mistr: A Computational Framework for Mixture and Composite Distributions

Lukas Sablica[a] **and Kurt Hornik**[a]

[a]Institute for Statistics and Mathematics, WU Wirtschaftsuniversität Wien, Austria; https://www.wu.ac.at/en/statmath

This version was compiled on December 7, 2018

The main aim of this vignette is to introduce several available options for the package mistr. In the first place, we introduce the computational and object oriented framework for the standard univariate distributions that uses S3 dispatch mechanism to create an object with all necessary information. These objects represent a random variable with certain properties and can be later used for evaluation of the cumulative distribution function (CDF), probability density function (PDF), quantile function (QF), and random numbers generation. Such objects can be then transformed using offered monotonic transformations or combined into mixtures and composite distributions and then transformed again. In the end, we provide and describe functions for data modeling using two specific composite distributions together with a numerical example, where a composite distribution is estimated to describe the log-returns of selected stocks.

distributions | composite | mixture | R | tails | Pareto | models | truncated | spliced

Introduction

During the history of the financial mathematics mankind has developed many useful theories how to describe financial markets. Some of these theories assume that the market behavior can be described using one simple distribution. For example, in case of stock log-returns it is typically a bad practice to assume the normal distribution, even if we see that the empirical distributions are generally heavy tailed. But can these market movements, which represent how we, highly advanced beings, think be described by only one standard distribution? The same way we think differently in different moods or extreme situations, the distribution describing our behavior in these situations can change. A simple illustration might be the distribution of SAP log-returns. Clearly, the tails are much heavier than in the case of normal distribution with the same mean and standard deviation. This behavior can be frequently found in a number of the financial assets.

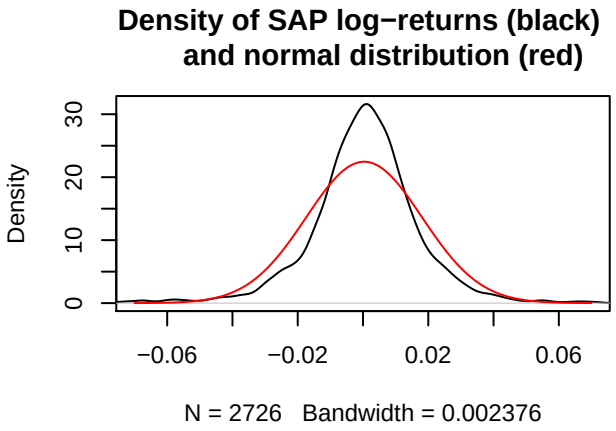
```
library(mistr)
```

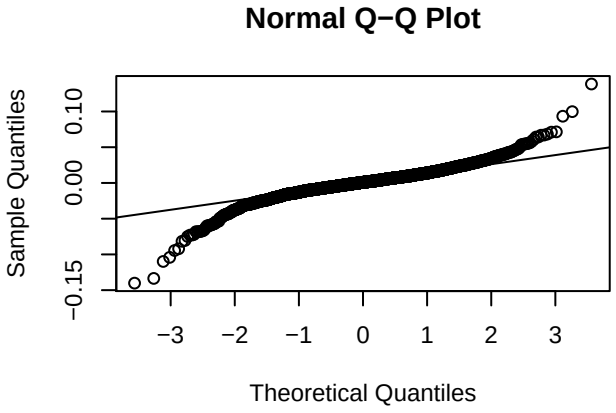
```
op <- par(mfrow = c(2, 1))

plot(density(stocks$SAP), xlim = c(-0.07, 0.07),
     main = "Density of SAP log-returns (black)
     and normal distribution (red)")

x <- seq(-0.07, 0.07, 0.001)
lines(x, dnorm(x, mean(stocks$SAP), sd(stocks$SAP)),
      col = "red")

qqnorm(stocks$SAP)
qqline(stocks$SAP)
```






```
par(op)
```

A more interesting result can be seen from the quantile-quantile plot. While the normal distribution fails in the tails, it excels in the center. This suggests to use a more suitable distribution, a distribution that can catch the fat tails presented above and yet follows a normal distribution in the center.

A simple answer to this idea is the concept of composite distributions and mixture models, where one assumes that the distribution is a finite mixture of component distributions. Clearly, a three-component composite model whose PDF is defined as

$$f(x) = \begin{cases} w_1 \frac{f_1(x)}{F_1(\beta_1)} & \text{if} \quad -\infty < x < \beta_1, \\ w_2 \frac{f_2(x)}{F_2(\beta_2) - F_2(\beta_1)} & \text{if} \quad \beta_1 \le x < \beta_2, \\ w_3 \frac{f_3(x)}{1 - F_3(\beta_2)} & \text{if} \quad \beta_2 \le x < \infty, \end{cases}$$

with first and third distribution being some heavy-tailed distribution and its negative transform, respectively, is something that might be appropriate for the presented data.

Moreover, composite models have gained a fair amount of attention in actuarial loss modeling. What is frequently used are mostly models composite of two distributions, where the models are based on the log-normal distribution and defined on the positive reals. The second distribution is chosen according to the data-set to model extreme measurements. Common choices for these tail-modeling distributions are, for instance, the generalized Pareto distribution or the Burr distribution. Such models have been proposed by various authors, for more details see (Nadarajah and Bakar, 2013), (Cooray and Ananda, 2005) and (Scollnik, 2007). In case of financial mathematics, these generated probability distributions are not enjoying such great popularity. The main reason is the difficulty to obtain a closed form of the whole distribution, or to even fit the parameters of such a distribution to some empirical data.

To offer a general framework for such univariate distributions and for mixtures in general, package **mistr** is specifically designed to create such models, evaluate or even fit them. This article introduces this package and illustrates with several examples how these distributions can be created and used.

Distributions in R

R is a very powerful and popular programming language, which people around the world use to work with distributions on a daily basis. It currently contains the standard naming convention [prefix][name], where the [name] corresponds to the name of the desired distribution and [prefix] stands for the popular p, d, q, and r. However, there are a lot of restrictions in such a concept. What would be desirable is that one would be able to treat a random variable as a variable and so to be able to send the variable to a function or perform transformations.

Naturally, one way to do this is by using the object oriented system in R. To even improve this idea, one can use some favored dispatching mechanism, like S3 or S4, to let the computer decide how to handle the corresponding distribution correctly and which functions to use. In particular, the prefixes p, d, q, and r can still be just smartly evaluated as generic functions with appropriate methods. Moreover, with such a system we can add other useful calls and so take the distribution operations to the next level, such as a monotonic transformation of a distribution. Additionally, once these objects containing all necessary information about the distributions are defined, they can be then reused for the purpose of the mixture and composite distributions.

