Some Contemporary Issues in Software Reliability

Vignesh T Subrahmaniam

Indian Statistical Institute, Kolkata

October 10, 2015

Some Contemporary Issues in Software Reliability

Vignesh T Subrahmaniam

Thesis submitted to the Indian Statistical Institute

in partial fulfillment of the requirements

for the award of the degree of

Doctor of Philosophy

March 2015

Indian Statistical Institute

203, B.T. Road, Kolkata, India.

Dedicated to

My Mother, Sarala Subrahmaniam

who has stood by me through all my trials and tribulations

&

My Father, Late T. C. Subrahmaniam

who has been my source of inspiration

Acknowledgments

I would like to express my heartfelt thanks and gratitude to my thesis supervisors Prof. Bimal Roy and Prof. Anup Dewanji of the Applied Statistics Unit, Indian Statistical Institute for their constant support, guidance and encouragement that helped me pursue my PhD. Much of what I have learnt about research in statistics is in a large part due to them. I would like to particularly thank both of them for their patience and flexibility in guiding me while I was working at GE Global Research. I would like to thank Prof. Bimal Roy for motivating me to take up a career in research. I also thank the Applied Statistics Unit for arranging accommodation for me during my numerous visits to the Institute.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Program testing can be used to show the presence of bugs, but never to show their absence!

Edsger W. Dijkstra

1.1. Preamble

The Statistician John Wilder Tukey was the first to coin the term “Software” (Tukey, 1958; Leonhardt, 2000) to describe programs running on electronic calculators in the 1950’s. Four decades before the “dot-com” boom, Tukey recognized the importance of computer programs for mankind and wrote that “It is at least as important” as the “hardware of tubes, transistors, wires, tapes and the like”. Since Tukey’s assessment, the importance of software to human civilization and the global economy has grown substantially. The size of the software industry can be judged from a Gartner report (Gartner Inc., 2014) which estimated the global software market revenue to be \$407.3 billion in 2013, a 100 billion dollars more than the revenue of the global semi-conductor market (Gartner Inc., 2013), representing the “transistors” Tukey alluded to in 1953.

The importance of software to mankind far exceeds the revenues of \$407.3 billion it generates for the companies that create software products and services. It forms a critical part of almost all aspects of modern living, from the monitoring programs running on ECG monitors in a hospital’s intensive care unit, to the auto-pilots flying an aircraft, to banking software recording financial transactions and to applications running on mobile phones. The list of technologies in which software plays a critical role is endless and is only expected to grow over the next decade due to the Internet of Things: a global network of human beings, machines and software programs.

The pervasiveness of software underscores the need for building reliable software products and solutions. For example, the “Y2K” bug is a famous software defect caused by abbreviation of four digit years to two digits cost the global economy anywhere between $300 million to $ 600 million (British Broadcasting Corporation, 2000). Another dramatic software error led to the loss of the $327 million Mars Climate Orbiter launched by NASA in 1998 due to a software bug in the orbiter’s guidance system (Isbell, Hardin and Underwood, 1999). Software defects related to security vulnerabilities can result in massive losses for users of the software; for example, the Code Red virus and Mellisa virus are estimated to have caused losses of $ 2.1 billion and $ 1.1 billion, respectively, (Telang and Wattal, 2007). Critical software defects not only affect the users but can also cause substantial loss of brand value for the software manufacturer and subsequently their market capitalization (Telang and Wattal, 2007). A National Institute of Standards (2002) report estimated the losses caused to the US economy due to defective software to be $60 billion in 2002.

The consequences of software defects are not just limited to economic losses but to loss of lives and productivity. The radiation overdose given by Therac-25 machines to cancer patients is another example of a catastrophic software fault

TABLE 1.1
Sub-characterization of software reliability as per ISO 9126

Sub-characteristic	Definition
Maturity	Attributes of software that bear on the frequency of software failure
Fault Tolerance	Attributes of software that bear on its ability to maintain a specified level of performance in case of software faults or of infringement of its specified interface
Recoverability	Attributes of software that bear on the capability to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort needed for it

(Jacky, 1989; Kapur et al., 2011). The power blackouts caused in the United States during August 2003 are attributed to an unanticipated race conditions in a power-grid control software (Fairley, 2004).

Creating a perfectly defect-free software is clearly the solution for mitigating the risks associated with defective software. The first step towards this objective is to develop a method of *measuring how reliable a software is*. There are quite a few definitions of software reliability. A commonality among these definitions is that the metric should correlate with the probability of failure-free operation of a software product over a specified duration of usage. The closer this probability is to one over large durations of future usage, the less defective the software will be and hence greater the reliability. Software reliability models estimate such a probability which provides (i) a critical input to determining whether the risk of a software failure in the future is acceptable and (ii) a method for comparing the reliability of two or more software products with the same functionality facilitating the selection of the most reliable product. Software reliability is one among six software quality aspects laid down in ISO standards 9126 (Abran et al., 2003) where it is divided into three sub-characteristics as described in Table 1.1.

Reliability metrics can be defined specifically for any one of these sub-characteristics, or for all of them together. The ability to create a sub-characteristic specific reliability measure would depend on whether software defects can be accurately classified into one of these sub-characteristics. For example, behavioral faults in a software can be attributed to a lack of maturity in the software. An example of this would be the Y2K bug. Software defects due to lack of sufficient safeguards against illegal user-inputs would correspond to lack of fault-tolerance, an example of which would be inadequate measures against “SQL code-injection” in database querying. It is possible that a software defect might point to lack of reliability across multiple sub-characteristics in which case the same defect would contribute to measuring the reliability across all these sub-characteristics.

1.2. Objectives

Software testing in the past decade has been revolutionized by two technology developments, namely, (i) the world-wide adoption of the Internet as a communication medium resulting in software users around the globe having the capability to report defects on a voluntary basis and (ii) distributed software product development involving 1000’s of software developers using platforms such as *Rational Rose* or *Github*. Under such coding platforms each developer is responsible for testing the code written by them resulting in white-box testing techniques, as opposed to black-box testing prevalent in the 1980’s and 1990’s, becoming the norm. For software reliability modeling both these developments mean that parametric models based on assumptions regarding (i) nature of reliability improvement after fixing a defect and (ii) the usage of the software during the testing procedure may no longer be appropriate.

In this thesis, we propose two new models for software reliability that cater to these developments. The first model is called the isotonic software reliability model that generalizes popular software reliability models. The model makes very little assumptions about the nature of reliability improvement after the repair of every additional defect and can also consider decrease in reliability after a debugging effort. We believe that the generic nature of this model will make them applicable to a wide range of testing frameworks including white-box testing. We provide novel and yet simple methods for estimating confidence bounds on the reliability of the software under this generic model.

The second model is devoted to analysis of software defect data arising out of uncontrolled usage of the software with the defects being voluntarily reported through the Internet and stored in bug-databases. The reported defects are typically classified into multiple types. We formulate a semi-parametric software reliability model that makes no assumptions about the underlying software usage and can be estimated from data retrieved from bug-databases. The classification of defects into multiple types is exploited to propose a novel partial-likelihood method for estimating the model parameters. We apply the methods developed to data retrieved from the Bug database of Python, a popular scripting language.

The models developed in the thesis have applications beyond software reliability. The methodologies described in Chapters 3 and 4 are not based on asymptotic methods. This means that they can be used for analysis of data-sets corresponding to catastrophic failures where the number of failures is expected to be small. For example, they can be used to analyze data regarding failures in critical processes such as fraud prevention in internet banking or hardware failures in nuclear power plants. The methods presented in Chapter 5 could have applications in epidemiology and demand estimation in online retail markets. We discuss such possibilities briefly in Chapters 5 and 6.

1.3. Literature Review

We begin our survey of the state of the art in statistical modeling and analysis of software reliability by first discussing software testing. This is because software testing provides the data for almost all software reliability modeling. An understanding of the different types of testing methodologies helps in choosing the appropriate software reliability model for analyzing software failure data.

1.3.1. Software Testing

Testing of a software product before its release to customers represents one of the most common methods of detecting faults in a software product. Software testing is a component of *software verification* which deals with ensuring that a software product adheres to all its customer requirements. Software verification has two parts; the first part deals with static verification and ensures that the software product meets coding or algorithm quality standards and the second part deals with dynamic verification and ensures functioning of the software product during run-time. Static verification deals with (i) inspecting the code to ensure it meets coding conventions such as variable naming conventions, commenting guidelines and readability requirements, (ii) adheres to specified software design patterns such as *Model-View-Controller* pattern or the *Service Location* pattern,

(iii) meets requirements on code complexity measured through metrics such as the cyclomatic measure (McCabe, 1976) or the Halstead complexity metric (Yu and Zhou, 2010) and (iv) the formal verification of the correctness of the underlying algorithm in the software (Bérard et al., 2010). Formal verification has its roots in theoretical computer science and uses algebraical methods to prove that an algorithm is correct. Formal verification of a software is often not done simply because of its complexity.

Dynamic verification, also known as software testing, is performed to detect faults during the execution of a software. More specifically, software testing is defined as “A process of executing a program with the goal of finding errors” (Abran et al., 2004). Testing involves identifying defective outputs of a software program through test cases which are “a set of inputs, execution preconditions, and expected outcomes developed for a particular objective such as to exercise a particular program path or to verify compliance with a specific requirement” (Radatz, Geraci and Katki, 1990). Software testing can, for example, detect run-time errors due to improper use of computer threads and inadequate safeguards against illegal user inputs. There are various software testing paradigms. These are briefly discussed.

White-Box Testing: In this type of testing, software testers leverage their knowledge of the internals of the software product to test it. The source code of the software product is required to be available to the software testers. White-box testing is highly dependent on the skill of the software tester. A skilled software tester can detect defects at a much faster rate than a less skilled tester. If there is a team of tester involved in white-box testing there may be substantial heterogeneity in the rate at which different testers detect defects. White-box testing can reveal major software errors in the initial stages of testing. A software reliability model analyzing test-data from a white-box testing procedure needs to consider variations in the skill levels of different testers and must allow for different defects to have different severity. White-box testing is commonly used in security testing by high skilled computer security engineers (Janardhanudu and van Wyk, 2005).

Black-Box Testing: This is the exact opposite of white-box testing in that the tester has no access to the source code and does not understand the internals of the software product. The tester can only control the sequence of test-cases and determine a fault if the program output of the test case does not match the expected output. The testing procedure can be automated and each tester can be assumed to have the same skill level. Many early software reliability models considered that software testing was black-box testing. In black-box testing it might be reasonable to assume that the severity of a defect has no dependence on the duration or number of test-cases it took to detect the defect.

Gray-Box Testing: As the name suggests, this form of testing bridges black-box and white-box testing. The tester is assumed to have some knowledge of how the software works but may not have as much information as a white-box tester. User-driven defect discovery which is becoming quite popular in the form of beta-testing is an example of gray-box testing. Software reliability models for gray-box testing are few in number as they need to make more generic assumptions regarding the defect discovery process as compared to models for black-box testing.

Equivalence-Partitioning Testing: This is a specific example of white-box testing and depends on identifying partitions of the input data, such that two inputs from the same partition will be processed by the same portions of the code. Once the partitions are identified, then a suite of test cases is designed to ensure a minimum number of test cases lie within each of the partitions. The advantage of this testing methodology is that it can lead to economic test plans. The method requires identification of the input data partitions which might be difficult for a complex software program.

Unit Testing: Unit-testing is a commonly used form of white-box testing. As the name suggest, it tests a "unit" of source code, where a unit is defined as the smallest part of the code that can be provided an input and whose output can be verified. Unit-tests are used to ensure that specific modules of a software program function as they are expected to. Unit testing is popular as it can detect problems during coding itself and also be used in a distributed programming environment where different teams code different modules. This framework is supported by many popular programming languages (see *junit* for unit-testing java code or *unittest* for unit-testing in Python).

Exploratory Testing: This is a specific form of gray-box testing where the test-cases are determined dynamically based on the outputs observed historically. In this type of testing, the assumption that there is uniform hazard of detecting a defect may not be appropriate. Also, the expertise of the tester would play an important role in determining the rate of detection of defects.

User-Driven Testing: The reliability of a software is increasingly defined through the eyes of its customers. Such a definition has economic benefits as a significant number of tests may verify functionality not important to a user. User driven testing has been made possible in the last decade due to the Internet becoming a pervasive communication medium. Users of a software can voluntarily report a defect through the internet with the defect being logged in specialized databases called bug-databases. User-driven testing represents one of the major innovations in software testing in the past decade and statistical models and methods for analyzing data from this testing procedure are the need of the hour. A software reliability model for such data would need to consider non-availability of information regarding the usage of the software (such as the number of users and how often the software being used) and the voluntary naturing of the reporting.

1.3.2. Software Reliability Modeling

Whatever be the software testing technique, their objective is to detect defects and repair them with the expectation that the reliability of the software will improve. The purpose of a software reliability model is to measure how much improvement in reliability can be attributed to every additional defect discovery and repair. Software testing is an expensive process and the cost involved in testing a software needs to be traded with the cost of releasing a defective software product. A software reliability model can extrapolate the reliability of the software for every additional unit of testing duration and for every additional defect discovered. An application of such a model would be the determination of the optimal duration of testing provided the cost of releasing a defective software product and the cost of testing are both available. Sometimes the cost of releasing a defective software product may not be known ahead of time, in which case the decision to stop testing a software maybe made after the software achieves a certain *target reliability* (Dewanji, Sengupta and Chakraborty, 2011). It hardly needs to be mentioned that any estimate of the reliability of the software needs to be accompanied by its corresponding uncertainty. Before we describe some of these models and discuss their relative merits we would like to discuss some commonly used definitions of software reliability. Note that these definitions are not specific to software but can also be used to measure the reliability of a hardware or the reliability of a process.

Software Reliability: Let T be a non-negative valued random variable representing the time to failure of any system which includes software products. The reliability of a system $R(t)$ over a duration of usage t is defined as

$$R(t) = P(T > t). \tag{1.1}$$

If $f(t)$ is the probability density function of T, then

$$R(t) = \int_t^\infty f(s)\,ds. \tag{1.2}$$

The hazard of failure occurring at time t, if $f(t)$ exists, is defined as

$$\lambda(t) = \frac{f(t)}{R(t)}. \tag{1.3}$$

The hazard can be used to approximate the instantaneous probability of a defect being observed at time t through the approximation $P(T \leq t + \delta t | T > t) \approx \lambda(t)\delta t$.

Also, $R(t)$ can be computed in terms of $\lambda(t)$ as

$$R(t) = \exp\left(-\int_0^t \lambda(s)\,ds\right). \tag{1.4}$$

Any continuous function $\lambda(t)$, s.t., $\lambda(t) \geq 0$ for $t \geq 0$ and $\int_0^\infty \lambda(t)\,dt = \infty$ is a hazard for a failure time distribution. Hence a software reliability model can be specified either through the probability density function or through the hazard function of the inter-failure times. A popular measure of reliability is the *expected time of defect free operation of a software* after its latest debugging and is called the Mean Time To Failure (MTTF) and is given by

$$MTTF = \int_0^\infty R(t)dt = \int_0^\infty \exp\left(-\int_0^t \lambda(s)\,ds\right)dt. \tag{1.5}$$

An accurate computation of the MTTF depends on an accurate specification of the underlying hazard. We shall discuss the importance of this requirement in Chapter 5 when we discuss analysis of software defect data from user-driven software testing. Now that we have defined software reliability, we will proceed to describe some software reliability models.

Software reliability models, also known as software reliability growth models (SRGM) or software reliability improvement models (SRIM), are commonly used to determine the cumulative improvements in the reliability of the software due to all the defects discovered and repaired so far and compute the current reliability of the software. A software reliability model needs to address the following challenges: (i) Uncertainty in the usage profile or testing profile of the software product. The usage profile determines the hazard of detecting a software defect, with higher usage implying higher hazard. Hence, from (1.2), the probability of defect-free operation of a software depends on assumptions about how the software will be used or tested in the future. For example, if the testing methodology were to change from black-box to white-box testing, then the estimates of reliability will no longer be valid. (ii) High costs of test generation resulting in few number of failures for estimating the software reliability model. This is particularly true for security testing where the number of security defects detected in a software can be as small as ten. Small software failure data-sets make computation of uncertainties in the estimates of reliability even harder as asymptotic and bootstrap techniques will no longer be appropriate. (iii) Lack of an accurate mathematical model for the testing process generating the test data leading to biased estimates of reliability. This is especially true for white-box testing or user-driven testing where assumptions regarding the skill of the testers and users in discovering defects may be hard to justify.

There are many different taxonomies of software reliability models, see (See, Musa, Laninio and Okumoto, 1987; Ramamoorthy and Bastani, 1982; Goševa-Popstojanova and Trivedi, 2001, for some examples). These taxonomies are based on the nature of the data used to estimate the models. For example, inter-failure time durations vs cumulative defect counts. Recently there have been attempts to classify the models based on Software Development Life Cycles (See, Kapur et al., 2011, for some examples). If we follow the classification used by Singpurwalla and Wilson (1994) then software reliability models can be classified into two broad categories: (i) Inter-failure time models and (ii) Counting process models. Since the review by Singpurwalla and Wilson (1994) a new class of models using machine learning techniques have been proposed. For the sake of completeness we shall also discuss such models briefly.

A commonality among the different classes of software reliability models is to assume an increasing reliability for the software with every additional defect discovery, although some models do allow for the possibility of a decrease in reliability with additional defect discoveries. A sofware reliability model assuming the contrary would imply that discovering more software defects would ultimately lead to decreased reliability of the software and would question the very need for software testing. Software reliability models which have an underlying stochastic assumption that explain the defect discovery process can lead to easily interpretable software reliability metrics. For example, from the Jelinski-Moranda model, one can estimate the number of remaining defects in the software. When such an estimate has a tight confidence bound and has little or no bias, it can be valuable to a product manager to determine if additional testing is required or not. A key weakness of stochastic software reliability models is that they can make assumptions that are hard to validate. Incorrect assumptions about the usage profile of the software can lead to biased estimates of software reliability. An example of this problem has been studied through simulations and presented in Section 5.3. An important weakness of stochastic software reliability models is that most of them do not prescribe goodness-of-fit tests that can be used to determine the validity of their assumptions using data from testing of the software. Another deficiency of many software reliability models is their inability to estimate the model when there are few failures. This is particularly important when the objective is to estimate the reliability of a software with respect to rarely occuring defects such as security vulnerabilities. This problem is further compounded by the lack of small sample statistical inference methods for the estimated model parameters.

Bayesian software reliability models can handle the availability of few software failures for model estimation. However, they may be criticized for their choice of priors. The methods proposed in Chapters 3 and 4 can help address this issue.

Inter-Failure Time Models

A software testing procedure would note the calendar time S_i, since the release of the software, when the i^{th} defect was detected, for $i = 1, \ldots, n$. The inter-failure time is defined as $T_i = S_i - S_{i-1}$, for $i = 2, \ldots, n$, with $T_1 = S_1$. The conditional distribution of $T_i | T_{i-1}, \ldots, T_1$ can be modeled to estimate the reliability of the software after, say, n defect discoveries. There are two approaches to modeling the conditional distribution; the first approach would model the hazard function, $\lambda_i(t)$ of $T_i | T_{i-1}, \ldots, T_1$ while the second approach would model $T_1, \ldots, T_n$ as an AR process. We present some important models from both these approaches

The Model of Jelinski and Moranda: This was one of the very first models of software reliability that assumed that $T_i \sim Exponential(\lambda(N-i+1))$ with $T_1, \ldots, T_n$ being independent of each other and $n \leq N$. The model implies that $\lambda_i(t) = \lambda(N-i+1)$. The parameter N is interpreted as the number of defects initially present in the software. We discuss this model in greater detail in Chapter 2. The reliability of the software over t units of usage after detecting and repairing the n^{th} defects is

$$R(t) = \exp(-(N-n+1)\lambda t). \tag{1.6}$$

The Jelinski and Moranda model pioneered the concept of decreasing hazard for the inter-failure times as a function of the number of defects discovered.

Bayesian Reliability Growth Model: Littlewood and Verrall (1973) proposed this model to extend the Jelinski-Moranda model by assuming that $\lambda_i(t) = \lambda_i$ with stochastically decreasing Gamma priors on λ_i as a function of i. The rate at which the gamma priors would decrease was controlled by an assumed parametric function of i. Littlewood (1981) and Mazzuchi and Soyer (1988) present some parametric forms for these Gamma priors as a function of i.

The De-Eutrophication Model of Moranda: The Jelinski-Moranda model implicitly assumed that every defect has the same severity. The de-eutrophication model was proposed by Moranda (1975) to counter this criticism by assuming that $T_i \sim \exp(\alpha + \beta i)$. The model estimation and inference are discussed in detail in Chapter 2.

The Model of Schick and Wolverton: In this model Schick and Wolverton (1978) assumed that $\lambda_i(t) = \lambda t(N - i + 1)$, i.e. the hazard is time-dependent. This model represents one of the first attempts at acknowledging non-constant hazards of detecting a software defect. This model is also discussed in more detail in Chapter 2.

Autoregressive Models of Software Reliability: Singpurwalla and Soyer (1985) pioneered the use of auto-regressive models for software reliability where they assumed that $T_i = \delta_i T_{i-1}^{\theta_i}$ with δ_i's being independent LogNormal random variables and θ_i's being modeled as an AR process. Extensions of similar types of models have been discussed by Singpurwalla and Soyer (1985, 1992) to name a few.

Software Reliability Models for Recapture Debugging: Nayak (1991) considered an interesting variation of software testing and debugging where a defect when discovered is not repaired immediately but remains in the software system to allow the number of re-discoveries of the defect to be determined. Nayak (1991) considered the problem of recapture debugging under the Jelinski and Moranda model assumptions and demonstrated efficiency gains in estimating the software reliability when compared to debugging without re-capture. Dewanji, Nayak and Sen (1995) considered the discovery and subsequent rediscoveries of each of the N defects to be governed by a renewal process with common renewal distribution $F(.|\theta)$ and discussed methods for estimating N and θ. Dewanji, Kundu and Nayak (2012) generalized the re-capture debugging model by estimating the renewal distribution non-parametrically.

Bayesian Software Reliability Models using Martingale Priors: Basu and Ebrahimi (2003) proposed priors for $\lambda_i(t) = \lambda_i$ using martingale processes, i.e., they assume $E_{prior}(\lambda_i|\lambda_{i-1}) = \lambda_{i-1}$. The justification being the discovery and repair of the i^{th} defect is expected to maintain the same level of reliability as before. Basu and Ebrahimi then proposed a series of prior for λ_i's that satisfy the martingale property and discuss the estimation of their posterior distributions through Monte-Carlo computations.

Counting Process Models

An alternative approach to software reliability is to model the cumulative defect counts $N(s)$ as a function of calendar time s. The non-homogenous Poisson process (NHPP) is the most commonly used counting process for modeling such data. A NHPP is defined through a positive and non-decreasing mean value function $m(s)$ and assumes that $N(s) \sim Poisson(m(s))$ and $N(s+t) - N(s) \sim$

$Poisson(m(s+t)-m(s))$ with $N(s+t)-N(s)$ being independent of $N(s)$ for all t,s with $t>0$. The function $m(s)$ represents the expected number of defects to be discovered up to and including time s. If $m(s)$ were known we can compute $P(N(s)=n)$ as

$$P(N(s)=n)=\frac{m(s)^n}{n!}\exp(-m(s)).$$

The reliability of a software product which is released at calendar time 0 over s units of future usage is $R(s)=P(N(s)-N(0)=0)$ and is given by

$$R(s)=\exp(-(m(s))). \tag{1.7}$$

The MTTF of the software under a counting process model would be

$$MTTF=\int_0^\infty \exp(-m(t))dt. \tag{1.8}$$

Equation (1.7) implies the hazard $\lambda(s)$ of a defect at calendar time s is given by

$$\lambda(s)=\frac{d}{ds}m(s), \tag{1.9}$$

when the derivative of $m(s)$ exists. Many counting process models specify $m(s)$ parametrically as a function of time s, i.e., $m(s)=f(s,\Theta)$ where Θ is the set of model parameter(s). Alternatively, they would express $\lambda(s)$ as a parametric function of $m(s)$ which due to (1.9) results in $m(s)$ being the solution to a differential equation with the boundary condition $m(0)=0$. Such models would typically assume a decreasing hazard $\lambda(s)$ as a function of $m(s)$. Estimates of the reliability of the software can be obtained through (1.7) and (1.8). If the reliability of the software is expected to improve with increasing test duration then $m(s)\to c$ as $s\to\infty$. Some models do not assume that this will happen allowing for what is known as imperfect debugging where the repair of a defect could introduce additional defects. Once the parametric form of $\lambda(s,\Theta)$ involving the model parameter Θ is available, estimates of Θ can be obtained through maximum likelihood procedures. The data from a testing procedure, governed by a NHPP process, can be represented as $D = \{n, s_1, \ldots, s_n, s\}$ where, s_i represents the calendar time of the i^{th} defect discovery of the n defects discovered in a software test of total duration s. The likelihood of the data D can be written as follows (V Basawa and Prakasa Rao, 1980):

$$L(\Theta|D) = \exp(-m(s)) \prod_{i=1}^{n} \lambda(s_i, \Theta). \tag{1.10}$$

The likelihood can be maximized to obtain maximum likelihood (ML) estimates of Θ. V Basawa and Prakasa Rao (1980) establish the consistency and asymptotic normality of these ML estimates. We now present some commonly assumed parametric forms for $m(s)$.

Goel and Okumoto Model: This was one of the first counting process models proposed by Goel and Okumoto (1979) where they assumed that the failure intensity of detecting a defect at calendar time s is a linearly decreasing function of the expected number of defects to be discovered by time s, i.e.,

$$\frac{d}{ds}m(s) = \lambda(s) = b(a - m(s)).$$

The solution to this differential equation along with the boundary condition $m(0) = 0$ results in

$$m(s) = a(1 - \exp(-bs)),$$

and $\lambda(s) = aexp(-bs)$, for unknown constants $a > 0$ and $b > 0$ with a being interpreted as the initial number of defects in the system.

Logarithmic Poisson Model: An important criticism of the Goel and Okumoto model was the linear decrease of $\lambda(s)$ as a function of $m(s)$. This was rectified by Musa and Okumoto (1984) by assuming that

$$\lambda(s) = \lambda_0 \exp\left(-\theta m(s)\right),$$

for some unknown constants $\lambda_0 > 0, \theta > 0$. This results in

$$m(s) = \frac{1}{\theta}\log(1+\lambda_0 s).$$

The justification for the model being that early defect result in greater improvement to the reliability as compared to defects discovered later on.

Hyper Exponential Growth Model: The defects detected in a software product occur in many different modules and all modules may not be tested equally nor will they have the same reliability. To address this concern Ohba (1984b) proposed what they called hyper exponential growth models in which the hazard of discovering a defect in the i^{th} module is governed by a module specific Goel and Okumoto model, that is $m_i(s) = a_i(1-b_i m_i(s))$ and $\lambda_i(s) = a_i(1-\exp(-b_i s))$, for $i = 1,\ldots,n$. The system-level mean value function, across all the modules, is given by $m(s) = \sum m_i(s)$. Ohba provided methods for estimating the model parameters when the information about the module in which each defect was discovered is missing.

S-Shaped Software Reliability Models: Starting from Ohba (1984a), a series of SRGM's were proposed in which $\lambda(s)$ was not a strictly decreasing function of $m(s)$, but assumed that $\lambda(s)$ would initially *increase* with $m(s)$ up to a calendar time point after which $\lambda(s)$ would decrease with $m(s)$. The justification for such a non-decreasing relationship comes from the following heuristic argument: defects discovered early in the test would trigger discovery of similar defects in other parts of the software leading to an initial spike in defect discovery that eventually decays. Ohba (1984a) first modeled this phenomenon by assuming that

$$\lambda(s) = am(s)(b - m(s)).$$

This results in m(t) being the sigmoid function

$$m(s) = \frac{a}{1 + \exp(-bs)}.$$

Since Ohba's model, a number of S-Shaped SRGMS have been proposed with increasingly sophisticated forms for $\lambda(s)$. See Yamada, Ohba and Osaki (1983); Yamada (1991); Yamada et al. (1994) for more details.

Some emerging trends in software reliability models

There are a couple of emerging trends in software reliability (SR) modeling. A popular trend, pioneered by Yamada et al. (1994), considers the use of stochastic differential equations (SDE) for modeling the mean value function of the stochastic process that counts the cumulative number of observed software defects. The use of SDE's is attractive, especially in white box testing or user-driven testing, as it provides more flexibility in the specification of the underlying stochastic process that governs the defect discovery process when compared to the more traditional parametric counting process models. A brief description of SDE based models is presented later in this section. One may view the emergence of SDE based SR models as a response to criticism that SR models have been making rigid assumptions about the underlying defect discovery process. SDE based models provide additional flexibility in parametrically specifying the mean value function of the underlying counting process through the use of stochastic differential equations. However, they still suffer from making parametric assumptions about the underlying defect discovery process that may be hard to validate.

Another emerging trend relates to SR models becoming more “ Statistical ” as compared to “ Stochastic”. The goal of the statistical model is to empirically estimate the distribution of time to discovery of the next defect using features from the past failure history of the software. The stochastic model, on the other hand, tries to explain the underlying defect discovery process using probabilistic models. This trend is partly motivated by a software product manager’s increasing need for estimating the probability of defect free operation of a released software with the greatest accuracy at (possibly) the expense of the resulting model not being easily explainable. The emergence of numerous code complexity metrics, derived from the source code of the software, has opened the possibility of using these metrics as covariates in SR models. A “Statistical” SR model can incorporate such metrics to better estimate the failure time distribution of the software. Some of the machine learning based SR models briefly described later follow this trend.

Stochastic Differential Equation Based Models: Stochastic differential equations (SDE) present a natural way of generalizing counting process models. SDE analogs for the model considered by Goel and Okumoto (1978) can be derived by replacing the mean value function of the NHPP $m(s)$ with $N(s)$, where $N(s)$ is the cumulative number of defects discovered till calendar time s. This results in

$$\frac{dN(s)}{ds} = \left(b(s) + \sigma\frac{dW(s)}{ds}\right)(a - N(s)), \tag{1.11}$$

where $W(s)$ is the standard Wiener process. Yamada et al. (1994) presented the first SDE based SRGM by assuming $b(s) = b$ in (1.11) and showed that the solution to the SDE along with boundary condition $N(0) = 0$ resulted in

$$\begin{aligned} N(s) &= a(1 - \exp(-bs - \sigma W(s))) && (1.12) \\ E(N(s)) &= a\left(1 - \exp(-bs + \frac{\sigma^2 s}{2})\right) && (1.13) \end{aligned}$$

Yamada, Nishigaki and Kimura (2003) proposed another model using (1.11) with $b(s)$ given by

$$b(s) = \frac{b^2}{1 + bs}$$

which resulted in what they called as the Delayed S-Shaped SRGM. SDE based software reliability models have been argued to model the stochastic nature of software usage and testing effort common in open-source software development. See Tamura and Yamada (2007) for an example.

Machine Learning based Software Reliability Models: The assumptions of many of the software reliability models presented till now have been criticized as being too restrictive or hard to justify (Yang and Li, 2007). This has led to data-driven approaches based on machine learning techniques such as Artificial Neural Networks (ANN's) and Support Vector Machines (SVM's). Their objective is to model the cumulative defect counts $N(s)$ as a function of total execution time s by leveraging the ability of ANN's and SVM's to model complex transfer functions. Cai et al. (2001) presented examples where an ANN based software reliability model performed better than parametric software reliability models. Yang and Li (2007) presented a software reliability growth model based on support vector machine regression model that predicted the i^{th} inter-failure time T_i as a function of $T_{i-1}, \ldots, T_{i-k}$ for a pre-determined lag k ensuring the model only used the most recent failure data in the prediction. Methods for determining uncertainty in the predictions are typically lacking and model over-fitting may be a cause for concern in these models.

1.4. Summary of Chapters

Parametric Models of Software Reliability (Chapter 2): Parametric models of software reliability (SR) based on failure time between consecutive failures (inter-failure times) are widely used. We present in detail some well-known models and discuss their shortcomings. We then propose two new classes of software reliability models; the first class of models extends well-known parametric SR models by considering dependence between the inter-failure times. The second class of models are a very generic class of SR models based on Isotonic regression involving Exponential, Gamma and Weibull distributional families. These models can also consider dependence between the inter-failure times. We discuss estimation methods for both classes of models, note the lack of large sample properties such as consistency and the difficulty in establishing the distribution of the estimators. Finally we compare the performance of maximum likelihood (ML) estimators of the proposed isotonic software reliability models with those obtained through parametric models using simulations. We find that the isotonic software reliability models can outperform parametric models for software reliability when the underlying parametric model is incorrect.

Independent Isotonic Software Reliability Models (Chapter 3): For the isotonic software reliability model involving scale-families such as Exponential, Weibull or Gamma distributions proposed in Chapter 2, the Pooled Adjacent Violator Algorithm (PAVA) can be used to estimate the failure time distribution of the latest version of the software through a maximum likelihood procedure. Obtaining the distribution of the estimator and subsequently confidence intervals for the reliability for the latest version of the software can be challenging. In this chapter, we propose a simple one-sided confidence bound, with a minimum coverage probability for the scale parameter based on the PAVA estimate. The proposed method makes use of neither asymptotics nor bootstrap procedures and is based on *establishing the PAVA estimator as a function of a sub-martingale process*. The performance of this method has been investigated through simulation. Applications of the proposed confidence bounds are illustrated on a software reliability data sets including a dataset retrieved from the bug-database of a popular scripting language.

Dependent Isotonic Software Reliability Models (Chapter 4): The generic nature of the dependent isotonic software reliability models proposed in Chapter 2 can be used to model failure data from a wide range of testing procedures. The methods developed in Chapter 3 make use of the independence of the inter-failure times and hence cannot be used when the inter-failure times are dependent. Also, they cannot considering right censoring of the failure times. To solve these problems, we propose a novel statistic, based upon the maximum of the observed inter-failure times, which is proved to be a confidence bound for the reliability of the latest version of the software with a minimum coverage probability. If needed, a method for ensuring the monotonicity of the confidence bounds is also proposed. The methods developed are studied through simulations and also applied to datasets relating to software debugging and a new dataset derived from an automated error logger.

Semi-Parametric Models for Analysis of Post-Release Software Defect Data (Chapter 5): Software bug-databases provide an important source of data for assessing the reliability of a software product after its release. Statistical analysis of these databases can be challenging when software usage is unknown, that is, there is no information about the usage, either in the form of a parametric model, or in the form of actual measurements. Reliability metrics, when defined on a calender time scale, would depend on this unknown and time-dependent usage of the software and hence cannot be estimated. This chapter proposes a semi-parametric analysis that makes use of defect classifications into multiple types to enable estimation of a model without making strict assumptions about the underlying usage of the software. New reliability metrics, based on number of failures rather than the calendar time, are introduced. The computation of these new reliability metrics do not depend on the unknown usage of the software. A method based on partial likelihood has been developed for estimating the model parameters and subsequently the reliability metrics. The performance of this method has been investigated through simulation. The proposed method has been illustrated using data retrieved from the bug-database of a popular scripting language, named Python. This illustration compares reliability of two versions of the software without making assumptions about their unknown usage.

Future Work (Chapter 6): Future research directions based on the thesis are presented. We discuss the joint analysis of post and pre-release software defect data under the semi-parametric model presented in Chapter 5. We also discuss non-parametric specification of the reliability improvement function in the same model. Finally we discuss applications of the methods in areas other than software reliability.

1.5. Summary of Relevant Statistical Techniques

Some of the key statistical concepts and techniques that will be used in the thesis are briefly discussed here.

