Journal of Statistical Software

April 2014, Volume 57, Issue 10. http://www.jstatsoft.org/

PaCAL: A Python Package for Arithmetic Computations with Random Variables

Marcin Korzeń
West Pomeranian University
of Technology

Szymon Jaroszewicz
National Institute
of Telecommunications

Abstract

In this paper we present **PaCAL**, a Python package for arithmetical computations on random variables. The package is capable of performing the four arithmetic operations: addition, subtraction, multiplication and division, as well as computing many standard functions of random variables. Summary statistics, random number generation, plots, and histograms of the resulting distributions can easily be obtained and distribution parameter fitting is also available. The operations are performed numerically and their results interpolated allowing for arbitrary arithmetic operations on random variables following practically any probability distribution encountered in practice. The package is easy to use, as operations on random variables are performed just as they are on standard Python variables. Independence of random variables is, by default, assumed on each step but some computations on dependent random variables are also possible. We demonstrate on several examples that the results are very accurate, often close to machine precision. Practical applications include statistics, physical measurements or estimation of error distributions in scientific computations.

Keywords: arithmetic on random variables, algebra of random variables, probabilistic computation, Python.

1. Introduction

Arithmetic operations on random variables have many applications in statistics, error propagation or probabilistic inference. In this paper we present **PaCAL**, a Python (van Rossum *et al.* 2011) package which allows for computation with random variables as one does with ordinary variables in a computer program. The resulting densities are computed numerically and approximated using Chebyshev interpolation, allowing for computations involving a very wide class of distributions which are allowed to have infinite supports and singularities in

their density functions. Thanks to the very high numerical stability of Chebyshev methods, excellent accuracy is typically achieved, comparable with that of double precision arithmetic. The package focuses mainly on independent random variables, however some functionality for computations on dependent variables is also available.

We have tested the package on hundreds of examples, including most examples and exercises from Springer (1979), as well as numerous relationships between distributions listed in Wikipedia and statistical textbooks, achieving, in almost all cases, excellent accuracy.

In this section we discuss the related work and give a high level overview of the package's structure and capabilities, as well as installation and basic usage instructions. Section 2 gives a detailed overview of the package with numerous examples, Section 3 presents the facilities available for computing with dependent random variables, Section 4 presents selected applications, and finally, Section 5 concludes and discusses future directions.

1.1. Basic usage

Let us start with a simple example corresponding to measuring an angle A by measuring the opposite Y and adjacent X whose values are not known exactly but follow uniform distributions on $[1, 2]$ and $[3, 4]$ respectively. The following code will produce the probability density function of A:

```
>>> from pacal import *
>>> Y = UniformDistr(1, 2)
>>> X = UniformDistr(3, 4)
>>> A = atan(Y / X)
>>> A.plot()
>>> show()
>>> print A.mean()
>>> print A.interval(0.95)
```

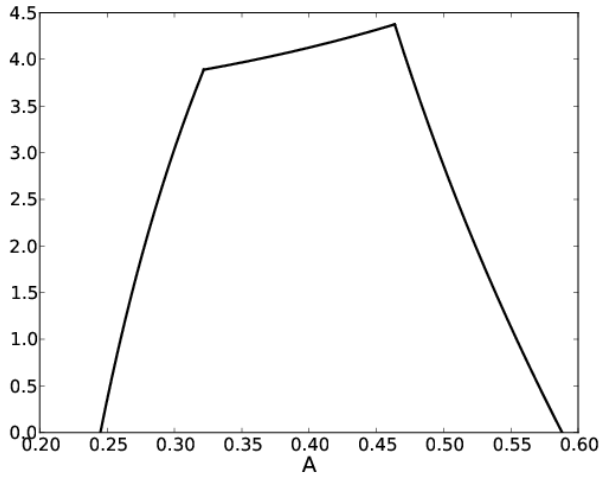

Figure 1: Indirect measurement of an angle: the probability density function of $A = \arctan(Y/X)$, where $Y \sim U(1, 2)$, $X \sim U(3, 4)$.

```
0.404871991791
(0.27259785174824225, 0.5466628704901051)
```

The resulting plot is shown in Figure 1. The density is clearly not normal and asymmetric. Of course `A` can be used in further computations.

In the rest of the paper we omit the `import` statements and commands related to subplots, legends, etc.

1.2. Related approaches

Related approaches fall into three general groups: symbolic, based on closed families of distributions, and sampling based.

The symbolic approach relies on the capabilities of computer algebra systems (CAS) to explicitly solve the integrals involved in performing operations on random variables. Such systems have the advantage of yielding exact results whenever the solver is capable of computing all the necessary integrals. In practice however, the integrals do not usually have closed form representations, and numerical fallbacks are necessary. Such fallbacks, if provided by the CAS, are typically very inefficient and not suitable for practical use. A prominent example of such a system is the **APPL** package (Glen, Evans, and Leemis 2001) based on Maple (Maplesoft 2008). Basic symbolic functionality for probabilistic arithmetic is also available in Mathematica (Wolfram 2003).

Another class of approaches is based on closed families of distributions. The most important of such approaches is described in Springer (1979) and is based on so called H-functions, which are a generalization of hypergeometric functions. The family of independent random variables whose densities are H-functions is closed under multiplication and division but not under addition and subtraction. The approach presented in Springer (1979) is primarily analytical, but numerical procedures based on moment estimation are also discussed. Lack of closure under addition and reliance on moments limit the usefulness of the approach. Another approach, closer in spirit to **PaCAL**, is presented in Williamson (1989) and Williamson and Downs (1990). The method is based on Laguerre polynomials and relies on the closure of such polynomials under Fourier and Mellin convolutions. The approach does not allow for division of random variables and does not allow for densities with discontinuities or singularities.

An approach based on polynomial interpolation (used also by **PaCAL**) has been demonstrated for a very general case of dependent random variables in Shenoy and West (2011). However, that approach uses symbolic integration by Mathematica and relies on closeness of polynomials under integration. As a result, multiplication and division are not possible except for special cases. Moreover, no convenient software package is provided.

The last group of methods are purely statistical approaches based on Monte Carlo (MC) sampling. The MC approach currently offers the greatest flexibility allowing for an arbitrary number of dependent random variables, Bayesian inference, etc. Examples of such systems are **BUGS** (Lunn, Thomas, Best, and Spiegelhalter 2000; Spiegelhalter, Thomas, Best, and Lunn 2003) and **PyMC** (Patil, Huard, and Fonnesbeck 2010). The MC approach is also suggested as the method of choice in the official guide on assessing uncertainty in measurements (JCGM 2009). The main weakness of the MC approach is slow convergence; the accuracy grows with square root of the sample size, thus to obtain one more correct decimal place of the result a 100 times larger sample is needed. Typically, sampling based methods only provide 3–4

decimal places of accuracy. The situation gets much worse in the tails of the distributions where few samples are available.

PaCAL can be viewed as a variant of the method based on closed representations, however, the class we use (that of approximable densities) is extremely broad and allows for heavy tails, singularities in densities, and the use of arbitrary arithmetic operations. Our earlier paper (Jaroszewicz and Korzeń 2012) explores the theoretical aspects of the approach, describes the closed family of distributions, and discusses details of interpolation and integration procedures, while this paper concentrates on a detailed description of the package itself. Several new examples are presented, as well as functionality not covered in the earlier paper, such as order statistics or optimized operations on independent identically distributed (i.i.d.) sequences. All parts concerning discrete distributions and dependent random variables are also new.

Finally, we need to mention the **Chebfun** package (Battels and Trefethen 2004; Trefethen and others 2011) which uses the same methodology to implement computations on arbitrary functions, and was the main inspiration for us. It is however not tailored towards probabilistic computation (in fact the only function useful in the probabilistic context is convolution). We focus solely on probabilistic arithmetic, which allows us to obtain better accuracy for a wider range of probabilistic operations.

1.3. Overview of PaCAL

We now give a high level overview of the structure and capabilities of the package.

The base class for all probability distributions is the class '`Distr`'. Objects of this class are random variables following some distribution. The result of any operation on objects of this class is again an instance of '`Distr`', which can be used in subsequent computations. The main part of the package implements efficient arithmetic on independent random variables with the four arithmetic operations: addition, subtraction, multiplication and division. This operations are implemented for the base class '`Distr`' and, thanks to operation overloading, can be performed using standard arithmetic symbols `+`, `-`, `*`, `/`. Besides basic arithmetic, three additional binary operations are available: `min`, `max`, and the power operation which overloads the `**` symbol. Moreover, a number of standard functions such as square root, exponential function or logarithm are available.

Summary statistics of random variables such as mean, variance, skewness, etc., can be computed through the methods of the '`Distr`' class. More advanced functionality like plots, such as histograms, quantiles and random number generation is also implemented.

Several standard continuous distributions are predefined in the package. Others, such as noncentral distributions, are defined, internally, using arithmetic operations on predefined distributions. Besides continuous random variables, discrete distributions are also available, but unlike the continuous case, they are restricted to finite domains. Continuous and discrete random variables can be freely mixed in calculations. Certain operations (e.g., `min` and `max`) can lead to mixed continuous-discrete distributions which are also supported. For example the random variable $Y = \min(X, 0)$, where $X \sim N(0, 1)$ is of such a mixed type with $\mathsf{P}(Y = 0) = 0.5$ and the continuous part being the right arm of the normal density.

