Modeling a System of Softwares Under Imperfect Debugging

Marcus A. Agustin
Ma. Zenia N. Agustin

Abstract

This paper considers a series system of $p$ softwares where the failure of a software follows a modified Jelinski-Moranda model. In this model, the debugging scheme of a software is imperfect in the sense that the number of bugs introduced to the software that caused system failure is a random variable. Moreover, tasks are assigned to the system with the task completion times assumed to be random. The main interest is to estimate the parameters that describe the failure process of each software as well as the system reliability. Finally, the finite-sample properties of these estimators are investigated via a simulation.

Key Words and Phrases: Competing risks, repairable system, software reliability.

AMS Subject Classification: Primary: 62N05 Secondary: 49M15, 62F10, 62N02

1. Introduction

A typical assumption in software reliability is the ‘perfect debugging’ of a software. This means that in the process of removing the bug(s) that caused software failure, no new bugs are introduced. One of the earliest papers in software reliability is by Jelinski and Moranda (1972) where software failure rate is assumed to be proportional to the residual number of bugs, and each bug has a constant failure rate contribution. In addition, the number of bugs decreases by one after each software failure to indicate a perfect removal of the bug that caused software failure. The finite-sample properties of the estimators in the Jelinski-Moranda model were established in van Pul (1992). Several other models have been proposed for software reliability as evidenced by the papers of Goel and Okumoto (1979), Littlewood (1980), van Pul (1993), Barendregt and van Pul (1995), Goseva-Popstojanova and Trivedi (2000), and Jeske and Pham (2001), among others. Goel and Okumoto (1979) extended the Jelinski-Moranda model by using a

---

*M. Agustin is Associate Professor in the Department of Mathematics and Statistics, Southern Illinois University Edwardsville, Edwardsville, IL 62026.
†Z. Agustin is Associate Professor in the Department of Mathematics and Statistics, Southern Illinois University Edwardsville, Edwardsville, IL 62026.
nonhomogeneous Poisson process to model failure times of a software. Jeske and Pham (2001) developed estimation procedures for the unknown parameters in the model proposed by Goel and Okumoto (1979). On the other hand, Littlewood (1980) introduced a model where the bugs in a software have varying failure rate contributions.

One can argue that perfect debugging is an unrealistic assumption due to the human element involved in debugging a software. In addition, the complexity of certain softwares may not only allow introduction of new bugs but also the possibility that the bug(s) that caused the failure may not have been completely eliminated. A partial, but not exhaustive, list of works that considered the imperfect debugging of a single software include the works by van Pul (1993), Fakhre-Zakeri and Slud (1995), Zeephongsekul (1996), Slud (1997), Shyur (2003), Xie and Yang (2003), and Pham (2007). In particular, Shyur (2003) extended the nonhomogeneous Poisson process model of Goel and Okumoto (1979) by assuming that during the debugging phase, new bugs are introduced to the system at different rates. Xie and Yang (2003) considered a cost model that took into account the resource cost associated with the debugging process. The effect of imperfect debugging on the total development cost as well as the time to release a software was studied. On the other hand, Zeephongsekul (1996) considered a software whose failures can be attributed to two types of errors, namely, primary and secondary errors. When a software failure is due to a primary error, there is a probability $1 - p_0$ of imperfect debugging which results in the introduction of secondary errors. van Pul (1993) extended the Jelinski-Moranda model by assuming that the number of bugs introduced to a software during the debugging period is a random variable that depends on an unknown parameter and the size of the software at that particular time.

The failure and subsequent debugging of a software, whether done perfectly or imperfectly, can be treated as a recurrent event. In particular, the debugging of a software can be viewed as a repair in the language of repairable systems, where a repair can be classified as perfect or minimal. A perfect repair occurs when the component is restored to the good-as-new state, while a minimal repair puts the component back to its condition prior to failure. Papers dealing with
repairable systems have considered cases where the probability of a perfect repair is constant as well as scenarios incorporating a time-dependent probability of perfect repair. Related papers include Brown and Proschan (1983), Block, Borges, and Savits (1985), Dorado, Hollander, and Sethuraman (1997), Peña, Strawderman, and Hollander (2001), Agustin (2002), and Agustin, Agustin, and Peña (2007).

