Topics in recurrent event prediction with generalized non-homogeneous Poisson process (NHPP) and electronic circuit troubleshooting with Bayesian inference

by

Qianqian Shan

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Statistics

Iowa State University

Ames, Iowa

2019

DEDICATION

I dedicate this work to my advisor for his guidance and my family for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this dissertation. First and foremost, Dr. William Q. Meeker for his guidance, patience and support throughout this research and the writing of this dissertation. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Stephen D. Holland, Dr. Jarad Niemi, Dr. Lily Wang and Dr. Huaiqing Wu.

I'm also very thankful to Dr. Yili Hong from Virginia Tech, the Chapter 2 of this dissertation would not have come to a successful completion without his help.

ABSTRACT

This dissertation consists of three projects focused on seasonal recurrent event prediction, electronic circuit troubleshooting and the development of an open source software, RSpice, respectively.

Big challenges when making recurrent event prediction include (1) the recurrent event rate may show a seasonal pattern and the pattern may also be affected by locations; (2) individual level variabilities can also affect the recurrent event rate. We present a general methodology to solve these challenges by using hierarchical clustering for the seasonal patterns and introducing random effects into our models. These help us in improving both the model fitting and prediction performance. This work is illustrated with two motivating product warranty applications in Chapter 2.

Electronic circuits are widely used in industry, and the troubleshooting process of an electronic circuit based on previous experience may suggest a replacement of the whole circuit or some components, however, the failure of the circuit may just due to a less number of specific electronic components. The unnecessary removal of components can increase the costs significantly. Motivated by finding a more precise and faster troubleshooting procedure based on limited data, we propose the use of data simulation and Bayesian inference in circuit troubleshooting in Chapter 3.

In the third project (Chapter 4), we address challenges that arise when doing data simulation and Bayesian inference in the second project (Chapter 3). In order to do exploratory analysis of an electronic circuit and make inference on which specific electronic components cause the failure of a circuit, we need to use a circuit simulator, ngspice, to generate the response (e.g., voltage values at specified nodes of a circuit) given the circuit setup. We also need to ensure that the component values generated by Bayesian inference algorithm can be passed to the circuit simulator and the output from the circuit simulator can be passed back to the algorithm for evaluation interactively. We present our work of writing an R package called RSpice to accomplish the above tasks.

CHAPTER 1. INTRODUCTION

Recurrent event data exist in a variety of fields such as the automobile warranty, cancer recurrences, the visits to doctors of patients, bull and bear markets in stock prices, recurrences of crime of previous offenders and so on. Accurate predictions on future events can be very useful in terms of saving lives, making profits, reducing the number of crimes and so on. For example, all companies that offer a warranty for their products are required, by law, to put into reserves a sufficient amount of cash so that they will be able to pay their warranty claims. Warranty predictions are important because good predictions can help the companies to maximize their profits and help adjusting their marking strategies. In Chapter 2, we consider a generalized way to make warranty prediction with two examples on product warranty returns. We cluster the warranty contracts hierarchically based on their locations and the seasonality of each location's empirical monthly recurrent event rate. Non-homogeneous Poisson process (NHPP) models are applied together with the covarites, cluster information and individual level random effects to model the monthly recurrent event rate. Predictions and prediction intervals on number of events for each warranty over a specific time period are obtained by integrating the predicted recurrent event rate.

Electronics has become a crucial part of a lot of commonly seen industrial systems, and quick troubleshooting of a failed electronic circuits plays an important role in ensuring the industrial systems working properly and preventing potential losses due to the breakdown of the systems. The troubleshooting of electronic circuits gets harder when the failures are soft failures (out-of-specification of components). With the limited measured voltage data from several testing points of a failed electronic circuit, we propose an efficient approach to make inference on which exact component(s) within the circuit is causing the circuit failure in Chapter 3. We apply adaptive Monte Carlo Markov Chain (MCMC) to map out the posterior distributions each circuit component parameter with the help of the circuit simulator software, ngspice. Then the results allow one to locate the out-of-specification components conveniently.

In Chapter 4, we illustrate how we integrate the open source circuit simulator software, ngspice, into R with an R package called RSpice. The electronic circuit setup can be passed to ngspice from R by calling the RSpice package, the commands to alter circuit component parameters can also be sent to ngspice with one line of command, which is a crucial step to run adaptive MCMC. After running the circuit simulations, the results (e.g., the voltage values at specified test points of a circuit given the component values) can be exported from ngspice to R directly for analysis.

Finally, our main findings and potential future work or improvements to what we have done are summarized in Chapter 5.

CHAPTER 2. SEASONAL WARRANTY PREDICTION BASED ON RECURRENT EVENT DATA

Qianqian Shan, Yili Hong and William Q. Meeker

2.1 Abstract

Warranty return data from repairable systems, such as home appliances, lawn mowers, computers, and automobiles, result in recurrent event data. The non-homogeneous Poisson process (NHPP) model is used widely to describe such data. Seasonality in the repair frequencies and other variabilities, however, complicate the modeling of recurrent event data. Not much work has been done to address the seasonality, and this paper provides a general approach for the application of NHPP models with dynamic covariates to predict seasonal warranty returns that motivated our work. The methods presented here, however, can be applied to other applications resulting in seasonal recurrent-event data. A hierarchical clustering method is used to stratify the population into groups that are more homogeneous than the overall population. The stratification facilitates modeling the recurrent event data with both time-varying and time-constant covariates. We demonstrate and validate the models using warranty claims data for two different types of products. The results show that our approach provides important improvements in the predictive power of monthly events compared with models that do not take the seasonality into account.

Keywords: EM algorithm, Hierarchical clustering, Missing data, NHPP, Random effects, Seasonal dynamic covariates

2.2 Introduction

2.2.1 Background

Predictions of warranty returns, based on recurrent event data from repairable systems are often needed by manufacturing companies so they can help to make decisions on the supply of replacement parts, warranty reserves, pricing of the warranty plans and so on. Monthly predictions of warranty returns are particularly helpful when repairable systems have recurrence rates affected by the month of a year and geographical locations. For example, some products may have a higher recurrence rate in warmer months and in a warmer location due to higher average usage rate. The predictions could be more accurate and useful when the variabilities in seasonality and locations are taken into consideration. Meeker and Escobar (1998)[Chapter 16], without giving details, describe how one might use a non-homogeneous Poisson process (NHPP) to make such predictions on the repairable systems, with the restrictive assumptions that all systems are independent and have the same failure recurrence rate function, $\nu(t)$. The assumptions of the simple NHPP model tend to be too strong for realistic complicated data structures, when products have staggered entry, different failure patterns and other system-to-system sources of variability. The purpose of this paper is to develop a general prediction methodology for applications with these complications. In addition to applications in warranty prediction, the methods are also applicable to many other applications such as the prediction of number of recurrent visits to hospitals of patients in health care industry. We use hierarchical clustering to partition the available data into groups within which there are similar seasonal patterns, and then use the NHPP model with time-varying covariates and random effects to describe the recurrent event warranty data. We illustrate the methods with two different warranty prediction applications.

2.2.2 Related Literature and Our Work

Application of NHPP models to warranty prediction has been discussed extensively in many places in the literature. Rigdon and Basu (2000) present a general review of NHPP models and their applications including the power law process and kinds of tests for the validity of the models. Ross

(2014) describes the NHPP model and shows how to simulate data from an NHPP model based on homogeneous Poisson process (HPP). Fredette and Lawless (2007) describe mixed Poisson models for the prediction of the aggregated number of events at specified calendar times across a population of processes. Koutsellis et al. (2017) present a modified generalized renewal process model for warranty prediction of repairable systems with effects of production dates and replacement of defective components/subsystems.

In other related literature, Hamada et al. (2008)[Section 6.4] and Ryan et al. (2011) apply NHPP models without covariates under a hierarchical Bayesian framework to describe the recurrent events on 48 shared-memory computer processors. Rai (2009) presents a warranty forecasting model with the monthly seasonality modeled by multiplicative seasonal indices based on data from a single representative production month. Wu (2012)[Section 3.5] gives a brief review of different types of coarse warranty data and methods to analyze such data. Xiao et al. (2015) develop nonparametric Bayesian methodology using a seasonal marked point process to predict hurricane occurrences. Cifuentes-Amado and Cepeda-Cuervo (2015) and Ngailo et al. (2016) use NHPP models with seasonality described by trigonometric functions of time in health diseases and seasonal rainfall events, respectively. Slimacek and Lindqvist (2016) add a piecewise constant rate of occurrence of failures (ROCOF) model with both observable and unobservable differences between repairable systems are taken into account. Therneau et al. (2003) show that fitting survival models with random effects can be done efficiently via penalized likelihood estimation. Klein (1992) presents an expectation maximization (EM) algorithm based on a profile likelihood for the semiparametric Cox model.

This paper focuses on developing a flexible warranty event prediction methodology by using the following.

- We develop a parametric recurrent event model to incorporate seasonal effects on the recurrence rates, which can improve the monthly warranty prediction significantly.
- We propose hierarchical clustering on the locations of the systems under warranty to differentiate among different seasonal patterns in the recurrent event processes.

- We take other available fixed covariates effects into consideration to further improve the predictive power of our model.
- We incorporate random effects into our model to describe heterogeneity not accounted for by the covariates.

2.2.3 Motivating Examples

All companies that offer a warranty for their products are required, by law, to put into reserves a sufficient amount of cash so that they will be able to pay their warranty claims. Warranty predictions are extremely important because there are penalties for not having enough cash in reserve and, of course, for holding too much in reserve. While it is common to use simple methods like percentage of sales to predict warranty needs, it is now recognized that there is valuable information in warranty databases that can be used to predict warranty returns more accurately.

We apply our models to two product warranty applications. These data sets differ in terms of the number of systems, number of years of data, recurrence rates, and available covariates. For both applications, we hold out the last 12 months of data for model checking and use the rest of the data to do exploratory analysis and model fitting.

For Product A, warranty/production information for 63,191 systems with 8,406 events from year 2011 to year 2016 is available for modeling. The Product A database contains variables such as in-service date of the systems, start and expiration date of the warranty contracts, country, model year, retail location, warranty price, model type, event date and cost. The warranty price can vary with different levels of coverage. Approximately 10% of the systems have had at least one warranty-return event.

For Product B, warranty/production information for 33,645 systems with 18,972 events from year 2014 to year 2016 is available for modeling. Each record in the data set contains the start and expiration date of the warranty contracts, product model year, retail location, warranty price, event date, and event cost (if any). All of the Product B systems have a 24-month warranty term. Approximately 19% of the systems had at least one warranty-return event.

In both examples, the main objective is to generate point predictions and prediction intervals of future warranty returns.

2.2.4 Overview

The remainder of the paper is organized as follows. Section 2.3 provides general ways to do exploratory analysis of warranty data to help suggest the form of an appropriate model and a clustering methodology to identify different seasonal recurrence rate patterns. Section 2.4 describes the NHPP-based models to be used in this paper. Section 2.5 presents maximum likelihood estimation of the model parameters with and without random effects. Section 2.6 discusses point predictions for the number of future events and compare the point predictions results for different models of Product A. Prediction intervals of the number of future events are presented in Section 2.7. Section 2.8 describes the application of the methodology to the warranty return predictions for Product B. Section 2.9 studies the effects of missing data on clustering and explores two different ways to deal with the missing data. Section 3.8 discusses our conclusions and ideas for future work.

2.3 Exploratory Analysis

Exploratory analysis is often useful for providing insight into the structure of a dataset and as an aid for model building. In this section, we explore the effects of the covariates on recurrence rate by examining the mean cumulative number of system recurrences for different levels of covariates, and apply clustering analysis for identifying different seasonal recurrence rate patterns based on warranty locations.

2.3.1 The Mean Cumulative Function

Nonparametric methods provide useful tools to explore data without making strong assumptions. Kaplan and Meier (1958) introduced the nonparametric estimator of a survival function based on censored time-to-event data. Similar to the survival function for time-to-event data, the mean cumulative function (MCF), giving the mean number of events across a population of systems as a function of system age, provides a useful baseline model for recurrence data. Nelson (1988) describes how to compute a nonparametric estimate using recurrent event data. For more details see Lawless and Nadeau (1995), Meeker and Escobar (1998)[Chapter 16] and Nelson (2003). The sample MCF can, for example, be used to compare the behavior of subpopulations defined by different levels of discrete covariates.

- If the different levels of a covariate have similar MCF curves, then the levels can be combined together for analysis.

- If the different levels of a covariate have importantly different MCF curves, then the terms could be added to the model to take account of the difference. This can be done by either adding the level information as a covariate of the model or by modeling the different levels separately.

Example 1. Exploratory Analysis for Product A Data

Exploratory analysis based on MCF curves of different subpopulations defined by the covariate levels can help to check if the subpopulations have significantly different behavior. For example, by checking the MCF curves for different countries with confidence intervals (CIs) computed by following Meeker and Escobar (1998)[Chapter 16], we can gain insights about how the recurrent event process behaves, as illustrated in Figure 2.1. There are few reported events for the first 180 days in both countries because of the nature of the warranty contracts and the MCF curve for Canadian systems is higher than that for US systems. The MCF curves suggest that the warranty process in the two countries are importantly different. The confidence intervals for the Canadian warranty process are wider because the number of recurrences are smaller.

□

The exploratory analysis using MCF curves provides information about the recurrence rate behaviors of different covariate levels on the system age scale. Other tools are needed if the behavior of the recurrence rate is also affected by factors on the calendar time scale, for example, the clustering analysis tool as introduced in Section 2.3.2.

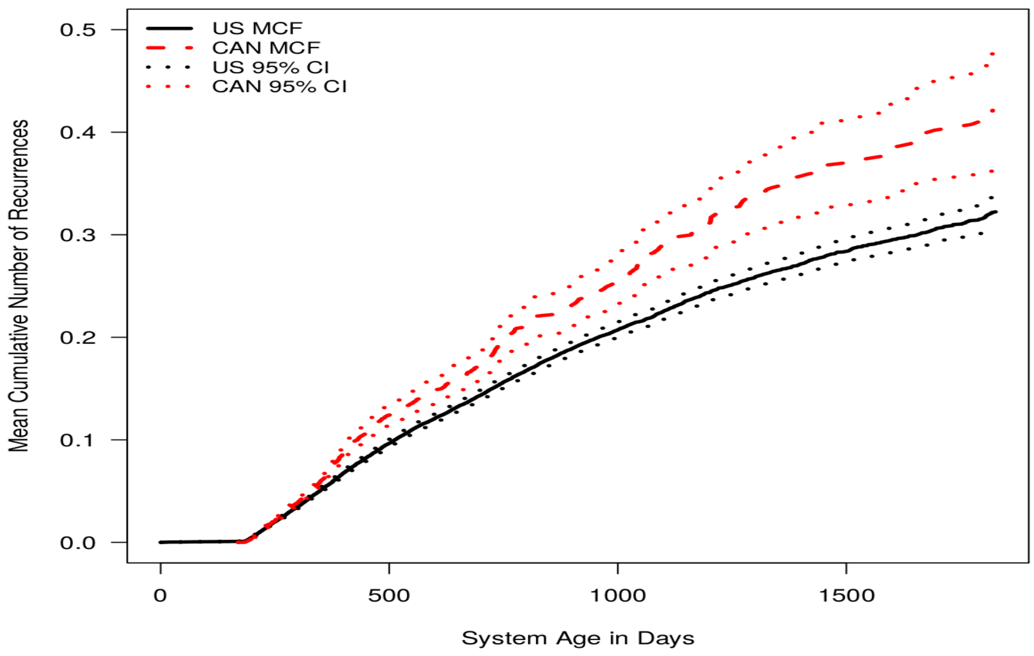

Figure 2.1: MCF versus age of systems in different countries of Product A with 95% pointwise CIs.

2.3.2 Data Clustering and Seasonality

2.3.2.1 Data for Clustering Analysis

The combination of climate differences and geographical locations can affect the usage of products and certain failure mechanisms, resulting in different seasonal patterns in different regions. For example, the usage rate of certain products could be higher during summer than in the winter in the northern US, while the seasonal pattern may be less pronounced in the southern US. Here we describe a data-based approach to group different locations across the US (or other geographical regions) into several clusters so locations within a cluster are, with respect to warranty report seasonality, more homogeneous. The variable to be clustered is the observed overall empirical monthly recurrence rate (the ratio of the number of claims in each month to the corresponding total number of repairable systems at risk) for each location, and we ignore the age effects on the number of events.

The following steps are used to construct the data to be used for clustering. For each location (for each state or province in our applications),

1. Select data that have at least one event and compute the number of events for each calendar month.

2. Compute the number of systems at risk for each calendar month.

3. Compute the empirical monthly recurrence rate as the ratio of the number of events to the number of systems at risk.

4. If the empirical monthly recurrence rates are computed for calendar months across multiple years, average the rates for each of the 12 months of the year.

5. For each location (e.g., US state and Canadian province), there will be 12 empirical recurrence rates for months from January to December. Use the rates as covariates (or features) to cluster the locations.

Note that if there are locations that have few events, we could observe their behavior and merge them to the locations with not dissimilar behavior until there is a substantial number of events in each location to do the clustering analysis.

2.3.2.2 Data with Missing Location Variables

It's common to have missing values in warranty (and other) databases. For our Product A data, 6.6% of the location variables are missing. This will affect our location-based clustering analysis. Every application is different, and there is no general approach for handling missing data that works for all applications. Three commonly used approaches are:

1. Group the missing values with a new category.

2. Impute the missing values based on information that is available.

3. Do a random assignment to replace missing values.

One can choose any one or a combination of the above methods depending on the specific available data, the missing mechanism, the missing percentage and so on. See Example 2 in Section 2.3.2.3 for an example of how we deal with the missing locations for Product A data.

2.3.2.3 Hierarchical Clustering Analysis

In unsupervised clustering of different locations, the empirical recurrence rates for each month of a year per location are the observations. We need to specify a clustering method in order to identify clusters of similar location groups. Popular clustering methods include the K-means algorithm Lloyd (1982); Hartigan (1975); Hartigan and Wong (1979), K-medoid, hierarchical clustering Rousseeuw and Kaufman (1990) and so on. K-means and K-medoid methods require specifying the number of clusters and initial centers of each cluster. In contrast, hierarchical clustering only requires a measure of the similarity (or equivalently, dissimilarity) among observations and a definition of how the dissimilarity of clusters is measured (Hastie et al. 2009 and James et al. 2013). We adopt hierarchical clustering analysis to take advantage of its convenience.

As there are no response variables to characterize each observation, a clear measure of the degree of similarity among the monthly recurrence rates in different locations needs also to be specified (e.g., see James et al. 2013). Possible choices of similarity measures include,

- Euclidean distance: compute the Euclidean distance for each pair of observations, and use it as the measure standard for clustering.
- Correlation-based distance: compute the correlation between each pair of the observations and group locations with highly correlated observations together.

The choice of a similarity measure depends on the data and the prior physical knowledge on the data. In the clustering analysis for different seasonal patterns of recurrent event data, a measure based on correlation performs better when the empirical recurrence rates are low (generally less than 0.01) and the recurrence rate differences are not obvious. A measure based on Euclidean distance performs better when the there exists obvious differences in recurrence rates across different locations. The largest dissimilarity of all the pairwise observations between two clusters is used to compare the dissimilarity of clusters.

The hierarchical clustering produces a dendrogram, a tree-based diagram in which each leaf represents one observation and the height (y-axis) is the specified distance metric. Cutting the dendrogram at different heights can split the observations into different clusters naturally. The selection of where to cut can be affected by factors such as the desired number of clusters, the minimum number of observations within each cluster, how many seasonal patterns exist in the data, and the expected degree of dissimilarity in the seasonal patterns after clustering analysis. The following example shows how the number of clusters can affect the prediction performance of models.

Example 2. Clustering of Product A Seasonal Patterns

Because 4142 systems (approximately 6.6% of Product A data) have missing location variable with no obvious missing patterns, we group these systems together by country and assign new location variables, NA.US and NA.CAN, for US and Canada systems, respectively. For the purpose of comparison, we also use the random assignment method for the missing locations, and the detailed explanation and results are in A.3 of supplemental materials. We use event data after year 2014 to do clustering analysis as there are more events with more repairable systems at risk for most of the locations after 2014. Figure 2.2 shows the dendrogram of the hierarchical clustering results for Product A using correlation distance. Cutting the dendrogram horizontally at around 1.55 naturally separates the data into four clusters with balanced number of locations and event counts in each cluster. The observed events and number of systems at risk by clusters as a function of calendar date are shown in Figure 2.3 and Figure 2.4. Figure 2.5 shows the overall monthly empirical recurrence rates for the four clusters, and it indicates that the seasonal patterns vary considerably.

A sensitivity analysis on the choice of number of clusters shows that cutting the dendrogram such that there are more than four clusters gives approximately the same results as using four clusters. But the amount of computing time required for model fitting can increase dramatically

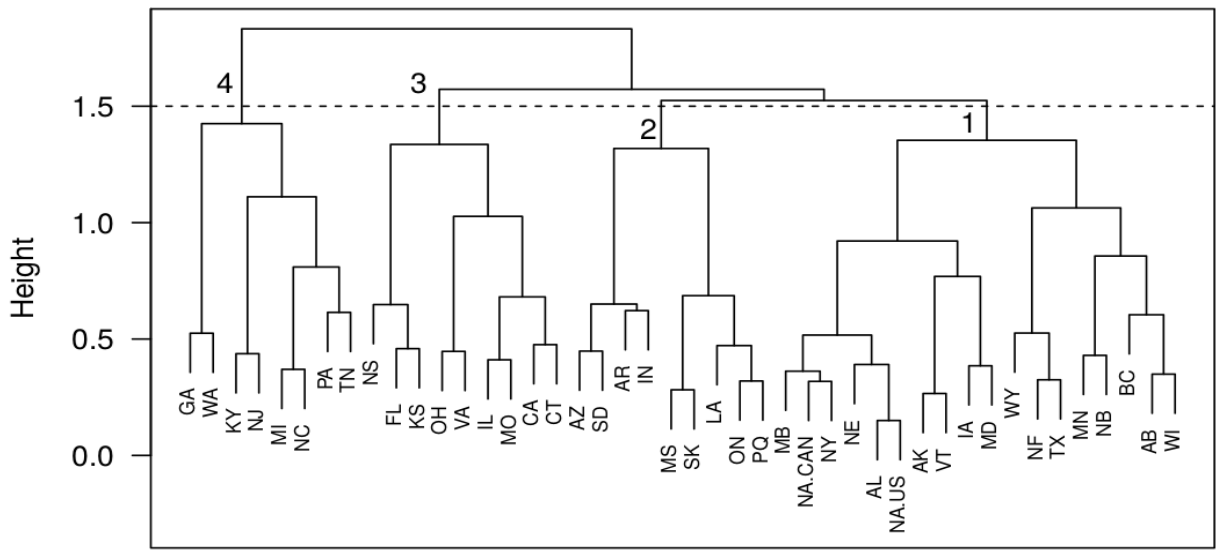

Figure 2.2: Dendrogram of correlation-based hierarchical clustering of Product A. The horizontal line indicates the cutoff location to divide the locations into different clusters. Denote the clusters from left to right as cluster 4, 3, 2 and 1, respectively.

(e.g., there are 50 parameters to be estimated with four clusters in Equation 18.7, however, there will be 122 parameters in the same model in the case of ten clusters). It will take even longer for more models with random effects involved, which requires more complicated algorithm to find the parameter estimates. Given that more clusters does not improve the prediction performance substantially and that it will take significantly longer time to find the model parameter estimates, we choose the number of clusters that provides a good trade-off between model performance and computation costs.

□

2.4 General Models for Recurrence Rates

The Poisson process is commonly used for modeling of repairable systems, but it has the assumption that the number of events in non-overlapping time intervals are statistically independent. For such situations, it's natural to model event counts with an NHPP model with nonconstant

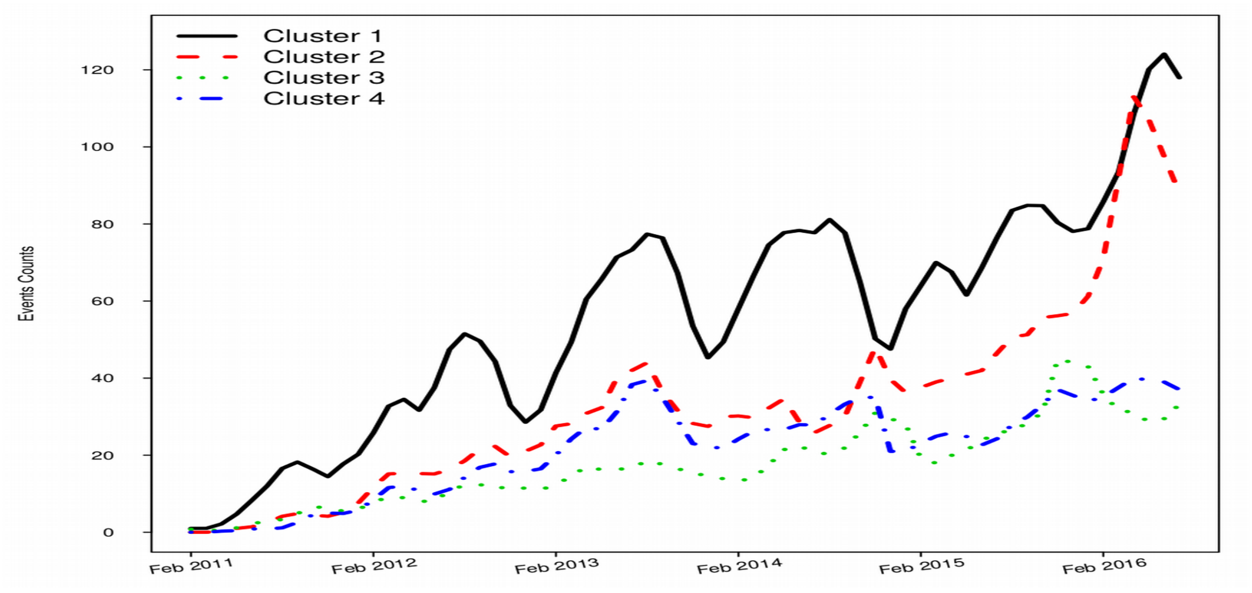

Figure 2.3: Observed event counts versus date for the different clusters of Product A.

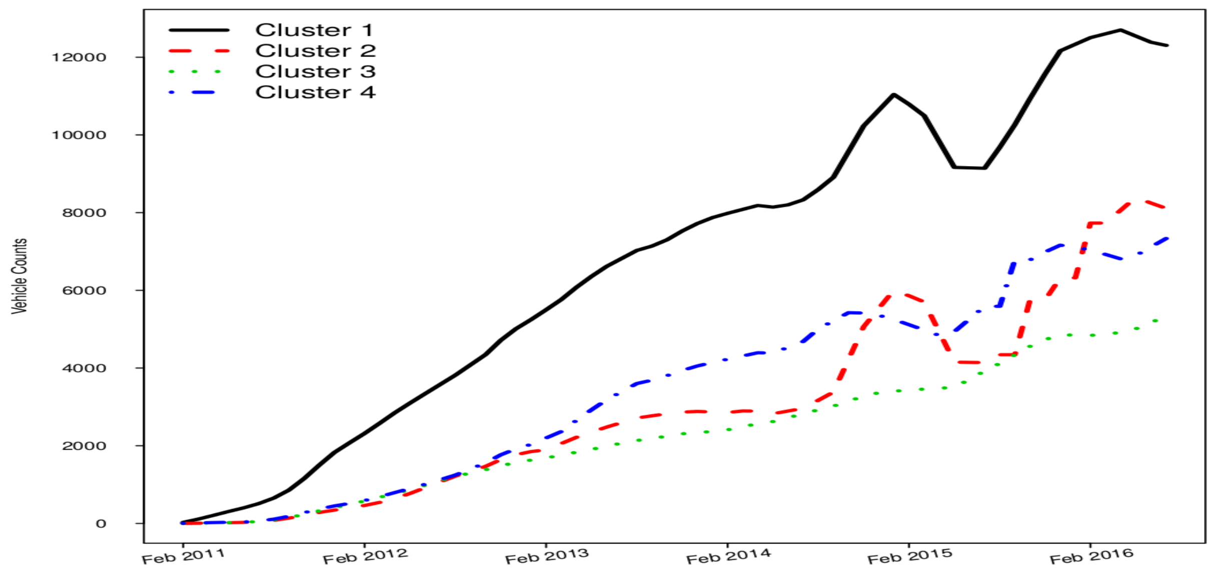

Figure 2.4: Number of systems at risk versus date for the different clusters of Product A.

