Journal of Statistical Software

August 2014, Volume 59, Issue 4. *http://www.jstatsoft.org/*

General Purpose Convolution Algorithm in S4 Classes by Means of FFT

Peter Ruckdeschel
Fraunhofer ITWM Kaiserslautern

Matthias Kohl
Furtwangen University

Abstract

Object orientation provides a flexible framework for the implementation of the convolution of arbitrary distributions of real-valued random variables. We discuss an algorithm which is based on the fast Fourier transform. It directly applies to lattice-supported distributions. In the case of continuous distributions an additional discretization to a linear lattice is necessary and the resulting lattice-supported distributions are suitably smoothed after convolution.

We compare our algorithm to other approaches aiming at a similar generality as to accuracy and speed. In situations where the exact results are known, several checks confirm a high accuracy of the proposed algorithm which is also illustrated for approximations of non-central χ^2 distributions.

By means of object orientation this default algorithm is overloaded by more specific algorithms where possible, in particular where explicit convolution formulae are available. Our focus is on R package **distr** which implements this approach, overloading operator `+` for convolution; based on this convolution, we define a whole arithmetics of mathematical operations acting on distribution objects, comprising operators `+`, `-`, `*`, `/`, and `^`.

Keywords: probability distributions, FFT, convolution, arithmetics for distributions, S4 classes.

1. Motivation

Convolution of (probability) distributions is a standard problem in statistics. For its implementation the fast Fourier transform (FFT) has been common practice ever since the appearance of Cooley and Tukey (1965). The upcoming of parallel computing architectures adds to the importance of FFT, as contrast to convolution by direct computations, it lends itself to easy parallelization, compare Gupta and Kumar (1993).

Combined with an object oriented programming (OOP) approach, this technique gets even more attractive: We may use it as a default algorithm in situations where no better alternative

is known, while in special cases as e.g., those of normal or Poisson random variables, where convolution reduces to transforming the corresponding parameters, a dispatching mechanism realizes this and replaces the general method by a particular (possibly exact) one. The user does not have to interfere with the dispatching mechanism, but is rather provided with one single function/binary operator for the task of convolution.

We discuss this approach within the R project (cf. R Core Team 2014) where it is implemented in package **distr**, available from the Comprehensive R Archive Network (CRAN) at http://CRAN.R-project.org/package=distr. Package **distr** provides classes for probability distributions within the S4 OOP concept of R; see Ruckdeschel, Kohl, Stabla, and Camphausen (2006, 2014).

In this context, convolution is the workhorse for setting up a whole arithmetics of mathematical operations acting on distribution objects, comprising, among others, operators `+`, `-`, `*`, `/`, and `^`. In this arithmetics, we identify distributions with corresponding (independent) random variables: If `X1` and `X2` are corresponding distribution variables, `X1 + X2` will produce the distribution of the sum of respective (independent) random variables, i.e., their convolution. Technically, speaking in terms of programming, we have overloaded the operator `+` for univariate distributions.

Convolution itself is computed according to the actual classes of the operands, with particular (exact) methods for e.g., normal or Poisson distributions.

```
R> library("distr")
R> N1 <- Norm(mean = 1, sd = 3)
R> N2 <- Norm(mean = -2, sd = 4)
R> N1 + N2

Distribution Object of Class: Norm
 mean: -1
 sd: 5
```

In the default method distributions are discretized to lattice form and the discrete Fourier transform (DFT) is applied. Thus, our general-purpose algorithm needs no assumptions like Lebesgue densities.

```
R> U1 <- Unif(Min = 0, Max = 1)
R> U3 <- convpow(U1, N = 3)
R> plot(U3, cex.inner = 1, inner = c("density", "cdf", "quantile function"))
```

While all our applied techniques are not novel in themselves, and much of the infrastructure (FFT in particular) has already been available in R for a long time, the combination as present in our approach is unique. Neither in core R nor in any other contributed add-on package available from CRAN (http://CRAN.R-project.org/), Bioconductor (http://www.Bioconductor.org/), or Rmetrics (https://www.Rmetrics.org/), there is a similarly general approach: We provide a `+` (aka convolution) operator applicable to (almost) *arbitrary* univariate distributions, no matter whether discrete or continuous. More specifically we cover every distribution that is representable as a convex combination of an absolutely continuous distribution and a discrete distribution. In addition, the return value of this `+` operator is

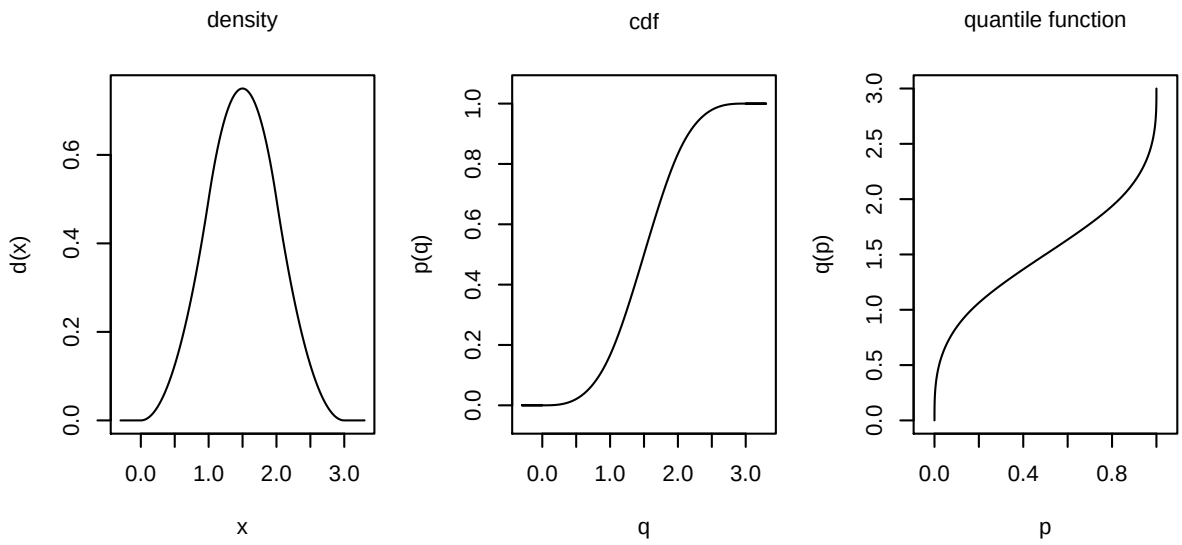

Figure 1: Plot of 3-fold convolution of a `Unif(0,1)` object.

again a distribution object, i.e., consisting not only of either a cumulative distribution function (CDF) or a density/probability function, but automatically of all four constitutive functions, i.e., CDF, density, quantile function, and random number generator. Accuracy of our default methods can be controlled through global options, see `?distroptions`. Just to illustrate our point, we take up the initial example and compute the 1/3 quantile of the convolution $\mathcal{N}(1,2) * \mathrm{Unif}(0,1)^{*3} * \mathrm{Poisson}(1)$, as well as evaluate its density at the vector $(0.5, 0.8)$

```
R> P <- Pois(lambda = 1)
R> D <- N1 + U3 + P
R> q(D)(1/3)

[1] 2.10923

R> d(D)(c(0.5, 0.8))

[1] 0.08110259 0.08815269
```

The approach is not restricted to academic purposes: The results are sufficiently accurate to be used in practice in many circumstances, be it for quite general compound distribution models as relevant in actuarial sciences, be it for very flexible model fitting techniques as described in detail in Kohl and Ruckdeschel (2010), or be it for very general robustification techniques as in packages **RobAStBase** (Kohl and Ruckdeschel 2013a), **ROptEst** (Kohl and Ruckdeschel 2013b), and specialized to biostatistical applications in **RobLoxBioC** (Kohl 2013), compare Kohl (2005) and Kohl and Deigner (2010).

When interest lies in multiple convolutions we provide a function `convpow` to quickly and reliably compute convolution powers; in particular sample size then is not an issue. Otherwise, i.e., for non-identically distributed summands, we either have to appeal to asymptotics in some way or do it summation by summation. We can say though, that our approach works reliably

for up to 40 (non-)iid summands. In each case, we automatically provide respective quantile functions which are of particular interest in actuarial sciences and risk management.

Our paper is organized as follows: In Section 2, we discuss how an object oriented framework could enhance implementations of both probability distributions in general and convolution algorithms in particular. To this end, we sketch our implementation of distribution classes in R package **distr**. We also briefly discuss the dispatching decisions involved when a new object of a distribution class is generated by convolution and indicate how our convolution methods are used to set up a whole arithmetics acting on distributions. In Section 3, we present the general purpose FFT algorithm and some ramifications. Some forerunners in this direction and connections to other approaches are discussed in Section 4. In Section 5 we present checks for the accuracy and computational efficiency of our algorithm. At the end of this paper we provide some conclusions in Section 6.