In the works cited above, the model of interest involved either only one software or component, or the components are not connected in any kind of structure. The case where softwares are connected in series was considered in Agustin and Peña (1999) and Agustin (1999). In these papers, each software in the series system was assumed to follow the Jelinski-Moranda model. However, in addition to observing software failures, the system was also assigned a task to complete. Agustin and Peña (1999) derived the statistical properties of estimators of relevant model parameters using the counting process framework originally used by van Pul (1992) for the case of a single software.

In this paper, we will consider the case where $p$ softwares are connected in series, a task is assigned to the system, and the debugging process is imperfect. For each software, the failure rate is assumed to depend on the residual number of bugs remaining in the software. In addition, at each observed failure and subsequent debugging, a random number of bugs is introduced into the system. Since softwares have to be released in a timely manner, we will consider a fixed observation period $[0, \tau]$ which can be viewed as the development phase of the software. The focus of this paper is the derivation of estimators of the unknown parameters governing the failure process of the softwares as well as the system reliability at the end of the development period. Finite-sample properties of resulting estimators will be examined via a Monte Carlo simulation. It is worth noting that even though the language used in this paper is that of software reliability, the results can be extended to the recurrent event setting such as repairable systems subject to imperfect repair. A discussion of recurrent event models can be found in Agustin, Agustin, and Peña (2007).
2. Model Formulation

Let \( S_1, S_2, \ldots \) denote the failure times of a software. Since we are assuming that the number of bugs introduced during the debugging phase is a random variable which depends on an unknown parameter as well as software size, the hazard rate function of the failure times can be expressed as

\[
\lambda(t; \phi) = \phi(N(t-) - n(t-))
\]

where \( n(t-) \) is the number of software failures observed just before time \( t \) and \( N(t-) \) is the number of bugs introduced to the software just before time \( t \). The parameter \( \phi \) represents the failure rate contribution of each bug. It should be noted that the hazard rate function in (2.1) is a generalization of the hazard rate function considered in van Pul (1992) where

\[
\lambda(t; \phi, B) = \phi(B - n(t-)).
\]

The parameter \( B \) is the unknown, but fixed, number of bugs that are present in a software at time \( t = 0 \).

Whenever a software failure is observed, the number of bugs that are introduced to the software is denoted by a random variable \( N_i \). We will assume that \( N_i \) follows a Poisson distribution with mean \( \mu K_i \), where \( K_i \) denotes the size of the software at the \( i \)th failure. In other words, the probability that there are \( n \) bugs introduced to the software after the \( i \)th debugging is

\[
P\{N_i = n\} = \frac{(\mu K_i)^n}{n!} e^{-\mu K_i}
\]

for \( n = 0, 1, 2, \ldots \). In addition, at time \( t = 0 \), we will assume that the initial number of bugs in the software is a Poisson random variable \( N_0 \) with mean \( \mu K_0 \). Thus, if the (operational) times in which software failures occur are given by \( T_1, T_2, \ldots \), then

\[
N(t) = \sum_{i=1}^{\infty} N_i I\{T_i \leq t\}.
\]
Consider a series system consisting of \( p \) softwares. Using (2.1), the time to failure of software \( j \) will have a hazard function

\[
\lambda_j(t; \phi_j) = \phi_j(N_j(t-) - n_j(t-)),
\]

for \( j = 1, \ldots, p \). The processes \( N_j(t-) \) and \( n_j(t-) \) denote the number of bugs introduced to software \( j \) and the number of system failures caused by the failure of software \( j \), respectively, just before time \( t \).

