

General Purpose Convolution Algorithm for Distributions in S4-Classes by means of FFT

Matthias Kohl*
Peter Ruckdeschel†
Thomas Stabla‡
Mathematisches Institut
Universität Bayreuth
D-95440 Bayreuth
Germany

e-Mail: `matthias.kohl@uni-bayreuth.de`

3rd March 2005

Abstract

In this paper we describe a realization of a general-purpose FFT-algorithm for the convolution of arbitrary distributions of real-valued random variables within an object orientated setup.

0 Motivation

Using Fast Fourier Transformation (FFT) for convolutions of distributions has been common practice ever since the appearance of Cooley and Tukey (1965).

In the framework of object orientated programming (OOP) this technique becomes even more attractive for this problem: The user may be given one single function/binary operand for the task of convolution, which applies to arbitrary distributions. In case nothing else is known, a well-designed “general-purpose” FFT-algorithm should automatically be used, while in special cases where more information or even an exact formula is available —like in case of normal or Poisson variables, a dispatching mechanism may realize this and replace the general algorithm by a particular (possibly exact) one.

We present this approach within the R-project, confer R Development Core Team (2005), where we have made available a package `distr` on CRAN¹, which implements distributions as classes within the OOP-concept of S. To use a simple notation, we

*..
†..
‡..

¹<http://cran.r-project.org/mirrors.html>

define the operator “+” for univariate distributions, which is then indeed specialized/overloaded for particular operands, e.g. normal distributions.

As default procedure, before applying the FFT-algorithm, we first discretize the distributions to lattice form, for which Discrete Fourier Transformation is available. We thus need no assumptions like Lebesgue densities for the involved distributions.

Our paper is organized as follows: We first document the need for OOP for (probability-)distributions. In section 2 we review the `S4-class` concept, by which OOP is realized in `S/R`, because, among others, for the purpose of convolution, we deviate from the intended OOP-style in our classes for distributions. We then sketch our implementation of distribution classes and briefly indicate how we generate a new object of a distribution class as a result of convolution. In section 3, we present the general purpose FFT-Algorithm and some ramifications. Some forerunners in that direction are discussed in section 4. In section 5 we discuss the exactitude of our algorithm.

1 Why 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` and its Open Source implementation `R`, but of course the considerations also apply for other extendable packages like `XploRe`, `Gauss`, `Simula`, `SAS` or even `MATLAB`. More or less all these packages contain techniques for most useful distributions, like normal, exponential, uniform, Poisson just to name a few.

There are limitations, however: You can only use distributions which are already implemented in the package or in some add-on library or for which you yourself have provided an implementation. In many natural settings you want to formulate algorithms once for all distributions, so you should be able to treat the actual distribution, say `D`, as argument to the function. This is an application particularly well-suited for OOP, as described in Booch (1995), with its paradigms ”inheritance” and ”method overloading”:

In the OOP concept, we would use a generic function for our algorithm and let the dispatching mechanism decide what to do at run-time. In particular, the result of such an algorithm may be a new distribution generated in this algorithm, as in the case when you consider the convolution of two distributions. To this end one may overload the operator “+” for distributions so that $X+Y$ for (univariate) distribution objects `X` and `Y` realizing independent random variables X and Y , will be a new random variable `Z`, with $\mathcal{L}(Z) = \mathcal{L}(X + Y)$.

In his package `HYDRA` for MCMC-simulation, Warnes (2002) pursuits a similar OOP approach, but in the `JAVA` setting. Under <http://statdistlib.sourceforge.net/>, the author provides a set of `JAVA` classes representing common statistical distributions, porting the `C`-code underlying the `R` implementation. But, quoting the author himself from his 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 OOP in S

2.1 The S4-class concept

In R, OOP is realized in the S3-class-concept as introduced in Chambers (1993a), (1993b), and by its successor, the S4-class-concept, as developed in Chambers (1998), (1999). We work with the S4-class-concept.

Using the terminology of Bengtsson (2003), this concept is intended to be *FOOP* (function-object-orientated programming) style, in contrast to *COOP* (class-object-orientated 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 part of the object, but belong to *generic functions* —abstract functions which allow 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 the arguments. Next to this concept in C++ are “overloaded functions” in the sense of Stroustrup (1987), section 4.6.6.