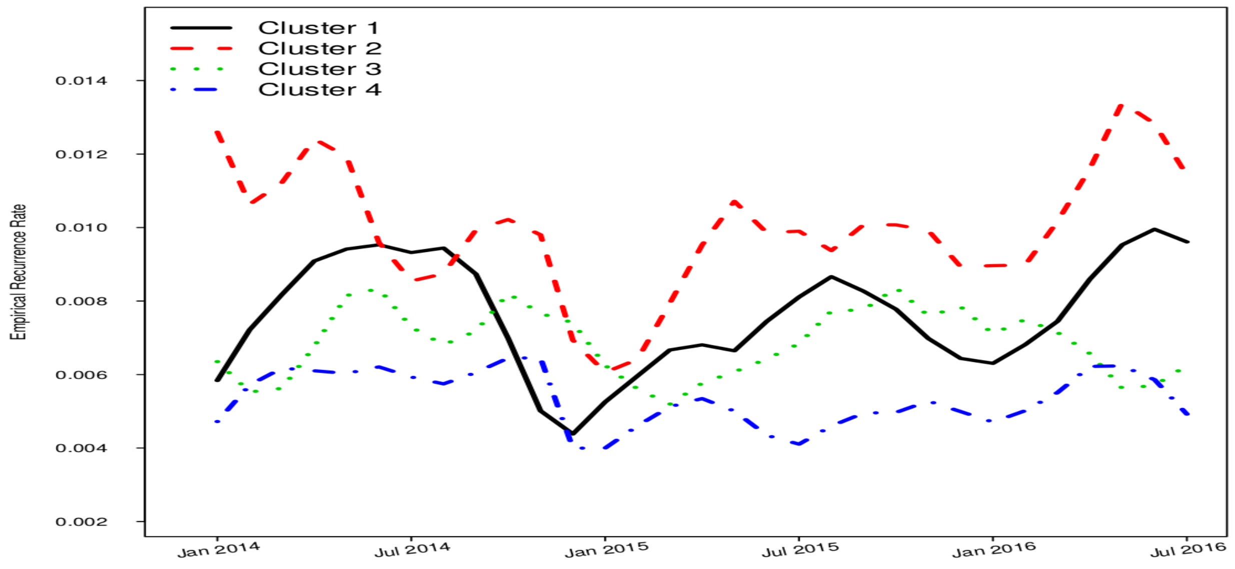

Figure 2.5: Empirical monthly recurrence rate by clusters of Product A systems since year 2014.

recurrence rate as described by Meeker and Escobar (1998)[Chapter 16]. We employ and adapt the widely used NHPP model for our analysis of warranty recurrent event data.

2.4.1 Notation

Let $\mathrm{N}_i(t) = \mathrm{N}_i(0, t)$ denote the observed total number of events up to system age t for repairable system i, where t is the number of days since the system is put into service. Then the process *recurrence rate function* for system i is

$$\nu_i(t) = \lim_{\Delta t \to 0} \frac{\mathrm{E}[\Delta \mathrm{N}_i(t)]}{\Delta t}, \tag{2.1}$$

where $\Delta \mathrm{N}_i(t) = \mathrm{N}_i(t + \Delta t^-) - \mathrm{N}_i(t^-)$ is the number of events in $[t, t + \Delta t)$. The process history at age t of system i, is defined as $\mathcal{H}_i(t) = \{\mathrm{N}_i(c), 0 \leq c < t\}$. We denote the parameter vector of a model as $\boldsymbol{\theta}$.

2.4.2 The Simple NHPP Model

The simple NHPP model assumes that all K systems have the same recurrence rate function and the rate function is defined as

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\boldsymbol{\theta}), \quad i = 1, \cdots, K, \tag{2.2}$$

where $\nu_0(t;\boldsymbol{\theta})$ is a function depending only on system age t and parameters $\boldsymbol{\theta}$. Here we use the power law process

$$\nu_0(t;\boldsymbol{\theta}) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1}, \tag{2.3}$$

with $\boldsymbol{\theta} = (\beta, \eta)^T$ and β without a subscript is the power law parameter. Subsequently we will use β with a subscript to denote regression parameters. The simple NHPP model has strong assumptions that are rarely appropriate for modeling complicated recurrence data structures such as the recurrences in a warranty database.

2.4.3 NHPP Model with Common Seasonal Effects

The NHPP model with simple seasonality assumes that the rate function of each system has the same seasonal behavior over $M = 12$ months of each year,

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\beta,\eta)\exp\left(\sum_{m=1}^{M}\beta_m \mathrm{I}_{m,i}(t)\right), \tag{2.4}$$

where $\nu_0(t;\beta,\eta)$ is as defined in (2.3), $\boldsymbol{\theta} = (\beta, \eta, \beta_1, \cdots, \beta_M)^T$ and

$$\mathrm{I}_{m,i}(t) = \begin{cases} 1 & \text{if system } i \text{ is in calendar month } m \text{ at age } t \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

We set one of the β_m values to be zero in order to have a full rank indicator matrix with its elements defined in (2.5). And the same rule applies to the rest of the models. Because the indicator $\mathrm{I}_{m,i}(\cdot)$ is obtained based on the number of days in service and the calendar date when the system is first put into service, it allows for systems to have staggered entry, as seen in typical warranty databases.

2.4.4 NHPP Model with Seasonal and Cluster Effects

As described in Section 2.3.2, for some applications, the seasonal behavior will depend on the geographical location. We account for this by generalizing the seasonal time-varying covariates. In particular, by assuming that the seasonal recurrence rate patterns vary in both shapes and levels among the clusters, the recurrence rate function of system i is

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\beta,\eta)\exp\left(\sum_{m=1}^{M}\sum_{n=1}^{N}\beta_{m,n}\mathrm{I}_{m,n,i}(t)\right), \tag{2.6}$$

where N is the number of clusters, $\boldsymbol{\theta} = (\beta,\eta,\beta_{1,1},\beta_{1,2},\cdots,\beta_{M,N})^T$ and

$$\mathrm{I}_{m,n,i}(t) = \begin{cases} 1 & \text{if system } i \text{ is in calendar month } m \text{ at age } t \text{ and from cluster } n \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

The model in (2.6) can be simplified if only the levels of the seasonal patterns change across different clusters. In this case,

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\beta,\eta)\exp\left(\sum_{m=1}^{M}\beta_m\mathrm{I}_{m,i}(t) + \sum_{n=1}^{N}\beta'_n\mathrm{I}_{n,i}\right), \tag{2.8}$$

where $\mathrm{I}_{m,i}(\cdot)$ is as defined in (2.5), $\boldsymbol{\theta} = (\beta,\eta,\beta_1,\cdots,\beta_M,\beta'_1,\cdots,\beta'_N)^T$ and the time independent cluster indicator is

$$\mathrm{I}_{n,i} = \begin{cases} 1 & \text{if system } i \text{ is in cluster } n \\ 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

2.4.5 NHPP Model with Seasonal, Cluster and Random Effects

If heterogeneity among systems cannot be completely explained by the Poisson process model with the adjustment of covariates, the incorporation of random effects of the repairable systems can be helpful to explain the system-to-system variation. A model for the recurrence rate for system i conditional on the random effects is introduced in a manner that is similar to the use of frailty models in the survival analysis (Aalen 1988, Therneau et al. 2003 and Cook and Lawless 2007). Then the intensity is

$$\nu_i(t;\boldsymbol{\theta},u_i) \quad = \quad u_i\,\nu_0(t;\beta,\eta)\exp\left(\sum_{m=1}^{M}\sum_{n=1}^{N}\beta_{m,n}\mathrm{I}_{m,n,i}(t)+\boldsymbol{x}_{i,fix}^{T}\boldsymbol{\beta}_{fix}\right). \tag{2.10}$$

Here,

- u_i denotes the i.i.d. random effects for system i. Because the u_i values are unknown and not observed, we assume that the random effects have an independent gamma distribution with mean 1 and variance ϕ, so the random effects are always positive and the distribution of the heterogeneity for individual systems can be reflected by the magnitude of the variance. The density function of u_i is
$$g(u_i;\phi)=\frac{u_i^{\phi^{-1}-1}\exp(-u_i/\phi)}{\phi^{\phi^{-1}}\Gamma(\phi^{-1})}. \tag{2.11}$$

- $\boldsymbol{x}_{i,fix}$ is a vector of fixed covariates that can help explain additional variability in the recurrence process, and $\boldsymbol{\beta}_{fix}$ is the corresponding column vector of regression coefficients. The fixed covariates can be identified in the exploratory analysis phase as described in Section 2.3.1 or by diagnostics based on model fitting and prediction performance.

In particular, $\boldsymbol{\theta}=\left(\beta,\eta,\beta_{1,1},\cdots,\beta_{M,N},\boldsymbol{\beta}_{fix}^{T},\phi\right)^{T}$ is the parameter vector to be estimated.

2.4.6 Comparison of Different Models

The NHPP model in (2.10) can be treated as a general model from the perspective that all of the other models listed above can be viewed as a special case of it:

- Set $\phi=0$, $\beta_{i,j}=0$ for $i=1,2,\cdots,M$ and any $j=1,2,\cdots,N$ and $\boldsymbol{\beta}_{fix}^{T}=\mathbf{0}$, (2.10) reduces to the simple NHPP model in (2.2).

- Set $\phi=0$, $\beta_{i,j}=\beta_{i,k}$ for $i=1,\cdots,M$ and any $j,k\in\{1,2,\cdots,N\}$ and $\boldsymbol{\beta}_{fix}^{T}=\mathbf{0}$, (2.10) reduces to the NHPP model with simple seasonality in (2.4).

- Set $\phi=0$ and $\boldsymbol{\beta}_{fix}^{T}=\mathbf{0}$, (2.10) reduces the NHPP model with seasonal and cluster covariates in (2.6). Further set $\beta_{i,j}=\mathrm{c}_{j-k}+\beta_{i,k}$ with c_{j-k} a constant related to $(j-k)$ for $i=1,2,\cdots,M$ and $1\le k<j\le N$, (2.10) reduces to (2.8).

2.5 Maximum Likelihood Estimation

2.5.1 Likelihood Function

By extending the approach in maximum likelihood estimation of the superimposed Poisson process likelihood which is used for the parameter estimation in Meeker and Escobar (1998), the total likelihood with the random effects u_i for system $i = 1, \cdots, K$ is

$$L\left(\boldsymbol{\theta}|\text{DATA}\right) = \prod_{i=1}^{K} \int \left\{ \left[\prod_{j=1}^{r_i} \nu_i(t_{ij}; \boldsymbol{\theta}) \right]^{\delta_{ij}} \exp\left[-\mu_i(0, t_{a_i}; \boldsymbol{\theta})\right] \right\} g(u_i; \phi) \mathrm{d}u_i, \tag{2.12}$$

where $g(\cdot)$ is defined in (2.11), δ_{ij} is an indicator of the j^{th} out of r_i observed event(s) for system i up to time t_{ij}, and t_{a_i} is the end-of-observation time or the end of warranty time for system i, whichever comes first.

We can re-write (2.10) as the multiplication of the random and non-random parts, $\nu_i(t_{ij}; \boldsymbol{\theta}) = u_i \nu_{b,i}(t_{ij}; \boldsymbol{\beta})$, where $\nu_{b,i}(t; \boldsymbol{\beta}) = \nu_0(t) \exp\left(\sum_{m=1}^{M} \sum_{n=1}^{N} \beta_{m,n} \mathrm{I}_{m,n,i}(t) + \boldsymbol{x}_{fix}^T \boldsymbol{\beta}_{fix}\right)$ and $\boldsymbol{\beta} = (\beta, \eta, \beta_{1,1}, \cdots, \beta_{M,N}, \boldsymbol{\beta}_{fix}^T)^T$. Similar to what is done in Lawless (1987), integrating over u_i for each system i in (2.12) gives the likelihood

$$L(\boldsymbol{\theta}|\text{DATA}) = \prod_{i=1}^{K} \left[\prod_{j=1}^{r_i} \nu_{b,i}(t_{ij}; \boldsymbol{\beta}) \right]^{\delta_{ij}} \frac{\Gamma(\zeta_i)}{\kappa_i^{\zeta_i}}, \tag{2.13}$$

where $\boldsymbol{\theta} = (\boldsymbol{\beta}^T, \phi)^T$, $\zeta_i = r_i + 1/\phi$ and $\kappa_i = \mu_{b,i}(t_{a_i}; \boldsymbol{\beta}) + 1/\phi$ and $\mu_{b,i}(t_{a_i}; \boldsymbol{\beta})$ is short for $\mu_{b,i}(0, t_{a_i}; \boldsymbol{\beta}) = \int_0^{t_{a_i}} \nu_{b,i}(x; \boldsymbol{\beta}) \mathrm{d}x$. Details of the derivation are given in Section A.1 of the supplemental materials.

Note that when there are no random effects (i.e., $\phi = 0$), the likelihood function in (2.12) reduces to

$$L(\boldsymbol{\theta}|\text{DATA}) = \prod_{i=1}^{K} \left\{ \left[\prod_{j=1}^{r_i} \nu_i(t_{ij}; \boldsymbol{\theta}) \right]^{\delta_{ij}} \exp\left[-\mu_i(0, t_{a_i}; \boldsymbol{\theta})\right] \right\}. \tag{2.14}$$

2.5.2 The EM Algorithm

For the model with random effects, we apply the EM algorithm based on the complete-data likelihood. The derivation of the formulas is based on the work of Klein (1992) for the semiparametric Cox model. If we could observe the random effects, $\boldsymbol{u} = (u_1, \ldots, u_i, \ldots, u_K)^T$, the complete-data log-likelihood of $\boldsymbol{\theta} = (\boldsymbol{\beta}^T, \phi)^T$ up to a constant is,

$$\begin{aligned}
&\mathcal{L}(\phi, \boldsymbol{\beta}|\text{DATA}, \boldsymbol{u}) \\
= \;& \sum_{i=1}^{K}\sum_{j=1}^{r_i}\{\log(u_i) + \log[\nu_{b,i}(t_{ij};\boldsymbol{\beta})]\} - \sum_{i=1}^{K} u_i\mu_{b,i}(t_{a_i};\boldsymbol{\beta}) + \sum_{i=1}^{K}\log[g(u_i;\phi)] \\
= \;& \sum_{i=1}^{K}\{r_i\log(u_i) - \log[g(u_i;\phi)]\} + \sum_{i=1}^{K}\left\{\sum_{j=1}^{r_i}\log[\nu_{b,i}(t_{ij};\boldsymbol{\beta})] - u_i\mu_{b,i}(t_{a_i};\boldsymbol{\beta})\right\} \\
= \;& \mathcal{L}_1(\phi|\text{DATA}, \boldsymbol{u}) + \mathcal{L}_2(\boldsymbol{\beta}|\text{DATA}, \boldsymbol{u}) \qquad (2.15)
\end{aligned}$$

where

$$\mathcal{L}_1(\phi|\text{DATA}, \boldsymbol{u}) = -K\left\{\frac{1}{\phi}\log(\phi) + \log\left[\Gamma\left(\frac{1}{\phi}\right)\right]\right\} + \sum_{i=1}^{K}\left[\left(r_i + \frac{1}{\phi} - 1\right)\log(u_i) - \frac{u_i}{\phi}\right],$$

is the part of the likelihood that is related to the parameter ϕ and

$$\mathcal{L}_2(\boldsymbol{\beta}|\text{DATA}, \boldsymbol{u}) = \sum_{i=1}^{K}\left\{\sum_{j=1}^{r_i}\log[\nu_{b,i}(t_{ij};\boldsymbol{\beta})] - u_i\mu_{b,i}(t_{a_i};\boldsymbol{\beta})\right\},$$

is the part of the likelihood that is related to the parameters in $\boldsymbol{\beta}$.

Simple calculations show that the distribution of u_i, conditional on the observed event process history $\mathcal{H}(t_{a_i})$, has a Gamma(ζ_i, κ_i) distribution, where ζ_i and κ_i are the same as in (2.13) and are shape and rate parameters, respectively. The expected complete-data log-likelihood in (2.15) is obtained by replacing the u_i values in $\mathcal{L}_1(\cdot)$ and $\mathcal{L}_2(\cdot)$ with their expected values given $\mathcal{H}(t_{a_i})$,

$$\begin{aligned}
\widehat{\mathcal{L}}_1(\phi|\text{DATA}, \widehat{\boldsymbol{u}}) \;=\;& -K\left\{\frac{1}{\phi}\log(\phi) + \log\left[\Gamma\left(\frac{1}{\phi}\right)\right]\right\} \\
&+ \sum_{i=1}^{K}\left\{\left(r_i + \frac{1}{\phi} - 1\right)[\psi(\zeta_i) - \log(\kappa_i)] - \frac{\zeta_i/\kappa_i}{\phi}\right\}, \qquad (2.16)
\end{aligned}$$

where $\psi(\cdot)$ is the digamma function derived from $\mathrm{E}[\log(u_i)|\mathrm{H}(t_{a_i})] = \psi(\zeta_i) - \log(\kappa_i)$. And,

$$\widehat{\mathcal{L}}_2(\boldsymbol{\beta}|\mathrm{DATA}, \widehat{\boldsymbol{u}}) = \sum_{i=1}^{K} \left\{ \sum_{j=1}^{r_i} \log\left[\nu_{b,i}(t_{ij};\boldsymbol{\beta})\right] - \left(\frac{\zeta_i}{\kappa_i}\right) \mu_{b,i}(t_{a_i};\boldsymbol{\beta}) \right\}. \tag{2.17}$$

In the maximization step, we maximize (2.16) and (2.17) with respect to the parameters ϕ and $\boldsymbol{\beta}$, and in the expectation step, we update the expected values of u_i. The EM algorithm proceeds as follows,

1. Obtain initial estimates of $\boldsymbol{\beta}$ by setting $u_i = 1$ for all systems (or equivalently, $\phi = 0$) and pick a nonzero initial value of ϕ to avoid infinite values of ζ_i and κ_i.
2. Update ζ_i and κ_i using the current values of $\boldsymbol{\beta}$, ϕ and $u_i = (\zeta_i/\kappa_i)$.
3. Update the estimates of ϕ and $\boldsymbol{\beta}$ by maximizing (2.16) and (2.17), respectively.
4. Repeat Step 2 and 3 until convergence.

Example 3. Model Fitting Performance for Product A We fit the following thirteen models labeled from (2.18.1) to (2.18.13) to the Product A data, and check the model fitting performance of different combinations of the seasonal effects, cluster effects, fixed covariates and random effects:

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\boldsymbol{\theta}) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} \tag{2.18.1}$$

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M} \beta_m \mathrm{I}_{m,i}(t)\right) \tag{2.18.2}$$

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M} \beta_m \mathrm{I}_{m,i}(t) + \boldsymbol{x}_{i,fix}^T \boldsymbol{\beta}_{fix}\right) \tag{2.18.3}$$

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M} \beta_m \mathrm{I}_{m,i}(t) + \sum_{n=1}^{N} \mathrm{I}_{n,i}\beta_n\right) \tag{2.18.4}$$

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M} \beta_m \mathrm{I}_{m,i}(t) + \sum_{n=1}^{N} \mathrm{I}_{n,i}\beta_n + \boldsymbol{x}_{i,fix}^T \boldsymbol{\beta}_{fix}\right) \tag{2.18.5}$$

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M} \sum_{n=1}^{N} \beta_{m,n} \mathrm{I}_{m,n,i}(t)\right) \tag{2.18.6}$$

$$\nu_i(t;\boldsymbol{\theta}) = \nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M}\sum_{n=1}^{N} \beta_{m,n}\mathrm{I}_{m,n,i}(t) + \boldsymbol{x}_{i,fix}^T\boldsymbol{\beta}_{fix}\right) \tag{2.18.7}$$

$$\nu_i(t;\boldsymbol{\theta}) = u_i\,\nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M} \beta_m\mathrm{I}_{m,i}(t)\right) \tag{2.18.8}$$

$$\nu_i(t;\boldsymbol{\theta}) = u_i\,\nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M} \beta_m\mathrm{I}_{m,i}(t) + \boldsymbol{x}_{i,fix}^T\boldsymbol{\beta}_{fix}\right) \tag{2.18.9}$$

$$\nu_i(t;\boldsymbol{\theta}) = u_i\,\nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M} \beta_m\mathrm{I}_{m,i}(t) + \sum_{n=1}^{N}\mathrm{I}_{n,i}\beta_{n,i}\right) \tag{2.18.10}$$

$$\nu_i(t;\boldsymbol{\theta}) = u_i\,\nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M} \beta_m\mathrm{I}_{m,i}(t) + \sum_{n=1}^{N}\mathrm{I}_{n,i}\beta_{n,i} + \boldsymbol{x}_{i,fix}^T\boldsymbol{\beta}_{fix}\right) \tag{2.18.11}$$

$$\nu_i(t;\boldsymbol{\theta}) = u_i\,\nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M}\sum_{n=1}^{N} \beta_{m,n}\mathrm{I}_{m,n,i}(t)\right) \tag{2.18.12}$$

$$\nu_i(t;\boldsymbol{\theta}) = u_i\,\nu_0(t;\boldsymbol{\theta}) \exp\left(\sum_{m=1}^{M}\sum_{n=1}^{N} \beta_{m,n}\mathrm{I}_{m,n,i}(t) + \boldsymbol{x}_{i,fix}^T\boldsymbol{\beta}_{fix}\right), \tag{2.18.13}$$

where $\boldsymbol{x}_{i,fix}$ denotes the fixed product country effects of Product A and $\boldsymbol{\beta}_{fix}$ is the corresponding regression coefficient vector. We use the Akaike information criterion (AIC) and the Bayesian information criterion (BIC) based on likelihood functions in (2.13) or (2.14) for model fitting evaluation. The model summarization in Table 2.1 shows that there are improvements in the model fitting as the model incorporates terms for the clusters, seasonality, fixed covariates and random effects. □

2.6 Point Predictions for the Number of Future Events

Predictions for the number of recurrences for a system in a future time-in-service interval, $[t_1, t_2)$ are based on the estimated expected value of the random variable $N_i(t_1, t_2)$.

- When there are no random effects, $N_i(t_1, t_2)$ has a Poisson distribution with mean $\mu_{b,i}(t_1, t_2; \boldsymbol{\beta}) = \int_{t_1}^{t_2} \nu_i(x; \boldsymbol{\beta})\mathrm{d}x$.

Table 2.1: Summary results of different models for the Product A data. The smallest AIC and BIC values are marked in **bold**.

No.	Model Components	AIC	BIC
1	Simple NHPP	152974	152992
2	NHPP with Common Seasonal Effects	152824	152942
3	NHPP with Country and Common Seasonal Effects	152776	152903
4	NHPP with Cluster and Common Seasonal Effects	152629	152774
5	NHPP with Cluster, Common Season and Country Effects	152625	152779
6	NHPP with Cluster and Seasonal Interactions	152568	153012
7	NHPP with Cluster and Seasonal Interactions and Country Effects	152563	153015
8	NHPP with Common Season and Random Effects	149997	150124
9	NHPP with Common Season, Country and Random Effects	149954	150090
10	NHPP with Cluster, Common Season and Random Effects	149833	**149987**
11	NHPP with Cluster, Common Season, Country and Random Effects	149830	149993
12	NHPP with Cluster and Seasonal Interactions and Random Effects	149772	150225
13	**NHPP with Cluster and Seasonal Interactions, Country and Random Effects**	**149768**	150230

- When there are random effects in the model, $N_i(t_1, t_2)$ has a negative binomial distribution with mean $[\zeta_i/\kappa_i]\, \mu_{b,i}(t_1, t_2; \boldsymbol{\beta})$ and probability function

$$\begin{aligned} & \Pr[N_i(t_1,t_2) = n | \text{DATA}, \boldsymbol{\theta}] \\ = \quad & \frac{\Gamma(n+\zeta_i)}{\Gamma(\zeta_i) n!} \left[\frac{\mu_{b,i}(t_1,t_2;\boldsymbol{\beta})}{\mu_{b,i}(t_1,t_2;\boldsymbol{\beta}) + \kappa_i}\right]^n \left[\frac{\kappa_i}{\mu_{b,i}(t_1,t_2;\boldsymbol{\beta}) + \kappa_i}\right]^{\zeta_i}. \end{aligned} \tag{2.19}$$

  That is, $N_i(t_1, t_2)$ has a $NB(\zeta_i, \kappa_i/\left[\mu_{b,i}(t_1, t_2; \boldsymbol{\beta}) + \kappa_i\right])$ distribution (see Section A.2 of the supplemental materials for more details).

Although the recurrence rate function contains time-dependent covariates, month-by-month integration is possible because the time dependent covariates remain unchanged within each calendar month. The total expected number of events in a future month is the sum of the expected number of events for each system at risk. Similarly, the total expected cumulative number of events for all systems up to a specified future month is the sum of cumulative number of events for each system at risk. A point prediction for these quantities can be made by replacing $\boldsymbol{\theta} = (\boldsymbol{\beta}^T, \phi)^T$ by the maximum likelihood estimator $\widehat{\boldsymbol{\theta}}$. A Monte Carlo simulation based procedure to produce point predictions (and prediction intervals) can also be applied as described in Xu et al. (2017).

In order to compare the prediction accuracy of different models, we compute the root mean square error (RMSE), mean absolute error (MAE), and mean absolute percent error (MAPE) of prediction errors for the hold-out data. Denote the non-negative observed and predicted monthly number of events as Y_i and $\widehat{Y}_i$, $i = 1, 2, \cdots, n$, respectively, then

$$\begin{aligned}
\text{RMSE} &= \sqrt{\frac{\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2}{n}} \\
\text{MAE} &= \frac{1}{n}\sum_{i=1}^{n}|Y_i - \widehat{Y}_i| \\
\text{MAPE} &= \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i - \widehat{Y}_i|}{Y_i} \times 100 \qquad (2.20)
\end{aligned}$$

Example 4. Comparisons of point predictions on Product A

Table 2.2 gives a comparison of the prediction performances of different models on the hold-out data. By incorporating the cluster information and assuming different seasonal effects for different clusters, the prediction power and the model fitting performance can be improved with smaller prediction errors and smaller AIC/BIC values. Although the model fitting performance improves with the more complicated model including cluster, season information, and random effects, Model 7 has the best prediction performance, especially for the first 6 months of the hold-out data. This model assumes both the shapes and levels of the seasonal patterns in the recurrence rates are different across clusters and that the recurrence rates vary in different countries. The incorporation of the random effects does not improve the prediction performance for this specific example. Figures 2.6 and 2.7 show the fitted Model 7 of the monthly and cumulative event counts respectively. The fitted model deviates from the observed counts between January 2013 and January 2015. Agreement is better after January 2015.

In order to select an appropriate model, one should take our business goal, data size, model fitting performance, model prediction performance, computation costs, and many other factors into consideration. For example, if our goal is to find a model with reasonably good prediction performance with as little computation costs as possible, Model 7 (sometimes even Model 3) could be a good trade-off between a complicated model and computation time. As we will see in the Product B example in Section 2.8, more complicated models can sometimes lead to a better performance

Table 2.2: Summary results of point predictions on different models for the Product A hold-out data. The superscript 6 and 12 indicate the RMSE/MAE/MAPE of the first 6 and 12 months, respectively. The model with smallest RMSE/MAE/MAPE values is marked in **bold**.

No.	RMSE^6	RMSE^{12}	MAE^6	MAE^{12}	MAPE^6	MAPE^{12}
1	58.2	46.2	46.5	38.7	16.4	15.3
2	45.7	38.8	39.9	33.6	15.0	14.1
3	39.4	31.8	33.1	28.0	12.6	11.8
4	41.2	34.9	34.5	29.5	13.1	12.5
5	38.9	32.8	32.9	27.8	12.5	11.8
6	39.7	34.1	34.0	29.6	12.8	12.4
7	**37.0**	**31.9**	**30.9**	**27.2**	**11.6**	**11.5**
8	50.0	39.9	41.3	34.7	15.6	14.6
9	40.8	34.3	34.2	29.0	13.0	12.3
10	42.7	36.1	36.2	30.8	13.7	13.0
11	40.6	34.2	34.1	29.0	13.0	12.3
12	41.4	35.7	35.9	31.1	13.6	13.1
13	38.9	33.4	32.9	28.8	12.5	12.2

(Model 9 has the best prediction performance), however, it will also take longer time for model fitting due to the need to estimate the random effects. As is usually the case, the modeling process requires judgment combined with experimentation and sensitivity analysis.

$\square$

2.7 Prediction Intervals

2.7.1 Prediction Interval Basics

Prediction intervals (PIs) for random variables and the calibration of PIs are introduced in literature such as Beran (1990), Meeker and Escobar (1998), Lawless and Fredette (2005), Fredette and Lawless (2007), and Fonseca et al. (2014). In our applications, the random variable of interest, Y, is the total number of monthly events or the cumulative number of events at up to a specified month across all systems at risk.

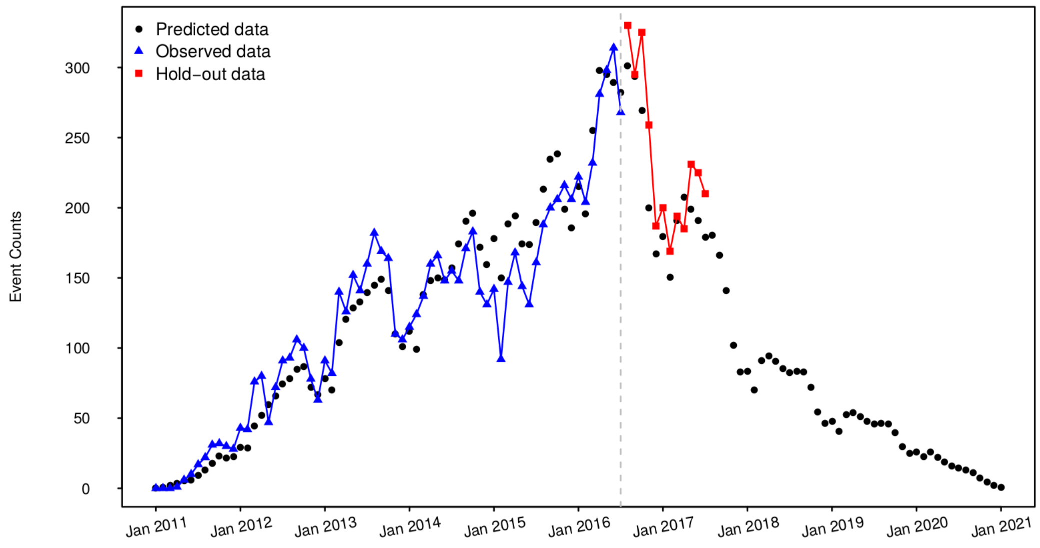

Figure 2.6: Monthly prediction of the event counts for Product A based on Model 7.