1.5.1. The Bootstrap Method for Estimating Standard Errors

Efron and Tibshirani (1986) presented a novel numerical approach for computing the distribution of estimators which were sophisticated functions of the observed data. They called this the bootstrap method. They proposed two types of bootstrap; the parametric and the non-parametric bootstrap.

The Non-parametric Bootstrap: Let $X_1, \ldots, X_n \overset{iid}{\sim} F(;\theta)$ with $\mathbf{x} = (x_1, \ldots, x_n)$ being a realization from $(X_1, \ldots, X_n)$. Let $\hat{\theta}_n(\mathbf{x})$ be a consistent estimator of θ. The standard deviation of the distribution of $\hat{\theta}_n(\mathbf{x})$, under $F(;\theta)$, is of interest for providing a confidence interval for θ and testing hypotheses regarding θ. In general, the distribution of $\hat{\theta}_n(\mathbf{x})$ can be difficult to derive algebraically. Efron and Tibshirani (1986) approximated this distribution by the distribution of $\hat{\theta}_n(\mathbf{x})$ under the empirical distribution of the observed $x_1, \ldots, x_n$, i.e., $\hat{F}_n(x) = 1/n \sum_1^n I(x_i \leq x)$. This approximation is computed numerically using Monte-Carlo simulations. The algorithm involving B bootstrap samples is presented in Algorithm 1.1. Efron and Tibshirani (1986) state that $B = 500$ is sufficient for estimating the distribution of

$\hat{\theta}_n$. However, depending on the computing power available, one can choose higher values of B.

Algorithm 1.1. The Non-parametric Bootstrap

1. Generate the b^{th} bootstrap sample, $\mathbf{x}^*(b) = (x_1^*, \ldots, x_n^*)$, from the empirical distribution $\hat{F_n}$ by a simple random sampling with replacement from $(x_1, \ldots, x_n)$.
2. Compute $\hat{\theta}_n^*(b) = \hat{\theta}_n(x^*(b))$.
3. Iterate through steps 1 and 2 over B times generating $\hat{\theta}_n^*(b)$ for $b = 1, \ldots, B$.
4. Compute $\hat{\theta}^* = 1/B \sum_1^B \hat{\theta}_n^*(b)$.
5. Compute the standard error of $\hat{\theta}_n$ as

$$SE(\hat{\theta}_n)^* = \frac{1}{B} \sum_{b=1}^{B} \left(\hat{\theta}_n^*(b) - \hat{\theta}^* \right)^2. \tag{1.14}$$

The Parametric Bootstrap: In the non-parametric bootstrap, the distribution of the data $X_1, \ldots, X_n$ is approximated by its empirical distribution $\hat{F_n}$. Alternatively, one can approximate the distribution by $F(.;\hat{\theta}_n(\mathbf{x}))$ which will provide a smooth estimate of $F(.;\theta)$ when compared to $\hat{F_n}$. Another advantage of this approximation is that $X_1, \ldots, X_n$ need not be independent as $F(;\theta)$ may specify the joint-distribution of the X_i's with $F(.;\hat{\theta}_n(\mathbf{x}))$ being its estimate. The Parametric Bootstrap is described in Algorithm 1.2.

Algorithm 1.2. The Parametric Bootstrap

1. Generate the b^{th} bootstrap sample, $\mathbf{x}^*(b) = (x_1^*, \ldots, x_n^*)$, from $F(.;\hat{\theta}_n(\mathbf{x}))$.
2. Compute $\hat{\theta}_n^*(b) = \hat{\theta}_n(x^*(b))$.
3. Iterate through steps 1 and 2 over B times generating $\hat{\theta}_n^*(b)$ for $b = 1, \ldots, B$.
4. Compute $\hat{\theta}^* = 1/B \sum_1^B \hat{\theta}_n^*(b)$.

5. Compute the standard error of $\hat{\theta}_n$ as

$$SE(\hat{\theta}_n)^* = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{\theta}^*_n(b) - \hat{\theta}^*\right)^2. \tag{1.15}$$

The parametric bootstrap is used in Chapters 3,4 and 5.

1.5.2. Inconsistency of the Bootstrap

Andrews (2000), in a seminal paper, presented a simple example where the distribution of the MLE is not consistently estimated by the non-parametric bootstrap of Efron and Tibshirani (1986). He made the following statement.

The bootstrap is not consistent if the parameter is on a boundary of the parameter space defined by linear or nonlinear inequality or mixed inequality or equality constraints.

Andrews provided the following example to substantiate his statement: Let $X_1,\ldots,X_n$ be independent and identically distributed as $N(\mu,1)$ with $\mu \geq 0$. The MLE of μ denoted by $\hat{\mu}_n = max(1/n\sum_1^n X_i, 0)$. Andrews showed that

$$\sqrt{n}(\hat{\mu}_n - \mu) \xrightarrow{d} Z, \text{ if } \mu > 0 \tag{1.16}$$

$$\sqrt{n}(\hat{\mu}_n - \mu) \xrightarrow{d} max(Z,0) \text{ if } \mu = 0, \tag{1.17}$$

where Z is the standard normal variate. The non-parametric bootstrap may be considered for estimating the distribution of $\hat{\mu}_n$. Let $X_i^* \overset{iid}{\sim} \hat{F}_n$, for $i = 1,\ldots,n$, where $\hat{F}_n$ is the empirical cumulative distribution function of the observations; i.e., $\hat{F}_n(x) = 1/n\sum_1^n I(x \leq X_i)$. Let $\overline{X}_n^* = 1/n\sum_1^n X_i^*$. Let $\hat{\mu}_n^* = max(\overline{X}_n^*, 0)$ be the random variable representing the ML estimate of μ from the bootstrap sample. The bootstrap probability distribution of $\hat{\mu}_n^*$, under $\hat{F}_n$, can be used to approximate the distribution of $\hat{\mu}_n$ only if the distribution of $\sqrt{n}(\hat{\mu}_n^* - \hat{\mu}_n)$, given $\hat{F}_n$, and the distribution of $\sqrt{n}(\hat{\mu}_n - \mu)$, for all $\mu \geq 0$, are the same, as $n \to \infty$. Andrews showed that, for $\mu = 0$, the distribution of $\sqrt{n}(\hat{\mu}_n^* - \hat{\mu}_n)$, given $\hat{F}_n$ and for large n, will be stochastically smaller than that of $max(Z,0)$, which is the asymptotic distribution of $\sqrt{n}(\hat{\mu}_n - \mu)$. In other words, the result states that, even if the exact bootstrap probability distribution of $\sqrt{n}(\hat{\mu}_n^* - \hat{\mu}_n)$, given $X_1,\ldots,X_n$ and $\hat{F}_n$, is computed (say, by using a very large number of bootstrap sample or theoretically), it would be an inconsistent estimator of the distribution of the MLE.

We reproduce the proof from Andrews (2000). When $\mu = 0$, by the law of iterated logarithms, there exists $c > 0$ s.t.

$$P\left(\bigcup_{m=1}^{\infty}\bigcap_{n=m}^{\infty}\{\sqrt{n}\overline{X}_n < -c\}\right) = 1.$$

This implies that, for every ω in the sample space, there exists a subsequence $\{n_k\}_{k\geq 1}$ s.t. $\sqrt{n_k}\overline{X}_{n_k}(\omega) < -c$ for all $k \geq 1$. Note that the bootstrap distribution of $\sqrt{n_k}(\hat{\mu}_{n_k}^* - \hat{\mu}_{n_k})$, given ω and hence $\{\hat{F}_{n_k}(\omega)\}_{k\geq 1}$, satisfies the following:

$$\begin{aligned}
&\sqrt{n_k}\left(\hat{\mu}_{n_k}^* - \hat{\mu}_{n_k}(\omega)\right)\\
&\qquad = max\left\{\sqrt{n_k}\left(\overline{X}_{n_k}^* - \overline{X}_{n_k}(\omega)\right) + \sqrt{n_k}\overline{X}_{n_k}(\omega), 0\right\} - max\left\{\sqrt{n_k}\overline{X}_{n_k}(\omega), 0\right\}\\
&\qquad \leq max\left\{\sqrt{n_k}\left(\overline{X}_{n_k}^* - \overline{X}_{n_k}(\omega)\right) - c, 0\right\}\\
&\qquad \xrightarrow{d} max(Z-c,0) \text{ as } k \to \infty\\
&\qquad \overset{st}{\leq} max(Z,0).
\end{aligned}$$

Note that the convergence in distribution is due to $E_{\hat{F}_{n_k}}(\overline{X}_{n_k}^*) = \overline{X}_{n_k}(\omega)$ and the triangular array central limit theorem. Since $c > 0$, the last stochastic inequality is strict with positive probability. Since this holds for almost every ω in the sample space with probability 1, we cannot have $\sqrt{n}(\hat{\mu}_n^* - \hat{\mu}_n) \xrightarrow{d} max(Z,0)$, which implies that the bootstrap based distribution of $\sqrt{n}(\hat{\mu}_n^* - \hat{\mu}_n)$ is an inconsistent estimator for the distribution of $\sqrt{n}(\hat{\mu}_n - \mu)$, when $\mu = 0$. The same methodology can be used to create counter-examples for other problems of parameter estimation where the unknown parameter could lie on the boundary of a parameter space defined by non-strict inequality constraints. Andrews mentioned about the existence of a similar problem for the parametric bootstrap. This result is referred to in Chapters 3 and 4.

1.5.3. Partial Likelihood

We now present a summary of the theory of partial likelihood as presented in Cox (1975). Let y be a vector of observations from a random variable Y with density $f_Y(y;\phi)$ with $\phi = (\theta, \eta)$. If we can transform y into (v, w), with the transformation not depending on the parameter θ, such that, the likelihood of the observation y can be written as

$$f_Y(y;\phi) = f_V(v;\phi) \times f_{W|V=v}(w;\phi), \tag{1.18}$$

with the transformation of y in to (v, w) ensuring that the second product in (1.18) depends only on θ, then $f_{W|V=v}(w;\phi) = f_{W|V=v}(w;\theta)$ is called a partial likelihood of θ. If the information on θ in the first part, i.e., $f_V(v;\phi)$ is non-extractable in the absence of any information on η, and the sole objective is estimation of θ, then, according to Cox (1975), $f_{W|V=v}(w;\theta)$ can be maximized to obtain what is called maximum partial likelihood estimate of θ. The method was proposed by Cox to eliminate the estimation of a nuisance parameter η which could possibly be infinite dimensional. Cox (1972) proposed this method for estimating the regression effects in his proportional hazards model for analysis of life-table data without having to estimate the unknown and arbitrary baseline hazard. Note that, by maximizing only $f_{W|V=v}(w;\theta)$, we are ignoring the first part in (1.18) which also depends on θ. Wong (1986) formally established that the partial likelihood estimator had many of the desired large sample properties of the Maximum likelihood estimators such as consistency and $\sqrt{n}$ normality. We state these results here. Another definition of partial likelihood as presented in Wong (1986) is as follows: Let $\mathbf{y} = (w_1, x_1, \ldots, x_n, w_n)$, the transformation chosen such that the full likelihood of the observations can be written as

$$f_\phi(\mathbf{y};\phi) = \left(\prod_{k=1}^{n} f_\phi(w_k|d_k)\right) \times \left(\prod_{k=1}^{n} f_\theta(x_k|c_k)\right), \tag{1.19}$$

where $d_k = (w_1, x_1, \ldots, w_{k-1}, x_{k-1})$ and $c_k = (w_1, x_1, \ldots, w_{k-1}, x_{k-1}, w_k)$. The second part of the factorization in (1.19) depending solely on θ is the partial-likelihood of θ. Wong assumed that the parameter space, $\Theta \ni \theta$, is a compact set.

Consistency of Partial Likelihood Estimator: The asymptotic consistency of the maximum partial likelihood estimator is stated in Theorem 2E of Wong (1986) as follows: Define

$$r_k(\theta) = log\left(f_{\theta_0}(x_k|c_k)/f_\theta(x_k|c_k)\right), \qquad R_n(\theta) = \sum_{k=1}^{n} r_k(\theta) \tag{1.20}$$

$$i_k(\theta) = E_{\theta_0}\left(r_k(\theta)|c_k\right), \qquad I_n(\theta) = \sum_{k=1}^{n} i_k(\theta) \tag{1.21}$$

$$j_k(\theta) = \underset{\theta_0}{Var}\left(r_k(\theta)|c_k\right), \qquad J_n(\theta) = \sum_{k=1}^{n} j_k(\theta) \tag{1.22}$$

$$m_k(\theta) = r_k(\theta) - i_k(\theta), \qquad M_n(\theta) = \sum_{k=1}^{n} m_k(\theta) \tag{1.23}$$

Theorem 1.3. *Assume that for any $\theta \neq \theta_0$ there exists an open neighborhood of θ whose closure, G_θ, does not contain θ_0 and there exists constants $\delta > 0$ and a* *sequence $a_n \to \infty$ such that*

$$a_n^{-2} J_n(\theta') \xrightarrow{p} 0, \forall \theta' \in G_\theta, \tag{1.24}$$

$$P\left(\inf_{\theta' \in G_\theta} a_n^{-1} I_n(\theta') > \delta \right) \to 1, \tag{1.25}$$

$$P\left(\sup_{\theta' \in G_\theta} a_n^{-1} || \bigtriangledown M_n(\theta') || > K \right) \to 0, \quad \textit{for some } K > 0, \tag{1.26}$$

$$P\left(L_n(\theta) \textit{ is strictly concave in } \theta\right) = 1 \textit{ for all } n, \tag{1.27}$$

where $\bigtriangledown M_n(\theta)$ is the Fretchet derivative of $M_n(\theta)$ and $L_n(\theta)$ is the log partial likelihood. Then, the maximum partial-likelihood estimate $\hat{\theta}_n \xrightarrow{p} \theta$.

Asymptotic Normality of Partial Likelihood Estimator of θ_0: Define

$$l_k(\theta) = \log(f_\theta(x_k|c_k)); \, l_k^{(k)}(\theta) = \frac{d^k l_k(\theta)}{dx^k}; \, L_n^{(k)}(\theta) = \sum_{n=1}^{N} l_n^{(k)}(\theta)$$

and

$$v_n(\theta) = \underset{\theta_0}{Cov}(l_n(\theta)|c_n); \, V_n(\theta) = \sum_{n=1}^{N} v_n(\theta)$$

The asymptotic normality of the partial maximum likelihood estimator is stated as in Theorem 4A of Wong (1986) as follows:

Theorem 1.4. *Suppose $\hat{\theta}_n$ is consistent for $\theta_0 \in$ interior of Θ and there exists constants $a_n \to \infty$ and a neighborhood $O(\theta_0)$ with*

$$a_N^{-1} V_n \xrightarrow{p} Q, \tag{1.28}$$

$$-a_N^{-1} L_n^{(2)}(\theta_0) \xrightarrow{p} Q_1, \tag{1.29}$$

$$P\left(a_n^{-1} \sup_{\theta \in O(\theta_0)} |L_n^{(3)}(\theta)| < M \right) \to 1, \textit{ for some } M > 0, \tag{1.30}$$

$$a_n^{-3/2} \sum_{k=1}^{n} E(||l_k(\theta)||^3 | c_k) \xrightarrow{p} 0. \tag{1.31}$$

Then,

$$a_n^{1/2}(\hat{\theta}_n - \theta_0) \xrightarrow{d} N(0, Q_1 Q^{-1} Q_1). \tag{1.32}$$

Note that in many situations, $Q_1 = Q$ resulting in

$$a_n^{1/2}(\hat{\theta}_n - \theta_0) \xrightarrow{d} N(0, Q). \tag{1.33}$$

Partial Likelihood based model estimation is used in Chapter 5.

1.5.4. Copulas: A Brief Discussion

A continuous copula is a multivariate distribution of the random variables $\{U_1, U_2, \ldots, U_n\}$ such that each U_i, for $i = 1, \ldots, n$, has a $Uniform(0,1)$ marginal distribution. Copulas are useful for modeling multivariate random variables, $\mathbf{X} = \{X_1, \ldots, X_n\}$, whose components are dependent and with each component having a specified marginal distribution. For example, we could use a Gaussian copula to construct a dependent multivariate random variable $\mathbf{X} = (X_1, X_2)$ with the $X_1 \sim Exponential(1)$ and $X_2 \sim LogNormal(0,1)$ as follows: Generate Z_1, Z_2 from a bivariate normal distribution with mean zero, standard deviations of one and correlation between Z_1 and Z_2 set to $\rho = 0.5$. Compute $U_1 = \Phi(Z_1), U_2 = \Phi(Z_2)$, where $\Phi(.)$ is the CDF of the standard normal distribution. Note that U_1, U_2 both have uniform marginal distribution. Compute $X_1 = -log(1 - U_1)$ and $X_2 = \exp(\Phi^{-1}(U_2))$. Since U_1, U_2 are both marginally distributed as $Uniform(0,1)$, we will have X_1 and X_2 distributed marginally as $Exponential(1)$ and $LogNormal(0,1)$, respectively. The dependence between U_1 and U_2 induces dependence between X_1 and X_2. Any multivariate distribution with continuous marginals can be used to generate a copula; this is due to

Sklar's theorem, a simplified version of which is presented as stated in Embrechts, Klüppelberg and Mikosch (1997)

Theorem 1.5. ***(Sklar's Theorem):*** *For any set of random variables $X_1,\dots,X_n$ with continuous marginal cumulative distribution functions (CDF's), $F_1,\dots,F_n$, and a joint cumulative distribution F, there exists a unique copula, $C(...)$, which is a cumulative distribution function on $[0,1]^n$ such that for all $\mathbf{x}=(x_1,\dots,x_n)$,*

$$F(x_1,\dots,x_n)=C(F_1(x_1),\dots,F_n(x_n)) \tag{1.34}$$

Conversely, given any marginal CDF's, $F_1,\dots,F_n$, and a copula C, the CDF $F(...)$ defined through (1.34) *is a multivariate CDF with marginals $F_1,\dots,F_n$.*

Sklar's Theorem enables the construction of a copula out of a multivariate distribution for which the dependence between the components can be easily specified and interpreted. The multivariate normal distribution with zero mean and unit variance for each component and *correlation* matrix R is used to construct the Gaussian copula. The T-copula, Archimedian copula and hyperbolic copulas are some of the other popularly used copulas. A particular form of a Gaussian copula, which is important for time series applications, is a copula derived from a Gaussian Auto-regression process (Manner and Reznikova, 2012). Copulas are used for defining new software reliability models in Chapter 2.

1.5.5. The Pooled Adjacent Violators Algorithm

Let $Y_1,\dots,Y_n$ be a sequence of independent random variables with $Y_i \sim F(.|\theta_i)$ for some real valued parameter θ_i. Denote $f(.|\theta_i)$ to be the PDF of $F(.|\theta_i))$. Let y_i be a realization of Y_i. The problem of estimating θ_i's, subject to the constraint

$\theta_1 \leq \ldots \leq \theta_n$, through the method of maximum likelihood has a variety of applications and was first considered by Ayer et al. (1955). The pooled adjacent violators algorithm was proposed by Ayer et al. to solve this problem and is described as follows:

Algorithm 1.6. The Pooled Adjacent Violators Algorithm

1. Denote $B(i,k)$ to be the block assigned to the observation y_i during the k^{th} iteration of the algorithm. Initialize the algorithm by assigning $B(i,0) = i$, for, $i = 1,\ldots,n$. Initialize $k = 0$.
2. Determine
$$\hat{\theta}_i^{(k)} = \underset{\theta}{arg\,max} \prod_{j:B(j,k)=B(i,k)} f(y_j|\theta), \text{ for } i = 1,\ldots,n.$$
3. If $\hat{\theta}_i^{(k)}$'s are all non-decreasing in i, then $\hat{\theta}_i^{(k)}$'s are the ML estimates of θ_i's.
4. If not, then increment $k = k+1$. For each $j = n, n-1, \ldots, 2$, determine if $\hat{\theta}_j^{(k-1)} < \hat{\theta}_{j-1}^{(k-1)}$ and assign $B(j',k) = B(j-1,k)$, $\forall j' \geq j$ with $B(j',k) = B(j,k)$. This is the pooling of adjacent violators into the same block.
5. Repeat Step 2.

A closed form expression for the PAVA estimate, useful for analytical purposes, is given by Robertson, Wright and Dykstra[pp-23] and presented in (1.35).

$$\hat{\theta}_i = \min_{j:j\geq i} \max_{k:k\leq i} \underset{\theta}{arg\,max} \prod_{l=k}^{l=j} f(y_l|\theta). \tag{1.35}$$

The PAVA algorithm is used in Chapters 2,3 and 4.

Chapter 2

Parametric Models for Software Reliability

2.1. Introduction

Pay attention to zeros. If there is a zero, someone will divide by it.

Cem Kaner

Basu and Ebrahimi (2003) defined software reliability as the “Probability of Failure Free” operation of a software product. This probability may be over the next use case of the software product or over a specified period of time. A key difference between software and hardware reliability is the absence of the notion of ageing. Once a software product fails, there is a possibility of detecting the cause of the failure (popularly called as a software bug) and repairing the bug (known as debugging) with a resulting increase in the reliability of the software. Since the software does not age, the increase in reliability should not depend on the age of the software. One would expect an increasing time to failure with the detection and debugging of every additional defect. A software reliability model quantifies this intuitive expectation mathematically. The inter-failure times observed historically can be used to estimate the model. An objective of the software reliability model is to estimate the distribution of the time to next failure of the software and use it to compute the reliability of the software in terms of the probability of failure over a specified period of future usage.

Consider a software product under controlled testing by a professional team of software testers. Let S_i, for $i = 1, 2, \ldots,$ denote the calendar times when defects are discovered since the release of the software. Let $T_1 = S_1, T_2 = S_2 - S_1, \ldots, T_i = S_i - S_{i-1}$, be the inter-failure time for the i^{th} failure with t_i denoting a realization of T_i. A generic software reliability model would compute the conditional distribution of the n^{th} inter-failure time, T_n, given $n, T_1 \ldots, T_{n-1}$. From such a model, one could compute the probability of failure-free operation over any specified duration of usage. The model would need to incorporate the expectation that every additional discovery increases the reliability of the software. In the vast majority of software reliability analysis, the number of failures and therefore the number of inter-failure time observations are limited necessitating the need for a model that is parsimonious in model parameters. This is particularly true for software reliability analysis involving critical failures such as those related to security vulnerabilities. Due to this practical constraint, it is common for software reliability models to assume the conditional distribution of T_i, given $T_{i-1}, \ldots, T_1$, to depend only on i or even on just T_{i-1}.

At this juncture, it is important to discuss how software failure times are measured. In the case of pre-release controlled software testing, the duration to the next failure is measured in terms of the number of test-cases successfully executed by the software product before it fails. An alternative definition would involve the number of days or hours of testing before the software fails. If there is a large team of software testers testing the software, with the team size varying over the testing duration, the number of man-hours of testing as a measurement unit may be more accurate. In the case of post-release software failure, with different customers beginning to use the software from different calendar times, the number of customer-hours, defined as the number of hours of use by all the customers before a failure is reported, may be a natural measure of the inter-failure time. A well-defined and accurate measurement of software operating duration adjusted for the number of users or testers is clearly important for any software reliability modeling and we assume that such a definition exists and is consistently used for all measurements of inter-failure time.

In this chapter, we describe in detail some key software reliability models that have motivated software reliability studies and discuss their shortcomings. The generalization of some such parametric models to incorporate dependent inter-discovery times using copulas induced by time series processes is considered. In particular, we will use the de-eutrophication model, which is one of the most popularly used model (Singpurwalla and Wilson, 1994; Derrenic and Le Gall, 1995; Farr, 1996), given by Moranda (1975), to illustrate the generalization. Copulas have been used in reliability modeling (for example, see Wang, Ephim and Pham; 2012) for modeling degradation process. The method proposed here considers a new approach that is specific to software reliability problems. We also propose a new family of software reliability models, which we term as Isotonic Software Reliability (ISR) models, that offers generalizations of many well-known software reliability models. The ISR model may be considered as an example of a "statistical" software reliability model, whereas the Jelinksi-Moranda model may be considered to be a 'stochastic' model since it has some explanation of the process by which the faults arise. We discuss maximum likelihood analysis of the proposed ISR models. A simulation study is then presented to compare the performance of the proposed ISR models with that of some of the existing parametric models.

A software reliability model needs to consider three modeling aspects when analyzing data corresponding to software failures: The first aspect, corresponding to the Defect Reporting Rate (DRR), represents the efficiency of the software testing and is defined as the intensity of discovering and reporting a defect at time t under the assumption that no defect has been discovered till time t (a kind of baseline reporting rate); the second aspect corresponds to the manner in which the history of previously detected defects affects the rate at which defects will be detected in the future and finally, the third aspect corresponds to stochastic dependence between the inter-discovery times of the defects. Currently, the most popular software reliability models are parametric in nature, considering all the three aspects discussed above in some limited manner. We intend to emphasize this in our further modeling generalizations.

The earliest models of software reliability considered a DRR which was constant with time, assumed a parametric form for the rate at which new defects are detected as a function of the cumulative number of defects already detected and simply considered independence between the inter-discovery times of the defects. The first among these models is the one, proposed by Jelinski and Moranda (1972), which considers a linear improvement in the reliability. The imperfect debugging model of Goel and Okumoto (1978) considered the possibility of a detected defect not being perfectly debugged. The de-eutrophication model of Moranda (1975) is similar to the model of Jelinski and Moranda but considered a geometric improvement in reliability as opposed to a linear improvement. The model of Schick and Wolverton (1978) is specified in terms of the hazard, $\lambda(t)$, of the distribution of the inter-discovery time T_i. One of the first Bayesian models of software reliability was due to Littlewood and Verrall (1973), where they assumed that $T_i \sim Exponential(\lambda_i)$, with prior distributions on λ_i belonging to the Gamma family. In particular, they assumed that $\lambda_i \sim Gamma(\Psi(i), \alpha)$ with a variety of parametric forms for $\Psi(i)$. Mazzuchi and Soyer (1988) gave details of a model which assumes $\Psi(i) = \beta_0 + \beta_1 i$.

The assumption of independence between the observed inter-discovery times has been a cause of concern for many software reliability practitioners, and in spite of this, software reliability models that can account for dependence between the time to discovery of the defects are few in number. The problem with assuming independence between the inter-discovery times is that if the discovery of an important defect takes a long time (or more test cases), it can hasten the discovery of subsequent defects through learnings the testing team has gained from this discovery. For example, consider a software product with modules $M_1, \ldots M_n$; if a security vulnerability, denoted by say S_1, is discovered in module M_1 after a very large number of test cases, there will be a tendency to use the test case that detected the defect S_1 in module M_1 on the other modules which may result in a shorter time to detecting security defects in other modules. In such a situation, the assumption of independence between the inter-discovery times may not be realistic.

The most notable software reliability model that does not assume dependence between the inter-discovery time is the auto-regressive model of
Singpurwalla and Soyer (1985), which assumes the conditional distribution T_i,

given $T_1, \cdots, T_{i-1}$, is log-normal with $log(T_i) = \theta_i log(T_{i-1}) + \varepsilon_i$, for $i = 2, 3, \cdots$, with $log(T_1) = \theta_1 + \varepsilon_1$ and $\varepsilon_1, \varepsilon_2, \ldots$ all being independent standard normals. The parameters θ_i's are assumed to have a distribution that is either assumed to be exchangeable or to be governed by an AR(1) process, with the latter being studied in depth by Singpurwalla and Soyer (1992) using an adaptive Kalman filter.

2.2. Some Existing Parametric Models of Software Reliability

2.2.1. Model of Jelinski and Moranda

The Jelinski and Moranda (JM) model, Jelinski and Moranda (1972), was one of the first models for software reliability. The genesis of the model lies in a well-known stochastic process model for the following pure death process: Let N individuals be confined to a room with N being unknown. An individual can be identified only when the individual "dies", with the time to death for any two individuals being independent of each other and having identical exponential distributions with scale parameter λ. If we denote the calendar time to death of the i^{th} individual as S_i, for $i = 1, \ldots, N$, then the time to the first "death" and hence the first discovery of an individual, denoted by $T_1 = min(S_1, \ldots, S_N)$. If we denote T_i to be the time to the i^{th} discovery of an individual since the $(i-1)^{th}$ discovery of an individual, then $T_i = S_{(i)} - S_{(i-1)}$, for $i = 2, \ldots, N$, with $S_{(i)}$ being the i^{th} order statistic of $S_1, \ldots, S_N$. Since $S_1, \ldots, S_N$ are independent and identically distributed as $Exponential(\lambda)$, we will have $T_i \sim Exponential((N - i + 1)\lambda)$ with T_i and T_j being independent of each other for $i \neq j$. This results is due to a well known property of the exponential distribution. Jelinski and Moranda likened the defects in a software product to the N individuals in the pure death process with their de-

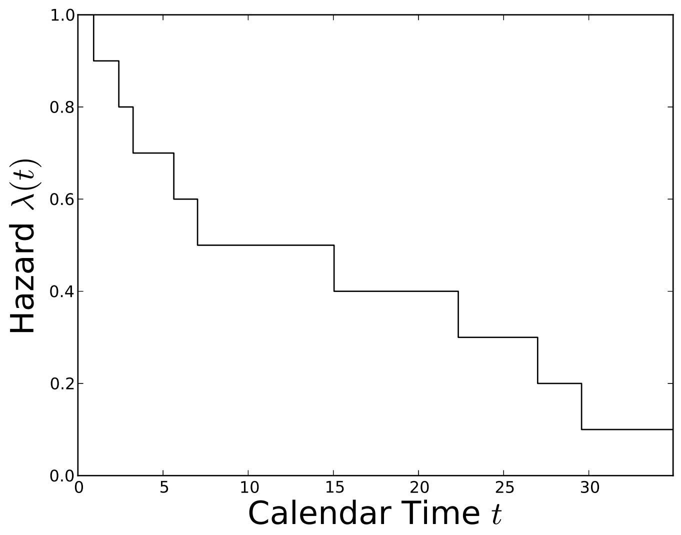


FIG 2.1. *Plot of $\lambda(t)$ vs calendar time t*

tection and debugging corresponding to the "death" of an individual. If one were to ignore the genesis of the model from a pure death process, then the JM model simply states the following:

Model 2.1. (Jelinski and Moranda Model) Let N be the number of defects in a software. Let T_i, for $i = 1,\ldots,N$, be the inter-failure times. Then $T_i \sim Exponential(\lambda(N-i+1))$ with T_i being independent of T_j for $i \neq j$.

To help visualize the model the instantaneous hazard of observing a failure at calendar time t, a plot of $\lambda(t)$ vs. ts for the JM model is presented in Figure 2.1 with $\lambda = 1$ and $N = 10$. There are a couple of properties of interest in the JM

Model: The first property corresponds to the expected value of the i^{th} inter-failure time $E(T_i)$, which keeps increasing with i, the number of detected defects, since

$E(T_i) = 1/\lambda(N - i + 1)$. This can be interpreted to mean that every defect discovery increases the reliability of the software, justifying the cost of finding defects and repairing them. Note that this increase in expectation has a very specific parametric form. The second feature corresponds to the inter-discovery time T_i not depending on the total testing time of the software at the time of its discovery, which represents the lack of ageing of the software. Both these properties have been used in subsequent software reliability models.

Criticisms of the JM Model: One of the key-assumptions of the model is that all defects are equal in terms of their impact on reliability improvement after their discovery and repair; an assumption that may not be realistic. For example, software defects could have different severities, with the discovery and repair of more severe defects increasing the reliability much more than the discovery and repair of less severe defects. The model also assumes that every defect can be fixed perfectly resulting in an immediate improvement in the reliability, an assumption that has often been criticized (See Littlewood and Verrall (1973); Goel and Okumoto (1978); Musa and Okumoto (1984) for some examples). This criticism has lead to the developement of several software reliability models (Singpurwalla and Wilson, 1994). The assumption that the reliability of the software must improve after every defect discovery has been challenged most recently by Basu and Ebrahimi (2003) leading them to propose a Bayesian model for software reliability.

Model Estimation: There are two software test procedures through which software defect data can be generated for estimating the JM model; the first procedure corresponds to testing the software till a pre-determined number of, say, n defects are discovered while the second procedure would test the software for a fixed calendar time duration irrespective of the number of defects discovered. We discuss first the likelihood and model estimation for the first testing procedure for which a typical data set would comprise of the inter-failure times of the n defects. Let $\mathbf{t} = \{t_1, \ldots, t_n\}$ denote the observed inter-failure times of the n defects. If one were to assume the JM model is appropriate for the software under consideration, then the likelihood of observing **t** will be given by

$$L(\mathbf{t}|\lambda, N) = \prod_{i=1}^{n} \lambda(N-i+i)\exp(-\lambda(N-i+1)t_i). \tag{2.1}$$

Jelinski and Moranda showed that if the likelihood in (2.1) had a unique maximum in N and λ, then the solution to finding the ML estimate for N was iterative and given by

$$\sum_{i=1}^{N} \frac{1}{N-i+1} = \frac{n}{N - \frac{1}{S_n}\sum_{i=1}^{n}(i-1)t_i}, \tag{2.2}$$

where $S_n = \sum_{i=1}^{n} t_i$, $N \geq n$. Once the ML estimate $\hat{N}$ of N is obtained, the ML estimate of λ can be obtained as

$$\hat{\lambda} = \frac{n}{\hat{N}S_n - \sum_{i=1}^{n}(i-1)t_i}. \tag{2.3}$$

If the software is tested for a specified period of time S, irrespective of the number of defects discovered, then the likelihood of the observation $(n, t_1, \ldots, t_n)$ will be

$$L(n, t_1, \ldots, t_n|\lambda, N) = \prod_{i=1}^{n} \lambda(N-i+i)e^{-\lambda(N-i+1)t_i} \times e^{-(N-n)\lambda S} \tag{2.4}$$

The likelihood in (2.4) is maximized at $\hat{N}^*$ which satisfies

$$G(\hat{N}^*) \leq \frac{1}{nS}\sum_{i=1}^{n} t_i < G(\hat{N}^* + 1), \tag{2.5}$$

where

$$G(N) = \{n(1 - [1 - (n/N)]^{1/n})\}^{-1} + 1 - (N/n).$$

Then, λ is estimated by $\hat{\lambda}^*$ given by

$$\hat{\lambda}^* = \frac{n}{\sum_{i=1}^{n}(\hat{N}^* - i + 1)t_i + ((\hat{N}^* - n)S)}. \tag{2.6}$$

Properties of the ML estimate $\hat{N}$ and $\hat{N}^*$: Blumenthal and Marcus (1975) have studied the properties of $\hat{N}$, obtained from (2.3), in detail. They show that $\sqrt{N}(\hat{N} - N) \sim Normal(0, \sigma)$ as $N \to \infty$ with σ being a function of the proportion of defects discovered $\delta = n/N$ and λ. The likelihood in (2.1) can sometimes be unbounded with $\hat{N} = \infty$. This has once again been studied by Blumenthal and Marcus where they compute the probability of this happening as a function of N and n. They note that $Prob(\hat{N} = \infty)$ can sometimes be as high as 0.40 for $N = 40$ and $n = 10$. Blumenthal and Marcus also note that $\hat{N}$ can be highly biased, especially when λ is very small and for moderately large values of N. Blumenthal and Marcus also showed similar results for $\hat{N}^*$ obtained through maximizing (2.4). In particular, they showed that for large N, $\hat{N}^*$ is consistent and $\sqrt{N}(\hat{N}^* - N) \sim Normal(0, \sigma)$, with σ being a function of λ and S.

