf{H}(\boldsymbol{\beta})$ are

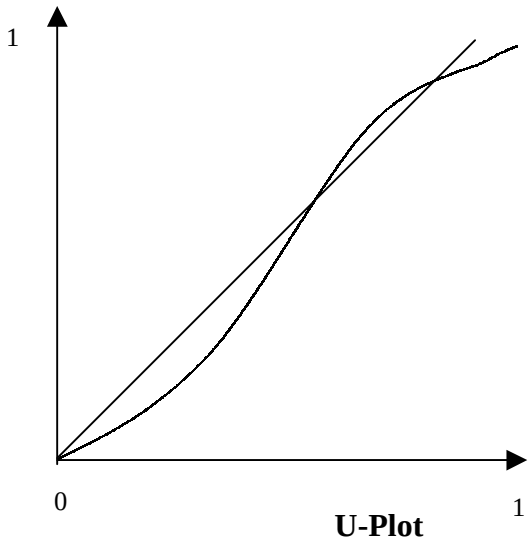
$$\mathbf{H}_{kl}(\boldsymbol{\beta})=f_{kl}(\boldsymbol{\beta})=\frac{\partial^2 \ln L(\boldsymbol{\beta})}{\partial \beta_k\, \partial \beta_l}, \quad k, l=0, 1, \ldots, w. \qquad \text{(C.7)}$$

The Newton-Raphson procedure converges rapidly if it does, which generally happens if the initial estimate is close enough to $\hat{\boldsymbol{\beta}}$. If the initial estimate is poor it may diverge. Also, the evaluation of the Hessian matrix, and its inversion, may pose formidable computational problems.

Appendix E:U-plot

The purpose of U-plot is to determine weather predictions from the distribution are close to the true distributions. It can be shown that if random variable T_j truly had the distribution $\hat{F}_j(t)$ the random variable $U_j = \hat{F}_j(t)$ will be uniformly distributed on (0,1). Any departure from such uniformity will indicate some kind of deviation between $\hat{F}_j(t)$ and truth $F_j(t)$

One way of looking for departure is by plotting the sample distribution function of the uj sequence. This is a step function constructed as follows: for a sequence of predictions, $F_j(t)$,j = s,…i on the interval (0,1), place the points u_s u_{s+1}…u_i; then from left to right plot an increasing step function, with each step of height 1/(i-s+2) at each u on the abscissa. The range of the resulting monotonically increasing function is (0,1) and is known as U-plot.



U-Plot

Appendix F: Code of the Software Developed

Editor Form

```
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.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
```

Reliability Calculator Form

```
Option Explicit
Private Sub cmdclear_Click()
txtmissiontime.Text = ""
txtreliability.Text = ""
End Sub
Private Sub cmdexponential_Click()
Dim frate As Single
mtime = txtmissiontime.Text
frate = Val(txtfailurerate.Text)
If txtmissiontime.Text = "" Then
MsgBox " Enter the Mission Time"
Exit Sub
End If
If txtfailurerate.Text = "" Then
MsgBox "Enter the parameters for Exponential distribution", vbExclamation
Exit Sub
End If
hrel = (Exp(-frate * mtime))
txtreliability.Text = hrel
```

```
End Sub
Private Sub cmdgamma_Click()
mtime = txtmissiontime.Text
If txtmissiontime.Text = "" Then
MsgBox " Enter the Mission Time"
Else
End If
Exit Sub
txtreliability.Text = hrel
End Sub
Private Sub cmdlognormal_Click()
Dim pi As Single
Dim X As Double, z As Double
Dim h As Single
Dim sum As Double
Dim lnshape As Double, lnloc As Double
mtime = txtmissiontime.Text
lnshape = Val(txtshapeparameter.Text)
lnloc = Val(txtlocation.Text)
If txtmissiontime.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
txtreliability.Text = hrel
End Sub
Private Sub cmdnormal_Click()
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
mtime = txtmissiontime.Text
nmean = Val(txtnormalmean.Text)
nstddev = Val(txtstandarddev.Text)
If txtmissiontime.Text = "" Then
MsgBox " Enter the Mission Time"
Else
Exit Sub
End If
If txtnormalmean.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
```

```
txtreliability.Text = hrel
End Sub

Private Sub cmdweibull_Click()
Dim wscale As Single
Dim wshape As Single
mtime = txtmissiontime.Text
wshape = Val(txtwshape.Text)
wscale = Val(txtwscale.Text)
If txtmissiontime.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 - wshape) / wscale) ^ wshape))
txtreliability.Text = hrel
End Sub
```

Menu Editor Form

```
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", _
End Sub
Private Sub mnucdopen_Click()
mciSendString "Set CDAudio Door Open Wait", _
End Sub
Private Sub mnuchi_Click()
frmchisquare.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 mnuhardware_Click()
