Loss modeling features of **actuar**

Christophe Dutang
Université Paris Dauphine

Vincent Goulet
Université Laval

Mathieu Pigeon
Université du Québec à Montréal

1 Introduction

One important task of actuaries is the modeling of claim amount and claim count distributions for ratemaking, loss reserving or other risk evaluation purposes. Package **actuar** features many support functions for loss distributions modeling:

1. support for heavy tail continuous distributions useful in loss severity modeling;
2. support for phase-type distributions for ruin theory;
3. functions to compute raw moments, limited moments and the moment generating function (when it exists) of continuous distributions;
4. support for zero-truncated and zero-modified extensions of the discrete distributions commonly used in loss frequency modeling;
5. extensive support of grouped data;
6. functions to compute empirical raw and limited moments;
7. support for minimum distance estimation using three different measures;
8. treatment of coverage modifications (deductibles, limits, inflation, coinsurance).

Vignette "distributions" covers the points 1–4 above in great detail. This document concentrates on points 5–8.

2 Grouped data

Grouped data is data represented in an interval-frequency manner. Typically, a grouped data set will report that there were n_j claims in the interval $(c_{j-1}, c_j]$, $j = 1, \ldots, r$ (with the possibility that $c_r = \infty$). This representation is much more compact than an individual data set — where the value of each claim is known — but it also carries far less information. Now that storage space in computers has essentially become a non issue, grouped data has somewhat fallen out of fashion. Still, grouped data remains useful as a means to represent data, if only graphically — for example, a histogram is nothing but a density approximation for grouped data. Moreover, various parameter estimation techniques rely on grouped data.

For these reasons, **actuar** provides facilities to store, manipulate and summarize grouped data. A standard storage method is needed since there are many ways to represent grouped data in the computer: using a list or a matrix, aligning n_j with c_{j-1} or with c_j, omitting c_0 or not, etc. With appropriate extraction, replacement and summary methods, manipulation of grouped data becomes similar to that of individual data.

Function `grouped.data` creates a grouped data object similar to — and inheriting from — a data frame. The function accepts two types of input:

1. a vector of group boundaries $c_0, c_1, \ldots, c_r$ and one or more vectors of group frequencies $n_1, \ldots, n_r$ (note that there should be one more group boundary than group frequencies);

2. individual data $x_1, \ldots, x_n$ and either a vector of breakpoints $c_1, \ldots, c_r$, a number r of breakpoints or an algorithm to determine the latter.

In the second case, `grouped.data` will group the individual data using function `hist`. The function always assumes that the intervals are contiguous.

Example 1. Consider the following already grouped data set:

Group	Frequency (Line 1)	Frequency (Line 2)
(0, 25]	30	26
(25, 50]	31	33
(50, 100]	57	31
(100, 150]	42	19
(150, 250]	65	16
(250, 500]	84	11

We can conveniently and unambiguously store this data set in R as follows:

```
> x <- grouped.data(Group = c(0, 25, 50, 100,
+                             150, 250, 500),
+                   Line.1 = c(30, 31, 57, 42, 65, 84),
+                   Line.2 = c(26, 33, 31, 19, 16, 11))
```

Internally, object `x` is a list with class

```
> class(x)
[1] "grouped.data" "data.frame"
```

The package provides a suitable `print` method to display grouped data objects in an intuitive manner:

```
> x
       Group Line.1 Line.2
1   (0,  25]     30     26
2  (25,  50]     31     33
3  (50, 100]     57     31
4 (100, 150]     42     19
5 (150, 250]     65     16
6 (250, 500]     84     11
```

□

Example 2. Consider Data Set B of Klugman et al. (2012, Table 11.2):

27	82	115	126	155	161	243	294	340	384
457	680	855	877	974	1,193	1,340	1,884	2,558	15,743

