R for Loss Data Analytics

An open text authored by the Actuarial Community

2018-08-07

Contents

Prerequisites

Welcome to R for Loss Data Analytics. This site provides files that generate R codes to support the online text Loss Data Analytics. Within the site, there are explanations available to guide you through the specific R functions used in a certain code chunk so that you can understand what the code is doing and why we need to use such a function. In order to use this site most effectively, it is helpful to learn and understand the following programming tools in R. By having this background knowledge, you can manipulate the given code to suit your application.

0.1 Basic Programming Tools

These are tools that *you should know* as you use this site. **Introduction to R** is a free course available on DataCamp that can help you understand them in detail.

- Assign values to variables
- Construct and select particular elements of
  - Vectors
  - Matrices
  - Data frames

0.2 Further Programming Tools

These are tools that are *helpful to know*. These are tools used throughout the site, though we deem them as optional to know since it is not required to understand the code provided.

- Install and call packages within RStudio
- Construct plots
  - Histograms
  - Lines
  - Curves
  - Multiple plots in one output
  - Include labels, x and y limits, color and a legend

- Construct and call functions
- Construct loops
- Construct if else statements
- Read and manipulate data
  - Read in files
  - Take a subset of the data
  - Get certain elements of the data

0.3 R Markdown

We suggest using R Markdown when *writing a report*. R Markdown is a package available in RStudio that allows you to record both your code and output in a single file. You can refer here to learn more about this package.

Chapter 1

Introduction to Loss Data Analytics

*This file contains illustrative **R** code for computing analysis on the Property Fund data. When reviewing this code, you should open an **R** session, copy-and-paste the code, and see it perform. Then, you will be able to change parameters, look up commands, and so forth, as you go. This code uses the dataset PropertyFundin_sample.csv*

1.1 Case Study: Wisconsin Property Fund

Read in Property Fund data.

```
in_sample <- read.csv("Data/PropertyFundInsample.csv", header = T,
                      na.strings = c("."), stringsAsFactors = FALSE)
in_sample_2010 <- subset(in_sample, Year == 2010)
```

A few quick notes on these commands:

- `read.csv` reads a csv file and creates a data frame from it, with cases corresponding to rows and variables to columns in the file.
- The assignment operator `<-` is analogous to an equal sign in mathematics. The command `in_sample <- read.csv("Data/PropertyFundInsample.csv", header=T, na.strings=c("."), stringsAsFactors = FALSE)` means we give the name `in_sample` to the data read.
- `The subset()` function is used to select variables and observations. In this illustration, we selected observations from year 2010.

1.1.1 Fund Claims Variables

1.1.1.1 Claim Frequency Distribution

In 2010 there were 1,110 policyholders in the property fund.

1.1.1.1.1 Property Fund Distribution for 2010

Table 1.1 shows the distribution of the 1,377 claims.

Table 1.1

```
library(pander)
table <- as.data.frame(table(in_sample_2010$Freq))
names(table) <- c("Number of Claims", "Frequency")
pander(t(table))
```

Table 1.1: Table continues below

Number of Claims	0	1	2	3	4	5	6	7	8	9	10	11
Frequency	707	209	86	40	18	12	9	4	6	1	3	2

Number of Claims	13	14	15	16	17	18	19	30	39	103	239
Frequency	1	2	1	2	1	1	1	1	1	1	1

The average number of claims for this sample was 1.24 (= 1377/1110). See table 1.2 below.

Table 1.2

```
pander(summary(in_sample_2010$Freq))
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0	0	0	1.241	1	239

A few quick notes on these commands:

- Many useful **R** functions come in packages and to use these functions you have to install them. One way to install a package is by using the command line `install.packages("<the package's name>")`. In addition, to read more about a function you use the command `help("function name")`.
- The `pander` function is used here to create nicer tables than regular **R** output. To use this function you need to download the `pander` package. For the normal **R** output in the illustration above, use the command line `summary(in_sample_2010$Freq)`.
- The `names()` function is used to to get or assign names of an object . In this illustration, we assigned `Number of Claims` and `Frequency` to the two columns in the data frame.
- The `t()` function is used to transpose a data frame or a matrix.

1.1.1.2 Average Severity Distribution for 2010

Table 1.3 summarizes the sample distribution of average severity from the 403 policyholders.

Table 1.3

```
in_sample_pos_2010 <- subset(in_sample_2010, yAvg > 0)
pander(summary(in_sample_pos_2010$yAvg))
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
166.7	2226	4951	56332	11900	12922218

```
length(in_sample_pos_2010$yAvg)
```

```
[1] 403
```

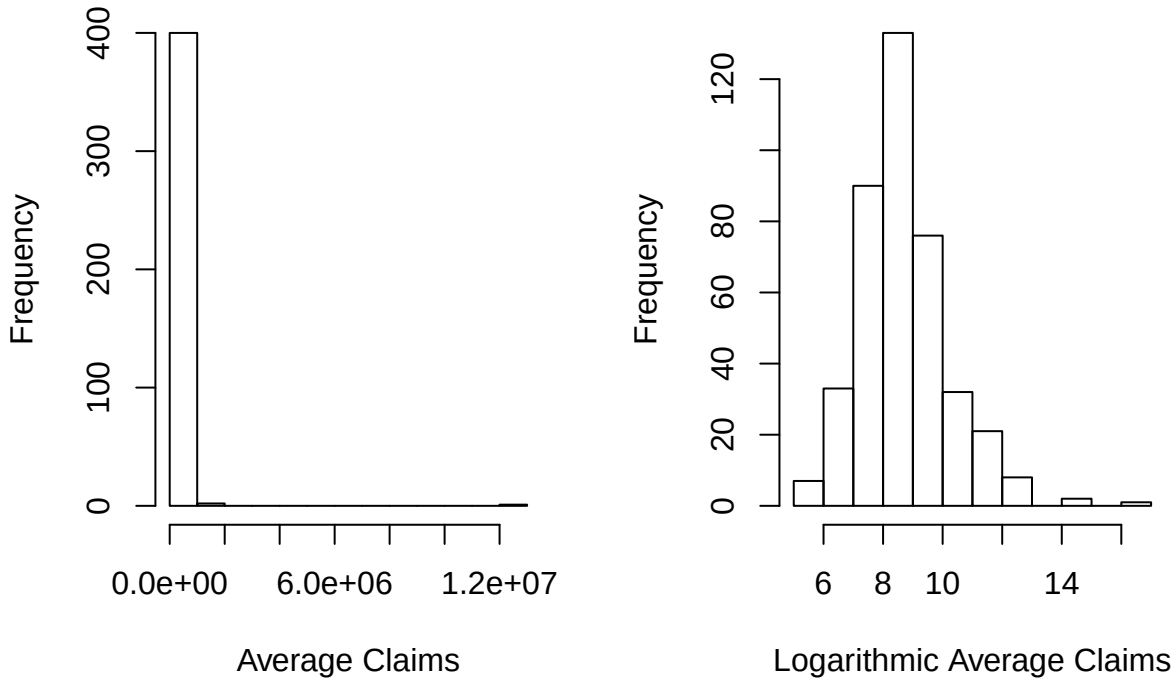
Note: The `length()` function sets the length of a vector (list) or other objects.

1.1.1.2.1 Plot of Average Claims

Figure 1.2 provides further information about the distribution of sample claims, showing a distribution that is dominated by this single large claim so that the histogram is not very helpful. Even when removing the large claim, you will find a distribution that is skewed to the right. A generally accepted technique is to work with claims in logarithmic units especially for graphical purposes; the corresponding figure in the right-hand panel is much easier to interpret.

Figure 1.2

```
par(mfrow = c(1, 2))
hist(in_sample_pos_2010$yAvg, main = "", xlab = "Average Claims")
hist(log(in_sample_pos_2010$yAvg), main = "", xlab = "Logarithmic Average Claims")
```



A few quick notes on these commands:

- The `par(mfrow)` function is handy for creating a simple multi-paneled plot. `mfrow` is a vector of length 2, where the first argument specifies the number of rows and the second the number of columns of plots.
- The `hist()` computes a histogram of the given data values. You put the name of your dataset in between the parentheses of this function.

1.1.2 Rating Variables

Earlier we considered a sample of 1,110 observations which may seem like a lot. However, as we will seen in our forthcoming applications, because of the preponderance of zeros and the skewed nature of claims, actuaries typically yearn for more data. One common approach that we adopt here is to examine outcomes from multiple years, thus increasing the sample size.

1.1.2.1 Average Claims Over Time

Table 1.4 shows that the average claim varies over time.

Table 1.4

```
library(doBy)
t_1a <- summaryBy(Freq ~ Year, data = in_sample,
                  FUN = function (x) { c(m = mean(x), num = length(x)) } )
t_1b <- summaryBy(yAvg ~ Year, data = in_sample,
                  FUN = function (x) { c(m = mean(x), num = length(x)) } )
t_1c <- summaryBy(BCcov ~ Year, data = in_sample,
                  FUN = function (x) { c(m = mean(x), num = length(x)) } )
table1_in <- cbind(t_1a[1], t_1a[2], t_1b[2], t_1c[2], t_1a[3])
names(table1_in) <- c("Year", "Average Freq", "Average Sev", "Average Coverage",
                      "No. of Policyholders")
pander(table1_in)
```

Year	Average Freq	Average Sev	Average Coverage	No. of Policyholders
2006	0.9515	9695	32498186	1154
2007	1.167	6544	35275949	1138
2008	0.9742	5311	37267485	1125
2009	1.219	4572	40355382	1112
2010	1.241	20452	41242070	1110

A few quick notes on these commands:

- The `summaryBy()` function provides summary statistics of a variable across different groups. You need to install the `doBy` package to use the command.
- The `cbind()` combines vector, matrix or data frame arguments by columns. The row number of the two datasets must be equal.
- The `c()` function combines its arguments to form a vector.

1.1.2.2 Frequency and Claims Statistics of Full Data

For a different look at this five-year sample, Table 1.5 summarizes the distribution of our two outcomes, frequency and claims amount. In each case, the average exceeds the median, suggesting that the distributions are right-skewed.

Table 1.5

```
BCcov.div1000 <- (in_sample$BCcov) / 1000

t_1 <- summaryBy(Freq ~ 1, data = in_sample,
          FUN = function (x) { c(ma = min(x), m1 = median(x), m = mean(x), mb = max(x)) } )
```

```
names(t_1) <- c("Minimum", "Median", "Average", "Maximum")
t_2 <- summaryBy(yAvg ~ 1, data = in_sample,
          FUN = function (x) { c(ma = min(x), m1 = median(x), m = mean(x), mb = max(x)) } )
names(t_2) <- c("Minimum", "Median", "Average", "Maximum")
t_3 <- summaryBy(Deduct ~ 1, data = in_sample,
          FUN = function (x) { c(ma = min(x), m1 = median(x), m = mean(x), mb = max(x)) } )
names(t_3) <- c("Minimum", "Median", "Average", "Maximum")
t_4 <- summaryBy(BCcov.div1000 ~ 1, data = in_sample,
          FUN = function (x) { c(ma = min(x), m1 = median(x), m = mean(x), mb = max(x)) } )
names(t_4) <- c("Minimum", "Median","Average", "Maximum")
table_2 <- rbind(t_1, t_2, t_3, t_4)
table_2a <- round(table_2, 3)
row_label <- rbind("Claim Frequency", "Claim Severity", "Deductible", "Coverage (000's)")
table_2aa <- cbind(row_label, as.matrix(table_2a))
pander(table_2aa)
```

	Minimum	Median	Average	Maximum
Claim Frequency	0	0	1.109	263
Claim Severity	0	0	9291.565	12922217.84
Deductible	500	1000	3364.87	1e+05
Coverage (000's)	8.937	11353.566	37280.855	2444796.98

A few quick notes on these commands:

- The `rbind()` combines vector, matrix or data frame arguments by rows. The column of the two datasets must be same.
- The `round()` function rounds the values in its first argument to the specified number of decimal places (default 0).

1.1.2.3 Rating Variable Description

Table 1.6 describes the rating variables considered in this chapter. To handle the skewness, we henceforth focus on logarithmic transformations of coverage and deductibles. See table 1.6 below for variables and variable descriptions.

Table 1.6

```
des <- read.table(header = TRUE, text = '
 Variable Description
  "BCcov"  "Total building and content coverage in dollars"
  "Deduct"  "Deductible in dollars"
"Entity Type"   "Categorical variable that is one of six types:
(Village, City, County, Misc, School, or Town)"
"alarm_credit"  "Categorical variable that is one of four types:
(0%, 5%, 10%, or 15%), for automatic smoke alarms in main rooms"
"NoClaimCredit" "Binary variable to indicate no claims in the past two years"
"Fire5" "Binary variable to indicate the fire class is below 5.
(The range of fire class is 0~10)" ')
```

```
pander(des)
```

Variable	Description
BCcov	Total building and content coverage in dollars
Deduct	Deductible in dollars
Entity Type	Categorical variable that is one of six types: (Village, City, County, Misc, School, or Town)
alarm_credit	Categorical variable that is one of four types: (0%, 5%, 10%, or 15%), for automatic smoke alarms in main rooms
NoClaimCredit	Binary variable to indicate no claims in the past two years
Fire5	Binary variable to indicate the fire class is below 5. (The range of fire class is 0~10)

1.1.2.4 Frequency and Claims by Rating Variables

To get a sense of the relationship between the non-continuous rating variables and claims, Table 1.7 relates the claims outcomes to these categorical variables. Table 1.7 shows claims summary by Entity Type, Fire Class, and No Claim Credit.

Table 1.7

```
# Table 1.7
by_var_summ <- function (datasub) {
  temp_a <- summaryBy(Freq ~ 1 , data = datasub,
                      FUN = function (x) { c(m = mean(x), num = length(x)) } )
  datasub_1 <- subset(datasub, yAvg > 0)
  temp_b <- summaryBy(yAvg ~ 1, data = datasub_1,
                      FUN = function (x) { c(m = mean(x)) } )
  temp_c <- merge(temp_a, temp_b, all.x = T)[c(2, 1, 3)]
  temp_c1 <- as.matrix(temp_c)
  return(temp_c1)
}

datasub <- subset(in_sample, TypeVillage == 1);
t_1 <- by_var_summ(datasub)
datasub <- subset(in_sample, TypeCity == 1);
t_2 <- by_var_summ(datasub)
datasub <- subset(in_sample, TypeCounty == 1);
t_3 <- by_var_summ(datasub)
datasub <- subset(in_sample, TypeMisc == 1);
t_4 <- by_var_summ(datasub)
datasub <- subset(in_sample, TypeSchool == 1);
t_5 <- by_var_summ(datasub)
datasub <- subset(in_sample, TypeTown == 1);
t_6 <- by_var_summ(datasub)
datasub <- subset(in_sample, Fire5 == 0);
t_7 <- by_var_summ(datasub)
datasub <- subset(in_sample, Fire5 == 1);
t_8 <- by_var_summ(datasub)
datasub <- subset(in_sample, in_sample$NoClaimCredit == 0);
t_9 <- by_var_summ(datasub)
```

```
datasub <- subset(in_sample, in_sample$NoClaimCredit == 1);
t_10 <- by_var_summ(datasub)
t_11 <- by_var_summ(in_sample)

table_a <- rbind(t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_10, t_11)
table_aa <- round(table_a, 3)
row_label <- rbind("Village", "City", "County", "Misc", "School",
                   "Town", "Fire5--No", "Fire5--Yes", "NoClaimCredit--No",
                   "NoClaimCredit--Yes", "Total")
table_4 <- cbind(row_label, as.matrix(table_aa))
```

```
pander(table_4)
```

	Freq.num	Freq.m	yAvg.m
Village	1341	0.452	10645.206
City	793	1.941	16924.035
County	328	4.899	15453.206
Misc	609	0.186	43036.076
School	1597	1.434	64346.394
Town	971	0.103	19831.048
Fire5–No	2508	0.502	13935.421
Fire5–Yes	3131	1.596	41421.263
NoClaimCredit–No	3786	1.501	31365.085
NoClaimCredit–Yes	1853	0.31	30498.714
Total	5639	1.109	31206.155

Table 1.8 shows claims summary by Entity Type and Alarm Credit

Table 1.8

```
by_var_summ <- function(datasub) {
  temp_a <- summaryBy(Freq ~ AC00 , data = datasub,
                      FUN = function(x) { c(m = mean(x), num = length(x)) } )
  datasub_1 <- subset(datasub, yAvg > 0)
  if (nrow(datasub_1) == 0) { n <- nrow(datasub)
    return(c(0, 0, n))
  } else
  {
    temp_b <- summaryBy(yAvg ~ AC00, data = datasub_1,
                        FUN = function(x) { c(m = mean(x)) } )
    temp_c <- merge(temp_a, temp_b, all.x = T)[c(2, 4, 3)]
    temp_c1 <- as.matrix(temp_c)
    return(temp_c1)
  }
}
alarm_c <- 1 * (in_sample$AC00 == 1) + 2 * (in_sample$AC05 == 1) +
           3 * (in_sample$AC10 == 1) + 4 * (in_sample$AC15 == 1)
by_var_credit<-function(ACnum){
datasub <-  subset(in_sample, TypeVillage == 1 & alarm_c == ACnum);
t_1 <- by_var_summ(datasub)
datasub <-  subset(in_sample, TypeCity == 1 & alarm_c == ACnum);
t_2 <- by_var_summ(datasub)
datasub <-  subset(in_sample, TypeCounty == 1 & alarm_c == ACnum);
```

```
t_3 <- by_var_summ(datasub)
datasub <-  subset(in_sample, TypeMisc == 1 & alarm_c == ACnum);
t_4 <- by_var_summ(datasub)
datasub <-  subset(in_sample, TypeSchool == 1 & alarm_c == ACnum);
t_5 <- by_var_summ(datasub)
datasub <-  subset(in_sample, TypeTown == 1 & alarm_c ==ACnum);
t_6 <- by_var_summ(datasub)
datasub <-  subset(in_sample, alarm_c == ACnum);
t_7 <- by_var_summ(datasub)
table_a <- rbind(t_1, t_2, t_3, t_4, t_5, t_6, t_7)
table_aa <- round(table_a, 3)
row_label <- rbind("Village", "City", "County", "Misc", "School", "Town","Total")
table_4 <- cbind(row_label,as.matrix(table_aa))
}
table_4a <- by_var_credit(1)  # claims summary by entity type and alarm credit == 00
table_4b <- by_var_credit(2)  # claims summary by entity type and alarm credit == 05
table_4c <- by_var_credit(3)  # claims summary by entity type and alarm credit == 10
table_4d <- by_var_credit(4)  # claims summary by entity type and alarm credit == 15
```

```
pander(table_4a)  # claims summary by entity type and alarm credit == 00
```

	Freq.m	yAvg.m	Freq.num
Village	0.326	11077.997	829
City	0.893	7575.979	244
County	2.14	16012.719	50
Misc	0.117	15122.127	386
School	0.422	25522.708	294
Town	0.083	25257.084	808
Total	0.318	15118.491	2611

```
pander(table_4b)  # claims summary by entity type and alarm credit == 05
```

		Freq.m	yAvg.m	Freq.num
t__3	Village	0.278	8086.057	54
	City	2.077	4150.125	13
	County	0	0	1
	Misc	0.278	13063.933	18
	School	0.41	14575.003	122
	Town	0.194	3937.29	31
	Total	0.431	10762.112	239

```
pander(table_4c)  # claims summary by entity type and alarm credit == 10
```

	Freq.m	yAvg.m	Freq.num
Village	0.5	8792.376	50
City	1.258	8625.169	31
County	2.125	11687.969	8
Misc	0.077	3923.375	26
School	0.488	11596.912	168
Town	0.091	2338.06	44

	Freq.m	yAvg.m	Freq.num
Total	0.517	10194.094	327

```
pander(table_4d)  # claims summary by entity type and alarm credit == 15
```

	Freq.m	yAvg.m	Freq.num
Village	0.725	10543.752	408
City	2.485	20469.514	505
County	5.513	15475.74	269
Misc	0.341	87020.878	179
School	2.008	85139.974	1013
Town	0.261	9489.613	88
Total	2.093	41458.312	2462

Chapter 2

Frequency Distributions

*This file contains illustrative **R** code for computing important count distributions. When reviewing this code, you should open an **R** session, copy-and-paste the code, and see it perform. Then, you will be able to change parameters, look up commands, and so forth, as you go.*

2.1 Basic Distributions

2.1.1 Poisson Distribution

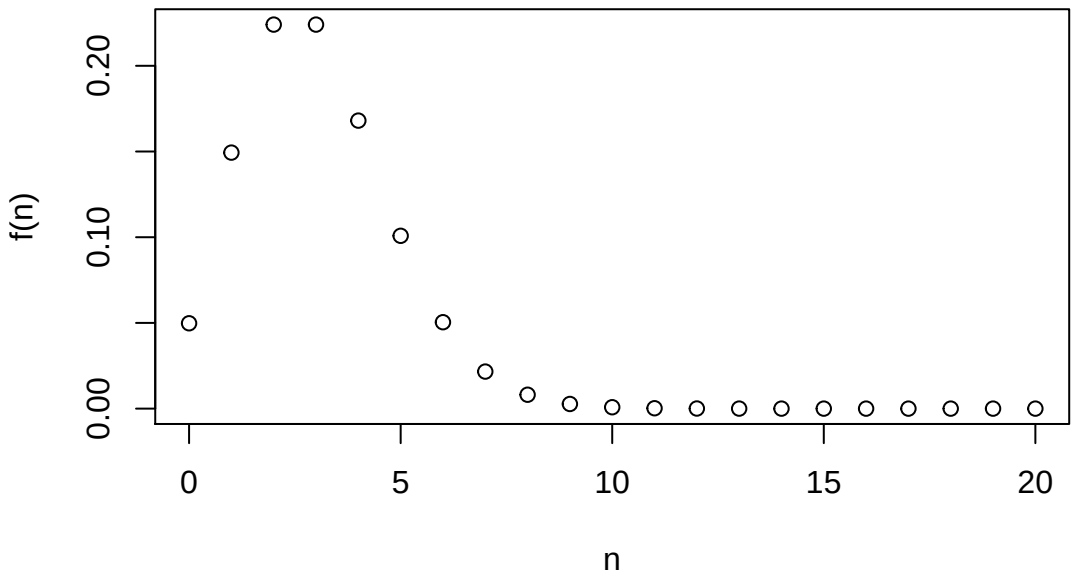
This sections shows how to compute and graph probability mass and distribution functions for the Poisson distribution.

2.1.1.1 Probability Mass Function (pmf)

```
lambda <- 3
N <- seq(0, 20, 1)
# Get the probability mass function using "dpois"
( fn <- dpois(N, lambda) )
```

```
 [1] 4.978707e-02 1.493612e-01 2.240418e-01 2.240418e-01 1.680314e-01
 [6] 1.008188e-01 5.040941e-02 2.160403e-02 8.101512e-03 2.700504e-03
[11] 8.101512e-04 2.209503e-04 5.523758e-05 1.274713e-05 2.731529e-06
[16] 5.463057e-07 1.024323e-07 1.807629e-08 3.012715e-09 4.756919e-10
[21] 7.135379e-11
```

```
# Visualize the probability mass function
plot(N, fn, xlab = "n", ylab = "f(n)")
```



A few quick notes on these commands.

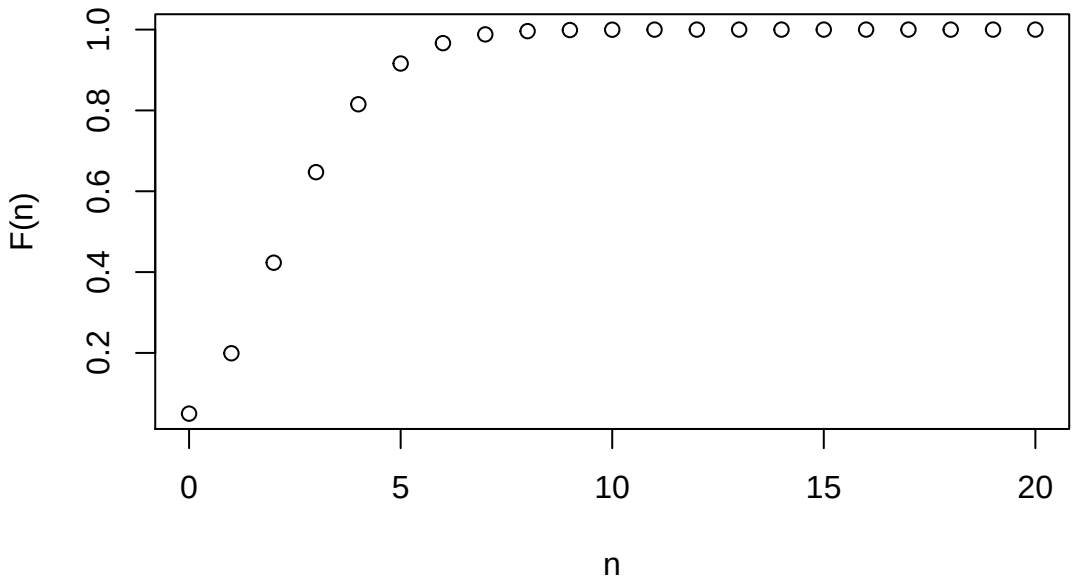
- The assigment operator `<-` is analogous to an equal sign in mathematics. The command `lambda <- 3` means to give a value of "3" to quantity *lambda*.
- `seq` is short-hand for sequence.
- `dpois` is a built-in command in **R** for generating the "density" (actually the mass) function of the Poisson distribution. Use the online help (`help("dpois")`) to learn more about this function.
- The open paren `(`, close paren `)` tells **R** to display the output of a calculation to the screen.
- `plot` is a very handy command for displaying results graphically.

2.1.1.2 (Cumulative) Probability Distribution Function (cdf)

```
# Get the cumulative distribution function using "ppois"
( Fn <- ppois(N, lambda) )
```

```
 [1] 0.04978707 0.19914827 0.42319008 0.64723189 0.81526324 0.91608206
 [7] 0.96649146 0.98809550 0.99619701 0.99889751 0.99970766 0.99992861
[13] 0.99998385 0.99999660 0.99999933 0.99999988 0.99999998 1.00000000
[19] 1.00000000 1.00000000 1.00000000
```

```
# Visualize the cumulative distribution function
plot(N, Fn, xlab = "n", ylab = "F(n)") # cdf
```



2.1.2 Negative Binomial Distribution

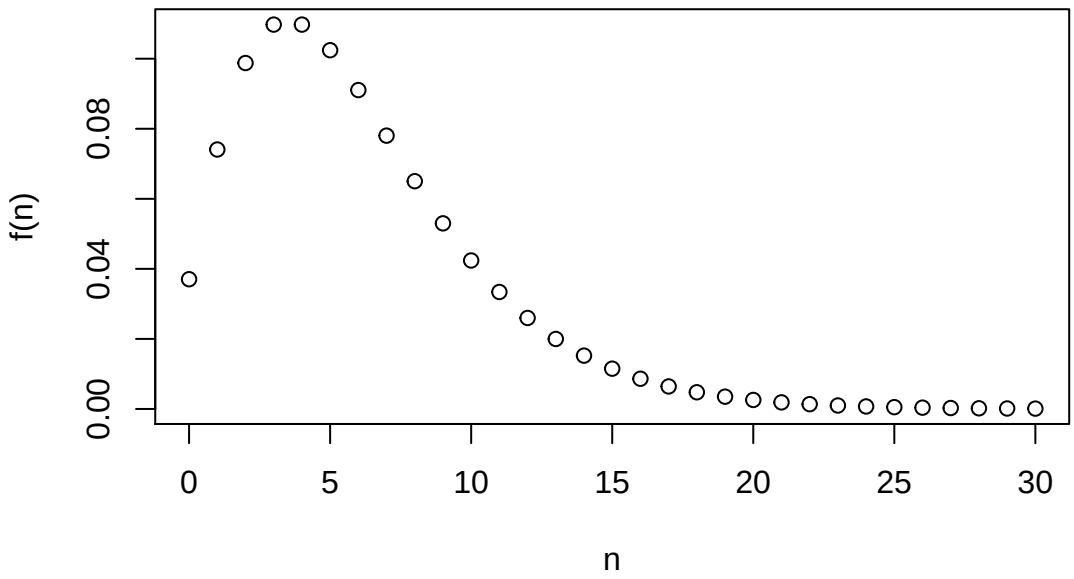
This section shows how to compute and graph probability mass and distribution functions for the negative binomial distribution. You will also learn how to plot two functions on the same graph.

