On Bayesian Inference of Software Reliability Measurement

Shinji Inoue¹, Shigeru Yamada²

¹Faculty of Informatics, Kansai University
2-1-1, Ryozenji, Takatsuki, Osaka 569-1095, Japan
ino@kansai-u.ac.jp
²Graduate School of Engineering, Tottori University
4-101, Koyama-Minami, Tottori, Tottori 680-8552, Japan
yamada@sse.tottori-u.ac.jp

Abstract: We discuss an interval estimation approach for parameters and software reliability assessment measures, which are derived from a discretized software reliability model. In our approach, we apply the Markov chain Monte Carlo (MCMC) method for conducting Bayesian interval estimations in software reliability assessment. Further, we show numerical examples of our approach by using actual fault count data.

Keywords: Software reliability assessment, Bayesian interval estimation, MCMC method.

I. INTRODUCTION

A software reliability growth model [8, 10, 12] is one of the fundamental technologies for quantitative software reliability assessment. In the software reliability measurement based on the models, we often use point estimation for obtaining the estimations of software reliability assessment measures. However, considering a practical situation, we encourage the software development managers to use the interval estimation method when we do not obtain the sufficient number of software reliability data. It should be noted that the interval estimation needs to derive the probability distribution function for the parameter of interest. However, it is very difficult or complex to derive the probability distribution functions analytically even if we use the asymptotic property assuming a large number of samples. And, it is not easy to derive some useful information for the statistical inference analytically on these software reliability assessment measures. Under such background, Kimura and Fujiwara [7], Kaneishi and Dohi [6] and Inoue and Yamada [5] discussed bootstrapping approaches for interval estimations of software reliability and optimal software shipping time based on several types of software reliability growth models. However, we should note that we can obtain software reliability data only once from the software testing. The bootstrapping approach is based on the randomly re-sampled data generated from the observed software fault count data or software failure-occurrence time data. And there is no guarantee that the re-sampling data reflects the software reliability growth process observed. Therefore, we need to discuss more suitable approach for conducting interval estimations of software reliability.

We discuss Bayesian estimation method [2, 9] for software reliability assessment based on a discretized NHPP model [4]. The discretized NHPP model conserves the basic properties of the continuous-time NHPP model and have good prediction and fitting performance for the actual data [4] because the discretized model has consistency with discrete fault count data collection activities. Then, we conduct interval estimation of the model parameters and software reliability assessment measures by Bayesian interval estimation approach. Finally, we show numerical examples of our approach by using actual software fault-count data, and show the results of interval estimations for the model parameters and the software reliability assessment measures.

II. DISCRETIZED EXPONENTIAL NHPP MODEL

We briefly discuss the aspect of the discretized NHPP model [4]. Now we define a discrete counting process \( \{ N_i, i = 0, 1, 2, \ldots \} \) representing the cumulative number of faults detected up to \( i \)-th testing-period. And we can say that the discrete counting process \( \{ N_i, i = 0, 1, 2, \ldots \} \) follows a discrete-time NHPP, which is the discrete analog of the continuous-time NHPP [4, 11],

\[
\Pr \{ N_i = x | N_0 = 0 \} = \frac{[A_i]^x}{x!} \exp \left[ -A_i \right],
\]

(1)

In Eq. (1), \( \Pr \{ \cdot \} \) means the probability of event \( A \) and \( \mu \) is the mean value function of the discrete-time NHPP. The mean value function, \( \mu \), also represents the expected cumulative number of faults detected up to the \( i \)-th testing-period.

Let denote a mean value function following a discretized exponential software reliability growth model [4]. The discretized exponential software reliability growth model is derived from the following difference equations:

\[
H_{i+1} - H_i = \delta b (a - H_i),
\]

(2)

which is the discrete analog of the differential equation of the corresponding continuous-time exponential software reliability growth model [1]. In Eq. (2), \( a \) is the expected initial fault content and \( b \) the fault detection rate per one fault.
Regarding the discretization method, we use the Hirota’s bilinearization methods [3] for conserving the property of the continuous-time exponential software reliability growth model. Solving the above integrable difference equation in Eq. (2), we can obtain an exact solution $H_i$ in Eq. (2) as

$$A_i \equiv H_i = a[1 - (1 - \delta b)^i] \quad (a > 0, b > 0),$$

where $\delta$ represents the constant time-interval. As $\delta \to 0$, Eq. (3) converges to the exact solution of the original continuous-time exponential software reliability growth model.