We can represent this data set as grouped data using either an automatic or a suggested number of groups (see `?hist` for details):

```
> y <- c(  27,   82,  115,   126, 155, 161, 243,  294,
+         340,  384,  457,   680, 855, 877, 974, 1193,
```

```
+        1340, 1884, 2558, 15743)
> grouped.data(y)
                   y
1     (0,  2000] 18
2  (2000,  4000]  1
3  (4000,  6000]  0
4  (6000,  8000]  0
5  (8000, 10000]  0
6 (10000, 12000]  0
7 (12000, 14000]  0
8 (14000, 16000]  1
> grouped.data(y, breaks = 5)
                   y
1     (0,  5000] 19
2  (5000, 10000]  0
3 (10000, 15000]  0
4 (15000, 20000]  1
```

The above grouping methods use equi-spaced breaks. This is rarely appropriate for heavily skewed insurance data. For this reason, `grouped.data` also supports specified breakpoints (or group boundaries):

```
> grouped.data(y, breaks = c(0, 100, 200, 350, 750,
+                            1200, 2500, 5000, 16000))
                 y
1    (0,   100] 2
2  (100,   200] 4
3  (200,   350] 3
4  (350,   750] 3
5  (750,  1200] 4
6 (1200,  2500] 2
7 (2500,  5000] 1
8 (5000, 16000] 1
```

□

The package supports the most common extraction and replacement methods for "`grouped.data`" objects using the usual `[` and `[<-` operators. In particular, the following extraction operations are supported. (In the following, object `x` is the grouped data object of Example 1.)

i) Extraction of the vector of group boundaries (the first column):

```
> x[, 1]

[1]   0  25  50 100 150 250 500
```

ii) Extraction of the vector or matrix of group frequencies (the second and third columns):

```
> x[, -1]

  Line.1 Line.2
1     30     26
2     31     33
3     57     31
4     42     19
5     65     16
6     84     11
```

iii) Extraction of a subset of the whole object (first three lines):

```
> x[1:3, ]

      Group Line.1 Line.2
1  (0,  25]     30     26
2 (25,  50]     31     33
3 (50, 100]     57     31
```

Notice how extraction results in a simple vector or matrix if either of the group boundaries or the group frequencies are dropped.

As for replacement operations, the package implements the following.

i) Replacement of one or more group frequencies:

```
> x[1, 2] <- 22; x

       Group Line.1 Line.2
1   (0,  25]     22     26
2  (25,  50]     31     33
3  (50, 100]     57     31
4 (100, 150]     42     19
5 (150, 250]     65     16
6 (250, 500]     84     11

> x[1, c(2, 3)] <- c(22, 19); x
```

```
       Group Line.1 Line.2
1   (0,  25]     22     19
2  (25,  50]     31     33
3  (50, 100]     57     31
4 (100, 150]     42     19
5 (150, 250]     65     16
6 (250, 500]     84     11
```

ii) Replacement of the boundaries of one or more groups:

```
> x[1, 1] <- c(0, 20); x

       Group Line.1 Line.2
1   (0,  20]     22     19
2  (20,  50]     31     33
3  (50, 100]     57     31
4 (100, 150]     42     19
5 (150, 250]     65     16
6 (250, 500]     84     11

> x[c(3, 4), 1] <- c(55, 110, 160); x

       Group Line.1 Line.2
1   (0,  20]     22     19
2  (20,  55]     31     33
3  (55, 110]     57     31
4 (110, 160]     42     19
5 (160, 250]     65     16
6 (250, 500]     84     11
```

It is not possible to replace the boundaries and the frequencies simultaneously.

The mean of grouped data is

$$\hat{\mu} = \frac{1}{n} \sum_{j=1}^{r} a_j n_j, \tag{1}$$

where $a_j = (c_{j-1} + c_j)/2$ is the midpoint of the jth interval, and $n = \sum_{j=1}^{r} n_j$, whereas the variance is

$$\frac{1}{n} \sum_{j=1}^{r} n_j (a_j - \hat{\mu})^2. \tag{2}$$

The standard deviation is the square root of the variance. The package defines methods to easily compute the above descriptive statistics:

```
> mean(x)
Line.1 Line.2
 188.0  108.2
> var(x)
Line.1 Line.2
 25050  14945
> sd(x)
Line.1 Line.2
 158.3  122.3
```

Higher empirical moments can be computed with `emm`; see section 4.