2.1.2.1 Probability Mass Function (pmf)

```
alpha <- 3
theta <- 2
prob <- 1 / (1 + theta)
N <- seq(0, 30, 1)
# Get the probability mass function using "dnbinom"
( fn <- dnbinom(N, alpha, prob) )
```

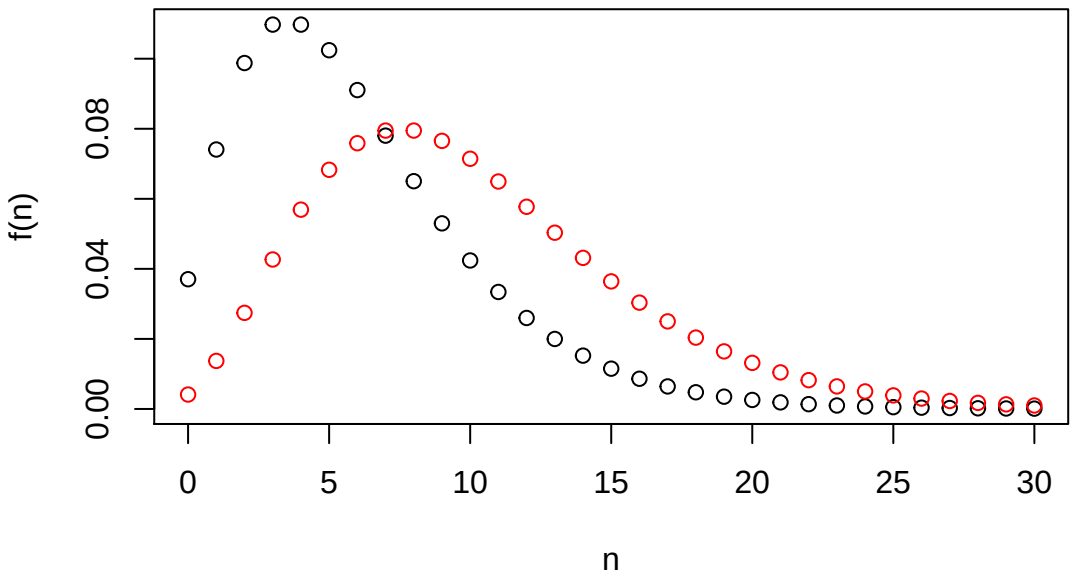
```
 [1] 3.703704e-02 7.407407e-02 9.876543e-02 1.097394e-01 1.097394e-01
 [6] 1.024234e-01 9.104303e-02 7.803688e-02 6.503074e-02 5.298801e-02
[11] 4.239041e-02 3.339850e-02 2.597661e-02 1.998201e-02 1.522439e-02
[16] 1.150287e-02 8.627153e-03 6.428075e-03 4.761537e-03 3.508501e-03
[21] 2.572901e-03 1.878626e-03 1.366273e-03 9.900532e-04 7.150384e-04
[26] 5.148277e-04 3.696199e-04 2.646661e-04 1.890472e-04 1.347233e-04
[31] 9.580323e-05
```

```
# Visualize the probability mass function
plot(N, fn, xlab = "n", ylab = "f(n)") # pmf
```



2.1.2.1.1 Plot Two Functions on The Same Graph

```
# Plot different negative binomial distributions on the same figure
alpha_1 <- 3
alpha_2 <- 5
theta <- 2
prob <- 1 / (1 + theta)
fn_1 <- dnbinom(N, alpha_1, prob)
fn_2 <- dnbinom(N, alpha_2, prob)
plot(N, fn_1, xlab = "n", ylab = "f(n)")
lines(N, fn_2, col = "red", type = "p")
```



A couple notes on these commands:

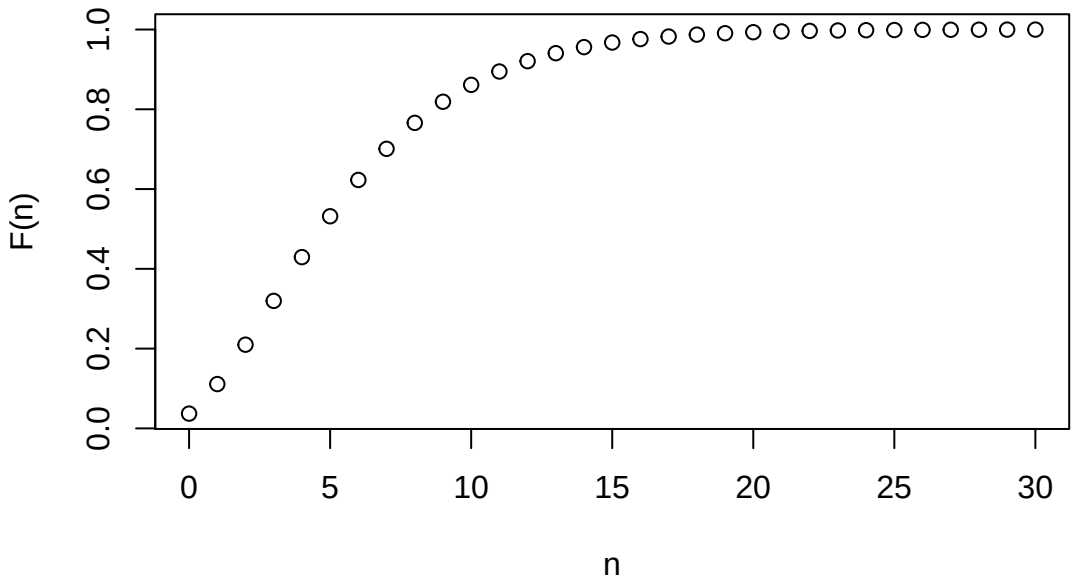
- You can enter more than one command on a line; separate them using the ; semi-colon.
- `lines` is very handy for superimposing one graph on another.
- When making complex graphs with more than one function, consider using different colors. The `col = "red"` tells **R** to use the color red when plotting symbols.

2.1.2.2 (Cumulative) Probability Distribution Function (cdf)

```
# Get the distribution function using "pnbinom"
( Fn <- pnbinom(N, alpha, prob) )
```

```
 [1] 0.03703704 0.11111111 0.20987654 0.31961591 0.42935528 0.53177869
 [7] 0.62282172 0.70085861 0.76588935 0.81887735 0.86126776 0.89466626
[13] 0.92064288 0.94062489 0.95584927 0.96735214 0.97597930 0.98240737
[19] 0.98716891 0.99067741 0.99325031 0.99512894 0.99649521 0.99748526
[25] 0.99820030 0.99871513 0.99908475 0.99934942 0.99953846 0.99967319
[31] 0.99976899
```

```
plot(N, Fn, xlab = "n", ylab = "F(n)")  # cdf
```

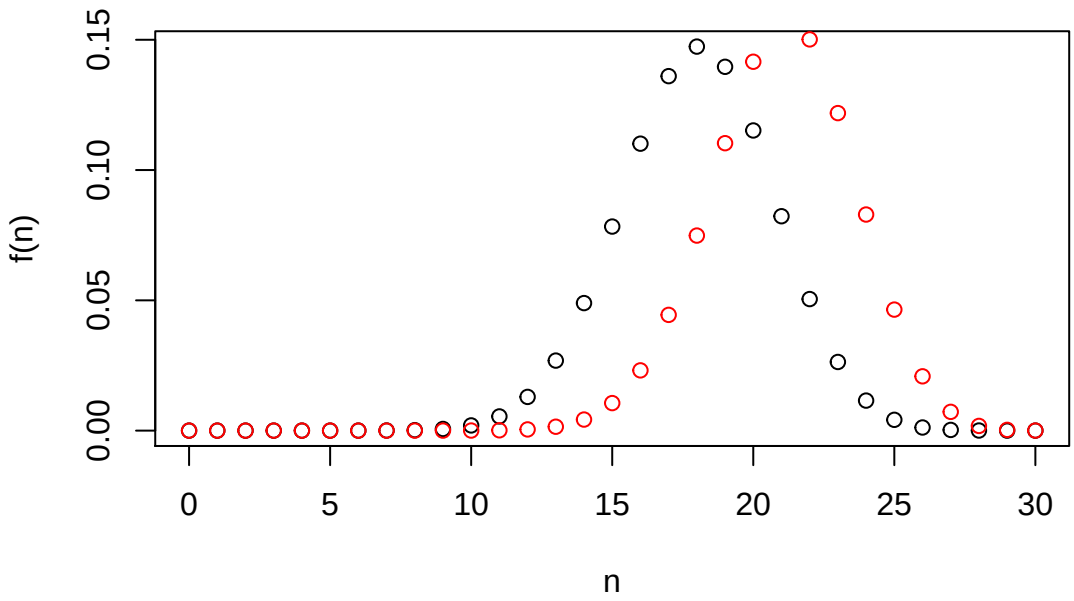


2.1.3 Binomial Distribution

This section shows how to compute and graph probability mass and distribution functions for the binomial distribution.

2.1.3.1 Probability Mass Function (pmf)

```
# Plot different binomial distributions on the same figure
size <- 30
prob <- 0.6
N <- seq(0, 30, 1)
fn <- dbinom(N, size, prob)
plot(N, fn, xlab = "n", ylab = "f(n)")  # pdf
fn2 <- dbinom(N, size, 0.7)
lines(N, fn2, col = "red", type = "p")
```

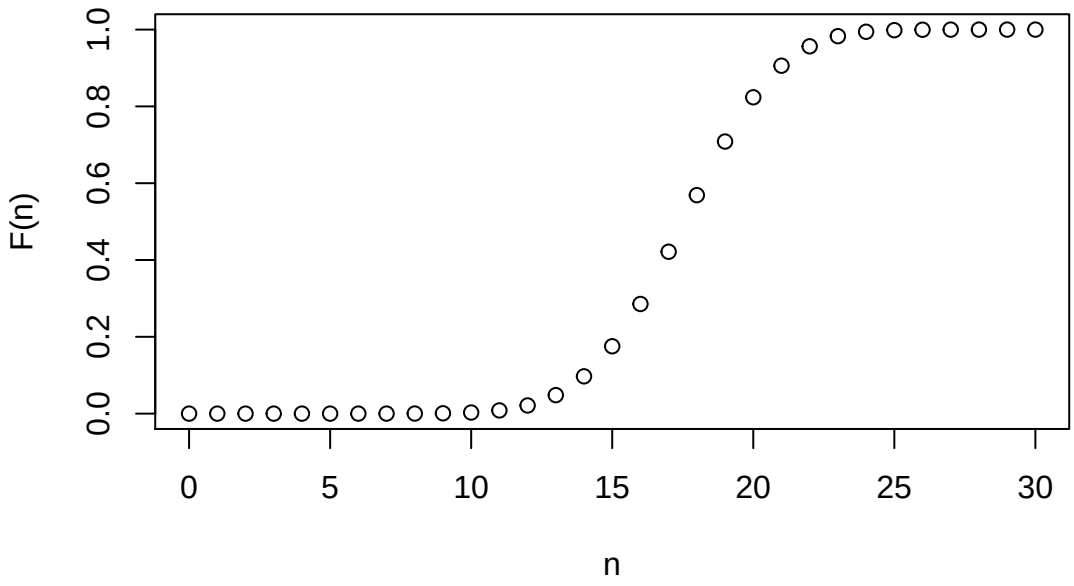


2.1.3.2 (Cumulative) Probability Distribution Function (cdf)

```
# Get the distribution function using "pbinom"
( Fn <- pbinom(N, size, prob) )
```

```
 [1] 1.152922e-12 5.303439e-11 1.181456e-09 1.697936e-08 1.769332e-07
 [6] 1.424573e-06 9.222321e-06 4.932503e-05 2.222679e-04 8.563920e-04
[11] 2.853883e-03 8.301584e-03 2.123988e-02 4.811171e-02 9.705684e-02
[16] 1.753691e-01 2.854956e-01 4.215343e-01 5.689095e-01 7.085281e-01
[21] 8.237135e-01 9.059888e-01 9.564759e-01 9.828170e-01 9.943412e-01
[26] 9.984899e-01 9.996867e-01 9.999526e-01 9.999954e-01 9.999998e-01
[31] 1.000000e+00
```

```
plot(N, Fn, xlab = "n", ylab = "F(n)")  # cdf
```



2.2 (*a*,*b*,0) Class of Distributions

This section shows how to compute recursively a distribution in the $(a,b,0)$ class. The specific example is a Poisson. However, by changing values of a and b, you can use the same recursion for negative binomial and binomial, the other two members of the $(a,b,0)$ class.

```
lambda <- 3
a <- 0
b <- lambda
# This loop calculates the (a,b,0) recursive probabilities for the Poisson distribution
p <- rep(0, 20)
# Get the probability at n = 0 to start the recursive formula
p[1] <- exp(-lambda)
for (i in 1:19)
  {
  p[i+1] <- (a + b / i) * p[i]  # Probability of i-th element using the ab0 formula
  }
p
```

```
 [1] 4.978707e-02 1.493612e-01 2.240418e-01 2.240418e-01 1.680314e-01
 [6] 1.008188e-01 5.040941e-02 2.160403e-02 8.101512e-03 2.700504e-03
[11] 8.101512e-04 2.209503e-04 5.523758e-05 1.274713e-05 2.731529e-06
[16] 5.463057e-07 1.024323e-07 1.807629e-08 3.012715e-09 4.756919e-10
```

```
# Check using the "dpois" command
dpois(seq(0, 20, 1), lambda = 3)
```

```
 [1] 4.978707e-02 1.493612e-01 2.240418e-01 2.240418e-01 1.680314e-01
 [6] 1.008188e-01 5.040941e-02 2.160403e-02 8.101512e-03 2.700504e-03
```

```
[11] 8.101512e-04 2.209503e-04 5.523758e-05 1.274713e-05 2.731529e-06
[16] 5.463057e-07 1.024323e-07 1.807629e-08 3.012715e-09 4.756919e-10
[21] 7.135379e-11
```

A couple notes on these commands.

- There are many basic math commands in **R** such as `exp` for exponentials.
- This demo illustrates the use of the `for` loop, one of many ways of doing recursive calculations.

2.3 Estimating Frequency Distributions

2.3.1 Singapore Data

This section loads the `SingaporeAuto.csv` dataset and checks the names of variables and the dimensions of the data. To have a glimpse at the data, the first 8 observations are listed.

```
Singapore <- read.csv("Data/SingaporeAuto.csv", quote = "", header = TRUE)
# Check the names, dimensions in the file and list the first 8 observations ;
names(Singapore)
```

```
 [1] "SexInsured"  "Female"      "VehicleType" "PC"          "Clm_Count"
 [6] "Exp_weights" "LNWEIGHT"    "NCD"         "AgeCat"      "AutoAge0"
[11] "AutoAge1"    "AutoAge2"    "AutoAge"     "VAgeCat"     "VAgecat1"
```

```
dim(Singapore)  # check number of observations and variables in the data
```

```
[1] 7483   15
```

```
Singapore[1:4, ]  # list the first 4 observations
```

```
  SexInsured Female VehicleType PC Clm_Count Exp_weights    LNWEIGHT NCD
1          U      0           T  0         0   0.6680356 -0.40341383  30
2          U      0           T  0         0   0.5667351 -0.56786326  30
3          U      0           T  0         0   0.5037645 -0.68564629  30
4          U      0           T  0         0   0.9144422 -0.08944106  20
  AgeCat AutoAge0 AutoAge1 AutoAge2 AutoAge VAgeCat VAgecat1
1      0        0        0        0       0       0        2
2      0        0        0        0       0       0        2
3      0        0        0        0       0       0        2
4      0        0        0        0       0       0        2
```

```
attach(Singapore)  # attach dataset
```

A few quick notes on these commands:

- The `names()`function is used to get or assign names of an object. In this illustration, it was used to get the variables names.
- The `dim()` function is used to retrieve or set the dimensions of an object.
- When you attach a dataset using the `attach()` function, variable names in the database can be accessed by simply giving their names.

2.3.2 Claim Frequency Distribution

The table below gives the distribution of observed claims frequency. The `Clm_Count` variable is the number of automobile accidents per policyholder.

```
table(Clm_Count)
```

```
Clm_Count
   0    1    2    3
6996  455   28    4
```

```
( n <- length(Clm_Count) )  # number of insurance policies
```

```
[1] 7483
```

2.3.3 Visualize The Loglikelihood Function

Before maximizing, let us start by visualizing the logarithmic likelihood function. We will fit the claim counts for the Singapore data to the Poisson model. As an illustration, first assume that $\lambda = 0.5$. The claim count, likelihood, and its logarithmic version, for five observations is

```
# Five typical observations
Clm_Count[2245:2249]
```

```
[1] 3 0 1 0 3
```

```
# Probabilities
dpois(Clm_Count[2245:2249], lambda = 0.5)
```

```
[1] 0.01263606 0.60653066 0.30326533 0.60653066 0.01263606
```

```
# Logarithmic probabilities
log(dpois(Clm_Count[2245:2249], lambda = 0.5))
```

```
[1] -4.371201 -0.500000 -1.193147 -0.500000 -4.371201
```

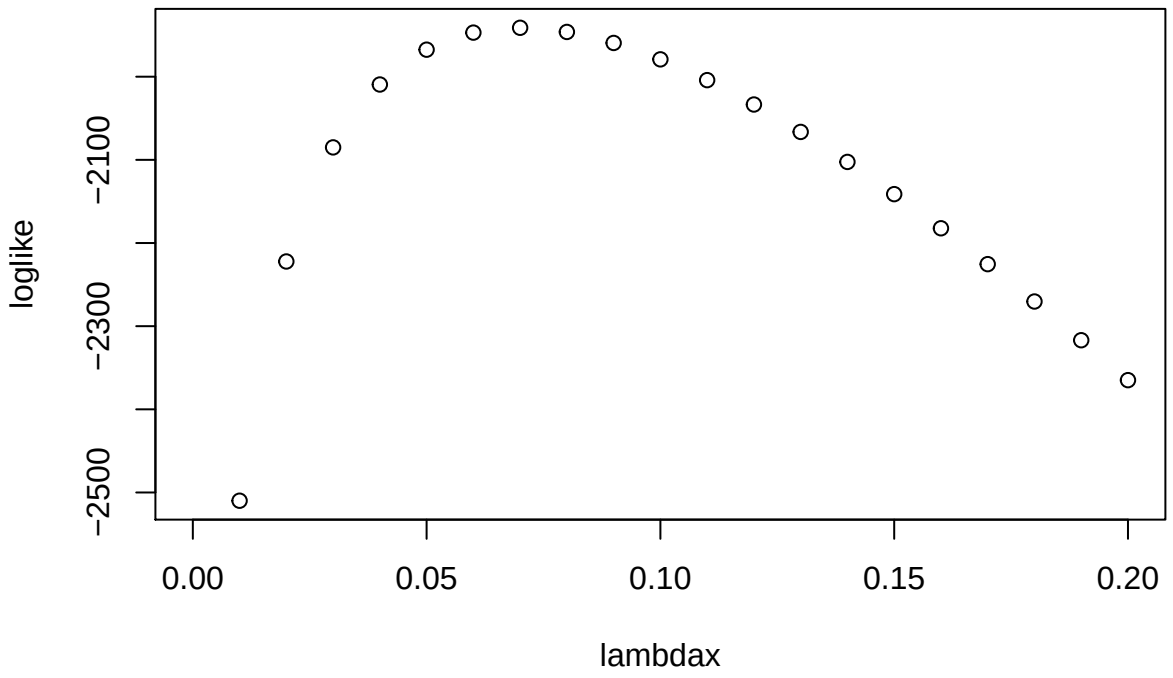
By hand, you can check that the sum of log likelihoods for these five observations is -10.9355492. In the same way, the sum of all 7483 observations is

```
sum(log(dpois(Clm_Count, lambda = 0.5)))
```

```
[1] -4130.591
```

Of course, this is only for the choice $\lambda = 0.5$. The following code defines the log likelihood to be a function of λ and plots the function for several choices of λ:

```
loglikPois <- function (parms){
# Defines the Poisson loglikelihood function
  lambda = parms[1]
  llk <- sum(log(dpois(Clm_Count, lambda)))
  llk
}
lambdax <- seq(0, .2, .01)
loglike <- 0 * lambdax
for (i in 1:length(lambdax))
  {
  loglike[i] <- loglikPois(lambdax[i])
}
plot(lambdax, loglike)
```



If we had to guess, from this plot we might say that the maximum value of the log likelihood was around 0.07.

2.3.4 The Maximum Likelihood Estimate of Poisson Distribution

From calculus, we know that the maximum likelihood estimator (*mle*) of the Poisson distribution parameter equals the average claim count. For our data, this is

```
mean(Clm_Count)
```

```
[1] 0.06989175
```

As an alternative, let us use an optimization routine `nlminb`. Most optimization routines try to minimize functions instead of maximize them, so we first define the *negative* loglikelihood function.

```
negloglikPois <- function (parms){
# Defines the (negative) Poisson loglikelihood function
  lambda <- parms[1]
  llk <- -sum(log(dpois(Clm_Count, lambda)))
  llk
}
ini.Pois <- 1
zop.Pois <- nlminb(ini.Pois, negloglikPois, lower = c(1e-6), upper = c(Inf))
print(zop.Pois)  # In output, $par = MLE of lambda, $objective = - loglikelihood value
```

```
$par
[1] 0.06989175
```

```
$objective
[1] 1941.178

$convergence
[1] 0

$iterations
[1] 17

$evaluations
function gradient
      23       20

$message
[1] "relative convergence (4)"
```

So, the maximum likelihood estimate, `zop.Pois$par` = 0.0698918 is exactly the same as the value that we got by hand.

Because actuarial analysts calculate Poisson mle's so regularly, here is another way of doing the calculation using the `glm`, *generalized linear model*, package.

```
count_poisson1 <- glm(Clm_Count ~ 1, poisson(link = log))
summary(count_poisson1)
```

```

Call:
glm(formula = Clm_Count ~ 1, family = poisson(link = log))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.3739  -0.3739  -0.3739  -0.3739   4.0861

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.66081    0.04373  -60.85   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 2887.2  on 7482  degrees of freedom
Residual deviance: 2887.2  on 7482  degrees of freedom
AIC: 3884.4

Number of Fisher Scoring iterations: 6
```

```
( lambda_hat <- exp(count_poisson1$coefficients) )
```

```
(Intercept)
 0.06989175
```

A few quick notes on these commands and results:

- The `glm()`function is used to fit generalized linear models. See `help(glm)` for other modeling options. In order to get the results we use the `summary()` function.

- In the output, `call` reminds us what model we ran and what options were specified.
- The `Deviance Residuals` shows the distribution of the deviance residuals for individual cases used in the model.
- The next part of the output shows the coefficient (maximum likelihood estimate of $\log(\lambda)$), its standard error, the z-statistic and the associated p-value.
- To get the estimated λ we take the exp(coefficient) `lambda_hat <- exp(count_poisson1$coefficients)`.

2.3.5 The Maximum Likelihood Estimate of The Negative Binomial Distribution

In the same way, here is code for determining the maximum likelihood estimates for the negative binomial distribution.

```
dnb <- function (y, r, beta){
# Defines the (negative) negative binomial loglikelihood function
  gamma(y + r) / gamma(r) / gamma(y + 1) * (1 / (1 + beta))^r * (beta / (1 + beta))^y
}
loglikNB <- function (parms){
  r = parms[1]
  beta = parms[2]
  llk <- -sum(log(dnb(Clm_Count, r, beta)))
  llk
}

ini.NB <- c(1, 1)
zop.NB <- nlminb(ini.NB, loglikNB, lower = c(1e-6, 1e-6), upper = c(Inf, Inf))
print(zop.NB)  # In output, $par = (MLE of r, MLE of beta), $objective = - loglikelihood value
```

```
$par
[1] 0.87401622 0.07996624

$objective
[1] 1932.383

$convergence
[1] 0

$iterations
[1] 24

$evaluations
function gradient
      30       60

$message
[1] "relative convergence (4)"
```

Two quick notes:

- There are two parameters for this distribution, so that calculation by hand is not a good alternative.
- The maximum likelihood estimator of r, 0.8740162, is not an integer.

2.4 Goodness of Fit

This section shows how to check the adequacy of the Poisson and negative binomial models for the Singapore data.

First, note that the variance for the *count* data is 0.0757079 which is greater than the mean value, 0.0698918. This suggests that the negative binomial model is preferred to the Poisson model.

Second, we will compute the *Pearson goodness-of-fit statistic*.

2.4.1 Pearson Goodness-of-Fit Statistic

The table below gives the distribution of fitted claims frequency using Poisson distribution $n \times p_k$

```
table_1p = cbind(n * (dpois(0, lambda_hat)),
                 n * (dpois(1, lambda_hat)),
                 n * (dpois(2, lambda_hat)),
                 n * (dpois(3, lambda_hat)),
                 n * (1 - ppois(3, lambda_hat)))
# or n * (1 - dpois(0,lambda_hat) - dpois(1, lambda_hat) -
# dpois(2, lambda_hat) - dpois(3, lambda_hat)))
actual <- data.frame(table(Clm_Count))[,2];
actual[5] <- 0  # assign 0 to claim counts greater than or equal to 4 in observed data

table_2p <- rbind(c(0, 1, 2, 3, "4+"), actual, round(table_1p, digits = 2))
rownames(table_2p) <- c("Number","Actual", "Estimated Using Poisson")
table_2p
```

```
                        [,1]      [,2]     [,3]    [,4]  [,5]
Number                  "0"       "1"      "2"     "3"   "4+"
Actual                  "6996"    "455"    "28"    "4"   "0"
Estimated Using Poisson "6977.86" "487.69" "17.04" "0.4" "0.01"
```

For goodness of fit, consider Pearson's chi-square statistic below. The degrees of freedom (df) equals the number of cells minus one minus the number of estimated parameters.

```
# PEARSON GOODNESS-OF-FIT STATISTIC
diff = actual - table_1p
( Pearson_p <- sum(diff * diff / table_1p) )
```

```
[1] 41.98438
```

```
# p-value
1 - pchisq(Pearson_p, df = 5 - 1 - 1)
```

```
[1] 4.042861e-09
```

The large value of the goodness of fit statistic 41.984382 or the small p value indicates that there is a large difference between actual counts and those anticipated under the Poisson model.

2.4.2 Negative Binomial Goodness-of-Fit Statistic

Here is another way of determining the maximum likelihood estimator of the negative binomial distribution.

```
library(MASS)
fm_nb <- glm.nb(Clm_Count ~ 1, link = log)
summary(fm_nb)
```

```
Call:
glm.nb(formula = Clm_Count ~ 1, link = log, init.theta = 0.8740189897)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.3667  -0.3667  -0.3667  -0.3667   3.4082

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.66081    0.04544  -58.55   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Negative Binomial(0.874) family taken to be 1)

    Null deviance: 2435.5  on 7482  degrees of freedom
Residual deviance: 2435.5  on 7482  degrees of freedom
AIC: 3868.8

Number of Fisher Scoring iterations: 1

              Theta:  0.874
          Std. Err.:  0.276

 2 x log-likelihood:  -3864.767
```

With these new estimates (or you could use the general procedure we introduced earlier), we can produce a table of counts and fitted counts and use this to calculate the goodness-of-fit statistic.

```
fm_nb$theta
```

```
[1] 0.874019
```

```
beta <- exp(fm_nb$coefficients) / fm_nb$theta
prob <- 1/(1+beta)

table_1nb = cbind(n * (dnbinom(0, size = fm_nb$theta, prob)),
                  n * (dnbinom(1, size = fm_nb$theta, prob)),
                  n * (dnbinom(2, size = fm_nb$theta, prob)),
                  n * (dnbinom(3, size = fm_nb$theta, prob)),
                  n * (dnbinom(4, size = fm_nb$theta, prob)))
table_2nb <- rbind(c(0, 1, 2, 3, "4+"), actual, round(table_1nb, digits = 2))
rownames(table_2nb) <- c("Number","Actual", "Estimated Using Neg Bin")
table_2nb
```

```
                        [,1]     [,2]     [,3]    [,4]   [,5]
Number                  "0"      "1"      "2"     "3"    "4+"
Actual                  "6996"   "455"    "28"    "4"    "0"
Estimated Using Neg Bin "6996.4" "452.78" "31.41" "2.23" "0.16"
```

```
# PEARSON GOODNESS-OF-FIT STATISTIC
diff = actual - table_1nb
(  Pearson_nb = sum(diff * diff / table_1nb) )
```

```
[1] 1.95024
```

```
# p-value
1 - pchisq(Pearson_nb, df = 5 - 2 - 1)
```

```
[1] 0.3771472
```

The small value of the goodness of fit statistic 1.9502395 or the high p value 0.3771472 both indicate that the negative binomial provides a better fit to the data than the Poisson.