2.2.2. The Imperfect Debugging Model of Goel and Okumoto

The Jelinski and Moranda (JM) model assumes that every failure corresponds to a unique software defect which can be perfectly fixed and will result in an improvement in the reliability. This assumption has been challenged by Goel and Okumoto

(1978) and they proposed a modification to the JM model by introduction of a parameter p, with $0 < p \leq 1$, which represents the probability that a defect has been fixed. The parameter p can be interpreted as the efficiency of debugging, with higher values representing better and more efficient debugging.

Model 2.2. (The Imperfect Debugging Model) Let N be the number of defects in a software. Let T_i, for $i = 1, \ldots, N$, be the inter-failure times. Then $T_i \sim Exponential(\lambda(N - p(i-1)))$ with T_i being independent of T_j for $i \neq j$.

The reason that we discuss this model is because it represents the first attempt by researchers at recognizing that fixing a defect need not necessarily improve the reliability of the software. In fact, the model suggests that, even after fixing N defects, if $p \neq 1$, then the software may still have a failure in the future. We refer to Goel and Okumoto (1978) for details regarding the model estimation which are very similar to that of the JM model.

2.2.3. The Model of Schlick and Wolverton

The JM model bestows a memory-less property to a software testing procedure by assuming exponential inter-failure times. A testing team that is monetarily rewarded for every software failure discovered can be expected to increase their testing intensity if no failure is observed by them for a long duration. Such a tendency would violate the memory-less assumption of the defect discovery process. Alternatively, immediately after a software failure is observed, there might be a tendency for the testing team to spend time analyzing the failure, leading to a temporary decrease in their testing intensity. Such possibilities require a model in which the hazard, $\lambda(s)$, of observing a software failure at calendar time s is not a constant. As a result, the inter-failure times are not exponentially distributed. The model of Schick and Wolverton (Schick and Wolverton, 1978) represents the first attempt at a software reliability model where the inter-failure time is assumed to be non-exponentially distributed.

Model 2.3. (Schick and Wolverton Model) Let N be the number of defects in a software. Let T_i, for $i = 1, \ldots, N$, be the inter-failure times. The hazard of observing the i^{th} failure at time $T_i = t$ since the $(i-1)^{th}$ failure is given by $\lambda_i(t) = \gamma(N - i + 1)t$.

The model implies that $T_i \sim Rayleigh(\gamma(N - i + 1))$, which is a Weibull distribution with shape parameter 2. The model estimation procedure is very similar to that of the JM model.

2.2.4. The De-Eutrophication Model of Moranda

One of the criticism's of the JM model is that fixing every defect improves the reliability of the software by the same quantum, which can also be interpreted as stating that all defect are equal in terms of their severity. The de-eutrophication model of Moranda (Moranda, 1975) challenges such an assumption and assumes that improvements in reliability due to defects that are detected early on are more than improvement to reliability due to defects detected later on. The heuristic reasoning being severe defects will be detected early on as their hazard of causing a failure will be more than those of less severe defects. The model is stated as follows:

Model 2.4. (De-Eutrophication model of Moranda) Let T_i, for $i = 1, \ldots, N$, be the inter-failure times. Then $T_i \sim Exponential(\exp(\alpha + \beta i))$ with T_i being independent of T_j for $i \neq j$. The parameters of the model are α and β. If $\beta < 0$, then the reliability of the software improves after every additional defect discovery.

Note that this model moves away from describing the reliability of the software in terms of the number of defects in the software. According to this model, the hazard of observing a defect in the future can never be zero irrespective of the number of defects already discovered. The parameter β measures the benefit of discovering and fixing an additional defect; if $\beta < 0$, then there is a positive improvement in the reliability for every additional defect fixed, if $\beta = 0$, there is no effect of discovering and fixing defects in the software and lastly if $\beta > 0$, every additional defect discovery decreases the reliability of the software. By testing for the sign of the β coefficient one may be able to judge whether software testing and debugging are beneficial to improving the reliability of the software.

Method of Moments Estimation: Under this model, $T_i \sim Exponential(\exp(\alpha + \beta i))$ which implies $-\log(T_i) \sim Gumbel(\alpha + \beta i, 1)$, which corresponds to a Gumbel distribution with location parameter $\alpha - \beta i$ and scale parameter of 1. This would in turn imply that $E(-\log(T_i)) = \alpha + \beta i + \gamma$, where $\gamma = 0.5772$ is the Euler-Mascheroni constant and standard deviation (SD) of $-log(T_i)$ is $\pi/\sqrt{6}$. Let t_i be a realization of T_i, for $i = 1, \ldots, n$. A least squares regression of $y_i = -\log(t_i) - \gamma$ against i, for $i = 1, \ldots, n$, would result in estimates $\hat{\alpha}_{LS}$ of α and $\hat{\beta}_{LS}$ of β, respectively. The estimates are given by

$$\begin{pmatrix} \hat{\alpha}_{LS} \\ \\ \hat{\beta}_{LS} \end{pmatrix} = D^{-1} \times \begin{pmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} i y_i \end{pmatrix}, \tag{2.7}$$

$$\text{where, } D = \begin{pmatrix} n & \sum_{i=1}^{n} i \\ \sum_{i=1}^{n} i & \sum_{i=1}^{n} i^2 \end{pmatrix}. \tag{2.8}$$

The estimates $\hat{\alpha}_{LS}$ and $\hat{\beta}_{LS}$ are unbiased estimates of α and β. Also note that the variance-covariance matrix of $(\hat{\alpha}_{LS}, \hat{\beta}_{LS})$ is given by $\pi^2/6 \times D^{-1}$, with D as defined in (2.8). Since T_i's are independent and the minimum eigenvalue of D tends to infinity as $n \to \infty$, the regression satisfies the conditions of Eicker (1963). Hence, the least square estimates are consistent and $\sqrt{n}(\hat{\alpha}_{LS} - \alpha, \hat{\beta}_{LS} - \beta)$ follows asymptotically a bivariate normal distribution with mean zero and variance-covariance matrix $\pi^2/6 \times (D/n)^{-1}$.

Maximum Likelihood Estimation: The log-likelihood under the de-eutrophication model of Moranda for the observations $t_1, \ldots, t_n$ can be written as

$$l(\alpha, \beta) = \prod_{i=1}^{n} (\alpha - \beta i) - \exp(\alpha - \beta i) t_i. \tag{2.9}$$

The score and the hessian matrix for the log-likelihood are given, respectively, by

$$\nabla l(\alpha, \beta) = \sum_{i=1}^{n} \left(1 - \exp(\alpha - \beta i) t_i\right) (1\, i)^T, \tag{2.10}$$

$$\nabla^2 l(\alpha, \beta) = \sum_{i=1}^{n} - \exp(\alpha - \beta i) t_i \,(1\, i) \times (1\, i)^T. \tag{2.11}$$

From (2.11), it is clear that the hessian matrix will be negative definite due to it being a sum of non-negative definite matrices. Hence, the log-likelihood will have a unique maximum and a Newton-Raphson procedure can be used to numerically maximize the log-likelihood to obtain maximum likelihood estimates $\hat{\alpha}_{ML}$ and $\hat{\beta}_{ML}$ of α and β, respectively. For large n, we have $\sqrt{n}(\hat{\alpha}_{ML} - \alpha, \hat{\beta}_{ML} - \beta)$ to be normally distributed with mean zero and a variance-covariance matrix that is estimated by $(\nabla^2 l(\hat{\alpha}_{ML}, \hat{\beta}_{ML}))^{-1}$. This asymptotic result can be used to compute confidence intervals for the parameters from their ML estimates and to test for the significance of the β coefficient.

2.3. Software Reliability Models with Dependence

The inter-discovery times for the defects, $T_1, \ldots, T_n$, have a natural ordering induced by the calendar time sequence of defect discoveries. This suggests that $T_1, \ldots, T_n$ can be modeled as a time series. The time series interpretation is advantageous in allowing specification of dependence between the inter-discovery times. The expectation that software reliability improves with every additional defect discovery suggests that the marginal distribution of T_{n+1} is stochastically larger than T_n. A simple way of introducing dependence in a software reliability model, while still maintaining the requirement of software reliability improvement, is to use copulas induced by a time series processes. We will use the de-eutrophication model as an example to demonstrate our modeling (See Section 2.3.1).

The purpose of a dependent software reliability model would be to estimate the conditional distribution of $T_{n+1}|T_1, \ldots, T_n$. Such an interpretation has been considered by Singpurwalla and Soyer (1985) for proposing an auto-regressive model for software reliability. While their model considers dependence, it is at odds with software reliability models such as the Jelinski and Moranda model since T_i is no longer distributed as $Exponential(\lambda(N-i+1)$.

For a software test data-set consisting of the inter-discovery times $T_1,\ldots,T_n$, a software reliability model usually specifies the distribution of T_i as a function of i. In order to preserve the specified marginal distributions of T_i's while allowing dependence between them, we propose using copulas. In particular, a Gaussian copula, constructed using a stationary auto-regressive process of order p, may seem well-suited for characterizing the nature of dependence between the T_i's. Recall that an auto-regressive process, $AR(p)$ of order p, is determined by the parameters $\rho_1,\ldots,\rho_p$ with ρ_i representing the i^{th} order auto-regression coefficient. A sequence of random variables, $V_1,\ldots,V_n$, is said to follow $AR(\rho_1,\ldots,\rho_p)$ when V_i's denote random variables corresponding to n successive observations from an $AR(p)$ process. A dependent software reliability model that makes use of an $AR(p)$ process is presented as follows:

Model 2.5. (Gaussian Copula based Software Reliability) Let $T_1,\ldots,T_n$ be the inter-failure times of n successive failures of a software product. Then,

$$T_i \sim F(.|\Theta,i), \text{ for } i=1,\ldots,n,$$
$$\text{with } \left(\Phi^{-1}(F^{-1}(T_1|\Theta,1)),\ldots,\Phi^{-1}(F^{-1}(T_n|\Theta,n))\right) \sim AR(\rho_1,\ldots,\rho_p).$$

Note that $\Phi(.)$ is the standard normal CDF and $F(\Theta,i)$ represents the marginal CDF of T_i as a function of the model parameters Θ and i. By specifying the dependence between the T_i's through a copula induced by a Gaussian AR process, one can consider the possibility of dependence between the inter-discovery times. Under this modeling framework, one may also consider a higher order AR process to allow for long term dependence between the T_i's.

Maximum Likelihood Estimation Assume that $f(t,\Theta,i)$, the density function corresponding to $F(.|\Theta,i)$, exists and is twice differentiable over the non-negative part of the real line. Let t_i be a realization of T_i. The log-likelihood of the observations, t_1,...,t_n, can be expressed in terms of $\mathbf{Z} = \{\Phi(F^{-1}(t_1|\Theta,1)),\ldots,\Phi(F^{-1}(t_n|\Theta,n))\}$ as follows:

$$\begin{aligned} l(\Theta,\rho,\Sigma) &= \sum_{i=1}^{n} \log(f(t_i|\Theta,i)) + \log(C_\rho(Z)), \\ \log(C_\rho(\mathbf{Z})) &= -\frac{p}{2} log(|\Sigma|) - Z'\Sigma^{-1}Z, \end{aligned}$$

with Σ being the covariance matrix of a p^{th} order Gaussian auto-regressive process and $\rho = (\rho_1,...,\rho_p)$. Numerical procedures need to be used for maximization of $l(\Theta,\rho,\Sigma)$ over Θ,ρ and Σ. The existence of a unique maximum would depend on the form of $f(.|\Theta,i)$ and the copula used.

2.3.1. De-Eutrophication Model with Dependence

A specific example of the Gaussian Copula based software reliability model would specify $F(.|\Theta,i)$ through the de-eutrophication model of Moranda. This is stated as follows:

Model 2.6. (De-Eutrophication Model with Dependence) Let $T_1,\ldots,T_n$ be the inter-failure times of n successive failures of a software product. Then,

$$T_i \sim Exponential(\alpha+\beta i),\ \text{for } i=1,\ldots,n,$$
$$\text{with } \left(\Phi^{-1}(1-e^{e^{\alpha+\beta 1}T_1}),\ldots,\Phi^{-1}(1-e^{e^{\alpha+n\beta}T_n})\right) \sim AR(\rho_1,\ldots,\rho_p).$$

The proposed model can be estimated through the method of maximum likelihood or through the method of moments. We will discuss the latter first.

Method of Moments Estimation Under the de-eutrophication model with dependence, $T_i \sim Exponential(\exp(\alpha+\beta i))$ which implies $-\log(T_i) \sim Gumbel(\alpha+\beta i, 1)$, which corresponds to a Gumbel distribution with location parameter $\alpha+\beta i$ and scale parameter of 1. This in turn implies that $E(-\log(T_i)) = \alpha+\beta i+\gamma$, where $\gamma \approx 0.5772$ is the Euler-Mascheroni constant and standard deviation of T_i is $\pi/\sqrt{6}$. A least squares regression of $Y_i = -\log(T_i)-\gamma$ against i, for $i=1,\ldots,n$, would result in estimates $\hat{\alpha}_{LS}$ of α and $\hat{\beta}_{LS}$ of β, respectively. The estimates are as follows:

$$\begin{pmatrix} \hat{\alpha}_n^{LS} \\ \\ \hat{\beta}_n^{LS} \end{pmatrix} = \begin{pmatrix} n & \sum_{i=1}^{n} i \\ \\ \sum_{i=1}^{n} i & \sum_{i=1}^{n} i^2 \end{pmatrix}^{-1} \times \begin{pmatrix} \sum_{i=1}^{n} y_i \\ \\ \sum_{i=1}^{n} i y_i \end{pmatrix}, \tag{2.12}$$

where $y_i = -\log(t_i)$ with t_i being a realization of T_i. The estimates $\hat{\alpha}_n^{LS}$ and $\hat{\beta}_n^{LS}$ are unbiased estimates of α and β, respectively, even if there is dependence between the T_i's. It is possible to estimate the coefficients of the copula of the underlying AR process governing the dependence between the T_i's as follows: Compute $v_i =$

$\Phi^{-1}(1-\exp(\exp(\hat{\alpha}_n^{LS}+\hat{\beta}_n^{LS})t_i))$, then the k^{th} order auto-correlation coeffecient of the underlying AR process is estimated as

$$\hat{\gamma}_n^k = \frac{\sum_{j=k+1}^n v_{j-k}v_j}{\sum_{j=k+1}^n v_{j-k}v_{j-k}}. \tag{2.13}$$

The estimated $\hat{\gamma}_n^k$'s, for $k=1,\ldots,p$, can be used in the Yule-Walker equation to obtain moment-estimates of $\rho_1,\ldots,\rho_p$ as given by

$$\begin{pmatrix} \hat{\rho}_1 \\ \vdots \\ \hat{\rho}_p \end{pmatrix} = \begin{pmatrix} 1 & \hat{\gamma}_n^1 & \hat{\gamma}_n^2 & \cdots \\ \hat{\gamma}_n^1 & 1 & \hat{\gamma}_n^1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ \hat{\gamma}_n^{p-1} & \hat{\gamma}_n^{p-2} & \hat{\gamma}_n^{p-3} & \cdots \end{pmatrix}^{-1} \times \begin{pmatrix} \hat{\gamma}_n^1 \\ \vdots \\ \hat{\gamma}_n^p \end{pmatrix}. \tag{2.14}$$

Properties of Method of Moments Estimator We will now study the large sample properties of $\hat{\alpha}_n^{LS}$ and $\hat{\beta}_n^{LS}$. An alternative representation of the model, in terms of $Y_i = -log(T_i) - 0.5772$, is

$$Y_i = \alpha + \beta i + \zeta_i, \tag{2.15}$$

where ζ_i is such that $\zeta_i + \gamma$ is a standard Gumbel random variable, where $\gamma \approx 0.5772$ is the Euler-Massacheroni constant. The definition of ζ_i implies that $E(\zeta_i) = 0$ and $E(\zeta_i^2) = \pi^2/6$. Making use of the representation in (2.15) in conjunction with (2.12), we have

$$\begin{pmatrix} \hat{\alpha}_n^{LS} \\ \hat{\beta}_n^{LS} \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} + \begin{pmatrix} n & \sum_{i=1}^n i \\ \sum_{i=1}^n i & \sum_{i=1}^n i^2 \end{pmatrix}^{-1} \times \begin{pmatrix} \sum_{i=1}^n \zeta_i \\ \sum_{i=1}^n i\,\zeta_i \end{pmatrix}. \tag{2.16}$$

Theorem 2.7. *Under the de-eutrophication model with dependence, $\hat{\beta}_n^{LS}$ is a consistent unbiased estimator of β.*

Proof. For $n > 1$, the matrix inverse in (2.12) can be explicitly computed, resulting in

$$\hat{\beta}_{LS}^{n} - \beta = \sum_{i=1}^{n} \left(\frac{12 \times i}{n(n+1)(n-1)} - \frac{6}{n(n-1)} \right) \zeta_i. \tag{2.17}$$

Since $E(\zeta_i^2) = \pi^2/6$ for all i, $|E(\zeta_i \zeta_j)| \leq \pi^2/6$. Due to the Cauchy-Schwartz inequality, we have $E((\sum_i a_i \zeta_i)^2) \leq (\sum_i |a_i|)^2 \pi^2/6$. If we set

$$a_i = \left(\frac{12 \times i}{n(n+1)(n-1)} - \frac{6}{n(n-1)} \right),$$

we get

$$\begin{aligned} E((\hat{\beta}_n^{LS} - \beta)^2) &\leq \frac{\pi^2}{6} \left(\sum_{i=1}^{n} \frac{12 \times i}{n(n+1)(n-1)} + \sum_{i=1}^{n} \frac{6}{n(n-1)} \right)^2 \\ &= O(1/n^2). \end{aligned} \tag{2.18}$$

When the number of defects in the software is expected to be large, the asymptotic property of the estimator as $n \to \infty$ maybe useful. Due to Equation (2.18), $E((\hat{\beta}_n^{LS} - \beta)^2) \to 0$ when $n \to \infty$. This implies that $\hat{\beta}_n^{LS}$ is a consistent estimator of β. The unbiasedness follows from (2.17) and noting that $E(\zeta_i) = 0$. □

The magnitude of the parameter β can be used to compare the reliability of two different software products. This is because the improvement in reliability after fixing the i^{th} defect can be measured as $E(log(T_{i+1}/T_i)) = -\beta$. Hence, larger the value of $-\beta$, greater the improvement in reliability. Note that if $\beta > 0$, then the reliability decreases for every additional defect discovered and debugged. Theorem 2.7 establishes that the moment estimate of β, given by (2.16), is a consistent unbiased estimator. The consistency of $\hat{\alpha}_n^{LS}$ is not clear. The estimate $\hat{\alpha}_n^{LS}$ can be represented as

$$\hat{\alpha}_{LS}^{n} = \alpha + \frac{1}{n}\sum_{i\leq n}(\hat{\beta}_n^{LS}-\beta)\, i + \frac{1}{n}\sum_{i\leq n}\zeta_i \tag{2.19}$$

$$= \alpha + \frac{n+1}{2}(\hat{\beta}_n^{LS}-\beta) + \frac{1}{n}\sum_{i\leq n}\zeta_i \tag{2.20}$$

To prove that $\hat{\alpha}_n^{LS}$ is a consistent estimator of α, we would require, (i) $E(n(\hat{\beta}_n^{LS} - \beta)) \rightarrow 0$, i.e a much faster rate of convergence for $\hat{\beta}_n^{LS}$ than we have established in Theorem 2.7, and (ii) the weak law of large numbers to hold for the sequence $\zeta_1, \zeta_2, \ldots$. The second requirement on the ζ's can be ensured by assuming that the underlying copula based AR process follow the requirements of the weak law of large numbers of correlated random variables, namely the ζ's are stationary and the $\sum_{t=-\infty}^{\infty}|\gamma(s,t)| < \infty$, where $\gamma(s,t)$ is the covariance between ζ_s and ζ_t. The problem with (i) arises from the fact that, unlike in classical linear regression, where it is assumed that $\bar{x} = 1/n\sum_{i=1}^{n} X_i \xrightarrow{p} \mu$ as $n \rightarrow \infty$, in the de-eutrophication model of Moranda, $\bar{x} = (1+\ldots+n)/n = (n+1)/2$ increases with n. To deal with this problem, one could place more restrictive assumptions on the relationship between T_i and i in the underlying software reliability model, for example, $-E(log(T_i)) \propto \exp(-i)$. Such assumptions could be hard to justify from a practitioner's perspective, motivating us to develop new software reliability models and estimation methodologies that do not require such restrictive assumptions in Section 2.4. To prove that $\hat{\alpha}_n^{LS}$ is a consistent estimator of α, we would require, (i) $E(n(\hat{\beta}_n^{LS} - \beta)) \rightarrow 0$, i.e a much faster rate of convergence for $\hat{\beta}_n^{LS}$ than we have established in Theorem 2.7, and (ii) the weak law of large numbers to hold for the sequence $\zeta_1, \zeta_2, \ldots$. The second requirement on the ζ's can be ensured by assuming that the underlying copula based AR process follow the requirements of the weak law of large numbers of correlated random variables, namely the ζ's are stationary and the $\sum_{t=-\infty}^{\infty} |\gamma(s,t)| < \infty$, where $\gamma(s,t)$ is the covariance between ζ_s and ζ_t. The problem with (i) arises from the fact that, unlike in classical linear regression, where it is assumed that $\bar{x} = 1/n \sum_{i=1}^{n} X_i \xrightarrow{p} \mu$ as $n \to \infty$, in the de-eutrophication model of Moranda, $\bar{x} = (1+...+n)/n = (n+1)/2$ increases with n. To deal with this problem, one could place more restrictive assumptions on the relationship between T_i and i in the underlying software reliability model, for example, $-E(log(T_i)) \propto \exp(-i)$. Such assumptions could be hard to justify from a practitioner's perspective, motivating us to develop new software reliability models and estimation methodologies that do not require such restrictive assumptions in Section 2.4.

Maximum Likelihood Estimation The log-likelihood for the de-eutrophication model with dependence induced by a copula corresponding to an AR(1) process can be written as

$$
\begin{aligned}
l(\alpha,\beta,\rho_1,\sigma) = & \sum_{j=1}^{n} \left(-(\alpha-\beta j)(t_j \exp(\alpha-\beta j))\right) \\
& - \frac{(n-1)}{2} \log\left(\frac{\sigma^2}{1-\rho_1^2}\right) - \frac{1}{2}\sum_{j=2}^{n} (\frac{z_j - \rho_1 z_{j-1}}{\sigma})^2, \qquad (2.21)
\end{aligned}
$$

where $z_j = \Phi^{-1}(1-\exp(-\exp(\alpha-\beta j)t_j))$, $j = 1, \ldots, n$.

Note that (2.21) is a product of two parts: the first part corresponds to the marginal log-likelihood of the de-eutrophication model and the second part considers the dependence between the observations, obtained by transforming the observed inter-discovery times to the underlying AR(1) process. The log-likelihood for a copula corresponding to a higher order AR process can be similarly derived. Numerical maximization can be used to maximize the likelihood. All these numerical algorithms will require a starting point for the iteration and this can be obtained from the moment estimates described previously. A Newton-Raphson procedure may be used to maximize the log likelihood with the moment estimates of $\alpha, \beta, \rho_1, \sigma$, as described in the previous section, being used to initialize the procedure. For an AR(1) Gaussian process copula, the score function of the log-likelihood can be explicitly derived and is given as follows. Define

$$\begin{aligned}
u_j &= 1-\exp(-\exp(\alpha-\beta j)t_j) \\
v_j &= \Phi^{-1}(u_j) \\
\theta_j &= \alpha-\beta j \\
\eta_{j1} &= \exp(\theta_j)(1+\theta_j) \\
\eta_{j2} &= (v_j-\rho_1 v_{j-1})\left(\tfrac{(1-u_j)\exp(\theta_j t_j)}{\phi(n_j)} - \tfrac{(1-u_{j-1})\exp(\theta_{j-1}t_{j-1})}{\phi(v_{j-1})}\right) \\
\eta_{j3} &= (v_j-\rho_1 v_{j-1})\left(j\tfrac{(1-u_j)\exp(\theta_j t_j)}{\phi(v_j)} - (j-1)\tfrac{(1-u_{j-1})\exp(\theta_{j-1}t_{j-1})}{\phi(v_{j-1})}\right) \\
\eta_{j4} &= -\tfrac{n-1}{2\sigma^2} - \tfrac{(v_j-v_{j-1})^2}{2\sigma^2} \\
\eta_{j5} &= -\tfrac{\rho_1}{1-\rho^2} - \tfrac{(v_j-\rho_1 v_{j-1})}{\sigma},
\end{aligned}$$

where $\phi(.)$ is the standard normal density function. The score function of the log-likelihood can be expressed as

$$Score(\alpha,\beta,\rho_1,\sigma^2) = \sum_{j=1}^{n}\begin{pmatrix}\eta_{j1}\\ j\eta_{j1}\\ 0\\ 0\end{pmatrix} + \sum_{j=2}^{n}\begin{pmatrix}\eta_{j2}\\ \eta_{j3}\\ \eta_{j4}\\ \eta_{j5}\end{pmatrix}. \tag{2.22}$$

Note that the Hessian matrix of the log-likelihood, denoted by $H(\alpha,\beta,\rho_1,\sigma^2)$, can be derived from the derivative of the score function and be used in a Newton-Raphson procedure for computing the ML estimate $(\hat{\alpha},\hat{\beta},\hat{\rho_1},\hat{\sigma}^2)$ of the parameters $(\alpha,\beta,\rho_1,\sigma^2)$. The ML estimate may be computed through a Newton-Raphson recursion. Details of the k^{th} recursion, resulting in the computation of $(\alpha^{(k)},\beta^{(k)},\rho_1^{(k)},\sigma^{(k)2})$, is given below,

$$\begin{aligned}(\alpha^{(k)},\beta^{(k)},\rho_1^{(k)},\sigma^{(k)2})' &= (\alpha^{(k-1)},\beta^{(k-1)},\rho_1^{(k-1)},\sigma^{(k-1)2})' \\ &+ H\left(\alpha^{(k)},\beta^{(k)},\rho^{(k)},\sigma^{(k)2}\right)^{-1} l\left(\alpha^{(k-1)},\beta^{(k-1)},\rho_1^{(k-1)},\sigma^{(k)2}\right),\end{aligned} \tag{2.23}$$

with the recursion being initialized by setting $(\alpha^{(0)},\beta^{(0)},\rho_1^{(0)},\sigma^{(0)2})$ to their corresponding moment estimates. The recursion would continue till numerical convergence is observed. The inverse of the Hessian matrix evaluated at the ML estimate provides an estimate of the variance-covariance matrix of $(\hat{\alpha},\hat{\beta},\hat{\rho},\hat{\sigma}^2)$ and may be used for statistical inference about the parameters. For a model with higher order AR terms, the score and Hessian matrix will be cumbersome to derive. As an alternative, one can use numerical maximization algorithms such as the Nedler-Mead Algorithm to maximize the log-likelihood.

It is evident that models with dependence can become increasingly cumbersome to estimate through ML methods. Consequently, derivation of both the small sample and asymptotic properties of the ML estimate can become even more difficult. In view of these issues, some dependent isotonic models are presented in this chapter which provide a generic model for software reliability with dependence. Estimation of confidence bounds for software reliability under the dependent isotonic software reliability model that is valid for small samples is presented in Chapter 4.

2.4. Isotonic Software Reliability Models

We will now propose a series of generic software reliability models which we term *isotonic software reliability models*. The models we propose have applications beyond that of software reliability and can be used to estimate the reliability of a product whose reliability is improved through a sequential quality improvement plan, where a series of product versions are created with each version addressing the failure mode of the previous product version. The genesis of these new models lies in observing that the process of improving the reliability of a product version, by fixing the failure mode of its previous versions, imposes a constraint that the reliability of successive versions must increase, or remain the same. More generally, software testing can be considered a special case of a sequential quality control plan, where the discovery of each defect and its corresponding debugging creates a new software version which is more reliable than its previous version.

Sequential quality improvement plans are used for industrial product development where a major flaw in the product results in product recalls. They are also used for improving the reliability of critical procedures such as air traffic control (ATC) protocols, where an investigation is launched after a major incident, to determine if existing protocols can be improved to prevent similar incidents in future, leading to a revised and safer protocol. In an Internet banking context, after a major fraudulent credit card transaction, the bank usually imposes stricter software measures for fraud prevention intended to make future credit card transactions safer. Software testing provides an example of sequential quality improvement procedures where significant defects are fixed as and when they are discovered. It is important to estimate the time to failure distribution of the most recent version of the product, with such estimates being used to compute reliability metrics. For example, one might compute metrics such as the expected time to failure, or the probability that the next use case of the product/procedure will result in a failure. The data for such analysis is the time to failure for each of the previous versions of the product. In the ATC context, one may consider the number of successful take-offs achieved through the use of the protocol before a major incident. In the Internet banking example, the available data could be the number of legitimate credit card transactions that were processed before a fraudulent transaction was erroneously processed. For making decisions based on the reliability of such critical products and services, a confidence bound on the reliability is more useful as it enables the decision maker to evaluate the margin of reliability available.

The failure distribution for a wide variety of hardware products is typically estimated by observing multiple failures in a cohort of identical products of the same variety. Such a strategy is only possible if the event of a failure in any one of the members of the cohort is not catastrophic in nature (from safety, operational as well as financial perspectives). For example, in order to estimate the reliability of an household appliance such as a light bulb, it is common to test a large cohort of identical light bulbs and record the failure times of multiple failures as the cost of failure of any one light bulb is inconsequential. However, for a software product, the defect responsible for the failure is fixed immediately, creating a new version of the software with improved reliability. From a data perspective, there is a single failure for every new version of software created as opposed to multiple failures. Sequential quality improvement plans are also commonly used on failures which are catastrophic in nature, in which case there is a need to prevent a recurrence after each failure by immediately addressing the cause of the failure. Due to the critical nature of these products and services, the number of failures observed in the past will naturally be small requiring the data analysis methodology to be valid for small datasets. The method of analysing data from such plans must also consider the possibility that for some consecutive versions there might be no improvement in reliability at all. An objective of this chapter concerns the parametric estimation of reliability from data generated from such plans.

Examples of sequential quality improvement plans abound, even if they are not explicitly called as such. The first example, as described previously, relates to aircraft traffic control procedures and the second relates to internet banking fraud prevention procedures. In both examples, any substantial failure in the procedure would require immediate attention to prevent recurrence. The third example relates to software reliability, in which a defect after detection is immediately fixed in all working copies of the software. Product recalls of consumer durables, food products and automobiles due to the discovery of a serious safety issue are other important examples. A confidence bound on the reliability can help the manufacturer of the product or custodian of the procedure to quantify risk of a future failure. Such a bound can help, for example, a bank engaging in internet banking to compute the minimum insurance premium required to insure every financial transaction being conducted. For a software manufacturer, it can result in better operational management by setting aside financial resources to cover a substantial software defect being discovered in the future.

The assumption of reliability improvement over the sequence of observations constraints the failure time model and consequently the successive failure times leading to *isotonicity* in the failure time distributions. Although we expect the reliability of the product not to decrease with every subsequent version, this might not happen in some situations. To address this issue, we demonstrate how the proposed method can be modified to consider a bounded decrease in reliability of the subsequent product versions. As mentioned before, sequential quality improvement procedures are used on products whose failure can be catastrophic and hence the number of product failures will be small. Due to the catastrophic nature of the failure, it is more important to provide confidence intervals for the reliability than the corresponding point estimates as it allows for an evaluation of the margin of safety in the product. In particular, one-sided bounds, which have at least a certain coverage for the parameter of interest of the failure time distribution, can be invaluable for decision makers. The failure time distribution of each version can be modeled parametrically while the nature of improvement in reliability is in general unknown. While computing confidence bounds, it is important to note that asymptotic procedures may not be appropriate since the size of the data-set comprising of the failure times for all versions can be small. Also, the procedure would need to deal with there being a single failure time observation for each version. Existing procedures for computing confidence bounds may be questioned, as they usually incorporate some parametric modeling of the nature of improvement in reliability over successive versions as in, for example, Jelinski and Moranda (1972). Independence of the failure times of different versions can be assumed in many situations, but may be questioned in some others.

The distributional assumptions on the time to failure distribution play a critical role in the proposed methodology. There are two ways of arriving at such assumptions. The first approach is based on the historical analysis of the failure time data for the particular product type. For many engineering components, the time to failure is commonly assumed to follow a Weibull distribution. Similarly, for software reliability, the time to failure has been noted by many researchers as being Exponential (Singpurwalla and Wilson, 1994). The second approach would be based on a simple probabilistic model for the failure distribution. For example, the distribution of the number of legitimate internet banking transactions that occurred before a fraudulent transaction was erroneously processed can be justified as a Geometric distribution. This is because the states of any two processed transactions (legitimate or fraudulent) can be modeled as independent and identical Bernoulli trials. We now introduce a series of new software reliability models that address some of the concerns raised in this chapter. We also discuss how to estimate them and compare their performance with the parametric models described in the previous section.

2.4.1. Independent Exponential Isotonic Software Reliability (EISR) Model

Exponentially distributed failure times for a software is a popular assumption for software reliability; hence an isotonic software reliability model that assumes exponentially distributed failure times is defined first.

Model 2.8. (Independent Exponential Isotonic Software Reliability Model) Let T_i, for $i = 1,\ldots,n$, be the inter-failure times for n observed software defects. Then $T_i \sim Exponential(\lambda_i)$, for $i = 1,\ldots,n$, with $\lambda_1 \geq \ldots \geq \lambda_n$ with T_i and T_j being independent of each other for $i \neq j$.

The parametric models of software reliability considered earlier in this chapter are special cases of the Independent-EISR model. In fact, by considering the Independent-EISR model, we would be considering almost all models of software reliability growth that assume an exponentially distributed failure time for each software defect. It clearly is a very generic model for software reliability.

Model Estimation The log-likelihood of the observed failure time t_i's, for $i = 1, \ldots, n$, is given by

$$l(\lambda_1, \ldots, \lambda_n) = \sum_{i=1}^{n} \log \lambda_i - \sum_{i=1}^{n} \lambda_i t_i, \ \lambda_1 \geq \ldots \geq \lambda_n, \tag{2.24}$$

where t_i is a realization of T_i. The maximization of this likelihood can be done using the pooled adjacent violators algorithm Ayer et al. (1955).

Theorem 2.9. *The ML estimate of λ_n, under the EISR model, is given by*

$$\hat{\lambda}_n = \frac{1}{max\left\{t_n, \frac{t_n+t_{n-1}}{2}, \ldots, \frac{t_n+\ldots+t_1}{n}\right\}}. \tag{2.25}$$

Proof. Let $\theta_i = 1/\lambda_i$, for $i = 1, \ldots, n$. Then, $\theta_1 \leq \cdots \leq \theta_n$ and (1.35) can be used to compute the ML estimate of θ_i as

$$\hat{\theta}_i = \min_{j:j\geq i} \ \max_{k:k\leq i} \frac{\sum_{l=k}^{l=j} t_l}{j-k+1}. \tag{2.26}$$

Setting $i = n$ in (2.26) and noting that $\hat{\lambda}_n = 1/\hat{\theta}_n$ proves the theorem. □

The distribution of $\hat{\lambda}_n$ is clearly difficult to derive. Nevertheless, in Chapter 3, we present a novel method of computing an upper-bound for $P(\hat{\lambda}_n \leq x)$ and use it to compute a confidence bound for λ_n.

2.4.2. Dependent Exponential Isotonic Software Reliability (EISR) Model

The copula based software reliability model, proposed earlier in Section 2.3, presents a generalization to many a software reliability model that assumes dependence between the software failure times. However, there are two main concerns regarding the model; the first concern relates to model estimation due to the increasing complexity of the model likelihood function and hence its maximization. The second concern relates to establishing large sample properties of the estimates of the model parameters. It is in the context of these concerns, we propose the dependent-EISR model. Due to Sklar's theorem, the dependent-EISR model is a very generic model for software reliability.

