2006

Regression approach to software reliability models

Abdelelah M. Mostafa
University of South Florida

Regression Approach to Software Reliability Models

by

Abdelelah M. Mostafa

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Mathematics
College of Arts and Sciences
University of South Florida

Major Professor: K. M. Ramachandran, Ph.D.
Chair: Tapas K. Das, Ph.D.
Chris Tsokos, Ph.D.
Marcus McWaters, Ph.D.
Geoffrey Okogbaa, Ph.D.
Jogindar Ratti, Ph.D.

Date of Approval:
June 21, 2006

Keywords: Linear Regression, Non-homogenous Poisson Process, Power Law Process, Non-parametric, Monotone Regression, Rank Regression

Dedication

To my parents, my wife Fatme, and our five children. Their constant love and support have guided me to where I am. My gratitude to them could never be expressed through words.

Acknowledgements

The completion of this dissertation was made possible through the support and cooperation of many individuals. I would like to thank my advisor, Dr. K.M. Ramachandran, who provided thoughtful guidance and encouragement through what seemed to be a never-ending process. I would also like to thank Dr. Marcus McWaters, Dr. Chris Tsokos, Dr. Geoffrey Okogbaa, and Dr. Jogindar Ratti, for helping me understand the significance of this research. In addition, I would like to thank Dr. Tapas K. Das for chairing my dissertation defense. Special thanks to Dr. A.N.V. Rao for his constant support and ideas that guided me through my research. My gratitude to my wife Fatme, for her understanding and love that encouraged me to work, persevere and continue pursuing my doctorate degree. Her steadfast and kind-hearted personality allowed me to face difficulty with courage. She always let me know that she was proud of me, motivating me to continue doing my best. Last but not least, I am greatly indebted to my children Sarah, Bilal, Noah, Sami, and Serene. They form the backbone and origin of my happiness. Their love and support without complaint has enabled me to complete this dissertation. I owe my every achievement to my wife and to my children.

Abdelelah M. Mostafa

Table of Contents

List of Figures

List of Tables

Regression Approach to Software Reliability Models

Abdelelah M. Mostafa

ABSTRACT

Many software reliability growth models have been analyzed for measuring the growth of software reliability. In this dissertation, regression methods are explored to study software reliability models. First, two parametric linear models are proposed and analyzed, the simple linear regression and transformed linear regression corresponding to a power law process. Some software failure data sets do not follow the linear pattern. Analysis of popular real life data showed that these contain outliers and leverage values. Linear regression methods based on least squares are sensitive to outliers and leverage values. Even though the parametric regression methods give good results in terms of error measurement criteria, these results may not be accurate due to violation of the parametric assumptions. To overcome these difficulties, nonparametric regression methods based on ranks are proposed as alternative techniques to build software reliability models. In particular, monotone regression and rank regression methods are used to evaluate the predictive capability of the models. These models are applied to real life data sets from various projects as well as to diverse simulated data sets. Both the monotone and the rank regression methods are robust procedures that are less sensitive to outliers and leverage values. In particular, the regression approach explains predictive properties of the mean time to failure for modeling the patterns of software failure times. In order to decide on model preference and to asses predictive accuracy of the mean time between failure time estimates for the defined data sets, the following error measurements evaluative

criteria are used: the mean square error, mean absolute value difference, mean magnitude of relative error, mean magnitude of error relative to the estimate, median of the absolute residuals, and a measure of dispersion. The methods proposed in this dissertation, when applied to real software failure data, give less error in terms of all the measurement criteria compared to other popular methods from literature. Experimental results show that the regression approach offers a very promising technique in software reliability growth modeling and prediction.

Chapter 1

Introduction and Review of Software Reliability Modeling

1.1 Introduction

Software has become an essential part of industry, medical systems, spacecraft and military systems, and many other commercial systems. The application of software in many systems has led software reliability to be an important research area Musa et al. [44], Rigdon et al. [52], Sinpurwalla et al. [63], Jelinski et al. [27], and Xie [70]. Researchers and engineers have worked to increase the chance that the software systems will perform satisfactorily during operation. This process required the removal of faults during the testing phase. Researchers used existing technologies in order to improve the software reliability significantly by avoiding the occurrence of faults in the design and development of software programs. Software reliability is a measure of the quality and performance of a software package. From the statistical point of view, software reliability deals with probabilistic methods applied to the analysis of random occurrences of failures in a software system. In this study, the statistical description is concerned with the time-intervals between failures. Thus, software reliability can be defined as the probability that no failure occurs up to time t. A failure is the departure of software behavior from the user requirements. This phenomenon must be distinguished from the fault (bug) in the software code which causes the occurrence of failure as soon as it is activated during program execution. If each time after a failure has been experienced, the underlying fault is detected and fixed correctly, then the reliability of software will improve with time.

Many software reliability models (SRM) are based on the inter-failure times. In this work, we are considering similar modeling principles. Our classification follows that of Horigome et al. [22] by analyzing the time between successive failures. As failure occurrences initiate the removal of faults, engineers reported failure times and time between failures (TBF). Both have been used to find the mean time between failures (MTBF), which is then used to investigate the reliability growth. Models that discuss the behavior of MTBF are called SRMs. Since 1970, scientists and engineers have been developing different models to analyze failure data in order to improve software reliability. These successive software failure times are assumed to be statistically independent.

It is now well-recognized that reliability growth models are better described by a non-homogeneous Poisson process (NHPP). A power law process (PLP) is a special case of a NHPP. Horigome et al. [22] proposed a power law process (PLP) model for describing the reliability growth, which relates the lifetimes of one stage to the next. By using a logarithmic transformation of the time between failures (TBF), they reduced the PLP model to an autoregressive process of order 1. An extension of this work, assuming stochastically monotone failure rate, is given by Littlewood et al. [32] (model I). Mazzuchi et al. [34] made extensions (model II) to Littlewood et al. [32] and proposed new computational techniques. Suresh [64] used a different approach to model II of Mazzuchi et al. [34] and observed an improvement in the following two error measures: the mean square error (MSE) and mean absolute value difference (MAVD). Qiao [49] and Qiao et al. [50] proposed a reliability growth model based on PLP. Roberts [54], used the ideas of Qiao et al. [50] to study software reliability models and reported an improvement in these error measurements.

In this chapter, we present some theoretical basis for some software reliability growth models. Over time, statisticians and scientists have discovered many software reliability growth models. Because of this vast variety of models, we will not present all of them in detail. Thus we will focus on the most widely-used models. This chapter concerns the relationship between failure time and time between failures. Therefore it is very important first to consider failure time distributions for homo-

geneous populations. Throughout the literature on failure time of software systems, certain parametric models have been used repeatedly; we will include and present the exponential model by Moranda [38] and Musa [41], the power model by Crow [11], and the Rayleigh model by Schick and Wolverton [58]. These distributions have closed form expressions for tail area probabilities and simple formulas for intensity functions. In this section, we discuss some of the standard failure time models for homogeneous populations. Therefore, we focus on models for the logarithm of failure time. For more details on different Software Reliability Growth Models, we refer the reader to the 'Recommended Practice for Software Reliability' of the American Institute of Aeronautics and Astronautics [2] and to the Handbook of Software Reliability Engineering by Michael Lyu [33].

1.2 Hardware and Software Reliability

It should be noted that software and hardware systems are different. Their environmental conditions vary, and their failure causes and failure consequences are dissimilar. However, the probabilistic definitions are identical and the theories of probability and statistics are also similar. After most engineering products have been completed, tested, and sold, it is expected that the products work reliably. But with software products, it is possible that one discovers that it has major bugs and the software system does not work reliably. It is possible that some software works well with one user while it does not work well with another user. The most competent programmers in the world cannot avoid error codes completely in the software products; it is not the maturity of the methods and tools used by software professionals that make software production so critical. Rather, it is the conceptual complexity of software that makes software unreliable. While software products are less reliable than other engineering products, many real world applications depend on these products and their required quality level.

Definition 1.2.1 Hardware reliability of a device is defined as: the probability that the device will perform its intended function for at least a specified period of time

under specified environmental conditions.

Definition 1.2.2 Software reliability is the probability that a software will function without failure under a given environmental condition during a specified period of time for which it was designed, given that it was used within the design limits and that the last failure occurred at a given time.

1.2.1 Hardware Reliability

1. Failures are caused by deficiencies in design, production, and maintenance.
2. Failures are due to wear or any other energy, parts related phenomena, but one can get a warning ahead of time.
3. Preventive maintenance is available and makes the system more reliable.
4. Reliability is time related. Failure rates may be decreasing, increasing or constant with respect to operating time.
5. Reliability is related to environmental conditions.
6. Reliability can be theoretically predicted from physical bases.
7. Reliability can be improved by redundancy.
8. Failure rates of the components of a system are predictable by analyzing the pattern of failure times.
9. Hardware interfaces are visual.
10. Hardware Design uses standard components.

1.2.2 Software Reliability

1. Failures are primarily due to design faults in the software. Modifying the design can make it robust versus conditions that could trigger a failure to make the repairs.

2. No wear phenomena in the software. Software errors occur without previous warning. Old codes can exhibit an increasing failure rate as a function of errors induced while making upgrades.

3. Failures occur when the logic path that contains an error is executed. Reliability growth observed as errors in the software can be detected and corrected.

4. External environmental conditions do not affect the software reliability, while the internal environmental conditions affect the reliability, these internal conditions are insufficient memory and inappropriate clock speeds.

5. Knowledge of design, usage, and environmental stress factors are not factors in predicting the reliability.

6. We can improve reliability by debugging and increasing the read access memory.

7. Reliability can be improved by replicating the same error.

8. Reliability can be improved by diversity. In other words; making the software work with different systems.

9. Software interfaces are not visual but are conceptual.

10. Software design does not use standard components; it depends on the qualifications of a programmer.

(Source: Adapted from table 1, page 7, Keene, S.J., 'Comparing Hardware and Software Reliability', ASQ Reliability Review, Vol. 14, Dec. 1994 and table 1, page 4, Walker, E. "Bridging the Software/Hardware Reliability Gap", RAC Journal, Vol. 4, No. 2,2Q96).

1.3 Some Basic Definitions, Concepts, and Terminology

Designing reliable software has acquired its importance due to the following reasons:

- Systems are becoming software intensive.

- Software users are demanding reliable and warranted systems.
- Cost of software development is increasing.

All of the above reasons cause the risk factor that software may fail when delivered to the end users, consequently increasing costs. To minimize the risk of the software failure, one should continuously verify and validate the software through each stage of the software development process. The failure intensity (FI) and the mean time to failure ($MTTF$) are two alternative ways of expressing software reliability. The FI is the expected number of failures per unit time while the $MTTF$ is the expected value of the failure interval. The principal objective of a software reliability model is to forecast failure behavior that will be experienced by the time the program is operational. This expected behavior changes rapidly and can be tracked during the periods in which the program is tested. In general, reliability improves within time as the failure intensity decreases. Previous research discusses various techniques used to determine software reliability. No matter how simple or complex a software program is, it is widely recognized that 100% percent reliability is impossible to obtain.
There are three generally accepted approaches used to pursue highly reliable software:

1. Design software by using structured programming that relies on formal specification languages.
2. Design fault-tolerant software systems that are able to perform satisfactorily, even in the presence of faults.
3. Improve reliability by debugging.

Debugging is still the primary method for achieving reliability. However, this process can consume a significant percentage of the lifespan of the program. The question remains: how does one measure software reliability? Several metrics were proposed to provide an answer to this question. For repeated runs of a program, where the inputs are planned to cover the expected operation, if N is the total number of runs, and N_s is the number of runs completed with no errors, then $\lim_{N\to\infty} \frac{N_s}{N}$ can be interpreted as an experimental reliability figure. Some authors have developed a technique known

as 'error-seeding'. A known number of fictitious faults are intentionally introduced into the program. A debugging team unaware of the location of the artificial faults may try to debug the faults. It can be expected that a certain number of errors may be detected by the team. Therefore, the percentage of discovered fictitious faults can be used to extrapolate the total number of errors originally inherent in the program. The following are inherent errors with this technique:

1. It increases the burden of testing effort.

2. It is difficult to uniformly seed errors in the program;

3. The idea that the total number of errors is representative of reliability is questionable.

4. Another general and more acceptable definition of software reliability can be defined in terms of the time interval between failures. The average processing time between two successive failures is often taken as a significant reliability index. Therefore, the occurrence of a failure is akin to a random event governed by some probabilistic law. The time interval between failures as well as the cumulative number of failures experienced up to a given time are random variables with given statistical distributions. The analysis of software reliability merges perfectly into the mainstream of classical reliability theory.

The bottom line is that as long as the test proceeds and errors are removed from the program, the reliability is expected to increase.

Definition 1.3.1 A repairable system is a system that: after failing to perform one or more of its designed functions satisfactorily, can be restored to an operation condition by some repair process other than replacing the entire system.

A software system is a repairable system. While a software is being developed and tested, programmers detect and correct failures. After the corrections are made, programmers check the software again until another failure is observed. They continue this process until they have a reliable system. Before they put the system in

the market, programmers would like to be sure that they have a desirable reliable system which reflects the quality and performance of the final design. This process of testing leads to the growth of the software system and maintains its reliability. For repairable systems, they observe failure times, let $T_1 < T_2 < ... < T_n$ denote the failure times of the software system, that is the time since the initial start up of the system operation. The times between failures will be denoted $Y_1, Y_2, ...$, and assigned the following: $Y_1 = T_1, Y_2 = T_2 - T_1, Y_3 = T_3 - T_2, ...$

Definition 1.3.2 A repairable system is deteriorating if the times between failure tend to get shorter with advancing time.

Definition 1.3.3 A repairable system is improving or growing if the times between failure tend to increase.

Definition 1.3.4 A failure occurs when one perceives that a software program ceases to deliver its expected outcome.

Definition 1.3.5 A fault is discovered when either an internal error is detected within the software codes or when a failure of the program occurs.

1.4 Fundamentals of Reliability

Software Reliability Modeling plays an important role in developing software systems and enhancing computer softwares. Software reliability theory deals with probabilistic methods applied to the analysis of random occurrence of failures in a given software system. In general, software reliability models fall into two categories depending on the operating domain. The most popular category of models depends on time, whose main feature is that probability measures, such as the mean time between failures and the failure intensity function depend on failure time. The second category of software reliability models measures reliability as the ratio of successful runs to the total number of runs. Because the amount of current research is devoted to time-based models, the first category will be considered. The time domain employs two approaches, the observed time between failures and the number of discovered

failures per time period. In this project, we will follow the time between failures. This section introduces some important terms, concepts, and notations which are frequently used in this dissertation.

1.4.1 Failure Interval Description (FID)

Suppose that a repairable system is observed until n failure times $t_1, t_2, \ldots, t_n$ occur, where $0 < t_1 < t_2 < \ldots < t_n$. Let $T > 0$ be the random variable representing the time to next failure. The reliability function $R(t)$ is the probability that a system will achieve its mission through time t. In other words, the probability of no failure occurs up to time t and is expressed by

$$R(t) = P(T > t) = \int_t^\infty f(x)\,dx \quad (t > 0), \tag{1.4.1}$$

where $f(t)$ is the probability density function (pdf) of the failure time $T > 0$. The cumulative distribution function (cdf) of the random variable T can be written in terms of $R(t)$ as follows:

$$F(t) = \int_0^t f(x)\,dx = P(T \leq t) = 1 - R(t) \tag{1.4.2}$$

The reliability function is also called the survival function of T. $R(t)$ decreases from 1 to 0 as $t = 0$ to $t = \infty$. Hence, $f(t)$, $F(t)$, and $R(t)$ are equivalent representatives of the random variable T.

A NHPP is described by the failure intensity function, which is denoted by $v(t)$.

Definition 1.4.1 The intensity (or failure rate) function is the probability that a failure occurs in an infinitesimal time interval $[t, \quad t + \delta t]$ given that no failure has occurred before time t. The *intensity* (or *failure rate*) function of T is defined as:

$$v(t) = \lim_{\Delta t \to 0} \frac{P(t < T \leq t + \Delta t \,|\, T > t)}{\Delta t}, \tag{1.4.3}$$

from this, we can get:

$$v(t) = \lim_{\delta t \to 0} \frac{R(t) - R(t + \delta t)}{\delta t R(t)} = \frac{f(t)}{R(t)}.$$

Other equivalent relations are:

$$v(t) = \frac{f(t)}{1 - F(t)} = \frac{d}{dt}[-ln(1 - F(t))] = \frac{d}{dt}[-lnR(t)]$$

The probability density function $f(t)$, the cumulative distribution function $F(t)$, the reliability function $R(t)$ and the failure rate function $v(t)$ are closely related to each other. Under general conditions, any one of these can be determined from the others given the failure rate function $v(t)$, and the reliability function $R(t)$. Failure rate function may be computed by:

$$R(t) = e^{-\int_0^t v(x)dx} \tag{1.4.4}$$

$$f(t) = v(t)e^{-\int_0^t v(x)dx} \tag{1.4.5}$$

If the intensity function increases, then the probability of failure over a specific interval of time becomes greater as long as time proceeds. This trend indicates that the software system deteriorates. On the contrary, if the intensity function decreases, this indicates that the software reliability is growing. Notice that in software systems it is reasonable to assume that the intensity function may change only when the program undergoes some modification in its codes (addition of new codes in the program, fault removal, and so on).

The time interval under which the system software will be used is important. In order to achieve a required outcome, it is essential that the system software involves functions properly with an extremely high reliability during a short time interval, which is usually shorter than other phases of software development. For software in commercial and industry applications, the time interval for which software is designed is supposed to be much longer.

There are three synthetic measures of reliability, namely:

- Mean-time-to-failure ($MTTF$).
- Mean time between failures ($MTBF$).
- Median of the random variable T.

The mean-time-to-failure ($MTTF$) is defined by:

$$MTTF = E(T) = \int_0^\infty tf(t)dt = \int_0^\infty R(t)dt. \tag{1.4.6}$$

where $MTTF$ is the average interval of time expected to the next failure time. In other words given the reliability function $R(t)$, $MTTF$ is thus a measure of the average time to failure for software system with life distribution $F(T)$.
The mean time between failures ($MTBF$) is the expected interval length from the current failure time, say $T_n = t_n$, to the next failure time $T_{n+1} = t_{n+1}$. Let $f(t|t_1, t_2, ..., t_n)$ denote the conditional distribution of failure time T_{n+1} given $T_1 = t_1, T_2 = t_2, \ldots, T_n = t_n$, then the $MTBF$ is defined by:

$$MTBF = \int_{t_n}^\infty f(t|t_1, t_2, \ldots, t_n)dt - t_n. \tag{1.4.7}$$

The reciprocal of the intensity function $\frac{1}{\nu(t)}$ is used to represent the expected time to the next failure time, given that the n^{th} failure time occurred at time t. That is, we consider $\frac{1}{\nu(t)}$ as the $MTBF$. In general, this representation is not accurate as it could be significantly different from the $MTBF$. In fact, under special conditions, $MTBF$ can be approximated by $\frac{1}{\nu(t)}$

$$\hat{MTBF} \approx \frac{1}{\nu(t)} \tag{1.4.8}$$

An alternative measure of reliability is the median of the random variable T. It is defined by:

$$F(\tilde{t}) = R(\tilde{t}) = \frac{1}{2} \tag{1.4.9}$$

The median is always well defined. However, there exists random variables T whose

distribution does not have a finite $MTTF$ value.

Consider an increasing intensity function $v(t)$. This equation indicates that the chance of failure over a specified short interval of time becomes greater as long as time proceeds. This trend is good whenever wear-out phenomena affect the operation of the system, such as in hardware products. If the hazard function is a decreasing function, then it is suitable for determining durability or the expected life-span of the product due to improper design and manufacturing defects. In software systems, it is reasonable to assume that the hazard rate may change only when the program undergoes some modification such as fault removal or new code addition where no physical deterioration effect occurs.

Since the failure intensity function $v(t)$ depends only on the cumulative failure time t and not on the previous pattern of failure times, then we can assume that a failed system is in exactly the same condition after a repair as it was just before the failure.

Definition 1.4.2 If the intensity function has the form:

$$v(t) = \lambda\beta t^{\beta-1}, t > 0, \beta > 0, \lambda > 0, \tag{1.4.10}$$

then the process is called the power law process (PLP), where λ is the scale parameter and β is the shape parameter of PLP [51], [53], and [64].

The power law process is a special case of NHPP. The model $v(t)$ demonstrates whether a software system is improving or deteriorating when one chooses the appropriate parameters. When $\beta > 1$, the failure intensity increases (TBF becomes shorter) at an exponential rate with time, and the PLP models the reliability of a repairable system with rapid deterioration. While, if $\beta < 1$, the intensity function is strictly decreasing (TBF becomes larger). This corresponds to modeling the reliability of a repairable system with rapid improvement. For the PLP, when $\beta = 1$, mean time between failures is equal to a constant value.
The PLP has proved to be useful in reliability modeling for several reasons.

1. It can be used to model deteriorating systems (TBF getting shorter) as well as to model improving systems (TBF getting larger).

2. Duane, Rigdon et al. [51] showed that the failure data of many systems used at General Electric fit a model that is closely related to PLP, and Statistical inference procedures can be used easily and applied to PLP models.

There are two statistical descriptions, namely:

1. Time-interval between failures.

2. Number of failures experienced in a given period.

1.5 Some Probability Distributions and Reliability Functions

In this section, we will describe some of the popular probability distributions that are commonly used in Software Reliability.