FOOP style is a good idea for functions like “+” having a natural meaning for many operand types/classes —e.g. distributions, where it should mean convolution. This style also has got advantages for distributed programming, as not every programmer providing functionality for some class has to interfere into the original class definition. Also, as S is interpreted, a method hanging in the class definition would not simply be a pointer but rather the whole function definition. So the COOP-style paradigm to declare methods as particular members of a class should even be avoided where possible.

Nevertheless, the S4-class-concept also allows COOP style, that is, members (or *slots* in S4-lingo) are permitted to be functions. This has been extensively used in Bengtsson’s (2003) R.oo package.

For our distribution classes, we, too, use the possibility for function-type members, albeit only in a very limited way.

2.2 Implementation of distribution classes within the S4-class concept

In S, any distribution is given through the four functions **r**, providing generation of pseudo-random numbers according to that distribution, **d**, the density or probability function/counting density, **p**, the cumulative distribution function (c.d.f.), and **q**, the quantile function. This is reflected in the naming convention `[prefix]<name>` where `[prefix]` stands for **r**, **d**, **p**, or **q** and `<name>` is the name of the distribution. We call these functions *constitutive*, because they should be regarded as integral part of a distribution object, and hence should be realized as members of a distribution —even though this may cause some “code weight” for these objects.

A justification for this approach may be seen in convolution: Assume we would like to automatically generate these functions/methods for the law of expressions like $X+Y$ for objects **X** and **Y** of class `UnivariateDistribution`. The constitutive functions vary from distribution to distribution, and the dispatching mechanism makes

its decision which method to use for a generic function “cdf” based the signature of a function. So we would need a new class for every new distribution, and for each distribution a new method “cdf”; i.e., very soon the dispatching mechanism would have to decide between lots of different signatures. Instead, when the c.d.f. is a member of a class, dispatching is not necessary.

This idea is extended for other mathematical operations in Ruckdeschel et al. (2005a), which is the documentation to our package “distr” available on CRAN; also, a short digest of the capabilities of this package is available in Ruckdeschel et al. (2005b).

2.3 Convolution as a particular method in “distr”

Convolution itself ideally fits in the FOOP-setup; method dispatching among various particular convolution methods goes as follows:

Either there are better algorithms or even exact convolution formulae for the given signature, as is the case for independent variables distributed according to $\text{Bin}(n_i, p)$, $i = 1, 2$, or $\text{Poisson}(\lambda_i)$, or $\mathcal{N}(\mu_i, \sigma_i^2)$ etc. Then the dispatching mechanism for S4-classes will hopefully realize e.g. that the signature is `Norm`, `Norm` and will use the best matching existing “+”-method, which in this case consists in generating a new object of class `Norm` with corresponding moments. This case will be exceptional, so that we do not have to dispatch among too many methods.

Or, by default, the convolution method generates a new object of either class `AbscontDistribution` or `DiscreteDistribution`, depending on whether the distribution of at least one summand is absolutely continuous or not.

The slots `p` and `d` of this new object are then filled by Algorithm 3.4, 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.

[was fuer diskrete???

The c.d.f.s F_1 and F_2 used in Algorithm 3.4 will be obtained from slot `p`. [WAS MACHEN WIR GENAU FUER DIE DICHTE f bzw $f_1 * f_2$????]

3 General purpose FFT algorithm

The main idea of the algorithm is to use Discrete Fourier Transformation (DFT), which may be calculated very fast using Fast Fourier Transformation (FFT). So, before coming back to the convolution of c.d.f.’s/densities in subsection 3.2, we start

with briefly introducing DFT and its convolution property stated in Theorem 3.2. In the presentation, we follow Lesson 8 of Gasquet and Witomski (1999).

