Summer 2018

A Comparison of Bridge Deterioration Models

Toktam Naderimoghaddam

A COMPARISON OF BRIDGE DETERIORATION MODELS

by

TOKTAM NADERIMOGHADDAM

(Under the Direction of Stephen Carden)

ABSTRACT

Predicting how bridges will deteriorate is the key to budgeting financial and personnel resources. Deterioration models exist for specific components of a bridge, but no models exist for the sufficiency rating which is an overall measure of the condition and relevance of a bridge used for determining eligibility for federal funds.

We have 25 years worth of data collected by the Georgia Department of Transportation from 1992 to 2016 about all bridges in the State of Georgia. More precisely, each row in this data set includes the characteristics of each bridge along with the sufficiency rating of that bridge in a specific year which was inspected by engineers.

In this thesis we introduce two models (Mixed Logistics-Gamma model and Poisson model) that predict the change of sufficiency rating of each bridge and tell us how much a specific bridge deteriorates in a specific period of time. We then compare these two models.

INDEX WORDS: Gamma distribution, Logistic distribution, Poisson distribution, Bridge deterioration, Sufficiency rating

2009 Mathematics Subject Classification: 62-07

A COMPARISON OF BRIDGE DETERIORATION MODELS

by

TOKTAM NADERIMOGHADDAM

B.S., Sharif University of Technology, Iran, 2002

M.S., Georgia Southern University, 2018

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial

Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

A COMPARISON OF BRIDGE DETERIORATION MODELS

by

TOKTAM NADERIMOGHADDAM

Electronic Version Approved:
July 2018

DEDICATION

This thesis is dedicated to my husband Saeed who was always a strong support to me with his unconditional love. To my beloved daughter Hanah who missed me every second that I was busy working on my thesis. To my lovely parents who inspired me the most, but will never read it.

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to my advisor, Dr. Carden, for introducing this subject to me and for his help and support during preparation of this thesis. I am also grateful to Dr. Iacob and Dr. Wanduku for kindly accepting to be a part of the Supervisory Committee and for reading my thesis. I very much appreciate Dr. Iacob for patiently answering my numerous LaTeX questions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The government of Georgia budgets around 162 million dollars to help maintain the state's roads and bridges [1]. The Georgia Department of Transportation (GDOT) inspects all bridges and bridge culverts of the State of Georgia (including county bridges) every two years. In 2017 there were 6,688 state-owned and 7,904 locally-owned bridges in Georgia [2]. Ninety-eight-percent of state-owned bridges are in fair to excellent condition and 2-percent of state-owned bridge structures are structurally deficient [2]. Eighteen-percent of local bridges are structurally deficient and functionally obsolete [3]. Bridges have a 75-year life cycle on average [4]. For years, the deficiency and uncertainty of federal funding have slowed down the maintenance of the bridges of the state and caused some maintenance projects to stop. To close any additional funding problems, FY 2016 budget contains 100 million dollars in bonds specifically for bridge maintenance and repair [4].

Budget authors have to predict how bridges will deteriorate in order to plan for maintenance costs and request funds from federal sources. Therefore, accurate prediction of bridge deterioration is of critical importance. There are two main methods of measuring the condition of a bridge.

1. Element-level Condition Assessment Method

Each bridge is constructed from a number of distinct elements (pieces of a bridge) generally ranging between 10 and 25. Some examples of elements are timber bridge railing, reinforced concrete bridge railing, metal bridge railing, steel open beam, timber open beam, reinforced concrete stringer, steel truss, etc. In this method, engineers inspect all the elements of bridges every two years and rate the condition of all those specific elements. The number of condition levels in the standard rating system is 4. These standard condition levels are GOOD, FAIR, POOR, and SEVERE. So this method uses a 1 through 4 system to assess the condition of each element in which 1 stands for good, 2 for fair, 3 for poor, and 4 for severe [6].

The standard models currently in use do not predict the condition rating directly, but they predict the change in condition rating that we call the "Drop". The two most popular models currently in use for modeling the drops in condition of elements include

- **Markov chain** models where the number of drops in condition rating between inspections is modeled with Poisson regression [5].
- **Weibull** models which count the time between drops. When a drop occurs, it is precisely one level [5].

Because there is a big difference between condition states, changes of more than one level at a time are rare. So the Weibull random variable model works well for this method. Also the drops are always integer in this method, so Poisson random variable works for the number of drops.

2. Overall Sufficiency Rating Assessment Method

In this method, the bridge as a whole is measured with a sufficiency rating. This is a real-valued number in [0,100], where 100 represents an entirely sufficient bridge and zero represents an entirely deficient bridge. Sufficiency rating is calculated by combining four different factors listed below [7].

- Structural adequacy and safety factor which is calculated from superstructure, substructure, culverts, and inventory rating items.

- Serviceability and functional obsolescence factor which is calculated from lanes on structure, average daily traffic, approach roadway alignment, main structure type, bridge roadway width, minimum vertical clearance(VC) over deck, deck condition, structural evaluation, deck geometry, under-clearances, waterway adequacy, STRAHNET(Strategic Highway Corridor Network) highway designation.

- Essentiality for public use factor which is calculated from detour length, average daily traffic, and STRAHNET(Strategic Highway Corridor Network) highway designation.

- Special reductions factor which is calculated from detour length, traffic safety features, and main structure type.

Twenty National Bridge Inventory items are used to calculate these four factors [7]. After calculating these four factors the sufficiency rating formula is:

Sufficiency rating (or SR for short) is defined to be $S_1 + S_2 + S_3 - S_4$, where

- S_1 is the structural adequacy and safety factor and ranges from 0 to 55.
- S_2 is the serviceability and functional obsolescence factor and ranges from 0 to 30.
- S_3 is the essentiality for public use factor and ranges from 0 to 15.
- S_4 is the special reductions factor and ranges from 0 to 13.

If the sufficiency rating is less than or equal to 80-percent the bridge is eligible for federal funds for rehabilitation, and if the sufficiency rating is less than 50-percent it is eligible for federal funds for replacement [8]. Some states use the sufficiency rating as the basis for establishing priority for repair or replacement of bridges; the lower the rating, the higher the priority. Therefore predicting how sufficiency rating will change could aid state agencies in predicting how much federal funds they are eligible for.

In the literature there is no research for modeling the sufficiency rating. In this thesis two models will be investigated to predict sufficiency rating to see which one works better. We will not be able to investigate the Weibull model because it assumes constant changes in rating which is reasonable for elements but not for sufficiency rating.

The Poisson model assumes integer drops, so it breaks the assumption in individual changes in sufficiency rating since reports are often not integer but it may be useful for predicting the average amount of change.

This suggests using a continuous random variable, but $P(\text{DROP} = 0) > 0$ which is not possible for a continuous random variable. Therefore we need a mixed-type random variable. We will consider setting $Z = XY$, where X is a $Bernoulli(p)$ and Y is a Gamma(α,β) random variable. Bernoulli random variable tells us if there is a drop in sufficiency rating or not. If there is a drop, then $X = 1$ and else, $X = 0$. Then if the drop exits the Gamma random variable says how much the drop is. Note that $P(Z = 0) = P(X = 0) = 1 - p > 0$ and for all $A \subseteq \mathbb{R}$ we have $P(Z \in A \mid X = 1) = P(Y \in A)$, that is, the probability function for the Gamma random variable.

The remainder of this thesis is organized as follows:

Chapter 2 describes the raw data and steps needed to transform it into something useable.

Chapter 3 describes the fitted models and variable selection to choose the best models.

Chapter 4 describes the simulations and curves to predict the deterioration under each model.

Chapter 5 has the comparison of our two models to see which model fits better.

Chapter 6 is the conclusion.

CHAPTER 2

DATA CLEANING

The data used in our investigation comes from the National Bridge Inventory (NBI) website. States, federal agencies, and tribal governments submit the data on bridges on an annual basis in the spring to the Federal Highway Administration. This data may be updated during the year in a way that complies with the National Bridge Inspection Standards, the Recording and Coding Guide for the Structure Inventory, and with the evaluation of the Nations Bridges. At the end of each year the data is finalized and published on the website of the NBI [8].

Data used in this thesis was collected during 25 years; from 1992 to 2016 about all bridges in state of Georgia. The number of bridges described per yearly files is shown in Table 2.1. In this thesis names of variables are denoted by capital letters and underscore symbols. Also data frames and models are respectively denoted by bold and italic font.

Data consists of 134 variables that represent several properties of the bridges. Some of the variables are related to the material used in the bridges like STRUCTURE_KIND, some others are about traffics like TRAFFIC_LANES on the bridge or under, some of the variables are related to the structure's measures like BYPASS, DETOUR_LENGTH or LONGTITUDE, etc., some others are about the geographic location of the bridge like ROUTE_PREFIX and DESIGNATED_LEVEL_OF_SERVICE, and others.

To keep the size of the data manageable we picked 13 variables that seem to be potentially more significant among 134 variables. Table 2.2 shows all the variables we used

Table 2.1: Yearly Number of Bridges

Year	**Number of Bridges**	**Year**	**Number of Bridges**
1992	16472	2005	16894
1993	16514	2006	16944
1994	16572	2007	16987
1995	16603	2008	17025
1996	16645	2009	17151
1997	16656	2010	17119
1998	16696	2011	17165
1999	16817	2012	17256
2000	11807	2013	14769
2001	11772	2014	14795
2002	16838	2015	14790
2003	16845	2016	14835
2004	16845		

along with their description, type of the variable and range or level of the variables.

Table 2.2: Variable Description

Variable	**Description**	**Type of Variable**	**Range/Level**
STRUCTURE_NUMBER	Unique identifier for each bridge/structure	Categorical	Represented as a variable-length integer.
SUFFICIENCY_RATING	An overall measure for structural quality	Continuous	Real number [0,100]
YEAR_BUILT	Obvious	Discrete	Integers from 1850 to 2015
TRAFFIC_LANES_ON	Number of lanes on the structure	Discrete	Integers 1, 2, ..., 19
TRAFFIC_LANES_UND	Number of lanes under the structure	Discrete	Integers 0, 1, ..., 34
ROUTE_PREFIX	Classifies according to government level and throughput	Categorical	1- Interstate 2- US numbered highway 3- State highway 4- County highway 5- City street 6- Federal lands road 7- State lands road 8- Other
SERVICE_LEVEL	Identifies the designated level of service	Categorical	0- None of the below 1- Mainline 2- Alternate 3- Bypass 4- Spur 6- Business 7- Ramp, Wye, Connector, etc 8- Service and/or/unclassified frontage road

Table 2.2: Variable Description

Variable	Description	Type of Variable	Range/Level
MEDIAN_CODE	A number (code) classifying the median into one of four levels	Categorical	0- No median 1- Open median 2- Closed median with no barrier 3- Closed with non-mountable barrier
NAV_VERT	Minimum Vertical Clearance	Continuous	Closed as a 4 digit value. Includes the hundreds, tens, units, and tenths place. Measured in decimeters.
STRUCTURE_KIND	Designated the primary construction material	Categorical	0- Other 1- Concrete 2- Concrete continuous 3- Steel 4- Steel continuous 5- Prestressed concrete 6- Prestressed concrete continuous 7- Wood or timber 8- Masonry 9- Aluminum, Wrought Iron, or Cast Iron
MAX_SPAN	Maximum length between supports	Continuous	Coded as a 4 digit value. Includes the hundreds, tens, units, and tenths place. Measured in decimeters.
STRUCTURE_LEN	Overall length of structure	Continuous	Coded as a 4 digit value. Includes the hundreds, tens, units, and tenths place. Measured in decimeters.

Table 2.2: Variable Description

Variable	**Description**	**Type of Variable**	**Range/Level**
STRUCTURE_TYPE	Designated the structural design of the structure, or even the type of the structure itself	Categorical	00- Other 01- Slab 02- Stringer/Multi-beam or Girder 03- Girder and floor-beam system 04- Tee Beam 05- Box Beam or Girders - Multiple 06- Box Beam or Girder - Single or spread 07- Frame (except frame culverts) 08- Orthotropic 09- Truss - Deck 10- Truss - Thru 11- Arch - Deck 12- Arch - Thru 13- Suspension 14- Stayed Girder 15- Movable - Lift 16- Movable - Bascule 17- Movable - Swing 18- Tunnel 19- Culvert (includes frame culverts) 20- Mixed Types 21- Segmental Box Girder 22- Channel Beam

In Table 2.3 some statistics of the quantitative variables such as mean, standard deviation, minimum and maximum of variables in the data set are summarized. Also there are five qualitative variables in our models and their level's frequency is shown in Table 2.4. In this table N is the number of bridges multiplied by 25 (years).

Table 2.3: Summary Table

Statistic	N	Mean	St. Dev.	Min	Max
YEAR_BUILT	293,993	1,969.076	19.242	1,872	2,014
TRAFFIC_LANES_ON	293,993	2.342	1.261	1	19
TRAFFIC_LANES_UND	293,993	0.565	1.950	0	34
NAV_VERT	293,993	0.023	0.567	0.000	20.400
MAX_SPAN	293,993	10.682	10.540	0.000	229.800
STRUCTURE_LEN	293,993	41.767	63.799	5.800	1,823.900
OLD_DATE	293,993	2,003.584	7.062	1,992	2,015
NEW_DATE	293,993	2,004.596	7.056	1,993	2,016
AGE	293,993	34.509	19.381	2	145
INSPECTION_GAP	293,993	1.012	0.155	1	16
SUFFICIENCY_RATING_Beg	293,993	79.015	21.124	0.000	100.000
DROP	293,993	0.670	3.173	0.000	76.900
DETPRES	293,993	0.174	0.379	0	1

Table 2.4: Categorical Variables Frequency Table

Categorical Variables	**Levels**	**Frequency**
ROUTE_PREFIX	1	1221
	2	2152
	3	3561
	4	8838
	5	1368
	6	0
	7	0
	8	276
SERVICE_LEVEL	0	209
	1	16738
	2	59
	3	42
	4	50
	6	63
	7	243
	8	12

Table 2.4: Categorical Variables Frequency Table

Categorical Variables	**Levels**	**Frequency**
MEDIAN_CODE	0	15645
	1	1198
	2	268
	3	305
STRUCTURE_KIND	0	6
	1	8972
	3	5327
	5	2392
	7	611
	8	108
STRUCTURE_TYPE (23 levels; from 0 to 22)	1	1433
	2	7284
	4	2337
	5	282
	6	119
	11	61
	19	5900
	other levels	0

One of the major tasks is to transform the data from a set of 25 yearly data sets to a single large data set that includes all the years with changes in sufficiency ratings. Algorithm 1 gives the pseudocode for the script that performs the transformation.

We noticed a minor and ultimately inconsequential error that the minimum AGE is -1 and the maximum AGE is 142 years but it doesn't match with the year built. The minimum and maximum of AGE should be 2 and 145 respectively to match with the year built. So we looked in the code and we found out that in the read_bridge_data function in import script file, instead of defining AGE to be one more than the NEW_DATE minus the YEAR_BUILT, we set it to be the OLD_DATE minus the YEAR_BUILT. Therefore all AGEs are shifted by three. We wanted to fix this, but it was too late, since the data is huge and it takes days to run the code again and change all numbers. However, since shifting in a variable's value does not affect the model's predictions or significance of variables, the error is inconsequential and we still can work with the shifted AGE. The result would be the same. So we kept our shifted AGE. In our paper we will fix that.

As part of the data cleaning, we saw a bridge with zero inspection gaps that showed up multiple times. This appeared to be an error so we removed that bridge from the data frame.

Some variables had levels with few observations taking those levels. Since they could be considered similar to other levels, some were combined. For ROUTE_PREFIX, categories 6, 7, and 8 are combined into the level 8. For STRUCTURE_KIND, categories 1 and 2 are combined into 1; Categories 3 and 4 are combined into 3; Categories 5 and 6 are combined into 5; Categories 8 and 9 are combined into 8. This new data frame is named "**reduced_df**".

To make a logistic model, we needed to classify "DROP" whether "DROP" is zero or non-zero. Algorithm 2 gives the pseudo code for the script that adds a new variable "DETPRES" that shows the presence of deterioration.

After we added the new variable DETPRES in **reduced_df**, we made **logistic data frame** which is one of the main data frames we used in our modeling.

The second main data frame that we used in modeling is **Gamma data frame**. As we know for any random variable Y that has Gamma distribution, since it is a continuous variable, $P(Y = 0) = 0$. Thus for Gamma data frame we just need the observations for which the "DROP" is non-zero and deterioration is present. So the data frame in use for Gamma model is a subset of logistic data frame with all DETPRES$= 1$ or DROP> 0. So

Algorithm 1 Pseudocode for the script that performs the transformation

```
Require: A collection of yearly data frames.
  Take a subset of variables for use in models later.
  Add the year as a new variable to every observation in every data frame.
  Combine all the data frames.
  Remove rows with missing observations.
  Initialize empty vectors for all variables to be stored.
  Make index $n$ for the rows of the data frame. Initialized at 1.
  Make a list of all bridges
  for $i$ from 1 to number of bridges do
      make a temporary data frame that contains only rows from bridge $i$.
      Make index $j$ for the rows of the new temporary data frame. Initialized at 1.
      while Row $j$ has missing value and $j$ is less than the number of rows do
          $j = j + 1$                                          ▷ Go to the next row
      end while
      while $j < numberofrows + 1$ do           ▷ $j$ reaches the end of the data frame.
          Make two new variables OLD_CONDITION and OLD_DATE and set them to be
          the SUFFICIENCY_RATING and YEAR of row $j$ respectively.
          Until reaching the end of the data frame enter the while loop.
          Go to the next row.
          while $j < numberofrows + 1$ do            ▷ The stopping condition is false.
              Search for the next non-missing value.
              while Row $j$ has a missing value and the stopping condition is false. do
                  Go to the next row.
              end while
              if $j$ is greater than the number of rows. then        ▷ You reach the end.
                  STOP!
              else                                    ▷ You haven't reached the end yet
                  CURRENT_CONDITION = next non-missing value row's SUFFICIENCY_RATING.
                  CURRENT_DATE = next non-missing value row's YEAR.
                  DROPPING = OLD_CONDITION - CURRENT_CONDITION.
                  if DROPPING$>= 0$ then                  ▷ Condition did not improve.
                      Make a vector for row $n$ that contains all the variables we need
                      including DROP and INSPECTION_GAP.
                      $n = n + 1$.                   ▷ Go to the next row of the data frame
                  end if
              end if
          end while
      end while
  end for
  Construct a data frame with all DROP$>= 0$.  ▷ Take and turn all vectors into a data frame.
  Cast all categorical variables as factors.
```

Algorithm 2 Pseudocode for Adding the new variable to make the Logistic data frame.

```
Require: reduced_df data frame.
  Find out the sample size n; Number of rows of the required data frame.
  Initialize a vector of all zeros for DETPRES.
  Turn zeros into ones when deterioration presents.        ▷ A for loop is used.
  for $i$ from 1 to the sample size $n$ do
      if The DROP of the observation $i$ is greater than zero then
          Set DETPRES=1.
      else
          DETPRES=0.
      end if
  end for
  Add the new variable DETPRES to the data frame.
```

we set **Gamma_df** to be a subset of **reduced_df** with all DROP> 0.