This approach has already been used in the package **distr** (Kohl and Ruckdeschel, 2010). Package **distr** provides a conceptual treatment of distributions by means of S4 classes. A mother class *Distribution* allows to create objects and contains a slot for parameters and also for the four methods mentioned above, `p()`, `d()`, `q()`, and `r()`. While **distr** provides several classes for distributions, like many similar packages, it does not support any tools to work with the composite distributions. In particular, the only packages available for composite models are the **CompLognormal** package (Nadarajah and Bakar, 2013), and the package **Gendist** (Bakar *et al.*, 2016), which can only deal with two-components composite distributions.

The proposed framework by the package **mistr** currently supports all distributions that are included in the **stats** package and, in addition, it offers some extreme value distributions like generalized Pareto, Pareto, Frechet, and Burr. In case that the user would like to use a distribution that is not directly supported by the framework, a new distribution can be implemented in a very simple way. This procedure is more deeply documented in "Extensions" vignette.

The objects can be created very easily. The creator-functions follow a standard naming convention from R where the name of a distribution is suffixed with "dist" suffix and the parameters are entered as arguments. Thus, an object representing normal distribution with mean equal to 1 and standard deviation equal to 3 can be created as follows:

```
N <- normdist(mean = 1, sd = 3)
N
```

```
#   Distribution       Parameters
#      Normal    mean = 1, sd = 3
```

Once the objects are created, they can be used for evaluation of various functions. Among the most used functions surely belong the `print()` function that was already demonstrated and the functions `p()`, `d()`, `q()` and `r()`. These can be easily evaluated as

```
d(N, c(1, 2, 3))
```

```
#  [1] 0.1329808 0.1257944 0.1064827
```

```
p(N, c(1, 2, 3))
```

```
#  [1] 0.5000000 0.6305587 0.7475075
```

```
q(N, c(0.1, 0.2, 0.3))
```

```
#  [1] -2.8446547 -1.5248637 -0.5732015
```

```
r(N, 3)
```

```
#  [1]  2.2822066 -0.2858569 -3.0163143
```

Another option is to use the wrappers `mistr_d()`, `mistr_p()`, `mistr_q()` and `mistr_r()` if the IDE catches the `q()` call (for example the R-Studio for Windows users).

Next important function provided by **mistr** is the left-hand limit of the cumulative distribution function. It might not look of crucial importance to be able to evaluate $F(x-) = \mathbb{P}(X < x)$, but this function plays a huge role in the transformations and composite distributions. Of course this function differs from the standard distribution function only if it is applied to a distribution with a positive probability mass in the point of interest. This function can be called using `plim()`.

```
B <- binomdist(size = 12, prob = 0.3)
plim(B, c(-3, 0, 3, 12))
```

```
#  [1] 0.0000000 0.0000000 0.2528153 0.9999995
```

Another very important function when dealing with transformations is the pseudoinverse for the left-hand limit of the CDF, i.e.

$$Q(p+) = \inf\{x \in \mathbb{R} : p < \mathbb{P}(X \leq x)\}.$$

These values can be obtained using the `qlim()` call, and just as `plim()`, in the case of continuous distributions it simplifies to `q()`.

```
qlim(B, plim(B, c(0, 3, 7, 12)))
```

```
#  [1]  0  3  7 12
```

Adding transformation

Once the objects that represent a single distribution are created, we can use this representation to go beyond the scope of a simple distribution function evaluation. The knowledge of support and class that is stored inside the object opens the doors to more complicated operations. One such an operation that we would like to introduce in this chapter is the ability to perform monotone transformations of defined random variables. This adds even more generality to the whole framework that can be later reused in the mixture and composite distributions. This innovation is also motivated by the fact that even though most of the extreme value distributions belong to the location-scale family, without a decreasing transformation these distributions cannot be used for modeling of the left tail.

The transformation framework currently allows for all standard monotone transformations (+ - * / log exp ^) and is provided with the knowledge of invariant and direct transformations that correspond to the distributions it offers. This information is stored as a generic function that directly dispatches on the class of distribution family and not on the class *univdist* to prevent from losing any information about the distribution. An example might be the exponential distribution where a multiplication with a positive scalar rather keeps the family and changes parameters to prevent losing any information about the distribution. On the other hand, a positive power transformation will directly create a Weibull distribution with appropriate parameters.

```
E <- expdist(2)

E * 2
```

```
#  Distribution  Parameters
#  Exponential   rate = 1
```

```
E^2
```

```
#  Distribution                Parameters
#    Weibull    shape = 0.5, scale = 0.25
```

If the transformation is necessary, the transformation dispatches on the class *univdist*. For any untransformed distribution this function will change the whole class and the class of distribution family is removed as the random variable does not follow the distribution anymore. However, the information is stored for the case the distribution would need to untransform itself later. In this case, the function builds an expression for the transformation and inverse transformation, along with a print expression and an expression for the derivative of the inverse transformation. Besides these list members, also a member history is stored. History is a list that stores the information about the old transformations and becomes really handy when it comes to an inverse transformation of the previous one, or updating a transformation. A simple example of a transformation and update follows.

```
E2 <- E * -2
E3 <- E2 * 5
E3
```

```
#    Trafo Distribution Parameters
#  -10 * X  Exponential   rate = 2
```

Once it is explained how the framework gathers information about the transformation and creates the required expressions, we can proceed with the functions that exploit this. An example is the transformation of the normal distribution that we created in the last chapter.

```
Norm_trafo <- (N - 1)^(1/3)
Norm_trafo
```

```
#    Trafo Distribution       Parameters
#  X^(1/3)       Normal mean = 0, sd = 3
```

Note that the $X - 1$ transformation is not displayed in the Trafo column as it is an invariant transformation that rather changed the parameter mean from 1 to 0.

The functions that evaluate the transformed distribution are called in the same fashion as in non-transformed distributions. Additionally, the new pseudo description of the support can be returned as `sudo_support()`.

```
Binom_trafo <- -3 * log(B + 4)

q(Binom_trafo, c(0.05, 0.5, 0.95))
```

```
#  [1] -6.907755 -6.238325 -4.828314
```

```
plim(Binom_trafo, c(-6, -5, 0))
```

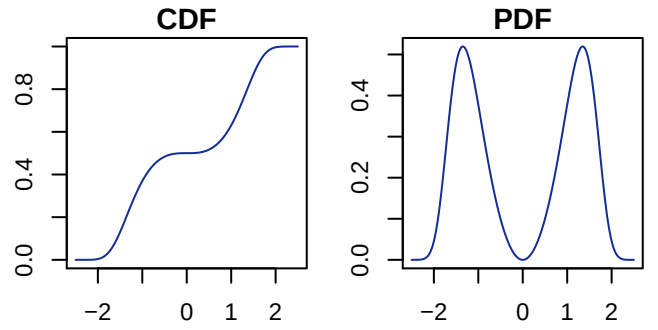
```
#  [1] 0.5074842 0.9149750 1.0000000
```

```
sudo_support(Binom_trafo)
```