Chapter 3

Modeling Loss Severities

*This file contains illustrative **R** code for computing important count distributions. When reviewing this code, you should open an **R** session, copy-and-paste the code, and see it perform. Then, you will be able to change parameters, look up commands, and so forth, as you go.*

3.1 Required packages

```
library(actuar)
library(VGAM)
```

3.2 Gamma Distribution

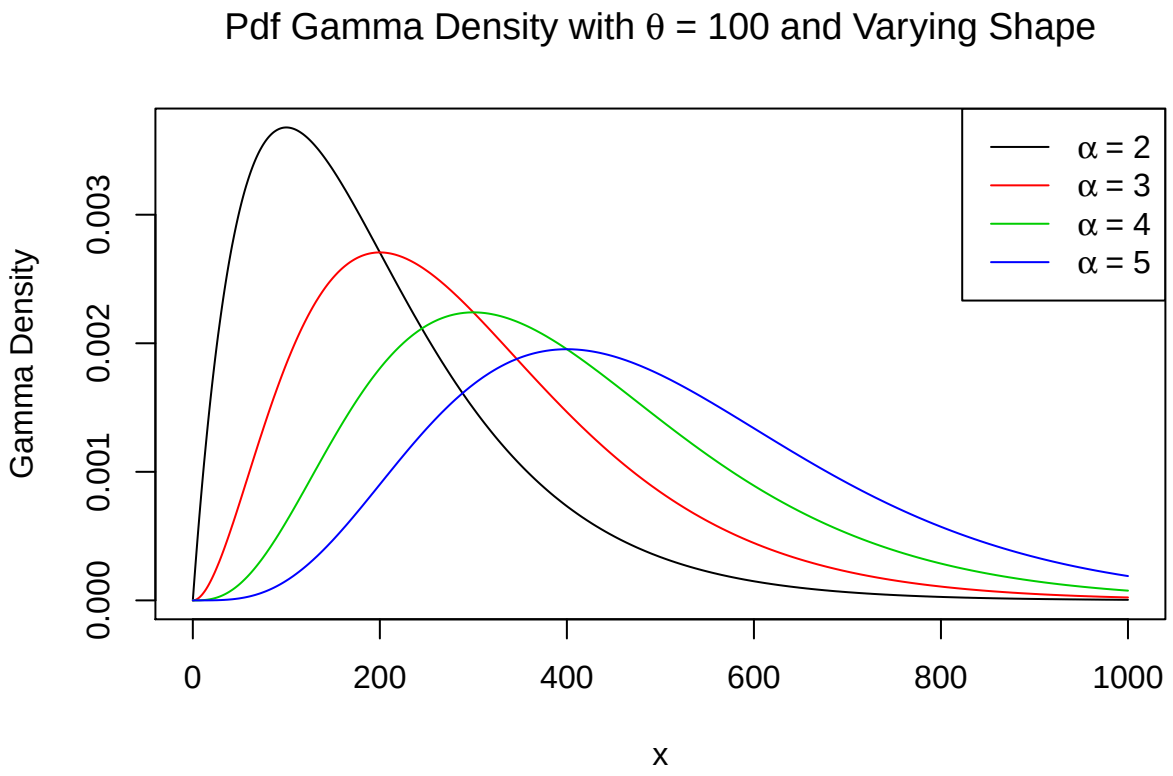
This section demonstrates the effect of the shape and scale parameters on the gamma density.

3.2.1 Varying The Shape Parameter

The graph shows the gamma density functions with varying shape parameters (α)

```
# Example 1: gamma distribution
# Define a grid
x <- seq(0, 1000, by = 1)

# Define a set of scale and shape parameters
scale_param <- seq(100, 250, by = 50)
shape_param <- 2:5

# Varying the shape parameter
plot(x, dgamma(x, shape = shape_param[1], scale = 100), type = "l", ylab = "Gamma Density")

for (k in 2:length(shape_param)) {
  lines(x, dgamma(x, shape = shape_param[k], scale = 100), col = k)
}
legend("topright", c(expression(alpha ~ '= 2'), expression(alpha ~ '= 3'),
                     expression(alpha ~ '= 4'), expression(alpha ~ '= 5')),
```

```
       lty = 1, col = 1:4)
title(substitute(paste("Pdf Gamma Density with"," ",theta," = 100"," ",
                       "and Varying Shape")))
```



A few quick notes on these commands :

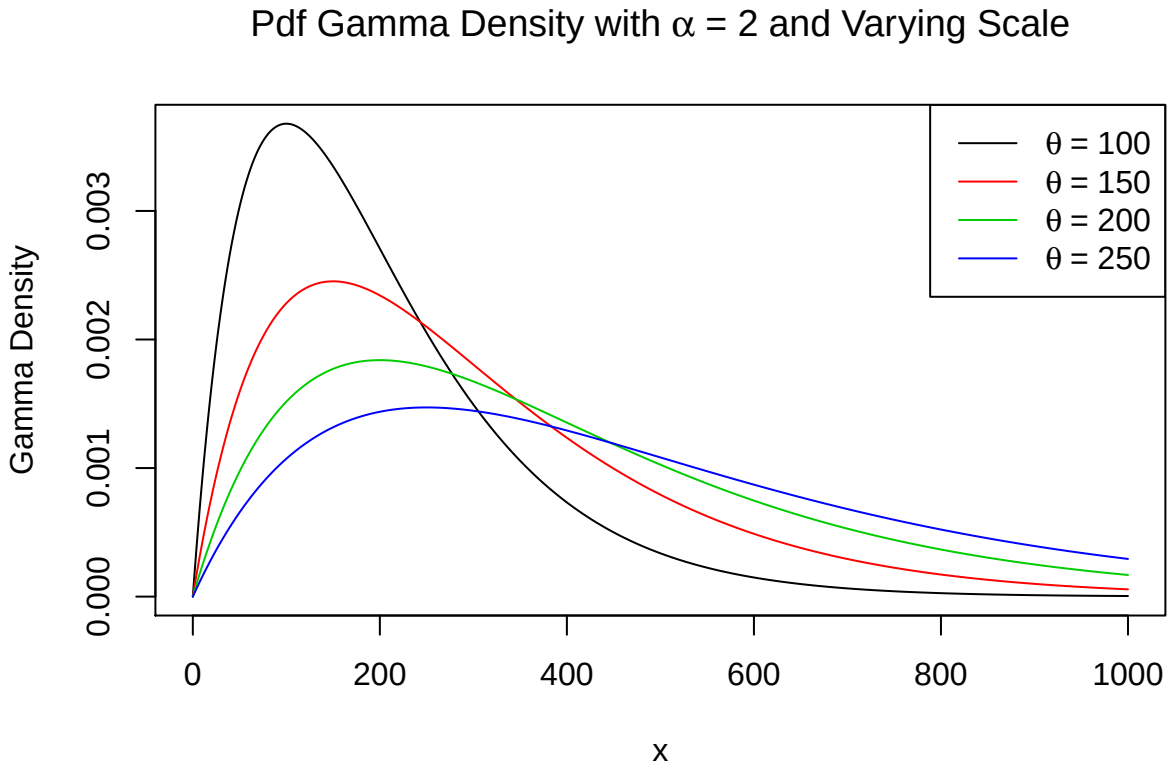
- `seq` is short-hand for sequence
- `dgamma` function is used for density of the Gamma distribution with shape and scale parameters .
- `plot` is a very handy command for displaying results graphically.
- The `lines()` function is used to add plots to an already existing graph.
- The `legend` function can be used to add legends to plots.

3.2.2 Varying The Scale Parameter

The graph shows the gamma density functions with varying scale parameters (θ)

```
plot(x, dgamma(x, shape = 2, scale = scale_param[1]), type = "l", ylab = "Gamma Density")

for (k in 2:length(scale_param)) {
  lines(x, dgamma(x, shape = 2, scale = scale_param[k]), col = k)
}

legend("topright", c(expression(theta ~ '= 100'), expression(theta ~ '= 150'),
                     expression(theta ~'= 200'), expression(theta ~ '= 250')),
       lty = 1, col = 1:4)
title(substitute(paste("Pdf Gamma Density with"," ", alpha, " = 2", " ",
```

```
                    "and Varying Scale")))
```



3.3 Pareto Distribution

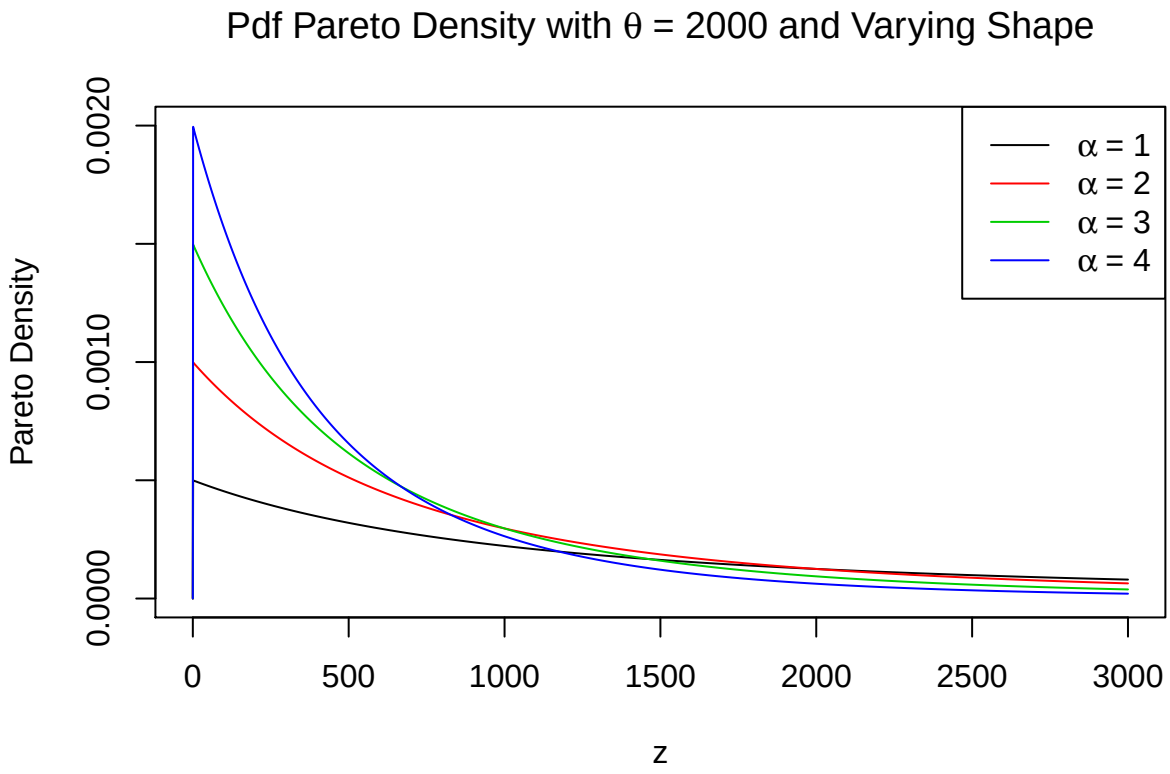
This section demonstrates the effect of the shape and scale parameters on the Pareto density function.

3.3.1 Varying The Shape Parameter

The graph shows the Pareto density functions with varying shape parameters (α)

```
z <- seq(0, 3000, by = 1)

scale_param <- seq(2000, 3500, 500)
shape_param <- 1:4

# Varying the shape parameter
plot(z, dparetoII(z, loc = 0, shape = shape_param[1], scale = 2000),
     ylim = c(0, 0.002),type = "l", ylab = "Pareto Density")

for (k in 2:length(shape_param)) {
  lines(z, dparetoII(z,loc = 0, shape = shape_param[k], scale = 2000), col = k)
}
legend("topright", c(expression(alpha ~ '= 1'), expression(alpha ~ '= 2'),
```

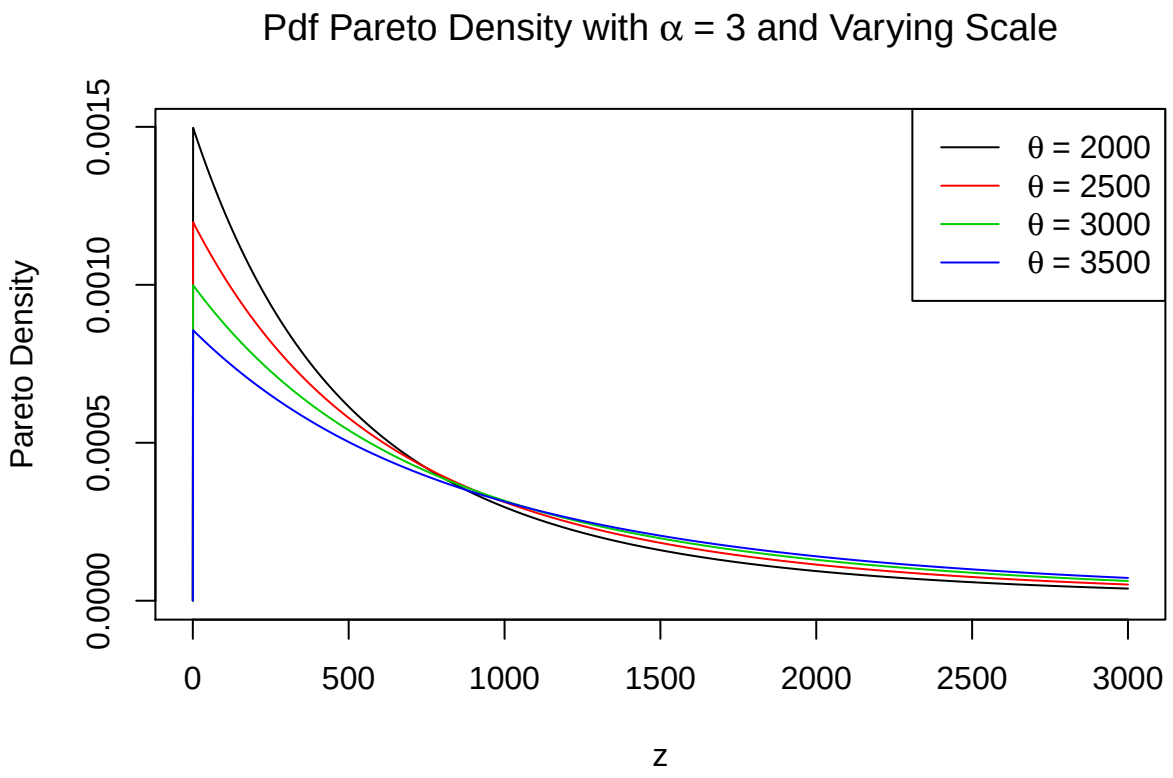
```
                    expression(alpha ~ '= 3'), expression(alpha ~ '= 4')),
       lty = 1, col = 1:4)
title(substitute(paste("Pdf Pareto Density with"," ",theta," = 2000"," ",
                       "and Varying Shape")))
```



3.3.2 Varying The Scale Parameter

The graph shows the Pareto density functions with varying scale parameters (θ)

```
plot(z, dparetoII(z, loc = 0, shape = 3, scale = scale_param[1]),
     type = "l", ylab = "Pareto Density")

for (k in 2:length(scale_param)) {
  lines(z, dparetoII(z, loc = 0, shape = 3, scale = scale_param[k]), col = k)
}

legend("topright", c(expression(theta ~ '= 2000'), expression(theta ~ '= 2500'),
                     expression(theta ~ '= 3000'), expression(theta ~ '= 3500')),
       lty = 1, col = 1:4)
title(substitute(paste("Pdf Pareto Density with"," ",alpha," = 3"," ",
                       "and Varying Scale")))
```



3.4 Weibull Distribution

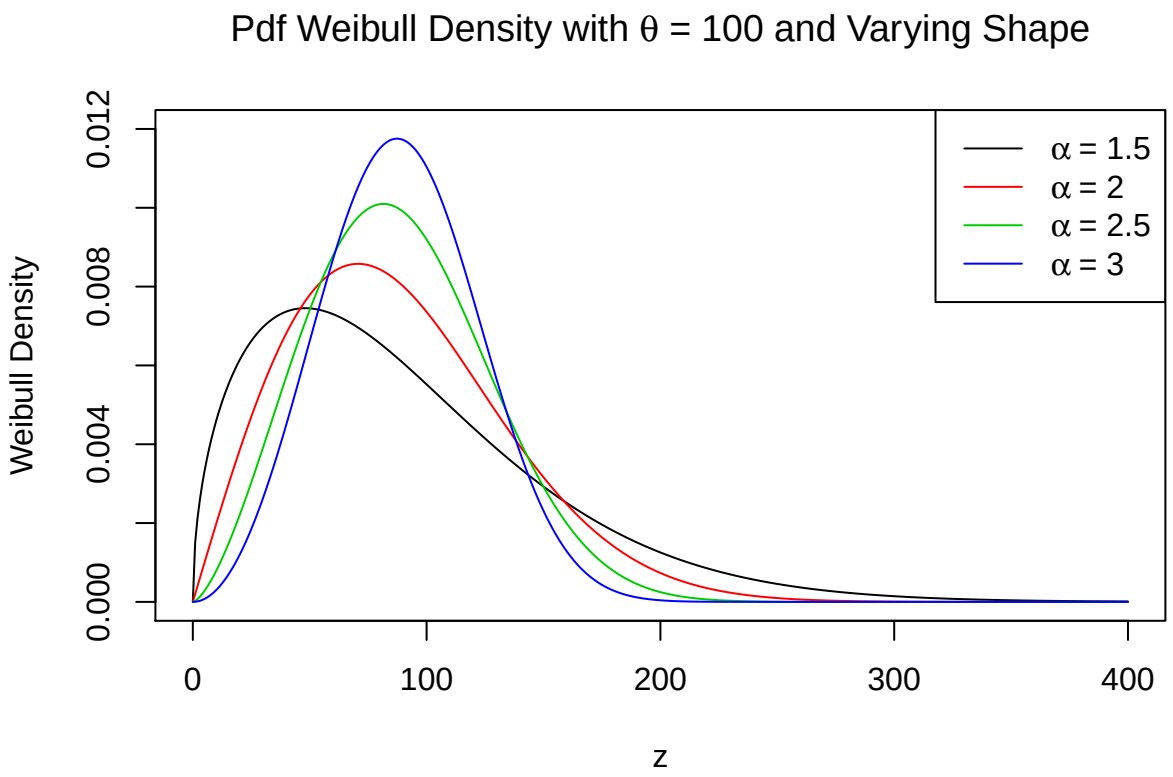
This section demonstrates the effect of the shape and scale parameters on the Weibull density function.

3.4.1 Varying The Shape Parameter

The graph shows the Weibull density function with varying shape parameters (α)

```
z <- seq(0, 400, by = 1)

scale_param <- seq(50, 200, 50)
shape_param <- seq(1.5, 3, 0.5)

# Varying the shape parameter
plot(z, dweibull(z, shape = shape_param[1], scale = 100),
     ylim = c(0, 0.012), type = "l", ylab = "Weibull Density")

for (k in 2:length(shape_param)) {
  lines(z, dweibull(z, shape = shape_param[k], scale = 100), col = k)
}

legend("topright",  c(expression(alpha ~ '= 1.5'), expression(alpha ~ '= 2'),
                      expression(alpha ~'= 2.5'), expression(alpha ~ '= 3')),
       lty = 1, col = 1:4)
```

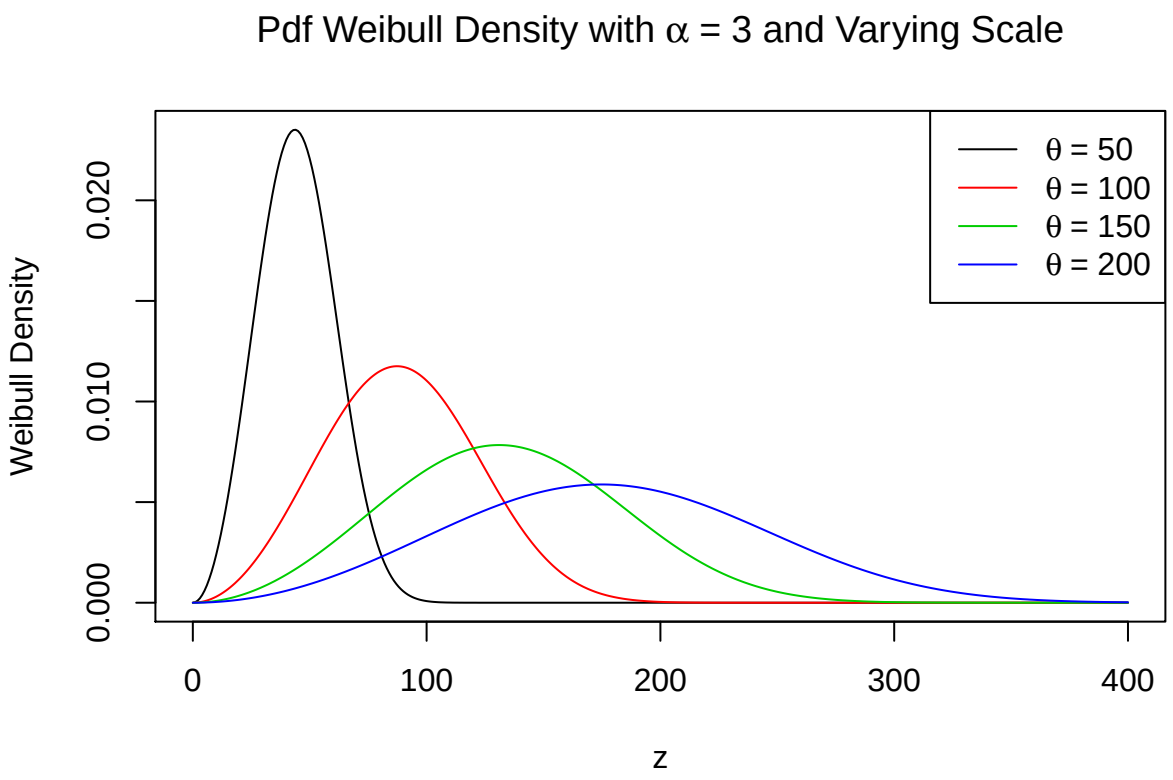
```
title(substitute(paste("Pdf Weibull Density with"," ",theta," = 100"," ",
                       "and Varying Shape")))
```



3.4.2 Varying The Scale Parameter

The graph shows the Weibull density function with varying scale parameters (θ)

```
plot(z, dweibull(z, shape = 3, scale = scale_param[1]),
     type = "l", ylab = "Weibull Density")

for(k in 2:length(scale_param)){
  lines(z,dweibull(z, shape = 3, scale = scale_param[k]), col = k)
}
legend("topright", c(expression(theta ~ '= 50'), expression(theta ~ '= 100'),
                     expression(theta ~ '= 150'), expression(theta ~ '= 200')),
       lty = 1, col = 1:4)
title(substitute(paste("Pdf Weibull Density with", " ", alpha, " = 3", " ",
                       "and Varying Scale")))
```



3.5 Generalized Beta Distribution of The Second Kind (GB2)

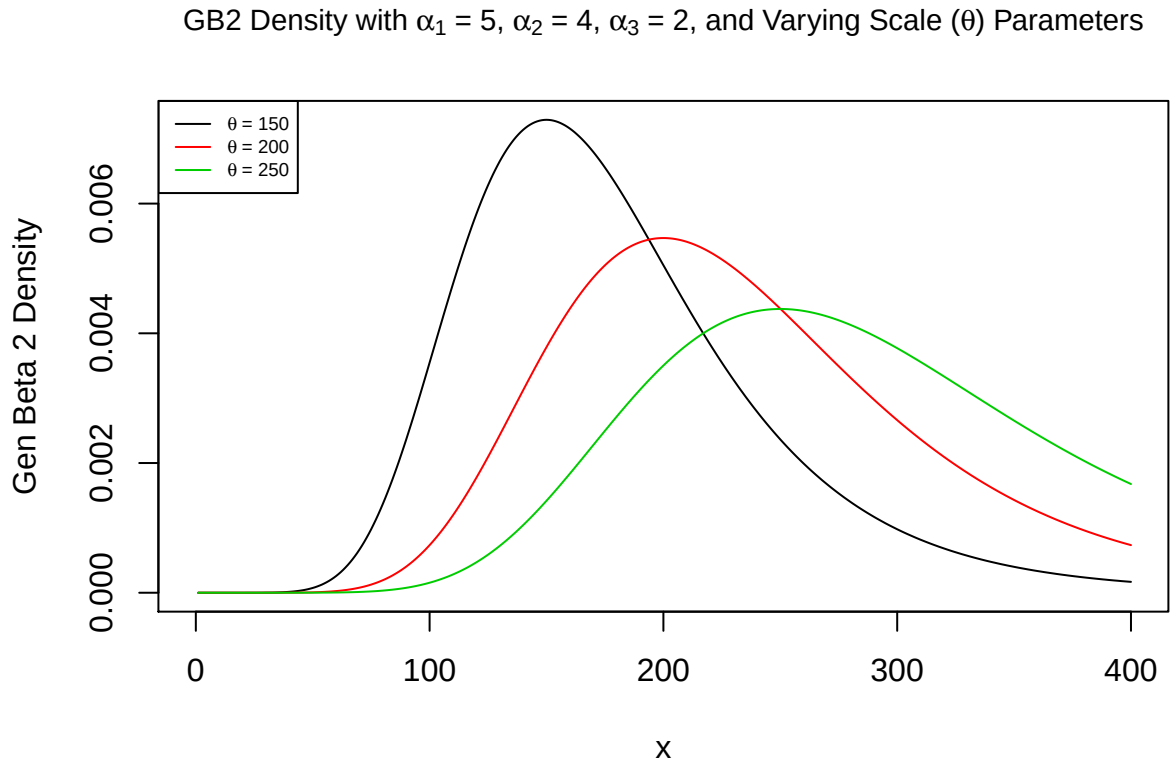
This section demonstrates the effect of the shape and scale parameters on the GB2 density function.

3.5.1 Varying The Scale Parameter

The graph shows the GB2 density function with varying scale parameter (θ)

```
# Example 4:GB2
gb2_density <- function (x, shape_1, shape_2, shape_3, scale){
  mu <- log(scale)
  sigma <- 1 / shape_3
  xt <- (log(x) - mu) / sigma
  logexpxt <- ifelse (xt > 23, yt, log(1 + exp(xt)))
  logdens <- shape_1 * xt - log(sigma) - log(beta(shape_1, shape_2)) -
    (shape_1 + shape_2) * logexpxt - log(x)
  exp(logdens)
}
x <- seq(0, 400, by = 1)

alpha_1 <- 5
alpha_2 <- 4
gamma <- 2
theta <- seq(150, 250, 50)
```

```r
# Varying the scale parameter
plot(x,
     gb2_density(x, shape_1 = alpha_1, shape_2 = alpha_2, shape_3 = gamma,
                 scale = theta[1]), type = "l", ylab = "Gen Beta 2 Density",
     main = expression(paste("GB2 Density with ", alpha[1], " = 5, ",
                             alpha[2], " = 4, ", alpha[3], " = 2, ",
                             "and Varying Scale (", theta, ") Parameters")),
     cex.main = 0.95 )

for(k in 2:length(theta)){
  lines(x, gb2_density(x, shape_1 = alpha_1, shape_2 = alpha_2, shape_3 = gamma,
                       scale = theta[k]), col = k)
}
legend("topleft", c(expression(theta ~ '= 150'), expression(theta ~ '= 200'),
                    expression(theta ~ '= 250')), lty = 1, cex = 0.6, col = 1:3)
```



Note: Here we wrote our own function for the density function of the GB2 density function.

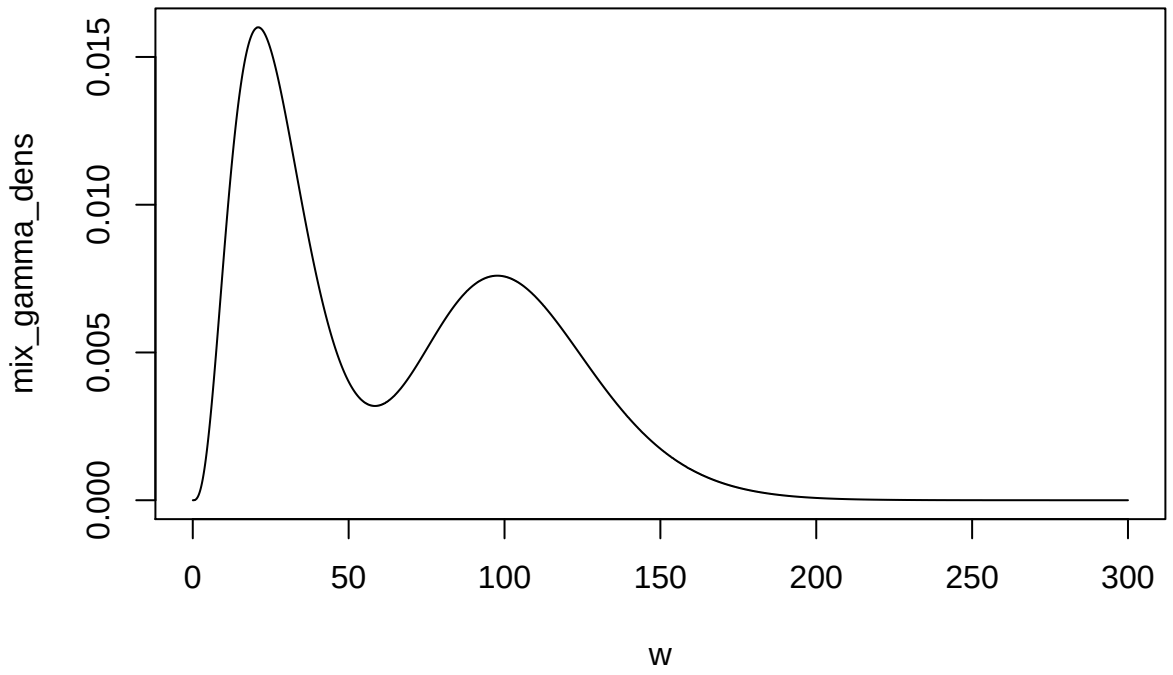
3.6 Methods of Creating New Distributions

This section shows some of the methods of creating new distributions.