1.5.1 Exponential Distribution

It is the most widely used distribution function in reliability analysis. Due to its important properties, the exponential distribution becomes the most commonly applied distribution in life. It is the simplest model for failure times. Thus the one parameter exponential distribution is obtained by taking the intensity function to be constant $\nu(t) = \lambda > 0$, over the range of T. The exponential distribution with parameter λ , denoted by $Exp(\lambda)$ is continuous, having a probability density function(p.d.f) of:

$$f(t) = \lambda e^{-\lambda t} \qquad \lambda > 0$$

The corresponding cumulative distribution function (c.d.f.), the reliability function (r.f.) and intensity function can be estimated as follows:

$$F(t) = 1 - e^{-\lambda t}$$

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

The failure rate function of an exponential distribution with parameter λ is a constant denoted by:

$$\nu(t) = \lambda$$

Both the expected and the standard deviation of an exponentially distributed random variable T are equal to λ.

Theorem 1.5.1 *If the random variable T has an exponential distribution with a cdf given by $F(t) = 1 - e^{-\lambda t}$, then*

$$E(T) = \lambda$$

and

$$V(T) = \lambda^2$$

[51]

Theorem 1.5.2 *The exponential distribution has a constant intensity function and is the only distribution with a constant intensity function.*

Proof. The intensity function is:

$$\nu(t) = \frac{f(t)}{R(t)} = \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} = \lambda$$

We must show that this distribution is the only distribution with a constant intensity function. Suppose that the intensity function is $\nu(t) = \theta$. Then:

$$F(t) = 1 - e^{-\int_0^t \nu(x)dx}$$

$$= 1 - e^{-\int_0^t \theta dx} = 1 - e^{-\theta x}$$

which is the cumulative distribution function of the exponential distribution of $\frac{1}{\theta}$ ($EXP(\frac{1}{\theta})$). The mean and the variance of this distribution is given by

$$E(T) = \frac{1}{\theta}$$

$$V(T) = \frac{1}{\theta^2}$$

Recall that for the exponential distribution, the intensity function is constant and is the reciprocal of the mean.

■

The parameter λ can be interpreted as the instantaneous failure rate, sometimes called failure intensity. It is independent of t, such that the conditional chance of failure in a specified time interval is the same regardless of how long the software system has been studied; thus by using the formula $MTTF = \frac{1}{\lambda}$, which is the reciprocal of the intensity function, it is shown in [6] that the median of the random variable is:

$$\tilde{t} = \frac{(ln2)}{\lambda}$$

1.5.2 Weibull Distribution

A direct generalization of the exponential distribution is the Weibull distribution. We are discussing the Weibull distribution for three reasons. First, it is most commonly used for the distribution of lifetimes. Second, it is related to the power law process and is used for repairable systems. Third, if debugging the system, that is bringing it back to a new system, then the assumption that the times between failures $T_1, T_2, ..., T_n$ are independent identical distribution Weibull random variables may be reasonable.
The Weibull distribution has the reliability function

$$R(t) = e^{-\lambda t^{\beta}} \tag{1.5.11}$$

If T is a random variable with this cdf, then X is distributed to $Weibull(\lambda, \beta)$. The cdf, pdf, and the intensity function are given by:

$$F(t) = 1 - R(t) = 1 - e^{-\lambda t^{\beta}}, \quad t > 0 \tag{1.5.12}$$

$$f(t) = F'(t) = \lambda \beta t^{\beta - 1} e^{-\lambda t^{\beta}}, \quad t > 0 \tag{1.5.13}$$

$$\nu(t) = \frac{f(t)}{R(t)} = \frac{\lambda\beta t^{\beta-1}e^{-\lambda t^{\beta}}}{e^{-\lambda t^{\beta}}}, \quad t > 0 \tag{1.5.14}$$

$$\nu(t) = \lambda\beta t^{\beta-1}, \quad t > 0. \tag{1.5.15}$$

The probability density function (p.d.f) is a two parameter ($\lambda > 0, \beta > 0$) function. The parameters β and λ are referred to as the shape and scale parameters, respectively.

The mean of the Weibull function can be expressed in terms of the gamma function.

Theorem 1.5.3 *If T is distributed with Weibull(λ, β) , then*

$$E(T) = \frac{1}{\lambda^{\frac{1}{\beta}}}\Gamma(1 + \frac{1}{\beta}) \tag{1.5.16}$$

Proof. The expectation is:

$$\begin{aligned} E(T) &= \int_0^{\infty} tf(t)dt \\ &= \int_0^{\infty} tf(t)\lambda\beta t^{\beta-1}e^{-\lambda t^{\beta}}dt \end{aligned}$$

Let $y = \lambda t^{\beta}$, thus $dy = \lambda\beta t^{\beta-1}dt$, and the limits of integration are the same. Therefore we have:

$$\begin{aligned} E(T) &= \int_0^{\infty} \lambda^{\frac{-1}{\beta}} y^{\frac{1}{\beta}} e^{-y}dy \\ &= \lambda^{\frac{-1}{\beta}} \int_0^{\infty} y^{1+\frac{1}{\beta}-1} e^{-y}dy \\ &= \lambda^{\frac{-1}{\beta}}\Gamma(1 + \frac{1}{\beta}) \end{aligned}$$

■

Therefore, the mean time until next failure is:

$$MTTF = \frac{1}{\lambda^{1/\beta}}\Gamma(\frac{1}{\beta} + 1) \tag{1.5.17}$$

where $\Gamma(\frac{1}{\beta}+1)$ is the Gamma function.

The intensity function $v(t)$ for the Weibull distribution is a power function of failure time. The value $\beta = 1$ corresponds to a constant intensity function, such that the exponential distribution (with $\lambda = \beta$) is obtained as a special case. The intensity rate decreases when $0 < \beta < 1$ and increases when $\beta > 1$. Therefore, the exponential function is a special case of the Weibull distribution when $\beta = 1$. But the gamma distribution is a useful model for the non-repairable system and has a relation with the Poisson Process.

1.5.3 Rayleigh Distribution

A Rayleigh distribution is a special case of a Weibull distribution. The p.d.f. of Rayleigh distribution is given by:

$$f(t) = kte^{-\frac{kt^2}{2}}$$

The cumulative distribution function is:

$$F(t) = -e^{-\frac{kt^2}{2}}$$

The reliability function is:

$$R(t) = 1 - F(t) = 1 + e^{-\frac{kt^2}{2}}$$

The intensity failure function is:

$$v(t) = \frac{f(t)}{R(t)} = \frac{kte^{-\frac{kt^2}{2}}}{1 + e^{-\frac{kt^2}{2}}}$$

The mean time between failures is:

$$M\hat{T}BF = \frac{1}{v(t)} = \frac{1 + e^{-\frac{kt^2}{2}}}{kte^{-\frac{kt^2}{2}}}$$

$$\hat{MTBF} = (kt)^{-1}e^{-\frac{kt^2}{2}} + (kt)^{-1}$$

It can be seen as a special case of Weibull distribution for $\beta = 2$ and $\lambda = (\frac{k}{2})^{1/2}$. In this case,

$$MTTF = (\frac{\pi}{2k})^{\frac{1}{2}}$$

In 1980, Goel [70] criticized the realism of this model for pure software. The main problem of, is that it is accepted that the failure intensity function is independent of time, provided that the software code is not changed and the test environment is random. Therefore, the assumptions of a time-dependent failure rate is not theoretically justified. But, although this model has suffered from critique for its un-realism, it is still worth considering it in applications. An advantage of this model is that it can be useful in combined hardware and software system.

1.6 Literature Review

1.6.1 Software Reliability Growth Model

A software is said to contain a fault if, for input data, the output result is incorrect. A fault is always an existing part in software codes. Therefore, the process of software debugging is a fundamental task of the life cycle of a software system. During this period, the software program is tested many times with the intent of discovering faults contained. When a failure is observed, the code is inspected to find the fault which caused the software failure. The fault is usually removed by correcting the software codes. As a result, one expects the software reliability to increase during the testing phase as more and more faults are removed. The reliability improvement phenomenon is called reliability growth. The size and the complexity of the software packages make it impossible to find and correct all existing faults. The best thing is to give software a reliability requirement and to try to attain a goal by testing the software and correcting the detected faults. However, obtaining the required software reliability is not an easy task. Thus, high reliability is usually estimated by using appropriate models applied on failure data from the software failure history. A

Software reliability model is a mathematical description of the debugging and fixing process built in the following three different stages:

1. Model structure is selected.
2. The free parameters in the model are tuned on the basis of the experimental data.
3. A rule is given to use the estimated model for predictive purposes.

A software reliability model falls into two categories that depend on the operating domain. Thus, the most popular models are based on time. Their main feature of reliability measures, such as the failure intensity which is derived as a function of time. The second kind of software reliability models have a different approach. This approach is made by using operational inputs as their main features, which measure reliability as the ratio of successful runs to total runs. The second approach has some problems such as: many systems have runs of large lengths with output measures that are incompatible with the time-based measures. Due to these problems, the work of this dissertation has been devoted to time-domain models. The time domain model employs either the observed time between failures or the number of discovered failures per time period. Thus, these two procedures were developed to estimate the model parameters from either failure count data or time between failures. Therefore, software reliability modeling and estimation can be grouped into two categories of general applicability:

1. Failure counting description (FCD).
2. Failure interval description (FID).

1.6.2 Littlewood-Verrall Model (LV)

Successive times to failures are seen as random variables with exponential distribution where the intensity function is no longer a decreasing deterministic function. The failure intensity is assumed to be a stochastic decreasing random variable with a

Gamma function of parameters α and $\theta(i)$.

$$f(\sigma_i) = \frac{[(\theta(i))^{\alpha}\sigma_i^{\alpha-1}e^{-\theta(i)\sigma_i}]}{\Gamma(\alpha)}$$

The substantial difference between the JM and LV models is that in the JM model, a fix always leads to a reduction of the failure intensity, which has always had the same value. While in the LV model, the failure intensity follows a random pattern so that the magnitude of its variations is not necessarily a constant. Furthermore, the sign of the variation may vary; that is, a fix may not result in a reliability improvement.

1.7 Non Homogenous Poisson Process (NHPP) Models

The Non-Homogeneous Poisson Process (NHPP) is a poisson process whose intensity function is nonconstant. For more details about the NHPP theory, we refer the reader to [51], [70], and [63]. In this section, we present briefly an introduction and present briefly some existing software reliability models for the the failure process that are described by the NHPP. NHPP models have been studied and used successfully in hardware reliability systems. It describes failure processes which have certain trends such as reliability growth or deterioration. The applications of NHPP models models have been implemented to software reliability. The cumulative number of failures up to time t, $N(t)$, can be described by NHPP. Many software reliability models belong to this category. The Poisson process model for describing the uncertainty of the counting process $N(t), t \geq 0$ is the simplest of counting process models. The counting process modeled by NHPP, where $N(t)$ follows a Poisson distribution with parameter $m(t)$, which is the mean value function. The probability that $N(t)$ is an integer is denoted by:

$$P(N(t) = n) = \frac{(m(t))^n}{n!}e^{-m(t)}, \quad n = 0, 1, 2, ... \tag{1.7.18}$$

where $m(t)$ is the mean residual time or expected cumulative number of failures in $[0, t)$. The assumptions of NHPP are:

- $N(0) = 0$
- $(N(t), \quad t \geq 0)$ has independent increments
- $P(N(t + \Delta t) - N(t) = 1) = \nu(t) + o(\Delta t)$
- $P(N(t + \Delta t) - N(t) \geq 2) = o(\Delta t)$

Where $o(\Delta t)$ approaches zero for small Δt.
The instantaneous failure intensity $\nu(t)$ is denoted by:

$$\nu(t) = Lim_{\Delta t \to 0_+} \frac{P(N(t + \Delta t) - N(t) > 0)}{\Delta t} \tag{1.7.19}$$

The mean value function is:

$$m(t) = E(N(T)) = \int_0^t \nu(x)dx \tag{1.7.20}$$

If $m(t)$ is known, the the failure intensity $\nu(t)$ is:

$$\nu(t) = \frac{dm(t)}{dt} \tag{1.7.21}$$

If $\nu(t)$ is constant, then we have a homogeneous Poisson process (HPP).
In 1975, Schneidewind [59] was the first to suggest the NHPP model. However, in 1979, Goel-Okumoto [15] was the first who presented a simple model for the software failure process. He assumed that the cumulative failure is a NHPP with a simple mean value function. Later, the Goel-Okumoto (GO) model became very well known among software reliability [70]. Goel and Okumoto [15] proposed the time dependent failure rate model based on NHPP. Ohba [47] and Ohba-Yamada [48] proposed some particular NHPP models such as the delayed S-Shaped software reliability models, and the inflection S-Shaped model. Musa and Okumoto [45] proposed the Logarithmic Poisson execution time model. Musa [42] proposed the basic execution time model. Goel [13], [14] introduced the test quality parameter. Littlewood assumed a modification of the Duane model based on NHPP. Yamada et al [71] assumed a model with two types of faults. In 1986, Yamada et al [73] assumed a discrete time

model. Yamada et al [72] assumed a testing effort dependent model that assumes the testing effort to follow either exponential, Weibull, or Raleigh distribution. Kapur and Garg [29] used a modified G-O model by introducing the concept of imperfect debugging. Kareer et al [30] assumed two types of fault models such that each fault type was modeled by an S-Shaped curve. Xie [70] assumed simple models with graphical interpretation. Yamada et al [72] assumed a model based on the testing domain.

1.7.1 Goel-Okumoto Model

The Goel-Okumoto Model assumes that the number of failures $n(t)$ follows a $NHPP$ distribution with expected value :

$$E(n(t)) = m(1 - e^{-\phi t})$$

Parameters m and ϕ are estimated with the maximum likelihood approach and predictions can be obtained. Both JM and GO models are conceptually indistinguishable on the basis of a single realization of the time to failure process. The only actual difference between the JM and GO models is the maximum likelihood function, which is differently built in the two models. For this reason, the predictions supplied by the two models do not coincide. In 1984, Musa discussed the possibility of classifying the models in terms of different attributes. The time domain was used for these models, where the calendar time or the execution time was adapted. Only a few models assume the execution time as the underlying time measure. For any dynamic model, failure time can be incorporated, and the probabilistic nature of the model assumptions is not changed.

1.7.2 Power Model

The power model was developed by Crow [11] in 1974 as a model for hardware reliability. This model has the ability to be applied for the prediction of software

reliability as well. The mean value function is:

$$\mu(t;\lambda,\beta) = \lambda t^{\beta}$$

The failure intensity function is:

$$\nu(t;\lambda,\beta) = \lambda\beta t^{(}\beta - 1)$$

If $\beta < 1$, then the software reliability improves.

1.7.3 Inverse Polynomial Model

The inverse polynomial model framework was first originated in 1974 by Littlewood and Verral [32]. John Musa applied this model successfully in 1987. The mean value function is:

$$\mu(t;\lambda,\beta) = 3\lambda(Q_1 + Q_2)$$

The failure intensity is:

$$\nu(t;\lambda,\beta) = \frac{\lambda}{\sqrt{t^2+\beta}}(Q_1 - Q_2)$$

where

$$Q_1 = \sqrt[3]{t + (t^2+\beta)^{\frac{1}{2}}}$$

$$Q_2 = \sqrt[3]{t - (t^2+\beta)^{\frac{1}{2}}}$$

1.7.4 Suresh and Robert Models

Cox-Lewis [10] proved that the mean time between failures for a stationary process is the reciprocal of the intensity function and Ascher-Feingold [3] proved that the mean time between failures approaches the reciprocal of the intensity function. Suresh [64] was the first person who applied this estimate procedure to software failure data sets by using the power law process. Suresh derived an extension of

the NHPP by considering the intensity function as a function of time and the time since the last failure. Suresh illustrated this process and proved that the mean time between failures estimates give better results than the models used by Singpurwalla-Horigom and Mazzuchi-Soyer. Suresh then extended a Bayes-Empirical-Bayes model for software reliability by assuming the time between failures to be Weibull random variables. She used an unbiased estimate of the failure rate for reliability prediction, and compared this model with previous models that used the general framework of Bayes-Empirical-Bayes procedure. Robert [54] analyzed the same software failure data sets by following two procedures for software reliability. Robert first used the same model and same software failure data sets as Suresh and proved that the numerical results of Suresh were more significant than previous ones. Later on, Robert applied the Qiao-Tsokos [50] model and computed the MTBF estimates of the time between failures by using different combinations of the estimate parameters of the scale and shape. They showed that the predictive accuracy of error is better than the Mazzuchi-Soyer, Singpurwalla-Horigom, and Suresh models. Robert developed Bayesian models for software systems whose rate of occurrence of failures (ROCOF) is characterized by the power law process.

1.7.5 Other Models

Here, we briefly describe some existing reliability growth models that deal with software systems. The Duane model, also referred to as the Weibull process model, can be interpreted as an NHPP model for software reliability growth. The model assumes the mean value function as:

$$m(t) = \lambda t^{\beta}, \quad \lambda > 0, \quad \gamma > 0$$

where α and β can be estimated by using the collected failure data. The rate of occurrence of failures (ROCOF) at time t is

$$\lambda(t) = \frac{dm(t)}{dt} = \lambda\beta t^{\beta-1}, \quad \lambda > 0, \quad \beta > 0$$

The main advantage of the Duane reliability growth model is the graphic representation of the cumulative number of failures versus the cumulative failure time on a log-log scale sheet. If the trend of the plotted graph is close to a straight line, then the model is valid [51]. The main disadvantages of the Duane model are ROCOF becomes zero at time infinity and becomes infinite at time zero. Littlewood modified the Duane model by assuming the mean value function of the modified Duane model as:

$$m(t) = k\left(1 - \left(\frac{\alpha}{\alpha+1}\right)^{\beta}\right) \quad \alpha > 0, \quad \beta > 0, \quad \kappa > 0$$

where κ represents the number of failures to be detected. The corresponding ROCOF is denoted by:

$$\lambda(t) = m'(t) = \kappa\beta\alpha^{\beta}(\alpha+t)^{-\beta-1} \quad \alpha > 0, \quad \beta > 0, \quad \kappa > 0$$

The logistic growth model is denoted by:

$$m(t) = \frac{\kappa}{1+\alpha e^{-bt}}, \quad \alpha > 0, \quad b > 0, \quad \kappa > 0$$

where α, b, and κ are constant parameters to be estimated by fitting the failure data. Also, κ is the expected number of failures. Note that $m(\infty) = \kappa$.
The Gompertz growth model is denoted by

$$m(t) = \kappa\alpha^{b^t}, \alpha > 0, \quad b < 1, \quad \kappa > 0$$

where κ is the expected number of failures, and α, and b are constant parameters. We also have deduced that $m(\infty) = \kappa$.

1.8 Model Selection and Comparison

Model Selection

Most of the commonly repairable systems are homogenous or non-homogenous poisson processes. We start by analyzing four data sets. This is done through using

graphical methods which are used for displaying data from repairable systems. In order to get an insight into the data and choose a reasonable model. We checked whether there was any trend in times between failure or whether the times between failure were changing or remaining constant. We displayed scatter plots of the cumulative number of failures through time t_i, $N(t_i)$ on the vertical axis, and the failure times along the horizontal axis.

If the time between failures tends to stay approximately the same, or, in other words, the intensity function remains constant over time and the graph shows a linear relationship, then, the possibility of a homogeneous poisson process may be considered to be an appropriate model if the times between failures are independent. If after removing bugs the time between failures tends to get longer, then we can consider the system to be an improving system, meaning that the intensity function decreases. This can be employed as a reliability growth model because the graph shows a concave down curvature which indicated reliability improvement, and hence is of interest for us. If the last condition is concave upward, then it illustrates a deteriorating system. This happens because after the removal of software bugs, the time between failures decreases, that is, the intensity function increases. If the graphs of $N(t_i)$ versus t_i have significant curvatures, the software data may be modeled by a non-stationary process which is capable of describing the occurrence of failure events in time. The intensity function is defined by:

$$v(t) = \lambda\beta(t)^{\beta-1}$$

This plays an important role in selecting the right model. A special case of the non-homogeneous process (Power Law Process) has been used for improving or deteriorating systems. Usually, the assumptions of independent and identical distributions for the time between failures in repairable systems are valid. The intensity function plays an important role for choosing the right model because it contains information about the likelihood of failure occurrence at or around any time t. The intensity function is proportional to the time raised to a power, that is, it changes as a system

ages. A way to verify software data is to use graphical methods for plotting the cumulative number of failures through time t_i, $N(t_i)$, on the vertical axis, and the failure time t_i along the horizontal axis. Consider the software failure times in Table (1.1) by assuming that each of the three systems was observed until the twelfth failure time. We selected the failure times in order to illustrate how the graphical methods help us analyze the data to give more information about verifying whether the system is improving.

Table 1.1: Failure Times for Three Assumed Systems (Systems A, B, and C)

Failure Number	System A	System B	System C
1	3	9	20
2	5	20	45
3	9	65	76
4	20	88	113
5	25	104	129
6	41	107	152
7	50	138	174
8	69	143	193
9	91	149	199
10	128	186	210
11	151	208	220
12	190	230	226
13	245	237	228

Model Comparison

In order to gain a good insight of the data and to help a researcher select a reasonable model for repairable systems, one can use graphical methods for displaying data. When one analyzes a software failure data, one should verify whether there is a trend in the times between failures and whether the time between failures increases,

decreases, or fluctuates within the testing time. One would use a simple way to plot the failure time along the x-axis and the cumulative number of failures along the y-axis. One could also plot the time between failures along the y-axis. There are two demonstrations in this dissertation. Figure (1.1) shows the graphs of $N(t_i)$ versus t_i. This graph contains three cases. The plot shows that there is a curvature for systems A and C. The plot of system A is curving down indicating that the number of failures is decreasing over an interval of time. In other words, system A is improving while the curvature of system C is curving up, that is system C is deteriorating. The plot of system B fluctuates up and down. For systems A and C, we have to consider the $NHPP$, for system B, the intensity function is a constant and the HPP is a proper choice for system B. Since we are choosing to select the $NHPP$, then the power law process is capable to set up the three conditions by applying different values of the shape parameter β. The following graph is a descriptive presentation of three systems. This will guide us to the next step.