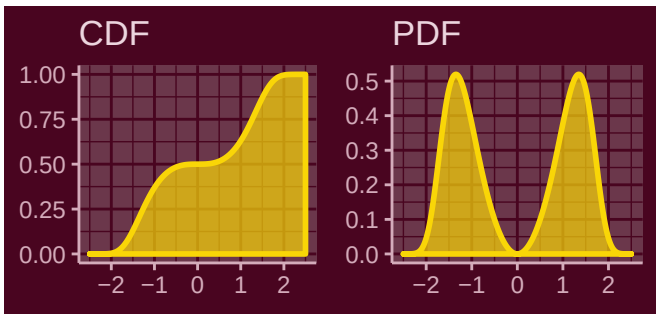
```
#       From        To
#  -8.317766 -4.158883
```

In addition, the `plot()` and `autoplot()` functions, which are also generic can be called. These functions are offered for any distribution object in the **mistr** package and return the plot of PDF or PMF and CDF of a given object. The function uses the introduced `d()` and `p()` functions to evaluate the required values. While the `plot()` call offers a plot constructed using the base plotting, the `autoplot()` offers an alternative plot that is created using the **ggplot2** package (Wickham, 2016).

```
par(mai = c(0.3, 0.3, 0.2, 0.2))
plot(Norm_trafo, xlim1 = c(-2.5, 2.5), ylab1 = "")
```




```
library(ggplot2)
autoplot(Norm_trafo, xlim1 = c(-2.5, 2.5))
```

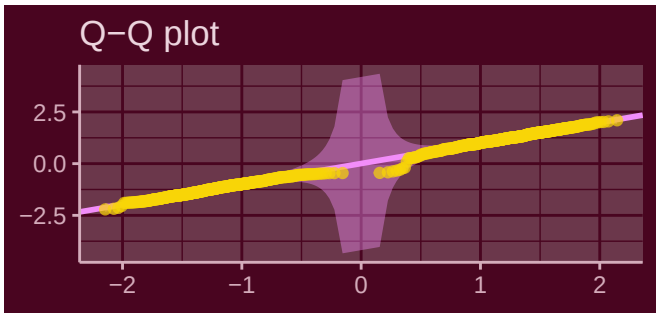


Another plot functions that are offered for the **mistr** distribution objects are `QQplot()` and `QQplotgg()`. The functions graphically compare the empirical quantiles of two data sets, or quantiles of two distribution objects, or quantiles of a distribution with the empirical quantiles of a sample. If quantiles of a continuous distribution are compared with a sample, a confidence bound for this data is offered. This confidence "envelope" is based on the asymptotic results of the order statistics. For a distribution F as n tends to infinity, the p^{th} sample quantile is asymptotically distributed as

$$X_{([np])} \sim AN\left(F^{-1}(p), \frac{p(1-p)}{n[f(F^{-1}(p))]^2}\right),$$

where $f(x)$ and $F^{-1}(p)$ are the density function and quantile function associated with $F(x)$, respectively. More details can be found for example on the order statistics Wikipedia page.

```
QQplotgg(Norm_trafo, r(Norm_trafo, 1000),
         conf = 0.99, ylab = NULL, xlab = NULL)
```



Combining objects

Mixtures. Mixture distributions defined as

$$F(x) = \sum_{i=1}^{n} w_i F_i(x),$$

are fully specified by the components $F_i(x)$ (i.e., the distributions) and by the weights w_i that correspond to these components. Thus, to create a mixture everything one needs is to specify these characterizations in the `mixdist()` call. This can be done in two ways. First, the user may specify the distribution names (names from [prefix][name] functions), the list of appropriate parameters of these distributions, and a sequence of weights. An example of such a call follows.

```
mixdist(c("norm", "unif"), list(c(2, 2), c(1, 5)),
                           weights = c(0.5, 0.5))
```

```
#  Mixture distribution with:
#
#    Distribution       Parameters       Weight
#  1    Normal    mean = 2, sd = 2          0.5
#  2   Uniform    min = 1, max = 5          0.5
```

Another way is to use the objects that have already been defined. Since the parameters are already stored inside the object, all the function requires are the objects and the weights. This also allows to use transformed distributions from the last chapter or more complicated objects, which will be presented later. This means that the transformed normal and binomial distributions together with an exponential distribution can be reused for mixture distribution as:

```
M <- mixdist(Norm_trafo, Binom_trafo, expdist(0.5),
                           weights = c(.4, .2, .4))
```

The information about the mixture can be accessed via generic functions. The objects can be extracted using square brackets `[]`, the weights can be obtained using `weights()`, and just as with standard distributions, the parameters are obtainable using `parameters()`.

Interesting functions are the functions `q()` and `qlim()`. While finding the CDF and PDF of the mixture model is straightforward, an explicit general expression for quantile function of the mixture model is not available. However, it can be found numerically as a solution of a unit-root problem:

$$\sum_{i=1}^{n} w_i F_i(Q(p)) - p = 0.$$

What is more, one can show that the quantile of a mixture distribution $Q(p)$ can always be found within the range of its components quantiles, and hence

$$\min_{i\in\{1,\ldots n\}} Q_i(p) \le Q(p) \le \max_{i\in\{1,\ldots n\}} Q_i(p),$$

where $Q_i(\cdot)$ is the quantile function of the i-th component. This specifies the needed interval for the root finder that will then iteratively find the solution. Additionally, further problems are solved to return the correct values. To show how this algorithm works we can perform a simple test. To make things harder, we will create a fully discrete mixture for which a decreasing transformation is performed.