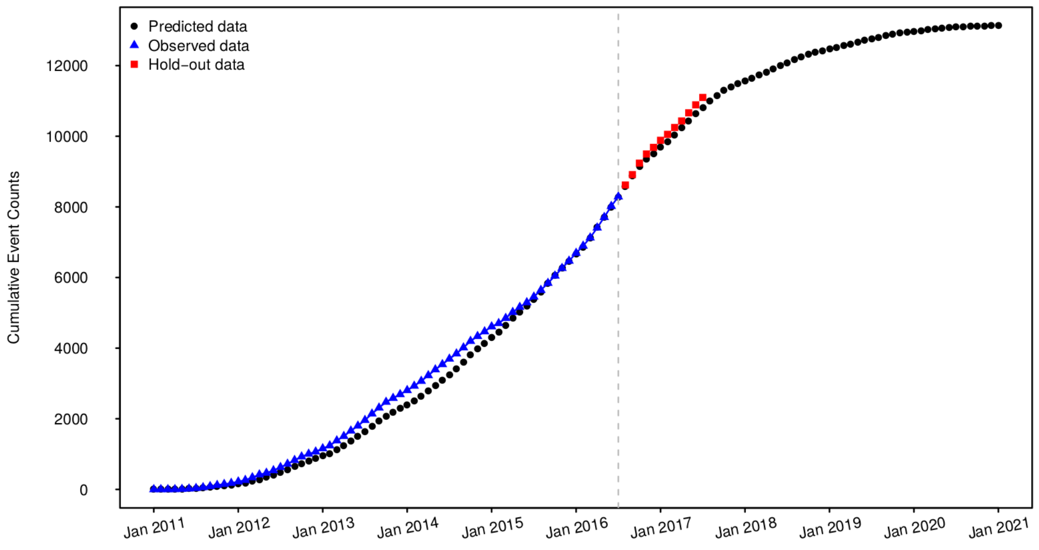

Figure 2.7: Cumulative prediction of the event counts for Product A based on Model 7.

Producing prediction intervals requires the distribution of the sum of the number of events across systems within specified intervals. Under the NHPP model without random effects, the number of events in non-overlapping intervals has a Poisson distribution and the sum of independent Poisson random variables has a Poisson distribution. In contrast, when assuming random effects $\boldsymbol{u}$ for the model, the total sum is a convolution of K negative binomial distributions, which does not have closed form.

Teerapabolarn (2014) shows that, when $\{\zeta_i \cdot \mu_{b,i}(\cdot\,;\boldsymbol{\beta})/\left[\mu_{b,i}(\cdot\,;\boldsymbol{\beta}) + \kappa_i\right]\}$ is small for each i, the distribution of the sum of independent negative binomial random variables can be approximated by Poisson distribution with mean, $\sum_i \left[(\zeta_i/\kappa_i)\mu_{b,i}(\cdot;\boldsymbol{\beta})\right]$, where the sum is across all systems at risk. We use this approximation for the distribution function of random variables when there are random effects in the model.

Here we compare plug-in prediction intervals, simple normal-approximation prediction intervals, and calibrated prediction intervals procedures for a random variable Y with the distribution function $G(Y;\boldsymbol{\theta})$.

2.7.2 Plug-in Prediction Intervals

The simple plug-in prediction interval is obtained by simply using the quantiles of $G(Y;\boldsymbol{\theta})$. Specifically, a two-sided $100\,(1-\alpha)\,\%$ plug-in prediction interval of a random variable Y is $[L,\,U]$ so that,

$$G\left[L < Y \leq U; \widehat{\boldsymbol{\theta}}\right] = 1 - \alpha. \tag{2.21}$$

The actual coverage probability of this procedure will generally be less than $(1-\alpha)$ because plug-in method ignores the uncertainty in $\boldsymbol{\theta}$. It is generally good practice to choose L and U such that $L = Y_{\alpha/2}$ and $U = Y_{1-\alpha/2}$.

2.7.3 Normal Approximate Prediction Intervals

A normal approximate $100(1-\alpha)\%$ prediction interval based on $Z_{\widehat{Y}} = \left(\widehat{Y} - Y\right)/\widehat{se}_{\widehat{Y}} \dot{\sim}$ $\text{NORM}(0,1)$ is,

$$[L,\, U] = \widehat{Y} \pm z_{1-\alpha/2}\widehat{se}_{\widehat{Y}}, \tag{2.22}$$

where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$ quantile of the standard normal distribution, $\widehat{Y}$ is the point prediction, and $\widehat{se}_{\widehat{Y}} = \sqrt{\widehat{\text{Var}}(\widehat{Y})}$, where $\widehat{\text{Var}}(\widehat{Y})$ is computed from the sum of variance of the independent random variables that are summed to obtain Y.

2.7.4 Calibrated Prediction Intervals

Bootstrap procedures to calibrate prediction intervals have been described by literature such as Beran (1990), Meeker and Escobar (1998), Lawless and Fredette (2005), and Fredette and Lawless (2007). Xu et al. (2015) give the following algorithm which is a simulation implementation of the general prediction calibration method described in Section 3 of Lawless and Fredette (2005).

1. Simulate the model estimates $\boldsymbol{\theta}_i^*$ with $i = 1, \cdots, B$ using a parametric bootstrap method.
2. Sample Y_i^* from the distribution function of the random variable, $G(Y;\widehat{\boldsymbol{\theta}})$, where $\widehat{\boldsymbol{\theta}}$ is the ML estimate of the parameters from the original data.
3. Compute $w_i = G(Y_i^*;\boldsymbol{\theta}_i^*)$ for $i = 1, \cdots, B$.
4. Let w_L and w_U be the $\alpha/2$ and $(1-\alpha/2)$ quantiles of the empirical distribution of $(w_1, \cdots, w_B)$.
5. Solve L and U from $w_L = G(L;\widehat{\boldsymbol{\theta}})$ and $w_U = G(U;\widehat{\boldsymbol{\theta}})$.

In Step 1, the parameters $\boldsymbol{\theta}_i^*$ are estimated from the simulated data sets based on $\widehat{\boldsymbol{\theta}}$, which would require a huge amount of computation time in our application. We approximate this procedure by simulating the model estimates $\boldsymbol{\theta}^*$ from the asymptotic multivariate normal distribution of the ML estimates. That is, let $\mathcal{L}(\boldsymbol{\theta})$ denote the total log likelihood of a specified model from K independent systems. Then,

$$\widehat{\boldsymbol{I}}_{\widehat{\boldsymbol{\theta}}} = -\left.\frac{\partial^2 \mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}^T}\right|_{\widehat{\boldsymbol{\theta}}} \tag{2.23}$$

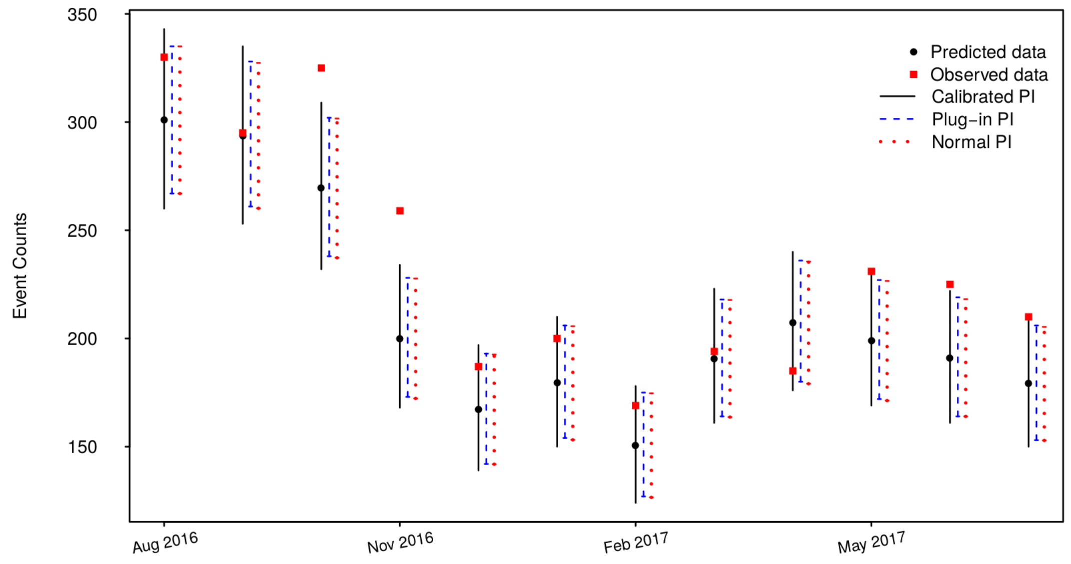

Figure 2.8: Monthly prediction of the event counts for Product A on the hold-out data based on Model 7.

is the observed Fisher information matrix for $\boldsymbol{\theta}$ evaluated at the ML estimate $\hat{\boldsymbol{\theta}}$. Then draws from the multivariate normal distribution $\text{MVN}\left(\hat{\boldsymbol{\theta}}, \widehat{\boldsymbol{I}}_{\hat{\boldsymbol{\theta}}}^{-1}\right)$ can be used in the calibration process. Because of the large amount of data, the approximation will be good.

Example 5. Prediction Intervals for Product A

The monthly and cumulative event predictions from Model 7 for Product A are shown in Figures 2.8 and 2.9, respectively. The calibrated prediction intervals are based on $B = 5{,}000$ simulations. Asymptotic theory (e.g. Beran 1990) suggests that the calibrated prediction interval procedure has coverage probabilities that will be close to the nominal $(1 - \alpha)$ confidence interval. For this example, 9 out of 12 observed event counts are within the calibrated prediction intervals and all observed cumulative event counts are within the calibrated prediction intervals. The plug-in and normal approximate predictions intervals are narrower when compared with the calibrated ones.

□

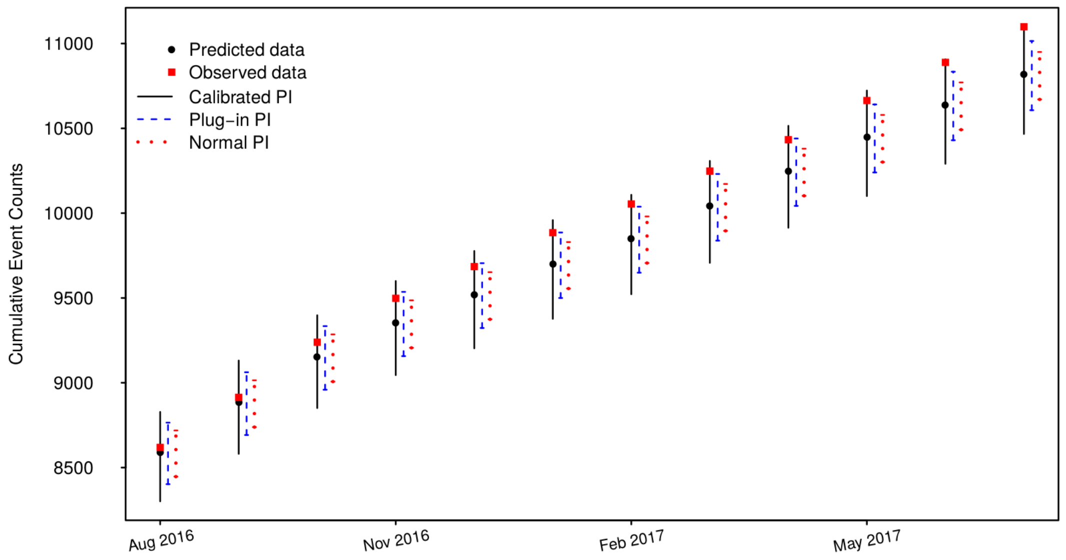

Figure 2.9: Cumulative prediction of the event counts for Product A on the hold-out data based on Model 7.

2.8 Models and Predictions for Product B

In this section, we present an example based on warranty data from Product B, initially described in Section 2.2.3. The models used in this section are similar to those that were applied to Product A, while the seasonal patterns and fixed covariates differ.

2.8.1 Exploratory Analysis

Similar to what we did for Product A, Figure 2.10 gives the MCF for different model years with 95% pointwise confidence intervals. Although we observed only the mean cumulative number of recurrences per system up to the first year for data of model year 2016, we could tell that the curves of the two different model years behave differently, which indicates that the model year information should be taken into account for modeling.

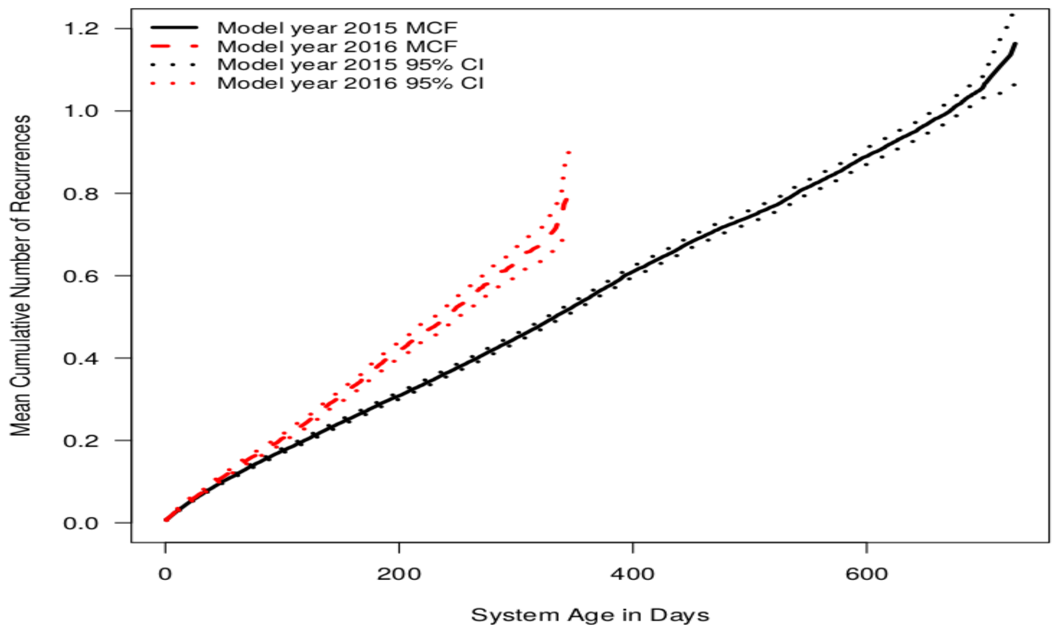

Figure 2.10: MCF versus system age for systems in model years 2015 and 2016 of Product B with 95% pointwise CIs.

2.8.2 Clustering for the Seasonal Models

Figure 2.11 shows the dendrogram of the hierarchical clustering results for Product B based on warranty information of the most recent year. The observed events and number of systems at risk by clusters as a function of calendar time are shown in Figure 2.12 and 2.13. The empirical monthly recurrence rates of the two clusters have the similar shape but different levels as shown in Figure 2.14.

2.8.3 NHPP Model Fitting

In this subsection, we fit the thirteen models in (2.18), and Tables 2.3 and 2.4 show the model fitting on training data and prediction performance hold-out data, respectively. Similar to the results for Product A, model fitting improves by adding cluster, season, fixed and random effects to the model. But the model with cluster factors does not improve the predictions because we only have two years of data for clustering and model fitting. When doing prediction, we implicitly

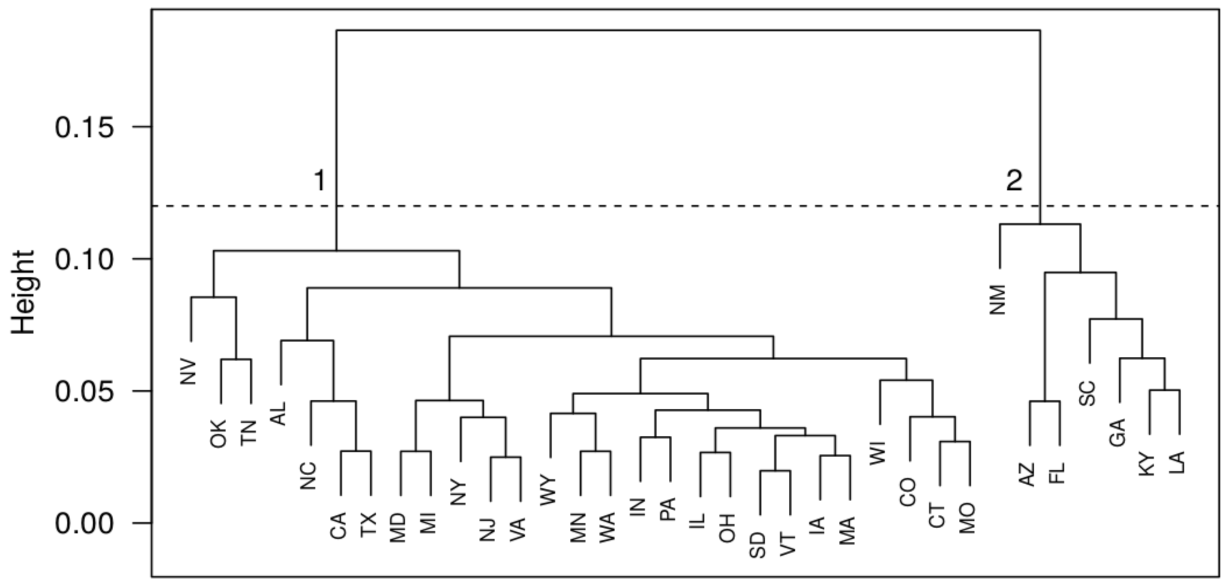

Figure 2.11: Dendrogram of hierarchical clustering for Product B. The horizontal line indicates the cutoff location to split the observations into different clusters. From left to right, the cluster numbers are 1 and 2, respectively.

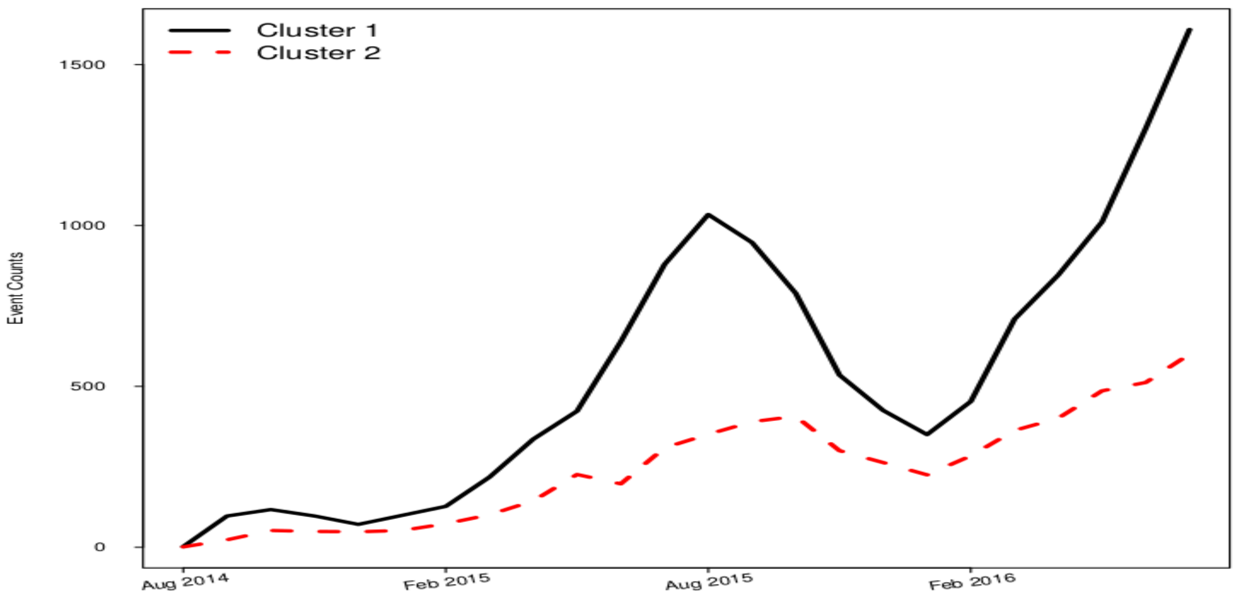

Figure 2.12: Observed event counts as a function of date for the two different clusters of Product B systems.

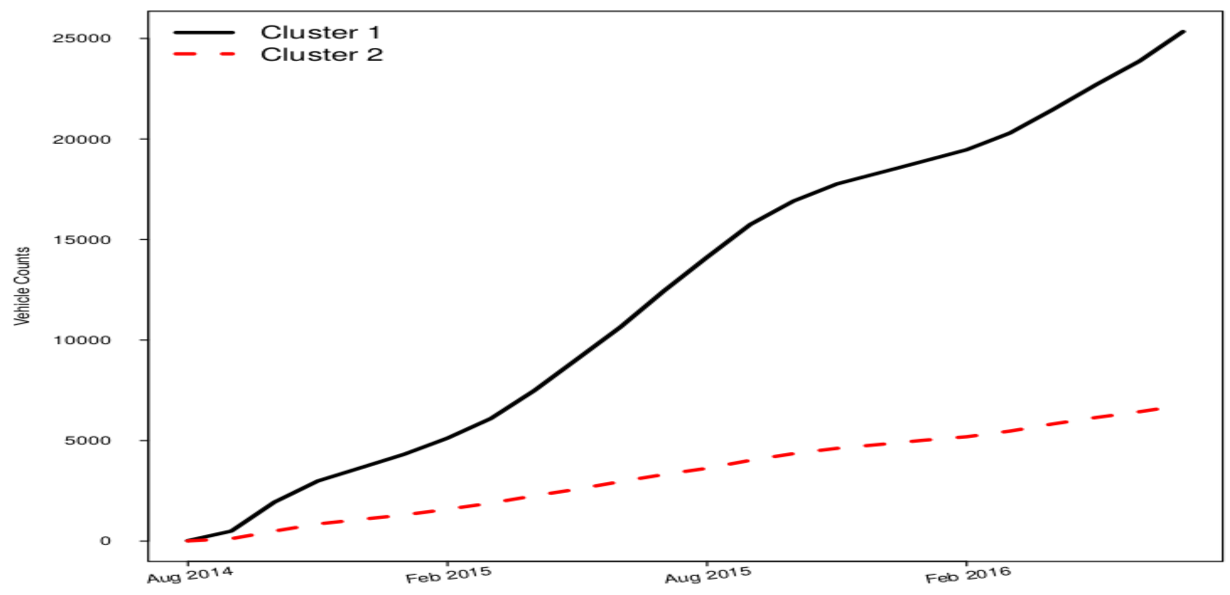

Figure 2.13: Number of systems at risk as a function of date for the two different Product B clusters.

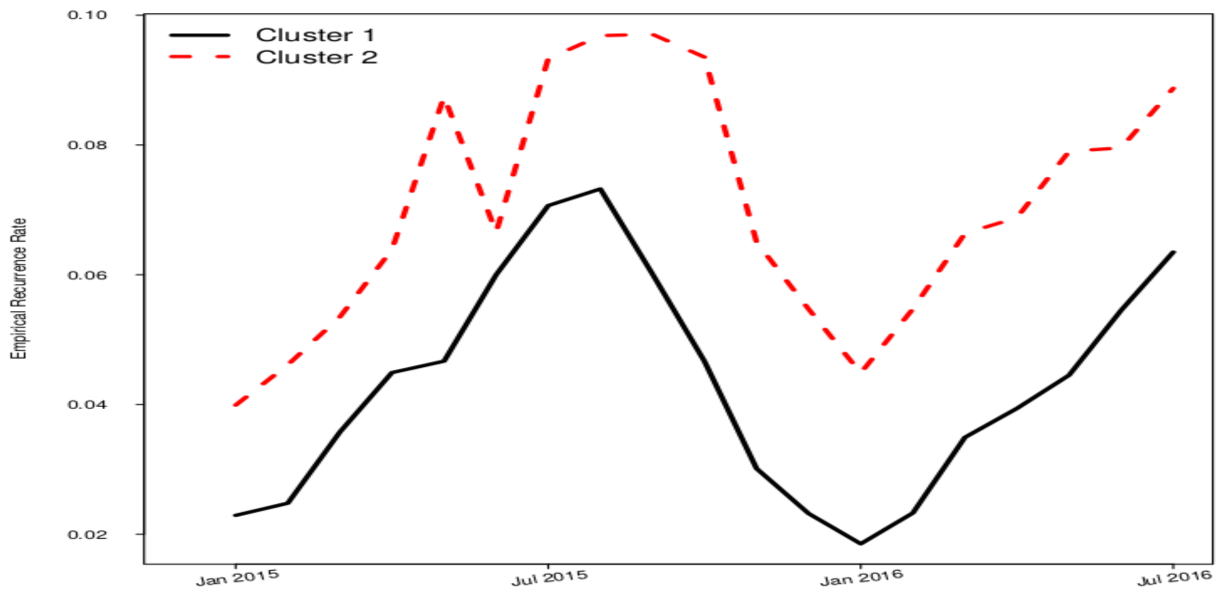

Figure 2.14: Empirical monthly recurrence rate for Product B since year 2015.

Table 2.3: Summary results of different models for the Product B data. The smallest AIC and BIC values are marked in **bold**.

No.	Model	AIC	BIC
1	Simple NHPP	282526	282543
2	NHPP with Common Seasonal Effects	280634	280743
3	NHPP with Model Year and Common Seasonal Effects	280561	280679
4	NHPP with Cluster and Common Seasonal Effects	279646	279764
5	NHPP with Cluster, Common Season and Model Year Effects	279598	279724
6	NHPP with Cluster and Seasonal Interactions	279540	279759
7	NHPP with Cluster and Seasonal Interactions and Model Year Effects	279490	279718
8	NHPP with Common Season and Random Effects	278007	278125
9	NHPP with Common Season, Model Year and Random Effects	279485	279612
10	NHPP with Cluster, Common Season and Random Effects	277410	277537
11	NHPP with Cluster, Common Season, Model Year and Random Effects	277372	277507
12	NHPP with Cluster and Seasonal Interaction and Random Effects	277296	277515
13	**NHPP with Cluster and Seasonal Interactions, Model Year and Random Effects**	**277259**	**277486**

assume that the seasonal patterns in the future behave like the past. Prediction accuracy may suffer if the future seasonal patterns behave in a different manner. Model 9 with the common seasonal covariates, model year effects, and random effects provides the best predictions among all of the models. The model fitting of monthly and cumulative event counts based on Model 9 are shown Figures 2.15 and 2.16, respectively.

2.8.4 Prediction Intervals

Figures 2.17 and 2.18 give the prediction intervals of the monthly events and cumulative events, respectively, while the calibrated prediction intervals are based on $B = 5{,}000$ simulations. Six out of eight observed counts fall within the calibrated prediction intervals for the first eight months of hold-out data. The number of events for three of the last four months, however, falls outside of the prediction intervals. Delayed event reports could be a reason why the monthly predictions are much higher than the observed events for June and July of 2017. The higher number of observed events for April and May could be because more of the product warranties expire in these two

Table 2.4: Summary results of point predictions on different models for the Product B hold-out data. The model with smallest RMSE/MAE/MAPE values is marked in **bold**.

No.	RMSE^6	RMSE^{12}	MAE^6	MAE^{12}	MAPE^6	MAPE^{12}
1	606.5	454.5	547.1	369.7	43.0	31.2
2	226.2	235.1	194.4	197.7	12.1	17.2
3	101.9	194.7	94.0	146.4	6.9	15.3
4	225.1	236.9	193.9	200.2	12.1	17.5
5	121.8	199.0	112.0	155.2	7.9	15.6
6	224.2	236.6	193.1	200.2	12.1	17.6
7	120.4	198.9	110.9	155.2	7.9	15.6
8	210.2	229.1	182.0	192.8	11.5	17.1
9	**84.2**	**192.4**	**76.1**	**137.5**	**6.4**	**15.1**
10	206.5	228.9	179.3	193.1	11.4	17.2
11	120.6	198.1	110.9	154.3	7.9	15.5
12	204.5	227.7	177.4	192.0	11.3	17.1
13	115.3	196.4	106.0	151.9	7.5	15.3

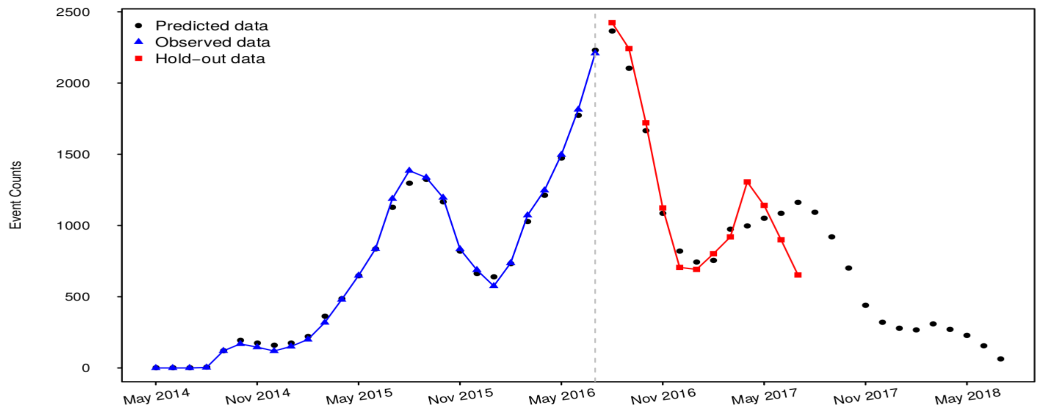

Figure 2.15: Monthly prediction of the event counts for Product B based on Model 9.