The **stats** sub-package contains some useful statistical procedures for i.i.d. variables. It contains two modules: `stats.distr_est` implementing distribution parameter fitting and `stats.iid_ops` containing optimized routines for probabilistic arithmetic on i.i.d. random

variables. These include products, averages (arithmetic and geometric), minimum, maximum and order statistics.

The remaining part of the package implements an early approach to computations with dependent random variables. Currently the functionality is limited to the two-variable case and densities without singularities. The base class for multidimensional distributions is '`NDDistr`', representing the joint density of two variables, from which all multidimensional distributions are derived. Currently, bivariate product distribution, bivariate normal distribution and a joint distribution of two ordered statistics are available. More complicated relationships can also be modeled using copulas. After defining the joint distribution of two variables, arithmetic operations can be performed on them.

1.4. Installation

This paper is based on **PaCAL** version 1.1. All examples will also work in the newer versions.

PaCAL is freely available at SourceForge (`http://pacal.sourceforge.net/`). The package requires:

- Python programming language version 2.7 (Python Software Foundation 2010),
- **NumPy** (≥1.6) (Ascher *et al.* 2001), a package for efficient manipulation of multidimensional arrays,
- **Matplotlib** (≥1.0.0) (Hunter 2007), a plotting package,
- **Sympy** (≥0.7.2) (**SymPy** Development Team 2008), a symbolic computation package needed for the dependent random variables module,
- **SciPy** (≥0.9) (Jones, Oliphant, Peterson *et al.* 2001), a Python scientific library used for optimization and root finding.

The following package is also recommended:

- **Cython** (Behnel, Bradshaw, Citro, Dalcin, Seljebotn, and Smith 2011) needed only for compilation of optimized code sections from source. **PaCAL** works correctly without the compiled parts, albeit less efficiently.

The latest version of **PaCAL** can be installed directly from the PyPI repository of Python packages using

```
easy_install pacal
```

For Windows users, we provide a Windows installer. To install from source, the `tar.gz` file needs to be downloaded and unpacked. Afterward, the standard command for installation of Python packages needs to be executed in the package directory:

```
python setup.py install
```

1.5. A remark on semantics

Two cases need to be distinguished: that of identical random variables and that of identically distributed random variables. If X and Y are i.i.d. random variables, then $X + Y$ and $X + X = 2X$ have, in general, different distributions. **PaCAL**, however, treats both cases in the same way – *arguments of all operations are always treated as independent.* Consider the following two lines:

```
>>> S1 = UniformDistr() + UniformDistr()
>>> U = UniformDistr(); S2 = U + U

Warning: arguments treated as independent
```

The distributions of `S1` and `S2` are the same, but the second expression gives a warning. We inform the user, that the result may be inconsistent with her intentions. If this is the case, one should rather write `S2 = 2 * U`.

The semantics is different in the dependent variable part of the package, where using the same instance several times always refers to the same random variable. Consequently `U + U` is equivalent to `2 * U`.

2. Detailed description of PaCAL's capabilities

In this section we present the capabilities of **PaCAL** in detail together with several examples. We begin by presenting the available distributions, later we discuss the methods of the core '`Distr`' class and other facilities, such as functions of random variables, conditioning, plotting and random number generation.

2.1. Distributions

Several standard continuous distributions are implemented directly (default parameters are given in the parentheses):

- `UniformDistr(a = 0.0, b = 1.0)`
- `BetaDistr(alpha = 1, beta = 1)`
- `NormalDistr(mu = 0.0, sigma = 1.0)`
- `CauchyDistr(gamma = 1.0, center = 0.0)`
- `StudentTDistr(df = 2)`
- `FDistr(df1 = 1, df2 = 1)`
- `ChiSquareDistr(df = 1)`
- `GammaDistr(k = 2, theta = 2)`
- `ExponentialDistr(lmbda = 1)`
- `LaplaceDistr(lmbda = 1.0, mu = 0.0)`
- `ParetoDistr(alpha = 1, xmin = 1)`

- `LevyDistr(c = 1.0, xmin = 0.0)`
- `WeibullDistr(k = 3.0, lmbda = 1.0)`
- `GumbelDistr(mu = 0, sigma = 1)`
- `FrechetDistr(alpha = 2, s = 1, m = 0)`

Further distributions are implemented (internally) by applying arithmetic operations and functions to the above distributions. For convenience we have provided the following important noncentral distributions:

- `NoncentralTDistr(df = 2, mu = 0)`
- `NoncentralChiSquareDistr(df, lmbda = 0)`
- `NoncentralFDistr(df1 = 1, df2 = 1, lmbda = 0)`
- `NoncentralBetaDistr(alpha = 1, beta = 1, lmbda = 0)`

One can also define new distributions by explicitly providing their density function and using the class generated by `FunDistr(f, breakPoints = None)`; the `breakPoints` argument is the list of discontinuities, which can include `-Inf` and `+Inf` to indicate infinite domain. Another possibility is inheriting from the '`Distr`' class and overriding the `init_piecewise_pdf` and `rand_raw` methods.

Discrete distributions in **PaCAL** are subclasses of the '`DiscreteDistr`' class, whose constructor takes lists of points and their respective probability masses. Currently the following subclasses are available:

- `ConstDistr(c = 1.0)` with subclasses `ZeroDistr()` and `OneDistr()`: degenerate distributions corresponding to constants,
- `BernoulliDistr(p)`,
- `BinomialDistr(n, p)`,
- `PoissonDistr(lmbda, trunk_eps = 1e-15)` defines the truncated Poisson distribution. The tail of the distribution is truncated such that the total mass of probability is `trunk_eps` close to 1.

Arbitrary finite discrete distributions can be created by instantiating the '`DiscreteDistr`' class directly. Remember that **PaCAL** allows only for finite supports in the discrete case. Truncation can be used to approximate infinite distributions. The strategy is very effective for distributions with exponentially decaying probabilities, such as the Poisson. The approach will, however, be inefficient for slowly decaying distributions, because in this case the number of points with high probability masses may be extremely large.

Direct implementation of discrete random variables with infinite supports using **PaCAL**'s methodology seems to be a challenging task, as it would require efficient methods of approximating infinite, irregularly spaced discrete functions.

2.2. Summary statistics

The following descriptive statistics are implemented as methods of the '`Distr`' class and are thus available for all random variables:

`mean()` expected value, the first moment: $m = \int_{-\infty}^{\infty} x f(x)\, dx$

`var()` variance: $v = \int (x-m)^2 f(x)\, dx$

`std()` standard deviation: square root of variance

`moment(k, c)` kth moment of about the value `c`, when `c` is omitted it returns central moments (about the mean).

`skewness()` skewness: `moment(3, mean())/std()**3`

`kurtosis()` kurtosis: `moment(4, mean())/std()**4`

`entropy()` entropy: $-\int f(x)\log(f(x))\, dx$

`quantile(level)` quantile: solution of the equation $\int_{-\infty}^{y} f(x)\, dx =$ `level`

`median()` median: `quantile(0.5)`

`medianad()` median absolute deviation: median$(|X - \text{median}(X)|)$

`iqrange(level = 0.025)`: inter-quantile range for a given quantile level: `quantile(1 - level) - quantile(level)`

`range()`: domain of the random variable

`interval(p = 0.95)`: an interval that supports the middle p-percent of the distribution: `(quantile((1 - p)/2), quantile(1 - (1 - p)/2))`

`tail_exponent()`: estimate of the exponent β in the expression $f(x) \sim \frac{1}{x^\beta}$ as $x \to \pm\infty$

`mode()`: mode of the distribution.

The `summary()` method prints the most important descriptive statistics.

2.3. Numerical accuracy

High numerical accuracy was a key issue for us while designing the package, it was thus necessary to be able to assess the accuracy of the result. Clearly, when the theoretical distribution or its parameters, such as the mean, are known, we may use them to evaluate the accuracy directly. The interpolation and integration procedures also allow for error estimation, and such estimates are available in **PaCAL**. However, a very useful estimate of the accuracy is the integral of the density over the whole real line (the 'zeroth' moment). Since we take no specific steps to guarantee that the densities integrate to 1, the quantity $|1 - \int_{-\infty}^{\infty} f|$ is a useful indicator of accuracy which takes into account the accumulation of error over subsequent operations. We call this quantity the $\|\cdot\|_1$ *error*; it is available via the `int_error()` method of class '`Distr`'. We have noticed that the $\|\cdot\|_1$ error is a good indicator of the order of magnitude of errors of other statistical indicators such as the mean, variance or quantiles.

We now give two examples demonstrating the accuracy of **PaCAL** on two numerically difficult cases involving operations on densities with singularities. A related example can be found in Section 2.10. An example involving distributions with heavy tails, another kind of numerical difficulty, can be found in Section 2.8 and in Section 4 (Hill's estimator); Section 2.4 contains examples of extremely heavy tails pushing the package to its limits.