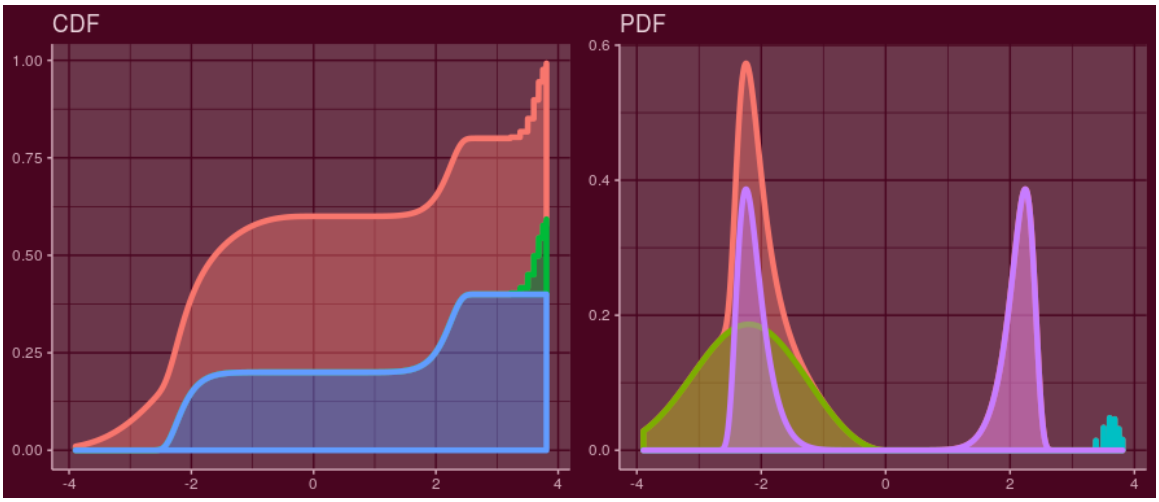
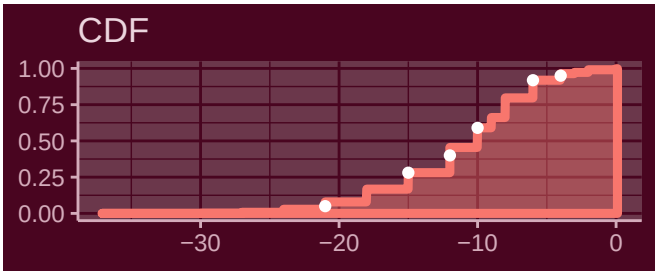


Fig. 1. autoplot output of M_trans

```
DM <- mixdist(3 * binomdist(12, 0.4),
          -2*poisdist(2) + 12, weights=c(0.5, 0.5))
y <- c(0.05, 0.4, p(-DM, c(-5, -10, -15)), 0.95)
x <- q(-DM, y)
autoplot(-DM, which = "cdf", only_mix = TRUE,
                          xlim1 = c(-37, 0)) +
   annotate("point", x, y, col = "white")
```



Finally, since the inverse transform sampling is not efficient for mixture distributions, it can be replaced by first sampling according to the weights w_i and then, conditionally on that, by drawing from the selected component. This is implemented in the corresponding method of the `r()` function. This approach allows to draw from a mixture much faster than the inverse quantile transform method, and can also be reused later for composite distributions.

```
system.time(r(M, 1e6))
```

```
#     user  system elapsed
#    0.386   0.012   0.397
```

Except the quantile function and other main functions for evaluation, one can call other generic functions that are designed for the class *mixdist*. As an example we suggest the `sudo_support()`.

```
sudo_support(M)
```

```
#  From   To
#  -Inf  Inf
```

Since the mixture models are in fact distributions, one can perform transformations of mixture random variables as well. It is easy to show that a transformation of a mixture random variable is the same as if we applied the same transformation to all its components. In addition, since the support of the components is a subset of the mixture's support, if the framework allows to transform the mixture, then it does the components as well. Now, using the mixture we created, we can perform a decreasing non-linear transformation. An example of `r()` and `autoplot()` functions follows.

```
M_trans <- -2 * (M)^(1/3)
r(M_trans, 4)
```

```
#  [1]  1.452721 -2.635702  3.681736 -1.092241
```

```
autoplot(M_trans)
```

Composite distributions. Let $B_1, B_2, \ldots, B_n$ be Borel sets giving a disjoint partition of the support, and $F_1, F_2, \ldots, F_n$ be the probability distributions over $\mathbb{R}^d$ with $F_i(B_i) > 0$ for all $i = 1, 2, \ldots, n$. In addition, assume that $w_1, w_2, \ldots, w_n$ are positive weights that sum up to one. Then the composite distribution over the partition (B_i) of (F_i) with weights (w_i) can be written as

$$F(A) = \sum_{i=1}^{n} w_i \frac{F_i(A \cap B_i)}{F_i(B_i)} = \sum_{i=1}^{n} w_i F_i(A|B_i).$$

Note that as with mixture models it is not necessary for the two arbitrary distributions to be identical. Obviously, the composite models are a specific case of the mixture models, where the corresponding probability distribution functions are truncated to some disjoint support.

The interval representation of the truncation allows to use a sequence of breakpoints

$$-\infty = \beta_0 < \beta_1 \leq \beta_2 \leq \cdots \leq \beta_{n-1} < \beta_n = \infty$$

to fully characterize the partitions B_i. Note that if F_i is continuous, to ensure that the interval has positive probability we must set $\beta_{i-1} < \beta_i$.

This allows to define $\lambda_1 = 0$ and for all $i = 2, \ldots, n$,

$$\lambda_i = \begin{cases} F_i(\beta_{i-1}) & \text{if } \beta_{i-1} \notin B_i, \\ F_i(\beta_{i-1}-) & \text{otherwise,} \end{cases}$$

and $\rho_n = 1$, and for all $i = 1, 2, \ldots, n-1$,

$$\rho_i = \begin{cases} F_i(\beta_i) & \text{if } \beta_i \in B_i, \\ F_i(\beta_i-) & \text{otherwise.} \end{cases}$$

Then for any $x \in B_i$

$$F_i((-\infty, x] \cap B_i) = \begin{cases} F_i(x) - F_i(\beta_{i-1}) & \text{if } \beta_{i-1} \notin B_i, \\ F_i(x) - F_i(\beta_{i-1}-) & \text{if } \beta_{i-1} \in B_i. \end{cases}$$

This means that for every $x \in B_i$ we can write the distribution as $F_i((-\infty, x] \cap B_i) = F_i(x) - \lambda_i$.

The straightforward implication of the above equations is that $\sup_{x \in B_i} F_i(x) = \rho_i$. Thus, this implies that

$$F_i(B_i) = \rho_i - \lambda_i,$$

which can be calculated using the `p()` and `plim()` functions. Hence, if we define $p_i = \sum_{j:j \le i} w_i$ the composite distribution satisfies

$$F(x) = p_{i-1} + w_i \frac{F_i(x) - \lambda_i}{F_i(B_i)} = p_{i-1} + w_i \frac{F_i(x) - \lambda_i}{\rho_i - \lambda_i}, \quad \forall x \in B_i.$$

Therefore, to fully specify a composite distribution, in addition to the mixture specifications, one needs to set the values that correspond to the breakpoints, which split $\mathbb{R}$ into disjoint partitions. Moreover, if at least one distribution is not absolutely continuous, it might be desired to specify to which adjacent interval should the breakpoint be included.