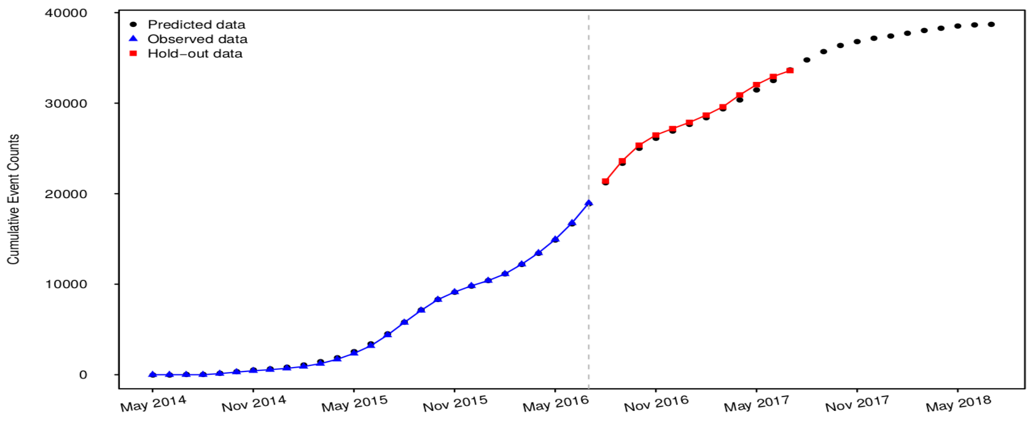

Figure 2.16: Cumulative prediction of the event counts for Product B based on Model 9.

months compared with previous years, which encourages people to use the warranty shortly before the expiration dates.

2.9 Simulation to Study Larger Amounts of Missing Data

In this section, we study the effects of missing data on the prediction performance of new data: we randomly erase $10\% - 30\%$ of location variables from the original data of Product A, and use these data to fit the models based on two different ways to handle missing locations data when clustering:

1. Assign all the missing locations to a new location category.

2. Make random weighted assignment for the missing locations.

In order to compare the prediction metrics such as the RMSE on the hold-out data, we keep using four clusters and compare Model 7 for different missing percentages. Figures 2.19 and 2.20 show the scatter plots of the 6-month and 12-month RMSE values based on the above two ways of handling the missing locations. We experiment with 10%, 15%, 20%, 25% and 30% missing locations. The random assignment methods shown in Section A.3 of the supplemental materials are used to fill the missing locations. We repeated the experiment 20 times by doing both the random erasing and assignment of the location data 20 times. Scatter plots of other prediction

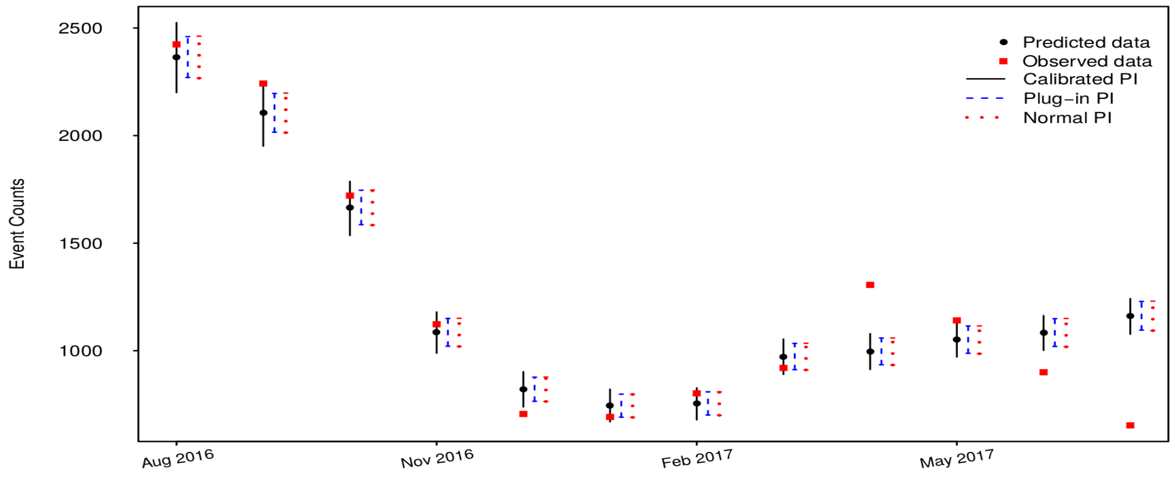

Figure 2.17: Monthly prediction of the event counts on the hold-out data based on Model 9 for Product B.

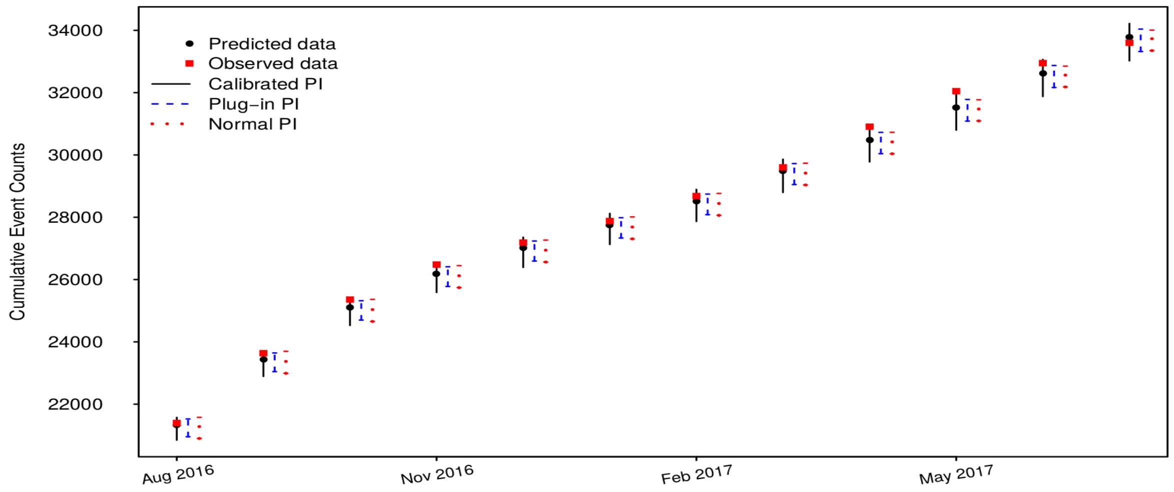

Figure 2.18: Cumulative prediction of the event counts on the hold-out data based on Model 9 for Product B.

metrics such as MAE and MAPE are presented in the supplemental materials in Figures 2.22, 2.23, 2.24, and 2.25. The results of this simulation study can be summarized as:

1. The model using the original data with the 6.6% missing locations assigned to separate categories by their country information has better prediction performance than the models with higher missing percentages.

2. When the missing percentage is relatively small (e.g., less than 15%), the average prediction metrics from two different ways to handle the missing data are close to each other. The random assignment of missing data gives prediction metrics with less variation over the 20 experiments. When the missing percentage is larger, there is no best way to handle the missing data.

3. When the missing percentage increases to around 30%, all of performance metrics have values close to the prediction performance of Model 3 in Section 2.6. An explanation for this is that: when we randomly erase 30% location variables and randomly assign a location to each of the missing values, we introduce noise for the seasonality clustering as the actual seasonality pattern in the data may be corrupted by the random assignment or the separate categories of missing locations. The clustering may not help anymore, and the model will behave more like Model 3, where all repairable systems have the same seasonality pattern.

4. In general, the model with 10% missing has better prediction performance in terms of the 6-month prediction metrics. The differences of 12-month prediction metrics among different missing percentages are relatively small.

5. When the missing percentage increases from 20% to 30%, the prediction performance degrades. For example, the MAPE^{12} values for the 25% missing data are slightly smaller than that of 20%, while the RMSE^{6} values for 25% are larger than that of 20%.

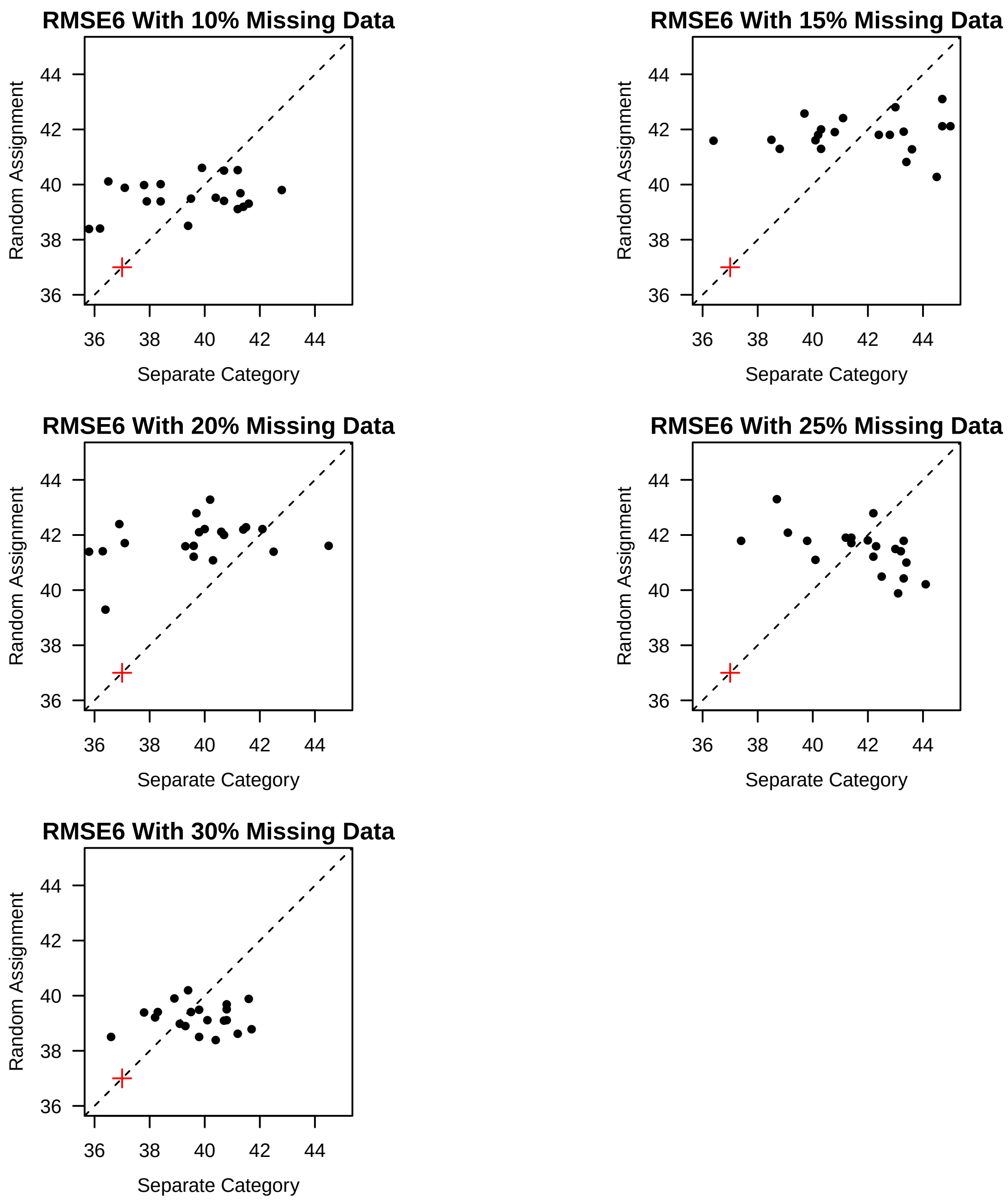

Figure 2.19: Scatter plots of 20 experimental prediction results with weighted random assignment and separate categories for RMSE^6. The cross marks indicate the prediction metrics in Model 7 of Section 2.6.

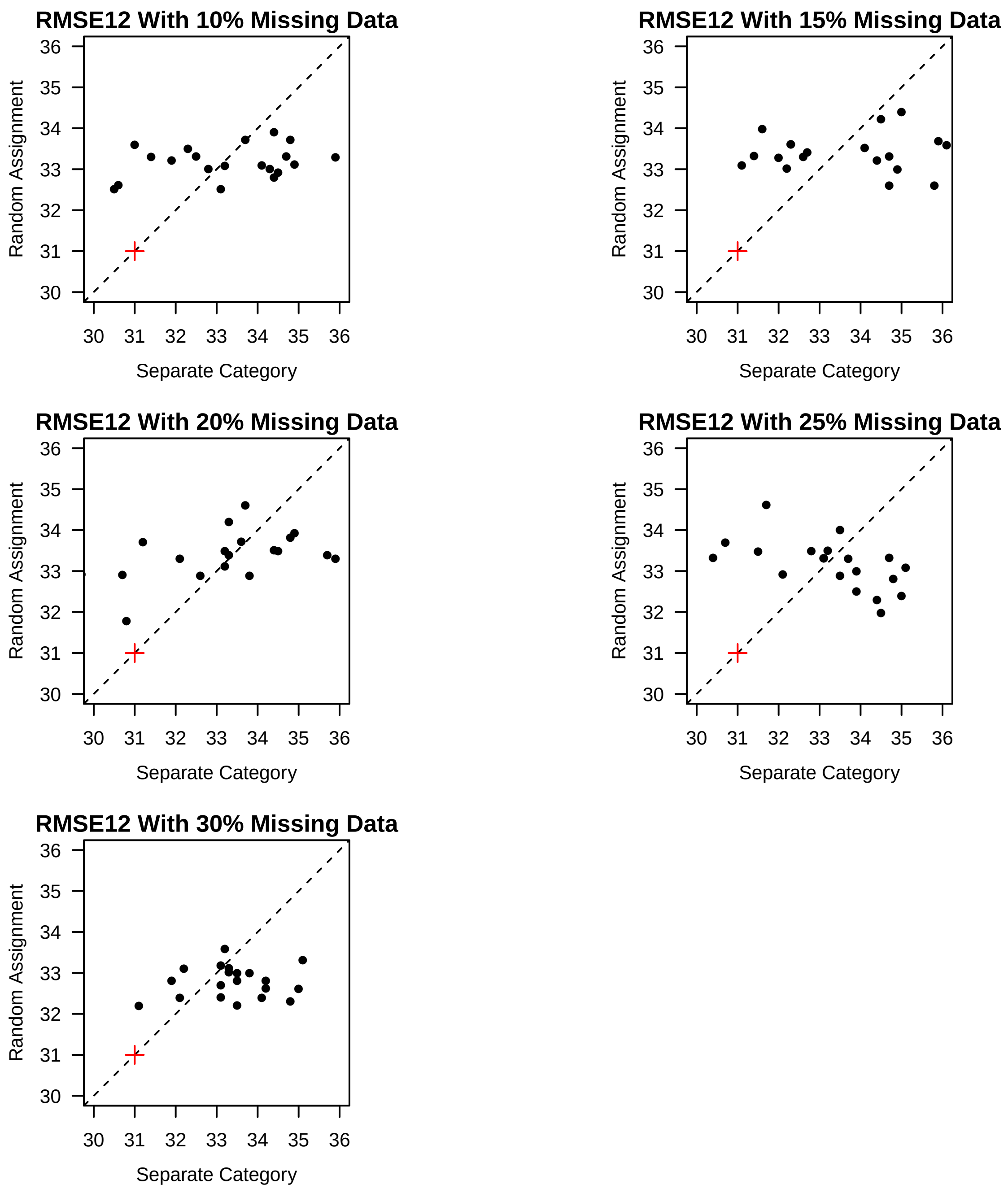

Figure 2.20: Scatter plots of 20 experimental prediction results with weighted random assignment and separate categories for RMSE^{12}. The cross marks indicate the prediction metrics in Model 7 of Section 2.6.

2.10 Concluding Remarks

In this paper, we introduce a general model for the recurrence rate of repairable systems that can be used to predict the number of future events. The model can be applied to various applications depending on the characteristics of the recurrent event processes. Our approach allows the use of covariates that may affect the recurrence rate (for example, the different seasonal trends for different locations), and provides better prediction results than the simple NHPP models. In particular, the use of the cluster and seasonality information improves the predictions of future monthly events for more useful decisions in industry.

Possible extensions of our current work include:

1. In this paper, we model the seasonal trends in the recurrence rate based on the calendar month and we assume implicitly that each month has the same number of business days. The number of business days varies from month to month because of holidays and the number of weekends in the month. Taking the number of business days of each month into the model could lead to more accurate modeling and prediction of events.

2. In some applications, claims are not reported immediately after the system failure. This introduces extra variability in the time-dependent seasonal patterns in the model and can lead to inaccuracies for data near the data-freeze date. Also, there can be spikes of warranty claims near to the end-of-warranty date. Such factors might be included in the prediction model.

3. Our paper focused on the prediction of future events. Sometimes it is important to predict future costs as well. Our model can be extended to a compound mixed Poisson process like that described in Grandell (1997). Marked point processes can also be used for claim cost prediction, as described in Brémaud (1981) and Karyagina et al. (1998). Information about failure modes could be helpful for the future claim cost prediction.

4. Two-dimensional warranty policies are widely used (e.g., in the North American automobile market). For example, the observation of a warranty contract will end when the mileage of

a product reaches 36 thousand miles or three years after the purchase date, whichever comes first. A model based on both system age and usage would be more appropriate. However, usage data are often not complete as the usage information may not be available until there is a claim and some of the systems may not have claims before the end of the warranty. Also, automobiles with claims may not be a representative sample of the entire population. Lawless and Crowder (2010) proposed joint models on the age and usage dimensions for the warranty data for dependence assessment and model parameter estimation. Some work has also been done on using a synthesized scale based on both age and usage as described in Ahn et al. (1998) and Duchesne and Lawless (2000).

5. The investigation of NHPP models with time-varying covariates and random effects under the Bayesian framework would be useful. The hierarchical modeling together with tools such as Markov Chain Monte Carlo (MCMC) can then be used conveniently for estimating model parameters and producing prediction intervals.

Acknowledgments

The authors gratefully acknowledge the help of After, Inc. for supporting and assisting in the work presented in this paper. We appreciate the insightful comments from the editor, associate editor and reviewers very much.

2.11 References

Aalen, O. O. (1988). Heterogeneity in survival analysis. *Statistics in medicine*, 7(11):1121–1137.

Ahn, C.-W., Chae, K.-C., and Clark, G. M. (1998). Estimating parameters of the power law process with two measures of failure time. *Journal of Quality Technology*, 30:127–132.

Beran, R. (1990). Calibrating prediction regions. *Journal of the American Statistical Association*, 85:715–723.

Brémaud, P. (1981). *Point Processes and Queues: Martingale Dynamics*, volume 50. Springer.

Cifuentes-Amado, M. V. and Cepeda-Cuervo, E. (2015). Non-homogeneous Poisson process to model seasonal events: Application to the health diseases. *International Journal of Statistics in Medical Research*, 4:337–346.

Cook, R. J. and Lawless, J. (2007). *The Statistical Analysis of Recurrent Events.* Springer.

Duchesne, T. and Lawless, J. (2000). Alternative time scales and failure time models. *Lifetime Data Analysis*, 6:157–179.

Fonseca, G., Giummole, F., and Vidoni, P. (2014). Calibrating predictive distributions. *Journal of Statistical Computation and Simulation*, 84:373–383.

Fredette, M. and Lawless, J. F. (2007). Finite-horizon prediction of recurrent events, with application to forecasts of warranty claims. *Technometrics*, 49:66–80.

Grandell, J. (1997). *Mixed Poisson Processes*, volume 77. CRC Press.

Hamada, M. S., Wilson, A., Reese, C. S., and Martz, H. (2008). *Bayesian Reliability.* Springer Science & Business Media.

Hartigan, J. A. (1975). *Clustering Algorithms.* Wiley.

Hartigan, J. A. and Wong, M. A. (1979). Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28:100–108.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning.* Springer.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning*, volume 112. Springer.

Kaplan, E. L. and Meier, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53:457–481.

Karyagina, M., Wong, W., and Vlacic, L. (1998). Life cycle cost modeling using marked point processes. *Reliability Engineering & System Safety*, 59:291–298.

Klein, J. P. (1992). Semiparametric estimation of random effects using the cox model based on the EM algorithm. *Biometrics*, 48:795–806.

Koutsellis, T., Mourelatos, Z., Hijawi, M., Guo, H., and Castanier, M. (2017). Warranty forecasting of repairable systems for different production patterns. *SAE International Journal of Materials and Manufacturing*, 10(2017-01-0209):264–273.

Lawless, J. and Fredette, M. (2005). Frequentist prediction intervals and predictive distributions. *Biometrika*, 92:529–542.

Lawless, J. F. (1987). Regression methods for Poisson process data. *Journal of the American Statistical Association*, 82:808–815.

Lawless, J. F. and Crowder, M. J. (2010). Models and estimation for systems with recurrent events and usage processes. *Lifetime Data Analysis*, 16:547–570.

Lawless, J. F. and Nadeau, C. (1995). Some simple robust methods for the analysis of recurrent events. *Technometrics*, 37:158–168.

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137.

Meeker, W. Q. and Escobar, L. A. (1998). *Statistical Methods for Reliability Data.* John Wiley & Sons.

Nelson, W. (1988). Analysis of repair data. In *1988. Proceedings., Annual Reliability and Maintainability Symposium, IEEE.*

Nelson, W. B. (2003). *Recurrent Events Data Analysis for Product Repairs, Disease Recurrences, and Other Applications*, volume 10. SIAM.

Ngailo, T., Shaban, N., Reuder, J., Rutalebwa, E., and Mugume, I. (2016). Non homogeneous Poisson process modelling of seasonal extreme rainfall events in Tanzania. *International Journal of Science and Research (IJSR)*, 5:1858–1868.

Rai, B. K. (2009). Warranty spend forecasting for subsystem failures influenced by calendar month seasonality. *IEEE Transactions on Reliability*, 58:649–657.

Rigdon, S. E. and Basu, A. P. (2000). *Statistical Methods for The Reliability of Repairable Systems.* Wiley.

Ross, S. M. (2014). *Introduction to Probability Models.* Academic Press.

Rousseeuw, P. J. and Kaufman, L. (1990). Finding groups in data. *Series in Probability & Mathematical Statistics*, 34:111–112.

Ryan, K. J., Hamada, M. S., and Reese, C. S. (2011). A Bayesian hierarchical power law process model for multiple repairable systems with an application to supercomputer reliability. *Journal of Quality Technology*, 43:209–223.

Slimacek, V. and Lindqvist, B. (2016). Reliability of wind turbines modeled by a poisson process with covariates, unobserved heterogeneity and seasonality. *Wind energy*, 19:1991–2002.

Teerapabolarn, K. (2014). Poisson approximation for independent negative binomial random variables. *International Journal of Pure and Applied Mathematics*, 93:779–781.

Therneau, T. M., Grambsch, P. M., and Pankratz, V. S. (2003). Penalized survival models and frailty. *Journal of Computational and Graphical Statistics*, 12:156–175.

Wu, S. (2012). Warranty data analysis: A review. *Quality and Reliability Engineering International*, 28:795–805.

Xiao, S., Kottas, A., Sansó, B., et al. (2015). Modeling for seasonal marked point processes: An analysis of evolving hurricane occurrences. *The Annals of Applied Statistics*, 9:353–382.

Xu, Z., Hong, Y., and Meeker, W. Q. (2015). Assessing risk of a serious failure mode based on limited field data. *IEEE Transactions on Reliability*, 64:51–62.

Xu, Z., Hong, Y., Meeker, W. Q., Osborn, B. E., and Illouz, K. (2017). A multi-level trend-renewal process for modeling systems with recurrence data. *Technometrics*, 59:225–236.

2.12 Supplemental Materials for Chapter 2

A.1 **Likelihood Function with Random Effects Term.**
Given that the random effects for the recurrence rate of each system follow independent Gamma$(1/\phi, 1/\phi)$, the integration over u_i for (2.12) is,

$$
\begin{aligned}
&\int \left\{ \left[\prod_{j=1}^{r_i} u_i \nu_{b,i}(t_{ij};\boldsymbol{\beta}) \right]^{\delta_{ij}} \exp\left[-u_i \mu_{b,i}(0, t_{a_i};\boldsymbol{\beta})\right] \right\} \frac{u_i^{1/\phi-1}\exp(-u_i/\phi)}{\phi^{1/\phi}\Gamma(1/\phi)} \mathrm{d}u_i \\
= \quad & \frac{1}{\phi^{1/\phi}\Gamma(1/\phi)} \int \left[\prod_{j=1}^{r_i} u_i \nu_{b,i}(t_{ij};\boldsymbol{\beta}) \right]^{\delta_{ij}} \exp\left[-u_i \mu_{b,i}(0, t_{a_i};\boldsymbol{\beta})\right] u_i^{1/\phi-1}\exp(-u_i/\phi) \mathrm{d}u_i \\
= \quad & \frac{1}{\phi^{1/\phi}\Gamma(1/\phi)} \left[\prod_{j=1}^{r_i} \nu_{b,i}(t_{ij};\boldsymbol{\beta}) \right]^{\delta_{ij}} \int u_i^{r_i+1/\phi-1} \exp\left\{-u_i \left[\mu_{b,i}(0, t_{a_i};\boldsymbol{\beta}) + 1/\phi\right]\right\} \mathrm{d}u_i \\
= \quad & \left[\prod_{j=1}^{r_i} \nu_{b,i}(t_{ij};\boldsymbol{\beta}) \right]^{\delta_{ij}} \frac{\Gamma(\zeta_i)}{\phi^{1/\phi}\Gamma(1/\phi)\kappa_i^{\zeta_i}},
\end{aligned}
$$

where $\zeta_i = r_i + 1/\phi$ and $\kappa_i = \mu_{b,i}(t_{a_i}) + 1/\phi$.

A.2 **Probability Function of Monthly or Cumulative Event Counts**

Suppose a random variable $Y_i(t)$ is the number of events to be predicted at calendar time t, and it can be re-written as $N_i(t_1, t_2)$, where (t_1, t_2) is the time period in terms of system age. Because the random effect u_i has a Gamma distribution with density given in (2.11), the probability mass function for $N_i(t_1, t_2)$ is,

$$
\begin{aligned}
& \Pr[N_i(t_1, t_2) = n | \text{DATA}, \boldsymbol{\theta}] \\
= \quad & \int_0^\infty P(N_i(t_1, t_2) = n | u_i) g(u_i | \text{DATA}, \phi) \mathrm{d}u_i \\
= \quad & \int_0^\infty \frac{\exp[-u_i \mu_{b,i}(t_1, t_2;\boldsymbol{\beta})] \left[u_i \mu_{b,i}(t_1, t_2;\boldsymbol{\beta})\right]^n}{n!} \frac{\kappa_i^{\zeta_i}}{\Gamma(\zeta_i)} u_i^{\zeta_i - 1} \exp(-u_i \kappa_i) \mathrm{d}u_i \\
= \quad & \frac{\kappa_i^{\zeta_i} \left[\mu_{b,i}(t_1, t_2;\boldsymbol{\beta})\right]^n}{\Gamma(\zeta_i) n!} \int_0^\infty u_i^{n+\zeta_i-1} \exp\left[-u_i \mu_{b,i}(t_1, t_2;\boldsymbol{\beta}) + \kappa_i\right] \mathrm{d}u_i \\
= \quad & \frac{\kappa_i^{\zeta_i} \left[\mu_{b,i}(t_1, t_2;\boldsymbol{\beta})\right]^n}{\Gamma(\zeta_i) n!} \frac{\Gamma(n+\zeta_i)}{\left[\mu_{b,i}(t_1, t_2;\boldsymbol{\beta}) + \kappa_i\right]^{n+\zeta_i}} \\
= \quad & \frac{\Gamma(n+\zeta_i)}{\Gamma(\zeta_i) n!} \left[\frac{\mu_{b,i}(t_1, t_2;\boldsymbol{\beta})}{\mu_{b,i}(t_1, t_2;\boldsymbol{\beta}) + \kappa_i} \right]^n \left[\frac{\kappa_i}{\mu_{b,i}(t_1, t_2;\boldsymbol{\beta}) + \kappa_i} \right]^{\zeta_i},
\end{aligned}
\tag{2.24}
$$

which is a negative binomial distribution. Also, when $\phi = 0$, the probability function reduces to a Poisson distribution with mean $\mu_{b,i}(t_1, t_2;\boldsymbol{\beta})$.

A.3 **Random Assignment of Missing Locations for Product A Clustering.**

For each repairable system i with missing location variable, we randomly assign a location to it with the following procedure.

(a) Find all systems that have non-missing locations that have the same effective month as system i. Denote the systems set as $\{S\}$.

(b) Compute the percentage of each distinct location for systems in $\{S\}$.

(c) Do a simple moving average smoothing of order 5 for the above percentages of each location with percentages computed from the previous and later months.

(d) Apply random weighted sampling of the locations with the above percentages as weights and assign the sampled location to system i.

The prediction results of Model 7 based on random assignment of missing locations were repeated 20 times. The results are shown in Table 2.5. Figure 2.21 shows the histogram of all 20 experiments, and their comparisons with the prediction results by grouping the missing location values into new categories. It can be seen that the latter shows consistently better prediction results.