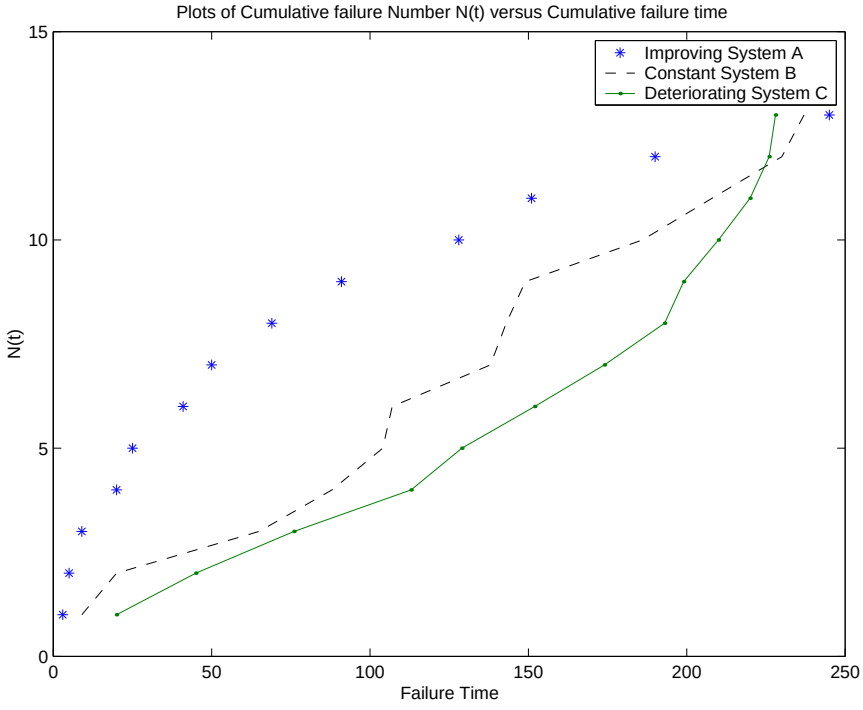

Figure 1.1: General Model Motivation Using Number of Failures

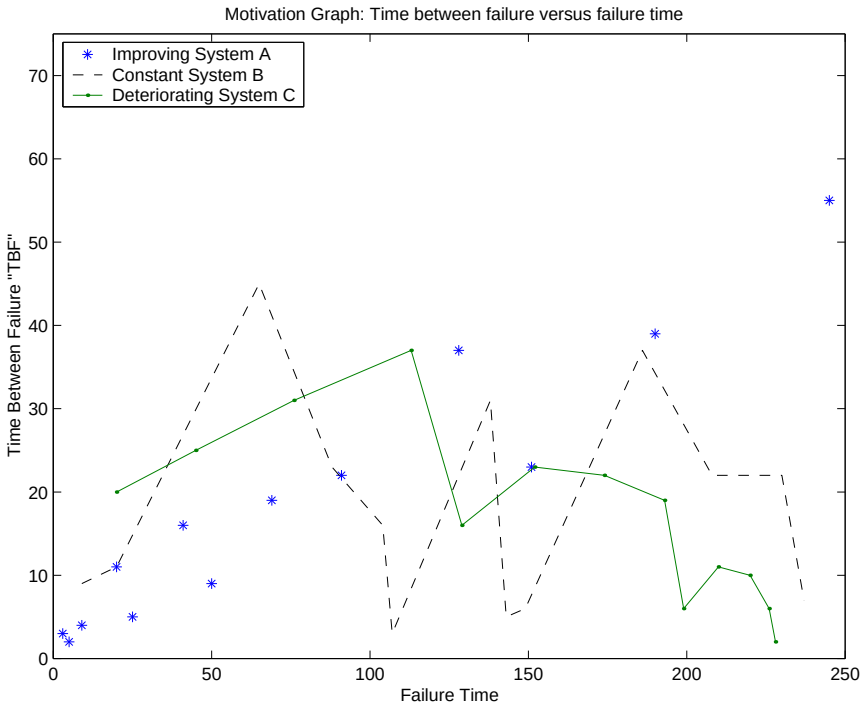

Figure 1.2: General Model Motivation Using Time between Failures

Figure (1.2) shows the same descriptive analysis as Figure (1.1). But in this case, one should watch the behavior of time between failures. The graph illustrates that: time between failures increases for system A. That is, system A is improving, meaning that after removing the code bugs, time between failures tends to become longer and the system should improve. Thus, the intensity function decreases, in other words, the probability of failures becomes smaller within time. This indicates a reliability growth model which is of interest to the researcher. The graph shows that System C is deteriorating, that is after removing the code bugs, the time between failures gets shorter and shorter. Thus the intensity function increases within time. The plot of system B indicates that time between failures fluctuates and tends to be somehow the same, that is, the system is stable and the intensity function is constant. For systems A and C, one should employ NHPP, while one should employ HPP is for system B. If NHPP is chosen, then one can use the power law process which is flexible enough to set up models for the three cases by applying different

values of the shape parameter β. More details and derivation will be given in the following chapters.

Software reliability models are the results of a trial and error processes aimed to achieve a reasonable trade-off between the model statistical characteristics especially in terms of predictive validity. To test the predictive reliability of a model, the software reliability up to a certain point, i, can be used to estimate the model parameters and to use the estimated model to predict the future value at point $i+j$ of a variable of interest A posteriori, the estimate can be compared with the observed "true" value taken by the variable at time $i+j$. This procedure can be repeated either by subsequently increasing i, so as to cover all available data, or by extending the prediction horizon d, while keeping the number i of data point constant.

1.9 Error Measurements Prediction Criteria

After one estimates the model parameters, there are a number of interesting questions: how accurate is the prediction and what are the good indicators to select a prediction? In this section, we will establish metrics to evaluate the accuracy of the estimates from the prediction models. We denote the estimated value of a measure of the time between failures with $\hat{MTBF}$ and the actual value with TBF. We are using the following metrics to evaluate the accuracy of estimates and to compare the software models.

Mean Squared Error (MSE)

MSE is the most commonly used error measurement criteria of prediction. The MSE of an estimator T of an unobservable parameter θ is defined by

$$MSE(T) = E((T-\theta)^2)$$

Let TBF be the actual time between failures, and $\hat{MTBF}$ is the predicted mean

time between failures. For simplicity of computation, use the following formula

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(TBF_i - M\hat{T}BF_i)^2 \tag{1.9.22}$$

Mean Absolute Value Difference ($MAVD$)

$MAVD$ is defined as the average of the difference between predicted mean time between failures and actual time between failure values, and computed by

$$MAVD = \frac{1}{n}\sum_{i=1}^{n}\left|TBF - M\hat{T}BF\right| \tag{1.9.23}$$

Magnitude of Relative Error (MRE)

Another popular evaluation criteria used in [46] is to assess the performance of predictive repairable models. It is the absolute value of the relative error, defined by

$$MRE = \left|\frac{M\hat{T}BF - TBF}{TBF}\right| \tag{1.9.24}$$

Mean Magnitude of Relative Error ($MMRE$)

It is the mean of MRE . Conte et al. [9] considered $MMRE \leq 0.25$ to be an acceptable value for prediction models effort. There are advantages for this assessment:

1. Comparisons can be made easy across failure time data sets [7], [69].
2. The Mean magnitude of relative error is independent of units of data.
3. Comparisons can be made across all types of prediction models [9].
4. Since $MMRE$ is independent of scale, that is the expected value of MRE does not vary with size.

Magnitude of Error Relative to the Estimate (MER)

Kitchenham et al. [31] proposed another measure, the magnitude of error relative to

the estimate. The MER is defined as

$$MER = \frac{\left|M\hat{T}BF - TBF\right|}{M\hat{T}BF} \tag{1.9.25}$$

Mean Magnitude of Error Relative to the Estimate ($MMER$)

$MMER$ as a measure of prediction error is defined as the mean of MER. In [46], showed that claims (2), and (4) hold, and that MER measure seems preferable to MRE because it measures the error relative to the estimate value of mean time between failures.

Median of Absolute Residual ($MdAR$)

Another measure proposed in Kitchenham [31] is median of the absolute error AR instead of $MMRE$, where the absolute error is defined as

$$AR = \left|TBF - M\hat{T}BF\right| \tag{1.9.26}$$

Then, $MdAR$ is the median of the values of AR. Also, they proposed MAR, which is the mean of AR. MAR is nothing but $MAVD$, which we have already calculated. In chapters two and three, we use these different methods of measurement (MSE , $MAVD$, $MMRE$, $MdAR$) for some of the software failure time data sets such as System 40, Project 1, and Project 5 in [40].

1.10 Data Sets

Apollo 8 software failure data is taken from resources of [54], and [64].

The Software Reliability Data sets (Project 1, Project 5, and System 40) was compiled by John Musa of Bell Telephone Laboratories [40], whose objective was to collect failure interval data to assist software managers in monitoring test status and predicting schedules and to assist software researchers in validating reliability models. These models are applied in the discipline of Software Reliability Engineering. Careful controls were employed during data collection to ensure that the data would

be of high quality. This data was collected throughout the mid 1970s. It represents projects from a variety of applications, including real time command and control, word processing, commercial, and military applications. For each software failure in the data set, the following items are recorded:

- Project Identification (System Code): an internally assigned identification number.
- Failure Number: a number identifying a particular failure. Failures are consecutively numbered from the first failure recorded.
- Failure Interval (TBF).

The time elapsed from the previous failure to the current failure. The time between failures in projects 1, 5, and 40 is the time given in wall-clock seconds. More detailed information on the specific characteristics of each project is available at Data and Analysis Center for Software [43]. The original data of Project 5 has 832 observations, but we consider 810 observations only, because the 832nd has a negative failure interval length, which is impossible while another 21 observations have zero values of TBF. Note: Since we are looking at TBF, not number of failures, then we replace these values. Project 1 data set is documented in [17], it is originally attributed to Musa in 1979. This data set has been applied in the software reliability community for model comparison. The original data has 137 observations for the time between failures, and the last observation (137th) is negative. We dropped the zero and the negative observations, hence we consider only 133 observations . The following graphical method for displaying the given software failure data sets (Apollo 8, System 40, Project 1, and Project 5) can be used to gain more insight into the data.

Figure (1.3) shows that the graphs of Apollo 8, Project 1, and System 40 are curving downward. This indicates the model is improving. However, the intensity function for these projects will show improvements for the model as we study it in the next chapter. While the graph of Project 5 fluctuates curving up and down, it indicates that the intensity function is approximately constant within failure time.

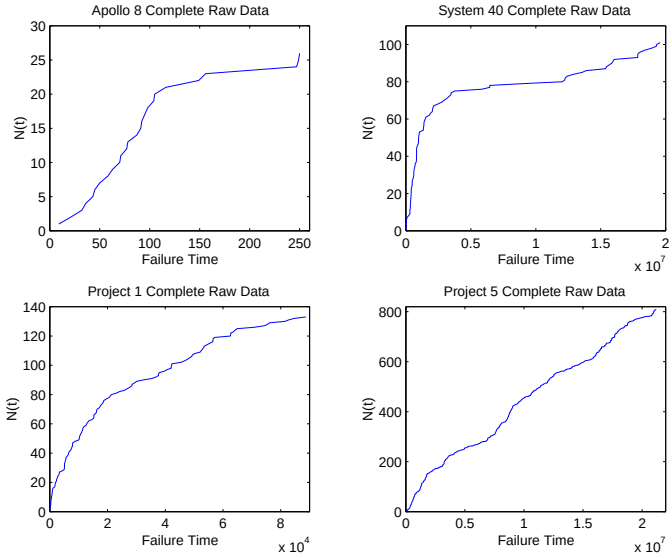

Figure 1.3: Model Motivation Using Software Failure Data Sets

This encourages us to consider the $NHPP$, for Apollo 8, Project 1, and System 40. The HPP can be considered as a proper choice for Project 5. Since we are choosing to select the $NHPP$, the power law process is capable of setting up the three conditions by estimating the shape parameter β.

1.11 Summary of Dissertation

It has been shown in the literature, Mazzuchi et al. [34], Qiao [49], Qiao et al. [50], Roberts [54], and Suresh [64], that the PLP model is a good modeling assumption to represent TBF. The idea of this work stems from the fact that the logarithm of the PLP equation shows that it is a linear function of logarithm of failure time data. This led us to propose that the model could be taken as a simple linear regression. In this work, based on PLP model, we will compute the predictive properties of mean time to failure ($MTTF$) using Maximum Likelihood Estimation (MLE) method, regression method using logarithm of data with PLP assumption, simple linear regression applied directly to the data, and successive predicted time between failures method. Since some of our failure data do not meet all the assumptions, we

proposed some non-parametric methods by using the monotone and the rank regression procedures. We will show that the proposed models significantly improve the predictive accuracy under several popular measures of error, the mean square error (MSE), the mean absolute value difference ($MAVD$), mean magnitude of relative error ($MMRE$), mean magnitude of error relative to the estimate ($MMER$), median absolute residual ($MAR, MdAR$), and the use of the convex dispersion function. We also simulate software failure times by taking different parameter values of the PLP, and the results were encouraging in terms of reducing our measurement indicators. The methods introduced in this work not only reduce the prediction errors, but are also intuitively appealing and easy to implement.

In Chapter 2, we present the new models; all are primarily based on linear regression. We use the goodness of fit tests and model validation to verify the software reliability models. In addition, we derive an appropriate intensity function when we base our analysis on linear regression. We compare the predictive errors for the new models with those of previous models.

In Chapter 3, we discuss some of the nonparametric regression methods, specifically the monotone regression method and the rank regression method. We consider analyzing the Apollo 8, Project 1, Project 5, and System 40 failure data sets.

In Chapter 4, we conclude our dissertation and present briefly the limitations and some other future research.

Chapter 2

Linear Regression Approaches to Software Reliability Models

2.1 Introduction

As mentioned in chapter 1, engineers, scientists, and statisticians studied the likelihood of failures of a software system in a mathematically and rigorous procedure. Thus, it is based on the idea of using failure times and time between failures in the past to model the failure behavior of a software system. At the same time, we illustrate software reliability modeling and some other related statistical concepts. The NHPP is a good model for repairable systems because it can model systems that are deteriorating or improving. It is shown that the power law process can be described as NHPP with intensity function $v(t)$. The shape parameter β shows how the system improves or deteriorates over time. However, in this chapter, we address some classical inference results on the power law process, and present this work for the parametric regression methods. This include the point estimation for the parameters, goodness of fit tests, model validation, estimation of the intensity function and estimation of the mean time between failures for power law process through regression. Then, we conclude this chapter by applying those new models on three sets of real software failure data.

2.2 Background Results

We assume that the failure times of the test are random. Given the failure rate function, $v(t)$, the reliability function, $R(t)$, and the probability density function,

$f(t)$, can be computed by:

$$R(t) = \exp\left(-\int_0^t v(x)\,dx\right),$$

and

$$f(t) = v(t)\exp\left(-\int_0^t v(x)dx\right) = v(t)\,R(t).$$

The mean time between failures of the repairable systems is the expected interval length from the current failure time $T_n = t_n$, to the next failure time $T_{n+1} = t_{n+1}$.

Let $f(t \,|\, t_1, t_2, \ldots, t_n)$ be the conditional distribution function of failure time T_{n+1} given $T_1 = t_1$, $T_2 = t_2, \ldots, T_n = t_n$. The joint pdf of the observed failure times can be derived using the following expression,

$$f(t_1, t_2, \cdots, t_n) = f_1(t_1) f_2(t_2 \,|\, t_1) f_3(t_3 \,|\, t_1, t_2) \cdots f_n(t_n \,|\, t_1, t_2, \ldots, t_{n-1}).$$

Markov property for the reliability function:

$$R_k(t_k \,|\, t_1, t_2, \ldots, t_{k-1}) = R(t_k \,|\, t_{k-1}) = \exp\left(-\int_{t_{k-1}}^{t_k} v(x)dx\right), \quad t_k > t_{k-1}. \quad (2.2.1)$$

As a consequence, we have

$$f(t_k \,|\, t_{k-1}) = v(t_k)\exp\left(-\int_{t_{k-1}}^{t_k} v(x)dx\right), \quad t_k > t_{k-1}. \quad (2.2.2)$$

We will use the following result from Rigdon et al. [52].

Theorem 2.2.1 *The joint pdf of the failure times $T_1, T_2, \ldots, T_n$ from the Non-Homogeneous Process (NHPP) with intensity function $v(t)$ is*

$$f(t_1, t_2, \ldots, t_n) = \left(\prod_{i=1}^{n} v(t_i)\right)\exp\left(-\int_0^{t_n} v(x)dx\right). \quad (2.2.3)$$

2.2.1 The MLE estimators of λ and β

Now, we describe the MLE approach for reliability estimation. The joint pdf of the failure times $T_1, T_2, \ldots, T_n$ from an NHPP with intensity function $\lambda(t)$ is given by:

$$f(t_1, t_2, \ldots, t_n) = \left(\prod_{i=1}^{n} v(t_i) \right) \exp\left(- \int_0^{t_n} v(x) dx \right)$$

Recall the density function under PLP model

$$v(t_i) = \lambda\beta(t_i)^{\beta-1}.$$

Under the assumptions that the data follows PLP model, the joint pdf of the failure times is

$$\begin{aligned} f(t_1, t_2, \ldots, t_n) &= \left(\prod_{i=1}^{n} \lambda\beta(t_i)^{\beta-1} \right) \exp\left(- \int_0^{t_n} \lambda\beta x^{\beta-1} dx \right) \\ &= (\lambda\beta)^n \left(\prod_{i=1}^{n} t_i \right)^{\beta-1} \exp\left(-\lambda t_n^{\beta}\right), \ 0 < t_1 < t_2 < \ldots < t_n < \infty. \end{aligned} \tag{2.2.4}$$

To get the MLE's, take the natural logarithm of the joint density and set the first partial derivatives (with respect to λ and β) equal to zero.
The natural log-likelihood function is

$$\begin{aligned} ln(f(t_1, t_2, \ldots, t_n)) &= \ell(\lambda, \beta \,|\, t) \\ &= \ln\lambda + n\ln\beta + (\beta - 1)\sum_{i=1}^{n} \ln(t_i) - \lambda(t_n)^{\beta}. \end{aligned} \tag{2.2.5}$$

Now, using the standard derivations Qiao et al. [50], and Rigdon et al. [51], the maximum likelihood estimators (MLEs) of the parameters, λ and β, are

$$\begin{aligned} \hat{\beta} &= \frac{n}{\sum_{i=1}^{n-1} \ln(t_n/t_i)} = \frac{n}{\sum_{i=1}^{n-1} (\ln(t_n) - \ln(t_i))} \\ \hat{\lambda} &= \frac{n}{t_n^{\hat{\beta}}} \end{aligned} \tag{2.2.6}$$

One of the most characteristic measures of reliability is the mean time-to-failure (MTTF) that represents length of the expected (average) interval of time to the next failure.

For the PLP,

$$MTBF_n = \int_{t_n}^{\infty} t\lambda\beta t^{\beta-1} \exp[-\lambda t^{\beta} + \lambda t_n^{\beta}]dt - t_n. \tag{2.2.7}$$

If $\beta = 1$, then

$$MTBF_n = \int_{t_n}^{\infty} \lambda t \exp(-\lambda t + \lambda t_n)dt - t_n = \frac{1}{\lambda}.$$

Thus, we can use $MTBF_n = \frac{1}{v_n(t)}$, Ascher et al. [3], Cox et al. [10], Roberts [54], and Suresh [64] as an approximation for the mean time between failures at nth stage.

Using the MLE, an estimator of the MTBF is given by

$$\hat{MTBF} = \frac{1}{\hat{v}(t)} = (\hat{\lambda}\hat{\beta})^{-1}t^{1-\hat{\beta}}. \tag{2.2.8}$$

In Table (2.5) of section (2.8), we will give the predictive errors, MSE and MAVD, resulting from using these estimators.

2.3 Regression Approach for Power Law Processes

In the current study, we shall apply regression methods for modeling the software failure times. First, we assume the PLP model for the data. For the PLP models, instead of estimating the parameters λ and β through the standard method of MLE, we will estimate these parameters through regression (least squares) approach. We will see that this will result in substantially smaller values of MSE and MAVD compared to models discussed in Crow et al. [12], Horigome et al. [22], Mazzuchi et al. [34], Roberts [54], and Suresh [64]. In addition, we also compute these measures in section (2.4) using the simple linear regression model directly without assuming the PLP model and derive the resulting intensity function. In section (2.5), we will calculate these error measures resulting from the successive prediction using the

regression model. For analysis in this chapter, we use following data sets from Musa [40]: Project 1, Project 5, and System 40.