CHAPTER 3

FITTED MODELS AND VARIABLE SELECTION

As we know, the Poisson model has only one parameter λ, but the other model (which is a combination of Bernoulli and Gamma random variables) has three parameters p, α and β. The model with more parameters has more complexity so it will more closely fit the particular data in training. Therefore the model with more parameters has more chance of over-fitting. One way to control this is dividing our data into two sets of training and testing, and evaluate the model on data different from what it is trained on. Hence we built our models based on training sets and tested them based on the testing sets. We set the training set for Logistic model to be 75-percent of the Logistic data frame and the remainder for the testing set. We named them **training_set_logistic** and **test_set_logistic** respectively. Then we eliminated STRUCTURE_NUMBER variable because the data frame was huge and we did not have access to super fast computers. With STRUCTURE_NUMBERs, the program took more than 3 days to run but at the end we did not get any result. (The computers ran out of memory; 16 Gigabyte was not enough.) Also we dropped the OLD_DATE, NEW_DATE and YEAR_BUILT from our data frame, since they were related to INSPECTION_GAP and AGE. These variables were set to be NULL in order to make the models from the rest of variables. The new training set is named **training_set_logistic2**.

Then we defined **training_set_gamma** which is the training set for Gamma model to be a subset of **training_set_logistic2** with all DROP> 0. Also we set the **test_set_gamma** to be a subset of **test_set_logistic** with all DROP> 0.

3.1 Logistic Model

To make the Logistic model first we set DROP to be NULL in the data frame because DETPRES is the response variable and we do not need the DROP in logistic model. Then we write the full logistic model with all the 13 variables. The model that we got which is named *fullmodel_logistic* is shown in the Table 3.1 with all the coefficients.

By looking at the summary of this model we see some of the variables are not significant and some of them are. Thus we tried stepwise variable seletion to get a possibly better model. Since we have some categorical variables with more than two levels, backward elimination method works better than forward selection method. This is because when we have categorical variables with more than two levels in our model the forward selection might wrongly show that our categorical variable is not significant [9]. Therefore we used backward elimination to get a smaller model[9]. Backward elimination method dropped TRAFFIC_LANES_ON and NAV_VERT. The result from backward method is *stepmodel_logistic*.

Based on smaller Akaike Information Criterion (AIC), we got the *stepmodel_logistic* out of *fullmodel_logistic*. Now we check it by ANOVA (Analysis of variance) and p-value = 0.4521 which is a large number so do not reject the null hypothesis (which is dropping TRAFFIC_LANES_ON and NAV_VERT), and support *stepmodel_logistic*.

Still in *stepmodel_logistic*, STRUCTURE_KIND is not significant. We thought it might be interactions between the levels of this variable. But since the data frame was huge

Table 3.1: Regression Results of Full Model Logistic

	Coefficients
(Intercept)	−3.3938717
factor(ROUTE_PREFIX)2	0.0169451
factor(ROUTE_PREFIX)3	−0.0459059
factor(ROUTE_PREFIX)4	−0.4588441
factor(ROUTE_PREFIX)5	−0.3182604
factor(ROUTE_PREFIX)8	−0.4499164
factor(SERVICE_LEVEL)1	0.8098013
factor(SERVICE_LEVEL)2	0.6781804
factor(SERVICE_LEVEL)3	0.3948153
factor(SERVICE_LEVEL)4	0.7683180
factor(SERVICE_LEVEL)6	0.6164090
factor(SERVICE_LEVEL)7	0.3257436
factor(SERVICE_LEVEL)8	0.1559793
TRAFFIC_LANES_ON	−0.0079939
TRAFFIC_LANES_UND	0.0404513
factor(MEDIAN_CODE)1	−0.1852925
factor(MEDIAN_CODE)2	0.0673280
factor(MEDIAN_CODE)3	−0.1912694
NAV_VERT	0.0015109
factor(STRUCTURE_KIND)1	1.0021030
factor(STRUCTURE_KIND)3	0.9550288
factor(STRUCTURE_KIND)5	0.7353310
factor(STRUCTURE_KIND)7	0.7211976
factor(STRUCTURE_KIND)8	1.3248433
factor(STRUCTURE_TYPE)2	0.0848590
factor(STRUCTURE_TYPE)4	−0.0354322
factor(STRUCTURE_TYPE)5	0.0903704
factor(STRUCTURE_TYPE)6	0.0951179
factor(STRUCTURE_TYPE)11	−0.1754007
factor(STRUCTURE_TYPE)19	−0.6044322
MAX_SPAN	−0.0047625
STRUCTURE_LEN	0.0004371
AGE	−0.0017637
INSPECTION_GAP	0.9253346
SUFFICIENCY_RATING_BEG	−0.0040583
Observations	220,494
Akaike Inf. Crit.	197,842.500

every way we tried to check the interactions took lots of time and it ran out of memory and stopped working before giving us a result. Thus we decided to find out how the proportion of DETPRES varies among levels of STRUCTURE_KIND in a way that is explained in Algorithm 3.

Algorithm 3 Pseudocode to see if proportion of DETPRES varies among levels of STRUCTURE_KIND.

```
Require: matrix A which is training_set_logistic2.
  Find out the levels of STRUCTURE_KIND.
  for i in levels of STRUCTURE_KIND do
      print levels.
      Print the average of DETPRESs for each level.
  end for
```

We observed that all proportions of the response variable being 1, are between 0.15 and 0.2 except for level 0. Then we checked if group zero has lots of observations or not. We saw that group 0 has just 18 observations which is very small compared to the size of data frame. Thus all the levels of STRUCTURE_KIND have almost the same effect on presence of deterioration in the way that we checked.

Then we did the same for STRUCTURE_TYPE to see if proportion of DETPRES varies among levels of STRUCTURE_TYPE or not. We observed that all proportions for STRUCTURE_TYPE are between 0.17 and 0.2, except for level 19, which has 85K observations. So STRUCTURE_TYPE is effective on DETPRES.

Therefore we decided to drop STRUCTURE_KIND and keep STRUCTURE_TYPE and make a new model. After we dropped STRUCTURE_KIND from *stepmodel_logistic*, we got a model named *Redmodel_logistic*. We checked by ANOVA and we saw that the p-value compared to the *stepmodel_logistic* is too small (p-value = 2.2e-16). We should reject the null based on the p-value. That means *stepmodel_logistic* was better and we do not need to drop STRUCTURE_KIND. But since when we drop STRUCTURE_KIND everything gets significant, we checked the models on the test set to see which one predicts better.

Then we thought there might be a correlation between our variables. For this reason we checked the correlation matrix for all continuous variables. We observed that X9 and X10, which are STRUCTURE_LEN and MAX_SPAN have relatively high correlation ($r = 0.82042742$). We decided to drop MAX_SPAN from our *fullmodel_logistic* and make a new model named *model_logistic* and repeat all the process on it again. After backward elimination on *model_logistic* we got a new model without NAV_VERT and MAX_SPAN named *smodel_logistic*.

The p-value of ANOVA comparing *model_logistic* and *smodel_logistic* was 0.8452 which is a large number. Hence *smodel_logistic* is a better model than *model-logistic*. Now we drop the STRUCTURE_KIND as we tried for *fullmodel_logistic*. But after we got this new model named *Rmodel_logistic* we checked the p-value of ANOVA and it was 2.2e-16 which is a very small number. Hence we reject the null and keep *smodel_logistic* compare to *Rmodel_logistic*.

Then it was time to check all of the models on the test set to see which model predicts better. We checked SSE of each model and we obtained the following results:

SSE(*fullmodel_logistic*) = 10393.81

SSE(*Redmodel_logistic*) = 10400.91

SSE(*stepmodel_logistic*) = 10393.84

SSE(*smodel_logistic*) = 10395.26

SSE(*Rmodel_logistic*) = 10402.05

SSE(*model_logistic*) = 10395.27

This shows that SSE of *smodel_logistic* is a little bit more but very close to the SSE of *stepmodel_logistic* and *fullmodel_logistic*. We preferred to keep *smodel_logistic* as our final logistic model because it does not have correlated variables. Furthermore, the test-set performance (SSE) is very close to the *stepmodel_logistic*. We renamed *smodel_logistic* to *finalmodel_logistic*.

In Table 3.2, *finalmodel_logistic* model is summarized along with its coefficients.

3.2 Gamma Model

To make the Gamma model first we set the **training_set_gamma** to be a subset of **training_set_logistic2** with all DROP> 0. Then we set DETPRES to be NULL in the data frame since DROP is the response variable for the Gamma model.

In Table 3.3, *fullmodel_gamma* is summarized with its coefficients.

Then we calculated the correlation matrix on our **training_set_gamma** and we observed that there is a relatively high correlation (Correlation = 0.81285040) between the

Table 3.2: Regression Results of Logistic Final Model

	Coefficients
(Intercept)	−3.3609934
factor(ROUTE_PREFIX)2	0.0198100
factor(ROUTE_PREFIX)3	−0.0433376
factor(ROUTE_PREFIX)4	−0.4518490
factor(ROUTE_PREFIX)5	−0.3169842
factor(ROUTE_PREFIX)8	−0.4610080
TRAFFIC_LANES_ON	−0.0092198
TRAFFIC_LANES_UND	0.0335640
factor(STRUCTURE_KIND)1	0.9789940
factor(STRUCTURE_KIND)3	0.9307112
factor(STRUCTURE_KIND)5	0.7070793
factor(STRUCTURE_KIND)7	0.7321097
factor(STRUCTURE_KIND)8	1.3061054
factor(STRUCTURE_TYPE)2	0.0492952
factor(STRUCTURE_TYPE)4	−0.0450203
factor(STRUCTURE_TYPE)5	0.0725919
factor(STRUCTURE_TYPE)6	−0.0151191
factor(STRUCTURE_TYPE)11	−0.2369287
factor(STRUCTURE_TYPE)19	−0.5803623
STRUCTURE_LEN	0.0002639
factor(SERVICE_LEVEL)1	0.7867188
factor(SERVICE_LEVEL)2	0.6494897
factor(SERVICE_LEVEL)3	0.3746947
factor(SERVICE_LEVEL)4	0.7489138
factor(SERVICE_LEVEL)6	0.5850863
factor(SERVICE_LEVEL)7	0.2955121
factor(SERVICE_LEVEL)8	0.1519145
factor(MEDIAN_CODE)1	−0.1889601
factor(MEDIAN_CODE)2	0.0566482
factor(MEDIAN_CODE)3	−0.1912564
AGE	−0.0016159
INSPECTION_GAP	0.9266020
SUFFICIENCY_RATING_BEG	−0.0043992
Observations	220,494
Akaike Inf. Crit.	197,842.500

Table 3.3: Regression Results of Full Model Gamma

	Coefficients
(Intercept)	−2.3886445
factor(ROUTE_PREFIX)2	−0.0100282
factor(ROUTE_PREFIX)3	−0.1451348
factor(ROUTE_PREFIX)4	0.3904420
factor(ROUTE_PREFIX)5	0.1083698
factor(ROUTE_PREFIX)8	1.3680943
TRAFFIC_LANES_ON	0.0662082
TRAFFIC_LANES_UND	−0.0073998
NAV_VERT	−0.0080975
factor(STRUCTURE_KIND)1	2.2826648
factor(STRUCTURE_KIND)3	3.0181367
factor(STRUCTURE_KIND)5	2.5356099
factor(STRUCTURE_KIND)7	3.2923866
factor(STRUCTURE_KIND)8	3.5240320
factor(STRUCTURE_TYPE)2	−0.8146550
factor(STRUCTURE_TYPE)4	−0.3049735
factor(STRUCTURE_TYPE)5	−0.7898207
factor(STRUCTURE_TYPE)6	−0.9199491
factor(STRUCTURE_TYPE)11	−0.1805673
factor(STRUCTURE_TYPE)19	−1.0346060
STRUCTURE_LEN	0.0007688
factor(SERVICE_LEVEL)1	0.8993690
factor(SERVICE_LEVEL)2	1.0027957
factor(SERVICE_LEVEL)3	1.0805396
factor(SERVICE_LEVEL)4	0.9608017
factor(SERVICE_LEVEL)6	0.8902926
factor(SERVICE_LEVEL)7	1.1506481
factor(SERVICE_LEVEL)8	1.5688987
factor(MEDIAN_CODE)1	−0.0057909
factor(MEDIAN_CODE)2	0.0891645
factor(MEDIAN_CODE)3	−0.1134823
MAX_SPAN	−0.0089077
AGE	0.0045925
INSPECTION_GAP	0.1210749
SUFFICIENCY_RATING_BEG	0.0045623
Observations	38,135
Akaike Inf. Crit.	155,308.900

two variables STRUCTURE_LEN and MAX_SPAN. Hence we decided to drop MAX_SPAN from the *fulmodel_gamma* and create a new model named *model-gamma*. Then we checked the summary of this model and we saw that NAV_VERT and MEDIAN_CODE are not significant. Thus we tried backward elimination to make a new model with a better AIC. Backward method dropped MEDIAN_CODE which was not significant from the *model-gamma*. We named the new model *Stepmodel_gamma* which contains all the 13 variables of *fullmodel_gamma* except MAX_SPAN and MEDIAN_CODE. The p-value of analysis of variance related to the two models *Stepmodel_gamma* and *model_gamma* was 0.2197 which is a relatively large number, so we didn't reject the nul and we picked *Stepmodel_gamma*. But then we saw that NAV_VERT is still in the model and still is not significant. We dropped NAV_VERT from the *Stepmodel_gamma* and made a new model named *Redmodel_gamma*. Checking the summary of *Redmodel_gamma* we saw that all the variables became significant. And also the p-value of ANOVA related to *Redmodel_gamma* and *Stepmodel_gamma* was 0.459 which was large. Therefore we should not reject the null and we should accept the *Redmodel_gamma*.

STRUCTURE_KIND is significant but not much, so we decided to check the box plot of all the levels of this categorical variable against DROP. As we see the box plot in Figure 3.1, different levels of STRUCTURE_KIND do not have similar affects on DROP. So this variable is effective on DROP and it must be kept in the model.

Then it was time to check all of the models on the test set to see which model predicts better. We checked SSE of each model and we obtained the following results:

SSE(*fullmodel_gamma*) = 561186.9

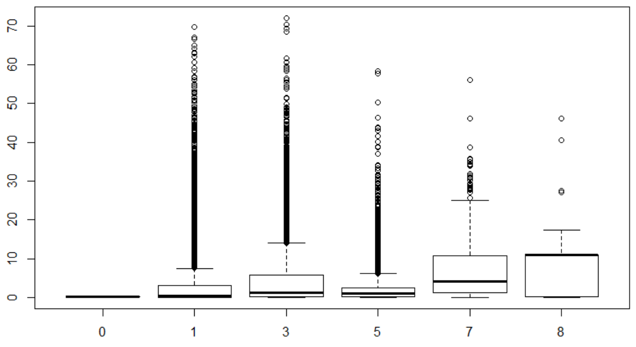

Figure 3.1: Box Plot for STRUCTURE_KIND

SSE(*Redmodel_gamma*) = 562069.4

SSE(*Stepmodel_gamma*) = 562074.8

SSE(*model_gamma*) = 562214.2

These numbers show that the SSE of *Redmodel_gamma* is strictly less than the SSE of *model_gamma*, *Stepmodel_gamma*, and *Redmodel_gamma*. The only model that has less SSE than *Redmodel_gamma* is the *fullmodel_gamma* that contains highly correlated variables. Since the difference between the SSE of *fullmodel_gamma* and *Redmodel_gamma* is relatively very small (0.1 percent) and *Redmodel_gamma* does not have high correlation between its variables, we decided to pick *Redmodel_gamma* as our final gamma model and we renamed it as *finalmodel_gamma*.

In Table 3.4, *finalmodel_gamma* is summarized with its coefficients.

Table 3.4: Gamma Regression Results

	Coefficients
(Intercept)	−2.4214972
factor(ROUTE_PREFIX)2	0.0166055
factor(ROUTE_PREFIX)3	−0.1187957
factor(ROUTE_PREFIX)4	0.4219894
factor(ROUTE_PREFIX)5	0.1336968
factor(ROUTE_PREFIX)8	1.3850939
TRAFFIC_LANES_ON	0.0591078
TRAFFIC_LANES_UND	−0.0194806
factor(STRUCTURE_KIND)1	2.2868109
factor(STRUCTURE_KIND)3	3.0183557
factor(STRUCTURE_KIND)5	2.5264149
factor(STRUCTURE_KIND)7	3.3593237
factor(STRUCTURE_KIND)8	3.5366407
factor(STRUCTURE_TYPE)2	−0.8769120
factor(STRUCTURE_TYPE)4	−0.3230127
factor(STRUCTURE_TYPE)5	−0.8090652
factor(STRUCTURE_TYPE)6	−1.1081179
factor(STRUCTURE_TYPE)11	−0.3024816
factor(STRUCTURE_TYPE)19	−0.9875559
STRUCTURE_LEN	0.0003616
factor(SERVICE_LEVEL)1	0.9033829
factor(SERVICE_LEVEL)2	1.0107013
factor(SERVICE_LEVEL)3	1.0952076
factor(SERVICE_LEVEL)4	0.9799581
factor(SERVICE_LEVEL)6	0.8882317
factor(SERVICE_LEVEL)7	1.1519028
factor(SERVICE_LEVEL)8	1.5957498
AGE	0.0048954
INSPECTION_GAP	0.1246739
SUFFICIENCY_RATING_BEG	0.0038034
Observations	51,052
Akaike Inf. Crit.	208,792.700

3.3 Poisson Model

To make the Poisson model first we set DETPRES to be NULL in **training_set-logistic2**, since DROP is the response variable for the Poisson model. In Poisson model since DROP is a Poisson variable, it is a discrete random variable but as we see in the data frame DROP is a continuous random variable. Thus, we decided to round DROPs in our training set to make them integers. This process makes DROP become a suitable Poisson random variable. Before rounding, the DROPs AIC of our models were infinity so we could not change our full model based on AIC. In the next step, we wrote the full Poisson model with all the 13 variables. We named this model *fullmodel_Poisson*. We observed the summary of the *fullmodel_Poisson* and we saw that there are some insignificant variables in the model. We checked the correlation matrix on **training_set_logistic2** and we noticed that there is a relatively high correlation (Correlation = 0.82042742) between STRUCTURE_LEN and MAX_SPAN. Thus we dropped MAX_SPAN from *fullmodel_poisson*, and we made a new model named *Rmodel_poisson*. Still we had some non-significant variables in *Rmodel_poisson*. Therefore we tried backward elimination on *Rmodel_poisson*, but this method did not drop any other variable from *Rmodel_poisson*. Hence our final Poisson model is *Rmodel_poisson* and we renamed it to *finalmodel_poisson*.

In Table 3.5, *finalmodel_Poisson* and *fullmodel_Poisson* are summarized with their coefficients.