Since the failure of a series system is caused by the failure of a single software, we will introduce an indicator variable to denote if system failure is due to the failure of a particular software. For \( i = 1, 2, \ldots \), let

\[
\delta_{ij} = \begin{cases} 
1 & \text{ith system failure is caused by software } j \\
0 & \text{otherwise}
\end{cases}.
\]

Moreover, we will define \( B_{ij} \) to be the number of bugs introduced to software \( j \) at the \( i \)th system failure. From (2.3), it follows that

\[
B_{ij} = \delta_{ij} N_{ij},
\]

where \( N_{ij} \) is a Poisson random variable with mean equal to \( \mu_j K_{ij} \) and with \( K_{ij} \) denoting the size of software \( j \) at the \( i \)th system failure. Note that we allow the introduction of bug(s) only to the software that caused system failure. In addition, at time \( t = 0 \), the initial number of bugs in software \( j \) is given by \( N_{0j} \) which is a Poisson random variable with mean \( \mu_j K_{0j} \) where \( K_{0j} \) is the initial size of software \( j \).

In addition to observing the failure of the system, jobs or tasks will be assigned to the system. Such tasks will have a random completion time and each task will either be completed successfully or censored by system failure. For example, one may consider a successful completion of an airplane flight from one city to another as a completed task. On the other hand, the failure of one of many softwares that are critical for airplane function will result in the flight not being completed. Let us denote the number of task completions observed up to time \( t \) by \( n_0(t) \). Hence, if we define a ‘system event’ to be either a task completion or a system failure due to the failure
of software \( j \), for \( j = 1, \ldots, p \), then the total number of system events observed by time \( t \), represented by \( n^*(t) \), is given by
\[
n^*(t) = n_0(t) + \sum_{j=1}^{p} n_j(t).
\]
In addition, since task completion does not result in any software failure, it follows that, at the \( i \)th system event and for \( j = 1, \ldots, p \), \( \delta_{ij} = 0 \) whenever a task completion is observed. Thus, bugs are not introduced to any software whenever a system task completion is observed. Moreover, if \( T_1, T_2, \ldots \) denote the (operational) times of observing a system event, then the number of bugs introduced to software \( j \) at time \( t \) is
\[
N_j(t) = \sum_{i=1}^{\infty} N_{ij} I\{T_i \leq t, \delta_{ij} = 1\}
\]
while the number of system failures caused by the failure of software \( j \) is
\[
n_j(t) = \sum_{i=1}^{\infty} I\{T_i \leq t, \delta_{ij} = 1\}.
\]
On the other hand, the number of task completions observed at time \( t \) is
\[
n_0(t) = \sum_{i=1}^{\infty} I\{T_i \leq t, \delta_{i1} = 0, \ldots, \delta_{ip} = 0\}.
\]
For a fixed value \( t \), the filtration \( \mathcal{F}_t \) represents all the available data observed in the time period \([0, t]\) (see Andersen, Borgan, Gill, and Keiding (1993)). In our model, the filtration is defined as \( \mathcal{F}_t = \sigma\{(T_i, \delta_{ij}) : T_i \leq t, i = 1, 2, \ldots, j = 0, 1, \ldots, p\} \). Note that the vector \( \mathbf{n}(t) = \{(n_0(t), n_1(t), \ldots, n_p(t)) : t \geq 0\} \) is measurable with respect to \( \mathcal{F}_t \). However, \( \mathbf{N}(t) = \{(N_1(t), \ldots, N_p(t)) : t \geq 0\} \) is unobservable. One can obtain the expected value of \( N_j(t-) \) with respect to \( \mathcal{F}_{t-} \) to be
\[
\mathbb{E}[(N_j(t-)|\mathcal{F}_{t-})] = \mu_j \sum_{h=0}^{n_j(t-)} K_{hj} e^{-\phi_j(t-T^*_{hj})} + n_j(t-),
\]
where we define
\[
T^*_{hj} = \min \left\{ T_i : \sum_{k=1}^{i} \delta_{kj} = h, i = 1, \ldots, n(\tau) \right\}
\]
to be the (operational) time when the $h$th failure of software $j$ occurred.