frmhardware1.Show
End Sub
Private Sub mnuks_Click()
frmks.Show
End Sub
Private Sub mnunewfile_Click()
editorform.Show
End Sub
Private Sub mnuphardware_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 mnupresults_Click()
On Error GoTo 3
frmresult.PrintForm
MsgBox "Done !", 64, vbExclamation
Exit Sub
3
MsgBox "There was problem printing to yout printer", vbExclamation
End Sub
Private Sub mnupsoftware_Click()
On Error GoTo 2
frmsoftware.PrintForm
MsgBox "Done !", 64, vbExclamation
Exit Sub
2
MsgBox "There was problem printing to yout printer", vbExclamation
End Sub
Private Sub mnurcal_Click()
frmhardware.Show
End Sub
Private Sub mnusoftware_Click()
frmsoftware.Show
End Sub
Private Sub mnusystem_Click()
frmsystem.Show
End Sub
Private Sub mnutile_Click()
frmret.Arrange vbTileVertical
End Sub
```

Hardware Reliability Form:

```
Option Explicit
Private Sub cmdchisquare_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 optexpdist = 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(txtnumberoffailuresh1.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
   If x(1) < barlett And barlett < x(2) Then
 lbldes.Caption = "As calculated Bartlett's test statistic," + Str(Format(barlett,
"#.000")) + ", is falls between the crritical 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(barlett, "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
End If
If optweibulldist = True Then
    Dim n1 As Double, n2 As Double, k1 As Double, k2 As Double, f As Double
    Dim ff As Double
    n = Val(txtnumberoffailuresh1.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 optlognormaldist = True Or optnormaldist = True Then
    Dim ks2 As Double
    n = Val(txtnumberoffailuresh1.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 optlognormaldist = True Then
        If k_s < ks2 Then
        lbldes.Caption = "As calculated Komogorov -Smirnov value," +
Str(Format(k_s, "0.000")) + ", is less than the critical Kolmogorov-Smirnov value,"
+ Str(Format(ks2, "#.000")) + ", hypothesis that data belongs to LogNormal
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_s, "#.000")) + ", hypothesis that data belongs to LogNormal
distribution is  not accepted"
        End If
    End If

    If optnormaldist = True Then
        If k_s < ks2 Then
        lbldes.Caption = "As calculated Komogorov -Smirnov value," +
Str(Format(k_s, "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_s, "#.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 cmdclear_Click()
Dim k As Integer
For k = 1 To 100
grddatah1.Col = 0
grddatah1.Row = k
grddatah1.Text = ""
Next k
For k = 1 To 100
grddatah1.Col = 1
grddatah1.Row = k
grddatah1.Text = ""
Next k
txtnumberoffailuresh1.Text = ""
txtsumtotalh1.Text = ""
txtfilename1.Text = ""
optlognormaldist.Value = False
optnormaldist.Value = False
optexpdist.Value = False
optweibulldist.Value = False
txtmtimeh1.Text = ""
txthrel1.Text = ""
txtalpha.Text = ""
lbldes.Caption = ""
Text1.Visible = False
Text2.Visible = False
Text3.Visible = False
```

```
Text4.Visible = False
Text5.Visible = False
txtalpha.Visible = False
lbldes.Visible = False
cmdchisquare.Visible = False
End Sub
Private Sub cmdhrel1_Click()
'================================================
'ERROR CHECK
If txtfilename1.Text = "" Then
MsgBox " Select a failure data file!", vbExclamation
Exit Sub
End If

grddatah1.Col = 1
grddatah1.Row = 1
If grddatah1.Text = "" Then
MsgBox " Open the selected file", vbExclamation
Exit Sub
End If

If txtmtimeh1.Text = "" Then
MsgBox " Enter the mission time ", vbExclamation
Exit Sub
End If

If IsNumeric(txtmtimeh1.Text) Then
mtime = Val(txtmtimeh1.Text)
If mtime <= 0 Then
MsgBox "Invalid input", vbExclamation