The first example is the sum of four χ^2 distributions with various numbers of degrees of freedom:

```
>>> X1 = ChiSquareDistr(1)
>>> X2 = ChiSquareDistr(17)
>>> X3 = ChiSquareDistr(82)
>>> X4 = ChiSquareDistr(100)
>>> S = X1 + X2 + X3 + X4
>>> S.summary()

  Chi2(1)+Chi2(17)+Chi2(82)+Chi2(100)
                mean  =  199.99999999999906
                 var  =  399.9999999999954
              median  =  199.33372983863111
           medianad  =  13.453154577985678
      iqrange(0.025)  =  78.32991300446642
      interval(0.95)  =  (162.72798250184633, 241.05789550631275)
             int_err  =  4.6629367034256575e-15
```

The χ^2 distribution is a good benchmark because of its known theoretical properties: the sum of independent χ^2 variables also has a χ^2 distribution whose number of degrees of freedom is the sum of degrees of freedom of the summation terms; all moments of the distribution are known and have integer values. Moreover, the distribution is numerically difficult: for one degree of freedom, the density has a singularity at zero which poses a challenge to interpolation and integration. The result `S` above follows a χ^2_{200} distribution, and we can compare its moments with known theoretical values. The $\|\cdot\|_1$ error is $4.66 \cdot 10^{-15}$. One can also immediately see, that the mean and variance are correct to at least 14 decimal digits. The median (not an integer) is also accurate to 14 decimal places.

A similar example demonstrates the noncentral χ^2 distribution. The density of this distribution does not have a closed form representation and, instead, is computed numerically as $X^2 + Y$, with $X \sim N(\lambda, 1)$ and $Y \sim \chi^2_{df-1}$, where df is the number of degrees of freedom and λ the noncentrality parameter. The density of the first term of the sum has a singularity at zero. Below we compute ten moments (m_k) and central moments (μ_k) of the distribution:

```
>>> X = NoncentralChiSquareDistr(df = 10, lmbda = 3)
>>> for k in range(11):
...   print k, "m_k=", repr(X.moment(k, 0)), ",\tmu_k=", repr(X.moment(k))

 0  m_k= 1.0000000000000004 ,        mu_k= 1.0000000000000004
 1  m_k= 13.000000000000005 ,        mu_k= -4.7739590058881731e-15
 2  m_k= 201.00000000000009 ,        mu_k= 32.000000000000028
 3  m_k= 3597.0000000000009 ,        mu_k= 151.99999999999955
 4  m_k= 73041.000000000044 ,        mu_k= 4127.9999999999991
 5  m_k= 1657773.0000000005 ,        mu_k= 58239.999999999927
 6  m_k= 41559129.00000003 ,        mu_k= 1336959.9999999991
 7  m_k= 1139822253.0000007 ,        mu_k= 29840639.999999974
 8  m_k= 33932500641.000019 ,        mu_k= 783211519.99999821
 9  m_k= 1089270783693.0005 ,        mu_k= 22124441599.999893
10  m_k= 37493858100968.984 ,        mu_k= 684756541439.99426
```

It is known that, for an integer λ, the moments of the noncentral χ^2 distribution are integers, so one can immediately see that the accuracy of 14–15 digits is achieved even for higher moments.

The noncentral χ^2 distribution can be computed using many equivalent expressions. The one described above was picked because it requires only a single convolution of two densities, only one of which involves a singularity, while the other is given by an explicit formula. As a rule of thumb, when several equivalent expressions exist, it is usually best to pick the one which involves the fewest operations on distributions with singularities and/or heavy tails, since they are more difficult to interpolate and integrate numerically.

For example, the noncentral χ^2 distribution can also be obtained using a more direct formula:

$$\sum_{i=1}^{df} X_i^2, \quad \text{where } X_i \sim N(\sqrt{\lambda/df}, 1).$$

This formula is much more challenging since it requires convolving 10 densities, each featuring a singularity at zero. Let us now compare both formulas in **PaCAL**, as well as the implementations of the noncentral χ^2 distribution in **scipy** and R (R Core Team 2013) using the following code fragment which computes the median of the distribution:

```
>>> mu = sqrt(0.3)
>>> df = 10
>>> lmbda = df * mu ** 2.0
>>> S = NormalDistr(mu, 1) ** 2
>>> for i in range(df - 1):
...   S += NormalDistr(mu, 1) ** 2
>>> X = NoncentralChiSquareDistr(df = 10, lmbda = 3)
>>> from scipy.stats import ncx2
>>> print "PaCAL simple                    : ", repr(X.median())
>>> print "PaCAL direct                    : ", repr(S.median())
>>> print "scipy.stats                     : ", repr(ncx2.median(df, lmbda))
>>> print "R: qchisq(0.5, df = 10, ncp = 3) : ", repr(12.210896063727795)
```

```
PaCAL simple                    :  12.21089606372774
PaCAL direct                    :  12.210896063728127
scipy.stats                     :  12.210897515189618
R: qchisq(0.5, df = 10, ncp = 3) :  12.210896063727795
```

As one can see, the results for both formulas are very similar in **PaCAL**. The simpler formula used in the '`NoncentralChiSquareDistr`' class achieves an excellent accuracy of 15 correct decimal digits. The direct formula, which requires significantly more operations also achieved a very high accuracy of 13 digits, only slightly worse than the simpler formula. It did however take much more time to perform the computations. The dedicated implementation in **scipy** (class '`ncx2`' from **scipy.stats**) achieved only 7 correct decimal places. The R implementation `qchisq()` gives results practically identical to **PaCAL**. Similar results have been obtained also for the moments of the distribution.

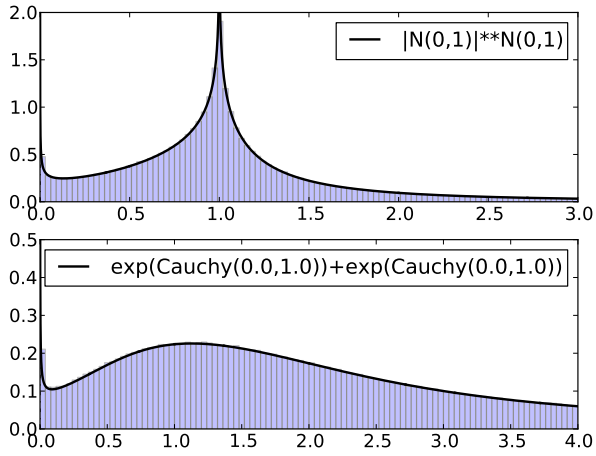

Figure 2: The power of two normal variables and the sum of two exponents of Cauchy random variables.

2.4. Functions of random variables

Functions of random variables are often used in statistics. We implement a number of standard functions: `exp`, `log`, `sqrt`, `atan`, `abs`, `sign`, `sin`, `cos` and `tan`[1]. Powers of random variables can be computed using the standard Python `**` operator. Moreover, the power operation is implemented also for the case when both arguments are random variables.

In the examples presented so far the accuracy has always been very high. Here we present two examples showing the behavior of the package in very difficult cases which push its capabilities to the limits:

```
>>> N = abs(NormalDistr()) ** NormalDistr()
>>> E = exp(CauchyDistr()) + exp(CauchyDistr())
>>> print N.int_error(), N.median()
>>> print E.int_error()

2.67842808e-06    1.00000013926
0.013874124313
```

The first line raises an absolute value of a normal variable to a power whose exponent is also a normal variable, the second computes the sum of exponents of two independent Cauchy random variables. Both densities are shown in Figure 2. In the first case, the $\|\cdot\|_1$ error of the result is $2.67 \cdot 10^{-6}$. Note the singularities at zero and at one; it is the second singularity which causes the loss of precision[2]. The distribution does not have any moments, but it can be seen that the median is accurate up to 6 digits. The second result has the $\|\cdot\|_1$ error of $1.38 \cdot 10^{-2}$. This might seem to be quite a weak result, note however that the exponent

[1]Trigonometric functions may currently suffer from some loss of accuracy, this will be fixed in future versions.

[2]We plan to improve the accuracy of singularities at nonzero locations in the future, but from the practical point of view, singularities at zero are the most important ones and are already handled with excellent accuracy.

of a Cauchy random variable decreases only as $\frac{1}{x \log^2 x}$ for $x \to \infty$. The tail is so heavy that $4.48 \cdot 10^{-4}$ of the probability mass lies beyond the range of double precision arithmetic! **PaCAL** is nevertheless able to add the two random variables and, as the histogram shows, the result is fairly accurate. See Section 2.8 and the Hill's estimator example in Section 4 for more typical heavy tailed distributions which **PaCAL** integrates with very high precision.

The next example demonstrates the `abs` and `sign` functions. It illustrates a property of symmetric distributions: the product of two independent random variables representing their absolute value and sign (a discrete random variable taking values -1 and $+1$) follows the original distribution.

```
>>> S = sign(NormalDistr()) * abs(NormalDistr())
>>> for i in range(11):
...   print i, S.moment(i, 0)

0 1.0
1 -5.55111512313e-17
2 1.0
3 -1.11022302463e-16
4 3.0
5 -1.33226762955e-15
6 15.0
7 -1.42108547152e-14
8 105.0
9 -8.52651282912e-14
10 945.0
```

The moments of the normal distribution have been recovered with very high accuracy.