A.4 **Scatter plots of MAE and MAPE for the simulation study on missing data in Section 2.9.**

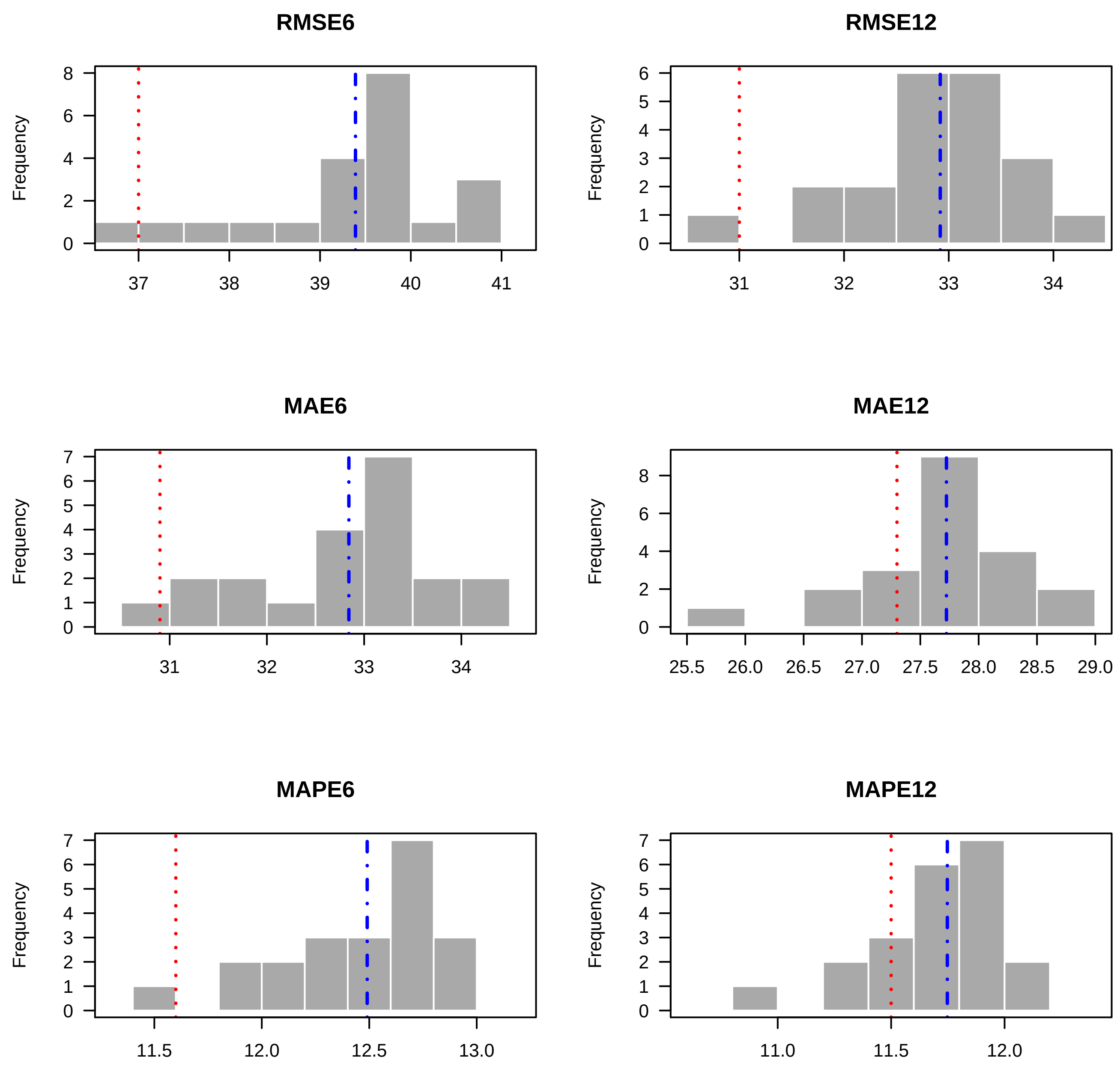

Figure 2.21: Histogram of 20 experimental prediction results with weighted random assignment of missing locations (Dotted lines shows the corresponding prediction values by grouping the missing locations into new categories, and dashed lines show the mean prediction values of the 20 experiments based on weighted random assignment for missing locations).

Table 2.5: Summary results of point predictions on Model 7 for the Product A hold-out data with 20 repeated times of random assignment experiment. The superscript 6 and 12 indicate the RMSE/MAE/MAPE of the first 6 and 12 months, respectively.

Repeat No.	RMSE^6	RMSE^{12}	MAE^6	MAE^{12}	MAPE^6	MAPE^{12}
1	38.8	32.4	32.0	27.1	12.1	11.5
2	39.4	32.9	32.5	27.4	12.3	11.6
3	37.6	31.9	31.2	27.2	12.0	11.6
4	37.3	31.7	31.2	26.8	11.9	11.4
5	39.8	32.8	34.0	28.0	12.9	11.8
6	39.6	33.2	33.1	28.0	12.6	11.9
7	40.3	33.8	33.3	28.3	12.7	12.0
8	39.9	33.2	33.2	27.9	12.7	11.9
9	39.4	33.0	32.9	27.9	12.6	11.9
10	39.7	33.2	33.2	28.1	12.7	12.0
11	39.2	33.1	32.7	27.9	12.4	11.8
12	39.6	33.0	32.7	27.6	12.4	11.7
13	38.4	32.2	31.9	27.0	12.1	11.4
14	36.9	30.7	30.6	25.7	11.5	10.8
15	41.0	34.2	34.4	28.8	13.0	12.1
16	40.6	33.7	33.5	28.0	12.7	11.8
17	39.9	33.3	33.3	28.1	12.7	11.9
18	40.8	33.9	34.2	28.7	13.0	12.2
19	39.9	33.4	33.3	28.1	12.7	11.9
20	39.7	32.8	33.7	27.9	12.8	11.8
Average	39.4	32.9	32.8	27.7	12.5	11.7

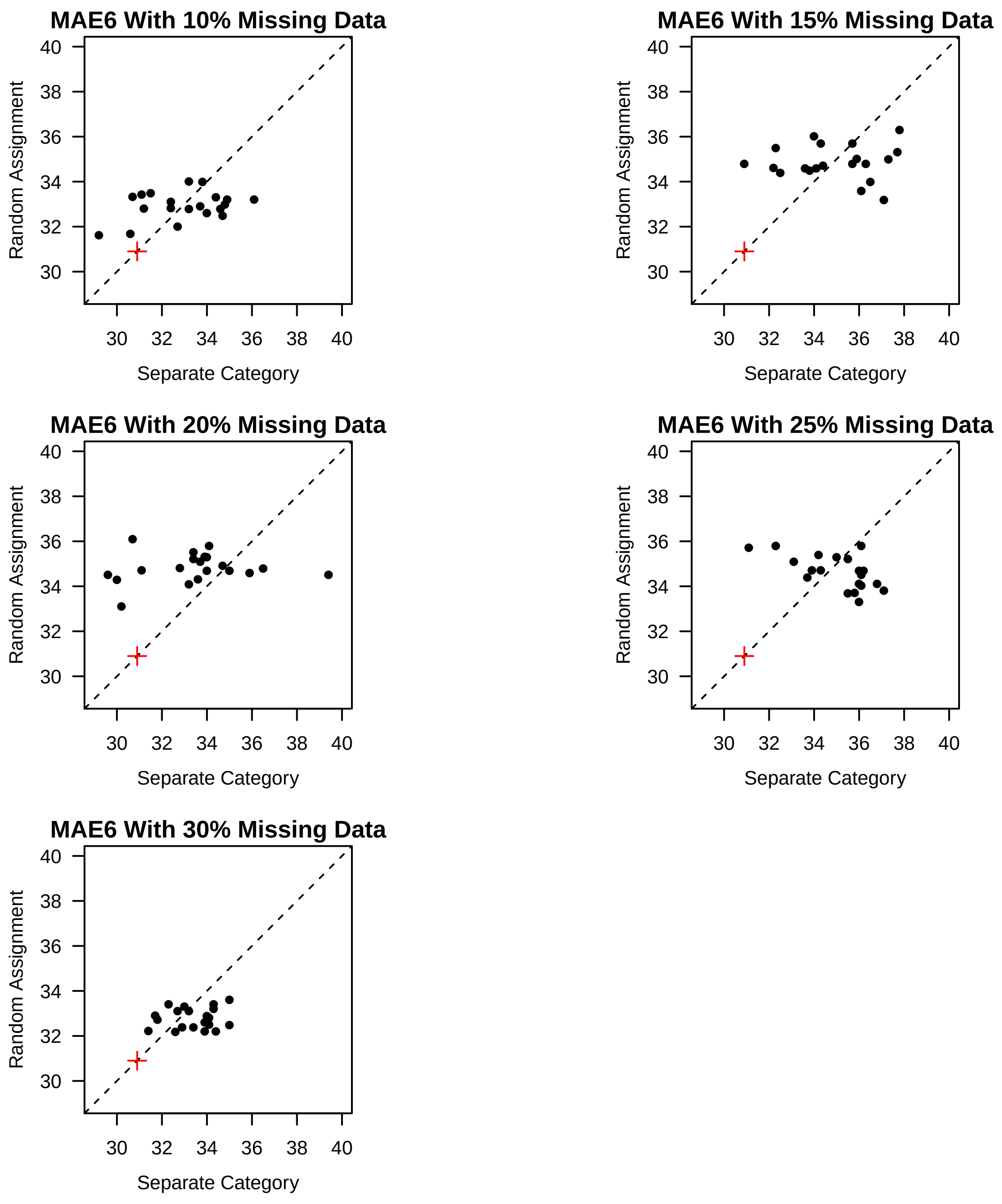

Figure 2.22: Scatter plots of 20 experimental prediction results with weighted random assignment and separate categories for MAE[6]. The cross marks indicate the prediction metrics in Model 7 of Section 2.6.

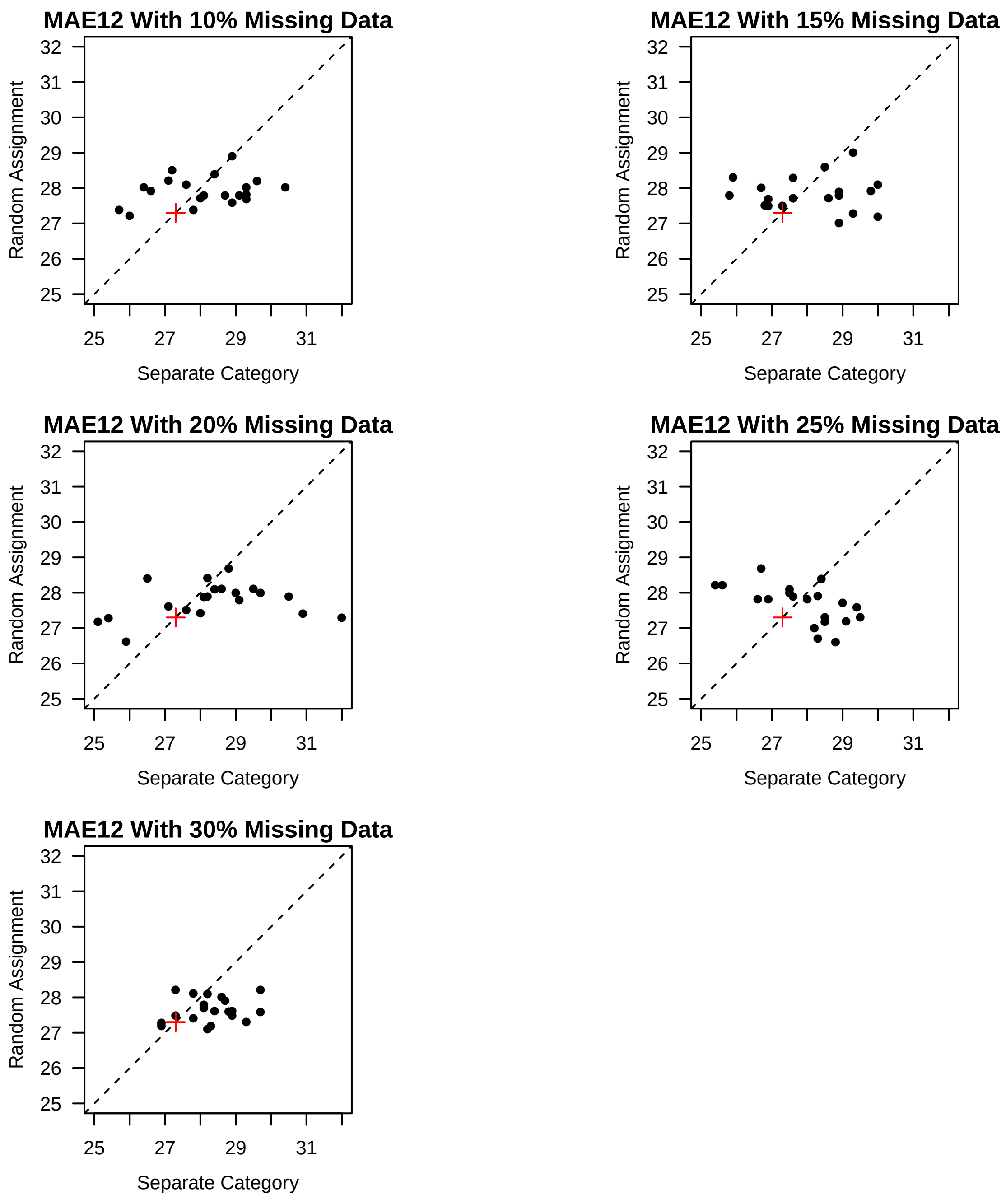

Figure 2.23: Scatter plots of 20 experimental prediction results with weighted random assignment and separate categories for MAE[12]. The cross marks indicate the prediction metrics in Model 7 of Section 2.6.

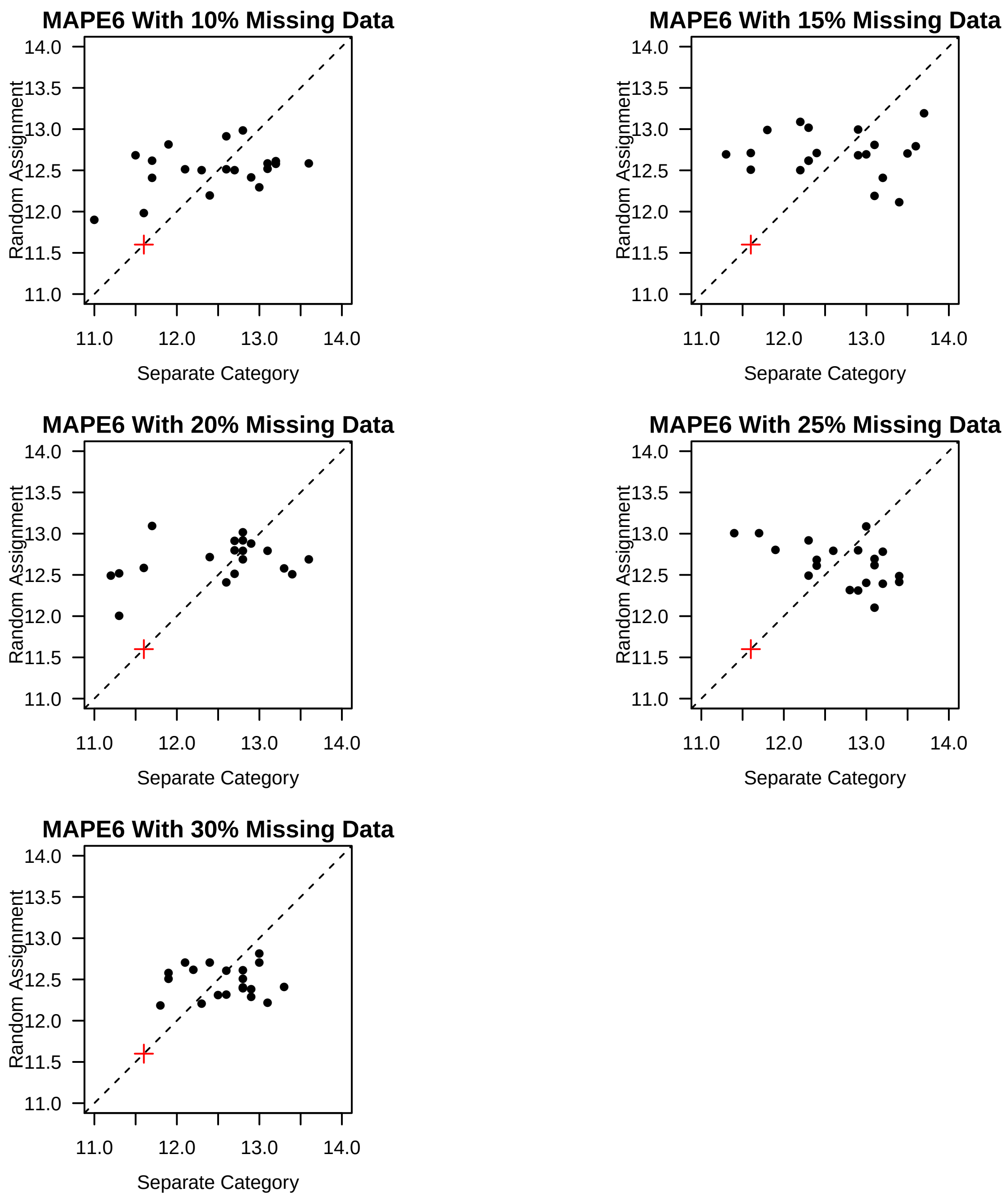

Figure 2.24: Scatter plots of 20 experimental prediction results with weighted random assignment and separate categories for MAPE^6. The cross marks indicate the prediction metrics in Model 7 of Section 2.6.

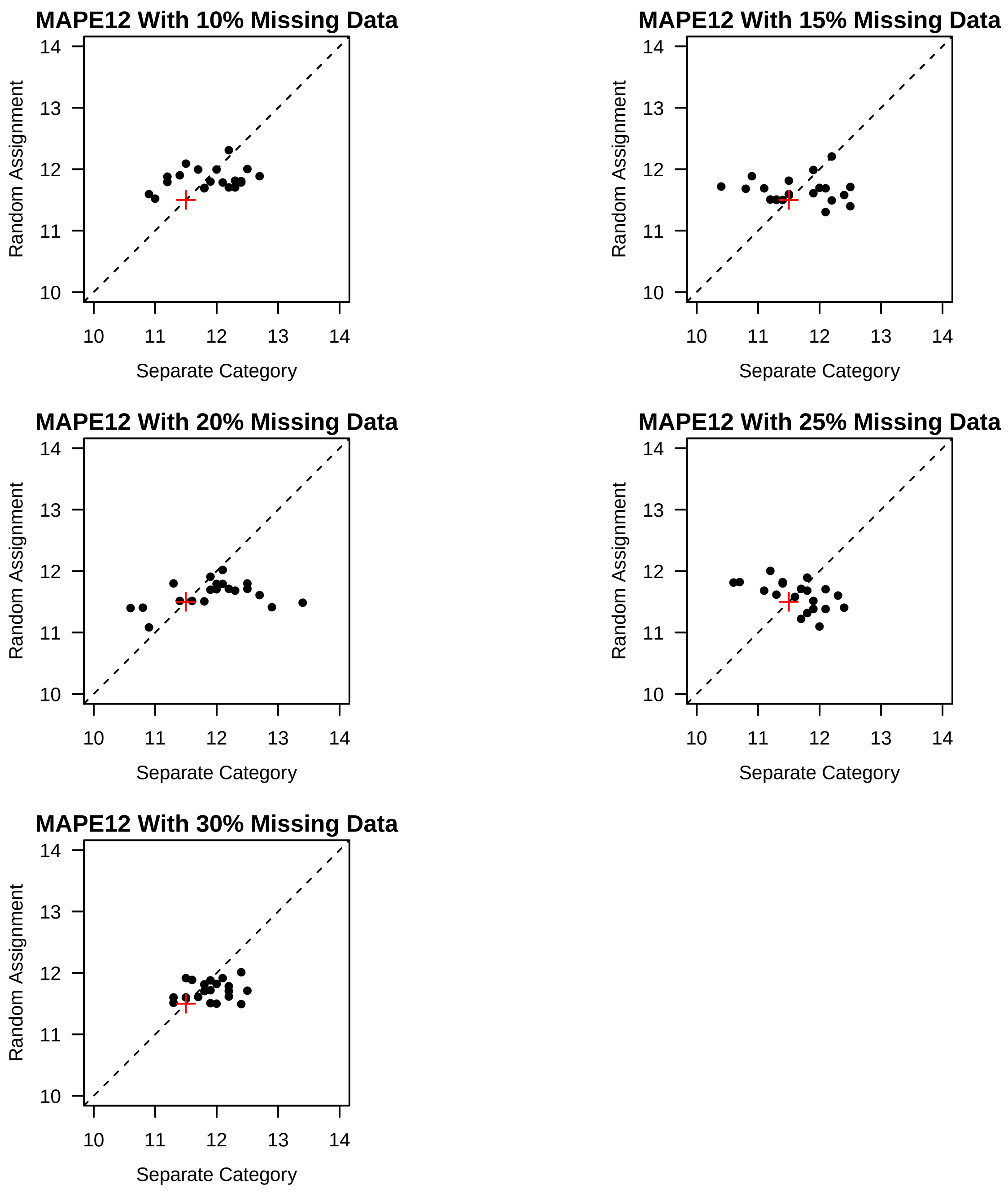

Figure 2.25: Scatter plots of 20 experimental prediction results with weighted random assignment and separate categories for MAPE^{12}. The cross marks indicate the prediction metrics in Model 7 of Section 2.6.

CHAPTER 3. ELECTRONIC CIRCUIT TROUBLESHOOTING USING SIMULATION AND BAYESIAN INFERENCE

Qianqian Shan, Stephen D. Holland and William Q. Meeker

3.1 Abstract

Electronics is an integral part of almost all industrial systems such as computers that control the systems and sensors that monitor and control systems. Electronic circuit component characteristics exhibit manufacturing variability just like mechanical parts, and this variability can lead to electronic circuit malfunctions or failures. The repair history for a particular kind of equipment may not be able to suggest which specific component(s) will fix a particular problem and costs could go up if we replace more components then needed. This paper focuses on developing an algorithm to automate electronic circuit troubleshooting. With the help of electronic circuit simulator, ngspice, we propose to apply a Bayesian approach with adaptive Markov Chain Monte Carlo (MCMC) to estimate the circuit system component values and to diagnose malfunctions in electronic systems. We demonstrate the methods using two simulated electronic circuits. The results show that the failed components of electronic circuits can be accurately and efficiently isolated.

Keywords: Adaptive MCMC, Bayesian inference,circuit simulation, circuit troubleshooting, forward model, inverse model.

3.2 Introduction

3.2.1 Background and Motivation

Electronic circuits are a crucial part of many industrial systems, and the troubleshooting a malfunctioned electronic circuit can be very useful to keep the systems working and save costs. Electronic circuit designs are based on nominal values of component parameters. Tolerances stack up and variability in physical properties of the components can have a major effect on the reliability of electronics. There has been much work on the use of simulation during the circuit design process. For example, Tuinenga (1991) introduces how to do electronic circuit simulation and analysis such as tolerancing problems with Pspice. However, little has been done to apply simulation-based methods to help troubleshoot electronic systems. Automation of assembly labor has been extensive, and automatic test equipment on circuit failure detection is used by almost all manufactures. What is missing is the automation of the circuit troubleshooting and repair process. Tests upon discovery of a problem will not be able to identify the exact component problem because many failures, especially in analog circuits, are not hard failures, but only out-of-specification failures. While hard failures are relatively easy to identify, soft failures, where one or more unusual parameter values combine to make the circuit malfunction are more difficult to locate. The repair history for a particular kind of equipment may suggest to a technician that replacing four specific components will usually fix the problem. If it is only one of the four components that is out of specification, then three good components are either thrown away or damaged by the removal and cost goes up. Across manufacturing, billions of dollars are spent to fix such problems and the dollars can be saved by having a more specific, faster, and less costly solutions for circuit troubleshooting.

3.2.2 Model-based Troubleshooting

Electronic component characteristics exhibit manufacturing variability just like mechanical parts, and this variability can lead to malfunctions. Moreover, relative variation can be quite large and parameter values can shift over time as the part degrades. The variation in electronic part properties comes from tolerances in photolithographic masking, layer thickness, presence of contaminants, and other variables, leading to variations in electronic parameters and part performance. Oftentimes, these variations are hidden through the use of feedback loops (e.g, negative feedback loops in an operational amplifier), but even so, extreme variation can exceed the ability of the feedback system to compensate.

With performance dependent on a large number of interacting variables and because electronic systems generally do not provide visual, tactile feedback like mechanical systems, the troubleshooting process that determines the root cause and feeds the knowledge back to the designers and component suppliers can be difficult. The process can be simplified by taking advantage of the circuit-simulation models to investigate the likely causes of measured symptoms from a failed circuit. For example, ngspice can be used to simulate a circuit's characteristics for given inputs of specified component parameter values.

We propose to apply adaptive MCMC to track and update estimates of the circuit components, and to diagnose malfunctions in electronic circuits. Given measured voltages of a failed circuit as observations, MCMC will help to detect which component parameters are out of specification, and thus help to identify the components that cause failures. For example, this process could be used to estimate the gains of several critical transistors, and determine which one or ones might have malfunctioned based on how the estimated values compare with their nominal values.

We assume weakly informative prior knowledge of a failed circuit, and the posterior distribution of each circuit component value is obtained for more accurate troubleshooting.

3.2.3 Literature Review

Bandler and Salama (1985) describes the theory and algorithms of four fault location approaches of analog circuits: the fault dictionary approach, the parameter identification approach, the fault verification approach and the approximation approach. Tadeusiewicz et al. (2002) proposes an approach using the linear programming and the simplex method for soft-fault diagnosis of linear and nonlinear circuits, and the approach assumes that any circuit element is specified by its nominal value and a tolerance. Alippi et al. (2002) introduces a sensitivity approach to select the most effective testing nodes and inputs. Aminian and Aminian (2001) provides analog circuit fault diagnosis methods with Bayesian neural networks and preprocessors such as wavelet transform and normalization, and assumes that the fault modes are known. Vasan et al. (2013) develops methods for detection of circuit fault conditions, fault troubleshooting, and prediction of remaining useful performance of analog circuits, and a one-against-one (OAO) multiclass least squares support vector machine (LS-SVM) algorithm for the training of a classifier. Other literature such as Cui and Wang (2011) and Vasan et al. (2014) also discuss circuit fault analysis with SVM.

In this paper, we propose a general methodology for electronic circuit troubleshooting, especially for the following cases:

- When there is not enough data to develop data-driven models.
- When there are multiple unknown failure modes.
- When the purpose is to isolate the malfunctioned components as accurately as possible (i.e., to identify the values of all component parameters for a failed circuit device).

3.2.4 Overview

The remainder of the paper is organized as follows. Section 3.3 describes the an electronic circuit simulation tool, ngspice and a related R package, RSpice. Section 3.4 provides the general ideas of electronic circuit troubleshooting with the help of simulated circuits. Section 3.5 describes the ideas in Section 3.4 in more detail with an illustrative power amplifier example. Section 3.6 presents the adaptive MCMC algorithm to solve the Bayesian inverse problem for troubleshooting. Section 3.7 applies the algorithm to another more complicated high dimensional application. Section 3.8 gives some concluding remarks.

3.3 Electronic Circuit Simulation and Ngspice

Ngspice is a mixed-level/mixed-signal electronic circuit simulator developed from three open source software packages: Spice3f5, Cider1b1, and Xspice, where SPICE is the acronym of Simulation Program with Integrated Circuit Emphasis. Ngspice can conveniently simulate different kinds of electronic circuits and provide voltage or current values at any specified circuit node. Circuit simulation is generally used to explore the properties of circuits as a function of circuit component values. In our application, we use ngspice in conjunction with statistical methods to make inference on the circuit component values which may be hard to measure, based on measured values of a circuit such as voltage observations.

R is a language providing a highly extensible environment for statistical computing and graphics. RSpice is an R package that makes it possible to run ngspice seamlessly from R via a shared library (or a dynamic linked library for a Windows operating system). The combination of ngspice and R can greatly simplify the simulation and analysis of electronic circuits in applications where statistical methods are needed.

3.4 Electronic Circuit Troubleshooting: The Basic Ideas

The aim of our approach is to estimate the parameter values of the hidden (not directly observable) parameters of an electronic circuit given its malfunction symptoms. We solve this inverse problem with the help of the forward model and a virtual population of the electronic circuit devices.

3.4.1 Forward Model

The forward model explores outputs of a given circuit with specified components values (e.g., transistor gains). With the help of the circuit simulation models, one can investigate the behaviors of a circuit that has variation in its component values. The behaviors may include the circuit voltages or currents at circuit nodes, the correlations among component values and so on. The forward model can help us to learn about the internal properties of the circuits and obtain insights about how the component's parameters affect the overall behavior of the circuit.

3.4.2 Inverse Problem

Troubleshooting diagnosis is done by estimating the posterior distributions of the failure causal factors (e.g., out of specification components) given the measured symptoms (e.g., voltages at test points) of a failed circuit. That is, voltage measurements at selected circuit nodes are used as observed data. Inferences about circuit component values are then performed to identify malfunctioned components of the failed circuit.