Exit Sub
End If
Else
MsgBox "Invalid Input", vbExclamation
Exit Sub
End If

If optexpdist.Value = False And optnormaldist.Value = False And
optlognormaldist.Value = False And optweibulldist.Value = False Then
MsgBox " Select a Distribution", vbExclamation
Exit Sub
End If

'EXPONENTIAL DISTRIBUTION
'************************
If optexpdist.Value = True Then
Dim lambda As Double, sumb1 As Double, sumb2 As Double
lambda = Val(txtnumberoffailuresh1.Text) / Val(txtsumtotalh1.Text)
mttf = 1 / lambda
hrel = (Exp(-lambda * mtime))
If hrel < 0 Then
txthrel1.Text = 0
Else
If hrel > 1 Then
txthrel1.Text = 1
Else
txthrel1.Text = hrel
End If
End If
n = Val(txtnumberoffailuresh1.Text)
sumb1 = 0
sumb2 = 0
For i = 1 To n
grddatah1.Col = 1
grddatah1.Row = i
sumb1 = sumb1 + grddatah1.Text
sumb2 = sumb2 + Log(grddatah1.Text)
Next i
barlett = 2 * n * ((Log((1 / n) * (sumb1))) - ((1 / n) * (sumb2))) / (1 + ((n + 1)
/ (n * 6)))
Text1.Text = "Distribution selected: Exponential"
Text2.Text = "Mean Time to Failure:" + Str(Format(mttf, "0.000"))
Text3.Text = "Lambda:" + Str(Format(lambda, "#.000000"))
Text4.Text = " Bartletts Test Statistic:" + Str(Format(barlett, "#.000"))
Text5.Text = " Bartletts test value is then compared with the Chisquared value
(also known as critical value) with degrees of freedom" + Str(n - 1) + " 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"
cmdchisquare.Caption = "Chi- square"
End If

'NORMAL DISTRIBUTION
'******************
If optnormaldist.Value = True Then
Dim mean As Single, evariance As Single, svariance As Single, tbar As Single
Dim evar As Single, hfailure(200) As Single
Dim x As Single, z As Double, kssum(200) As Single, l1 As Integer, index1 As
Integer
Dim h As Single, D1(200) As Single, D2(200) As Single, min1 As Single
Dim nstddev As Double, newd1(200) As Single, newd2(200) As Single
n = Val(txtnumberoffailuresh1.Text)
mean = Val(txtsumtotalh1.Text) / n
mttf = Val(txtsumtotalh1.Text) / n
tbar = 0
evar = 0
For i = 1 To n
grddatah1.Col = 1
grddatah1.Row = i
evar = evar + ((grddatah1.Text) ^ 2)
tbar = tbar + grddatah1.Text / n
Next i
evariance = 0
For j = 1 To n
evariance = evariance + ((grddatah1.Text - tbar) ^ 2)
Next j
svariance = Sqr(evariance / n - 1)
nstddev = Sqr(((n * evar) - ((sum) ^ 2)) / ((n) * (n - 1)))
pi = 3.14
x = -10
mtime = Val(txtmtimeh1.Text)
z = (mtime - mean) / nstddev
h = 0.005
sumh = 0
Do While x < z
sumh = sumh + (((1 / Sqr(2 * pi)) * Exp((-(x ^ 2) / 2))) * h)
x = x + h
Loop
hrel = 1 - sumh
If hrel < 0 Then
txthrel1.Text = 0
Else
txthrel1.Text = hrel
End If
min1 = 100000                          ' sorts the failure data
For m = 1 To n
For l1 = 1 To n
If failure(l1) < min1 Then
min1 = failure(l1)
index1 = l1
End If
Next l1
hfailure(m) = min1
failure(index1) = 100000
min1 = 100000
Next m
For i = 1 To n
x = -4
z = (hfailure(i) - tbar) / (nstddev)
h = 0.005
kssum(i) = 0
Do While x < z
kssum(i) = kssum(i) + (((1 / Sqr(2 * pi)) * 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 m = 1 To n
For l1 = 1 To n
If D1(l1) < min1 Then
```

```
min1 = D1(l1)
index1 = l1
End If
Next l1
newd1(m) = min1
D1(index1) = 100000
min1 = 100000
Next m
min1 = 100000                          ' sorts the failure data
For m = 1 To n
For l1 = 1 To n
If D2(l1) < min1 Then
min1 = D2(l1)
index1 = l1
End If
Next l1
newd2(m) = min1
D2(index1) = 100000
min1 = 100000
Next m
If newd1(n) < newd2(n) Then
k_s = newd2(n)
Else
k_s = newd1(n)
End If
Text1.Text = " Distribution selected: Normal"
Text2.Text = " Mean:" + Str(Format(mean, "0.000"))
Text3.Text = " Standarad Deviation:" + Str(Format(nstddev, "#.000000"))
Text4.Text = " Kolgomorov-Smirnov Test Stat:" + Str(Format(k_s, "#.0000"))