2. OOP for probability distributions and convolution

2.1. OOP for probability distributions

There is a huge amount of software available providing functionality for the treatment of probability distributions. In this paper we will mainly focus on S, more specifically, on its open source implementation R, but of course the considerations also apply for other extensible software like XploRe (MD*Tech 2007), Gauss (Aptech Systems, Inc. 2006), Simula, SAS (SAS Institute Inc. 2011) or MATLAB (The MathWorks, Inc. 2011). All these packages provide standard distributions, like normal, exponential, uniform, Poisson just to name a few.

There are limitations, however: Only distributions are available which either already ship with the software or in some add-on library or package, or for which oneself has provided an implementation. Automatic generation of new distributions is left out in general.

In many natural settings one wants to formulate algorithms once for all distributions, so that one should be able to use the actual distribution, say `D`, as argument to some function. This requires particular data types for distributions. Going ahead in this direction, this allows formulation of statements involving the expectation or variance of functions of random variables as usual in mathematics; i.e., no matter if the expectation involves a finite sum, a sum of infinite summands, or a (Lebesgue) integral. This idea is particularly well-suited for OOP, as described in Booch (1995), with its paradigms "inheritance" and "method overloading".

In the OOP concept, a dispatching mechanism selects the method to use at run-time. In our convolution case, the result of such an algorithm may then be a new distribution.

In his Java MCMC simulation package **HYDRA**, Warnes (2002) heads for a similar OOP approach providing a set of Java classes representing common statistical distributions, porting the C code underlying the R implementation. But, quoting the author from the respective web page, "[o]ther than grouping the PDF, CDF, etc into a single class for each distribution, the files don't (yet) make much use of OO design."

2.2. OOP in S: The S4 class concept

In base R, OOP is realized in the S3 class concept as introduced in Chambers (1993a,b), and by its successor, the S4 class concept, as described in detail in Chambers (2008). We work

with the S4 class concept.

Using the terminology of Bengtsson (2003), this concept is intended to be *FOOP* (function-object-oriented programming) style, in contrast to *COOP* (class-object-oriented programming) style, which is the intended style in C++, for example.

In COOP style, methods providing access to or manipulation of an object are part of the object, while in FOOP style, they are not, but rather belong to so-called *generic functions* which are abstract functions allowing for arguments of varying type/class. A dispatching mechanism then decides on run-time which method best fits the *signature* of the function, that is, the types/classes of (a certain subset of) its arguments. In C++, "overloaded functions" in the sense of Stroustrup (1987, Section 4.6.6) come next to this concept.

FOOP style has some advantages for functions like `+` having a natural meaning for many operand types/classes as in our convolution case. It also helps collaborative programming, as not every programmer providing functionality for some class has to interfere with the original class definition. In addition, as S, respectively R, is an interpreted language, a method incorporated in a S4 class definition would not simply be a pointer but rather the whole function definition and environment. Hence, the COOP-style paradigm in (standard) R entails arguable draw-backs and hence is not generally advisable within the S4 class system. This has been overcome to some extent, however, with the introduction of reference classes.

Since its introduction to R, the S4 class concept has allowed COOP style, that is, members (or *slots* in S4 lingo) have always been permitted to be functions, but we may say that use of functional slots in S4 is not standard, which may be judged against a thread on the R-devel mailing list on 2004-01-31 (`http://tolstoy.newcastle.edu.au/R/devel/04a/0185.html`). Use of functional slots has been extensively used in Bengtsson's (2003) **R.oo** package where the author circumvents the above-mentioned problems by a non-standard *call-by-reference* semantic.

For our distribution classes, we, too, use the possibility for function-type members, albeit only in a very limited way, and not extending the standard S4 system in any respect. Still, others have suggested to rather follow the S4 generic way for slots `r`, `d`, `p`, `q`, which however, in our opinion, would lead to many class definitions (a new one generated at each call to the convolution operation) instead of only few class definitions as in our design.

2.3. Implementation of distribution classes within the S4 class concept

In S/R, any distribution is given through four functions: `r`, generating pseudo-random numbers according to that distribution, `d`, the density or probability function/counting density, `p`, the CDF, and `q`, the quantile function. This is also reflected in the naming convention `[prefix]<name>` where `[prefix]` stands for `r`, `d`, `p`, or `q` and `<name>` is the (abbreviated) name of the distribution.

We call these functions *constitutive* as we regard them as integral part of a distribution object, and hence realize them as members (slots) of our distribution classes (with corresponding accessor functions `r`, `d`, `p`, and `q`) even though this causes some "code weight" for the corresponding objects.

Some people dislike our use of function names `r`, `d`, `p`, and `q`, because making function `q` generic, we overload the base R function `q` for quitting an R session (`q` retains in base R usage this way, though); this causes some problems in **RStudio** (which masks `q` for technical

reasons). The criticism against our use is that `q` then means quite different things according to the signature of the first argument. Still, we think that R should support context-dependent meanings if this entails, as here, self-explanatory and succinct notation.

A real benefit of our approach is grouping of routines which represent one distribution instead of having separate functions `rnorm`, `dnorm`, `pnorm`, and `qnorm` which otherwise are only connected by gentleman's agreement/naming convention. Consistency may become an issue then, of course: We cannot exclude the possibility that someone (inadvertently) puts together inadequate `r`, `d`, `p`, or `q` slots, manipulating the slots by assignments of the like `a@b <- 4`. As mentioned in Ruckdeschel, Kohl, Stabla, and Camphausen (2013, Section 9 and Example 13.7), this is not the intended way to generate distribution objects, though. To warrant a certain level of consistency, we provide generating functions for standard distributions and, following Gentleman (2003), corresponding accessor and replacement functions for each of the slots. Of course, consistency also holds for automatically generated distributions arising as return values from arithmetic operations.

Another justification for this approach can be given by considering convolution: Assume we would like to automatically generate the constitutive functions for the law of expressions like `X + Y` for objects `X` and `Y` of some distribution class. Following the FOOP paradigm the function `cdf` to compute the CDF would not be part of the class but some method of a corresponding generic function. Then, as each distribution has its own constitutive functions and the CDF method is dispatched on the signature, `cdf` would need to have a particular method for every (new) distribution and in particular would need a new class for every newly generated distribution. That is, very soon the dispatching mechanism would have to decide between lots of different signatures. In contrast, when `cdf` is a member of a class, dispatching is not necessary and calculations are more efficient. This efficiency is not a question of extracting the CDF as a functional slot, instead of getting it from dispatch, but rather due to the necessity to have sufficiently many classes for the return values of convolutions of arbitrary distributions: As a rule, the convolution of two arbitrary distributions f and g will generate a new distribution $f * g$ for which there has not been an implementation before. So in order to have access to $f * g$ in FOOP manor, the CDF or density or quantile function has either to be computed "on the fly" for each evaluation or a new S4 class and a corresponding convolution method has to be generated when calling something like `cdf(conv(f, g))`, or the class of admitted operands (arguments) of `conv()` has to be restricted, such that the result object is again a member of a (possibly parametric) set of distribution functions.

In fact, in R package **actuar**, Goulet (2008) pursues the FOOP approach just sketched in their function `aggregateDist`. To escape the possible multitude of new distribution classes, the authors restrict themselves to particular probability distributions; i.e., the "(a, b, 0) or (a, b, 1) families of distributions" (see cited reference for their definition) and so offer alternatives to compute the convolution (see help to `aggregateDist`).

Their approach and ours combine well though: Our extension package **distrEx** even depends on package **actuar**, using some of the additional root distributions provided there.

2.4. Convolution as a particular method in distr

Contrary to the `r`, `d`, `p`, and `q` functions just discussed, the computation of convolutions ideally fits in the FOOP setup where method dispatch works as follows:

In case there are better algorithms or even exact convolution formulae for the given signa-

ture, as for independent variables distributed according to $\mathrm{Bin}(n_i, p)$, $i = 1, 2$, or $\mathrm{Poisson}(\lambda_i)$ or $\mathcal{N}(\mu_i, \sigma_i^2)$ etc., the dispatching mechanism for S4 classes will realize that, will use the best matching existing `+`-method and will generate a new object of the corresponding class. However, this case is exceptional and we do not have to dispatch among too many methods.

As our object oriented framework allows to override the default procedure easily by more specialized algorithms by method dispatch, the focus of our default algorithm, Algorithm 1, is not to provide the most refined techniques to achieve high accuracy but rather to be applicable in a most general setting. This default algorithm is based on FFT and will be described in detail in the next section. It originally applies to distribution objects of class '`LatticeDistribution`' where a lattice distribution is a discrete distribution whose support is a lattice of the form $a_0 + iw$, $a_0 \in \mathbb{R}$, $w \in \mathbb{R} \setminus \{0\}$ with $i \in \mathbb{N}_0$ (or $\{0, 1, \ldots, n\}$, $n \in \mathbb{N}$). In our implementation this class is a subclass of class '`DiscreteDistribution`' which in addition to its respective mother class '`UnivariateDistribution`' has an extra slot `support`, a numerical vector containing the support (if finite, and else a truncated version carrying more than $1 - \varepsilon$ mass). Besides '`DiscreteDistribution`', class '`UnivariateDistribution`' has subclasses '`AbscontDistribution`' for absolutely continuous distributions, i.e., distributions with a (Lebesgue) density, and '`UnivarLebDecDistribution`' for a distribution in Lebesgue decomposed form, i.e., a mixture of an absolutely continuous part and a discrete part. Such distributions e.g., arise from truncation operations, or when a discrete distribution (with point mass at $\{0\}$) is multiplied with a(n) (absolutely) continuous one.