Model 2.10. (Dependent Exponential Isotonic Software Reliability Model) Let T_i, for $i = 1, \dots, n$, be the inter-failure times. Then $T_i \sim Exponential(\lambda_i)$, for $i = 1, \dots, n$ with $\lambda_1 \geq \dots \geq \lambda_n$. We assume that T_i's have dependence induced between them by an arbitrary copula $C(\cdots)$.

The estimation of λ_n through maximum likelihood procedures would require an exact specification of the copula governing the dependence between the T_i's. In Chapter 4, we present one of the key contributions of the thesis, which is the derivation of a novel confidence bound for λ_n without making any assumptions about the nature of the dependence.

2.4.3. Weibull Isotonic Software Reliability Models

A natural way to extend the EISR model is to consider the failure times T_i to be distributed either as a Weibull or a Gamma distribution. Failure times of a software

are not always Exponentially distributed, as first noted by Schick and Wolverton (1978) where they are assumed to be distributed as a Weibull distribution with a shape parameter of 2 (i.e. a Rayleigh distribution). The Weibull distribution for failure time is useful to counter criticism regarding a constant hazard of failure assumed by early software reliability models. Isotonic regression involving Weibull distribution would be a very generic and useful model for software reliability. As in the case of the EISR model, there are two variations that are possible, one that assumes independence and another that assumes dependence between the failure times governed by an arbitrary copula.

Model 2.11. (Independent Weibull Isotonic Software Reliability (WISR) Model) Let T_i, for $i = 1, \ldots, n$, be the inter-failure times. Then $T_i \sim Weibull(\lambda_i, \alpha)$, for $i = 1, \ldots, n$ with $\lambda_1 \geq \ldots \geq \lambda_n$. Assume that T_i's are independent of each other.

An important assumption being made in the Independent-WISR model is that all the failure times have the same shape parameter, though they can have arbitrary scale parameters with isotonic constraints. In this thesis, we propose a method for obtaining confidence upper-bounds on λ_n under the WISR model with unknown shape parameter α with the constraint $L \leq \alpha \leq U$. The methodology is discussed in Chapter 3. The model of Schick and Wolverton (1978) is a special case of the Independent-WISR model where $T_i \sim Weibull(\lambda(N-i+1), 2)$ independently for $i = 1, \ldots, n$.

Model Estimation Note that if α were known, then $T_i^\alpha \sim Exponential(\lambda_i^\alpha)$. Let t_i be a realization of T_i, then we can use the estimator in (2.25) applied to t_i^α to obtain the MLE of λ_n^α from which the MLE of λ_n can be computed. The estimator will be a function of α and is given by

$$\hat{\lambda}_n(\alpha) = \frac{1}{max\left\{t_n^\alpha, \frac{t_n^\alpha + t_{n-1}^\alpha}{2}, \ldots, \frac{t_n^\alpha + \ldots + t_1^\alpha}{n}\right\}^{1/\alpha}}. \tag{2.27}$$

The Independent-WISR model can be generalized to the Dependent-WISR by not requiring the T_i's to be independent of each other. This model is stated as follows;

Model 2.12. (Dependent Weibull Isotonic Software Reliability (WISR) Model) Let T_i, for $i = 1, \ldots, n$, be the inter-failure times. Then $T_i \sim Weibull(\lambda_i, \alpha)$, for $i = 1, \ldots, n$ with $\lambda_1 \geq \ldots \geq \lambda_n$. Assume $T_1, \ldots, T_n$ are dependent with their dependence governed by an arbitrary copula $C(\cdots)$.

Estimation of λ_n without specification of the copula C is a challenge. In Chapter 4 we come up with a method for computing a confidence upper-bound for λ_n without making any assumption on the nature of dependence between the T_i's.

2.4.4. Gamma Isotonic Software Reliability Model (GISR)

The next extension of the EISR model is to consider $T_i \sim Gamma(\lambda_i, \alpha)$, where α is the shape parameter. The EISR model is a special case of the GISR model with $\alpha = 1$. As with EISR and WISR models, there are two variations to the GISR model; the first variations considers independence between the observed failure times T_i's and is stated as follows:

Model 2.13. (Independent Gamma Isotonic Software Reliability (GISR) Model) Let T_i, for $i = 1, \ldots, n$, be the inter-failure times. Then $T_i \sim Gamma(\lambda_i, \alpha)$, for $i = 1, \ldots, n$ with $\lambda_1 \geq \ldots \geq \lambda_n$. We assume that T_i's are independent of each other.

If α were known, the MLE of λ_i can once again be obtained by the PAV algorithm and is given by

$$\hat{\lambda}_n(\alpha) = \frac{1}{\alpha \times max\left\{t_n, \frac{t_n+t_{n-1}}{2}, \ldots, \frac{t_n+\ldots+t_1}{n}\right\}}, \tag{2.28}$$

where t_i is a realization of T_i. The problem of estimating both α and λ_n simultaneously will be challenging. A key contribution of this thesis involves a method for obtaining confidence upper-bounds on λ_n under the GISR model with α unknown under the constraint $\alpha \leq U$. The methodology is discussed in Chapter 3. The dependent gamma isotonic regression model is an obvious generalization. The model is stated as follows:

Model 2.14. (Dependent Gamma Isotonic Software Reliability (GISR) Model) Let T_i, for $i = 1, \ldots, n$, be the inter-failure times. Then $T_i \sim Gamma(\lambda_i, \alpha)$, for $i = 1, \ldots, n$ with $\lambda_1 \geq \ldots \geq \lambda_n$. Assume that the dependence between the T_i's is governed by an arbitrary copula $C(\cdots)$.

Estimation of λ_n will require specification of the copula $C(\cdots)$. As in the case of the dependent-WISR model, a method of constructing a confidence upper-bound on λ_n under the dependent-GISR model is derived in Chapter 4.

2.5. Simulation Study

We now proceed to a simulation study that compares the ML estimator of λ_n obtained using the Exponential isotonic software reliability model with parametric models of software reliability. In particular, for a data-set with n failure times, $T_1, \ldots, T_n$, we will be interested in studying the bias and the root mean squared error (RMSE) of the ML estimate of λ_n obtained through (2.25) and comparing them with the ML estimates obtained from parametric software reliability models, such as the Jelinski and Moranda model (Model 2.1) and the de-eutrophication model (Model 2.4). The robustness of the ML estimators, when there are deviations in the underlying software reliability improvement model, is much desired not only in the context of software reliability but for any statistical estimation method as noted by Freedman (1991). The robustness of the estimator of λ_n is particularly important for estimating the reliability of a software product with respect to catastrophic failures. We have designed the simulation study with the object of charachterizing the robustness of the estimators, by estimating their bias and root mean squared error (RMSE), when the underlying software reliability model is incorrect.

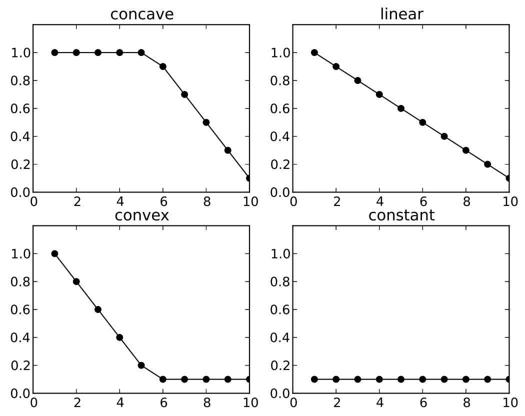
The simulation study considers four different patterns of software reliability improvement. These patterns, for $n = 10$, are presented algebraically in Table 2.1 and presented graphically in Figure 2.2. The patterns are chosen so as to reflect four different non-increasing patterns of the λ_k's under the EISR model. Note that, while the Jelinski and Moranda model represents linear decrease in λ_k's, the de-eutrophication model of Moranda represents a convex exponential decrease in λ_k's, the convex graph in Figure 2.2 approximating it the best. Failure times T_i are simulated such that $T_i \sim Exponential(\lambda_i)$, for $i = 1, \ldots, n$, where λ_i's are obtained from one of the models in Table 2.1.

The patterns of reliability improvement that we chose was motivated by 1) the need to have one pattern to correspond exactly to a known parametric software reliability model and 2) have at least two patterns that are not modeled by that parametric model of software reliability. The first requirement would help in characterizing the performance of the PAVA based estimator when the corresponding

TABLE 2.1
Expressions for λ_k, for $k = 1, \ldots, n$, corresponding to four patterns of improvement

Constant	$\theta_k = 1/3$
Linear	$\theta_k = 1 - 2k/3n$
Convex	$I(2k \leq n) \times (1 - 4k/3n) + I(2k > n) \times 1/3$
Concave	$I(2k \leq n) + I(2k > n) \times (1 - (4k - 2n)/3n)$

FIG 2.2. *Plot of λ_k vs. k for four different patterns of software reliability improvement, with $n = 10$.*



parametric model is correctly specified. The second requirement would enable us to study the effects of model mis-specification. With this context, we chose the linear pattern of improvement which is modeled exactly by Jelinski-Moranda model. The convex and concave patterns of improvement were chosen as they cannot be approximated by the Jelinski-Moranda model. While the convex model may be approximated by the de-eutrophication model of Moranda, the concave model cannot be modeled by either the Jelinski-Moranda or the de-eutrophication model of Moranda.

We define the relative bias of an estimator $\hat{\lambda}_n^{\mathcal{M}}$ obtained using model $\mathcal{M}$ as $E((\hat{\lambda}_n^{\mathcal{M}} - \lambda_n)/\lambda_n)$. The RMSE of the estimator is defined as $\sqrt{E((\hat{\lambda}_n^{\mathcal{M}} - \lambda_n)/\lambda_n)^2}$. The expectations in the definitions of bias and RMSE are computed using Monte-

Carlo simulations as follows: Simulate $T_1, \ldots, T_n$ from one of the patterns of reliability improvement described in Table 2.1; estimate λ_n by $\hat{\lambda}_n^{\mathcal{M}}$ obtained by using the model $\mathcal{M}$, compute the bias as $(\hat{\lambda}_n^{\mathcal{M}} - \lambda_n)/\lambda_n$ and squared error as $((\hat{\lambda}_n^{\mathcal{M}} - \lambda_n)/\lambda_n)^2$ and store both values; repeat the procedure 1000 times and compute the empirical mean of the bias values and square root of the empirical mean of squared error values over the 1000 repetitions. These values, based on the simulations will be estimates of $E((\hat{\lambda}_n^{\mathcal{M}} - \lambda_n)/\lambda_n)$ and $\sqrt{E((\hat{\lambda}_n^{\mathcal{M}} - \lambda_n)/\lambda_n)^2}$, respectively. The results of the simulation study, for the different models $\mathcal{M}$, are provided in Table 2.2.

From Table 2.2 it is clear that for data simulated under the linear pattern of improvement, the estimate of λ_n from the Jelinski-Moranda model has the least bias and RMSE, for n = 10, 25 and 100, as seen from the second row of each corresponding sub-table. Note that the second row of each sub-table in Table 2.2 corresponds to data simulated from the linear model and represents the case when the Jelinski-Moranda model is indeed the true model. For the linear pattern of improvement, when n = 100, the estimate of λ_n from the Jelinski-Moranda model has negligible bias and the least RMSE. This leaves us to conclude that the linear pattern of improvement is similar to the Jelinski-Moranda model. The deeutrophication model of Moranda assumes that $\lambda_i = \exp(\alpha - \beta i)$, which corresponds to an exponential improvement in software reliability. Note that the constant model of software reliability is a special case of the Moranda model with $\beta = 0$. Table 2.2, shows that the Moranda model has the least bias and RMSE in estimating λ_n when used on data generated from the constant model, with the bias becoming negligible when $n = 100$. When we exclude the constant model,

TABLE 2.2
Bias and RMSE of the estimates of λ_n. The values in () are the RMSE values while the unbracketed values are the bias.

$n = 10$

Model	Jelinski Moranda	Moranda	$\hat{\lambda}_n$
constant	-0.43 (0.5)	0.43 (1.2)	-0.17 (0.4)
linear	0.68 (1.1)	1.23 (2.2)	0.71 (1.6)
concave	1.26 (1.8)	2.30 (3.8)	1.08 (2.2)
convex	-0.21 (0.5)	-0.23 (0.7)	-0.04 (0.6)

$n = 25$

Model	Jelinski Moranda	Moranda	$\hat{\lambda}_n$
constant	-0.55 (0.6)	0.12 (0.5)	-0.27 (0.4)
linear	0.19 (0.6)	0.89 (1.2)	0.27 (0.9)
concave	0.73 (1.2)	2.07 (2.6)	0.59 (1.4)
convex	-0.45 (0.5)	-0.44 (0.5)	-0.21 (0.4)

$n = 100$

Model	Jelinski Moranda	Moranda	$\hat{\lambda}_n$
constant	-0.59 (0.6)	0.03 (0.2)	-0.30 (0.4)
linear	-0.03 (0.3)	0.80 (0.9)	-0.06 (0.5)
concave	0.40 (0.6)	1.87 (2.0)	0.08 (0.6)
convex	-0.56 (0.6)	-0.49 (0.5)	-0.28 (0.4)

the convex pattern of software reliability improvement is the closest to the de-eutrophication model of Moranda. This is evident from column 3 of Table 2.2, where the Moranda model has significantly lesser bias and RMSE in estimating λ_n when the true model is indeed convex as compared to when the true model is concave or linear. From this observation, we can infer that the convex model is best approximated by the de-eutrophication model of Moranda.

The results in Table 2.2 reveal useful insights. The ML estimator of λ_n using (2.25), presented in the fourth column of Table 2.2, outperforms the ML estimates of the parametric models, in terms of bias and RMSE, when the parametric model is mis-specified. This indicates the ML estimator (2.25), obtained using the independent-EISR model, is a more robust estimator of software reliability compared to estimators derived through parametric models. Note that, when the true model is the Jelinski and Moranda model, (that is, the linear model), the performance of the estimator obtained by the independent EISR model is similar to that of the true model. On the other hand, the performance of the Moranda model is much worse when the true model is the linear model. Similar observations can be made for the other models presented in Figure 2.2. For all the three size of n considered, the ML estimate obtained through the independent-EISR model appears to be superior to those obtained from both the parametric models when the reliability improvement is convex or concave. This may be due to the parametric models considered being unable to approximate convex or concave patterns of reliability improvement. This result alone should warn us about the dangers of using parametric models when there is little or no justification for the parametric form of the underlying model.

2.6. Concluding Remarks

In this chapter, in addition to discussing existing parametric models for software reliability, we have introduced copula based dependent failure time models and isotonic software reliability models. The latters generalizes a very broad class of software reliability models. Maximum likelihood estimation of the reliability of the software are discussed. The isotonic software reliability models can be modified to consider bounded decrease in the reliability of the software after a defect is detected and fixed. For example, in the independent-EISR model, one may assume that $\lambda_{i+1} \leq \beta\lambda_i$ with a known $\beta > 1$. This assumption would allow for λ_i to increase with i which would allow the reliability of the software to sometimes decrease after a software defect is detected and fixed. The possibility of the software reliability decreasing after a defect has been detected is a commonly studied contemporary issue in software reliability. A number of Bayesian models have been proposed for addressing such a possibility. The bounded increase model provides a simpler alternative to some of these sophisticated Bayesian models. The parameter λ_n can be estimated by (3.2) using the sequence $\beta^{-n+i}t_i$, for $1 = 1,\ldots,n$, instead of using the original inter-failure time sequence t_i.

The simulation study reveals that the isotonic regression model is preferable to a parametric software reliability model when the justification for the parametric model is not clear. The distribution of the PAVA based ML estimator is needed in order to compute confidence bounds for λ_n. The problem of determining this distribution, even for the independent-EISR model, is challenging because the PAVA based estimator is a complex non-linear function of the failure times, T_i's, and will depend on all the n parameters $\lambda_1,\ldots,\lambda_n$. In Chapter 3, we shall address all these challenges and derive an innovative confidence bound based on the PAVA estimate in (3.2). The problem of estimation of a confidence interval for λ_n, under the dependent-EISR model, is considered in Chapter-4.

Chapter 3

Independent Isotonic Software Reliability Models

Failure saves lives. In the airline industry, every time a plane crashes the probability of the next crash is lowered by that.

Nassim N. Taleb

3.1. Introduction

In this chapter we propose a method for estimating an upper bound for scale parameter of the failure time distribution of a software product under the independent EISR, WISR and GISR models introduced in Chapter 2. The method can be used to provide a lower bound for the reliability $R(t)$ of the latest version of the software product. To re-capitulate, the reliability of a software product is improved through a process of discovering software defects and fixing them as soon as they are discovered. A software reliability model estimates the distribution of the time to discovery of the next defect by incorporating the effect of the debugging process. Such an estimate can be useful to customers of the software product as well as the software quality control engineers. For example, based on the estimated distribution, a software testing team might choose to stop the test process and release the software product (Singpurwalla and Wilson, 1994).

Let T_i denote the time to failure of the software after the discovery of the i^{th} defect, for $i = 1, \ldots n$. The assumption that the reliability of a software must increase after every subsequent discovery and fixing of a defect is widely used. See (Jelinski and Moranda, 1972; Moranda, 1975; Goel and Okumoto, 1978) for some examples. A parametric model may assume that $T_i \sim F(.|\Theta, i)$, where $F(.|\Theta, i)$ is a CDF, Θ is the model parameter and T_i is stochastically larger than T_{i-1}, for $i = 2, \ldots, n$. For example, the Jelinski and Moranda Model (Jelinski and Moranda, 1972) assumed that $T_i \sim Exponential(\lambda(N - i + 1))$, for $1 \leq i \leq N$, where N is the number of defects in the software. Subsequent to the Jelinski and Moranda model, increasingly sophisticated parametric forms for the improvement in reliability have been proposed.

A generic software reliability model assuming exponential failure time for the T_i's would consider the EISR model introduced in Chapter 2 where $T_i \sim Exponential(\lambda_i)$ with $\lambda_{i+1} \leq \lambda_i$. The maximum likelihood estimate of λ_n can be obtained from the observations $T_1, \ldots, T_n$ through the Pooled Adjacent Violator Algorithm (PAVA) Ayer et al. (1955). Performing statistical inference using the PAVA based ML estimator of λ_n can be challenging as the estimator is a complex non-linear function of the observations and standard asymptotic theory or bootstrap procedures may not be used due to possibility of some of the λ_i's being equal. In particular, the parameter space comprising of $\lambda_1 \geq \ldots \geq \lambda_n > 0$ is not an open set. See Andrews (2000) for a criticism on using bootstrap procedures or asymptotic theory for obtaining SE of estimators when the parameter space is not an open set. The problem of computing confidence intervals for a fixed number of proportions under isotonic constraints has been studied in the context of dosage studies using new bootstrap procedures (Bhattacharya and Kong, 2007). These procedures cannot be adapted for the problem at hand as the number of model parameters, i.e. $\lambda_1, \ldots, \lambda_n$'s can be equal to the number of observations $T_1, \ldots, T_n$.

One can further generalize the isotonic exponential regression based software reliability model to non-exponential distributions as follows: Let $T_i \sim \lambda_i^{-1} E_i$ with $\lambda_1 \leq \ldots \leq \lambda_n$, $E_1, \ldots, E_n \overset{iid}{\sim} F$ where $E_F(X^P) < \infty$ for $P \geq 1$. The objective would be the estimation of λ_n along with confidence lower-bounds for it. In this article, we derive a new statistic that can be used to obtain an upper-bound on λ_n with atleast $100 \times (1-q)\%$ confidence. For the EISR model, an upper bound $\hat{\lambda}_n^q$ with $P(\lambda_n \leq \hat{\lambda}_n^q) \geq 1-q$ can be used to compute a lower-bound on $R_n(t)$, the reliability of the software after n failures over t units of future usage through

$$\hat{R}_n(t)^q = \exp(-\hat{\lambda}_n^q t). \tag{3.1}$$

Since $\hat{\lambda}_n^q$ is an upper-bound for λ_n, it is clear that $P(\hat{R}(t)^q < R(t)) \geq 1-q$. Note that the model being considered is quite generic; for example it would include the WISR model where $T_i \sim Weibull(\lambda_i, \alpha)$ with $\lambda_1 \geq \ldots \geq \lambda_n$ and the Weibull shape parameter, α, being constant. The model would also include the GISR model with $T_i \sim Gamma(\lambda_i, \alpha)$, $\lambda_1 \geq \ldots \geq \lambda_n$ and the Gamma shape parameter, α, being constant during the entire testing period of the software.

3.2. The Main Result

Theorem 3.1. *Let $T_i \sim \lambda_i^{-1} E_i$, $\lambda_1 \geq \ldots \geq \lambda_n$, $E_1, \ldots, E_n \overset{iid}{\sim} F$ with $P_F(X < 0) = 0$ and $E_F(X^P) < \infty$ for $P \geq 1$. Define*

$$\hat{\lambda}_n = \frac{1}{max\left\{t_n, \frac{t_n + t_{n-1}}{2}, \ldots, \frac{t_n + \ldots + t_1}{n}\right\}}, \tag{3.2}$$

where t_i is a realization of T_i. Then, for any $0 \le q \le 1$ *and* $P \ge 1$,

$$P\left(\hat{\lambda_n}\left(\frac{E_F(X^P)}{q}\right)^{\frac{1}{P}} \le \lambda_n\right) \le q \tag{3.3}$$

Proof. Define $S_1 = T_n + \cdots + T_1, S_2 = T_n + \cdots + T_2, \ldots, S_n = T_n$ and $Y_j = S_j/(n-j+1)$ for $j = 1, \ldots, n$. We will show that $Y_1, \ldots, Y_n$ is a sub-martingale and then use the Doob's maximal inequality (Doob, 1953) on the sub-martingale to prove the theorem. For $j \le k \le n$, we have $E(T_k|S_j) = \lambda_k^{-1}E(E_k|S_j) = (\lambda_j/\lambda_k)E(T_j|S_j)$. Note that $E(T_j|S_j) + \ldots + E(T_n|S_j) = S_j$ and hence,

$$E(T_j|S_j)\sum_{k=j}^{n}\frac{\lambda_j}{\lambda_k} = S_j \implies E(T_j|S_j) = \frac{S_j}{\sum_{k=j}^{n}\lambda_j/\lambda_k} \implies E(T_j|S_j) \le \frac{S_j}{n-j+1} = Y_j,$$

since $\lambda_j/\lambda_k \ge 1$, for $j \le k \le n$. By the definition of Y_j's, we have

$$\begin{aligned} Y_{j+1} = \frac{(n-j+1)Y_j - T_j}{n-j} \implies E(Y_{j+1}|Y_j) &= \frac{(n-j+1)Y_j}{n-j} - \frac{E(T_j|Y_j)}{n-j} \\ &\ge \frac{(n-j+1)Y_j - Y_j}{n-j} = Y_j, \end{aligned}$$

because $E(T_j|S_j) \le Y_j$. This proves $E(Y_{j+1}|Y_j) \ge Y_j$. In order to show that $Y_1, \ldots, Y_n$ is a sub-martingale, we only need to show that $(Y_{j+1}|Y_j, \ldots, Y_1) \overset{d}{=} (Y_{j+1}|Y_j)$. This can be proved as follows: note that $S_{j+1} = S_j - T_j$ with T_j being independent of $T_1, \ldots, T_{j-1}$. Hence the conditional distribution $(S_{j+1}|S_j, T_1, \ldots, T_{j-1}) \overset{d}{=} (S_{j+1}|S_j)$. Note that conditioning on $S_1, \ldots, S_j$ is equivalent to conditioning on $S_j, T_1, \ldots, T_{j-1}$. Hence $(S_{j+1}|S_j, \ldots, S_1) \overset{d}{=} (S_{j+1}|S_j)$. Since $Y_j = S_j/(n-j+1)$ for $j = 1, \ldots, n$, we will have $(Y_{j+1}|Y_j, \ldots, Y_1) \overset{d}{=} (Y_{j+1}|Y_j)$. The Doob's maximal inequality (Doob, 1953) on the non-negative submartingale $Y_1, \ldots, Y_n$ states that

$$P(max(Y_1, \ldots, Y_n) > \theta) \le \frac{E(Y_n^P)}{\theta^P}, \quad P \ge 1, \theta > 0. \tag{3.4}$$

By definition, we have $Y_n = T_n$, $\hat{\lambda}_n = 1/max(Y_1,\ldots,Y_n)$ and $E(Y_n^P) = \lambda_n^P E_F(X^P)$. Setting $\theta = (1/\lambda_n)\left(E_F(X^P)/q\right)^{1/P}$ in Equation (3.4) completes the proof.

□

Theorem 3.1 gives a conservative upper-bound for λ_n in the sense that $P(\lambda_n \leq \hat{\lambda_n}\left(\frac{E_F(X^P)}{q}\right)^{\frac{1}{P}}) \geq 1-q$; that is, $\hat{\lambda_n}\left(\frac{E_F(X^P)}{q}\right)^{\frac{1}{P}}$ is an upper confidence bound for λ_n with atleast $100\times(1-q)\%$ confidence. One can create the sharpest upper-bound by minimizing the scaling factor $\sigma(P,q) = \left(E_F(X^P)/q\right)^{(1/P)}$ over $P \geq 1$. In general, the minimization would depend on the distribution F. We proceed to discussing application of Theorem 3.1 to specific distributional families.

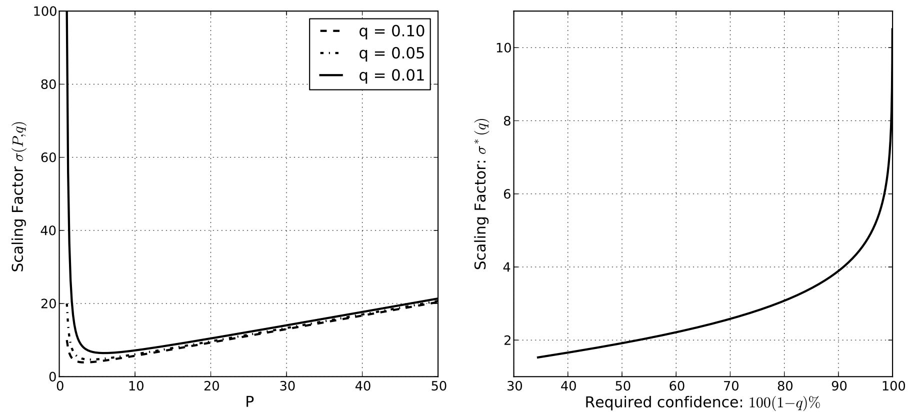
3.2.1. Exponential isotonic software reliability model

Consider the exponential isotonic software reliability model, with $T_i \sim Exponential(\lambda_i)$, $\lambda_1 \geq \ldots \geq \lambda_n$, i.e., the EISR model. The statistic $\hat{\lambda}_n$ is the ML estimate of λ_n through the PAV algorithm. Clearly the EISR model is a special case of the model described in Theorem 3.1, with $F = Exponential(1)$ and hence $\sigma(P,q) = (\Gamma(P+1)/q)^{(1/P)}$. There is scope for minimizing $\sigma(P,q)$ by optimally choosing P as illustrated in the plots of $\sigma(P,q)$ vs. P for various values of q provided in the left panel of Figure 3.1. It can be shown that the optimal scaling factor, $\sigma^*(q) = \min_P (\Gamma(P+1)/q)^{(1/P)}$, is attained when P is the solution to the equation

$$\log(\Gamma(P+1)) - P\,\Psi(P+1) = \log(q), \tag{3.5}$$

where $\Psi(.)$ is the digamma function. This is equivalent to stating that for $q = \Gamma(P+1)\exp(-P\Psi(P+1))$, the optimal scaling factor $\sigma^*(q) = \exp(\Psi(P+1))$. This result is stated as follows:

FIG 3.1. *Plot of* $\sigma(P,q)$ *vs. P.*



Result 3.2. *Let* $T_i \sim Exponential(\lambda_i)$*, with* $\lambda_1 \geq \ldots \geq \lambda_n$*. Define* $\hat{\lambda_n}$ *as in* (3.2)*, then*

$$P\left(e^{\Psi(P+1)}\hat{\lambda}_n \leq \lambda_n\right) \leq \Gamma(P+1)e^{-P\Psi(P+1)}, \quad P \geq 1. \tag{3.6}$$

A plot of $\sigma^*(q) = \exp(\Psi(P+1))$ vs. $1-q = 1-\Gamma(P+1)\exp(-P\Psi(P+1))$, for $P \geq 1$ can visually present the optimal scaling factor as a function of the required confidence. The right panel of Figure 3.1 presents such a plot. From this plot, we gather that for an EISR model, $4.71 \times \hat{\lambda}_n$ is an upper-bound for λ_n with atleast 95% confidence. Table 3.1 presents the scaling factor required for some commonly used confidence levels.

TABLE 3.1
Values of $\sigma^*(q)$ *for some values of* $100 \times (1-q)\%$.

$100 \times (1-q)\%$	50%	90%	95%	99%
$\sigma^*(q)$	1.90	3.80	4.71	6.62

3.2.2. Weibull isotonic software reliability model

The EISR model can be generalized to a Weibull isotonic software reliability model (WISR) by assuming that $F(.)$ in Theorem 3.1 is a $Weibull(1,\alpha)$, with known shape parameter α. The scaling factor would be a function of α with $\sigma(P,q,\alpha) = (\Gamma(P/\alpha+1)/q)^{(1/P)}$. Just as in the EISR model, the optimum value for the scaling factor, $\sigma^*(q,\alpha) = \min_P \sigma(P,q,\alpha)$, is achieved when P is solution to

$$\frac{P}{\alpha}\Psi\left(\frac{P}{\alpha}+1\right) = \log\left(\Gamma\left(\frac{P}{\alpha}+1\right)\right) - \log(q). \tag{3.7}$$

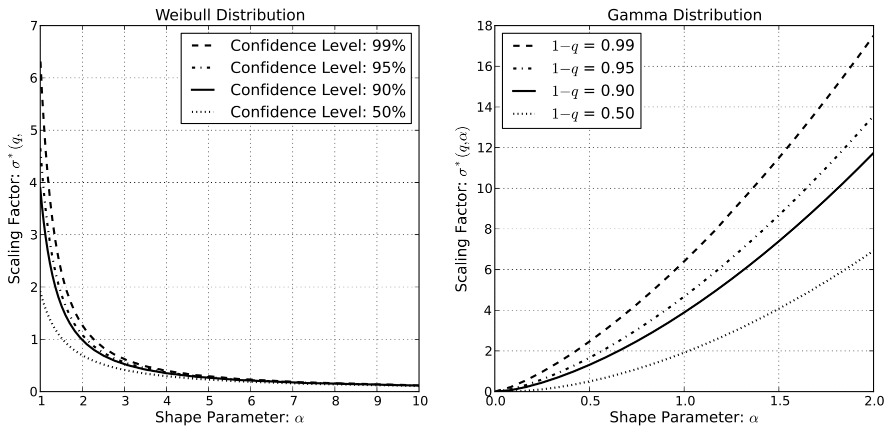
This can be interpreted to mean that, for $q = \Gamma((P/\alpha)+1)\exp(-(P/\alpha)\Psi((P/\alpha)+1))$, the optimal scaling factor $\sigma^*(q,\alpha) = \exp(\Psi(P/\alpha+1)/\alpha)$. Note that $\hat{\lambda}_n$ as defined through (3.2) is no longer the MLE of λ_n. Nevertheless, Theorem 3.1 can still be used to obtain a confidence upper-bound for λ_n, as stated in Result 2 below.

Result 3.3. *Let $T_i \sim Weibull(\lambda_i,\alpha)$, with $\lambda_1 \geq \ldots \geq \lambda_n$ and $\alpha \geq 1$.*
Define $\hat{\lambda}_n$ as in (3.2)*, then*

$$P\left(\exp\left(-\frac{\Psi\left(\frac{P}{\alpha}+1\right)}{\alpha}\right)\hat{\lambda}_n \leq \lambda\right) \leq \Gamma\left(\frac{P}{\alpha}+1\right)\exp\left(-\frac{P}{\alpha}\Psi\left(\frac{P}{\alpha}+1\right)\right), \quad P \geq 1. \tag{3.8}$$

Plots of $\sigma^*(q,\alpha)$ vs. α are presented for some values of q in the left panel of Figure 3.2. It can be shown that $\sigma^*(q,\alpha)$ is decreasing in $\alpha \geq 1$. This implies that if α was unknown but had a constraint $\alpha \geq L \geq 1$, then the worst possible upper-bound computed through (3.8) is at $\alpha = L$. Stated otherwise, $\sigma^*(q,L)\hat{\lambda}_n$ would be an atleast $100\times(1-q)\%$ confidence bound for λ_n with unknown α and $\alpha \geq L \geq 1$.

FIG 3.2. *Plot of* $\sigma^*(q,\alpha)$ *vs.* α.



3.2.3. Gamma isotonic software reliability model

Another extension of the EISR model is the Gamma isotonic software reliability model (GISR) in which we have $T_i \sim Gamma(\lambda_i,\alpha)$ with $\lambda_1 \geq \ldots \geq \lambda_n$. Theorem 3.1 can be used on the GISR model to obtain confidence upper-bound for λ_n by assuming F to be the CDF of $Gamma(1,\alpha)$. As in the case of the WISR model, the scaling factor would be a function of α given by $\sigma(P,q,\alpha) = (\Gamma(P+\alpha)/(q\Gamma(\alpha)))^{(1/P)}$. The optimum value for the scaling factor, $\sigma^*(q,\alpha) = \min_P \sigma(P,q,\alpha)$, is achieved when P is the solution to

$$\log(\Gamma(P+\alpha)) - \log(\Gamma(\alpha)) - P\,\Psi(P+\alpha) = \log(q), \tag{3.9}$$

This can be interpreted to mean that, for $q = \Gamma(P+1)\exp(-P\Psi(P+1))$, the optimal scaling factor $\sigma^*(q,\alpha) = \exp(\Psi(P+1))$. This result is stated as follows:

Result 3.4. *Let* $T_i \sim Gamma(\lambda_i,\alpha)$, *with* $\lambda_1 \geq \ldots \geq \lambda_n$ *and* $\alpha \geq 0$.

Define $\hat{\lambda}_n$ *as in* (3.2)*, then*

$$P\left(e^{\Psi(P+\alpha)}\hat{\lambda}_n \leq \lambda\right) \leq \frac{\Gamma(P+\alpha)}{\Gamma(\alpha)} e^{-P\,\Psi(P+\alpha)}, \quad P \geq 1. \tag{3.10}$$

Plots of $\sigma^*(q,\alpha)$ vs. α are presented for some values of q in the right panel of Figure 3.2. It can be shown that $\sigma^*(q,\alpha)$ is increasing in α, which implies that, if $\alpha \leq U$, then the largest possible upper-bound computed through (3.10) is at $\alpha = U$. Alternatively, $\sigma^*(q,U)\hat{\lambda}_n$ would be an atleast $100 \times (1-q)\%$ confidence bound for λ_n with α unknown and with the constraint $\alpha \leq U$.

3.3. Extension to Non-Isotonic Software Reliability Models

The assumption that reliability improves after the detection and repair of every failure in a software product has been criticized by many. Sophisticated Bayesian models have been proposed to address this problem (see Basu and Ebrahimi (2003) for an example). We provide a simpler alternative for addressing this issue. Instead of assuming that $T_i \sim Exponential(\lambda_i)$ with $\lambda_{i+1} \leq \lambda_i$, we shall assume that $\lambda_{i+1} \leq \beta\lambda_i$, for some known $\beta \geq 1$. The assumption considers the possibility of $\lambda_{i+1} > \lambda_i$ (i.e., the reliability of the software could have worsened after the detection and fixing of the i^{th} defect).