The R function `hist` splits individual data into groups and draws an histogram of the frequency distribution. The package introduces a method for already grouped data. Only the first frequencies column is considered (see Figure 1 for the resulting graph):

```
> hist(x[, -3])
```

Remark 1. One will note that for an individual data set like `y` of Example 2, the following two expressions yield the same result:

```
> hist(y)
> hist(grouped.data(y))
```

R has a function `ecdf` to compute the empirical cdf $F_n(x)$ of an individual data set:

$$F_n(x) = \frac{1}{n}\sum_{j=1}^{n} I\{x_j \le x\}, \tag{3}$$

where $I\{\mathcal{A}\} = 1$ if $\mathcal{A}$ is true and $I\{\mathcal{A}\} = 0$ otherwise. The function returns a `"function"` object to compute the value of $F_n(x)$ in any x.

The approximation of the empirical cdf for grouped data is called an ogive (Klugman et al., 2012; Hogg and Klugman, 1984). It is obtained by joining the known values of $F_n(x)$ at group boundaries with straight line segments:

$$\tilde{F}_n(x) = \begin{cases} 0, & x \le c_0 \\ \dfrac{(c_j - x)F_n(c_{j-1}) + (x - c_{j-1})F_n(c_j)}{c_j - c_{j-1}}, & c_{j-1} < x \le c_j \\ 1, & x > c_r. \end{cases} \tag{4}$$

The package includes a generic function `ogive` with methods for individual and for grouped data. The function behaves exactly like `ecdf`.

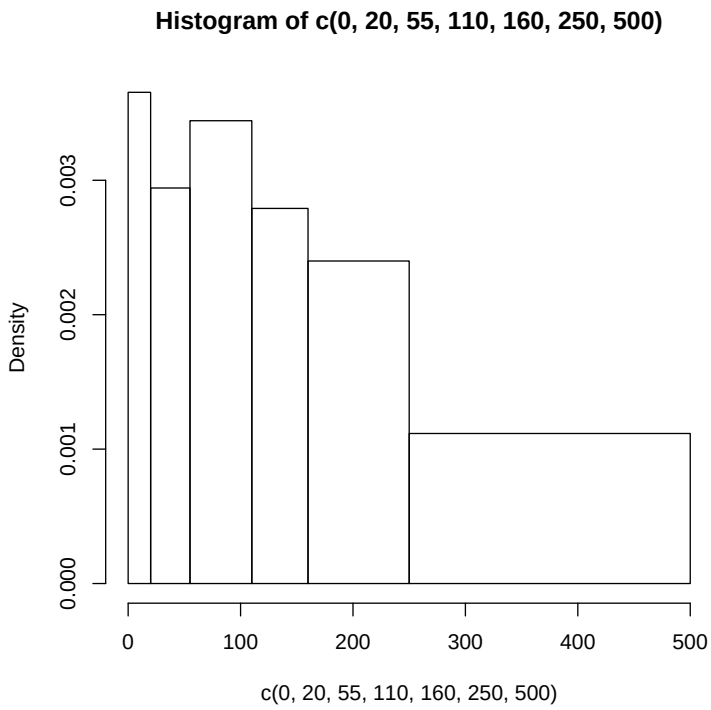

Figure 1: Histogram of a grouped data object

Example 3. Consider first the grouped data set of Example 1. Function `ogive` returns a function to compute the ogive $\tilde{F}_n(x)$ in any point:

```
> (Fnt <- ogive(x))
Ogive for grouped data
Call: ogive(x = x)
    x =   0,  20,  55,  ..., 2.5e+02, 5e+02
 F(x) =   0, 0.073, 0.18,  ..., 0.72,   1
```