3.1 Discrete Fourier Transformation

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 Discrete Fourier Transformation (DFT) of order m is,

$$\text{DFT}_m: \mathbb{C}^m \rightarrow \mathbb{C}^m, (x_0, x_1, \dots, x_{m-1}) \mapsto (\hat{x}_0, \hat{x}_1, \dots, \hat{x}_{m-1}) \quad (3.1)$$

where

$$\hat{x}_n = \frac{1}{m} \sum_{j=0}^{m-1} x_j \omega_m^{jn} \quad \omega_m = e^{-2\pi i/m}, i = \sqrt{-1} \quad (3.2)$$

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} ($j, k = 0, 1, \dots, m-1$) and inverse $\Omega_m^{-1} = 1/m \bar{\Omega}_m$ ($\bar{\Omega}_m$ the conjugate DFT_m); i.e., DFT_m is linear and bijective.

Remark 3.1 (a) Computing $\hat{x}_0, \hat{x}_1, \dots, \hat{x}_{m-1}$ directly from equation (3.2), requires $(m-1)^2$ complex multiplications and $m(m-1)$ complex additions. But, Fast Fourier Transformation (FFT), as introduced by Cooley and Tukey (1965), is of just order $m \log m$. It works best for the case $m = 2^p$ ($p \in \mathbb{N}$); confer 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; confer Table 9.1 (ibid.).

(b) If $(x_n)_{n \in \mathbb{Z}}$ is a sequence of real numbers, it is possible to reduce the cost of computation by half; confer Section 8.3 of Gasquet and Witomski (1999).

(c) FFT is available already in R as function `fft`; confer R Development Core Team (2005). ////

For DFTs we have the following convolution theorem:

Theorem 3.2 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,

$$x * y = \left(\sum_{j=0}^{m-1} x_j y_{n-j} \right)_{n \in \mathbb{Z}} \quad (3.3)$$

and it holds,

$$\hat{z} = m \hat{x} \hat{y} \quad \text{with} \quad z = x * y \quad (3.4)$$

where $\hat{x} \hat{y} = (\hat{x}_n \hat{y}_n)_{n \in \mathbb{Z}}$.

PROOF :[REFERENZ????] ////

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

Proposition 3.3 *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,*

$$\widehat{*_{i=1}^N x} = m^{N-1} \hat{x}^N \quad N \in \mathbb{N} \quad (3.5)$$

PROOF :Immediately follows from Theorem 3.2 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

$$F_i(x) = \sum_{j=0}^{m-1} p_{i,j} \mathbf{I}_{[j,\infty)}(x) \quad i = 1, 2 \quad (3.6)$$

with

$$p_{i,j} \geq 0 \quad j = 0, 1, \dots, m-1 \quad \sum_{j=0}^{m-1} p_{i,j} = 1 \quad (3.7)$$

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

$$p_{i,j} = 0 \quad i = 1, 2 \quad j = m, \dots, 2m-1 \quad (\text{zero padding}) \quad (3.8)$$

and

$$p_{i,k+2m} = p_{i,k} \quad \forall k \in \mathbb{Z} \quad (3.9)$$

Then, the convolution F of F_1 and F_2 is an integer lattice distribution given by

$$F(x) = (F_1 * F_2)(x) = \sum_{j=0}^{m-1} \sum_{k=0}^{m-1} p_{1,j} p_{2,k} \mathbf{I}_{[k,\infty)}(x-j) \quad (3.10)$$

$$= \sum_{j=0}^{2m-1} \pi_j \mathbf{I}_{[j,\infty)}(x) \quad \text{with} \quad \pi_j := \sum_{k=0}^{2m-1} p_{1,k} p_{2,j-k} \quad (3.11)$$

where in particular $\pi_{2m-1} = 0$. Hence, in view of Theorem 3.2, $\pi = (\pi_n)_{n \in \mathbb{Z}} = p_1 * p_2$ and we can compute π using FFT and its inverse. This result forms the basis of Algorithm 3.4.

[SIMILARLY (ausformulieren) : for its densities w.r.t. counting measure...]

As it stands, Algorithm 3.4 will be presented for the case of absolutely continuous distributions, but with slight and obvious modifications this algorithm works for quite general distributions, thus justifying the title of this paper; also confer Subsection 3.3.