Table 3.5: Poisson Regression Results

Variable	Full Model	Final Model
(Intercept)	−1.189e+01	−1.177e+01
factor(ROUTE_PREFIX)2	1.806e-02	3.060e-02
factor(ROUTE_PREFIX)3	−1.440e-01	−1.331e-01
factor(ROUTE_PREFIX)4	4.707e-02	7.495e-02
factor(ROUTE_PREFIX)5	−1.546e-01	−1.471e-01
factor(ROUTE_PREFIX)8	1.182e+00	1.151e+00
TRAFFIC_LANES_ON	3.401e-02	3.009e-02
TRAFFIC_LANES_UND	2.724e-02	2.376e-03
NAV_VERT	−8.647e-04	−1.566e-02
factor(STRUCTURE_TYPE)2	−7.528e-01	−8.680e-01
factor(STRUCTURE_TYPE)4	−3.067e-01	−3.436e-01
factor(STRUCTURE_TYPE)5	−6.754e-01	−7.377e-01
factor(STRUCTURE_TYPE)6	−6.647e-01	−1.031e+00
factor(STRUCTURE_TYPE)11	−3.836e-01	−5.632e-01
factor(STRUCTURE_TYPE)19	−1.600e+00	−1.521e+00
factor(STRUCTURE_KIND)1	9.455e+00	9.366e+00
factor(STRUCTURE_KIND)3	1.018e+01	1.009e+01
factor(STRUCTURE_KIND)5	9.464e+00	9.362e+00
factor(STRUCTURE_KIND)7	1.033e+01	1.036e+01
factor(STRUCTURE_KIND)8	1.094e+01	1.088e+01
STRUCTURE_LEN	1.205e-03	6.677e-04
MAX_SPAN	−1.630e-02	0
factor(SERVICE_LEVEL)1	1.952e+00	1.861e+00
factor(SERVICE_LEVEL)2	1.855e+00	1.743e+00
factor(SERVICE_LEVEL)3	1.722e+00	1.645e+00
factor(SERVICE_LEVEL)4	1.967e+00	1.894e+00
factor(SERVICE_LEVEL)6	1.877e+00	1.761e+00
factor(SERVICE_LEVEL)7	1.811e+00	1.687e+00
factor(SERVICE_LEVEL)8	1.429e+00	1.405e+00
factor(MEDIAN_CODE)1	−1.600e-01	−1.758e-01
factor(MEDIAN_CODE)2	1.291e-01	8.539e-02
factor(MEDIAN_CODE)3	−1.631e-01	−1.599e-01
AGE	2.736e-03	3.336e-03
INSPECTION_GAP	3.591e-01	3.613e-01
SUFFICIENCY_RATING_BEG	2.553e-03	1.489e-03
Observations	220,494	
Akaike Inf. Crit.	838,988.800	

CHAPTER 4

DETERIORATION PREDICTION

4.1 SEGMENT DATA FRAME

The purpose of the models is not to predict a single deterioration at a time. We want to show a sequence of how a bridge will deteriorate over years. When we look at the SUFFICIENCY_RATING numbers we see that sometimes they go up. This means that some parts of the bridge that were not in good condition have been fixed or replaced. On the other hand, we want to study the natural deterioration of the bridges. So we decided to make segments in which we only have non-increasing SUFFICIENCY_RATINGs since, normally, if the bridge exists without getting fixed, then the sufficiency rating would not get better. To manipulate the data into these non-increasing segments, we dropped all the increases of SUFFICIENCY_RATINGs by dropping the rows in which the SUFFICIENCY_RATING was increasing. Hence, in the new data frame we did not have such rows (Years). Therefore, when we wrote the function that makes the segments we did not need to check SUFFICIENCY_RATINGs if they increase or not. We just looked for years that do not exist in the data frame. Algorithm 4 explains how we created the segment data frame.

After this function that makes segments, we created our **segment_df** data frame by passing **Reduced_df** described in the second chapter, into this function. After generating the segmented data frame, it was time to find out how a bridge deteriorates over years under each model. First we changed the name of our models from *finalmodel_logistic*, *finalmodel_gamma*, *finalmodel_Poisson*, respectively to *model_logistic*, *model_gamma*, and

Algorithm 4 Pseudocode for the script that performs the segments

```
Require: df data frame.
  Start make_segment_df function using df data frame.
  Initialize empty vectors for STRUCTURE_NUMBER, segmentBegins, and segmentEnds.
  Make index $n$ to store all segment's information. Initialized at 1.
  Make bridgeIDs vector.                                    ▷ Vector of all bridges.
  for $i$ from 1 to number of bridges do
      make a temporary data frame that contains only rows from bridge $i$.
      Make index $j$ for the rows of the temporary data frame. Initialized at 1.
      The stopping condition of bridge is False.
      while Did not go to the next bridge yet do
          segmentBegin is the Old_Date of row j.
          Until reaching the end of the temporary data frame enter the while loop.
          while Did not get to the next segment do
              segmentEnd is the New_Date of row $j$.
              if $j + 1 > numberofrowsof$ tempdf then   ▷ You passed the end of the data frame.
                  End the segment and the bridge.
              else                                  ▷ have not reached the End of the bridge yet.
                  if The Old_Date of row $j + 1$ does not match the New_Date of row $j$ then
                      End the segment.
                  end if
              end if
              Go to the next row.
          end while
          Store this segment's information; STRUCTURE_NUMBER, segmentBegin and segmentEnd.
          $n = n + 1$                                       ▷ Go to the next segment
      end while
  end for
  Make newdf data frame of all segments of all bridges with STRUCTURE_NUMBER, segmentBegins and segmentEnds.
  Return newdf.
```

model_Poisson. The goal is to compare the mixed *Logistic_Gamma* model against *Poisson* model to see which model works better at predicting bridge deterioration.

4.2 Predicted Deterioration for Mixed Logistic_Gamma Model

We start with our mixed-type Logistic_Gamma distribution. This model first determines if there is any drop in SUFFICIENCY_RATING, with a logistic random variable $X \sim Bernoulli(p)$. Such that if DROP> 0, $X = 1$ and if DROP≤ 0, $X = 0$. It then finds the amount of deterioration with a Gamma random variable $Y \sim Gamma(\alpha, \beta)$. Finally, the random variable Z which is our mixed model's random variable is generated by XY. Parameter α in Gamma distribution is fixed by the assumption of the Generalized Linear Model (GLM). Hence, in this model each bridge deteriorates with probability of p and the amount of each deterioration (if non-zero) can be estimated by the expected value of the Gamma distribution, which is $\mu = \frac{\alpha}{\beta}$.

In order to be able to simulate, first we needed to write a function which gets the special bridge's information and the coefficients of our models and calculates the predicted probability of deterioration ($\hat{p}$, logistic distribution's parameter), and the predicted deterioration amount ($\hat{\mu}$, Gamma distribution's expected value), for the given bridge. The name of this function is get_eta. In this section get_eta function that finds the Predicted Deterioration of this model is explained.

First we found the dispersion parameter and $\hat{\alpha}$, which are the fixed parameters of Gamma distribution to use in simulation.

If $X \sim Gamma(\alpha, \beta)$, then $f(x) = (\beta^{\alpha}/\Gamma(\alpha))x^{\alpha-1}e^{x\beta}$; $x > 0$, $\alpha > 0$ and $\beta > 0$.

We know $E[X] = \alpha/\beta$ and var$(X) = \alpha/\beta^2$. Hence, $\beta = \alpha/E[X]$ and

$$\text{var}(X) = \frac{\alpha}{\beta^2} = \frac{\alpha}{(\frac{\alpha}{E[X]})^2} = \frac{E^2[X]}{\alpha}.$$

Also, we have var$(X) = E[X^2] - E^2[X] = \frac{E^2[X]}{\alpha}$, so $\alpha = \frac{E^2[X]}{E[X^2]-E^2[X]}$. We define dispersion parameter ϕ to be $\phi = \frac{1}{\alpha}$. Then

$$\phi = \frac{1}{\alpha} = \frac{E[X^2] - E^2[X]}{E^2[X]} = \frac{\text{var}(X)}{E^2[X]}.$$

By the method of moments we have $\widehat{E[X]} = \hat{\mu}$ and $\widehat{\text{var}(X)} = \frac{\Sigma(X_i-\hat{\mu})^2}{(n-p)}$; where p is the number of variables and n is the number of observations.

Therefore, the estimated dispersion parameter by method of moments is

$$\hat{\phi} = \frac{1}{\hat{\alpha}} = \frac{\Sigma(X_i - \hat{\mu})^2}{(\hat{\mu})^2(n-p)}.$$

Before using the function we noticed that all the categorical variables have the word “factor” in their names in the models. So first we changed the name of all categorical variables to their own names without the word “factor”. Then we defined two vectors of variable names for both logistic and gamma models. We defined a vector named “is_categorical”, which assigns “zero” to all quantitative variables including intercept, and “one” to all categorical variables. For each categorical variable of our bridge we need to clarify the level and corresponding coefficient. Therefore, we described bridges in terms of their explanatory variables. After describing the bridge and finding $\beta\_logistic$ and β_gamma which are the coefficients arising from the logistic and gamma models for each bridge, we wrote the function of calculating the predicted probability of deterioration for this bridge and predicted deterioration amount. To find those predictions, we know that for logistic distribution we have

$$\eta = X^T\beta$$

and

$$\hat{p} = \frac{e^\eta}{1+e^\eta},$$

in which β is the coefficients of *model_logistic* and X is the vector of values of variables in logistic model for each bridge and $\hat{p}$ is the predicted probability of deterioration. Also we know when using the log-link for gamma distribution we have

$$\eta = X^T\beta$$

and

$$\hat{\mu} = e^\eta,$$

in which β is the coefficients of *model_gamma* and X is the vector of values of variables in gamma model for each bridge and $\hat{\mu}$ is predicted amount of deterioration.

This function, which is named get_eta is described in Algorithm 5.

4.3 Predicted Sufficiency Rating For Mixed Logistic_Gamma Model

After we wrote the function which predicts the probability and amount of deterioration of a given bridge, we needed to write another function that calculates the predicted sufficiency rating for our given bridge in our given segment. This function which is named generate_history is explained in this section. This function's job is to get a bridge's information, coefficients of logistic model, coefficients of gamma model, $\hat{\alpha}$, number of iterations, and

Algorithm 5 Pseudo code For get_eta Function

```
Require: model_logistic and model_gamma.
    Start get_eta function by passing a bridge, coefficients of logistic and gamma model into
    it.
    Change the names of categorical variables to match their names in data frame.
    Define a vector of variables_logistic names including the "intercept".
    Define a vector of variables_gamma names including the "intercept".
    Define is_categorical vector that assigns 1 to all categorical variables
    and zero for other variables and intercept.
    Describe a bridge in terms of the explanatory variables and coefficients of model_logistic
    ▷ beta_logistic.
    Describe a bridge in terms of the explanatory variables and coefficients of model_gamma
    ▷ beta_gamma.
    $\eta_l = 0$.                                        ▷ Initialize eta logistic at zero
    for i in variables_logistic do
        if i is a categorical variable then                          ▷ $\eta_l$=$\eta$_logistic.
            $\eta_l = \eta_l + \beta_i$;                               ▷ $\beta$ is $\beta$_logistic here.
        else                                                ▷ The variable is not categorical.
            $\eta_l = \eta_l + x_i\beta_i$                             ▷ $\beta$ is $\beta$_logistic here.
        end if
        Print $\eta_l$.
    end for
    $\hat{p} = \frac{e^{\eta_l}}{1+e^{\eta_l}}$           ▷ Calculate the predicted probability of deterioration
    Initialize $\eta_g = 0$
    for i in variables_gamma do
        if i is a categorical variable then                           ▷ $\eta_g$=$\eta$_gamma.
            $\eta_g = \eta_g + \beta_i$                                 ▷ $\beta$ is $\beta$_gamma here.
        else                                              ▷ The variable i is not categorical.
            $\eta_g = \eta_g + x_i\beta_i$                              ▷ $\beta$ is $\beta$_gamma here.
        end if
        Print $\eta_g$.
    end for
    $\hat{\mu} = e^{\eta_g}$.                    ▷ Calculate the predicted amount of deterioration.
    $predictions = (\hat{p}, \hat{\mu})$.
    Return Predictions.
```

the segment-length, and return a matrix of sufficiency rating history which is named sufficiency_history. The sufficiency_history matrix has as many rows as number of iterations and as many columns as the years of the specified segment. All of the rows of the first column of this matrix are the first year of the segment's sufficiency rating for that specified bridge. Each row has all years of the segment's predicted sufficiency ratings for each simulation.

The beginning of each row of this matrix starts with the sufficiency rating of the first year of the segment for that specified bridge. For the next year (which is the next column of the same row) $\hat{p}$ and $\hat{\mu}$ that we found from the model are used to calculate a random deterioration amount which is the DROP. After that, by deducting this random simulated DROP from the previous year's sufficiency rating, it finds a number for this year's sufficiency rating. It then replaces the new sufficiency rating as the beginning sufficiency rating and adds one year to the age of the bridge, for the next step. Continuing the same way, the function calculates the sufficiency ratings for all years of the segment for that iteration. After it finishes the first row, it goes to the next row and does the same for the second simulation. It continues till it reaches the end of the last iteration. At the end, the function returns the sufficiency_history matrix. This function, which is named generate_history is described in Algorithm 6.

At the end of the generate_history function, the sufficiency_history matrix which has multiple rows of simulated sufficiency rating for each year in its columns is returned. To find the predicted sufficiency rating of each year in the segment we take the median of all sufficiency ratings in the related column of the sufficiency_history matrix.

Algorithm 6 Pseudo code For generate_history Function

```
Require: $model\_logistic$ and $model\_gamma$.
  num_iter = number of iterations
  segment_length = $segmentEnd - segmentBegin + 1$.        ▷ years in segment
  Pass a bridge, coefficient of logistic and gamma model, $\hat{\alpha}$, num_iter and
  segment_length into generate_history function.
  Initialize an empty matrix named sufficiency_history with
  number of rows = num_iter and number of columns = segment_length.
  for i in 1 to num_iter do
      Define a bridge named tempBridge. ▷ For each iteration start from the bridge's data.
      Store the sufficiency rating for the first year in the segment.
      for k in 2 to segment-length do                 ▷ All years after the first year.
          pass a row from the data frame into get_eta function.      ▷ Find $\hat{p}$ and $\hat{\mu}$.
          if The uniform randomly generated number< $\hat{p}$ then   ▷ The bridge deteriorates
              $\hat{\beta} = \hat{\alpha}/\hat{\mu}$
              Simulate the Gamma portion using $\hat{\alpha}$ and $\hat{\beta}$.
              Decrease the sufficiency rating by the deterioration amount.
              Increase the age of the bridge by one year.
          else                                                 ▷ No deterioration.
              Increase the age of the bridge by one year.
          end if
          Put the numbers in their place in the sufficiency_history matrix.
      end for
  end for
  Return sufficiency_history matrix.
```

4.4 DETERIORATION CURVES FOR MIXED LOGISTIC_GAMMA MODEL

In the next step, we wrote a function that plots the sufficiency ratings versus years for any bridge with any defined segment length. Thus it is very useful to predict the deterioration after a given number of years have passed. We pass a bridge's information, coefficients of *model_logistic*, coefficients of *model_gamma*, $\hat{\alpha}$, number of iterations, and the segment_length which is the number of all years in a segment, into this function. The function generates three curves for each segment's predicted sufficiency ratings versus years.

One of the curve's predicted sufficiency ratings are the medians of each column of the sufficiency_history matrix for each year which is graphed by black color. The second curves's predicted sufficiency ratings are the first quartile of each column of the sufficiency_history matrix for each year which is plotted by blue. Finally, the last curve's predicted sufficiency ratings are the third quartile of each column of the sufficiency_history matrix for each year which is plotted by red.

At the beginning the sufficiency history matrix is generated by passing a bridge's information, coefficients of *model_logistic*, coefficients of *model_gamma*, $\hat{\alpha}$, number of iterations, and segment_length into the generate_history function. Then it finds the median, first quartile, and the third quartile of each column of this sufficiency_history matrix. It makes a vector of all medians, a vector of all first quartiles, and a vector of all third quartiles for the years of the specified segment. It makes a sequence from one to the number of years in the segment and names it Years. At the end it plots median, Q1s and Q3s versus Years in the colors mentioned above.

I described this function, which is named generate_curve function, in Algorithm 7.

Algorithm 7 Pseudo code For generate_curve Function

```
Require: A bridge, logistic and gamma model, $\hat{\alpha}$, number of iterations, segment-length.
  Pass a bridge, logistic and gamma coefficients, $\hat{\alpha}$, number of iterations and
  segment-length into generate-history function.
  Initialize empty vectors median, Q1s, Q3s.
  for k in 1 to segment-length do ; k is index over time.
      Calculate Q1s[k]                         ▷ Vector of first quartiles
      Calculate medians[k]                     ▷ Vector of medians
      Calculate Q3s[k]                         ▷ Vector of third quartiles
  end for
  Generate Years.        ▷ Sequence of numbers from 1 to number of years of a segment.
  Plot Q1s vs. Years in Blue. Set the Y-axis to be "Sufficiency Rating".
  Plot medians vs. Years in Black.
  Plot Q3s vs. Years in Red.
  Make the legend in the bottom left.
```

Then we can use this function to simulate and generate curves. We need to define a bridge, set a number of iterations and a segment length and pass them in generate_curve function to generate a curve that predicts deteriorations throughout and at the end of the segment.

4.5 Predicted Deterioration for Poisson Model

In Poisson model, the "DROP", which is the deterioration amount of each bridge at each inspection, is a Poisson random variable. We know that a Poisson random variable can only take integer values 0, 1, 2, 3, etc. The parameter of Poisson random variable is λ. For variable $Y \sim Poisson(\lambda)$, the probability density function is

$$f(k) = P(Y = k) = \frac{e^{-\lambda}\lambda^k}{k!}; k = 0, 1, 2, ...$$

.

$$E[Y] = \text{var}(Y) = \lambda$$

Suppose that the mean of a Poisson random variable Y depends on a set of explanatory variables x_1, x_2, x_3, ..., and x_p, and $\beta_0, \beta_1, \beta_2, ..., \beta_p$ are the coefficients of the Poisson model. Since the mean is always positive, a common way of modeling the conditional expectation of Y is to set it to be the exponential function of the response. Hence we define:

$$\lambda = E[Y \mid x_1, x_2, x_3, ..., x_p] = e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_p x_p}$$

By taking the natural logarithm of each side we have,

$$\ln(\lambda) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p = \eta$$

This uses the natural log as the link function for a Poisson regression. Therefore:

$$E[Y] = Var(Y) = e^{\eta} = \lambda$$

Since in our model “DROP” is the Poisson random variable, the expected value of “DROP” can be estimated by λ. Hence, in order to simulate and predict the deterioration in a segment, first we need a function that takes a bridge’s information and the coefficients of our Poisson model and calculates the predicted amount of deterioration $\hat{\lambda}$, which is an estimation for the Poisson distribution’s expected value, for the given bridge. This function is named get_lambda function.

Before starting the function we noticed that all the categorical variables have the word “factor” in their names in the model. Thus we started this function with changing the name of all categorical variables to their own names without the word “factor”. Then we defined a vector of variable names which are in use in Poisson model. We defined a vector named “is_categorical”, which assigns “zero” to all quantitative variables including intercept, and “one” to all categorical variables. Since we had categorical variables in each bridge’s data, it is necessary to assign the regression coefficient corresponding to the bridge’s level of each variable. Therefore, we described bridges in terms of their explanatory variables. After describing the bridge, we found $\beta\_Poisson$, which is the coefficients arising from the Poisson model for each bridge. At the end, we calculated $\hat{\lambda}$ which is an exponential function of $\beta\_Poisson$. As we explained, $\hat{\lambda}$ is the expected value of deterioration of a bridge in this model.

This function, which is named get_lambda is described in Algorithm 8.