Methods for functions `knots` and `plot` allow, respectively, to obtain the knots $c_0, c_1, \ldots, c_r$ of the ogive and to draw a graph (see Figure 2):

```
> knots(Fnt)
[1]   0  20  55 110 160 250 500
> Fnt(knots(Fnt))
[1] 0.00000 0.07309 0.17608 0.36545 0.50498 0.72093
```

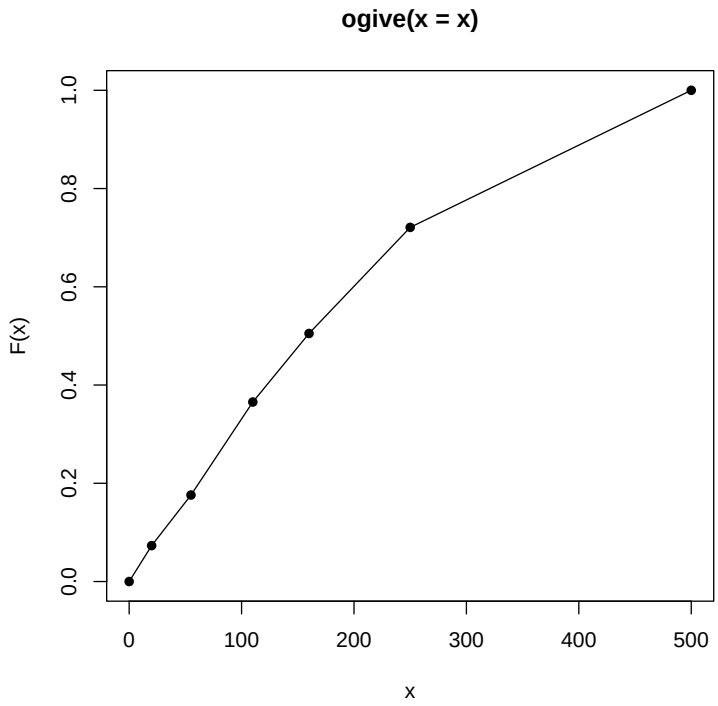

Figure 2: Ogive of a grouped data object

```
[7] 1.00000
> plot(Fnt)
```

To add further symmetry between functions `hist` and `ogive`, the latter also accepts in argument a vector individual data. It will call `grouped.data` and then computes the ogive. (Below, y is the individual data set of Example 2.)

```
> (Fnt <- ogive(y))
Ogive for grouped data
Call: ogive(x = y)
    x =   0, 2e+03, 4e+03,  ..., 1.4e+04, 1.6e+04
 F(x) =   0, 0.9, 0.95,  ..., 0.95,   1
> knots(Fnt)
[1]     0  2000  4000  6000  8000 10000 12000 14000
[9] 16000
```

□

A method of function `quantile` for grouped data objects returns linearly smoothed quantiles, that is, the inverse of the ogive evaluated at various points:

```
> quantile(x)
    0%    25%    50%    75%   100%
  0.00  76.47 158.21 276.04 500.00
> Fnt(quantile(x))
[1] 0.00 0.25 0.50 0.75 1.00
```

Finally, a `summary` method for grouped data objects returns the quantiles and the mean, as is usual for individual data:

```
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0    76.5   158.2   188.0   276.0   500.0
```

3 Data sets

This is certainly not the most spectacular feature of **actuar**, but it remains useful for illustrations and examples: the package includes the individual dental claims and grouped dental claims data of Klugman et al. (2012):

```
> data("dental"); dental
 [1]  141   16   46   40  351  259  317 1511  107
[10]  567
> data("gdental"); gdental
             cj nj
1      (0,  25] 30
2    ( 25,  50] 31
3    ( 50, 100] 57
4    (100, 150] 42
5    (150, 250] 65
6    (250, 500] 84
7   (500, 1000] 45
8  (1000, 1500] 10
9  (1500, 2500] 11
10 (2500, 4000]  3
```