Algorithm 3.4

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,

$$F_i((-\infty, A_i)) = \frac{\varepsilon}{2} \quad \text{and} \quad F_i((B_i, \infty)) = \frac{\varepsilon}{2} \quad (3.12)$$

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

$$G_i(x) := \sum_{j=0}^{m-1} p_{i,j} \mathbf{I}_{[A+(j+0.5)h, \infty)}(x) \quad h = \frac{B-A}{m} \quad (3.13)$$

where

$$p_{i,j} = F_i([A + jh, A + (j+1)h]) \quad (3.14)$$

for $j = 0, 1, \dots, m-1$.

Step 3: (Transformation to an integer grid)

Based on G_i ($i = 1, 2$), we define the integer lattice distributions

$$\tilde{G}_i(x) := \sum_{j=0}^{m-1} p_{i,j} \mathbf{I}_{[j, \infty)}(x) \quad i = 1, 2 \quad (3.15)$$

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

$$p_{i,j} = 0 \quad i = 1, 2 \quad j = m, \dots, 2m-1 \quad (\text{zero padding}) \quad (3.16)$$

and

$$p_{i,k+2m} = p_{i,k} \quad \forall k \in \mathbb{Z} \quad (3.17)$$

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 (3.11); i.e.,

$$\tilde{G}(x) = \sum_{j=0}^{2m-1} \pi_j \mathbf{I}_{[j, \infty)}(x) \quad \pi_j := \sum_{k=0}^{2m-1} p_{1,k} p_{2,j-k} \quad (3.18)$$

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,

$$G(x) = \sum_{j=0}^{2m-2} \pi_j \mathbb{I}_{[2A+(j+1.5)h, \infty)}(x) \quad (3.19)$$

That is, we additionally use a continuity correction of $h/2$, which clearly improves the results; confer Remark 5.6 (a).

Step 6: (Smoothing)

Next, we use interpolation of the values of G on $\{2A, 2A + 1.5h, \dots, 2B - 0.5h, 2B\}$ 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 $\{0, \pi_0, \pi_1, \dots, \pi_{2m-2}, 0\}$ by h and interpolating these values on the grid $\{2A, 2A + h, \dots, 2B - h, 2B\}$ (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 $F^{\natural}([2A, 2B])$ and $\int f^{\natural}(x) dx$, respectively, where $\int f^{\natural}(x) dx$ may be calculated numerically exact, since f^{\natural} is a piecewise linear function.

3.3 Ramifications and extensions of this algorithm

Algorithm 3.4 for lattice distributions: Obviously, Algorithm 3.4 may easily be applied to lattice distributions F_1, F_2 on \mathbb{R} which are defined on the same grid. In this case the algorithm can essentially be reduced to steps 1-5 and 7. Moreover, the results are numerically exact if the lattice distributions have finite support; confer 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 of Algorithm 3.4 is, to some extent, controllable and in many cases negligible.

Availability: This algorithm is implemented in the R package `distr` which was developed in joint work of the present authors with F. Camphausen; confer Ruckdeschel et al. (2005b). For the sources also see [wo??] on

<http://www.uni-bayreuth.de/departments/math/org/mathe7/DISTR/>

Richardson Extrapolation: A technique to enhance the exactitude of Algorithm 3.4 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 `+`; confer also Remark 4.1.

Exponential Tilting: To better cope with the aliasing error, which occurs as

a wrap-around effect due to summation modulo m (cf. equation (3.3)), especially for heavy-tailed distributions, Algorithm 3.4 can largely be improved by a suitable change of measure (exponential tilting) — at the cost of additional smoothness conditions for the densities. So one might conceive a further subclass `HeavyTailedSmoothDistribution` and overload "+" for objects of these classes using exponential tilting. Also confer Remark 4.1.

Examples: There are some instructive examples, like the computation of an approximation to the stationary regressor distribution of an AR(1) process, together with the corresponding R sources included in Ruckdeschel et al. (2005b).

Modification for M-Estimators: In view of Proposition 3.3, Algorithm 3.4 may easily be modified to compute an approximation of the exact finite-sample distribution of M estimates. As the underlying distributions are not continuous, special care has to be taken; under these precautions, one obtains very accurate results; confer Ruckdeschel and Kohl (2004). In the cited reference, we compare the results obtainable with this modified algorithm with 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

Remark 4.1 (a) 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 nice mathematical formulation of the corresponding algorithm (cf. Algorithm 1) is included in Grübel and Hermesmeier (1999). However, the main purpose of their article is the investigation of the aliasing error. In case of a claim size distribution of lattice type they obtain a simple general bound for this error and show that it can be eliminated by exponential tilting. But, even without the smoothness assumptions needed for exponential tilting, the aliasing error can also be made very small if we choose ε in step 1 of Algorithm 3.4 small enough and q in step 2 large enough. Thus, in many cases this effect is negligible.