2.3.1 Intensity Function for the Regression Model:

For the power law process model, the MTBF can be approximated by the inverse of the intensity function Ascher et al. [3], Cox et al. [10], Roberts [54], and Suresh [64]. That is,

$$MTBF = \frac{1}{v(t)} = (\lambda\beta)^{-1}t^{(1-\beta)}, \tag{2.3.9}$$

where t is the failure time. Taking the natural logarithm, we get:

$$\ln(MTBF) = -\ln(\lambda\beta) + (1-\beta)\ln(t). \tag{2.3.10}$$

By writing, $Y = \ln(MTBF)$, $b = -\ln(\lambda\beta)$, $a = (1-\beta)$, and $Z = \ln(t)$, we can rewrite (2.3.10) as a linear equation

$$Y = b + aZ. \tag{2.3.11}$$

Using the method of least squares for the linear regression model, Ryan [56], we can derive least squares estimators of a and b as $\hat{a}$ and $\hat{b}$, where

$$\hat{a} = \frac{\sum Z_i Y_i - (\sum Z_i)(\sum Y_i)/n}{\sum Z_i^2 - (\sum Z_i)^2/n}, \tag{2.3.12}$$

and

$$\hat{b} = \overline{Y} - \hat{a}\overline{Z}. \tag{2.3.13}$$

Now, using the following derivation

$$a = (1-\beta) \text{ implies } \beta = 1 - a,$$
$$b = -\ln(\lambda\beta) \text{ implies } \lambda = \frac{1}{1-a}\frac{1}{e^b} = \frac{e^{-b}}{1-a},$$

we get the regression estimators of the PLP parameters λ and β, denoted by: $\hat{\beta}_{reg}$,

and $\hat{\lambda}_{reg}$, as

$$\hat{\beta}_{reg} = 1 - \hat{a} \tag{2.3.14}$$

and

$$\hat{\lambda}_{reg} = \frac{1}{1-\hat{a}} \frac{1}{e^{\hat{b}}}. \tag{2.3.15}$$

Using these estimators, we can obtain an estimator of MTBF as:

$$\hat{MTBFa}_{reg} = (\hat{\lambda}_{reg}\hat{\beta}_{reg})^{-1} t^{(1-\hat{\beta}_{reg})}. \tag{2.3.16}$$

We will call this as the MTBFa-regression model.

We now present some numerical results using MTBFa-regression model by using real software failure data Musa [40]. The following table gives the MSE and MAVD calculated for three different software failure data of System 40, Project 1, and Project 5.

Table 2.1: MSE and MAVD of the MTBFa-Regression Model

Data	System 40 ($\log(t)$, t is in seconds)	Project 1 ($\log(t)$, t is in seconds)	Project 5 ($\log(t)$, t is in seconds)
MSE	0.0653	0.1074	0.0483
MAVD	0.1660	0.2312	0.1588

Comparison of these values with the corresponding values for the models discussed in section (2.1) will be given in section (2.5). One of the major advantages of this model lies in the simplicity of computing and comprehending while substantially reducing MSE and MAVD.

Instead of assuming a PLP model, we suppose the natural logarithm of data, $\ln(t)$, follows a linear relationship, that is, $Y = \ln(MTTF) = a \ln t + b$. Then, the question becomes what is the corresponding intensity function. To answer this question, we will introduce the following. Let T be a continuous random variable

(r.v) on $(0, \infty)$ denoting failure time. The mean residual time (MRT) is the average time to the next failure, given that no failure occurs up to time t, and is defined by

$$m(t) = E(T - t \,|\, T > t).$$

The following result from Bartoszynski [4] gives a relationship between MRT and the reliability function.

We use the following theorem to find the intensity function $\nu(t)$.

Theorem 2.3.1 *Let T be a random variable of continuous type, with density $f(t)$ and CDF ($F(t)$), if we assume that T is nonnegative, so that $f(t) = F(t) = 0$ for $t < 0$. Then*

$$E(T) = \int_0^\infty (1 - F(t))dt = \int_0^\infty R(t)\, dt \tag{2.3.17}$$

and the MRT is

$$m(t) = \frac{\int_t^\infty R(u)\, du}{R(t)}. \tag{2.3.18}$$

From the assumed linear relationship of equation (2.3.11),

$$Y = \ln(MTTF) = a \ln t + b, \tag{2.3.19}$$

we get

$$MTTF = e^{a \ln t + b} = e^{a \ln t} e^b = e^b t^a.$$

Now, equating the MRT with MTTF in order to find the intensity failure function, $v(t)$, $MTTF = m(t)$, we have

$$\begin{aligned} \frac{\int_t^\infty R(u)\, du}{R(t)} &= e^b t^a, \\ e^b t^a R(t) &= \int_t^\infty R(u)\, du \end{aligned} \tag{2.3.20}$$

Call $k = e^b$ for simplicity.

By differentiating (2.3.20) and using the following formula

$$\frac{d}{dt}\left\{\int_t^\infty R(u)\,du\right\} = -\frac{d}{dt}\left\{\int_\infty^t R(u)\,du\right\} = -R(t),$$

we obtain the following:

$$akt^{a-1}R(t) + kt^a\frac{d}{dt}R(t) = -R(t)$$

or

$$\begin{aligned}\frac{d}{dt}R(t) + \left(\frac{1+akt^{a-1}}{kt^a}\right)R(t) &= 0,\\ \frac{d}{dt}R(t) + (k^{-1}t^{-a} + at^{-1})R(t) &= 0\end{aligned} \tag{2.3.21}$$

Since,

$$v(t) = \frac{f(t)}{R(t)},$$

where

$$f(t) = \frac{dF(t)}{dt} = \frac{d(1-R(t))}{dt} = -\frac{dR(t)}{dt}.$$

Now from (2.3.21),

$$-\frac{dR(t)}{dt} = [k^{-1}t^{-a} + at^{-1}]R(t).$$

Thus,

$$\begin{aligned}v(t) = \frac{f(t)}{R(t)} &= \frac{-\frac{dR(t)}{dt}}{R(t)}\\ &= \frac{(k^{-1}t^{-a} + at^{-1})R(t)}{R(t)}\\ &= (k^{-1}t^{-a} + at^{-1}),\ t > 0.\end{aligned}$$

Hence, if we assume the model, $Y = \ln(MTTF) = alnt + b$, then the resulting intensity function will be

$$v(t) = \frac{1+akt^{a-1}}{kt^a}, \quad t > 0. \tag{2.3.22}$$

2.4 Linear Regression Approach

Encouraged by the results of section (2.3), the question becomes what happens if we take directly a simple linear regression model, instead of assuming PLP model. That is, we consider the basic model as

$$TBF = b + at + \epsilon,$$

where TBF (time between failure) and t (time of failure) denote the dependent and independent variables respectively, and a and b are parameters that need to be estimated and ϵ represents the error term with mean zero. Now, the least squares estimates of the parameters a and b are given by

$$\hat{a} = \frac{\sum_{i=1}^{n}(t_i)(TBF_i) - \left(\sum_{i=1}^{n} t_i\right)\left(\sum_{i=1}^{n}(TBF_i)\right)/n}{\sum_{i=1}^{n} t_i^2 - \left(\sum_{i=1}^{n} t_i\right)^2/n} \tag{2.4.23}$$

and

$$\hat{b} = \overline{TBF} - \hat{a}\bar{t}, \tag{2.4.24}$$

where $\overline{TBF}$ denotes the average. Thus, the following prediction equation represents the estimating mean time between failures

$$M\hat{TBF} = \hat{b} + \hat{a}t. \tag{2.4.25}$$

For this simple linear model (we will call this SRRM1 model), MSE and MAVD are given in the Table (2.2).

Table 2.2: MSE and MAVD of the SRRM1 Model

Data	System40 $\log(t)$, t is in seconds	Project 1 $\log(t)$, t is in seconds	Project 5 $\log(t)$, t is in seconds
MSE	2.56	1.80	2.71
MAVD	1.24	0.95	1.33

Though these values are higher than the values obtained in Table (2.1) for the MTBFa-regression model, the advantage is that we do not assume the power law process model. Instead a simple linear regression is assumed. We will observe in section (2.5) that these values are still comparable to the corresponding values obtained in the literature described in section (2.1).

As in section (2.3), one of the questions that can come up becomes what should the intensity function corresponding to the simple linear regression model be. For the linear model, we will derive the intensity function. We will once again use the mean residual time (MRT).

For a simple linear regression model,

$$MTTF = at + b \tag{2.4.26}$$

equating the MRT with MTTF in order to find the intensity failure function $v(t)$,

$$MTTF = m(t),$$

we have

$$\frac{\int_t^\infty R(u)\,du}{R(t)} = at + b, \tag{2.4.27}$$

following the same steps as in section (2.3), we obtain the intensity function corre-

sponding to the simple linear model as

$$v(t) = \frac{a+1}{at+b}, \quad t \geq 0, \quad t \neq -\frac{b}{a}. \tag{2.4.28}$$

It should be noted that in a simple linear regression model,

$$MTTF = at + b,$$

for $t = -(b/a)$ leads to $MTTF = 0$.

Remarks:

(1) If the slope of the regression line is positive, then the MTBF increases. Let k be the desired optimal time of satisfactory operation (specified by the design). The testing of the software will be terminated when $MTBF = k$.

(2) If the slope of the regression line remains negative for a certain software failure data, then we have to discard the system.

2.5 Successive Prediction Using Regression

One of the main objectives of a model is to predict the software failure ahead of time. This can be done by using the two models described in Sections (2.3) and (2.4). Here we describe another method without using the power law process or the power law process through regression, but our prediction task is achieved by doing repeated one-step ahead predictions by using the estimated parameters of the past failure data, which we call the iterative (successive) prediction regression method. This method is a variation of the SRRM1 model in the way that we can predict time to failure (PTBF) directly.

The successive prediction method works as follows: it predicts only the next failure time one step ahead by applying the linear regression model with the iterative estimated parameters from past failure data. That is we apply the linear regression line for computing the estimated parameters, then by using the parameters at every

single failure time successively, we found the predicted time to next failure, then we computed the error. Follow this pattern to find all the predicted values of failure time, then compute MSE and MAVD.

The predictive performance of this method is assumed by computing mean squared error (MSE) and mean absolute error (MAVD). We are listing the results of this model in Table (2.3). These results will be compared with other models in Table (2.8) at the end of this chapter.

Table 2.3: MSE and MAVD of the Successive Recurrence Regression Model

Data	System 40 $\log(t)$, t is in seconds	Project 1 $\log(t)$, t is in seconds	Project 5 $\log(t)$, t is in seconds
MSE	4.35	2.30	3.01
MAVD	1.69	1.20	1.40

2.6 Model Validation

In this section, we will present some results related to validation of the assumptions made in Section (2.3). First we will give some quantile quantile plots (qqplots), and then give some goodness of fit results.

2.6.1 Quantile-Quantile Plot of TBF and Predicted Values

Our objective is to verify if the actual times between failure and the predicted values of TBF of the models specified in this paper are coming from the same assumed distribution. For this purpose, we applied the $qqplot(MTBF, TBF)$; this displays a quantile-quantile plot of the two samples. If these samples are coming from the same distribution, then the plot will be linear.

After applying the *qqplot* for the PLP and linear regression models by using System 40, Project 1, and Project 5 data sets, we notice that the actual TBF fits

the linear regression model better than the PLP model. We can see clearly by looking at the following graphs in Figure (2.1).

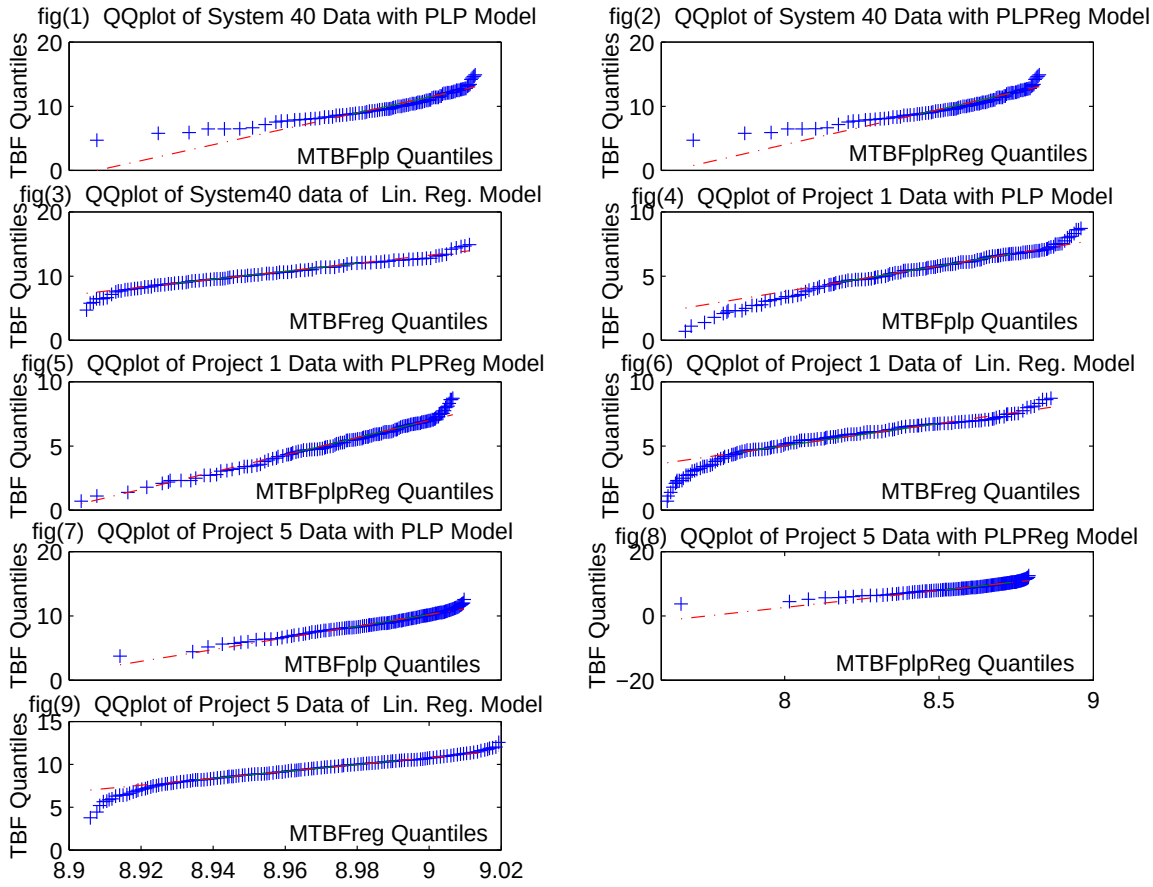

Figure 2.1: Quantile-Quantile Plot of TBF and Predicted Values

Figure (2.1) contains nine sub-figures, which demonstrate the validity of the models. The *qqplot* of System 40 in Fig. (1) and Fig. (2) are very closely the same, while Fig. (3) for the linear model fits the data better.

Project 1 data: Fig. (4) and Fig. (6) for the PLP and linear regression models behave the same, while in Fig. (5) for the PLP_Reg models seem to fit the data better. Project 5 data: Fig.(7) and Fig. (8) are very close, while Fig. (9) fits most of the data except for lowest quartiles. The quantile quantile plot indicates that the samples failure data are coming from the PLP, PLPReg, and the linear regression distributions.

2.7 The Goodness-of-Fit-Test for the PLP Process Model and the Regression Approach

Now we will test the adequacy of the power law process model by a goodness-of-fit test. Consider the hypothesis,

H_0: The power law process is the correct model.

H_1: The power law process is not the correct model.

The ratio power transformation is used for constructing the goodness-of-fit test. If $\Lambda(t)$ is the expected number of failures before time t, then

$$R_i = \frac{\Lambda(t_i)}{\Lambda(t_n)}, i = 1, 2, \ldots, n$$

are distributed approximately as order statistics from a uniform distribution over the interval $[0, 1]$, Rigdon et al. [52].

For the power law process of intensity function $v(t) = \lambda\beta t^{\beta-1}$

$$\Lambda(t) = \int_0^t v(t)\, dx = \int_0^t \lambda\beta x^{\beta-1} dx = \lambda t^{\beta}.$$

Then, the ratio power transformation is

$$R_i = \frac{\Lambda(t_i)}{\Lambda(t_n)} = \frac{\lambda t_i^{\beta}}{\lambda t_n^{\beta}} = \left(\frac{t_i}{t_n}\right)^{\beta}, \quad i = 1, 2, 3, \ldots, n-1,$$

But in most cases, the parameter β is unknown and must be estimated by using the observed failure time data. We used the MLE estimates $\hat{\beta} = n/\left(\sum_{i=1}^{n-1} \log(t_n/t_i)\right)$, but it is biased. The unbiased estimator value $\bar{\beta}$ of β is

$$\bar{\beta} = \frac{n-2}{n}\hat{\beta} = \frac{n-2}{\sum_{i=1}^{n-1} \log(t_n/t_i)}.$$

Then

$$\hat{R}_i = (t_i/t_n)^{\beta}, \tag{2.7.29}$$

The test statistic for the Cramer-von-Mises test denoted by C_R^2 can be applied for the goodness-of-fit test

$$C_R^2 = \frac{1}{12(n-1)} + \sum_{i=1}^{n-1} \left(\hat{R}_i - E\left(\hat{R}_i\right) \right)^2,$$

where $E\left(\hat{R}_i\right) = \frac{2i-1}{2(n-1)}$. Thus

$$C_R^2 = \frac{1}{12(n-1)} + \sum_{i=1}^{n-1} \left(\hat{R}_i - \frac{2i-1}{2(n-1)} \right)^2. \tag{2.7.30}$$

Large values of C_R^2 means that there is an evidence of a departure from the power law process (Rigdon et al. [52]). For certain values of confidence level, the critical values for the Cramer-von Mises goodness-of-fit test with $m = (n-1)$ can be obtained using Table (A 6), page 256, of Rigdon et al. [52]. For $m > 100$, we know the critical values will be smaller than the values given in this table. This information will be used in the decision making for the data sets we have.

Now, we test the adequacy of the linear regression model goodness-of-fit test using the Cramer-von-Mises test. Let

H_0: The linear regression is the correct model.

H_1: The linear regression is not the correct model.

From equation (2.4.28), the intensity function for the simple linear regression model is given by

$$v(t) = \frac{a+1}{at+b}.$$

The expected number of failures before time t is:

$$\begin{aligned} \Lambda(t) = E\left(v(t)\right) &= \int_0^t v\left(x\right) dx \\ &= \int_0^t \frac{a+1}{ax+b}\, dx = \frac{a+1}{a} \ln \left| \frac{at+b}{b} \right|. \end{aligned}$$

Thus

$$\hat{R}_i = \frac{\Lambda(t_i)}{\Lambda(t_n)}, \quad i = \frac{\ln|(at_i + b)/b|}{\ln|(at_n + b)/b|}.$$

The parameters a and b are unknown. We can compute the estimators of a and b by using the least squares method of the observed failure data. The least squares estimators $\hat{a}$ and $\hat{b}$ are unbiased.

Now, the ratio power transformation is:

$$\hat{R}_i = \frac{\ln\left|\left(\hat{a}t_i + \hat{b}\right)/\hat{b}\right|}{\ln\left|\left(\hat{a}t_n + \hat{b}\right)/\hat{b}\right|}, \quad i = 1, 2, \ldots, n. \tag{2.7.31}$$

Thus, the test statistic for the Cramer-Von Mises test is:

$$C_R^2 = \frac{1}{12(n-1)} + \sum_{i=1}^{n-1}\left(\frac{\ln\left|\left(\hat{a}t_i + \hat{b}\right)/\hat{b}\right|}{\ln\left|\left(\hat{a}t_n + \hat{b}\right)/\hat{b}\right|} - \frac{2i-1}{2(n-1)}\right)^2. \tag{2.7.32}$$

Table 2.4: Goodness-of-Fit Test Statistical Values

Software Model	System 40 $n = 101$	Project1 $n = 133$	Project5 $n = 810$
PLP	0.04231272276	0.0121209429	0.008258820298
Simple Linear Regression	0.009400859828	0.007124003038	0.01730674729

We conclude based on this test, both the PLP assumption and the derived linear intensity function assumptions are valid. Thus, not only do we have good error values, but the assumptions are valid as well.

2.8 Comparison of the MSE and MAVD Values

In this section, we present the comparison of our models with that of some relevant models from literature. The error measures MSE and MAVD are used for these comparisons.

Table (2.5) gives the comparison between MLE approach and regression approach with assuming the logarithmic failure data of System 40, Project1, and Project 5 follow PLP, where we use the notations, MTBFa_mle for the PLP model by using the MLE estimates. and MTBFa_reg the based PLP model through regression.

Table 2.5: Comparing MSE and MAVD of MLE and Regression Methods

Model Type	Error Measurement	System 40 $\log(t)$, t is in seconds	Project 1 $\log(t)$, t is in seconds	Project 5 $\log(t)$, t is in seconds
MTBFa_reg	MSE	0.0653	0.1074	0.0483
MTBFa_mle	MSE	23.4327	16.6756	292.5728
MTBFa_reg	MAVD	0.1660	0.2312	0.1588
MTBFa_mle	MAVD	4.7957	3.7663	16.3511

We can observe that the linear regression model, with the PLP assumption, outperforms the MLE method across all of the different data set. In Table (2.6), we see that this continues to be true in comparison with the models suggested in the literature.

We now give results corresponding to simple linear regression (SRRM1) model, where we did not assume the PLP structure to the data. Given that there are virtually no distributional assumptions (except that is required in the least squares method of regression analysis), our results are competitive.

We will make comparison using the following eight software models, which in-

cludes the published models: Singpurwalla and Soyer models (model I, model II, model III, and model IV), Horigome, Singpurwalla, and Soyer model (MTBFhs model), Suresh model (MTBFa / SRGM models), Henry's model (MTBFq model), and the proposed SRRM1 model.

Table 2.6: Comparing MSE for Different Models (System 40)

Model Name	Mean Square Error "MSE"	Mean Absolute Value Difference "MAVD"
Singpurwalla and Soyer Model I	4.92	Not Available (NA)
Singpurwalla and Soyer Model II	12.99	NA
Singpurwalla and Soyer Model III	9.58	NA
Singpurwalla and Soyer Model IV	16.2029	NA
MTBFhs (Horigome, Singpurwalla, & Soyer)	5.19	1.6865
MTBFa / SRGM / Suresh	4.72	1.85
MTBFa / Computed by Henry	4.15	NA
MTBFq / Henry	3.17	1.5677
SRRM1 /(Linear Regression)	2.57	1.2412

As can be seen from Table (2.6), the results illustrate the error measurements of MSE and MAVD of the above models by applying System 40 software failure data set. These results shows that the linear regression model SRRM1 has the minimum of MSE and MAVD error measurement relative to the other models. This indicates that SRRM1 model is better that SRRM1 has better results than previous models mentioned in the Table (2.6). Although, the linear regression method is attractive due to its simplicity and practicability.