Algorithm 8 Pseudo code For get_lambda Function

```
Require: model_Poisson.
  Start get_lambda function by passing a bridge and coefficients of Poisson model into it.
  Change the names of categorical variables to match their names in the data frame.
  Define a vector of variables_Poisson names including the “intercept”.
  Define is_categorical vector that assigns 1 to all categorical variables
  and zero for other variables and intercept.
  Describe a bridge in terms of the explanatory variables and coefficients of
  model_Poisson.                                              ▷ beta_Poisson.
  $\eta_P = 0$                                    ▷ Initialize eta_Poisson at zero.
  for i in variables_Poisson do
      if i is a categorical variable then
          $\eta_P = \eta_P + \beta_i$                     ▷ $\beta$ is $\beta$_Poisson here.
      else                                    ▷ The variable is not categorical.
          $\eta_P = \eta_P + x_i\beta_i$                  ▷ $\beta$ is $\beta$_Poisson here.
      end if
  end for
  $\hat{\lambda} = e^{\eta_P}$                    ▷ Find predicted amount of deterioration.
  Return $\hat{\lambda}$.
```

4.6 Predicted Sufficiency Rating For Poisson Model

After we wrote the function which predicts the amount of deterioration of a given bridge, we needed to write another function that calculates the predicted sufficiency rating for our given bridge in our given segment. I explain this function which is named Poisson_Pred_Suff in this section.

This function takes a bridge's information, coefficients of Poisson model, number of iterations, and the segment-length. The purpose of this function is to simulate the sufficiency rating for each year of the segment, as many times as the number of iterations that we passed into it, and calculate the matrix of sufficiency rating history which is named suff_history matrix. Finally, it finds the medians, first quartile, and the third quartile of each column of this matrix, plots these medians, first quartiles, and the third quartiles versus all years of a segment for each bridge. At the end it returns the medians of all columns of the suff_history matrix, which are the predicted sufficiency ratings for those related years (column) in the segment.

The suff_history matrix has as many rows as the number of iterations and as many columns as the years of the specified segment. All of the rows of the first column of this matrix is the first year of the segment's sufficiency rating for that specified bridge. Each row includes all years of the segment's predicted sufficiency ratings for each simulation.

The beginning of each row of this matrix starts with the sufficiency rating of the first year of the segment for that specified bridge. For the next year (which is the next column of the same row) $\hat{\lambda}$ that we found from the model is used to generate a random deterioration amount which is the "Deterioration". By deducting this random simulated deterioration from the previous year's sufficiency rating it finds a number for this year's sufficiency rating. Next it replaces the new sufficiency rating as the beginning sufficiency rating and it adds one year to the age of the bridge, for the next step.

It continues the same way to calculate the sufficiency ratings for all years of the segment for that iteration. After it finishes the first row, it goes to the next row and starting from the defined bridge's information, does the same for the second simulation. It continues till it reaches the end of the last iteration. This way it generates the suff_history matrix.

In the next step, it finds the median, first quartile and the third quartile of each column of this suff_history matrix and gives us the vectors of sufficiency ratings for the years of the segment for that bridge. Then it plots them versus the Years of the segment. From this we can see how the bridge will deteriorate through the segment and at the end of it.

The vector of medians has the predicted sufficiency ratings for all the years of the segment for that bridge. At the end the Poisson_Pred_Suff function returns the predicted sufficiency ratings for the years of the segment which is the median vector that it made. This function is described in Algorithm 9. We can use this function to simulate and generate curves. We need to define a bridge, a number of iterations and a segment length and pass them in Poisson_Pred_Suff function to generate a curve that predicts deteriorations throughout and at the end of the segment.

Example 4.1. *We choose the first bridge out of the* ***logistic_df*** *data frame and plot the curves of both models as we see in Figures* (4.1) *and* (4.2). *We see the medians of both mod-*

els are almost the same, but the first and third quartiles of the SUFFICIENCY_RATINGs for the Poisson *model are very close to the median.*

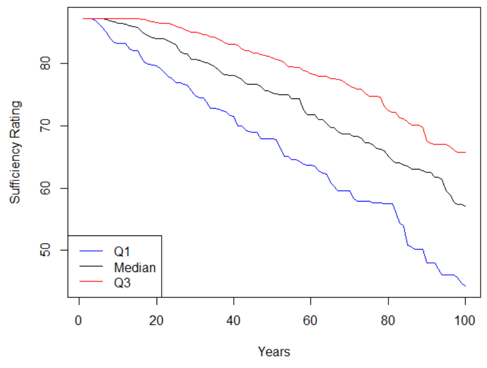

Figure 4.1: Logistic_Gamma Predictions for Example 4.1

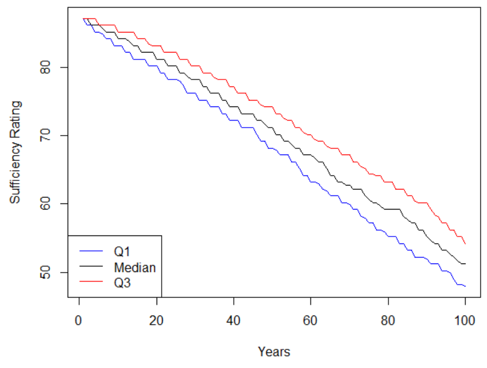

Figure 4.2: Poisson Predictions for Example 4.1

Algorithm 9 Pseudo code For Poisson_Pred_Suff Function

```
Require: model_Poisson.
  num_iter = number of iterations.
  segment_length=$segmentEnd - segmentBegin + 1$.                ▷ Years in segment.
  Pass a bridge, coefficient of model_Poisson, num_iter and
  segment_length into Poisson_Pred_Suff function.
  Initialize an empty num_iter by segment_length matrix named suff_history.
  for i in 1 to num_iter do                              ▷ i is index over number of rows.
      Define a bridge named tempBridge. ▷ For each iteration start from the bridge's data.
      Store the sufficiency rating for the first year in the segment.
      for k in 2 to segment-length do                ▷ All years in segment after the first year.
          Pass a row from the data frame to the get_lambda function to get $\hat{\lambda}$.
          Randomly generate Poisson variable using $\hat{\lambda}$. ▷ simulated Deterioration amount.
          Decrease the sufficiency rating by the deterioration amount.
          Increase the age of the bridge by one year.
          Put the numbers in their place in the suff_history matrix.
      end for
  end for
  Initialize empty vectors median, Q1s, Q3s.
  for k in 1 to segment-length do
      Calculate Q1s[k].                                          ▷ Vector of first quartiles
      Calculate medians[k].                                          ▷ Vector of medians
      Calculate Q3s[k].                                          ▷ Vector of third quartiles
  end for
  Generate Years.           ▷ Sequence of numbers from 1 to number of years of a segment.
  Plot Q1s vs. Years in Blue. Set the Y-axis to be "Sufficiency Rating".
  Plot medians vs. Years in Black.
  Plot Q3s vs. Years in Red.
  Make the legend in the bottom left.
  Set medians to be predicted sufficiency ratings of the years in the segment.
  Return Pois_Pred_Suff.        ▷ Predicted sufficiency ratings of the years in the segment.
```

CHAPTER 5

COMPARISON

5.1 SUM OF SQUARED ERRORS

In this section we compare our two models' sum of squared errors (SSE) of prediction to see which one is smaller and we also we look at some curves and examples to compare these two models. For all the bridges in our data frame we have all the sufficiency ratings of all years. For each bridge we can predict a sufficiency rating for each year using the functions that we created in Chapter 4. Hence, each bridge has a real sufficiency rating from the data frame (Y) and a predicted one $(\hat{Y})$ from the simulations. For each bridge, the error of predicting the sufficiency rating, is $e_i = Y_i - \hat{Y}_i$, where i is the index over years of each segment. The squared errors of prediction for that bridge in a segment is:

$$\sum_{i=1}^{segment\_length} (Y_i - \hat{Y}_i)^2.$$

Hence, we find the SSE's of all segments of all bridges and then we add them all together to find the SSE of the model.

The script that finds the SSE of the mixed *Logistic_Gamma* model is explained in Algorithm 10 and the script that finds the SSE of the *Poisson* model is explained in Algorithm 11.

The results of calculating SSE for the models were:

SSE of the mixed *Logistic_Gamma* model = 9,570,996

SSE of *Poisson* model = 36,092,381.

Algorithm 10 Pseudocode for the script that finds SSE of mixed Logistic_Gamma model.

```
Require: logistic_df and segment_df data frame and generate_history function.
  num_iter =100                                          ▷ Number of times to simulate.
  Make bridgeIDs vector.                                 ▷ A vector of all bridges.
  Initialize sse at zero.
  for m from 1 to number of bridges do
      Print m.               ▷ To make sure the sse is for all of the bridges at the end.
      Make the temporary data frame that contains only rows from bridge m of
      the logistic_df.                                   ▷ temp_bridge_df
      Make the temporary data frame that contains only rows from bridge m of
      the segment_df.                                    ▷ temp_segment_df.
      for j from 1 to number of rows of temp_segment_df do
          Find segment_length.                   ▷ How many years the segment lasts.
          Make temp_bridge_in_segment_df         ▷ subset of temp_bridge_df for
          the $j$-th segment.
          Extract tempBridge.           ▷ The bridge at the beginning of the segment.
          Pass tempBridge, coefficients of models, $\hat{\alpha}$, nume_iter and segment_length
          into generate_history function.        ▷ Make sufficiency_history matrix.
          Initialize empty vector medians.
          for k from 1 to segment_length do
              Set the $k$-th element of the medians vector equal to the median of the $k$-th
              column of the sufficiency_history matrix.
          end for
          Pred_Suff = medians
          Y = SUFFICIENCY_RATING_BEG of the temp_bridge_in_segment_df.
          sse = sse $+ \sum(Y - Pred\_Suff)^2$
      end for
  end for
  Print sse.
```

Algorithm 11 Pseudocode for the script that finds SSE of Poisson model.

```
Require: logistic_df and segment_df data frame and generate_history function.
  num_iter = 100                                   ▷ Number of times to simulate.
  Make vector.                                     ▷ Vector of all bridges.
  Initialize sse at zero.
  for m from 1 to number of bridges do
      Print m.                 ▷ To make sure the sse is for all of the bridges at the end.
      Make a temporary data frame that contains only rows from bridge m of
      the logistic_df.                             ▷ temp_bridge_df.
      Make a temporary data frame that contains only rows from bridge m of
      the segment_df.                              ▷ temp_segment_df.
      for j from 1 to number of rows of temp_segment_df do
          Find segment_length.                     ▷ How many years the segment lasts.
          Make temp_bridge_in_segment_df.   ▷ Subset of temp_bridge_df for the $j$-th segment.
          Initialize suff_history matrix.   ▷ An empty num_iter by segment_length matrix.
          for i from 1 to num_iter do
              Extract tempBridge.           ▷ The bridge at the beginning of the segment.
              Store the SUFFICIENCY_RATING of the first year in the segment in the
              first column of suff_history matrix.
              for k from 2 to segment_length do
                  Find $\hat{\lambda}$ by passing the tempBridge and coefficients of
                  Poisson model into get_lambda function.
                  Randomly generate Poisson variable using $\hat{\lambda}$.   ▷ Deterioration amount.
                  Decrease the SUFFICIENCY_RATING by the deterioration amount.
                  Increase the AGE of the bridge by one year.
                  Put the numbers in their place in the suff_history matrix.
              end for
          end for
          Pass the tempBridge, model_Poisson's coefficients, num_iter and segment_length
          into Poisson_Pred_Suff function.         ▷ Find Pois_Pred_Suff.
          Y = SUFFICIENCY_RATING_BEG of the temp_bridge_in_segment_df.
          SSE=SSE $+ \sum(Y - Pois\_Pred\_Suff)^2$
      end for
  end for
  Print SSE.
```

We see that the SSE of the mixed *Logistic_Gamma* model is much smaller than *Poison* model which shows the mixed model predicts the deteriorations better than *Poisson* model.

5.2 Examples

In this section we want to look at some bridges' deterioration curves under both mixed *logistic_Gamma* model and the *Poisson* model and observe the differences.

Example 5.1. *We defined a bridge in Table 5.1, and we set the number of iterations to be 400 and we want to observe how this bridge deteriorates 50 years from now. We set the segment_length equal to 50 and we passed the bridge, number of iterations and segment_length into both generate_curve function and Poisson_Pred_Suff function to see the difference of curves for our two different models. Looking at Figures* (5.1) *and* (5.2)*, we observe that the medians for both models are almost the same but the first quartiles line and the third quartiles line of the* Poisson *model are too close to the median line. We think it is not realistic and they should not be that close.*

Table 5.1: Bridge Data

Variables	***Value***	***Variables***	***Value***
Intercept	*1*	*STRUCTURE_LEN*	*85.6*
ROUTE_PREFIX	*4*	*SERVICE_LEVEL*	*1*
TRAFFIC_LANES_ON	*2*	*MEDIAN_CODE*	*0*
TRAFFIC_LANES_UND	*8*	*AGE*	*30*

Table 5.1: Bridge Data

Variables	***Value***	***Variables***	***Value***
STRUCTURE_KIND	*3*	*INSPECTION_GAP*	*1*
STRUCTURE_TYPE	*2*	*SUFFICIENCY_RATING_BEG*	*78.9*
NAV_VERT	*24*		

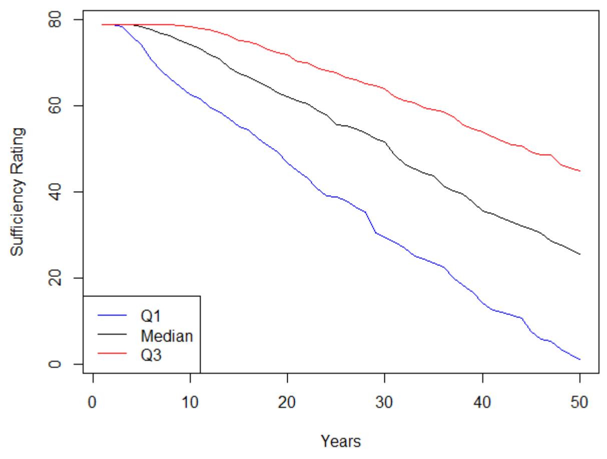

Figure 5.1: Logistic_Gamma Predictions for Example 5.1

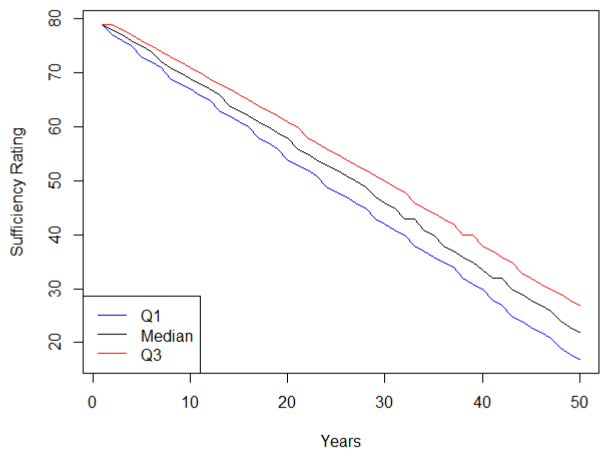

Figure 5.2: Poisson Predictions for Example 5.1

Example 5.2. *In this example we chose the bridge with STRUCTURE_NUMBER "000000003500130" out of the data frame and we set the number of iterations to be 100 and we want to observe how this bridge deteriorates in 60 years from now. We compared the curves of the two different models for that bridge as we see in Figures* (5.3) *and* (5.4)*, and we see that the Poisson deterioration is too slow and also the quartiles are too close to the median again. The reason is that when we have a Poisson parameter with small* λ *close to zero, the change is almost always zero or one. Therefore there can not be much variation. So* Poisson *model does not really allow for variation as much as the mixed* Logistic_Gamma *model does.*

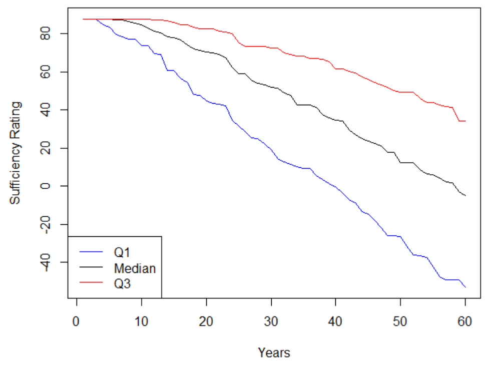

Figure 5.3: Mixed Model Predictions for Example 5.2

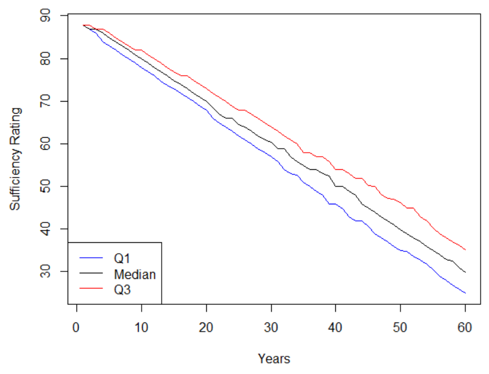

Figure 5.4: Poisson Model Predictions for Example 5.2

CHAPTER 6

CONCLUSION

Data about all bridges in the State of Georgia that is used in this thesis was collected for a 25 year period from 1992 to 2016. We picked 13 variables among 134 variables from the original data frame that seem to be potentially more significant. In this thesis, two models (Mixed *Logistics_Gamma* model and *Poisson* model) are introduced. They both predict the change of sufficiency rating of each bridge and tell us how much a specific bridge deteriorates in a specific period of time. The mixed *Logistics_Gamma* model is made of two different models; *finalmodel_logistic* and *finalmodel_gamma*. The *finalmodel_logistic* tells us if there is any drop in SUFFICIENCY_RATING. If DROP is greater than zero, then the *finalmodel_gamma* gives the amount of that deterioration. In Poisson model, since DROP is a Poisson variable, it must be a discrete random variable but as we see in the data frame, DROP is a continuous random variable. Therefore, we decided to round DROPs in our training set to the closest integers. We then compare these two models by checking their SSE. As it is explained in Chapter 5, the SSE of the mixed *Logistics_Gamma* model is 9,570,996 and the SSE of the *Poisson* model is 36,092,381 which is much larger than the SSE of mixed *Logistics_Gamma* model. Also we compared the deterioration curves of these two models in Chapter 5, and it is clear that the medians for both models are almost the same but the first quartile curves and the third quartile curves of the *Poisson* model are too close to the median curve which is not realistic and they should not be that close. The reason is that the λ of the Poisson parameter is too small and close to zero and it forces the change to be almost always zero or one. Thus the mixed *Logistics_Gamma* model is a better model to predict sufficiency rating of a bridge in future. This is not only because of the SSE and the curves, but also because the sufficiency rating and deterioration are continuous and the Gamma portion works in continuous case.

REFERENCES

[1] Governor Nathan Deal, Office of the Governor, *https://gov.georgia.gov/press-releases/2017-05-01/deal-signs-fy-2018-budget*

[2] Georgia Department of Transportation, *GDOT 2017 Reference Guide, http://www.dot.ga.gov/PartnerSmart/Public/Documents/publications/ReferenceGuide*

[3] Georgia Department of Transportation, *GDOT 2016 Reference Guide, http://www.dot.ga.gov/PartnerSmart/Public/Documents/publications/ReferenceGuide*

[4] Andria.Brooks, *Bridge maintenance starts following 100 million dollar infusion*, Atlanta Journal Constitution, July 07, 2015.

[5] A. K. Agrawal, A. Kawaguchi, and Z. Chen, *Deterioration rates of typical bridge elements in New York*, Journal of Bridge Engineering, Volume 15, Issue 4, July 2010.

[6] American Association of State Highway And Transportation Officials (AASHTO), *AASHTO Manual For Bridge Element Inspection*, Published by American Association of State Highway And Transportation Officials, ISBN.978-1-56051-622-4, First Edition 2013.

[7] U. S. Department of Transportation, Federal Highway Administration, *Recording and coding guide for the structure inventory and appraisal of the nation's bridges*, (Report No. FHWA-PD-96-001), 1995.