The discretized exponential software reliability growth model in Eq. (3) has the parameters, $a$ and $\delta b$, which have to be estimated by using actual data. In the point estimation, the parameter estimations of $a$ and $\delta b$, $\hat{a}$ and $\hat{\delta b}$, can be obtained by the following procedure using the method of least-squares. Suppose that we have observed fault counting data $\mathcal{D} \equiv (i, y_i) \ (i = 1, 2, \ldots, n)$, where $y_i$ represents the cumulative number of faults detected up to $i$-th testing-period. We can derive the following regression equation from Eq. (2):

$$c_i = \alpha + \beta d_i,$$

where

$$\begin{align*}
    c_i &= H_{i+1} - H_i \equiv y_{i+1} - y_i, \\
    d_i &= H_i - y_i, \\
    \alpha &= \delta a b, \\
    \beta &= -\delta b.
\end{align*}$$

Based on the regression analysis, we can estimate $\hat{a}$ and $\hat{\delta b}$, which are the estimations of $a$ and $\delta b$ in Eq. (3). Then, the parameter estimations, $\hat{a}$ and $\hat{\delta b}$, can be obtained as

$$\begin{align*}
    \hat{a} &= -\frac{\sum_{i=1}^{n} y_i \cdot (y_i - \hat{a} d_i)}{\sum_{i=1}^{n} d_i^2}, \\
    \hat{\delta b} &= -\frac{\sum_{i=1}^{n} d_i}{\sum_{i=1}^{n} y_i}.
\end{align*}$$

respectively. It is worth noting that $c_i$ in Eq. (4) is independent of $\delta$ because $\delta$ is not used in calculating $c_i$ as showing Eq. (5). Hence, we can obtain the same parameter estimates $\hat{a}$ and $\hat{\delta b}$, respectively, when we choose any constant value of $\delta$ [4].

Regarding software reliability assessment measures, the discrete version of the expected number of remaining faults, $M_i$, represents the expected number of undetected faults in the software system at arbitrary testing-period. Then, we have

$$M_i \equiv \mathbb{E}[N_{\infty} - N_i] = a - A_i = a(1 - \delta b)^i,$$

if we assume that $N_i$ follows the discrete-time NHPP with mean value function $H_i$ in Eq. (3). In Eq. (7), $\mathbb{E}[N_i]$ represents the expectation of $N_i$. And the discrete-time software reliability function, $R(i, h)$, is defined as the probability that a software failure does not occur in the time-interval $(i, i + h) (h = 1, 2, \ldots)$ given that the testing has been going up to the $i$-th testing-period. Then, we have

$$R(i, h) \equiv \Pr[N_{i+h} - N_i = 0 | N_i = x] = \exp[-H_h(1 - \delta b)^i].$$

III. MCMC METHOD FOR BAYESIAN INFERENCE

The point estimations of the parameters in Eq. (3) can be obtained by the linear regression approach as discussed in Section 2. This implies that the parameter $a$ and $\delta b$ are estimated by the method of maximum-likelihood assuming $c_i \sim N(\alpha + \beta d_i, \sigma^2)$, in which $c_i$ follows the normal distribution with mean $\alpha + \beta d_i$ and standard deviation $\sigma^2$.

Now, we derive the posterior distribution of $a$ based on the Bayes’ theorem. The Bayes’ theorem gives us the following relationship between the prior and posterior being related to $a$:

$$p(a|\beta, \sigma^2, \mathcal{D}) \propto p(\mathcal{D}|a, \beta, \sigma^2)p(a),$$

while $\mathcal{D}$, $\beta$ and $\sigma^2$ are given. Eq. (9) implies that the prior distribution of $\alpha$, $p(\alpha)$, is updated as the posterior, $p(a|\beta, \sigma^2, \mathcal{D})$, by the likelihood function for $\mathcal{D}$, $p(\mathcal{D}|a, \beta, \sigma^2)$. Assuming $a \sim N(\mu_a, \tau_a^2)$, we can derive the posterior for $a$ as

$$a|\beta, \sigma^2, \mathcal{D} \sim N \left( \frac{n \hat{\alpha} + \mu_a \overline{\sigma}_1^2 + \sigma^2 \mu_a}{\tau_a^2 n + \sigma^2}, \frac{\tau_a^2 \overline{\sigma}_1^2}{\tau_a^2 n + \sigma^2} \right).$$

The posterior of $\beta$ given $a$, $\sigma^2$ and $\mathcal{D}$ is also derived as

$$\beta|a, \sigma^2, \mathcal{D} \sim N \left( \frac{n \hat{\beta} \overline{\sigma}_1^2 + \sigma^2 \overline{\sigma}_2}{\tau_b^2 \overline{\sigma}_1^2 + \sigma^2}, \frac{\tau_b^2 \overline{\sigma}_1^2}{\tau_b^2 \overline{\sigma}_1^2 + \sigma^2} \right),$$

where the prior of $\beta$ is assumed that $\beta \sim N(\mu_b, \tau_b^2)$. Regarding the posterior of $\sigma^2$, we apply an inverse gamma distribution to the prior because the inverse gamma distribution is the conjugate distribution of the variance for data following the normal distribution. Then, the posterior is derived as

$$\sigma^2|a, \beta, \mathcal{D} \sim IG \left( \frac{n + \tau_0}{2}, \frac{\sum_{i=1}^{n} (y_i - a - \beta d_i)^2 + s_0}{2} \right).$$

In Eq. (12),

$$IG \left( \frac{\tau_0}{2}, \frac{s_0}{2} \right) = \frac{\tau_0^{\tau_0/2}}{\Gamma(\tau_0/2)} \left( \sigma^2 \right)^{-\tau_0/2-1} \exp \left[ -\frac{s_0}{2\sigma^2} \right].$$

where $\tau_0/2 > 0$ and $s_0/2 > 0$, respectively.