Just as mixture distributions, composite models can be created in two ways. Either one can directly use the objects or let the function create these objects by specifying the sequence of names and a list of parameters. In the following example we will directly proceed with the first mentioned method where we define some objects inside the `compdist()` call to create a composite distribution. Besides these objects one needs to set the sequences of weights and breakpoints. Additionally, one may determine for each breakpoint to which partition should the breakpoint be included. This can be set by the argument break.spec with values 'R' or 'L', where 'R' and 'L' stand for right (i.e., include breakpoint to the interval on the right of the breakpoint) and left (i.e., include to the interval on the left), respectively. If this argument is not stated, the algorithm will by default set all intervals to be left-closed, i.e., right-open. This can be nicely seen from the following example where a linearly transformed Pareto distribution and a geometric distribution are combined with a normal distribution into a composite model.

```
C <- compdist(-paretodist(1, 1), normdist(0, 2),
              geomdist(0.3) + 2,
              weights = c(0.15, 0.7, 0.15),
              breakpoints = c(-3, 3),
              break.spec = c("L", "R"))
C
```

```
#  Composite distribution with:
#
#    Trafo Distribution           Parameters Weight
#  1  -X       Pareto    scale = 1, shape = 1   0.15
#  2 none      Normal      mean = 0, sd = 2     0.70
#  3 X + 2    Geometric       prob = 0.3        0.15
#    Truncation
#  1  (-Inf,-3]
#  2   (-3,3)
#  3   [3,Inf)
```

The argument break.spec is set to ("L", "R"), and thus the breakpoint -3 belongs to the first partition while the second breakpoint is affiliated to the partition on the right. This can be observed from the print of the distribution, more precisely from the Truncation column, where the parentheses are printed according to this argument.

The package also permits to use the same breakpoint twice. This possibility allows to define a partition on a singleton, and hence to create a mass of probability. If this feature is used, the break.spec needs to be specified with "R" and "L", for the first and the second identical breakpoint, respectively, or not set at all. If the break.spec is not used, the source code will change the break.spec such that this single point with probability mass is a closed set. This feature can become particularly useful when the user wants to create a distribution that is, for example, absolutely continuous on both the negative and positive reals and has positive mass at zero.

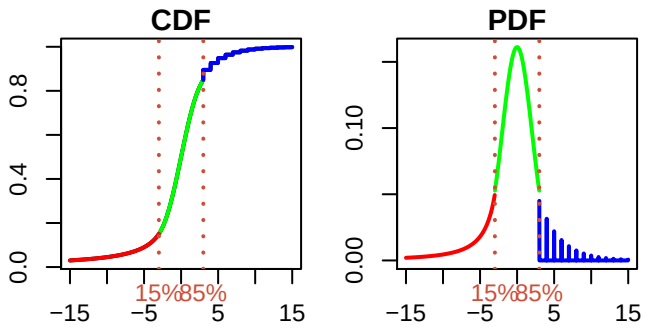
```
C2 <- compdist(-expdist(2), poisdist(),
               expdist(2),
               weights = c(0.25, 0.5, 0.25),
               breakpoints = c(0, 0))
C2
```

```
#  Composite distribution with:
#
#    Trafo Distribution Parameters Weight Truncation
#  1  -X    Exponential  rate = 2    0.25   (-Inf,0)
#  2 none     Poisson   lambda = 1   0.50    [0,0]
#  3 none   Exponential  rate = 2    0.25   (0,Inf)
```

Note that the distribution assigned to this singleton has to be a discrete distribution with support on that point, otherwise the interval will have zero probability.

As for any distribution, the framework also offers many generic functions that can be used to obtain additional information or evaluate the distribution. One can extract the parameters, weights, or the support in the same manner as with mixture distributions. In addition, calling `breakpoints()` extracts the splicing points. Finally, the functions `plot()` and `autoplot()` are offered where by default the components are visible. As with mixtures, this can be turned off using only_mix = TRUE argument.

```
par(mai = c(0.3, 0.3, 0.2, 0.2))
plot(C, xlim1 = c(-15, 15), ylab1 = "")
```



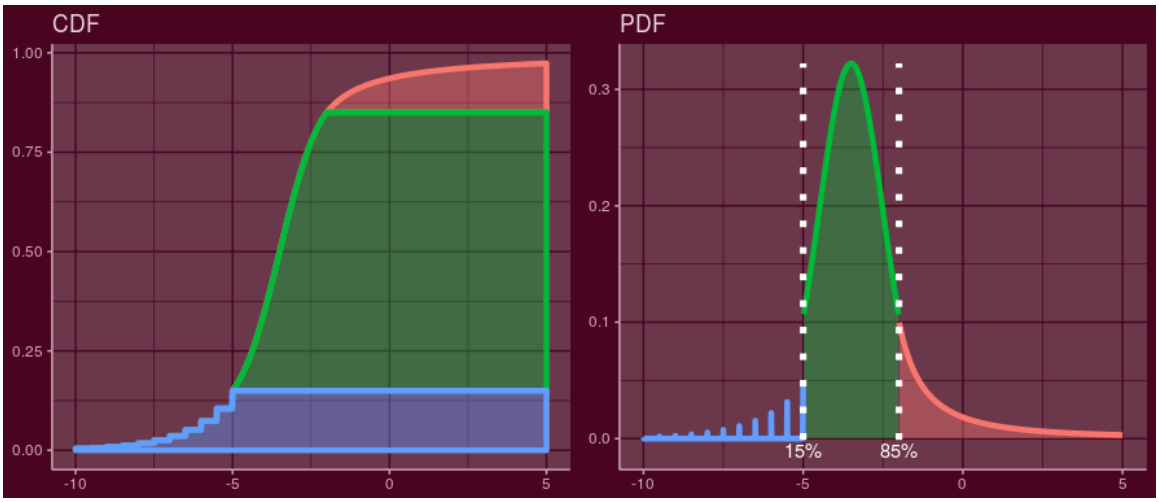
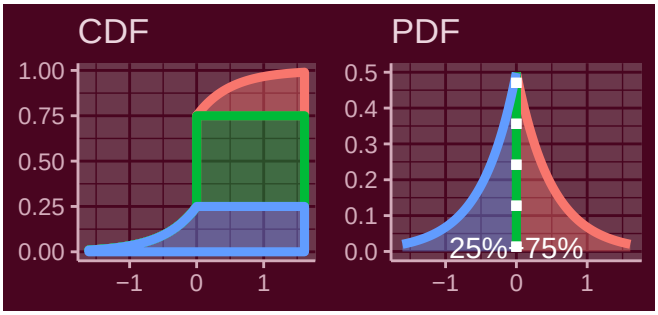


Fig. 2. autoplot output of C_trans

```
autoplot(C2, text_ylim = 0.01)
```



Analogously to the mixture distributions, the framework offers to transform also composite random variables. Thus, using the composite distribution we defined, we propose an example of a linear transformation.

```
C_trans <- -0.5 * (C + 7)
```

Even with such a distribution, the user still can evaluate any desired presented function. To support this, we again propose an example where the function `q()` and `r()` are used, and the functions `p()` and `d()` are represented graphically using the function `autoplot()`.

```
q(C_trans, c(0.075, 0.5, 0.7, 0.9))
```

```
#  [1] -5.500000 -3.500000 -2.833235 -1.250000
```

```
r(C_trans, 4)
```

```
#  [1] -2.767421 -4.682717 -3.928681 -3.989454
```

```
autoplot(C_trans, xlim1 = c(-10,5))
```