Our FFT-based algorithm starts with two lattice distributions with compatible lattices; i.e., we assume that the support of the resulting convolved distribution has length strictly smaller than the product of the lengths of the supports of the operands. For discrete distributions, we check whether they can be cast to lattice distributions with compatible lattices. If one operand is absolutely continuous, the other one discrete, we proceed by "direct computation". If both operands are absolutely continuous, as described in Algorithm 1, we first discretize them to lattice distributions with same width w. The CDFs F_1 and F_2 used in this algorithm will be obtained from the corresponding `p`-slots. For objects of class '`UnivarLebDecDistribution`', we proceed component-wise.

Slots `p` and `d` of the resulting new object are then filled by Algorithm 1, described in detail in the next section. More precisely we will use variants of this algorithm for the absolutely continuous and the discrete/lattice case, respectively.

Slot `r` of the new object consists in simply simulating pairs of variables by means of the `r` slots of the convolutional summands and then summing these pairs. Slot `q` is obtained by numerical inversion: For a continuous approximation of the quantile function we evaluate the function in slot `p` on an x-grid, exchange x- and y-axis and interpolate linearly between the grid points, for discrete distributions `D` we start with the vector `pvec <- p(D)(support(D))` and search for the support-point belonging to the largest member of `pvec` smaller than or equal to the argument of `q`.

2.5. General arithmetics of distributions in distr

Implementing distributions as classes enables us to implement fairly complete and accurate arithmetics acting on correspondingly distributed random variables.

The first observation to be made is that the image distribution of affine linear transformations can be explicitly spelled out for each of the slots `r`, `d`, `p`, and `q`. Hence, if `X` and `Y` are

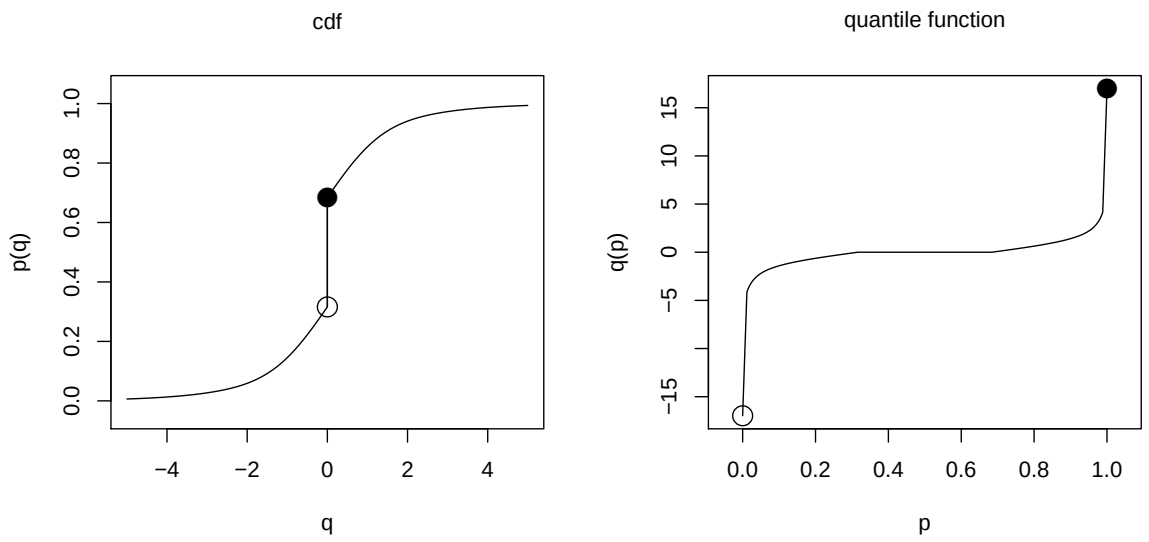

Figure 2: Distribution of $X = N * P$ if $N \sim \mathcal{N}(0,1)$ and $P \sim \text{Poisson}(\lambda)$

both univariate distributions, we define `X - Y` to mean the convolution of `X` and `-Y`. For distributions with support contained in $(0, \infty)$, also multiplication is easy: as log and exp are strictly monotone and differentiable transformations, the respective image distributions may also be spelled out explicitly, for each of the slots `r`, `d`, `p`, and `q`, and the `X * Y = exp(log(X) + log(Y))`. Splitting up the support of a distribution into positive, negative, and 0-part (where each of the intersections may be empty), and interpreting this as a mixture of possibly three distinct distributions, we can also allow general $\mathbb{R}$-valued distributions as factors in multiplications; the result can then possibly be a mixture of a Dirac distribution in 0 and an absolutely continuous distribution. For division we note that for distributions with positive support, `X/Y = exp(log(X) - log(Y))`, and similar arguments also allow us to cover powers, i.e., expressions like `X^Y`. As an example, let us see how the distribution of $X = N * P$ looks like if $N \sim \mathcal{N}(0,1)$ and $P \sim \text{Poisson}(\lambda)$ (see Figure 2):

```
R> X <- Norm() * Pois(lambda = 1)
R> q(X)((1:3)/4)

[1] -3.471003e-01 -3.333333e-07  3.471003e-01

R> plot(X, cex.inner = 1, to.draw.arg = c(1, 2),
+    inner = c("cdf", "quantile function"))
```

3. General purpose FFT algorithm

The main idea of our algorithm is to use DFT, which may be calculated very fast by FFT. Hence, we start with a brief introduction to DFT and its convolution property (cf. Theorem 1) where we follow Lesson 8 of Gasquet and Witomski (1999). Afterwards, we describe the convolution of CDFs/densities in Section 3.2.

3.1. Discrete Fourier transform (DFT)

Let $m \in \mathbb{N}$ and let $(x_n)_{n\in\mathbb{Z}}$ be a sequence of complex numbers with period m; i.e., $x_{n+m} = x_n$ for all $n \in \mathbb{Z}$. Then, the DFT of order m is,

$$\mathrm{DFT}_m \colon \mathbb{C}^m \to \mathbb{C}^m,\ (x_0, x_1, \ldots, x_{m-1}) \mapsto (\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{m-1}), \tag{1}$$

where

$$\hat{x}_n = \frac{1}{m} \sum_{j=0}^{m-1} x_j \omega_m^{jn}, \qquad \omega_m = e^{-2\pi i/m},\ i = \sqrt{-1}. \tag{2}$$

We obtain the DFT $(\hat{x}_n)_{n\in\mathbb{Z}}$ of $(x_n)_{n\in\mathbb{Z}}$ by the periodic extension $\hat{x}_{n+m} = \hat{x}_n$ for all $n \in \mathbb{Z}$. DFT_m is represented by a matrix Ω_m with entries ω_m^{jk} $(j, k = 0, 1, \ldots, m-1)$ and inverse $\Omega_m^{-1} = 1/m\,\overline{\Omega}_m$ ($\overline{\Omega}_m$ the conjugate DFT_m); i.e., DFT_m is linear and bijective.

Remark 1 **(a)** *Computing $\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{m-1}$ directly from Equation 2, requires $(m-1)^2$ complex multiplications and $m(m-1)$ complex additions. But, FFT as introduced by Cooley and Tukey (1965), is of just order $m \log m$. It works best for the case $m = 2^p$ $(p \in \mathbb{N})$; see Lesson 9 of Gasquet and Witomski (1999). In case $m = 2^{10}$, direct computation needs* 1046529 *multiplications and* 1047552 *additions, whereas FFT only requires* 4097 *multiplications and* 10240 *additions; see also Table 9.1 (ibid.).*

(b) *If $(x_n)_{n\in\mathbb{Z}}$ is a sequence of real numbers, it is possible to halve the computational costs, see Section 8.3 of Gasquet and Witomski (1999).*

(c) *FFT is available in R as function* `fft`.

For DFTs we have the following theorem, proven, e.g., in Kohl (2005, Theorem C.1.2).

Theorem 1 *Let $x = (x_n)_{n\in\mathbb{Z}}$ and $y = (y_n)_{n\in\mathbb{Z}}$ be two sequences of complex numbers with period m and let $\hat{x} = (\hat{x}_n)_{n\in\mathbb{Z}}$ and $\hat{y} = (\hat{y}_n)_{n\in\mathbb{Z}}$ be the corresponding DFTs. Then, the circular convolution product of x and y is defined as,*

$$x * y = \left(\sum_{j=0}^{m-1} x_j y_{n-j} \right)_{n\in\mathbb{Z}} \tag{3}$$

and, with $\hat{x}\,\hat{y} = (\hat{x}_n \hat{y}_n)_{n\in\mathbb{Z}}$, it holds,

$$\hat{z} = m\,\hat{x}\,\hat{y} \qquad \text{with} \qquad z = x * y. \tag{4}$$

This theorem implies the following result for N-fold convolution products.