Moreover, if one considers absolutely continuous probability distributions, a initial discretization step is necessary; confer also step 2 of Algorithm 3.4. The corresponding error is studied in Grübel and Hermesmeier (2000) and it is shown that this error, under certain smoothness conditions, can be reduced by an extrapolation technique (Richardson extrapolation).

(b) In Embrechts et al. (1993) the authors describe how one can use FFT to determine various quantities of interest in risk theory and insurance mathematics including the computation of the total claim size distribution, the mean and the variance of the process and the probability of ruin. Moreover, using FFT it is also possible to find the stationary waiting time distribution for a given customer interarrival time distribution and a given service time distribution in the G/G/1

queueing model; confer Grübel (1991). ////

4.2 Algorithms based on Fourier inversion

5 Exactitude of our algorithm

To assess the exactitude of our algorithm, we present checks for n -fold convolution products. We determine the precision of this convolution algorithm in terms of the total variation distance of the densities,

$$d_v(P, Q) = \frac{1}{2} \int |p - q| d\mu = \sup_{B \in \mathbb{B}} |P(B) - Q(B)| \quad (5.20)$$

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 cumulative distribution functions,

$$d_\kappa(P, Q) = \sup_{t \in \mathbb{R}} |P((-\infty, t]) - Q((-\infty, t])| \quad (5.21)$$

Obviously, $d_\kappa \leq d_v$ as the supremum in case of the total variation distance is taken over more sets. In the sequel d_v^\sharp and d_κ^\sharp stand for the numerical approximations of d_v and d_κ .

The first example treats Binomial distributions and shows that the convolution algorithm is numerically exact for integer lattice distributions with finite support. In particular, this implies that the computation of the FFT and its inverse is numerically exact.

Example 5.1 Assume $F = \text{Bin}(k, p)$ with $k \in \mathbb{N}$ and $p \in (0, 1)$. Then, the n -fold convolution product is $F^{*n} = \text{Bin}(nk, p)$ ($n \in \mathbb{N}$). Let f_n and f^\sharp be the probability functions of F^{*n} and F^\sharp , respectively. Then, we may determine d_v^\sharp and d_κ^\sharp numerically exact by,

$$d_v^\sharp(F, F^\sharp) = \frac{1}{2} \sum_{j=0}^{nk} |f_n(j) - f^\sharp(j)| \quad (5.22)$$

and

$$d_\kappa^\sharp(F, F^\sharp) = \max_{j \in \{0, \dots, nk\}} |F^{*n}([0, j]) - F^\sharp([0, j])| \quad (5.23)$$

We obtain the results contained in Table 1 which show that Algorithm 3.4 is numerically exact in case of binomial distributions, where the values of k and p are chosen arbitrarily. The computation time for the convolution at $n = 1000$, $k = 50$ and $p = 0.4$ on an Athlon with 2.5 GHz and 512 MB RAM is about 0.2 seconds. ////

In case of the Poisson distribution the results of the convolution algorithm turn out to be very accurate, too.

n	k	p	d_v^\sharp	d_κ^\sharp
2	10	0.5	1.9e-16	1.1e-16
5	20	0.7	5.9e-16	3.9e-16
10	30	0.8	1.4e-15	1.1e-15
100	15	0.2	8.5e-15	8.3e-15
1000	50	0.4	7.0e-14	6.6e-14

Table 1: The Precision of the convolution of binomial distributions via FFT; confer Example 5.1.