3.4.3 A Virtual Population of Devices

We use a simulated virtual population of circuit devices to mimic an actual population. We define a circuit device as failed when its output voltage is more than 5% different from the output voltage when all component values are equal to their nominal values.

3.5 Illustrative Application: Class A Power Amplifier

We illustrate data preprocessing and analysis with a simple single-stage Class A power amplifier.

3.5.1 Class A Power Amplifier Background

An amplifier takes a signal (usually small) as input and amplifies the signal as output. The class A amplifier features a full 360° of the input signal (Boylestad and Nashelsky (2012)). Figure 3.1 shows the circuit schematic of the amplifier, where $R1$ and $R2$ are biasing resistors, RL is the load resistor, RE is the emitter resistor, and $Q1$ is a type 2N3055 NPN transistor. For the purpose of demonstration, we assume that the failure of the amplifier is caused by only low transistor gains of $Q1$ or high resistance of RL.

3.5.2 Virtual Population of Devices and Failure Definition

To illustrate our troubleshooting methodology, we use a simulated population of virtual class A amplifier devices to mimic an actual population of such devices. An amplifier is considered as failed if the amplitude of its output voltage (voltage across resistor RL) is more than 5% different from its nominal value. We built 1,000 simulated power amplifier devices with a nominal theoretical circuit output gain of 10. Even with modern manufacturing tolerances, transistor gains can be highly variable. The change of the load resistance can also affect the output voltage directly. For simplicity, we restrict our investigation of failed devices to these two components but could be

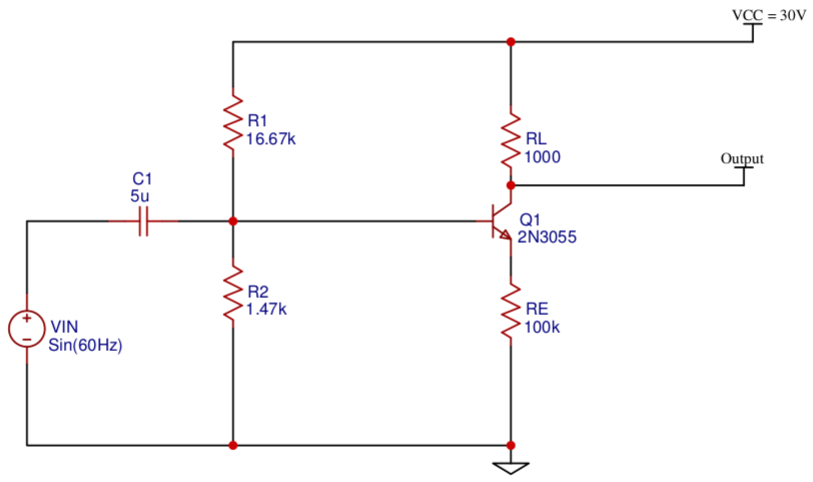

Figure 3.1: Schematic of a class-A amplifier.

easily extended to all circuit components. We assume that the distributions of the RL resistance and $Q1$ transistor gain are independent and follow a mixture of two truncated normal distributions (truncated because both the RL resistance and transistor $Q1$ gain must be non-negative).

- **Sub-population of non-defective components**: Component values have truncated normal distributions. The mean of the corresponding untruncated normal distributions of non-defective components, μ_{ND}, is set to equal nominal component values and the standard deviation, σ_{ND}, is determined such that 99.99% of the component values are from a population is in the interval $[0.9\mu_{ND}, 1.1\mu_{ND}]$.

- **Sub-population of malfunctioned (defective) components**: Component values also have truncated normal distributions. The mean of the corresponding untruncated normal distribution, μ_D, and standard deviation, σ_D, are based on physical knowledge of the components in the defective devices.

- The overall population of component values is a mixture of the non-defective and defective devices with the weights chosen to control the proportion of devices that would be defective in practice. Denote a vector of parameter values as $\boldsymbol{\theta} = (\theta_1, \theta_2)$, then the cdf of θ_i is

$$F(\theta_i; \mu_{D,i}, \sigma_{D,i}, \mu_{ND,i}, \sigma_{ND,i}) = pF(\theta_i; \mu_{D,i}, \sigma_{D,i}) + (1-p)F(\theta_i; \mu_{ND,i}, \sigma_{ND,i}), \tag{3.1}$$

where p is the proportion of defective devices, $F(\boldsymbol{\theta}; \mu_{D,i}, \sigma_{D,i})$ and $F(\boldsymbol{\theta}; \mu_{ND,i}, \sigma_{ND,i})$ are cdfs of components parameter i from defective and non-defective devices, respectively. For example, suppose that there are 1% of 1000 devices are defective, then $p = 0.01$. Table 3.1 shows the values of other parameters.

Table 3.1: Parameter values of the mixture of truncated normal distributions for class A amplifier.

Component Number	Name	μ_{ND}	σ_{ND}	μ_D	σ_D
1	$RL(\Omega)$	1000	10	1100	10
2	$Q1$	129	5.49	7	5

3.5.3 Data Pre-processing

Because the simulated results from ngspice are based on the circuit parameter specification, and the solutions of a set of equations based on physical laws (for example, equations based on Kirchhoff's Laws at each node) provide deterministic voltage values, $\mathbf{y}^*$, with,

$$\mathbf{y}^* = \boldsymbol{g}\left(\boldsymbol{\theta}\right), \tag{3.2}$$

where $\mathbf{y}^*$ is a vector of raw voltage values from ngspice at the specified circuit nodes, $\boldsymbol{g}(\cdot)$ is the function computed by ngspice for a given vector of parameter values in $\boldsymbol{\theta}$. Our goal is to estimate the joint posterior distribution of $\boldsymbol{\theta}$.

We can obtain voltage values for all nodes from ngspice. There may, however, be voltages at certain nodes that are insensitive to the change of component parameters. These nodes have the same voltage values for all the population devices up to the third decimal, (i.e., the differences of voltages at a node for all devices in a population would not be detectable if they were measured by a real voltmeter). As we are trying to mimic the real measurement process and no information can be obtained from these nodes, their voltages need not be obtained. Although currents can also be computed from ngspice at given nodes, we do not to use them because currents are harder to measure in practice. Suppose the data consist of J voltage values from J circuit nodes as our raw "observations", that is, $\mathbf{y}_i^* = (y_{i,1}^*, \cdots, y_{i,J}^*)^T$ represents J raw voltage values obtained from ngspice for device i. The preprocessing of the raw voltage values is performed as follows,

- **Rounding**: Ngspice returns float voltage/current values with more than 8 decimal places. With a real voltmeter, however, one can usually measure up to three digits of precision. In order to have the simulated data mimic the real data as much as possible, we round the J selected raw voltage values for each device from ngspice to three significant digits.

- **Standardization**: When characterizing the population with ngspice, there are 1,000 voltage values for each node from the 1,000 devices, however, the ranges for these values for different test points vary. For example, the 1,000 values ranges from 0.221 to 0.342 for one node, while another node has values ranging from 1.0 to 9.0. Moreover, the distributions of the 1000 voltage values for different nodes also vary: some distributions are symmetric while others are either left or right skewed. We propose a robust standardization method to re-scale the voltages so that all standardized "observations" will have approximately a variance of 1 and median of 0:

  For $j = 1, \cdots J$,

  1. With the 1,000 voltage values at node j from the population, $(\mathrm{y}_{1,j}^*, \cdots, \mathrm{y}_{1000,j}^*)$, find median m_j and interquartile range IQR_j.
  2. Compute the standardized voltage values for node j of device i as $\mathrm{y}_{i,j} = [\mathrm{y}_{i,j}^* - m_j]/\mathrm{IQR}_j$.

3.5.4 Forward Model

The behaviors of voltages at any of the circuit nodes as a function of component values can be studied with ngspice. In the class A power amplifier example, by using different values of resistance RL and transistor gains of $Q1$ as inputs, ngspice will return the corresponding voltage values, which can be used to compute the overall circuit gain (the ratio of output and input). If we only change one component value within a certain range and keep the others fixed at their nominal values, the effect of this specific component on output voltage can be studied. Similarly, if we

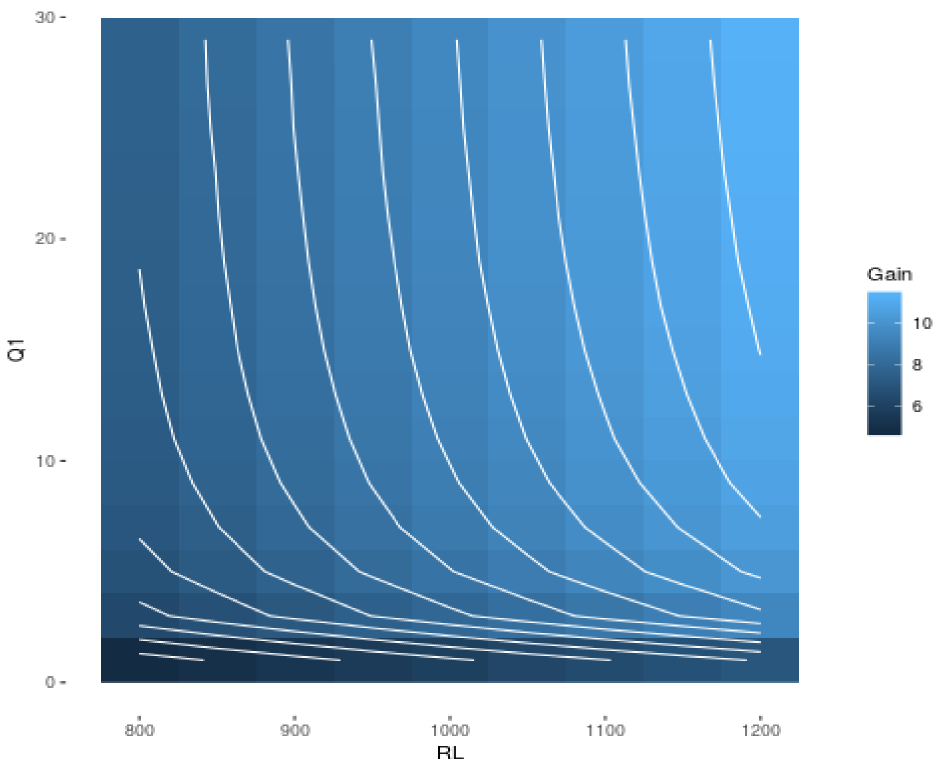

Figure 3.2: Class-A power amplifier gain as a function of resistance of RL and gain of transistor $Q1$.

change the values of the two components systematically and keep the others fixed at nominal, we can obtain the contour plots of output voltages (amplitude) for these two components. These kinds of explorations provide information about the forward model. Figure 3.2 shows contours of circuit gains as a function of RL and $Q1$. The plot indicates that circuit gains are sensitive to low $Q1$ and RL values. Similar contour plots can be done with different ranges of $Q1$ and RL for a detailed exploration of how the component values affect the behavior of the circuits.

3.5.5 Likelihood and Weakly Informative Prior Distribution

Prior distribution: A weakly informative prior distribution on the component values, $p(\boldsymbol{\theta})$ with support Θ, is used. By using a weakly informative prior (e.g., a heavy tailed t-distribution), we place more weight on more reasonable component values while affecting the information in likelihood as little as possible (Gelman et al. (2017)).

Likelihood function: Let $h(\boldsymbol{\theta})$ denote the standardized voltage data following steps in Section 3.5.3 with component values $\boldsymbol{\theta}$. Denote $\mathbf{y}$ as the standardized "observations" from a defective device,

$$\mathbf{y} = h(\boldsymbol{\theta}) + \boldsymbol{\epsilon}, \tag{3.3}$$

where $\boldsymbol{\epsilon} = (\epsilon_1, \cdots, \epsilon_j, \cdots, \epsilon_J)^T$ with $\epsilon_j \overset{iid}{\sim} N\left(0, \sigma^2_{\epsilon_j}\right)$ is the error term for the observations. The error term arises from the measurement variability and rounding to simulate field observations. In the ngspice simulation, the error term arises from a combination of rounding and convergence accuracy of ngspice computations. The likelihood function for the observations from one device is,

$$p(\mathbf{y}|\boldsymbol{\theta}) = p(\mathbf{y}|h(\boldsymbol{\theta})) = \phi_{\text{nor}}\left[\frac{\mathbf{y} - h(\boldsymbol{\theta})}{\boldsymbol{\sigma}}\right], \tag{3.4}$$

where $\phi_{\text{nor}}(\cdot)$ is the probability density function (pdf) of the standard normal distribution.

3.5.6 Posterior Distribution

With the above prior and likelihood functions, the posterior likelihood function can be described according to Bayes theorem as,

$$p(\boldsymbol{\theta}|\mathbf{y}) = \frac{p(\mathbf{y}|\boldsymbol{\theta}) \cdot p(\boldsymbol{\theta})}{\int_\Theta p(\mathbf{y}|\boldsymbol{\theta}) \cdot p(\boldsymbol{\theta}) \mathrm{d}(\boldsymbol{\theta})}, \tag{3.5}$$

where the denominator is a normalizing constant. Then the posterior distribution can also be expressed as,

$$p(\boldsymbol{\theta}|\mathbf{y}) \propto p(\mathbf{y}|\boldsymbol{\theta}) \cdot p(\boldsymbol{\theta}). \tag{3.6}$$

The corresponding log-posterior for each device with support Θ after substituting $p(\boldsymbol{\theta}|\mathbf{y})$ and $p(\boldsymbol{\theta})$ can be expressed as

$$\mathcal{L}(\boldsymbol{\theta}|\mathbf{y}) = \log\left[p(\boldsymbol{\theta}|\mathbf{y})\right] = -\sum_{j=1}^{J}\left[\frac{y_j - h_j(\boldsymbol{\theta})}{\sigma_{\epsilon_j}}\right]^2 + \log[p(\boldsymbol{\theta})], \tag{3.7}$$

where y_j is the corresponding standardized voltage at circuit node j with $j = 1, \cdots, J$.

3.6 Use of Adaptive MCMC to Solve the Inverse Problem

Various MCMC methodologies have been developed to sample from complex probability distributions such as a joint posterior distribution in Bayesian inference. One of the simplest approaches is to use the random-walk Metropolis (RWM) algorithm as illustrated in Gilks et al. (1995). To speed up the convergence of the Markov chain and increase the acceptance rate, Haario et al. (2001) introduce an adaptive Metropolis algorithm to adapt the covariance matrix of the proposal distribution empirically according to the historical Markov chain values. Many other adaptive algorithms have been developed such as Andrieu and Robert (2001), Sahu et al. (2003), Haario et al. (2005), and Andrieu and Thoms (2008). Vrugt et al. (2009) proposed a self-adaptive MCMC algorithm based on differential evolution to deal with cases when inappropriate initial proposal distributions are used. RWM generates new Markov chain values from a symmetric proposal distribution with a specified covariance matrix, however, has a low acceptance rate when the proposal distribution is not similar to the (unknown) true joint distribution of the parameters or when there exist strong correlations among parameters to be estimated. In this section, we propose a modified adaptive RWM algorithm that can tune the covariance matrix of the proposal distribution and determine the number of tuning iterations automatically.

3.6.1 Adaptive Metropolis Algorithm

The general idea of an adaptive Metropolis algorithm is to use the historical Markov chain values to tune the covariance matrix of the proposal distribution, so the acceptance rate will increase because the covariance matrix is reflecting more information about the true joint distribution of the parameters. In case of high dimensional parameters, we first reduce the dimension by doing an initial sensitivity analysis to find out how each component parameter can affect the output. To run the MCMC in an efficient way, we tune the covariance matrix of the proposal distribution of parameters, which can be used to characterize the covariance matrix of joint posterior distribution of the parameters. We develop a tunning algorithm based on methods described in Haario et al.

(2001) and Andrieu and Thoms (2008). At the end, we apply the standard MCMC to ensure ergodicity of the Markov chain values. Our proposed algorithm is as follows,

1. Initial stage.

   - For each of the circuit component, $i = 1, \cdots, n$, do a component-wise sensitivity analysis of the component's effect on the overall posterior function $p(\boldsymbol{\theta}|\mathbf{y})$. Record the maximum observed log posterior value that each transistor has as L_i.

   - Split the components to be estimated into two groups: one group that has large posterior values, which will be more strongly influenced by changes in the parameter values. The other group of components will not be influential to the posterior distribution. Denote the groups of components that have large log posterior values by G_1 and the other group by G_2.

2. Training stage. To speed up the movement towards the posterior mode and avoid degeneracy problems so we can tune the covariance matrix of the proposal distribution efficiently, we first run N_{train} iterations with a fixed initial covariance matrix. Initialize the starting component values, $X_{train,0} \in \mathbb{R}^n$ and the positive definite covariance matrix, $C_{train,0}$. Propose new component values with RWM algorithm and only accept the proposed values when the log posterior value is increasing.

3. Tuning stage on components in G_1: Keep all other components in G_2 at their nominal values, initialize the starting values of components in G_1 with the last training stage chain values, $X_{G_1,0} = X_{G_1,N_{train}}$, and take the empirical covariance matrix of components in G_1 of training stage, $C_{G_1,0}$ as the starting covariance matrix for the proposal distribution. At each step t:

   - Update the values of Markov chain values, $X_{G_1,t}$ using the acceptance rules from RWM algorithm, (i.e., acceptance rate determined by the ratio of the posterior distribution of the proposed values and $X_{G_1,t-1}$).

- Drop the older chain values to avoid excessive chain values far from the mode and use only the newer ones to further update the covariance matrix.

For every n_{batch} step:

- Update the proposal covariance matrix as $C_{G_1,t}$ and propose new component values with the updated covariance matrix.
- Keep track of the value of each element in the covariance matrix $C_{G_1,t}$ after each update, compute the quartile coefficient of dispersion for each element separately every n_1 iterations (i.e., check the value of $[(Q_3 - Q_1)/(Q_3 + Q_1)]$). As the magnitude of dispersion reflects the variation of the historical update, smaller quartile dispersion values indicate that the specific element is relatively stable and a further update will not change it much. The proposal covariance matrix becomes stable when the quartile coefficient of each element of the matrix is close to zero.

4. Tunning stage on all components: Expand the tuning to the whole parameter space. Obtain a final covariance matrix for all parameters.

5. Metropolis stage: Use the final tuned covariance matrix obtained from the tuning stage as the covariance matrix of the proposal distribution. Apply the standard Metropolis algorithm and save Markov chain values for inference.

3.6.2 Application of Adaptive MCMC to A Failed Class A Power Amplifier

We pick one failed device from the virtual population and apply the above adaptive MCMC algorithm to find the posterior distributions of resistor RL and transistor $Q1$, and determine which components cause the failure. We use a heavy tail scaled t distribution as a weakly informative prior for the component values based on suggestions on Gelman et al. (2017). Figure 3.3 shows the prior distributions of RL and $Q1$. The prior distributions indicate our knowledge that these component values for a failed device: resistors tend to have a higher resistance and transistors tend to have lower gains for failed devices.

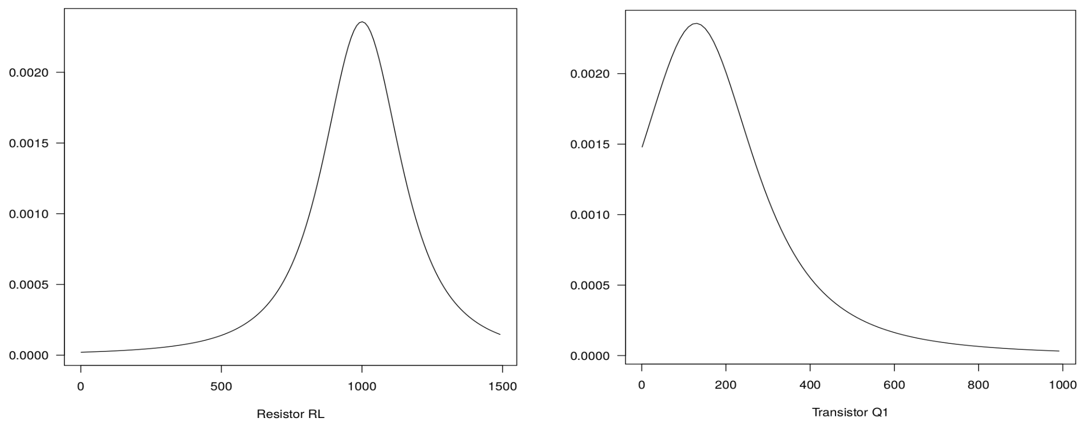

Figure 3.3: Prior distributions of RL and $Q1$.

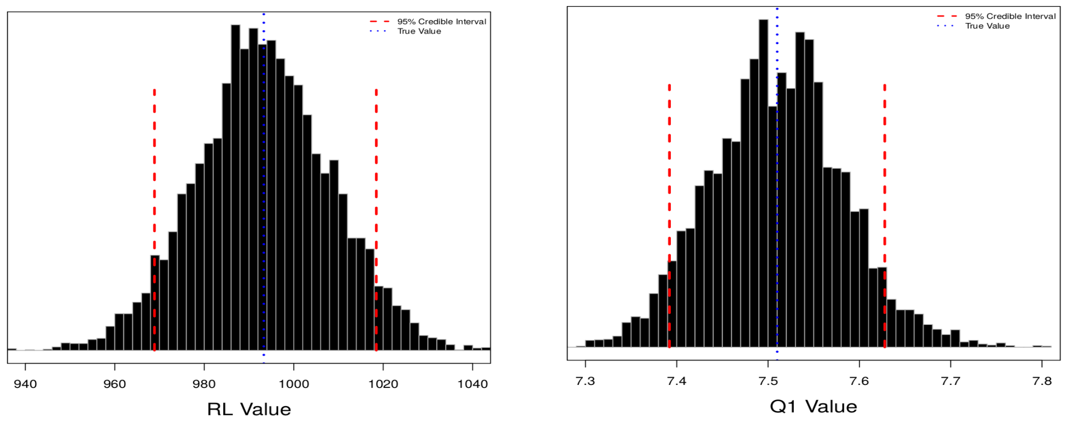

Figure 3.4: Marginal posterior distributions of RL and $Q1$.

The posterior distributions based on 10,000 iterations of MCMC of RL and $Q1$ are shown in Figure 3.4. The marginal posterior distribution of $Q1$ has mode at around 7.51, which is far from its nominal value (129). We can conclude that $Q1$ is causing the circuit to fail.

3.7 A Second Example: 741 Op-Amp Circuit

The integrated-circuit (IC) operational amplifier (op-amp) was first introduced in mid-1960s and became popular due to its versatility and low prices. We use a virtual version of the well-known op-amp to illustrate our failure-detection methodology with an example of a failed operational

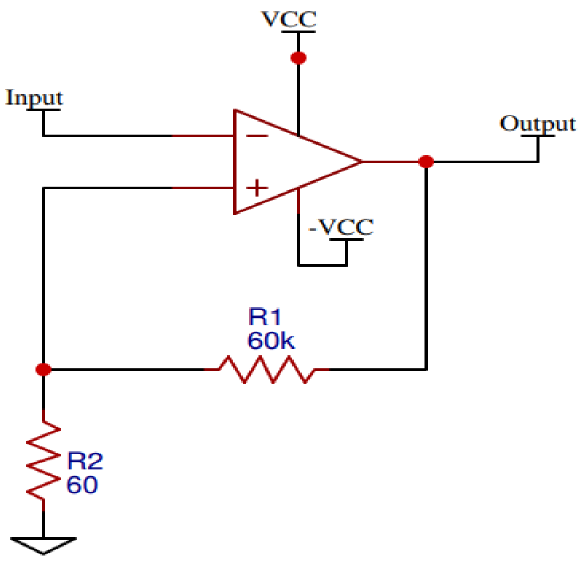

Figure 3.5: Negative feedback amplifier circuit.

amplifier. We consider two different kinds of gains. The overall gain of an op-amp circuit (the ratio of output and input voltage) is used to determine if the device is failed. The gain of the individual transistors that comprise the op-amp are the component values to be estimated.

3.7.1 741 Op-Amp Background

An op-amp is usually composed of a number of transistors, resistors, and capacitors, and the overall op-amp gain can be less than its nominal value when certain transistors malfunction (e.g., when transistor gains are far lower than their nominal values). In this example, we use a negative feedback amplifier circuit based on a 741 op-amp and employ ngspice to simulate the circuit as shown in Figure 3.5. There are 25 transistors in the op-amp as shown in Figure 3.6. More specifically, we used 15 NPN-type 2N3904 bipolar transistors and 10 PNP-type 2N3906 bipolar transistors to simulate this 741 op-amp. Nominal specifications of the transistors are given in data sheets such as Fairchild (2001). Failures of the op-amp circuit are usually caused by one or more transistors with gains out of specification (generally gain values that are too low).

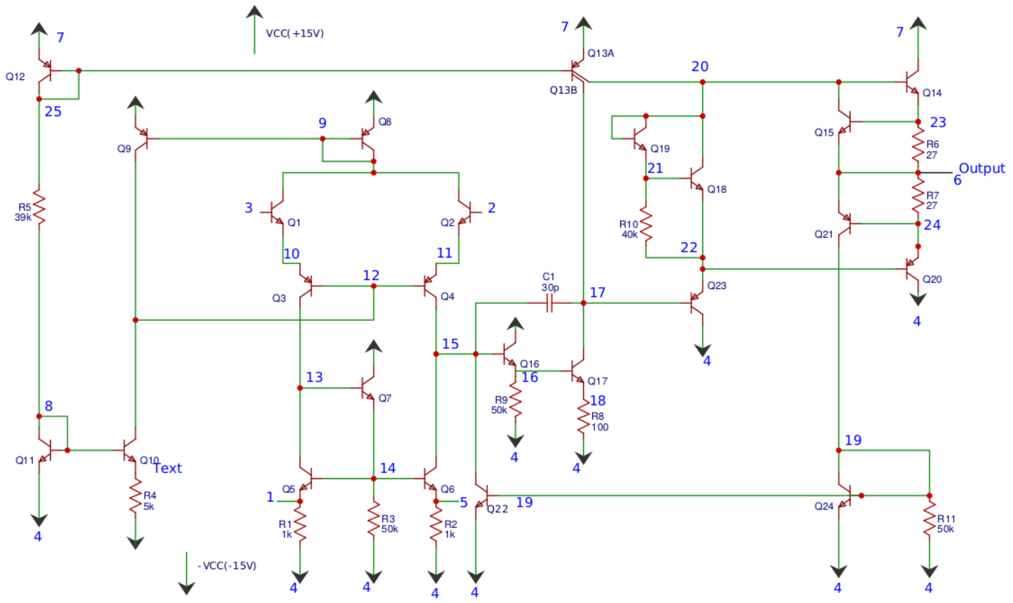

Figure 3.6: 741 op-amp schematic with numbered circuit nodes (Redrawn based on the circuit on page 812 of Adel S. Sedra (1998)).

3.7.2 Application of the Adaptive MCMC Algorithm

Using the same standard as in the previous class A power amplifier example, we also define an op-amp as failed if its output voltage is more than 5% different from its nominal value. We built 1,000 simulated op-amp devices with a theoretical op-amp circuit gain equals to 1001 (determined by the fixed negative feedback loop set up in Figure 3.5) and the 25 transistor gains for each device were drawn from truncated normal distributions. We assume that the distributions of all transistor gains are independent and the gains follow a mixture of two truncated normal distributions. Table 3.2 shows the values of other parameters.

Table 3.2: Parameter values of the mixture of truncated normal distributions.

Transistor Type	μ_{ND}	σ_{ND}	μ_D	σ_D
NPN	416.4	12.65	50	10
PNP	180.7	5.49	50	10

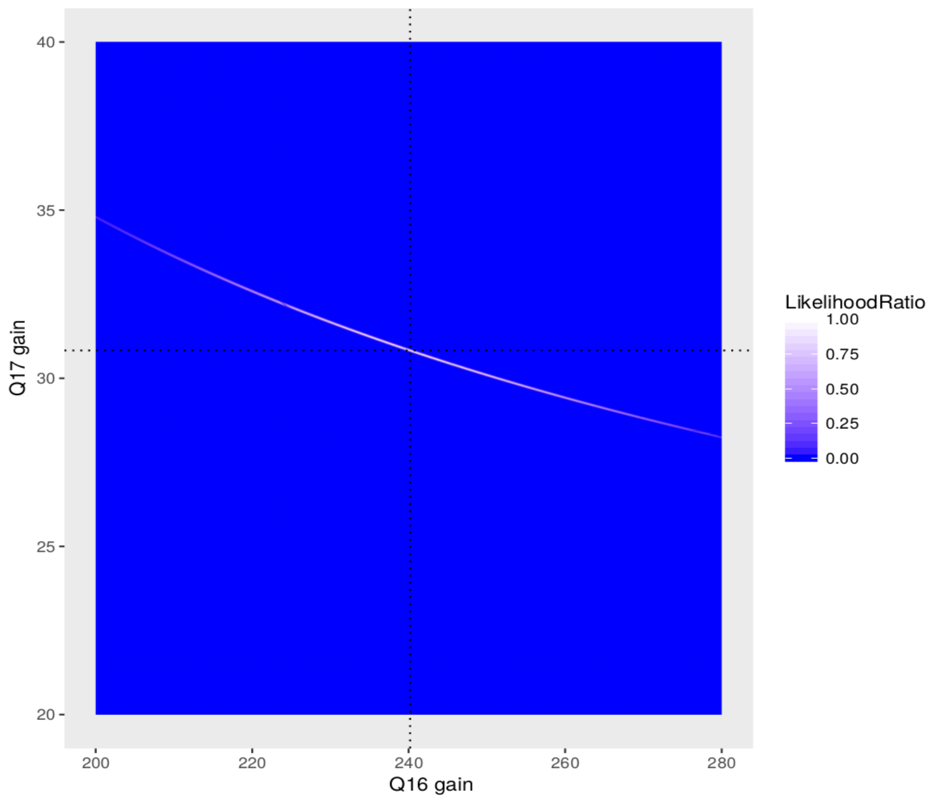

Figure 3.7: 741 op-amp circuit likelihood values as a function of $Q16$ and $Q17$ gains.