Information about β may be derived from prior experience of a software test engineer, much like the Bayesian prior distributions regarding the failure rate of a software product in Bayesian software reliability models. The software test engineer could derive β from the failure rate of other software products that have been tested in the past. The parameter β can be interpreted as the ratio of two subsequent inter-failure times. Hence one possible estimate of β may be the mean observed ratio of successive inter-failure times of a similar/related software product.

Define $S_i = \beta^{-(n-i)} T_i$ and note that $S_i \sim Exponential(\gamma_i)$, where $\gamma_i = \lambda_i \beta^{n-i}$. Since $\lambda_{i+1} \leq \beta\lambda_i$, we have $\gamma_{i+1} \leq \gamma_i$ and hence $S_1, \ldots, S_n$ conforms to the requirements of Result 3.2. A confidence bound on $\gamma_n = \lambda_n$ can now be obtained by applying Result 3.2 on $S_i = \beta^{-(n-i)} T_i$. This is stated as follows:

Result 3.5. *Let $T_i \sim Exponential(\lambda_i)$ independently, with $\lambda_{i+1} \leq \beta\lambda_i$, for $i = 1, \ldots, n-1$, $\beta \geq 1$ known. Let t_i be the realization of T_i. Define*

$$\hat{\lambda}(\beta)_n = \frac{1}{max\left(\frac{\beta^{-n+1}t_1 + \cdots + t_n}{n}, \frac{\beta^{-n+2}t_2 + \cdots + t_n}{n-1}, \cdots, t_n\right)} \tag{3.11}$$

Then for $P \geq 1$,

$$P\left(e^{\Psi(P+1)}\hat{\lambda}(\beta)_n \leq \lambda_n\right) \leq \Gamma(P+1)e^{-P\Psi(P+1)}. \tag{3.12}$$

The extension of this result to non-isotonic Gamma or Weibull software reliability models is very similar. In particular, $\hat{\lambda}(\beta)_n$ can be used in (3.8) and (3.10) in place of $\hat{\lambda}_n$, as given by (3.11).

3.4. Extension to Geometric Isotonic Software Reliability

Instead of considering $T_i \sim Exponential$, one may consider $T_i \sim Geometric(p_i)$, with $p_1 \geq p_2 \geq \cdots \geq p_n$. Such a model is natural when the measured time to failure has discrete measurement units. The PAV algorithm can be used to obtain the ML estimate of p_n and is given by.

$$\hat{p}_n = \frac{1}{max\left\{t_n, \frac{t_n + t_{n-1}}{2}, \ldots, \frac{t_n + \ldots + t_1}{n}\right\}}, \tag{3.13}$$

where t_i is a realization of T_i. A confidence upper-bound for p_n in terms of $\hat{p}_n$ is of interest. Using arguments similar to those used in the proof of Theorem 3.1, it

TABLE 3.2

Expressions for λ_k, for $k = 1, \ldots, n$, corresponding to four patterns of improvement

Constant	$\theta_k = 1/3$
Linear	$\theta_k = 1 - 2k/3n$
Convex	$I(2k \leq n) \times (1 - 4k/3n) + I(2k > n) \times 1/3$
Concave	$I(2k \leq n) + I(2k > n) \times (1 - (4k - 2n)/3n)$

can be shown that $Y_1 = (T_1 + \cdots + T_n)/n, Y_2 = (T_2 + \cdots + T_n)/(n-1), \cdots, Y_n = T_n$ is a martingale and the Doob's maximal inequality can be applied to $\hat{p}_n$. Theorem 3.1 however, can no longer be used to obtain a confidence bound for p_n because $E(T_n^P)$ is no longer proportional to p_n^P; for example, $E(T_n^2) = (2 - p_n)/p_n^2$. An approximate confidence interval is possible by replacing p_n with $\hat{p}_n$ in the corresponding formula for the P^{th} moment. Such an approximation would compute an upper-bound for p_n. Result 3.6 states the result using the second moment of the geometric distribution.

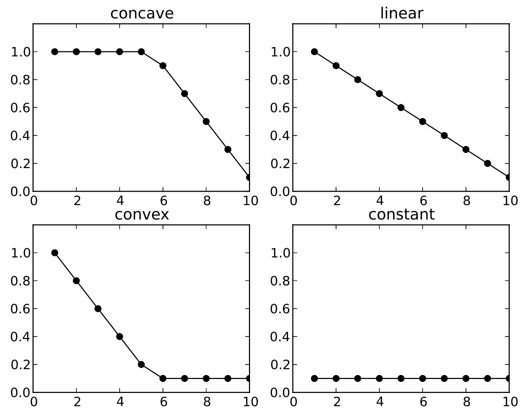
Result 3.6. *Let $T_i \sim Geometric(p_i)$ independently for $i = 1, \ldots, n$ with $p_1 \geq \ldots \geq p_n$. Define $\hat{p}_n$ as in* (3.13). *Then,*

$$P\left(\hat{p}_n\sqrt{\frac{2 - \hat{p}_n}{q}} \leq p_n\right) \leq q. \tag{3.14}$$

3.5. Simulation Study

We now investigate the performance of the upper-bound through simulations. We consider four different patterns of software reliability improvement. These are presented algebraically in Table 3.2 and presented graphically in Figure 3.3 for $n = 10$. Failure times T_i are simulated such that $T_i \sim Exponential(\lambda_i)$, for $i = 1, \ldots, n$, where λ_i's are obtained from one of the models in Table 3.2.

FIG 3.3. *Plot of λ_k vs. k for four different patterns of software reliability improvement with $n = 10$.*



The simulated data is analyzed using parametric models for software reliability, namely the Jelinski and Moranda model (Jelinski and Moranda, 1972) and the Moranda model (Moranda, 1975), which correspond to linear and convex models, respectively. The estimates of λ_n are obtained through the method of maximum likelihood. The Jelinski and Moranda model assumes $\lambda_k = \Lambda(N - k + 1)$, while the Moranda model assumes that $\lambda_k = \exp(\alpha - \beta k)$. Upper-bounds for λ_n with a required coverage probability of $100 \times (1 - q)\%$ are obtained from these model through parametric bootstrap. The simulated data is also analyzed through the EISR model and ML estimate of λ_n is obtained through (3.2). Also, $100 \times (1 - q)\%$ upper-bound for λ_n is obtained through (3.6). The performance evaluation studies the coverage probability and the relative size of the upper-bounds for λ_n, using 1000 simulations, from each of the four patterns of reliability improvement presented in Table 3.2. The evaluation is done for $n = 10, 25$ and 100, respectively, and the results are presented in Table 3.3. Coverage probabilities with substantial shortfall in the expected 95% coverage are highlighted in the table.

In the simulation study, the linear model was chosen as it exactly models the

Jelinski and Moranda model, thus providing an example where the properties of the proposed confidence bound can be studied when the correct model is available. The convex model was chosen as it can be approximated by the de-eutrophication model of Moranda, while the concave model was chosen as it cannot be approximated by either the Jelinski and Moranda model or the de-eutrophication model of Moranda. The simulation study reveals astonishing results concerning the coverage probability of the parametric bootstrap based confidence upper-bound for λ_n. The Jelinski and Moranda model when applied to data simulated from a model in which there is no improvement in reliability (i.e., the constant model of Table 3.2) computes 95% confidence bounds that have 35.6% coverage for $n = 10$, 0.7% for $n = 25$ and no coverage for $n = 100$, respectively. This indicates the bias of the results when the model is not true. For the concave model, the Jelinski and Moranda model produces 95% confidence bounds that have 92.6%, 47.1% and 0% coverage for $n = 10, 25$ and 100, respectively. When the true model is linear, the 95% confidence upper-bound from the Jelinski and Moranda model comes close to attaining the required coverage probability only when n= 100. These results alone should warn against using parametric model when the exact parametric nature of the underlying reliability improvement is not clear. The Moranda model fares slightly better than the Jelinski and Moranda model, especially for data simulated from the constant reliability improvement model. This can be attributed to $\beta = 0$ in the Moranda model being equivalent to the constant reliability model. The performance of the confidence bound from the moranda model is poor when applied to data simulated from the concave model; the 95% confidence upper-bound for λ_n has only 82%, 50.4% and 10% coverage for $n = 10, 25$ and 100, respectively.

All these anomalous coverage values have been highlighted in Table 3.3. Clearly, parametric models should not be used when the nature of the underlying reliability improvement is unknown. The confidence upper-bound for λ_n obtained through EISR model, on the other hand, has more than the required confidence across the different models, as expected. The relative size of the confidence bounds obtained through (3.6) are generally larger (except for $n = 10$), as expected, since these are upper-bounds. Since the parametric models have substantial shortfall of the required coverage in certain cases, we note that method based on (3.6) may be more appropriate for obtaining a confidence upper-bound for λ_n when the underlying model for software reliability is not known.

3.6. Applications

Failure times of a software sub-system for 136 iterations of debugging for a commercial software was obtained from Musa (2012) (Dataset 6). Based on this dataset we consider the problem of estimating the failure time distribution of the next software failure using data corresponding to the last 10 iterations. The data is provided in Table 3.4.

We assume that every inter-failure time T_i, for $i = 1, \ldots, n$, is exponentially distributed with rate parameter λ_i. However, unlike the parametric models, we make a generic assumption of $\lambda_{i+1} \leq \lambda_i$ which corresponds to non-decreasing reliability of the software. The PAV algorithm based ML estimate of λ_{10} and the corresponding confidence intervals using (3.6) with a minimum confidence of $50\%, 90\%$ and 95% are provided in Table 3.5.

We now consider another example concerning user-reported software defects.

TABLE 3.3
The relative size of the 95% upper confidence bound of λ_n with the coverage probabilities in parentheses

$n = 10$

Model	Jelinski Moranda	Moranda	$\hat{\lambda}_n$
Constant	0.95 **(35.6)**	4.71 (98.3)	3.87 (99.1)
Linear	4.08 (100.0)	7.26 (100.0)	7.98 (99.1)
Convex	5.53 (100.0)	10.87 (100.0)	9.72 (98.9)
Concave	1.87 (92.6)	2.51 **(82.0)**	4.47 (99.0)

$n = 25$

Model	Jelinski Moranda	Moranda	$\hat{\lambda}_n$
Constant	0.57 **(0.7)**	2.28 (97.4)	3.44 (99.1)
Linear	2.42 (100.0)	3.85 (100.0)	5.92 (99.0)
Convex	3.45 (100.0)	6.24 (100.0)	7.43 (99.2)
Concave	1.03 **(47.1)**	1.14 **(50.4)**	3.68 (98.9)

$n = 100$

Model	Jelinski Moranda	Moranda	$\hat{\lambda}_n$
Constant	0.38 **(0.0)**	1.43 (96.2)	3.29 (99.6)
Linear	1.41 (96.0)	2.60 (100.0)	4.39 (99.1)
Convex	2.02 (99.6)	4.21 (100.0)	5.07 (99.4)
Concave	0.60 **(0.0)**	0.75 **(10.1)**	3.35 (99.7)

TABLE 3.4
The inter-failure times (CPU seconds) for the last 10 iterations.

Iteration	1	2	3	4	5	6	7	8	9	10
Failure Time	40	2	86	221	6	891	23	4	437	66

TABLE 3.5
ML estimate $\hat{\lambda}_{10}$ and its corresponding confidence bound, $\hat{\lambda}_{10}^{q}$, with at least $100 \times q\%$ coverage.

$\hat{\lambda}_{10}$	$\hat{\lambda}_{90}^{0.50}$	$\hat{\lambda}_{10}^{0.95}$	$\hat{\lambda}_{10}^{0.99}$
0.0035	0.0067	0.013	0.016

Bug-databases record defects discovered by the users of the software after its release. Software bug-database have become an important source of data for estimating the reliability of a software after its release. A bug-database can be queried to determine the number of days between successive defect discoveries. Typically, bug-databases will classify the defects into a number of defect classes with security defects being an important class of defects for the software community. Table 3.6 presents data corresponding to 15 most recent security defects retrieved from the publicly available bug-database of Python 2.6, a popular scientific scripting language. The bug-database was accessed at http://bugs.python.org/issue?@template=search on January 31, 2012 and the 15 most recent security defects that were confirmed as on January 31, 2012 were retrieved.

TABLE 3.6
Number of inter-discovery days for the discovery of the k^{th} defect; T_1 corresponds to days since release of the first version of the software.

k	1	2	3	4	5	6	7	8	9	
t_k	2150	83	137	5	28	32	309	62	164	38
k	10	11	12	13	14	15				
t_k	70	170	251	16	285	24				

Typically after the discovery and reporting of a defect in the bug-database, the defect is fixed and the reliability of the software is expected to improve or remain the same. After the defect is reported and fixed, the quantum of improvement in reliability is typically not known. The isotonic geometric regression model is used, for the sake of illustration, to estimate the reliability of a software, with respect to a critical security defect, using (3.14). In particular, the number of inter-discovery days for observing the k^{th} security defect can be assumed to be $Geometric(p_k)$, for $k = 1, \ldots, n$, with $p_1 \geq, \ldots, \geq p_n$. The PAV algorithm can be used to obtain the MLE $\hat{p}_n$ of p_n. One can use (3.14) to obtain a confidence upper-bound for p_n with a minimum required coverage probability. The results are presented in Table 3.7 for minimum coverage probabilities of 50%, 90% and 95%. The upper-bounds on p_n, with $n = 15$, can be converted to lower-bounds on the expected number of days till the discovery of the next security defect. In particular, from Table 3.7, we may conclude that the expected number of days for the 16^{th} security defect to be discovered is $1/\hat{p}_{16} = 154$ days, with a 95% lower bound being $1/\hat{p}_{15}^{0.95} = 24$ days.

TABLE 3.7
ML estimate $\hat{p}_{15}$ and its corresponding confidence bounds with minimum required coverage of 50%, 90%, 95% and 99%.

$\hat{q}_{16}$	$\hat{q}_{16}^{0.50}$	$\hat{q}_{16}^{0.90}$	$\hat{q}_{16}^{0.95}$
0.0064	0.013	0.028	0.040

3.7. Concluding Remarks

To conclude, three generic models for software reliability improvement have been proposed using Exponential, Gamma and Weibull distributional families. The proposed method for computing confidence bounds, based on the PAVA based ML estimate, presents a new approach of constructing confidence intervals that circumvent the important concerns raised by Andrews (2000) in using bootstrap procedures in constrained parameter spaces. The approach also avoids using asymptotic procedures as they suffer from concerns related to their being as many model parameters as data-points. The proposed confidence bounds are simple to compute and interpret as demonstrated in Section 3.6. The simulations study indicates that the proposed method is superior to a parametric model for obtaining confidence bounds for software reliability in the absence of information regarding the nature of software reliability improvement. Extension of the method for bounded improvement in reliability has been discussed along with extensions to Geometric isotonic regression models. The proposed methodology may be particularly useful to estimate the reliability of software with rare and catastrophic failures, where failures will be few and very few assumptions can be made regarding the improvement in reliability after each debugging effort.

Chapter 4

Dependent Isotonic Software Reliability Models

The consequences of an act affect the probability of its occurring again.

B. F. Skinner

4.1. Introduction

Estimation and statistical inference for the dependent isotonic software reliability models, proposed earlier in Chapter 2, are developed in this chapter. In particular, we develop conservative bounds similar to those in Chapter 3. The bounds proposed in Chapter 3 cannot be used for the dependent isotonic software reliability models because we make use of the independence of the inter-failure times to prove Theorem 3.1. Also, the bounds of Chapter 3 cannot consider right censoring of the most recent failure time, i.e, it cannot consider failure free operation of the software since the last failure. Another problem with the bounds proposed in Chapter 3 is that even under the assumption of independence there is no guarantee that they will be monotonic. In particular, the lower confidence limits for reliability, after successive discovery of defects and their repair, may not be increasing even though we assume an increasing reliability. For a non-statistician this may be hard to interpret as these estimates will be at odds with the assumption of increasing reliability after every defect discovery and repair. To address both these problems we propose a novel and simple method for constructing an *exact* one-sided confidence bound, with a minimum coverage probability, for the single index parameter of the failure time distribution corresponding to the isotonic software reliability model. The method is based on the distribution of $max(T_1, \ldots, T_n)$, where T_k's denote the observed inter-failure times, of say, n successive failures. The proposed bound is valid even for small data sizes, can be modified to consider non-independence of the T_k's and be used when there is no improvement in reliability after fixing a defect. Additionally, the bound can be used even when the time to failure T_n of the current version of the product is right censored. This chapter is organized as follows. Section 4.2 formulates the problem and presents the main result in a general framework. Section 4.3 studies the properties of the proposed upper-bound through simulation. Section 4.4 presents the application of the proposed upper-bound to two publicly available software reliability datasets and a new dataset retrieved from the automatic error-logger of a workstation. We would like to highlight that the application of the method to the dataset in section 4.4 reveals that two popular models for software reliability might underestimate the reliability of the underlying software. Although we expect the reliability of the product not to decrease with every subsequent version, this might not happen in some situations. To address this issue, we demonstrate how the method can be modified to consider a bounded decrease in reliability of the subsequent product versions (See Section 4.2). Section 4.5 ends with some concluding remarks.

The dependent isotonic reliability model is a very generic software reliability growth model that can consider arbitrary dependence between successive failure times. Estimating the failure time distribution of the latest version of the software and providing confidence intervals for metrics derived from the estimated failure time distribution is a challenge. This is because of (i) the arbitrary nature of the dependence between the failures (ii) the possibility of equality in some or all of the failure time distributions. Recently, there has been increased interest in characterizing the asymptotic distribution of PAV estimator in the context of dosage studies (Jewell and Kalbfleisch, 2004; Tebbs and Swallow, 2003; Bhattacharya and Kong, 2007). The inconsistency of bootstrap procedures when the parameter space has non-strict inequality constraints has been noted by Andrews (2000). Li, Taylor and Nan (2010) proposed a modified bootstrap estimator for estimating two binomial proportions in the presence of order restrictions in small samples. Similar bootstrap method seems difficult for the problem being considered here since there is exactly one observed failure time for every version. Every additional version introduces an additional unknown parameter, whereas in the method of Li, Taylor and Nan the number of parameters is fixed at two. The method of constructing a conservative one-sided confidence bound proposed in this chapter is a novel solution that addresses both the problems.

4.2. The Main Result

Let $T_k \sim F_k(.)$, for $k = 1, \ldots, n$, where $F_k(.)$ is the cumulative distribution function for T_k. Since successive versions of the product have non-decreasing reliability, we will impose the constraint that $F_k(t) \leq F_{k-1}(t)$ for all t. An estimate of $F_n(t)$, for any $t \in \mathbb{R}$, along with an upper confidence bound is of interest. If $F_1, \ldots, F_n$ are assumed to be arbitrary CDF's with the above stochastic ordering, then estimation of $F_n(t)$, for all $t \in \mathbb{R}$, may not be feasible, specifically for $t > max(T_1, \ldots, T_n)$. However, if F_k's belong to a family of distributions indexed by a real-valued single parameter, i.e $F_k(t) = F(t; \theta_k)$, with $F(t, \theta)$ increasing in θ for all t in the support of F, then under the assumption of mutual independence, maximum likelihood estimation of θ_k's, and in particular θ_n, may be possible for certain families of distribution. In order to ensure a margin of safety, it is more important to obtain a confidence bound with a specified minimum coverage probability for θ_n rather than just obtaining a point estimate for θ_n. As in the context of software reliability models (Jelinski and Moranda 1972), we assume $\theta_1 \geq \ldots \geq \theta_n$, leading to $F(t; \theta_1) \geq \cdots \geq F(t; \theta_n)$ for all t. For example, one may consider T_k's to follow a Pareto distribution with support on $(0, \infty)$ and assume $F_k(t) = 1 - (1/x)^{\theta_k}$, for $k = 1, \ldots,$ with $\theta_1 \geq \theta_2 \geq \ldots$. Alternatively we could assume T_k's to follow a Rayleigh distribution with $F_k(t) = 1 - exp(-t^2/\theta_k)$ with $\theta_1 \geq \theta_2 \geq \ldots \geq \theta_k$.

The assumption that θ_k may possibly be equal to θ_{k-1} implies that the number of unknown parameters is not specified. Such an observation indicates that asymptotic methods for computing confidence bounds for θ_n as $n \rightarrow \infty$ may not be appropriate. The possibility of $T_1, \ldots, T_n$ being a dependent sequence of observations, with the nature of the dependence also being unknown, only complicates the problem. It is in these contexts that we propose Theorems 4.1 and 4.2, which can be used to obtain confidence bounds for the parameter θ_n with a minimum coverage probability. Theorem 4.1 is formulated under the assumption of $T_1, \ldots, T_n$ being independent, while Theorem 4.2 is formulated without the assumption of independence.

Define $\mathcal{F} = \{F(., \theta), \theta \in \mathbb{R}\}$ to be a family of cumulative distribution functions with support in $(-\infty,\infty)$ indexed by the parameter $\theta \in \Theta \subseteq \mathbb{R}$, with the property that $F(t,\theta) > F(t,\theta')$, if $\theta > \theta'$, $\forall t \in (-\infty,\infty)$. Let $T_1 \sim F(.,\theta_1), T_2 \sim F(.,\theta_2), \ldots, T_n \sim F(.,\theta_n), \ldots$ with $\theta_1 \geq \theta_2 \geq \ldots \geq \theta_n \geq \ldots$, be an independent sequence of observations. Define $F^{-1}(p,\theta) = inf\ \{t\ :\ F(t,\theta) \geq p\}$. Let $t_1, t_2, \ldots, t_n$ be realizations of $T_1, \ldots, T_n$. Based on $t_1, t_2, \ldots, t_n$, define the statistic

$$\hat{\theta}_n^p = min\left\{\theta : max(t_1,\ldots,t_n) > F^{-1}\left(p^{1/n},\theta\right), \theta \in \Theta\right\}. \tag{4.1}$$

Note that, since $F^{-1}(.;\theta)$ is a non-increasing function of θ, this minimum $\hat{\theta}_n^p$ exists.

Theorem 4.1. *The statistic $\hat{\theta}_n^p$ has the property $P(\hat{\theta}_n^p < \theta_n) \leq 1-p$. This implies that $\hat{\theta}_n^p$ is an at least $100 \times p$% upper-bound for the parameter θ_n, that is, $P(\hat{\theta}_n^p \geq \theta_n) \geq p$. In other words, $\left\{\theta : \theta \leq \hat{\theta}_n^p\right\}$ is an at least $100 \times p$% one-sided confidence interval.*

Proof. Note that $P\left[max\ (T_1,\ldots,T_n) \leq \lambda\right] = \prod_{j=1}^n F(\lambda,\theta_j)$
$\geq [F(\lambda,\theta_n)]^n$, since $F(\lambda,\theta_n) \leq F(\lambda,\theta_j)$, for $j = 1,\ldots,n$.
Define $A_k = \left\{max(T_1,\ldots,T_n) > F^{-1}\left(p^{1/n},\theta_n - \frac{1}{k}\right)\right\}$, for $k = 1,2,\ldots$. Then,

$$\begin{aligned} P(A_k) &\leq 1 - \left[F\left(F^{-1}\left(p^{1/n},\theta_n - 1/k\right),\theta_n\right)\right]^n \\ &\leq 1 - \left[F\left(F^{-1}\left(p^{1/n},\theta_n\right),\theta_n\right)\right]^n \\ &\leq 1-p \end{aligned}$$

since $F^{-1}\left(p^{1/n},\theta_n - \frac{1}{k}\right) \geq F^{-1}\left(p^{1/n},\theta_n\right)$ and $F\left(F^{-1}\left(p^{1/n},\theta_n\right),\theta_n\right) \geq p^{1/n}$.
Now, $P(\hat{\theta}_n^p < \theta_n) =$

$P\left(\exists\,\varepsilon > 0\, s.t.\, max(T_1,\ldots,T_n) > F^{-1}\left(p^{1/n},\theta_n - \delta\right), \forall\, 0 < \delta \leq \varepsilon\right) =$
$P\left(\bigcup_{j=1}^{\infty}\bigcap_{k=j}^{\infty} A_k\right) = P\left(\liminf_k A_k\right) \leq \liminf_k P\left(A_k\right) \leq 1 - p.$

$\square$

It is important to note that while $\hat{\theta}_1^p$ is also a confidence bound for θ_n since $\theta_n \leq \theta_1$, the bound $\hat{\theta}_n^p$ can be expected to be smaller as it is based on n observations $T_1,\ldots,T_n$, while $\hat{\theta}_1^p$ is only based on T_1.

Define $\mathcal{F} = \{F(.,\theta), \theta \in \mathbb{R}\}$ to be a family of cumulative distribution as before. Let $T_1 \sim F(.,\theta_1), T_2 \sim F(.,\theta_2), \ldots, T_n \sim F(.,\theta_n), \ldots$ with $\theta_1 \geq \theta_2 \geq \ldots \geq \theta_n \geq \ldots$. be a possibly dependent sequence of observations. Let t_i be a realization of T_i. Based on $t_1,\ldots,t_n$, define the statistic

$$\tilde{\theta}_n^p = min\left\{\theta : max(t_1,\ldots,t_n) > F^{-1}\left(1 - \frac{1-p}{n}, \theta\right), \theta \in \Theta\right\}. \qquad (4.2)$$

Theorem 4.2. *The statistic* $\tilde{\theta}_n^p$ *has the property* $P(\tilde{\theta}_n^p < \theta_n) \leq 1 - p$, *for* $k = 1,\ldots,n$. *This implies that* $\tilde{\theta}_n^p$ *is an at least* $100 \times p\%$ *upper-bound for the parameter* θ_n, *even if* $T_1,\ldots,T_n$ *are dependent.*

Proof. Note that $P\left(max\left\{T_1,\ldots,T_n\right\} > \lambda\right) = P\left(\bigcup_j \left\{T_j > \lambda\right\}\right)$
$\leq \sum_{j=1}^{n} 1 - F(\lambda,\theta_j) \leq n[1 - F(\lambda,\theta_n)]$, since $F(\lambda,\theta_n) \leq F(\lambda,\theta_j)$, for $j = 1,\ldots,n$. The rest of the proof follows the arguments of Theorem 4.1. $\square$

In situations when T_n is right censored at t_0, the proposed bounds $\hat{\theta}_n^p$ and $\tilde{\theta}_n^p$ may be computed by replacing T_n with t_0. The statistics will still be a valid upper-bound for the parameter with a minimum coverage probability, since $max(T_n,\ldots,T_n) \geq max(T_n,\ldots,T_{n-1},t_0)$. This is particularly useful as the time to failure for the latest version of the product may often be right censored. Note that, with T_n being right censored, the maximum likelihood estimate of θ_n does not exist. However, $\hat{\theta}_n^p$ can still be obtained and used as an at least $100 \times p$ % upper-bound for θ_n.

Monotonicity of the Confidence Bounds

There is no guarantee that the bounds $\hat{\theta}_n^p$ and $\tilde{\theta}_n^p$ will monotonically decrease with n. For example, an observation T_{n+1} s.t. $T_{n+1} \leq max(T_1, \ldots, T_n)$ will result in $\hat{\theta}_{n+1}^p > \hat{\theta}_n^p$. Due to the nature of the sequential quality improvement, this increase is to be expected as one would intuitively require the time to failure of $n+1^{th}$ version of the product to be more than the time to failure of all the preceding versions. Hence a quicker time to failure of the product in the $n+1^{th}$ iteration would indicate that a decrease of the θ parameter over the preceding versions is doubtful, requiring an increase in $\hat{\theta}_{n+1}^p$. It is important to note that the problem of non-monotonicity of bounds is also present in the MLE estimator using the Pooled Adjacent Violator Algorithm. See Figure 4.3 for an example. The monotonicity of the confidence bound or an any estimate for θ_n may however be desirable as it is easily interpreted by practitioners who will otherwise find a sudden increase in the bound for θ_n hard to explain when the underlying model postulates that the θ_n decreases with n. To this end, we propose the statistics

$$\hat{\theta}_{n,m}^p = min\{\theta : max(t_1, \ldots, t_n) > F^{-1}(p^{1/m}, \theta)\} \tag{4.3}$$

$$\tilde{\theta}_{n,m}^p = min\{\theta : max(t_1, \ldots, t_n) > F^{-1}(1-(p/m), \theta)\}, \tag{4.4}$$

where m is chosen to be an upper-limit for the number of sequential improvements that will be conducted.

Theorem 4.3. *The statistics $\hat{\theta}^p_{n,m}$ and $\tilde{\theta}^p_{n,m}$ are at least $100 \times p\%$ confidence bound for θ_n under the assumption of Theorem 4.1 and 4.2, respectively. Also, $\hat{\theta}^p_{n,m}$ and $\tilde{\theta}^p_{n,m}$ are both non-increasing in n for $n \leq m$.*

Proof. Define $A_{k,n,m} = \left\{max(T_1,\ldots,T_n) > F^{-1}\left(p^{1/m}, \theta_n - \frac{1}{k}\right)\right\}$, for $k = 1,2,\ldots$ By following the steps of the proof of Theorem 4.1, it is can be shown that $P(A_{k,n,m}) \geq p^{n/m} \geq p$. With this result, the proof of $P(\hat{\theta}^p_{n,m} > \theta_n) \geq p$ follows the same arguments as the rest of the proof of Theorem 4.1.

To prove $\hat{\theta}^p_{n,m}$ is decreasing in n, for $n \leq m$, note that $max(T_1,\ldots,T_n)$ is increasing in n and $F^{-1}(p^{1/m},\theta)$ is decreasing in θ. Hence if $max(T_1,\ldots,T_n) \geq F^{-1}(p^{1/m},\theta)$, we will have $max(T_1,\ldots,T_{n+1}) \geq F^{-1}(p^{1/m},\theta)$, which proves that $\hat{\theta}_{n+1,m} \leq \hat{\theta}_{n,m}$. The proof for $\tilde{\theta}^p_{n,m}$ follows similar arguments. $\square$

It is easy to see that $\hat{\theta}^p_n$ and $\tilde{\theta}^p_n$ are less than or equal to $\hat{\theta}^p_{n,m}$ and $\tilde{\theta}^p_{n,m}$, respectively (See also Figure 4.3). For the purpose of analyzing sequential improvement plans, with the requirement of monotonic confidence bounds, one may choose $m = n + c$, where c is the number of sequential improvements expected in the future.

4.3. A Simulation study

In order to study the performance of the proposed conservative upper-bounds numerically, we consider a model in which $T_k \sim Exponential(\theta_k)$, for $k = 1,\ldots,n$. Theorems 4.1 and 4.2 can be used to provide a conservative $100 \times p\%$ confidence bounds for θ_n. We consider four patterns of reliability improvement, namely, constant, linear, convex and concave, as algebraically described in Table 4.1 and presented visually in in Figure 4.3 for $n = 10$.

TABLE 4.1

Expressions for θ_k, for $k = 1, \ldots, n$, corresponding to four patterns of reliability improvement

Constant	$\theta_k = 1/3$
Linear	$\theta_k = 1 - 2k/3n$
Convex	$I(2k \leq n) \times (1 - 4k/3n) + I(2k > n) \times 1/3$
Concave	$I(2k \leq n) + I(2k > n) \times (1 - (4k - 2n)/3n)$

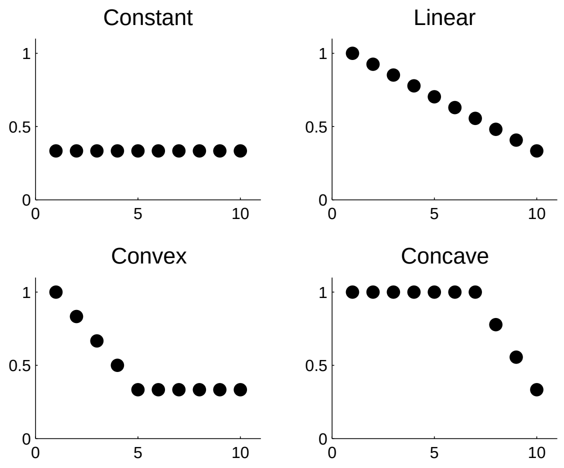


FIG 4.1. *Plots of θ_k vs k for $k = 1, \ldots, 10$ corresponding to four patterns of reliability improvement.*

If we consider the case when $n = 10$, the conservative upper-bound $\hat{\theta}_{10}^{p}$ for θ_{10}, under the assumption of independence, is obtained by simulating $T_1, \ldots, T_{10}$ independent of each other with θ_k's as per each of the four patterns presented in Figure 4.3. To consider the possibility of dependence between the T_k's, we simulate the correlated T_i's, for $i = 1, \ldots, 10$, using a multivariate Gaussian copula with a copula correlation of 0.5 (Nelson 1999) as follows: simulate $\{Z_1, \ldots, Z_{10}\}$ from a multivariate normal distribution with dimension 10, such that each Z_j has mean 0, variance 1 and with a covariance between any Z_j and Z_k set to 0.5, for $1 \leq j < k \leq 10$. Compute $U_k = \Phi(Z_k)$, for $k = 1, \ldots, 10$, where $\Phi(.)$ is the standard normal CDF. Note that each U_k is $Uniform(0,1)$, and since Z_k and Z_j are not independent, U_k and U_j are also not independent. Compute $T_k = -log(1-U_k)/\theta_k$, for $k = 1, \ldots, 10$. Each T_k is marginally distributed as an $Exponential(\theta_k)$; however, due to the dependence between $U_1, \ldots, U_{10}$, the variables $T_1, \ldots, T_{10}$ are also dependent. The simulated $T_1, \ldots, T_{10}$ are used to obtain $\tilde{\theta}_{10}^{p}$. The actual coverage probabilities for $\hat{\theta}_{10}^{p}$ and $\tilde{\theta}_{10}^{p}$ are estimated from 1000 such simulations with $p = 0.90$, 0.95 and 0.99 and are presented in Table 4.2.

The results of the simulation study are in concordance with the statements of Theorems 4.1 and 4.2 as the upper-bounds achieve their intended minimum coverage probabilities. Note that, when all the θ_k's are equal, for $k = 1, \ldots, 10$, and the T_k's are independent, the coverage probability of $\hat{\theta}_{10}^{p}$ is close to p, for $p = 0.90$,0.95 and 0.99, indicating that, in such a case, the bounds will be tight, as is expected from the proof of Theorem 4.1. We will next study the average size of the conservative upper-bound which we define as the ratio of the upper-bound to the true parameter value, given by $\hat{\theta}_{n}^{p}/\theta_n$ and $\tilde{\theta}_{n}^{p}/\theta_n$, respectively. The results are presented in Table 4.3 for p = 90%,95% and 99%. From Figure 4.3, it is clear that for any k, θ_k is on an average the largest for the concave model, followed by the linear, convex and then the constant model. Such an ordering leads to $max(T_1,...,T_n)$ being the largest, on the average, for the constant model and least for the concave model. This implies the reverse ordering, on the average size, for the proposed upper-bounds from Theorems 4.1 and 4.2. This is evident in Table 4.3.