Proposition 1 *Let $x = (x_n)_{n\in\mathbb{Z}}$ be a sequence of complex numbers with period m and let $\hat{x} = (\hat{x}_n)_{n\in\mathbb{Z}}$ be the corresponding DFT. Then, it holds,*

$$\widehat{*_{i=1}^{N} x} = m^{N-1}\,\hat{x}^N \qquad N \in \mathbb{N} \tag{5}$$

The proof immediately follows from Theorem 1 by induction.

3.2. Convolution algorithm

DFT is formulated for discrete (equidistant) sequences of complex numbers, as which we may interpret the probability function of the following special integer lattice distributions

$$F_i(x) = \sum_{j=0}^{m-1} p_{i,j}\, \mathrm{I}_{[j,\infty)}(x) \qquad i = 1, 2, \tag{6}$$

with $p_{i,j} \geq 0$ for $j = 0, 1, \ldots, m-1$ and $\sum_{j=0}^{m-1} p_{i,j} = 1$ and where $x \in \mathbb{R}$ and $m = 2^q$ ($q \in \mathbb{N}$). We extend $p_{i,j}$ ($i = 1, 2$, $j = 0, \ldots, m-1$) to two sequences $p_i = (p_{i,n})_{n\in\mathbb{Z}}$ of real numbers with period $2m$ via,

$$p_{i,j} = 0 \qquad i = 1, 2 \qquad j = m, \ldots, 2m-1 \qquad \text{(zero padding)} \tag{7}$$

and $p_{i,k+2m} = p_{i,k}$ for all $k \in \mathbb{Z}$. Then, the convolution F of F_1 and F_2 is an integer lattice distribution given by

$$F(x) = (F_1 * F_2)(x) = \sum_{j=0}^{2m-1} \pi_j\, \mathrm{I}_{[j,\infty)}(x) \qquad \text{with} \qquad \pi_j := \sum_{k=0}^{2m-1} p_{1,k} p_{2,j-k}, \tag{8}$$

where in particular $\pi_{2m-1} = 0$. Hence, in view of Theorem 1, $\pi = (\pi_n)_{n\in\mathbb{Z}} = p_1 * p_2$ and we can compute π using FFT and its inverse. This result forms the basis of Algorithm 1.

As it stands, Algorithm 1 will be presented for the case of absolutely continuous distributions, but with slight and obvious modifications this algorithm works for quite general distributions; for more details see also Section 3.3.

Algorithm 1 *Assume two absolutely continuous distributions F_1, F_2 on $\mathbb{R}$.*

Step 1: (Truncation)
If the support of F_i ($i = 1, 2$) is unbounded or "too large", we define numbers $A_i, B_i \in \mathbb{R}$, for given $\varepsilon > 0$, such that,

$$F_i\big((-\infty, A_i)\big) = \frac{\varepsilon}{2} \qquad \text{and} \qquad F_i\big((B_i, \infty)\big) = \frac{\varepsilon}{2} \tag{9}$$

and set $A = \min\{A_1, A_2\}$ and $B = \max\{B_1, B_2\}$. If this is not the case, we define $A := \min\{F_1^{-1}(0), F_2^{-1}(0)\}$ and $B := \max\{F_1^{-1}(1), F_2^{-1}(1)\}$, where F_i^{-1} ($i = 1, 2$) are the quantile functions of F_i.

Step 2: (Discretization on a real grid)
Given $m = 2^q$ ($q \in \mathbb{N}$) and F_i ($i = 1, 2$), we define the lattice distributions

$$G_i(x) := \sum_{j=0}^{m-1} p_{i,j}\, \mathrm{I}_{[A+(j+0.5)h,\infty)}(x), \qquad h = \frac{B-A}{m}, \tag{10}$$

where $p_{i,j} = F_i\big([A + jh, A + (j+1)h]\big)$ for $j = 0, 1, \ldots, m-1$.

Step 3: (Transformation to an integer grid)
Based on G_i $(i = 1, 2)$, we define the integer lattice distributions

$$\tilde{G}_i(x) := \sum_{j=0}^{m-1} p_{i,j}\, \mathrm{I}_{[j,\infty)}(x), \qquad i = 1, 2, \tag{11}$$

and extend $p_{i,j}$ $(i = 1, 2,\ j = 0, \ldots, m-1)$ to two sequences $p_i = (p_{i,n})_{n\in\mathbb{Z}}$ of real numbers with period $2m$ via,

$$p_{i,j} = 0, \qquad i = 1, 2, \qquad j = m, \ldots, 2m-1, \qquad \textit{(zero padding)} \tag{12}$$

and $p_{i,k+2m} = p_{i,k}$ for all $k \in \mathbb{Z}$.

Step 4: (Convolution by FFT on integer grid)
We calculate $\tilde{G} = \tilde{G}_1 * \tilde{G}_2$ by FFT and its inverse as given in Equation 8; i.e.,

$$\tilde{G}(x) = \sum_{j=0}^{2m-1} \pi_j\, \mathrm{I}_{[j,\infty)}(x), \qquad \pi_j := \sum_{k=0}^{2m-1} p_{1,k} p_{2,j-k}, \tag{13}$$

where in particular $\pi_{2m-1} = 0$.

Step 5: (Back-transformation to real grid)
Given $\tilde{G}$, we obtain $G = G_1 * G_2$ by,

$$G(x) = \sum_{j=0}^{2m-2} \pi_j\, \mathrm{I}_{[2A+(j+1.5)h,\infty)}(x). \tag{14}$$

Here we improve the accuracy of the results using a continuity correction of $h/2$.

Step 6: (Smoothing)
Next, we use interpolation of the values of G on $\{2A, 2A + 1.5h, \ldots, 2B - 0.5h, 2B\}$ by linear functions to get a continuous approximation $F^\natural$ of $F = F_1 * F_2$. We obtain a continuous approximation $f^\natural$ of the density f of F by multiplying $\{0, \pi_0, \pi_1, \ldots, \pi_{2m-2}, 0\}$ by h and interpolating these values on the grid $\{2A, 2A + h, \ldots, 2B - h, 2B\}$ (no continuity correction) using linear functions.

Step 7: (Standardization)
To make sure that the approximation $F^\natural$ is indeed a probability distribution, we standardize $F^\natural$ and $f^\natural$ by $F^\natural\big([2A, 2B]\big)$ and $\int f^\natural(x)\,dx$, respectively, where $\int f^\natural(x)\,dx$ may be calculated numerically exactly, since $f^\natural$ is a piecewise linear function.

For some instructive code examples like the computation of (an approximation to) the stationary regressor distribution of an AR(1) process see Ruckdeschel *et al.* (2013).

3.3. Ramifications and extensions of this algorithm

Algorithm 1 for lattice distributions: Obviously, Algorithm 1 applies to lattice distributions F_1, F_2 on $\mathbb{R}$ defined on the same grid. In this case the algorithm essentially reduces to Steps 1–5 and 7. Moreover, the results are numerically exact if the lattice distributions have finite support; see Section 5. In this case the algorithm consists only of Steps 2–5.

Specification of "too large": In Step 1, a support is considered as "too large" if a uniform grid with a reasonable step-length produces too many grid points. In the same sense, the loss of mass included in Step 1 is, to some extent, controllable and in many cases negligible.

Richardson extrapolation: A technique to enhance the accuracy of Algorithm 1 for given q is extrapolation. But, for this to work properly, we need additional smoothness conditions for the densities. We could take this into account by introducing a new subclass '`SmoothDistribution`' for distributions with sufficiently smooth densities and a corresponding new method for the operator `+`; see also Section 4.1.

Exponential tilting: As a wrap-around effect, summation modulo m (cf. Equation 3) induces an aliasing error. Especially for heavy-tailed distributions – again at the cost of additional smoothness conditions for the densities – Algorithm 1 can thus be improved by a suitable change of measure (exponential tilting). So one might conceive a further subclass '`HeavyTailedSmoothDistribution`' and overload `+` for objects of these classes using exponential tilting; see also Section 4.1.

Modification for M-estimators: In view of Proposition 1, Algorithm 1 may easily be modified to compute an approximation of the exact finite-sample distribution of M-estimates, compare Ruckdeschel and Kohl (2010). In the cited reference, we compare the results obtainable with this modified algorithm to other approximations of the exact finite-sample distribution of M-estimates, like the saddle point approximation and higher order asymptotics.

4. Connections to other approaches

4.1. Algorithms based on DFT

A very similar algorithm was proposed by Bertram (1981) to numerically evaluate compound distributions in insurance mathematics where he assumes claim size distributions of lattice type. Numerical examples and comparisons to other methods can be found in Bühlmann (1984) and Feilmeier and Bertram (1987).

A mathematical formulation of the corresponding algorithm is included in Grübel and Hermesmeier (1999), with a focus on the investigation of the aliasing error which, as they show, can be eliminated by exponential tilting in case of a lattice type claim size distribution. But, even without the smoothness assumptions needed for exponential tilting, in many cases, the aliasing error can also be made very small if we choose ε in Step 1 of Algorithm 1 small enough and q in Step 2 large enough. This strategy is no cure-all remedy, of course, as a too small ε or a too large q would lead to numerical problems by cancellation effects, and, in addition, could cause memory problems.