3.6.1 Mixture Distributions

The graph below creates a density function from two random variables that follow a gamma distribution.

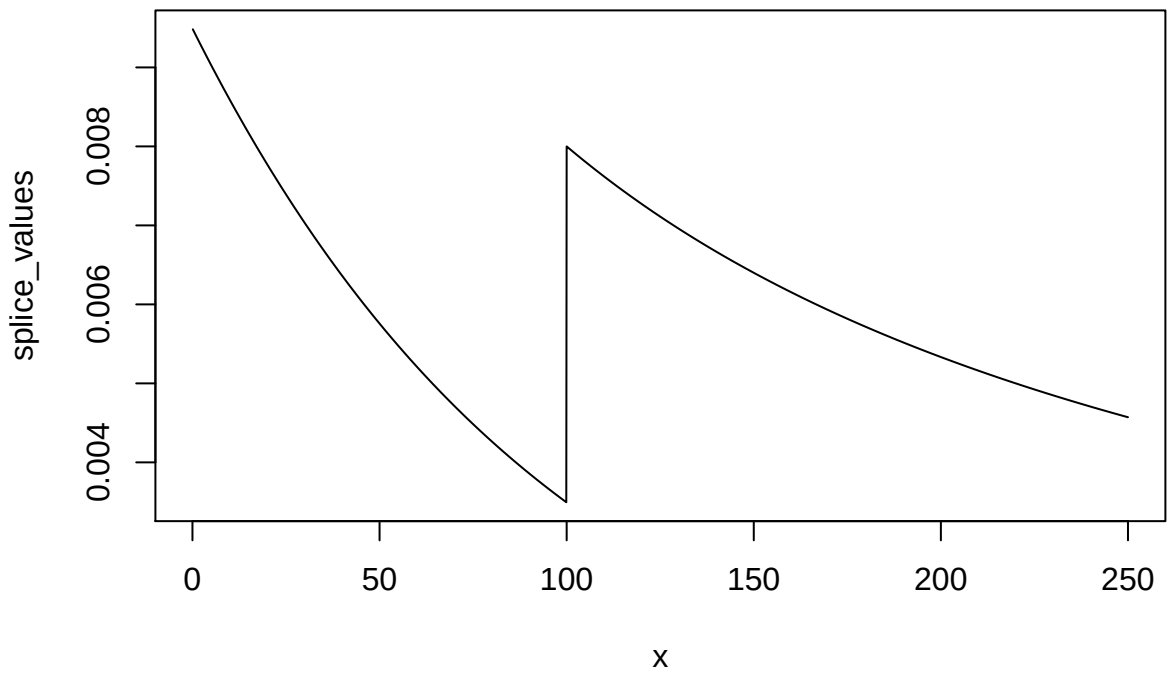
```
# Example 5: A mixed density
# Specify density of a mixture of 2 gamma distributions
mixture_gamma_density <- function (x, a_1, a_2, alpha_gamma1, theta_gamma1, alpha_gamma2,
                                   theta_gamma2){
  a_1 * dgamma(x, shape = alpha_gamma1, scale = theta_gamma1) + a_2 *
    dgamma(x, shape = alpha_gamma2, scale = theta_gamma2)
}

w <- 1:30000 / 100
a_1 <- 0.5
a_2 <- 0.5
alpha_1 <- 4
theta_1 <- 7
alpha_2 <- 15
theta_2 <- 7

mix_gamma_dens <- mixture_gamma_density(w, a_1, a_2, alpha_1, theta_1, alpha_2, theta_2)

plot(w, mix_gamma_dens, type = "l")
```



3.6.2 Density Obtained Through Splicing

The graph below shows a density function through splicing by combining an exponential distribution on $(0, c)$ with a Pareto distribution on (c, ∞)

```
# Example 6: density obtained through splicing
# Combine an Exp on (0,c) with a Pareto on (c,\infty)

splice_exp_par <- function (x, c, v, theta, gamma, alpha){
  if (0 <= x & x < c) {return(v * dexp(x, 1 / theta) / pexp(c, 1 / theta))} else
    if (x >= c) {return((1 - v) * dparetoII(x, loc = 0, shape = alpha, scale = theta) /
                          (1 - pparetoII(x, loc = 0, shape = alpha, scale = theta)))}
}

x <- t(as.matrix(1:2500 / 10))

splice_values <- apply(x, 2, splice_exp_par, c = 100, v = 0.6,
                       theta = 100, gamma = 200, alpha = 4)

plot(x, splice_values, type = 'l')
```



3.7 Coverage Modifications

3.7.1 Load Required Package

The `actuar` package provides functions for dealing with coverage modifications. In the following sections we will check the functionalities of the `coverage` command.

```
library(actuar)
```

3.7.2 Ordinary Deductible

This section plots the modified probability density functions due to deductibles for the payment per loss and payment per payment random variables.

3.7.2.1 Payment Per Loss with Ordinary Deductible

Let X be the random variable for loss size. The random variable for the payment per loss with deductible d is $Y^L = (X - d)_+$. The plot of the modified probability density function is below.

```
f <- coverage(dgamma, pgamma, deductible = 1, per.loss = TRUE)  # create the object
mode(f)  # it's a function. Here deductible is 1
```

```
[1] "function"
```

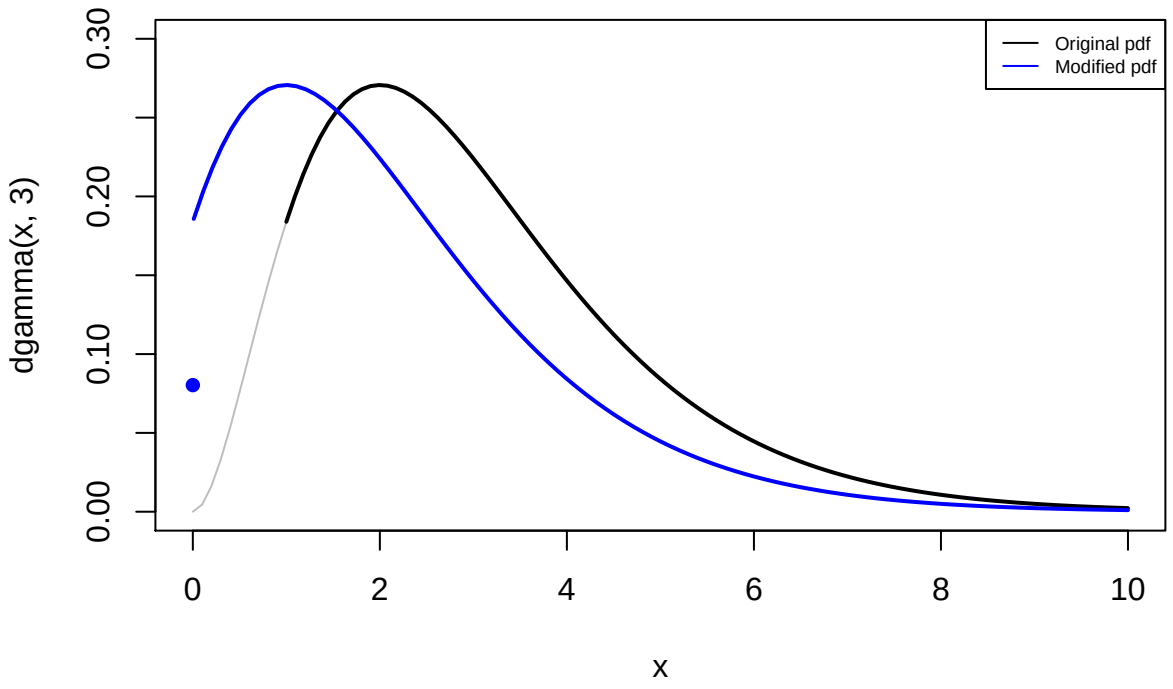
```
# Check the pdf for Y^L at 0 and the original loss at 1
f(0, 3)  # mass at 0
```

```
[1] 0.0803014
```

```
pgamma(0 + 1, 3)  # idem
```

```
[1] 0.0803014
```

```
curve(dgamma(x, 3), from = 0, to = 10, ylim = c(0, 0.3), lwd = 1, col = "gray")  # original
curve(dgamma(x, 3), from = 1, to = 10, ylim = c(0, 0.3), lwd = 2, add = TRUE)
curve(f(x, 3), from = 0.01, col = "blue", add = TRUE, lwd = 2)  # modified
points(0, f(0, 3), pch = 16, col = "blue")
legend("topright", c("Original pdf", "Modified pdf"),
       lty = 1, cex = 0.6, col = c("black","blue"))
```



A few quick notes on these commands:

- The `coverage()` function computes probability density function or cumulative distribution function of the payment per payment or payment per loss random variable under any combination of the following coverage modifications: deductible, limit, coinsurance, inflation. In this illustration we used it to compute the probability density function of the payment per loss random variable with a deductible of 1.
- The `f(0, 3)` function calculates the pdf when the payment per loss variable is 0 with gamma parameters shape=3 and rate=1. Because we used a deductible of 1 , this should be equal to `pgamma(0 + 1, 3)`.

3.7.2.2 Payment Per Payment with Ordinary Deductible

Y^P with pdf $f_{Y^P}(y) = f_X(y+d)/S_X(d)$

```
f <- coverage(dgamma, pgamma, deductible = 1)  # create the object

f(0, 3)  # calculate in x = 0, shape = 3, rate = 1
```

```
[1] 0
```

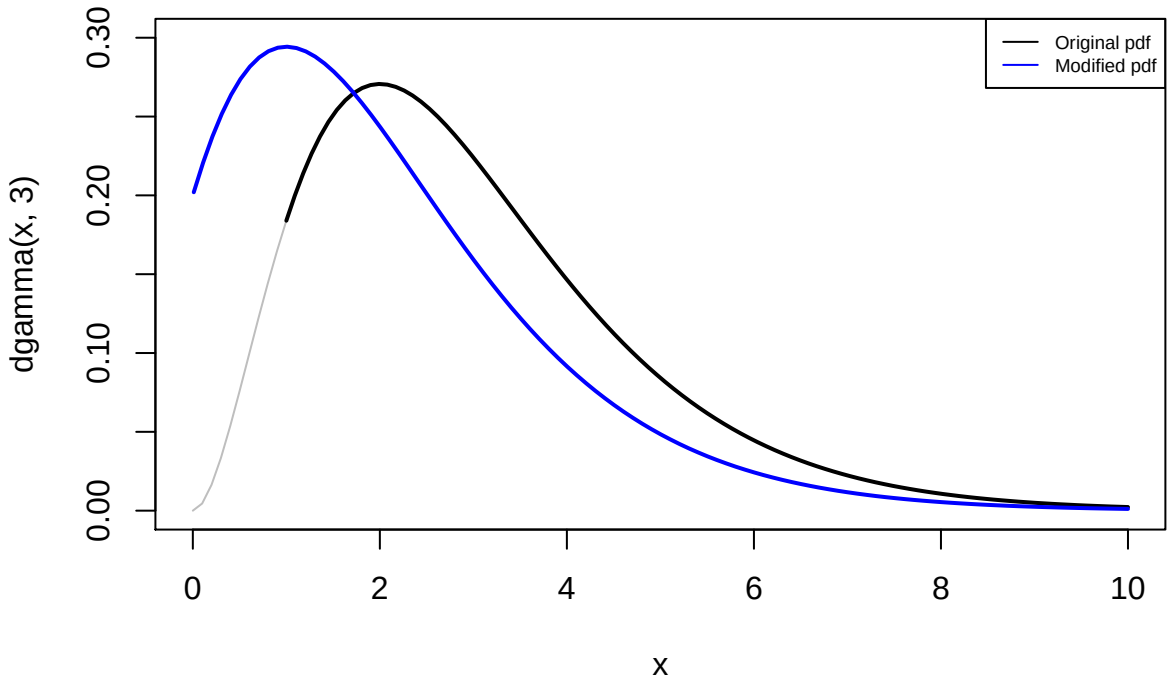
```
f(5, 3)  # calculate in x = 5, shape = 3, rate = 1
```

```
[1] 0.04851322
```

```
dgamma(5 + 1, 3) / pgamma(1, 3, lower = FALSE)  # DIY
```

```
[1] 0.04851322
```

```
curve(dgamma(x, 3), from = 0, to = 10, ylim = c(0, 0.3),
      lwd = 1, col = "gray")  # original pdf
curve(dgamma(x, 3), from = 1, to = 10, ylim = c(0, 0.3), add = TRUE, lwd = 2)
curve(f(x, 3), from = 0.01, col = "blue",
      add = TRUE, lwd = 2)     # modified pdf
legend("topright", c("Original pdf", "Modified pdf"),
       lty = 1, cex = 0.6, col = c("black","blue"))
```



3.7.2.3 Per Payment Variable with Policy Limit, Coinsurance and Inflation

```
f <- coverage(dgamma, pgamma, deductible = 1, limit = 100,
              coinsurance = 0.9, inflation = 0.05)
         # create the object

f(0, 3)  # calculate in x = 0, shape = 3, rate = 1
```

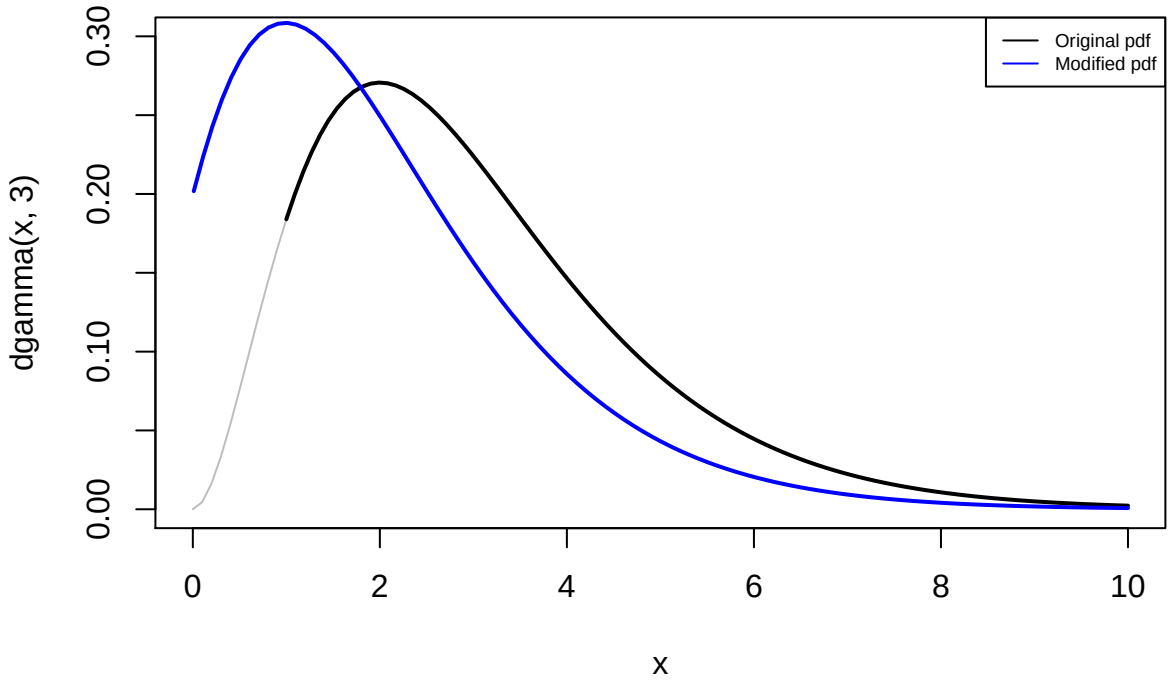
```
[1] 0
```

```
f(5, 3)  # calculate in x = 5, shape = 3, rate = 1
```

```
[1] 0.0431765
```

```
curve(dgamma(x, 3), from = 0, to = 10, ylim = c(0, 0.3),
      lwd = 1, col = "gray")  # original pdf
curve(dgamma(x, 3), from = 1, to = 10, ylim = c(0, 0.3), add = TRUE, lwd = 2)
curve(f(x, 3), from = 0.01, col = "blue", add = TRUE, lwd = 2)  # modified pdf
```

```
legend("topright", c("Original pdf", "Modified pdf"),
       lty = 1, cex = 0.6, col = c("black", "blue"))
```



3.7.3 Franchise Deductible

A policy with a *franchise deductible* of d pays nothing if the loss is no greater than d, and pays the full amount of the loss if it is greater than d. This section plots the pdf for the per payment and per loss random variable.

3.7.3.1 Payment Per Loss with Franchise Deductible

```
# Franchise deductible
# Per loss variable
f <- coverage(dgamma, pgamma, deductible = 1, per.loss = TRUE, franchise = TRUE)
f(0, 3)  # mass at 0
```

```
[1] 0.0803014
```

```
pgamma(1, 3)  # idem
```

```
[1] 0.0803014
```

```
f(0.5, 3)  # 0 < x < 1
```

```
[1] 0
```

```r
f(1, 3)  # x = 1
```

```
[1] 0
```

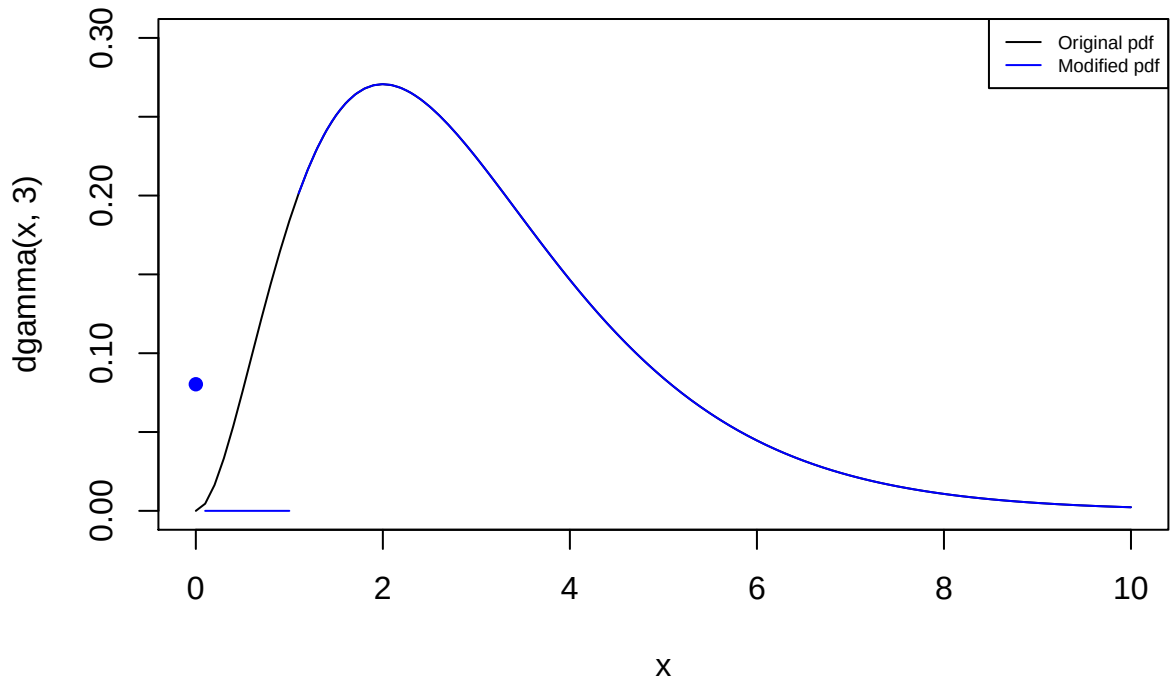
```r
f(5, 3)  # x > 1
```

```
[1] 0.08422434
```

```r
dgamma(5, 3)
```

```
[1] 0.08422434
```

```r
curve(dgamma(x, 3), from = 0, to = 10, ylim = c(0, 0.3))  # original
curve(f(x, 3), from = 1.1, col = "blue", add = TRUE)  # modified
points(0, f(0, 3), pch = 16, col = "blue")  # mass at 0
curve(f(x, 3), from = 0.1, to = 1, col = "blue", add = TRUE)  # 0 < x < 1
legend("topright", c("Original pdf", "Modified pdf"),
       lty = 1, cex = 0.6, col = c("black", "blue"))
```



Note : To use the franchise deductible , we have to add the option `franchise = TRUE` in the coverage function.

3.7.3.2 Payment Per Payment with Franchise Deductible

```r
# Franchise deductible
# Per payment variable
```

```
f <- coverage(dgamma, pgamma, deductible = 1, franchise = TRUE)
f(0, 3)  # x = 0
```

```
[1] 0
```

```
f(0.5, 3)  # 0 < x < 1
```

```
[1] 0
```

```
f(1, 3)  # x = 1
```

```
[1] 0
```

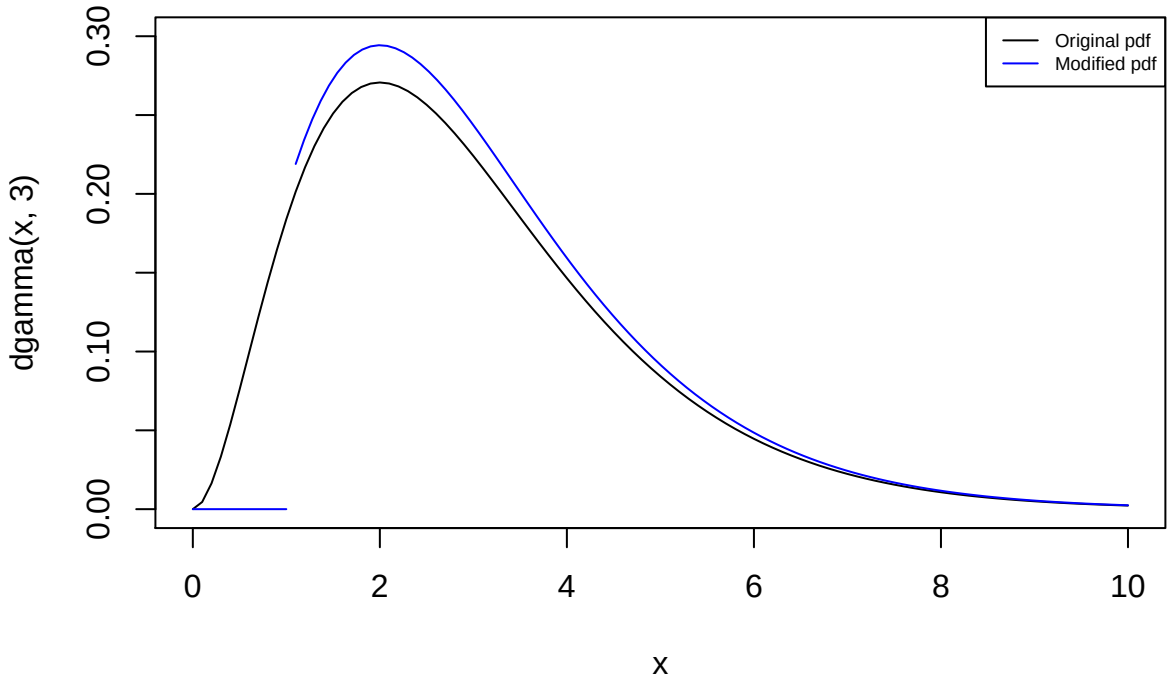
```
f(5, 3)  # x > 1
```

```
[1] 0.09157819
```

```
dgamma(5, 3) / pgamma(1, 3, lower = FALSE)  # idem
```

```
[1] 0.09157819
```

```
curve(dgamma(x, 3), from = 0, to = 10, ylim = c(0, 0.3))  # original
curve(f(x, 3), from = 1.1, col = "blue", add = TRUE)  # modified
curve(f(x, 3), from = 0, to = 1, col = "blue", add = TRUE)  # 0 < x < 1
legend("topright", c("Original pdf", "Modified pdf"),
       lty = 1, cex = 0.6, col = c("black", "blue"))
```



Chapter 4

Model Selection

*This file contains illustrative **R** code for computing important count distributions. When reviewing this code, you should open an **R** session, copy-and-paste the code, and see it perform. Then, you will be able to change parameters, look up commands, and so forth, as you go.This code uses the dataset `CLAIMLEVEL.csv`*

4.1 Claim Level Data of Property Fund

This section summarizes claims from the property fund for year 2010 and plots the data.

4.1.1 Claims Data

The results below considers individual claims from the property fund for year 2010.

```
## Read in data and get number of claims.
claim_lev <- read.csv("Data/CLAIMLEVEL.csv", header = TRUE)
nrow(claim_lev)  # 6258
```

```
[1] 6258
```

```
# 2010 subset
claim_data <- subset(claim_lev, Year == 2010);
length(unique(claim_data$PolicyNum))  # 403 unique policyholders
```

```
[1] 403
```

```
n_tot <- nrow(claim_data)  # 1377 individual claims
n_tot
```

```
[1] 1377
```

```
# As an alternative, you can simulate claims
# n_tot <- 13770
# alpha_hat <- 2
# theta_hat <- 100
# claim <- rgamma(n_tot, shape = alpha_hat, scale = theta_hat)
# claim <- rparetoII(n_tot, loc = 0,  shape = alpha_hat, scale = theta_hat)
# GB2
# claim <- theta_hat * rgamma(n_tot, shape = alpha_hat, scale = 1) /
#          rgamma(n_tot, shape = 1, scale = 1)
```

```
# claim_data <- data.frame(claim)

#####################################################
```

4.1.2 Summary of Claims

The output below provides summary on claims data for 2010 and summary in logarithmic units.

```
# Summarizing the claim data for 2010
summary(claim_data$Claim)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
       1      789     2250    26623     6171 12922218
```

```
sd(claim_data$Claim)
```

```
[1] 368029.7
```

```
# Summarizing logarithmic claims for 2010
summary(log(claim_data$Claim))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   6.670   7.719   7.804   8.728  16.374
```

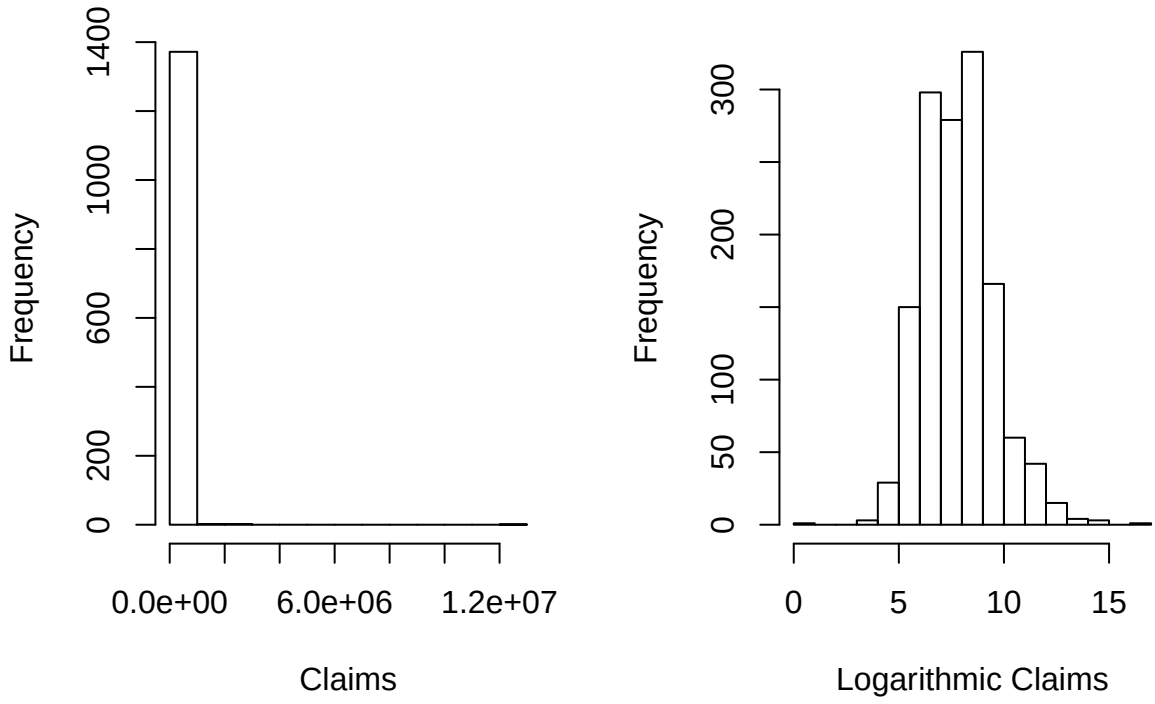
```
sd(log(claim_data$Claim))
```

```
[1] 1.683297
```

4.1.3 Plot of Claims

The plots below provides further information about the distribution of sample claims.

```
# Histogram
par(mfrow = c(1, 2))
hist(claim_data$Claim, main="", xlab = "Claims")
hist(log(claim_data$Claim), main = "", xlab = "Logarithmic Claims")
```




```
# dev.off()
```

4.2 Fitting Distributions

This section shows how to fit basic distributions to a data set.