To illustrate the proposed confidence bounds visually, we consider data simulated from the convex model with n = 25 (see Table 4.1) and compute the upper-bounds for θ_k, namely $\hat{\theta}_k^p = -log(1-p^{1/k})/max(T_1,\ldots,T_k)$ and $\hat{\theta}_{k,m}^p = -log(1-p^{1/m})/max(T_1,\ldots,T_k)$, with $m = 25$ and plot them for $k = 1,\ldots,25$. Also, for comparison, the MLE of θ_k and the corresponding upper-bounds $\hat{\theta}_k^p$ based on $T_1,\ldots,T_k$ are shown on the same plot. Such a plot, for one set of simulated data, is presented in Figure 4.3 with $p = 0.95$. As alluded to before, the ML estimate $\hat{\theta}_k^{ML}$ of θ_k and its upper-bound $\hat{\theta}_k^p$, based on $T_1,\ldots,T_k$, are not monotonic with k as shown in Theorem 4.3. Further $\hat{\theta}_{k,m}^p$ is monotonic with k as required by Theorem 4.3. Note that the size of this monotonic bound is larger than the non-monotonic bound $\hat{\theta}_k^p$. We leave it up to the practitioner to choose between the two bounds based upon the requirements of the problem being solved. In all subsequent analysis, we will present the results based on the bound $\hat{\theta}_n^p$.

We now proceed to comparing the confidence bound $\hat{\theta}_k^p$ with confidence bounds estimated by two well known parametric models, namely the Jelinski and Moranda model and the Moranda model. The Jelinski and Moranda model assumes that $\theta_k = \alpha - \beta k$ for $k = 1,\ldots,n$, while the Moranda model assumes

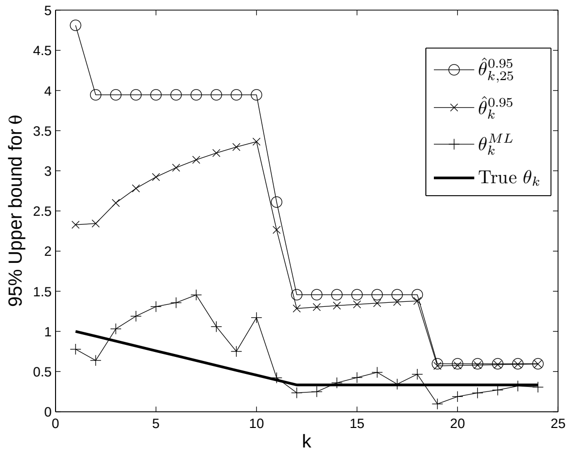


FIG 4.2. *Comparison of the two confidence bounds $\hat{\theta}^p_{k,m}$ and $\hat{\theta}^p_k$ along with the MLE for θ_k obtained using the PAV Algorithm for a simulation of data from the convex model*

TABLE 4.2
Estimated coverage probability of the upper-bounds for the four patterns of decrease of reliability improvement.

	Independent			Dependent		
Pattern	$\hat{\theta}^{0.90}_{10}$	$\hat{\theta}^{0.95}_{10}$	$\hat{\theta}^{0.99}_{10}$	$\tilde{\theta}^{0.90}_{10}$	$\tilde{\theta}^{0.95}_{10}$	$\tilde{\theta}^{0.99}_{10}$
Constant	0.90	0.95	0.99	0.94	0.97	0.99
Linear	0.99	0.99	0.99	0.98	0.99	1.00
Concave	0.94	0.97	0.99	0.96	0.99	0.99
Convex	0.98	0.99	1.00	0.99	0.99	1.00

TABLE 4.3
Average values of $\hat{\theta}^p_{10}/\theta_{10}$ and $\tilde{\theta}^p_{10}/\theta_{10}$ for the four patterns of reliability improvement.

	Independent			Dependent		
Model	$\hat{\theta}^{0.90}_{10}$	$\hat{\theta}^{0.95}_{10}$	$\hat{\theta}^{0.99}_{10}$	$\tilde{\theta}^{0.90}_{10}$	$\tilde{\theta}^{0.95}_{10}$	$\tilde{\theta}^{0.99}_{10}$
Constant	1.82	2.14	2.90	3.60	3.84	5.13
Linear	5.55	6.50	8.32	9.67	10.37	15.56
Convex	2.59	3.05	3.89	4.67	5.48	6.91
Concave	7.07	8.07	10.63	13.70	12.79	17.48

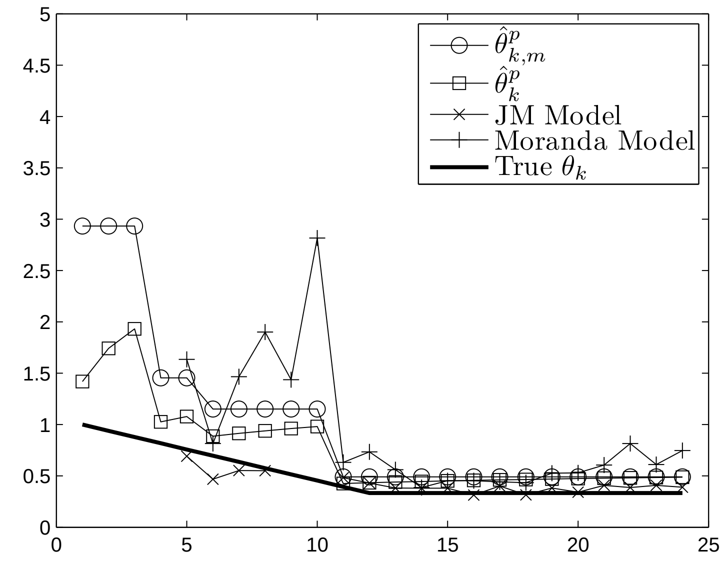


FIG 4.3. *Comparison of the two confidence bounds $\hat{\theta}^p_{k,m}$ and $\hat{\theta}^p_k$ with 95% confidence bounds obtained using parametric models*

that $\theta_k = exp(\alpha - \beta k)$ for $k = 1, \ldots, n$. The parameter θ_k can be estimated as $\theta_k = \hat{\alpha} - \hat{\beta}k$ for the Jelinski and Moranda model, while for the Moranda model, it can be estimated as $\theta_k = exp(\hat{\alpha} - \hat{\beta}k)$, where $\hat{\alpha}$ and $\hat{\beta}$ are the ML estimators from the corresponding models. $100 \times p$% confidence intervals for θ_n can be obtained through parametric bootstrap. For a typical simulation from the convex pattern with $m = 25$, a graph of the proposed confidence bounds $\hat{\theta}^p_k$, its monotonic counterpart $\hat{\theta}^p_{k,m}$, for $p = 0.95$ and the 95% confidence bounds obtained from the Jelinski and Moranda and the Moranda models through parametric bootstrap are plotted against $k = 1, \ldots, m$ in Figure 4.3. Observe from the figure that the confidence bounds obtained through the parametric models are not monotonic.

In order to evaluate the precision of the upper-bound $\hat{\theta}^p_n$, we compute the average size of the upper-bound, as defined before, over 1000 simulations. We present such a comparison for n = 10, 25 and 100 using the four models of reliability improvement described in Table 4.1. The results are presented in Table 4.4 for p = 0.95. Note that the average size and coverage probabilities of the confidence bounds computed through the two parametric models are similar to those presented in Table 3.3 in Chapter 3. Observe that the conservative upper-bound $\hat{\theta}_n^p$ has the minimum required coverage probability of 95% across all the four types of reliability improvement while the parametric model has a coverage that is much lesser than the intended coverage. This shortfall in the required coverage for the confidence bounds obtained through the Jelinski and Moranda model increases dramatically with n for the convex and constant patterns. This may be due to the Jelinski and Moranda model not being a good approximation for the constant and concave reliability improvement models. The confidence bounds obtained through the Moranda model also suffers from a similar problem for the concave reliability improvement patterns. The simulation study clearly demonstrates the dangers of using an incorrect parametric model for estimating the reliability of the software. While the proposed bounds are larger, they consistently have the required coverage both in small and large samples as expected from Theorem 4.1.

4.4. Applications

In this section, we consider three application areas for the proposed conservative upper-bound and illustrate with an analysis of a dataset from each area.

TABLE 4.4

Comparison of average value of $\hat{\theta}_n^{95}/\theta_n$ to 95 % confidence bounds obtained by using parametric bootstrap. The coverage probability percentages are given in brackets.

Method	constant	linear	convex	concave
		n = 10		
$\hat{\theta}_n^p$	2.16 (95.6)	6.11 (99.5)	3.10 (97.1)	8.11 (99.8)
Jelinski and Moranda	0.96 (35.2)	4.09 (99.8)	5.52 (99.2)	1.87 (92.1)
Moranda Model	4.71 (98.9)	7.26 (99.9)	10.87 (99.4)	2.51 (82.0)
		n = 25		
$\hat{\theta}_n^p$	1.80 (95.0)	4.93 (99.6)	2.26 (97.5)	6.49 (99.8)
Jelinski and Moranda	0.57 (0.7)	2.38 (100.0)	3.44 (100.0)	1.03 (47.1)
Moranda Model	2.27 (99.2)	5.92 (99.0)	7.43 (99.3)	3.68 (98.7)
		n = 100		
$\hat{\theta}_n^p$	1.53 (94.7)	3.62 (99.9)	1.80 (97.1)	4.83 (99.9)
Jelinski and Moranda	0.38 (0.0)	1.43 (96.1)	2.02 (99.4)	0.60 (0.0)
Moranda Model	1.46 (96.1)	2.62 (100.0)	4.22 (100.0)	0.73 (10.6)

4.4.1. *A Sequential quality improvement plan with non-decreasing exponential failure times*

Consider a sequential quality improvement plan for a product with n quality improvement iterations. First, a prototype of the product is tested till its first failure, at time T_1 following $Exponential(\theta_1)$ distribution with $E(T_1) = 1/\theta_1$, is observed. After the first failure, the production process goes through a quality improvement exercise resulting in the second prototype which is again tested till its failure, after time T_2 following $Exponential(\theta_2)$ distribution, is observed. We assume that due technical diligence is followed during the quality improvement exercise (i.e., product's quality does not deteriorate) so that $\theta_1 \geq \theta_2$ can be assumed. The n such successive improvement exercises give rise to n failure times $T_1, \dots, T_n$, with $T_k \sim Exponential(\theta_k)$, for $k = 1, \dots, n$, and $\theta_1 \geq \dots \geq \theta_n$. Let t_i be a realization of T_i. A statistician is expected to estimate θ_n and provide an at least 95% upper-bound (one-sided confidence interval) for θ_n. Such an upper-bound can be used to obtain an at least 95% lower bound for the probability that the nth improved version does not fail before a certain time t_0 (that is, the reliability). Theorem 4.2 can be used to compute an at least $100 \times p$ % upper-bound for θ_n *without making any assumption about the nature of dependence* between T_k's as

$$\tilde{\theta}_n^p = \frac{log(n) - log(1-p)}{max(t_1, \ldots, t_n)}, \tag{4.5}$$

where t_i is a realization of T_i.

Under the assumption of independence of T_k's, the MLE of θ_n can be obtained through the PAV algorithm (Ayers 1955) as given by

$$\hat{\theta}_n = \frac{1}{max\left\{t_n, \frac{t_n+t_{n-1}}{2}, \ldots, \frac{t_n+\ldots+t_1}{n}\right\}}. \tag{4.6}$$

However, since the asymptotic distribution of $\hat{\theta}_n$ is difficult to obtain (See Li, Taylor and Nan 2010 for an example relating to binomial distribution), finding a $100 \times p$% confidence interval for θ_n can be difficult. On the other hand, the proposed upper-bound $\hat{\theta}_n^p$ as given by

$$\hat{\theta}_n^p = \frac{-log\left(1 - p^{1/n}\right)}{max(t_1, \ldots, t_n)} \tag{4.7}$$

provides a conservative one-sided confidence interval, when the T_k's can be assumed to be independent.

The Dataset of Musa (2012) consists of software failure times for 136 iterations of debugging for a sub-system of a commercial software. In reality, failure times

TABLE 4.5
The inter-failure times (CPU seconds) for the last 10 iterations

Iteration (k)	1	2	3	4	5	6	7	8	9	10
Failure Time (t_k)	40	2	86	221	6	891	23	4	437	66

from this many iterations are hardly available and hence we consider the more realistic problem of estimating the reliability of the software using only the observed failure times in the last 10 iterations. The corresponding data is provided in Table 4.5.

Two popular models for software failure data are the Jelinski and Moranda (1972) model and Moranda (1975) model which assume the failure time T_k to follow $Exponential(\theta_k)$ distribution with $\theta_k = \lambda(N-k)$ and $\theta_k = exp(\alpha - \beta k)$, respectively. Assuming independence of the T_k's, the parameters (λ, N) and (α, β) can be estimated through the maximum likelihood principle using the respective models. The corresponding 95 % one-sided confidence intervals for θ_{10} can be obtained using parametric bootstrap. Alternatively, an at least 95% upper-bound $\hat{\theta}_{10}^{0.95}$ for θ_{10}, without making any assumption about the nature of decrease in θ_k's can be obtained through Equation (4.7). Moreover, $\tilde{\theta}_n^p$ can be obtained through Equation (4.5) without assuming T_k's are independent. The corresponding conservative lower bounds for θ_{10} are given in the third and fourth rows of Table 4.6. Those against Jelinski and Moranda (first row) and Moranda (second row) are obtained through parametric bootstrap of the respective models.

The simulation study presented in Section 4.3 indicates the 95% confidence bounds obtained through our method are, on an average, larger than those obtained through parametric models. However, the coverage probability of the con-

TABLE 4.6
Comparison of upper-bounds for θ_{10}

	Confidence		
Model	90%	95%	99%
Jelinski and Moranda	0.0072	0.0089	0.0123
Moranda	0.0065	0.0083	0.0141
$\hat{\theta}_n^p$	0.0051	0.0059	0.0077
$\tilde{\theta}_n^p$	0.0052	0.0060	0.0078

fidence bounds obtained through a parametric model may have coverage that is substantially less than the required 95% coverage when the parametric model is incorrect. Hence, a smaller size of the confidence bound from a parametric model need not imply the confidence bound will have the required coverage probability. The upper-bounds of θ_{10} obtained through the two parametric models are smaller than $\hat{\theta}_{10}^p$ and $\tilde{\theta}_{10}^p$, casting doubts on the two parametric models. Inspite of the confidence bounds we propose being larger on an average based on simulation results, the confidence bound proposed by us is actually smaller than those obtained by the parametric methods. This indicates that the reliability of the software could be increasing much faster than postulated by either the Jelinski and Moranda or the Moranda model. In the face of uncertainty regarding the parametric assumptions and small sample size, the proposed conservative bounds may be more acceptable.

4.4.2. Sequential quality improvement plans with bounded decrease in reliability

Continuing with the model of the previous sub-section, the assumption of non-decreasing reliability of subsequent product iterations cannot be justified sometimes and hence the assumption, $\theta_1 \geq \ldots \geq \theta_n$, may not be valid. In such a situation, an assumption such as $\beta\theta_{k-1} \geq \theta_k$, for a known $\beta > 0$ might be appropriate. If $\beta > 1$, then the parameter sequence, $\theta_1 \geq \ldots \geq \theta_n$, would be allowed to increase in a bounded manner. For example, if failure times $T_1, \ldots, T_n$ are independently distributed as exponential variables, but the order restriction of $E(T_1) \leq \ldots \leq E(T_n)$ is hard to justify, one may consider a restriction of the form $E(T_{k-1}) \leq \beta E(T_k)$, with a known $\beta > 1$. Consider the parameter sequence $\gamma_1 = \theta_1, \gamma_2 = \theta_2/\beta, \ldots, \gamma_n = \theta_n/\beta^{n-1}$. Note that the parameter sequence $\gamma_1, \ldots, \gamma_n$ is non-increasing and $\beta^{k-1}T_k \sim exp(\gamma_k)$. Theorem 4.1 can be applied to the sequence $T_1, \beta T_2, \ldots, \beta^{n-1}T_n$ to obtain an at least $100 \times p$% confidence upper-bound $\hat{\gamma}_n^p$ for γ_n. Let t_i be a realization of T_i. The at least $100 \times p$% upper-bound for θ_n can be obtained as $\beta^{n-1}\hat{\gamma}_n^p$. Also, the PAVA based ML can be used to compute the MLE of θ_n as

$$\hat{\theta}_n = \frac{\beta^{n-1}}{max\left\{\beta^{n-1}t_n, \frac{\beta^{n-1}t_n+\beta^{n-2}t_{n-1}}{2}, \ldots, \frac{\beta^{n-1}t_n+\ldots+t_1}{n}\right\}}, \tag{4.8}$$

where t_i is a realization of T_i. Jelinski and Moranda (1972) present the time to failures of a software subsequent to fixing the defect which caused the previous failure. They provide the failure time data for 26 iterations of debugging and testing. As with the previous analysis in Section 4.1, we consider only the last 10 failure times. The data is presented in Table 4.7. Jelinski and Moranda assume that the kth failure time $T_k \sim exp(\theta_k)$ with $\theta_k = \lambda(N-k)$. The linear decrease model and the assumption that the parameters θ_k must decrease with k has often been questioned. For this example, considering the failure times in Table 4.7, the assumption of successive T_k's being stochastically larger does not seem to be right and has been questioned recently by Basu and Ebrahimi (2003). Therefore,

TABLE 4.7
The inter-failure times (days) for the last 10 iterations

Iteration (k)	1	2	3	4	5	6	7	8	9	10
Failure Time (t_k)	3	3	6	1	11	33	7	91	2	1

the assumption of bounded increase of the θ_k's seems more appropriate. Theorem 4.1 can be used to compute a conservative 95% upper-bound for θ_n when we assume $\theta_k \leq \beta\theta_{k-1}$. For the illustration, we consider $\theta_k \leq 2\theta_{k-1}$. The application of Theorem 4.1 results in a conservative 95% upper-bound for θ_{10}, as $\hat{\theta}_{10}^{0.95} = 0.23$, which is surprisingly close to the value of 0.21 (obtained from the graphically presented posterior mean and posterior standard deviation for θ_{10}) reported in Basu and Ebrahimi (2003).

4.4.3. Software Reliability from bug-databases and error loggers

Software bug-databases that record user-reported defects provide an increasingly important source of data for software reliability assessment. A bug-database is often used to track the set of known defects in the software up to a calender time s. Error loggers perform a similar function but the errors are not user-reported, but system-reported. We denote the set of known defects up to calendar time s by $D(s)$, with each defect identified by a unique defect ID. Software updates are often based on fixing the set of known defects in the bug-database. When a defect is reported at time s, the defect is compared with the known set of defects $D(s-)$ to determine whether it is already known and, if it is a new defect, then $D(s)$ is updated with the new defect. Define $p(s)$ to be the probability that a defect reported at time s is not contained in the known set of defects $D(s-)$. In order to minimize the number of software updates, it is important to release an update by fixing the defects in $D(s)$ only if the probability $p(s)$ is less than a certain threshold (i.e., the probability of recording an unknown defect is small). A simple and generic model for $p(s)$ is to assume that $p(s_1) = p(s_2)$ if $D(s_1) = D(s_2)$ and $p(s_2) \leq p(s_1)$ if $D(s_1) \subseteq D(s_2)$. Let $s_1, \ldots, s_n$ be the calender times when the first n "new" defects are observed and let M_k be the random variable denoting the number of defects that are observed between calendar times s_{k-1} and s_k, for $k = 1, \cdots, n$, with $M_0 = 0$. Let $p_k = p(s_k)$, for $k = 1, \ldots, n$. Then, clearly, $p_1 \geq p_2 \geq \ldots \geq p_n$. It may be reasonable to assume that $M_k \sim Geometric(p_k)$, for $k = 1, \ldots, n$, and M_i is independent of M_j for $i \neq j$. Denote m_i to a be realization of M_i. The PAV algorithm can be used to obtain the MLE of p_n, but as argued previously, finding a confidence interval for p_n is difficult. Theorem 4.1 can be used to obtain an at least $100 \times p\%$ upper-bound for p_n as given by

$$\hat{p}_n^p = 1 - exp\left(\frac{log(1 - p^{1/n})}{max(m_1, \ldots, m_n)}\right). \tag{4.9}$$

If the T_k's cannot be assumed to be independent, then Theorem 4.2 gives an at least $100 \times p$ % upper-bound for p_n as

$$\tilde{p}_n^p = 1 - exp\left(\frac{log((1 - p)/n)}{max(m_1, \ldots, m_n)}\right). \tag{4.10}$$

Many operating systems provide software for monitoring system errors as and when they arise using error loggers. After observing the error logs till n distinct errors are reported, it may be necessary to compute the probability that a subsequent error that is logged will be a new error that does not belong to the observed set of errors. For an operating system observed by the authors, the data consisting of

TABLE 4.8
Number of known defects reported from the operating system under use.

Iteration (k)	1	2	3	4	5	6	7	8	9	10
Number of defects (m_k)	7	97	1	9	54	87	5	14	48	49

the number of additional errors observed before the kth new error was observed, for $k = 1, \ldots, 10$, is presented in Table 4.8. Using (4.9), an at least 90%, 95% and 99% upper-bounds for p_{10} are obtained as 0.047, 0.054 and 0.071, respectively. The bounds using (4.10), without assuming independence, are 0.048, 0.055 and 0.072, respectively.

4.5. Concluding remarks

Computing confidence bounds for reliability, with a minimum coverage probability, is important for assessing the risk of failure in a product whose quality has been improved sequentially. A non-parametric model for the increase in reliability may provide a worst-case scenario for failure, which in turn may be used for risk management. As demonstrated in Section 4.4, the proposed methodology can be modified for applications which may allow the reliability to decrease with a subsequent version of the product. The proposed method can be used as an alternative to Bayesian procedures proposed by various authors for considering possibilities of a decrease in reliability after fixing a defect. The generality of Theorems 4.1 and 4.2 can be used for computing one-sided confidence bounds for any parametric family of failure time distributions indexed by a single unknown parameter. As mentioned in Section 4.1, the method makes few assumptions regrading the nature of the reliability improvement and does not make use of asymptotic theory;

hence, it is ideal for analysis of data-sets with few failures such as in sequential quality improvement plans for products/procedures with catastrophic failures.

The upper reliability bound proposed in this chapter is a confidence interval. It is conservative because the $100 \times (1-p)\%$ confidence interval obtained for the reliability parameter through Theorem 4.1 or Theorem 4.2 is guaranteed to have *atleast* $100 \times (1-p)\%$ coverage and not exactly $100 \times (1-p)\%$ coverage. This implies that the actual coverage can be much more than $100 \times (1-p)\%$ which in turn means that the upper-reliability bound can be much larger than an exact confidence bound. This is indeed indicated through results of a simulation study presented in Section 4.3.

Often the failure time distribution, $F(t;\theta)$, is continuous and monotonic in the single index parameter θ. In case $F(t;\theta)$ is decreasing in θ and successive θ_k's are non-decreasing, unlike the conditions in Theorem 4.1, one can re-parametrize $F(t;\theta)$ to satisfy the conditions and apply Theorem 4.1. Also, when the distribution involves more than one parameter, often there is one parameter of interest which keeps changing over different iterations while the others remain unchanged. Theorem 4.1 can be used to compute confidence bounds for the parameter of interest.

When T_k's are independent and identically distributed (i.i.d), one may use likelihood methods to obtain asymptotic confidence bounds. Alternatively, Theorem 4.1 can be used to obtain a conservative upper-bound even for small sample sizes. Moreover, Theorem 4.2 gives such a bound even when T_k's are not independent and can be used when successive software failures are suspected to be dependent. From the proofs of Theorems 4.1 and 4.2, the conservative bound in the IID case is expected to be tighter with more accurate coverage probability and relatively smaller size, possibly because of sharing of more information. Our simulation study (see Tables 4.2 and 4.3), for positively correlated inter-failure times, also indicates this.

Chapter 5

Semi-Parametric Software Reliability Models for Post-Release Data

An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem.

John W. Tukey

5.1. Introduction

Controlled testing of a software product by a team of in-house software engineers is an expensive process which is limited by the number of test runs. To improve the reliability of the software product, after a limited in-house testing procedure, the software product is released to allow its users to voluntarily report defects, if any, using the Internet. These user-reported defects are recorded in specialized databases popularly known as bug-databases. Software updates to fix the reported defects are released on a continuing basis. A number of software products allow defects to be reported by any user on a continuing basis. Beta-testing provides another distinct example of user-driven defect reporting, where the software is released to a select user community for voluntary usage and testing for a limited period of time. Unlike a controlled testing environment, where the software usage along with the discovery and reporting of a defect is strictly monitored, a user-driven defect reporting process is based on voluntary reporting of defects that depends on an uncontrolled usage of the software.

Bug databases containing user-reported defects have become ubiquitous for both commercial and open source software and they are viewed as an important data source for software reliability assessments. In fact, after the release of a software product, bug databases can be the only source of data for assessing the reliability of a software in the field. Traditionally, software reliability models, such as those discussed in Chapter 2, have focused on estimating the reliability of the software based upon data generated from in-house testing where the testing methodology and reporting of defects is strictly controlled. These reliability models cannot be directly applied to data retrieved from bug databases as they contain defects recorded due to uncontrolled software usage and reporting of software defects. The usage rate of a software is typically a function of time and is in general unknown. The reporting pattern also depends on the type of users. The distribution of the severity or type of defects reported by users may be quite different from those detected by a controlled testing procedure. There is a tendency on the part of an average user to discover behavior defects more often as compared to, say, critical security defects. Such a tendency can complicate the development of a software reliability model. The usage of the software is determined by the number of users of the software, the type of users and the frequency of usage across all the users. It is analogous to the notion of operating profile as described in Musa (2005)[pp. 93]. This paper addresses the problem of analyzing data retrieved from a bug-database, where information regarding the software usage across its users is uncontrolled and unavailable.

This chapter addresses the lacuna in the statistical analysis of data retrieved from bug-databases using a novel approach that uses the classification of user-reported software defects into multiple types (based, for example, on their severity). The method formulates and estimates a software reliability model that is non-parametric with respect to the usage rate and takes into account differences in defect reporting rates (DRR's) of different types of defects. A partial likelihood approach is used for model estimation. Based upon the proposed model, reliability metrics are proposed that do not depend upon the usage rate. An additional advantage of the proposed model is that it can be estimated using generalized linear model procedures found in most statistics packages.

An analysis of data retrieved from a bug-database also needs to address another problem: The distribution of the type of defects reported by the users may be different from those reported through a controlled testing procedure. Such a difference may exist because software defects can be classified into multiple types and certain defect types are of significant importance to the software community and require specialized knowledge for their discovery, an example of which is the type of defects related to security loopholes in the software (Musa, 2005)[pp. 198]. There may be a tendency on the part of an average user to discover behavior defects more often as compared to, say, critical security defects. Hence, a very low rate of reporting of security defects in a user-reported bug-database does not necessarily mean that the software product is reliable with respect to security defects. Such tendencies can complicate the development of a software reliability model.

Almost all existing models for software reliability make predictions on a calendar time scale, for example, the mean time to failure (MTTF). Predictions on a calendar time scale are only valid if the assumptions regarding the usage continue to hold in the future. While such an assumption may be appropriate in controlled testing of a software product, this may be hard to justify for a software that is subject to uncontrolled usage. For example, predictions made using data retrieved from the bug-database of a software product, freely available in the public domain, will no longer be valid if the number of users of the software increase (or decrease) substantially in an unknown manner over the next few months. Such unknown changes in usage are common in markets which have several competing software products that offer similar functionality, examples include scripting languages, web-browsers and operating systems. This unknown and time-dependent usage of the software does not allow defining a reliability metric in the time scale. To address this deficiency, we propose reliability metrics which make predictions on the scale of number of defects to be observed and, not on the calendar time scale, thereby delinking the unknown usage from the prediction process. The purpose of such metrics is to measure reliability of the software with respect to a particular defect type using the distribution of number of failures to be observed before a failure of that particular type is observed.

Another equally important objective for a software reliability model is to provide metrics that can compare the reliability of two software versions. For example, the software development manager or the customer may need to decide which among two versions of a software product is more reliable. Predictions on a calendar time scale do not delink the reliability of the software from its unknown usage and hence cannot be used for comparison. For example, if one version of a software product is predicted to have an MTTF of 150 days, while another version has the prediction of 250 days, the conclusion of the latter version being more reliable may not be appropriate in the absence of information on the usage of the two versions. The two versions may be equally reliable and the differences in MTTF may only be due to differences in their usage.

The rate at which defects are reported in a bug-database is determined by two components. The first component, termed as the Defect Reporting Rate (DRR), is a function of the usage and is defined as the intensity of discovering and reporting a defect at time s under the assumption that no defect has been discovered till time s (a kind of baseline reporting rate). Since the usage of the software is unknown, the DRR is assumed to be an arbitrary time varying function. The second component represents the cumulative effects of previous defect discoveries on the rate at which new defects will be discovered. This is modeled as a parametric function which assigns a, possibly decreasing, propensity of reporting new defects as a function of the number of distinct defects already reported. The reliability of a software must be a function of this second component, and not the first one, in order to delink it from the unknown usage.

Almost all existing models for software reliability recognize the need for incorporating the usage of the software through the DRR. For example, the Jelinski and Moranda (1972) model assumes the DRR to be constant with time, which may be reasonable in a controlled testing procedure. The model proposed in this paper considers, for the first time as a special case, a Jelinski-Moranda model with a time varying non-parametric specification of the DRR. The proposed modeling can be extended to other popular software reliability models. A class of software reliability models starting with Jelinski and Moranda (1972) and proceeding through

Schick and Wolverton (1978), Goel and Okumoto (1979), Ohba (1984b), Singpurwalla and Wilson (1994) have used a similar decomposition, but assumed increasingly sophisticated parametric models for the DRR as a function of time. These models may not be universally applicable as the assumed parametric forms may not calibrate to the unknown usage of the software. Tamura and Yamada (2007) considered the analysis of a bug-databases from an open source software by using a parametric stochastic process to model the underlying usage rate. Wang, Wang and Liang (2007) considered nonparametric estimation of the intensity of the defect reporting process using kernel regression. Their model could be more appropriate for user-driven defect discovery when compared to parametric models; however, their model does not consider the effect of previous defect discoveries and hence they could not define appropriate reliability metrics. The model proposed in this paper extends these models by treating the DRR non-parametrically. This extension is crucial for analyzing data from user reported bug-databases as the defects are discovered during uncontrolled and unknown usage of the software.

The proposed method can also consider dependencies between the discovery processes of different types of defects. The method, however, requires that there be at least two types of defects in the software so that the reporting rate of a defect type of primary interest can be considered in relation to that of other types of defects as a measure of reliability. A partial likelihood approach is used to eliminate the arbitrary DRR component and estimate the parametric component representing the effect of previous defect discoveries. We would like to note that the proposed model can be estimated using generalized linear model procedures found in most statistics packages.

Section 5.2 formulates the model and describes the method of partial likelihood for estimating the parameters of interest along with estimation of the reliability metrics. Section 5.3 considers a simulation study to investigate the finite sample properties of the estimators and Section 5.4 considers an analysis of a publicly available bug-databases to illustrate our methodology. Section 5.5 ends with some concluding remarks.

5.2. The Modeling and the Method

Usually a bug-database provides information on each defect that is discovered and reported (See, for example, `http://www.bugzilla.org/docs/4.2/en/html/bug_page.html`). For each new defect, there is a record carrying information on (i) time of reporting (typically the calendar time of when the defect was first reported), (ii) the version of the software in which the defect was found, (iii) priority of releasing a fix to the defect (urgent, high, medium and low), (iv) software components affected by the defect and (v) classification of the defect in one of several types. Most defect classification schemes contain security and crash related defects. From now on we will write reporting of a defect to mean "discovering and reporting" of the defect by a user of the software.

Consider data related to all new defects reported up to a certain calendar time S from the release date of a software. This may be represented as the sequence of tuples $(S_1, Z_1), \ldots, (S_n, Z_n)$, where S_i is the calendar time of reporting the i^{th} new defect reported since the release of the software and Z_i is the type of the defect, for $i = 1, ..., n$, where n is the number of new defects reported till time S. Suppose there are m special types of defects, denoted by $1, ..., m$, and all other types are pooled into one "baseline" type denoted by 0. Then Z_i takes values in $\{0, \cdots, m\}$. Alternatively, such a data set can be represented by the sequence of tuples $(S_1, \mathbf{N}(S_1)) \ldots, (S_n, \mathbf{N}(S_n))$, where the vector $\mathbf{N}(s) = [N_0(s), \ldots, N_m(s)]$ denotes the multivariate counting process with $N_i(s)$ being the number of new defects of type i reported upto and including time s, for $i = 0, \cdots, m$.

The model proposed in this chapter considers the defect discovery and reporting of a software product to be stochastic in nature through the defect reporting rates (DRR's) for the different defect types. While the model considers randomness in the process of defects being discovered and reported, it assumes that the bug-database stores all defect reports received at any point of time and faithfully stores them. If there is any dropping of defect reports, or if defect reports are being deliberately deleted, then there will be missing data in the bugzilla database, in which case the model we propose may provide biased estimates of reliability.

5.2.1. The Model

Existing software reliability models for software failure data usually incorporate the debugging process directly or indirectly into the failure rate. Since we consider reporting of only the new defects, the debugging process can be ignored for the purpose of modeling such data. The modeling may be done through the multivariate counting process $\mathbf{N}(s) = [N_0(s), \ldots, N_m(s)]$. A natural model is to describe each $N_i(s)$ as a self-exciting point process (Snyder and Miller, 1991, pp-287) with intensity function $\lambda_i(s)$, the rate at which the i^{th} type of defects are reported.

The intensity of discovering a new software defect at time s, as explained in the introduction, is a function of the number of defects already reported up to time s along with the type of defect and usage of the software. The self-exciting nature of the process $N_i(s)$ is due to possible dependence of $\lambda_i(s)$ on the history of the multivariate process $\mathbf{N}(s)$. Suppose that the DRR corresponding to each $N_i(s)$, denoted by $\gamma_i(s)$, depends only on the usage of the software and reporting rate of the i^{th} type type of defect. The DRR $\gamma_i(s)$ may be calibrated to model the effect of the history of the process $\mathbf{N}(s)$ on the intensity process $\lambda_i(s)$, as discussed in Section 5.1, through a proportional intensity model as given by

$$\lambda_i(s) \quad = \quad \gamma_i(s) f_i(\mathbf{N}(s-)), \text{ for } i = 0, \cdots, m, \tag{5.1}$$

where $\gamma_i(s)$ is considered to be unknown and arbitrary and $f_i() \geq 0$ with $f_i(\mathbf{0}) = 1$. We intend to model $f_i(\mathbf{N}(s))$ parametrically leading to a semi-parametric model for $\lambda_i(s)$. In particular, if $f_i(\mathbf{N}(s))$ depends on $\mathbf{N}(s)$ only through $N_i(s)$, the component processes $N_i(s)$'s are independent. Note that the set of functions $\{f_i(\mathbf{N}(s)), i = 0, \cdots, m\}$ can be used to obtain reliability metrics that are agnostic to the usage and reporting rates of the defects. For example, the rate at which f_i or $\log(f_i)$ decreases for every additional defect discovery, can be interpreted as a measure of reliability of the software with respect to the i^{th} defect type (See Section 5.2.4). A further simplification can be achieved by assuming that each $\gamma_i(s)$ is proportional to a common rate parameter $\gamma(s)$ across all types of defects. This assumption greatly simplifies the estimation of model parameters apart from enabling inference on the reliability with respect to a given type of defect. Therefore, with $\gamma_0(s) = \gamma(s)$, we may assume

$$\gamma_i(s) = \gamma(s)e^{\alpha_i}, \text{ for } i = 1, \cdots, m. \tag{5.2}$$

The parameters $\alpha_i's$ can be interpreted as the differences in reporting rates (DRR's) of different types of new defects. Noting that $f_i(\mathbf{N}(s))$ represents multiplicative changes in $\lambda_i(s)$ due to reporting of different types of new defects, it may be natural to assume that $f_i(\mathbf{N}(s))$ decreases with each component of $\mathbf{N}(s)$. However, because of the user-reported nature of new defects, the nature of this decrease is not clear. There may be several choices for $f_i(\mathbf{N}(s))$. In this context, one may recall the Jelinski and Moranda (1972) model for software testing data, which postulated a linear decrease in the intensity of reporting a new defect with the number of already detected defects, and consider a linear decrease model as given by

$$f_i(\mathbf{N}(s)) = max(0, 1 - \sum_{j=0}^{m} \beta_{ji} N_j(s)) \tag{5.3}$$

.

