Operational Risk

Anna Zalewska

June 12, 2010

Contents

Chapter 1

Introduction

Operational risk is defined by *The Basel Committee on Banking Supervision* [1] as *the risk of loss resulting from inadequate or failed internal processes, people and systems or from external events; this definition includes legal risk, but excludes strategic and reputational risk.* In *Basel* document we have three methods for calculating operational risk presented: *The Basic Indicator Approach*, *The Standarised Approach* and *Advanced Measurement Approach.* It is stated that banks *are encouraged to move along the spectrum of available approaches as they develop more sophisticated operational risk measurement systems and practices.* This paper aims to present some techniques for computing operational risk that could perhaps be useful for bank applying *Advanced Measurement Approach.* Business lines and risk categories classification will be the same as in *Annexes 8* and *9* of *Basel II.* [2]

1.1 Data

The example loss data is included into `opVaR` package. It is an R list consisting of 5 elements. Let us see some simple summary of that list:

```
> data(loss.data.object)
> summary(loss.data.object)

         Length Class      Mode
losses   4      data.frame list
risk     2      data.frame list
rcateg   7      -none-     character
business 2      data.frame list
blines   8      -none-     character
```

[1]See http://www.bis.org/ and http://www.bis.org/publ/bcbsca.htm for more information about *Basel II* (provided by *BIS* - Bank for International Settlements).

[2]See http://www.bis.org/publ/bcbs128.htm, *Basel II*, June 2006, *Annexes.*

Two elements not being data.frames, namely `rcateg` and `blines`, are names of risk categories and business lines.

```
> loss.data.object$rcateg

[1] "Business Disruption and System Failures"
[2] "Clients, Products & Business Practices"
[3] "Damage to Physical Assets"
[4] "Employment Practices and Workplace Safety"
[5] "Execution, Delivery & Process Management"
[6] "External Fraud"
[7] "Internal Fraud"

> loss.data.object$blines

[1] "Agency Services"      "Asset Management"     "Commercial Banking"
[4] "Corporate Finance"    "Payment & Settlement" "Retail Banking"
[7] "Retail Brokerage"     "Trading & Sales"
```

Two other elements, `loss.data.object$risk` and `loss.data.object$business`, are intended to assign some numbers to risk categories and business lines. As it can be seen, it has connection with some more detailed losses' division because there is more than one number corresponding to given risk category or business line. For example:

```
> loss.data.object$business[loss.data.object$business[,2] == "Retail Banking",]

    Internal_BL_ID Path_Component_1
58             650   Retail Banking
59             651   Retail Banking
60             652   Retail Banking
61             653   Retail Banking
62             654   Retail Banking
294         500816   Retail Banking
295         510817   Retail Banking
296         510818   Retail Banking
297         520819   Retail Banking
298         520820   Retail Banking
299         520821   Retail Banking
300         520822   Retail Banking
301         520823   Retail Banking
302         530824   Retail Banking
303         530825   Retail Banking
304         530826   Retail Banking
305         530827   Retail Banking
306         540828   Retail Banking
307         540829   Retail Banking
308         540830   Retail Banking
309         540831   Retail Banking
```

That finds all numbers corresponding to business line `"Retail Banking"`. All that Internal_BL_ID numbers are `"Retail Banking"` line numbers.

Let us see first six rows of `loss.data.object$losses`:

```
> head(loss.data.object$losses)
```

```
  Internal_BL_ID Internal_RC_ID First_Date_of_Event Gross_Loss_Amount
1         240401            183          2002-01-03           1642.26
2         570946             77          2002-01-06           2498.33
3          20105             54          2002-01-08           7420.72
4          20107             57          2002-01-09          27019.26
5         300424             95          2002-01-10           1829.98
6          20105             92          2002-01-11          12164.67
```

We can ask to which business line and risk category that loss from first row is assigned.

```
> loss.data.object$risk[loss.data.object$risk[,1] ==183,]
```

```
    Internal_RC_ID Path_Component_1
182            183   Internal Fraud
```

So risk category is `"Internal Fraud"` ...

```
> loss.data.object$business[loss.data.object$business[,1] == 240401,]
```

```
    Internal_BL_ID   Path_Component_1
179         240401 Commercial Banking
```

...and business line is `"Commercial Banking"`.

1.2 Note

For real data estimation we should have full periods and the fact is that we do not have them. All performed estimations are in fact estimations for about-four-years period between `"2002-01-03"` and `"2006-02-12"`.

Chapter 2

Data Reading

Let us start with data processing. We want to have that data presented before classified by business lines and risk categories. We will use `read.loss()` function. This function takes as outputs business line number, risk category number and our list `loss.data.object`. Let us have business line number 1 and risk category number 2.

```
>  loss.data.object$blines[1]

[1] "Agency Services"

> loss.data.object$rcateg[2]

[1] "Clients, Products & Business Practices"
```

Function value is `x12`:

```
>  x12<- read.loss(b=1,r=2,loss.data.object)
> head(x12)

   First_Date_of_Event Gross_Loss_Amount
3           2002-01-08           7420.72
4           2002-01-09          27019.26
6           2002-01-11          12164.67
8           2002-01-16           4983.20
9           2002-01-16           5894.24
10          2002-01-16           3162.60

>  dim(x12)

[1] 806   2
```

Function `read.loss()` reads losses (dates and amounts).

2.1 Loss summaries matrix

It could be also convenient to have matrix of losses summaries, `D`. Note that `D[i,1,j]` is maximum loss, `D[i,2,j]` mean loss, `D[i,3,j]` minimum loss, `D[i,4,j]` number of losses (`i` being number of business line and `j` number of risk category respectively).

```
> D <- loss.matrix(loss.data.object)
> D

, , 1

     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
[4,]    0    0    0    0
[5,]    0    0    0    0
[6,]    0    0    0    0
[7,]    0    0    0    0
[8,]    0    0    0    0

, , 2

           [,1]      [,2]    [,3] [,4]
[1,] 1790529.75 47079.516  794.22  806
[2,]       0.00     0.000    0.00    0
[3,]   59600.88  4428.156  891.80  285
[4,]       0.00     0.000    0.00    0
[5,]       0.00     0.000    0.00    0
[6,]       0.00     0.000    0.00    0
[7,] 1181407.58 19857.792 1003.45  459
[8,]       0.00     0.000    0.00    0

, , 3

          [,1]      [,2]    [,3] [,4]
[1,]      0.00     0.000    0.00    0
[2,]      0.00     0.000    0.00    0
[3,]      0.00     0.000    0.00    0
[4,]  88421.37  5601.253 1028.65  128
[5,] 214527.15 14215.301  969.82  328
[6,]      0.00     0.000    0.00    0
[7,]      0.00     0.000    0.00    0
[8,] 274861.47  7164.078 1004.44  192

, , 4
```

```
          [,1]     [,2]    [,3] [,4]
[1,]      0.00     0.00    0.00    0
[2,]      0.00     0.00    0.00    0
[3,]  32229.74  4248.81 1009.01   61
[4,]      0.00     0.00    0.00    0
[5,] 143152.28 16925.76  974.28  137
[6,]      0.00     0.00    0.00    0
[7,]      0.00     0.00    0.00    0
[8,]      0.00     0.00    0.00    0

, , 5

           [,1]      [,2]    [,3] [,4]
[1,]       0.00     0.000    0.00    0
[2,] 2222723.89 91837.982 1008.43  139
[3,]       0.00     0.000    0.00    0
[4,]   27339.64  3715.261  951.42   38
[5,]   80946.39 11526.309 1005.58  126
[6,]   59086.99 10369.983 1083.44   15
[7,]  214323.74 11388.459  866.25  119
[8,]       0.00     0.000    0.00    0

, , 6

           [,1]      [,2]    [,3] [,4]
[1,] 2272729.37 64208.694  890.15  102
[2,]       0.00     0.000    0.00    0
[3,]   57369.05  6624.523 1066.77   54
[4,]   46101.82  5882.500 1095.27   32
[5,]       0.00     0.000    0.00    0
[6,]       0.00     0.000    0.00    0
[7,]       0.00     0.000    0.00    0
[8,]       0.00     0.000    0.00    0

, , 7

           [,1]       [,2]    [,3] [,4]
[1,]  310442.89  32131.851 1019.38   80
[2,] 3875031.69 132019.721 1151.49  173
[3,]   13110.72   2985.808 1005.67   64
[4,]       0.00      0.000    0.00    0
[5,]  144495.59  16282.654 1024.10  125
[6,]   29386.38   6080.325 1177.93   20
[7,]  111428.79  11412.683 1029.68  126
[8,]       0.00      0.000    0.00    0
```

It could be also useful to have it all in a table. Let us use `loss.matrix.image()` function:

```
> loss.matrix.image(data = loss.data.object)
```

Figure 2.1: Matrix of loss summaries

It could be also:

```
> loss.matrix.image(D,loss.data.object$blines,loss.data.object$rcateg)
```

It is clear now that there are no losses that comes from business line `"Business Disruption and System Failures"` and any of risk categories.
In `"Clients, Products & Business Practices"` and `"Agency Services"` cell we have 806 losses. Mean loss is equal to 47079.52, maximum loss is equal to 1790529.75. These values could be found in `D[1, ,2]` (and `D[1,3,2] = 794.22` is minimum loss there). This cell has `col3` colour which is intended to distinguish cells with mean loss more than or equal to mean loss for `D` plus standard deviation of mean loss for `D`.

2.2 Losses aggregation

We would like to have possibility of changing periods from `days` to `weeks` or `months` or even `quarters` and it is connected with loss aggregation. Let us see that on example:

```
> x65 <- read.loss(6,5,loss.data.object)
> dim(x65)

[1] 15  2
```

That data comes from 6th business line and 5th risk category. Using `dim()` function allows us to see numbers of `x65` rows and columns.

```
> x65

     First_Date_of_Event Gross_Loss_Amount
63            2002-02-20           6118.85
165           2002-04-23           7942.78
622           2003-01-21           3325.98
711           2003-03-06           3405.60
984           2003-07-01           1426.88
1211          2003-10-14           1083.44
1325          2003-12-03           3284.29
1444          2004-01-28           6855.69
1956          2004-08-09           1894.01
1957          2004-08-09          37248.23
2141          2004-10-19          13553.71
2249          2004-12-03           5690.19
2496          2005-03-02           1570.91
2500          2005-03-02           3062.20
3032          2005-08-23          59086.99
```

No aggregation at all:

```
> t0 <- period.loss(x65,"none"); t0

 [1]  6118.85  7942.78  3325.98  3405.60  1426.88  1083.44  3284.29  6855.69  1894.01
[10] 37248.23 13553.71  5690.19  1570.91  3062.20 59086.99
```

Losses merged by days . . .

```
> t1 <- period.loss(x65,"days"); t1

 [1]  6118.85  7942.78  3325.98  3405.60  1426.88  1083.44  3284.29  6855.69 39142.24
[10] 13553.71  5690.19  4633.11 59086.99
```

. . . by weeks . . .

```
>  t2 <- period.loss(x65,"weeks"); t2
```

```
 [1]  6118.85  7942.78  3325.98  3405.60  1426.88  1083.44  3284.29  6855.69 39142.24
[10] 13553.71  5690.19  4633.11 59086.99
```

. . . by months . . .

```
>  t3 <- period.loss(x65,"months"); t3
```

```
 [1]  6118.85  7942.78  3325.98  3405.60  1426.88  1083.44  3284.29  6855.69 39142.24
[10] 13553.71  5690.19  4633.11 59086.99
```

. . . and by quarters:

```
> t4 <- period.loss(x65,"quarters"); t4
```

```
 [1]  6118.85  7942.78  6731.58  1426.88  4367.73  6855.69 39142.24 19243.90  4633.11
[10] 59086.99
```

But sometimes we would like see dates as well; then it is reccomended to use `dts` ("dates") option.

```
> t0 <- period.loss(x65,"none",dts=TRUE); t0
```

```
2002-02-20 2002-04-23 2003-01-21 2003-03-06 2003-07-01 2003-10-14 2003-12-03
   6118.85    7942.78    3325.98    3405.60    1426.88    1083.44    3284.29
2004-01-28 2004-08-09 2004-08-09 2004-10-19 2004-12-03 2005-03-02 2005-03-02
   6855.69    1894.01   37248.23   13553.71    5690.19    1570.91    3062.20
2005-08-23
  59086.99
```

```
>  t1 <- period.loss(x65,"days",dts=T); t1
```

```
2002-02-20 2002-04-23 2003-01-21 2003-03-06 2003-07-01 2003-10-14 2003-12-03
   6118.85    7942.78    3325.98    3405.60    1426.88    1083.44    3284.29
2004-01-28 2004-08-09 2004-10-19 2004-12-03 2005-03-02 2005-08-23
   6855.69   39142.24   13553.71    5690.19    4633.11   59086.99
```

```
> t2 <- period.loss(x65,"weeks",dts=T); t2
```

```
2002-02-18 2002-04-22 2003-01-20 2003-03-03 2003-06-30 2003-10-13 2003-12-01
   6118.85    7942.78    3325.98    3405.60    1426.88    1083.44    3284.29
2004-01-26 2004-08-09 2004-10-18 2004-11-29 2005-02-28 2005-08-22
   6855.69   39142.24   13553.71    5690.19    4633.11   59086.99
```

```
>  t3 <- period.loss(x65,"months",dts=T); t3
```

```
2002-02-01 2002-04-01 2003-01-01 2003-03-01 2003-07-01 2003-10-01 2003-12-01
   6118.85    7942.78    3325.98    3405.60    1426.88    1083.44    3284.29
2004-01-01 2004-08-01 2004-10-01 2004-12-01 2005-03-01 2005-08-01
   6855.69   39142.24   13553.71    5690.19    4633.11   59086.99
```

```
> t4 <- period.loss(x65,"quarters",dts=T); t4

2002-01-01 2002-04-01 2003-01-01 2003-07-01 2003-10-01 2004-01-01 2004-07-01
   6118.85    7942.78    6731.58    1426.88    4367.73    6855.69   39142.24
2004-10-01 2005-01-01 2005-07-01
  19243.90    4633.11   59086.99
```

These are the same results as before but with dates. Note that only for days and no period there are original dates; for weeks, months and quarters there are only dates opening period in which loss occurred.

Chapter 3

Loss frequency

3.1 Loss histograms

We would like to fit loss frequency distribution. The function `hist.period()` might be used for plotting histograms of frequency of losses. Let us see these histogram for the fifth business line and risk category. What is important in this function is possibility to choose periods (losses could be aggregated by days, weeks, months or quarters). According to *Basel II*, *"A bank's risk measurement system must be sufficiently 'granular' to capture the major drivers of operational risk affecting the shape of the tail of the loss estimates"*.

```
> x55<- read.loss(5,5,loss.data.object)
```

The 5th risk category is called `"Execution, Delivery & Process Management"` and 5th business line is called `"Payment & Settlement"`.

```
> z<- {}
> par(mfrow=c(2,2))
>  z$days <- hist.period(x55,"days",col = "pink1")
>  z$weeks <- hist.period(x55,"weeks",col = "lightblue")
>  z$months <- hist.period(x55,"months",col = "khaki1" )
>  z$quarters <- hist.period(x55,"quarters",col = "lightgreen")
```

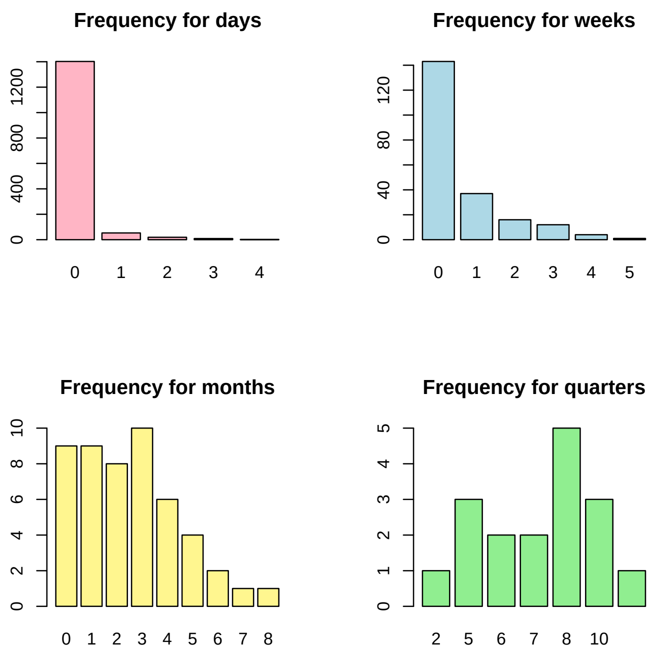


Figure 3.1: Frequency histograms for x55 data and different periods

For every histogram another colour is used.

```
> z

$days
$days$y

   0    1    2    3    4
1402   53   19    9    2


$weeks
$weeks$y
y
  0   1   2   3   4   5
143  37  16  12   4   1


$months
$months$y
```

```
y
 0  1  2  3  4  5  6  7  8
 9  9  8 10  6  4  2  1  1


$quarters
$quarters$y
y
 2  5  6  7  8 10 13
 1  3  2  2  5  3  1
```

As it can be easily found (checking `z$days$y`), there were `1402` days without losses, `53` with one loss, `19` with two, `9` with three and only `2` with four. There was no day with five losses. From `z$weeks$y` we learn that there was `143` weeks without losses, ...

Of course there is one crucial question connected with day losses aggregation: should be weekends counted or not?
Perhaps there can be no loss in given business line and risk category at weekends? Could we possibly check that?

Let us see that for our `x55`. Function `weekdays()` is function extracting the weekday; `x55[,1]` are dates, but `as.Date` must be used as well.

As we could easily see, there are some weekend days in our data. But there still could be some assymetry expected. A flooding does not distinguish between business and weekend days, but someone committing internal fraud can and perhaps that crime could be possible only in business days.
Let us check whether there are weekend days in our data:

```
> weekdays(as.Date(x55[,1]))

  [1] "Sunday"    "Thursday"  "Thursday"  "Thursday"  "Friday"
  [6] "Friday"    "Sunday"    "Wednesday" "Sunday"    "Wednesday"
 [11] "Thursday"  "Thursday"  "Tuesday"   "Sunday"    "Monday"
 [16] "Monday"    "Wednesday" "Saturday"  "Tuesday"   "Saturday"
 [21] "Thursday"  "Wednesday" "Thursday"  "Thursday"  "Monday"
 [26] "Monday"    "Monday"    "Thursday"  "Wednesday" "Wednesday"
 [31] "Saturday"  "Thursday"  "Friday"    "Sunday"    "Monday"
 [36] "Saturday"  "Tuesday"   "Sunday"    "Sunday"    "Thursday"
 [41] "Thursday"  "Tuesday"   "Tuesday"   "Sunday"    "Thursday"
 [46] "Thursday"  "Thursday"  "Thursday"  "Sunday"    "Sunday"
 [51] "Thursday"  "Friday"    "Friday"    "Monday"    "Thursday"
 [56] "Tuesday"   "Friday"    "Tuesday"   "Friday"    "Friday"
 [61] "Monday"    "Sunday"    "Sunday"    "Sunday"    "Saturday"
 [66] "Saturday"  "Thursday"  "Saturday"  "Saturday"  "Sunday"
 [71] "Saturday"  "Tuesday"   "Tuesday"   "Tuesday"   "Thursday"
 [76] "Thursday"  "Monday"    "Monday"    "Tuesday"   "Friday"
```

```
 [81] "Friday"    "Friday"    "Friday"    "Wednesday" "Tuesday"
 [86] "Friday"    "Friday"    "Sunday"    "Friday"    "Tuesday"
 [91] "Tuesday"   "Tuesday"   "Tuesday"   "Sunday"    "Saturday"
 [96] "Saturday"  "Saturday"  "Tuesday"   "Tuesday"   "Wednesday"
[101] "Friday"    "Sunday"    "Sunday"    "Tuesday"   "Saturday"
[106] "Saturday"  "Wednesday" "Wednesday" "Wednesday" "Tuesday"
[111] "Tuesday"   "Tuesday"   "Sunday"    "Thursday"  "Sunday"
[116] "Sunday"    "Sunday"    "Sunday"    "Friday"    "Wednesday"
[121] "Saturday"  "Monday"    "Wednesday" "Sunday"    "Sunday"
[126] "Sunday"
```

Yes, there are many of them. Maybe we could treat them as business days. But there is also one example without weekend days in our data set. That simple code will help us to prove it:

```
> for(i in 1:length(loss.data.object$blines)){
+ for(j in 1:length(loss.data.object$rcateg)){
+ y<- read.loss(b = i, r = j, loss.data.object)
+ if(dim(y)[1]!=0){
+ w<- weekdays(as.Date(y[,1]))
+ if(!is.element("Sundays", w)&!is.element("Saturday", w)){
+ print(paste(loss.data.object$blines[i], loss.data.object$rcateg[j],i,j))}
+ }
+ }}

[1] "Retail Banking Execution, Delivery & Process Management 6 5"
```

For given `i` and `j`, `y` are losses from `ith` business line and `jth` risk category. Then we check if there are any losses in y. If there are losses we change dates to week days and search for `"Saturday"` or `"Sunday"`. In case there are weekend days, names and numbers of risk category and business line are printed.
As we could easily see, there is only one cell in our `loss.matrix.image` table without weekend days. It is for 6th business line and 5th risk category. One look at that table tells that there are only 15 losses. It seems too little to decide whether there are possible losses in weekend days in that cell or not. If we could decide that answer is "no", we could use `wknd` option. It is intended for `period = "days"` only (of course it makes no sense to use it having other `period`).

```
> x65<- read.loss(b = 6, r = 5,loss.data.object)
> x65

     First_Date_of_Event Gross_Loss_Amount
63            2002-02-20           6118.85
165           2002-04-23           7942.78
622           2003-01-21           3325.98
711           2003-03-06           3405.60
984           2003-07-01           1426.88
```

```
1211           2003-10-14          1083.44
1325           2003-12-03          3284.29
1444           2004-01-28          6855.69
1956           2004-08-09          1894.01
1957           2004-08-09         37248.23
2141           2004-10-19         13553.71
2249           2004-12-03          5690.19
2496           2005-03-02          1570.91
2500           2005-03-02          3062.20
3032           2005-08-23         59086.99
```

Let us check dates:

```
> weekdays(as.Date(x65[,1]))

 [1] "Wednesday" "Tuesday"   "Tuesday"   "Thursday"  "Tuesday"   "Tuesday"
 [7] "Wednesday" "Wednesday" "Monday"    "Monday"    "Tuesday"   "Friday"
[13] "Wednesday" "Wednesday" "Tuesday"
```

So what is the difference between data histogram with and without weekend days in that case? How to choose the right one? That would perhaps be task for risk managers. We just see the results:

```
> h1 <- hist.period(x65,"days",col = "plum")$y
> h1

   0    1    2
1268   11    2
```

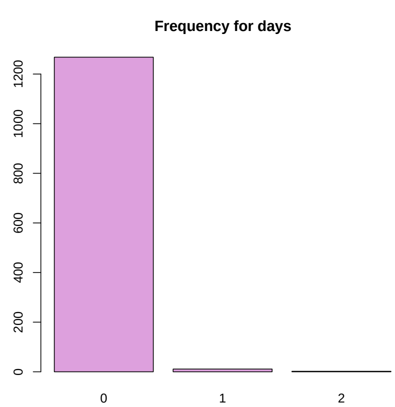


Figure 3.2: histogram for days for x65

```
> h2 <- hist.period(x65,"days",col = "pink1",wknd= F)$y
> h2
```

```
  0   1   2
902  11   2
```

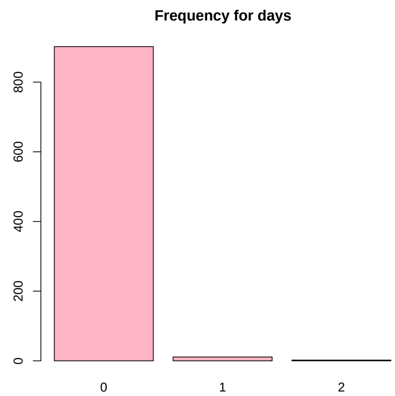


Figure 3.3: histogram for days for x65; wknd=F

Naturally, quantity of days with given positive number of losses does not differ regardless `wknd`. Let us have:

```
> dates<- as.Date(x65[,1])
> max(dates) - min(dates) +1
```

```
Time difference of 1281 days
```

Time difference is number of days between begin date and end date, including both begin and end.

Of course:

```
> sum(h1)
```

```
[1] 1281
```

. . . which is equal to `max(dates) - min(dates) +1`.

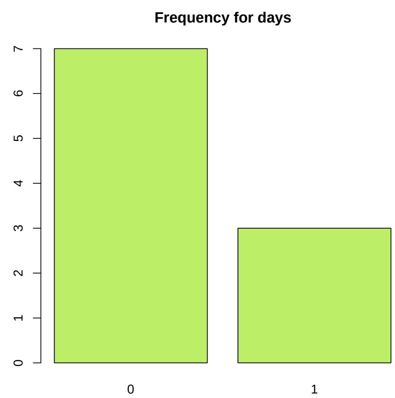
It seems a right time for a warning. Let us have some dates:

```
> new.dates <- c("2010-01-01", "2010-01-03", "2010-01-10")
> weekdays(as.Date(new.dates))
```

```
[1] "Friday" "Sunday" "Sunday"
```

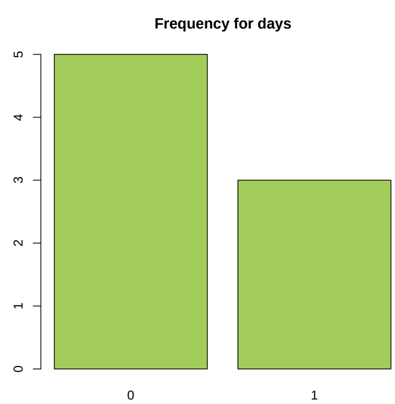
We bind that dates with whichever loss data (only because our data must be two-dimensional: dates and losses).

```
> a<- cbind(new.dates,1:3)
> hist.period(a,"days", col =  "darkolivegreen2")$y

0 1
7 3
```



That is correct: 7 days without losses and 3 with one; in total 10. We have `10` days (from `"2010-01-01"` to `"2010-01-10"`) and 7 of them is with no loss. Using `wknd = F` having some loss dates being weekend days (in that case these are `"2010-01-03"` and `"2010-01-10"`) is of course improper and results some further errors. In following example we have:

```
> hist.period(a,wknd=F, col =  "darkolivegreen3")$y

0 1
5 3
```

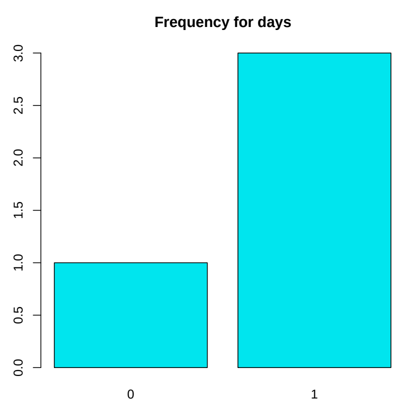


Days are default so there is no need to write it.
We have 8 in total and that is not correct because we should have 4 days excluded from 10:

```
> weekdays(seq(as.Date(new.dates[1]), length.out=10, by="1 day"))

 [1] "Friday"    "Saturday"  "Sunday"    "Monday"    "Tuesday"
 [6] "Wednesday" "Thursday"  "Friday"    "Saturday"  "Sunday"
```

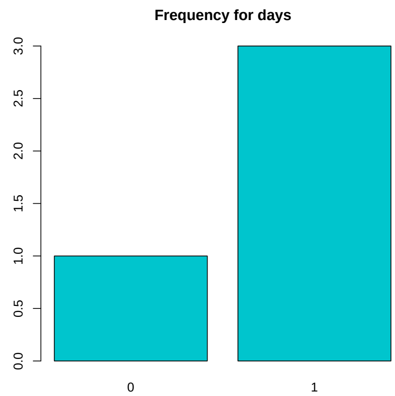
Those two losses in Sundays should not be counted, so result is wrong.
. . . and it could be even more irrational:

```
> new.dates2 <- c("2010-01-01", "2010-01-03","2010-01-04")
> weekdays(as.Date(new.dates2))

[1] "Friday" "Sunday" "Monday"

> b <- cbind(new.dates2,1:3)
> hist.period(b, col = "turquoise2")$y

0 1
1 3
```



That is of course correct, `wknd = TRUE`.

```
> hist.period(b,wknd=F,col = "turquoise3")$y

0 1
1 3
```



No difference despite `wknd` being `FALSE`!
From `"2010-01-01"` to `"2010-01-04"` we have 4 days, not counting weekends it makes only two days: Friday (one loss) and Monday (one loss) so there could not be 3 days with one loss and that makes contradiction.