Text5.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"
cmdchisquare.Caption = "KS - Value"
End If
'LOG NORMAL DISTRIBUTION
'**********************
If optlognormaldist.Value = True Then
Dim lnmean As Single, lnloc As Single, lnsum As Single
Dim lnshape As Single, newsum As Single
Dim q As Integer
Dim lnvar As Double
n = Val(txtnumberoffailuresh1.Text)
lnsum = 0
For i = 1 To n
grddatah1.Col = 1
grddatah1.Row = i
lnsum = lnsum + Log(grddatah1.Text)
Next i
lnmean = lnsum / n
lnloc = Exp(lnmean)
newsum = 0
For q = 1 To n
grddatah1.Col = 1
grddatah1.Row = q
newsum = newsum + ((Log(grddatah1.Text) - lnmean) ^ 2)
Next q
lnshape = Sqr(newsum / n)
pi = 3.14
x = -20
z = (1 / lnshape) * Log(mtime / lnloc)
h = 0.005
sumh = 0
Do While x < z
sumh = sumh + ((1 / (Sqr(2 * pi))) * (Exp((-x ^ 2 / 2)))) * h
x = x + h
Loop
hrel = 1 - sumh
lnvar = (lnloc ^ 2) * (Exp(lnshape ^ 2)) * ((Exp(lnshape ^ 2)) - 1)
If hrel < 0 Then
txthrel1.Text = 0
Else
txthrel1.Text = hrel
End If
min1 = 100000                          ' sorts the failure data
```

```
For m = 1 To n
For l1 = 1 To n
If failure(l1) < min1 Then
min1 = failure(l1)
index1 = l1
End If
Next l1
hfailure(m) = min1
failure(index1) = 100000
min1 = 100000
Next m
For i = 1 To n
x = -4
z = (1 / lnshape) * Log(hfailure(i) / lnloc)
h = 0.005
kssum(i) = 0
Do While x < z
kssum(i) = kssum(i) + (((1 / Sqr(2 * pi)) * 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 m = 1 To n
For l1 = 1 To n
If D1(l1) < min1 Then
min1 = D1(l1)
index1 = l1
End If
Next l1
newd1(m) = min1
D1(index1) = 100000
min1 = 100000
Next m
min1 = 100000                          ' sorts the failure data
For m = 1 To n
For l1 = 1 To n
If D2(l1) < min1 Then
min1 = D2(l1)
index1 = l1
End If
Next l1
newd2(m) = min1
D2(index1) = 100000
min1 = 100000
Next m
If newd1(n) < newd2(n) Then
k_s = newd2(n)
Else
k_s = newd1(n)
End If
Text1.Text = " Distribution selected: LogNormal"
Text2.Text = " Shape Parameter:" + Str(Format(lnshape, "0.000"))
Text3.Text = " Location Parameter:" + Str(Format(lnloc, "#.000000"))
Text4.Text = " Kolgomorov-Smirnov Test Stat:" + Str(Format(k_s, "#.0000"))
Text5.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 Log-Normal distribution"
cmdchisquare.Caption = "KS- Value"
End If

'WEIBULL DISTRIBUTION
'--------------------
If optweibulldist.Value = True Then

Dim l As Integer, p As Integer, r As Integer, t As Integer
Dim a(200) As Single, sdata(200) As Single, index As Integer
Dim min2 As Single, data(200) As Single, xj(200) As Double
Dim xbar As Double, yj(200) As Double, ybar As Double, sumxj As Double
Dim sumyj As Single, fj(200) As Single, b As Double, c As Double
Dim b1 As Double, B2 As Double, wscale As Double, D As Double
Dim k1 As Double, k2 As Double, den1 As Double
Dim m1(100) As Double, z1(100) As Double, i1 As Double, num1 As Double
```

```
Dim i2 As Integer, i3 As Integer, num As Double, den As Double, wshape As Double
n = Val(txtnumberoffailuresh1.Text)
For p = 1 To n
grddatah1.Col = 1
grddatah1.Row = p
data(p) = grddatah1.Text
Next p
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
sumxj = 0
sumyj = 0
For r = 1 To n
xj(r) = Log(sdata(r))
sumxj = sumxj + xj(r)
fj(r) = (r - 0.3) / (n + 0.4)
yj(r) = (Log(Log(1 / (1 - fj(r)))))
sumyj = sumyj + yj(r)
Next r
xbar = sumxj / n
ybar = sumyj / n
b1 = 0
B2 = 0
For t = 1 To n
b1 = b1 + ((xj(t) - (xbar)) * (yj(t) - (ybar)))
B2 = B2 + ((xj(t) - (xbar)) ^ 2)
Next t
b = b1 / B2
c = (ybar) - (b * xbar)
wshape = b
wscale = Exp(-c / b)
    hrel = Exp(-((mtime / wscale) ^ b))
    If hrel < 0 Then
    txthrel1.Text = 0
    Else
    End If
    If hrel > 1 Then
    txthrel1.Text = 1
    Else
    txthrel1.Text = hrel
    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)