4 Calculation of empirical moments

The package provides two functions useful for estimation based on moments. First, function `emm` computes the kth empirical moment of a sample, whether in individual or grouped data form. For example, the following expressions compute the first three moments for individual and grouped data sets:

```
> emm(dental, order = 1:3)
[1] 3.355e+02 2.931e+05 3.729e+08
> emm(gdental, order = 1:3)
[1] 3.533e+02 3.577e+05 6.586e+08
```

Second, in the same spirit as `ecdf` and `ogive`, function `elev` returns a function to compute the empirical limited expected value — or first limited moment — of a sample for any limit. Again, there are methods for individual and grouped data (see Figure 3 for the graphs):

```
> lev <- elev(dental)
> lev(knots(lev))
 [1]  16.0  37.6  42.4  85.1 105.5 164.5 187.7 197.9
 [9] 241.1 335.5
> plot(lev, type = "o", pch = 19)
> lev <- elev(gdental)
> lev(knots(lev))
 [1]   0.00  24.01  46.00  84.16 115.77 164.85
 [7] 238.26 299.77 324.90 347.39 353.34
> plot(lev, type = "o", pch = 19)
```

5 Minimum distance estimation

Two methods are widely used by actuaries to fit models to data: maximum likelihood and minimum distance. The first technique applied to individual data is well covered by function `fitdistr` of the package **MASS** (Venables and Ripley, 2002).

The second technique minimizes a chosen distance function between theoretical and empirical distributions. Package **actuar** provides function `mde`, very similar in usage and inner working to `fitdistr`, to fit models according to any of the following three distance minimization methods.

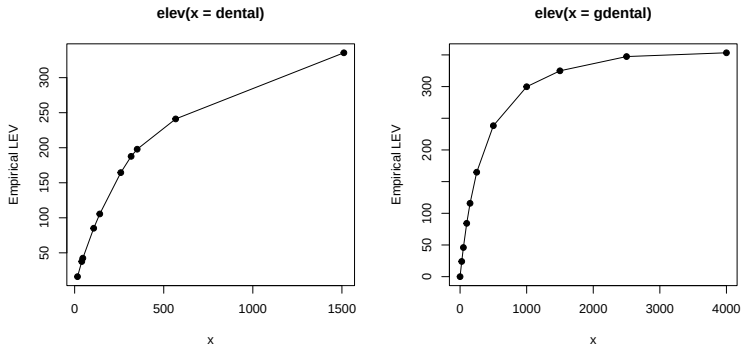

Figure 3: Empirical limited expected value function of an individual data object (left) and a grouped data object (right)

1. The Cramér-von Mises method (`CvM`) minimizes the squared difference between the theoretical cdf and the empirical cdf or ogive at their knots:

$$d(\theta) = \sum_{j=1}^{n} w_j [F(x_j;\theta) - F_n(x_j;\theta)]^2 \tag{5}$$

   for individual data and

$$d(\theta) = \sum_{j=1}^{r} w_j [F(c_j;\theta) - \tilde{F}_n(c_j;\theta)]^2 \tag{6}$$

   for grouped data. Here, $F(x)$ is the theoretical cdf of a parametric family, $F_n(x)$ is the empirical cdf, $\tilde{F}_n(x)$ is the ogive and $w_1 \geq 0, w_2 \geq 0, \ldots$ are arbitrary weights (defaulting to 1).

2. The modified chi-square method (`chi-square`) applies to grouped data only and minimizes the squared difference between the expected and observed frequency within each group:

$$d(\theta) = \sum_{j=1}^{r} w_j [n(F(c_j;\theta) - F(c_{j-1};\theta)) - n_j]^2, \tag{7}$$

   where $n = \sum_{j=1}^{r} n_j$. By default, $w_j = n_j^{-1}$.