All that shows that `wknd` must be used very carefully and perhaps that we should pay more attention to `weeks` to verify our theories. There are also `months` and `quarters` but as we can see, the shape of our histograms does not seem to be very regular - we have got less data. Let us check:

```
> min(as.Date(loss.data.object$losses[,3]))

[1] "2002-01-03"

> max(as.Date(loss.data.object$losses[,3]))

[1] "2006-02-12"
```

These are the earliest date in our dataset and the latest date.

So first quarter to appear is that with begin date `"2002-01-01"` and last is with begin date `"2006-01-01"` and that makes only `17` quarters.
One should be aware that `hist.period()` assumes that loss data was collected in complete periods i.e. in our case data was registered from `"2002-01-01"` to `"2006-01-01"` and that dates should be given as `begin` and `end` arguments. If not given, `begin` and `end` become first and last losses' dates and that is what happens in our case. We could repeat that argumentation with `months` - but of course there is more data then; and `weeks` seem rather reliable - maybe more than `days` with that `wknd` option (and `crt` option - correction for holidays, not used in this example).

3.2 Fitting loss frequency distributions

We would like to fit some frequency distributions to our data. Of course it would be better to have a large dataset for our example purpose, because fitting distribution having 3 points makes no sense and using 500 points seem more reliable. Let us see:

```
> D[,4,]
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    0  806    0    0    0  102   80
[2,]    0    0    0    0  139    0  173
[3,]    0  285    0   61    0   54   64
[4,]    0    0  128    0   38   32    0
[5,]    0    0  328  137  126    0  125
[6,]    0    0    0    0   15    0   20
[7,]    0  459    0    0  119    0  126
[8,]    0    0  192    0    0    0    0
```

That shows how many losses has given business line and given risk category. Of course we have maximum at `D[1,4,2]` (that means first business line and second risk category):

```
> max(D[,4,])
```

```
[1] 806
```

We will choose that line and that category.

3.2.1 "Agency Services"/"Clients, Products & Business Practices" cell

```
> x12<- read.loss(b=1,r=2,loss.data.object)
> dim(x12)
```

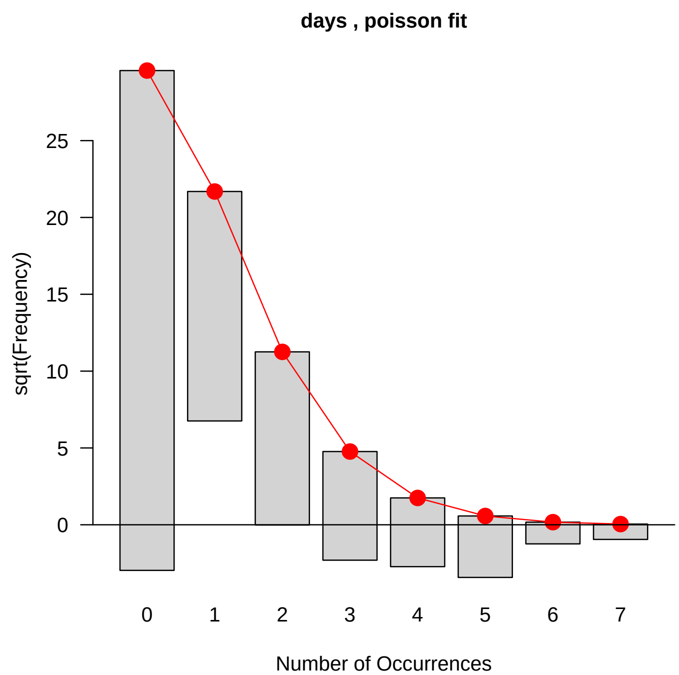
```
[1] 806   2
```

Poisson fit

We would like to fit `"poisson"` distribution to our data.

```
> y1<- root.period(x12,"days","poisson")
```

```
         Goodness-of-fit test for poisson distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 379.8419  6 6.012827e-79
```



> t <- y1$table; t

```
Observed and fitted values for poisson distribution
with parameters estimated by `ML'

 count observed       fitted
     0     1058 8.737622e+02
     1      223 4.704424e+02
     2      127 1.266455e+02
     3       50 2.272907e+01
     4       20 3.059391e+00
     5       16 3.294414e-01
     6        2 2.956243e-02
     7        1 2.273816e-03
```

It is a very poor fit and one can tell that just looking at the image. Let us see our function values:

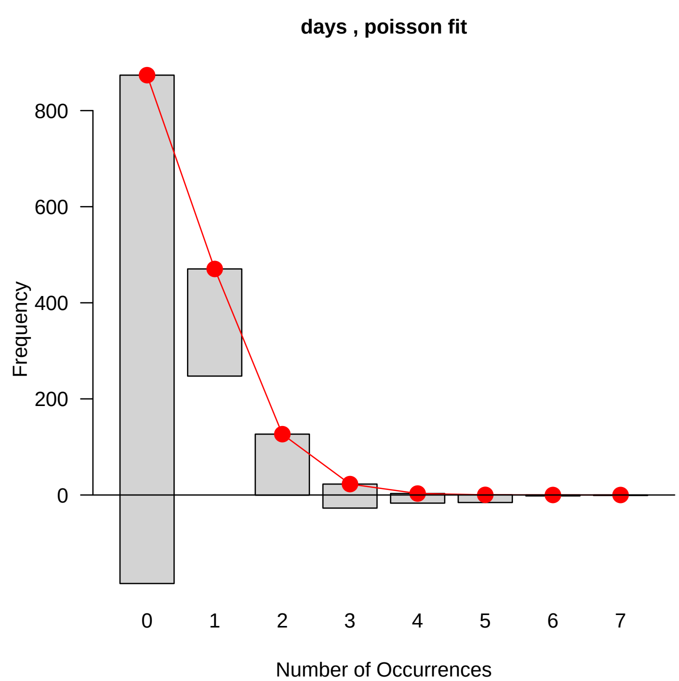
> par <- y1$param; par

```
$lambda
[1] 0.5384102
```

```
> p <- y1$p; p
[1] 6.012827e-79
```

It is clear that this distribution is indeed poorly fitted, because `p` is very small. And we must be aware of using `sqrt` scale and remember that it looks really like that:

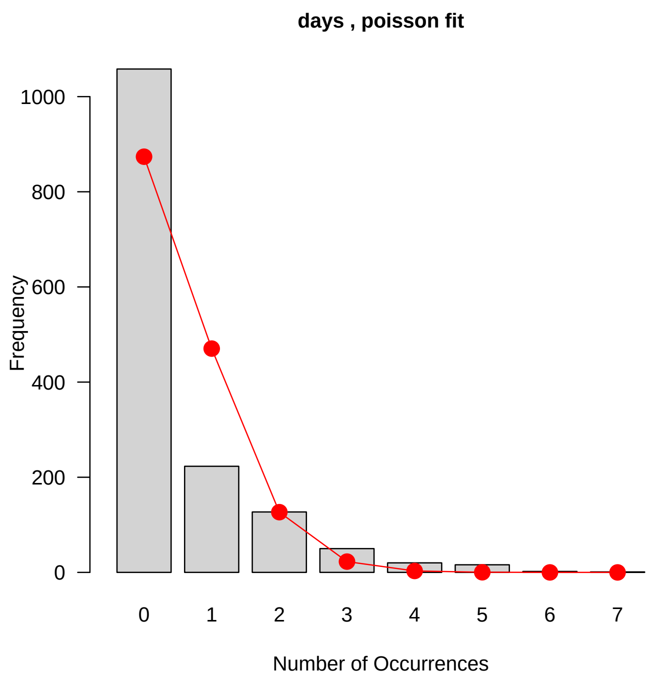
```
> y2 <- root.period(x12,"days","poisson",scale = "raw")

         Goodness-of-fit test for poisson distribution

                     X^2 df     P(> X^2)
Likelihood Ratio 379.8419  6 6.012827e-79
```



Note using `"raw"` scale!

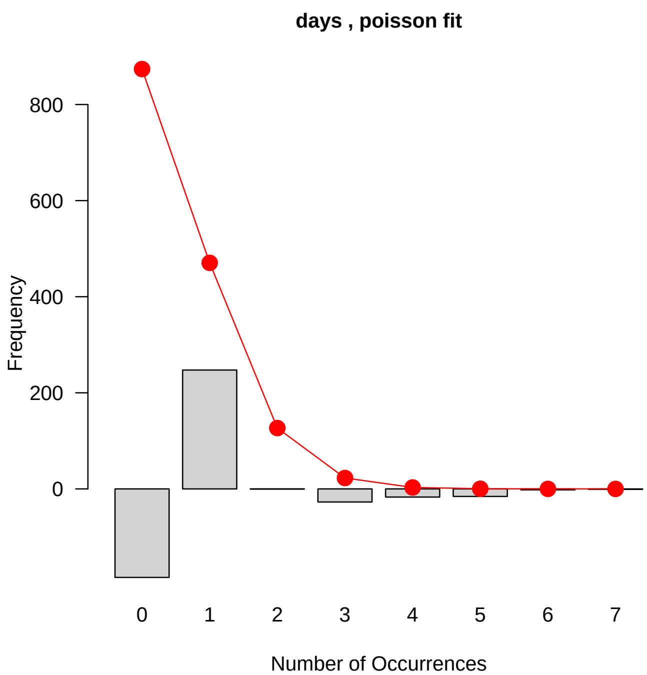
And now let us try another option:

```
> y3<-root.period(x12,"days","poisson",scale = "raw",bar = "standing")

         Goodness-of-fit test for poisson distribution

                     X^2 df     P(> X^2)
Likelihood Ratio 379.8419  6 6.012827e-79
```



That is plot with `raw` scale and `standings` bars (default is `hanging`).

```
> y4<-root.period(x12,"days","poisson",scale = "raw",bar = "deviation")

         Goodness-of-fit test for poisson distribution

                        X^2 df      P(> X^2)
Likelihood Ratio 379.8419  6 6.012827e-79
```



And that plot shows deviation.

Definitely, it is unreliable fit. Unfortunately, `"binomial"` fit is no better or even worse, looking at `p` value:

Binomial fit

```
> root.period(x12,"days","binomial")

         Goodness-of-fit test for binomial distribution

                      X^2 df      P(> X^2)
Likelihood Ratio 529.9351  6 2.983201e-111
$table

Observed and fitted values for binomial distribution
with parameters estimated by `ML'

 count observed       fitted
     0     1058 8.548920e+02
     1      223 4.986354e+02
     2      127 1.246460e+02
     3       50 1.731015e+01
```

```
     4       20 1.442363e+00
     5       16 7.211071e-02
     6        2 2.002868e-03
     7        1 2.384121e-05

$param
$param$prob
[1] 0.07691574

$param$size
[1] 7


$p
[1] 2.983201e-111
```

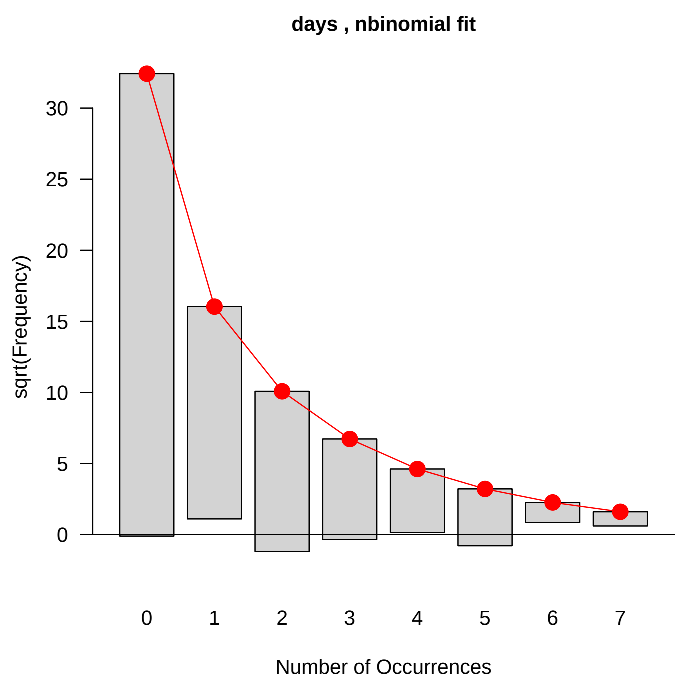
That warning message:
`warning("size was not given, taken as maximum count")`
comes from `goodfit` and it is always like that unless `size` is given; see help for `goodfit` for more details.

Negative binomial fit

And now we will try `"nbinomial"`:

```
> root.period(x12,"days","nbinomial")

         Goodness-of-fit test for nbinomial distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 23.01709  5 0.0003350356
$table

Observed and fitted values for nbinomial distribution
with parameters estimated by ‘ML’

 count observed      fitted
     0     1058 1051.085049
     1      223  257.118985
     2      127  101.585990
     3       50   45.231189
     4       20   21.273528
     5       16   10.325656
     6        2    5.115405
     7        1    2.570862

$param
```

```
$param$size
[1] 0.4483843

$param$prob
[1] 0.4544358


$p
[1] 0.0003350356
```



First of those warnings is:
`1: In dnbinom\mu(x, size, mu, log) : NaNs produced` and rest is the same; these are warnings from `goodfit()` (and `dnbinom()` in fact).

That `"nbinomial"` fit is better than `"poisson"` and `"binomial"` fits.
It is time to present well fitted distribution.

3.2.2 "Corporate Finance"/"Execution, Delivery & Process Management" cell

Let us have some other loss data:

```
> x45 <- read.loss(b=4,r=5,loss.data.object)
> dim(x45)
```

```
[1] 38  2
```

Let us check the best parameters.

```
> fit <- {}
> i = 1
> for(period in c("days","weeks","months","quarters")){
+ dist <- c("poisson","binomial","nbinomial")
+ p1 <- root.period(x45,period,"poisson")$p
+ p2 <- root.period(x45,period,"binomial")$p
+ p3 <- root.period(x45,period,"nbinomial")$p
+ p <- c(p1,p2,p3)
+ number <- which(p == max(p))
+ fit[[i]] <- paste(dist[number],period); i <- i +1
+ }
```

We have obtained `fit`: a list of distributions to choose. We need maximum `p` value. For every `period` value one distribution from `dist` is chosen.

```
> fit
```

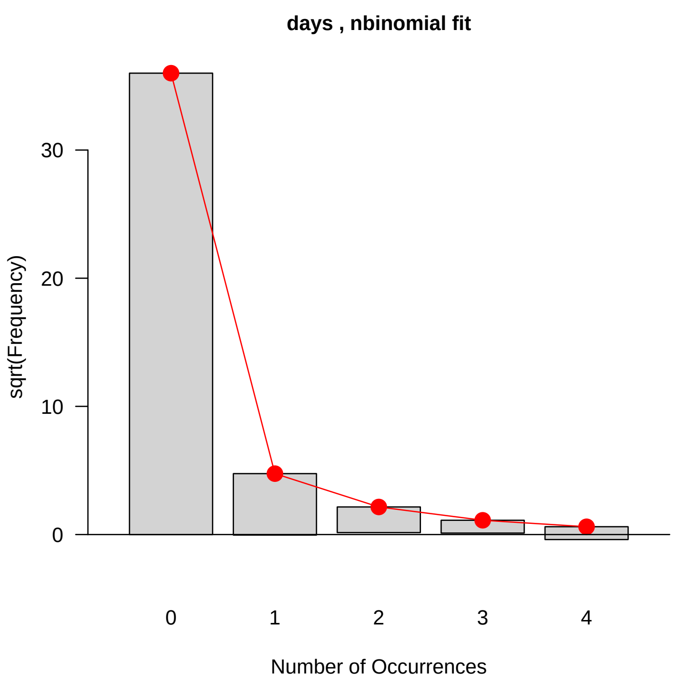
```
[1] "nbinomial days"   "nbinomial weeks"  "nbinomial months" "poisson quarters"
```

It is clear which distributions we should choose for our periods (`"nbinomial"` for `days`, `weeks`, `months`; `"poisson"` for `quarters`).
For days:

Days

```
> root.period(x45,"days","nbinomial")

         Goodness-of-fit test for nbinomial distribution

                      X^2 df P(> X^2)
Likelihood Ratio 1.227877  2 0.541215
$table

Observed and fitted values for nbinomial distribution
with parameters estimated by ‘ML’

 count observed       fitted
     0     1296 1296.0023507
     1       23   22.5832754
     2        4    4.6294474
     3        1    1.2384585
     4        1    0.3700255
```

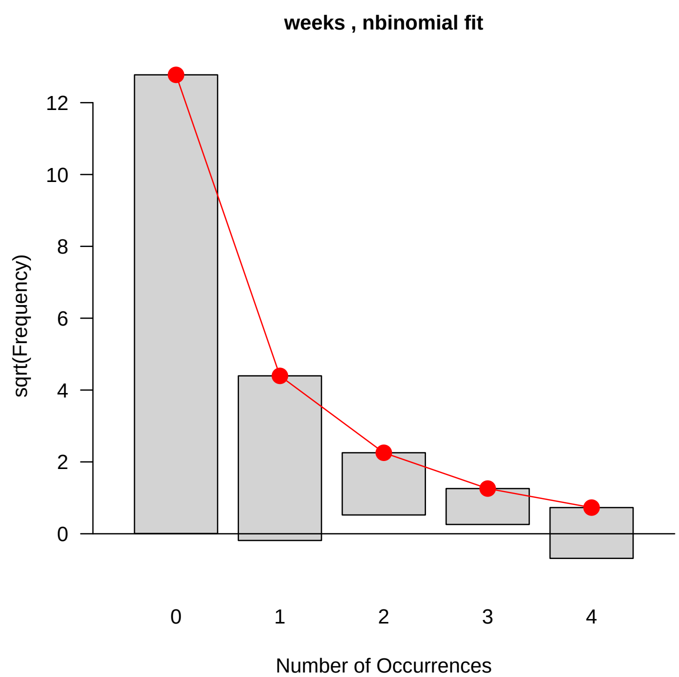
```
$param
$param$size
[1] 0.04438856

$param$prob
[1] 0.6074363


$p
[1] 0.541215
```



Weeks

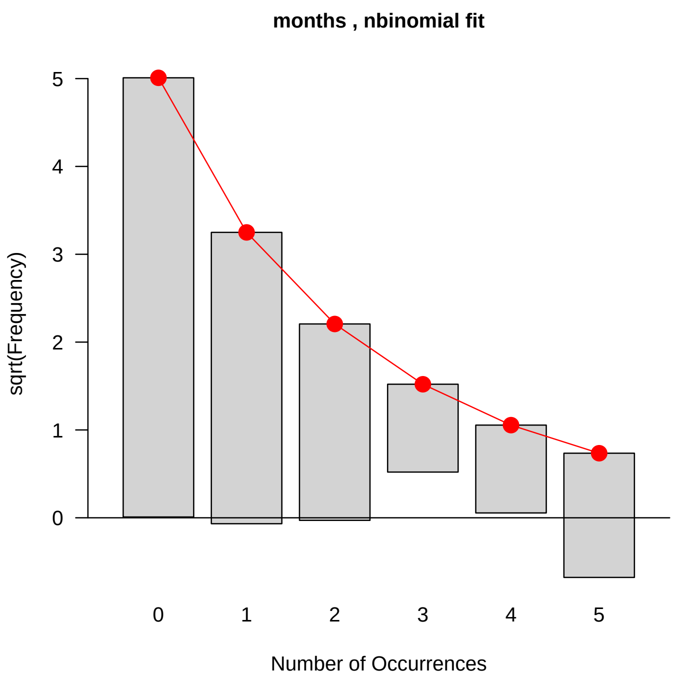
```
> root.period(x45,"weeks","nbinomial")

         Goodness-of-fit test for nbinomial distribution

                      X^2 df  P(> X^2)
Likelihood Ratio 4.342171  2 0.1140537
$table
```

```
Observed and fitted values for nbinomial distribution
with parameters estimated by `ML'

 count observed      fitted
     0      163 163.1849680
     1       21  19.3219847
     2        3   5.0854239
     3        1   1.5838878
     4        2   0.5315334

$param
$param$size
[1] 0.2902222

$param$prob
[1] 0.5920181


$p
[1] 0.1140537
```



Months

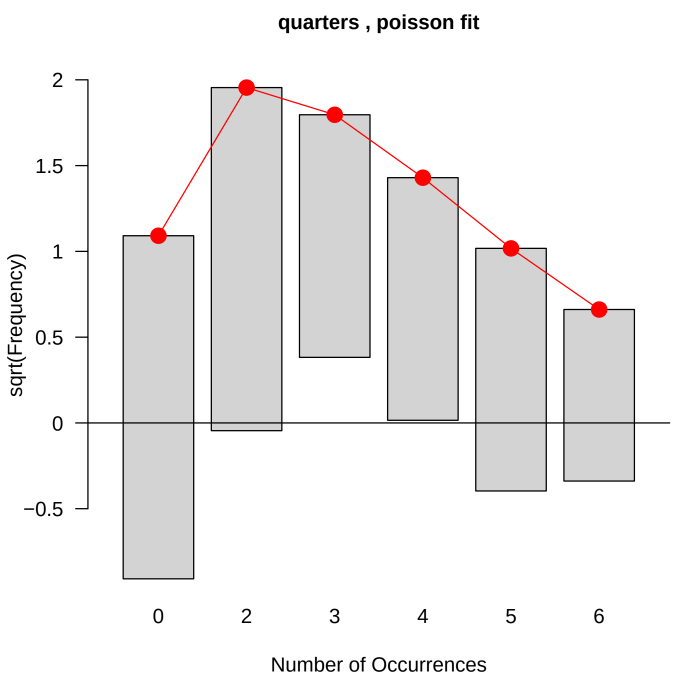
```
> root.period(x45,"months","nbinomial")

         Goodness-of-fit test for nbinomial distribution

                      X^2 df  P(> X^2)
Likelihood Ratio 4.333902  3 0.2275930
$table

Observed and fitted values for nbinomial distribution
with parameters estimated by `ML'

 count observed     fitted
     0       25 25.0902201
     1       11 10.5566972
     2        5  4.8692328
     3        1  2.3116422
     4        1  1.1130419
     5        2  0.5404305

$param
$param$size
[1] 0.838577

$param$prob
[1] 0.4982578


$p
[1] 0.2275930
```



Quarters

```
> root.period(x45,"quarters","poisson")

         Goodness-of-fit test for poisson distribution

                        X^2 df    P(> X^2)
Likelihood Ratio 12.34514  4 0.01496106
$table

Observed and fitted values for poisson distribution
with parameters estimated by `ML'

 count observed    fitted
     0        4 1.1909090
     2        4 3.8214946
     3        2 3.2270399
     4        2 2.0437919
     5        2 1.0355212
     6        1 0.4372201

$param
```

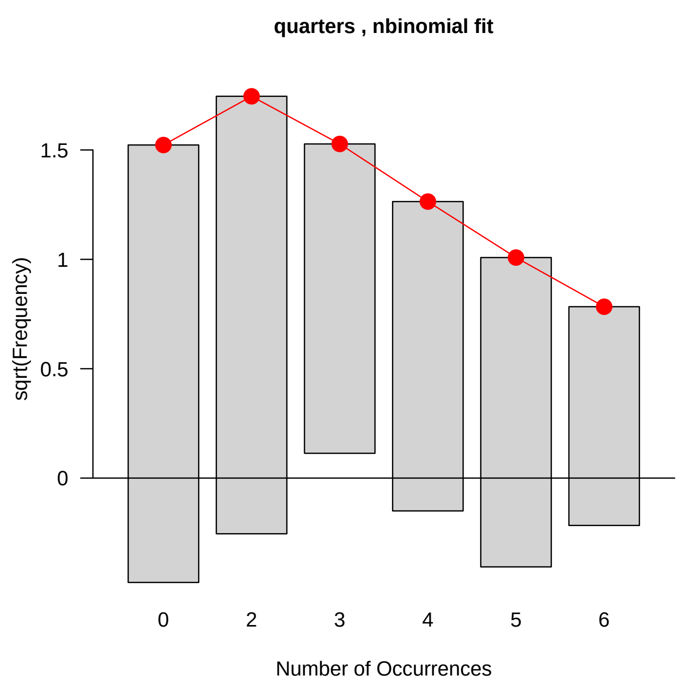
```
$param$lambda
[1] 2.533333


$p
[1] 0.01496106
```



That `p` value does not seem very big comparing to that for `days`, `weeks` or `months` but as we said quarters tend to be unreliable because there is not enough data and data should be given for full periods but in our case it is not.
In fact there is no big difference between `"poisson"` and `"nbinomial"` fits:

```
> root.period(x45,"quarters","nbinomial")

         Goodness-of-fit test for nbinomial distribution

                      X^2 df    P(> X^2)
Likelihood Ratio 10.50511  3 0.01472627
$table

Observed and fitted values for nbinomial distribution
with parameters estimated by `ML'
```

```
 count observed    fitted
     0        4 2.3188191
     2        4 3.0463495
     3        2 2.3334455
     4        2 1.5980008
     5        2 1.0165374
     6        1 0.6136513

$param
$param$size
[1] 3.206557

$param$prob
[1] 0.5586445


$p
[1] 0.01472627
```

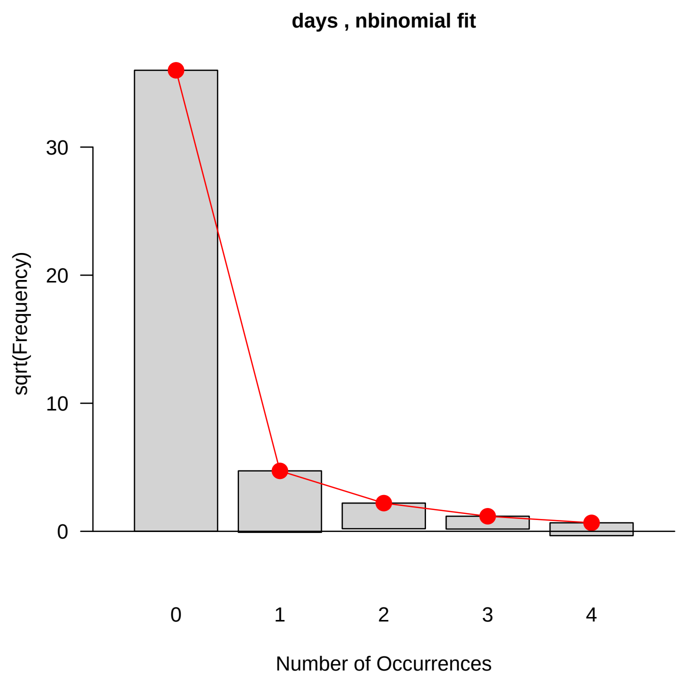


For Poisson `p` was equal to 0.01496106 and the difference is only 0.00023479.