We will assume that the task completion time is an exponentially distributed random variable with parameter $\beta$. The unknown parameter vector is therefore given by

$$\theta = (\mu_1, \ldots, \mu_p, \phi_1, \ldots, \phi_p, \beta).$$

In order to obtain an estimator of the parameter vector $\theta$, we need the likelihood function based on the observed completion and software failure times up to time $\tau$. The likelihood function will be formulated by making use of the technique presented for the single software case in van Pul (1993). It should be noted that the resulting likelihood function follows from the work by Aalen (1978) using the machinery based on counting processes. For a more detailed discussion on the likelihood formulation, the reader is referred to Andersen, Borgan, Gill, and Keiding (1993). Using the hazard function given in (2.2) and observing that both $N_j(t)$ and $n_j(t)$ change whenever a system failure due to the failure of software $j$ is observed, one obtains the integral

$$\int_0^\tau [N_j(t) - n_j(t)] \, dt = \sum_{h=1}^{n_j(\tau)} (\tau - T_{hj}^*) N_{hj} + \sum_{i=1}^{n(\tau)} \delta_{ij} T_i - n_j(\tau) \tau,$$

where $N_{hj}$ is the random variable that represents the number of bugs introduced to software $j$ at its $h$th failure. Since $\delta_{ij}$ is an indicator of failure of software $j$ at the $i$th system event, it follows that

$$\int_0^\tau (N_j(t) - n_j(t)) \, dt = \sum_{h=1}^{n_j(\tau)} (\tau - T_{hj}^*) N_{hj} + \sum_{h=1}^{n(\tau)} T_{hj}^* - n_j(\tau) \tau.$$

To simplify the expressions, let us define $a_{ij} = \mu_j K_{ij} \exp\{-\phi_j (\tau - T_{ij}^*)\}$ for $i = 0, 1, \ldots, n_j(\tau)$. Consequently, the likelihood function is

$$L_\tau(\theta) = \beta^{n_0(\tau)} \exp\{-\beta \tau\} \times \prod_{j=1}^p \phi_j^{n_j(\tau)} \exp\{\phi_j \sum_{h=1}^{n_j(\tau)} (\tau - T_{hj}^*) - \mu_j \sum_{h=0}^{n_j(\tau)} K_{hj}\} \times \left(\sum_{N_0 j} N_0^j a_{0j} \frac{N_0^j}{N_{i j}}\right) \left(\sum_{N_{i j}} (N_{0 j} + N_{1 j} - 1) a_{1 j} \frac{N_{1 j}}{N_{i j}}\right) \times \cdots \times$$
Setting (2.4) equal to 0, we get the MLE of \( \beta \) while the partial derivative of \( \beta \) with respect to \( \phi \) is

\[
\frac{\partial l_T(\theta)}{\partial \phi_j} = \frac{n_j(\tau)}{\phi_j} + \sum_{h=1}^{n_j(\tau)} (\tau - T_{hj}^*) - \mu_j \sum_{h=0}^{n_j(\tau)} K_{hj} (\tau - T_{hj}^*) e^{-\phi_j(\tau - T_{hj}^*)}
\]

(2.5)

On the other hand, for \( j = 1, \ldots, p \), the partial derivative of \( l_T(\theta) \) with respect to \( \mu_j \) is

\[
\frac{\partial l_T(\theta)}{\partial \mu_j} = - \sum_{h=0}^{n_j(\tau)-1} \left[ \sum_{l=0}^{h} K_{lj} e^{-\phi_j(\tau - T_{lj}^*)} \right]^{-1} \left[ \sum_{l=0}^{h} K_{ij} (\tau - T_{lj}^*) e^{-\phi_j(\tau - T_{lj}^*)} \right].
\]

Setting (2.4) equal to 0, we get the MLE of \( \mu_j \) to be

\[
\hat{\mu}_j = \frac{n_j(\tau)}{\sum_{h=0}^{n_j(\tau)} K_{hj} \left( 1 - e^{-\phi_j(\tau - T_{hj}^*)} \right)}.
\]