In Table (2.7), we compute the percentage reduction of MSE achieved by SRGM, MTBFq, and SRRM1 models. These results shows that SRRM1 linear regression model has the maximum error reduction. It is clearly shown that the SRRM1 regression model has the best error reduction.

Table 2.7: Percentage Reduction Achieved by SRRM1 (System 40)

Model Name	% Reduction Achieved by SRGM / Suresh, and Rao Model	% Reduction Achieved by MTBFq / Henry, Tsokos, and Rao Model	% Reduction Achieved by SRRM1 Model
Singpurwalla and Soyer Model I	4.06	35.57	47.76
Singpurwalla and Soyer Model II	63.66	75.60	80.22
Singpurwalla and Soyer Model III	50.73	66.91	73.17
Singpurwalla and Soyer Model IV	70.87	80.44	84.14
MTBFhs (Horigome, Singpurwalla & Soyer)	9.06	38.92	50.48
MTBFa / SRGM / Suresh & Rao Model	—	32.84	45.55
MTBFa / Computed by Henry	—	23.61	38.07
MTBFq /Henry & Tsokos	—	—	18.93

We used System 40 and Project 1 failure data sets to demonstrate the graphical comparisons of the proposed four software reliability models in this chapter. Figure (2.2) and Figure (2.3), indicate that our proposed models have less MSE and MAVD error measurements than the MLE approach.

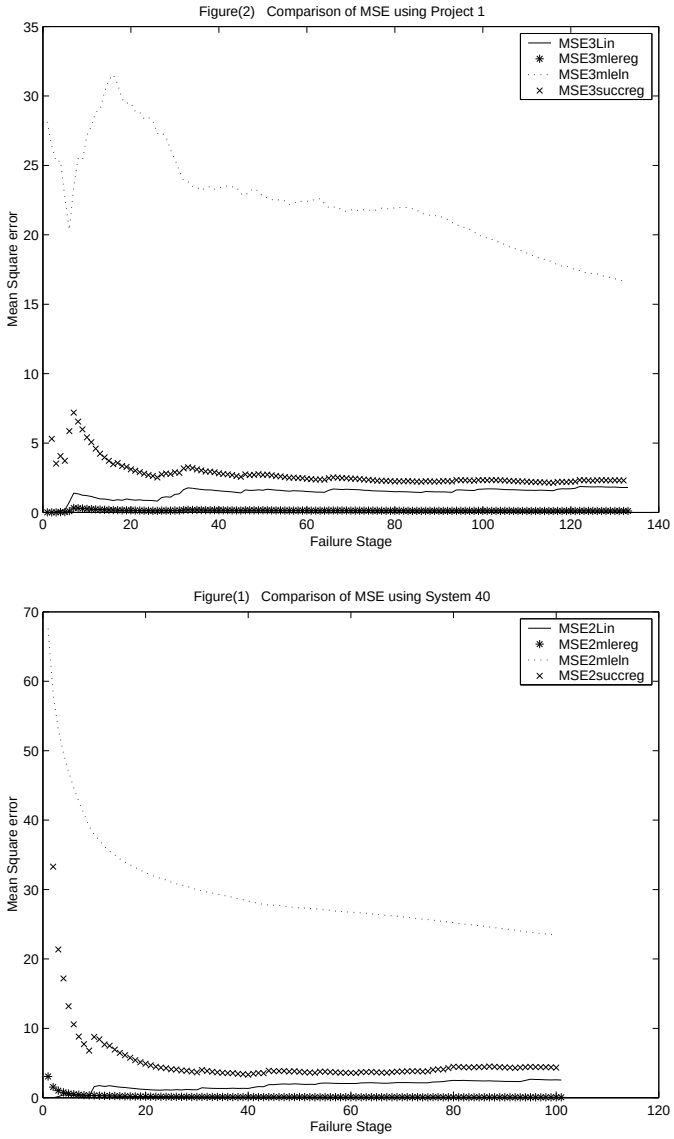

Figure 2.2: MSE Comparison of Software Reliability Models

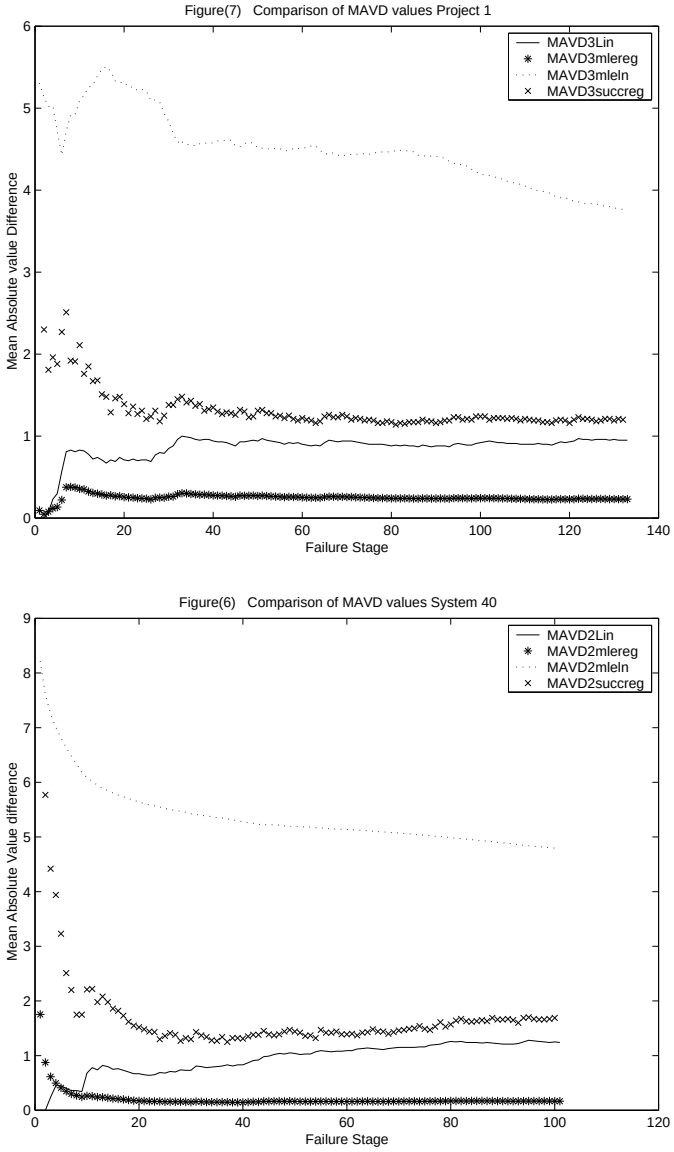

Figure 2.3: MAVD Comparison of Software Reliability Models

These figures illustrates that our proposed models, the PLPReg, the linear regression and the successive recurrence models are better that the PLP based on MLE estimates.

Now, we present some results using different measurement methods in order to compare these models. Table (2.8) shows these methods of measurement (MSE, MAVD, MMRE, MdAR) for the software reliability model for System 40, Project 1, and Project 5 software failure time data sets.

Table 2.8: Comparing Predictive Models by Using Various Error Criteria

System 40	PLP_mle	Simple Lin_reg	PLP_Reg	Successive Recurrence
MSE	23.4327	2.5657	0.0653	4.3483
MAVD	4.7957	1.2370	0.1660	1.6919
MMRE	2.1000	0.1320	0.0762	0.1737
MdAR	4.7961	1.0701	0.1375	1.3917
Project 1				
MSE	16.6756	1.7965	0.1074	2.3011
MAVD	3.7663	0.9479	0.2313	1.2164
MMRE	0.7424	0.2733	0.1704	0.3298
MdAR	4.1110	0.6596	0.1628	1.0105
Project 5				
MSE	292.6700	2.7122	0.0483	3.0068
MAVD	16.3511	1.3260	0.1588	1.4071
MMRE	1.8385	0.1679	0.0778	0.1714
MdAR	15.2034	1.1517	0.1343	1.2794

Based on the observations made in Nyrtveit [46], the standard MLE method under the PLP assumption does not perform well and the successive recurrence model for Project 1 data set is also not a good model. In general, the models introduced in this paper, the linear regression, PLP_Reg, and the successive recurrence models used for prediction perform better in comparison to MLE method for prediction. Overall, for the data sets considered, the PLP regression method outperforms all other models.

2.9 Conclusion

The power law process has proved to be a good model for repairable systems, from which the maximum likelihood estimators can be found through the likelihood function of the failure times. In this study, we presented regression procedure for estimating the parameters of the power law process, which resulted in much-improved results in terms of the error measure MSE and MAVD than the MLE-based method. In addition, for the data sets considered here, we have shown that the simple linear regression model without any NHPP assumptions gives a comparable result to that derived in the literature. Additionally, based on the mean magnitude of relative error values, PLP regression and simple linear regression models perform well. This suggests that, in software reliability modeling, the first step should be to consider a simple linear model for software reliability prediction. The error measures resulting from any other model then could be compared to the error measure resulting from the simple linear regression model. A comparison of error measures with some of the models from the literature suggests that the linear regression models suggested here are competitive and in some cases outperforms. If the assumptions of the PLP model can be validated then the resulting linear regression model, by far, outperforms all the other models compared.

Chapter 3

Non-Parametric Regression Models for Software Reliability

3.1 Introduction

In this chapter, we analyze some of the software failure data using two specific non-parametric methods: namely, monotonic regression and rank regression.

In [39], it has been demonstrated that the linear regression models outperform other popular models from literature in terms of the predictive accuracy of the Mean Time Between Failure (MTBF) estimates. These estimates were measured using the Mean Square Error (MSE) and Mean Absolute Value Difference (MAVD). However, for some of the software failure data, certain parametric assumptions, such as the normality assumption, do not hold. As a result, the inferential procedures described in [39] and many other references may not be applicable and hence may not be accurate. Also, the assumption of uncorrelated errors is more often violated when one uses software failure data, since it is collected over time. There are many robust alternatives that have been developed in the last decades to deal with these situations. See (That is [23], [16] or Hampel et al., [55], [18], and [19]). One of the alternative ways is to model the regression function non-parametrically so as to let the data decide on the functional form. Silverman [61] states that "An initial non-parametric estimate may well suggest a suitable parametric model (such as a linear regression), but nevertheless will give the data more of a chance to speak for themselves in choosing the model to be fitted". In these cases, the fitted values determined non-parametrically are superior to the fitted values obtained from a parametric model. A

simple procedure which involves replacing the data by their corresponding ranks has been used in regression by Iman and Conover [25]. Also, non-parametric estimates of the regression coefficients of a linear regression model were proposed by [28] and [26] by using the Rank-based estimates. In this method, instead of minimizing the least square errors, the regression coefficients are estimated by minimizing the dispersion of the residuals consecutively. These estimates are based on the Wilcoxon scores and the generalized Wilcoxon-Mann-Whitney tests. Non-parametric procedures make less stringent demands on the data and normality is not required to make inferences about the predicted next failure time.

In this work, we apply two nonparametric methods: monotonic regression and rank regression.The main idea of the monotone rank regression approach is to use the rank transformation in regression problems. If there are observations of the dependent time between failures Y, we replace these observations by their corresponding ranks, where $R(Y)$ is the assigned rank to the value of time between failures. Similarly, replace each of the independent failure time T_j, $j = 1, 2, ..., k$ by its corresponding ranks. Ties are replaced by assigning average ranks. Then, we perform the least squares regression analysis on these ranks to obtain the predictive regression estimates.

For the rank regression method, we present an asymptotically distribution-free rank based procedure by computing the estimators of the regression parameter β. Let $R_i(\beta)$ denote the rank of the residual as a function of β. The estimator β is the value that minimizes the dispersion $D(Y_i - T_i\beta)$ defined in (3.2.2). These approaches are less sensitive to outliers. They represent an alternate approach for the parametric regression and the least squares approaches in case the parametric assumptions are not appropriate. In addition, the monotone and the rank regression methods limit the influence of outliers and high leverage values, as shown in Section (3.4) and (3.5).

In Section (3.2), we introduce some preliminary notations, definitions and motivation for using non-parametric methods. Much of the analysis of outliers and influential high leverage are discussed mainly in Section (3.3). In Sections (3.4)

and (3.5), we present new models. They are primarily based on rank regression and consist of estimation and diagnostic tools for checking the adequacy of model fit. Mainly two models are presented, a monotonic regression and a rank regression which estimate procedures by using slope estimation in one dimension. Numerical studies and practical examples are demonstrated with the proposed procedures as well. In Section (3.6), we compare the predictive errors, mean square errors (MSE) and mean absolute value difference (MAVD), and the minimum values of the convex function for the new models with those of the least squares models. We conclude the chapter in Section (3.7).

3.2 Preliminaries

In this section, first, we describe the motivation of using the nonparametric approach. Then, we introduce some important terms and definitions.

Now, we compute some descriptive statistics of two failure data sets, and then verify if there is any linear relationship between the failure time and the time between failures. We can visually verify the histogram and the scatter plot of the two variables to ensure that the underlying relationship is linear. However, the correlation is one of the most useful in describing the degree of relationship between the failure time and the time between failures. The correlation coefficient is nothing more than a quantitative estimate of the relationship and it measures the strength of the linear relationship between two variables.

$$\rho = \frac{n\sum_{i=1}^{n} t_i y_i - \sum_{i=1}^{n} t_i \sum_{i=1}^{n} y_i}{\sqrt{(n\sum_{i=1}^{n} t_i^2 - (\sum_{i=1}^{n} t_i)^2)(n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i^2)^2}}$$

Once we have computed a correlation for the data set, we can determine the probability that the observed correlation occurred by chance, by conducting a significance test:

Null Hypothesis $H_0 : \quad \rho = 0 \quad$ versus $\quad$ Alternate Hypothesis: $H_1 : \quad \rho \neq 0.$

We will use the Spearman's rho and applying the hypothesis test to two data sets System 40 and Project 1. The coefficient correlation between the failure time and the time between failures of System 40 is $\rho = 0.597$. In this case, we are testing the hypothesis. The test shows that the correlation is significant at the 0.01 level (2-tailed).

The test shows that we have to reject the null hypothesis that $\rho = 0$ at level 0.01. Therefore, the underlying relationship is linear. Similarly for Project 1, the correlation is $\rho = 0.544$ and the test shows that we should reject the null hypothesis that $\rho = 0$ at level 0.01. Thus, we can assume that the relationship between the failure data and the time between failures is linear.

A similar test for relationship between the ranks of failure time and the time between failures results in the following: The correlation between ranks for system 40 is $\rho = 0.608$. The result is to reject the null hypothesis that the correlation is zero. The ranks correlation of Project 1 failure data is $\rho = 0.581$ and the conclusion is to reject the null hypothesis. We have fairly good evidence for proceeding with the linear regression and the rank regression for System 40 and Project 1 data sets.

Now, we introduce some important terms and definitions. Let $T_1, T_2, ..., T_n$ be the failure times of the software, and let Y be the random variable representing the time to the next failure. Consider the following linear regression model for software failure prediction:

$$Y_i = \alpha + \beta t_i + \epsilon_i, \ 1 \leq i \leq n, \tag{3.2.1}$$

where α is the intercept parameter, β is the parameter representing the regression coefficient, and ε_i represents the random errors.

Definition 3.2.1 [20] Let $D(.)$ be a measure of variability that satisfies the following two properties:

1. $D(T + a) = D(T)$
2. $D(-T) = D(T)$

Then $D(.)$ is called an even location-free measure of variability.
Thus, $D(Y_i - \alpha - \beta T_i) = D((Y - \beta T_i) - \alpha) = D(Y - \beta T_i)$.
Hence, the intercept α has no effect on the measure of variability and we only need to analyze the residuals $Y_i - \beta T_i$.

Definition 3.2.2 Assume that $D(.)$ satisfies Definition (3.2.1) and let $\hat{\beta}$ be the value that minimizes $D(Y_i - \beta T_i)$.
The corresponding minima is denoted by $V = D(Y_i - \hat{\beta} T_i)$

Definition 3.2.3 The rank regression estimator $\hat{\beta}$ is the value of β that minimizes the sum:

$$D(\beta) = \sum_{i=1}^{n} R_i^c(\beta)(Y_i - \beta T_i) \tag{3.2.2}$$

where

$$R_i^c(\beta) = R_i(\beta) - \frac{(n+1)}{2}, \tag{3.2.3}$$

are the centered ranks or mid-ranks and $R_i(\beta)$ are the ranks of the residuals $Y_i - \beta T_i$.

In order to motivate the rank regression and to provide some insight into the distribution problem, we recall the least squares method (for details see [60]). In the least square regression case, long tailed error distributions have effects which impair the efficiency of the least squares estimates. A single outlier observation may change the regression estimates. Additionally, outliers are easier to spot in the simple location case than in the least square case. Therefore, robust alternatives to the least squares method are needed.

The purpose of the rank regression is to suggest a simple modification of the least squares method which yields an even location-free measure of dispersion. Consider the residual $\varepsilon_i = Y_i - \beta T_i - \alpha$, in the least squares regression we have $\sum_{i=1}^{n} \varepsilon_{(i)}^2 = \sum_{i=1}^{n} \varepsilon_{(i)} \varepsilon_{(i)}$, where ε_i is the ith ordered residual from a symmetric distribution centered at zero. For the rank regression, replace the residuals by its ranks. Let $R_i(\beta)$, $R_i^c(\beta)$,$i = 1, ..., n$, respectively be the ranks and centered ranks of the

residuals, $Y_i - \beta T_i$ as a function of β. Hence, the dispersion function $D(\beta)$ is a linear combination of ordered residuals. For the rank procedure, the residuals are no longer squares as the least squares method, but are weighted according to their ranks. This helps to reduce the effects of outliers and makes it desirable for heavy tailed residual distributions (see [24]).

It follows that $\sum_{i=1}^{n} R_i^c(\beta) = 0$,

and

$$R_1^c(\beta) \leq R_2^c(\beta) \leq ... \leq R_n^c(\beta) \quad \text{(monotone)},$$

and

$$R_1^c(\beta) = -R_{n-i+1}^c(\beta) \quad \text{(antisynmmetric)}$$

Now, we introduce the error measure criteria that we will use in lieu of least squares.

Theorem 3.2.4 *$D(\beta)$ defined in equation (3.2.2) is indeed an even location-free measure of variability.*

Proof. From Definition (3.2.1), it is enough to prove that

1. $D(\beta, \alpha) = D(\beta)$,
2. $D(-\beta) = D(\beta)$.

1. Show that $D(\beta, \alpha) = D(\beta)$

$$\begin{aligned} D(\beta, \alpha) &= D((Y_i - \beta T_i) - \alpha) \\ &= \sum_{i=1}^{n} R_i^c(\beta)(Y_i - \beta T_i - \alpha) \\ &= \sum_{i=1}^{n} R_i^c(\beta)(Y_i - \beta T_i) - \sum_{i}^{n} R_i^c(\beta)\alpha \\ &= \sum_{i=1}^{n} R_i^c(\beta)(Y_i - \beta T_i) - \alpha \sum_{i=1}^{n} R_i^c(\beta) \\ &= \sum_{i=1}^{n} R_i^c(\beta)(Y_i - \beta T_i) = D(\beta) \end{aligned}$$

2. Show that $D(-\beta) = D(\beta)$

$$\begin{aligned} D(-\beta) &= D(-Y_i - \beta T_i) \\ &= \sum_{i=1}^{n} R_i^c(-\beta)(-Y_i + \beta T_i) \\ &= \sum_{i=1}^{n} (-1) R_i^c(-\beta)(Y_i - \beta T_i) \\ &= \sum_{i=1}^{n} R_i^c(\beta)(Y_i - \beta T_i) = D(\beta) \end{aligned}$$

Note that since $\sum_{i=1}^{n} R_i^c(\beta) = 0$, we have an even and location-free measure of dispersion.

■

3.3 Detecting Outliers and Leverage Values

It has been shown in [39] that the least squares methods are very sensitive to outlier values. Even if a single observation is omitted, it has a large impact on the regression results. In regression analysis, outliers, or over-influential observations represent observations that need careful examination. The simple residuals are the most commonly used measures for detecting outliers. There are two common ways to calculate the standardized residual for the ith observation. One uses the residual mean square error from the model fitted to the full data set (internally studentized residuals). The other uses the residual mean square error from the model fitted to all of the data except the ith observation (externally studentized residuals). The externally studentized residuals follow a t distribution [5]. Least squares regression line rests on $(\bar{y}, \bar{t})$ and prediction is adjusted based on distance of predictor values from the mean. The following formula gives the amount of contribution of the ith observation to its own fitted value. It is useful in determining the influence of each observation.

$$h_i = \frac{1}{n} + \frac{(t_i - \bar{t})^2}{S_{tt}} = \frac{1}{n} + \frac{(t_i - \bar{t})^2}{\sum_{i=1}^{n} (t_i - \bar{t})^2} \tag{3.3.4}$$

Note that:

$$h_i \geq \frac{1}{n} \text{and} \sum_{i=1}^{n} h_i = 2 \tag{3.3.5}$$

The standardized residuals can provide a useful tool to detect outlier values. A standardized residual that exceeds ± 2 suggests that the residuals are more than 2 standard deviations, above or below the regression line. Assuming normally distributed errors, standardized residuals should fall outside this range only 5% of the time [65]. It is clear that the further the T value from its mean, the larger the leverage value. We used $h_i \geq 6/n$ in our computations as a threshold condition to obtain leverage observations. A good leverage point is a point that is unusually large or small among the observations and is not a regression outlier. The point is relatively removed from the majority of the observations, but reasonably close to the regression line. In project 1 data set, there are two outliers (0.69, and 1.10) and three leverage values (0.69, 1.10, and 1.39). Thus, we can consider the observation 1.39 as a good leverage point. Additionally, in System 40, there are no outliers but one leverage point that is considered to be a good leverage value. A bad leverage point is a point that has an unusually large residual corresponding to some regression lines. This point is situated far from the regression line of the bulk observations data. A bad leverage point is an outlier among all observations as well. Bad leverage points can rapidly affect the estimated value of the slope. Such an effect has been seen in the case of Apollo 8, with the leverage value 33. The normality test reveals that Apollo 8 data is not normal. Project 1 and Project 5 have outliers and slightly depart from the normal distribution. In such cases, it can substantially reduce our ability to detect a true association between the failure time and time between failures. However, the outliers among the failure time T values inflate the sample variance s_t^2 , and decrease the standard error of the least squares estimate of the slope. This suggests that outlier points are beneficial in terms of increasing our ability to detect regression

lines which have a nonzero slope. However, there is another concern that must be taken into consideration. If the T value is a leverage point and the time between failures is an outlier, then we have a regression outlier that might completely distort how the bulk of the points are related. In a similar manner, we can fail to detect a situation where the slope differs from zero. This occurs not because the slope is indeed zero, but because regression outliers mask an association among the bulks of the points under study. Also, the error measurements of the least squares regression line are not resistant to outliers as we have seen in the following tables described in [39] for the Apollo 8, System 40, Project1, and project 5 data sets.