"Maximum Likelihood" and "Minimum Chi-squared" methods

And just one more remark. Distributions are fitted via `"ML"` (i.e. Maximum Likelihood) or via `"MinChisq"` (i.e. Minimum Chi-squared) while `"ML"` is default. Let us see `x45` loss frequency fitted via `"MinChisq"` with `period` = `days`:

```
> root.period(x45,"days","nbinomial",method = "MinChisq")

         Goodness-of-fit test for nbinomial distribution

              X^2 df P(> X^2)
Pearson 0.4405477  2 0.802299
$table

Observed and fitted values for nbinomial distribution
with parameters estimated by ‘MinChisq’

 count observed       fitted
     0     1296 1295.8037516
     1       23   22.2756990
     2        4    4.8601571
     3        1    1.3860155
     4        1    0.4416922

$param
$param$size
[1] 0.04101081

$param$prob
[1] 0.5808266


$p
[1] 0.802299
```



`p` is greater than `p` obtained for `days` using `"ML"` and it is large difference, but `table` is almost identical. We have also some warnings.

Summarizing, `"nbinomial"` rather satisfactionary fits our `x45` data.

Chapter 4

Loss severity

Having frequency distribution we will try fit severity distribution too.

4.1 Density

First of all, let us plot density for some of our cells. There is `loss.density()` function, plotting all densities for a given risk category (business line) and all business lines (risk categories).

```
> loss.density( a=1,b=7,loss.data.object)
```

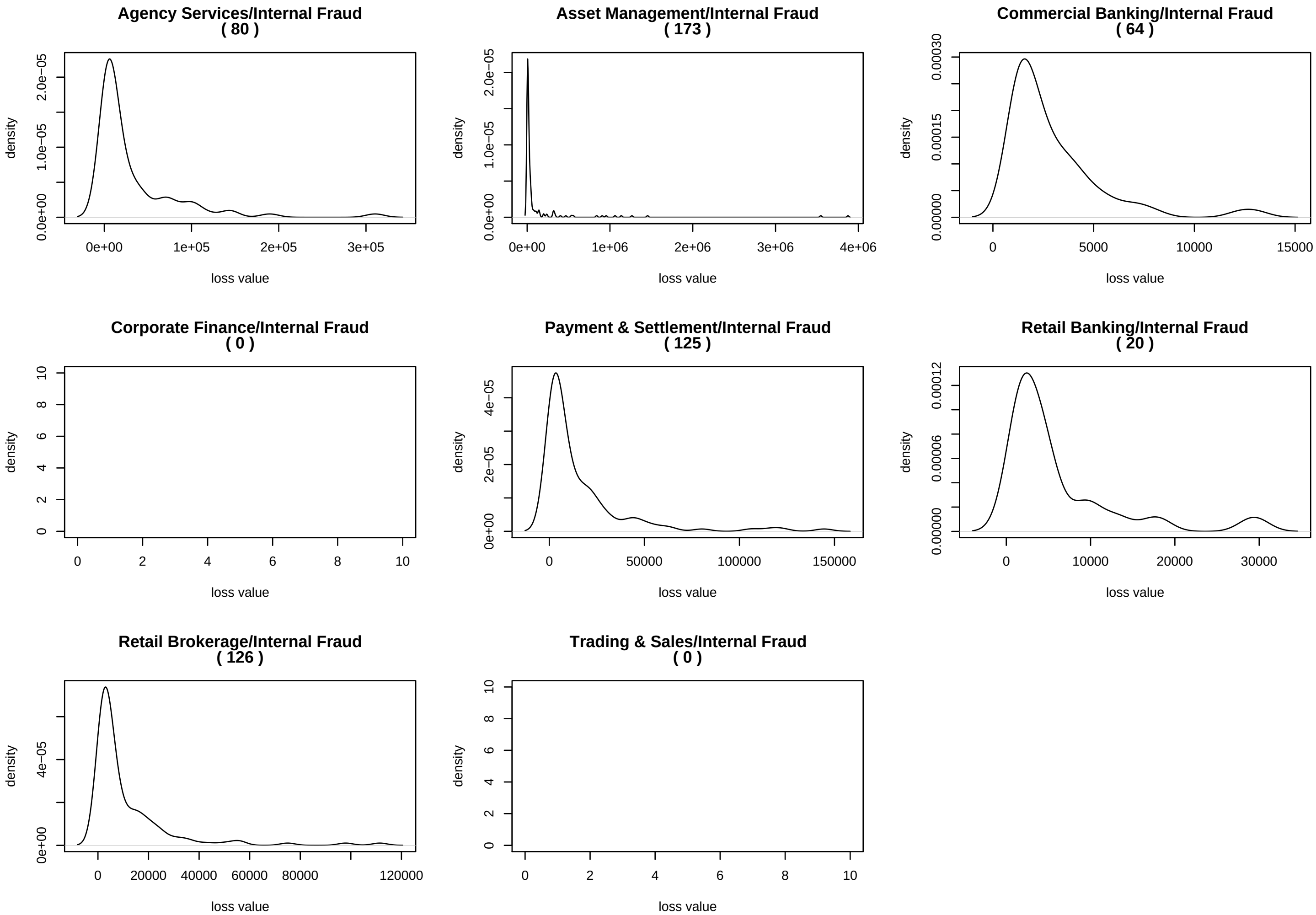


Figure 4.1: Loss densities for 7th risk category and all business lines

`a` = 1 means loss density for all business lines (`loss.data.object$blines`) and `b` = 7 means risk category `"Internal Fraud"`; `period` = `"none"` (default).

As we could easily check, there is no loss data in some of "business line"/"risk category" cells; see `loss.matrix.image(D,loss.data.object$blines,loss.data.object$rcateg)`. There are numbers of losses printed above plots.
But there can be many empty plots - for example `loss.density(1,1,loss.data.object)`: there is no data at all, so `no` (no plotting empty data) option could be useful.
Let us see the difference:

```
> loss.density(a=1,b=6,loss.data.object)
```

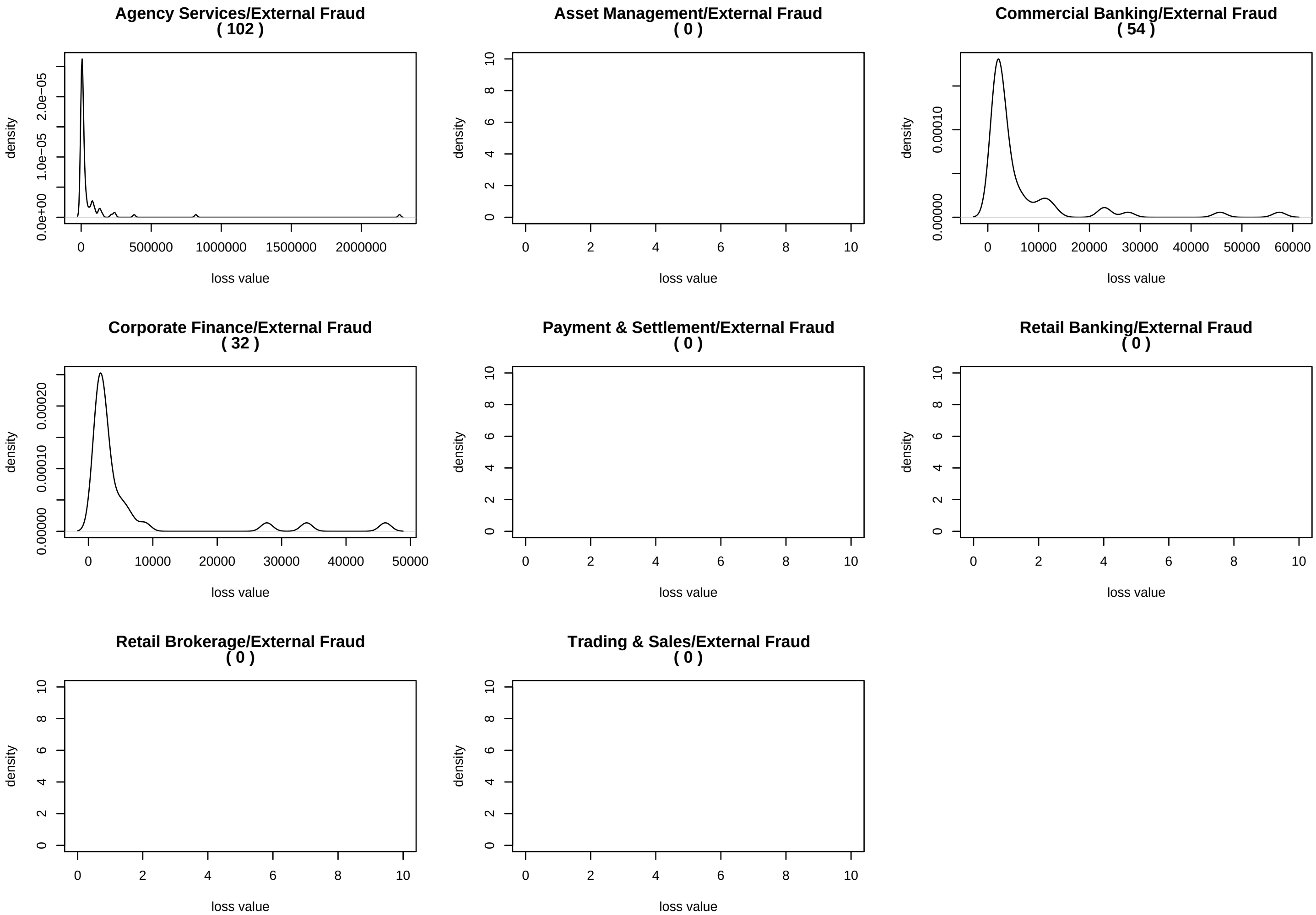


Figure 4.2: Loss densities for 6th risk category and all business lines

Risk category is "External Fraud". There are five empty plots. We can omit them.

```
> loss.density(a=1,b=6,loss.data.object,no=TRUE)
```

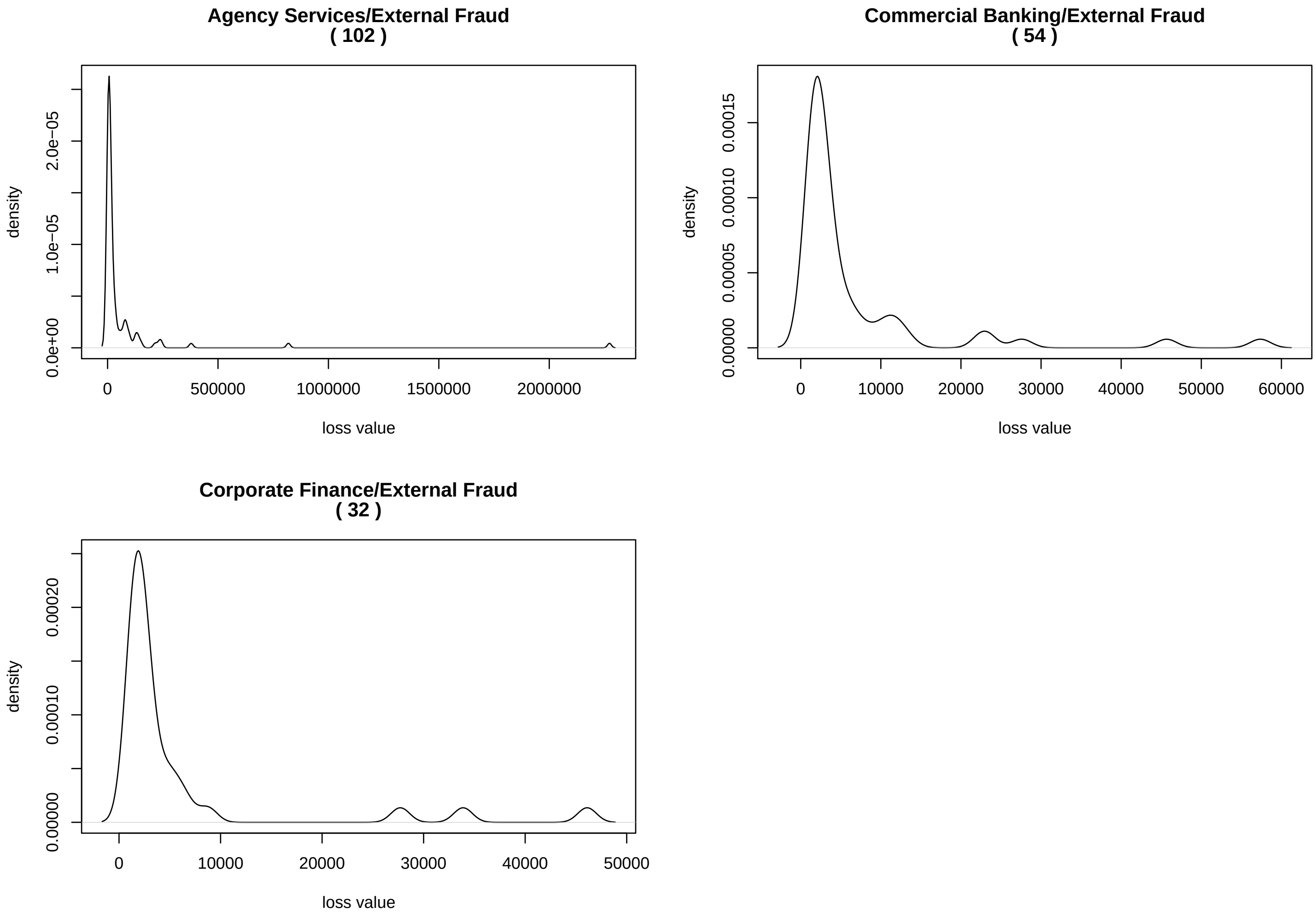


Figure 4.3: Loss densities for 6th risk category and all business lines with no empty plots

That one is more clear.

There is also possibility of having that function for one risk category (business line) and some chosen positions from business lines (risk categories).
For example, let us have 3rd business line (`"Commercial Banking"`) and vector of risk categories consisting of 6th and 7th risk category (`"External Fraud"` and `"Internal Fraud"`).

```
> loss.density(a=2,b=3,loss.data.object,rnumb=c(6,7))
```

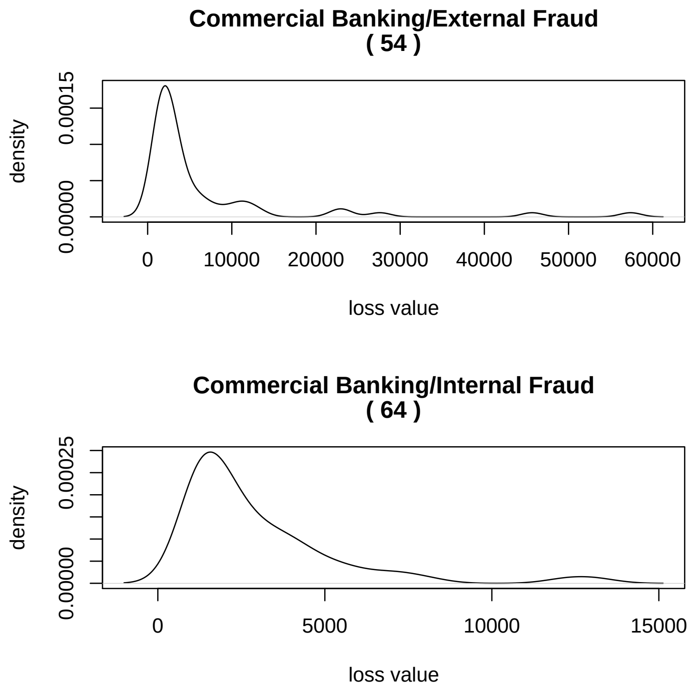


Figure 4.4: Loss densities for 3rd business line and risk categories with numbers from rnumb = c(6,7)

`a` = 2 means that we plot for risk categories (here: `rnumb`) and we choose one business line and its number is `b` = 3.
Of course it would be useless to give any `bnumb` because we choose only one business line and its number is `b`. Instruction like

```
> loss.density(a=2,b=3,loss.data.object,rnumb=c(6,7),bnumb = c(11))
```

are formally wrong (note that there is no 11th business line at all!) but correct in result (there is no warning related to that non existing 11 number because that instruction is omitted). We get the same figure as in 4.4.

Let us see density for only one risk category and business line cell:

```
> loss.density(a=1,b=7,bnumb=c(5),loss.data.object)
```

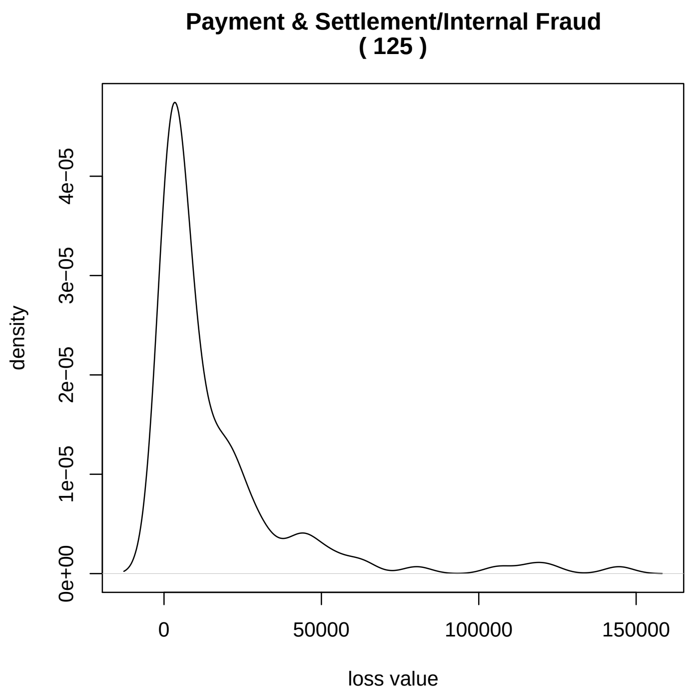


Figure 4.5: Loss density for 5th business line and 7th risk category

Risk category is `loss.data.object$rcateg[7]` and because `bnumb=c(5)`; there is only one business line, `loss.data.object$blines[5]`.

We will change `period` (which is `none` default). Code to generate that plots is obtained by changing `loss.density()` code i.e. putting "#" to hide `par(mfrow=c(n1,n2))` option and adding `period` to `main`:

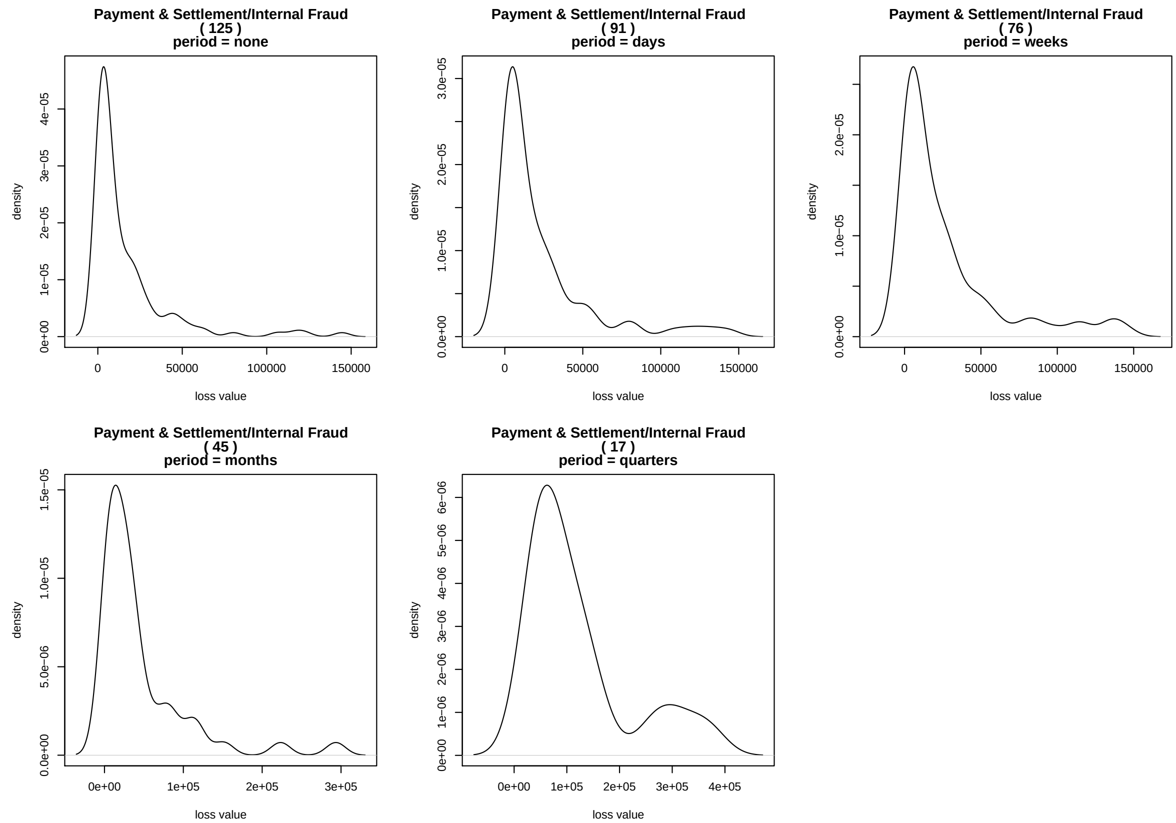
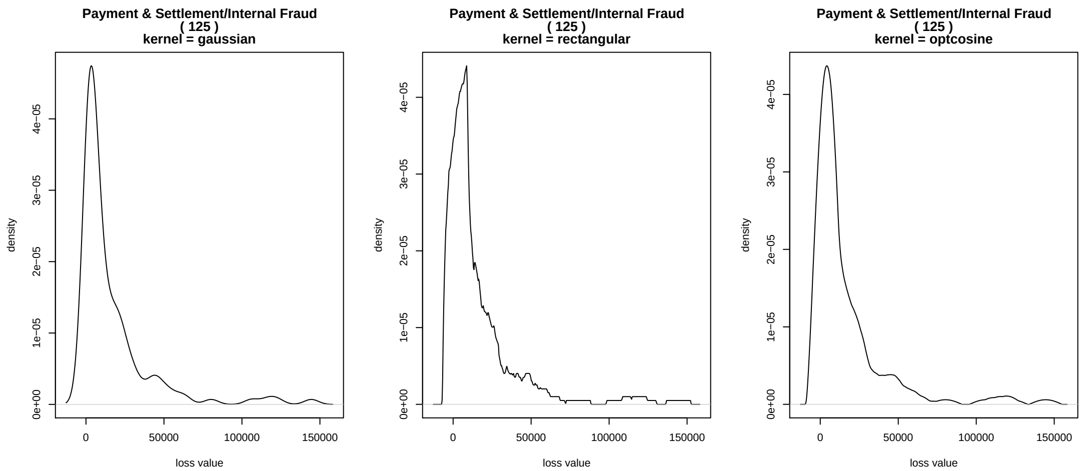
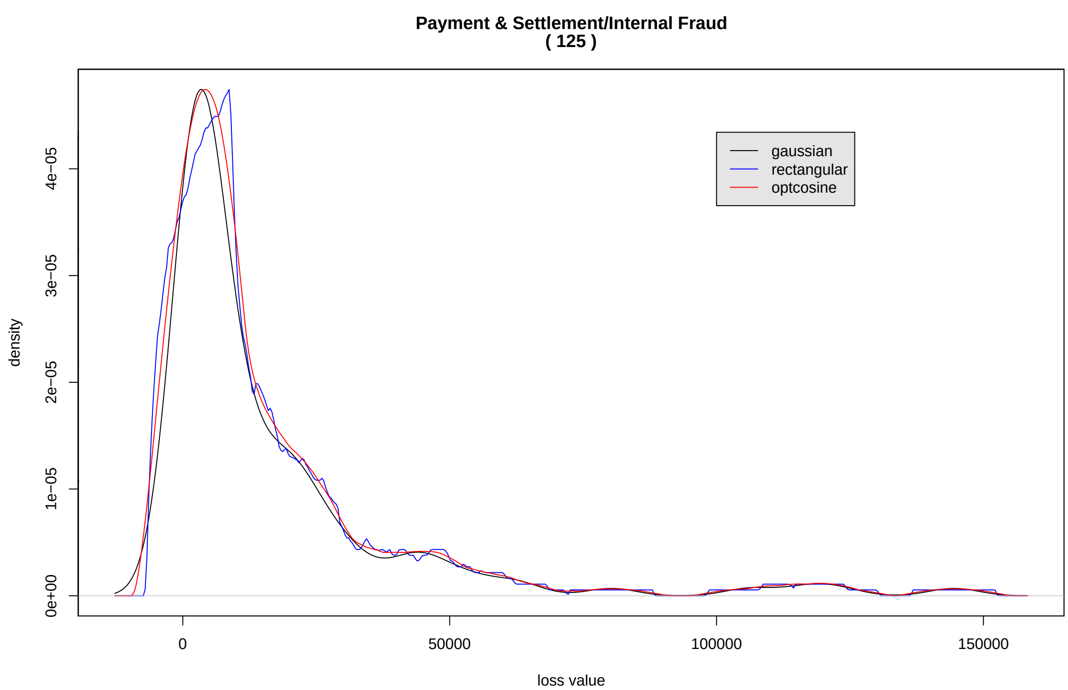


Figure 4.6: Loss density for 5th business line and 7th risk category; periods equal to ”none”, ”days”, ”weeks”, ”months” and ”quarters”

We can also use some `density` option, for example we could change `kernel` - let us choose `"gaussian"` (default), `"rectangular"` and `"optcosine"` (code to generate that plots is obtained by changing `loss.density` code):


Payment & Settlement/Internal Fraud
(125)
kernel = gaussian
Payment & Settlement/Internal Fraud
(125)
kernel = rectangular
Payment & Settlement/Internal Fraud
(125)
kernel = optcosine
density
loss value

On one plot:


Payment & Settlement/Internal Fraud
(125)
gaussian
rectangular
optcosine
density
loss value

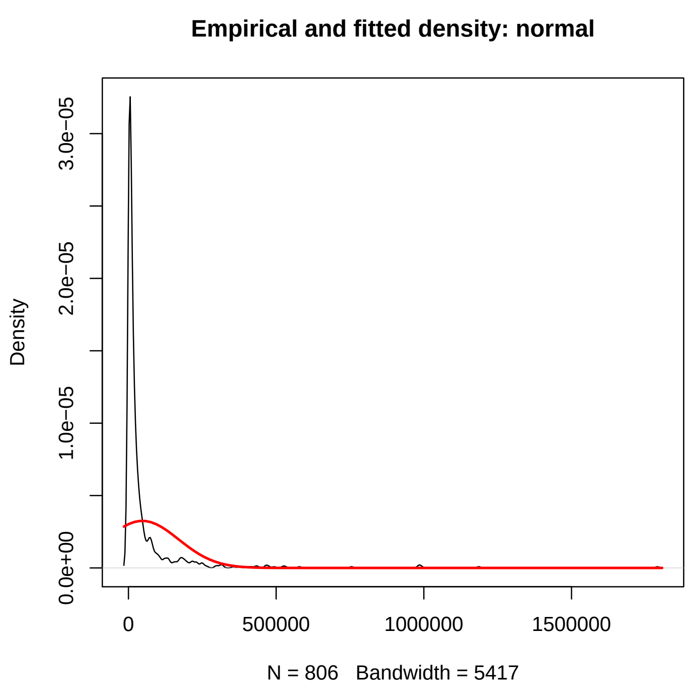
4.2 Fitting loss severity distributions

Let us fit some severity distributions.