2.5. Random number generation

Random number generation is supported through the `rand` method of the '`Distr`' class:

`rand(n)` returns a random sample of size `n` taken from the distribution of the random variable on which the method is called.

For standard distributions, the method uses native random generators from the **numpy.random** package; arithmetic operations are implemented by performing those operations on samples returned by the `rand` method of the arguments. A more difficult case arises for conditional (Section 2.9) or user defined distributions, where such methods are not available. In those cases we resort to sampling based on the inverse of the cumulative distribution function implemented by the method `rand_invcdf(n, use_interpolated = True)`. If `use_interpolated` is set to `True`, the inverse CDF is precomputed (using numerical root finding) and stored in an interpolated form before generating random numbers, greatly speeding up the approach at the cost of a slight decrease in accuracy.

For comparison, taking a sample of size 1000000 from the normal distribution using `rand` takes 0.06 s and is about 40 times faster then using `rand_invcdf` with the flag `use_interpolated =`

`True` which takes 2.2 s. Using `rand_invcdf`, without interpolation is much slower: generating a sample of size 10000 takes about 26 s.

2.6. Plots and histograms

Graphical methods available in the '`Distr`' class are:

`plot(xmin = None, xmax = None, **kwargs)` plots the probability density function or the random variables distribution. For discrete and mixed distributions points of nonzero probability are marked using upward pointing arrows of appropriate height. `xmin` and `xmax` allow for specifying the range of the plot (by default chosen automatically) and `kwargs` contains extra formatting arguments to be passed to **matplotlib**'s `plot` function,

`hist(n = 1000000, xmin = None, xmax = None, bins = 50, **kwargs)` draws a histogram based on a random sample of size `n`. When `xmin` and `xmax` are given, samples are generated only from the specified interval using rejection sampling[3]. `kwargs` are extra arguments passed directly to **matplotlib**'s plotting routines.

2.7. Operating on probability densities and cumulative distributions

The '`Distr`' class serves as an external representation of the random variable, but most of the computations are performed internally on probability density functions (PDFs) which are implemented as piecewise interpolated functions. It is possible to directly access the PDF as well as the cumulative distribution function (CDF) of a distribution using the following methods:

`pdf(x)` returns the value of the PDF at `x`,

`cdf(x)` returns the value of the CDF at `x`,

`quantile(y)` returns the quantile function of the random variable at `y`,

`get_piecewise_pdf()` returns an object representing the PDF,

`get_piecewise_cdf()` returns an object representing the CDF,

`get_piecewise_invcdf()` returns an object representing the inverse of the CDF (the quantile function).

The last three methods return objects of the '`PiecewiseFunction`' class which represents piecewise continuous functions. Several operations can be performed on objects of this class like plotting or integrating. It is also possible to perform pointwise arithmetic operations such as addition or subtraction (appropriate operators have been overloaded). The '`PiecewiseFunction`' class is thus a simple Python version of the '`chebfun`' class form the **Chebfun** library (Battels and Trefethen 2004; Berrut and Trefethen 2004), containing, however, only the few operations needed to implement probabilistic arithmetic.

The following example shows an application of this functionality to compute the difference between the true and empirical CDFs, and the Kolmogorov-Smirnov statistic:

[3]This can lead to very long computation times for narrow intervals.

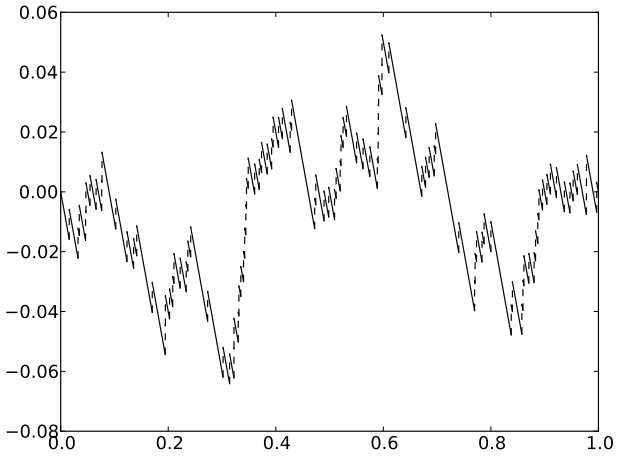

Figure 3: Difference between empirical and true CDFs for a uniform sample of size 50.

```
>>> U = UniformDistr()
>>> X = U.rand(50)
>>> D = DiscreteDistr(xi = X, pi = ones_like(X) / len(X))
>>> r = D.get_piecewise_cdf() - U.get_piecewise_cdf()
>>> r.plot()
>>> x, D_n = r.max_abs()
>>> print "D_n =", D_n, ",  K_n =", sqrt(50) * D_n

D_n = 0.0641160362945 ,  K_n = 0.453368840466
```

`K_n` is the value of the Kolmogorov-Smirnov statistic and `D_n` the maximum absolute difference between the cumulative distribution functions. The plot is shown in Figure 3. This is in fact a finite approximation of a Brownian bridge process.

2.8. Discrete and mixed distributions

Discrete distributions can be used in the same way as continuous ones. Mixed continuous-discrete distributions are also possible. Below are some examples.

In Feller (1971, Chapter V, Chapter 4) there is an interesting example showing that the uniform distribution is a convolution of two singular Cantor-type distributions. In **PaCAL**, due to finite representation, we cannot represent such distributions directly. They can however be approximated:

```
>>> V, U = ZeroDistr(), ZeroDistr()
>>> for i in range(4):
...     V += BernoulliDistr() / 4 ** (i + 1)
...     U += 2 * BernoulliDistr() / 4 ** (i + 1)
...
>>> X = V + U
```

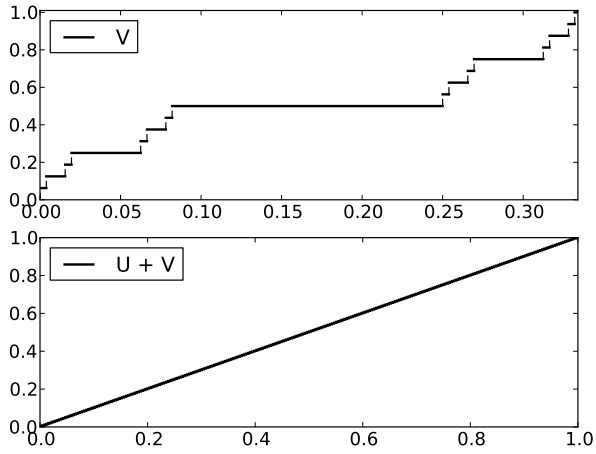

Figure 4: Approximation of Cantor-type distributions whose sum is the uniform distribution.

Figure 4 shows the CDFs of `V` and `X` (the CDF of `U` is similar to `V`). It can be seen that `X` approximates the uniform distribution quite well; in fact, its CDF is composed of 256 steps.

The next example is somewhat artificial, but is difficult from a numerical point of view because of heavy tails and discontinuities in the result. Consider the quotient of a mixture of uniform distributions[4]:

```
>>> NUM = UniformDistr(0, 1) + BinomialDistr(n = 2, p = 0.6)
>>> DEN = UniformDistr(-1, 0) + BinomialDistr(n = 2, p = 0.3)
>>> Q = NUM / DEN
>>> Q.summary()
>>> Q.plot(xmin = -10, xmax = 10, linewidth = 2.0)
>>> Q.hist(xmin = -10, xmax = 10)

  (U(0,1)+Binom(2,0.6))/(U(-1,0)+Binom(2,0.3))
                mean  =  nan
                 var  =  nan
             entropy  =  3.4886677329629441
              median  =  0.1811594202898548
           medianad  =  3.117602787965399
     iqrange(0.025)  =  61.879999999999626
     interval(0.95)  =  (-33.31999999999968, 28.559999999999945)
              range  =  (-inf, inf)
             tailexp  =  (-2.0000000000001492, -2.0000000000001492)
             int_err  =  0.0
```

The result `Q` (shown in Figure 5) is very accurate, the $\|\cdot\|_1$ error turned out to be zero; it

[4]Convolving a discrete and a continuous distribution is equivalent to forming a mixture of the continuous distribution.

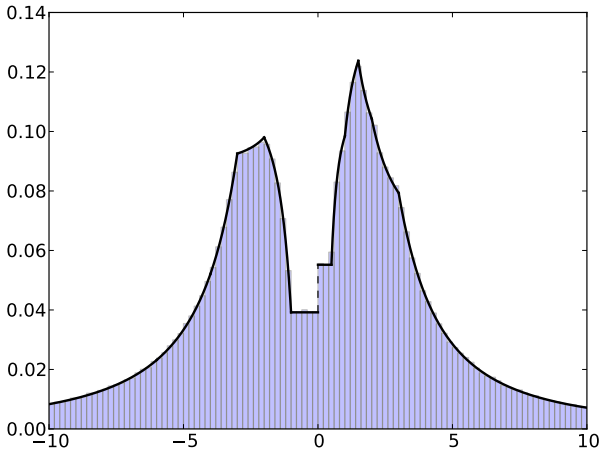

Figure 5: A quotient of two mixtures of uniform random variables.

also agrees well with the histogram. Tail exponents are correctly estimated as 2, mean and variance do not exist (due to heavy tails), which is also correctly indicated in the summary.