Moreover, if one considers absolutely continuous probability distributions with possibly non-compact support, initial truncation and discretization steps are necessary; see Steps 1 and 2 of Algorithm 1. This is also discussed in Monahan (2011, Chapter 14, Example (C)). The corresponding discretization error is studied in Grübel and Hermesmeier (2000) and, under certain smoothness conditions, shown to be reducible by Richardson extrapolation.

Efficient and precise algorithms based on FFT for the convolution of heavy-tailed distributions are considered in Schaller and Temnov (2008).

Applications of FFT-based convolutions of distributions cover computation of the total claim size distribution and probability of ruin in risk theory and insurance mathematics (compare Embrechts, Grübel, and Pitts 1993) as well as computation of stationary waiting time distribution in queuing models for general customer inter-arrival time distributions and service time distributions, see Grübel (1991).

4.2. Other algorithms

For continuous distributions, instead of starting with a discretization of the CDF right away, we could also use the actual characteristic functions, i.e., the Fourier transform of the corresponding distributions which then get inverted by the usual Fourier inversion formulae, see e.g., Chung (1974, Section 6.2). Lumely (2007) mentions this approach for linear combination of independent χ^2 distributions where closed form expressions for the characteristic functions are available, see also Monahan (2011, Chapter 14, Example (E)). Still, inverting characteristic functions is not always advisable – see, e.g., convolution powers of the uniform distribution on $[-1/2, 1/2]$: The corresponding characteristic functions are $(\sin(t/2)/t)^n$ which if naïvely inverted causes quite some numerical problems. A more comprehensive account of this approach can be found in Cavers (1978) and Abate and Whitt (1992, 1995).

Similarly, but with a restricted application range due to integrability one could stay on the real line using Laplace transforms; compare Abate and Whitt (1992, 1995).

In actuarial science, recursive schemes to compute convolution powers, the so-called *Panjer recursions*, have been in use for a long time. As Temnov and Warnung (2008) show, these methods are slower than FFT when a sufficient precision of the estimated quantile is needed.

5. Accuracy and computational efficiency of our algorithm

To assess the accuracy and computational efficiency of our algorithm, we present checks for n-fold convolution products where the exact results are known. In addition, we approximate probabilities of non-central χ^2 distributions.

5.1. Accuracy

We determine the precision of the convolution algorithm in terms of the total variation distance of the distributions,

$$d_v(P,Q) = \tfrac{1}{2}\int |p-q|\,d\mu = \sup_{B\in\mathbb{B}} \big|P(B) - Q(B)\big|, \tag{15}$$

where $P, Q \in \mathcal{M}_1(\mathbb{B})$ with $dP = p\,d\mu$, $dQ = q\,d\mu$ for some σ-finite measure μ on $(\mathbb{R}, \mathbb{B})$ and the Kolmogorov distance of the CDFs,

$$d_\kappa(P,Q) = \sup_{t\in\mathbb{R}} \big|P\big((-\infty,t]\big) - Q\big((-\infty,t]\big)\big|. \tag{16}$$

In the sequel $d_v^\natural$ and $d_\kappa^\natural$ stand for the numerical approximations of d_v and d_κ. Due to numerical inaccuracies we obtain $d_\kappa^\natural > d_v^\natural$ in some cases, although, of course, $d_\kappa \le d_v$.

The first example treats binomial distributions and shows that the convolution algorithm is very accurate for integer lattice distributions with finite support.

Example 1 Assume $F = \mathrm{Bin}\,(k, p)$ with $k \in \mathbb{N}$ and $p \in (0, 1)$. Then, the n-fold convolution product is $F^{*n} = \mathrm{Bin}\,(nk, p)$ $(n \in \mathbb{N})$. Let f_n and $f^{\natural}$ be the probability functions of F^{*n} and $F^{\natural}$, respectively. Then, we may determine $d_v^{\natural}$ and $d_\kappa^{\natural}$ numerically exact by,

$$d_v^{\natural}(F, F^{\natural}) = \frac{1}{2} \sum_{j=0}^{nk} |f_n(j) - f^{\natural}(j)| \tag{17}$$

and

$$d_\kappa^{\natural}(F, F^{\natural}) = \max_{j \in \{0,\ldots,nk\}} \left|F^{*n}([0, j]) - F^{\natural}([0, j])\right|. \tag{18}$$

We obtain the results contained in Table 1 which show that Algorithm 1 is very accurate in case of binomial distributions, where the values of k and p are chosen arbitrarily. To get the corresponding results we use our R packages **distr** and **distrEx**. For example,

```
R> library("distrEx")
R> distroptions(TruncQuantile = 1e-15)
R> B1 <- Binom(size = 30, prob = 0.8)
R> B2 <- convpow(B1, N = 10)
R> D1 <- as(B1, "LatticeDistribution")
R> D2 <- convpow(D1, N = 10)
R> TotalVarDist(B2, D2)

total variation distance
            2.918596e-16

R> KolmogorovDist(B2, D2)

Kolmogorov distance
       2.220446e-16
```

where `B2` is computed using the exact formula and `D2` is the approximation via FFT. To increase accuracy we change the default value of option `TruncQuantile` from 1e−5 to 1e−15.

We skip the example for the Poisson distribution where the results of the convolution algorithm are very accurate, too. Corresponding code is included in the supplementary material. In the next two examples we consider the convolution of absolutely continuous distributions. We determine $d_v^{\natural}(F, F^{\natural})$ by numerical integration using the R function `integrate`. To compute the Kolmogorov distance, we evaluate $d_\kappa^{\natural}(F, F^{\natural})$ on a grid obtained by the union of a deterministic grid of size 1e05 and two random grids consisting of 1e05 pseudo-random numbers of the considered distributions. We begin with results for normal distributions.

Example 2 Assume $F = \mathcal{N}\,(\mu, \sigma^2)$ with $\mu \in \mathbb{R}$ and $\sigma \in (0, \infty)$. Then, for $n \in \mathbb{N}$, it holds, $F^{*n} = \mathcal{N}\,(n\mu, n\sigma^2)$. Starting with $\mathcal{N}\,(0, 1)$ and A and B as defined in Step 1 of Algorithm 1

n	k	p	$d_v^\natural$	$d_\kappa^\natural$
2	10	0.5	2.9e−16	2.2e−16
5	20	0.7	1.7e−15	1.1e−15
10	30	0.8	2.4e−15	1.0e−15
100	15	0.2	4.5e−15	3.4e−15
1000	50	0.4	8.3e−13	4.2e−13

Table 1: Precision of the convolution of binomial distributions via FFT; see Example 1. Note: For values beyond 1e−13, minor differences may occur between different R hardware architectures; e.g., on Ubuntu, 64bit, in case, $n = 10$, $k = 30$, $p = 0.8$, we get $d_v^\natural = 2.4\text{e}{-15}$, while in Windows 7, 32bit, we get 2.5e−15 (both on R 3.1.1); the tables in this paper come from Ubuntu, 64bit.

n	ε	q	μ	σ	$d_v^\natural$	$d_\kappa^\natural$
			−10.0	100.0	1.2e−06	2.1e−06
			−2.0	5.0	1.1e−06	2.1e−06
2	1e−08	12	0.0	1.0	1.2e−06	2.1e−06
			1.0	50.0	1.2e−06	2.1e−06
			100.0	1000.0	1.2e−06	2.1e−06

Table 2: Precision of the convolution of normal distributions via FFT is independent of the parameters μ and σ; see Example 2.

we obtain $\tilde{A} = \sigma A + \mu$ and $\tilde{B} = \sigma B + \mu$ in case of $\mathcal{N}(\mu, \sigma^2)$. That is, the grid transforms in the same way as the normal distributions do. Thus, we expect the precision of the results to be independent of μ and σ. This is indeed confirmed by the numerical calculations; see Table 2. We therefore may consider $\mu = 0$ and $\sigma = 1$ for the study of the accuracy of the convolution algorithm subject to $n \in \mathbb{N}$, $\varepsilon > 0$ (Step 1) and $q \in \mathbb{N}$ (Step 2). The results included in Table 3 show that the precision is almost independent of n. It mainly depends on q where the maximum accuracy, we can reach, is of order ε. The results can be computed with our R packages **distr** and **distrEx** similarly to the binomial and Poisson case.

```
R> library("distrEx")
R> distroptions(TruncQuantile = 1e-10)
R> distroptions(DefaultNrFFTGridPointsExponent = 14)
R> N1 <- Norm(mean = 0, sd = 1)
R> N2 <- convpow(N1, N = 2)
R> D1 <- as(N1, "AbscontDistribution")
R> D2 <- convpow(D1, N = 2)
R> distroptions(TruncQuantile = 1e-15)
R> TotalVarDist(N2, D2, rel.tol = 1e-10)

total variation distance
            9.768635e-08

R> KolmogorovDist(N2, D2)
```

n	ε	q	$d_v^\natural$	$d_\kappa^\natural$
2	1e−06	8	2.2e−04	3.9e−04
		10	1.3e−05	2.3e−05
		12	3.4e−06	1.8e−06
2	1e−08	10	1.9e−05	3.4e−05
		12	1.2e−06	2.1e−06
		14	8.2e−08	1.2e−07
2	1e−10	12	1.6e−06	2.7e−06
		14	9.8e−08	1.7e−07
		18	4.9e−10	5.3e−10

n	ε	q	$d_v^\natural$	$d_\kappa^\natural$
5	1e−08	12	3.3e−06	9.7e−04
		16	6.6e−08	6.1e−05
10	1e−08	12	1.2e−05	1.1e−05
		16	6.3e−08	3.5e−08
50	1e−08	12	1.6e−04	9.6e−05
		18	1.0e−07	5.3e−08

Table 3: Precision of the convolution of normal distributions via FFT; see Example 2.

```
Kolmogorov distance
       1.700888e-07
```

Next we turn to the convolution of exponential distributions leading to gamma distributions.