 num = 0
 For i2 = k1 To n - 1
 num = num + (((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 = num * k1
 den1 = den * k2
 mann = num1 / den1
Text1.Text = " Distribution selected: Weibull"
Text2.Text = " Shape Parameter:" + Str(Format(wshape, "0.000"))
Text3.Text = " Scale Parameter:" + Str(Format(wscale, "0.000000"))
Text4.Text = " Mann's Test Statistic:" + Str(Format(mann, "0.000"))
Text5.Text = " Manns test value is then compared with the F-distribution value(also
known as critical value)from the standard table F-distribution table; with degrees
of freedom" + Str(2 * k1) + " and " + Str(2 * k2) + ";and level of significance
```

```
alpha (user decides this value). If the calculated Mann's test statistic is less
than the critical value than the data is good fit for Weibull distribution"
cmdchisquare.Caption = "F - Value"
End If
Text1.Visible = True
Text2.Visible = True
Text3.Visible = True
Text4.Visible = True
Text5.Visible = True
cmdchisquare.Visible = True
txtalpha.Visible = True
lblalpha.Visible = True
End Sub
Private Sub cmdopenfileh1_Click()
Dim sumh As Single
If txtfilename1.Text = "" Then
MsgBox "You must Select a Data File!"
Exit Sub
End If
pathname = dirdirectoryh1.path
If Right(dirdirectoryh1.path, 1) <> "\" Then
path = dirdirectoryh1 + "\"
Inp1 = path + txtfilename1.Text
Else
Inp1 = pathname + txtfilename1.Text
End If
Open Inp1 For Input As #2
i = 0
Do While Not EOF(2)
i = i + 1
Input #2, failure(i)
grddatah1.Col = 0
grddatah1.Row = i
grddatah1.Text = i
grddatah1.Col = 1
grddatah1.Row = i
grddatah1.Text = Val(failure(i))
Loop
txtnumberoffailuresh1.Text = i
Close #2
sum = 0
For j = 1 To Val(txtnumberoffailuresh1.Text)
grddatah1.Col = 1
grddatah1.Row = j
sum = sum + grddatah1.Text
Next j
txtsumtotalh1.Text = sum
End Sub
Private Sub dirdirectoryh1_Change()
'Here we change the files according to the directory
 filfilesh1.path = dirdirectoryh1.path
End Sub
Private Sub drvdriveh1_Change()
'The next statement checks the error in the path
On Error GoTo driveerror
'change the path of the new drive
dirdirectoryh1.path = drvdriveh1.Drive
Exit Sub
driveerror:
' Here we restore the original drive
MsgBox "Device is not ready!", vbCritical, "Error"
drvdriveh1.Drive = dirdirectoryh1.path
End Sub
Private Sub filfilesh1_Click()
txtfilename1.Text = filfilesh1.FileName
End Sub
Private Sub Form_Load()
frmret.mnuphardware.Enabled = True
Text1.Visible = False
Text2.Visible = False
Text3.Visible = False
Text4.Visible = False
Text5.Visible = False
End Sub
```

Software Reliability Form

```
'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 = l
frmresult.grdresult.Text = Format(eval1, "0.00000")
eval(l) = eval1
Next l
End If
' GAMMA MODEL
'============
If optgamma.Value = True Then
For l = 1 To TNF
grddata.Col = 1
grddata.Row = l
fintensity = (b1new ^ 2) * b0 * (Exp(-b1new * grddata.Text)) * grddata.Text
srel = (Exp(-(srtime) * (fintensity)))
SSE = SSE + (((l) - ((b0) * ((1) - ((1 + b1new * grddata.Text) * (Exp(-
b1new * grddata.Text)))))) ^ 2)
CDF1 = ((1) - ((1 + b1new * grddata.Text) * (Exp(-b1new * grddata.Text))))
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
'GEOMETRIC MODEL
'===============
If optlp.Value = True Then
For l = 1 To TNF
grddata.Col = 1
grddata.Row = l
fintensity = (b0 * b1new) / (1 + (b1new * grddata.Text))
srel = (Exp(-(srtime) * (fintensity)))
SSE = SSE + (((l) - (b0 * (Log(1 + b1new * grddata.Text)))) ^ 2)
CDF1 = (Log(1 + b1new * grddata.Text) / Log(1 + b1new * failure(TNF)))
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
'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 l = 1 To TNF
grddata.Col = 1
grddata.Row = l
fintensity = 2 * b0 * b1new * grddata.Text * (Exp(-b1new * grddata.Text *