Combining mixture and composite distributions. A significant advantage of object oriented programming is that the dispatching mechanism automatically knows how to treat a given object. This allows to combine mixture and composite models into more complicated mixtures and composite distributions. Therefore, we can take the transformed mixture and the transformed composite distributions we created to compose a composite distribution with these distributions as components. What is more, we can perform a transformation of such a distribution.

```
C3 <- compdist(M_trans - 3,
               C_trans, weights = c(0.5, 0.5),
               breakpoints = -4.5)
C3_trans <- -2 * C3 + 2
```

Thus, the object C3_trans is a transformed composite distribution that contains a transformed mixture and a transformed composite distribution, from which both additionally contain many transformed and untransformed distributions. Even in such complex models, the user may evaluate the most complicated functions as `plim()` and `qlim()`. The functions `d()` and `p()` can be again best represented graphically, where both distributions can easily be recognized from previous chapters.

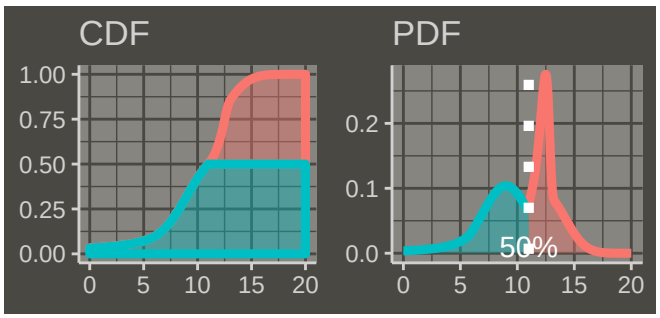
```
plim(C3_trans, c(6, 10, 12))
```

```
#  [1] 0.09667553 0.42195189 0.62458021
```

```
qlim(C3_trans, c(0.3, 0.5, 0.7))
```

```
#  [1]  8.785363 11.000000 12.327907
```

```
autoplot(C3_trans, xlim1 = c(0,20), text_ylim = 0.01,
                                    grey = TRUE)
```



Although the print for such a hierarchical distribution does not contain a lot of information, an alternative function can be used. We recommend to use the function `summary()`, which is designed mainly for the more complicated distributions. The printed result of this call contains all the necessary information, and much more as well. Additionally, since the result of `summary()` on the C3_trans object is two pages long, the demonstration is left to the reader.

To finish this chapter and to show that the user may go even further, we would like to present an example where we will combine the last object with another distribution from this chapter into a mixture distribution. The distribution is directly plotted using the autoplot() call.

```
autoplot(mixdist( C3_trans, C2 + 5,
          weights = c(0.7, 0.3)),
                xlim1 = c(0, 15))
```

Data modeling

While the previous chapters were aimed at showing the "muscles" (i.e., generality) of the framework, in this last section we will focus on examples using real data. In particular, we will present a simple fitting for two specific composite distributions.

As motivated in the introduction, the models in financial mathematics suggest a distribution that can capture the wide variety of behavior in tails while still following the normal distribution in the center. This offers to use a three components composite distribution. The first and third component will be used to model the extreme cases, i.e., the tails, and the second component will try to catch the center of the empirical distribution.

The first model offered by **mistr** is the Pareto-Normal-Pareto (PNP) model. This means that a $-X$ transformation of a Pareto random variable will be used for the left tail, normal distribution for the center and again Pareto for the right tail. From this it follows that the PDF of the model can be written as:

$$f(x) = \begin{cases} w_1 \frac{f_{-P}(x)}{F_{-P}(\beta_1)} & \text{if} \quad -\infty < x < \beta_1, \\ w_2 \frac{f_N(x)}{F_N(\beta_2) - F_N(\beta_1)} & \text{if} \quad \beta_1 \le x < \beta_2, \\ w_3 \frac{f_P(x)}{1 - F_P(\beta_2)} & \text{if} \quad \beta_2 \le x < \infty, \end{cases}$$

where $f_P(x) = f_{-P}(-x)$ and $F_P(x) = 1 - (K/x)^\alpha$ are the density and distribution function of a Pareto distribution with $F_{-P}(x) = 1 - F_P(-x)$. $f_N(x)$ and $F_N(x)$ are the PDF and CDF of the normal distribution, respectively.

If we follow the properties of the Pareto distribution, the conditional probability distribution of a Pareto-distributed random variable, given that the event is greater than or equal to $\gamma > K$, is again a Pareto distribution with parameters γ and α. This means that the conditional distribution $f_P(x|K,\alpha)/(1 - F_P(\beta_2|K,\alpha)) = f_P(x|\beta_2,\alpha)$ if $\beta_2 > K$. On the other hand, if $\beta_2 < K$ the distribution cannot be continuous as the support of the Pareto distribution starts at K. The same can be shown for the transformed distribution. Since we are interested only in the continuous case we can rewrite the PDF as

$$f(x) = \begin{cases} w_1 f_{-P}(x|-\beta_1,\alpha_1) & \text{if} \quad -\infty < x < \beta_1, \\ w_2 \frac{f_N(x|\mu,\sigma)}{F_N(\beta_2|\mu,\sigma) - F_N(\beta_1|\mu,\sigma)} & \text{if} \quad \beta_1 \le x < \beta_2, \\ w_3 f_P(x|\beta_2,\alpha_2) & \text{if} \quad \beta_2 \le x < \infty, \end{cases}$$

$$\text{where} \quad \begin{array}{l} \beta_1 < 0 < \beta_2, \\ \alpha_1, \alpha_2 > 0. \end{array}$$

The condition $\beta_1 < 0 < \beta_2$ follows from the fact that the scale parameter has to be positive. Thus, such a model can be fully used only with demeaned data sample or with data with a mean close to zero. This is of course not a problem for stock returns, which are the aim of this vignette. What is more, one can show that the density is continuous if it holds for the shape parameters that

$$\alpha_1 = -\beta_1 \frac{w_2 f_N(\beta_1|\mu,\sigma)}{w_1 \left(F_N(\beta_2|\mu,\sigma) - F_N(\beta_1|\mu,\sigma)\right)},$$

$$\alpha_2 = \beta_2 \frac{w_2 f_N(\beta_2|\mu,\sigma)}{w_3 \left(F_N(\beta_2|\mu,\sigma) - F_N(\beta_1|\mu,\sigma)\right)}.$$