Taking the logarithm we get the log-likelihood function to be

\[
l_T(\theta) = n_0(\tau) \log \beta - (\beta \tau) + \sum_{j=1}^{p} n_j(\tau) \log (\phi_j \mu_j)
\]

\[
+ \phi_j \sum_{h=1}^{n_j(\tau)} (\tau - T_{hj}^*) - \mu_j \sum_{h=0}^{n_j(\tau)} K_{hj} \left( 1 - \exp\{-\phi_j(\tau - T_{hj}^*)\} \right)
\]

\[
+ \sum_{h=0}^{n_j(\tau)-1} \left[ \sum_{l=0}^{h} K_{lj} e^{-\phi_j(\tau - T_{lj}^*)} \right]^{-1} \left[ \sum_{l=0}^{h} K_{ij} (\tau - T_{lj}^*) e^{-\phi_j(\tau - T_{lj}^*)} \right].
\]

To obtain estimators of the unknown parameters, we will utilize the maximum likelihood principle. In particular, to obtain the maximum likelihood estimator (MLE) of \( \beta \), we set the partial derivative of \( l_T(\theta) \) with respect to \( \beta \) equal to 0 to obtain

\[
\hat{\beta} = \frac{n_0(\tau)}{\tau}.
\]
where $\hat{\phi}_j$ is the MLE of $\phi_j$. It is obtained by solving the equation $g(\phi_j) = 0$ numerically for $\phi_j$.

Straightforward manipulation of (2.5) will lead us to the function

$$
g(\phi_j) = \frac{n_j(\tau)}{\phi_j} + \sum_{h=1}^{n_j(\tau)} \left( \tau - T_{hj}^* \right) - n_j(\tau) \frac{\sum_{h=0}^{n_j(\tau)} K_{hj} (\tau - T_{hj}^*) e^{-\phi_j (\tau - T_{hj}^*)}}{\sum_{h=0}^{n_j(\tau)} K_{hj} \left( 1 - e^{-\phi_j (\tau - T_{hj}^*)} \right)}
$$

A possible simplification suggested in van Pul (1993) is to consider all $K_{ij}$ to be equal to some $K_j$, with the exception of $K_{0j}$ which is assumed to be much larger than $K_j, j = 1, 2, \ldots, p$.

Using $K_j$ in place of $K_{ij}$, for $j = 1, 2, \ldots, p$, allows us to express (2.6) as

$$
\hat{\mu}_j = \frac{n_j(\tau)}{K_j \sum_{h=0}^{n_j(\tau)} \left( 1 - e^{-\phi_j (\tau - T_{hj})} \right)}
$$

and (2.7) as

$$
g(\phi_j) = \frac{n_j(\tau)}{\phi_j} + \sum_{h=1}^{n_j(\tau)} \left( \tau - T_{hj}^* \right) - n_j(\tau) \frac{\sum_{h=0}^{n_j(\tau)} (\tau - T_{hj}^*) e^{-\phi_j (\tau - T_{hj}^*)}}{\sum_{h=0}^{n_j(\tau)} \left( 1 - e^{-\phi_j (\tau - T_{hj}^*)} \right)}
$$

At the end of the development phase of the system, it is important to have a measure of the system reliability, that is, the probability that the system will function adequately in practice. We define the system reliability as the probability that any given task will be performed successfully before the failure of the system is observed. Thus, if we represent the (random) task completion time by $C$, the software failure times by $S_1, \ldots, S_p$, and keeping in mind that we have a series system of softwares, the system reliability at the end of the development period
where $f_C(c)$ is the density function of $C$. Assuming that the parameter vector is known, the system reliability will depend on $\theta$, the observed processes $n_0(\tau), n_1(\tau), \ldots, n_p(\tau)$, and the corresponding system event times. Moreover, the expected value of $N_j(\tau)$ given the filtration $F_\tau$ is

$$E[N_j(\tau)] = \mu_j \sum_{h=0}^{n_j(\tau)} K_{hj} e^{-\phi_j(\tau-T_{kj}^*)} + n_j(\tau).$$