4.2.1 Inference Assuming a Lognormal Distribution

The results below assume that the data follow a lognormal distribution and usesVGAM library for estimation of parameters.

```
# Inference assuming a lognormal distribution
# First, take the log of the data and assume normality
y <- log(claim_data$Claim)
summary(y)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   6.670   7.719   7.804   8.728  16.374
```

```
sd(y)
```

```
[1] 1.683297
```

```
# Confidence intervals and hypothesis test
t.test(y, mu = log(5000))  # H0: mu_o = log(5000) = 8.517
```

```
    One Sample t-test

data:  y
t = -15.717, df = 1376, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 8.517193
95 percent confidence interval:
 7.715235 7.893208
sample estimates:
mean of x
 7.804222
```

```
# Mean of the lognormal distribution
exp(mean(y) + sd(y)^2 / 2)
```

```
[1] 10106.82
```

```
mean(claim_data$Claim)
```

```
[1] 26622.59
```

```
# Alternatively, assume that the data follow a lognormal distribution
# Use "VGAM" library for estimation of parameters
library(VGAM)
fit.LN <- vglm(Claim ~ 1, family = lognormal, data = claim_data)
summary(fit.LN)
```

```

Call:
vglm(formula = Claim ~ 1, family = lognormal, data = claim_data)


Pearson residuals:
                Min      1Q   Median     3Q    Max
meanlog     -4.6380 -0.6740 -0.05083 0.5487  5.093
loge(sdlog) -0.7071 -0.6472 -0.44003 0.1135 17.636

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept):1  7.80422    0.04535  172.10   <2e-16 ***
(Intercept):2  0.52039    0.01906   27.31   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of linear predictors:  2

Names of linear predictors: meanlog, loge(sdlog)

Log-likelihood: -13416.87 on 2752 degrees of freedom

Number of iterations: 3

No Hauck-Donner effect found in any of the estimates
```

```
coef(fit.LN)                    # coefficients
```

```
(Intercept):1 (Intercept):2
```

```
    7.8042218     0.5203908
```

```
confint(fit.LN, level = 0.95)  # confidence intervals for model parameters
```

```
                  2.5 %    97.5 %
(Intercept):1 7.7153457 7.8930978
(Intercept):2 0.4830429 0.5577387
```

```
logLik(fit.LN)                 # loglikelihood for lognormal
```

```
[1] -13416.87
```

```
AIC(fit.LN)                    # AIC for lognormal
```

```
[1] 26837.74
```

```
BIC(fit.LN)                    # BIC for lognormal
```

```
[1] 26848.2
```

```
vcov(fit.LN)                   # covariance matrix for model parameters
```

```
              (Intercept):1 (Intercept):2
(Intercept):1   0.002056237  0.0000000000
(Intercept):2   0.000000000  0.0003631082
```

```
# Mean of the lognormal distribution
exp(mean(y) + sd(y)^2 / 2)
```

```
[1] 10106.82
```

```
exp(coef(fit.LN))
```

```
(Intercept):1 (Intercept):2
  2450.927448      1.682685
```

A few quick notes on these commands:

- The `t.test()` function can be used for a variety of t-tests. In this illustration, it was used to test $H_0 = \mu_0 = \log(5000) = 8.517$.
- The `vglm()` function is used to fit vector generalized linear models (VGLMs). See `help(vglm)` for other modeling options.
- The `coef()` function returns the estimated coefficients from the `vglm` or other modeling functions.
- The `confint` function provides the confidence intervals for model parameters.
- The `loglik` function provides the log-likelihood value for the lognormal estimation from the `vglm` or other modeling functions.
- `AIC()` and `BIC()` returns Akaike's Information Criterion and BIC or SBC (Schwarz's Bayesian criterion) for the fitted lognormal model. AIC $= -2*(\text{loglikelihood}) + 2*\text{npar}$, where `npar` represents the number of parameters in the fitted model, and BIC $= -2 * \text{log-likelihood} + \log(n) * \text{npar}$ where n is the number of observations.
- `vcov()` returns the covariance matrix for model parameters.

4.2.2 Inference Assuming a Gamma Distribution

The results below assume that the data follow a gamma distribution and uses`VGAM` library for estimation of parameters.

```
# Inference assuming a gamma distribution
# Install.packages("VGAM")
library(VGAM)
```

```r
fit.gamma <- vglm(Claim ~ 1, family = gamma2, data = claim_data)
summary(fit.gamma)
```

```
Call:
vglm(formula = Claim ~ 1, family = gamma2, data = claim_data)


Pearson residuals:
                 Min      1Q  Median      3Q     Max
loge(mu)      -0.539 -0.5231 -0.4935 -0.4141 261.117
loge(shape) -153.990 -0.1024  0.2335  0.4969   0.772

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept):1 10.18952    0.04999  203.82   <2e-16 ***
(Intercept):2 -1.23582    0.03001  -41.17   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of linear predictors:  2

Names of linear predictors: loge(mu), loge(shape)

Log-likelihood: -14150.59 on 2752 degrees of freedom

Number of iterations: 13

No Hauck-Donner effect found in any of the estimates
```

```r
coef(fit.gamma)                 # This uses a different parameterization
```

```
(Intercept):1 (Intercept):2
    10.189515     -1.235822
```

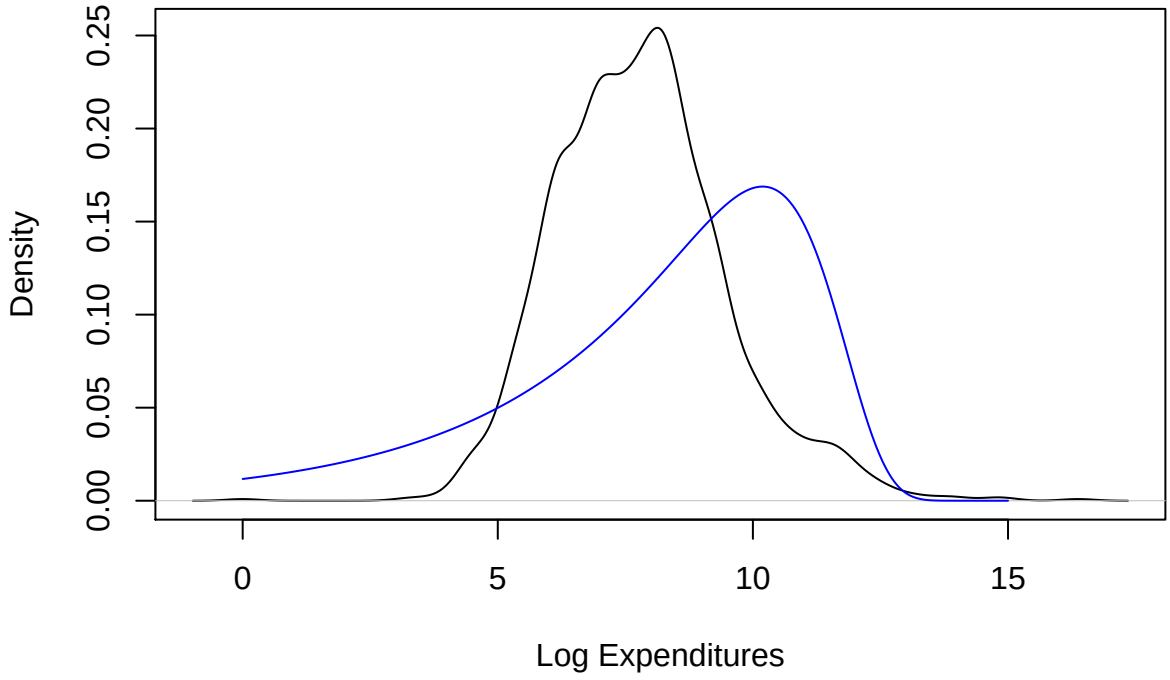
```r
( theta <- exp(coef(fit.gamma)[1]) / exp(coef(fit.gamma)[2]) )  # theta = mu / alpha
```

```
(Intercept):1
     91613.78
```

```r
( alpha <- exp(coef(fit.gamma)[2]) )
```

```
(Intercept):2
    0.2905959
```

```r
plot(density(log(claim_data$Claim)), main = "", xlab = "Log Expenditures")
x <- seq(0, 15, by = 0.01)
fgamma_ex <- dgamma(exp(x), shape = alpha, scale = theta) * exp(x)
lines(x, fgamma_ex, col = "blue")
```




```
confint(fit.gamma, level = 0.95)  # confidence intervals for model parameters
```

```
                   2.5 %    97.5 %
(Intercept):1 10.091533 10.287498
(Intercept):2 -1.294648 -1.176995
```

```
logLik(fit.gamma)                 # loglikelihood for gamma
```

```
[1] -14150.59
```

```
AIC(fit.gamma)                    # AIC for gamma
```

```
[1] 28305.17
```

```
BIC(fit.gamma)                    # BIC for gamma
```

```
[1] 28315.63
```

```
vcov(fit.gamma)                   # covariance matrix for model parameters
```

```
              (Intercept):1 (Intercept):2
(Intercept):1   0.002499196  0.0000000000
(Intercept):2   0.000000000  0.0009008397
```

```
# Here is a check on the formulas
# AIC using formula : -2 * (loglik) + 2 * (number of parameters)
-2 * (logLik(fit.gamma)) + 2 * (length(coef(fit.gamma)))
```

```
[1] 28305.17
```

```
# BIC using formula : -2 * (loglik) + (number of parameters) * (log(n))
-2 * (logLik(fit.gamma)) + length(coef(fit.gamma, matrix = TRUE)) * log(nrow(claim_data))
```

```
[1] 28315.63
```

```
# Alternatively, we could a gamma distribution using glm
library(MASS)
fit.gamma_2 <- glm(Claim ~ 1, data = claim_data, family = Gamma(link = log))
summary(fit.gamma_2, dispersion = gamma.dispersion(fit.gamma_2))
```

```

Call:
glm(formula = Claim ~ 1, family = Gamma(link = log), data = claim_data)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-4.287  -2.258  -1.764  -1.178  30.926

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 10.18952    0.04999   203.8   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Gamma family taken to be 3.441204)

    Null deviance: 6569.1  on 1376  degrees of freedom
Residual deviance: 6569.1  on 1376  degrees of freedom
AIC: 28414

Number of Fisher Scoring iterations: 14
```

```
( theta <- exp(coef(fit.gamma_2)) * gamma.dispersion(fit.gamma_2) )  #theta = mu / alpha
```

```
(Intercept)
   91613.78
```

```
( alpha <- 1 / gamma.dispersion(fit.gamma_2) )
```

```
[1] 0.2905959
```

```
logLik(fit.gamma_2)  # log - likelihood slightly different from vglm
```

```
'log Lik.' -14204.77 (df=2)
```

```
AIC(fit.gamma_2)     # AIC
```

```
[1] 28413.53
```

```
BIC(fit.gamma_2)     # BIC
```

```
[1] 28423.99
```

Note : The output from `coef(fit.gamma)` uses the parameterization $\mu = \theta * \alpha$. `coef(fit.gamma)[1]` $= \log(\mu)$ and `coef(fit.gamma)[2]` $= \log(\alpha)$, which implies , $\alpha =$ `exp(coef(fit.gamma)[2])` and $\theta = \mu/\alpha =$ `exp(coef(fit.gamma)[1]) / exp(coef(fit.gamma)[2])`.

4.2.3 Inference Assuming a Pareto Distribution

The results below assume that the data follow a Pareto distribution and usesVGAM library for estimation of parameters.

```
fit.pareto <- vglm(Claim ~ 1, paretoII, loc = 0, data = claim_data)
summary(fit.pareto)
```

```
Call:
vglm(formula = Claim ~ 1, family = paretoII, data = claim_data,
    loc = 0)


Pearson residuals:
                Min      1Q Median     3Q   Max
loge(scale)  -6.332 -0.8289 0.1875 0.8832 1.174
loge(shape) -10.638  0.0946 0.4047 0.4842 0.513

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept):1  7.7329210  0.0933332  82.853   <2e-16 ***
(Intercept):2 -0.0008753  0.0538642  -0.016    0.987
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of linear predictors:  2

Names of linear predictors: loge(scale), loge(shape)

Log-likelihood: -13404.64 on 2752 degrees of freedom

Number of iterations: 5

No Hauck-Donner effect found in any of the estimates
```

```
head(fitted(fit.pareto))
```

```
        [,1]
[1,] 2285.03
[2,] 2285.03
[3,] 2285.03
[4,] 2285.03
[5,] 2285.03
[6,] 2285.03
```

```
coef(fit.pareto)
```

```
(Intercept):1 (Intercept):2
 7.7329210483 -0.0008752515
```

```
exp(coef(fit.pareto))
```

```
(Intercept):1 (Intercept):2
 2282.2590626     0.9991251
```

```
confint(fit.pareto, level = 0.95)  # confidence intervals for model parameters
```

```
                   2.5 %    97.5 %
(Intercept):1  7.5499914 7.9158507
(Intercept):2 -0.1064471 0.1046966
```

```
logLik(fit.pareto)                 # loglikelihood for Pareto
```

```
[1] -13404.64
```

```
AIC(fit.pareto)                    # AIC for Pareto
```

```
[1] 26813.29
```

```
BIC(fit.pareto)                    # BIC for Pareto
```

```
[1] 26823.74
```

```
vcov(fit.pareto)                   # covariance matrix for model parameters
```

```
              (Intercept):1 (Intercept):2
(Intercept):1   0.008711083   0.004352904
(Intercept):2   0.004352904   0.002901350
```

4.2.4 Inference Assuming an Exponential Distribution

The results below assume that the data follow an exponential distribution and uses`VGAM` library for estimation of parameters.

```
fit.exp <- vglm(Claim ~ 1, exponential, data = claim_data)
summary(fit.exp)
```

```
Call:
vglm(formula = Claim ~ 1, family = exponential, data = claim_data)


Pearson residuals:
              Min     1Q Median     3Q Max
loge(rate) -484.4 0.7682 0.9155 0.9704   1

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -10.18952    0.02695  -378.1   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of linear predictors:  1

Name of linear predictor: loge(rate)

Residual deviance: 6569.099 on 1376 degrees of freedom

Log-likelihood: -15407.96 on 1376 degrees of freedom

Number of iterations: 6
```

```
No Hauck-Donner effect found in any of the estimates
```

```
( theta = 1 / exp(coef(fit.exp)) )
```

```
(Intercept)
   26622.59
```

```
# Can also fit using the "glm" package
fit.exp2 <- glm(Claim ~ 1, data = claim_data, family = Gamma(link = log))
summary(fit.exp2, dispersion = 1)
```

```

Call:
glm(formula = Claim ~ 1, family = Gamma(link = log), data = claim_data)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-4.287  -2.258  -1.764  -1.178  30.926

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 10.18952    0.02695   378.1   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Gamma family taken to be 1)

    Null deviance: 6569.1  on 1376  degrees of freedom
Residual deviance: 6569.1  on 1376  degrees of freedom
AIC: 28414

Number of Fisher Scoring iterations: 14
```

```
( theta <- exp(coef(fit.exp2)) )
```

```
(Intercept)
   26622.59
```

4.2.5 Inference Assuming a Generalized Beta Distribution of the Second Kind (GB2) Distribution

The results below assume that the data follow a GB2 distribution and uses the maximum likelihood technique for parameter estimation.

```
# Inference assuming a GB2 Distribution - this is more complicated
# The likelihood functon of GB2 distribution (negative for optimization)
lik_gb2 <- function (param) {
  a_1 <- param[1]
  a_2 <- param[2]
  mu <- param[3]
  sigma <- param[4]
  yt <- (log(claim_data$Claim) - mu) / sigma
  logexpyt <- ifelse(yt > 23, yt, log(1 + exp(yt)))
  logdens <- a_1 * yt - log(sigma) - log(beta(a_1,a_2)) -
    (a_1+a_2) * logexpyt - log(claim_data$Claim)
```

```
  return(-sum(logdens))
}
# "optim" is a general purpose minimization function
gb2_bop <- optim(c(1, 1, 0, 1), lik_gb2, method = c("L-BFGS-B"),
                 lower = c(0.01, 0.01, -500, 0.01),
                 upper = c(500, 500, 500, 500), hessian = TRUE)

# Estimates
gb2_bop$par
```

```
[1] 2.830928 1.202500 6.328981 1.294552
```

```
# Standard error
sqrt(diag(solve(gb2_bop$hessian)))
```

```
[1] 0.9997743 0.2918469 0.3901929 0.2190362
```

```
# t-statistics
( tstat <- gb2_bop$par / sqrt(diag(solve(gb2_bop$hessian))) )
```

```
[1]  2.831567  4.120313 16.220133  5.910217
```

```
# density for GB II
gb2_density <- function(x){
  a_1 <- gb2_bop$par[1]
  a_2 <- gb2_bop$par[2]
  mu <- gb2_bop$par[3]
  sigma <- gb2_bop$par[4]
  xt <- (log(x) - mu) / sigma
  logexpxt<-ifelse (xt > 23, yt, log(1 + exp(xt)))
  logdens <- a_1 * xt - log(sigma) - log(beta(a_1, a_2)) -
    (a_1+a_2) * logexpxt - log(x)
  exp(logdens)
}

# AIC using formula : -2 * (loglik) + 2 * (number of parameters)
-2 * ( sum(log(gb2_density(claim_data$Claim))) ) + 2 * 4
```

```
[1] 26768.13
```

```
# BIC using formula : -2 * (loglik) + (number of parameters) * (log(n))
-2 *( sum(log(gb2_density(claim_data$Claim))) ) + 4 * log(nrow(claim_data))
```

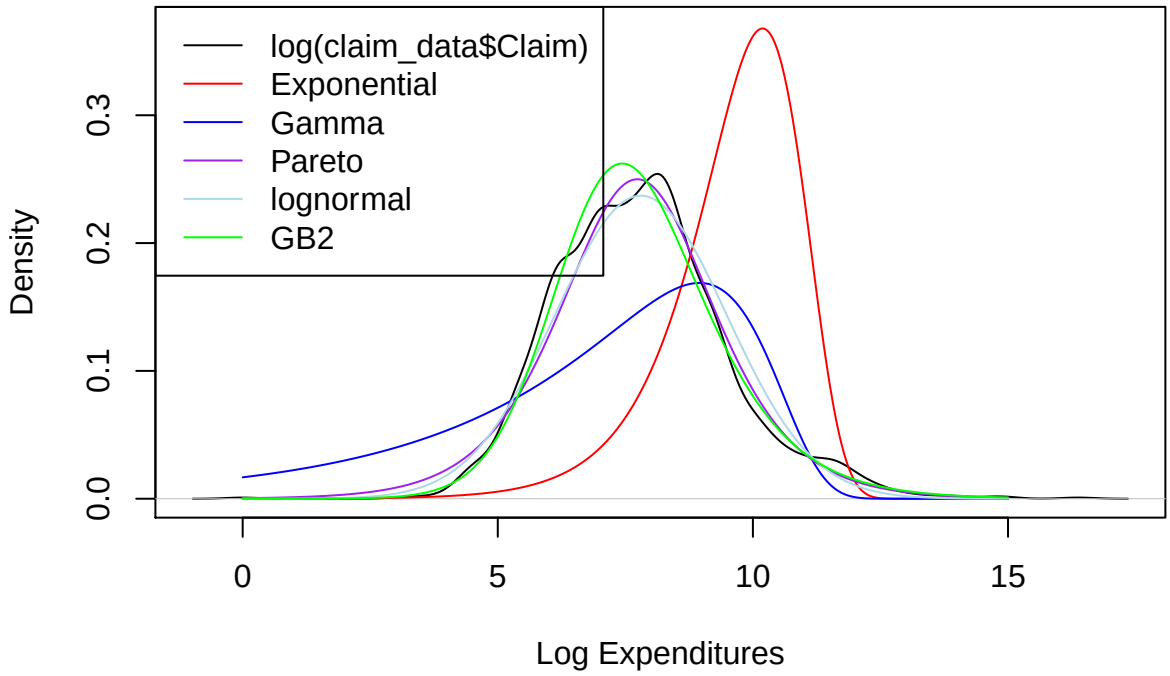
```
[1] 26789.04
```

4.3 Plotting the Fit Using Densities (on a Logarithmic Scale)

This section plots on a logarithmic scale, the smooth (nonparametric) density of claims and overlays the densities of the distributions considered above.

```
# None of these distributions is doing a great job....
plot(density(log(claim_data$Claim)), main = "", xlab = "Log Expenditures",
     ylim = c(0 ,0.37))
x <- seq(0, 15, by = 0.01)
fexp_ex <- dgamma(exp(x), scale = exp(-coef(fit.exp)), shape = 1) * exp(x)
lines(x, fexp_ex, col = "red")
```

```
fgamma_ex <- dgamma(exp(x), shape = alpha, scale = theta) * exp(x)
lines(x, fgamma_ex, col = "blue")
fpareto_ex <- dparetoII(exp(x), loc = 0, shape = exp(coef(fit.pareto)[2]),
                        scale = exp(coef(fit.pareto)[1])) * exp(x)
lines(x, fpareto_ex, col = "purple")
flnorm_ex <- dlnorm(exp(x), mean = coef(fit.LN)[1],
                    sd = exp(coef(fit.LN)[2])) * exp(x)
lines(x, flnorm_ex, col = "lightblue")
# Density for GB II
gb2_density <- function (x) {
  a_1 <- gb2_bop$par[1]
  a_2 <- gb2_bop$par[2]
  mu <- gb2_bop$par[3]
  sigma <- gb2_bop$par[4]
  xt <- (log(x) - mu) / sigma
  logexpxt <- ifelse (xt > 23, yt, log(1 + exp(xt)))
  logdens <- a_1 * xt - log(sigma) - log(beta(a_1, a_2)) -
    (a_1+a_2) * logexpxt -log(x)
  exp(logdens)
  }
fGB2_ex = gb2_density(exp(x)) * exp(x)
lines(x, fGB2_ex, col="green")

legend("topleft", c("log(claim_data$Claim)", "Exponential", "Gamma", "Pareto",
                    "lognormal", "GB2"),
       lty = 1, col = c("black","red","blue","purple","lightblue","green"))
```



4.4 Nonparametric Inference

4.4.1 Nonparametric Estimation Tools

This section illustrates non-parametric tools including moment estimators, empirical distribution function, quantiles and density estimators.

4.4.1.1 Moment Estimators

The kth moment EX^k is estimated by $\frac{1}{n}\sum_{i=1}^{n} X_i^k$. When $k = 1$ then the estimator is called the sample mean. The central moment is defined as $E(X-\mu)^k$. When $k = 2$, then the central moment is called variance. Below illustrates the mean and variance.

```
# Start with a simple example of ten points
( x_example <- c(10, rep(15,3), 20, rep(23,4), 30) )
```

```
 [1] 10 15 15 15 20 23 23 23 23 30
```

```
# Summary
summary(x_example)  # mean
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   10.0    15.0    21.5    19.7    23.0    30.0
```

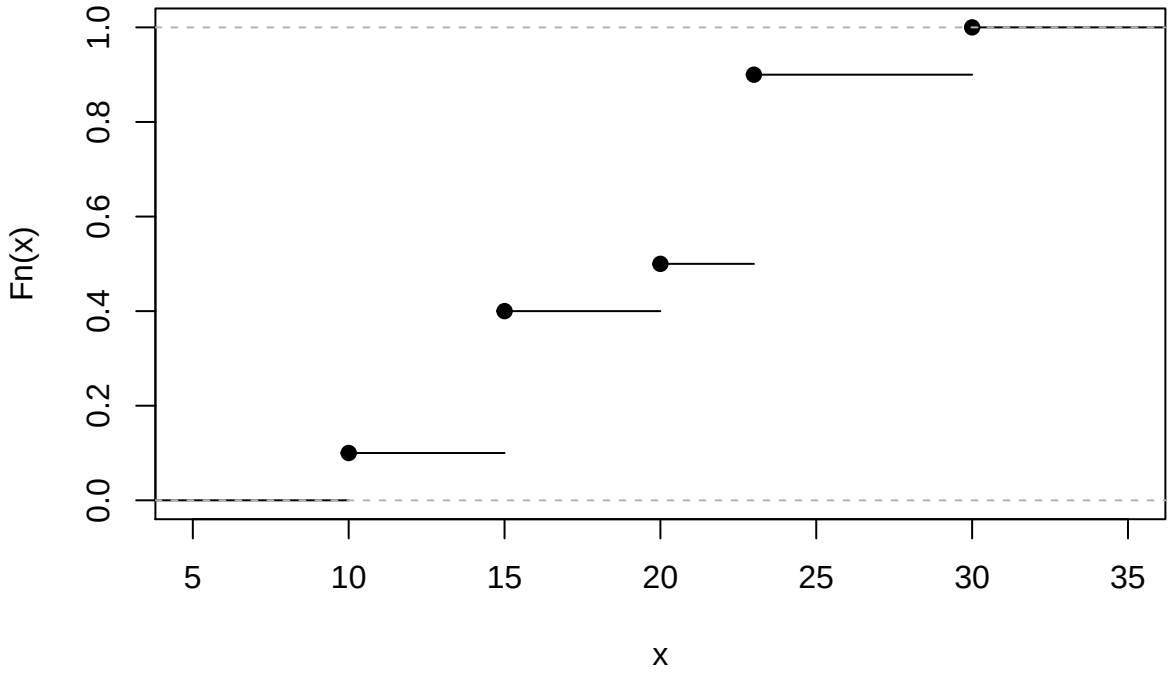
```r
sd(x_example)^2  # variance
```

```
[1] 34.45556
```

4.4.1.2 Empirical Distribution Function

The graph below gives the empirical distribution function `x_example` dataset.

```r
percentiles_x_example <- ecdf(x_example)

# Empirical distribution function
plot(percentiles_x_example, main = "", xlab = "x")
```



4.4.1.3 Quantiles

The results below gives the quantiles.

```r
# Quantiles
quantile(x_example)
```

```
  0%  25%  50%  75% 100%
10.0 15.0 21.5 23.0 30.0
```

```r
# Quantiles : set you own probabilities
quantile(x_example, probs = seq(0, 1, 0.333333))
```

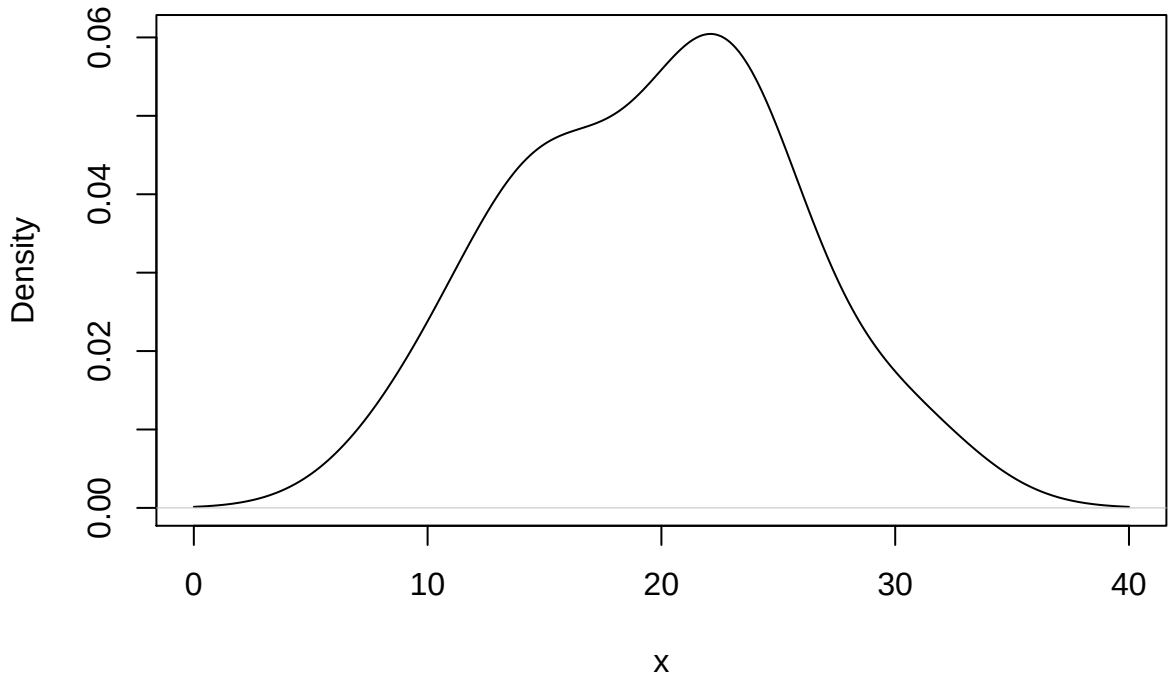
```
      0% 33.3333% 66.6666% 99.9999%
10.00000 15.00000 23.00000 29.99994
```

```
# help(quantile)
```

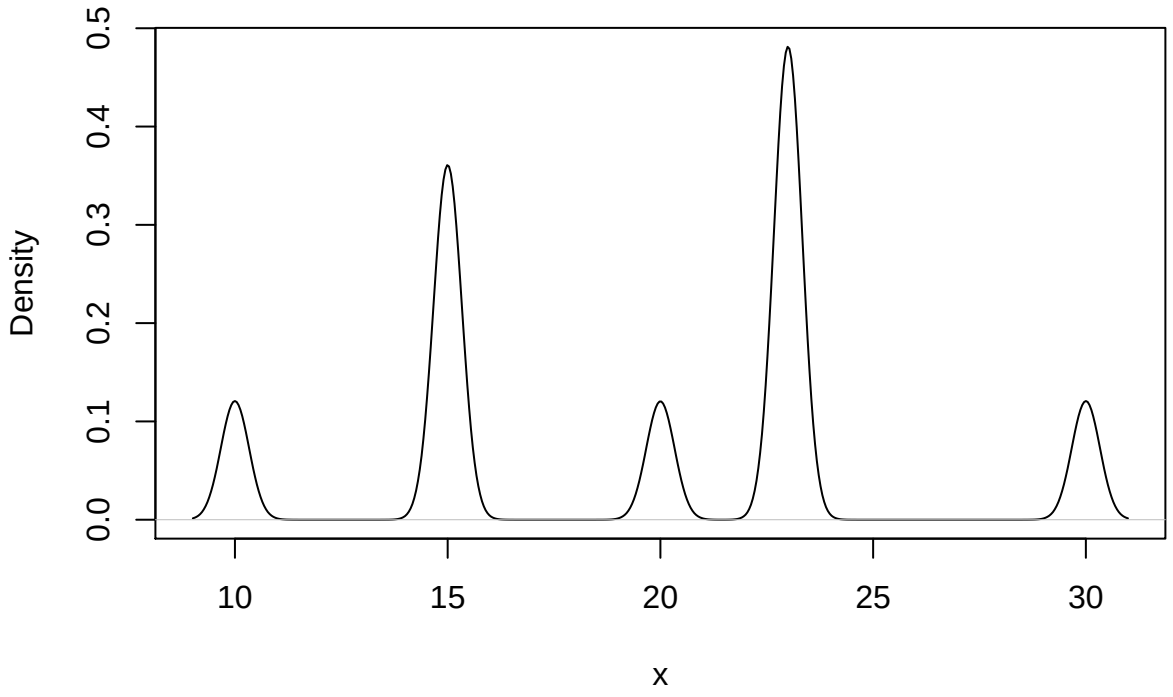
4.4.1.4 Density Estimators

The results below gives the density plots using the uniform kernel and triangular kernel.