4.2.1 "Agency Services/Clients, Products & Business Practices" cell

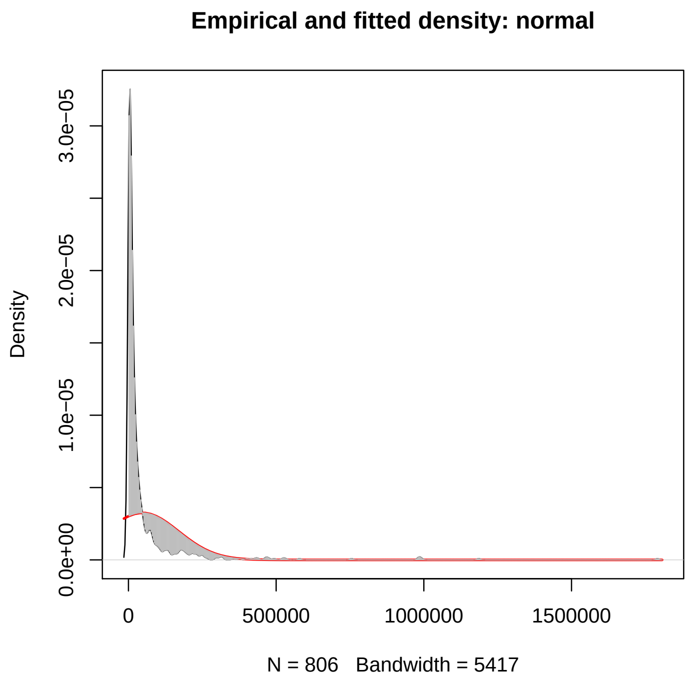
We will use `loss.fit.dist()` function to fit `"normal"` distribution to our data:

```
> x12<- read.loss(b=1,r=2,loss.data.object)
> loss.fit.dist("normal",x12)

$mean
[1] 47079.52

$sd
[1] 122694.0

           ad
0.0002364375
```



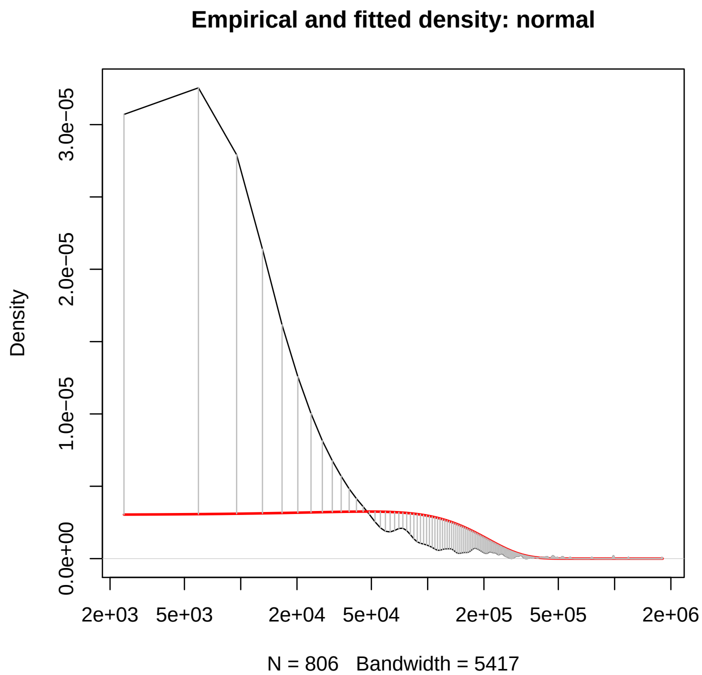
As we could see, it does not fit that data but we would like to have some numerical criterion and `ad` - sum of absolute differences between empirical and fitted density - provide it. To understand it better let us use `draw.diff` option (logical, `FALSE` default):

```
> x <- loss.fit.dist("normal",x12,draw.diff = T)
```



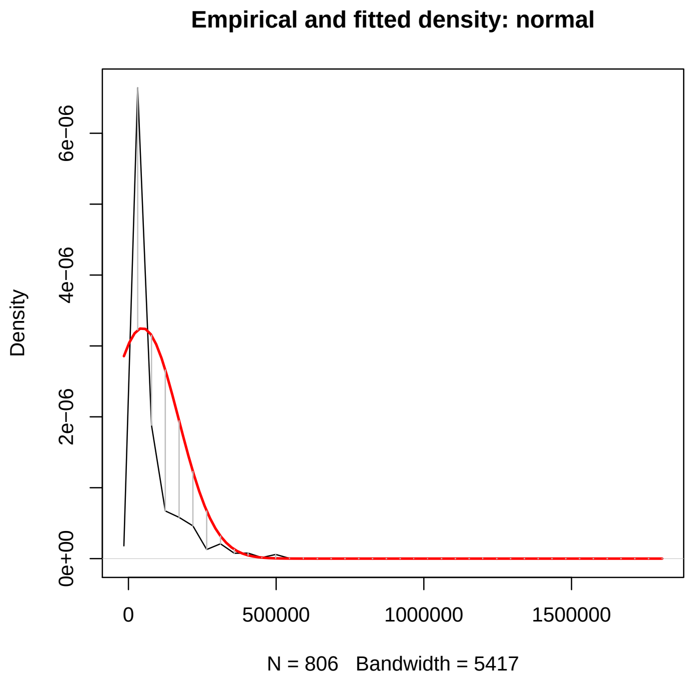
It could be perhaps more clear with `xlog.scale=T`:

```
> loss.fit.dist("normal", x12, draw.diff = T, xlog.scale = T)
```



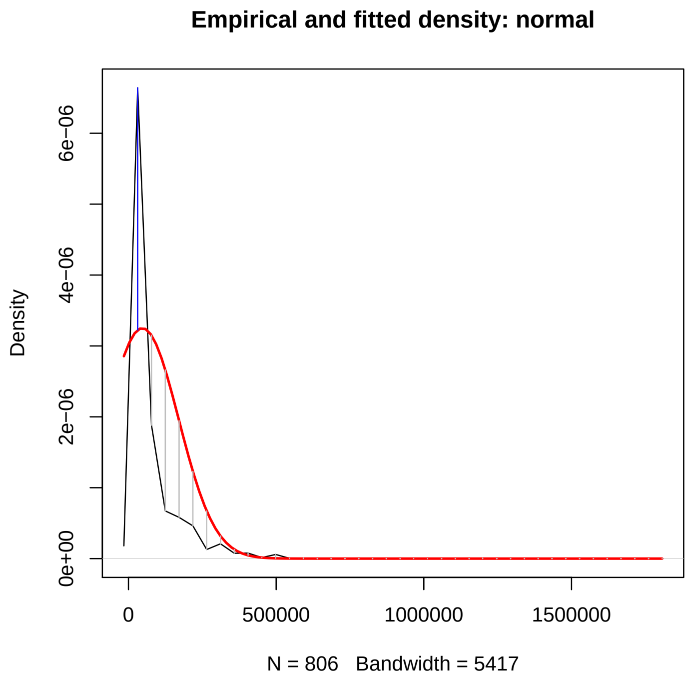
Empirical and theoretical density `y` points are connected by grey lines; we could change number of point to plot that densities and to compute that differences for them although it does not affect distribution parameters estimation:

```
> loss.fit.dist("normal",x12,draw.diff = T,n = 40)

$mean
[1] 47079.52

$sd
[1] 122694.0

         ad
9.61874e-06
```



Of course it was not a very good idea to have only 40 points because `ad` criterion is less reliable now; there should be many points because if there would be many non overlapping lines (which have in plot some width) with (almost) no breaks between them, we could multiply `ad` by that "line width" and obtain something rather resembling integral.
There is also option to draw maximum of that absolute differences:

```
> loss.fit.dist("normal",x12,draw.diff = T,n = 40,draw.max = T)

$mean
[1] 47079.52

$sd
[1] 122694.0

          ad
9.61874e-06
```



Let us see list of all `loss.fit.dist()` values for our `x`:

```
> summary(x)
```

```
           Length Class  Mode
loglik       1    -none- numeric
param        2    -none- list
sd           2    -none- numeric
q.t          0    -none- NULL
q.e          0    -none- NULL
ad           1    -none- numeric
teor.dens  507    -none- numeric
emp.dens   507    -none- numeric
maxdiff      1    -none- numeric
meandiff     1    -none- numeric
```

Where `loglik` (the log-likelihood), `param` (the parameter estimates) and `sd` (the estimated standard errors) are, in fact, values of `fitdistr()`; `q.t` and `q.e` are theoretical and empirical quantiles computed only for option `qq = T` (`FALSE` default); `teor.dens` and `emp.dens` are values of theoretical and empirical density values.
But why there are only 507 points in that two elements while `density()` computes `512` values (default)? We could also check that:

```
> length(density(x12[,2])$y)
```

```
[1] 512
```

But that is simple - computing absolute differences for non positive `x` seems to make no sense and:

```
> length(which(density(x12[,2])$x >0))
```

```
[1] 507
```

`fit.plot()` used in `loss.fit.dist()` does take only that positive `x` values.

Two last values are `maxdiff` (maximum absolute difference) and `meandiff` (mean absolute difference):

```
> x$maxdiff
```

```
[1] 2.946671e-05
```

```
> x$meandiff
```

```
[1] 4.663462e-07
```

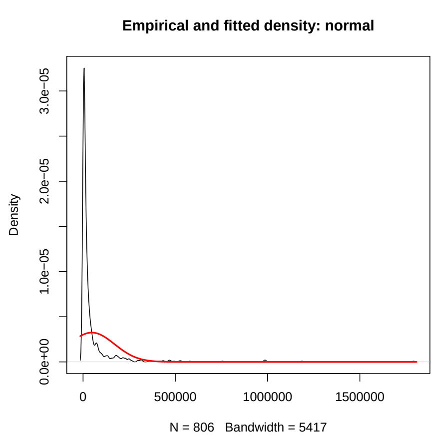
Let us use `qq` option.

```
> z<- loss.fit.dist("normal",x12,qq=T)
> summary(z$q.e)
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
    794.2    3845.0   11180.0   46110.0   34600.0 1791000.0
```

```
> summary(z$q.t)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-927300  -35680   47080   47080  129800 1021000
```



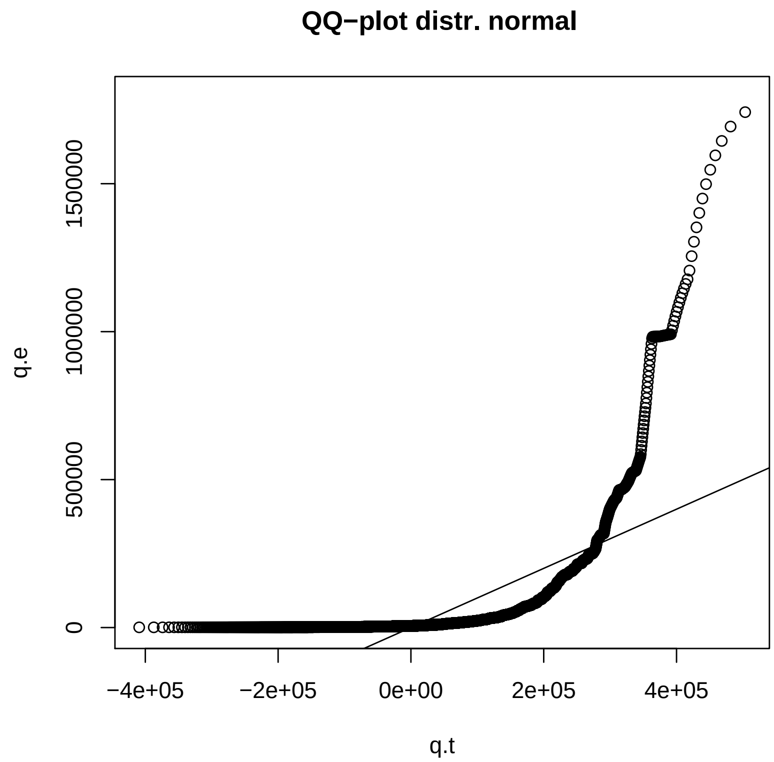


Figure 4.7: Theoretical quantiles for fitted normal distribution and empirical quantiles for x12 data

Of course quantiles are not close to `y = x` line but we do not expect that as we know "from image" that fit is very poor. We have:

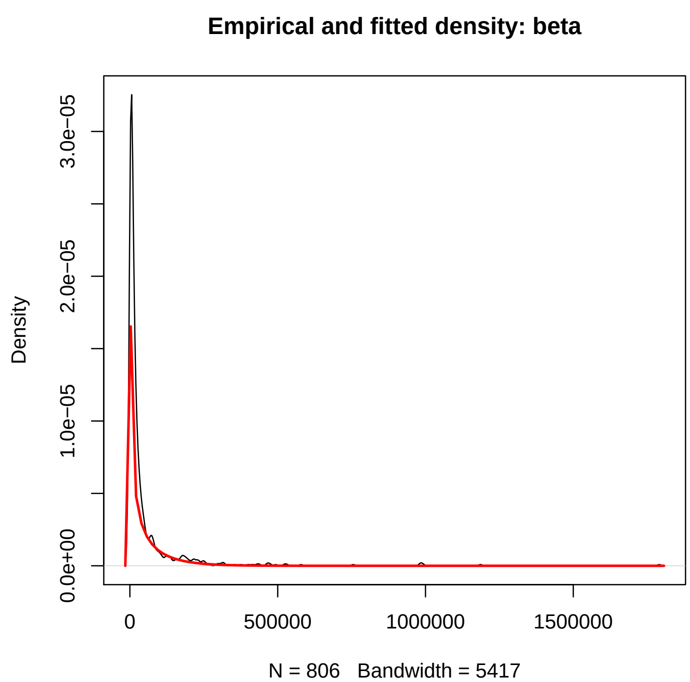
```
> head(cbind(z$q.e,z$q.t))

                     [,1]       [,2]
1.000000e-13% 794.2200 -927275.6
1.000100e-02% 807.3251 -409218.3
2.000200e-02% 820.4302 -387264.2
3.000300e-02% 833.5353 -373955.5
4.000400e-02% 846.6404 -364284.8
5.000500e-02% 859.7455 -356644.8
```

As we see there are negative values in theoretical quantiles and they are extreme. That also confirms that fit is not good.
Note that there is `length.out` option (10000 default) and it determines desired length of the sequence `p`, where `p = seq(from, to, length.out, by)` is

probs from quantile.

In our list of recognized distributions some are rather special. There is "beta" distribution which is continuous probability distribution defined on the interval (0,1). Of course losses are could be scaled but there still remains one big problem: our losses are bounded by their maximum and so are simulations. Perhaps it would not be a good idea to use it to simulate losses, particularly unexpected.
But we just will see one example:

```
> loss.fit.dist("beta",x12)

[1] "Argument scaled; x<- x/max(x)"
$shape1
[1] 0.471598

$shape2
[1] 17.10989

         ad
0.0001251046
```



We have:

```
> max(x12[,2])
```

```
[1] 1790530
```

...so there could not be loss bigger than that. Though "beta" seems well fitted, one should remember that.

But fitdistr() (and in result loss.fit.dist()) can fit distributions not being one on the list of distributions known by name (character string). We will try that other way on "dnorm" (with no name given it would be treated as an unknown distribution); we will need start values, so:

```
> new.start = list(sd= sd(x12[,2]),mean = mean(x12[,2]))
```

```
> loss.fit.dist(dnorm,x12,start = new.start)
```

That gives error message:

```
Error in solve.default(res$hessian) :
Lapack routine dgesv:  system is exactly singular
```

Start values were rather good (method of moments):

```
> new.start
```

```
$sd
[1] 122770.1

$mean
[1] 47079.52
```

Indeed, let us compare them with the estimated parameters obtained for loss.fit.dist("normal",x12). The problem comes from fitdistr:

```
> fitdistr(x12[,2],dnorm,start = new.start)
```

...and we have the same warning message as before. As we could check in help(fitdistr), "direct optimization of the log-likelihood is performed using optim", then, from help(optim) we learn that there are only 500 iterations for "Nelder-Mead" method used in fitdistr() (and "Defaults to 100 for the derivative-based methods"). Only for "SANN" method we have total number of function evaluations (iterations) - maxit defaults to 10000 and there is no other stopping criterion. Try:

```
> fitdistr(x12[,2],dnorm,start = new.start,method = "SANN")
```

That method is based on simulations therefore sometimes it gives some results and sometimes not. Try typing above command a few times. If one is lucky, one can obtain some results soon or later (and sometimes rather good). For "normal" distribution treated as known distribution parameters values are simply computed as:

```
> x<- x12[,2]
> n <- length(x)
>         sd0 <- sqrt((n - 1)/n) * sd(x)
>       mx <- mean(x)
>       estimate <- c(mx, sd0)
>       sds <- c(sd0/sqrt(n), sd0/sqrt(2 * n))
>       names(estimate) <- names(sds) <- c("mean", "sd")
> estimate

     mean        sd
 47079.52 122693.96
```

This could be easily verified by typing `fitdistr` or `fix(fitdistr)` to see that function code.

Maybe for big datasets and unrecognized by name distributions (in case of `loss.fit.dist()` it would be also `"inverse gaussian"` which is unrecognized by `fitdistr()`) more subtle methods are needed, for `fitdistr()` does not compute required values. Of course one can always change method, use method of moments, and/or cut data to obtain some results:

```
> loss.fit.dist(dnorm,x12[1:500,2],start = new.start)

$sd
[1] 122770.1

$mean
[1] 47079.52

          ad
0.0002690216
```

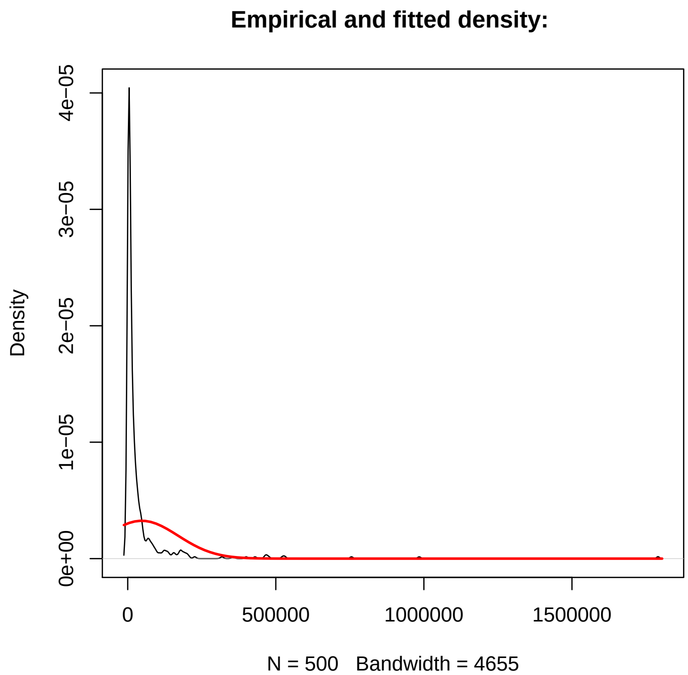
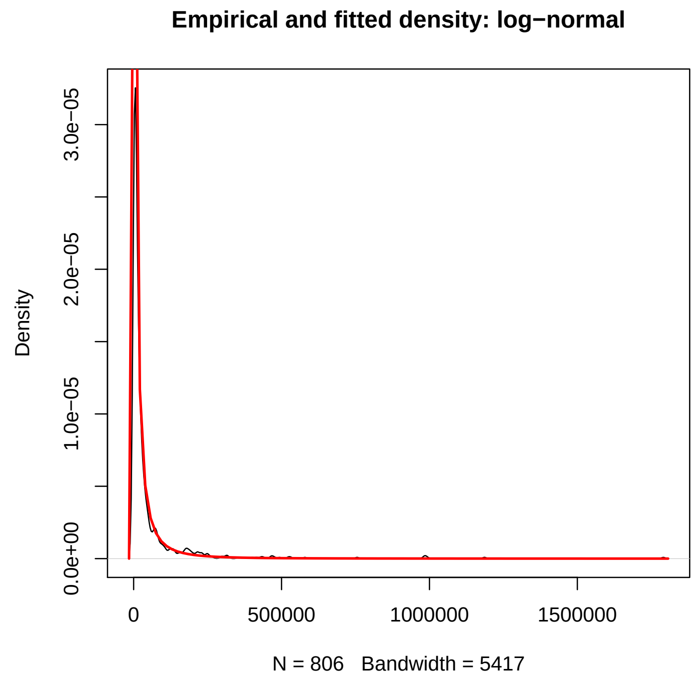


Figure 4.8: Empirical density and fitted normal density for first 500 rows of x12. Note that distname argument was not given.

These are not very good values but our data is different - cut by about 38 percent.

It is the time to show well fitted distribution and that would be `"log-normal"`:

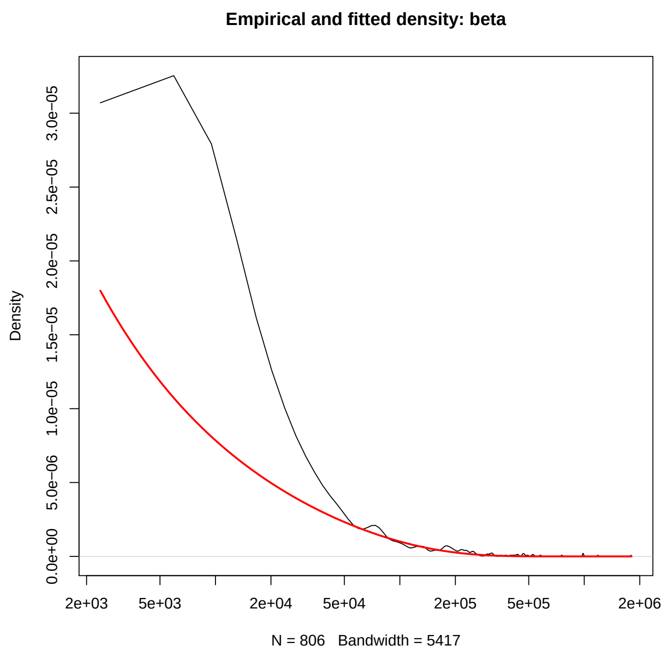
```
> loss.fit.dist("log-normal",x12)

$meanlog
[1] 9.455174

$sdlog
[1] 1.542905

           ad
6.158446e-05
```

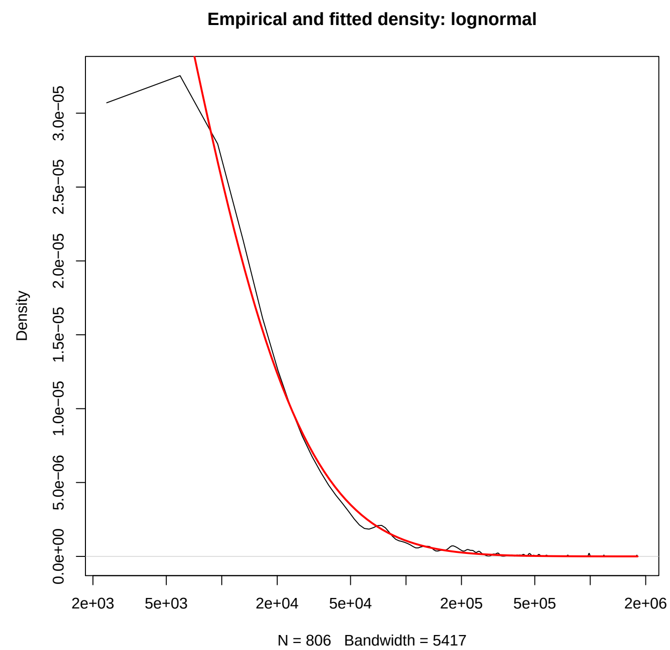


Though "`beta`" seemed rather good, "`log-normal`" is much better and seems perfect. Let us compare that two using with x logarithmic scale - that would be `TRUE` for `xlog.scale` option:

```
> par(mfrow = c(1,2))
>  loss.fit.dist("beta",x12,xlog.scale=T)

[1] "Argument scaled; x<- x/max(x)"
$shape1
[1] 0.471598

$shape2
[1] 17.10989

          ad
0.0001251046

>  loss.fit.dist("lognormal",x12,xlog.scale=T)

$meanlog
[1] 9.455174

$sdlog
[1] 1.542905
```

```
            ad
6.158446e-05
```





Beta, Exponential, F, Gamma, Log-normal, Logistic, Normal and Weibull fits

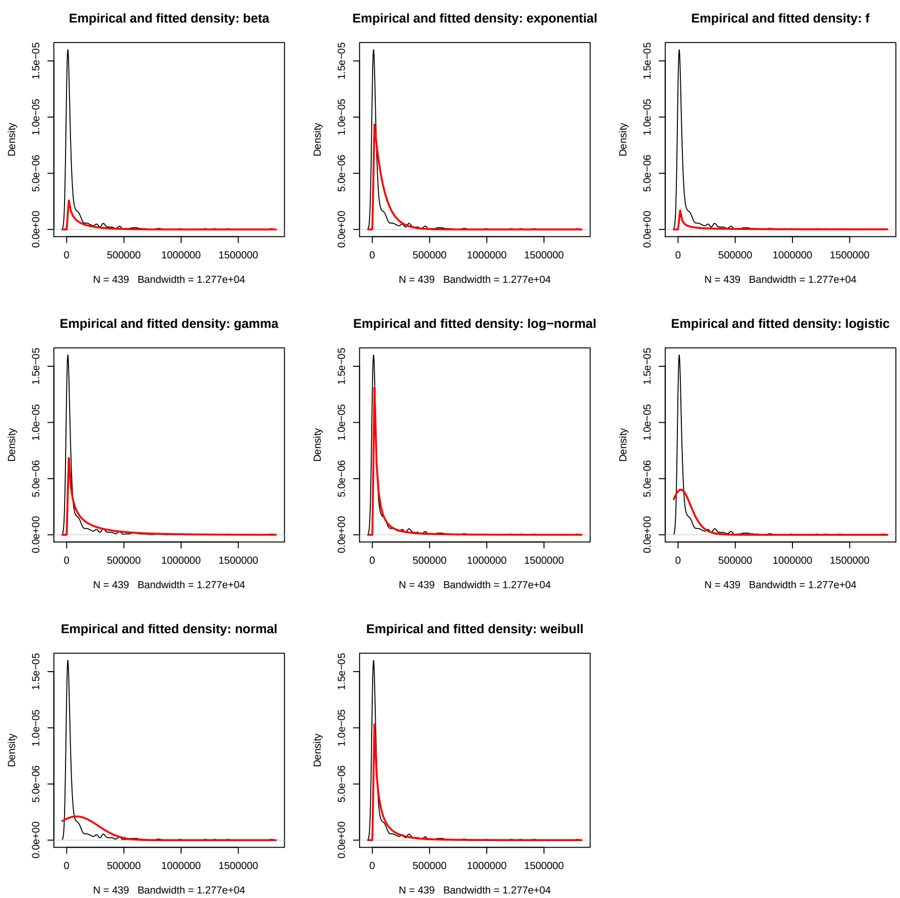
One can see that `"log-normal"` is undeniably better than `"beta"`. It has also smaller `ad` of course.
Let us see that simple code:

```
> dlist = c("beta", "exponential", "f", "gamma", "log-normal",
+                   "logistic", "normal", "weibull")
> par(mfrow = c(3,3))
> ad.days <- {}
> k <- 1
> for(i in dlist){
+ u<- loss.fit.dist(x=x12, densfun = i, period = "days")$ad
+ ad.days[[k]]<- u; k <- k+1
+ }
```

```
[1] "Argument scaled; x<- x/max(x)"
```

```
> names(ad.days)<- dlist
> ad.days<- sort(ad.days,decreasing = T); ad.days

      normal            f     logistic         beta  exponential        gamma
2.140493e-04 1.787725e-04 1.595774e-04 1.545514e-04 1.396397e-04 1.163744e-04
     weibull   log-normal
8.308964e-05 7.989927e-05

> names(which(ad.days ==min(ad.days)))

[1] "log-normal"
```

`dlist` is list of our distributions. We have an 3-by-3 array on the device - `par(mfrow=c(3,3))` - and we compute `ad` and draw plot for each one of that distributions, then we have `ad.days` - list of our `ad` values and we choose the smallest of them; `"log-normal"` is the best.