3. The layer average severity method (`LAS`) applies to grouped data only and minimizes the squared difference between the theoretical and empirical lim-

ited expected value within each group:

$$d(\theta) = \sum_{j=1}^{r} w_j [\mathrm{LAS}(c_{j-1}, c_j; \theta) - \tilde{\mathrm{LAS}}_n(c_{j-1}, c_j; \theta)]^2, \tag{8}$$

where $\mathrm{LAS}(x, y) = E[X \wedge y] - E[X \wedge x]$, $\tilde{\mathrm{LAS}}_n(x, y) = \tilde{E}_n[X \wedge y] - \tilde{E}_n[X \wedge x]$ and $\tilde{E}_n[X \wedge x]$ is the empirical limited expected value for grouped data.

The arguments of `mde` are a data set, a function to compute $F(x)$ or $E[X \wedge x]$, starting values for the optimization procedure and the name of the method to use. The empirical functions are computed with `ecdf`, `ogive` or `elev`.

Example 4. The expressions below fit an exponential distribution to the grouped dental data set, as per example 2.21 of Klugman et al. (1998):

```
> mde(gdental, pexp, start = list(rate = 1/200),
+     measure = "CvM")
   rate
  0.003551

 distance
  0.002842
> mde(gdental, pexp, start = list(rate = 1/200),
+     measure = "chi-square")
   rate
  0.00364

 distance
     13.54
> mde(gdental, levexp, start = list(rate = 1/200),
+     measure = "LAS")
   rate
  0.002966

 distance
     694.5
```

□

It should be noted that optimization is not always as simple to achieve as in Example 4. For example, consider the problem of fitting a Pareto distribution to the same data set using the Cramér–von Mises method:

```
> mde(gdental, ppareto,
+     start = list(shape = 3, scale = 600),
+     measure = "CvM")
```

```
Error in mde(gdental, ppareto, start = list(shape = 3,
             scale = 600), measure = "CvM") :
  l'optimisation a échoué
```

Working in the log of the parameters often solves the problem since the optimization routine can then flawlessly work with negative parameter values:

```
> pparetolog <- function(x, logshape, logscale)
+     ppareto(x, exp(logshape), exp(logscale))
> (p <- mde(gdental, pparetolog,
+           start = list(logshape = log(3),
+                        logscale = log(600)),
+           measure = "CvM"))
 logshape   logscale
    1.581      7.128

  distance
  0.0007905
```

The actual estimators of the parameters are obtained with

```
> exp(p$estimate)
logshape logscale
   4.861 1246.485
```

This procedure may introduce additional bias in the estimators, though.

6 Coverage modifications

Let X be the random variable of the actual claim amount for an insurance policy, Y^L be the random variable of the amount paid per loss and Y^P be the random variable of the amount paid per payment. The terminology for the last two random variables refers to whether or not the insurer knows that a loss occurred. Now, the random variables X, Y^L and Y^P will differ if any of the following coverage modifications are present for the policy: an ordinary or a franchise deductible, a limit, coinsurance or inflation adjustment (see Klugman et al., 2012, chapter 8 for precise definitions of these terms). Table 1 summarizes the defi-

Coverage modification	Per-loss variable (Y^L)	Per-payment variable (Y^P)
Ordinary deductible (d)	$\begin{cases} 0, & X \leq d \\ X - d, & X > d \end{cases}$	$\begin{cases} X - d, & X > d \end{cases}$
Franchise deductible (d)	$\begin{cases} 0, & X \leq d \\ X, & X > d \end{cases}$	$\begin{cases} X, & X > d \end{cases}$
Limit (u)	$\begin{cases} X, & X \leq u \\ u, & X > u \end{cases}$	$\begin{cases} X, & X \leq u \\ u, & X > u \end{cases}$
Coinsurance (α)	αX	αX
Inflation (r)	$(1 + r)X$	$(1 + r)X$

Table 1: Coverage modifications for per-loss variable (Y^L) and per-payment variable (Y^P) as defined in Klugman et al. (2012).

nitions of Y^L and Y^P.