Thus, from (2.1), the hazard function of the $j$th software at time $\tau$ is

$$\lambda_j(\tau; \mu_j, \phi_j) = \mu_j \sum_{h=0}^{n_j(\tau)} K_{hj} e^{-\phi_j(\tau-T_{kj}^*)}.$$

Based on the observed data up to time $\tau$, the hazard function $\lambda_j(\tau; \mu_j; \phi_j)$ is measurable with respect to the filtration $F_\tau$. Since both $C$ and $\min(S_1, \ldots, S_p)$ are independent exponentially distributed random variables with respective parameters $\beta$ and $\sum_{j=1}^{p} \lambda_j(\tau; \mu_j, \phi_j)$, it follows that the system reliability is

$$\rho_{\tau}(\theta) = \int_0^\infty e^{-c \left( \sum_{j=1}^{p} \lambda_j(\tau; \mu_j, \phi_j) \right)} \beta e^{-\beta c} \, dc$$

$$= \frac{\beta}{\beta + \sum_{j=1}^{p} \lambda_j(\tau; \mu_j, \phi_j)}.$$

Clearly, the system reliability at $\tau$ depends on the unknown parameter vector $\theta$. Using the invariance property of the MLE, an estimator of the system reliability is

$$\hat{\rho}_{\tau} = \rho_{\tau}(\hat{\theta}) = \frac{\hat{\beta}}{\hat{\beta} + \sum_{j=1}^{p} \lambda_j(\tau; \hat{\mu}_j, \hat{\phi}_j)}.$$

3. Finite-Sample Properties

In order to examine the finite-sample properties of $\hat{\theta}$ and $\hat{\rho}_{\tau}$, a simulation study was performed. A series system with $p = 2$ components was observed and debugged up to some fixed time $\tau = 5$. Arbitrarily chosen but fixed values of $\theta = (\mu_1, \mu_2, \phi_2, \phi_2, \beta)$ were considered.
The initial number of bugs for software \( j \) was assumed to be distributed according to a Poisson distribution with mean \( \mu_j K_{0j} \), with \( K_{01} = K_{02} \). In addition, the number of bugs introduced to software \( j \) after a failure and subsequent debugging was assumed to be a Poisson random variable with mean \( \mu_j \bar{K}_j \). For each value of \( \theta \), the experiment was repeated 1000 times.

The first scenario that was simulated corresponds to \( \theta = (0.1,0.1,0.1,0.1,2) \) and \( \bar{K}_1 = \bar{K}_2 = 1 \). In this case, the two softwares are assumed to be almost the same in terms of the initial number of bugs as well as the average number of bugs introduced at each failure and subsequent debugging. The means and standard deviations of the parameter estimates are given in Table 1. On the other hand, the histograms corresponding to the estimates are given in Figure 1. A close look at the values in Table 1 shows that the means of the estimates of \( \mu_1 \) and \( \mu_2 \) are closer to the true values compared to those of \( \phi_1 \) and \( \phi_2 \). The mean of the system reliability at the end of the testing period is noticeably low. This low value, however, should not be a surprise since we have imperfect debugging. In fact, each software started with roughly 50 bugs, on the average. At the end of the testing period, there were still about 32 bugs, on the average, in each software. We also performed simulations where we let the testing period get longer but we did not find marked improvements in the system reliability. Examining the histograms in Figure 1, we notice that only the histogram corresponding to \( \beta \) exhibits some semblance of symmetry. The rest are definitely right skewed.