In the last example of this section, we compute the minimum `M` of a continuous uniform random variable `C` and a discrete random variable `D`, which has a mixed continuous-discrete distribution:

```
>>> xi = [0.0, 0.2, 0.4, 0.6, 0.8]
>>> pi = [0.2, 0.2, 0.2, 0.2, 0.2]
>>> D = DiscreteDistr(xi = xi, pi = pi)
>>> C = UniformDistr()
>>> M = min(C, D)
>>> S = M + UniformDistr()
>>> M.plot(linewidth = 1.0, label = M.getName())
>>> S.plot(linestyle = '-', linewidth = 2.0, label = S.getName())
>>> legend()
```

When we add a uniform random variable to `M`, we obtain a random variable with a purely continuous distribution. The resulting plots are shown in Figure 6, note the arrows marking points with nonzero probability mass in the upper plot.

2.9. Conditional distributions

In **PaCAL**, there are two classes representing conditional (truncated) distributions:

`CondGtDistr(X, L)` the distribution of `X` conditioned on $\texttt{X} > L$,

`CondLtDistr(X, U)` the distribution of `X` conditioned on $\texttt{X} < U$,

where U and L are constants. We have overridden the symbol `|` (Python's or), to make our syntax closer to standard probabilistic notation. The syntax is: `X |` *Condition* where

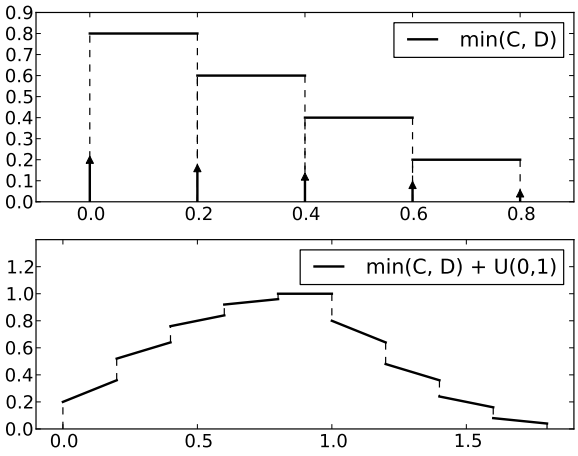

Figure 6: Minimum of discrete and continuous uniform random variables and its sum with a uniform random variable. Note the arrows marking points with nonzero probability mass.

`X` is a random variable (instance of class '`Distr`') and *Condition* is one of: `Gt(L)`, `Lt(U)`, `Between(L, U)`. For example, let us verify that the exponential distribution is a memoryless distribution:

```
>>> E = ExponentialDistr()
>>> CE = E | Gt(7)
>>> r = E.get_piecewise_pdf() - (CE - 7).get_piecewise_pdf()
>>> print "absolute_error =", r.max_abs()[1]

absolute_error = 4.3520742565306136e-13
```

2.10. Operations on i.i.d. random variables

The module `pacal.stats.iid_ops` contains optimized operations for sequences of i.i.d. variables. The following functions are provided:

`iid_sum(X, n)`, `iid_average(X, n)`: sum and average of n i.i.d. r.v.s with distribution identical to that of `X`,

`iid_prod(X, n)`, `iid_average_geom(X, n)`: product and geometric mean of n i.i.d. r.v.s,

`iid_order_stat(X, n, k)`: the distribution of kth order statistic of an i.i.d. sample size of n.

`iid_min(X, n)`, `iid_max(X, n)`, `iid_median(X, n)`: special cases of order statistics.

All the above methods take an additional argument `all`, by default set to `False`. When set to `True`, the function returns a vector of results for all $i = 1, \ldots, n$. E.g., `iid_sum(X, n, all = True)` returns the vector of partial sums: $S_1 = X_1, S_2 = X_1 + X_2, \ldots, S_n = X_1 + \ldots + X_n$. If `all` were set to `False`, only a single value, S_n, would be returned.

The example below demonstrates the extreme value theorem for the exponential distribution.

```
>>> X = ExponentialDistr()
>>> for n in range(4, 25, 5):
...   M = iid_max(X, n)
...   (M - log(n)).plot()
...
>>> GumbelDistr().plot()
```

The result illustrates how the distribution of the maximum of i.i.d. exponential variables rapidly approaches the Gumbel distribution. The plot is omitted to save space.

2.11. Distribution fitting

The `stats.distr_est` module contains a single class '`LoglikelihoodEstimator`' which can be used for simple fitting of the parameters of continuous distributions to data using the maximum likelihood method. Below we use a sample taken form a beta distribution to recover its parameters:

```
>>> sample = BetaDistr(2, 4).rand(100)
>>> MLE = LoglikelihoodEstimator(BetaDistr, xi = sample)
>>> par = MLE.find_params()
>>> print par
>>> B = BetaDistr(**par)

Optimization terminated successfully.
         Current function value: -38.641360
         Iterations: 47
         Function evaluations: 89
{'alpha': 2.0546203507634977, 'beta': 4.229057983016892}
```

3. Towards dependent random variables

All variables in the previous sections have been assumed to be independent. Sometimes, however, this assumption does not hold, and we need to compute with dependent random variables. This section describes the facilities for such computations available currently in **PaCAL**. They are essentially limited to the two-variable case, extension to larger number of variables is a work in progress. Operations on dependent variables are more restricted then in the case of independence. Only continuous variables are implemented, densities are not allowed to have singularities, maximum and minimum operations are currently not available.

The process of computing with dependent random variables requires two steps: specifying the joint distribution of the variables and computing the distribution of the result of arithmetic operations on them. We describe those steps in turn.

The abstract class '`NDDistr`' is the base class for multidimensional (currently two-dimensional) distributions. The main method of this class is `pdf(*X)` which defines the joint probability density function of the two random variables. Its arguments are the values of each dimension of the variable. Several subclasses are available including multidimensional normal distributions, copulas etc. They are described in detail below.

The class '`TwoVarsModel`' performs the actual arithmetic on dependent random variables. One needs to supply the joint distribution of the two variables and the function of the random variables to be computed. The class has a method `eval` which performs the actual computations and returns the distribution of the desired function of the two variables. While functions of only two dependent variables are currently supported, the function itself may be defined using an arbitrary number of intermediate variables.

It is currently assumed that the function can be inverted symbolically. Mathematical details of the computation are described in the Appendix. The class depends on the **sympy** package for symbolic computation (**SymPy** Development Team 2008). Symbolic computations are used to find the inverse of the function or the random variables and its Jacobian. Note that this is different from purely symbolic approaches, as the distributions themselves are represented numerically and need not be expressed explicitly.

3.1. Multidimensional distributions

We now describe the multidimensional distributions available in **PaCAL**:

'`NDNormalDistr`' class implements the multidimensional normal distribution with given mean vector `mu` and covariance matrix `Sigma`.

'`IJthOrderStatsNDDistr`' class represents the joint distribution of the ith and jth order statistics of an i.i.d. random sample. The constructor takes four arguments: `X` the distribution from which the sample is taken, `n` the size of the sample, `i`, `j` the order statistics to use.

Copulas. The most flexible option available in **PaCAL** are copula distributions which are described in detail below. The constructors take the argument `marginals` containing a list of marginal distributions, and possibly other arguments: parameters of the copula.

Copulas

Copulas allow for construction of multidimensional distributions with given marginal distributions. They allow one, for example, to obtain a joint distribution with given marginals and given rank correlation coefficient. Detailed discussion of copulas is beyond the scope of this paper, for more information see e.g., Nelsen (2006); some basic information is also included in the Appendix.

There are four copula types available in **PaCAL** (we follow the naming used by Nelsen 2006). The simplest is the `PiCopula` which implements the product distribution corresponding to independent random variables. The three remaining types of copulas are: `FrankCopula`, `GumbelCopula`, `ClaytonCopula`. See the Appendix, the examples below, and Nelsen (2006) for details. We do not aim to provide a full featured copula implementation at this moment, just a usable solution to do useful work with dependent random variables.