Often, one will want to use data $Y_1^P, \ldots, Y_n^P$ (or $Y_1^L, \ldots, Y_n^L$) from the random variable Y^P (Y^L) to fit a model on the unobservable random variable X. This requires expressing the pdf or cdf of Y^P (Y^L) in terms of the pdf or cdf of X. Function `coverage` of **actuar** does just that: given a pdf or cdf and any combination of the coverage modifications mentioned above, `coverage` returns a function object to compute the pdf or cdf of the modified random variable. The function can then be used in modeling like any other `dfoo` or `pfoo` function.

Example 5. Let Y^P represent the amount paid by an insurer for a policy with an ordinary deductible d and a limit $u - d$ (or maximum covered loss of u). Then the definition of Y^P is

$$Y^P = \begin{cases} X - d, & d \leq X \leq u \\ u - d, & X \geq u \end{cases} \tag{9}$$

and its pdf is

$$f_{Y^P}(y) = \begin{cases} 0, & y = 0 \\ \dfrac{f_X(y + d)}{1 - F_X(d)}, & 0 < y < u - d \\ \dfrac{1 - F_X(u)}{1 - F_X(d)}, & y = u - d \\ 0, & y > u - d. \end{cases} \tag{10}$$

Assume X has a gamma distribution. Then an R function to compute the pdf (10) in any y for a deductible $d = 1$ and a limit $u = 10$ is obtained with `coverage` as follows:

```
> f <- coverage(pdf = dgamma, cdf = pgamma,
+               deductible = 1, limit = 10)
> f
function (x, shape, rate = 1, scale = 1/rate)
{
    Call <- match.call()
    Sd <- Call
    Sd$lower.tail <- FALSE
    Sd[[1L]] <- as.name("pgamma")
    names(Sd)[2L] <- "q"
    Sd[[2L]] <- 1
    Su <- Sd
    Su[[2L]] <- 10
    f <- Call
    f[[1L]] <- as.name("dgamma")
    res <- numeric(length(x))
    w <- which(0 < x & x < 9)
    f[[2L]] <- x[w] + 1
    res[w] <- eval.parent(f)/(p <- eval.parent(Sd))
    res[x == 9] <- eval.parent(Su)/p
    res
}
<environment: 0x12ac351a8>
> f(0, shape = 5, rate = 1)
[1] 0
> f(5, shape = 5, rate = 1)
[1] 0.1343
> f(9, shape = 5, rate = 1)
[1] 0.02936
> f(12, shape = 5, rate = 1)
[1] 0
```

□

Note how function `f` in the previous example is built specifically for the

coverage modifications submitted and contains as little useless code as possible.

The function returned by `coverage` may be used for various purposes, most notably parameter estimation, as the following example illustrates.

Example 6. Let object `y` contain a sample of claims amounts from policies with the deductible and limit of Example 5. One can fit a gamma distribution by maximum likelihood to the claim severity distribution as follows:

```
> library(MASS)
> fitdistr(y, f, start = list(shape = 2, rate = 0.5))
   shape     rate
  3.9542   0.8060
 (0.6988) (0.1476)
```

□

Vignette `"coverage"` contains more detailed formulas for the pdf and the cdf under various combinations of coverage modifications.

References

R. V. Hogg and S. A. Klugman. *Loss Distributions*. Wiley, New York, 1984. ISBN 0-4718792-9-0.

S. A. Klugman, H. H. Panjer, and G. Willmot. *Loss Models: From Data to Decisions*. Wiley, New York, 1998. ISBN 0-4712388-4-8.

S. A. Klugman, H. H. Panjer, and G. Willmot. *Loss Models: From Data to Decisions*. Wiley, New York, 4 edition, 2012. ISBN 978-1-118-31532-3.

W. N. Venables and B. D. Ripley. *Modern applied statistics with S*. Springer, New York, 4 edition, 2002. ISBN 0-3879545-7-0.