Alternatively, a non-linear decrease model in the spirit of the logarithmic Poisson model Musa and Okumoto (1984) as given by

$$f_i(\mathbf{N}(s)) = \exp\left(-\sum_{j=0}^{m} \beta_{ji} N_j(s)\right) \tag{5.4}$$

may also be considered. In the special case, when the component processes $N_i(S)$'s are independent, it would be appropriate to consider $f_i(\mathbf{N}(s)) = exp(-\beta_i N_i(s))$ with a scalar β_i, for $i = 0, \cdots, m$. The two choices in (**??**) and (5.4) result in

$$\lambda_i(s) = \gamma(s)\exp(\alpha_i)\left[1 - \sum_{j=0}^{m} \beta_{ji} N_j(s-)\right]$$

and

$$\lambda_i(s) = \gamma(s)\exp\left(\alpha_i - \sum_{j=0}^{m} \beta_{ji} N_j(s-)\right), \tag{5.5}$$

respectively, for $i = 0, 1, 2 \ldots, m$, with $\alpha_0 = 0$. Write $\alpha = \{\alpha_1, \cdots, \alpha_m\}$, $\beta = \{\beta_0, \cdots, \beta_m\}$ with $\beta_i^s = [\beta_{0i}, \cdots, \beta_{mi}]$ and $\gamma(.)$ as the infinite dimensional intensity $\gamma(s)$ to determine the set of unknown parameters of the model. The logarithmic Poisson model has been noted as an extensively applied software reliability model (Farr, 1996) and shown to provide accurate predictions for large software systems (Jones, 1991; Derrennic and Le Gall, 1995), leading us to use (5.5) as the underlying model of choice for illustration of the method developed in this article.

5.2.2. Partial Likelihood Estimation

Let us write $S^{(j)} = (S_j, S_{j-1}, \ldots, S_1)$ and $Z^{(j)} = (Z_j, Z_{j-1}, \ldots, Z_1)$. The likelihood of the data sequence $(S_1, Z_1), \ldots, (S_n, Z_n)$ is proportional to

$$\prod_{j=1}^{n} P_{S_j|S^{(j-1)},Z^{(j-1)}}(.|\alpha,\beta,\gamma(.)) \times \prod_{j=1}^{n} P_{Z_j|S^{(j)},Z^{(j-1)}}(.|\alpha,\beta). \tag{5.6}$$

The second product in (5.6), under the modeling assumptions (5.1) and (5.2), does not depend on $\gamma(s)$, and is the partial likelihood for estimating α and β (Cox, 1975). Note that the conditional probability $P_{Z_j|S^{(j)},Z^{(j-1)}}(.|\alpha,\beta)$ in the second product can be derived as the following multinomial probability under assumptions (5.1) and (5.2):

$$P(Z_j = i | S^{(j)}, Z^{(j-1)}) = \frac{e^{\alpha_i} f_i(\mathbf{N}(S_j-))}{\sum_{k=0}^{m} e^{\alpha_k} f_k(\mathbf{N}(S_j-))}, \tag{5.7}$$

for $i = 0, 1, \ldots, m$. In the special case given by (5.4), this partial likelihood simplifies to the likelihood of the multinomial logistic regression model as given by,

$$\prod_{j=1}^{n} P(Z_j | S^{(j)}, Z^{(j-1)}) = \prod_{j=1}^{n} \prod_{i=0}^{m} \left(\frac{e^{\alpha_i - \beta_i' \mathbf{N}(S_j-)}}{\sum_{k=0}^{m} e^{\alpha_k - \beta_k' \mathbf{N}(S_j-)}} \right)^{I(Z_j = i)} . \tag{5.8}$$

Maximizing the partial likelihood to estimate α and β can now be performed through a multinomial logistic regression between Z_j and the vector $\mathbf{N}(S_j-)$. A Newton-Raphson procedure can be used to maximize the log partial-likelihood in order to estimate the parameters. Alternatively, any standard software package to analyze a multinomial logistic regression model can be used. For example, one may use the *multinom* function in *nnet* package of the R software package (Venables and Ripley, 2002). The estimated asymptotic variance-covariance matrix of the parameter estimates can be obtained from the observed information matrix based on the partial likelihood. The asymptotic normality of the partial likelihood estimates (Wong, 1986) can be used to perform tests of significance and obtain confidence intervals for the parameters of interest and functions thereof.

5.2.3. Goodness of Fit

The proposed modeling consists of the forms for the set of functions $\{f_i(\mathbf{N}(s)), i = 0, \ldots, m\}$ as defined in (5.1) and the proportionality of the baseline rates $\gamma_i(s)$'s as defined in (5.2). Violations of these assumptions would lead to $P\left(Z_j | S^{(j)}, Z^{j-1}\right)$ being incorrectly specified. Hence, it suffices to check the goodness of fit for the form of $P\left(Z_j | S^{(j)}, Z^{j-1}\right)$ for different j. In the special case when the model is being fit with only two types of defects (i.e., Z_j takes binary values) and the choice of $f_0(\mathbf{N}(s))$ and $f_1(\mathbf{N}(s))$ is the logarithmic Poisson form as given in (5.4) leading to (5.8), the Hosmer and Lemeshow (1980) test can be used to assess the goodness of fit of $P\left(Z_j|S^{(j)},Z^{j-1}\right)$. For data sets with more than two types of defects, with the logarithmic Poisson form for $\{f_i(\mathbf{N}(s)), i=0,\ldots,m\}$, we would need to assess the goodness of fit of a multinomial logistic regression model. For this, we appeal to tests proposed by Begg and Gray (1984), Pigeon and Heyse (1999), or Fagerland, Hosmer and Bofin (2008).

For a graphical check, one may consider comparing $\sum_{j=1}^{k} P\left(Z_j = i|S^{(j)},Z^{(j-1)}\right)$ with the observed number of defects of type i up to time S_k, given by $N_i(S_k)$. A plot of $\sum_{j=1}^{k} \hat{P}\left(Z_j = i|S^{(j)},Z^{(j-1)}\right)$ (as predicted) and $N_i(S_k)$ (as observed) over k can be used to visually check the goodness of fit, where $\hat{P}(\cdots|\cdots)$ denotes the estimate of the corresponding $\hat{P}(\cdots|\cdots)$, evaluated at the parameter estimates $\hat{\alpha}$ and $\hat{\beta}$.

The integrated intensity function $\Gamma(s) = \int_0^s \gamma(u)du$ (See Section 5.2.1) may be estimated using a Breslow type estimator (Breslow, 1972), as given by, for the logarithmic Poisson model,

$$\hat{\Gamma}(s) = \int_0^s \frac{dN_.(u)}{\sum_{i=0}^{m} \exp(\hat{\alpha}_i - \hat{\beta}_i N_i(u-))} du, \quad 0 < s \leq S, \tag{5.9}$$

where $N_.(s) = \sum_{i=0}^{m} N_i(s)$. This type of estimator may also be used for testing the proportionality assumption (5.2). For example, the individual integrated intensity $\Gamma_i(s) = \int_0^s \gamma_i(u)du$ may be estimated, without the assumption (5.2), by

$$\hat{\Gamma}_i(s) = \int_0^s \frac{dN_i(u)}{\exp(-\hat{\beta}_i N_i(u-))} du, \quad 0 < s \leq S, \tag{5.10}$$

for $i = 0,\ldots,m$. Plots of $log(\hat{\Gamma}_i(s))$ for different i on the same graph should be near parallel if (5.2) is true.

5.2.4. Reliability Metrics

As mentioned earlier, certain defect types are of more importance to the software community than others, an example being defects which are related to security loopholes or crashes in the software. A traditional metric for the reliability of a software with respect to, say, crash defects would be the probability of observing no crash defects in the subsequent year, or alternatively the mean time to observing a crash defect. These metrics depend upon the usage rate of the software. For example, if the usage of the software were to increase manifold, these metrics would no longer be correct. Hence, there is a need for a metric that does not depend upon future usage rate of the software. An intuitive metric that does not depend upon the usage rate is the probability of discovering no crash related defect (corresponding to, say, type m) in the next N defects of any type. Let us denote this by $R(N)$. This metric, under the modeling of Section 5.2.1, would not depend upon the usage of the software. As a result, the time required for the N defects to be reported will not be known (since the usage is unknown). One may have a rough guess of this time by assuming a particular pattern of reporting. Similarly, we can also think of the number of defects to be observed before observing a new crash related defect and call it Mean Number of Defects to Failure ($MNDF$) as an alternative to mean time to failure ($MTTF$).

Theoretical derivations of these reliability metrics in general can be a challenge because of the stochastic nature of the model (5.1). For example, the probability of no crash related defect requires a huge sum of m^N joint probability terms each corresponding to an ordered set of N defects of types other than crash related defect. This may be calculated when N is small; however, even for moderate N (say, 10 or 15) and with $m = 2$ or 3, this is computationally difficult. One option is to pool all other defect types into one type resulting in only two types (with $m = 1$) so that there is only one joint probability term as given by

$$R(N) = \prod_{j=n+1}^{n+N} P\left(Z_j = 0 | S^{(j)}, Z^{(j-1)}; \alpha, \beta\right). \tag{5.11}$$

Note that $Z^{(j-1)}$ in the conditioning event in each term of (5.11) is given by $(Z_1, \ldots, Z_n, Z_{n+1} = 0, \ldots, Z_{j-1} = 0)$. Also, each such term is independent of $S^{(j)}$, except through $\mathbf{N}(S_j)$ as in (5.7). In this case, the other reliability measure MNDF can be calculated as the infinite sum

$$MNDF = \sum_{l=n+1}^{\infty} \prod_{j=n+1}^{l} P\left(Z_j = 0 | S^{(j)}, Z^{(j-1)}; \alpha, \beta\right). \tag{5.12}$$

For the independent model with $f_i(\mathbf{N}(s)) = exp(\alpha_i - \beta_i N_i(s))$, for $i = 0, 1$, with $\alpha_0 = 0$, the individual probability term in (5.11) and (5.12) is given by

$$P(Z_j = 0 | S^{(j)}, Z^{(j-1)}, \alpha, \beta) = \frac{e^{-\beta_0(n_0+j-1)}}{e^{-\beta_0(n_0+j-1)} + e^{\alpha_1 - \beta_1 n_1}}, \tag{5.13}$$

for $j = n+1, n+2, \ldots$, where $n_i = N_i(S_n)$, for $i = 0, 1$. Since these probability terms are decreasing in j, the MNDF in (5.12) has a finite value. In the special case, when $f_0(\mathbf{N}(s)) = 1$, or $\beta_0 = 0$, meaning that there is no effect of history on $\lambda_0(s)$, we have $MNDF = exp(-\alpha_1 + \beta_1 n_1)$. In this special case, $R(N)$ simplifies to $[1 + exp(\alpha_1 - \beta n_1)]^{-N}$. The metrics in (5.11) and (5.12) are to be estimated by evaluating them at the partial likelihood estimates of α, β. The standard errors of these estimates can be obtained by applying the delta method.

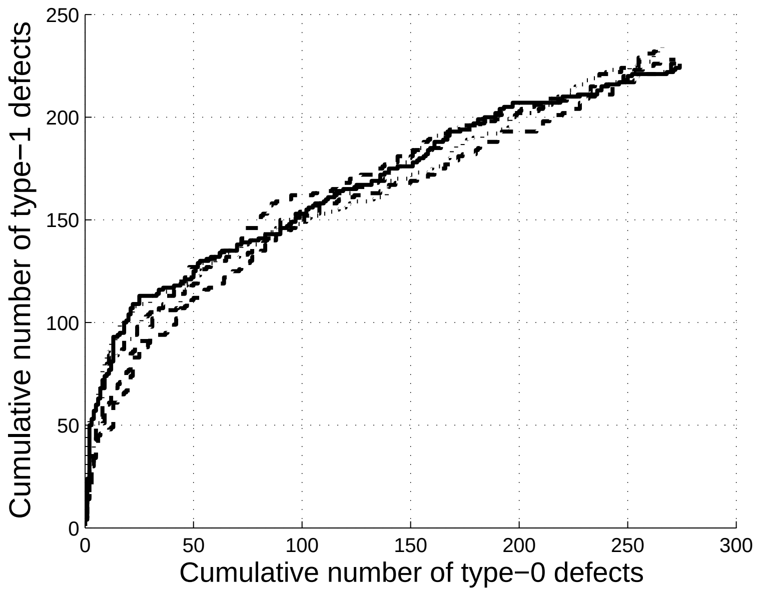
In general, both $R(N)$ and $MNDF$ can be estimated by simulating Z_{n+i}'s successively using (5.7) and the estimates of the model parameters. For example, the reliability metric, $R(N)$ can be estimated by the proportion of times none of $Z_{n+1},\ldots,Z_{n+N}$ equals m over, say, $M = 1000$ simulations of $Z_{n+1},\ldots,Z_{n+N}$. Similarly, for the reliability metric MNDF, one needs to simulate Z_{n+i}'s till $Z_{n+L+1} = m$, where L is the minimum number of detected defects before observing a type m defect in the simulation. Then, MNDF is estimated by the mean of the values of L over the M simulations. If a simulation approach were to be used for estimating the metrics, standard errors can be obtained by using a parametric bootstrap method (Efron and Tibshirani, 1986) as follows: simulate the bth bootstrap sample comprising of the defect types $(Z_j^{(b)}, j = 1,2,\cdots,n)$ using the estimates of α and β, and (5.7). Based on this bootstrap sample, obtain estimates $\hat{\alpha}^{(b)}$ and $\hat{\beta}^{(b)}$ of α and β, respectively, using the method of Section 5.2.2. Estimate the reliability metrics using $\hat{\alpha}^{(b)}$ and $\hat{\beta}^{(b)}$ by the method of simulation as described above and denote them by $R(N)^{(b)}$ and $MNDF^{(b)}$, respectively. Repeat the bootstrap process for, say, $B = 500$ times. The standard deviations of the estimated reliability metrics $R(N)^{(b)}$ and $MNDF^{(b)}$ over the B bootstrap samples estimate their corresponding standard errors.

5.3. A Simulation Study

The simulation study presented here consists of three parts. In the first part, we study the asymptotic properties of the estimator. In the second part, we compare the proposed model to software reliability models that assume a parametric form for the underlying DRR. In the third part of the study the effects of model mis-specification is studied through a limited set of simulations. While planning a simulation study, note that, for the purpose of estimating the model parameters, information on only the defect types $Z_1, \cdots, Z_n$ is needed and the same on the reporting times $S_1, \cdots, S_n$ may be ignored. Also, while estimating the reliability metrics $R(N)$ and $MNDF$ by simulation, it is enough to simulate only the future defect types $Z_{n+1}, Z_{n+2}, \ldots$. Note that simulation of Z_i's can be successively carried out using (5.7). In our simulation study, we consider two types of defects (i.e., $m = 1$) and type 1 is assumed to correspond to an important defect type such as a crash related defect. We consider the independent logarithmic Poisson model given by $f_i(\mathbf{N}(s)) = exp(-\beta_i N_i(s))$, for $i = 0, 1$, with $\beta_0 = 0.03$, $\beta_1 = 0.01$ and the type-specific differential rate parameter $\alpha_1 = 3$. For each simulation, $Z_1, \ldots, Z_n$ are generated using (5.7) with $n = 500$ and 2000 to reflect moderate to large sample sizes. We visually present five simulation histories with $n = 500$ in Figure 5.1, where the cumulative count of type 1 defects is plotted against the cumulative count of type 0 defects for each simulation history.

The maximum partial likelihood estimates of α_1, β_0 and β_1 are then obtained by maximizing (5.8) along with the corresponding standard errors and the asymptotic 95% confidence intervals of the parameters using a normal approximation for the distributions of their estimates (See Wong 1986). Reliability metrics $R(N)$ (with $N = 10$) and $MNDF$ are also estimated using the simplified forms in (5.11) and (5.12), respectively, with each probability term given by (5.13), along with their standard errors and asymptotic 95% confidence intervals. This simulation is repeated 1000 times and mean and standard deviation of the estimates over the 1000 simulations are obtained. The coverage probabilities of the asymptotic 95%

FIG 5.1. *Sample simulations*



confidence intervals are also estimated. Results of this simulation study, for five sets of parameters, are presented in Table 5.1. In the first set, by setting the parameter $\alpha = 0$, we consider the case when there is no difference in the reporting rate of the two types of defects. In the next four parameter sets, we set $\alpha = 3$ and $\alpha = 10$, which correspond to the reporting rates of the first defect type being $exp(3) \approx 20$ times and $exp(10) \approx 22000$ times the reporting rate of the second defect type. Also, in the same simulations, we study the effect of small and large difference in the β parameters of the two types of defects. As the average of the standard errors over 1000 simulations is close to the standard deviation of the corresponding estimates over the 1000 simulations, we report only the latter quantity for the measure of standard error. The true values corresponding to $R(N)$ and $MNDF$ are computed from (5.11) and (5.12) with (5.13) evaluated at the true parameter values, given the simulated $Z^{(n)}$ giving values of n_0 and n_1. The corresponding values in Table 5.1 are averages of those over the 1000 simulations. Note that $R(N)$ and $MNDF$ depend on n, the number of defects discovered, and the history of the n defects discovered including the relative accumulation of the different types of defects. This explains the difference in the true values for $R(N)$ and $MNDF$ for $n = 500$ and $n = 2000$. The results indicate consistency of the estimates and the standard errors (SE) decrease with n, as expected. Also, the estimated coverage probability is closer to 95% for larger n.

The second part of the simulation study compares the proposed model with software reliability models that assume a parametric form for the underlying DRR. For this purpose, we simulate data from four different parametric models for the underlying DRR $\gamma(s)$ by assuming $\gamma(s) = c$ and $\gamma(s) \propto \sqrt{s}, s$ and s^2. These four models correspond to a constant, concave, linear and convex DRR's as functions of time. We once again use the logarithmic Poisson Model with $f_i(\mathbf{N}(s)) = exp(-\beta_i N_i(s))$, for $i = 0, 1$ corresponding to two types of defects. For the purpose of the study, we consider $\beta_0 = 0.3$, $\beta_1 = 0.1$ and $\alpha = 3$. $n = 500$ defects were simulated in each simulation. The coverage probability and the average half-width of the 95% Confidence Interval (CI) are presented using 1000 simulations.

The first panel in Table 5.2 presents the comparisons of the estimates of β_0 obtained through parametric models for DRR to those obtained through the proposed model. The second panel in Table 5.2 presents the same for β_1. Coverage probability (%) for the estimated 95% CI for β_0 is reported. The value in () represents 1000 times the average width of the CI. The simulation results presented

TABLE 5.1
Empirical evaluation of the asymptotic properties of the estimates

	n=500				n=2000			
Parameter	True	Estimate	SE×10	Coverage	True	Estimate	SE×10	Coverage
α	0.00	0.29	3.80	89.2%	0.00	0.14	2.53	93.2%
β_0	0.03	0.05	0.23	86.2%	0.03	0.03	0.08	92.1%
β_1	0.01	0.02	0.10	86.4%	0.01	0.01	0.03	91.8%
R(N)	0.05	0.04	0.20	80.2%	0.05	0.05	0.16	95.6%
MNDF	2.86	2.64	4.59	88.0%	2.95	2.88	3.50	95.4%
α	3.00	3.56	7.40	91.4%	3.00	3.22	4.12	93.1%
β_0	0.03	0.06	0.37	88.2%	0.03	0.03	0.08	92.1%
β_1	0.01	0.02	0.08	88.3%	0.01	0.01	0.02	92.2%
R(N)	0.61	0.82	0.38	95.0%	0.05	0.05	0.16	95.6%
MNDF	3.16	3.51	7.20	96.5%	2.91	2.95	3.50	95.4%
α	3.00	2.83	9.33	81.0%	3.00	2.92	3.47	90.2%
β_0	0.30	0.77	3.93	76.6%	0.30	0.40	1.31	88.2%
β_1	0.01	0.02	0.14	77.7%	0.01	0.01	0.04	88.4%
R(N)	0.70	0.67	1.49	77.6%	0.05	0.05	0.97	94.1%
MNDF	25.38	21.92	97.15	90.3%	2.91	2.95	80.84	89.5%
α	10.00	11.40	25.91	95.3%	10.00	10.28	10.06	96.1%
β_0	0.03	0.03	0.08	94.4%	0.03	0.03	0.03	95.4%
β_1	0.01	0.01	0.11	93.5%	0.01	0.01	0.01	95.0%
R(N)	0.01	0.01	0.18	74.6%	0.05	0.05	0.13	93.2%
MNDF	1.93	1.85	5.65	87.60%	2.92	2.89	3.04	94.8%
α	10.00	11.67	29.30	94.0%	10.00	10.77	18.6	95.0%
β_0	0.30	0.34	0.91	94.1%	0.30	0.32	0.05	94.3%
β_1	0.01	0.01	0.1	93.5%	0.01	0.01	0.01	94.1%
R(N)	0.71	0.69	0.81	94.1%	0.71	0.71	0.08	95.0%
MNDF	26.26	23.20	60.08	87.10%	26.03	25.20	59.04	93.1%

TABLE 5.2
Comparisons of estimates of β_0 *and* β_1 *obtained using parametric models for DRR to those obtained through the proposed model in terms of coverage percentage and 1000 times the average width of the CIs (in parentheses).*

		Fitted Model				Proposed Model
	True Model	$\gamma(s) = c$	$\gamma(s) \propto \sqrt{s}$	$\gamma(s) \propto s$	$\gamma(s) \propto s^2$	$\gamma(s)$ arbitrary
β_0	$\gamma(s) = c$	94.4 (0.62)	0.0	0.0	0.0	92.7 (43.4)
	$\gamma(s) \propto \sqrt{s}$	0.0	95.7 (0.62)	0.0	0.0	90.6 (44.2)
	$\gamma(s) \propto s$	0.0	0.0	95.7 (0.62)	0.0	91.7 (43.8)
	$\gamma(s) \propto s^2$	0.0	0.0	0.0	94.1 (0.62)	91.9 (43.6)
β_1	$\gamma(s) = c$	95.8 (0.31)	0.0	0.0	0.0	94.9 (15.6)
	$\gamma(s) \propto \sqrt{s}$	0.0	94.8 (0.31)	0.0	0.0	93.5 (15.7)
	$\gamma(s) \propto s$	0.0	0.0	94.9 (0.31)	0.0	94.0 (15.6)
	$\gamma(s) \propto s^2$	0.0	0.0	0.0	95.7 (0.31)	94.2 (15.6)

in the two panels of Table 5.2 indicate that the parametric model for the DRR, when correct, results in accurate estimation of the β parameters. However, when the underlying model is incorrect, the estimates are inaccurate, as seen in coverage probability being equal to 0 in the estimated confidence intervals. On the other hand, the proposed model estimates confidence intervals which contain the true parameter with nearly the required confidence. However, from the first panel of Table 5.2, the proposed model has a much higher (nearly 70 times) standard error for β_0 when compared to the estimates from the correct parametric model, as expected.

The third part studies the effect of model mis-specification for $f_i(\mathbf{N}(s))$ on the estimation of the reliability metrics, we consider simulation of $Z^{(n)}$ from the independent linear decrease model with two types of defects, given by $f_i(\mathbf{N}(s)) = 1 - \beta_i N_i(s)$, for $i = 0, 1$. We assume $\beta_0 = 0.001$, $\beta_1 = 0.00005$ and $\alpha_1 = 0.5$ for the purpose of simulation. This parameter setting implies that there are $1/\beta_0 = 1000$

defects of type 0 and $1/\beta_1 = 20000$ defects of type 1 in the beginning. For each simulation of $Z^{(n)}$ with $n = 1000$, estimation of the model parameters α_1, β_0 and β_1 is carried out by fitting the incorrectly specified independent logarithmic Poisson model. Reliability metrics $R(N)$ with $N = 10$ and $MNDF$ are estimated by using (5.11) and (5.12), respectively, with each probability term given by (5.13), as before. For the purpose of comparison, we also compute (5.11) and (5.12) but with each probability term given by the correct model and using the true parameter values and simulated $Z^{(n)}$. The averages of these two sets of estimates of $R(N)$ and $MNDF$, over 1000 simulations, are compared to understand the effect of model mis-specification. To assess the severity of the mis-specification, the Hosmer-Lemeshow goodness of fit test (described in Section 5.2.3) was used to determine whether the logarithmic Poisson model being fit was appropriate or not. It was determined that the test, when used at 5% level, rejected the null hypothesis of the logarithmic Poisson model being appropriate, 34.0% of the 1000 realizations. This indicates that the deviation considered is fairly large. The estimates for $R(N)$ and $MNDF$ obtained by using the correct model are 0.11 and 4.17, respectively, the corresponding estimates obtained by using the incorrect model are 0.10 and 3.85, with standard errors 0.03 and 0.61, respectively. Therefore, in this limited study, the effect of model mis-specification on the estimates of $R(N)$ and $MNDF$ seems to be minimal.

5.4. Analysis of Python Software

Python is a general purpose scripting language that is extensively used in a variety of applications. It is an open source software which is maintained and developed by its community. The Python project maintains a bug-database that records defects in the software reported by the Python user community. When a defect is reported by the user community, it is classified and tagged with auxiliary information. Every defect reported by the user community is classified into one of several types. Defects of type “crash” or “security” are of significant importance. We shall analyze the corresponding bug-database to estimate the reliability metrics for these two types of defects.

Python’s bug-database provides a method for querying its database (http://bugs.python.org/issue?@template=search). Only those defects whose resolution was “fixed” or “fixed and accepted” as on 31 January 2012 are retrieved. This is because defects with other resolutions comprise of duplicates of already reported defects, or defects that have not been confirmed as genuine. In addition to the date of first reporting of each defect, information on the defect type is also extracted. Python version 2.7 has 2273 reported defects, while Python version 2.6 has 1975 defects reported until 31st January 2012, a summary of which with defect types is given in Table 5.3. We would like to mention that Python 2.6 is built on Python 2.1 and hence shares much of its code base with Python 2.1 which was released in 2001. Hence, a defect in Python 2.1 will be tagged as having occurred in Python 2.6 if the defect occurred in a module which is still a part of Python 2.6. When we queried Python’s bug-database to retrieve all defects of Python 2.6 since 31 January 2012, it also returned defects from Python 2.1 that occurred in modules which are still a part of Python 2.6. Hence, the date of discovery of the first defect in Python 2.6 was in 2001. The same observation holds for Python 2.7.

We first analyze the data using all three types of defects, namely, crash defects,

TABLE 5.3
Summary of defects in the two versions of Python

	Python 2.7		Python 2.6	
Defect Type	Count	Percentage	Count	Percentage
Crash	130	5.7 %	124	6.3 %
Security	19	0.8%	16	0.8%
Others	2124	93.5 %	1835	92.9%
Total defects	2273	100%	1975	100 %

security defects and defects of other types. Let us denote these defect types as type 2, type 1 and type 0, respectively. We use the independent logarithmic Poisson model given by $f_i(\mathbf{N}(s)) = exp(\alpha_i - \beta_i N_i(s))$, for $i = 0,1,2$, with $\alpha_0 = 0$. For the analysis of data from each version using the method of Section 5.2.2, we only need the vector $Z^{(n)} = (Z_1,\ldots,Z_n)$ with $n = 2273$ and 1975 for versions 2.7 and 2.6, respectively, where each Z_i takes values 0,1 or 2. The parameters $\alpha_1,\alpha_2,\beta_0,\beta_1,\beta_2$ are estimated along with their variance-covariance matrix, by performing a multinomial logistic regression between Z_j and n_{0j},n_{1j},n_{2j}, where $n_{ij} = N_i(S_j-) = \sum_{l=1}^{j-1} I(Z_l = i)$ is the number of type i defects reported till prior to time S_j, for $i = 0,1,2$. These parameter estimates are used to estimate reliability metrics $R(N)$ with $N = 10$ and $MNDF$ using simulation, as described in the end of Section 5.2.4. The corresponding standard errors are obtained by using the parametric bootstrap method, as described therein. The estimates of the parameters and the reliability metrics, for both crash and security related defects, along with their standard errors are presented in the top panel of Table 5.4.

From the estimates of $R(N)$ and $MNDF$, Python 2.6 seems more reliable with respect to “crashes” when compared to Python 2.7. However, by looking at the

TABLE 5.4

Estimates of the parameters and reliability metrics with their standard errors.

		Python 2.7		Python 2.6	
Analysis	Parameters	Estimate	SE	Estimate	SE
Three-type	α_1	-5.60	0.49	-5.88	0.53
	α_2	-3.87	0.32	-3.36	0.26
	$\beta_0(\times 10^2)$	0.56	0.15	0.25	0.11
	β_1	0.64	0.18	0.22	0.15
	β_2	0.08	0.02	0.03	0.01
	$R(N=15)$(Crash)	0.43	0.10	0.49	0.13
	$MNDF$(Crash)	12.02	2.67	15.06	5.50
	$R(N=15)$(Security)	0.98	0.02	0.90	0.10
	$MNDF$(Security)	55.91	17.67	55.15	20.28
Two-type	α_1	-3.85	0.37	-3.56	0.33
	$\beta_0(\times 10^2)$	0.55	0.18	0.35	0.15
	β_1	0.08	0.02	0.04	0.02
	$R(N)$ (Crash)	0.44	0.07	0.47	0.07
	$MNDF$ (Crash)	11.23	2.09	12.28	2.26

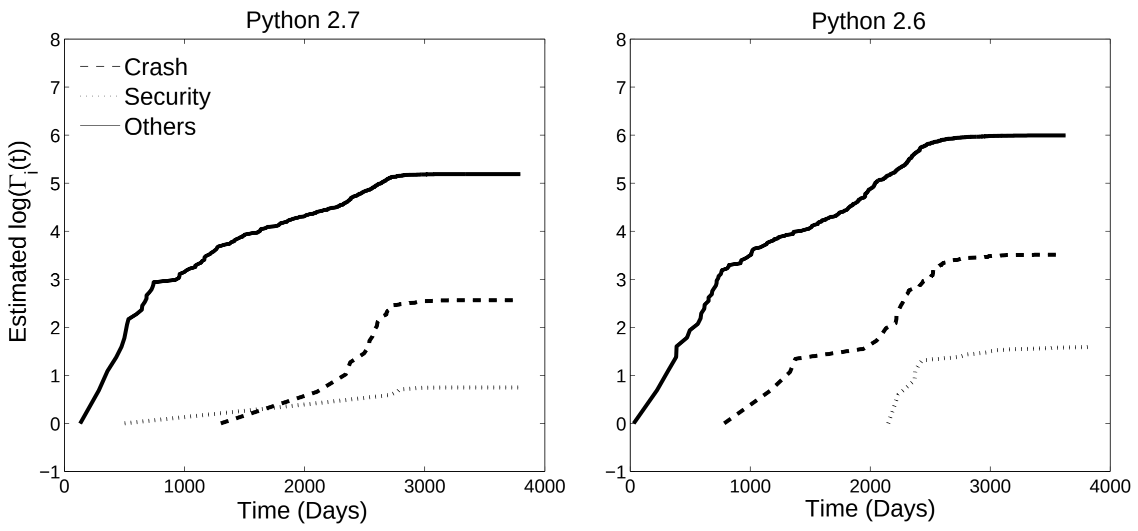
standard errors, we cannot conclude that the reliabilities of the two versions of the software are significantly different. Note that the coefficients $\beta_0, \beta_1, \beta_2$ are all statistically significant except for β_1 in the analysis of Python 2.6 data. The β coefficients are positive, which implies that the propensity of discovering a defect of the i^{th} type ($\lambda_i(s)$) decreases with the discovery of every additional defect of the i^{th} type, since we assumed $\lambda_i(s) = \gamma(s)exp(\alpha_i - \beta_i N_i(s))$. Also, in the analysis of both versions of the software, the parameters α_1 and α_2, which correspond to differences in the reporting rates of security and crash related defects, respectively, when compared to other defect types, are significantly different with $\alpha_2 > \alpha_1$. This indicates that security related defects are less likely to be reported when compared to crash related defects and, surprisingly, to other types of defects as well. Note that the parameters α_1 and α_2 are not significantly different across the two versions of the software, while they themselves are significant.

The goodness of fit for a multinomial logistic regression model, as suggested by Fagerland, Hosmer and Bofin (2008), is carried out to assess the fit of the form of $P\left(Z_j|S^{(j)},Z^{(j-1)}\right)$, as derived from the assumed independent logarithmic Poisson model. The test is conducted by quantizing the estimated $P\left(Z_j=2|S^{(j)},Z^{(j-1)}\right)$, for $j=1,2,\ldots,n$, into 10 levels using its deciles so that there are roughly the same number of values assigned to each quantized level. The quantized values of $P\left(Z_j=2|S^{(j)},Z^{(j-1)}\right)$ are used as the grouping variable for performing the goodness of fit test, as suggested by Fagerland, Hosmer and Bofin (2008), of the model with all the categories of the dependent variable taken together. The corresponding test statistic has an asymptotic chi-square distribution with 16 degrees of freedom. The corresponding p-value for Python 2.7 is 0.08, while the same for Python 2.6 is 0.02. This indicates that the proposed three-type defect model fits reasonably well for Python 2.7 while the fit for Python 2.6 may be questionable.

There is an apparent lack of fit of the three-type model to the data from Python 2.6 due to the non-significance of β_1 corresponding to security related defects in the three-type analysis. This lack of fit and the simplicity of a two-type model for the calculation of reliability metrics (see Section 5.2.4) lead us to pool security related defects with other types, and consider a two-type model involving just crash related defects ($i=1$) and other defects ($i=0$). The independent logarithmic Poisson model given by $f_i(\mathbf{N}(s))=exp(\alpha_i-\beta_i N_i(s))$, for $i=0,1$, with $\alpha_0=0$, is once again used. As in the three-type analysis, we only need the vector $Z^{(n)} = (Z_1, \ldots, Z_n)$, where each Z_i takes values 0 or 1. The parameters α_1, β_0 and β_1 are estimated along with their variance-covariance matrix, by performing a logistic regression between Z_j and n_{0j}, n_{1j}, where n_{ij}'s are similar to those defined before but with the new labeling of type ($i = 0, 1$). The estimates of the parameters and the reliability metrics $R(N)$, with $N = 10$, and $MNDF$, along with their standard errors, are presented in the bottom panel of Table 5.4. In order to assess the fit of the form of $P\left(Z_j | S^{(j)}, Z^{(j-1)}\right)$, the Hosmer-Lemeshow test for logistic regression is performed. The test is conducted by splitting the estimated $P\left(Z_j = 1 | S^{(j)}, Z^{(j-1)}\right)$ into ten probability deciles, which leads to an asymptotic chi-square distribution for the test statistic with 8 degrees of freedom. The corresponding p-values for Python 2.7 and 2.6 are 0.45 and 0.55, respectively. This indicates a good fit of the model to the data. To visually assess the fit, plots of $\sum_{j=1}^{k} \hat{P}\left(Z_j = 1 | S^{(j)}, Z^{(j-1)}\right)$ and n_{1k} over k are considered and shown in the bottom most panel of Figure 5.3. The plots indicate a reasonably good fit of the model for both the versions. Comparison of the reliability metrics for crash related defects across the two versions, along with their standard errors, indicates that these are not significantly different across the two versions. It is interesting to note that the estimated reliability metrics for crash related defects are quite similar to those obtained from the three-type analysis presented earlier. The parameters β_0 and β_1 turn out to be statistically significant and positive for both versions of the software, as before. The estimate of β_1 is similar to the corresponding estimate (i.e., of β_2) in the three-type analysis. The estimate of α_1 is also similar to the corresponding estimate (i.e., of α_2) in the three-type analysis.