Due to the fact that a composite distribution can be represented as a mixture of truncated distributions that are truncated to a disjoint support, the weight of each component can be estimated as the proportion of points that correspond to each of the truncated regions. Obviously, this condition ensures that the empirical and estimated CDF match on each of the breakpoints. Thus, conditionally on the fact that the breakpoints are known, the weights can be computed as

$$w_1 = \frac{\sum_{i=1}^n 1_{\{x_i < \beta_1\}}}{n}, w_2 = \frac{\sum_{i=1}^n 1_{\{\beta_1 \le x_i < \beta_2\}}}{n}, w_3 = \frac{\sum_{i=1}^n 1_{\{\beta_2 \le x_i\}}}{n},$$

where $1_{\{\cdot\}}$ is the indicator function, and x_i is the i-th data value. These conditions decrease the number of parameters from 11 to 4, and imply the density function of a form:

$$f(x|\beta_1,\beta_2,\mu,\sigma).$$

This model if offered by the call `PNP_fit()`. The function `PNP_fit()` takes the data and a named vector of starting values with names break1, break2, mean, and sd and returns a list of class *comp_fit*. Other arguments are passed to the optimizer. To demonstrate this, we will take the same data we used in the introduction to fit a PNP model with the default starting values.

```
PNP_model <- PNP_fit(stocks$SAP)

PNP_model
```

```
#  Fitted composite Pareto-Normal-Pareto
#  distribution:
#
#  Breakpoints: -0.019304 0.020518
#  Weights: 0.092443 0.82135 0.086207
#
#  Parameters:
#    scale1   shape1     mean       sd   scale2
#  0.019304 1.773598 0.000961 0.012950 0.020518
#    shape2
#  2.198590
#
```

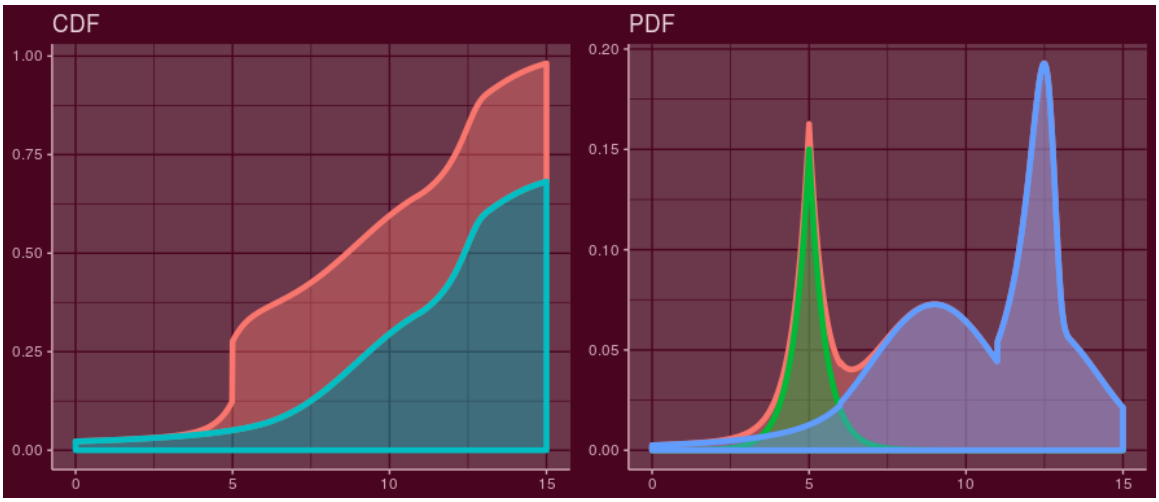


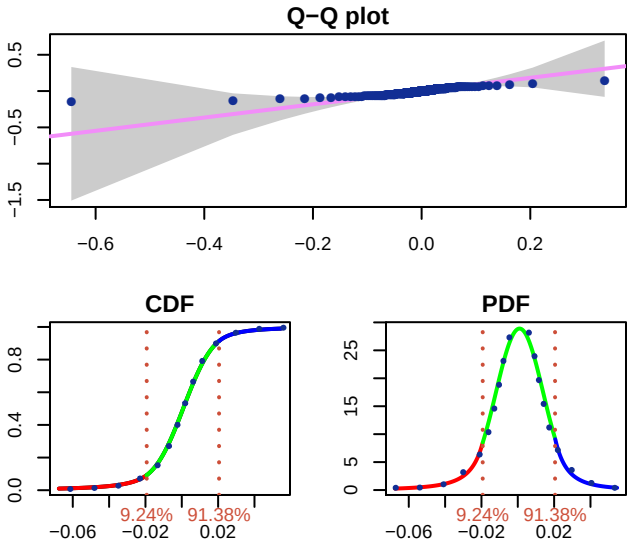
Fig. 3. autoplot output of the mixture model containing two composite models

```
#  Log-likelihood: 7400.117,
#  Average log-likelihood: 2.7146
```

If the fitted object is printed, the function will print all the parameters together with the log-likelihood that was achieved by the optimization. In addition, the average log-likelihood is printed, which is just the log-likelihood divided by the size of the data-set. The user can extract parameters using the call `parameters()`, weights using the call `weights()`, and breakpoints using `breakpoints()`. The `distribution()` function can be used to extract the distribution with fitted parameters that can be used for evaluation.

Finally, the `plot()` and `autoplot()` functions are offered. The functions plot the Q-Q plot of the fitted distribution and data, and the CDF and PDF plot of the fitted distribution, which overlap with the empirical CDF and PDF of the data-set. Again, the which argument can extract the proposed plots separately (i.e., which = "pdf"). Other arguments are passed to the the plot calls.

```
plot(PNP_model, ylab1 = "", ylab2 = "")
```



The plots indicate an overall nice fit where all quantiles are in the confidence bound.

The second offered model is a similar distribution to the previous one, except we will replace the Pareto distributions by the generalized Pareto distributions (GPD)

$$F_{GPD}(x) = \begin{cases} 1-\left(1+\xi\frac{x-\theta}{\gamma}\right)^{-1/\xi} & \text{if } \xi \neq 0, \\ 1-\exp\left(-\frac{x-\theta}{\gamma}\right) & \text{if } \xi = 0. \end{cases}$$

This means that the PDF of this model can be written as:

$$f(x) = \begin{cases} w_1 \frac{f_{-GPD}(x)}{F_{-GPD}(\beta_1)} & \text{if} \quad -\infty < x < \beta_1, \\ w_2 \frac{f_N(x)}{F_N(\beta_2)-F_N(\beta_1)} & \text{if} \quad \beta_1 \leq x < \beta_2, \\ w_3 \frac{f_{GPD}(x)}{1-F_{GPD}(\beta_2)} & \text{if} \quad \beta_2 \leq x < \infty. \end{cases}$$

The same way as in the PNP model, the scale parameters can be eliminated by the continuity conditions, weights by the above mentioned condition and in addition, under current settings and the continuity conditions, the value of the conditional GPD distribution depends on the location parameter only through the conditions $-\beta_1 \geq \theta_1$ and $\beta_2 \geq \theta_2$. This suggests to choose without any loss in the model $-\beta_1 = \theta_1$ and $\beta_2 = \theta_2$. Such a PDF is fully characterized by

$$f(x|\beta_1, \beta_2, \mu, \sigma, \xi_1, \xi_2),$$

where the only restriction on the parameters is $-\infty < \beta_1 < \beta_2 < \infty$.