Example 5.2 We consider $F = \text{Pois}(\lambda)$ with $\lambda \in (0, \infty)$ where $F^{*n} = \text{Pois}(n\lambda)$ ($n \in \mathbb{N}$). Since the support of F is \mathbb{N}_0 , we use $A = 0$ and $B = F^{-1}(1 - 1e-15)$ in step 1 of Algorithm 3.4 and determine d_v^\sharp and d_κ^\sharp numerically exact by,

$$d_v^\sharp(F, F^\sharp) = \frac{1}{2} \sum_{j=0}^M |f_n(j) - f^\sharp(j)| \tag{5.24}$$

and

$$d_\kappa^\sharp(F, F^\sharp) = \max_{j \in \{0, \dots, M\}} |F^{*n}([0, j]) - F^\sharp([0, j])| \tag{5.25}$$

where M is the $1 - 3e-16$ quantile of F^{*n} . For further details on the computation of d_v^\sharp and d_κ^\sharp we refer to Remark 5.3. We obtain the results contained in Table 2 which demonstrate the high precision of the convolution algorithm in case of Poisson distributions where the parameter λ is chosen arbitrarily. The computation time for the convolution at $n = 1000$ and $\lambda = 50$ on an Athlon with 2.5 GHz and 512 MB RAM is about 0.6 seconds. ///

n	λ	d_v^\sharp	d_κ^\sharp
2	0.1	2.9e-16	3.3e-16
5	10.0	2.8e-15	2.9e-15
10	7.5	4.9e-15	4.8e-15
100	15.0	1.9e-14	1.4e-14
1000	50.0	3.4e-13	3.3e-13

Table 2: The Precision of the convolution of Poisson distributions via FFT; confer Example 5.2.

Remark 5.3 (a) The differences between d_κ^\sharp and d_v^\sharp are below the computational accuracy in case of $n = 2, 3, 5, 10$ which is about $2e-16$. That is, the values of d_κ^\sharp and d_v^\sharp are numerically not distinguishable.

(b) By checking the output of the convolution algorithm, we found that the cumulative distribution function of the Poisson distribution `ppois` included in R

becomes less exact for bigger λ ; confer also the **R-help** mail archive of January 2004 (<http://maths.newcastle.edu.au/~rking/R/help/04/01/0513.html>), respectively of March 2004 (<http://maths.newcastle.edu.au/~rking/R/help/04/03/0036.html>). Thus, we directly sum over the numerically exact values of the probability function `dpois` in order to obtain the numerically exact Kolmogorov distances given in Table 2. `////`

In the next two examples we consider the convolution of absolutely continuous distributions. We determine the total variation distance $d_v^h(F, F^h)$ by numerical integration using the **R** function `integrate`. To compute an approximation of the Kolmogorov distance, we evaluate $d_\kappa^h(F, F^h)$ on a random grid which we obtain by sampling 1e06 pseudo-random numbers of `Unif` (`Min`, `Max`) where `Min` and `Max` are the $\varepsilon/10$ and $1 - \varepsilon/10$ quantile of F^{*n} . We first present the results for normal distributions.

Example 5.4 Assume $F = \mathcal{N}(\mu, \sigma^2)$ with $\mu \in \mathbb{R}$ and $\sigma \in (0, \infty)$. Then it holds, $F^{*n} = \mathcal{N}(n\mu, n\sigma^2)$ ($n \in \mathbb{N}$). Starting with $\mathcal{N}(0, 1)$ and A and B as defined in step 1 of Algorithm 3.4 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 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; confer Table 3. 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 4 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 . `////`

n	ε	q	μ	σ	d_v^h	d_κ^h
2	1e-08	12	-10.0	100.0	3.2e-07	1.4e-07
			-2.0	5.0	3.2e-07	1.4e-07
			0.0	1.0	3.2e-07	1.4e-07
			1.0	50.0	3.2e-07	1.4e-07
			100.0	1000.0	3.2e-07	1.4e-07

Table 3: The Precision of the convolution of normal distributions via FFT is independent of the parameters μ and σ ; confer Example 5.4.

Our last example treats the convolution of exponential distributions which leads to gamma distributions.