Before applying adaptive MCMC algorithm, we explore the op-amp circuit behavior as a function of the gains of the different transistors (part of the forward model). We find that there are several pairs of transistors that are strongly correlated with each other. For example, Figure 3.7 shows the relative likelihood ratio (original likelihood function divided by the maximum of the likelihood function) as a function of the transistors $Q16$ and $Q17$ gains. The contour plot shows that the estimated gains of $Q16$ and $Q17$ have a strong correlation because the gain of the cascaded $Q16$ and $Q17$ is approximately equal to the product of the individual gains of $Q16$ and $Q17$. Thus it would take the standard MCMC algorithm very long to converge and find the posterior distributions of component values if the covariance matrix of the proposal distribution does not take the correlation into account. Our adaptive MCMC can help to characterize these kinds of correlations automatically and thus improve the sampling efficiency of the MCMC algorithm and significantly speed up convergence.

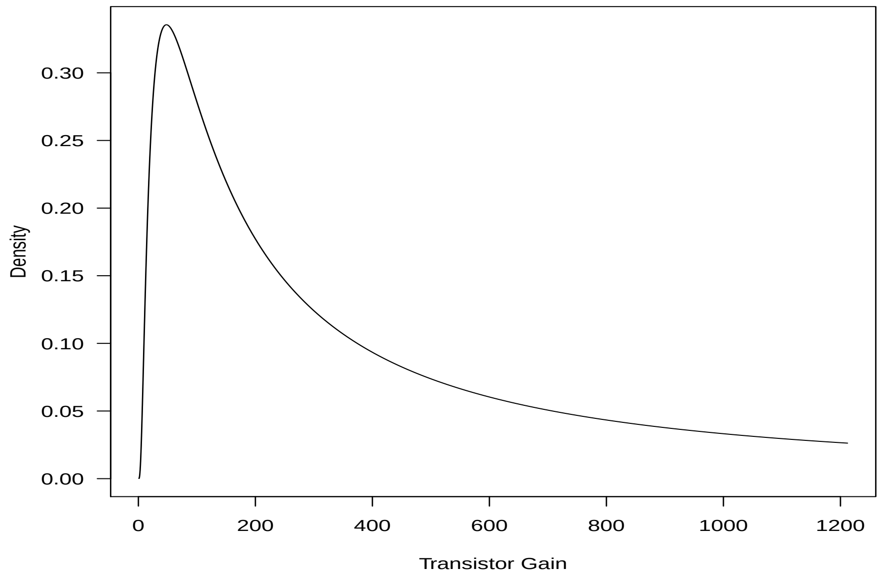

Figure 3.8: Prior distribution of transistor gains.

We select one failed device from the virtual population and apply the above adaptive MCMC algorithm to find the posterior distribution of the 25 transistor gains, and isolate the one or ones that caused the failure. We use a scaled t distribution with heavy tails as the weakly informative prior distribution for the component parameter values, as suggested by Gelman et al. (2017). Figure 3.8 shows the prior distribution on the original gain scale. The prior distribution indicates our knowledge that the gain values for a failed device could have small values, but there are still significant weights around their nominal values (416.4 and 180.7 for NPN- and PNP-type transistors, respectively) with heavy tails.

The algorithm estimates the gain values of the failed device well for all of the transistors. Plots in Figure 3.9 show the representative marginal posterior gain distributions for $Q3$, $Q5$, $Q13B$, $Q16$, and $Q17$, respectively. The posterior gain distribution of $Q17$ has the narrowest credible interval with the mode located at around 29.2, which is significantly lower than its nominal value (416.4). Even with the strong negative correlation between $Q16$ and $Q17$, our algorithm can estimate the transistor gains of $Q16$ and $Q17$ with reasonable precision. The posterior distributions of Q3, Q5, Q13B, and Q16 have modes close to their nominal values (416.4 for Q5 and Q16, and 180.7 for Q3 and Q13B) with wider credible intervals, which indicate that our algorithm can effectively discriminate between defective and non-defective components. The algorithm can deal with the higher dimensional problems as well.

Depending on the starting Markov chain values and the proposal distribution, the standard MCMC will take hours or longer to obtain converged Markov chains and stable posterior estimates of component values. It will, however, take only a few minutes to achieve the same results with adaptive MCMC, and our proposed algorithm is especially efficient if some of the circuit components have complicated correlations with each other.

3.8 Concluding Remarks

In this paper, we introduce a general methodology for electronic circuit troubleshooting that can be used when knowledge of failure modes is unavailable. It can be applied to various applications flexibly. The methodology works well even when the electronic circuit has many components. The methodology makes inferences on the electronic circuit component values with the help of circuit simulation and Bayesian inference methods. Then the information about the component values can be used to identify the possible causes of failure. In the examples of two relatively simple simulated electronic circuits, voltages from the circuit nodes are required to make inverse Bayesian inference. In practice, this kind of data need to be available to apply our methodology.

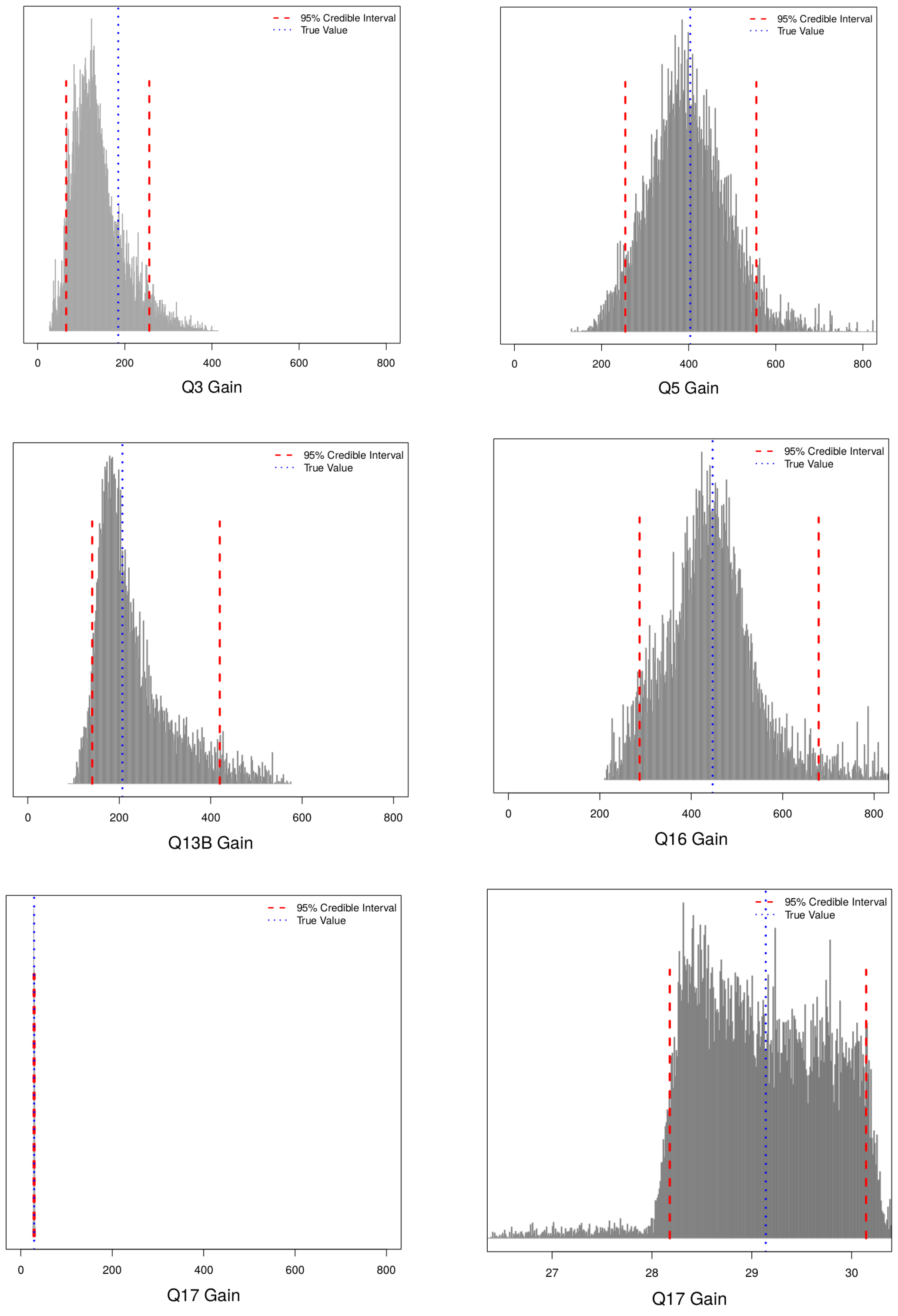

Figure 3.9: Posterior gain distribution of Q3, Q5, Q16, Q13B and Q17.

3.9 References

Adel S. Sedra, K. C. S. (1998). Microelectronic circuits.

Alippi, C., Catelani, M., Fort, A., and Mugnaini, M. (2002). SBT soft fault diagnosis in analog electronic circuits: a sensitivity-based approach by randomized algorithms. *IEEE Transactions on Instrumentation and Measurement*, 51:1116–1125.

Aminian, F. and Aminian, M. (2001). Fault diagnosis of analog circuits using bayesian neural networks with wavelet transform as preprocessor. *Journal of Electronic Testing*, 17:29–36.

Andrieu, C. and Robert, C. P. (2001). *Controlled MCMC for optimal sampling*. National Institute of Statistics and Economic Studies.

Andrieu, C. and Thoms, J. (2008). A tutorial on adaptive MCMC. *Statistics and Computing*, 18:343–373.

Bandler, J. W. and Salama, A. E. (1985). Fault diagnosis of analog circuits. *Proceedings of the IEEE*, 73:1279–1325.

Cui, J. and Wang, Y. (2011). A novel approach of analog circuit fault diagnosis using support vector machines classifier. *Measurement*, 44:281–289.

Fairchild (2001). Fairchild Transistors. https://www.fairchildsemi.com/datasheets/2N/ ,https://www.onsemi.com/PowerSolutions/supportDoc.do?type=models&rpn=2N3055. [Online; accessed 30-December-2018].

Gelman, A., Simpson, D., and Betancourt, M. (2017). The prior can often only be understood in the context of the likelihood. *Entropy*, 19(10):555.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. (1995). *Markov Chain Monte Carlo in Practice*. CRC press.

Haario, H., Saksman, E., and Tamminen, J. (2005). Componentwise adaptation for high dimensional MCMC. *Computational Statistics*, 20(2):265–273.

Haario, H., Saksman, E., Tamminen, J., et al. (2001). An adaptive Metropolis algorithm. *Bernoulli*, 7:223–242.

Sahu, S. K., Zhigljavsky, A. A., et al. (2003). Self-regenerative Markov chain Monte Carlo with adaptation. *Bernoulli*, 9:395–422.

Tadeusiewicz, M., Halgas, S., and Korzybski, M. (2002). An algorithm for soft-fault diagnosis of linear and nonlinear circuits. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49:1648–1653.

Tuinenga, P. W. (1991). *SPICE: A guide to circuit simulation and analysis using Pspice*. Prentice Hall PTR.

Vasan, A., Long, B., and Pecht, M. (2014). Experimental validation of ls-svm based fault identification in analog circuits using frequency features. *Engineering Asset Management 2011*, pages 629–641.

Vasan, A. S. S., Long, B., and Pecht, M. (2013). Diagnostics and prognostics method for analog electronic circuits. *IEEE Transactions on Industrial Electronics*, 60:5277–5291.

Vrugt, J. A., Ter Braak, C., Diks, C., Robinson, B. A., Hyman, J. M., and Higdon, D. (2009). Accelerating Markov chain Monte Carlo simulation by differential evolution with self-adaptive randomized subspace sampling. *International Journal of Nonlinear Sciences and Numerical Simulation*, 10:273–290.

CHAPTER 4. RSPICE: A PACKAGE FOR USING NGSPICE FROM R

Qianqian Shan, Stephen D. Holland, William Q. Meeker

4.1 Abstract

The RSpice package provides functions to call ngspice, an open source electronic circuit simulator, from R. The circuit setup can be passed to the circuit simulator, ngspice, through R. R functions can then start/stop the simulation, alter device or model paramters, read the simulation output back to R and so on. It is a friendly and easy to use interface to combine the flexible R tools and the ngspice circuit analysis. This article illustrates how to conveniently use the package to run ngspice in R.

Keywords: Electronic circuit simulator, ngspice, interface, R.

4.2 Introduction

Electronics devices have become ubiquitous in industry, home, and our personal lives. Electronic circuit simulation softwares are used widely by product designer and manufacturers to solve design and manufacturing problems such as exploring the effect of tolerance levels of equipment components and optimization. Ngspice[1] is an open source mixed-level/mixed-signal circuit simulator which allows the simulation of kinds of circuits. It has been developed based on three open source software packages that are used for circuit simulation: Spice3f5, Cider1b1 and Spice. It is a fast and reliable tool for circuit simulation (Vogt et al., 2018). There are numerous applications of circuit simulation where statistical analysis of the results is needed.

Although ngspice provides built-in basic user interfaces, it would be complicated to visualize, compare and analyze multiple results at the same time. See Chapter 18 of Vogt et al. (2018) for more

[1] http://ngspice.sourceforge.net/

details on existing ngspice interfaces. R is a software system with highly extensible environment for statistical computing and graphics. RSpice is an R package that makes it possible to run ngspice in R directly by loading an ngspice shared library. The combination of ngspice and R can greatly simplify the simulation and post-processing analysis of electronic circuits properties in applications where statistical methods (e.g., Monte Carlo Markov Chain) are needed.

In this article, three examples are presented to demonstrate the usage and advantages of RSpice. Section 4.3 describes the implementation details of RSpice functions and demonstrates the basic usage of the functions with a toy example. Section 4.4 introduces two more complicated examples to show more ways to take advantage of RSpice. Section 4.5 briefly summarizes the RSpice package and examples.

4.3 Implementation of RSpice

In this section, we first introduce the functions in RSpice and then demonstrate the use of RSpice with a toy example.

4.3.1 Implementation

Running ngspice in R is implemented by calling the shared library (or dynamic linked library in Windows) of ngspice within R. The shared library provides exported functions and callback functions to control ngspice and return results from ngspice, respectively. RSpice provides an interactive interface to send commands to ngspice and receive results from ngspice interactively within R.

The RSpice package includes R wrapper functions to interact with ngspice via R functions that call a C interface library through R's ".C" interfacing functions. The R and C wrappers initialize ngspice by calling the exported function `ngSpice_Init` from the shared library, send commands to ngspice by `ngSpice_Command`, including sending a circuit, reading output names and length via their corresponding exported functions and so on. A brief illustration of the R functions is as follows:

1. The `circuitLoad(circarray, dylibpath, dylibname, listing)` function is used to:

   (a) Locate the path of ngspice shared library and initialize ngspice if it's not initialized. The function will search the default paths and default names for the ngspice shared library if `dylibpath` and `dylibname` are not specified.

   (b) Send the `circarray` file to ngspice via exported function `ngSpice_Circ` in the shared library, where `circarray` should be an array of strings representing the lines of an ngspice netlist file that defines a circuit and the operations to be simulated on the circuit. Examples of netlists are given in Section 4.3.2 and 4.4.

2. The `runSpice()` function sends a "run" command to ngspice to start the simulation specified in the netlist passed to ngspice.

3. The `exportResults(location)` function reads the simulation results from ngspice and makes the results available in R. The function first calls the exported function `ngSpice_CurPlot` to find out the name of the current circuit output, then calls `ngGet_Vec_Info` for all circuit node names in the current circuit output, and finally calls `ngSpice_AllVecs` to visit the simulation results for the specified node names by their `location` among all node names. The `location` information can be obtained by running `getPlotNames()`. `getLength()` is called within `exportResults(location)` to determine the length of output for each specified node (e.g., if we specify a linear scan of input voltage for a circuit, then the length of the output for a specified node will be corresponding to the different number of input voltages scanned).

4. The `spiceCommand(cmd)` function allows a valid command, `cmd`, to be sent to ngspice via `ngSpice_Command`. The commands include control and interactive commands introduced in Vogt et al. (2018).

5. The `unloadSpice()` function will unload the ngspice shared library.

4.3.2 Toy Example

A simple circuit with one voltage source and two resistors is shown in Figure 4.1, the voltage source has a constant voltage of $10V$ and is connected between node 1 and ground node (node 0). The two $5k\Omega$ resistors are connected in series. We use this circuit to demonstrate how to use the functions of RSpice to run the simulation and use the results in R.

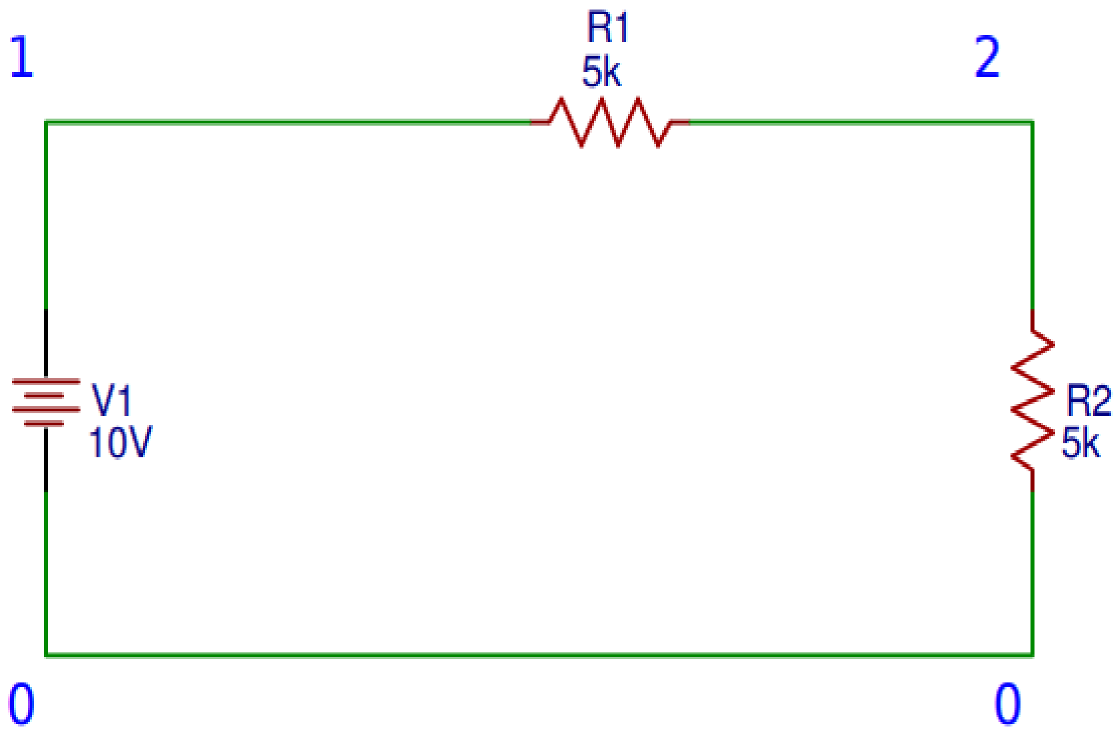

Figure 4.1: A simple circuit with one voltage source and two resistors.

The `toyexample` is a built-in RSpice object which contains a vector of character strings to define the above simple circuit (a netlist). Ngspice uses the character strings to set up a circuit, and it can be read from RSpice and is displayed below:

```
R> data("toyexample", package = "RSpice")
R> toyexample
```

```
[1] ".title toyexample "
```

```
[2] "R1 1 2 5k"
[3] "R2 2 0 5k"
[4] "VDD 1 0 DC 10V"
[5] ".op"
[6] ".end"
```

The netlist starts with a title line, and the following lines define that resistor R1 is placed in between node 1 and node 2 with resistance $5k\Omega$, R2 is in between node 2 and node 0, also with resistance $5k\Omega$, VDD is a direct current (DC) voltage source between node 0 and node 1 with $10V$ and node 1 is the positive node (anode). The `.op` command specifies a operating point analysis of the current circuit, so voltages for all nodes and currents in voltage sources are computed. The netlist should be ended with “.end”.

An important note about the naming rules of the circuit components in ngspice is that, the component type is determined by the first letter of its name, for example, the ”R” in ”R1” stands for a resistor and the following number 1 together with ”R” specifies a customized name of the particular resistor between node 1 and node 2 (Vogt et al., 2018, Chapter 2). Also, when the circuit is loaded, the names of circuit components will be converted to lower cases and it is necessary to specify the converted lowercase names in the `spiceCommand()` function if further modifications on the circuit components are needed. One can use function `spiceCommand("listing")` to print the current circuit from ngspice to check the names of the circuit components (Vogt et al., 2018, Chapter 17.5.39).

We first analyze the original circuit and then vary the value of R1 to check the relationship between the voltage at node 2 and the resistance of R1. Before sending any circuit information to ngspice, we need to make sure that the ngspice shared library exists and initialize it with the code:

```
R> circ <- circuitLoad(toyexample, dylibpath = NULL, dylibname = NULL)
```

The `circuitLoad()` function will first call `initializeSpice()` to search the ngspice shared library. If `dylibpath` and `dylibname` are not specified, the function will search the pre-specified default path and name for the ngspice shared library. One can specify the ngspice path and name explicitly using these two arguments if the ngspice shared library are not stored in the default path with its default name. It's not required to give the exact path of the shared library as `initializeSpice()` will search the specified `dylibpath` and its subdirectories recursively for the shared library. The more complete the path is, however, the faster the searching process will be. For example, if the shared library is located in "C:/Spice/bin", the `dylibpath` can be specified as "C:", "C:/Spice", or "C:/Spice/bin" and the last one cost the least amount of time to find the shared library.

The argument `dylibname` is the name of the ngspice shared library, and depending on the operating systems, the default name could be "ngspice", "libngspice", or "libngspice-0" when installing the ngspice shared library. It is also possible to rename the shared library with a customized name and specify it by argument `dylibname` to overwrite the default name. The shared library extension is automatically added by R, and it will be `.so` or `.dll`, depending on the operating systems.

In particular,

1. The default paths for ngspice shared library when installing ngspice shared library are
   - "/usr/local/bin" for unix-like systems.
   - "C:/Spice64/bin" for 64-bit Windows systems.
   - "C:/Spice/bin" for 32-bit Windows systems.
2. Ngspice will read its standard configuration file, `spinit`, everytime it starts. The default location of the `spinit` file is "/usr/local/share/ngspice/scripts" for unix-like systems, "C:/Spice/share/ngspice/scripts" for 32-bit Windows systems, or "C:/Spice64/share/ngspice/scripts" for 64-bit Windows systems. If the standard configuration file is not found when initializing ngspice, a warning message will be issued but ngspice continues. For more details, see Chapter 16.5 of Vogt et al. (2018).
3. If ngspice shared library was downloaded directly from online websites such as the ngspice official website, and it was not compiled directly by users on their own computers, error

may raise because of the setup in `spinit` file. Ngspice predefines six "Code Models" with the extension `.cm` as library files that can be used to model different kinds of electronic circuits in a way that is similar to the standard device models. The default path to search for these files is "/usr/local/lib/ngspice/" for Unix/Linux systems, and "C:/Spice/lib/ngspice" or "C:/Spice64/lib/ngspice" for Windows systems. The `spinit` file will load these libraries by default and the `initializeSpice()` function will return an error if these libraries can not be found. One could comment out the commands to load the code models in `spinit` if they are not needed for the electronic circuits to be specified. For example, here are the commands in `spinit` to load the code models for a Linux system,

```
codemodel /usr/local/lib/ngspice/spice2poly.cm
codemodel /usr/local/lib/ngspice/analog.cm
codemodel /usr/local/lib/ngspice/digital.cm
codemodel /usr/local/lib/ngspice/xtradev.cm
codemodel /usr/local/lib/ngspice/xtraevt.cm
codemodel /usr/local/lib/ngspice/table.cm
```

One could either comment out the lines with an asterisk ($*$) when the code models are not needed, or edit the path to locate the code models.

When the ngspice shared library is loaded successfully, the circuit netlist can be passed to ngspice for simulation. The `runSpice()` function is used to start the simulation. The argument `bgrun = FALSE` specifies that the simulation should be done in the same thread as the calling process; the simulation will otherwise run in a background thread. Plot is a type of SPICE terminology that implies a group of vectors containing ngspice outputs (Vogt et al., 2018, Chapter 17.3). The `getPlotNames()` function reads the vector names and their corresponding storage indices (location) from ngspice. The `location` is important as we may not be interested in exporting all simulated outputs from ngspice, and we usually would only export interesting results by specifying the corresponding locations. In this example, the outputs are stored with an order of vdd#branch, V(2) and V(1), where vdd#branch is the current through the voltage source VDD, V(2) is the voltage at node 2 and V(1) is the voltage at node 1:

```
R> PlotNames <- getPlotNames()
R> PlotNames

location       Name
1          vdd#branch
2          V(2)
3          V(1)
```

Suppose that our goal is to investigate the relationship between the resistance of R1 and the voltage at node 2. We can do this by sending commands to alter the R1 value to ngspice and collect the corresponding voltage values of V(2). The commands to alter the resistance of R1 from $1K\Omega$ to $10K\Omega$ with an increment of $1K\Omega$ are:

```
R> r1.values <- paste0(seq(1, 10, 1), "k")
R> cmds <- paste0("alter r1=", r1.values)
R> print(cmds)

[1] "alter r1=1k"  "alter r1=2k"
[3] "alter r1=3k"  "alter r1=4k"
[5] "alter r1=5k"  "alter r1=6k"
[7] "alter r1=7k"  "alter r1=8k"
[9] "alter r1=9k"  "alter r1=10k"
```

The voltage values at node 2 can be obtained from ngspice with `exportResults(2)` each time after running the circuit with the new resistance value of R1, where the argument `2` for `exportResults()` indicates the location of voltage at node 2.

```
R> v2 <- double(length(cmds))
R> for (i in 1:length(r1.values)) {
R>spiceCommand(cmds[i])
R>runSpice(bgrun= FALSE)
R>v2[i] <- exportResults(2)
R> }
R> print(v2)
```

```
[1] 8.333333 7.142857 6.250000 5.555556
[5] 5.000000 4.545455 4.166667 3.846154
[9] 3.571429 3.333333
```

Figure 4.2 shows the above relationship of voltage at node 2 and the resistance of R1.

```
R> plot(seq(1, 10, 1), v2, xlab = expression(paste(R[1], " (k", Omega,")" )),
+  ylab = "Voltage at Node 2", pch = 16, las = 1)
```

In this example, we show that ngspice can interact with R via RSpice and display the results conveniently. The real advantage of using RSpice arises when the circuit is much more complicated that there is no analytical solution for the voltages and currents at the circuit nodes of interest.

4.4 More Complicated Examples

In this section, we demonstrate two more examples to show the versatility of RSpice to do different kinds of statistical analysis.

4.4.1 Linear Regulator

In electronics, it is often necessary to keep the output voltage independent of fluctuations in the input and the load. A linear regulator is a system that provides constant voltage output by observing the output power demand and adjusting a series resistor. Figure 4.3 is a schematic

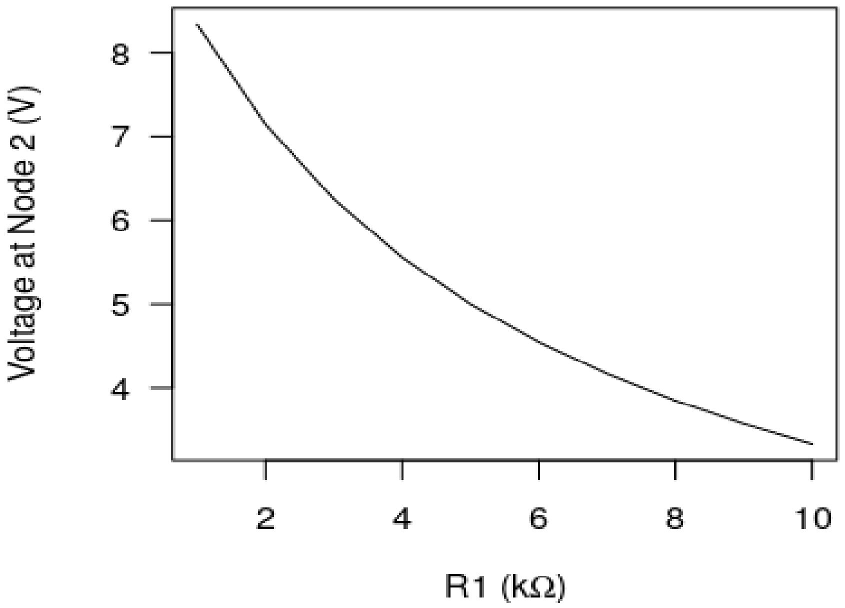

Figure 4.2: Voltage at node 2 as a function of the resistance of $R1$.

diagram for one type of linear regulators. The circuit is composed of a reference voltage V_{ref}, an NPN bipolar junction transistor Q1, a voltage-controlled voltage source E1 and a resistive divider with R_{upper} and R_{lower}. More details on how the circuit works can be found in Basso (2008)[Chapter 1]. The linear regulator is used as an example to present how the combination of ngspice and R can help the visualization and exploration of circuits.