[8] Website of the U. S. Department of Transportation, Federal Highway Administration, Bridges and Structures, *https://www.fhwa.dot.gov/bridge/nbi.cfm*.

[9] Ayala Cohen, *Dummy Variables In Stepwise Regression*, Published by Taylor & Francis, Ltd. on behalf of the American Statistical Association, Volume 45, No.3 , August 1991 , pp. 226-228.

[10] P. McCullagh, J. A. Nelder, *Generalized Linear Models*, Chapman & Hall, London, Second Edition, 1989.

[11] M. Müller, *Generalized Linear Models*, Fraunhofer Institute for Industrial Mathematics (ITWM), P.O. Box 3049, D67663 Kaiserslautern (Germany), January 6, 2004.

[12] T. Naderimoghaddam and S. Carden, *A Comparison of Bridge Deterioration Models*, paper in preparation.

Appendix A

DATA CLEANING

A.1 Import Script

```
setwd("G:/Thesis/Thesis␣data␣Toki")

library(dplyr)

Year1 = "ga92del.txt"
Year2 = "ga93del.txt"
Year3 = "ga94del.txt"
Year4 = "ga95del.txt"
Year5 = "ga96del.txt"
Year6 = "ga97del.txt"
Year7 = "ga98del.txt"
Year8 = "ga99del.txt"
Year9 = "ga2000del.txt"
Year10 = "ga2001del.txt"
Year11 = "ga2002del.txt"
Year12 = "ga2003del.txt"
Year13 = "ga2004del.txt"
Year14 = "ga2005del.txt"
Year15 = "ga2006del.txt"
Year16 = "ga2007del.txt"
Year17 = "ga2008del.txt"
Year18 = "ga2009del.txt"
Year19 = "ga2010del.txt"
Year20 = "ga2011del.txt"
Year21 = "ga2012del.txt"
Year22 = "ga2013del.txt"
Year23 = "ga2014del.txt"
Year24 = "ga2015del.txt"
Year25 = "ga2016del.txt"

df1 = read.csv(Year1)
df2 = read.csv(Year2)
df3 = read.csv(Year3)
df4 = read.csv(Year4)
df5 = read.csv(Year5)
df6 = read.csv(Year6)
```

```
df7 = read.csv(Year7)
df8 = read.csv(Year8)
df9 = read.csv(Year9)
df10 = read.csv(Year10)
df11 = read.csv(Year11)
df12 = read.csv(Year12)
df13 = read.csv(Year13)
df14 = read.csv(Year14)
df15 = read.csv(Year15)
df16 = read.csv(Year16)
df17 = read.csv(Year17)
df18 = read.csv(Year18)
df19 = read.csv(Year19)
df20 = read.csv(Year20)
df21 = read.csv(Year21)
df22 = read.csv(Year22)
df23 = read.csv(Year23)
df24 = read.csv(Year24)
df25 = read.csv(Year25)

df1 = df1[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING', 'ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df1$YEAR = rep(1992,nrow(df1))

df2 = df2[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df2$YEAR = rep(1993,nrow(df2))

df3 = df3[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df3$YEAR = rep(1994,nrow(df3))

df4 = df4[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
```

```
df4$YEAR = rep(1995,nrow(df4))

df5 = df5[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df5$YEAR = rep(1996,nrow(df5))

df6 = df6[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df6$YEAR = rep(1997,nrow(df6))

df7 = df7[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df7$YEAR = rep(1998,nrow(df7))

df8 = df8[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df8$YEAR = rep(1999,nrow(df8))

df9 = df9[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df9$YEAR = rep(2000,nrow(df9))

df10 = df10[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df10$YEAR = rep(2001,nrow(df10))

df11 = df11[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df11$YEAR = rep(2002,nrow(df11))

df12 = df12[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df12$YEAR = rep(2003,nrow(df12))

df13 = df13[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df13$YEAR = rep(2004,nrow(df13))

df14 = df14[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df14$YEAR = rep(2005,nrow(df14))

df15 = df15[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df15$YEAR = rep(2006,nrow(df15))

df16 = df16[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df16$YEAR = rep(2007,nrow(df16))

df17 = df17[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df17$YEAR = rep(2008,nrow(df17))
```

```
df18 = df18[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df18$YEAR = rep(2009,nrow(df18))

df19 = df19[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df19$YEAR = rep(2010,nrow(df19))

df20 = df20[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df20$YEAR = rep(2011,nrow(df20))

df21 = df21[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df21$YEAR = rep(2012,nrow(df21))

df22 = df22[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df22$YEAR = rep(2013,nrow(df22))

df23 = df23[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df23$YEAR = rep(2014,nrow(df23))

df24 = df24[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df24$YEAR = rep(2015,nrow(df24))

df25 = df25[c('STRUCTURE_NUMBER_008','SUFFICIENCY_RATING','ROUTE_
   PREFIX_005B','SERVICE_LEVEL_005C', 'YEAR_BUILT_027', 'TRAFFIC_
   LANES_ON_028A','TRAFFIC_LANES_UND_028B','MEDIAN_CODE_033','NAV
   _VERT_CLR_MT_039','STRUCTURE_KIND_043A','STRUCTURE_TYPE_043B',
   'MAX_SPAN_LEN_MT_048','STRUCTURE_LEN_MT_049')]
df25$YEAR = rep(2016,nrow(df25))


df = bind_rows(df1,df2,df3,df4,df5,df6,df7,df8,df9,df10,df11,df12,
   df13,df14,df15,df16,df17,df18,df19,df20,df21,df22,df23,df24,
   df25)
df = na.omit(df)
}

read_bridge_data=function(df){
  STRUCTURE_NUMBER_008=vector(,1)
  ROUTE_PREFIX_005B=vector(,1)
  SERVICE_LEVEL_005C =vector(,1)
  YEAR_BUILT_027 =vector(,1)
  TRAFFIC_LANES_ON_028A =vector(,1)
  TRAFFIC_LANES_UND_028B =vector(,1)
  MEDIAN_CODE_033 =vector(,1)
  #DEGREES_SKEW_034 =vector(,1)
  NAV_VERT_CLR_MT_039 =vector(,1)
  STRUCTURE_KIND_043A =vector(,1)
  STRUCTURE_TYPE_043B =vector(,1)
  MAX_SPAN_LEN_MT_048 =vector(,1)
  STRUCTURE_LEN_MT_049 =vector(,1)
  Old_Date =vector(,1)
  New_Date =vector(,1)
  Age = vector(,1)
  Inspection_Gap =vector(,1)
  SUFFICIENCY_RATING_Beg =vector(,1)
  Drop=vector(,1)
  SegmentList = list()
  n = 1
  bridgeIDs = unique(df$STRUCTURE_NUMBER_008)
  print(bridgeIDs)
  for (i in 1:length(bridgeIDs)) {
    print(i)
    tempdf <- subset(df, STRUCTURE_NUMBER_008 == bridgeIDs[i])
    tempBridgeList = list()
```

```
j = 1
while (is.na(tempdf$SUFFICIENCY_RATING[j]) &
       j < nrow(tempdf) + 1) {
  j = j + 1
}
while (j < nrow(tempdf) + 1) {
  old_condition = tempdf$SUFFICIENCY_RATING[j]
  old_date = tempdf$YEAR[j]
  j = j + 1
  flag = FALSE
  while (!flag) {
    while (is.na(tempdf$SUFFICIENCY_RATING[j]) & j < nrow(tempdf
        ) + 1) {
      j = j + 1
    }
    if (j > nrow(tempdf)) {
      flag = TRUE
    } else{
      cur_condition = tempdf$SUFFICIENCY_RATING[j]
      cur_date = tempdf$YEAR[j]
      Dropping = old_condition - cur_condition
      if (Dropping >= 0) {
        STRUCTURE_NUMBER_008[n] = bridgeIDs[i]
        ROUTE_PREFIX_005B[n] = tempdf$ROUTE_PREFIX_005B[j]
        SERVICE_LEVEL_005C[n] = tempdf$SERVICE_LEVEL_005C[j]
        YEAR_BUILT_027 [n] = tempdf$YEAR_BUILT_027[j]
        TRAFFIC_LANES_ON_028A[n] = tempdf$TRAFFIC_LANES_ON_028A[j
            ]
        TRAFFIC_LANES_UND_028B[n] = tempdf$TRAFFIC_LANES_UND_028B
            [j]
        MEDIAN_CODE_033[n] = tempdf$MEDIAN_CODE_033[j]
        NAV_VERT_CLR_MT_039[n] = tempdf$NAV_VERT_CLR_MT_039[j]
        STRUCTURE_KIND_043A[n] = tempdf$STRUCTURE_KIND_043A[j]
        STRUCTURE_TYPE_043B[n] = tempdf$STRUCTURE_TYPE_043B[j]
        MAX_SPAN_LEN_MT_048[n] = tempdf$MAX_SPAN_LEN_MT_048[j]
        STRUCTURE_LEN_MT_049[n] = tempdf$STRUCTURE_LEN_MT_049[j]
        Old_Date[n] = old_date
        New_Date[n] = cur_date
        Age[n] = New_Date - tempdf$YEAR_BUILT_027[j] + 1
        Inspection_Gap[n] = New_Date[n] - Old_Date[n]
        SUFFICIENCY_RATING_Beg[n] = old_condition
        Drop[n] = Dropping
        n = n + 1
      }
      flag = TRUE
    }
```

```
      }
    }
  }
  newdf = data.frame(STRUCTURE_NUMBER_008, ROUTE_PREFIX_005B,
     SERVICE_LEVEL_005C, YEAR_BUILT_027 ,TRAFFIC_LANES_ON_028A,
     TRAFFIC_LANES_UND_028B, MEDIAN_CODE_033,
  NAV_VERT_CLR_MT_039, STRUCTURE_KIND_043A, STRUCTURE_TYPE_043B,
     MAX_SPAN_LEN_MT_048 , STRUCTURE_LEN_MT_049, Old_Date, New_
     Date, Age, Inspection_Gap, SUFFICIENCY_RATING_Beg,
  Drop)
  newdf$MEDIAN_CODE_033=as.factor(newdf$MEDIAN_CODE_033)
  newdf$ROUTE_PREFIX_005B=as.factor(newdf$ROUTE_PREFIX_005B)
  newdf$SERVICE_LEVEL_005C=as.factor(newdf$SERVICE_LEVEL_005C)
  newdf$STRUCTURE_KIND_043A=as.factor(newdf$STRUCTURE_KIND_043A)
  newdf$STRUCTURE_TYPE_043B=as.factor(newdf$STRUCTURE_TYPE_043B)
  return(newdf)
}
```

A.2 SEGMENTS

```
make_segment_df=function(df){
  STRUCTURE_NUMBER=vector(,1)
  segmentBegins = vector(,1)
  segmentEnds = vector(,1)
  n=1
  bridgeIDs = unique(as.character(df$STRUCTURE_NUMBER_008))
  for (i in 1:length(bridgeIDs)) {
    print(i)
    tempdf <- subset(df,STRUCTURE_NUMBER_008 == bridgeIDs[i])
    j=1
    bridgeFlag = FALSE
    while (!bridgeFlag){
      segmentBegin = tempdf$Old_Date[j]
      segmentFlag = FALSE
      while(!segmentFlag){
        segmentEnd = tempdf$New_Date[j]
        if ((j+1)>length(tempdf$Old_Date)){
          segmentFlag = TRUE
          bridgeFlag = TRUE
        }else{
          if (tempdf$Old_Date[j+1] != segmentEnd){
            segmentFlag = TRUE
          }
        }
        j = j + 1
      }
      STRUCTURE_NUMBER[n] = bridgeIDs[i]
      segmentBegins[n] = segmentBegin
      segmentEnds[n] = segmentEnd
      n = n + 1
      print(n)
    }
  }
  newdf = data.frame(STRUCTURE_NUMBER, segmentBegins, segmentEnds)
  return(newdf)
}
```

A.3 Construct a data frame with DROP ≥ 0

```
full_df = read_bridge_data(df)
full_df = full_df[-which(full_df$Inspection_Gap==0),]
reduced_df = subset(full_df, STRUCTURE_TYPE_043B %in% c
    (1,2,4,5,6,11,19))
reduced_df$ROUTE_PREFIX_005B[reduced_df$ROUTE_PREFIX_005B %in% c("
   6","7","8")] <- "8"
reduced_df$STRUCTURE_KIND_043A <- as.character(reduced_df$
   STRUCTURE_KIND_043A)
reduced_df$STRUCTURE_KIND_043A[reduced_df$STRUCTURE_KIND_043A %in%
    c("1","2")] <- "1"
reduced_df$STRUCTURE_KIND_043A[reduced_df$STRUCTURE_KIND_043A %in%
    c("3","4")] <- "3"
reduced_df$STRUCTURE_KIND_043A[reduced_df$STRUCTURE_KIND_043A %in%
    c("5","6")] <- "5"
reduced_df$STRUCTURE_KIND_043A[reduced_df$STRUCTURE_KIND_043A %in%
    c("8","9")] <- "8"
reduced_df$STRUCTURE_KIND_043A <- factor(reduced_df$STRUCTURE_KIND
   _043A)
segment_df = make_segment_df(reduced_df)
```

A.4 Classify ”Drop” as either 1 or 0

```
n = length(reduced_df$Drop)
DetPres = rep(0,length=n)
for (i in 1:n){
  if (reduced_df$Drop[i]>0){
    DetPres[i]=1
  } else{
    DetPres[i]=0
  }
}
reduced_df$DetPres = DetPres
```

A.5 LOGISTIC AND GAMMA DATA FRAME SETUP

```
logistic_df = reduced_df
posdrops_df = subset(reduced_df, Drop>0)
gamma_df = posdrops_df
```

A.6 CLEANING HELPER DATA FRAMES

```
rm(df, df1, df2, df3, df4, df5, df6, df7, df8, df9, df10, df11,
    df12, df13, df14, df15, df16, df17, df18, df19, df20,
   df21, df22, df23, df24, df25, full_df, posdrops_df, reduced_df,
       Year1, Year2, Year3, Year4, Year5, Year6, Year7, Year8,
   Year9, Year10, Year11, Year12, Year13, Year14, Year15, Year16,
      Year17, Year18, Year19, Year20, Year21, Year22, Year23,
   Year24, Year25, i, n, make_segment_df, read_bridge_data,
      DetPres)
```

A.7 FREQUENCY TABLE FOR CATEGORICAL VARIABLES

```
new_df = subset(logistic_df, STRUCTURE_NUMBER_008 == bridgeIDs[1])
new_df = new_df[1,]
for (i in 1:length(bridgeIDs)){
  print(i)
  temp_df = subset(logistic_df, STRUCTURE_NUMBER_008 == bridgeIDs[
     i])
  new_df[i,] = temp_df[1,]
}
w= table(new_df$ROUTE_PREFIX_005B)
v=table(new_df$STRUCTURE_KIND_043A)
h=table(new_df$STRUCTURE_TYPE_043B)
l=table(new_df$SERVICE_LEVEL_005C)
c=table(new_df$MEDIAN_CODE_033)
```

Appendix B

REGRESSION MODELS

B.1 Training And Test Sets

```
sample_size = floor(.75*nrow(logistic_df))
set.seed(30461)
training_set_indices = sample(seq_len(nrow(logistic_df)),size =
   sample_size)
training_set_logistic = logistic_df[training_set_indices,]
test_set_logistic = logistic_df[-training_set_indices,]
training_set_logistic2 = training_set_logistic
training_set_logistic2$STRUCTURE_NUMBER_008 = NULL
training_set_logistic2$Old_Date = NULL
training_set_logistic2$New_Date = NULL
training_set_logistic2$YEAR_BUILT_027 = NULL
```

B.2 Logistic Model

B.2.1 Full Model and Backward Elimination

```
training_set_logistic2$Drop = NULL
fullmodel_logistic = glm (DetPres ~ factor(ROUTE_PREFIX_005B)
                           + factor(SERVICE_LEVEL_005C)
                           + TRAFFIC_LANES_ON_028A
                           + TRAFFIC_LANES_UND_028B
                           + factor(MEDIAN_CODE_033)
                           + NAV_VERT_CLR_MT_039
                           + factor(STRUCTURE_KIND_043A)
                           + factor(STRUCTURE_TYPE_043B)
                           + MAX_SPAN_LEN_MT_048
                           + STRUCTURE_LEN_MT_049
                           + Age + Inspection_Gap
                           + SUFFICIENCY_RATING_Beg,
                           data=training_set_logistic2,
                           family = binomial,
                           y = FALSE, model = FALSE)
null=glm(DetPres ~ 1, data = training_set_logistic2,
       family = binomial)
backwards1 = step(fullmodel_logistic)
stepmodel_logistic = glm (DetPres ~ factor(ROUTE_PREFIX_005B)
                           + TRAFFIC_LANES_UND_028B
                           + factor(STRUCTURE_TYPE_043B)
                           + factor(STRUCTURE_KIND_043A)
                           + STRUCTURE_LEN_MT_049
                           + MAX_SPAN_LEN_MT_048
                           + factor(SERVICE_LEVEL_005C)
                           + factor(MEDIAN_CODE_033)
                           + Age + Inspection_Gap
                           + SUFFICIENCY_RATING_Beg,
                           data=training_set_logistic2,
                           family = binomial,
                           y = FALSE, model = FALSE)
anova(stepmodel_logistic,fullmodel_logistic, test="Chisq")
```

B.2.2 PROPORTION OF DETPRES AMONG LEVELS OF STRUCTURE_KIND

```
A=training_set_logistic2
levels(A$STRUCTURE_KIND_043A)
for (i in levels(A$STRUCTURE_KIND_043A)){
 print(as.numeric(i))
 print(mean(A$DetPres[A$STRUCTURE_KIND_043A == as.numeric(i)]))
}
table(A$STRUCTURE_KIND_043A)
```

B.2.3 PROPORTION OF DETPRES AMONG LEVELS OF STRUCTURE_TYPE

```
levels(A$STRUCTURE_TYPE_043B)
for (i in levels(A$STRUCTURE_TYPE_043B)){
 print(as.numeric(i))
 print(mean(A$DetPres[A$STRUCTURE_TYPE_043B == as.numeric(i)]))
}
table(A$STRUCTURE_TYPE_043B)
```

B.2.4 DROP STRUCTURE_KIND, KEEP STRUCTURE_TYPE

```
Redmodel_logistic = glm (DetPres ~ factor(ROUTE_PREFIX_005B)
                          + TRAFFIC_LANES_UND_028B
                          + factor(STRUCTURE_TYPE_043B)
                          + STRUCTURE_LEN_MT_049
                          + MAX_SPAN_LEN_MT_048
```

```
                              + factor(SERVICE_LEVEL_005C)
                              + factor(MEDIAN_CODE_033)
                              + Age + Inspection_Gap
                              + SUFFICIENCY_RATING_Beg,
                              data=training_set_logistic2,
                              family = binomial,
                              y = FALSE, model = FALSE)
anova(Redmodel_logistic, stepmodel_logistic , test="Chisq")
```

B.2.5 Correlation Matrix

```
A=training_set_logistic2
y=A[,14]
x1=A[,1]
x2=A[,2]
x3=A[,3]
x4=A[,4]
x5=A[,5]
x6=A[,6]
x7=A[,7]
x8=A[,8]
x9=A[,9]
x10=A[,10]
x11=A[,11]
x12=A[,12]
x13=A[,13]
one = rep(1, length(y))
X= cbind(one, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10,
       x11, x12 , x13 )
cor(X, use="all.obs", method="spearman" )
```