grddata.Text))
srel = (Exp(-(srtime) * (fintensity)))
SSE = SSE + (((l) - ((b0) * ((1 - Exp(-b1new * ((grddata.Text) ^ 2)))))) ^
2)
CDF1 = ((1 - Exp(-b1new * ((grddata.Text) ^ 2))))
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) = Str(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
'INVERSE LINEAR MODEL
'====================
If optinverse.Value = True Then
For l = 1 To TNF
grddata.Col = 1
```

```
grddata.Row = l
fintensity = b0 * (1 / (2 * (b1new + grddata.Text) ^ (1 / 2)))
srel = (Exp(-(srtime) * (fintensity)))
SSE = SSE + (((l) - ((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 = 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
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
```

Result Form:

```
Option Explicit
```

```
Private Sub cmdgraph_Click()
Dim a As Integer, b As Integer, c As Integer
Dim introw1 As Integer, reldata(400) As Double, sreldata(400) As Double
Dim introw2 As Integer
For p = 1 To TNF
grdresult.Col = 2
grdresult.Row = p
fidata(p) = grdresult.Text
Next p
min = 1                          ' sorts the failure intensity data
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
MSChart.Plot.UniformAxis = False
With MSChart.Plot.Axis(VtChAxisIdX)
.AxisScale.Type = VtChScaleTypeLinear
.AxisGrid.MinorPen.Style = VtPenStyleNull
.CategoryScale.Auto = False
.ValueScale.Maximum = Val(FD(TNF))
.ValueScale.Minimum = Val(FD(1))
.AxisTitle = "Failure Data"
End With
With MSChart.Plot.Axis(VtChAxisIdY)
.AxisScale.Type = VtChScaleTypeLinear
.AxisGrid.MinorPen.Style = VtPenStyleNull
.CategoryScale.Auto = False
.ValueScale.Maximum = (sfidata(TNF))
.ValueScale.Minimum = (sfidata(1))
.AxisTitle = "Failure Intensity"
End With
MSChart.chartType = VtChChartType2dXY
MSChart.ColumnCount = 2
MSChart.RowCount = TNF
For introw = 1 To TNF
MSChart.Row = introw
MSChart.Column = 1
MSChart.data = values(1, introw)
Next introw
For introw = 1 To TNF
MSChart.Column = 2
MSChart.Row = introw
MSChart.data = values(2, introw)
Next introw

' RELIABILITY GROWTH CURVE
For j = 1 To TNF
grdresult.Col = 3
grdresult.Row = j
reldata(j) = grdresult.Text
Next j
min = 100000                   ' sorts the reliability data
For a = 1 To TNF
For b = 1 To TNF
If reldata(b) < min Then
min = reldata(b)
index = b
End If
Next b
sreldata(a) = min
fidata(index) = 10000000
min = 10000000
Next a
    For a = 1 To TNF
    values(1, a) = FD(a)
```

```
    values(3, a) = r(a)
    Next a
    MSChart1.Plot.UniformAxis = False
    With MSChart1.Plot.Axis(VtChAxisIdX)
    .AxisScale.Type = VtChScaleTypeLinear
    .AxisGrid.MinorPen.Style = VtPenStyleNull
    .CategoryScale.Auto = False
    .ValueScale.Maximum = Val(FD(TNF))
    .ValueScale.Minimum = Val(FD(1))
    .AxisTitle = "Failure Data"
    End With
    With MSChart1.Plot.Axis(VtChAxisIdY)
    .AxisScale.Type = VtChScaleTypeLinear
    .AxisGrid.MinorPen.Style = VtPenStyleNull
    .CategoryScale.Auto = False
    .ValueScale.Maximum = 1
    .ValueScale.Minimum = 0
    .AxisTitle = "Reliability"
    End With
    MSChart1.chartType = VtChChartType2dXY
    MSChart1.ColumnCount = 2
    MSChart1.RowCount = TNF
    For introw1 = 1 To TNF
    MSChart1.Row = introw1
    MSChart1.Column = 1
    MSChart1.data = values(1, introw1)
    Next introw1
    For introw1 = 1 To TNF
    MSChart1.Column = 2
    MSChart1.Row = introw1
    MSChart1.data = values(3, introw1)
    Next introw1
'-------------------------------------------------------------
    For c = 1 To TNF
    values(6, c) = eval(c)
    values(4, c) = CDF(c)
    values(7, c) = c / TNF
    Next c
  MSChart3.Plot.UniformAxis = False
    With MSChart3.Plot.Axis(VtChAxisIdX)
    .AxisScale.Type = VtChScaleTypeLinear