3.2. Examples

Consider the sum of two dependent random variables $X \sim U(0,1)$ and $Y \sim U(0,1)$. The extreme case of complete positive correlation corresponds to $X+Y = 2X = 2Y$ and the result

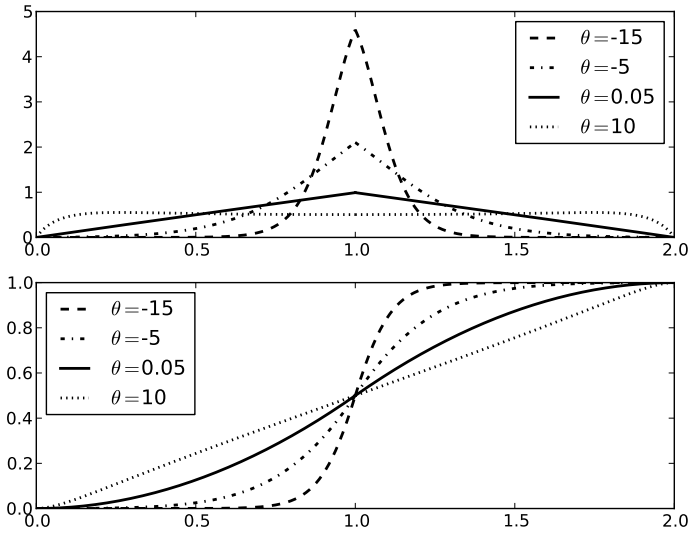

Figure 7: Sum of two dependent uniform random variables for various values of the θ parameter of the Frank copula defining their joint distribution.

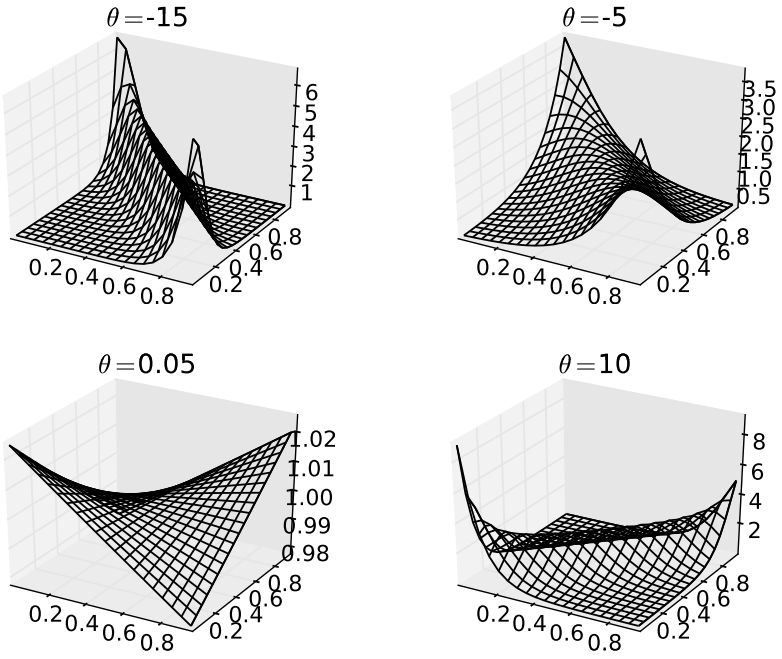

Figure 8: Joint density functions for Frank copula for different values of the θ parameter.

is uniform on $[0, 2]$. If, on the other hand, correlation is maximal negative then the result is a degenerate one point distribution with all mass concentrated at 1. Here we use the Frank copula with parameters $\theta = \{-15, -5, 0.05, 10\}$, such that the correlation varies from strong negative to strong positive. The correlation coefficients in those four cases are about -0.93, -0.64, 0.008, and 0.86 respectively.

```
>>> X = UniformDistr(0, 1, sym = "X")
>>> Y = UniformDistr(0, 1, sym = "Y")
>>> for theta, ls in [(-15, "--"), (-5, "-."), (0.05, "-"), (10, ":")]:
...   Copula = FrankCopula(marginals = [X, Y], theta = theta)
...   M = TwoVarsModel(Copula, X + Y)
...   S = M.eval()
...   S.plot(linestyle = ls)
```

The resulting PDFs and CDFs are shown in Figure 7. It can be seen that the more negatively correlated the variables, the more the distribution concentrates around one; when the variables are close to independence ($\theta = 0.05$) the distribution is close to triangular; for highly positively correlated variables the distribution is close to uniform. Figure 8 shows the joint density functions of X and Y for different values of the θ parameter. Notice also the `sym` argument given to the `UniformDistr` constructor. It is the name of the variable to be used in symbolic computations. The name may be omitted, but providing it results in a more readable output.

Parallel circuit of two resistors

The equivalent resistance of a parallel combination of two resistors is equal to

$$R = \frac{R_1 R_2}{R_1 + R_2},$$

where R_1 and R_2 are the resistances of the two resistors. We would like to apply **PaCAL** to this formula when the resistances are not known exactly, and instead, their probability distributions are given. Even though R_1 and R_2 are independent, we cannot use the arithmetic of independent random variables because the numerator and the denominator of the fraction are dependent. Note that in this specific case we can rewrite the expression as $(\frac{1}{R_1} + \frac{1}{R_2})^{-1}$ for which arithmetic of independent variables can be used, however, the modules of **PaCAL** which operate on dependent variables can handle this expression directly. Consider for example the case where $R_1 = 1 \pm 0.5\,\Omega$, $R_2 = 2 \pm 0.5\,\Omega$:

```
>>> R1 = UniformDistr(0.5, 1.5)
>>> R2 = UniformDistr(1.5, 2.5)
>>> pr = PiCopula(marginals = [R1, R2])
>>> M = TwoVarsModel(pr, R1 * R2 / (R1 + R2))
>>> R_equiv = M.eval()
>>> R_equiv.plot(linewidth = 2.0, color = 'k')
>>> R_equiv.hist()
>>> R_equiv.summary()
```

The result is shown in Figure 9. Note that the result is an object of class '`Distr`' and supports all its methods, for example using `R_equiv.cdf(0.75)` we obtain $\mathsf{P}(R < 0.75\,\Omega) = 0.7266$.

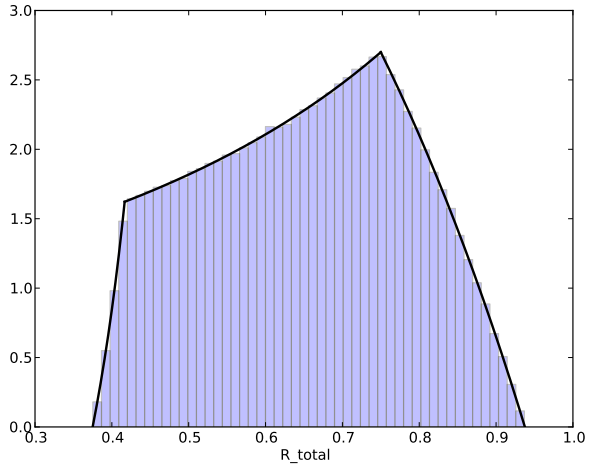

Figure 9: Distribution of the equivalent resistance of a parallel circuit of two resistors.

General regression

This example does not directly involve arithmetic on random variables, but it demonstrates other capabilities of **PaCAL** for computing with dependent variables. Given two dependent random variables X and Y, one can define several types of regressions of Y on X. Typically, one models the expectation of Y conditioned on X, but there are other possibilities, including conditional median, mode or quantiles. Given a joint distribution of X and Y one can use **PaCAL** to easily obtain such regression curves. For a given value x we build a conditional distribution of Y conditioned on $X = x$ and calculate any parameter of the distribution such as the mean, median, mode, etc. Repeating the procedure for various values of x a regression curve is obtained. Below is a sample piece of code which draws a regression curve for the mean of Y when the joint distribution of X and Y is defined using the Frank copula.

```
>>> def condmean(x, F, X, Y):
...   fun = lambda y: F.pdf(x, y) / X.pdf(x)
...   bpt = Y.get_piecewise_pdf().getBreaks()
...   distr = FunDistr(fun = fun, breakPoints = bpt)
...   return distr.mean()
...
>>> X = UniformDistr(0, 1) + UniformDistr(0, 1)
>>> X.setSym("X")
>>> Y = BetaDistr(1, 4, sym = "Y")
>>> F = FrankCopula(theta = 4, marginals = [X, Y])
>>> F.contour()
>>> xx, yy = linspace(0.01, 1.99, 30), []
>>> for x in xx:
...   yy.append(condmean(x, F, X, Y))
>>> plot(xx, yy)
```

Figure 10 shows the contour plot of the joint probability distribution F and regression curves

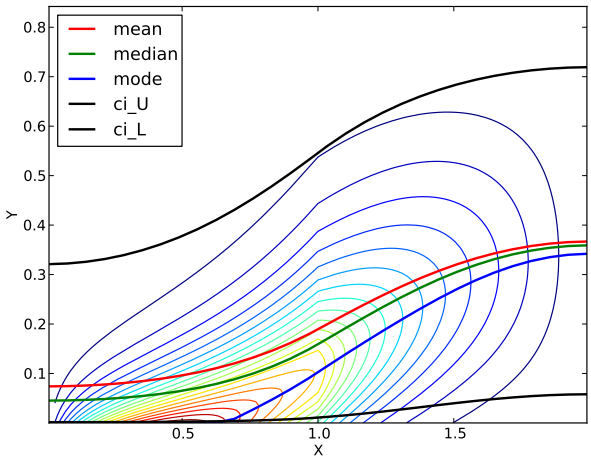

Figure 10: Conditional excepted value, median, mode and confidence interval for Frank copula with $\theta = 4$.

for the mean, median, mode and 95% confidence interval of Y conditioned on X.

4. Applications

In this section we present examples of statistical applications of **PaCAL**.