Example 3 We consider $F = \mathrm{Exp}(\lambda) = \Gamma(1, \lambda)$ with $\lambda \in (0, \infty)$. Then, for $n \in \mathbb{N}$, it holds $F^{*n} = \Gamma(n, \lambda)$. Analogously to the normal case in Example 2, the grid transforms according to $\tilde{A} = A/\lambda$ and $\tilde{B} = B/\lambda$. As expected, the precision of the results is confirmed to be independent of λ by our numerical computations; see Table 4. Next we study the dependence of the accuracy of Algorithm 1 on $n \in \mathbb{N}$, $\varepsilon > 0$ and $q \in \mathbb{N}$ where we may choose $\lambda = 1.0$. As in Example 2 the precision is almost independent of n. It mainly depends on q where the maximum accuracy, we can reach, is of order ε; see Table 5. The results can be computed with our R packages **distr** and **distrEx** similarly to the previous cases.

```
R> library("distrEx")
R> distroptions(TruncQuantile = 1e-8)
R> distroptions(DefaultNrFFTGridPointsExponent = 16)
R> E1 <- Exp(rate = 1)
R> E2 <- convpow(E1, N = 5)
R> D1 <- as(E1, "AbscontDistribution")
R> D2 <- convpow(D1, N = 5)
R> distroptions(TruncQuantile = 1e-15)
R> TotalVarDist(E2, D2, rel.tol = 1e-10)

total variation distance
            1.39883e-07

R> KolmogorovDist(E2, D2)

Kolmogorov distance
       9.455868e-08
```

n	ε	q	λ	$d_v^\natural$	$d_\kappa^\natural$
			0.01	5.7e−06	4.0e−05
			0.5	5.7e−06	4.0e−05
2	1e−08	12	1.0	5.7e−06	4.0e−05
			5.0	5.7e−06	4.0e−05
			10.0	5.7e−06	4.0e−05

Table 4: Precision of the convolution of exponential distributions via FFT is independent of the parameter λ; see Example 3.

n	ε	q	$d_v^\natural$	$d_\kappa^\natural$
2	1e−06	8	7.4e−04	4.7e−03
		10	5.0e−05	3.4e−04
		12	4.5e−06	2.2e−05
2	1e−08	10	7.9e−05	6.0e−04
		12	5.7e−06	4.0e−05
		16	3.6e−08	1.6e−07
2	1e−10	12	8.0e−06	6.2e−05
		14	4.9e−07	3.9e−06
		20	2.7e−10	9.6e−10

n	ε	q	$d_v^\natural$	$d_\kappa^\natural$
5	1e−08	12	2.6e−05	2.8e−05
		16	1.4e−07	9.5e−08
10	1e−08	12	1.4e−04	1.4e−04
		16	6.2e−07	5.3e−07
50	1e−08	12	4.9e−03	4.9e−03
		20	3.8e−07	3.8e−07

Table 5: Precision of the convolution of exponential distributions via FFT; see Example 3.

Remark 2 Example 3 reveals one minor flaw of Algorithm 1. The support of $\Gamma(n, \lambda)$ is $[0, \infty)$ whereas the convolution algorithm is only very accurate in $[2A+(n/2+0.5)h, \ldots, 2B-(n/2+0.5)h]$. That is, for small n ($n \leq 5$) the Kolmogorov distance is $F\big([0, 2A + (n/2 + 0.5)h)\big) - F^\natural\big([0, 2A + (n/2 + 0.5)h)\big)$. However, for bigger n this inaccuracy disappears as there is less and less mass in $[0,\ 2A + (n/2 + 0.5)h)$. Moreover, since $(n/2 + 0.5)h$ is very small, this also causes the numerical inaccuracy of $d_v^\natural$ for small n and leads to $d_\kappa^\natural > d_v^\natural$.

Example 4 In this last example we show how our FFT approach can be used to compute probabilities for non-central χ^2 distributions where the exact values are difficult to obtain. Let X be a non-central χ^2 distributed random variable with `df` degrees of freedom and non-centrality parameter `ncp`; i.e., $X \sim \chi^2_{\texttt{df}}(\texttt{ncp})$. Our goal is to approximate the CDF $\mathsf{P}(X \leq x)$ at $x \in (0, \infty)$. In Table 6 we reproduce the values of Patnaik (1949) and Ittrich, Krause, and Richter (2000), and compare them to approximations by function `pchisq` of package **stats** (R Core Team 2014), as well as the results of three FFT approaches (FFT1–FFT3). In the first case (FFT1) we approximate X by

$$X \approx Z_1^2 + Z_2^2 + \ldots + Z_{\texttt{df}}^2 \qquad \text{with} \qquad Z_i \sim \mathcal{N}\left(\sqrt{\frac{\texttt{ncp}}{\texttt{df}}}, 1\right). \tag{19}$$

Secondly (FFT2) we use

$$X \approx Z_1^2 + Z_2^2 + \ldots + Z_{\texttt{df}-1}^2 + Z_{\texttt{df}}^2, \tag{20}$$

where $Z_i \sim \mathcal{N}(0, 1)$ for $i = 1, \ldots, \texttt{df} - 1$ and $Z_{\texttt{df}} \sim \mathcal{N}(\sqrt{\texttt{ncp}}, 1)$. Our third approximation (FFT3) reads

$$X \approx Y + Z^2, \tag{21}$$

df	ncp	x	Patnaik	Ittrich *et al.*	R Core Team	FFT1	FFT2	FFT3
		1.765	0.0500	0.0499994	.....	86	2	.....
	4	10.000	0.7118	0.7117928	.....	5	.....	.....
4		17.309	0.9500	0.9499957	.....	.....	.....	.....
		24.000	0.9925	0.9924604	.....	.....	.....	.....
	10	10.000	0.3148	0.3148207	.....	4	6	.....
	1	4.000	0.1628	0.1628330	.....	13	15	.....
		16.004	0.9500	0.9500015	.....	4	.....	6
7		10.257	0.0500	0.0499942	.....	39	39	.....
	16	24.000	0.5898	0.5863368	.....	6	4	.....
		38.970	0.9500	0.9499992	.....	.....	.....	.....
	6	24.000	0.8187	0.8173526	.....	10	11	.....
12	18	24.000	0.2901	0.2900495	.....	73	71	.....
	8	30.000	0.7880	0.7880015	.....	79948	79994	.....
16		40.000	0.9632	0.9632255	.....	43	1	.....
	32	30.000	0.0609	0.0628420	.....	392	09	.....
		60.000	0.8316	0.8315635	.....	14	23	.....
		36.000	0.1567	0.1567111	.....	018	023	.....
24	24	48.000	0.5296	0.5296284	.....	177	174	.....
		72.000	0.9667	0.9666954	.....	44	41	.....

Table 6: Approximations of the CDF of non-central χ^2 distributions via FFT; see Example 4. ($\varepsilon = 1e{-}08$, $q = 18$, only the decimal places which are different to Ittrich *et al.* are given).

where $Y \sim \chi^2_{\mathtt{df}-1}(0)$ (a central χ^2 distribution) and $Z \sim \mathcal{N}\left(\sqrt{\mathtt{ncp}}, 1\right)$.