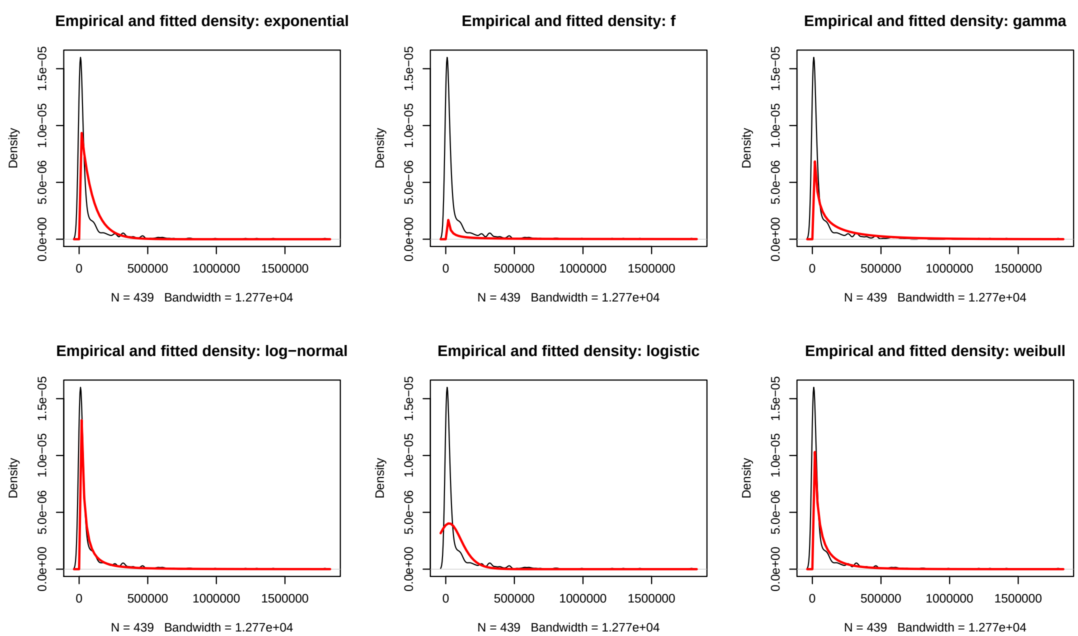
We could also exclude "beta" and "normal" at once:

```
> dlist2 = c("exponential", "f", "gamma", "log-normal", "logistic", "weibull")
> par(mfrow = c(2,3))
> ad.days2 <- {}
> k <- 1
> for(i in dlist2){
+ u<- loss.fit.dist(x=x12, densfun = i, period = "days")$ad
+ ad.days2[[k]]<- u; k <- k+1
+ }
> names(ad.days2)<- dlist2
> ad.days2<- sort(ad.days2,decreasing = T); ad.days2
```

```
           f     logistic  exponential        gamma      weibull   log-normal
1.787725e-04 1.595774e-04 1.396397e-04 1.163744e-04 8.308964e-05 7.989927e-05
```

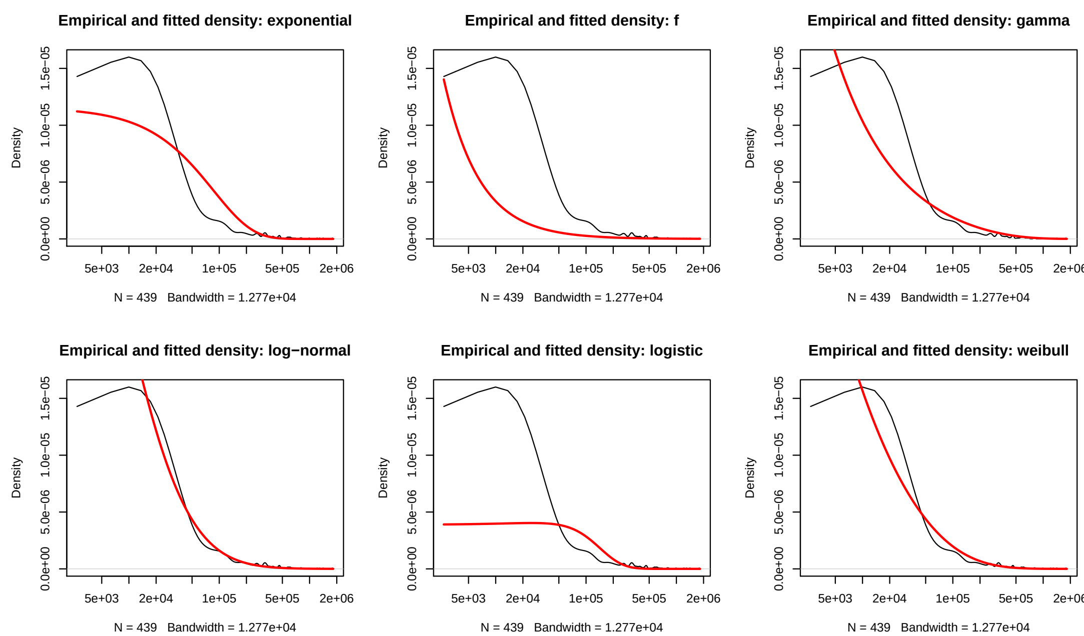
```
> names(which(ad.days2 ==min(ad.days2)))
```

```
[1] "log-normal"
```



And the same using `xlog.scale=T`:

```
> par(mfrow = c(2, 3))
> for (i in dlist2) {
+     u <- loss.fit.dist(x = x12, densfun = i, period = "days",
+         xlog.scale = T)$ad
+ }
```

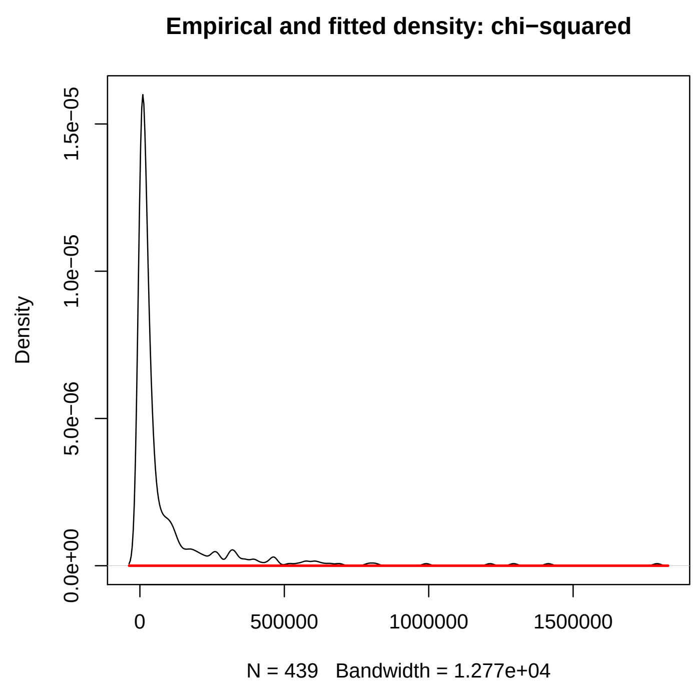


But there remain some other distributions - `"cauchy"`, `"chi-squared"` and `"inverse gaussian"`. They should be called with some additional parameters. Maybe one of them is better? Let us check:

Chi-squared fit

We will use `loss.fit.dist()` for `"chi-squared"` distribution:

```
> k <- loss.fit.dist("chi-squared", x12, period = "days")
> k

$df
[1] 22521.04

           ad
0.0002301894
```



That does not seem fit our data. Maybe we should change method?

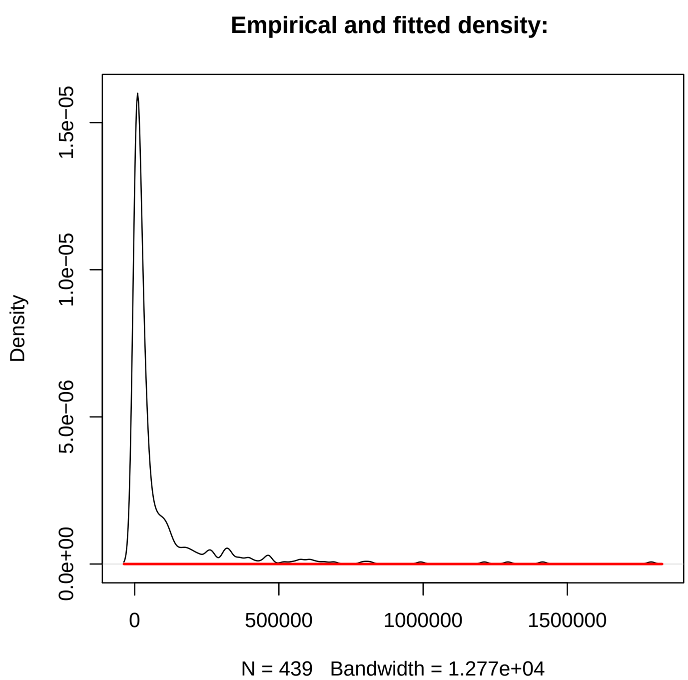
```
> loss.fit.dist("chi-squared",x12,period="days",method = "BFGS")

$df
[1] 22504.64

           ad
0.0002301894
```

Very similar plot - not included.

Why we have line drawn for `"chi-squared"` distribution? For `fit.plot()`, which is used in `loss.fit.dist()`, called with parameters obtained before we have the same result:

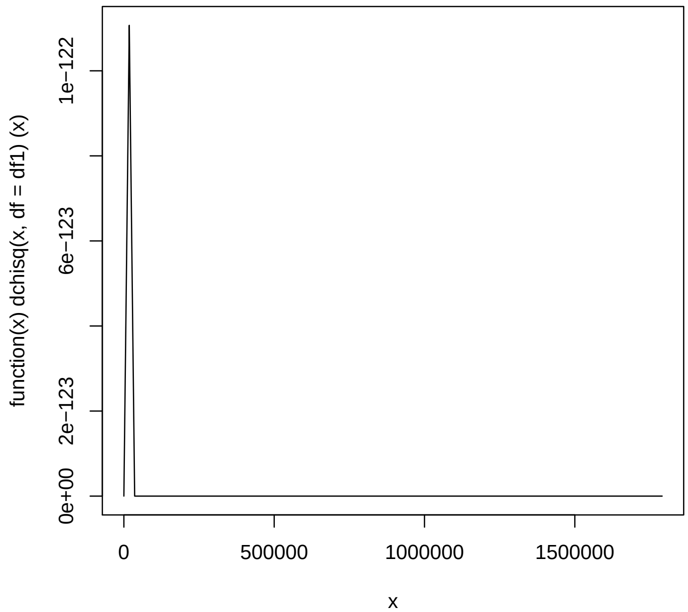
```
> df1 <- as.numeric(k$param)
> z <- period.loss(x12, "days")
> fit.plot(z, dchisq, param = list(df = df1))
```

```
[1] 0.0002301894
```



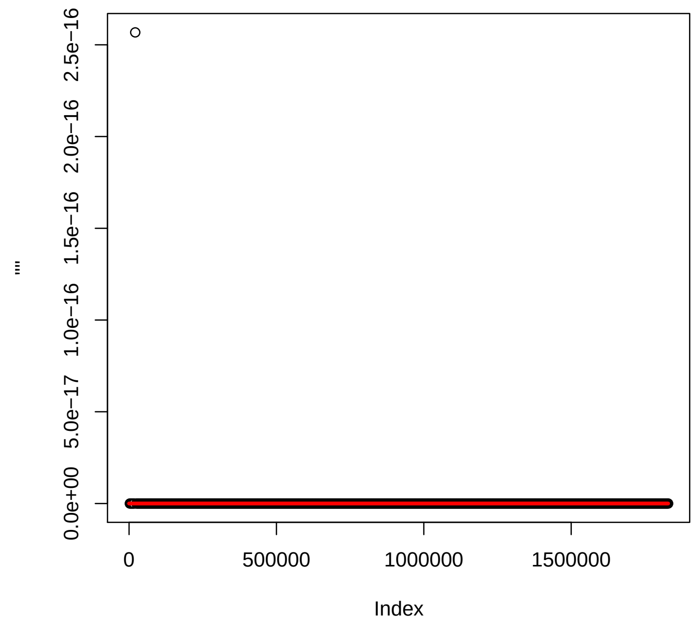
Why does it looks like that? Let us check:

```
> plot(function(x) dchisq(x,df = df1),xlim = c(0,max(z)))
```



This is our "chi-squared" density function.
Why it is not plotted like that? Let us see:

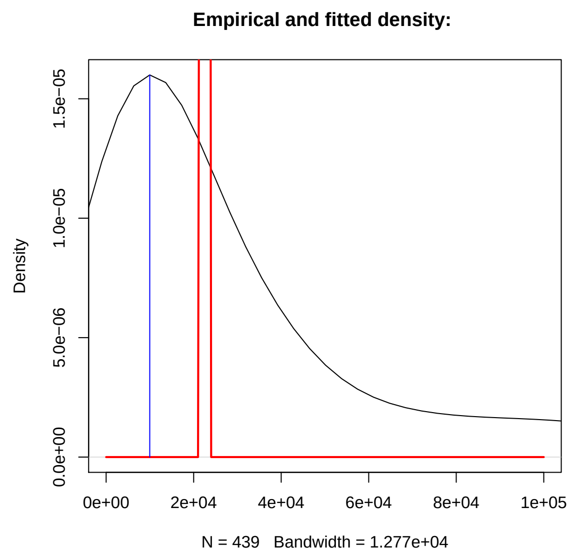
```
> nmbrs <- which(density(z)$x > 0)
> plus.nmbrs <- density(z)$x[nmbrs]
> plus <- which(dchisq(plus.nmbrs, df = df1) > 0)
> plus.values <- dchisq(plus.nmbrs[plus], df = df1)
> plus.values

[1] 3.219630e-161  2.567757e-16  1.037219e-22 2.640469e-138

> plot("", xlim = c(0, max(plus.nmbrs)), ylim = c(0, max(plus.values)))
> points(plus.nmbrs, dchisq(plus.nmbrs, df = df1))
> curve(dchisq(x, df = df1), add = T, col = "red", lwd = 3)
```



`nmbrs` are that of `density(z)$x` which are strictly positive; then we check thosewith strictly positive `density()` values and plot them (three of them are so small in relation to fourth that we do see them as zeros on the plot). Afterwards `curve()` is used and it seems that it could not join all those points - `max(plus.values)` is excluded.

We could also check that some `xlim` manipulation allow us to see better plot:

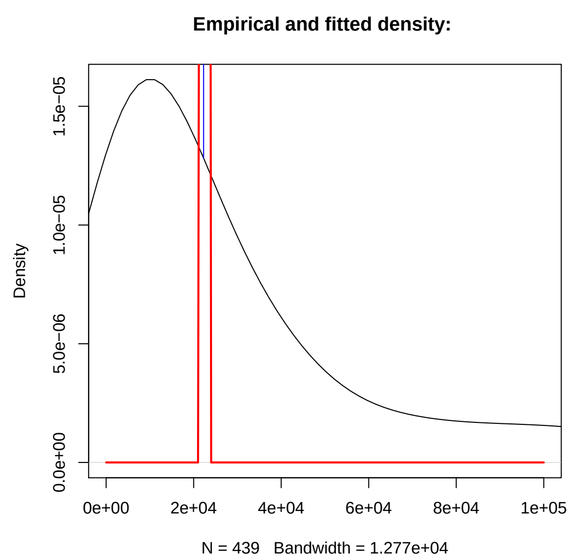
```
> fit.plot(z, dchisq, param = list(df = df1), draw.max = T, xlim = c(0,1e+05))

[1] 0.0002301894
```



We could also do some `n` manipulation, where `n` is number of points to compute values:

```
> fit.plot(z, dchisq, param = list(df = df1), draw.max = T,
+      xlim = c(0,1e+05), n = 1000)
```

```
[1] 0.001365987
```



. . . but as we see, that distribution does not fit that data well, so we leave it.

Cauchy fit

We would like to fit "cauchy" distribution to x12 with period = "days":

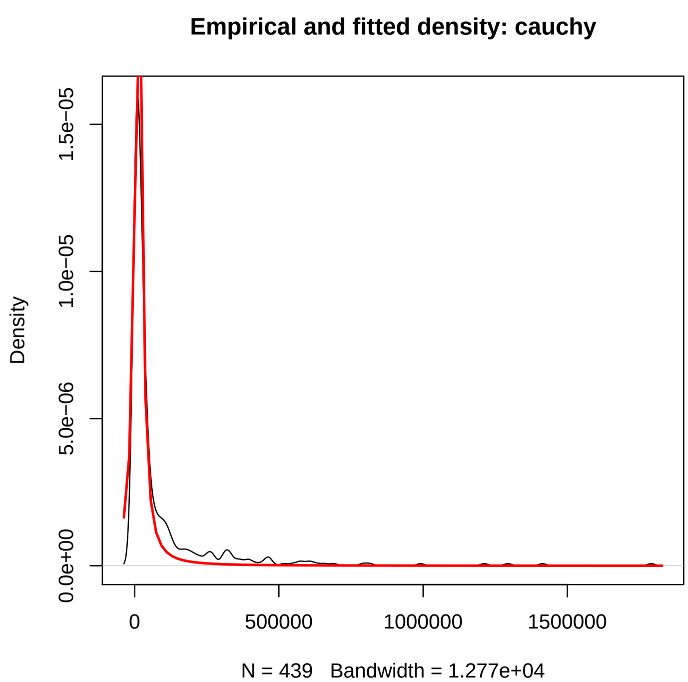
```
> loss.fit.dist("cauchy",x12,period = "days")
```

That gives error message:

```
Error in solve.default(res$hessian) :
Lapack routine dgesv:  system is exactly singular
```

But we can change method:

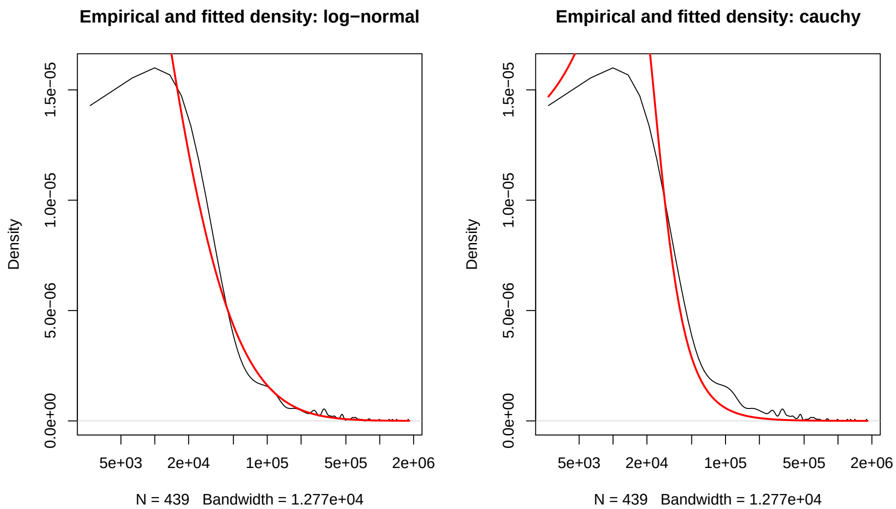
```
> loss.fit.dist("cauchy",x12,period = "days",method = "Nelder-Mead")

$location
[1] 12959.27

$scale
[1] 14188.55

           ad
8.679948e-05
```



It is a good fit, but "log-normal" is still better for that loss data.

We could also compare that fits using `xlog.scale=T`:

```
> par(mfrow = c(1, 2))
> loss.fit.dist("log-normal", x12, period = "days", xlog.scale = T)
> loss.fit.dist("cauchy", x12, period = "days", method = "Nelder-Mead",
+     xlog.scale = T)
```



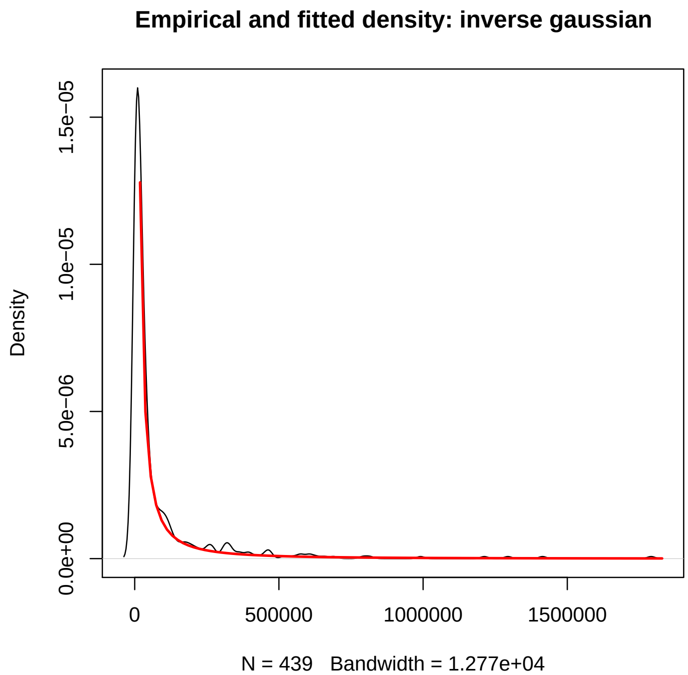
Only `"inverse gaussian"` remains:

Inverse gaussian fit

```
> loss.fit.dist("inverse gaussian",x12,period = "days")
```

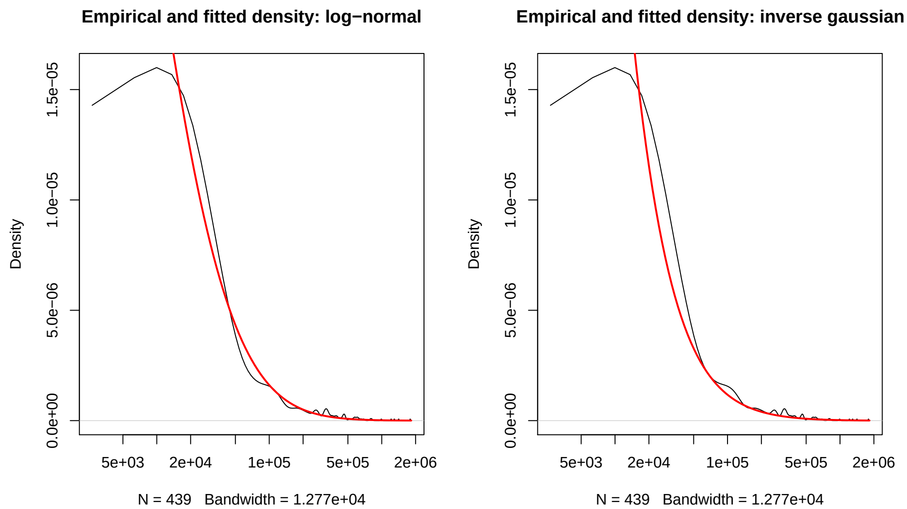
Gives the same error message as for `"cauchy"`. Changing `method` we have:

```
> loss.fit.dist("inverse gaussian",x12,period = "days",method = "Nelder-Mead")

$lambda
[1] 8653.568

$nu
[1] 86341.77

          ad
0.0001247034
```



Of course `ad` for `"inverse gaussian"` is more than for `"log-normal"`. We could also compare that fits using `xlog.scale=T`:

```
> par(mfrow = c(1, 2))
> loss.fit.dist("log-normal", x12, period = "days", xlog.scale = T)
> loss.fit.dist("inverse gaussian", x12, period = "days", method = "Nelder-Mead",
+     xlog.scale = T)
```

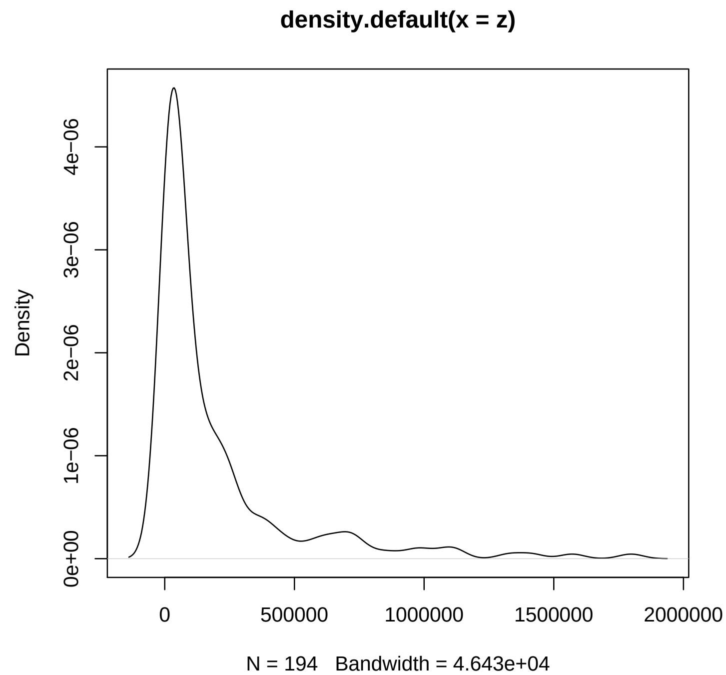


It is also good but no better than `"log-normal"`.