Sample distribution of the difference of proportions

Some tests for equality of two proportions (Newcombe 1998) rely on the exact distribution of their difference. Suppose we have two Bernoulli distributions with success probabilities p_1 and p_2, draw independent random samples of size $n_1 = 8$ and $n_2 = 7$ according to those distributions, and want to compute the distribution of the difference of proportions of successes in the two samples. Suppose that $n_1 = 8$, $n_2 = 7$, $p_1 = 1/2$, $p_2 = 2/7$; the desired distribution can easily be computed in **PaCAL**:

```
>>> B1 = BinomialDistr(8, p = 1.0/2.0)
>>> B2 = BinomialDistr(7, p = 2.0/7.0)
>>> D = B1 / 8 - B2 / 7
>>> D.plot()
>>> D.get_piecewise_cdf().plot()
>>> print D.interval(0.05)

(-0.30357142857142855, 0.625)
```

Figure 11 shows the density and cumulative distribution function of the result.

Hill's tail exponent estimator

Suppose that our data comes from a heavy tailed distribution with density decreasing as $x^{-\beta}$ when $x \to \infty$. We want to estimate β based on an i.i.d. sample of size N. A popular choice

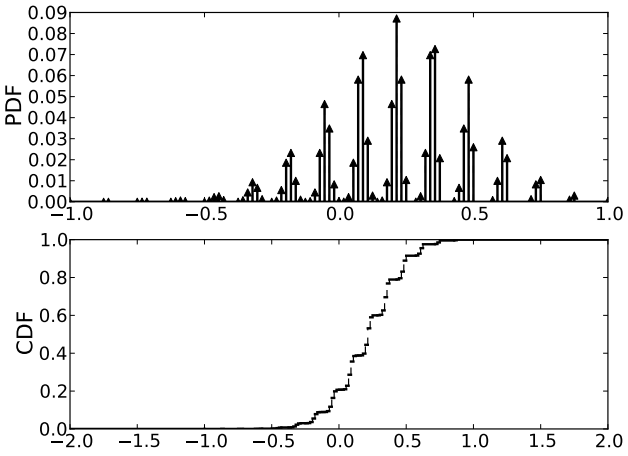

Figure 11: Difference between sample proportions in two binomial populations.

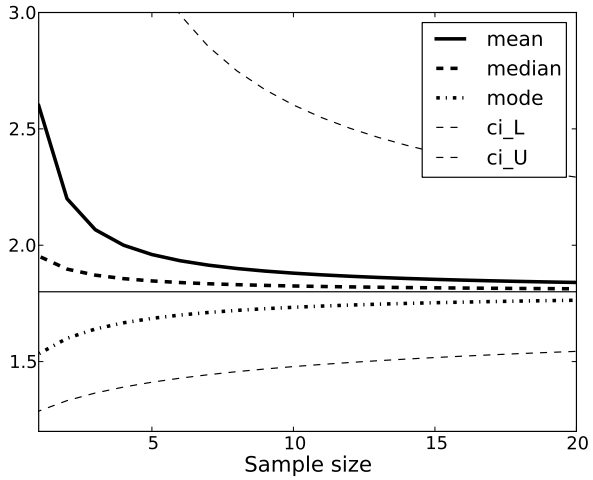

Figure 12: Hill's tail exponent estimator for various sample sizes. Mean, median, mode and 95% confidence intervals (ci_L, ci_U) are shown.

is the Hill's estimator given by the following equation (Clauset, Shalizi, and Newman 2009)

$$\hat{\beta} = 1 + N \left[\sum_{i=1}^{N} \log \frac{X_i}{x_{\min}} \right]^{-1},$$

where X_i are the samples and $x_{\min} > 0$ a threshold; only samples X_i greater than or equal to $x_{\min}$ are included in the sample. The function below returns the sample distribution of Hill's estimator for a Pareto sample with parameter `alpha` (tail exponent equal to `alpha` + 1) and size `n`.

```
>>> def Hills_estimator(n, alpha, xmin = 1):
...   d = ParetoDistr(alpha) | Gt(xmin)
```

```
...   s = iid_sum(log(d / xmin), n)
...   return 1 + n / s
```

Figure 12 shows the mean and 95% confidence interval of the estimator for various sample sizes taken from the Pareto distribution with parameter `alpha = 0.8` and $x_{\min} = 1$; the true tail exponent equals 1.8. It can be seen that the estimator is biased and that the bias decreases with sample size; the median and mode of the estimator are also shown. Using **PaCAL**, we can easily obtain any parameters of the estimate, e.g., for $N = 21$ and significance level of 0.05 we get:

```
>>> H = Hills_estimator(21, 0.8)
>>> print H.mean()
>>> print H.median()
>>> print H.interval()

          mean = 1.8400000000000083
        median = 1.8128655833263683
interval(0.95) = (1.543893889570158, 2.2923742014553357)
```

The $\|\cdot\|_1$ error is about $5.32 \cdot 10^{-15}$. It can also be seen that the bias of the estimator for $N = 21$ is already quite small.

Weighted sum of t distributions

Some statistical procedures use weighted sums of Student's t distributions. For example one of the solutions to the Behrens-Fisher problem uses the statistic $t_1 \sin\theta - t_2 \cos\theta$ for some constant θ (Springer 1979). Using **PaCAL** one can easily work with such distributions. For example if t_1 and t_2 are Student's t distributed with 10 and 20 degrees of freedom respectively, and $\theta = \pi/4$, the distribution can be obtained using the following code

```
>>> theta = pi/4
>>> weightedT = sin(theta) * StudentTDistr(10) - \
...   cos(theta) * StudentTDistr(20)
```

The interval supporting the middle 95% of the distribution is $[-2.146, 2.146]$. The $\|\cdot\|_1$ error of the computed distribution is $6.66 \cdot 10^{-16}$, so one can expect the results to be very accurate.

Distributions of sample range and interquantile range

Order statistics have been implemented in **PaCAL** as discussed in Section 2.10. However, different order statistics of the sample are not independent, so arithmetic of independent random variables cannot be used. We have thus also implemented the joint probability distribution of the ith and jth order statistic (see Appendix A.1 for the formulas used). The class constructed by `IJthOrderStatsNDDistr(X, n, i, j)` implements the two dimensional joint distribution of ith and jth order statistic for a sample of size n taken from a population whose distribution is given by `X`.

Consider a sample of size 8 taken from a normal population. Suppose we want to obtain the distribution of the difference between the 7th and the 2nd value in the ordered sample.

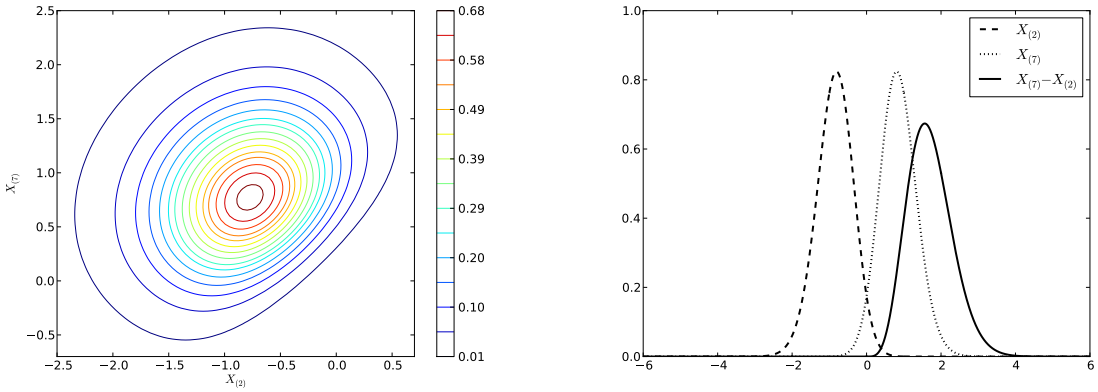

Figure 13: The 2nd and the 7th order statistics of a sample of size 8 taken form a normal population: Joint distribution (left) and distribution of the difference (right).

The class `IJthOrderStatsNDDistr` defines the joint distribution and makes each of the order statistics available in the `marginals` field:

```
>>> X = NormalDistr(0, 1, sym = "X")
>>> OIJ = IJthOrderStatsNDDistr(X, n = 8, i = 2, j = 7)
>>> X2, X7 = OIJ.marginals
>>> M = TwoVarsModel(OIJ, X7 - X2)
>>> R = M.eval()
>>> R.plot(); R.summary()
>>> X2.plot()
>>> X7.plot()
```

```
                mean  =  1.7044497241256709
                 var  =  0.35230886834402886
            skewness  =  0.45001967571540297
            kurtosis  =  3.1581082698082792
             entropy  =  0.87728490170498374
              median  =  1.6584918445680656
           medianad  =  0.40137447447604147
     iqrange(0.025)  =  2.3011403582330137
     interval(0.95)  =  (0.6848524640923669, 2.9859928223253807)
               range  =  (-inf, inf)
             int_err  =  -5.7208016102094916e-11
```

The left panel of Figure 13 shows the joint distribution of both order statistics and the right panel the distributions of each order statistic and of their difference. Notice that the result is an asymmetric distribution (kurtosis is greater than 3). Accuracy is estimated at $5.72 \cdot 10^{-11}$. We can also easily estimate the accuracy of the mean (mean of the sum/difference of random variables is the sum/difference of means even for dependent variables) as $1.17 \cdot 10^{-9}$.