<table>
<thead>
<tr>
<th>True Value</th>
<th>Estimated Values</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu_1 = 0.1 )</td>
<td>.099</td>
<td>.079</td>
</tr>
<tr>
<td>( \mu_2 = 0.1 )</td>
<td>.105</td>
<td>.079</td>
</tr>
<tr>
<td>( \phi_1 = 0.1 )</td>
<td>.185</td>
<td>.116</td>
</tr>
<tr>
<td>( \phi_2 = 0.1 )</td>
<td>.163</td>
<td>.100</td>
</tr>
<tr>
<td>( \beta = 2 )</td>
<td>1.995</td>
<td>.631</td>
</tr>
<tr>
<td>( \rho )</td>
<td>.053</td>
<td>.045</td>
</tr>
</tbody>
</table>

Table 1: Mean and standard deviation of parameter estimates for \( \theta = (0.1,0.1,0.1,0.1,2) \) with \( K_{01} = K_{02} = 500 \) and \( \bar{K}_1 = \bar{K}_2 = 1 \)

Now, let us examine what happens to the distribution of the parameter estimates if we
Figure 1: Histograms of $\hat{\theta}$ and $\hat{\rho}$ for $\theta = (0.1, 0.1, 0.1, 0.1, 2)$ with $K_{01} = K_{02} = 500$ and $K_{1} = K_{2} = 1$
increase the initial number of bugs. A higher initial number of bugs translates to observing more failures. Thus, what is of interest to us is seeing how fast the distribution of the maximum likelihood estimates converge to normality. We retain the same values $\theta = (0.1, 0.1, 0.1, 0.1, 2)$ and $\overline{K}_1 = \overline{K}_2 = 1$, but we increase $K_{01}$ and $K_{02}$ to 1000. The means and standard deviations of the resulting estimates are presented in Table 2, while the corresponding histograms are given in Figure 2. The average system reliability for this setting is smaller than that of the previous scenario. This is to be expected since on the average, each software started and ended with about 100 and 64 bugs, respectively. In terms of the shapes of the distributions, it seems that the convergence to the asymptotic normal behavior is quite slow. This phenomenon was also noted in van Pul (1991). For this model, it appears that the convergence is slowest for the estimates of $\mu_1$ and $\mu_2$.

<table>
<thead>
<tr>
<th>True Value</th>
<th>Estimated Values</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1 = 0.1$</td>
<td>.106</td>
<td>.067</td>
</tr>
<tr>
<td>$\mu_2 = 0.1$</td>
<td>.105</td>
<td>.068</td>
</tr>
<tr>
<td>$\phi_1 = 0.1$</td>
<td>.145</td>
<td>.082</td>
</tr>
<tr>
<td>$\phi_2 = 0.1$</td>
<td>.143</td>
<td>.080</td>
</tr>
<tr>
<td>$\beta = 2$</td>
<td>2.013</td>
<td>.659</td>
</tr>
<tr>
<td>$\rho$</td>
<td>.022</td>
<td>.016</td>
</tr>
</tbody>
</table>

Table 2: Mean and standard deviation of parameter estimates for $\theta = (0.1, 0.1, 0.1, 0.1, 2)$ with $K_{01} = K_{02} = 1000$ and $\overline{K}_1 = \overline{K}_2 = 1$

Consider next the scenario where one software has a higher initial number of bugs as well as a higher average number of bugs introduced at each failure and subsequent debugging. We used $\theta = (0.1, 0.2, 0.1, 0.1, 2)$, $K_{01} = K_{02} = 500$, and $\overline{K}_1 = \overline{K}_2 = 1$. Thus, software 2 is worse than software 1. The means and standard deviations of the resulting estimates are given in Table 3, while the corresponding histograms are presented in Figure 3. Note that the average system reliability in this setting lies between those of the first two scenarios discussed. This can be attributed to the fact that, on the average, the initial number of bugs present in softwares 1 and 2 are about 50 and 100, respectively. At the end of the fixed testing period, there were still
Figure 2: Histograms of $\hat{\theta}$ and $\hat{\rho}$ for $\theta = (0.1, 0.1, 0.1, 0.1, 2)$ with $K_{01} = K_{02} = 1000$ and $K_1 = K_2 = 1$
Table 3: Mean and standard deviation of parameter estimates for $\theta = (0.1, 0.2, 0.1, 0.1, 2)$ with $K_{01} = K_{02} = 500$ and $K_1 = K_2 = 1$