For the FFT computations we used $\varepsilon = 1e{-}08$ and $q = 18$. All three FFT approaches give very good approximations. In particular, FFT3 yields results which have the same accuracy as `pchisq` and the approximation of Ittrich *et al.* (2000).

```
R> library("distr")
R> distroptions(withgaps = FALSE, TruncQuantile = 1e-8)
R> distroptions(DefaultNrFFTGridPointsExponent = 18)
R> df0 <- ncp0 <- 4
R> x0 <- 1.765
R> Z2 <- Norm(mean = sqrt(ncp0/df0))^2
R> res1 <- convpow(Z2, N = df0)
R> Z2 <- Norm()^2
R> X2 <- convpow(Z2, N = df0 - 1)
R> Y2 <- Norm(mean = sqrt(ncp0))^2
R> res2 <- X2 + Y2
R> res3 <- Chisq(df = df0 - 1) + Y2
R> res <- c(FFT1 = p(res1)(x0), FFT2 = p(res2)(x0), FFT3 = p(res3)(x0),
+    R = pchisq(x0, df = df0, ncp = ncp0))
R> res

      FFT1       FFT2       FFT3          R
0.04999865 0.04999924 0.04999936 0.04999937
```

5.2. Computational efficiency

To judge the computational efficiency of our algorithm, let us check it in a situation where the exact solution of the convolution is known, i.e., at the 10-fold convolution of independent $\chi^2_1(0)$ distributions. As timings are of course subject to hardware considerations we report relative timings, where as reference we use the implementation in R package **actuar**. As for general distributions, **actuar** already needs probabilities evaluated on a grid, we have to wrap the respective function `aggregateDist` into a function `convActuar` first, providing a respective discretization.

```
R> gc()
R> library("actuar")
R> distroptions(TruncQuantile = 1e-5)
R> distroptions(DefaultNrFFTGridPointsExponent = 12)
R> convActuar <- function(N = 2, df = 1, ncp = 0, method = "lower") {
+    D1 <- Chisq(df = df, ncp = ncp)
+    lo <- getLow(D1);  up <- getUp(D1)
+    dGPExp <- getdistrOption("DefaultNrFFTGridPointsExponent")
+    M <- 2^max(dGPExp - floor(log(N)/log(2)), 5)
+    h <- (up - lo)/M
+    probs <- discretize(pchisq(x, df = df, ncp = ncp), from = lo,
+      to = up, by = h, method = method)
+    x <- seq(from = N * lo + N/2 * h, to = N * up - N/2 * h, by = h)
+    x <- c(x[1] - h, x[1], x + h)
+    dx <- aggregateDist(method = "convolution",
+      model.freq = c(rep(0, N), 1), model.sev = probs)
+    list(d = dx, x = x)
+  }
```

Our algorithm compares well as to both timings and accuracy with each of the methods implemented in function `discretize` of package **actuar**, i.e., `"rounding"`, `"lower"`, or `"upper"`:

```
R> res1 <- convActuar(method = "rounding")
R> res2 <- convActuar(method = "upper")
R> res3 <- convActuar(method = "lower")
R> D1 <- as(Chisq(), "AbscontDistribution")
R> D2 <- convpow(D1 = D1, N = 2)
R> .rd <- summary(abs(res1$d(knots(res1$d)) - p(D2)(
+    res1$x[c(-1, -length(res1$x))])))
R> .lo <- summary(abs(res3$d(knots(res3$d)) - p(D2)(res3$x)))
R> .up <- summary(abs(res2$d(knots(res2$d)) - p(D2)(
+    res2$x[c(-1, -length(res2$x))])))
R> rbind(.rd, .lo, .up)

         Min.   1st Qu.    Median      Mean   3rd Qu.     Max.
.rd 2.005e-05 2.005e-05 2.018e-05 1.905e-04 4.636e-05 0.002910
.lo 1.463e-18 2.000e-05 2.017e-05 2.006e-04 4.664e-05 0.004855
.up 3.597e-08 1.968e-05 2.000e-05 7.301e-05 2.000e-05 0.001317
```

To see the differences more clearly, let us repeat this 100 times.

```
R> speedref <- function(expr.ref, rep.times = 100) {
+    ref.time <- system.time(for(i in 1:rep.times)
+      res <- eval(expr.ref))[1]
+    names(ref.time) <- NULL
+    return(list(res = res, ref.time = ref.time))
+  }
R> speedcheck <- function(expr, ref.time, rep.times = 100) {
+    r.time <- system.time(for(i in 1:rep.times)
+      res <- eval(expr))[1]/ref.time
+    names(r.time) <- NULL
+    return(list(res = res, r.time = r.time))
+  }
```

Comparing the relative timings we get the following result (where timings are reported as percentages relative to `convActuar`):

```
R> rep <- 100
R> refset <- speedref(quote(convActuar(N = 10, method = "lower")),
+    rep.times = rep)
R> r1 <- speedcheck(expr = quote(convpow(D1 = D1, N = 10)),
+    ref.time = refset$ref.time, rep.times = rep)
R> r2 <- speedcheck(expr = quote(convpow(D1 = Chisq(), N = 10)),
+    ref.time = refset$ref.time, rep.times = rep)
R> r3 <- speedcheck(expr = quote(Chisq(df = 10)),
+    ref.time = refset$ref.time, rep.times = rep)
R> res <- refset$res
R> D10 <- r1$res; Dex <- r2$res; Dcheck <- r3$res
R> round(refset$ref.time, 2)

[1] 3.8

R> print(round(c("actuar" = 1,"FFT" = r1$r.time,
+    "Chisq-Meth" = r2$r.time, "exact" = r3$r.time) * 100, 2))

    actuar        FFT Chisq-Meth      exact
    100.00      29.92      12.96       3.16
```

As to accuracy, our algorithm still is competitive:

```
R> .actu.D10 <- summary(abs(res$d(knots(res$d))[-c(1:8)] - p(D10)(res$x)))
R> .actu.Dex <- summary(abs(res$d(knots(res$d))[-c(1:8)] - p(Dex)(res$x)))
R> .Dex.D10 <- summary(abs(p(Dex)(res$x) - p(D10)(res$x)))
R> rbind(.actu.D10, .actu.Dex, .Dex.D10)
```

```
               Min.  1st Qu.    Median      Mean   3rd Qu.      Max.
.actu.D10 4.927e-17 1.00e-04 1.000e-04 1.927e-04 1.000e-04 0.001900
.actu.Dex 2.011e-08 1.00e-04 1.000e-04 2.455e-04 1.000e-04 0.003120
.Dex.D10  0.000e+00 2.22e-16 4.441e-16 6.111e-05 1.758e-07 0.001253
```

Note that the computations with `aggregateDist` of **actuar** get considerably more expensive when passing to finer discretizations, as we show in the following illustration which now cuts off lower and upper 10^{-6} quantiles (instead of 10^{-5} beforehand) and which uses 4 times as many discretization points (with only 30 replications)—again we report percentages relative to `convActuar`:

```
R> distroptions(TruncQuantile = 1e-6)
R> distroptions(DefaultNrFFTGridPointsExponent = 14)
R> rep <- 30
R> round(refset$ref.time,2)

[1] 83.25

R> print(round(c("actuar" = 1,"FFT" = r1$r.time,
+    "Chisq-Meth" = r2$r.time, "exact" = r3$r.time) * 100, 2))

    actuar        FFT Chisq-Meth      exact
    100.00       1.38       0.19       0.05

R> .actu.D10 <- summary(abs(res$d(knots(res$d))[-c(1:8)] - p(D10)(res$x)))
R> .actu.Dex <- summary(abs(res$d(knots(res$d))[-c(1:8)] - p(Dex)(res$x)))
R> .Dex.D10 <- summary(abs(p(Dex)(res$x) - p(D10)(res$x)))
R> rbind(.actu.D10, .actu.Dex, .Dex.D10)

               Min.   1st Qu.   Median      Mean  3rd Qu.      Max.
.actu.D10 7.384e-17 1.00e-05 1.000e-05 3.401e-05 1.00e-05 0.0005745
.actu.Dex 5.694e-11 1.00e-05 1.000e-05 4.172e-05 1.00e-05 0.0007835
.Dex.D10  0.000e+00 2.22e-16 4.441e-16 8.643e-06 1.56e-09 0.0002149
```

6. Conclusion

With our implementation of a general default convolution algorithm for distributions in the object oriented framework of R, we provide a flexible framework which combines scalable accuracy and reasonable computational efficiency. This framework lends itself for introductory courses in statistics where students can easily sharpen their intuition about how convolution and other arithmetic operations work on distributions. It is however not limited to educational purposes but can be fruitfully applied to many problems where one needs accurate representations of distributions of convolutions, as arising e.g., in finite sample risk of M-estimators (Ruckdeschel and Kohl 2010), actuarial sciences and risk management (Singh 2010), linguistics (Schaden 2012), and Bingo premia calculations (Kroisandt and Ruckdeschel 2012).

Acknowledgments

The first implementation of our FFT algorithm from end of 2003 is due to our former student, T. Stabla, who also collaborated with us on this topic until he left academia in 2006 and whom we warmly thank for his efforts. We thank Prof. Grübel for drawing our attention to relevant literature on this topic and two anonymous referees for their valuable comments. Financial support from Volkswagen Foundation in the framework of project "Robust Risk Estimation" (http://www.mathematik.uni-kl.de/~wwwfm/RobustRiskEstimation/) for which **distr** provides indispensable infrastructure, is gratefully acknowledged.

References

Abate J, Whitt W (1992). "The Fourier-Series Method for Inverting Transforms of Probability Distributions." *Queueing Systems*, **10**(1–2), 5–88.

Abate J, Whitt W (1995). "Numerical Inversion of Laplace Transforms of Probability Distributions." *ORSA Journal on Computing*, **7**(1), 36–43.

Aptech Systems, Inc (2006). *GAUSS Mathematical and Statistical System 8.0*. Aptech Systems, Inc., Black Diamond, Washington. URL http://www.Aptech.com/.