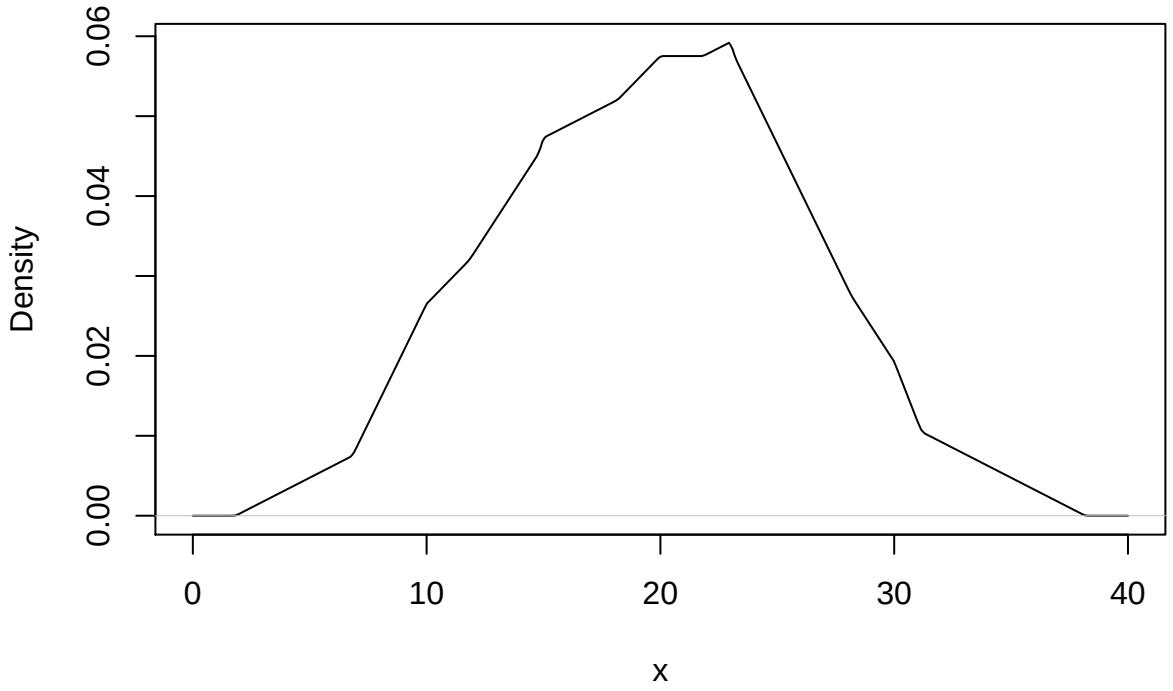
```
# Density plot
plot(density(x_example), main = "", xlab = "x")
```




```
plot(density(x_example, bw = .33), main = "", xlab = "x")  # change the bandwidth
```




```
plot(density(x_example, kernel = "triangular"), main="", xlab = "x")  # change the kernel
```



4.4.2 Property Fund Data

This section employs non-parametric estimation tools for model selection for the claims data of the Property Fund.

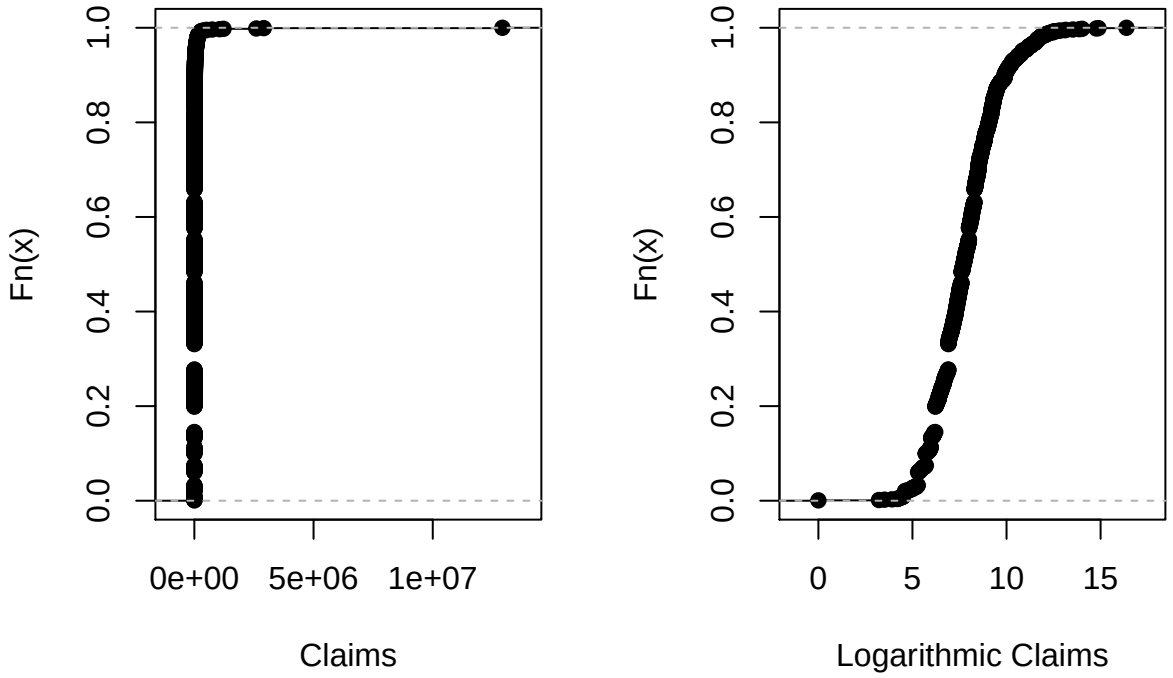
4.4.2.1 Empirical Distribution Function of Property Fund

The results below gives the empirical distribution function of the claims and claims in logarithmic units.

```
claim_lev <- read.csv("DATA/CLAIMLEVEL.csv", header=TRUE)
nrow(claim_lev)  # 6258
```

```
[1] 6258
```

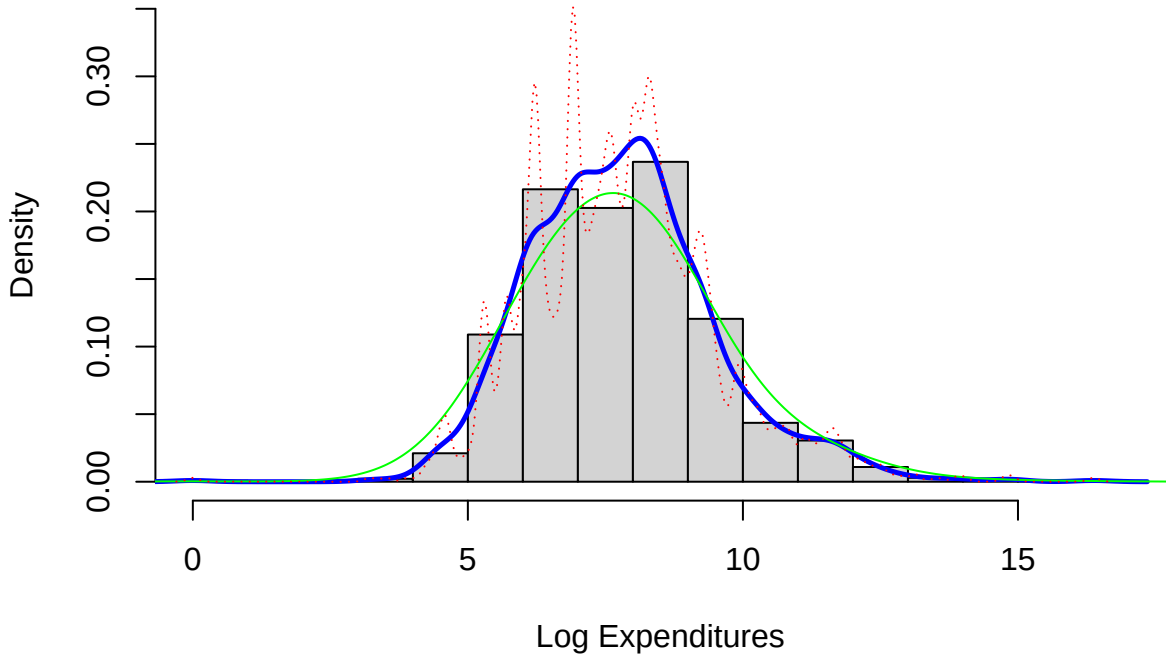
```
claim_data <- subset(claim_lev, Year==2010)  # 2010 subset
# Empirical distribution function of Property Fund
par(mfrow = c(1, 2))
percentiles  <- ecdf(claim_data$Claim)
log_percentiles  <- ecdf(log(claim_data$Claim))
plot(percentiles,  main = "", xlab = "Claims")
plot(log_percentiles, main = "", xlab = "Logarithmic Claims")
```



4.4.2.2 Density Comparison

Shows a histogram (with shaded gray rectangles) of logarithmic property claims from 2010. The blue thick curve represents a Gaussian kernel density where the bandwidth was selected automatically using an ad hoc rule based on the sample size and volatility of the data.

```
# Density comparison
hist(log(claim_data$Claim), main = "", ylim = c(0, .35), xlab = "Log Expenditures",
     freq = FALSE, col = "lightgray")
lines(density(log(claim_data$Claim)), col = "blue", lwd = 2.5)
lines(density(log(claim_data$Claim), bw = 1), col = "green")
lines(density(log(claim_data$Claim), bw = .1), col = "red", lty = 3)
```




```
density(log(claim_data$Claim))$bw  # default bandwidth
```

```
[1] 0.3255908
```

4.4.3 Nonparametric Estimation Tools For Model Selection

4.4.3.1 Fit Distributions To The Claims Data

The results below fits gamma and Pareto distribution to the claims data.

```
library(MASS)
library(VGAM)
# Inference assuming a gamma distribution
fit.gamma_2 <- glm(Claim ~ 1, data = claim_data, family = Gamma(link = log))
summary(fit.gamma_2, dispersion = gamma.dispersion(fit.gamma_2))
```

```
Call:
glm(formula = Claim ~ 1, family = Gamma(link = log), data = claim_data)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-4.287  -2.258  -1.764  -1.178  30.926

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
```

```
(Intercept) 10.18952    0.04999   203.8   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Gamma family taken to be 3.441204)

    Null deviance: 6569.1  on 1376  degrees of freedom
Residual deviance: 6569.1  on 1376  degrees of freedom
AIC: 28414

Number of Fisher Scoring iterations: 14
```

```
( theta <- exp(coef(fit.gamma_2)) * gamma.dispersion(fit.gamma_2))  # mu = theta / alpha
```

```
(Intercept)
   91613.78
```

```
( alpha <- 1 / gamma.dispersion(fit.gamma_2) )
```

```
[1] 0.2905959
```

```
# Inference assuming a Pareto distribution
fit.pareto <- vglm(Claim ~ 1, paretoII, loc = 0, data = claim_data)
summary(fit.pareto)
```

```
Call:
vglm(formula = Claim ~ 1, family = paretoII, data = claim_data,
    loc = 0)


Pearson residuals:
                Min      1Q Median     3Q   Max
loge(scale)  -6.332 -0.8289 0.1875 0.8832 1.174
loge(shape) -10.638  0.0946 0.4047 0.4842 0.513

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept):1  7.7329210  0.0933332  82.853   <2e-16 ***
(Intercept):2 -0.0008753  0.0538642  -0.016    0.987
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of linear predictors:  2

Names of linear predictors: loge(scale), loge(shape)

Log-likelihood: -13404.64 on 2752 degrees of freedom

Number of iterations: 5

No Hauck-Donner effect found in any of the estimates
```

```
head(fitted(fit.pareto))
```

```
        [,1]
[1,] 2285.03
```

```
[2,] 2285.03
[3,] 2285.03
[4,] 2285.03
[5,] 2285.03
[6,] 2285.03
```

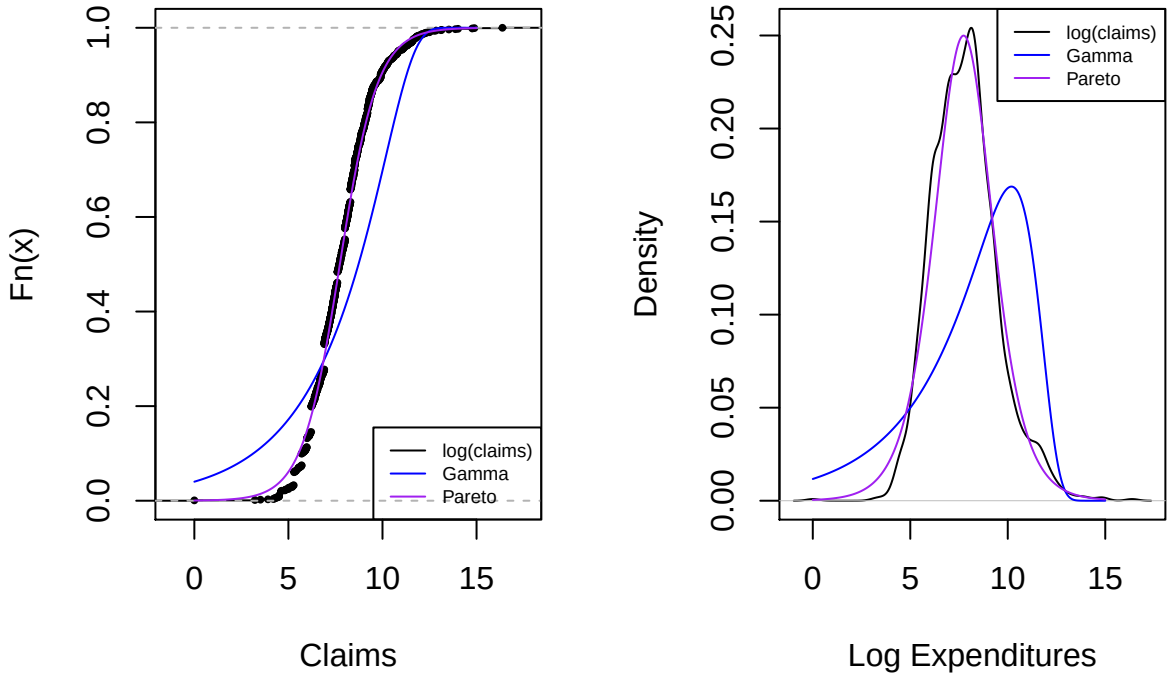
```
exp(coef(fit.pareto))
```

```
(Intercept):1 (Intercept):2
 2282.2590626     0.9991251
```

4.4.3.2 Graphical Comparison of Distributions

The graphs below reinforces the technique of overlaying graphs for comparison purposes using both the distribution function and density function. Pareto distribution provides a better fit.

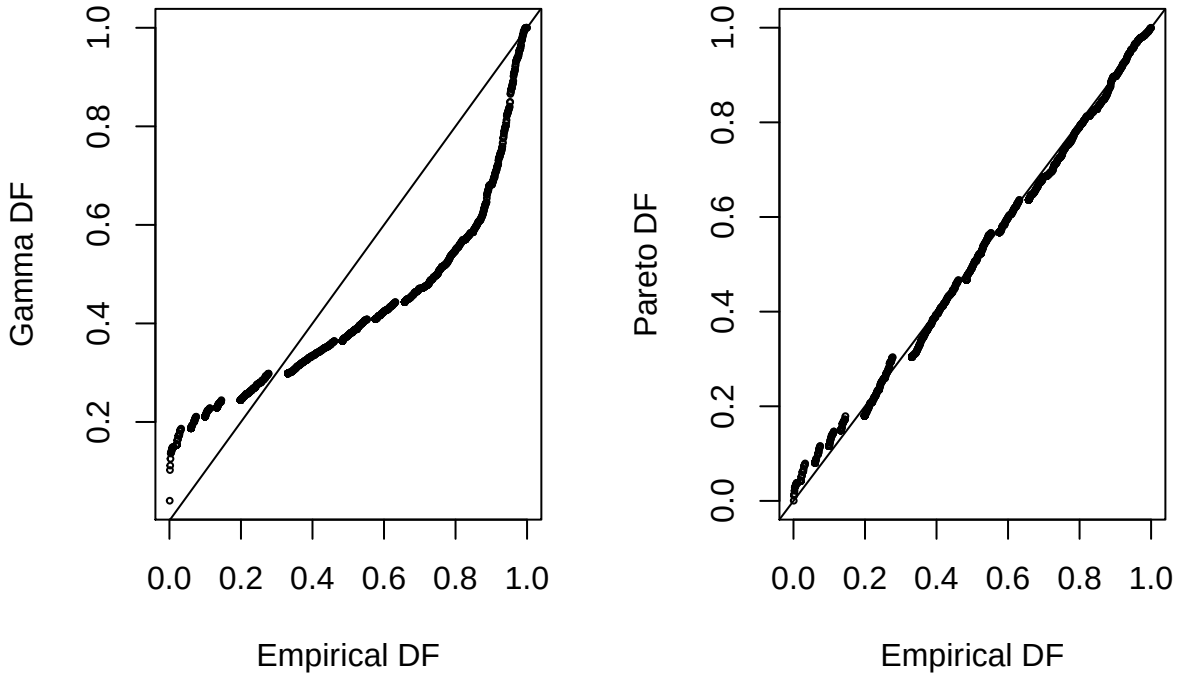
```
# Plotting the fit using densities (on a logarithmic scale)
# None of these distributions is doing a great job....
x <- seq(0, 15, by = 0.01)

par(mfrow = c(1, 2))
log_percentiles  <- ecdf(log(claim_data$Claim))
plot(log_percentiles,  main = "", xlab = "Claims", cex = 0.4)
Fgamma_ex <- pgamma(exp(x), shape = alpha, scale = theta)
lines(x, Fgamma_ex, col = "blue")
Fpareto_ex <- pparetoII(exp(x), loc = 0,shape = exp(coef(fit.pareto)[2]),
                        scale = exp(coef(fit.pareto)[1]))
lines(x, Fpareto_ex, col = "purple")
legend("bottomright", c("log(claims)", "Gamma", "Pareto"), lty = 1, cex = 0.6,
       col = c("black","blue","purple"))

plot(density(log(claim_data$Claim)) , main = "", xlab = "Log Expenditures")
fgamma_ex <- dgamma(exp(x), shape = alpha, scale = theta) * exp(x)
lines(x, fgamma_ex, col = "blue")
fpareto_ex <- dparetoII(exp(x), loc = 0, shape = exp(coef(fit.pareto)[2]),
                        scale = exp(coef(fit.pareto)[1])) * exp(x)
lines(x, fpareto_ex, col = "purple")
legend("topright", c("log(claims)", "Gamma", "Pareto"), lty = 1, cex = 0.6,
       col = c("black","blue","purple"))
```



4.4.3.3 P-P Plots

Shows *pp* plots for the Property Fund data; the fitted gamma is on the left and the fitted Pareto is on the right. Pareto distribution provides a better fit again.

```
# PP Plot
par(mfrow = c(1, 2))
Fgamma_ex <- pgamma(claim_data$Claim, shape = alpha, scale = theta)
plot(percentiles(claim_data$Claim), Fgamma_ex, xlab = "Empirical DF",
     ylab = "Gamma DF", cex = 0.4)
abline(0, 1)
Fpareto_ex <- pparetoII(claim_data$Claim, loc = 0,
                        shape = exp(coef(fit.pareto)[2]),
                        scale = exp(coef(fit.pareto)[1]))
plot(percentiles(claim_data$Claim), Fpareto_ex, xlab = "Empirical DF",
     ylab = "Pareto DF", cex = 0.4)
abline(0, 1)
```

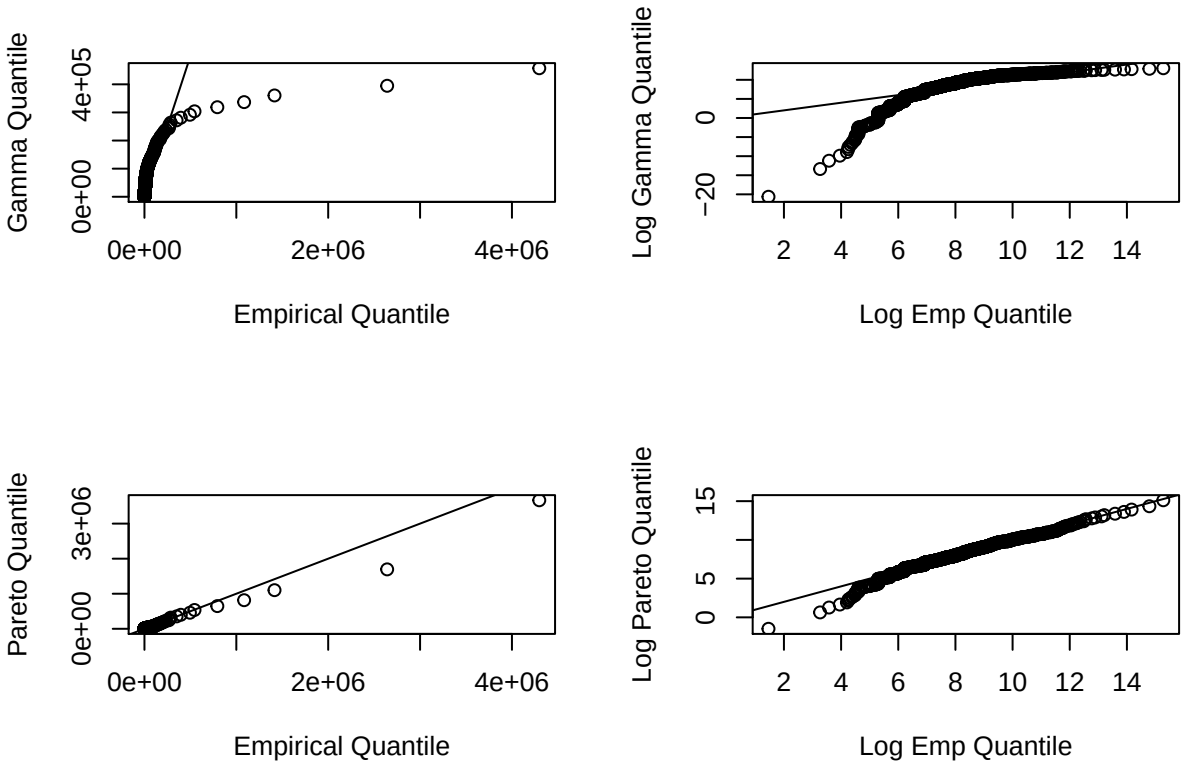



```
#dev.off()
```

4.4.3.4 Q-Q Plots

In the graphs below the quantiles are plotted on the original scale in the left-hand panels, on the log scale in the right-hand panel, to allow the analyst to see where a fitted distribution is deficient.

```
# Q-Q plot
par(mfrow = c(2, 2))
x_seq <- seq(0.0001, 0.9999, by = 1 / length(claim_data$Claim))
emp_quant <- quantile(claim_data$Claim, x_seq)
gamma_quant <- qgamma(x_seq, shape = alpha, scale = theta)
plot(emp_quant, gamma_quant, xlab = "Empirical Quantile", ylab = "Gamma Quantile")
abline(0, 1)
plot(log(emp_quant), log(gamma_quant), xlab = "Log Emp Quantile",
     ylab = "Log Gamma Quantile")
abline(0, 1)
pareto_quant <- qparetoII(x_seq, loc = 0, shape = exp(coef(fit.pareto)[2]),
                          scale = exp(coef(fit.pareto)[1]))
plot(emp_quant, pareto_quant, xlab = "Empirical Quantile", ylab = "Pareto Quantile")
abline(0, 1)
plot(log(emp_quant), log(pareto_quant), xlab = "Log Emp Quantile",
     ylab="Log Pareto Quantile")
abline(0, 1)
```



4.4.3.5 Goodness of Fit Statistics

For reporting results, it can be effective to supplement graphical displays with selected statistics that summarize model goodness of fit. The results below provides three commonly used goodness of fit statistics.

```
library(goftest)
# Kolmogorov-Smirnov # the test statistic is "D"
ks.test(claim_data$Claim, "pgamma", shape = alpha, scale = theta)
```

```
    One-sample Kolmogorov-Smirnov test

data:  claim_data$Claim
D = 0.26387, p-value < 2.2e-16
alternative hypothesis: two-sided
```

```
ks.test(claim_data$Claim, "pparetoII", loc = 0, shape = exp(coef(fit.pareto)[2]),
        scale = exp(coef(fit.pareto)[1]))
```

```
    One-sample Kolmogorov-Smirnov test

data:  claim_data$Claim
D = 0.047824, p-value = 0.003677
alternative hypothesis: two-sided
```

```
# Cramer-von Mises  # the test statistic is "omega_2"
cvm.test(claim_data$Claim, "pgamma", shape = alpha, scale = theta)
```

```
    Cramer-von Mises test of goodness-of-fit
    Null hypothesis: Gamma distribution
    with parameters shape = 0.290595934110839, scale =
    91613.779421033

data:  claim_data$Claim
omega2 = 33.378, p-value = 2.549e-05
```

```
cvm.test(claim_data$Claim, "pparetoII", loc = 0, shape = exp(coef(fit.pareto)[2]),
         scale = exp(coef(fit.pareto)[1]))
```

```
    Cramer-von Mises test of goodness-of-fit
    Null hypothesis: distribution 'pparetoII'
    with parameters shape = 0.999125131378519, scale =
    2282.25906257586

data:  claim_data$Claim
omega2 = 0.38437, p-value = 0.07947
```

```
# Anderson-Darling  # the test statistic is "An"
ad.test(claim_data$Claim, "pgamma", shape = alpha, scale = theta)
```

```
    Anderson-Darling test of goodness-of-fit
    Null hypothesis: Gamma distribution
    with parameters shape = 0.290595934110839, scale =
    91613.779421033

data:  claim_data$Claim
An = Inf, p-value = 4.357e-07
```

```
ad.test(claim_data$Claim, "pparetoII", loc = 0, shape = exp(coef(fit.pareto)[2]),
        scale = exp(coef(fit.pareto)[1]))
```

```
    Anderson-Darling test of goodness-of-fit
    Null hypothesis: distribution 'pparetoII'
    with parameters shape = 0.999125131378519, scale =
    2282.25906257586

data:  claim_data$Claim
An = 4.1264, p-value = 0.007567
```