This example performs a transient analysis of a linear regulator from time 0 to $0.06s$ with an increment of $0.1ms$ to investigate that if the linear regulator is functioning well by checking the relationship between the input and output voltages and to see how the output voltage changes with varied load R_{load}.

```
R> data("linearregulator", package = "RSpice")
R> circuitLoad(linearregulator)
```

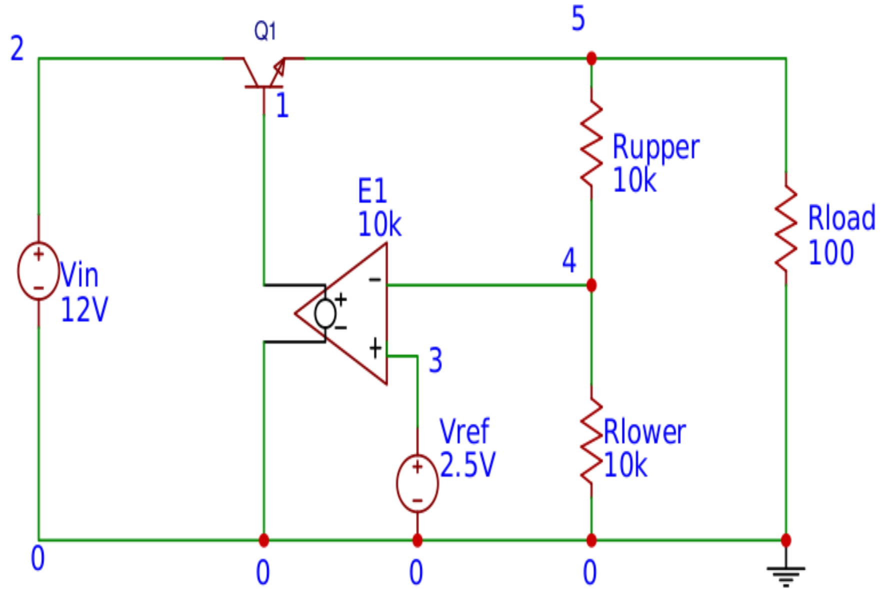

Figure 4.3: Linear regulator schematic (modified example from Chapter 1 of Basso (2008)).

```
R> linearregular

[1] ".title Linear Regular, modified from Basso: Switched-Mode Power Supplies"

[2] "Q1 2 1 5 QN2222"

[3] ".MODEL QN2222 NPN BF=105 BR=4 CJC=12.2P CJE=35.5P IKF=.5"

[4] "+ IKR=.225 IS=15.2F ISE=8.2P NE=2 NF=1 NR=1 RB=1.49 RC=.149"

[5] "+ RE=.373 TF=500P TR=85N VAF=98.5 VAR=20 XTB=1.5"

[6] "Vin 2 0 sin(0 12 60 0 0)"

[7] "E1 1 0 3 4 10k"

[8] "Vref 3 0 DC=2.5"

[9] "Rlower 4 0 10k"

[10] "Rupper 5 4 10k"

[11] "Rload 5 0 100"

[12] ".tran 0.1m 0.06"

[13] ".END "
```

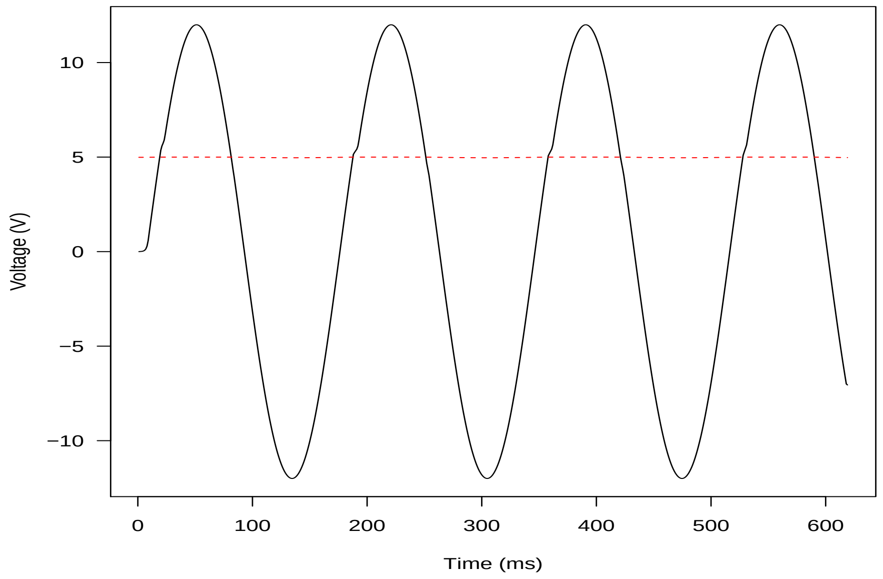

Figure 4.4: Input (solid line) vs output (dashed line) of the linear regulator.

We apply an alternating current (AC) voltage source as input with magnitude of $12V$ and frequency of $60Hz$. Output and input voltage values are exported from locations 8 and 6 of the ngspice simulation results, respectively. Figure 4.4 shows that the output voltage is nearly a constant around $5V$ even though the input is a sine wave with amplitude as $12V$.

```
R> runSpice(bgrun= FALSE)
R> PlotNames <- getPlotNames()
R> results <- exportResults(c(8, 6))
R> plot(c(1, data.length), c(min(results), max(results)), type = "n",
+  xlab = "Time (ms)", ylab = "Voltage (V)", las = 1)
R> for (i in 1:2) {
R>lines(results[i, ], col = i)
R> }
```

Next we alter the values of the load resistor from 10Ω to 200Ω to see how its value affects the output voltage (voltage at node 5 with `location` = 6) at time 0 of the transient analysis of the circuit. The results are shown in Figure 4.5.

```
R> rload.values <- paste0(seq(10, 200, 10))
R> cmds <- paste0("alter rload=", rload.values)
R> v5 <- double(length(cmds))
R> for (i in 1:length(rload.values)) {
R>spiceCommand(cmds[i])
R>runSpice()  # run the simulation
R>v5[i] <- exportResults(6)[, 1]
R> }


R> plot(rload.values, v5, xlab = expression(paste( "R"[load]," (", Omega,")" )),
R>      ylab = "Output Voltage", type = "l")
```

It can be seen that the value of the output remains a constant up to the fourth decimal even when the linear regulator has AC input and the resistance of the load changes from 10Ω to 200Ω.

4.4.2 741 Operational Amplifier

Integrated-circuit (IC) operational amplifiers (op-amp) were first introduced in mid-1960s and became popular due to their versatility and low price. We use a simulation of the well-known 741 op-amp to illustrate the usage of statistical methods on ngspice results. Figure 4.6 shows the internal structure of a 741 op-amp. The op-amp is composed of transistors, resistors and capacitors, and Figure 4.7 gives the pin-out diagram on how the op-amp can be connected to other electronic devices. The op-amp usually operates close to its expected performance level even with substantial variance in the component values due to its robustness (Adel S. Sedra, 1998). The overall op-amp output can, however, be far from its nominal value when certain transistors malfunction (e.g., when a transistor's gain is far lower than the nominal value). We study the behavior of a 741 op-amp in a

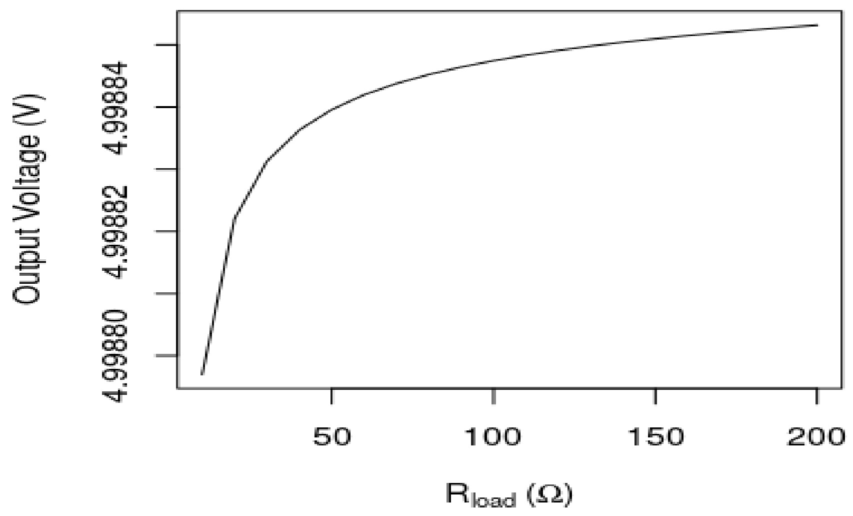

Figure 4.5: Output voltage versus the resistance of load for the linear regular.

negative feedback circuit loop as shown in Figure 4.8 (i.e., a circuit with its output node connected to its inverting input node).

In this example, we use the interaction between ngspice and R more extensively: We check the effects of the variations of the transistor gains on the voltage output of op-amp circuit. We only test the effects of transistor gains on the output for the sake of simplicity in description. One could also check the effects of variability in the resistors and capacitor characteristics on the output in a similar way. Denote the gain values of the 25 transistors as $\boldsymbol{\beta} = (\beta_1, \cdots, \beta_{25})^T$. Suppose that 99.9% of the transistors gain values are within 10% of their corresponding nominal values and each transistor gain value follows a normal distribution centered at its nominal value $\beta_{i,0}$ for each $i = 1, 2, \cdots, 25$. The corresponding standard deviation σ_i is derived so that $\Pr(\beta_i \in [0.9\beta_{i,0},\ 1.1\beta_{i,0}]) = 0.999$. The steps used to test the effects of transistors gains within their tolerance intervals on the circuit output are as follows:

For $i = 1, \cdots, 1000$,

1. Simulate the 25 transistor gain values, $\boldsymbol{x^i} = (x_1^i, \cdots, x_{25}^i)$, independently from their corresponding normal distributions.

2. Re-run the circuit analysis with the simulated transistor gain values, $\boldsymbol{x^i}$.

3. Export the output voltage value from ngspice.

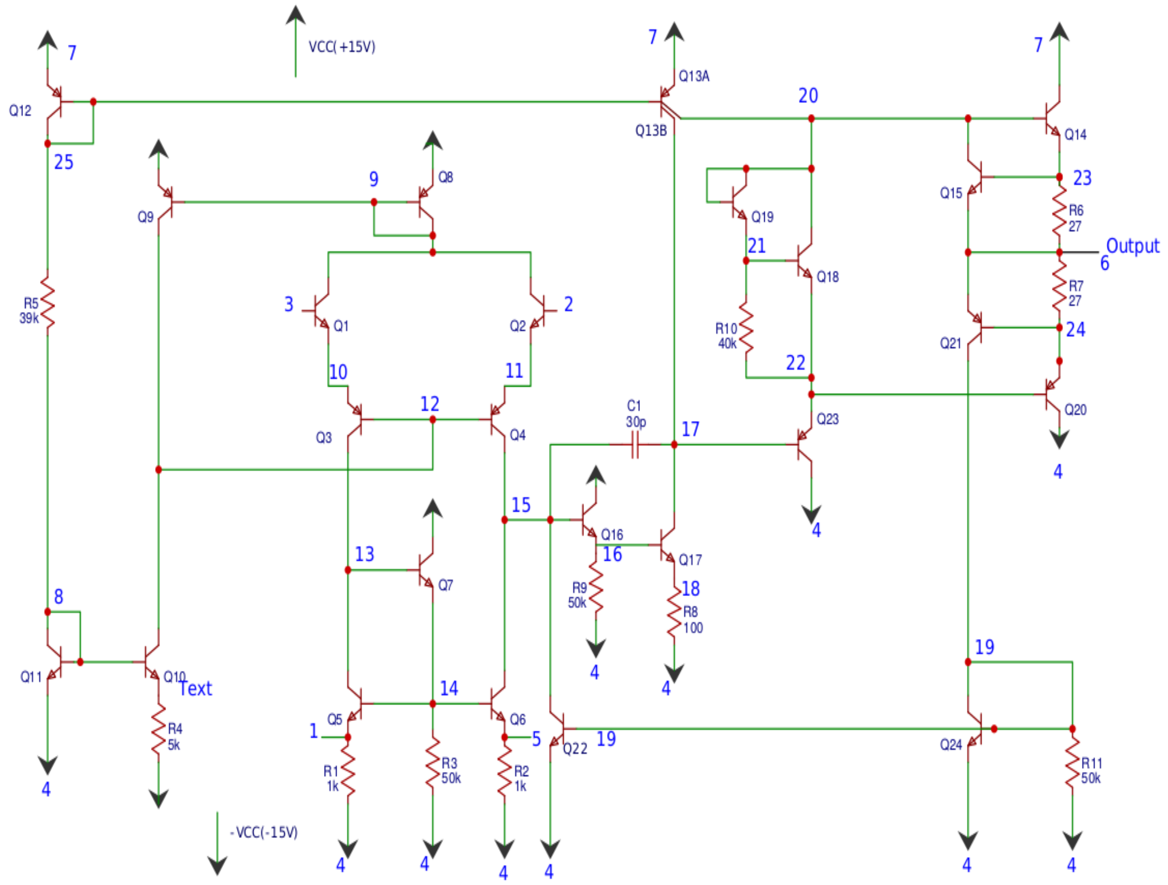

Figure 4.6: Internal structure of 741 op-amp.

```
R> circuitLoad(opAmp)

R> runSpice()

R> PlotNames <- getPlotNames()

R> location <-  c(which(PlotNames\$Name == "output"))
```

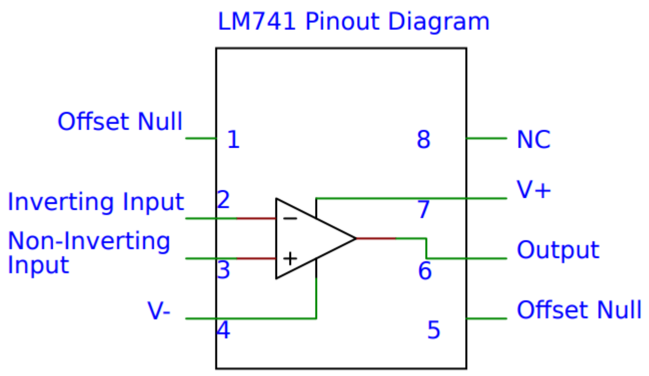

Figure 4.7: Pin-out diagram of 741 op-amp.

```
R> nominal.output <- exportResults(location)

R> head(PlotNames, 11)

location          Name

1              vplus#branch

2              vminus#branch

3              vinp#branch

4              xun.19

5              xun.24

6              xun.22

7              xun.21

8              xun.18

9              xun.16

10             output

11             xun.23

R> nominal.output

0.4845526
```

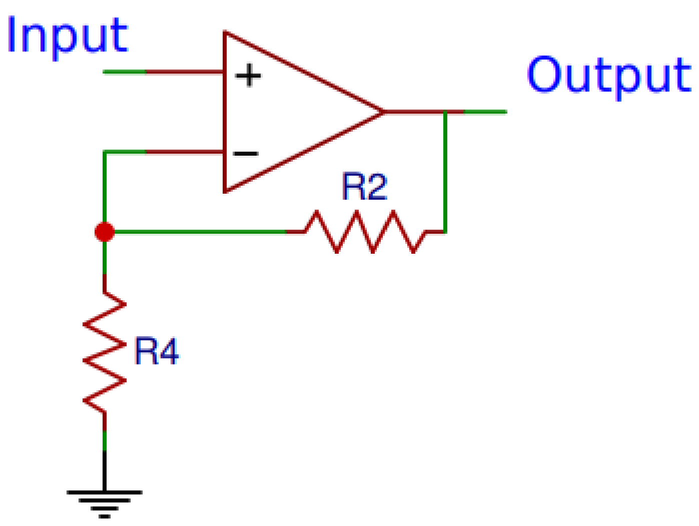

Figure 4.8: Op-amp in a negative feedback circuit.

It is straightforward to show that the standard deviations for NPN and PNP type transistors are `sd.npn = 12.65` and `sd.pnp = 5.49`, respectively. Then a 1000 by 25 matrix in which each row is a set of simulated gains, `gain.mat`, is generated:

```
R> gain.mat <- matrix(NA, nrow = n, ncol = 25)
R> npn.vector <- c(1, 2, 5, 6, 7, 10, 11, 15, 16, 17, 18, 19, 20, 23, 25)
R> for(i in 1:n){
R>gain.mat[i, ] <- sapply(1:25, function(i) ifelse(i %in% npn.vector, 416.4, 180.7))
R>for(j in 1:25) {
R>if (j %in% npn.vector) {
R>mus <- c(416.4)
R>sds <- c(sd.npn)
R>} else {
R>mus <- c(180.7)
R>sds <- c(sd.pnp)
R>}
R>gain.mat[i, j] <-  rnorm(1, mean = mus, sd = sds)
R>                }
R>             }
```

All the commands to be sent to ngspice are `altercmd`:

```
R> tem <- c("q2n3904q1", "q2n3904q2", "q2n3906q3", "q2n3906q4","q2n3904q5",
R>"q2n3904q6", "q2n3904q7", "q2n3906q8", "q2n3906q9", "q2n3904q10",
R>"q2n3904q11", "q2n3906q12", "q2n3906q13a", "q2n3906q13b","q2n3904q14",
R>"q2n3904q15", "q2n3904q16", "q2n3904q17", "q2n3904q18", "q2n3904q19",
R>"q2n3906q20", "q2n3906q21", "q2n3904q22", "q2n3906q23", "q2n3904q24")
R> altercmd <- paste("altermod @", tem, "[bf]=", as.vector(t(gain.mat)),
R>                   sep = "")
```

The output voltage values can then be exported and plotted:

```
R> results.voutput <- matrix(NA, nrow = n, ncol = getLength())
R> for (m in 1:n){
R>  # Initialize the commands vector to be passed to ngspice
R>  commands <- character(25)
R>  # Assign  to be sent to ngspice
R>  for (q in 1:25){
R>  commands[q] <- altercmd[(m-1)*(25) + q]
R>  }
R>  # Send alter commands down
R>  spiceCommand(commands)
R>  runSpice()
R>  # Export the output voltage from ngspice
R>  results.voutput[m, ] <- as.vector(exportResults(location))
R>   }
```

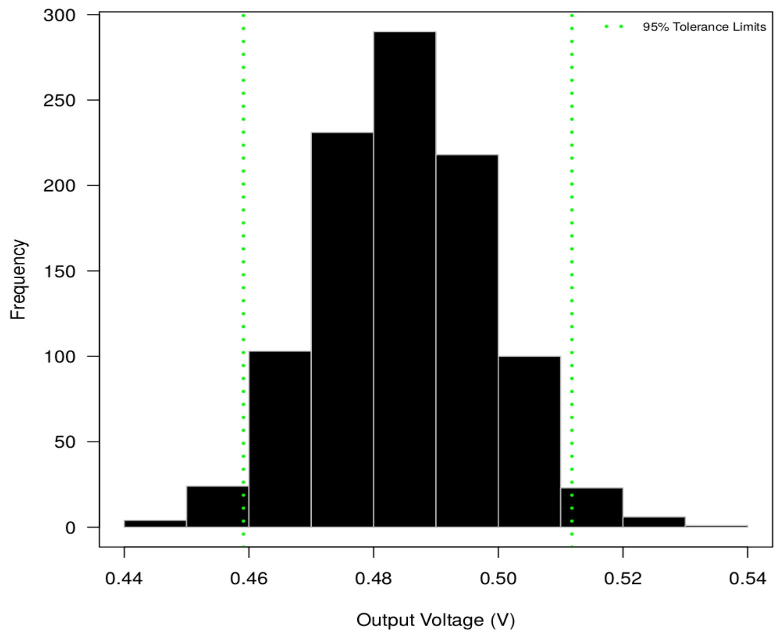

Figure 4.9: Histogram of the 1000 voltage outputs for a negative feedback loop from 1000 simulated 741 op-amps.

```
R> # Plot the all output voltages
R> hist(results.voutput, main = "", border = "grey", col = "black", las = 1,
R> xlab = "Output Voltage")
R> abline(v = res, lwd = 3, col = "red", lty = 2)
R> abline(v = tolerance.int, lwd = 2, col = "grey", lty = 3)
R> box()
R> legend("topright", c("Nominal Output", "95% Approximate Interval"),
R>        col = c("red", "grey"), lwd = 2 , lty = c(2, 3), bty = "n",
R>        cex = 0.7)
```

Figure 4.9 shows the distribution of the output voltage from the simulated population. The use of RSpice makes it easy to show how the tolerance levels of the circuit components are affecting the distribution of the output.

4.5 Summary and discussion

We introduce the RSpice package for running ngpsice in R with three examples. Various kinds of analysis of electronic circuits such as the tolerance analysis and setup checking of circuits can be done and visualized in R directly and interactively.

Help Page for the R Functions

1. `circuitLoad`: Load Specified Circuit to Ngspice

   Description:

   Initialize ngspice by linking to its shared library, load the specified circuit and send the circuit netlist to ngspice. If the ngspice shared library has been linked, only the function will only load the specified circuit.

   Usage:

   ```
   circuitLoad(circarray, dylibpath = NULL, dylibname = NULL,
   listing = TRUE, verbose = TRUE)
   ```

   Arguments:

`circarray`	a list of character strings which are used to define
`dylibpath`	the path of the ngspice shared library file. If `dylibpath = NULL`, the program will search the default path for the shared library.
`dylibname`	the name of the ngspice shared library without extensions. If `dylibpath = NULL`, the program will search the shared library with its default name.
`listing`	logical; if TRUE, print a listing of the current circuit after loading the circuit.
`verbose`	logical; if TRUE, print the stdout, stderr etc information exported from ngspice.

   Value:

The outputs from printf, fprintf and fputs of ngspice simulator if `verbose = TRUE`.

The listing of the current circuit will be returned if `listing = TRUE`.

Examples:

```
# Load a simple circtui with name test, the resistor R1 is 5k Ohm and
# connected to node 1 and 2, R2 is 1k Ohm and connected to node 2 and
# 0, and a voltage source VDD with 10V between node 0 and 1. A point
# analysis is performed.
circuitload(c('.title test', 'R1 1 2 5k', 'R2 2 0 1k',
'VDD 0 1 DC 10', '.op', '.end'))
```

2. `initializeSpice`: Initialize Ngspice

Description:

Load the ngspice shared library and initialize Ngspice with the exported functions. See Chapter 19 of the ngspice manual for more details.

Usage:

```
initializeSpice(dylibpath, dylibname, verbose)
```

Arguments:

`dylibpath`	the path of the ngspice shared library file.
`dylibname`	the name of the ngspice shared library without extensions such as .so or .dll.
`verbose`	logical; if TRUE, print the stdout, stderr etc information exported from ngspice.

Value:

NULL.

```
# Initialize Ngspice.
initializeSpice(dylibpath = "/usr/local/bin", dylibname = 'libngspice')
```

3. `findSpice`: Find The Path of The Ngspice Shared Library

Description:

Find if the specified Ngspice shared library exists and return the first path if there the shared library exists in multiple paths.

Usage:

```
findSpice(dylibpath, dylibname)
```

Arguments:

`dylibpath`	a character string with the general path of the Ngspice shared library. If NULL, the function will search across the computer, which is not recommended as it may search some directories that requires root/admin authorization. If a path is specified, the search will be conducted under the path recursively with a faster speed.
`dylibname`	a character string with the name of the Ngspice shared library.

Value:

The function will stop if no shared library is found, and the first available file path containing the shared library will be returned.

Examples:

```
# under windows os
```

```
findSpice(dylibpath = 'C:/Users/bin', dylibname = 'ngspice.so')
# under *unix os
findSpice(dylibpath = '/usr/local/lib', dylibname = 'libngspice.so')
```

4. `runSpice`: Send Command to Ngspice to Run the Circuit Simulation

Description:

Send command to ngspice to run the simulation of the circuit.

Usage:

```
runSpice(bgrun = FALSE, verbose = FALSE)
```

Arguments:

`bgrun`	logical; indicator of if the simulation is run in main thread or background thread.
`verbose`	logical; if TRUE, print the stdout, stderr etc information exported from ngspice.

Value:

Simulation status obtained from Ngspice is printed.

Examples:

```
# Run the simulation in main thread
runSpice()

# Run the simulation in background thread
runSpice()
```

5. `getLength`: Extract the Length of the Output

   Description:

   > Extract the length of the output from Ngspice.

   Usage:

   > `getLength()`

   Value:

   > An integer showing the length of the ouput after running a simulation of a circuit in ngspice.

   Examples:

   > `getLength()`

6. `getPlotNames`: Read All Output Names from Ngspice

   Description:

   > Read the available output names with their corresponding order from ngspice after the simulation of a circuit.

   Usage:

   > `getPlotNames()`

   Value:

   > The output names from ngspice after running a circuit simulation. A data frame with two columns: `location` and `Name`. The `location` column shows the order of

output names retuned from Ngspice and the `Name` column shows the corresponding output names returned from ngspice.

7. `exportResults`: Export the Output Values from NgspiceexportResults

   Description:

   Export the specified output values from Ngspice.

   Usage:

   ```
   exportResults(location)
   ```

   Arguments:

   `location` a vector of the location(s) of the outputs we want to export, i.e. the location-th output from the `getPlotNames()`.

   Value:

   A matrix containing the output values with the ouputs for each location stored in one row. The number of rows of the matrix equals `length(location)` and the number of columns equals the length of each output value.

   Examples:

   ```
   # obtain the location and Name information by running getPlotNames()
   getPlotNames()
   # Export the output values for location 1, 2 and 4.
   exportResults(c(1, 2, 4))
   ```

8. `spiceCommand`: Send Command to NgspicespiceCommand

   Description:

Send a valid command from caller to ngspice. See more control or interactive commands details in http://ngspice.sourceforge.net/docs/ngspice-manual.pdf

Usage:

```
spiceCommand(cmd)
```

Arguments:

`cmd` a vector of commands which to be sent to ngspice for one time change.

Examples:

```
# To print a listing of the current circuit
spiceCommand("listing")
```

9. `unloadSpice`: Unload NgSpice

Description:

Unload the ngspice shared library.

Usage:

```
unloadSpice()
```

Value:

A message ”Ngspice shared library unloaded.” or ”Ngspice shared library already unloaded.” dependening on if the `unloadSpice()` function is run for the first time or not.

4.6 References

Adel S. Sedra, K. C. S. (1998). Microelectronic circuits.

Basso, C. (2008). *Switch-mode power supplies SPICE simulations and practical designs*. McGraw-Hill, Inc.

Vogt, H., Hendrix, M., and Nenzi, P. (2018). *Ngspice Users Manual*.

CHAPTER 5. FUTURE WORK SUMMARY AND DISCUSSION

5.1 Summary and Discussion

We have developed several generalized NHPP models for recurrent event data with two product warranty examples in Chapter 2, and demonstrate the fitting and prediction performance of these models. Our general model incorporates the effects of different seasonality trends for different locations, the covariates and the individual level random effects, and is well suited for predictions in various recurrent event situations. The clustering of locations provides guidance not only for model fitting as shown in Chapter 2, but can also be helpful to provide data-driven insights on strategic planning for the future by indicating what locations are similar or dissimilar. The general model also gives practitioners the flexibility to compare and choose different models based on their own needs. For example, although the use of random effects for Product A can improve the model fitting performance in terms of AIC and BIC, it doesn't help to improve the prediction metrics on the hold-out data. Then we may prefer to choose Model 7 as our final model for prediction if we take the computation costs into account: Model 7 takes much shorter model fitting time than the models with random effects, as no parameters on random effects are needed and thus no iterations from the EM algorithm are needed.

In Chapter 3, we present a convenient implementation of electronic circuit trobleshooting by adaptive MCMC with the help of the electronic circuit simulator, ngspice. As all the data we need are the observed voltage values from several testing points of the failed electronic circuit, it's especially useful when we do not have a lot of the same kind of electronic circuits to experiment for possible failure modes or patterns. The adaptive MCMC algorithm works well for high dimension problems, and it can also provide us the marginal posterior distributions of each electronic circuit component, which can help to identify the components that are out-of-specification, and can also alert on some components that are close to be out-of-specification by checking if the modes of the posterior distributions are still in the manufactured tolerance limit.

5.2 Future Work

Potential directions to improve our current work include:

1. Seasonal warranty prediction based on recurrent event data:

    (a) In addition to the predictions of monthly and cumulative event counts of the recurrent event data, the predictions on the event costs are also very important, the costs include monthly, cumulative costs and costs by time and locations.

    (b) When we consider the random effect for each repairable system, we take the factor of heavy users vs. light users into our account. This factor may be modeled better if the product usage information of users is available as well. It's also common to have two-dimension product warranty, for example, a warranty may state that its term is either 36 months or 36,000 miles, which ever comes first. The usage information will not only affect the recurrent rate of events, but also the length of the warrany term.

    (c) The clustering of locations on seasonal effects is based on calendar months, and we use the monthly empirical recurrent rate values for the clustering. In practice, the recurrence rate for Decembers and Januaries (or similarly, July and August) are usually very similar to each other, models may be further simplified by studying how to group neighbor months together.

2. Electronic circuit troubleshooting using simulation and Bayesian inference

    (a) In Chapter 4, we illustrate a data-drive strategy to conduct electronic circuit troubleshooting. In practice, we may also use a hybrid model that uses both data and physical properties of the circuit information for modeling.