Bengtsson H (2003). "The **R.oo** Package – Object-Oriented Programming with References Using Standard R Code." In K Hornik, F Leisch, A Zeileis (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*. Vienna, Austria. URL http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/.

Bertram J (1981). "Numerische Berechnung von Gesamtschadenverteilungen. (Numerical Computation of Aggregate Loss Distributions)." *Blätter der Deutschen Gesellschaft für Versicherungs- und Finanzmathematik e.V. (DGVFM)*, **15**(2), 175–194.

Booch G (1995). *Objektorientierte Analyse und Design. (Object Oriented Analysis and Design)*. 1st edition. Addison-Wesley. Corrected reprint, German translation.

Bühlmann H (1984). "Numerical Evaluation of the Compound Poisson Distribution: Recursion or Fast Fourier Transform?" *Scandinavian Actuarial Journal*, **1984**(2), 116–126.

Cavers JK (1978). "On the Fast Fourier Transform Inversion of Probability Generating Functions." *IMA Journal of Applied Mathematics*, **22**(3), 275–282.

Chambers JM (1993a). "Classes and Methods in S. I: Recent Developments." *Computational Statistics*, **8**(3), 167–184.

Chambers JM (1993b). "Classes and Methods in S. II: Future Directions." *Computational Statistics*, **8**(3), 185–196.

Chambers JM (2008). *Software for Data Analysis. Programming with R*. Springer-Verlag, Berlin.

Chung KL (1974). *A Course in Probability Theory*. 2nd edition. Academic Press, New York.

Cooley JW, Tukey JW (1965). "An Algorithm for the Machine Calculation of Complex Fourier Series." *Mathematics of Computation*, **19**(90), 297–301.

Embrechts P, Grübel R, Pitts SM (1993). "Some Applications of the Fast Fourier Transform Algorithm in Insurance Mathematics." *Statistica Neerlandica*, **47**(1), 59–75.

Feilmeier M, Bertram J (1987). *Anwendung numerischer Methoden in der Risikotheorie. (Application of Numerical Methods in Risk Theory)*, volume 16 of *Schriftenreihe Angewandte Versicherungsmathematik*. Deutsche Gesellschaft für Versicherungsmathematik. Verlag Versicherungswirtschaft e.V., Karlsruhe.

Gasquet C, Witomski P (1999). *Fourier Analysis and Applications. Filtering, Numerical Computation, Wavelets*, volume 30 of *Texts in Applied Mathematics*. Springer-Verlag. Translated from the French by R. Ryan.

Gentleman R (2003). *Object Oriened Programming. Slides of a Short Course Held in Auckland*. URL http://www.stat.auckland.ac.nz/S-Workshop/Gentleman/Methods.pdf.

Goulet V (2008). "**actuar**: An R Package for Actuarial Science." *Journal of Statistical Software*, **25**(7), 1–37. URL http://www.jstatsoft.org/v25/i07/.

Grübel R (1991). "Algorithm AS 265: G/G/1 via Fast Fourier Transform." *Applied Statistics*, **40**(2), 355–365.

Grübel R, Hermesmeier R (1999). "Computation of Compound Distributions. I. Aliasing Errors and Exponential Tilting." *ASTIN Bulletin*, **29**(2), 197–214.

Grübel R, Hermesmeier R (2000). "Computation of Compound Distributions. II. Discretization Errors and Richardson Extrapolation." *ASTIN Bulletin*, **30**(2), 309–331.

Gupta A, Kumar V (1993). "The Scalability of FFT on Parallel Computers." *IEEE Transactions on Parallel and Distributed Systems*, **4**(8), 922–932.

Ittrich C, Krause D, Richter WD (2000). "Probabilities and Large Quantiles of Noncentral Generalized Chi-Square Distributions." *Statistics: A Journal of Theoretical and Applied Statistics*, **34**(1), 53–101.

Kohl M (2005). *Numerical Contributions to the Asymptotic Theory of Robustness*. PhD Thesis, Universität Bayreuth, Bayreuth. URL http://www.stamats.de/ThesisMKohl.pdf.

Kohl M (2013). ***RobLoxBioC**: Infinitesimally Robust Estimators for Preprocessing Omics Data*. R package version 0.9, URL http://robast.R-Forge.R-project.org/.

Kohl M, Deigner HP (2010). "Preprocessing of Gene Expression Data by Optimally Robust Estimators." *BMC Bioinformatics*, **11**(583).

Kohl M, Ruckdeschel P (2010). "R package **distrMod**: Object-Oriented Implementation of Probability Models." *Journal of Statistical Software*, **35**(10), 1–27. URL http://www.jstatsoft.org/v35/i10/.

Kohl M, Ruckdeschel P (2013a). ***RobAStBase**: Robust Asymptotic Statistics*. R package version 0.9, URL http://robast.R-Forge.R-project.org/.

Kohl M, Ruckdeschel P (2013b). ***ROptEst**: Optimally Robust Estimation*. R package version 0.9, URL http://robast.R-Forge.R-project.org/.

Kroisandt G, Ruckdeschel P (2012). "Bingo und Stochastik: Wieviele Spieler wie häufig und wieviel im allgemeinen Bingo gewinnen. (Bingo and Stochastics. How Many Players Win How Often and How Much in General Bingo)." *Mathematische Semesterberichte*, **59**(2), 155–181.

Lumely T (2007). "Tail Area of Sum of Chi-square Variables." Message posted to the R-help mailing list, archived at http://tolstoy.newcastle.edu.au/R/e2/help/07/03/13491.html.

MD*Tech (2007). *XploRe Version 4.8*. MD*Tech – Method and Data Technologies. URL http://www.xplore-stat.de/.

Monahan JF (2011). *Numerical Methods of Statistics*. 2nd edition. Cambridge University Press, Cambridge.

Patnaik PB (1949). "The Non-Central χ^2- and F-Distributions and Their Applications." *Biometrika*, **36**(1–2), 202–232.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Ruckdeschel P, Kohl M (2010). "Computation of the Finite Sample Risk of M-Estimators on Neighborhoods." *Technical report*, Bayreuth university. URL http://www.mathematik.uni-kl.de/~ruckdesc/pubs/howtoap.pdf.

Ruckdeschel P, Kohl M, Stabla T, Camphausen F (2006). "S4 Classes for Distributions." *R News*, **6**(2), 2–6. URL http://CRAN.R-project.org/doc/Rnews/.

Ruckdeschel P, Kohl M, Stabla T, Camphausen F (2013). *S4 Classes for Distributions – A Manual for Packages **distr**, **distrEx**, **distrEllipse**, **distrMod**, **distrSim**, **distrTEst**, and **distrTeach***. Version 2.5, URL http://CRAN.R-project.org/package=distrDoc.

Ruckdeschel P, Kohl M, Stabla T, Camphausen F (2014). ***distr**: S4 Classes for Distributions*. R package version 2.5.3, URL http://distr.R-Forge.R-project.org/.

SAS Institute Inc (2011). *The SAS System, Version 9.3*. SAS Institute Inc., Cary, NC. URL http://www.sas.com/.

Schaden G (2012). "Modelling the "Aoristic Drift of the Present Perfect" as Inflation An Essay in Historical Pragmatics." *International Review of Pragmatics*, **4**(2), 261–292.

Schaller P, Temnov G (2008). "Efficient and Precise Computation of Convolutions: Applying FFT to Heavy Tailed Distributions." *Computational Methods in Applied Mathematics*, **8**(2), 187–200.

Singh R (2010). *A Comparison of the Methods Used to Determine the Portfolio Credit Loss Distribution and the Pricing of Synthetic CDO Tranches*. M.Sc. thesis, University of the Witwatersrand, Faculty of Science, Witwatersrand, South Africa. URL http://wiredspace.wits.ac.za/bitstream/handle/10539/9273/Renay%20Singh%20MSc.pdf?sequence=1.

Stroustrup B (1987). *Die C++ Programmiersprache. (The C++ Programming Language).* Internationale Computer-Bibliothek. Addison-Wesley, Bonn. German translation.

Temnov G, Warnung R (2008). "A Comparison of Loss Aggregation Methods for Operational Risk." *Journal of Operational Risk*, **3**(1), 3–23.

The MathWorks, Inc (2011). *MATLAB – The Language of Technical Computing, Version R2011b.* The MathWorks, Inc., Natick, Massachusetts. URL http://www.mathworks.com/products/matlab/.

Warnes G (2002). "**HYDRA**: A Java Library for Markov Chain Monte Carlo." *Journal of Statistical Software*, **7**(4), 1–32. URL http://www.jstatsoft.org/v07/i04/.

Affiliation:

Peter Ruckdeschel
Fraunhofer ITWM Kaiserslautern
Department of Financial Mathematics
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany
E-mail: Peter.Ruckdeschel@itwm.fraunhofer.de

Matthias Kohl
Department of Medical and Life Sciences Furtwangen University
Jakob-Kienzle-Str. 17
78054 Villingen-Schwenningen, Germany
E-mail: Matthias.Kohl@stamats.de

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 59, Issue 4 *Submitted:* 2012-10-02
August 2014 *Accepted:* 2014-04-15