B.2.6 Drop MAX_SPAN

```
model_logistic = glm (DetPres ~ factor(ROUTE_PREFIX_005B)
                            + TRAFFIC_LANES_ON_028A
                            + TRAFFIC_LANES_UND_028B
                            + NAV_VERT_CLR_MT_039
                            + factor(STRUCTURE_TYPE_043B)
                            + factor(STRUCTURE_KIND_043A)
                            + STRUCTURE_LEN_MT_049
                            + factor(SERVICE_LEVEL_005C)
                            + factor(MEDIAN_CODE_033)
                            + Age + Inspection_Gap
                            + SUFFICIENCY_RATING_Beg,
                            data=training_set_logistic2,
                            family = binomial, y = FALSE,
                            model = FALSE)
    anova(model_logistic , fullmodel_logistic , test="Chisq")
```

B.2.7 Backward Elimination for *model_logistic*

```
null=glm(DetPres ~ 1, data = training_set_logistic2,
       family = binomial)
backwards2 = step(model_logistic)
smodel_logistic = glm(DetPres ~ factor(ROUTE_PREFIX_005B)
                        + TRAFFIC_LANES_ON_028A
                        + TRAFFIC_LANES_UND_028B
                        + factor(STRUCTURE_TYPE_043B)
                        + factor(STRUCTURE_KIND_043A)
                        + STRUCTURE_LEN_MT_049
                        + factor(SERVICE_LEVEL_005C)
                        + factor(MEDIAN_CODE_033)
                        + Age + Inspection_Gap
                        + SUFFICIENCY_RATING_Beg ,
                        data=training_set_logistic2,
                        family = binomial,
                        y = FALSE, model = FALSE)
anova(smodel_logistic, model_logistic, test="Chisq")
```

B.2.8 Drop STRUCTURE_KIND From *smodel_logistic*

```
Rmodel_logistic = glm(DetPres ~ factor(ROUTE_PREFIX_005B)
                        + TRAFFIC_LANES_ON_028A
                        + TRAFFIC_LANES_UND_028B
                        + factor(STRUCTURE_TYPE_043B)
                        + STRUCTURE_LEN_MT_049
                        + factor(SERVICE_LEVEL_005C)
                        + factor(MEDIAN_CODE_033)
                        + Age + Inspection_Gap
                        + SUFFICIENCY_RATING_Beg ,
                        data=training_set_logistic2,
                        family = binomial,
                        y = FALSE, model = FALSE)
anova(Rmodel_logistic, smodel_logistic , test="Chisq")
```

B.2.9 Final Logistic Model

```
fullmodel_predictions = predict(fullmodel_logistic,
                        newdata = test_set_logistic,
                        type = "response")
Redmodel_predictions = predict(Redmodel_logistic,
                       newdata = test_set_logistic,
                       type = "response")
stepmodel_predictions = predict(stepmodel_logistic,
                        newdata = test_set_logistic,
                        type = "response")
smodel_predictions = predict(smodel_logistic,
                     newdata = test_set_logistic,
                     type = "response")
sum((fullmodel_predictions - test_set_logistic$DetPres)^2)
sum((Redmodel_predictions - test_set_logistic$DetPres)^2)
sum((stepmodel_predictions - test_set_logistic$DetPres)^2)
sum((smodel_predictions - test_set_logistic$DetPres)^2)
finalmodel_logistic = smodel_logistic
```

B.3 Gamma Model

B.3.1 Full Gamma Model

```
training_set_gamma = subset(training_set_logistic2, Drop>0)
test_set_gamma = subset(test_set_logistic, Drop>0)
training_set_gamma$DetPres = NULL
fullmodel_gamma = glm(Drop ~ factor(ROUTE_PREFIX_005B)
                     + TRAFFIC_LANES_ON_028A
                     + TRAFFIC_LANES_UND_028B
                     + NAV_VERT_CLR_MT_039
                     + factor(STRUCTURE_KIND_043A)
                     + factor(STRUCTURE_TYPE_043B)
                     + STRUCTURE_LEN_MT_049
                     + factor(SERVICE_LEVEL_005C)
                     + factor(MEDIAN_CODE_033)
                     + MAX_SPAN_LEN_MT_048
                     + Age + Inspection_Gap
                     + SUFFICIENCY_RATING_Beg,
                     data=training_set_gamma,
                     family = Gamma(link="log"),
                     y = FALSE, model = FALSE)
```

B.3.2 Correlation Matrix

```
B=training_set_gamma
U=B[,14]
v1=B[,1]
v2=B[,2]
v3=B[,3]
v4=B[,4]
v5=B[,5]
v6=B[,6]
v7=B[,7]
v8=B[,8]
v9=B[,9]
v10=B[,10]
v11=B[,11]
v12=B[,12]
v13=B[,13]
one = rep(1, length(U))
V = cbind(one, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10,
       v11, v12 , v13 )
cor(V, use="all.obs", method="spearman" )
```

B.3.3 DROP MAX_SPAN

```
model_gamma = glm(Drop ~ factor(ROUTE_PREFIX_005B)
                   + TRAFFIC_LANES_ON_028A
                   + TRAFFIC_LANES_UND_028B
                   + NAV_VERT_CLR_MT_039
                   + factor(STRUCTURE_KIND_043A)
                   + factor(STRUCTURE_TYPE_043B)
                   + STRUCTURE_LEN_MT_049
                   + factor(SERVICE_LEVEL_005C)
                   + factor(MEDIAN_CODE_033)
                   + Age + Inspection_Gap
                   + SUFFICIENCY_RATING_Beg,
                   data=training_set_gamma,
                   family = Gamma(link="log"),
                   y = FALSE, model = FALSE)
```

B.3.4 BACKWARD ELIMINATION

```
null = glm(Drop~1, data = training_set_gamma,
        family = Gamma(link="log"))
backwards3 = step(model_gamma )
Stepmodel_gamma = glm(Drop ~ factor(ROUTE_PREFIX_005B)
                      + TRAFFIC_LANES_ON_028A
```

```
                          + TRAFFIC_LANES_UND_028B
                          + NAV_VERT_CLR_MT_039
                          + factor(STRUCTURE_KIND_043A)
                          + factor(STRUCTURE_TYPE_043B)
                          + STRUCTURE_LEN_MT_049
                          + factor(SERVICE_LEVEL_005C)
                          + Age + Inspection_Gap
                          + SUFFICIENCY_RATING_Beg,
                          data=training_set_gamma,
                          family = Gamma(link="log"),
                          y = FALSE, model = FALSE)
anova(Stepmodel_gamma,model_gamma, test = "F")
```

B.3.5 Drop NAV_VERT

```
Redmodel_gamma = glm(Drop ~ factor(ROUTE_PREFIX_005B)
                         + TRAFFIC_LANES_ON_028A
                         + TRAFFIC_LANES_UND_028B
                         + factor(STRUCTURE_KIND_043A)
                         + factor(STRUCTURE_TYPE_043B)
                         + STRUCTURE_LEN_MT_049
                         + factor(SERVICE_LEVEL_005C)
                         + Age + Inspection_Gap
                         + SUFFICIENCY_RATING_Beg,
                         data=training_set_gamma,
                         family = Gamma(link="log"),
                         y = FALSE, model = FALSE)
anova(Redmodel_gamma,Stepmodel_gamma, test = "F")
```

B.3.6 Box Plots of Levels of STRUCTURE_KIND and STRUCTURE_TYPE

```
boxplot(Drop~STRUCTURE_KIND_043A, data=training_set_gamma)
boxplot(Drop~STRUCTURE_TYPE_043B, data=training_set_gamma)
```

B.3.7 Final Gamma Model

```
fullmodel_predictions2 = predict(fullmodel_gamma,
                              newdata = test_set_gamma,
                              type = "response")
Redmodel_predictions2 = predict(Redmodel_gamma,
                             newdata = test_set_gamma,
                             type = "response")
stepmodel_predictions2 = predict(Stepmodel_gamma,
                             newdata = test_set_gamma,
                             type = "response")
model_predictions2 = predict(model_gamma,
                           newdata = test_set_gamma,
                           type = "response")
sum((fullmodel_predictions2 - test_set_gamma$Drop)^2)
sum((Redmodel_predictions2 - test_set_gamma$Drop)^2)
sum((stepmodel_predictions2 - test_set_gamma$Drop)^2)
sum((model_predictions2 - test_set_gamma$Drop)^2)
finalmodel_gamma = Redmodel_gamma
```

B.4 Poisson Model

B.4.1 Full Poisson Model and Backward Elimination

```
training_set_logistic2$DetPres = NULL
training_set_logistic2$Drop = round(training_set_logistic2$Drop)
fullmodel_Poisson = glm(Drop ~ factor(ROUTE_PREFIX_005B)
                        + TRAFFIC_LANES_ON_028A
                        + TRAFFIC_LANES_UND_028B
                        + NAV_VERT_CLR_MT_039
                        + factor(STRUCTURE_TYPE_043B)
                        +  factor(STRUCTURE_KIND_043A)
                        + STRUCTURE_LEN_MT_049
                        + MAX_SPAN_LEN_MT_048
                        + factor(SERVICE_LEVEL_005C)
                        + factor(MEDIAN_CODE_033)
                        + Age
                        + Inspection_Gap
                        + SUFFICIENCY_RATING_Beg ,
                        data=training_set_logistic2,
                        family = poisson )
```

B.4.2 Correlation Matrix

```
C = training_set_logistic2
T =C[,14]
l1=C[,1]
l2=C[,2]
l3=C[,3]
l4=C[,4]
l5=C[,5]
l6=C[,6]
```

```
l7=C[,7]
l8=C[,8]
l9=C[,9]
l10=C[,10]
l11=C[,11]
l12=C[,12]
l13=C[,13]
one = rep(1, length(T))
L = cbind(one, l1, l2, l3, l4, l5, l6, l7, l8, l9, l10,
       l11, l12 , l13)
cor(L, use="all.obs", method="spearman")
```

B.4.3 DROP MAX_SPAN

```
Rmodel_Poisson = glm(Drop ~ factor(ROUTE_PREFIX_005B)
                        + TRAFFIC_LANES_ON_028A
                        + TRAFFIC_LANES_UND_028B
                        + NAV_VERT_CLR_MT_039
                        + factor(STRUCTURE_TYPE_043B)
                        + factor(STRUCTURE_KIND_043A)
                        + STRUCTURE_LEN_MT_049
                        + factor(SERVICE_LEVEL_005C)
                        + factor(MEDIAN_CODE_033)
                        + Age
                        + Inspection_Gap
                        + SUFFICIENCY_RATING_Beg ,
                        data=training_set_logistic2,
                        family = poisson )
```

B.4.4 BACKWARD METHOD ON RMODEL_POISSON

```
null = glm(Drop~1, data = training_set_logistic2, family = poisson
   )
backwards5 = step(Rmodel_Poisson)
finalmodel_Poisson = Rmodel_Poisson
```

B.5 FINAL MODELS

```
model_logistic = glm(DetPres ~ factor(ROUTE_PREFIX_005B)
                       + TRAFFIC_LANES_ON_028A
                       + TRAFFIC_LANES_UND_028B
                       + factor(STRUCTURE_TYPE_043B)
                       + factor(STRUCTURE_KIND_043A)
```

```
                        + STRUCTURE_LEN_MT_049
                        + factor(SERVICE_LEVEL_005C)
                        + factor(MEDIAN_CODE_033)
                        + Age + Inspection_Gap
                        + SUFFICIENCY_RATING_Beg ,
                        data=logistic_df,
                        family = binomial,
                        y = FALSE, model = FALSE)

model_gamma = glm(Drop ~ factor(ROUTE_PREFIX_005B)
                  + TRAFFIC_LANES_ON_028A
                  + TRAFFIC_LANES_UND_028B
                  + factor(STRUCTURE_KIND_043A)
                  + factor(STRUCTURE_TYPE_043B)
                  + STRUCTURE_LEN_MT_049
                  + factor(SERVICE_LEVEL_005C)
                  + Age + Inspection_Gap
                  + SUFFICIENCY_RATING_Beg,
                  data=gamma_df,
                  family = Gamma(link="log"),
                  y = FALSE, model = FALSE)

dispersion = sum((gamma_df$Drop - model_gamma$fitted.values)^2/
   model_gamma$fitted.values^2)/(nrow(gamma_df)-length(model_
   gamma$coeff))
alpha_hat = 1/dispersion


model_Poisson = glm(Drop ~ factor(ROUTE_PREFIX_005B)
                   + TRAFFIC_LANES_ON_028A
                   + TRAFFIC_LANES_UND_028B
                   + NAV_VERT_CLR_MT_039
                   + factor(STRUCTURE_TYPE_043B)
                   + factor(STRUCTURE_KIND_043A)
                   + STRUCTURE_LEN_MT_049
                   + factor(SERVICE_LEVEL_005C)
                   + factor(MEDIAN_CODE_033)
                   + Age + Inspection_Gap
                   + SUFFICIENCY_RATING_Beg ,
                   data=logistic_df,
                   family = poisson )
```