    .AxisGrid.MinorPen.Style = VtPenStyleNull
    .CategoryScale.Auto = False
    .ValueScale.Maximum = 1
    .ValueScale.Minimum = 0
    End With
    With MSChart3.Plot.Axis(VtChAxisIdY)
    .AxisScale.Type = VtChScaleTypeLinear
    .AxisGrid.MinorPen.Style = VtPenStyleNull
    .CategoryScale.Auto = False
    .ValueScale.Maximum = 1
    .ValueScale.Minimum = 0
     End With
    MSChart3.chartType = VtChChartType2dXY
    MSChart3.ColumnCount = 4
    MSChart3.RowCount = TNF
    For introw2 = 1 To TNF
    MSChart3.Row = introw2
    MSChart3.Column = 1
    MSChart3.data = values(4, introw2)
    Next introw2
    For introw2 = 1 To TNF
    MSChart3.Column = 2
    MSChart3.Row = introw2
    MSChart3.data = values(6, introw2)
    Next introw2
    For introw2 = 1 To TNF
    MSChart3.Row = introw2
    MSChart3.Column = 3
    MSChart3.data = values(7, introw2)
    Next introw2
    For introw2 = 1 To TNF
    MSChart3.Row = introw2
    MSChart3.Column = 4
    MSChart3.data = values(6, introw2)
    Next introw2
```

```
    Label3.Caption = "Reliability of the software for next" + Str(srtime) + " unit
time with failure intensity = " +   Str(Format(FI(TNF), "0.00000")) + " is" +
Str(Format(r(TNF), "0.0000"))
    Label4.Caption = "Sum of Square of Errors is" + Str(SSE)
   MSChart.Visible = True
    MSChart1.Visible = True
    MSChart3.Visible = True
   Label1.Visible = True
    Label2.Visible = True
    Label3.Visible = True
    Label4.Visible = True
    Label5.Visible = True
End Sub
Private Sub Form_Load()
frmret.mnupresults.Enabled = True
MSChart.Visible = False
MSChart1.Visible = False
Label4.Visible = False
For i = 1 To TNF
grdresult.Col = 7
grdresult.Row = i
j = i / TNF
grdresult.Text = "j"
Next i
End Sub
```

System Reliability

```
Option Explicit
Dim k As Integer, l As Integer, m As Integer, n As Integer
Dim c As Integer, TT As Integer, CC As Integer
Dim Tset(100, 100) As Integer, Cset(100, 100) As Integer, Crel(100) As Single
Dim R1 As Single, R2 As Single, R3 As Single, R4 As Single, R5 As Single
Dim R6 As Single, R7 As Single, R8 As Single, R9 As Single, R10 As Single
Dim p As Single
Dim RU1 As Single, RU2 As Single, RU3 As Single, RU4 As Single
Dim RL1 As Single, RL2 As Single, RL3 As Single, RL4 As Single
Private Sub cmdcls1_Click()
Dim i As Integer, j As Integer
For i = 0 To 1
For j = 0 To 99
grdcrel.Col = i
grdcrel.Row = j
grdcrel.Text = ""
Next j
Next i
For i = 0 To 29
For j = 0 To 99
grdcutset.Col = i
grdcutset.Row = j
grdcutset.Text = ""
Next j
Next i
For i = 0 To 29
For j = 0 To 59
grdtieset.Col = i
grdtieset.Row = j
grdtieset.Text = ""
Next j
Next i
optd1 = False
optd2 = False
optd3 = False
optd4 = False
txtcutset.Text = ""
txtcutsetfile.Text = ""
txtnumcomp.Text = ""
txttieset.Text = ""
txttiesetfile.Text = ""
txtcrel.Text = ""
lblfinal.Caption = ""
End Sub
Private Sub cmdcrel_Click()
If txtnumcomp.Text = "" Then
MsgBox "Enter number of components for the system ", vbCritical
```

```
Exit Sub
End If
c = Val(txtnumcomp.Text)
If txtcrel.Text = "" Then
MsgBox "You must Select a Componenet Reliability Matrix File!"
Exit Sub
End If
pathname = Dirdirectorycrel.path
If Right(Dirdirectorycrel.path, 1) <> "\" Then
path = Dirdirectorycrel + "\"
Inp = path + txtcrel.Text
Else
Inp = pathname + txtcrel.Text
End If
Open Inp For Input As #7
For i = 1 To c
grdcrel.Col = 0
grdcrel.Row = i
grdcrel.Text = "Component" + Str(i)
Next i
grdcrel.Row = 0
grdcrel.Col = 1
grdcrel.Text = "Reliability"
For i = 1 To c
Input #7, Crel(i)
grdcrel.Col = 1
grdcrel.Row = i
grdcrel.Text = Val(Crel(i))
Next i
Close #7
End Sub
Private Sub cmdsysrel_Click()
If txtcrel.Text = "" Then
MsgBox "You must Select a Componenet Reliability Matrix File!"