Now, we demonstrate in Table (3.1) the effect of outlier and leverage values in Apollo 8 data on the error measurements of the software reliability regression model of the first degree (SRRM1), using the least square procedure.

Table 3.1: The Effects of Outlier and Leverage Values on SRRM1(Apollo 8)

Model	Measurements Criteria	Complete Data	Data without Leverage Values '91'	Data without Outliers '33, 91'
SRRM1	MSE	151.9420	29.3014	7.1063
SRRM1	MAVD	6.3229	3.1597	1.9085
SRRM1	MMER	0.5914	0.57	0.5147
SRRM1	MMRE	2.2680	1.01	0.7385
SRRM1	MdAR	2.2045	2.18	2.0853
SRRM1	V(LSR)	341.25	11.64	73.6

Let's illustrate this point by using the actual data of Apollo 8 software failure times. We have been shown that the least squares regression line has an estimated slope of 0.1132, $MSE = 151.9420$, and $MAVD = 6.3229$ in the case of complete data, while the slope of the regression is -0.0428, $MSE = 7.1063$, and $MAVD = 1.9805$ in the case of removing the two outliers 33 and 91 of TBF. This negative association is missed by the least squares regression line because of the

outliers. In fact the two outliers, 33 and 91, caused problems. Even if we remove the most extreme outlier 91 and keep the other less hazard outlier 33, we get the following results: the slope of the regression line is 0.0124, $MSE = 29.3014$, and $MAVD = 3.1597$. Furthermore, a single outlier can cause some problems if we use the least squares regression method. Beside this disadvantage, the mean square error is improved by removing the outliers. Therefore, the least squares regression line offers no hint of a strong association, even though the mean square error is low compared to previous models. The slope estimator is not resistant to outliers and if there is a small departure from normality, devastating consequences would erupt when trying to use the least squares regression method. In other words, this method is not robust and the slope of the regression line can be extremely sensitive to small changes in the probability curve or to the existence of outliers. The removal of the outliers from Apollo 8 data improves the SRRM1 model by decreasing its MSE and MMRE values. This model is not acceptable for the Apollo 8 effort prediction, even if it has better results than both of Henry [54] and Suresh [64] models.

Table 3.2: The Effects of Outlier and Leverage Values on SRRM1 (System 40)

Model	Measurements Criteria	Complete Data	Data without Leverage Values '4.70'	Data has no Outliers
SRRM1	MSE	2.5657	2.3428	*NoOutliers*
SRRM1	MAVD	1.2370	1.1736	*NoOutliers*
SRRM1	MMER	0.1224	0.1138	*NoOutliers*
SRRM1	MMRE	0.1320	0.1223	*NoOutliers*
SRRM1	MdAR	1.0701	1.0336	*NoOutliers*
SRRM1	V(LSR)	404	400	*NoOutliers*

The System 40 software failure data has no outliers, but it has one leverage value. Table (3.2) demonstrates the effect of even one single outlier on the error measurements of the least squares regression model. The SRRM1 model is very

sensitive to leverage values, too. Since $MMRE = 0.12 \leq 0.25$, applying this model on System 40 data can be considered as acceptable for effort prediction. Therefore, there is no need for us to eliminate the leverage values from System 40 failure data.

Table 3.3: The Effects of Outlier and Leverage Values on SRRM1 (Project 1)

Model	Measurements Criteria	Complete Data	Data without Leverage Values	Data without Outliers
SRRM1	MSE	1.7965	1.6384	1.7209
SRRM1	MAVD	0.9479	0.9195	0.9428
SRRM1	MMER	1.1691	0.1628	0.0.1694
SRRM1	MMRE	0.2733	0.2277	0.2423
SRRM1	MdAR	0.6596	0.6125	0.0.6577
SRRM1	V(LSR)	343	357.65	355

Similarly, one can follow the same discussion for Tables (3.1) and (3.2). Table (3.3) illustrates the effect of outliers and leverage values on error measurements. This model is sensitive to outliers. But in case the outliers or leverage values are removed from analysis, then measurements of $MMRE < 0.25$ imply that SRRM1 is an acceptable regression model for predicting the time between failures of Project 1 data. While in the case of complete Project 1 data, $MMRE = 0.2733 \approx 0.25$, which indicates that the model SRRM1 may be accepted for prediction, depends on the designer of the software. Otherwise, remove the leverage values in order to have a good prediction model. The least squares regression as shown in the previous tables might poorly reflect how the bulk of observations are associated and its nonresistance to outliers. These analysis of the data sets show the need to use more robust methods in lieu of least square method for software reliability analysis. Even though, the regression methods in terms of error measures introduced in [39] gave very good results, there was an issue of small values. This casts doubt about the appropriateness of a simple linear regression model. Since we are dealing with real

life software failure time, it is not acceptable to drop the outliers from analysis. We now utilize two methods, the monotone and rank regression, that are less sensitive to outliers and high leverage values.

3.4 Monotone Regression

Iman, and Conover [25] introduced the idea of rank transform procedure in regression as an alternative method to formulating a nonlinear data [56]. In this section, monotonic regression method for the estimation of predicted time between failure values is proposed as a robust alternative to least square regression. Assume that $E(Y|T)$ increases (at least, it does not decrease) as T increases. Therefore, the regression is monotonically increasing. If $E(Y|T)$ decreases as T increases, then the regression is monotonically decreasing. Since there is a monotonic relationship between T and Y for Apollo 8, Project 1, and System 40 failure data sets, the relationship between T and Y is nonlinear. While in Project 5 data set, the relationship is not strictly monotonic and linear. This encourages us to use these data sets to illustrate the monotonic rank regression. The procedure used in this work is to replace the failure data by their corresponding ranks. Replace the dependent variable Y by its corresponding ranks from 1 to n , where $R(Y_i)$ represents the assigned rank to the ith value of Y . Similarly, replace each of the failure times T with its corresponding ranks $R(T_i)$. Ties are assigned by their average ranks. The monotonic regression procedures depend on the fact that the ranks of these two variables have a linear relationship if the corresponding variables have a monotonic relationship. The linear regression equation based on ranks is given by

$$R(Y_i) = \alpha + \beta R(T_i) + \varepsilon_i \tag{3.4.6}$$

The least squares regression analysis is performed on the ranks of T and Y. The regression equation which expresses $\hat{R}(Y_i)$ in terms of $R(T_i)$ is

$$\hat{R}(Y_i) = (n+1)/2 + \hat{\beta}(R(T_i) - (n+1)/2) + \varepsilon_i \tag{3.4.7}$$

Since these monotonic procedures can be viewed as the usual parametric procedures applied to ranks, but the hypothesis test of the ranked data suggests using the linear regression, it is reasonable to ask ourselves the question of what we gained by using ranks in place of the raw failure data. Our objective is to predict Y rather than the rank of Y, so we have to obtain predicted value of Y from the predicted value of $R(Y)$. Equation (3.4.6) generates the mean predicted values of the ranks of Y once the ranks of T and Y are substituted in the linear regression equation. This method is similar to the least squares in such a way that we are just simply trying to fit a line through the ranks of the failure data points. The main point here is the analysis of Apollo 8, System 40, Project 1, and Project 5 indicate the linear relationship between the failure time and the time between failures.
We will apply the following algorithms on software failure data to obtain the regression curve.

3.4.1 Algorithm of Obtaining the Estimate of $E(Y|T)$ at a Particular Point

In order to estimate the regression of the time between failures Y on T at a particular failure time $T = t_0$, apply the following algorithm [8].

1. Acquire the ranks $R(T_i)$ and $R(Y_i)$ of the T and Y respectively. In case of ties use average of tied ranks.

2. Identify the least squares regression estimators of equation (3.4.6),

$$\hat{\beta} = \frac{\sum_{i=1}^{n} R(T_i)R(Y_i) - n(n+1)^2/4}{\sum_{i=1}^{n} [R(T_i]^2 - n(n+1)^2/4} \tag{3.4.8}$$

$$\hat{\alpha} = (1 - \hat{\beta})(n+1)/2. \tag{3.4.9}$$

3. A rank $R(t_0)$ for t_0 can be acquired by applying the next algorithms:

(a) If t_0 equals one of the observed points T_i, let $R(t_0)$ equal the rank of that T_i.

(b) If t_0 exists between two adjacent values T_i and T_j where $T_i < t_0 < T_j$, interpolate between their respective ranks to get $R(t_0)$.

$$R(t_0) = R(T_i) + \frac{t_0 - T_i}{T_j - T_i}[R(T_j) - R(T_i)] \tag{3.4.10}$$

Note that this "rank" is not necessarily an integer.

(c) If t_0 is less than the smallest observed T or greater than the largest observed T, do not attempt to extrapolate. Information on the regression of Y on T is available within the observed range of T.

4. Substitute $R(t_0)$ for t in Equation (3.4.6) to get an estimated rank $R(y_0)$ for the corresponding value of $E(Y|T = t_0)$,

$$\hat{R}(y_0) = \hat{\alpha} + \hat{\beta}R(t_0). \tag{3.4.11}$$

5. Transform $R(y_0)$ into $\hat{E}(Y|T = t_0)$, an estimate of $E(Y|T = t_0)$, by referring to the observed Y_i as follows.

(a) If $R(y_0)$ equals the rank of one of the observations Y_i , let the estimate $\hat{E}(Y|T = t_0)$ equal observation Y_i.

(b) If $R(y_0)$ is between the ranks of two adjacent values of Y, say Y_i and Y_j where Y_i<Y_j, so that $R(Y_i) < R(y_0) < R(Y_j)$, interpolate between Y_i and Y_j:

$$\hat{E}(Y|T = t_0) = Y_i + \frac{R(y_0) - R(Y_i)}{R(Y_j) - R(Y_i)}(Y_j - Y_i). \tag{3.4.12}$$

(c) If $R(y_0)$ is greater than the largest observed rank of Y, let $E(\hat{Y}|T = t_0)$ equal the largest observed value Y. If $R(y_0)$ is less than the smallest observed rank of Y, let $E(\hat{Y}|T = t_0)$ equal the smallest observed value Y.

3.4.2 The Estimate of the Regression of Y on T

The following procedure is used to get the entire regression curve consisting of all points.

1. Obtain the end points of the regression curve by using the smallest $T^{(1)}$ and the largest $T^{(n)}$ observations in the preceding procedure to obtain $E(\hat{Y}|T = t^{(1)})$ and $E(\hat{Y}|T = t^{(n)})$.

2. For each rank of Y, find the estimated rank of T_i and $\hat{R}(T_i)$ from Equation (3.4.7)

$$\hat{R}(T_i) = [R(Y_i) - \hat{\alpha}]/\hat{\beta}, \quad i = 1, 2, ..., n. \tag{3.4.13}$$

3. Transform each $\hat{R}(T_i)$ to an estimate $\hat{T}_i$ in the manner of the preceding step (5). More Specifically:

   (a) If $\hat{R}(T_i)$ equals the rank of some observation T_j , let $\hat{T}_i$ equal that observed value.

   (b) If $\hat{R}(T_i)$ is between the ranks of two adjacent observations T_j and T_k , where $T_j - T_k < 0$, obtain $\hat{T}_i$ by using the following equation:

$$\hat{T}_i = T_j + \frac{\hat{R}(T_i) - R(T_j)}{R(T_k) - R(T_j)}[T_k - T_i]. \tag{3.4.14}$$

   (c) For all observed ranks of T values, if $\hat{R}(T_i) < \min(R(T_i))$ or $\hat{R}(T_i) > \max(R(T_i))$, then there is no estimate for $\hat{T}_i$.

4. Plot each of the points found in step (3), with Y_i as the ordinate and $\hat{T}_i$ as the abscissa. Also plot the end points found in step (1), with $E(\hat{Y}|T)$ as the ordinate and $T^{(1)}$ or $T^{(n)}$ as the abscissa. All these points are monotonic, increasing if $\hat{\beta} > 0$ and decreasing if $\hat{\beta} < 0$.

5. The estimate of the regression of Y on T is represented by lines joining points in step (4).

3.4.3 Results of Monotone Regression

Now, we apply the monotone regression procedure described in Section (3.4) to some of the software failure data sets.

Table 3.4: Analysis of Software Monotone Regression Method (Apollo 8)

Measurements Criteria	Complete Data	No Outliers (33), and (91)	No Leverage Values (91)
MSE	326.7174	9.8692	42.9201
MAVD	6.9378	2.5810	3.7246
MMER	1.4040	0.5796	0.7987
MMRE	1.2132	0.9779	1.1072
MdAR	2.3256	2.1249	2.4462

The above table (3.4) demonstrates its sensitivity to outliers and leverage values.

Table 3.5: Analysis of Software Monotone Regression Method (System 40)

Measurements Criteria	Complete Data	No Outliers Exist	No Leverage Values
MSE	2.9877	No Outliers exist	2.7345
MAVD	1.3286	No Outliers exist	1.2868
MMER	0.1295	No Outliers exist	0.1249
MMRE	0.1460	No Outliers exist	0.1365
MdAR	1.0331	No Outliers exist	1.0166

There are no outliers in System 40, but one can observe that in Table (3.5) the impact of removing the leverage values on the measurements output is very little. This means that the monotone regression is insensitive to leverage values for System 40 data. At the same time, notice that $MMRE = 0.1460 < 0.25$ for complete System 40 data, indicating that the monotone regression is an acceptable predictive

model for applying System 40. On the other hand, if we remove the leverage values, then $MMRE = 0.1365 < 0.025$. In this situation, we use the complete System 40 data set.

Table 3.6: Analysis of Software Monotone Regression Method (Project 1)

(Project 1)

Measurements Criteria	Complete Data	No Outliers	No Leverage Values
MSE	2.0517	19.6718	16.05
MAVD	1.0592	4.1360	3.77
MMER	0.0408	2.9756	2.10
MMRE	0.1981	0.7150	0.64
MdAR	0.7494	4.31	3.91

Table (3.6) The Monotone regression method is very good for complete data. MSE and MAVD values are small, and MMRE=0.1981. This information proves that the Monotone regression for project 1 data set is an acceptable prediction model. While if we remove the outliers or the leverage values, then MMRE=0.7150 and MMRE=0.64 respectively lead us to decide that this model is not acceptable for prediction in the absence of outliers and leverage values. Since we are dealing with real life software data, it is crucial to keep these values.

Table 3.7: Analysis of Software Monotone Regression Method (Project 5)

Measurements Criteria	Complete Data	No Outliers	No Leverage Values
MSE	2.7975	2.4576	2.3085
MAVD	1.3348	1.2819	1.2525
MMER	0.0148	0.0109	0.0090
MMRE	0.0597	0.0451	0.0399
MdAR	1.1471	1.1108	1.1

Since the value $MMRE = 0.0597 \leq 0.25$ in Table (3.7), it is significant that the monotone regression model is an acceptable model for prediction for complete data. The percent effect of outliers is 12.15 percent, while the percentage effect of leverage values is 17.48 percent. This result shows that the monotone regression is insensitive to outliers for Project 5.

The following graphs represent the lines (curves) obtained using linear regression and monotonic regression methods, along with the scatter plot of the data values.

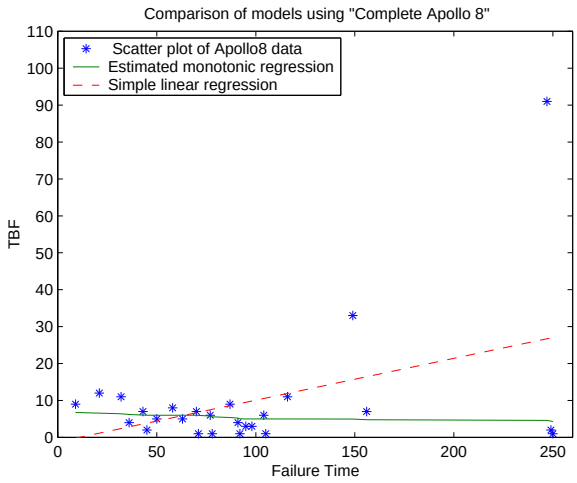

Figure 3.1: Analysis of Monotone Regression (Apollo 8, Complete Data)

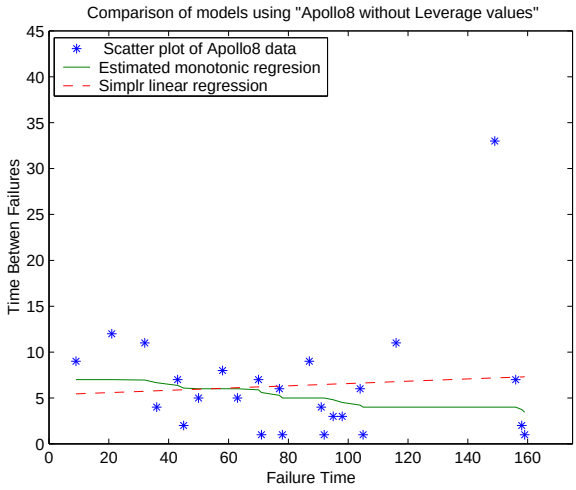

Figure 3.2: Analysis of Monotone Regression (Apollo 8, No Leverage Values)

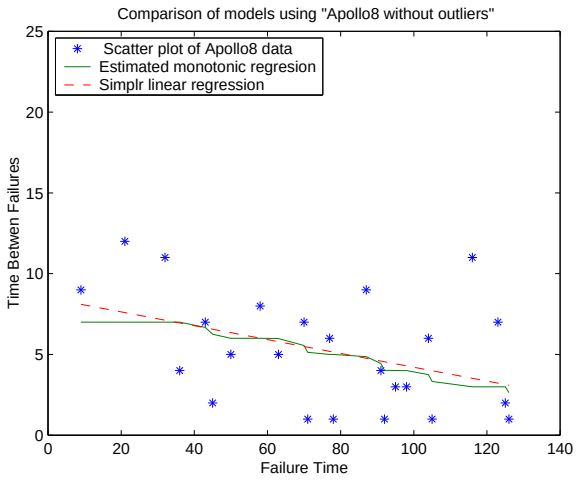

Figure 3.3: Analysis of Monotone Regression (Apollo 8, No Outliers)

3.5 Rank Regression Based on Robust Slope Estimation

For the monotone regression, we were investigating errors by assuming that the ranks of the failure data and the time between failures were linearly related. However, in this section we are taking into consideration the ranks of the residuals. Ranking the residuals reduces the effects of outliers as were discussed by Huber [23], [24]. Conovar, Iman, [25], Sawyer [57], and Theil [66], [67], and [68] introduced the rank regression method. In this section, we adopt this research to study the software reliability models. Recall equations (3.2.1), (3.2.2), (3.2.3).

Since

$$\sum_{i=1}^{n} R_i^c(\beta) = \sum_{i=1}^{n} (R_i(\beta) - \frac{(n+1)}{2}) = 0$$

then

$$D(\beta, \alpha) = \sum_{i=1}^{n} R_i^c(\beta).(Y_i - \beta T_i - \alpha)$$

$$D(\beta, \alpha) = \sum_{i=1}^{n} R_i^c(\beta).(Y_i - \beta T_i) = D(\beta). \tag{3.5.15}$$

Equation (3.5.15) represents the sum that involves the residuals $(Y_i - \beta T_i - \alpha)$ of the regression equation (3.2.1). Instead of equation (3.5.15), the least-squares estimators (LSE) of α and β are found by minimizing:

$$C(\beta, \alpha) = \sum_{i=1}^{n} (Y_i - \alpha - \beta T_i)^2 \tag{3.5.16}$$

The LSE of $\hat{\beta}_c$ from equation (3.5.16) is:

$$\hat{\beta}_c = \frac{\sum_{i=1}^{n} (Y_i - \bar{Y})(T_i - \bar{T})}{\sum_{i=1}^{n} (T_i - \bar{T})^2} \tag{3.5.17}$$

Equation (3.5.15) can be minimized for β , but the computations are very difficult.

However, a natural generalization of the LSE $\hat{\beta}_c$ in equation (3.5.17) is to minimize

$$E(\beta, \alpha) = \sum_{i=1}^{n} |Y_i - \alpha - \beta T_i| \tag{3.5.18}$$

instead of $C(\beta, \alpha)$ in equation (3.5.16). But the parameter estimates $\hat{\beta}_1$, and $\hat{\alpha}_1$, computed by minimizing the function $E(\beta, \alpha)$ in equation (3.5.18), are not easy to analyze for the proposed rank regression model.

This leads us to use Theil's statistics procedure [21], [66], [67], and [68] to compute the slope estimator of equation (3.2.1).