The Gibbs sampling method, which is one of the MCMC methods, is used for obtaining the posterior distribution of each parameter because we have the full conditional posterior distributions for each parameter. When the software fault-count data $\mathcal{D}$ is obtained, the Gibbs sampler is concretely given by the following steps:

(Step 1) Estimate $\hat{a}$ and $\hat{\beta}$ from the observed data $\mathcal{D}$ by using the regression analysis discussed in Section 2.

(Step 2) Set $\hat{a}$ and $\hat{\beta}$ and $\sigma^2 = 1$ as $a^{(1)}, \beta^{(1)}, \sigma^2^{(1)}$, which are the initial values of $a$, $\beta$ and $\sigma^2$.

(Step 3) Generate $a^{(r)}$ from $p(a^{(r)}|\beta^{(r-1)}, \sigma^2^{(r-1)}, \mathcal{D})$ in Eq. (10).
Fig. 1 The MCMC samples and posterior distribution for the expected number of remaining faults at \( i=25, M_{25} \)

Fig. 2 The MCMC samples and posterior distribution for the software reliability at \( i=25, R(25,1) \)

(Step 4) Generate \( \beta^{(r)} \) from \( p(\beta^{(r)}|\alpha^{(r)}, \sigma^{2(r-1)}, \mathcal{D}) \) in Eq. (11).

(Step 5) Obtain \( \alpha^{(r)} \) and \( \delta \beta^{(r)} \) by \( -\alpha^{(r)}/\beta^{(r)} \) and \( -\beta^{(r)} \) respectively. And calculate software reliability assessment measures.

(Step 6) Generate \( \sigma^{2(r)} \) from \( p(\sigma^{2(r)}|\alpha^{(r)}, \beta^{(r)}, \mathcal{D}) \) in Eq. (12).

(Step 7) \( r \leftarrow r + 1 \), then back to (Step 2).

**IV: NUMERICAL EXAMPLE**

We apply the following data: \((n, y_n) (n = 1, 2, \ldots, 25; y_{25} = 136) \) [4]. We generated \( r = 10,000 \) samples for all parameters and software reliability assessment measures by following the steps discussed in Section 3. And the first 1,000 samples were discarded as the burn-in samples.

Figures 1 and 2 show the MCMC samples and the posterior distributions of the expected number of remaining faults at \( i = 25, M_{25} \) in Eq. (7), and the software reliability, \( R(25,1) \) in Eq. (8), respectively. From these posterior distributions, we can obtain the interval estimations of the parameters and the software reliability assessment measures. The interval estimation can be obtained by following the notion of the credible interval. The 100\((1 - \gamma)\)% credible interval, denoted by \( C \), satisfies

\[
\int_{C} p(\theta|\mathcal{D})d\theta = 1 - \gamma, \tag{14}
\]

where \( p(\theta|\mathcal{D}) \) is the posterior of the parameter of interest. The HPD (highest posterior density) interval is often used for the interval estimation in Bayesian approach. The 100\((1 - \gamma)\)% HPD interval, which is denoted by \( C_{\text{HPD}} \), is obtained as
Table 1. Results of interval estimations based on 95% HPD interval ($\gamma = 0.05$).

<table>
<thead>
<tr>
<th>Expected initial fault content: $a$</th>
<th>HPD Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lower</td>
</tr>
<tr>
<td>Fault-detection rate: $b$</td>
<td>0.1106</td>
</tr>
<tr>
<td>Expected number of remaining faults: $M_{25}$</td>
<td>6.244</td>
</tr>
<tr>
<td>Software reliability: $R(25,1)$</td>
<td>0.429</td>
</tr>
</tbody>
</table>

\[ C_{\text{HPD}} = \{ \theta \in \Theta | p(\theta|D) \geq k(\gamma) \} \]  

(15)

where $\Theta$ is the set of the value of parameter and $k(\gamma)$ is the largest value satisfying

\[ \int_{C_{\text{HPD}}} p(\theta|D)d\theta = 1 - \gamma. \]  

(16)

Table 1 shows the results of interval estimations based on the 95% HPD interval for the parameters and the software reliability assessment measures. The Bayesian interval estimation is obtained from the posterior distribution, which is updated by the likelihood for the data obtained. Further, the interval estimation in the Bayesian approach is conducted by sampling the parameter repetitively from the full conditional posterior distribution.

CONCLUSIONS

We discussed a method for interval estimation based on Bayesian approach. Especially, applying a discretized NHPP model, which has the high-performance fitting and predictive performance, we derived the full conditional posterior distributions for each parameter. Then, we discussed the interval estimation procedure by applying the Gibbs-sampling method, which is one of the MCMC methods for obtaining the probability distributions of the parameters of interest. Further, we showed numerical examples of our Bayesian interval estimation approach by applying to the software fault-count data observed in an actual software testing.

ACKNOWLEDGMENT

This research was supported in part by the Grant-in-Aid for Scientific Research (C), Grant No. 16K00098, from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

REFERENCES