4.5 MLE for Grouped Data

4.5.1 MLE for Grouped Data- SOA Exam C # 276

Losses follow the distribution function $F(x) = 1 - (\theta/x), \quad x > 0$. A sample of 20 losses resulted in the following:

Interval	Number of Losses
(0,10]	9
(10,25]	6
(25,infinity)	5

Calculate the maximum likelihood estimate of θ.

```
# Log likelihood function
lik_grp <- function (theta) {
  log_like <- log(((1 - (theta / 10))^9) * (((theta / 10) - (theta / 25))^6) *
                    (((theta / 25))^5))
  return(-sum(log_like))
}
# "optim" is a general purpose minimization function
grp_lik <- optim(c(1), lik_grp, method = c("L-BFGS-B"), hessian = TRUE)
# Estimates - Answer "B" on SoA Problem
grp_lik$par
```

```
[1] 5.5
```

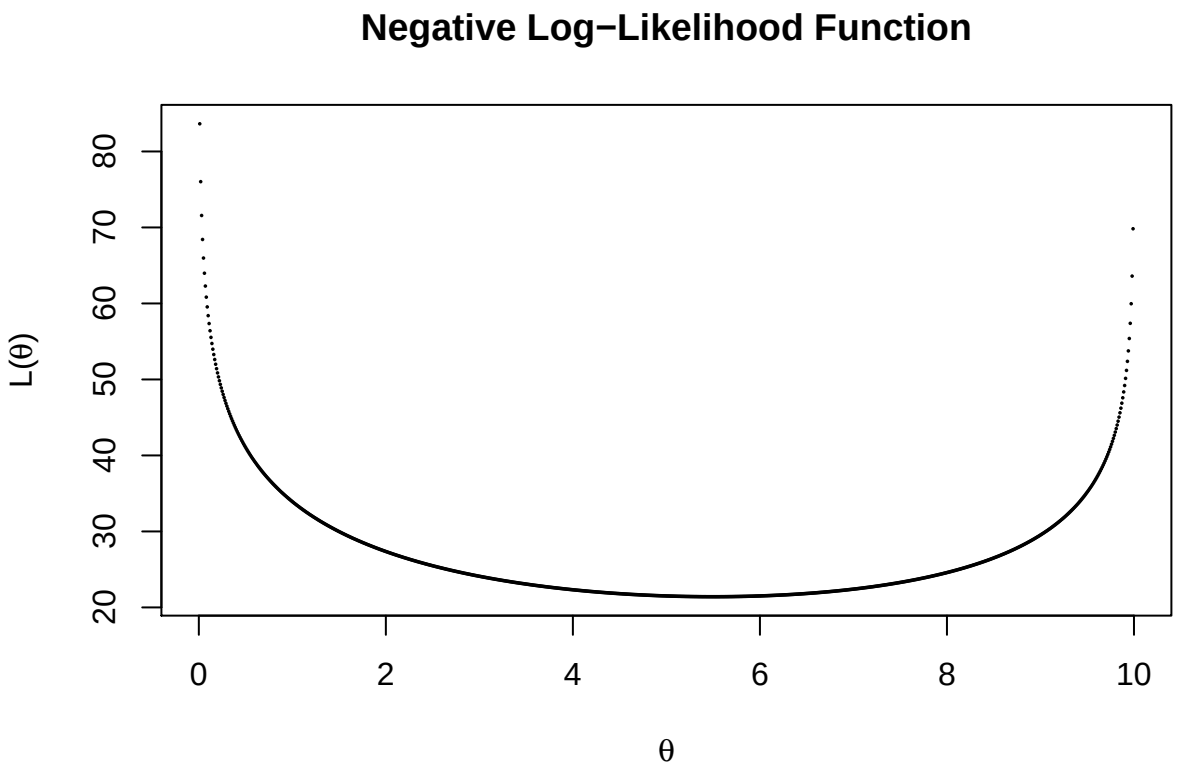
```
# Standard error
sqrt(diag(solve(grp_lik$hessian)))
```

```
[1] 1.11243
```

```
# t-statistics
( tstat <- grp_lik$par / sqrt(diag(solve(grp_lik$hessian))) )
```

```
[1] 4.944132
```

```
# Plot of Negative Log-Likelihood function
vllh <- Vectorize(lik_grp, "theta")
theta <- seq(0, 10, by = 0.01)
plot(theta, vllh(theta), pch = 16, main = "Negative Log-Likelihood Function", cex = .25,
     xlab = expression(theta), ylab = expression(paste("L(", theta, ")")))
```


Negative Log−Likelihood Function
L(θ)
20
30
40
50
60
70
80
0
2
4
6
8
10
θ

Chapter 5

Simulation

*This file contains illustrative **R** code for computing important count distributions. When reviewing this code, you should open an **R** session, copy-and-paste the code, and see it perform. Then, you will be able to change parameters, look up commands, and so forth, as you go.*

5.1 Simulation - Inversion Method

This section shows how to use the inversion method to simulate claims from a gamma distribution. The results below are summary statistics from the simulated data.

```
# Simulation - gamma
library(moments)
set.seed(2)  # set seed to reproduce work
n_tot <- 20000  # number of simulations
alpha <- 2
theta <- 100

losses <- rgamma(n_tot, alpha, scale = theta)
summary(losses)
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
   0.0921   96.3265  167.8035  200.1747  268.2257 1110.1298
```

```
k <- 0.95
percentile_loss <- quantile(losses,k)  # Kth percentile of losses
percentile_loss
```

```
     95%
473.8218
```

```
########################################

# OR you can use this method to simulate losses
# Fx <- runif(n_tot)
# losses <- qgamma(Fx, alpha, scale = theta)

########################################

# For the Pareto Distribution, use
```

```
# library(VGAM)
# n_tot <- 10000  # number of simulations
# alpha <- 3
# theta <- 25000
# losses <- rparetoII(n_tot, scale = theta, shape = alpha)
# rparetoII(n_tot, scale = theta, shape = alpha)
```

A few quick notes on these commands:

- The `rgamma()` function randomly generates data from the Gamma distribution. In this illustration the data was generated from a gamma distribution with parameters `shape = alpha = 2` and `scale = theta = 100`.
- The `quantile()` function provides sample quantiles corresponding to the given probabilities. Here we wanted the simulated loss data corresponding to the 95th percentile.

5.2 Comparing Moments from The Simulated Data to Theoretical Moments

```
library(pander)
# Raw moments for k = 0.5
# Theoretical
k <- 0.5
T_0.5 <- round(((theta^k) * gamma(alpha + k)) / gamma(alpha), 2)

# Simulated data raw moments
S_0.5 <- round(moment(losses, order = k, central = FALSE), 2)


# Raw moments for k = 1
# Theoretical
k <- 1
T_1 <- ((theta^k) * gamma(alpha + k)) / gamma(alpha)

# Simulated data raw moments
S_1 <- round(moment(losses, order = k, central = FALSE), 2)

# Raw moments for k = 2
# Theoretical
k <- 2
T_2 <- ((theta^k) * gamma(alpha + k)) / gamma(alpha)

#Simulated data raw moments
S_2<-round(moment(losses, order = k, central = FALSE),2)

# Raw moments for k = 3
# Theoretical
k <- 3
T_3<-((theta^k) * gamma(alpha + k)) / gamma(alpha)

# Simulated data raw moments
S_3 <- round(moment(losses, order = k, central = FALSE), 2)
```

```
# Raw moments for k = 3

# Theoretical
k <- 4
T_4 <- ((theta^k) * gamma(alpha + k)) / gamma(alpha)

# Simulated data raw moments
S_4 <- round(moment(losses, order = k, central = FALSE), 2)

pander(rbind(c("k", 0.5, 1, 2, 3, 4), c("Theoretical", T_0.5, T_1, T_2, T_3, T_4),
             c("Simulated", S_0.5, S_1, S_2, S_3, S_4)))
```

k	0.5	1	2	3	4
Theoretical	13.29	200	60000	2.4e+07	1.2e+10
Simulated	13.3	200.17	60069.73	23993892.53	11897735665.89

A few quick notes on these commands:

- The `moment()` function computes all the sample moments of the chosen type up to a given order. In this illustration, we wanted the raw sample moments of the *kth* order.
- The `round()` function was used to round values.

Chapter 6

Aggregate Claim Simulation

*This file demonstrates simulation of aggregate claim distributions. When reviewing this code, you should open an **R** session, copy-and-paste the code, and see it perform. Then, you will be able to change parameters, look up commands, and so forth, as you go.*

6.1 Collective Risk Model: without coverage modifications

S = X_1 + ... + X_N

Assume N ~ Poisson(lambda=2) and X ~ Pareto(alpha=3,theta=5000)

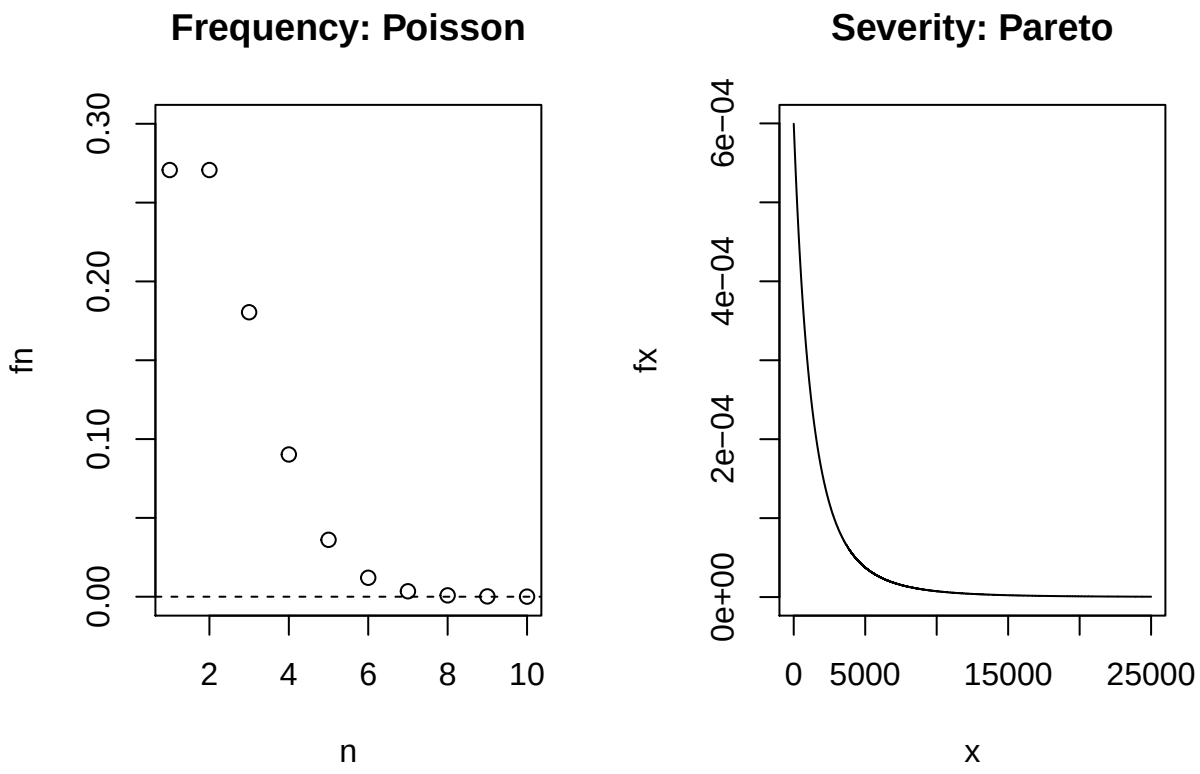
6.1.1 Set Parameters

```
lambda <- 2
alpha <- 3
theta <- 5000
```

6.1.2 Show frequency and severity distributions

Graphing the our frequency (N) and severity (X) distributions

```
par(mfrow=c(1,2))

n <- 1:10
fn <- dpois(1:10,lambda)

plot(n,fn,ylim=c(0,0.3),main="Frequency: Poisson")
abline(h=0,lty=2)

x <- seq(1,25000,1)
fx <- alpha*theta^alpha/(x+theta)^(alpha+1)

plot(x,fx,type="l",main="Severity: Pareto")
```



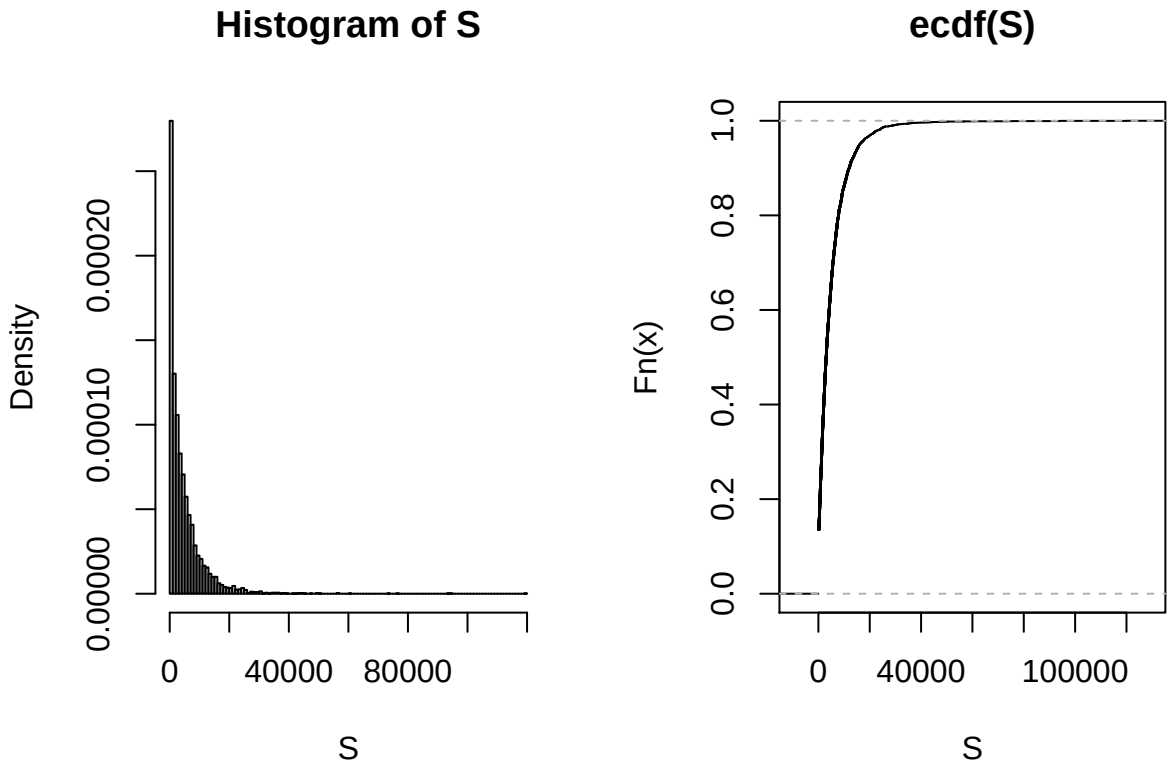
6.1.3 Set sample size for the simulation

We're going to simulate 5000 observations of S

```
set.seed(123)
size <- 5000
S <- rep(NA,size)
N <- rpois(size,lambda)
for (i in 1:size){
  uu <- runif(N[i])
  X <- theta*((1-uu)^(-1/alpha)-1)
  S[i] <- sum(X)
}
```

6.1.4 Show distribution of aggregate loss S

```
par(mfrow=c(1,2))
hist(S,freq=F,breaks=100)
plot(ecdf(S),xlab="S")
```



6.2 Applications

6.2.1 Find descriptive statistics

Here we show numerical descriptions of our simulated distribution S

```
mean(S)                                  # sample mean
```

```
[1] 4829.894
```

```
sd(S)                                    # sample standard deviation
```

```
[1] 6585.567
```

```
quantile(S,prob=c(0.05,0.5,0.95))        # percentiles
```

```
       5%       50%       95%
    0.000  2846.073 15983.408
```

6.2.2 Calculate cdf

```
sum((S==0))/size
```

```
[1] 0.1348
```

Pr(S=0)

```
sum(S<=mean(S))/size
```

```
[1] 0.6578
```

Pr(S<=E(S))

```
sum(S>mean(S))/size
```

```
[1] 0.3422
```

Pr(S>E(S))

6.2.3 Calculate risk measures

CTE is also known as TVaR

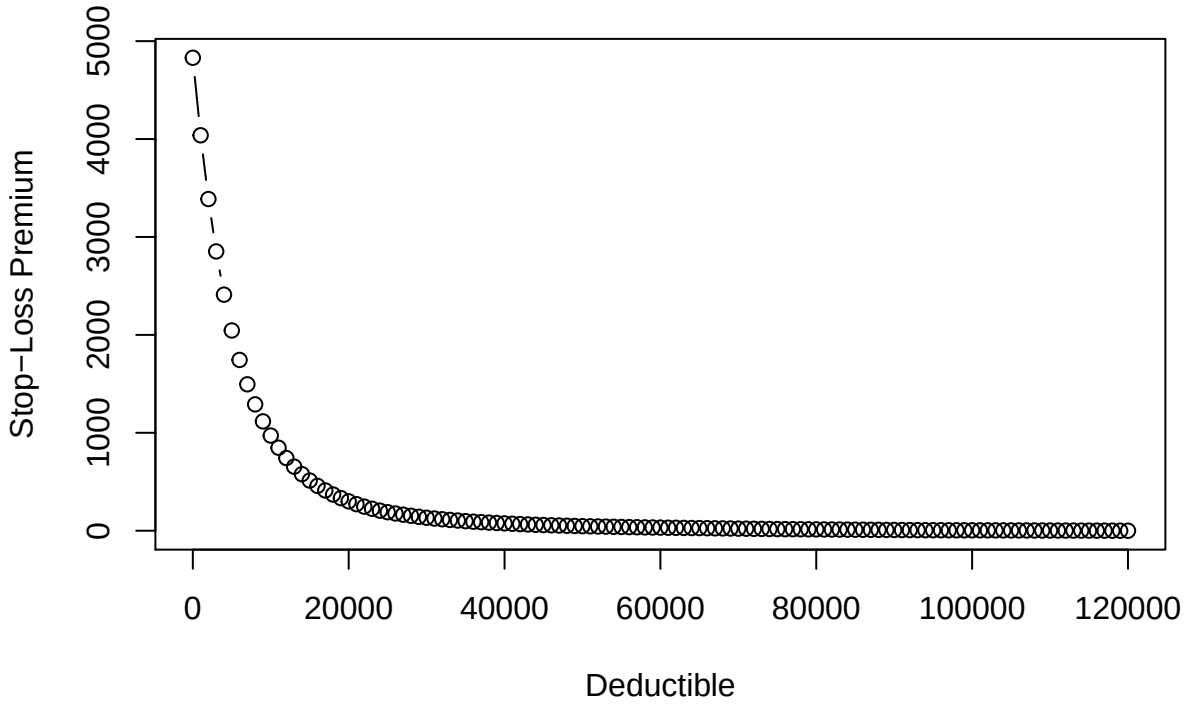
```
VaR <- quantile(S,prob=0.99)          # significance level = 0.01
CTE <- sum(S*(S>VaR))/sum((S>VaR))
rm <- c(VaR,CTE)
names(rm) <- c("VaR","CTE")
print(rm)
```

```
     VaR      CTE
28636.56 43193.19
```

6.2.4 Pricing stop-loss insurance - Set deductible

Here we plot how the premium for a stop-loss insurance product changes based on the size of the deductible

```
par(mfrow=c(1,1))
d <- seq(0,120000,1000)
price <- rep(NA,length(d))
for (i in 1:length(d)){
  price[i] = sum((S-d[i])*(S>d[i]))/size
}
plot(d,price,xlab="Deductible",ylab="Stop-Loss Premium",type="b")
```


Stop−Loss Premium
0
1000
2000
3000
4000
5000
0
20000
40000
60000
80000
100000
120000
Deductible

Chapter 7

FreqSev

*This file contains illustrative **R** code for calculations involving frequency and severity of distributions. When reviewing this code, you should open an **R** session, copy-and-paste the code, and see it perform. Then, you will be able to change parameters, look up commands, and so forth, as you go.*

7.1 Getting the Data

Before we can do any analysis we must import the data.

7.1.1 read data "MassAuto.csv"

Import the excel file into R.

```
dat <- read.csv(file = "Data/MassAuto.csv",header=TRUE)
```

7.1.2 Check Variable Names

This code outputs a list of all the variable names of the excel file. This is useful for determining what kind of data you re working with.

```
names(dat)
```

```
[1] "VIN"       "Territory" "Class"     "Loss1"     "Loss2"
```

7.1.3 Calculate Total Losses in "dat"

This code creates a new column representing the sum of the two loss columns.

```
dat$Loss <- dat$Loss1 + dat$Loss2
```

7.2 Fit Frequency Models

7.2.1 Prepare Data for Frequency Models

You may have to install the package "dplyr" to run this code.

```
library(dplyr)
freq.dat <- dat %>% group_by(VIN) %>% summarise(tLoss = sum(Loss),count = sum(Loss>0))
dim(freq.dat)
```

```
[1] 49894     3
```

7.2.2 Fit Poisson distribution

Here we fit a poisson distrubtion to the data and run log likelihood to determine the most likely parameter for the distribution. We then calculate the standard error of this estimate.

7.2.2.1 Define the pmf for the Poisson Distribution

```
loglikPois<-function(parms){
  lambda=parms[1]
  llk <- -sum(log(dpois(freq.dat$count,lambda)))
  llk
}
ini.Pois <- 1
zop.Pois <- nlminb(ini.Pois,loglikPois,lower=c(1e-6),upper=c(Inf))
print(zop.Pois)
```

```
$par
[1] 0.04475488

$objective
[1] 9243.476

$convergence
[1] 0

$iterations
[1] 17

$evaluations
function gradient
      24       20

$message
[1] "relative convergence (4)"
```

7.2.2.2 Obtain Standard Error

```
library(numDeriv)
est <- zop.Pois$par
names(est) <- c("lambda")
hess<-hessian(loglikPois,est)
se <-sqrt(diag(solve(hess)))
print(cbind(est,se))
```

```
             est           se
lambda 0.04475488 0.0009471004
```

7.2.3 Fit Negative Binomial Distribution

Now we fit a negative binomial distribution to the data using log likelihood.

We then calculate the standard error of this estimate.

7.2.3.1 Define pmf for Negative Binomial

```
dnb <- function(y,r,beta){
  gamma(y+r)/gamma(r)/gamma(y+1)*(1/(1+beta))^r*(beta/(1+beta))^y
}
loglikNB<-function(parms){
  r=parms[1]
  beta=parms[2]
  llk <- -sum(log(dnb(freq.dat$count,r,beta)))
  llk
}
ini.NB <- c(1,1)
zop.NB <- nlminb(ini.NB,loglikNB,lower=c(1e-6,1e-6),upper=c(Inf,Inf))
print(zop.NB)
```

```
$par
[1] 0.86573901 0.05169541

$objective
[1] 9218.902

$convergence
[1] 0

$iterations
[1] 27

$evaluations
function gradient
      32       63

$message
[1] "relative convergence (4)"
```

7.2.3.2 Obtain Standard Error

```
library(numDeriv)
est <- zop.NB$par
names(est) <- c("r","beta")
hess<-hessian(loglikNB,est)
se <-sqrt(diag(solve(hess)))
print(cbind(est,se))
```

```
            est          se
r    0.86573901 0.161126426
beta 0.05169541 0.009686412
```

7.2.4 Goodness-of-Fit

Here we calculate goodness of fit for the emipircal, poission, and negative binomial models.