Theil's estimator is:

$$\hat{\beta}_{TH} = median\,\{S_{ij}\} \quad 1 \leq i < j \leq n \tag{3.5.19}$$

where

$$S_{ij} = \frac{(Y_j - Y_i)}{(T_j - T_i)} \quad 1 \leq i < j \leq n$$

Compute $N = n(n-1)/2$ individual sample slope values of S_{ij}.

Let $S^{(1)} \leq S^{(2)} \leq ... \leq S^{(N)}$ be the ordered sample slope values of S_{ij}.

If $N = 2k + 1$ odd, then $\hat{\beta}_{TH} = S^{(k+1)}$, where $k = (N-1)/2$.

If $N = 2k$, then $k = N/2$ and $\hat{\beta}_{TH} = [S^{(k)} + S^{(k+1)}]/2$.

If the values of T_i are equally spaced, then $\hat{\beta}_{TH}$ and the rank regression estimator $\hat{\beta}$ from (3.2.2) can be shown to be asymptotically equally powerful for estimating β [21]. If the T_i are not equally spaced as in the software failure time data sets [40], then the rank-regression estimator $\hat{\beta}$ is asymptotically more powerful, or is more accurate for the same size of failure data [57].

Theorem 3.5.1 *If $R_i^c(\beta) = R_i(\beta) - \frac{n+1}{2}$ is the centered rank of $Y_i - \beta T_i$, prove that $D(\beta) = \sum_{i=1}^{n} R_i^c(\beta)(Y_i - \beta T_i)$ is continuous, piecewise linear, and convex upwards. i.e*

$$k_T = \min\{k : S_k = -Q + \sum_{p=1}^{k} |T_{jp} - T_{ip}|\} > 0. \tag{3.5.20}$$

for $k = k_T$, the rank regression estimator $\hat{\beta}$ is:

$$\hat{\beta} = W_k = \frac{Y_{jk} - Y_{ik}}{T_{jk} - T_i}, \quad if \quad S_{k-1} < 0 < S_k \tag{3.5.21}$$

Equation (3.5.21) corresponds to the case of a unique minimum value for $D(\beta)$.

$$\hat{\beta} = \frac{W_{k-1} + W_k}{2}, \quad if \quad S_{k-1} = 0 < S_k \tag{3.5.22}$$

Equation (3.5.22) corresponds to an interval of minimum values. [21], [57], and [26]

Proof.

i. $D(\beta)$ is piecewise linear function

For simplicity, assume there are no ties among the failure data pairs (T_i, Y_i) . Let $R_a(\beta)$ be the ranks of $Z_a = Y_a - \beta T_a$, for $1 \leq a \leq n$. If there is a solution of $Y_j - \beta T_j = Y_i - \beta T_i$ for $i \neq j$, then the values of Z_a have tied values of (T_a, Y_a) .

$$Y_j - \beta T_j = Y_i - \beta T_i \Leftrightarrow Y_j - Y_i = \beta(T_j - T_i)$$

$$\beta = Z(i,j) = \frac{Y_j - Y_i}{T_j - T_i};\ T_i \neq T_j \tag{3.5.23}$$

Equation (3.5.23) $\Rightarrow$ $\{Z_a\} = \{Y_a - \beta T_a\}$ has no tied values as

$$\beta \neq Z(i,j) = \frac{Y_j - Y_i}{T_j - T_i},\ (T_j \neq T_i)$$

which are at most $N = \frac{n(n-1)}{2}$ in number.

Sort the values of $Z_a = Y_a - \beta T_a$

$(sorted)(Y_a - \beta T_a)$ moves with rate $-T_a$ as β increases.

The sorted values $(sorted)(Y_a - \beta T_a)$ keeps the same relative order as soon as:

$$\beta \neq Z(i,j) = \frac{Y_j - Y_i}{T_j - T_i}.$$

In other words, the ranks $R_i(\beta)$ are constant in each of the $(N+1) = \frac{n(n-1)}{2} + 1$ intervals (W_k, W_{k+1}) for $0 \leq k \leq N$, where W_k are the sorted values $Z(i,j)$, such that it is sorted in an increasing order with

$$W_0 = -\infty, \text{ and } W_{N+1} = +\infty$$

Since the ranks $R_i(\beta)$ are constant in each interval (W_k, W_{k+1}) , then the function

$$D(\beta) = \sum_{i=1}^{n} R_i^c(\beta)(Y_i - \beta T_i)$$

is piecewise linear

$$D(\beta) = A_k + \beta B_k \tag{3.5.24}$$

where

$$W_k < \beta < W_{k+1}$$

$$A_k = \sum_{i=1}^{n} R_i^c(\beta) Y_i$$

and

$$B_k = -\sum_{i=1}^{n} R_i^c(\beta) T_i$$

ii. $D(\beta)$ **is convex upward**

If $\beta < W_1$, then the relative order of $Y_i - \beta T_i$ stays the same as β becomes arbitrarily large and negative. This implies that $Y_i - \beta T_i$ has the same relative order as T_i , or $R(T_i)$. With this said, the slope then the slope of $D(\beta)$ is

$$B_0 = -Q < 0, \quad \text{where} \quad Q = \sum_{i=1}^{n} R_i^c(T) T_i > 0.$$

If $\beta > W_N$, where $N = \frac{n(n-1)}{2}$, then the slope of $D(\beta)$ is

$$B_N = Q > 0, \qquad \sum_{i=1}^{n} R_i^c(T) T_i > 0$$

Therefore $D(\beta)$ has a minimum value, i.e $D(\beta)$ is convex upwards.

iii. $D(\beta)$ **is Continuous**

Now what happens if $W_1 < \beta < W_N$, and β crosses one of the values of $W_k, \quad \ni \ 1 \leq k \leq N$?

If there is a single tie, then:

$$Y_j - \beta T_j = Y_i - \beta T_i$$

$$\text{at} \quad \beta = W_k \quad \text{with} \quad T_i < T_j$$

$$Y_a - \beta T_a = Y_a - W_k T_a + (W_k - \beta) T_a$$

$$if \quad \beta < W_k \Rightarrow Y_j - \beta T_j > Y_i - \beta X_i$$

$$\text{if } \beta > W_k \Rightarrow Y_j - \beta T_j < Y_i - \beta T_i$$

If β is sufficiently close to W_k, then no other values $Y_a - \beta T_a$ lie between the previous two values. Therefore, there exists an integer m such that for sufficiently small $b > 0$, we have:

$$R_j(\beta) = m + 1 \quad \text{and} \quad R_i(\beta) = m \quad \text{for} \quad W_k - b < \beta < W_k \tag{3.5.25}$$

$$R_j(\beta) = m \quad \text{and} \quad R_i(\beta) = m + 1 \quad \text{for} \quad W_k < \beta < W_k + b \tag{3.5.26}$$

and we have no other change in ranks $R_a(\beta)$ $(a \neq i, j)$ in the interval

$$W_k - b < \beta < W_k + b$$

This implies that:

a. $\mathrm{D}(\beta) = \sum_{a=1}^{n} R_a(\beta)(Y_a - RT_a)$ is a continuous function across $\beta = W_k$.

By equations (3.5.24), (3.5.25), and (3.5.26), there is only one change in the ranks of $R_a(\beta)$. This change happens when both ranks $R_i(\beta)$ and $R_j(\beta)$ switch their values at $\beta = W_k$. However, since the coefficients $R_i^c(\beta)$, $R_j^c(\beta)$

have the tied values $Y_j - \beta T_j = Y_i - \beta T_i$ for $D(\beta)$ under the condition $\beta = W_k$, $D(\beta)$ is continuous. We can then follow the same procedures if $\{Y_a - \beta T_a\}$ has multiple tied values at $\beta = W_k$.

b. By assuming $T_i < T_j$ and using equations (3.5.24), (3.5.25), and (3.5.26) as β increases through W_k the change in the slope parameter $D(\beta)$ is

$$\begin{aligned} B_k - B_{k-1} &= -\left(\sum_{a=1}^{n} R_a^c(\beta + b)T_a - \sum_{a=1}^{n} R_a^c(\beta - b)T_a\right) \\ &= -((mT_j + (m+1)T_i - ((m+1)T_j + mT_i)) \\ &= T_j - T_i \end{aligned} \tag{3.5.27}$$

However, consider what happens if we assume $T_i > T_j$.

Then $B_k - B_{k-1} = T_i - T_j$.

In general for all the values of T_i and T_j, we have $B_k - B_{k-1} = |T_j - T_i|$.

Assume that the T_i are increasing, then the slope of $D(\beta)$ increases by:

$T_j - T_i > 0$ at each $\beta = W_k$ for $1 \le k \le N$.

If $\beta < W_1$, then the slope of $D(\beta) = -Q$.

If $\beta > W_N$, then the slope of $D(\beta) = Q$.

For verification, consider the pair (i, j) for $W_k = Z(i, j)$ and (3.5.23), where

$$\{W_k : 1 \le k \le N\} = (sorted)\left\{\frac{Y_j - Y_i}{T_j - T_i} : 1 \le i < j \le n, T_i \ne T_j\right\} \tag{3.5.28}$$

If the values of T_i are strictly increasing, then

$$\begin{aligned} &\sum\sum_{(i,j)} (T_j - T_i) = \sum \sum_{1 \le i < j \le n} (T_j - T_i) = \sum_{j=1}^{n}\sum_{i=1}^{j-1} T_j - \sum_{i=1}^{n}\sum_{j=i+1}^{n} T_i \\ &= \sum_{j=1}^{n} (j - 1 - (n - j))T_j = 2\sum_{j=1}^{n} (j - \frac{(n+1)}{2})T_j = 2Q \end{aligned} \tag{3.5.29}$$

This is another proof that the slope of $D(\beta)$ is $B_N = Q$.

■

Find the minimum value of $D(\beta)$ [57]

Using equation (3.5.28) implies that the slope of $D(\beta)$ in (W_k, W_{k+1}) is:

$$B_k = B_0 + \sum_{p=1}^{k} (B_p - B_{p-1}) = B_0 + \sum_{p=1}^{k} (T_{jp} - T_{ip}) = S_k \tag{3.5.30}$$

Thus, $S_k = B_k$ in equation (3.5.30) is the slope of $D(\beta)$ in (W_k, W_{k+1}).

i. If $S_{k-1} < 0 < S_k$, then $\beta = \hat{\beta} = W_k$ is the unique minimum value of $D(\beta)$.
Resulting in:

$$\hat{\beta} = W_k = \frac{Y_{jk} - Y_{ik}}{T_{jk} - T_i}.$$

ii. $S_{k-2} < 0, \quad S_{k-1} = 0, \quad S_k > 0$

$$\hat{\beta} = \frac{W_{k-1} + W_k}{2}.$$

Note that equation (3.5.21) represents a unique minimum value of $D(\beta)$.
While, equation (3.5.22) corresponds to an interval estimation of minimum values.

3.5.1 Algorithm Criteria for Computing the Slope Estimator $\hat{\beta}$

I. Compute and sort the $N = n(n-1)/2$ individual sample slope value S_{ij}.

$$S_{ij} = \frac{Y_j - Y_i}{T_j - T_i},\ 1 \le \mathrm{i} \le \mathrm{j} \le \mathrm{n} \tag{3.5.31}$$

$$\{W_k : 1 \le k \le N\} = (sorted)\left\{S_{ij} = \frac{Y_j - Y_i}{T_j - T_i} : 1 \le i < j \le n, T_i \ne T_j\right\} \tag{3.5.32}$$

where

$$N \begin{cases} = \dfrac{n(n-1)}{2}, & \text{if } T_i \text{ have no ties} \\ < \dfrac{\mathrm{n(n\ -\ 1)}}{2}, & \text{otherwise} \end{cases}$$

II. Define

$$Q = \sum_{i=1}^{n} R_i^c(T)T_i,\ \mathrm{R_i^c}(T) = R_i(T) - (n+1)/2 \tag{3.5.33}$$

If T_i observations are not constant, then $Q > 0$. Since the failure time data is cumulative $T_1 < T_2 < ... < T_n$, then $R_i(T) = i$, and

$$Q = \sum_{i=1}^{n} R_i^c(T)T_i = \sum_{i=1}^{n} (i - \frac{(n+1)}{2})T_i \quad = \\ \sum_{i=1}^{(n+1)/2} \left| i - \frac{(n+1)}{2} \right| (T_{n+1-i}T_i) > 0. \tag{3.5.34}$$

Use the same argument to obtain

$$D(\beta) = \sum_{i=1}^{c} R_i^c(\beta)(Y_i - \beta T_i \geq 0. \tag{3.5.35}$$

III. If (i_p, j_p) are the integers (i, j) that correspond to $k = p$ in (3.5.32). Define $S_0 = -Q$, and for $k \geq 1$

$$k_T = \min\{k : S_k = -Q + \sum_{p=1}^{k} |T_{jp} - T_{ip}| > 0\} \tag{3.5.36}$$

for $k = k_T$, the rank regression estimator $\hat{\beta}$ is:

$$\hat{\beta} = W_k = \frac{Y_{jk} - Y_{ik}}{T_{jk} - T_{ik}}, \quad if \quad S_{k-1} < 0 < S_k. \tag{3.5.37}$$

$$\hat{\beta} = \frac{W_{k-1} + W_k}{2}, \quad if \quad S_{k-1} = 0 < S_k \tag{3.5.38}$$

$S_N = Q > 0$, $\hat{\beta} \leq W_N$

The function $D(\beta)$ is piecewise linear (linear on each segment (W_k, W_{k+1}) and continuous. The number S_k in equation (3.5.36) is the slope of $D(\beta)$ for $W_k < \beta < W_{k+1}$.

Since S_k is strictly increasing, then the graph of $D(\beta)$ concaves upward. The

minimum value of $D(\beta)$ is either a single value at $\beta = W_k$ at the vertex of the curve, or in a closed interval (W_{k-1}, W_k).

It is clear that from (3.5.36), (3.5.37), and (3.5.38), the estimator $\hat{\beta}$ is a weighted median of N slope estimators $W_k = \frac{Y_{jk}-Y_{ik}}{T_{jk}-T_{ik}}$, and is less sensitive to gross errors than the least squares estimator [21]. The least squares estimator is a weighted average of the S_{ij}'s .

$$\bar{\beta} = \frac{\sum_{i=1}^{n} (Y_i - \bar{Y})(t_i - \bar{t})}{\sum_{j=1}^{n} (t_j - \bar{t})^2}, \tag{3.5.39}$$

where $\bar{t}, \bar{Y}$, are the sample means.

3.6 Results and Comparisons

Table 3.8: Analysis of Software Rank Regression (Apollo 8)

Measurements Criteria	Complete Data	No Outlier Values (33), and (91)	No Leverage Values values (91)
MSE	429.4732	19.4206	73.7114
MAVD	8.2009	3.3299	4.8218
MMER	7.5311	2.0602	0.1770
MMRE	1.5341	NA	NA
MdAR	NA	NA	NA
V(RRS)	245.7090	209.9932	243.2727

The results of Table (3.8) indicate that the minimum value for the convex function is $V(RRS) = 245.7090$ for complete Apollo 8 failure data, $V(RRS) = 209.9932$ for Apollo 8 with no outliers, and $V(RRS) = 243.2727$ for Apollo 8 data with no leverage values.

Table 3.9: Analysis of Software Rank Regression (System 40)

Measurements Criteria	Complete Data	No Outliers Exist	No Leverage Values
MSE	28.56	No Outliers Exist	24.41
MAVD	4.59	No Outliers Exist	4.26
MMER	0.73	No Outliers Exist	0.58
MMRE	0.46	No Outliers Exist	0.42
MdAR	4.44	No Outliers Exist	4.27
V(RRS)	1274.97	No Outliers Exist	1199.35

Similarly, in Table (3.9), $V(RRS) = 1274.97$ for System 40 complete failure data, while $V(RRS) = 1199.35$ for System 40 data set without leverage values. The main result is that the rank regression is insensitive to outliers and leverage values.

Table 3.10: Analysis of Software Rank Regression (Project1)

Measurements Criteria	Complete Data	No Outlier Values	No Leverage Values
MSE	21.8342	21.1284	20.1449
MAVD	3.9773	3.9337	3.8370
MMER	6.7568	0.2167	0.1967
MMRE	0.9864	0.8452	0.7964
MdAR	3.8216	3.7546	3.6655
V(RRS)	1343.90	1334	1305

Results of Table (3.10), the values of the minimum values of the convex function $V(RRS)$ are very closed to each other for the complete Project 1, Project 1 data with no outliers, and to Project 1 with no leverage values. These results indicate that the rank regression is not sensitive to outliers and to leverage values.

Table 3.11: Analysis of Software Linear Regression Method (Apollo 8)

Measurements Criteria	Complete Data	No Outliers Values (33), and (91)	No Leverage Values (91)
MSE	151.9420	7.1063	29.3014
MAVD	6.3229	1.9085	3.1597
MMER	0.5914	0.5147	0.57
MMRE	2.2680	0.7385	1.01
MdAR	2.2045	2.0853	2.18

MSE and $MAVD$ results of Table (3.11) show that the linear regression method is very sensitive to outliers and to leverage values.

Table 3.12: Analysis of Software Linear Regression Method (System 40)

Measurements Criteria	Complete Data	No Outliers Exist	No Leverage Values
MSE	2.5657	No Outliers Exist	2.3428
MAVD	1.2370	No Outliers Exist	1.1736
MMER	0.1219	No Outliers Exist	0.1138
MMRE	0.1320	No Outliers Exist	0.1223
MdAR	1.0701	No Outliers Exist	1.0426

Table 3.13: Analysis of Software Linear Regression Method (Project 1)

Measurements Criteria	Complete Data	No Outlier Values	No Leverage Values
MSE	1.7965	1.7209	1.6384
MAVD	0.9479	0.9428	0.9195
MMER	1.1691	0.1694	0.1628
MMRE	0.2733	0.2423	0.2277
MdAR	0.6596	0.6577	0.6125

Table 3.14: Analysis of Software Linear Regression Method (Project 5)

Measurements Criteria	Complete Data	No Outlier Values	No Leverage Values
MSE	2.7122	2.3911	2.2460
MAVD	1.3260	1.2738	1.2417
MMER	0.1435	0.1370	0.1329
MMRE	0.1679	0.1524	0.1459
MdAR	1.1517	1.1194	1.0949

The linear regression method is very sensitive to outliers and to leverage values.

3.7 Conclusion

Table 3.15: The % Effect of Outlier and Leverage Values of Apollo 8 (MSE,V)

Models	Percentage Effects of Outliers	Percentage effects of Leverage Values
Least Squares Regression	95.32	80.72
Monotone Regression	96.98	86.86
Rank Regression	14.5358	0.9915

Table 3.16: The % Effect of Outlier and Leverage Values of Apollo 8 (MAVD)

Models	Percentage Effects of Outliers	Percentage Effects of Leverage Values
Least Squares Regression	69.8161	50.0277
Monotone Regression	62.7980	46.3144
Rank Regression	59.3959	41.2040

Table 3.17: The % Effect of Outlier and Leverage Values of System 40 (MSE,V)

Models	Percentage Effects of Outliers	Percentage Effects of Leverage Values
Least Squares Regression	Data has no outliers	3.6793
Monotone Regression	Data has no outliers	8.4747
Rank Regression	Data has no outliers	5.9311

Table 3.18: The % Effect of Outlier and Leverage Values of System 40 (MAVD)

Models	Percentage Effects of Outliers	Percentage Effects of Leverage Values
Least Squares Regression	Data has no Outliers	3.2094
Monotone Regression	Data has No Outliers	3.1462
Rank Regression	Data has no Outliers	7.1895

Table 3.19: The % Effect of Outlier and Leverage Values of Project 1 (MSE,V)

Models	Percentage Effects of Outliers	Percentage Effects of Leverage Values
Least Squares Regression	4.21	8.8
Monotone Regression	−858.80	−682.28
Rank Regression	0.7367	2.8946

Table 3.20: The % Effect of Outlier and Leverage Values of Project 1 (MAVD)

Models	Percentage Effects of Outliers	Percentage Effects of Leverage Values
Least Squares Regression	0.7579	3.2105
Monotone Regression	−290.4834	−255.9290
Rank Regression	1.0962	3.5275

Table 3.21: The % Effect of Outlier and Leverage Values of Project 5 (MSE)

Models	Percentage Effects of Outliers	Percentage Effects of Leverage Values
Least Squares Regression	11.84	17.19
Monotone Regression	12.15	17.48
Rank Regression	NA	NA

Table 3.22: The % Effect of Outlier and Leverage Values of Project 5, (MAVD)

Models	Percentage Effects of Outliers	Percentage Effects of Leverage Values
Least Squares Regression	3.9366	6.3574
Monotone Regression	3.9631	6.1657
Rank Regression	NA	NA

Table 3.23: Analysis of SRRM1, Monotone, and Rank Regression (SL n=50)

Simulated Skewed Laplace Complete Data (n=50)

Models	SRRM1	Monotone Regression	Rank Regression
MSE	0.7973	8.0781	4.5430
MAVD	0.6556	2.6587	1.7903
MMER	0.1681	2.2445	1.4355
MMRE	0.2130	0.6840	0.5821
MdAR	0.4806	2.4248	1.6064
V			291.5592

Table 3.24: Analysis of SRRM1, Monotone, and Rank Regression (SL n=44)

Simulated Skewed Laplace Data without Outliers (n=44)

Models	SRRM1	Monotone Regression	Rank Regression
MSE	0.3725	2.1137	2.0108
MAVD	0.4451	1.2772	1.1598
MMER	0.1237	0.5549	0.4667
MMRE	0.1280	0.3334	0.3276
MdAR	0.3180	1.1966	1.0412
V			203.4474

Table 3.25: MSE and V Percentage Effect of Outlier Values (SL n=50)

Simulated Skewed Laplace Failure Data of Size 50

Models	Complete Data	No Outlier Values	Percentage Effect of Outliers
Least Squares Regression	MSE=0.7973	MSE=0.3725	53.2798
Monotone Regression	MSE=8.0781	MSE=2.1137	73.835
Rank Regression	V=291.5592	V=203.4474	30.2208

Table 3.26: Analysis of SRRM1, Monotone, and Rank Regression (TSL n=50)

Simulated Truncated Skewed Laplace Complete Data (50)

Models	SRRM1	Monotone Regression	Rank Regression
MSE	1.2079	3.6117	6.0844
MAVD	0.8585	1.4174	1.9490
MMER	0.5010	99.0884	1.5016
MMRE	4.8654	0.9498	7.6142
MdAR	0.8108	1.1812	1.4425
V			223.3501

Table 3.27: Analysis of SRRM1, Monotone, and Rank Regression (TSL n=42)

Simulated Truncated Skewed Laplace Data Without Outliers (n=42)

Models	SRRM1	Monotone Regression	Rank Regression
MSE	0.2709	1.2747	1.4043
MAVD	0.4273	0.9481	0.9449
MMER	0.4562	66.2997	4.0417
MMRE	2.5277	0.9410	4.8870
MdAR	0.4446	0.8877	0.8425
V			107.0164

Table 3.28: MSE and V % Effect of Outlier Values (TSL n=50)

Simulated Truncated Skewed Laplace Failure Data of Size 50

Models	Complete Data	No Outlier Values	Percentage Effect of Outliers
Least Squares Regression	MSE=1.2079	MSE=0.2709	77.5726
Monotone Regression	MSE=3.6117	MSE=1.2747	64.7063
Rank Regression	V=223.3501	V=107.0164	52.0858

The following Table (3.29) represents the output of three models: least squares regression, Monotone regression, and the rank regression by applying the three procedures on the monotonic data that is taken from a book [56]. Table (3.29) supports the theoretical point of view that if the data is monotonic, then the monotonic regression method is the best choice for software reliability.