These conditions decrease the number of parameters from 13 to 6. What is more, the function `GNG_fit()` contains the argument break_fix, which fixes the breakpoints from the vector of starting values, and so decreases the number of parameters to 4 if TRUE is assigned. In this case, the breakpoints are fixed and weights are computed before the optimization. The function `GNG_fit()` takes the data, the named vector of starting values with names break1, break2, mean, sd, shape1 and shape2, the break_fix argument and the argument midd, which is by default set to be equal to the mean of the data. The midd values are used to split $\mathbb{R}$ into two sub-intervals and then the first breakpoint is optimized on the left of the midd value and the second breakpoint on the right.

The call returns a list of class *comp_fit*. The results can be then extracted, printed or visualized in the same way as the results of `PNP_fit()`.

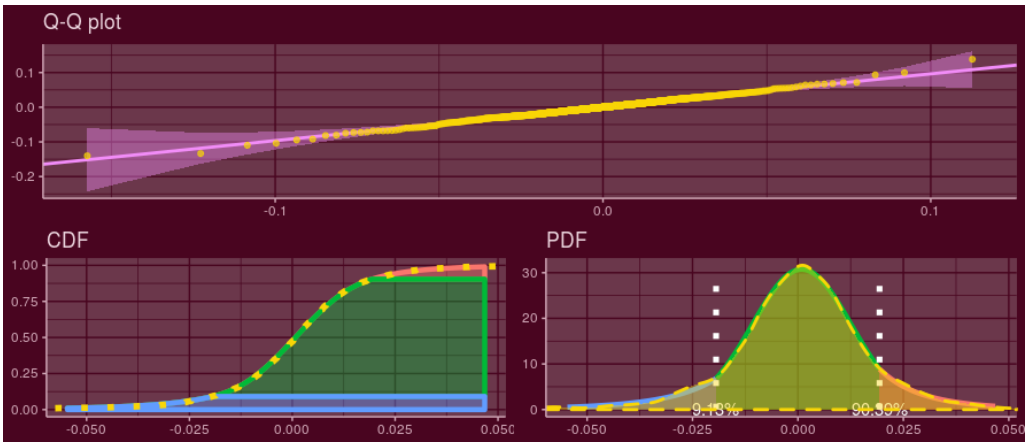


Fig. 4. autoplot output of GNG_fit

```
GNG_model <- GNG_fit(stocks$SAP)

GNG_model
```

```
#  Fitted composite GPD-Normal-GPD distribution:
#
#  Breakpoints: -0.019414 0.019353
#  Weights: 0.091343 0.812546 0.096112
#
#  Parameters:
#       loc1    scale1    shape1      mean        sd
#  0.019414 0.013439 0.150141 0.000907 0.011696
#       loc2    scale2    shape2
#  0.019353 0.010842 0.096832
#
#  Log-likelihood: 7423.245,
#  Average log-likelihood: 2.7231
```

```
autoplot(GNG_model)
```

The log-likelihood has increased to 7423.2 with the average of 2.723 per data-point. In this model, the generalized Pareto distribution explains the first 9.1% from the left tail and the last 9.6% from the right tail. In addition, since GPD generalizes the Pareto distribution, the higher likelihood is a reasonable result. Additionally, the QQ-plot suggests an almost perfect fit.

The result of these estimations is a proper continuous parametric set-up that describes the distribution of the data. What is more, the distribution has been fitted as a whole with respect to the continuity conditions. This means that the tails take into account the whole distribution, which allows to calculate the risk measures with an even higher precision as when only the tails are modeled.

Risk measures. Package **mistr** provides a function `risk()` which can be used for rapid calculations of point estimates of prescribed quantiles, expected shortfalls and expectiles. As an input parameter this function needs the output of the function `PNP_fit()` or `GNG_fit()` and a vector of desired levels. As an example we illustrate these functions on our fitted object.

```
risk(GNG_model, c(0.02, 0.05, 0.07, 0.1, 0.2, 0.3))
```

```
#    level         VaR         ES         Exp
#  1  0.02 0.042341368 0.06220519 0.032240507
#  2  0.05 0.027889909 0.04520065 0.021949976
#  3  0.07 0.023062791 0.03952074 0.018526202
#  4  0.10 0.018245509 0.03380624 0.015091608
#  5  0.20 0.010642738 0.02386181 0.008851640
#  6  0.30 0.006167236 0.01867190 0.005168659
```

These results can be also visualized if arguments plot or ggplot are set to TRUE.

References

Bakar S, Nadarajah S, Kamarul Adzhar Z, Mohamed I (2016). "Gendist: An R Package for Generated Probability Distribution Models." *P L o S One*, **11**(6). ISSN 1932-6203. .

Bolker B, Team RDC (2017). *bbmle: Tools for General Maximum Likelihood Estimation*. R package version 1.0.20, URL https://CRAN.R-project.org/package=bbmle.

Cooray K, Ananda MM (2005). "Modeling actuarial data with a composite lognormal-Pareto model." *Scandinavian Actuarial Journal*, **2005**(5), 321–334. . https://www.tandfonline.com/doi/pdf/10.1080/03461230510009763, URL https://www.tandfonline.com/doi/abs/10.1080/03461230510009763.

Kohl M, Ruckdeschel P (2010). "R Package distrMod: S4 Classes and Methods for Probability Models." *Journal of Statistical Software, Articles*, **35**(10), 1–27. ISSN 1548-7660. . URL https://www.jstatsoft.org/v035/i10.

Nadarajah S, Bakar S (2014). "New composite models for the Danish fire insurance data." *Scandinavian Actuarial Journal*, **2014**(2), 180–187. . https://doi.org/10.1080/03461238.2012.695748, URL https://doi.org/10.1080/03461238.2012.695748.

Nadarajah S, Bakar SAA (2013). "CompLognormal: An R Package for Composite Lognormal Distributions." *The R Journal*, **5**(2), 97–103. URL https://journal.r-project.org/archive/2013/RJ-2013-030/index.html.

Ryan JA, Ulrich JM (2018). *quantmod: Quantitative Financial Modelling Framework*. R package version 0.4-13, URL https://CRAN.R-project.org/package=quantmod.

Scollnik DPM (2007). "On composite lognormal-Pareto models." *Scandinavian Actuarial Journal*, **2007**(1), 20–33. . https://doi.org/10.1080/03461230601110447, URL https://doi.org/10.1080/03461230601110447.

Wickham H (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN 978-3-319-24277-4. URL http://ggplot2.org.