7.2.4.1 Set Parameters

```
lambda<-zop.Pois$par
r<-zop.NB$par[1]
beta<-zop.NB$par[2]
numrow<-max(freq.dat$count)+1
```

7.2.4.2 Empirical Model

```
emp<-rep(0,numrow+1)
for(i in 1:(numrow+1)){
  emp[i]<-sum(freq.dat$count==i-1)
}
```

7.2.4.3 Poisson Model

```
pois<-rep(0,numrow+1)
for(i in 1:numrow){
  pois[i]<-length(freq.dat$count)*dpois(i-1,lambda)
}
pois[numrow+1]<- length(freq.dat$count)-sum(pois)
```

7.2.4.4 Negative Binomial Model

```
nb<-rep(0,numrow+1)
for(i in 1:numrow){
  nb[i]<-length(freq.dat$count)*dnb(i-1,r,beta)
}
nb[numrow+1]<- length(freq.dat$count)-sum(nb)
```

7.2.4.5 Output

```
freq <- cbind(emp,pois,nb)
rownames(freq) <- c("0","1","2","3",">3")
colnames(freq) <- c("Empirical","Poisson","NegBin")
round(freq,digits=3)
```

```
   Empirical   Poisson    NegBin
0      47763 47710.232 47763.629
1       2036  2135.266  2032.574
2         88    47.782    93.203
3          7     0.713     4.376
>3         0     0.008     0.218
```

7.2.5 Chi Square Statistics

Here we run chi square to determine the goodness of fit

```
chi.pois <- sum((pois-emp)^2/pois)
chi.negbin <- sum((nb-emp)^2/nb)
chisq <- c(Poisson=chi.pois, NegBin=chi.negbin)
print(chisq)
```

```
  Poisson    NegBin
93.986694  2.087543
```

7.3 Fit Severity Models

7.3.1 Prepare Data for Severity Models

```
sev.dat <- subset(dat,Loss>0)
dim(sev.dat)
```

```
[1] 2233    6
```

7.3.2 Log-normal distribution

7.3.2.1 Use "VGAM" Library for Estimation of Parameters

You may have to install the package "VGAM" to run this code.

```
library(VGAM)
fit.LN <- vglm(Loss ~ 1, family=lognormal, data = sev.dat)
summary(fit.LN)
```

```

Call:
vglm(formula = Loss ~ 1, family = lognormal, data = sev.dat)


Pearson residuals:
                Min      1Q   Median      3Q    Max
meanlog     -4.1427 -0.4450  0.05912  0.5917  2.254
loge(sdlog) -0.7071 -0.6636 -0.51680 -0.1437 11.428

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept):1  7.16870    0.03662  195.76   <2e-16 ***
(Intercept):2  0.54838    0.01496   36.65   <2e-16 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of linear predictors:  2

Names of linear predictors: meanlog, loge(sdlog)

Log-likelihood: -20400.73 on 4464 degrees of freedom

Number of iterations: 3

No Hauck-Donner effect found in any of the estimates
```

Coefficients (note scale parameter is in log scale).

```
coef(fit.LN)
```

```
(Intercept):1 (Intercept):2
    7.1687024     0.5483791
```

Confidence intervals for model parameters.

```
confint(fit.LN, level=0.95)
```

```
                  2.5 %    97.5 %
(Intercept):1 7.0969291 7.2404757
(Intercept):2 0.5190507 0.5777076
```

Loglikelihood for lognormal.

```
logLik(fit.LN)
```

```
[1] -20400.73
```

AIC for lognormal.

```
AIC(fit.LN)
```

```
[1] 40805.47
```

BIC for lognormal.

```
BIC(fit.LN)
```

```
[1] 40816.89
```

Covariance matrix for model parameters.

```
vcov(fit.LN)
```

```
              (Intercept):1 (Intercept):2
(Intercept):1   0.001341004   0.000000000
(Intercept):2   0.000000000   0.000223914
```

7.3.2.2 User-Defined Likelihood Function

Here we estimate sigma directly instead of in log scale.

```
loglikLN<-function(parms){
  mu=parms[1]
  sigma=parms[2]
  llk <- -sum(log(dlnorm(sev.dat$Loss, mu, sigma)))
```

```
  llk
}
ini.LN <- c(coef(fit.LN)[1],exp(coef(fit.LN)[2]))
zop.LN <- nlminb(ini.LN,loglikLN,lower=c(-Inf,1e-6),upper=c(Inf,Inf))
print(zop.LN)
```

```
$par
(Intercept):1 (Intercept):2
     7.168702      1.730446

$objective
[1] 20400.73

$convergence
[1] 0

$iterations
[1] 1

$evaluations
function gradient
       2        2

$message
[1] "relative convergence (4)"
```

7.3.2.3 Obtain Standard Error

```
library(numDeriv)
est <- zop.LN$par
names(est) <- c("mu","sigma")
hess<-hessian(loglikLN,est)
se <-sqrt(diag(solve(hess)))
print(cbind(est,se))
```

```
           est         se
mu    7.168702 0.03661961
sigma 1.730446 0.02589397
```

7.3.3 Pareto Distribution

7.3.3.1 Use "VGAM" Library for Estimation of Parameters

You may have to install the package "VGAM" to run this code.

```
library(VGAM)
fit.pareto <- vglm(Loss ~ 1, paretoII, loc=0, data = sev.dat)
summary(fit.pareto)
```

```
Call:
vglm(formula = Loss ~ 1, family = paretoII, data = sev.dat, loc = 0)
```

```
Pearson residuals:
               Min      1Q  Median     3Q    Max
loge(scale) -2.044 -0.7540 0.02541 0.8477 1.2958
loge(shape) -5.813  0.1526 0.36197 0.4322 0.4559

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept):1  7.99629    0.08417  95.004   <2e-16 ***
(Intercept):2  0.52653    0.05699   9.239   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Number of linear predictors:  2

Names of linear predictors: loge(scale), loge(shape)

Log-likelihood: -20231.91 on 4464 degrees of freedom

Number of iterations: 5

No Hauck-Donner effect found in any of the estimates
```

```
head(fitted(fit.pareto))
```

```
         [,1]
[1,] 1502.569
[2,] 1502.569
[3,] 1502.569
[4,] 1502.569
[5,] 1502.569
[6,] 1502.569
```

```
coef(fit.pareto)                #note both parameters are in log scale
```

```
(Intercept):1 (Intercept):2
    7.9962932     0.5265276
```

```
exp(coef(fit.pareto))           #estimate of parameters
```

```
(Intercept):1 (Intercept):2
  2969.928635      1.693043
```

```
confint(fit.pareto, level=0.95)  #confidence intervals for model parameters
```

```
                2.5 %    97.5 %
(Intercept):1 7.831327 8.1612593
(Intercept):2 0.414834 0.6382213
```

```
logLik(fit.pareto)              #loglikelihood for pareto
```

```
[1] -20231.91
```

```
AIC(fit.pareto)                 #AIC for pareto
```

```
[1] 40467.83
```

```
BIC(fit.pareto)                 #BIC for pareto
```

```
[1] 40479.25
```

```
vcov(fit.pareto)                    #covariance matrix for model parameters
```

```
              (Intercept):1 (Intercept):2
(Intercept):1   0.007084237   0.004453555
(Intercept):2   0.004453555   0.003247586
```

7.3.3.2 User-Defined Likelihood Function

Here we estimate alpha and theta directly to define the pareto density.

```
dpareto <- function(y,theta,alpha){
  alpha*theta^alpha/(y+theta)^(alpha+1)
}
loglikP<-function(parms){
  theta=parms[1]
  alpha=parms[2]
  llk <- -sum(log(dpareto(sev.dat$Loss,theta,alpha)))
  llk
}
ini.P <- exp(coef(fit.pareto))
zop.P <- nlminb(ini.P,loglikP,lower=c(1e-6,1e-6),upper=c(Inf,Inf))
print(zop.P)
```

```
$par
(Intercept):1 (Intercept):2
  2969.928635      1.693043

$objective
[1] 20231.91

$convergence
[1] 0

$iterations
[1] 1

$evaluations
function gradient
       1        2

$message
[1] "both X-convergence and relative convergence (5)"
```

7.3.3.3 Obtain Standard Error

```
library(numDeriv)
est <- zop.P$par
names(est) <- c("theta","alpha")
hess<-hessian(loglikP,est)
se <-sqrt(diag(solve(hess)))
print(cbind(est,se))
```

```
             est          se
theta 2969.928635 248.24191162
alpha    1.693043   0.09590892
```

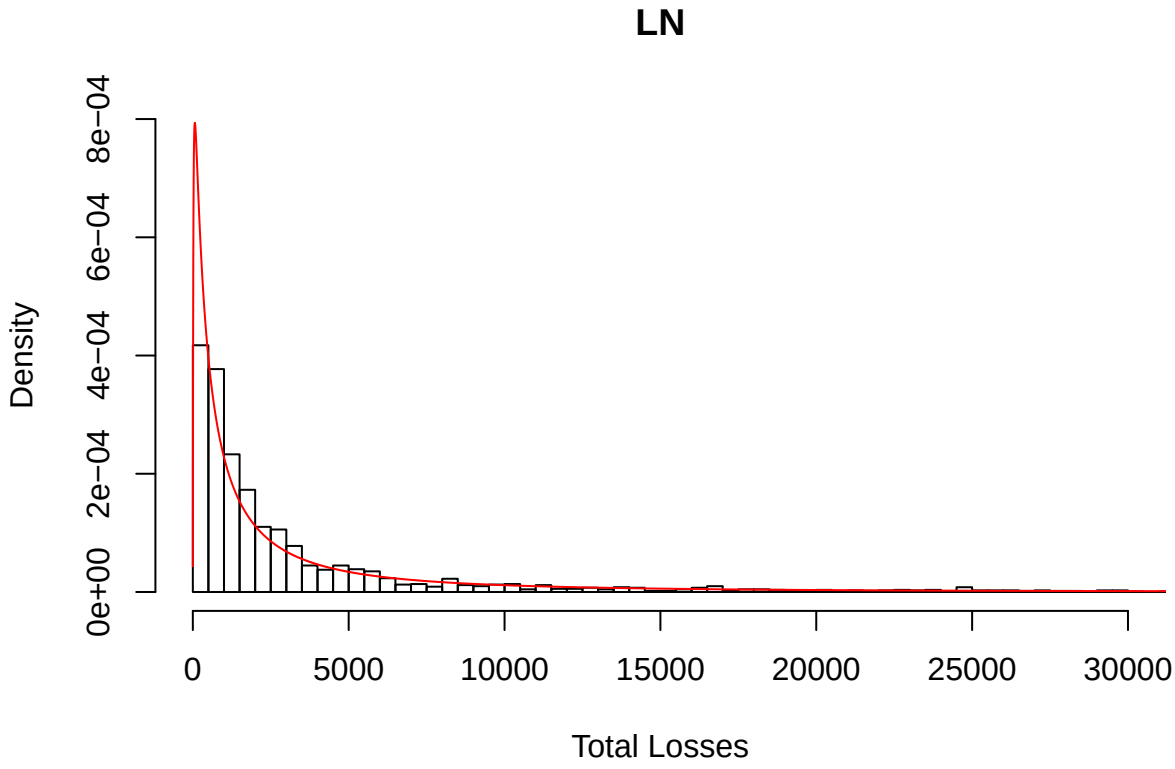
7.3.4 Histogram

prepare the display window parameters to properly fit the histograms

```
par(mfrow=c(1,2))
```

7.3.4.1 LN

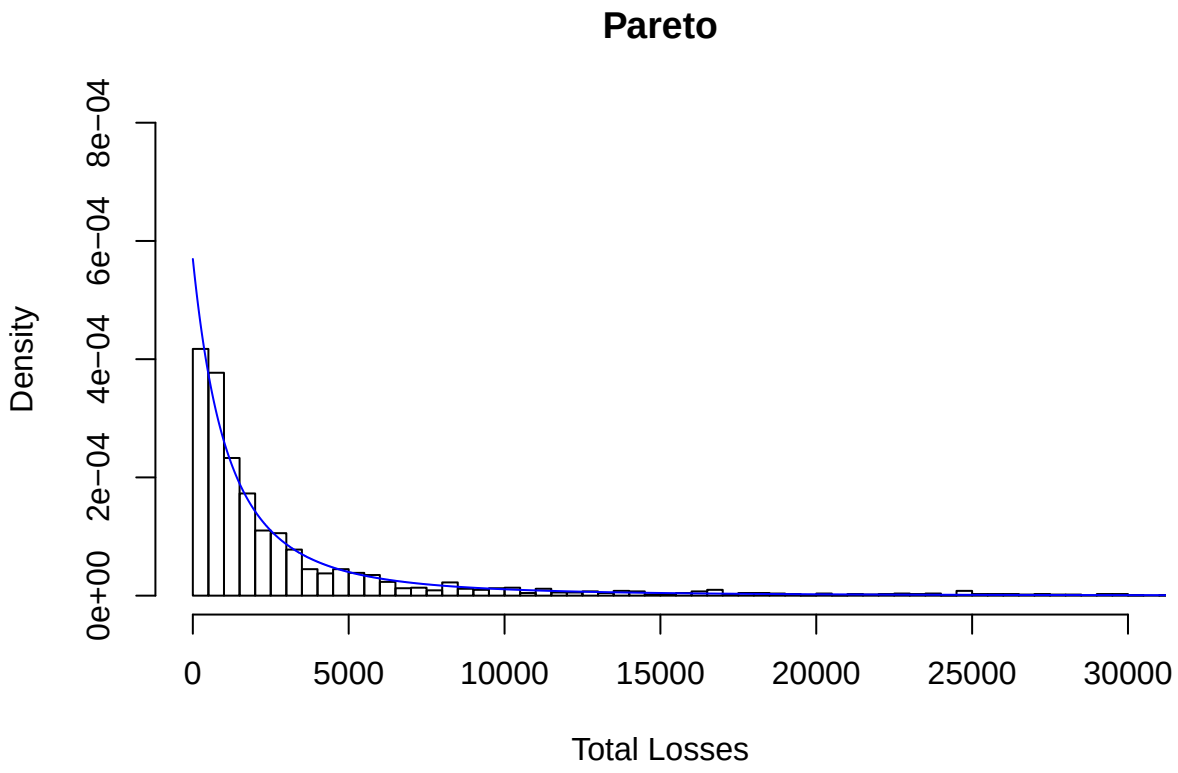
```
hist(sev.dat$Loss,xlab="Total Losses",main="LN",breaks=100,freq=F,xlim=c(0,3e4),ylim=c(0,8e-4))
x <- seq(1,max(sev.dat$Loss),1)
mu <- zop.LN$par[1]
sigma <- zop.LN$par[2]
lines(x,dlnorm(x,mu,sigma),col="red")
```



7.3.4.2 Pareto

```
hist(sev.dat$Loss,xlab="Total Losses",main="Pareto",breaks=100,freq=F,xlim=c(0,3e4),ylim=c(0,8e-4))
x <- seq(1,max(sev.dat$Loss),1)
theta <- zop.P$par[1]
```

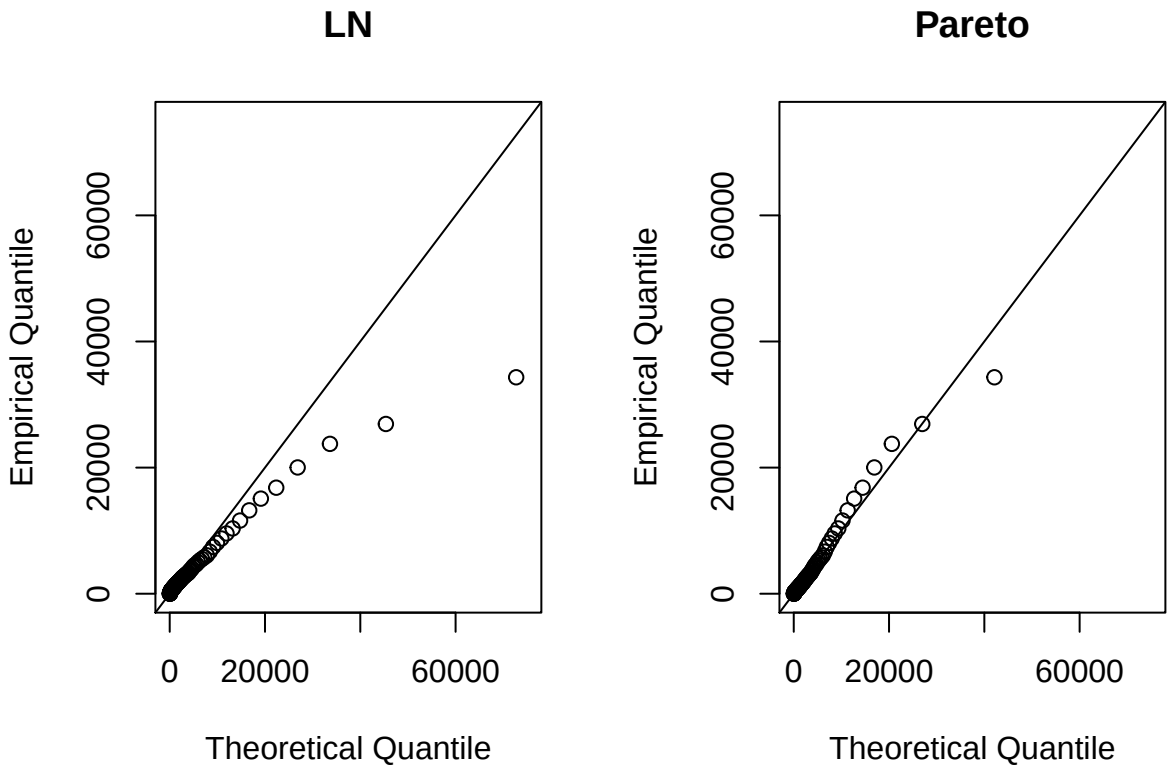
```
alpha <- zop.P$par[2]
lines(x,dpareto(x,theta,alpha),col="blue")
```



7.3.5 qq Plots

7.3.5.1 Define Quantile Function of Pareto

```
qpareto <- function(p,theta,alpha){theta*((1-p)^(-1/alpha)-1)}
pct <- seq(0.01,0.99,0.01)
par(mfrow=c(1,2))
plot(qlnorm(pct,mu,sigma),quantile(sev.dat$Loss,probs=pct),
     main="LN", xlab="Theoretical Quantile", ylab="Empirical Quantile",
     xlim=c(0,7.5e4),ylim=c(0,7.5e4))
abline(0,1)
plot(qpareto(pct,theta,alpha),quantile(sev.dat$Loss,probs=pct),
     main="Pareto", xlab="Theoretical Quantile", ylab="Empirical Quantile",
     xlim=c(0,7.5e4),ylim=c(0,7.5e4))
abline(0,1)
```


LN
Empirical Quantile
0
20000
40000
60000
0
20000
60000
Theoretical Quantile
Pareto
Empirical Quantile
0
20000
40000
60000
0
20000
60000
Theoretical Quantile

Chapter 8

Tweedie

*This file contains illustrative **R** code for the Tweedie distribution. When reviewing this code, you should open an **R** session, copy-and-paste the code, and see it perform. Then, you will be able to change parameters, look up commands, and so forth, as you go.*

8.1 Tweedie distribution

8.1.1 Load Tweedie Package

First bring in the package Tweedie (you may need to first install this package).

```
library(tweedie)
```

8.1.2 Set Paramteres for Tweedie(p,mu,phi)

Setting parameters p, mu and phi defines the specific features of the distribution.

Furthermore, setting a specific seed allows us to generate the same randomn numbers so we can produce identical distributions

```
set.seed(123)
p <- 1.5
mu <- exp(1)
phi <- exp(1)
```

8.1.3 Set Sample Size

Sample size is set to 500 for this example. "y" holds all 500 obserations from tweedie distribution with the given parameters.

```
n <- 500
y <- rtweedie(n,p,mu,phi)
```

8.1.4 Show Summary Statistics

Here we calculate important statisitics like mean, median, standard deviation and quantiles.

```
summary(y)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   1.438   2.687   3.878  24.181
```

```
sd(y)
```

```
[1] 3.346954
```

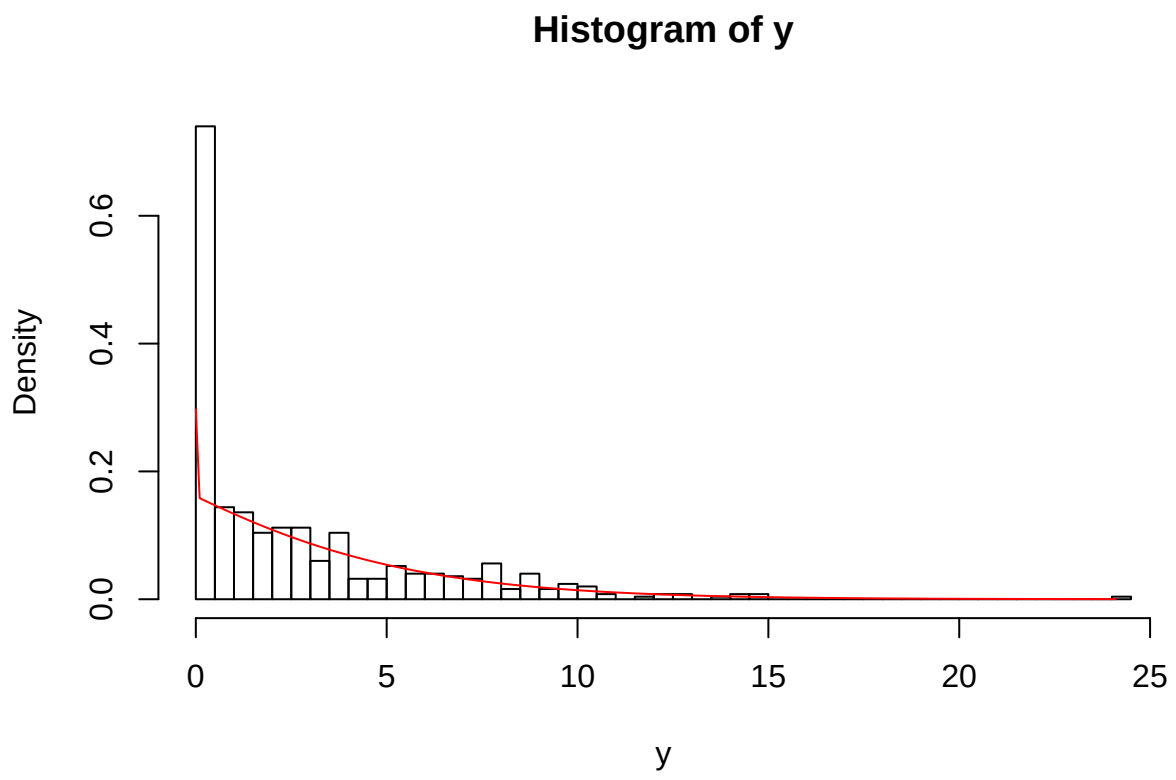
```
quantile(y,seq(0,1,0.1))
```

```
        0%        10%        20%        30%        40%        50%
 0.0000000  0.0000000  0.0000000  0.0000000  0.7275496  1.4378933
        60%        70%        80%        90%       100%
 2.3767214  3.4212150  5.2317625  7.7471281 24.1813833
```

8.1.5 Show Histogram

Histograms are useful for visually interpreting data. Sometime summary statistics aren't enough to see the full picture.

```
hist(y, prob=T,breaks=50)
x <- seq(0,max(y),0.1)
lines(x,dtweedie(x,p,mu,phi),col="red")
```



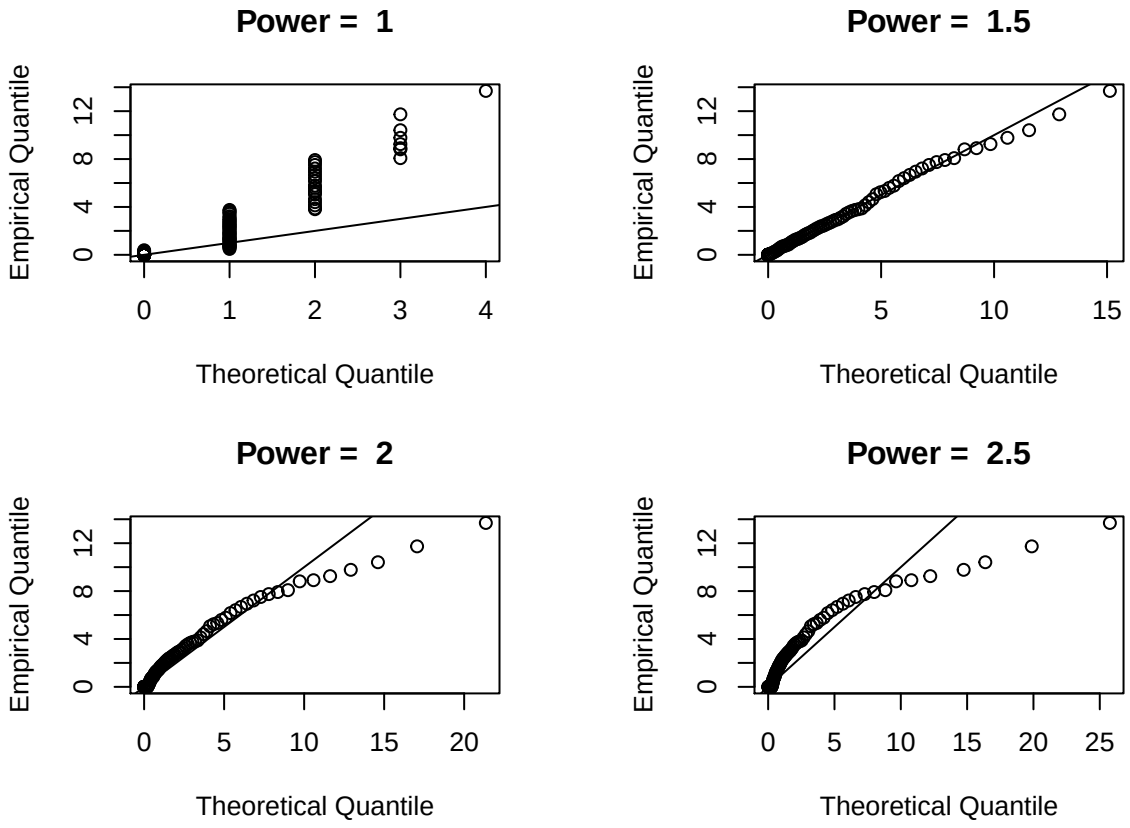
8.1.6 QQ Plots for Different p Values

A QQ plot is a plot of the quantiles of the first data set against the quantiles of the second data set.

This is graphical technique for determining if two data sets come from populations with a common distribution.

It appears here that a power of 1.5 matches the distribution best.

```
par(mfrow=c(2,2),mar=c(4,4,4,4))
qqTweedie <- function(xi,pct,mu,phi) {
  plot(qtweedie(pct,xi,mu,phi),quantile(y,probs=pct),
       main=paste("Power = ",xi), xlab="Theoretical Quantile", ylab="Empirical Quantile")
  abline(0,1)
}
pct <- seq(0.01,0.99,0.01)
lapply(c(1,1.5,2,2.5),qqTweedie,pct=pct,mu=mu,phi=phi)
```




```
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

[[4]]
```

```
NULL
```

8.1.7 Fit Tweedie Distribution

Here we run a "glm" for the Tweedie distribution. you may need to first install the "statmod" package

```
library(statmod)
fit <- glm(y~1,family=tweedie(var.power=1.5,link.power=0))
summary(fit)
```

```
Call:
glm(formula = y ~ 1, family = tweedie(var.power = 1.5, link.power = 0))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5607  -2.5607  -0.6876   0.5155   5.1207

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.9885     0.0557   17.75   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Tweedie family taken to be 2.542861)

    Null deviance: 1618.8  on 499  degrees of freedom
Residual deviance: 1618.8  on 499  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 4
```

8.1.8 Show Parameter Estimates

We now display the parameter estimates calculated in the glm.

```
summary(fit)$coefficient
```

```
             Estimate Std. Error  t value    Pr(>|t|)
(Intercept) 0.9885415  0.0556989 17.74795 5.42159e-55
```

```
summary(fit)$dispersion
```

```
[1] 2.542861
```

8.1.9 Maximum Likelihood Estimation

Here we run a "MLE" to determine the most likely parameters of the Tweedie distribution.

```
loglik<-function(parms){
  p=parms[1]
  mu=exp(parms[2])
  phi=exp(parms[3])
  llk <- -sum(log(dtweedie(y, p, mu, phi)))
```

```
  llk
}
ini <- c(1.5,1,1)
zop <- nlminb(ini,loglik, lower =c(1+1e-6,-Inf,-Inf),upper =c(2-1e-6,Inf,Inf))
print(zop)
```

```
$par
[1] 1.4823346 0.9885411 0.9871154

$objective
[1] 1122.992

$convergence
[1] 0

$iterations
[1] 14

$evaluations
function gradient
      23       77

$message
[1] "relative convergence (4)"
```

8.1.10 Obtain Standard Error

Now we calculate the standard errors of our parameter estimates from the MLE. You may need to first install the "numDeriv" package.

```
library(numDeriv)
est <- zop$par
names(est) <- c("p","mu","phi")
hess<-hessian(loglik,est)
se <-sqrt(diag(solve(hess)))
print(cbind(est,se))
```

```
          est         se
p   1.4823346 0.02260226
mu  0.9885411 0.05672086
phi 0.9871154 0.05060983
```

Chapter 9

Bootstrap Estimation

This file demonstrates both empirical and parametric boostrap simulation. When reviewing this code, you should open an ***R*** *session, copy-and-paste the code, and see it perform. Then, you will be able to change parameters, look up commands, and so forth, as you go.*

9.1 Empirical Bootstrap

Example: 90% confidence interval for the mean. Consider outcomes of rolling a fair die.

9.1.1 Random sample

Here we input a sample of 10 observations, which we are going to build our estimates off of.

```
y <- c(1,3,2,5,4,5,5,6,6,6)
n <- length(y)
n
```

```
[1] 10
```

9.1.2 Sample mean

Finding the mean of the random sample.

```
xbar <- mean(y)
xbar
```

```
[1] 4.3
```

9.1.3 Random resamples from y

9.1.3.1 Set bootstrap sample size

We're going to generate 30 different observations using the original sample data, called the *boostrap sample*

```
nboot <- 30
tmpdata <- sample(y,n*nboot,replace=TRUE)
bootstrap.sample <- matrix(tmpdata,nrow=n,ncol=nboot)
```

9.1.3.2 Compute sample mean for each bootstrap sample

Here we find the mean for all 30 of our boostrap samples.

```
bsmeans <- colMeans(bootstrap.sample)
bsmeans
```

```
 [1] 3.8 4.8 4.5 4.4 4.4 4.3 4.5 4.5 5.2 4.6 3.5 4.1 4.6 4.3 4.5 4.9 4.6
[18] 4.0 4.8 4.0 4.5 4.0 4.8 4.3 4.2 4.6 3.3 3.4 4.4 3.6
```

9.1.4 90% confidence interval

Using the generated boostrap sample, we calculate a 90% confidence interval for our sample mean.

```
CI <- c(quantile(bsmeans,prob=0.05), quantile(bsmeans,prob=0.95))
CI
```

```
   5%   95%
3.445 4.855
```

9.2 Parametric Bootstrap

Example: confidence interval for 1/theta. Consider y ~ exponential(theta=10).

9.2.1 random sample of size 250

This time we generate a random sample of 250 observations from the exponential sampling distribution.

```
y <- rexp(250,rate=0.1)
n <- length(y)
n
```

```
[1] 250
```

9.2.2 The MLE for lambda: 1/xbar

Remember from earlier that the MLE of theta for an exponential distribution is its mean.

```
theta.mle <- mean(y)
rate.hat <- 1/theta.mle
```

9.2.3 Generate bootstrap samples

Using the MLE we calculated we now generate 500 *bootstrap samples* of 250 observations each.

```
nboot <- 500
tmpdata <- rexp(n*nboot,rate=rate.hat)
bootstrap.sample <- matrix(tmpdata,nrow=n,ncol=nboot)
```

9.2.4 compute botstrap statistics

We now find the rate parameter for each of our *boostrap samples*

```
rate.star <- 1/colMeans(bootstrap.sample)
```

9.2.5 calculate deviation from sample statistics

Subtracting the original sample rate parameter from each of our simulated rate parameters, we can find the 5th and 95th percentiles of the differences.

```
delta.star <- rate.star - rate.hat
delta.lb <- quantile(delta.star,prob=0.05)
delta.ub <- quantile(delta.star,prob=0.95)
```

9.2.6 90% confidence intervel

We use the percentiles to create a 90% CI for the rate parameter.

```
CI <- rate.hat - c(delta.ub,delta.lb)
print(CI)
```

```
       95%         5%
0.08817793 0.10884260
```