The other periods

We should do the same for `weeks`, `months` and `quarters` periods. Let us see density for `weeks`:

```
> z<- period.loss(x12,"weeks")
> plot(density(z))
```



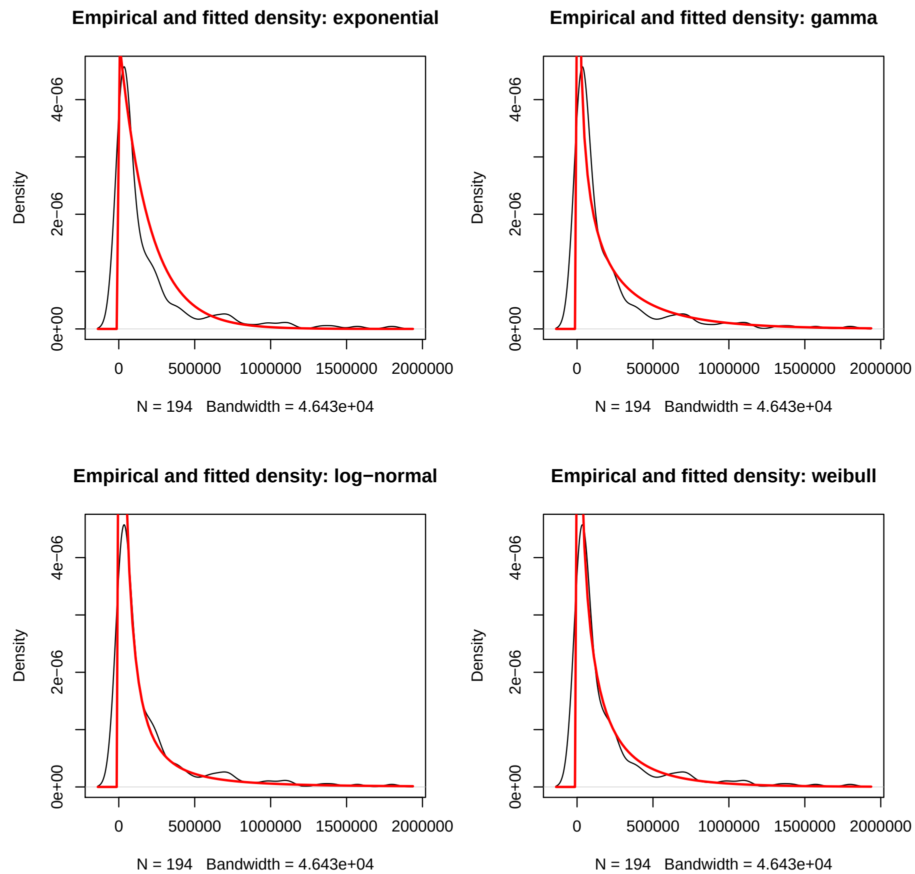
We have good fits for `"weibull"`, `"gamma"`, `"log-normal"` and `"exponential"` - in that order. Differences are not big:

```
> dlist3  =  c("exponential","gamma", "log-normal", "weibull")
> par(mfrow = c(2,2))
> ad.weeks <- {}
> k <- 1
> for(i in dlist3){
+ u<- loss.fit.dist(x=x12, densfun = i, period = "weeks")$ad
+ ad.weeks[[k]]<- u; k <- k+1
+ }
> names(ad.weeks)<- dlist3
> ad.weeks <- sort(ad.weeks ,decreasing = T); ad.weeks
```

```
 exponential   log-normal         gamma      weibull
7.216382e-05 6.797123e-05 6.345749e-05 5.706141e-05
```

```
> names(which(ad.weeks  == min(ad.weeks)))
```

```
[1] "weibull"
```



All that distributions seems very well fitted; we would like to choose the same distribution for `days` and for `weeks`. We should be aware that changing `n` could change our choice:

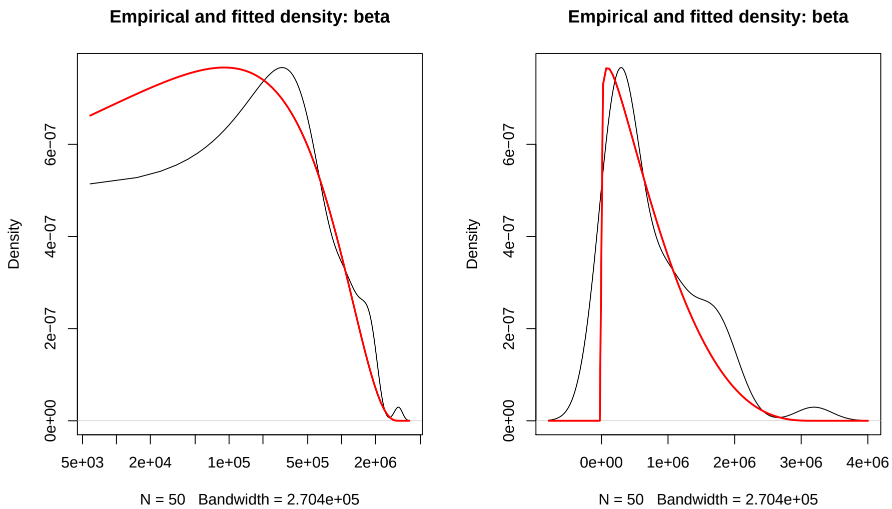
```
> dlist3  =  c("exponential","gamma", "log-normal", "weibull")
> par(mfrow = c(2,2))
> ad.weeks2 <- {}
> k <- 1
> for(i in dlist3){
+ u<- loss.fit.dist(x=x12, densfun = i, period = "weeks",n=10000)$ad
+ ad.weeks2[[k]]<- u; k <- k+1
+ }
> names(ad.weeks2)<- dlist3
> ad.weeks2 <- sort(ad.weeks2 ,decreasing = T); ad.weeks2

      gamma exponential     weibull  log-normal
0.001599348 0.001427725 0.001345542 0.001344964
```

```
> names(which(ad.weeks2==min(ad.weeks2)))

[1] "log-normal"
```

As we can see, now `"log-normal"` is the best distribution. Of course we could change `n` also for `days` but we will simply choose that `"log-normal"` distribution.

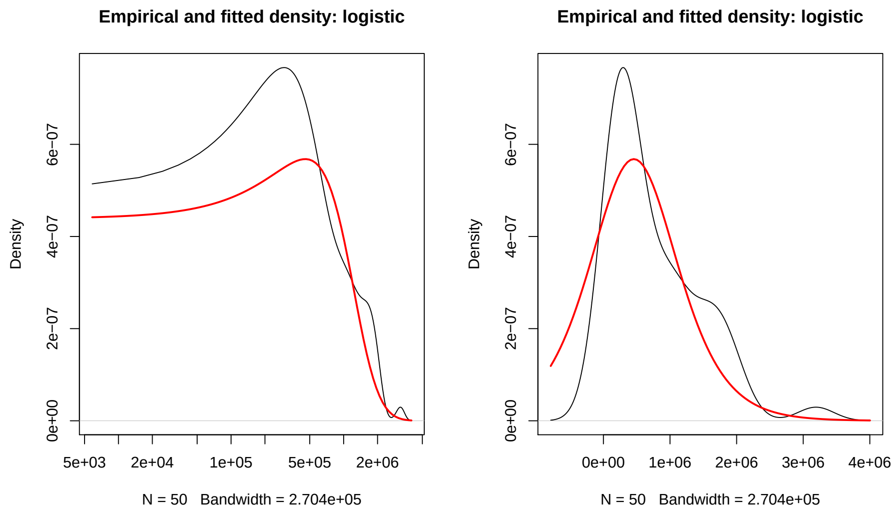
For `months` we have `"beta"` or `"logistic"` distribution:

```
> par(mfrow = c(1,2))
> loss.fit.dist(x=x12, densfun = "beta", period = "months",xlog.scale=T)

[1] "Argument scaled; x<- x/max(x)"
$shape1
[1] 1.080420

$shape2
[1] 3.753931

          ad
1.705283e-05

> loss.fit.dist(x=x12, densfun = "beta", period = "months")

[1] "Argument scaled; x<- x/max(x)"
$shape1
[1] 1.080420

$shape2
[1] 3.753931

          ad
1.705283e-05
```



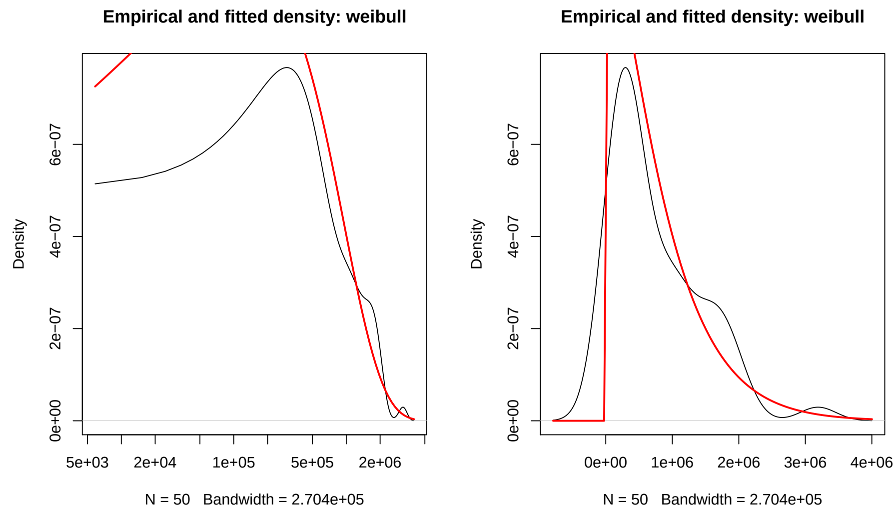
. . . and . . .

```
> par(mfrow = c(1,2))
> loss.fit.dist(x=x12, densfun = "logistic", period = "months",xlog.scale=T)

$location
[1] 456530.1

$scale
[1] 440127.1

          ad
2.350981e-05

> loss.fit.dist(x=x12, densfun = "logistic", period = "months")

$location
[1] 456530.1

$scale
[1] 440127.1

          ad
2.350981e-05
```



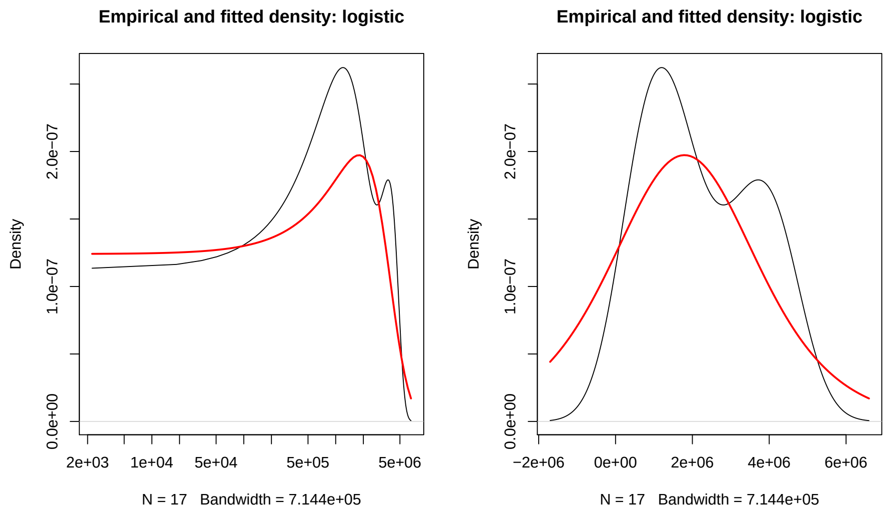
There is also "weibull" distribution which seems a reasonable choice.

```
> par(mfrow = c(1,2))
> loss.fit.dist(x=x12, densfun = "weibull", period = "months",xlog.scale=T)

$shape
[1] 1.136458

$scale
[1] 797806.5

          ad
2.486692e-05

> loss.fit.dist(x=x12, densfun = "weibull", period = "months")

$shape
[1] 1.136458

$scale
[1] 797806.5

          ad
2.486692e-05
```



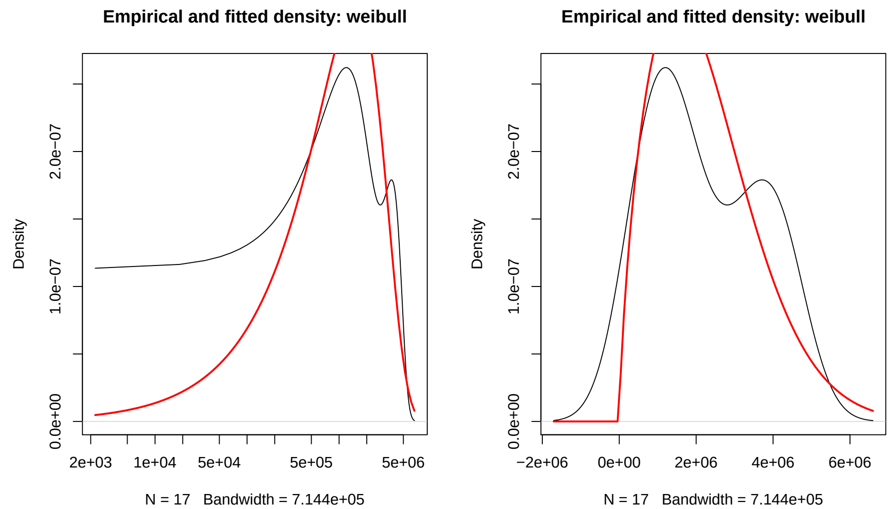
And the same about `quarters` only now `"logistic"` is the best and then `"weibull"` and `"beta"`.

```
> par(mfrow = c(1,2))
> loss.fit.dist(x=x12, densfun = "logistic", period = "quarters",xlog.scale=T)

$location
[1] 1793798

$scale
[1] 1266969

          ad
1.399808e-05

> loss.fit.dist(x=x12, densfun = "logistic", period = "quarters")

$location
[1] 1793798

$scale
[1] 1266969

          ad
1.399808e-05
```



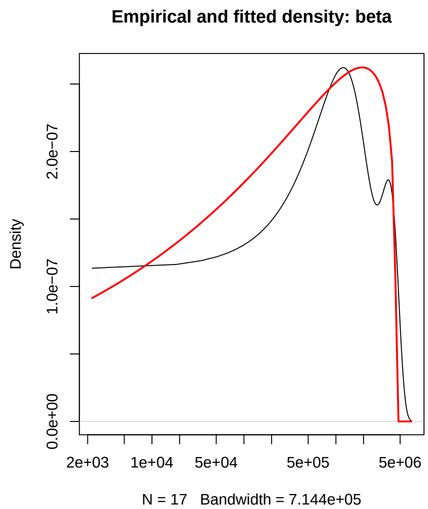
The same for "weibull":

```
> par(mfrow = c(1,2))
> loss.fit.dist(x=x12, densfun = "weibull", period = "quarters",xlog.scale=T)

$shape
[1] 1.705159

$scale
[1] 2528889

          ad
1.634660e-05

> loss.fit.dist(x=x12, densfun = "weibull", period = "quarters")

$shape
[1] 1.705159

$scale
[1] 2528889

          ad
1.634660e-05
```

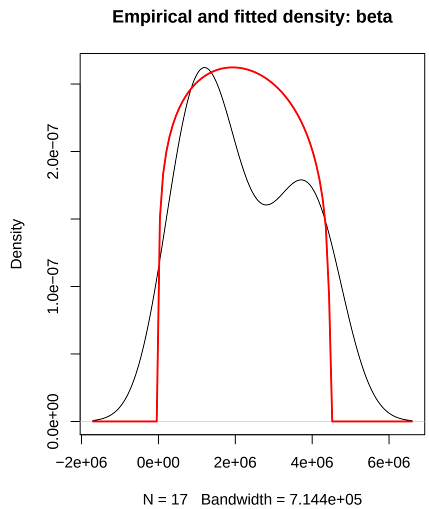


And the same for "beta":

```
> par(mfrow = c(1,2))
> loss.fit.dist(x=x12, densfun = "beta", period = "quarters",xlog.scale=T)

[1] "Argument scaled; x<- x/max(x)"
$shape1
[1] 1.175090

$shape2
[1] 1.228670

          ad
1.741724e-05

> loss.fit.dist(x=x12, densfun = "beta", period = "quarters")

[1] "Argument scaled; x<- x/max(x)"
$shape1
[1] 1.175090

$shape2
[1] 1.228670

          ad
1.741724e-05
```



Empirical and fitted density: beta
Density
0.0e+00
1.0e−07
2.0e−07
2e+03
1e+04
5e+04
5e+05
5e+06
N = 17   Bandwidth = 7.144e+05



Empirical and fitted density: beta
Density
0.0e+00
1.0e−07
2.0e−07
−2e+06
0e+00
2e+06
4e+06
6e+06
N = 17   Bandwidth = 7.144e+05

Chapter 5

Value at Risk

We will estimate operational VaR for our data using `mc()` function. First, let us estimate it for some chosen cells.

5.1 Value at Risk for chosen cells

Let us start with `x32`:

5.1.1 "Commercial Banking/Clients, Products & Business Practices" cell

```
> l1 = mc(x32)$table

         Goodness-of-fit test for poisson distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 124.9302  4 4.723739e-26

         Goodness-of-fit test for binomial distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 169.8749  4 1.112539e-35

         Goodness-of-fit test for nbinomial distribution

                    X^2 df  P(> X^2)
Likelihood Ratio 3.08669  3 0.3784514
nbinomial
0.3784514

         Goodness-of-fit test for nbinomial distribution
```

```
                     X^2 df  P(> X^2)
Likelihood Ratio 3.08669  3 0.3784514
$size
[1] 0.2834215

$prob
[1] 0.5978599

[1] "Argument scaled; x<- x/max(x)"
[1] "log-normal"
$meanlog
[1] 8.19784

$sdlog
[1] 0.9626437
```

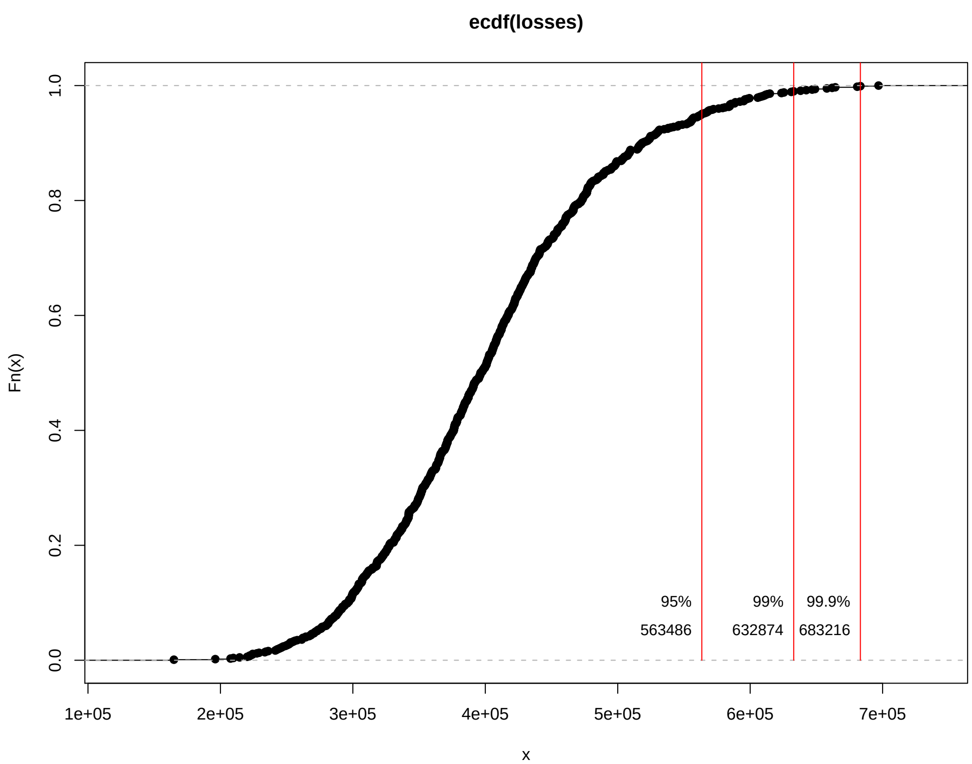


Figure 5.1: Empirical Cumulative Distribution Function for thousand simulated yearly losses for x32 data

These are our losses' quantiles:

```
> l1$q

     95%      99%    99.9%
563486.0 632874.1 683216.3
```

Of course one can change that confidence levels which is default to `p = c(0.95, 0.99, 0.999)`.

There are `1000` losses simulated:

```
> length(l1$losses)

[1] 1000

> head(l1$losses)

[1] 376690.0 430624.6 384730.4 421828.0 358475.7 351422.3
```

If missing, `period` becomes `days` and `iterate` becomes `years`.
Losses are simulated for `period` periods (here: days) and then summed to year losses, `iterate` is time interval (here: years) and `nmb` is number of `iterate` iterations.
In that example we have `nmb` (default to `1000`) `iterate` periods and that means `365*nmb ==365000` day losses to simulate. Losses are aggregated by `days`, then by `years`. We obtain `nmb` yearly losses.

In next example losses would be simulated only for `10 nmb`, `iterate = "quarters"` and `weeks` periods. That means that we have `nmb=10` quarters (`iterate = "quarters"`), `13` weeks a quarter makes `10*13=130` weeks and is equal to `130/52=2.5` years; we have always yearly losses simulated. That does not make full years and we have a warning message:

```
> l2 = mc(x32, period = "weeks", iterate = "quarters", nmb = 10)$table

[1] "note that these are not full years"

         Goodness-of-fit test for poisson distribution

                       X^2 df     P(> X^2)
Likelihood Ratio 28.42763  6 7.804346e-05

         Goodness-of-fit test for binomial distribution

                       X^2 df     P(> X^2)
Likelihood Ratio 65.80124  6 2.959408e-12
```

```
         Goodness-of-fit test for nbinomial distribution

                      X^2 df  P(> X^2)
Likelihood Ratio 4.644747  5 0.4607534
nbinomial
0.4607534

         Goodness-of-fit test for nbinomial distribution

                      X^2 df  P(> X^2)
Likelihood Ratio 4.644747  5 0.4607534
$size
[1] 2.092517

$prob
[1] 0.6110669

[1] "Argument scaled; x<- x/max(x)"
[1] "exponential"
$rate
[1] 0.0001061786
```

Let us see function `mc()` values: simulated losses, computed quantiles and `ad` value for fitted distribution:

```
> l2

$losses
[1] 762077.6 585109.8 729420.7

$q
     95%      99%    99.9%
758811.9 761424.4 762012.2

$ad
         ad
0.001141128
```

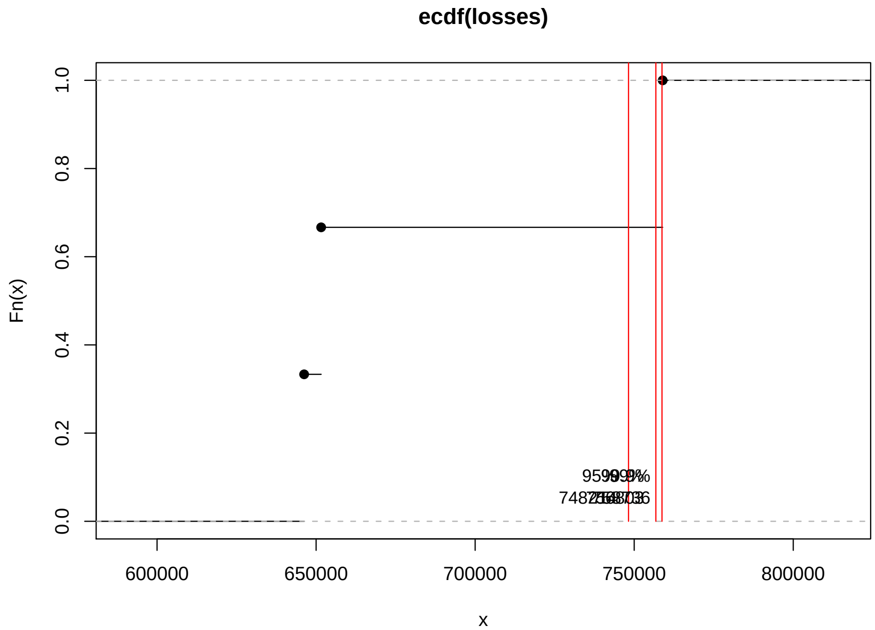


Figure 5.2: Empirical Cumulative Distribution Function for three simulated yearly losses for x32 data

...but we have `3` simulated year losses. How is that? Let us see:

```
> matrix(c(1,2,3,4,4,4),3)

     [,1] [,2]
[1,]    1    4
[2,]    2    4
[3,]    3    4
```

That makes from `c(1,2,3,4,4,4)` matrix with `3` rows. And now:

```
> matrix(c(1,2,3,4,4,4,5),3)

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

We have a warning message, but matrix is created because: "If there are too few elements in data to fill the array, then the elements in data are recycled.";

see `help(matrix)`.
That shows that we should be aware that our data can be not exactly entirely simulated when using inadequate `nmb`, `iterate` or `period`.

5.1.2 "Agency Services"/"Clients, Products & Business Practices" cell

Let us have our `x12` data. We know best frequency and severity distribution to fit to that data, so we will give them as function arguments:

```
> l3 = mc(x12, rfun = "log-normal", type = "nbinomial")$table

         Goodness-of-fit test for nbinomial distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 23.01709  5 0.0003350356
$size
[1] 0.4483843

$prob
[1] 0.4544358

$meanlog
[1] 10.02143

$sdlog
[1] 1.636008

> length(l3$losses)

[1] 1000
```

Let us see quantiles:

```
> l3$q

     95%      99%    99.9%
24397159 29449732 33912439
```

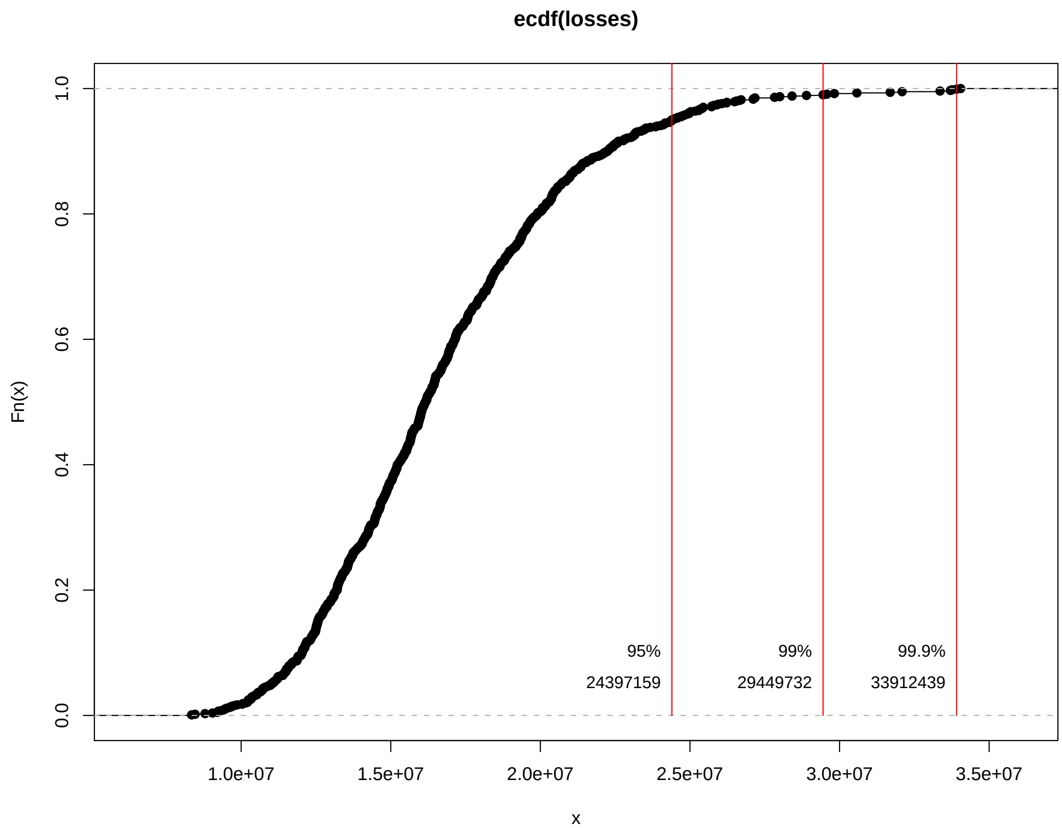


Figure 5.3: Empirical Cumulative Distribution Function for thousand simulated yearly losses for x12 data

5.2 All business lines

And now for all business lines. First of all we will assign losses to business lines:

```
> b.loss <- {}
> for(b in 1: length(loss.data.object$blines)){
+         b.loss[[b]]<- read.loss(b=b,r=1,loss.data.object)
+ for(r in 2: length(loss.data.object$rcateg)){
+         b.loss[[b]]<- rbind(b.loss[[b]],read.loss(b=b,r=r,loss.data.object))
+ }
+ }
```

`b[[i]]` are all losses assigned to `loss.data.object$blines[i]`. These are total number of losses in business lines; we can compare our result with `loss.matrix.image(data = loss.data.object)` to see that.