Table 3.29: Comparison of Models Using Different Measurement's Criteria

(Using a Monotonic Data)

Model	Least Squares Method	Monotone Regression	Rank Regression
MSE	3.2814	0.2011	7.1656
MAVD	1.4008	0.2829	1.8184
MMER	0.1020	0.0188	0.1175
MMRE	0.1454	0.0193	0.2450
V	36.9	116.10	99.7500
R-Square)	0.9512	0.9692	0.5767
R-adj	0.9485	0.9675	0.5532

The results in Table (3.29) support this study for the need of the monotone regression and the rank regression models. Since the hypothetical data is normal, we are expecting the least squares regression model to be an acceptable model for prediction. But the good thing is that MMRE of the three models is smaller than 0.25, meaning that all three models are considered to be acceptable for prediction. At the same time the error measurements of the monotonic regression model are the best among these three. The reason behind it supports our theory that if the data is monotonic, then we can expect the monotonic regression model to be the best choice. The value of R-square is 0.9512 for the least squares method, and $R^2 = 0.9692$ for the monotonic regression method, while the value R-square for the rank regression method is 57.67 percent. These results are very good for the simulated Monotonic normal data. The error measurements of the rank regression illustrate an acceptable model for prediction. The data is drawn from a normal distribution.

Table 3.30: MSE and V Percentage Effect of Outlier Values (n=20)

Normal Simulated Failure Data of Size 20 (mean=203.3626, std=38.4871)

(One Outlier =296.0650)

Models	Complete Data	No Outlier Values	Percentage Effect of Outliers
Least Squares Regression	MSE=0.0197	MSE=0.0154	21.8274
Monotone Regression	MSE=0.0240	MSE=0.0188	21.6667
Rank Regression	V=57.1451	V=52.1208	8.7922

Table 3.31: MSE and V Percentage Effect of Outlier values (n=50)

Normal Simulated Failure Data of Size 50 (mean=249.7735, std=38.34)

Three outliers (158.9660, 341.3610, 355.6750)

Models	Complete Data	No Outlier Values	Percentage Effect of Outliers
Least Squares Regression	MSE=0.0157	MSE=0.0114	27.3885
Monotone Regression	MSE=0.0192	MSE=0.0129	32.8125
Rank Regression	V=157.2260	V=142.9994	9.0485

Even if both two data sets are normal, we have better results for the rank regression model, while the monotone regression is better that the least squares regression for the data of sample size 20. The above results show that the rank regression is insensitive to outliers.

Chapter 4

Conclusion and Future Research

4.1 Summary of Dissertation

In this dissertation, we prepare and analyze an array of different procedures related to software reliability based on regression methods. A chapter by chapter summary is presented below.

4.1.1 Chapter One

Chapter one contains an introduction and review of software reliability modeling, including basic definitions, concepts, terminology, fundamentals of reliability, and a discussion on hardware and software reliability. Under fundamentals of reliability, two key concepts are covered: Failure Interval Description (FID) and Failure Time-Interval (FTI). Chapter 1 also includes the discussion of differences between Hardware and Software reliability. This chapter incorporates some important probability distributions used in reliability studies and literature review on software reliability models, which include non-homogenous Poisson process (NHPP) Models. This chapter also incorporates a fundamental theory of reliability, gives a summary of frequently-used error measurements prediction criteria, and provides a discussion of the software failure data sets used in the dissertation. At the end of this chapter, we have an overall summary of the dissertation.

4.1.2 Chapter Two

This chapter shows that linear regression methods can be effectively used to model software reliability failure prediction problems. This approach is motivated through a logarithmic transformation of power law processes. Linear regression approaches to software reliability models encompass our contribution of regression approaches to the power law process model, simple linear regression model, and successive prediction model by using regression. Model validation covers Quantile-Quantile Plots of TBF and predicted values, which show that the actual times between failures and the predicted values of TBF come from the same proposed models. Goodness-of-fit tests used for model validation of the PLP modeling and the simple linear regression model (SRRM1) show that evidence for both the PLP and the derived linear intensity function assumptions are valid. A comparison of models using the MSE, MAVD, MMRE, and MdAR error measurement criteria shows that there are significant improvements in comparison to previous MLE-based models of: Singpurwalla, Horigome, Suresh, and Robert in terms of these error measures. The main result of Chapter three is that the simple linear regression model without any NHPP assumptions gives comparable results to those derived in literature. Based on MMRE measure values, PLP-Regression and simple linear regression models perform very well. The first step to consider in software reliability should be a linear model for prediction.This outcome suggests that, in Software reliability, the first step should be to consider a simple linear model for prediction.

4.1.3 Chapter Three

This chapter follows the non-parametric approaches by proposing two main models, the monotone regression and the rank robust regression models. Chapter three illustrates that if we relax the assumptions of the least square procedure described in Chapter two, the monotone regression and rank regression models perform well for: System 40, Project 1, and Project 5 failure data sets. Both of these models are capable of predicting the next failures with smaller error measurements. They

are also less sensitive to outliers and leverage values of the time between failures, in comparison with the parametric models. The percentage effects of outliers and leverage values illustrate that the rank regression model is less sensitive to outliers and leverage values than either the least squares regression or the monotone regression.

4.2 Limitations

All of these methods are data specific. There is not a single method that is superior to all data sets. The main contribution of the results of this dissertation is that one should first try regression models and compute the error prediction measurements. Any other methods should be compared to the results obtained through the regression models.

With the concepts of error measurement prediction criteria discussed in this dissertation, one is able to analyze and compare the prediction models after the actual values of the time between failure have been obtained. This can be done by computing the relative error. That is, the difference between actual time between failure and the predicted value of the time between failure at a certain stage. This could be sufficient, if one has an excellent prediction software system that is better than other prediction models, all the time or most of the time, and applies the same software failure data sets. But this is not the case for all proposed models in this dissertation. There is no way to select a single best Software Reliability Growth system based on the properties of a data set. Our results show that by investigating the $MMRE$ measurement of the monotone regression model, some results works very well for System 40, Project 1, and Project 5 data sets, while performing poorly for the Apollo 8 data set. Because of this situation, it is necessary to have an adequate measurement criterion or a number of measurement criteria which allows us to select a prediction system that performs very well for the real life current data set. Therefore, it is preferable to adapt the validity criteria of the IEEE Standard for a Software Methodology [36], and take into consideration the reliability and validity in comparative studies of software reliability models [46], or apply the MSE and MAVD

criteria. According to [35], the analysis and research for a valid Software Reliability Growth Model that can be applied to all software failure data sets to make accurate prediction results under all circumstances seems to be difficult to achieve. If one's concern is to predict the next failure time, then there are certain techniques used in this work for comparing the software reliability growth models when using the same failure data set. But if one is looking for a software reliability growth model that has a long-term prediction, then after the first prediction or the next time to failure, one has to compute the parameter estimators for the new data including the predicted value. Next, the selected model has to be executed again to find the next-next time to failure and so on [1].

4.3 Future Directions

Some of future research could include: High breakdown rank regression, Kernel regression approach. It may be advisable to explore the Bayesian and non-parametric regression methods in detail.

References

[1] A. A. Abdallah, P. Y. Chan, and B. Littlewood Evaluation of Competing Software Reliability Predictions. In *IEEE Trans. Software Eng.*, vol. 12, no. 9, pp.950–967, 1986.

[2] ANSI: Recommended Practice for Software Reliability. In *American Institute of Aeronautics and Astronautics*, 1992.

[3] H. Ascher and H. Feingold. *Repairable Systems Reliability, Inference, Misconceptions and their Causes.* Marcel Dekker, New York, 1984.

[4] R. Bartoszynski and M. Niewiadomska-Bugaj. *Probability and Statistical Inference.* John Wiley and Sons, New York, 1996.

[5] D. A. Besley, E. Kuh, and R. E. Welsch. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity.* John Wiley and Sons, New York 1980.

[6] S. Bittani. *Lecture Notes in Computer Science.* Springer-Verlag, Berlin Heidelberg New York London Paris Tokyo, 1988.

[7] L. C. Brian, T. Langley, and I. Wieczorek. A Replicated Assessment and Comparison of Common Software Cost Modeling Techniques. In *Proc. Int'l. Conf. Software Eng.*, vol. 22, pp.377–386, 2000.

[8] W. J. Conover. *Practical Non-parametric Statistics 2ed.* John Wiley and Sons, 1980.

[9] S. D. Conte, H. E. Dunsmore and V. Y. Shen. *Software Engineering Metrics and Models.* Benjamin/Cummings, Menlo Park, California, 1984.

[10] D. R. Cox and P. A. Lewis. *The Statistical Analysis of Series of Events*. Chapman and Hall, London, 1996.

[11] L. H. Crow. Reliability for Complex Systems, Reliability and Biometry. *Society for Industrial and Applied Mathematics (SIAM)*, pp.379–410, 1974.

[12] L. H. Crow and N. D. Singpurwalla. An empirically Developed Fourier Series Model for Describing Failures. *IEEE Trans. Reliability*, R-33(2), pp.176–183, 1984.

[13] A. L. Goel. A Guide Book for software Reliability Asessment. *Technical report*, RADC-TR-83-176, Rome Air Development Center, Rome, New York, 1983.

[14] A. L. Goel. Software Reliability Models: Assumptions, Limitations and Applicability. *IEEE Trans. on Soft. Eng*, SE-11, pp.1411–1423, 1985.

[15] A. L. Goel, and K. Okumoto. Time-Dependenter or Detection rate Model for software reliability and Other Performance Measures. *IEEE Trans. Reliability*, R-28, pp.206–211, 1979.

[16] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel. *Robust Statistics: The Approach Based on Influence Functions*. John Wiley and Sons, New York, 1986.

[17] D. J. Hand. *Small Data Sets*. Chapman and Hall/CRC: 1st edition, November 1, 1993.

[18] T. Hastie, and R. Tibshirani. Generalized Additive Models: Some Applications. *Journal of the American Statistical Association*, vol. 82, no. 5, pp.371–386, 1987.

[19] T. Hastie, and R. Tibshirani. *Generalized Additive Models*. Chapman and Hall, 1990.

[20] T. P. Hettmansperger, and J. W. McKean. A Robust Alternative Based on Ranks to Least Squares in Analyzing Linear Models. *Technometrics*, vol. 19, no. 3, August 1977.

[21] M. Hollander, and D. A. Wolf. *Non-paramtric Statistical Methods.* John Wiley and Sons, New York, 1999.

[22] M. Horigome, N. D. Singpurwalla, and R. Soyer. A Bayes Empirical Bayes Approach for Software Reliability growth. In *Computer Science and Statistics*, (16th Symp. Interface, Atlanta, GA), North-Holland, pp.47–55, 1984.

[23] P. Huber. *Robust Statistics.* John Wiley and Sons, New York, 1981.

[24] P .Huber. Robust Regression: Asymptotics, Conjectures and Monte Carlo. *Ann. Statist.*, vol. 1, pp.799–821, 1973.

[25] R. L. Iman, and W. J. Conover. The Use of Rank Transform in Regression. *Technometrics*, vol. 21, no. 4, pp.499–506, Nov. 1979.

[26] A. L.Jackel. Estimating Regression Coefficients by Minimizing the Dispersion of the Residuals. *Ann. Math. Statist.*, vol. 43, no. 5, pp.1449–1458, Oct. 1972.

[27] Z. Jelinski, and P. B. Moranda. *Software Reliability Research. Statistical Computer Performance Evaluation.* Academic Press, NewYork, pp.465–484, 1972.

[28] J. Jureckova. Non-parametric Estimate of Regression Coefficients. *Ann. Math. Statist.*, vol. 42, 1971.

[29] P. K. Kapur, and R. B. Garg. Optimum Softwarte Release Policies for Software Reliability Growth Models Under Imperfect Debugging. *R.A.I.R.O.*, 1991.

[30] N. Kareer, et al. An S-shaped Software Reliability Growth Model with Two Types of Error. *Microelectronics and Reliability*, vol. 30, pp.1085–1090, 1990.

[31] B. A. Kitchenham, S. G. MacDonell, L. Pickard, and M. J. Shepperd. What Accuracy Statistics Really Measure. *IEEE Pro. Software Engineering*, 148, pp.81–85, 2001.

[32] B. Littlewood and J. L. Verral. A Bayesian Reliability Model with a Stochastically Monotone Failure rate. *IEEE Trans. Reliab.*, R-23(1), pp.108–114, 1974.

[33] M. R. Lyu. *Handbook of Software Reliability Engineering.* McGraw-Hill, 1996.

[34] T. A. Mazzuchi and R. Soyer. A Bayes Empirical-Bayes Model for Software Reliability. *IEEE Trans. Reliability*, 37(2), pp.248–254, 1988.

[35] R. L. Michael. *Handbook of Software Reliability Engineering.* McGraw-Hill, 1996.

[36] C. E. Modell. IEEE Standards for a Software Quality Metrics Methodology. *IEEE Standard 1061-1998*, Sodftware Engineering Committee, New York, Dec. 1998.

[37] D. C. Montgomery, E. A. Peck, and G. G. Vining. *Introduction to Linear regression Analysis.* John Wiley and Sons, New York, 2001.

[38] P. B. Moranda. Predictions of Software Reliability During Debugging. *Proceedings of Annual Reliability and Maintainability Symposium*, pp.327–332, 1975.

[39] A. Mostafa, K. M.Ramachandran, and A. N. V. Rao. Regression Approach to Software Reliability Models. *International Journal of Pure and Applied Mathematics (IJPAM)*, vol. 26, no. 2, 2006.

[40] J. D. Musa. `http://www.dacs.dtic.mil/databases/sled/swrel.shtml`.

[41] J. D. Musa. Theory of Software Reliability and its Application. *IEEE Press, Orlando, Florida, United States*, S. 312–327, 1975.

[42] J D. Musa. Validity of the Execution Time Theory of Software Reliability. *IEEE Trans. Reliability*, R-28, pp.181–191, 1979.

[43] J. D. Musa. Theory of Software Reliability. *Data and Analysis Center for Software*, January 1980.

[44] J. D. Musa, A. Iannino, and K. Okumoto. *Software reliability.* McGraw-Hill, New York, 1987.

[45] J. D. Musa, and K. Okumoto. *Application of Basic and Logarithmic Poisson Execution Time Models in Software Reliability Measurement.* In Software System Design Methods, Ed. J.K. Skwirzynski, Nato Series, vol. F22, Springer-Verlag, Berlin, pp.275–298, 1986.

[46] I. Nyrtveit, E. Stensrud, and Martin Shepperd. Reliability and Validity in Comparative Studies of Software Prediction Models. *IEEE Transactions on Software Engineering*, 31(5), pp.380–391, 2005.

[47] H. Ohba, et al. S-shaped Software Reliability Growth Curve: How Good is it? *COMPSAC'82*, pp.38–44 1982.

[48] H. Ohba, and S. Yamada. S-shaped Software Reliability Models. *4th Int. Conf. on Reliability and Maintainability*, pp.430–436, 1984.

[49] H. Qiao. *Parametric and Nonparametric Statistical Modeling: Reliability Analysis.* Ph.D. thesis, University of South Florida, 1993.

[50] H. Qiao and C. Tsokos. Best Efficient Estimates of the Intensity Function of the Power Law Process. *Journal of Applied Statistics*, 25(1), pp.111–120, 1998.

[51] S. E. Rigdon and A. P. Basu. The Power Law Process: a Model for the Reliability of Repairable Systems. *Journal of Quality Technology*, 21(4), pp.251–259, 1989.

[52] S. E. Rigdon and A. P. Basu. *Statistical Methods for the Reliability of Repairable Systems.* John Wiley and Sons, New York, 2000.

[53] S. E. Rigdon, M. Xiaolin, and K. M. Boden. Statistical Inference for Repairable Systems Using the Power Law Process. *Journal of Quality Technology*, 30(4), 1998.

[54] H. Roberts. *Predicting the Performance of Software Systems via the Power Law Process.* Ph.D. thesis, University of South Florida, Tampa, FL, 2000.

[55] P. J.Rousseeuw, and A. M.Leroy. *Robust Regression and Outlier Detection.* John Wiley and Sons, New York, 1987.

[56] P. T.Ryan. *Modern Regression Methods.* Wiley, New York, 1996.

[57] S. Sawyer. A Rank Regression Estimating Procedure. *www.math.wustl.edu/ sawyer/handouts*, April 25th, 2003.

[58] G. J. Schick, and R.W. Wolverton An analysis of competing software reliability models. *IEEE Trans. Software Eng.*, SE-4, pp.104–120, 1978.

[59] N. F. Schneidewind. Analysis of Error Processes in Computer Software. *Proceedings of the International Conference on Reliable Software, IEEE Computer Society*, 21–23, pp.337–346, 1975.

[60] S. R.Searle. *Linear Models.* John Wiley and Sons, New York, 1971.

[61] B .W.Silverman. Some Aspects of the Spline Smoothing Approach to Non-parametric Regression Curve Fitting. *Journal of the Royal Statistical Society*, Series B, 47, pp.1–21, (Discussion pp.21–52), 1985.

[62] N. D. Singpurwalla, M. Horigome, and R. Soyer. A Bayes Empirical Approach for Software Reliability Growth. *Computer Science and Statistics, The Interface*, North Holland, pp.47–55, 1985.

[63] N. D. Singpurwalla and S. P. Wilson. *Statistical Methods in Software Engineering.* Springer, New York, 1999.

[64] N. Suresh. *Modeling and Analysis of Software Reliability.* Ph.D. thesis, University of South Florida, Tampa, FL, 1992.

[65] `www.sussex.ac.uk/Units/economics/appl_stats/aslec_09`

[66] H .Theil. A Rank-Invariant Method of Linear and Polynomial Regression Analysis. *I. Proc. Kon. Ned. Akad.V. wetensch* A. 53, pp.386–392, 1950a.

[67] H .Theil. A Rank-Invriant Method of Linear and Polynomial Regression Analysis. *II. Proc. Kon. Ned. Akad. V. wetensch*, A. 53, pp.521–525, 1950b.

[68] H .Theil. A Rank-Invariant Method of Linear and Polynomial Regression. *III. Proc. Kon. Ned. Akad. V. wetensch*, A. 53, pp.1397–1412, 1950c.

[69] F. Walkerden and R. Jeffrey. An Empirical Study of Analogy-Based Software Effort Estimation. *Empirical Software Eng.*, 4(2), pp.135–158, 1999.

[70] M. Xie. *Software Reliability Modeling*. World Scientific, Singapore, 1991.

[71] S. Yamada, et al. A Software Reliability Growth Model with two Types of Errors. *R.A.I.R.O.*, vol. 19, pp.87–104, 1985.

[72] S. Yamada, H. Ohtera, and H. Narihisa. Software Reliability Growth Models with Testing Effort. *IEEE Trans. Reliability*, R-35, pp.19-23, April, 1986.

[73] S. Yamada, and S. Osaki. Discrete Models for Software Reliability Evaluation. *In Reliability and Quality control*, Ed. A.P. Basu, Elsevier, London, pp.401–412, 1986.

About the Author

Abdelelah Mostafa had interests in Mathematics and Science. He received a Bachelor's degree in Electrical Engineering in 1980 from Beirut Arab University. He taught mathematics and electricity at UNICEF in Lebanon. He earned a Master degree in Mathematics in 1991 from Texas Southern University (TSU). He joined the University of South Florida (USF) to achieve his Ph.D. in Mathematics/Statistics. He taught Mathematics at USF, TSU, and HCC. At this time, he was juggling the crucial tasks of giving his five children the best of care and offering them the greatest education while taking care of his graduate studies. Nevertheless, he volunteered to teach Mathematics at the Urban and other educational programs at USF on weekends. Today, his dream of starting an educational program which taught Mathematics, Science, and Languages has come to life; and he continues to devote his very life to a most priceless tool, education.