<table>
<thead>
<tr>
<th>True Value</th>
<th>Mean</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1 = 0.1$</td>
<td>.109</td>
<td>.011</td>
</tr>
<tr>
<td>$\mu_2 = 0.2$</td>
<td>.219</td>
<td>.162</td>
</tr>
<tr>
<td>$\phi_1 = 0.1$</td>
<td>.177</td>
<td>.113</td>
</tr>
<tr>
<td>$\phi_2 = 0.1$</td>
<td>.146</td>
<td>.085</td>
</tr>
<tr>
<td>$\beta = 2$</td>
<td>2.000</td>
<td>.636</td>
</tr>
<tr>
<td>$\rho$</td>
<td>.031</td>
<td>.026</td>
</tr>
</tbody>
</table>

Finally, we investigate the setting where one software has a higher initial number of bugs but the other software has a higher average number of bugs introduced at each failure and subsequent debugging. This situation corresponds to $\theta = (0.1, 0.2, 0.1, 0.1, 2)$, $K_{01} = K_{02} = 500$, $K_1 = 3$, and $K_2 = 1$. Thus, software 2 has more bugs at the start of the testing period, but more bugs, on the average, are introduced at each failure of software 1. The means and standard deviations of the parameter estimates are presented in Table 4, while the corresponding histograms are given in Figure 4. Note that there is not much difference between the mean system reliability in this scenario and that of the previous setting. This can be explained by the fact that at the end of the testing period, there were still about 35 bugs left in software 1 and 67 bugs in software 2, on the average. In all the settings considered in this simulation study, it seems that the estimates of $\phi_1$ and $\phi_2$ appear to be positively biased.

4. Conclusion

A model for a series system of softwares under an imperfect debugging scheme was introduced. At each system failure, the software that caused the failure was debugged. In the process of debugging, the possibility of introducing a random number of new bugs was incorporated. At the end of a fixed observation period, the system reliability was obtained. The maximum likelihood principle was used to derive estimators of the relevant parameters as well as the system reliability. A simulation study was carried out to investigate the finite-sample properties of
Figure 3: Histograms of $\hat{\theta}$ and $\hat{\rho}$ for $\theta = (0.1, 0.2, 0.1, 0.1, 2)$ with $K_{01} = K_{02} = 500$ and $K_1 = K_2 = 1$
Figure 4: Histograms of $\hat{\theta}$ and $\hat{\rho}$ for $\theta = (0.1, 0.2, 0.1, 0.1, 2)$ with $K_{01} = K_{02} = 500$, $K_1 = 3$, and $K_2 = 1$
<table>
<thead>
<tr>
<th>True Value</th>
<th>Estimated Values</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1 = 0.1$</td>
<td>.098</td>
<td>.078</td>
</tr>
<tr>
<td>$\mu_2 = 0.2$</td>
<td>.184</td>
<td>.089</td>
</tr>
<tr>
<td>$\phi_1 = 0.1$</td>
<td>.186</td>
<td>.118</td>
</tr>
<tr>
<td>$\phi_2 = 0.1$</td>
<td>.152</td>
<td>.079</td>
</tr>
<tr>
<td>$\beta = 2$</td>
<td>2.004</td>
<td>.652</td>
</tr>
<tr>
<td>$\rho$</td>
<td>.032</td>
<td>.025</td>
</tr>
</tbody>
</table>

Table 4: Mean and standard deviation of parameter estimates for $\theta = (0.1, 0.2, 0.1, 0.1, 2)$ with $K_{01} = K_{02} = 500$, $K_1 = 3$, and $K_2 = 1$

the estimators. The results of the simulation reinforce the observation of van Pul (1993) that for software models with imperfect debugging, the convergence of estimators to the asymptotic normal behavior is fairly slow.

References