```
> for(i in 1: length(loss.data.object$blines)){
+ print(paste(loss.data.object$blines[i],dim(b.loss[[i]])[1]))
+ }
```

```
[1] "Agency Services 988"
[1] "Asset Management 312"
[1] "Commercial Banking 464"
[1] "Corporate Finance 198"
[1] "Payment & Settlement 716"
[1] "Retail Banking 35"
[1] "Retail Brokerage 704"
[1] "Trading & Sales 192"
```

5.2.1 "Agency Services"

For `"Agency Services"` we can check all `flist` distributions apart `"inverse gaussian"` using that command:

```
>  mc(b.loss[[1]], flist =  c("beta", "cauchy", "chi-squared", "exponential", "f",
+                   "gamma", "log-normal", "logistic", "normal", "weibull"), nmb=1)
```

...and the best is `"weibull"`; `"inverse gaussian"`, not running with any of `fitdistr` method, is no better - we use method of moments.

Compare `"weibull"` with `"inverse gaussian"` for `b.loss[[1]]`:

```
> x<- period.loss(b.loss[[1]],"days")
> m <- mean(x)
> v <- var(x)
> lambda = max(m^3/v, 0.1^(100))
> nu = max(m, 0.1^(100))
> par(mfrow = c(2,2))
> loss.fit.dist("weibull",x,xlog.scale=T)
```

```
$shape
[1] 0.611712

$scale
[1] 53922.56

          ad
3.713546e-05
```

```
> fit.plot(x,dinvGauss,distname = "i.g.",param = list(lambda = lambda, nu = nu),log="x")
```

```
[1] 5.641139e-05
```

```
> loss.fit.dist("weibull",x)
```

```
$shape
[1] 0.611712

$scale
[1] 53922.56

          ad
3.713546e-05

> fit.plot(x,dinvGauss,distname = "i.g.",param = list(lambda = lambda, nu = nu))

[1] 5.641139e-05
```

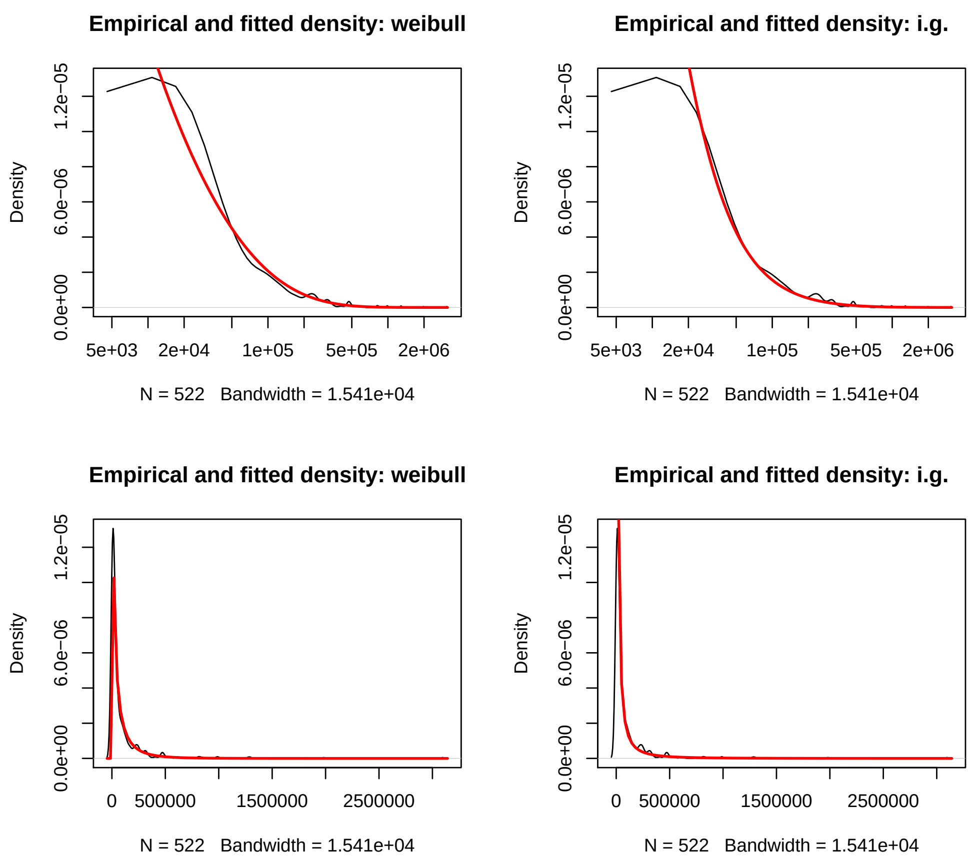
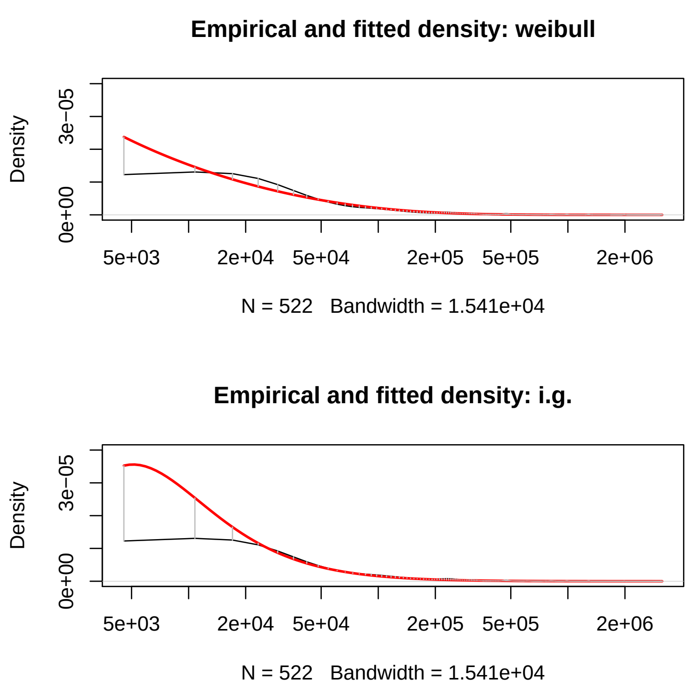


Figure 5.4: Inverse gaussian and weibull fits for ”Agency Services” loss data

Both seem very good but "inverse gaussian" could seem better. Why it was not chosen? Below we have an answer on the plot

```
> par(mfrow = c(2,1))
> loss.fit.dist("weibull",x,xlog.scale=T,draw.diff=T,ylim = c(0,40e-06))

$shape
[1] 0.611712

$scale
[1] 53922.56

          ad
3.713546e-05

> fit.plot(x,dinvGauss,distname = "i.g.",param = list(lambda = lambda, nu = nu),
+           log="x",draw.diff=T,ylim = c(0,40e-06))

[1] 5.641139e-05
```



Maybe if we want better tail estimation, "inverse gaussian" could be a bit better. However, we will choose "weibull" this time:

```
> b1 <- mc(b.loss[[1]], rfun = "weibull")$table

         Goodness-of-fit test for poisson distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 377.8318  7 1.348104e-77

         Goodness-of-fit test for binomial distribution

                      X^2 df      P(> X^2)
Likelihood Ratio 530.4053  7 2.319596e-110

         Goodness-of-fit test for nbinomial distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 22.55354  6 0.0009606621
   nbinomial
0.0009606621

         Goodness-of-fit test for nbinomial distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 22.55354  6 0.0009606621
$size
[1] 0.5730788

$prob
[1] 0.464733

$shape
[1] 0.611712

$scale
[1] 53922.56

> b1$q

     95%      99%    99.9%
23757338 25758312 28510517
```

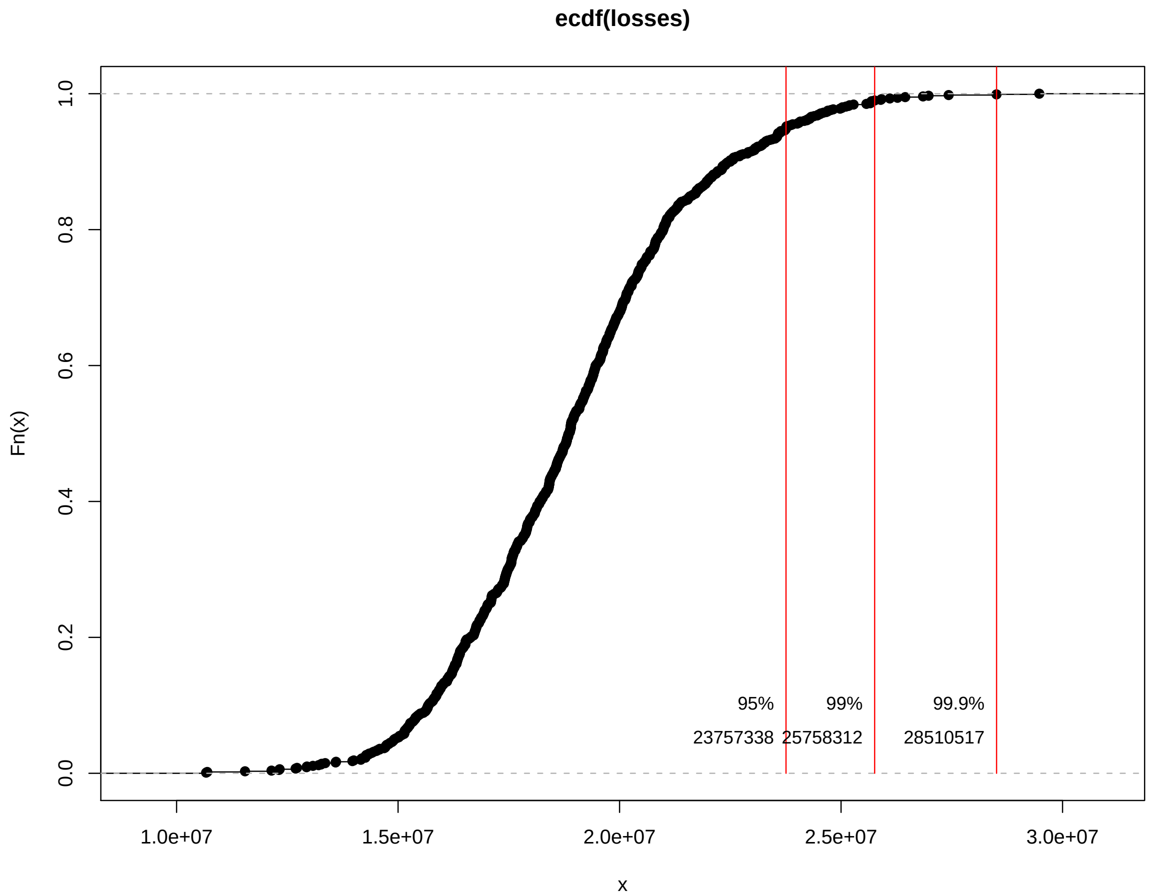


Figure 5.5: Empirical Cumulative Distribution Function for thousand simulated yearly losses for ”Agency Services”

5.2.2 ”Asset Management”

For `b.loss[[2]]` we have `"cauchy"` distribution being the best in respect of `ad`, but `"log-normal"` still could be better. Let us use `loss.fit.dist()` function:

```
> par(mfrow = c(2, 1))
> x <- period.loss(b.loss[[2]], "days")
> loss.fit.dist("log-normal", b.loss[[2]], xlog.scale = T,
+     n = 1000, ylim = c(0, 5e-05))

$meanlog
[1] 9.656914
```

```
$sdlog
[1] 1.844185

          ad
7.532465e-05

> loss.fit.dist("cauchy", b.loss[[2]], xlog.scale = T,
+     n = 1000, ylim = c(0, 5e-05))

$location
[1] 11273.96

$scale
[1] 22714.10

          ad
7.237527e-05
```

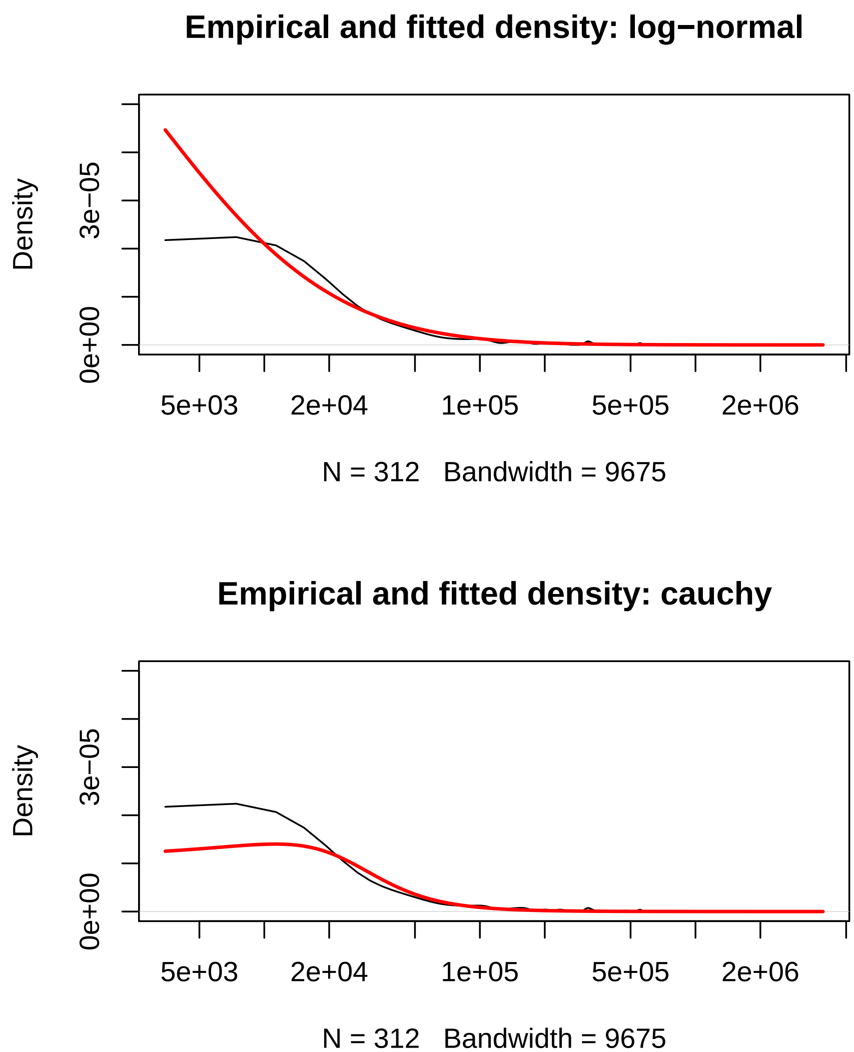


Figure 5.6: Cauchy and log-normal fits for "Asset Management"

As we could see, `"log-normal"` density values are rather more than empirical density values while `"cauchy"` density gives values rather less than empirical. Perhaps it would be more safe to have `"lognormal"`:

```
> b2 <- mc(b.loss[[2]], rfun = "log-normal")$table

        Goodness-of-fit test for poisson distribution
```

```
                      X^2 df     P(> X^2)
Likelihood Ratio 208.9491  5 3.454384e-43

         Goodness-of-fit test for binomial distribution

                     X^2 df     P(> X^2)
Likelihood Ratio 266.269  5 1.770535e-55

         Goodness-of-fit test for nbinomial distribution

                     X^2 df  P(> X^2)
Likelihood Ratio 6.64929  4 0.1556236
nbinomial
0.1556236

         Goodness-of-fit test for nbinomial distribution

                     X^2 df  P(> X^2)
Likelihood Ratio 6.64929  4 0.1556236
$size
[1] 0.2164420

$prob
[1] 0.4979963

$meanlog
[1] 10.17238

$sdlog
[1] 1.902329
```

> b2$q

```
     95%      99%    99.9%
25419740 40447504 79467922
```

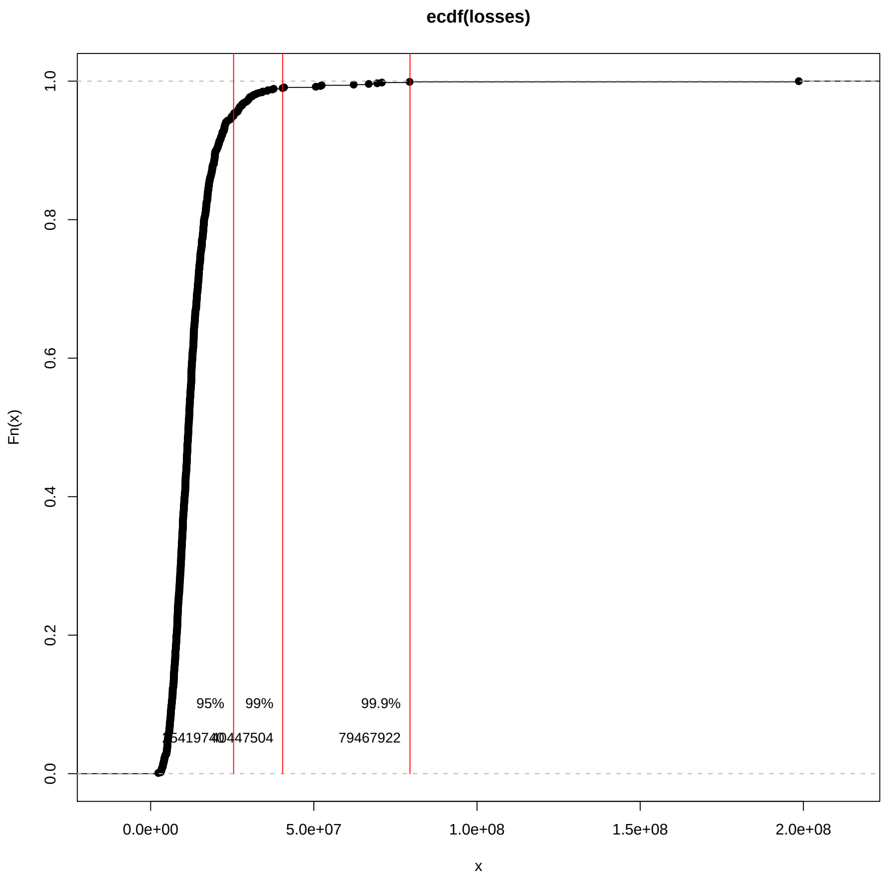


Figure 5.7: Empirical Cumulative Distribution Function for thousand simulated yearly losses for "Asset Management"

There are big differences between different quantiles; see `b2$q` and 5.7 figure. It is no better, or even worse, when using `"cauchy"`.

5.2.3 "Commercial Banking"

For `b.loss[[3]]` we have `"log-normal"` distribution; `"inverse gaussian"` is very good, too.

```
> b3 <- mc(b.loss[[3]], rfun = "log-normal")$table

        Goodness-of-fit test for poisson distribution

                     X^2 df     P(> X^2)
```

```
Likelihood Ratio 173.4832  4 1.869881e-36

         Goodness-of-fit test for binomial distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 253.3477  4 1.236929e-53

         Goodness-of-fit test for nbinomial distribution

                      X^2 df   P(> X^2)
Likelihood Ratio 8.722379  3 0.03321907
 nbinomial
0.03321907

         Goodness-of-fit test for nbinomial distribution

                      X^2 df   P(> X^2)
Likelihood Ratio 8.722379  3 0.03321907
$size
[1] 0.4113998

$prob
[1] 0.5711386

$meanlog
[1] 8.331045

$sdlog
[1] 0.9357786
```

```
> b3$q

     95%       99%     99.9%
 932310.2 1054075.5 1168380.9
```

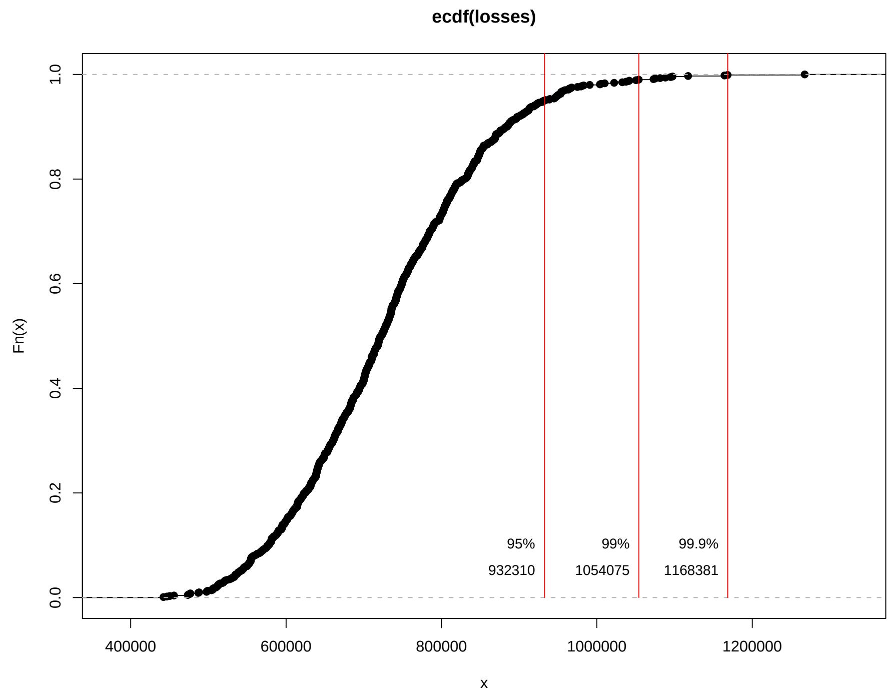


Figure 5.8: Empirical Cumulative Distribution Function for thousand simulated yearly losses for "Commercial Banking"

5.2.4 "Corporate Finance"

For `b.loss[[4]]` we have `"log-normal"` distribution, but, again, `"inverse gaussian"` is very good, too.