Example 5.5 We consider $F = \text{Exp}(\lambda) = \Gamma(1, \lambda)$ with $\lambda \in (0, \infty)$. Then it holds, $F^{*n} = \Gamma(n, \lambda)$ ($n \in \mathbb{N}$). Analogously to the normal case (cf. Example 5.4), the grid transforms the same as the exponential distributions do; i.e. $\tilde{A} = 1/\lambda A$ and $\tilde{B} = 1/\lambda B$. Thus, we expect the precision of the results to be independent of λ . This is again confirmed by our numerical computations; confer Table 5. Next we study the dependence of the accuracy of Algorithm 3.4 on $n \in \mathbb{N}$, $\varepsilon > 0$ and

n	ε	q	d_v^{\sharp}	d_{κ}^{\sharp}
2	1e-06	8	5.6e-05	2.6e-05
		10	4.5e-06	1.7e-06
		12	2.1e-06	1.9e-06
	1e-08	10	4.8e-06	2.4e-06
		12	3.2e-07	1.4e-07
		14	3.5e-08	1.9e-08
	1e-10	12	4.0e-07	1.9e-07
		14	2.5e-08	1.2e-08
		18	2.9e-10	2.0e-10
5	1e-08	12	1.9e-07	7.5e-08
		14	5.0e-08	3.5e-08
10	1e-08	12	1.4e-07	4.3e-08
		14	1.0e-07	4.8e-08
50	1e-08	12	5.0e-07	8.2e-09
		14	5.0e-07	6.7e-08

Table 4: The Precision of the convolution of normal distributions via FFT; confer Example 5.4.

$q \in \mathbb{N}$ where we may choose $\lambda = 1.0$. As in Example 5.4 the precision is almost independent of n . It mainly depends on q where the maximum accuracy, we can reach, is of order ε ; confer Table 6. ////

n	ε	q	λ	d_v^{\sharp}	d_{κ}^{\sharp}
2	1e-08	12	0.01	1.3e-06	2.5e-06
			0.5	1.3e-06	2.5e-06
			1.0	1.3e-06	2.5e-06
			5.0	1.3e-06	2.5e-06
			10.0	1.3e-06	2.5e-06

Table 5: The Precision of the convolution of exponential distributions via FFT is independent of the parameter λ ; confer Example 5.5.

Remark 5.6 (a) Without continuity correction (step 6) the Kolmogorov distances would clearly increase. For $n = 2$, $\varepsilon = 1e-08$ and $q = 12$ we obtain in case of normal distributions $d_{\kappa}^{\sharp} = 3.9e-04$ instead of $1.4e-07$ and in case of exponential distributions $d_{\kappa}^{\sharp} = 8.3e-04$ instead of $2.5e-06$.

(b) In case of normal and exponential distributions the computation time for $n = 2$, $\varepsilon = 1e-10$ and $q = 18$ on an Athlon with 2.5 GHz and 512 MB RAM is about 30 seconds, for $n = 50$, $\varepsilon = 1e-08$ and $q = 14$, it is about 14 seconds and for $n = 2$, $\varepsilon = 1e-08$ and $q = 12$ it is about 0.4 seconds.

n	ε	q	d_v^\sharp	d_κ^\sharp
2	1e-06	8	1.9e-04	3.6e-04
		10	1.3e-05	2.2e-05
		12	1.8e-06	2.0e-06
	1e-08	10	2.1e-05	4.0e-05
		12	1.3e-06	2.5e-06
		14	9.0e-08	1.6e-07
	1e-10	12	2.1e-06	3.9e-06
		14	1.3e-07	2.5e-07
		18	6.0e-10	9.6e-10
5	1e-08	12	1.7e-06	1.7e-06
		14	1.3e-07	9.7e-08
10	1e-08	12	2.3e-06	2.2e-06
		14	1.9e-07	1.1e-07
50	1e-08	12	4.6e-06	4.6e-06
		14	4.0e-07	3.2e-07

Table 6: The Precision of the convolution of exponential distributions via FFT; confer Example 5.5.