Appendix C

DETERIORATION CURVES

C.1 MIXED LOGISTIC_GAMMA MODEL'S CURVES

C.1.1 GET_ETA FUNCTION

```
get_eta = function(bridge, coeff_logistic, coeff_gamma){
  names(coeff_logistic)[names(coeff_logistic)=="factor(ROUTE_
     PREFIX_005B)2"] = "ROUTE_PREFIX_005B2"
  names(coeff_logistic)[names(coeff_logistic)=="factor(ROUTE_
     PREFIX_005B)3"] = "ROUTE_PREFIX_005B3"
  names(coeff_logistic)[names(coeff_logistic)=="factor(ROUTE_
     PREFIX_005B)4"] = "ROUTE_PREFIX_005B4"
  names(coeff_logistic)[names(coeff_logistic)=="factor(ROUTE_
     PREFIX_005B)5"] = "ROUTE_PREFIX_005B5"
  names(coeff_logistic)[names(coeff_logistic)=="factor(ROUTE_
     PREFIX_005B)8"] = "ROUTE_PREFIX_005B8"

  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     TYPE_043B)2"] = "STRUCTURE_TYPE_043B2"
  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     TYPE_043B)4"] = "STRUCTURE_TYPE_043B4"
  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     TYPE_043B)5"] = "STRUCTURE_TYPE_043B5"
  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     TYPE_043B)6"] = "STRUCTURE_TYPE_043B6"
  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     TYPE_043B)11"] = "STRUCTURE_TYPE_043B11"
  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     TYPE_043B)19"] = "STRUCTURE_TYPE_043B19"

  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     KIND_043A)1"] = "STRUCTURE_KIND_043A1"
  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     KIND_043A)3"] = "STRUCTURE_KIND_043A3"
  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     KIND_043A)5"] = "STRUCTURE_KIND_043A5"
  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     KIND_043A)7"] = "STRUCTURE_KIND_043A7"
  names(coeff_logistic)[names(coeff_logistic)=="factor(STRUCTURE_
     KIND_043A)8"] = "STRUCTURE_KIND_043A8"

  names(coeff_logistic)[names(coeff_logistic)=="factor(SERVICE_
     LEVEL_005C)1"] = "SERVICE_LEVEL_005C1"
  names(coeff_logistic)[names(coeff_logistic)=="factor(SERVICE_
     LEVEL_005C)2"] = "SERVICE_LEVEL_005C2"
  names(coeff_logistic)[names(coeff_logistic)=="factor(SERVICE_
     LEVEL_005C)3"] = "SERVICE_LEVEL_005C3"
  names(coeff_logistic)[names(coeff_logistic)=="factor(SERVICE_
     LEVEL_005C)4"] = "SERVICE_LEVEL_005C4"
  names(coeff_logistic)[names(coeff_logistic)=="factor(SERVICE_
     LEVEL_005C)6"] = "SERVICE_LEVEL_005C6"
  names(coeff_logistic)[names(coeff_logistic)=="factor(SERVICE_
     LEVEL_005C)7"] = "SERVICE_LEVEL_005C7"
  names(coeff_logistic)[names(coeff_logistic)=="factor(SERVICE_
     LEVEL_005C)8"] = "SERVICE_LEVEL_005C8"

  names(coeff_logistic)[names(coeff_logistic)=="factor(MEDIAN_CODE
     _033)1"] = "MEDIAN_CODE_0331"
  names(coeff_logistic)[names(coeff_logistic)=="factor(MEDIAN_CODE
     _033)2"] = "MEDIAN_CODE_0332"
  names(coeff_logistic)[names(coeff_logistic)=="factor(MEDIAN_CODE
     _033)3"] = "MEDIAN_CODE_0333"

  names(coeff_gamma)[names(coeff_gamma)=="factor(ROUTE_PREFIX_005B
     )2"] = "ROUTE_PREFIX_005B2"
  names(coeff_gamma)[names(coeff_gamma)=="factor(ROUTE_PREFIX_005B
     )3"] = "ROUTE_PREFIX_005B3"
  names(coeff_gamma)[names(coeff_gamma)=="factor(ROUTE_PREFIX_005B
     )4"] = "ROUTE_PREFIX_005B4"
  names(coeff_gamma)[names(coeff_gamma)=="factor(ROUTE_PREFIX_005B
     )5"] = "ROUTE_PREFIX_005B5"
  names(coeff_gamma)[names(coeff_gamma)=="factor(ROUTE_PREFIX_005B
     )8"] = "ROUTE_PREFIX_005B8"

  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_TYPE_
     043B)2"] = "STRUCTURE_TYPE_043B2"
  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_TYPE_
     043B)4"] = "STRUCTURE_TYPE_043B4"
  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_TYPE_
     043B)5"] = "STRUCTURE_TYPE_043B5"
  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_TYPE_
     043B)6"] = "STRUCTURE_TYPE_043B6"
  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_TYPE_
     043B)11"] = "STRUCTURE_TYPE_043B11"
  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_TYPE_
     043B)19"] = "STRUCTURE_TYPE_043B19"

  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_KIND_
     043A)1"] = "STRUCTURE_KIND_043A1"
  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_KIND_
     043A)3"] = "STRUCTURE_KIND_043A3"
  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_KIND_
     043A)5"] = "STRUCTURE_KIND_043A5"
  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_KIND_
     043A)7"] = "STRUCTURE_KIND_043A7"
  names(coeff_gamma)[names(coeff_gamma)=="factor(STRUCTURE_KIND_
     043A)8"] = "STRUCTURE_KIND_043A8"

  names(coeff_gamma)[names(coeff_gamma)=="factor(SERVICE_LEVEL_005
     C)1"] = "SERVICE_LEVEL_005C1"
  names(coeff_gamma)[names(coeff_gamma)=="factor(SERVICE_LEVEL_005
     C)2"] = "SERVICE_LEVEL_005C2"
  names(coeff_gamma)[names(coeff_gamma)=="factor(SERVICE_LEVEL_005
     C)3"] = "SERVICE_LEVEL_005C3"
  names(coeff_gamma)[names(coeff_gamma)=="factor(SERVICE_LEVEL_005
     C)4"] = "SERVICE_LEVEL_005C4"
  names(coeff_gamma)[names(coeff_gamma)=="factor(SERVICE_LEVEL_005
     C)6"] = "SERVICE_LEVEL_005C6"
  names(coeff_gamma)[names(coeff_gamma)=="factor(SERVICE_LEVEL_005
     C)7"] = "SERVICE_LEVEL_005C7"
  names(coeff_gamma)[names(coeff_gamma)=="factor(SERVICE_LEVEL_005
     C)8"] = "SERVICE_LEVEL_005C8"

variables_logistic = c("(Intercept)", "ROUTE_PREFIX_005B",
                 "TRAFFIC_LANES_ON_028A",
                 "TRAFFIC_LANES_UND_028B",
                 "STRUCTURE_KIND_043A",
                 "STRUCTURE_TYPE_043B",
                 "STRUCTURE_LEN_MT_049",
                 "SERVICE_LEVEL_005C",
                 "MEDIAN_CODE_033",
                 "Age", "Inspection_Gap",
                 "SUFFICIENCY_RATING_Beg")
variables_gamma = c("(Intercept)", "ROUTE_PREFIX_005B",
               "TRAFFIC_LANES_ON_028A",
               "TRAFFIC_LANES_UND_028B",
               "STRUCTURE_KIND_043A",
               "STRUCTURE_TYPE_043B",
               "STRUCTURE_LEN_MT_049",
               "SERVICE_LEVEL_005C",
               "Age", "Inspection_Gap",
               "SUFFICIENCY_RATING_Beg")
is_categorical = vector()
is_categorical["(Intercept)"] = 0
is_categorical["ROUTE_PREFIX_005B"] = 1
is_categorical["TRAFFIC_LANES_ON_028A"] = 0
is_categorical["TRAFFIC_LANES_UND_028B"] = 0
is_categorical["STRUCTURE_KIND_043A"] = 1
is_categorical["STRUCTURE_TYPE_043B"] = 1
is_categorical["STRUCTURE_LEN_MT_049"] = 0
is_categorical["SERVICE_LEVEL_005C"] = 1
is_categorical["MEDIAN_CODE_033"] = 1
is_categorical["Age"] = 0
is_categorical["Inspection_Gap"] = 0
is_categorical["SUFFICIENCY_RATING_Beg"] = 0

bridge["(Intercept)"] = 1
beta_logistic = vector(, length = 0)
beta_logistic["(Intercept)"] = coeff_logistic["(Intercept)"]
if (bridge["ROUTE_PREFIX_005B"]==1){
 beta_logistic["ROUTE_PREFIX_005B"] = 0
} else if(bridge["ROUTE_PREFIX_005B"]==2){
 beta_logistic["ROUTE_PREFIX_005B"] = coeff_logistic["ROUTE_
    PREFIX_005B2"]
} else if(bridge["ROUTE_PREFIX_005B"]==3){
 beta_logistic["ROUTE_PREFIX_005B"] = coeff_logistic["ROUTE_
    PREFIX_005B3"]
} else if(bridge["ROUTE_PREFIX_005B"]==4){
 beta_logistic["ROUTE_PREFIX_005B"] = coeff_logistic["ROUTE_
    PREFIX_005B4"]
} else if(bridge["ROUTE_PREFIX_005B"]==5){
 beta_logistic["ROUTE_PREFIX_005B"] = coeff_logistic["ROUTE_
    PREFIX_005B5"]
} else if(bridge["ROUTE_PREFIX_005B"]==8){
 beta_logistic["ROUTE_PREFIX_005B"] = coeff_logistic["ROUTE_
    PREFIX_005B8"]
}
beta_logistic["TRAFFIC_LANES_ON_028A"] = coeff_logistic["TRAFFIC_
   LANES_ON_028A"]
beta_logistic["TRAFFIC_LANES_UND_028B"] = coeff_logistic["TRAFFIC_
   LANES_UND_028B"]

if (bridge["STRUCTURE_KIND_043A"]==0){
 beta_logistic["STRUCTURE_KIND_043A"] = 0
} else if(bridge["STRUCTURE_KIND_043A"]==1){
 beta_logistic["STRUCTURE_KIND_043A"] = coeff_logistic["STRUCTURE
    _KIND_043A1"]
```

```
} else if(bridge["STRUCTURE_KIND_043A"]==3){
  beta_logistic["STRUCTURE_KIND_043A"] = coeff_logistic["STRUCTURE
      _KIND_043A3"]
} else if(bridge["STRUCTURE_KIND_043A"]==5){
  beta_logistic["STRUCTURE_KIND_043A"] = coeff_logistic["STRUCTURE
      _KIND_043A5"]
} else if(bridge["STRUCTURE_KIND_043A"]==7){
  beta_logistic["STRUCTURE_KIND_043A"] = coeff_logistic["STRUCTURE
      _KIND_043A7"]
} else if(bridge["STRUCTURE_KIND_043A"]==8){
  beta_logistic["STRUCTURE_KIND_043A"] = coeff_logistic["STRUCTURE
      _KIND_043A8"]
}

if (bridge["STRUCTURE_TYPE_043B"]==1){
  beta_logistic["STRUCTURE_TYPE_043B"] = 0
} else if(bridge["STRUCTURE_TYPE_043B"]==2){
  beta_logistic["STRUCTURE_TYPE_043B"] = coeff_logistic["STRUCTURE
      _TYPE_043B2"]
} else if(bridge["STRUCTURE_TYPE_043B"]==4){
  beta_logistic["STRUCTURE_TYPE_043B"] = coeff_logistic["STRUCTURE
      _TYPE_043B4"]
} else if(bridge["STRUCTURE_TYPE_043B"]==5){
  beta_logistic["STRUCTURE_TYPE_043B"] = coeff_logistic["STRUCTURE
      _TYPE_043B5"]
} else if(bridge["STRUCTURE_TYPE_043B"]==6){
  beta_logistic["STRUCTURE_TYPE_043B"] = coeff_logistic["STRUCTURE
      _TYPE_043B6"]
} else if(bridge["STRUCTURE_TYPE_043B"]==11){
  beta_logistic["STRUCTURE_TYPE_043B"] = coeff_logistic["STRUCTURE
      _TYPE_043B11"]
}else if(bridge["STRUCTURE_TYPE_043B"]==19){
  beta_logistic["STRUCTURE_TYPE_043B"] = coeff_logistic["STRUCTURE
      _TYPE_043B19"]
}

if (bridge["SERVICE_LEVEL_005C"]==0){
  beta_logistic["SERVICE_LEVEL_005C"] = 0
} else if(bridge["SERVICE_LEVEL_005C"]==1){
  beta_logistic["SERVICE_LEVEL_005C"] = coeff_logistic["SERVICE_
      LEVEL_005C1"]
} else if(bridge["SERVICE_LEVEL_005C"]==2){
  beta_logistic["SERVICE_LEVEL_005C"] = coeff_logistic["SERVICE_
      LEVEL_005C2"]
} else if(bridge["SERVICE_LEVEL_005C"]==3){
  beta_logistic["SERVICE_LEVEL_005C"] = coeff_logistic["SERVICE_
```

```
      LEVEL_005C3"]
} else if(bridge["SERVICE_LEVEL_005C"]==4){
  beta_logistic["SERVICE_LEVEL_005C"] = coeff_logistic["SERVICE_
      LEVEL_005C4"]
} else if(bridge["SERVICE_LEVEL_005C"]==6){
  beta_logistic["SERVICE_LEVEL_005C"] = coeff_logistic["SERVICE_
      LEVEL_005C6"]
} else if(bridge["SERVICE_LEVEL_005C"]==7){
  beta_logistic["SERVICE_LEVEL_005C"] = coeff_logistic["SERVICE_
      LEVEL_005C7"]
} else if(bridge["SERVICE_LEVEL_005C"]==8){
  beta_logistic["SERVICE_LEVEL_005C"] = coeff_logistic["SERVICE_
      LEVEL_005C8"]
}

if (bridge["MEDIAN_CODE_033"]==0){
  beta_logistic["MEDIAN_CODE_033"] = 0
} else if(bridge["MEDIAN_CODE_033"]==1){
  beta_logistic["MEDIAN_CODE_033"] = coeff_logistic["MEDIAN_CODE_
      0331"]
} else if(bridge["MEDIAN_CODE_033"]==2){
  beta_logistic["MEDIAN_CODE_033"] = coeff_logistic["MEDIAN_CODE_
      0332"]
} else if(bridge["MEDIAN_CODE_033"]==3){
  beta_logistic["MEDIAN_CODE_033"] = coeff_logistic["MEDIAN_CODE_
      0333"]
}

beta_logistic["STRUCTURE_LEN_MT_049"] = coeff_logistic["STRUCTURE_
    LEN_MT_049"]
beta_logistic["Age"] = coeff_logistic["Age"]
beta_logistic["Inspection_Gap"] = coeff_logistic["Inspection_Gap"]
beta_logistic["SUFFICIENCY_RATING_Beg"] = coeff_logistic["
    SUFFICIENCY_RATING_Beg"]


beta_gamma = vector(, length = 0)
beta_gamma["(Intercept)"] = coeff_gamma["(Intercept)"]
if (bridge["ROUTE_PREFIX_005B"]==1){
  beta_gamma["ROUTE_PREFIX_005B"] = 0
} else if(bridge["ROUTE_PREFIX_005B"]==2){
  beta_gamma["ROUTE_PREFIX_005B"] = coeff_gamma["ROUTE_PREFIX_005
      B2"]
} else if(bridge["ROUTE_PREFIX_005B"]==3){
  beta_gamma["ROUTE_PREFIX_005B"] = coeff_gamma["ROUTE_PREFIX_005
      B3"]
```

```
} else if(bridge["ROUTE_PREFIX_005B"]==4){
 beta_gamma["ROUTE_PREFIX_005B"] = coeff_gamma["ROUTE_PREFIX_005
    B4"]
} else if(bridge["ROUTE_PREFIX_005B"]==5){
 beta_gamma["ROUTE_PREFIX_005B"] = coeff_gamma["ROUTE_PREFIX_005
    B5"]
} else if(bridge["ROUTE_PREFIX_005B"]==8){
 beta_gamma["ROUTE_PREFIX_005B"] = coeff_gamma["ROUTE_PREFIX_005
    B8"]
}

beta_gamma["TRAFFIC_LANES_ON_028A"] = coeff_gamma["TRAFFIC_LANES_
    ON_028A"]
beta_gamma["TRAFFIC_LANES_UND_028B"] = coeff_gamma["TRAFFIC_LANES_
    UND_028B"]
#beta_gamma["NAV_VERT_CLR_MT_039"] = coeff_gamma["NAV_VERT_CLR_MT_
    039"]

if (bridge["STRUCTURE_KIND_043A"]==0){
 beta_gamma["STRUCTURE_KIND_043A"] = 0
} else if(bridge["STRUCTURE_KIND_043A"]==1){
 beta_gamma["STRUCTURE_KIND_043A"] = coeff_gamma["STRUCTURE_KIND_
    043A1"]
} else if(bridge["STRUCTURE_KIND_043A"]==3){
 beta_gamma["STRUCTURE_KIND_043A"] = coeff_gamma["STRUCTURE_KIND_
    043A3"]
} else if(bridge["STRUCTURE_KIND_043A"]==5){
 beta_gamma["STRUCTURE_KIND_043A"] = coeff_gamma["STRUCTURE_KIND_
    043A5"]
} else if(bridge["STRUCTURE_KIND_043A"]==7){
 beta_gamma["STRUCTURE_KIND_043A"] = coeff_gamma["STRUCTURE_KIND_
    043A7"]
} else if(bridge["STRUCTURE_KIND_043A"]==8){
 beta_gamma["STRUCTURE_KIND_043A"] = coeff_gamma["STRUCTURE_KIND_
    043A8"]
}

if (bridge["STRUCTURE_TYPE_043B"]==1){
 beta_gamma["STRUCTURE_TYPE_043B"] = 0
} else if(bridge["STRUCTURE_TYPE_043B"]==2){
 beta_gamma["STRUCTURE_TYPE_043B"] = coeff_gamma["STRUCTURE_TYPE_
    043B2"]
} else if(bridge["STRUCTURE_TYPE_043B"]==4){
 beta_gamma["STRUCTURE_TYPE_043B"] = coeff_gamma["STRUCTURE_TYPE_
    043B4"]
} else if(bridge["STRUCTURE_TYPE_043B"]==5){
  beta_gamma["STRUCTURE_TYPE_043B"] = coeff_gamma["STRUCTURE_TYPE_
      043B5"]
 } else if(bridge["STRUCTURE_TYPE_043B"]==6){
  beta_gamma["STRUCTURE_TYPE_043B"] = coeff_gamma["STRUCTURE_TYPE_
      043B6"]
 } else if(bridge["STRUCTURE_TYPE_043B"]==11){
  beta_gamma["STRUCTURE_TYPE_043B"] = coeff_gamma["STRUCTURE_TYPE_
      043B11"]
 }else if(bridge["STRUCTURE_TYPE_043B"]==19){
  beta_gamma["STRUCTURE_TYPE_043B"] = coeff_gamma["STRUCTURE_TYPE_
      043B19"]
 }

beta_gamma["STRUCTURE_LEN_MT_049"] = coeff_gamma["STRUCTURE_LEN_MT
    _049"]

if (bridge["SERVICE_LEVEL_005C"]==0){
  beta_gamma["SERVICE_LEVEL_005C"] = 0
 } else if(bridge["SERVICE_LEVEL_005C"]==1){
  beta_gamma["SERVICE_LEVEL_005C"] = coeff_gamma["SERVICE_LEVEL_
      005C1"]
 } else if(bridge["SERVICE_LEVEL_005C"]==2){
  beta_gamma["SERVICE_LEVEL_005C"] = coeff_gamma["SERVICE_LEVEL_
      005C2"]
 } else if(bridge["SERVICE_LEVEL_005C"]==3){
  beta_gamma["SERVICE_LEVEL_005C"] = coeff_gamma["SERVICE_LEVEL_
      005C3"]
 } else if(bridge["SERVICE_LEVEL_005C"]==4){
  beta_gamma["SERVICE_LEVEL_005C"] = coeff_gamma["SERVICE_LEVEL_
      005C4"]
 } else if(bridge["SERVICE_LEVEL_005C"]==6){
  beta_gamma["SERVICE_LEVEL_005C"] = coeff_gamma["SERVICE_LEVEL_
      005C6"]
 } else if(bridge["SERVICE_LEVEL_005C"]==7){
  beta_gamma["SERVICE_LEVEL_005C"] = coeff_gamma["SERVICE_LEVEL_
      005C7"]
 } else if(bridge["SERVICE_LEVEL_005C"]==8){
  beta_gamma["SERVICE_LEVEL_005C"] = coeff_gamma["SERVICE_LEVEL_
      005C8"]
 }

beta_gamma["Age"] = coeff_gamma["Age"]
beta_gamma["Inspection_Gap"] = coeff_gamma["Inspection_Gap"]
beta_gamma["SUFFICIENCY_RATING_Beg"] = coeff_gamma["SUFFICIENCY_
    RATING_Beg"]
```

```
eta_logistic = 0
for (i in variables_logistic){
 if (is_categorical[i] == 1){
   eta_logistic = eta_logistic + as.numeric(beta_logistic[i])
  } else{
   eta_logistic = eta_logistic + as.numeric(bridge[i])*as.numeric(
      beta_logistic[i])
  }
}
p_hat = exp(eta_logistic)/(1 + exp(eta_logistic)) # Here is the
   predicted probability

eta_gamma = 0
for (i in variables_gamma){
 if (is_categorical[i] == 1){
   eta_gamma = eta_gamma + as.numeric(beta_gamma[i])
  } else{
   eta_gamma = eta_gamma + as.numeric(bridge[i])*as.numeric(beta_
      gamma[i])
  }
}
mu_hat = exp(eta_gamma)
predictions = c(p_hat, mu_hat)
return(predictions)
}
```

C.1.2 GENERATE_HISTORY FUNCTION

```
generate_history = function(bridge, coeff_logistic, coeff_gamma,
   alpha_hat, num_iter, segment_length){
sufficiency_history = matrix(0, nrow = num_iter, ncol = segment_
   length)
for (i in 1:num_iter){
   tempBridge = bridge
   sufficiency_history[i, 1] = as.numeric(tempBridge["SUFFICIENCY_
      RATING_Beg"])
   for (k in 2:segment_length) {
    predictions = get_eta(tempBridge, coeff_logistic, coeff_gamma
       )
    p_hat = predictions[1]
    mu_hat = predictions[2]
    if (runif(1) < p_hat) {
      beta_hat = alpha_hat / mu_hat
      deterioration = rgamma(1, shape = alpha_hat, rate = beta_hat
         )
      tempBridge["SUFFICIENCY_RATING_Beg"] = tempBridge["
         SUFFICIENCY_RATING_Beg"] - deterioration
      tempBridge["Age"] = tempBridge["Age"] + 1
    } else{
      tempBridge["Age"] = tempBridge["Age"] + 1
    }
    sufficiency_history[i, k] = as.numeric(tempBridge["
       SUFFICIENCY_RATING_Beg"])
  }
 }
return(sufficiency_history)
}
```

C.1.3 Curves

```
generate_curve = function(bridge, coeff_logistic, coeff_gamma,
   alpha_hat, num_iter, segment_length){
sufficiency_history = generate_history(bridge, coeff_logistic,
   coeff_gamma, alpha_hat, num_iter, segment_length)
   medians = vector(,length=segment_length)
   Q1s = vector(,length=segment_length)
   Q3s = vector(,length=segment_length)
   for (k in 1:segment_length){
    Q1s[k] = quantile(sufficiency_history[,k], probs = .25, na.rm
       =T)
    medians[k] = median(sufficiency_history[,k], na.rm=T)
    Q3s[k] = quantile(sufficiency_history[,k], probs = .75, na.rm
       =T)
   }
   Years = seq(1,segment_length,1)
   plot(Years, Q1s, col = "blue", type = "l", ylab = "Sufficiency␣
      Rating")
   points(Years,medians, col="black", type = "l")
   points(Years,Q3s, col="red", type = "l")
   legend("bottomleft", legend = c("Q1", "Median", "Q3"), col=c("
      blue", "black", "red"), lty = c(1,1))
   Pred_Suff = medians
}
```

C.1.4 Define a Bridge to Simulate

```
var_names = c("(Intercept)", "ROUTE_PREFIX_005B",
              "TRAFFIC_LANES_ON_028A",
            "TRAFFIC_LANES_UND_028B",
            "STRUCTURE_KIND_043A",
            "STRUCTURE_TYPE_043B",
            "STRUCTURE_LEN_MT_049",
            "SERVICE_LEVEL_005C",
            "MEDIAN_CODE_033",
            "Age", "Inspection_Gap",
            "SUFFICIENCY_RATING_Beg")
bridge = vector(, length = length(var_names))
names(bridge) = var_names
bridge["(Intercept)"] = 1
bridge["ROUTE_PREFIX_005B"] = 4
bridge["TRAFFIC_LANES_ON_028A"]= 2
bridge["TRAFFIC_LANES_UND_028B"] = 8
bridge["STRUCTURE_KIND_043A"] = 3
bridge["STRUCTURE_TYPE_043B"] = 2
bridge["STRUCTURE_LEN_MT_049"] = 85.6
bridge["SERVICE_LEVEL_005C"] = 1
bridge["MEDIAN_CODE_033"] = 0
bridge["Age"] = 30
bridge["Inspection_Gap"] = 1
bridge["SUFFICIENCY_RATING_Beg"] = 78.9
num_iter = 400
segment_length = 50
A = generate_curve(bridge, model_logistic$coeff, model_gamma$coeff
   , alpha_hat, num_iter, segment_length)
```