```
> b4 <- mc(b.loss[[4]], rfun = "log-normal")$table

         Goodness-of-fit test for poisson distribution

                      X^2 df      P(> X^2)
Likelihood Ratio 95.11894  3 1.740823e-20

         Goodness-of-fit test for binomial distribution

                      X^2 df      P(> X^2)
Likelihood Ratio 130.0241  3 5.344058e-28

         Goodness-of-fit test for nbinomial distribution
```

```
                     X^2 df  P(> X^2)
Likelihood Ratio 3.90774  2 0.1417245
nbinomial
0.1417245

         Goodness-of-fit test for nbinomial distribution

                     X^2 df  P(> X^2)
Likelihood Ratio 3.90774  2 0.1417245
$size
[1] 0.2181131

$prob
[1] 0.6194936

$meanlog
[1] 8.306907

$sdlog
[1] 0.9849417

> b4$q

     95%      99%    99.9%
472351.9 549452.4 700082.3
```

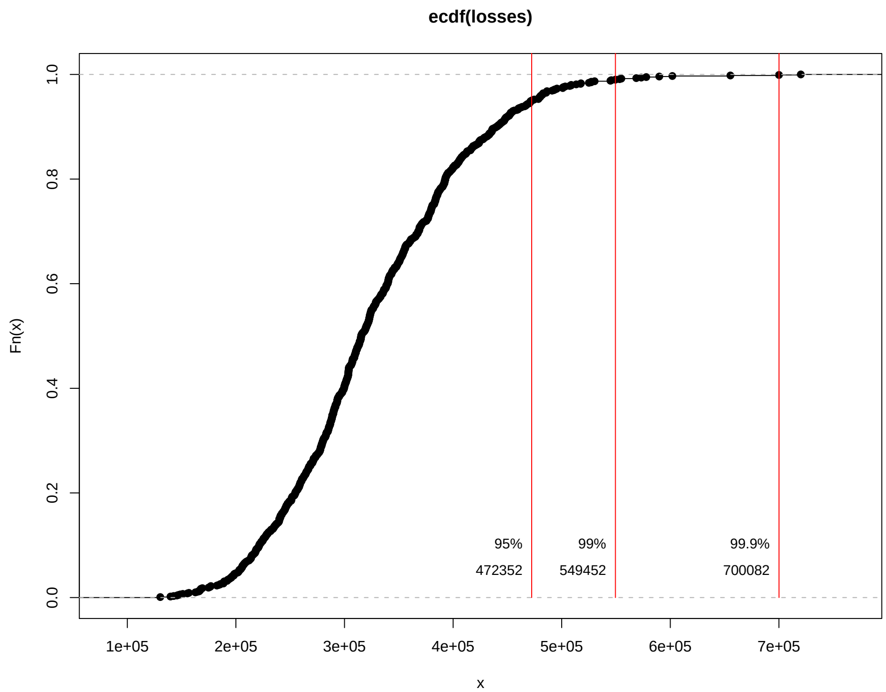


Figure 5.9: Empirical Cumulative Distribution Function for thousand simulated yearly losses for ”Corporate Finance”

5.2.5 ”Payment & Settlement”

And the same with `b.loss[[5]]`:

```
> b5 <- mc(b.loss[[5]], rfun = "log-normal")$table

         Goodness-of-fit test for poisson distribution

                     X^2 df     P(> X^2)
Likelihood Ratio 280.0119  5 1.978837e-58

         Goodness-of-fit test for binomial distribution

                     X^2 df     P(> X^2)
Likelihood Ratio 418.9128  5 2.485152e-88

         Goodness-of-fit test for nbinomial distribution

                     X^2 df   P(> X^2)
```

```
Likelihood Ratio 10.18445  4 0.03743263
 nbinomial
0.03743263

         Goodness-of-fit test for nbinomial distribution

                      X^2 df   P(> X^2)
Likelihood Ratio 10.18445  4 0.03743263
$size
[1] 0.4899842

$prob
[1] 0.5052239

$meanlog
[1] 9.325801

$sdlog
[1] 1.296436

> b5$q

    95%     99%   99.9%
6143788 6937562 7751518
```

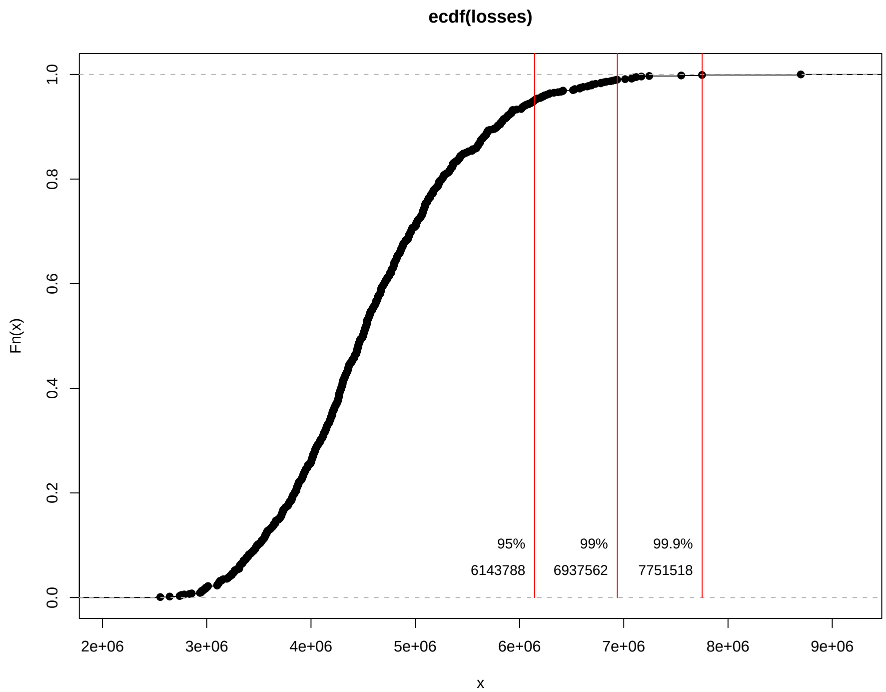


Figure 5.10: Empirical Cumulative Distribution Function for thousand simulated yearly losses for "Payment & Settlement"

5.2.6 "Retail Banking"

There are similar problems with `"cauchy"` for `b.loss[[6]]` as for business line `"Asset Management"` and `b.loss[[2]]`, so we will choose `"log-normal"` again:

```
> b6 <- mc(b.loss[[6]], rfun = "log-normal")$table

         Goodness-of-fit test for poisson distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 12.57757  1 0.0003904038

         Goodness-of-fit test for binomial distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 17.69507  1 2.592976e-05

         Goodness-of-fit test for nbinomial distribution
```

```
                      X^2 df P(> X^2)
Likelihood Ratio 1.295086  0        0
     poisson
0.0003904038

         Goodness-of-fit test for poisson distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 12.57757  1 0.0003904038
$lambda
[1] 0.02433936

$meanlog
[1] 8.502203

$sdlog
[1] 1.027541
```

> b6$q

```
     95%      99%    99.9%
152303.6 223119.8 389235.0
```

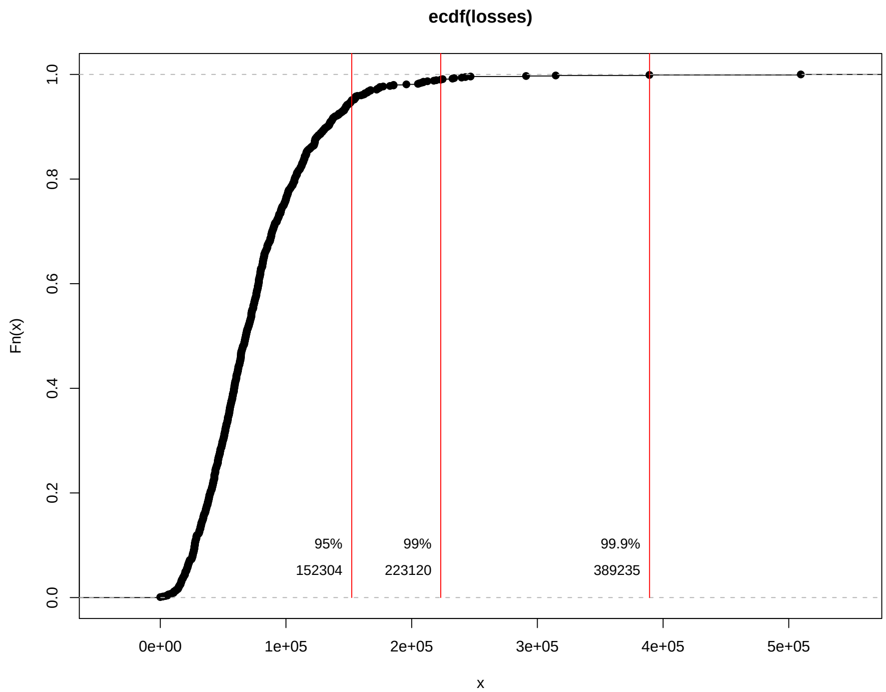


Figure 5.11: Empirical Cumulative Distribution Function for thousand simulated yearly losses for "Retail Banking"

5.2.7 "Retail Brokerage"

We have `"log-normal"` again for `b.loss[[7]]`:

```
> b7 <- mc(b.loss[[7]], rfun = "log-normal")$table

         Goodness-of-fit test for poisson distribution

                        X^2 df     P(> X^2)
Likelihood Ratio 223.5155  5 2.623081e-46

         Goodness-of-fit test for binomial distribution

                        X^2 df     P(> X^2)
Likelihood Ratio 335.5067  5 2.306178e-70

         Goodness-of-fit test for nbinomial distribution

                        X^2 df   P(> X^2)
```

```
Likelihood Ratio 12.11575  4 0.01651093
 nbinomial
0.01651093

         Goodness-of-fit test for nbinomial distribution

                      X^2 df   P(> X^2)
Likelihood Ratio 12.11575  4 0.01651093
$size
[1] 0.5650693

$prob
[1] 0.5454443

$meanlog
[1] 9.351597

$sdlog
[1] 1.265011
```

> b7$q

```
    95%     99%   99.9%
5867162 6651370 8242026
```

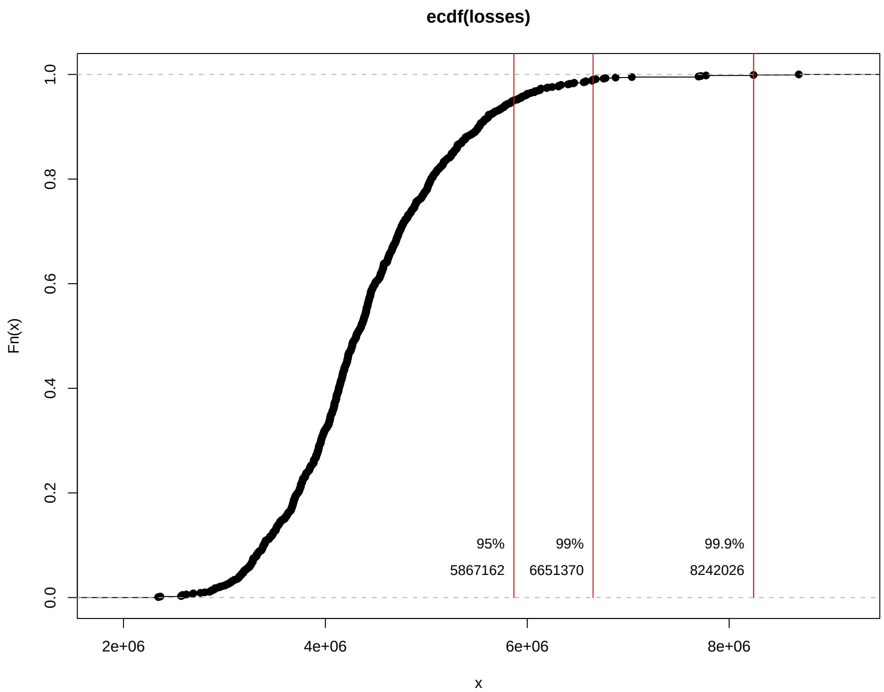


Figure 5.12: Empirical Cumulative Distribution Function for thousand simulated yearly losses for "Retail Brokerage"

5.2.8 "Trading & Sales"

...and for `b.loss[[8]]` too:

```
> b8 <- mc(b.loss[[8]], rfun = "log-normal")$table

         Goodness-of-fit test for poisson distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 116.2838  3 4.871768e-25

         Goodness-of-fit test for binomial distribution

                      X^2 df     P(> X^2)
Likelihood Ratio 157.1107  3 7.702578e-34

         Goodness-of-fit test for nbinomial distribution

                      X^2 df  P(> X^2)
```

```
Likelihood Ratio 4.099084  2 0.1287939
nbinomial
0.1287939

         Goodness-of-fit test for nbinomial distribution

                      X^2 df  P(> X^2)
Likelihood Ratio 4.099084  2 0.1287939
$size
[1] 0.1796925

$prob
[1] 0.5794828

$meanlog
[1] 8.490445

$sdlog
[1] 1.065124
```

> b8$q

```
     95%      99%    99.9%
622662.9 723614.4 831673.0
```

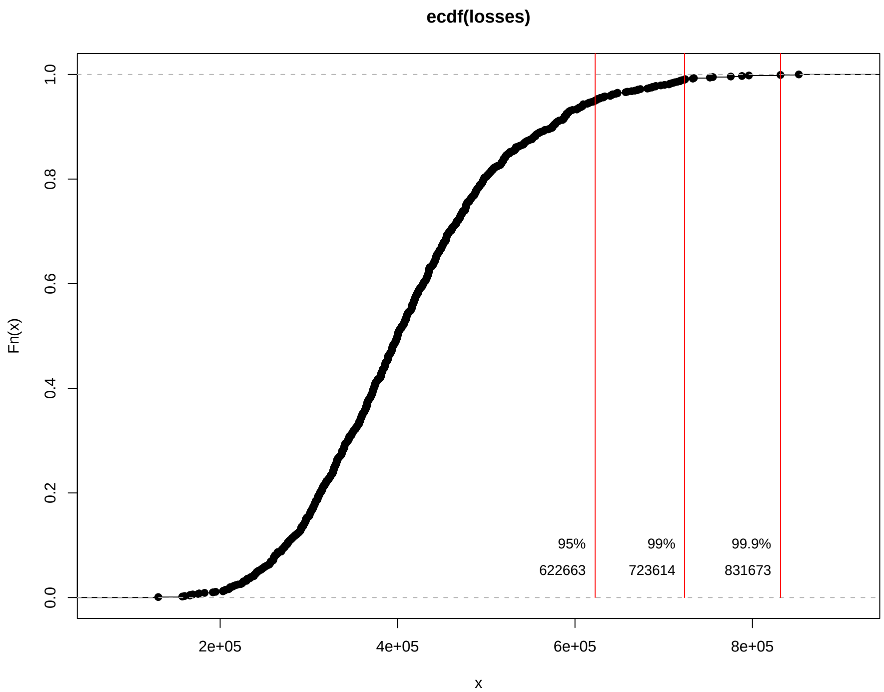


Figure 5.13: Empirical Cumulative Distribution Function for thousand simulated yearly losses for "Trading & Sales"

5.2.9 A summary

A short summary:

```
> b <- rbind(b1$q, b2$q, b3$q, b4$q, b5$q, b6$q, b7$q,
+     b8$q)
> b
```

```
            95%        99%      99.9%
[1,] 23757338.3 25758312.0 28510517.4
[2,] 25419739.6 40447503.7 79467922.3
[3,]   932310.2  1054075.5  1168380.9
[4,]   472351.9   549452.4   700082.3
[5,]  6143787.8  6937561.6  7751517.9
[6,]   152303.6   223119.8   389235.0
[7,]  5867162.1  6651369.7  8242026.3
[8,]   622662.9   723614.4   831673.0
```

These are our business lines' quantiles binded by rows. Let us sum loss data by business lines:

```
> b.loss.sum <- NULL
> for (i in 1:length(loss.data.object$blines)) {
+     b.loss.sum[i] <- sum(b.loss[[i]][, 2])
+ }
```

We have `b.loss.sum/4` being mean yearly losses for business lines in that about-four-years period:

```
> yearly.b.loss<- b.loss.sum/4
> yearly.b.loss

[1] 11766481.26  8901222.83   517504.47   261595.07
[5]  2617273.66    69289.07  2976987.77   343875.76
```

What amount do we need to keep in relation to that mean yearly losses for business lines?

```
> fraction1 <- b[, 1]/yearly.b.loss
> fraction1

[1] 2.019069 2.855758 1.801550 1.805660
[5] 2.347400 2.198090 1.970838 1.810721
```

And that means that we need about `2` times `yearly.b.loss[1]` amount for `Agency Services` to have `95` percent confidence. That amount is equal to:

```
> fraction1[1]*yearly.b.loss[1]

[1] 23757338
```

Let us see real yearly losses from first business line; `y` will be `"Agency Services"` loss data and `y2` is sum of losses from 2002; `min(as.Date(y[,1]) = "2002-01-08"`.

```
> y<- b.loss[[1]]
> y2<- sum(y[as.Date(y[,1])<as.Date("2003-01-01"),2])
> y2

[1] 3390290
```

. . . `y3` is sum of losses from `2003`:

```
> y3<- sum(y[as.Date("2003-01-01")<=as.Date(y[,1]) &
+                    as.Date(y[,1])<as.Date("2004-01-01"),2])
> y3

[1] 8792141
```

. . . `y4` is sum of losses from `2004`:

```
> y4<- sum(y[as.Date("2004-01-01")<=as.Date(y[,1]) &
+                    as.Date(y[,1])<as.Date("2005-01-01"),2])
> y4
```

```
[1] 12663557
```

...and `y5` is sum of losses from `2005`:

```
> y5<- sum(y[as.Date("2005-01-01")<=as.Date(y[,1]) &
+                    as.Date(y[,1])<as.Date("2006-01-01"),2])
> y5
```

```
[1] 19903260
```

There is also `2006` year loss data, but `max(as.Date(y[,1])) = "2006-02-12"`, so data from that year - `44` losses - is incomplete.

Let us compare output for `fraction1[1]*yearly.b.loss[1] = 23757338` with:

```
> yearly.b.loss[1] + 2*sd(c(y2,y3,y4,y5))
```

```
[1] 25655443
```

For normal distribution that would give about `96` percent confidence level. We have quite big standard deviation for our yearly losses:

```
> sd(c(y1,y2,y3,y4))
```

```
[1] 6944481
```

...and in relation of percentage to mean yearly loss for `"Agency Services"` it would be:

```
>100*sd(c(y1,y2,y3,y4))/yearly.b.loss[1]
```

```
[1] 59.01918
```

...therefore we could expect `fraction1[1]*yearly.b.loss[1]` value to be quite different from `yearly.b.loss[[1]]`.

Let us do the same computations for `99` and `99.9` percent confidence levels:

```
> fraction2 <- b[, 2]/yearly.b.loss
> fraction2
```

```
[1] 2.189126 4.544039 2.036843 2.100393
[5] 2.650683 3.220130 2.234262 2.104290
```

```
> fraction3 <- b[, 3]/yearly.b.loss
> fraction3

fraction3
[1] 2.423028 8.927753 2.257721 2.676206
[5] 2.961677 5.617553 2.768579 2.418528
```

Note that for second and sixth business lines amounts needed are increasing while changing confidence level rather fast. That was business lines with densities well fitted to `"cauchy"` distribution which has no mean nor standard deviation.

Of course VaR is not a coherent risk measure and `b` values should not be summed and compared with sum of losses without some further assumptions.

5.3 All cells

We will compute VaRs for every one business line/risk category non-empty cell. We need to choose best fit for every cell. Let us have zero matrix `m`:

```
> m <- matrix(0, length(loss.data.object$blines), length(loss.data.object$rcateg))
```

For every non-empty cell number of best fitted distribution from `flist` will be assigned to `m[i,j]` where `i`, `j` are business line number of that cell and risk category number of that cell respectively. List `flist` consists only from `5` distributions, because it happens that these are distribution best fitting that data. There are also good `"cauchy"` fits, but we exclude that distribution for similar reasons as before.

```
> flist = c("log-normal", "weibull", "gamma", "beta",  "exponential")
>  for (i in 1:8) {
+  for (j in 1:7) {
+         y <- read.loss(b = i, r = j, loss.data.object)
+         if (dim(y)[1] != 0) {
+         ad <- 100
+         value <- 1
+                 for (k in flist) {
+                 ad.new <- loss.fit.dist(densfun = k,
+                 x = y, period = "days", n = 10000)$ad
+                 if (ad.new < ad) {
+                   ad <- ad.new
+                   value <- which(flist == k)
+                 }
+             }
+             m[i, j] <- value
```

```
+           }
+ }
+ }
```

Obtained matrix:

```
> m
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    0    1    0    0    0    1    3
[2,]    0    0    0    0    2    0    1
[3,]    0    1    0    4    0    1    3
[4,]    0    0    4    0    1    1    0
[5,]    0    0    1    5    5    0    2
[6,]    0    0    0    0    1    0    4
[7,]    0    1    0    0    5    0    5
[8,]    0    0    1    0    0    0    0
```

For example: `m[5,7]=2` means that for 5th business line and 7th risk category cell we have `flist[2]= "weibull"` distribution fitted.

We can also sort values of that matrix …

```
> sort(as.vector(m))
```

```
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[29] 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 2 2 3 3 4 4 4 5 5 5 5
```

…so we can easily see that there are 34 empty cells and that we have `"lognormal"` fit 11 times, `"weibull"` fit 2 times, `"gamma"` fit 2 times, `"beta"` fit 3 times and `"exponential"` fit 4 times.

For every non-empty cell three quantiles will be computed using `mc()` function with argument `rfun =flist[m[i,j]]`, i, j being business line number of cell and risk category number of cell respectively. We would obtain sum of all non-empty cells quantiles for three confidence levels.

```
> q <- c(0, 0, 0)
> for (i in 1:length(loss.data.object$blines)) {
+     for (j in 1:length(loss.data.object$rcateg)) {
+         if (m[i, j] != 0) {
+             y <- read.loss(b = i, r = j, loss.data.object)
+             q.new <- mc(y, rfun = flist[m[i, j]])$table$q
+             q <- apply(rbind(q, q.new), 2, sum)
+         }
+     }
+ }
```

```
> sum.all.rect <- q
```

We obtain:

```
> sum.all.rect

      95%       99%     99.9%
 65876010  92248519 153280063
```

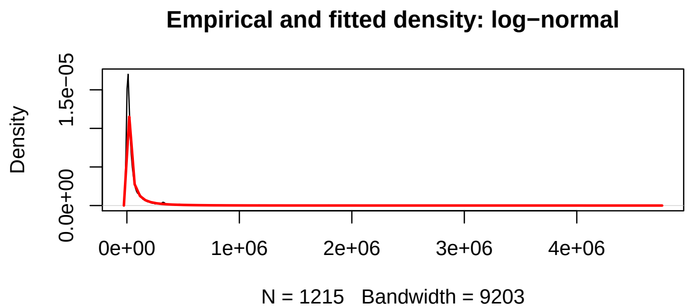
5.4 All losses

And now for all losses together without categorization:

```
> all.losses <- loss.data.object$losses[, 3:4]
> head(all.losses)

  First_Date_of_Event Gross_Loss_Amount
1          2002-01-03           1642.26
2          2002-01-06           2498.33
3          2002-01-08           7420.72
4          2002-01-09          27019.26
5          2002-01-10           1829.98
6          2002-01-11          12164.67
```

We will choose "log-normal" which is best fit for that data:

```
> par(mfrow = c(2, 1))
> loss.fit.dist("log-normal", all.losses, period = "days")

$meanlog
[1] 9.99138

$sdlog
[1] 1.638526

          ad
4.036907e-05

> loss.fit.dist("log-normal", all.losses, period = "days",
+     xlog.scale = T)

$meanlog
[1] 9.99138

$sdlog
[1] 1.638526

          ad
4.036907e-05
```



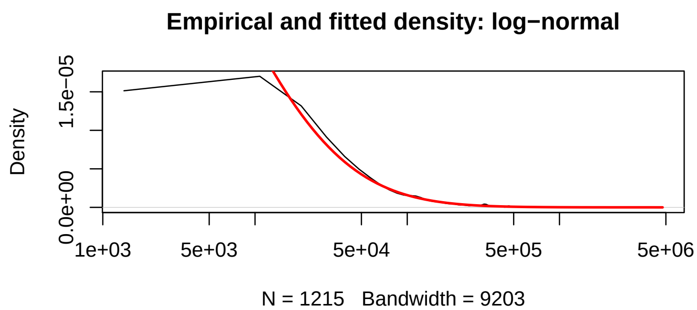


Figure 5.14: Lognormal fit for all losses; normal and logarithmic scales

And now we will compute VaR:

```
> sum.all <- mc(all.losses, "log-normal")$table$q

         Goodness-of-fit test for poisson distribution

                          X^2 df      P(> X^2)
Likelihood Ratio 415.5948 11 2.983464e-82

         Goodness-of-fit test for binomial distribution

                          X^2 df       P(> X^2)
Likelihood Ratio 879.2554 11 1.784866e-181

         Goodness-of-fit test for nbinomial distribution

                          X^2 df  P(> X^2)
Likelihood Ratio 12.80476 10 0.2347937
nbinomial
0.2347937
```

```
          Goodness-of-fit test for nbinomial distribution

                          X^2 df  P(> X^2)
Likelihood Ratio 12.80476 10 0.2347937
$size
[1] 2.389882

$prob
[1] 0.4987338

$meanlog
[1] 9.99138

$sdlog
[1] 1.638526
```

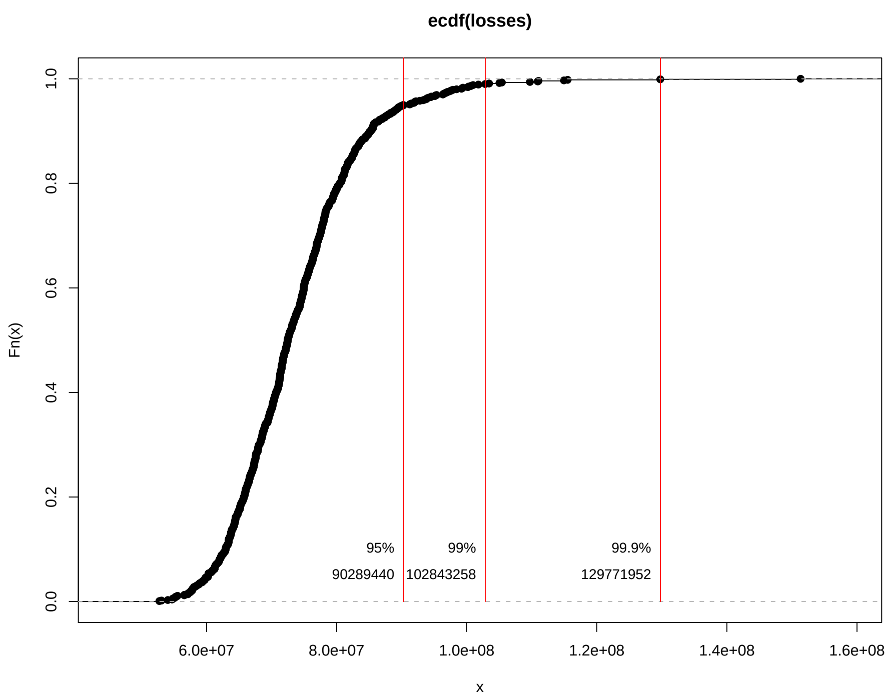


Figure 5.15: Empirical Cumulative Distribution Function for thousand yearly losses simulated for all data

5.5 Value at Risk for cells, business lines and for all losses

Let us summarize:

```
> sum.all.blines

      95%       99%     99.9%
 63367656  82345009 127061355

> sum.all.rect

      95%       99%     99.9%
 65876010  92248519 153280063

> sum.all

      95%       99%     99.9%
 90289440 102843258 129771952
```

Differences between relevant quantiles are not very big, given that we have not had any assumption and there is no universal relation between sum of VaRs computed for data divided by some categories and VaR computed for all data.

For reliable simulation we should have more information about that data to make some assumptions. For better VaR estimation we should compute also at least week estimation of frequency and severity and then simulate VaR, but it would cause a big growth of that guide content so to let it go at that.