(c) The Example 5.5 reveals one minor flaw of Algorithm 3.4. The support of $\Gamma(n, \lambda)$ is $[0, \infty)$ whereas the convolution algorithm is only very exact in $[2A + (n/2 + 0.5)h, \dots, 2B - (n/2 + 0.5)h]$. That is, for small n ($n \leq 5$) the Kolmogorov distance is $F([0, 2A + (n/2 + 0.5)h]) - F^\sharp([0, 2A + (n/2 + 0.5)h])$. However, for bigger n this inexactness 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^\sharp for small n and leads to $d_\kappa^\sharp > d_v^\sharp$. ////

6 Acknowledgement

We thank Prof. Rieder for.... Grübel for

References

- Bengtsson H. (2003): The R.oo package - object-oriented programming with references using standard R code. In: Hornik K., Leisch F. and Zeileis A. (Eds.) *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*. Vienna, Austria. Published as <http://www.ci.tuwien.ac.at/Conferences/DSC-2003/>.
- Bertram J. (1981): Numerische Berechnung von Gesamtschadenverteilungen. *Bl., Dtsch. Ges. Versicherungsmath.*, **15**: 175–194.

- Booch G. (1995): *Objektorientierte Analyse und Design. (Object oriented analysis and design)*. Addison-Wesley, 1., corrected reprint edition. German Translation.
- Bühlmann H. (1984): Numerical evaluation of the compound Poisson distribution: Recursion or fast Fourier transform? *Scand. Actuar. J.*, **1984**: 116–126.
- Chambers J.M. (1993a): Classes and methods in S. I: Recent developments. *Comput. Stat.*, **8**(3): 167–184. <http://cm.bell-labs.com/stat/doc/93.26.ps>.
- (1993b): Classes and methods in S. II: Future directions. *Comput. Stat.*, **8**(3): 185–196. <http://cm.bell-labs.com/stat/doc/93.27.ps>.
- (1998): *Programming with data. A guide to the S language*. Springer. <http://cm.bell-labs.com/stat/Sbook/index.html>.
- (1999): Computing with Data: Concepts and Challenges. *The American Statistician*, **53**(1): 73–84. <http://cm.bell-labs.com/stat/doc/Neyman98.ps>.
- Cooley J. and Tukey J.W. (1965): An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, **19**: 297–301.
- Embrechts P., Grübel R. and Pitts S. (1993): Some applications of the fast Fourier transform algorithm in insurance mathematics. *Stat. Neerl.*, **47**(1): 59–75.
- Feilmeier M. and Bertram J. (1987): *Anwendung numerischer Methoden in der Risikotheorie. (Application of numerical methods in risk theory)*, Vol. 16 of *Schriftenreihe Angewandte Versicherungsmathematik*. Deutsche Gesellschaft für Versicherungsmathematik. Verlag Versicherungswirtschaft e.V., Karlsruhe.
- Gasquet C. and Witomski P. (1999): *Fourier analysis and applications. Filtering, numerical computation, wavelets. Transl. from the French by R. Ryan.*, Vol. 30 of *Texts in Applied Mathematics*. Springer.
- Grübel R. (1991): Algorithm AS 265: G/G/1 via fast Fourier transform. *Applied Statistics*, **40**(2): 355–365.
- Grübel R. and Hermesmeier R. (1999): Computation of compound distributions. I. Aliasing errors and exponential tilting. *Astin Bull.*, **29**(2): 197–214.
- (2000): Computation of compound distributions. II. Discretization errors and Richardson extrapolation. *Astin Bull.*, **30**(2): 309–331.
- R Development Core Team (2005): *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. <http://www.R-project.org>.
- Ruckdeschel P. and Kohl M. (2004): Computation of the Finite Sample Risk of M-Estimators on Neighborhoods. Technical report, Feb. 2005. [URL-reference soon]

Ruckdeschel P., Kohl M., Stabla T., and Camphausen F. (2005a): S4 Classes for Distributions —a manual for package "distr" version 1.4, Mar. 2005. <http://www.uni-bayreuth.de/departments/math/org/mathe7/DISTR/distr.pdf>.

— (2005b): S4 Classes for Distributions. Submitted. Also available in <http://www.uni-bayreuth.de/departments/math/org/mathe7/RUCKDESCHEL/pubs/distr-rnews.pdf>.

Stroustrup B. (1987): *Die C++ Programmiersprache. (The C++ programming language)*. Internationale Computer-Bibliothek. Addison-Wesley Verlag. German Translation.

Warnes G. (2002): *HYDRA A JAVA library for Markov Chain Monte Carlo*. <http://research.warnes.net/downloads/Hydra/UserGuide.pdf>.