Exit Sub
End If
If txtcutsetfile.Text = "" Then
MsgBox "You must Select a Cutset Matrix File!"
Exit Sub
End If
If txttiesetfile.Text = "" Then
MsgBox "You must Select a Tieset Matrix File!"
Exit Sub
End If
If txtnumcomp.Text = "" Or txtcutset.Text = "" Or txttieset.Text = "" Then
MsgBox "Enter number of components for the system and/or number of cutset/tieset
for the system", vbCritical
Exit Sub
End If
If optd1 = False And optd2 = False And optd3 = False And optd4 = False Then
MsgBox "Select presicion for reliability"
Exit Sub
End If
R1 = 0
For i = 1 To TT
p = 1
For j = 1 To c
If Tset(i, j) = 1 Then
p = p * Crel(j)
End If
Next j
R1 = R1 + p
Next i
R2 = 0
For i = 1 To TT - 1
For k = i + 1 To TT
p = 1
For j = 1 To c
If Tset(i, j) = 1 Or Tset(k, j) = 1 Then
p = p * Crel(j)
End If
Next j
R2 = R2 + p
Next k
Next i
R3 = 0
```

```
For i = 1 To TT - 2
For k = i + 1 To TT - 1
For l = k + 1 To TT
p = 1
For j = 1 To c
If Tset(i, j) = 1 Or Tset(k, j) = 1 Or Tset(l, j) = 1 Then
p = p * Crel(j)
End If
Next j
R3 = R3 + p
Next l
Next k
Next i
R4 = 0
For i = 1 To TT - 3
For k = i + 1 To TT - 2
For l = k + 1 To TT - 1
For m = l + 1 To TT
p = 1
For j = 1 To c
If Tset(i, j) = 1 Or Tset(k, j) = 1 Or Tset(l, j) = 1 Or Tset(m, j) = 1 Then
p = p * Crel(j)
End If
Next j
R4 = R4 + p
Next m
Next l
Next k
Next i
R5 = 0
For i = 1 To CC
p = 1
For j = 1 To c
If Cset(i, j) = 1 Then
p = p * (1 - Crel(j))
End If
Next j
R5 = R5 + p
Next i
R6 = 0
For i = 1 To CC - 1
For k = i + 1 To CC
p = 1
For j = 1 To c
If Cset(i, j) = 1 Or Cset(k, j) = 1 Then
p = p * (1 - Crel(j))
End If
Next j
R6 = R6 + p
Next k
Next i
R7 = 0
For i = 1 To CC - 2
For k = i + 1 To CC - 1
For l = k + 1 To CC
p = 1
For j = 1 To c
If Cset(i, j) = 1 Or Cset(k, j) = 1 Or Cset(l, j) = 1 Then
p = p * (1 - Crel(j))
End If
Next j
R7 = R7 + p
Next l
Next k
Next i
R8 = 0
For i = 1 To CC - 3
For k = i + 1 To CC - 2
For l = k + 1 To CC - 1
For m = l + 1 To CC
p = 1
For j = 1 To c
If Cset(i, j) = 1 Or Cset(k, j) = 1 Or Cset(l, j) = 1 Or Cset(m, j) = 1 Then
p = p * (1 - Crel(j))
End If
Next j
```

```
R8 = R8 + p
Next m
Next l
Next k
Next i
R10 = 0
For i = 1 To CC - 4
For k = i + 1 To CC - 3
For l = k + 1 To CC - 2
For m = l + 1 To CC - 1
For n = m + 1 To CC
p = 1
For j = 1 To c
If Cset(i, j) = 1 Or Cset(k, j) = 1 Or Cset(l, j) = 1 Or Cset(m, j) = 1 Or Cset(n,
j) = 1 Then
p = p * (1 - Crel(j))
End If
Next j
R10 = R10 + p
Next n
Next m
Next l
Next k
Next i
If optd1 = True Then
lblfinal.Caption = "The system reliability is :" + Format((1 - R5 + R6 - R7 + R8 -
R10), "0.00")
End If
If optd2 = True Then
lblfinal.Caption = "The system reliability is :" + Format((1 - R5 + R6 - R7 + R8 -
R10), "0.000")
End If
If optd3 = True Then
lblfinal.Caption = "The system reliability is :" + Format((1 - R5 + R6 - R7 + R8 -
R10), "0.0000")
End If
If optd4 = True Then
lblfinal.Caption = "The system reliability is :" + Format((1 - R5 + R6 - R7 + R8 -
R10), "0.00000")
End If
End Sub
```