C.1.5 SSE Calculation For Mixed Logistic_Gamma Model

```
num_iter = 100
bridgeIDs = unique(as.character(segment_df$STRUCTURE_NUMBER))
sse = 0
for (m in 1:length(bridgeIDs)){
  print(m)
  temp_bridge_df = subset(logistic_df, STRUCTURE_NUMBER_008 ==
     bridgeIDs[m])
  temp_segment_df = subset(segment_df, STRUCTURE_NUMBER ==
     bridgeIDs[m])
  for (j in 1:nrow(temp_segment_df)){
    segment_length = temp_segment_df$segmentEnds[j] - temp_segment_
       df$segmentBegins[j] + 1
    temp_bridge_in_segment_df = subset(temp_bridge_df, (Old_Date >=
        temp_segment_df$segmentBegins[j]) &
                                (New_Date <= temp_segment_df$
                                        segmentEnds[j]))

    tempBridge = temp_bridge_in_segment_df[1, ]
    sufficiency_history= generate_history(tempBridge, model_
       logistic$coeff, model_gamma$coeff, alpha_hat, num_iter,
       segment_length)
    medians = vector(,length=segment_length)
    for (k in 1:segment_length){
     medians[k] = median(sufficiency_history[,k], na.rm=T)
    }
    Pred_Suff = medians
    Y= temp_bridge_in_segment_df$SUFFICIENCY_RATING_Beg
    sse = sse + sum((Y - Pred_Suff)^2)
  }
}
print(sse)
```

C.1.6 Mixed Logistic_Gamma Model Examples

```
m = logistic_df[logistic_df$STRUCTURE_NUMBER_008 == "
   000000003500130", ]
tempbri = m[1,]
coeff_logistic = model_logistic$coefficients
coeff_gamma = model_gamma$coefficients
predictions = get_eta(tempbri, model_logistic$coeff, model_gamma$
   coeff)
print(predictions)
num_iter = 100
segment_length = 60
generate_curve(tempbri, coeff_logistic, coeff_gamma, alpha_hat,
   num_iter, segment_length)

bridge = logistic_df[1,]
coeff_logistic = model_logistic$coefficients
coeff_gamma = model_gamma$coefficients
num_iter = 100
segment_length = 100
generate_curve(bridge, coeff_logistic, coeff_gamma, alpha_hat, num
   _iter, segment_length)
```

C.2 Poisson Model's Curves

C.2.1 get_Lambda Function

```
get_lambda = function(bridge, coeff_Poisson){
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(ROUTE_PREFIX_
     005B)2"] = "ROUTE_PREFIX_005B2"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(ROUTE_PREFIX_
     005B)3"] = "ROUTE_PREFIX_005B3"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(ROUTE_PREFIX_
     005B)4"] = "ROUTE_PREFIX_005B4"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(ROUTE_PREFIX_
     005B)5"] = "ROUTE_PREFIX_005B5"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(ROUTE_PREFIX_
     005B)8"] = "ROUTE_PREFIX_005B8"

  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     TYPE_043B)2"] = "STRUCTURE_TYPE_043B2"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     TYPE_043B)4"] = "STRUCTURE_TYPE_043B4"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     TYPE_043B)5"] = "STRUCTURE_TYPE_043B5"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     TYPE_043B)6"] = "STRUCTURE_TYPE_043B6"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     TYPE_043B)11"] = "STRUCTURE_TYPE_043B11"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     TYPE_043B)19"] = "STRUCTURE_TYPE_043B19"

  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     KIND_043A)1"] = "STRUCTURE_KIND_043A1"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     KIND_043A)3"] = "STRUCTURE_KIND_043A3"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     KIND_043A)5"] = "STRUCTURE_KIND_043A5"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     KIND_043A)7"] = "STRUCTURE_KIND_043A7"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(STRUCTURE_
     KIND_043A)8"] = "STRUCTURE_KIND_043A8"

  names(coeff_Poisson)[names(coeff_Poisson)=="factor(SERVICE_LEVEL
     _005C)1"] = "SERVICE_LEVEL_005C1"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(SERVICE_LEVEL
     _005C)2"] = "SERVICE_LEVEL_005C2"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(SERVICE_LEVEL
     _005C)3"] = "SERVICE_LEVEL_005C3"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(SERVICE_LEVEL
     _005C)4"] = "SERVICE_LEVEL_005C4"
  names(coeff_Poisson)[names(coeff_Poisson)=="factor(SERVICE_LEVEL
     _005C)6"] = "SERVICE_LEVEL_005C6"
```

```
names(coeff_Poisson)[names(coeff_Poisson)=="factor(SERVICE_LEVEL
   _005C)7"] = "SERVICE_LEVEL_005C7"
names(coeff_Poisson)[names(coeff_Poisson)=="factor(SERVICE_LEVEL
   _005C)8"] = "SERVICE_LEVEL_005C8"

names(coeff_Poisson)[names(coeff_Poisson)=="factor(MEDIAN_CODE_
   033)1"] = "MEDIAN_CODE_0331"
names(coeff_Poisson)[names(coeff_Poisson)=="factor(MEDIAN_CODE_
   033)2"] = "MEDIAN_CODE_0332"
names(coeff_Poisson)[names(coeff_Poisson)=="factor(MEDIAN_CODE_
   033)3"] = "MEDIAN_CODE_0333"

variables_Poisson = c("(Intercept)", "ROUTE_PREFIX_005B", "
   TRAFFIC_LANES_ON_028A",
                 "TRAFFIC_LANES_UND_028B", "NAV_VERT_CLR_MT_039"
                 "STRUCTURE_KIND_043A", "STRUCTURE_TYPE_043B",
                 "STRUCTURE_LEN_MT_049", "SERVICE_LEVEL_005C", "
                    MEDIAN_CODE_033",
                 "Age", "Inspection_Gap", "SUFFICIENCY_RATING_Beg
                    ")

is_categorical = vector()
is_categorical["(Intercept)"] = 0
is_categorical["ROUTE_PREFIX_005B"] = 1
is_categorical["TRAFFIC_LANES_ON_028A"] = 0
is_categorical["TRAFFIC_LANES_UND_028B"] = 0
is_categorical["NAV_VERT_CLR_MT_039"] = 0
is_categorical["STRUCTURE_KIND_043A"] = 1
is_categorical["STRUCTURE_TYPE_043B"] = 1
is_categorical["STRUCTURE_LEN_MT_049"] = 0
is_categorical["SERVICE_LEVEL_005C"] = 1
is_categorical["MEDIAN_CODE_033"] = 1
is_categorical["Age"] = 0
is_categorical["Inspection_Gap"] = 0
is_categorical["SUFFICIENCY_RATING_Beg"] = 0

bridge["(Intercept)"] = 1
beta_Poisson = vector(, length = 0)
beta_Poisson["(Intercept)"] = coeff_Poisson["(Intercept)"]
if (bridge["ROUTE_PREFIX_005B"]==1){
 beta_Poisson["ROUTE_PREFIX_005B"] = 0
} else if(bridge["ROUTE_PREFIX_005B"]==2){
 beta_Poisson["ROUTE_PREFIX_005B"] = coeff_Poisson["ROUTE_PREFIX
    _005B2"]
} else if(bridge["ROUTE_PREFIX_005B"]==3){
 beta_Poisson["ROUTE_PREFIX_005B"] = coeff_Poisson["ROUTE_PREFIX
```

```
      _005B3"]
  } else if(bridge["ROUTE_PREFIX_005B"]==4){
   beta_Poisson["ROUTE_PREFIX_005B"] = coeff_Poisson["ROUTE_PREFIX
      _005B4"]
  } else if(bridge["ROUTE_PREFIX_005B"]==5){
   beta_Poisson["ROUTE_PREFIX_005B"] = coeff_Poisson["ROUTE_PREFIX
      _005B5"]
  } else if(bridge["ROUTE_PREFIX_005B"]==8){
   beta_Poisson["ROUTE_PREFIX_005B"] = coeff_Poisson["ROUTE_PREFIX
      _005B8"]
  }
 beta_Poisson["TRAFFIC_LANES_ON_028A"] = coeff_Poisson["TRAFFIC_
    LANES_ON_028A"]
 beta_Poisson["TRAFFIC_LANES_UND_028B"] = coeff_Poisson["TRAFFIC_
    LANES_UND_028B"]
 beta_Poisson["NAV_VERT_CLR_MT_039"] = coeff_Poisson["NAV_VERT_
    CLR_MT_039"]

 if (bridge["STRUCTURE_KIND_043A"]==0){
   beta_Poisson["STRUCTURE_KIND_043A"] = 0
  } else if(bridge["STRUCTURE_KIND_043A"]==1){
   beta_Poisson["STRUCTURE_KIND_043A"] = coeff_Poisson["STRUCTURE_
      KIND_043A1"]
  } else if(bridge["STRUCTURE_KIND_043A"]==3){
   beta_Poisson["STRUCTURE_KIND_043A"] = coeff_Poisson["STRUCTURE_
      KIND_043A3"]
  } else if(bridge["STRUCTURE_KIND_043A"]==5){
   beta_Poisson["STRUCTURE_KIND_043A"] = coeff_Poisson["STRUCTURE_
      KIND_043A5"]
  } else if(bridge["STRUCTURE_KIND_043A"]==7){
   beta_Poisson["STRUCTURE_KIND_043A"] = coeff_Poisson["STRUCTURE_
      KIND_043A7"]
  } else if(bridge["STRUCTURE_KIND_043A"]==8){
   beta_Poisson["STRUCTURE_KIND_043A"] = coeff_Poisson["STRUCTURE_
      KIND_043A8"]
  }

 if (bridge["STRUCTURE_TYPE_043B"]==1){
   beta_Poisson["STRUCTURE_TYPE_043B"] = 0
  } else if(bridge["STRUCTURE_TYPE_043B"]==2){
   beta_Poisson["STRUCTURE_TYPE_043B"] = coeff_Poisson["STRUCTURE_
      TYPE_043B2"]
  } else if(bridge["STRUCTURE_TYPE_043B"]==4){
   beta_Poisson["STRUCTURE_TYPE_043B"] = coeff_Poisson["STRUCTURE_
      TYPE_043B4"]
  } else if(bridge["STRUCTURE_TYPE_043B"]==5){
```

```
 beta_Poisson["STRUCTURE_TYPE_043B"] = coeff_Poisson["STRUCTURE_
    TYPE_043B5"]
} else if(bridge["STRUCTURE_TYPE_043B"]==6){
 beta_Poisson["STRUCTURE_TYPE_043B"] = coeff_Poisson["STRUCTURE_
    TYPE_043B6"]
} else if(bridge["STRUCTURE_TYPE_043B"]==11){
 beta_Poisson["STRUCTURE_TYPE_043B"] = coeff_Poisson["STRUCTURE_
    TYPE_043B11"]
}else if(bridge["STRUCTURE_TYPE_043B"]==19){
 beta_Poisson["STRUCTURE_TYPE_043B"] = coeff_Poisson["STRUCTURE_
    TYPE_043B19"]
}

if (bridge["SERVICE_LEVEL_005C"]==0){
 beta_Poisson["SERVICE_LEVEL_005C"] = 0
} else if(bridge["SERVICE_LEVEL_005C"]==1){
 beta_Poisson["SERVICE_LEVEL_005C"] = coeff_Poisson["SERVICE_
    LEVEL_005C1"]
} else if(bridge["SERVICE_LEVEL_005C"]==2){
 beta_Poisson["SERVICE_LEVEL_005C"] = coeff_Poisson["SERVICE_
    LEVEL_005C2"]
} else if(bridge["SERVICE_LEVEL_005C"]==3){
 beta_Poisson["SERVICE_LEVEL_005C"] = coeff_Poisson["SERVICE_
    LEVEL_005C3"]
} else if(bridge["SERVICE_LEVEL_005C"]==4){
 beta_Poisson["SERVICE_LEVEL_005C"] = coeff_Poisson["SERVICE_
    LEVEL_005C4"]
} else if(bridge["SERVICE_LEVEL_005C"]==6){
 beta_Poisson["SERVICE_LEVEL_005C"] = coeff_Poisson["SERVICE_
    LEVEL_005C6"]
} else if(bridge["SERVICE_LEVEL_005C"]==7){
 beta_Poisson["SERVICE_LEVEL_005C"] = coeff_Poisson["SERVICE_
    LEVEL_005C7"]
} else if(bridge["SERVICE_LEVEL_005C"]==8){
 beta_Poisson["SERVICE_LEVEL_005C"] = coeff_Poisson["SERVICE_
    LEVEL_005C8"]
}

if (bridge["MEDIAN_CODE_033"]==0){
 beta_Poisson["MEDIAN_CODE_033"] = 0
} else if(bridge["MEDIAN_CODE_033"]==1){
 beta_Poisson["MEDIAN_CODE_033"] = coeff_Poisson["MEDIAN_CODE_
    0331"]
} else if(bridge["MEDIAN_CODE_033"]==2){
 beta_Poisson["MEDIAN_CODE_033"] = coeff_Poisson["MEDIAN_CODE_
    0332"]
```

```
  } else if(bridge["MEDIAN_CODE_033"]==3){
    beta_Poisson["MEDIAN_CODE_033"] = coeff_Poisson["MEDIAN_CODE_
        0333"]
  }

  beta_Poisson["STRUCTURE_LEN_MT_049"] = coeff_Poisson["STRUCTURE_
      LEN_MT_049"]
  beta_Poisson["Age"] = coeff_Poisson["Age"]
  beta_Poisson["Inspection_Gap"] = coeff_Poisson["Inspection_Gap"]
  beta_Poisson["SUFFICIENCY_RATING_Beg"] = coeff_Poisson["
      SUFFICIENCY_RATING_Beg"]

  eta_Poisson = 0
  for (i in variables_Poisson){
    if (is_categorical[i] == 1){
      eta_Poisson = eta_Poisson + as.numeric(beta_Poisson[i])
    } else{
      eta_Poisson = eta_Poisson + as.numeric(bridge[i])*as.numeric(
          beta_Poisson[i])
    }
  }
  Lambda_hat = exp(eta_Poisson)
  return(Lambda_hat)
}
```

C.2.2 POISSON CURVES

```
Poisson_Pred_Suff = function(bridge, coeff_Poisson, num_iter,
    segment_length) {
  suff_history = matrix(0, nrow = num_iter, ncol = segment_length)
  for (i in 1:num_iter){
    tempBridge = bridge
    suff_history[i, 1] = as.numeric(tempBridge["SUFFICIENCY_RATING_
        Beg"])
    for (k in 2:segment_length) {
      Lambda_hat = get_lambda(tempBridge, coeff_Poisson)
      deterioration = rpois(1 , Lambda_hat)
        tempBridge["SUFFICIENCY_RATING_Beg"] = tempBridge["
            SUFFICIENCY_RATING_Beg"] - deterioration
        tempBridge["Age"] = tempBridge["Age"] + 1
        suff_history[i,k] = as.numeric(tempBridge["SUFFICIENCY_
            RATING_Beg"])
      }
    }
  medians = vector(,length=segment_length)
  Q1s = vector(,length=segment_length)
  Q3s = vector(,length=segment_length)
  for (k in 1:segment_length){
    Q1s[k] = quantile(suff_history[,k], probs = .25, na.rm=T)
    medians[k] = median(suff_history[,k], na.rm=T)
    Q3s[k] = quantile(suff_history[,k], probs = .75, na.rm=T)
  }
  Years = seq(1,segment_length,1)
  plot(Years, Q1s, col = "blue", type = "l", ylab = "Sufficiency␣
     Rating")
  points(Years,medians, col="black", type = "l")
  points(Years,Q3s, col="red", type = "l")
  legend("bottomleft", legend = c("Q1", "Median", "Q3"), col=c("
     blue", "black", "red"), lty = c(1,1))
  Pois_Pred_Suff = medians
  return(Pois_Pred_Suff)
}
```

C.2.3 SSE for Poisson Model

```
num_iter = 100
bridgeIDs = unique(as.character(segment_df$STRUCTURE_NUMBER))
SSE = 0
for (m in 1:length(bridgeIDs)){
  print(m)
  temp_bridge_df = subset(logistic_df, STRUCTURE_NUMBER_008 ==
     bridgeIDs[m])
  temp_segment_df = subset(segment_df, STRUCTURE_NUMBER ==
     bridgeIDs[m])
  for (j in 1:nrow(temp_segment_df)){
    segment_length = temp_segment_df$segmentEnds[j] - temp_segment_
       df$segmentBegins[j] + 1
    temp_bridge_in_segment_df = subset(temp_bridge_df, (Old_Date >=
        temp_segment_df$segmentBegins[j]) &
                              (New_Date <= temp_segment_df$
                                segmentEnds[j]))
    suff_history = matrix(0, nrow = num_iter, ncol = segment_length
       )
    for (i in 1:num_iter){
      tempBridge = temp_bridge_in_segment_df[1, ]
      suff_history[i, 1] = as.numeric(tempBridge["SUFFICIENCY_
         RATING_Beg"])
      for (k in 2:(segment_length)) {
        Lambda_hat = get_lambda(tempBridge, model_Poisson$coeff)
          deterioration = rpois(1 , Lambda_hat)
         tempBridge["SUFFICIENCY_RATING_Beg"] = tempBridge["
            SUFFICIENCY_RATING_Beg"] - deterioration
         tempBridge["Age"] = tempBridge["Age"] + 1
        suff_history[i, k] = as.numeric(tempBridge["SUFFICIENCY_
           RATING_Beg"])
      }
    }
    Pois_Pred_Suff = Poisson_Pred_Suff(tempBridge, model_Poisson$
       coeff, num_iter, segment_length)
    Y= temp_bridge_in_segment_df$SUFFICIENCY_RATING_Beg
    SSE = SSE + sum((Y - Pois_Pred_Suff)^2)
   }
}
print(SSE)
```

C.2.4 Poisson Model Examples

```
var_names = c("(Intercept)", "ROUTE_PREFIX_005B",
           "TRAFFIC_LANES_ON_028A",
           "TRAFFIC_LANES_UND_028B",
           "NAV_VERT_CLR_MT_039",
           "STRUCTURE_KIND_043A",
           "STRUCTURE_TYPE_043B",
           "STRUCTURE_LEN_MT_049",
           "SERVICE_LEVEL_005C",
           "MEDIAN_CODE_033",
           "Age", "Inspection_Gap",
           "SUFFICIENCY_RATING_Beg")
bridge = vector(, length = length(var_names))
names(bridge) = var_names
bridge["(Intercept)"] = 1
bridge["ROUTE_PREFIX_005B"] = 4
bridge["TRAFFIC_LANES_ON_028A"]= 2
bridge["TRAFFIC_LANES_UND_028B"] = 8
bridge["NAV_VERT_CLR_MT_039"]= 24
bridge["STRUCTURE_KIND_043A"] = 3
bridge["STRUCTURE_TYPE_043B"] = 2
bridge["STRUCTURE_LEN_MT_049"] = 85.6
bridge["SERVICE_LEVEL_005C"] = 1
bridge["MEDIAN_CODE_033"] = 0
bridge["Age"] = 30
bridge["Inspection_Gap"] = 1
bridge["SUFFICIENCY_RATING_Beg"] = 78.9
num_iter = 400
segment_length = 50
```

```
C = Poisson_Pred_Suff(bridge, model_Poisson$coeff, num_iter,
   segment_length)

m = logistic_df[logistic_df$STRUCTURE_NUMBER_008 == "
   000000003500130", ]
tempbri = m[1,]
coeff_Poisson = model_Poisson$coefficients
num_iter = 100
segment_length = 60
B = Poisson_Pred_Suff(tempbri, coeff_Poisson, num_iter, segment_
   length)

bridge = logistic_df[1,]
coeff_Poisson = model_Poisson$coefficients
num_iter = 100
segment_length = 100
A = Poisson_Pred_Suff(bridge, coeff_Poisson, num_iter, segment_
   length)
```