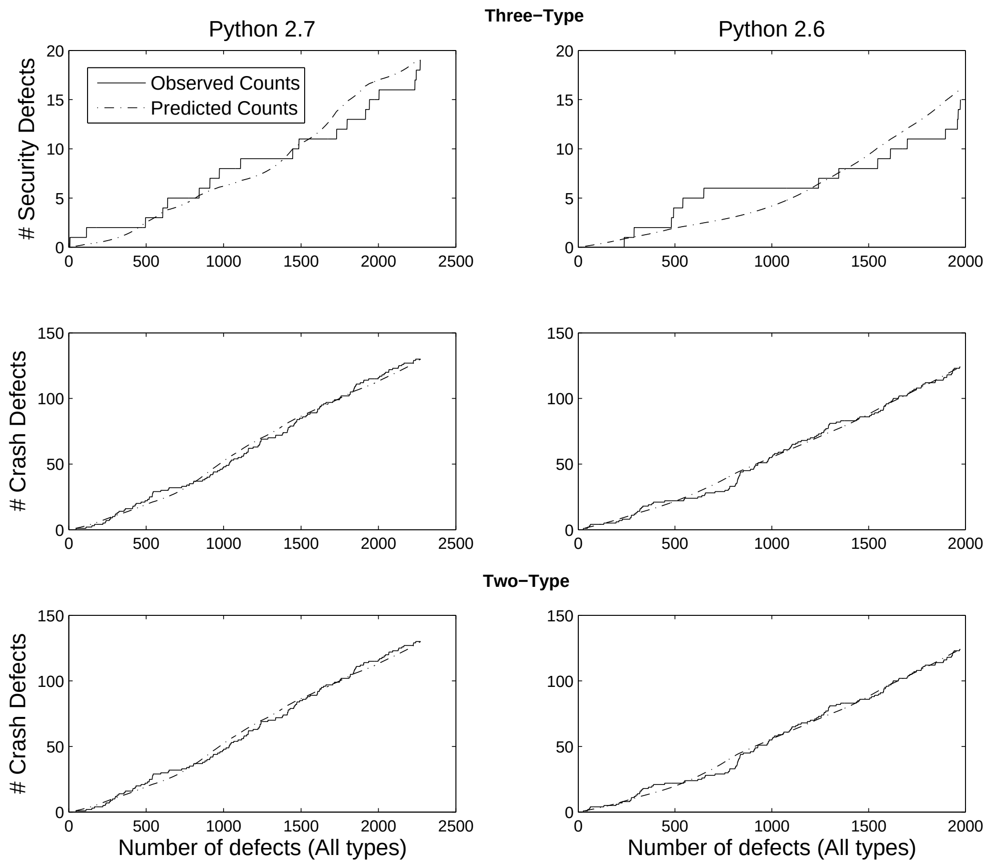
To justify the fit of the model to the Python data, the plot of the estimates of

FIG 5.2. *Plot of* $log(\hat{\Gamma}_i(s))$ *vs s.*



the log of the integrated intensity functions, $log(\hat{\Gamma}_i(s))$, obtained using (5.10), for $i = 0, 1, 2$, for the three-type analysis of Python software are presented in Figure 5.2 with time 0 being the date when the first defect was reported. The plots show that $log(\hat{\Gamma}_i(s))$ are nearly parallel to each other and give evidence in favor of the proportionality assumption (5.2), except for security related defects in version 2.7. To visually assess the fit of the model even further, plots of $\sum_{j=1}^{k} \hat{P}\left(Z_j = i | S^{(j)}, Z^{(j-1)}\right)$ and n_{ik} over k (See Section 5.2.3), for $i = 0, 1, 2$, are considered. The top two panels of Figure 5.3 shows the plots of crash and security related defects, respectively, for the three type model. The third panel shows the plot for crash related defects for a two type model. From this figure one may conclude that fit of the three type model to security related defects is not satisfactory, especially for Version 2.6. However, one needs to be cautious in making such a conclusion since the data corresponding to security defects used for constructing the plot is sparse (less than 20 data points). Nevertheless, the model seems appropriate for crash related defects in both the three-type and the two-type model. This lack of fit of the three-type model for security defects could be the reason why the goodness of fit test for the three type model applied to Python 2.6 has a small p-value of 0.02.

FIG 5.3. *Plots of predicted and actual numbers of crash and security related defects.*



5.5. Concluding Remarks

Bug-databases, which record user reported defects, have become a norm for commercial software. Statistical models for analyzing this increasingly important data source will help end-users to objectively assess the reliability of a software. This paper makes an attempt in that direction by considering, possibly for the first time, an analysis of software defects classified into multiple types. Usage and reporting rate of the software, which are modeled by an infinite dimensional confounding factor $\gamma(s)$ in the analysis of such data, is considered non-parametrically. The partial likelihood approach facilitates estimation of important model parameters. An added advantage of the proposed method is that it can be carried out using standard statistical software packages for easy implementation by practitioners. The proposed reliability metrics are easily interpretable and can be used to compare different software versions. For example, in the analysis of Python software, we gather from the estimate of *MNDF* for Python 2.7 that, on average, about 12 non-crash related defects will be discovered before the discovery of a crash related defect, which is about the same when compared to that of Python 2.6. This indicates that there might not be any additional gain in reliability with respect to "crashes" in using Python 2.7 over Python 2.6. Such analysis may help decision makers to objectively choose between migrating from one version of a software to another. It is interesting to note that, in the limited simulation study, the estimated reliability metrics seem to have better asymptotic convergence properties than the model parameters themselves. Note that the different defect types need to be well-defined for the application of the model. Also, there is possibility of a defect being classified as more than one type. This can be dealt with by considering an additional defect type.

A key application of the proposed model is to compare the reliability of two versions of a software product in a manner that does not depend on the usage of the software. We envisage the methodology to be used primarily for software that has already been released. However, if the method is used for pre-release software, for example from data collected during beta-testing, the prediction of the expected time to discovery of next defect can be a metric used to decide the release of the beta version of the software. Such a prediction would involve extrapolation of the underlying baseline hazard for that defect type using Equation 5.10. An extrapolation method would have to be carefully designed to model the unknown future usage of the software. This could be a direction for future research.

The method considered here may have applications in many areas other than software reliability. As an example, in the context of disease epidemiology, diseased individuals in a geographic location may have different disease types (for example, drug-resistant and non drug-resistant Tuberculosis). The voluntary reporting of a software defect by users corresponds to diseased individuals voluntarily reporting to a hospital to seek treatment, while usage rate may correspond to unknown exposure of the individual. Using an independent logarithmic Poisson model, one could determine whether a particular disease type is spreading or not by checking the sign of the corresponding β_i. The proposed reliability metrics may also be relevant in this epidemiological context. For example, the metric MNDF, could be interpreted as the expected number of patients with the non drug-resistant disease who will report to the hospital before a patient with the drug-resistant disease reports.

Chapter 6

Future Work

The unavoidable price of reliability is simplicity.

C. A. R. Hoare

6.1. Introduction

In this chapter, we briefly discuss future research directions based on the contributions in this thesis. The first relates to the joint analysis of pre- and post-release software defect data using the method of Chapter 5, where we demonstrated how the defect reporting rate DRR can be considered non-parametrically while considering the reliability improvement function parametrically. The second topic concerns the possibility of the reliability improvement function itself being considered semi-parametrically and non-parametrically. The methods developed in Chapter 5 have applications beyond software reliability. We briefly touch upon an application concerning the analysis of data corresponding to product purchases made by customers through an online retailer.

6.2. Joint Analysis of Pre- and Post-Release Software Defect Data

In Chapter 5, we considered the analysis of post-release software defect data retrieved from a bug-database. Many software programs will have pre-release software defects, discovered during controlled testing before the release of the software, in addition to post-release defects recorded in a bug-database. It is natural to ask whether a joint analysis of the two data sets will help improve the estimation of software reliability in the post-release phase. A joint analysis method needs to consider different defect reporting rates for the pre- and post-release software testing phases. However, one may argue that the improvement in reliability due to every additional defect discovery will be the same since in both the pre- and post-release phases as the software is still the same.

6.2.1. Some Existing Methods

A defect discovered during the post-release phase of a software product may not be immediately repaired leading to the defect being reported recurrently for some time after its first reporting. The imperfect debugging model of Goel and Okumoto (1978) allowed for a defect not being repaired immediately by considering the probability, $0 < p < 1$, of not repairing a reported defect immediately after its detection. Let S be the duration of pre-release testing. Jeske, Zhang and Pham (2001) considered the following NHPP model with mean value functions $m_{pre}(s), m_{post}(s)$ and hazards $\lambda_{pre}(s), \lambda_{post}(s)$ for defects discovered in pre- and post-release software, respectively, based on the model of Goel and Okumoto. This is given by

$$m_{pre}(s) = a(1-\exp(-bs)),\ \lambda_{pre}(s) = ab\exp(-bs), \tag{6.1}$$

$$m_{post}(s) = a(1+\mu b)(1-\exp(-\frac{b}{1+n\mu b}s)),\ \lambda_{post}(s) = ab\exp(-\frac{b}{1+n\mu b}s), \tag{6.2}$$

where $p = 1/(1+n\mu b)$ is interpreted as the probability of debugging a software defect in the post-release phase immediately after its reporting and n is the number of users of the software product. Note that the time s in (6.2) is the time since release of the software. The parameters a,b are estimated from pre-release software defect data while the parameter μ may be obtained from the post-release data. This model requires one to know the number of users, n, using the software system and assumes that each of the users have the same usage and the same reporting probability. Both these requirements may not be satisfied in practice. Jeske, Zhang and Pham (2005) suggested another model where they considered the logarithmic Poisson model for both the pre- and post-release software data with

$$m_{pre}(s) = a(1-\exp(-bs)), \quad \lambda_{pre}(s) = ab\exp(-bs), \tag{6.3}$$

$$m_{post}(s) = c(1-\exp(-ds)), \quad \lambda_{pre}(s) = ab\exp(-bs). \tag{6.4}$$

They assumed $c = a/K_c$ and $d = b/K_d$, where K_c and K_d were calibration factors to be obtained by using data from previous software releases. They also suggested a likelihood ratio test for testing $K_c = 1$ vs $K_c \neq 1$. The model of Jeske, Zhang and Pham (2005) is the first in a series in a models that made use of a correction or calibration factor for the parameters of post-release software reliability model based on the pre-release model.

6.2.2. An Alternative Approach

If the defects are classified into $m+1$ types for $i = 0,\ldots,m$, the model described in Chapter 5 can be used for joint consideration of both pre- and post-release software defect data. Let S be the calendar time when the software was released.

The DRR, $\lambda_i(s)$, of discovering a defect of type i at time s can be modeled as

$$\lambda_i(s) = \begin{cases} \exp(\alpha_i^{pre}) \times \gamma^{pre}(s) \times f_i(\mathbf{N(s)};\theta), & s \le S, \alpha_0^{pre} = 0 \\ \exp(\alpha_i^{post}) \times \gamma^{post}(s) \times f_i(\mathbf{N(s)},\theta), & \gamma^{post}(s) \text{ arbitrary } , s > S, \alpha_0^{post} = 0. \end{cases} \tag{6.5}$$

Here, as in Chapter 5, $\mathbf{N(s)} = (N_0(s),\ldots,N_m(s))$ denotes the multivariate process counting the number of defects of various types up to and including time s. The reliability improvement functions, $f_i(\mathbf{N(s)},\theta)$, for $i = 0,\ldots,m$, can be assumed to be the same for both pre- and post-release data as we are dealing with the same software product. The reason we allow α_i's and $\gamma(s)$ for pre- and post-release phases to be different is to allow for differences in how the software is being used before and after its release. During the software testing, that is, the pre-release phase, when the software is used by experts, the α_i^{pre} may tend to be closer to zero since all the defect types get presumably similar attention for scrutiny. For example, in the pre-release phase there may be more emphasis on discovering security defects due to which α_i^{pre}, corresponding to security defects, may have a large value. The α_i^{post}'s, on the other hand, may may be more varied depending on the nature of the users. In particular, the baseline DRR for post release phase $\gamma^{pre}(s)$ is expected to be larger than $\gamma^{post}(s)$, since the software is subjected to more rigorous scrutiny during the pre-release phase than in the post-release phase. Note that the model in (6.5) is a special case of the model considered in (5.1) and hence the methodology described in Section 5.2 of Chapter 5 can be used to estimate the model parameters. In particular, with only two defect types and assuming $f_i(\mathbf{N}(s)) = \exp(-\beta_i N_i(s))$, for $i = 0,1$, we have, from (5.7),

$$P(Z_j = 1|S^{(j)}, Z^{(j-1)}) = \begin{cases} \frac{e^{\alpha_i^{pre} - \beta_1 N_1(S_j)}}{e^{\alpha_i^{pre} - \beta_1 N_1(S_j)} + e^{-\beta_0 N_0(S_j)}}, & S_j \leq S \\ \frac{e^{\alpha_i^{post} - \beta_1 N_1(S_j)}}{e^{\alpha_i^{post} - \beta_1 N_1(S_j)} + e^{-\beta_0 N_0(S_j)}}, & S_j > S, \end{cases} \tag{6.6}$$

where Z_j indicates the type of the j^{th} defect, amongst the two defect types, reported at time S_j, for $j = 1, \ldots, n$. The parameters, $\alpha_1^{pre}, \alpha_1^{post}, \beta_0, \beta_1$, can be estimated through maximum likelihood estimation involving a logistic regression analysis between Z_j on $N_0(S_j)$, $N_1(S_j)$ and $I(S_j \leq S)$. A test of hypothesis for $\alpha_1^{pre} = \alpha_1^{post}$ vs $\alpha_1^{pre} > \alpha_1^{post}$ can reveal whether the reporting pattern of the type 1 defects with respect to type 0 defects has changed between pre- and post-release phases. Extensions to more than two types of defect can be considered similarly.

Applications of this model to a data-set that contains both pre- and post-release software defect data along with classification of defects into multiple types would be insightful. Parametric specification of the DRR during the pre-release testing phase and non-parametric specification in the post-release testing phase would be another possibility. Under such a model, the estimation of the model parameters would consider products of the likelihood for the observations in the pre-release phase and the partial likelihood in the post-release phase. Establishing the asymptotic distribution of the estimates when the number of pre- and post-release defects are large would be a challenging research problem.

6.3. Non-Parametric Estimation

In the analysis of multi-type defects with $m+1$ defect types considered in Chapter 5, we assumed parametric forms for the reliability improvement functions, $f_i(\mathbf{N(s)})$, for $i = 1, \ldots, m$. In some situation, these parametric forms may be questioned. Hence, a non-parametric consideration of these functions would be useful. Let us consider the case with $m = 1$.

6.3.1. Non-Parametric Estimation With Kernel Regression

Let $\mathbf{N}(s) = (N_0(s), N_1(s))$ be the multivariate process that counts the cumulative number of defects reported across the two types at calendar time s. Let $S_1, \ldots, S_n$ be the calendar times when defects were reported and let Z_j represent the defect type of the j^{th} defect, for $j = 1, \ldots, n$. Under the model specified in (5.7),

$$P(Z_j = 1 | H(S_j)) = \frac{1}{1 + \exp(\alpha_1) \frac{f_1(\mathbf{N}(S_j))}{f_0(\mathbf{N}(S_j))}} = G(\mathbf{N}(S_j)), \text{ say }, \tag{6.7}$$

where $H(S_j) = \{S^{(j)}, Z^{(j-1)}\}$. For a general s, $H(s)$ is the history upto time $s-$ (consisting of all the defect discovery times and their types occurring prior to time s) and the fact that a defect discovery takes place at time s. Then, denoting $Z(s)$ to be the defect type discovered at time s, we have $P(Z(s) = 1|H(s)) = G(\mathbf{N}(s))$. Note that $\mathbf{N}(s)$ is a function of $H(s)$. A non-parametric specification of $G(\mathbf{N}(s))$ would waive the requirement of a parametric specification of $f_0(\mathbf{N(s)})$ and $f_1(\mathbf{N(s)})$. Then, $E(Z(s)|H(s))$ can be modeled non-parametrically as an arbitrary function $G(\mathbf{N}(s))$, depending on $H(s)$ only through $\mathbf{N}(s)$, except that $G(.)$ lies between 0 and 1. A Nadaraya-Watson regression Watson (1964) between $Z(s)$ and $\mathbf{N}(s)$ would estimate $G(\mathbf{N}(s))$, for a given $\mathbf{N}(s)$, using a bandwidth parameter h and a specified bivariate kernel $K(.,.)$, as given by

$$\hat{G}(\mathbf{N}(s)) = \frac{\sum_{j=1}^{n} K\left(\frac{\mathbf{N}(S_j) - \mathbf{N}(s)}{h}\right) Z_j}{\sum_{j=1}^{n} K\left(\frac{\mathbf{N}(S_j) - \mathbf{N}(s)}{h}\right)} \tag{6.8}$$

A choice for the kernel $K(.,.)$ can be the independent bivariate Gaussian kernel with $K(x_1, x_2) \propto \exp(-(x_1^2 + x_2^2))$. The estimated $G(\mathbf{N}(s))$ maybe graphically compared with the one estimated through parametric specification of $f_0(\mathbf{N}(s))$, $f_1(\mathbf{N}(s))$ and α_1 using the method of Chapter 5. A good visual match between the two estimates could justify the assumed parametric forms. Designing a goodness of fit test for the assumed parametric form would be a challenging research problem while at the same time being valuable to a software reliability practitioner. The choice of an appropriate kernel is another challenge . A future research topic would involve determination of an optimal kernel for binary regression involving variables derived from counting processes.

6.3.2. Constrained Non-parametric estimation using PAVA

Another approach to specifying $G(\mathbf{N}(s))$, with two types of defects, would involve making an assumption that $G(\mathbf{N}(s)) = L(N_0(s) - N_1(s))$ for some $L(x)$. The quantity $N_0(s) - N_1(s)$ measures the difference in the number of type 0 defects from the number of type 1 defects reported till calendar time s. One may argue that, more this difference, the higher the chance that the next defect after time s will be of type 1, implying that $L(x)$ is increasing in x. The pooled adjacent violators algorithm (PAVA) can be used to estimate $L(\cdot)$ as a function of the difference $N_0(S_j) - N_1(S_j)$. Let Z_j be an indicator of the j^{th} defect being a type 1 defect. Let $X_j = N_0(S_j) - N_1(S_j)$, for $j = 1, \ldots, n$, and $X_{(1)} \leq \ldots \leq X_{(n)}$ be the ordered X_j's with corresponding Z-values denoted by $Z_{(1)}, \ldots, Z_{(n)}$, respectively. For any x, define

$$j(x) = \sum_{i=1}^{n} I(X_{(i)} \leq x). \tag{6.9}$$

Then,

$$\hat{H}(x) = \min_{j(x) \leq k \leq n} \max_{1 \leq j \leq j(x)} \left\{ \frac{\sum_{i=j}^{k} Z_{(i)}}{k - j + 1} \right\} \tag{6.10}$$

This estimated $\hat{H}(x)$ can be used to compute reliability metrics such as $R(N)$ and $MNDF$ using the methods described in Section 5.3 of Chapter 5. Computing standard errors for $\hat{H}(x)$ and the reliability metrics computed from the estimate can be another direction for future research.

6.4. Applications to Online Retail Analytics

In an online market there are thousands of products available for sale to millions of customers. Online retailing is a fast growing multi-billion dollar industry. What differentiates an online market from a traditional market, from a data analysis perspective, is that every product and customer are uniquely identified and every sale is electronically recorded. An online retailer would benefit from estimates of future demand for different products. In particular, identification of products which are not expected to sell over the next few months can help in inventory control and bring down costs. If we denote $D_i(s)$ to be the cumulative demand of a product i, upto calendar time s among $m+1$ products, estimates of $P(D_i(s+t)-D_i(s)=0)$ for a future duration t is the metric of importance. For example, if $P(D_i(s+t)-D_i(s)=0)$ is close to 1, then the probability of the i^{th} product selling over a future t units of time is negligible and the retailer might consider reducing the inventory of this product. A NHPP model for the counting process $D_i(s)$ seems appropriate for computing this probability. For a variety of products, if a customer has bought the product, then it can be assumed the same customer will not buy the same product again in the near future resulting in a decreased demand for the product. Examples of such products include books where a customer will not buy the same book title twice. Other examples include consumer electronics such as television sets and software products such as mobile applications. The modeling of $D_i(s)$, in such situations, has parallels with the modeling of the counting process representing the cumulative number of defects discovered in a software.

The intensity function $\lambda_i(s)$ of the NHPP governing $D_i(s)$ maybe modeled as a function of $D_i(s-)$. If we let T_i to be the duration between the i^{th} and $(i-1)^{th}$ purchase, we could assume a de-eutrophication model and assume that $T_i \sim Exponential(\alpha - \beta i)$. We would expect $\beta > 0$, but it is possible that $\beta < 0$ for some products where the demand grows exponentially. A more realistic model for $\lambda_i(s)$ would be similar to the model introduced in Chapter 5 as given by

$$\lambda_i(s) = \gamma_i(s)\, f_i(N_i(s-)) \tag{6.11}$$

with $f_i(0) = 1$. The function $\gamma_i(s)$, represents the baseline demand for the product at calendar time s assuming that no customer has purchased the product till time s. The function $\gamma_i(s)$ cannot be assumed to be constant with calendar time s, as one would expect it to be influenced by a variety of market forces such as product pricing, effectiveness of the product marketing and macro-economic conditions, to name a few. A non-parametric specification of the baseline demand function along the lines of (5.5) would be appropriate. This is stated as follows,

$$\lambda_i(s) = \gamma(s) \exp(\alpha_i)\, f_i(N_i(s-)), \tag{6.12}$$

for $i = 0, 1, \ldots, m$, with $\alpha_0 = 1$.

Methods for estimating the proposed model using a partial likelihood approach as discussed in Chapter 5 along with estimation of future demand by extrapolating the Breslow estimate of $\gamma(s)$ can be developed. Demand metrics similar to the reliability metrics $R(N)$ and $MNDF$ can also be proposed. The number of products, m, can be as large as 1000 resulting in a large number of model parameters which increases the risk of over-fitting. Shrinkage or lasso methods of estimation may be used to solve this problem. Incorporation of covariates, such as product pricing and macro-economic variables such as bank interest rates, into the model would make it more realistic and would provide another direction for research. Such methods would provide a new way of analyzing data concerning online retail and provide crucial estimates of product demand for this highly competitive industry.

Bibliography

ABRAN, A., KHELIFI, A., SURYN, W. and SEFFAH, A. (2003). Usability meanings and interpretations in ISO standards. *Software Quality Journal* **11** 325–338.

ABRAN, A., MOORE, J. W., BOURQUE, P., DUPUIS, R. and TRIPP, L. (2004). Guide to the software engineering body of knowledge, 2004 version. *IEEE Computer Society* **1**.

ANDREWS, D. W. (2000). Inconsistency of the bootstrap when a parameter is on the boundary of the parameter space. *Econometrica* **68** 399–405.

AYER, M., BRUNK, H. D., EWING, G. M., REID, W., SILVERMAN, E. et al. (1955). An empirical distribution function for sampling with incomplete information. *The annals of mathematical statistics* **26** 641–647.

BASU, A. and EBRAHIMI, N. (2003). Bayesian Software Reliability Models Based on Martingale Processes. *Technometrics* **54** 150-158.

BEGG, C. B. and GRAY, R. . (1984). Calculation of Polychotomous Logistic Regression Parameters using Individualized Regressions. *Biometrika* **71** 11-18.

BÉRARD, B., BIDOIT, M., FINKEL, A., LAROUSSINIE, F., PETIT, A., PETRUCCI, L. and SCHNOEBELEN, P. (2010). *Systems and software verification: model-checking techniques and tools*. Springer-Verlag, Heidelberg.

BHATTACHARYA, R. and KONG, M. (2007). Consistency and asymptotic normality of the estimated effective doses in bioassay. *Journal of Statistical planning* *and Inference* **137** 643–658.

BLUMENTHAL, S. and MARCUS, R. (1975). Estimating population size with exponential failure. *Journal of the American Statistical Association* **70** 913–922.

BRESLOW, N. (1972). Comment on D. R. Cox (1972) paper. *Journal of the Royal Sta- tistical Society: Series B* **34** 216-217.

BRITISH BROADCASTING CORPORATION, (2000). News Article. Retrieved on 14/07/2014 from `http://news.bbc.co.uk/2/hi/science/nature/585013.stm`.

CAI, K. Y., CAI, L., WANG, W. D., YU, Z. Y. and ZHANG, D. (2001). On the neural network approach in software reliability modeling. *Journal of Systems and Software* **58** 47–62.

COX, D. R. (1972). Regression Models and Life-Tables. *Journal of the Royal Statistical Society. Series B (Methodological)* **34** pp. 187-220.

COX, D. R. (1975). Partial Likelihood. *Biometrika* **62** 269-276.

DERRENNIC, H. and LE GALL, G. (1995). Use of Failure-Intensity Models in the Software- Validation Phase for Telecommunications. *IEEE Transactions on Reliability* **44** 658-665.

DEWANJI, A., KUNDU, S. and NAYAK, T. K. (2012). Nonparametric estimation of the number of components of a superposition of renewal processes. *Journal of Statistical Planning and Inference* **142** 2710–2718.

DEWANJI, A., NAYAK, T. K. and SEN, P. K. (1995). Estimating the number of components of a system of superimposed renewal processes. *Sankhyā: The Indian Journal of Statistics, Series A* **57** 486–499.

DEWANJI, A., SENGUPTA, D. and CHAKRABORTY, A. K. (2011). A discrete time model for software reliability with application to a flight control software. *Applied Stochastic Models in Business and Industry* **27** 723–731.

DOOB, J. L. (1953). *Stochastic processes* **101**. New York Wiley.

EFRON, B. and TIBSHIRANI, R. . (1986). Bootstrap Methods for Standard Errors, Con- fidence Intervals and Other Measures of Statistical Accuracy. *Statistical Science* **1** 54-77.

EICKER, F. (1963). Asymptotic normality and consistency of the least squares estimators for families of linear regressions. *The Annals of Mathematical Statistics* **34** 447–456.

EMBRECHTS, P., KLÜPPELBERG, C. and MIKOSCH, T. (1997). *Modelling extremal events: for insurance and finance*. *Application of Mathematics* **33**. Springer.

FAGERLAND, M. W., HOSMER, D. W. and BOFIN, A. M. (2008). Multinomial goodness-of-fit tests for logistic regression models. *Statistics in medicine* **27** 4238–4253.

FAIRLEY, P. (2004). The unruly power grid. *Spectrum, IEEE* **41** 22–27.

FARR, W. (1996). *Handbook of Software Reliability Engineering* Software Reliability Modeling Survey 71-117. McGraw-Hill, New York.

FREEDMAN, D. A. (1991). Statistical models and shoe leather. *Sociological methodology* **21** 201–313.

GARTNER INC. , (2013). Semiconductor Market Share 2013. Retrieved on 14/07/2014 from `http://www.gartner.com/newsroom/id/2632415`.

GARTNER INC. , (2014). Software Market Share 2013. Retrieved on 14/07/2014 from `http://www.gartner.com/newsroom/id/2696317`.

GOEL, A. L. and OKUMOTO, K. (1978). An Analysis of Recurrent Software Failures on a Real-time Control System. In *Proceedings of the ACM Annual Technical Conference, 1978* 496-500.

GOEL, A. L. and OKUMOTO, K. (1979). Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures. *IEEE Transactions on Reliability* **R-28** 206-211.

GOŠEVA-POPSTOJANOVA, K. and TRIVEDI, K. S. (2001). Architecture-based approach to reliability assessment of software systems. *Performance Evaluation* **45** 179–204.

HOSMER, D. W. and LEMESHOW, S. (1980). Goodness of Fit Tests for the Multiple Logistic Regression Model. *Communcations in Statistics* **A9** 1043-1069.

ISBELL, D., HARDIN, M. and UNDERWOOD, J. (1999). Mars Climate Orbiter team finds likely cause of loss. *NASA news release*.

JACKY, J. (1989). Programmed for disaster. *The Sciences* **29** 22–27.

JANARDHANUDU, G. and VAN WYK, K. (2005). White box testing. *Build Security In,US Department of Homeland Security, https://buildsecurityin.us-cert.gov/articles/best-practices/white-box-testing/white-box-testing*.

JELINSKI, Z. and MORANDA, P. (1972). *Statistical computer performance evaluation* 465-484. New York: Academic Press.

JESKE, D. R., ZHANG, X. and PHAM, L. (2001). Accounting for realities when estimating the field failure rate of software. In *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on* 332–339. IEEE.

JESKE, D. R., ZHANG, X. and PHAM, L. (2005). Adjusting software failure rates that are estimated from test data. *Reliability, IEEE Transactions on* **54** 107–114.

JEWELL, N. P. and KALBFLEISCH, J. D. (2004). Maximum likelihood estimation of ordered multinomial parameters. *Biostatistics* **5** 291–306.

JONES, W. D. (1991). Reliability Models for Very Large Software Systems in Industry. In *Proceedings of the Second International Symposium on Software Reliability Engineering* 35-42.

KAPUR, P., PHAM, H., GUPTA, A. and JHA, P. (2011). *Software reliability assessment with OR applications*. Springer.

LEONHARDT, D. (2000). John Tukey, 85, Statistician; Coined the Word ’Software. *New York Times*.

LI, Z., TAYLOR, J. M. and NAN, B. (2010). Construction of confidence intervals and regions for ordered binomial probabilities. *The American Statistician* **64** 291–298.

LITTLEWOOD, B. (1981). A Bayesian differential debugging model for software reliability. In *ACM SIGMETRICS Performance Evaluation Review* **10** 129–130. ACM.

LITTLEWOOD, B. and VERRALL, J. (1973). A Bayesian reliability growth model for computer software. *Applied statistics* **22** 332–346.

MANNER, H. and REZNIKOVA, O. (2012). A survey on time-varying copulas: Specification, simulations, and application. *Econometric Reviews* **31** 654–687.

MAZZUCHI, T. and SOYER, R. (1988). A Bayes empirical-Bayes model for software reliability. *Reliability, IEEE Transactions on* **37** 248–254.

MCCABE, T. J. (1976). A complexity measure. *Software Engineering, IEEE*

Transactions on 4 308–320.

MORANDA, P. (1975). Prediction of software reliability during debugging. In *Proceedings of the Annual Reliability and Maintainability Symposium* 327-332.

MUSA, J. D. (2005). *Software Reliability Engineering*, 2nd ed. Tata McGraw-Hill.

MUSA, J. D. (2012). Dataset 6. Retrieved on 14/07/2014 from `https://sw.thecsiac.com/databases/sled/swrel.php`.

MUSA, J. D., LANINIO, A. and OKUMOTO, K. (1987). *A. lannino, and K. Okumoto* **1** 5–15. McGraw-Hill, New York.

MUSA, J. D. and OKUMOTO, K. (1984). A logarithmic Poisson execution time model for software reliability measurement. In *Proceedings of the 7th international conference on Software engineering* 230–238. IEEE Press.

NATIONAL INSTITUTE OF STANDARDS, (2002). The Economic Impacts of Inadequate Infrastructure for Software Testing. Retrieved on 14/07/2014 from `www.nist.gov/director/planning/upload/report02-3.pdf`.

NAYAK, T. K. (1991). Estimating the number of component processes of a superimposed process. *Biometrika* **78** 75–81.

OHBA, M. (1984a). *Inflection S-shaped software reliability growth model* 144–162. Springer, Berlin-Hiedelberg.

OHBA, M. (1984b). Software reliability analysis models. *IBM Journal of research and Development* **28** 428–443.

PIGEON, J. G. and HEYSE, J. F. (1999). An improved goodness of fit statistic for probability prediction models. *Biometrical Journal* **41** 71–82.

RADATZ, J., GERACI, A. and KATKI, F. (1990). IEEE standard glossary of software engineering terminology. *IEEE Std* **610121990** 121990.

RAMAMOORTHY, C. and BASTANI, F. B. (1982). Software reliability- Status and perspectives. *IEEE Trans. Software Eng.* **8** 354–371.

ROBERTSON, T., WRIGHT, F. and DYKSTRA, R. Order Restricted Statistical Inference, 1988.

SCHICK, G. J. and WOLVERTON, R. W. (1978). An analysis of competing software reliability models. *Software Engineering, IEEE Transactions on* 2 104–120.

SINGPURWALLA, N. D. and SOYER, R. (1985). Assessing (software) reliability growth using a random coefficient autoregressive process and its ramifications. *Software Engineering, IEEE Transactions on* 12 1456–1464.

SINGPURWALLA, N. D. and SOYER, R. (1992). Non-homogeneous autoregressive processes for tracking (software) reliability growth, and their Bayesian analysis. *Journal of the Royal Statistical Society. Series B (Methodological)* **54** 145–156.

SINGPURWALLA, N. D. and WILSON, S. P. (1994). Software relability modeling. *International Statistical Review* **62** 289–317.

SNYDER, L. D. and MILLER, I. M. (1991). *Random Point Processes in Time and Space, Springer*. *A Wiley-Interscience Publication*. Wiley, New York.

TAMURA, Y. and YAMADA, S. (2007). Software reliability growth model based on stochastic differential equations for open source software. In *Mechatronics, ICM2007 4th IEEE International Conference on* 1–6. IEEE.

TEBBS, J. M. and SWALLOW, W. H. (2003). Estimating ordered binomial proportions with the use of group testing. *Biometrika* **90** 471–477.

TELANG, R. and WATTAL, S. (2007). An empirical analysis of the impact of software vulnerability announcements on firm stock price. *Software Engineering, IEEE Transactions on* **33** 544–557.

TUKEY, J. W. (1958). The teaching of concrete mathematics. *American Mathematical Monthly* **65** 1–9.

V BASAWA, I. and PRAKASA RAO, B. (1980). Asymptotic inference for stochastic processes. *Stochastic Processes and their Applications* **10** 221–254.

VENABLES, W. N. and RIPLEY, B. D. (2002). *Modern applied statistics with S*. Springer.

WANG, Z., WANG, J. and LIANG, X. (2007). Non-parametric estimation for NHPP software reliability models. *Journal of Applied Statistics* **34** 107–119.

WATSON, G. S. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A* **26** 359–372.

WONG, W. H. (1986). Theory of partial likelihood. *The Annals of Statistics* **14** 88–123.

YAMADA, S. (1991). Software quality/reliability measurement and assessment: Software reliability growth models and data analysis. *Journal of Information Processing* **14** 254–266.

YAMADA, S., NISHIGAKI, A. and KIMURA, M. (2003). A stochastic differential equation model for software reliability assessment and its goodness-of-fit. *International Journal of Reliability and Applications* **4** 1–11.

YAMADA, S., OHBA, M. and OSAKI, S. (1983). S-shaped reliability growth modeling for software error detection. *Reliability, IEEE Transactions on* **32** 475–484.

YAMADA, S., KIMURA, M., TANAKA, H. and OSAKI, S. (1994). Software reliability measurement and assessment with stochastic differential equations. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* **77** 109–116.

YANG, B. and LI, X. (2007). A study on software reliability prediction based on support vector machines. In *Industrial Engineering and Engineering Management, 2007 IEEE International Conference on* 1176–1180. IEEE.

YU, S. and ZHOU, S. (2010). A survey on metric of software complexity. In *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on* 352–356. IEEE.