Distributions of the sample range (difference between the largest and the smallest elements of the sample) have applications in quality control. The **SAS/QC** (SAS Institute 2011) package

implements functions `D2` and `D3` which return the mean and standard deviation of sample range of n independent normal random variables. Setting `n = 5`, `i = 1`, and `j = 5` in the last example, we obtain `R.mean() = 2.3259289472810414` and `R.std() = 0.86408194109950442`, the same values the **SAS/QC** package gives (respectively 2.3259289473 and 0.8640819411).

4.1. Other examples

We have tested **PaCAL** on hundreds of examples, including most examples and exercises from Springer (1979), as well as several properties of various distributions given in Wikipedia and statistical textbooks. Many examples are available on the package's website at `http://pacal.sourceforge.net/gallery/`.

5. Conclusions and future research

The **PaCAL** package provides accurate and efficient arithmetic on independent random variables. As our examples have shown, even complicated computations can be performed with accuracy close to machine precision. Besides arithmetic, the package provides many functions which are useful in statistical calculations, such as means, moments, medians, plots, histograms, etc.

A somewhat limited, but nevertheless useful, module for computation with dependent variables is also available. Extending this module to allow for arithmetic with more than two dependent variables and more complex dependency models will be the main goal of future development.

Acknowledgments

This work was supported by Research Grant no. N N516 068537 of the Polish Ministry of Science and Higher Education (Ministerstwo Nauki i Szkolnictwa Wyższego) from research funds for the period 2009–2011.

References

Ascher D, *et al.* (2001). "Numerical Python." *Technical Report UCRL-MA-128569*, Lawrence Livermore National Laboratory. URL `http://numpy.scipy.org/`.

Battels Z, Trefethen LN (2004). "An Extension of MATLAB to Continuous Functions and Operators." *SIAM Journal of Scientific Computing*, **25**(5), 1743–1770.

Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn DS, Smith K (2011). "**Cython**: The Best of Both Worlds." *Computing in Science Engineering*, **13**(2), 31 –39.

Berrut JP, Trefethen LN (2004). "Barycentric Lagrange Interpolation." *SIAM Review*, **46**(3), 501–517.

Boyd JP (1987). "Exponentially Convergent Fourier-Chebshev Quadrature Schemes on Bounded and Infinite Intervals." *Journal of Scientific Computing*, **2**(2), 99–109.

Boyd JP (2001). *Chebyshev and Fourier Spectral Methods*. Dover Publishers.

Clauset A, Shalizi CR, Newman MEJ (2009). "Power-Law Distributions in Empirical Data." *SIAM Review*, **51**(4), 661–703.

Davis PJ, Rabinowitz P (1984). *Methods of Numerical Integration*. Academic Press, London.

Feller W (1971). *An Introduction to Probability Theory and Its Applications*, volume 2. John Wiley & Sons.

Glen AG, Evans DL, Leemis LM (2001). "**APPL**: A Probability Programming Language." *The American Statistician*, **55**(2), 156–166.

Hoffmann-Jørgensen J (1994a). *Probability with a View toward Statistics*, volume 2. Chapman and Hall, New York.

Hoffmann-Jørgensen J (1994b). *Probability with a View toward Statistics*, volume 1. Chapman and Hall, New York.

Hunter JD (2007). "**Matplotlib**: A 2D Graphics Environment." *Computing in Science & Engineering*, **9**(3), 90–95.

Jaroszewicz S, Korzeń M (2012). "Arithmetic Operations on Independent Random Variables: A Numerical Approach." *SIAM Journal of Scientific Computing*, **34**(3), A1241–A1265.

JCGM (2009). "Evaluation of Measurement Data – An Introduction to the 'Guide to the Expression of Uncertainty in Measurement' and Related Documents." *Technical Report 104*, Bureau International des Poids et Mesures, Sèvres Cedex, France.

Jones E, Oliphant T, Peterson P, *et al.* (2001). "**SciPy**: Open Source Scientific Tools for Python." URL http://www.scipy.org/.

Knuth DE (1998). *The Art of Computer Programming*, volume 2. Addison Wesley Longman.

Lunn D, Thomas A, Best NG, Spiegelhalter DJ (2000). "**WinBUGS** – A Bayesian Modelling Framework: Concepts, Structure, and Extensibility." *Statistics and Computing*, **10**(4), 325–337.

Maplesoft (2008). *Maple 12*. Waterloo Maple Inc., Waterloo. URL http://www.maplesoft.com/products/maple/.

Nelsen RB (2006). *An Introduction to Copulas*. Springer-Verlag, New York.

Newcombe RG (1998). "Interval Estimation for the Difference between Independent Proportions: Comparison of Eleven Methods." *Statistics in Medicine*, **17**(8), 873–890.

Patil A, Huard D, Fonnesbeck CJ (2010). "**PyMC**: Bayesian Stochastic Modelling in Python." *Journal of Statistical Software*, **35**(4). URL http://www.jstatsoft.org/v35/i04.

SymPy Development Team (2008). ***SymPy**: Python Library for Symbolic Mathematics*. URL http://www.sympy.org.

Python Software Foundation (2010). *Python Software, Version 2.7*. URL http://www.python.org/.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

SAS Institute (2011). ***SAS/QC** 9.3 User's Guide*. SAS Institute, Cary, NC.

Shenoy PP, West JC (2011). "Inference in Hybrid Bayesian Networks Using Mixtures of Polynomials." *International Journal of Approximate Reasoning*, **52**(5), 641–657.

Spiegelhalter DJ, Thomas A, Best NG, Lunn D (2003). ***WinBUGS** Version 1.4 User Manual*. MRC Biostatistics Unit, Cambridge. URL http://www.mrc-bsu.cam.ac.uk/bugs/.

Springer MD (1979). *The Algebra of Random Variables*. John Wiley & Sons.

Trefethen LN (2008). "Is Gauss Quadrature Better than Clenshaw-Curtis?" *SIAM Review*, **50**(1), 67–87.

Trefethen LN, others (2011). ***Chebfun** Version 4.0*. The **Chebfun** Development Team. URL http://www.maths.ox.ac.uk/chebfun/.

van Rossum G, *et al.* (2011). *Python Programming Language*. URL http://www.python.org.

Waldvogel J (2006). "Fast Construction of the Fejer and Clenshaw-Curtis Quadrature Rules." *BIT Numerical Mathematics*, **46**(1), 195–202.

Williamson RC (1989). *Probabilistic Arithmetic*. Ph.D. thesis, Department of Electrical Engineering, University of Queensland.

Williamson RC, Downs T (1990). "Probabilistic Arithmetic. I. Numerical Methods for Calculating Convolutions and Dependency Bounds." *International Journal of Approximate Reasoning*, **4**(2), 89–158.

Wolfram S (2003). *The Mathematica Book*. 5th edition. Wolfram Media.

A. Implementation details

In the **PaCAL** project we use a purely numerical approach. Distributions of random variables are represented using their probability density functions (PDFs). Binary operations on random variables are implemented using numerical integration. The PDFs are represented as interpolated piecewise smooth functions. The results of the operations use the same representation, so they can be used during further computations. In the following sections we briefly describe the probabilistic context and sketch out the implementation details. More theoretical results and implementation information can be found in Jaroszewicz and Korzeń (2012).

A.1. Probabilistic background

Detailed expositions concerning arithmetic operations on random variables can be found in Springer (1979); Williamson (1989); Hoffmann-Jørgensen (1994a). Let f and g be probability density functions (PDFs) of i.random variables X and Y respectively. The corresponding cumulative distribution functions (CDFs) will be denoted by capital letters F, G. The densities of random variables $X+Y$, $X-Y$, $X \cdot Y$, X/Y are given by (Springer 1979):

$$\begin{aligned} PDF_{X+Y}(t) &= \int_{-\infty}^{+\infty} f(x)g(t-x)\,dx, \\ PDF_{X-Y}(t) &= \int_{-\infty}^{+\infty} f(x)g(x-t)\,dx, \\ PDF_{X\cdot Y}(t) &= \int_{-\infty}^{+\infty} f(x)g(t/x)\frac{1}{|x|}\,dx, \\ PDF_{X/Y}(t) &= \int_{-\infty}^{+\infty} f(xt)g(x)|x|\,dx. \end{aligned}$$

Note that the integration is performed along the path on which $x \diamond y = t$, where $\diamond$ is one of $+, -, \cdot, /$. Section A.4 contains an illustrative figure. If u is a function consisting of n strictly monotone and differentiable pieces u_i, the distribution of $u(X)$, where X is a random variable with density f, is given by (Springer 1979)

$$PDF_{u(X)}(t) = \sum_{i}^{n} f({u_i}^{-1}(t)) \left|({u_i}^{-1})'(t)\right|,$$

where, u' denotes the derivative of u.

The above formulas for arithmetic operations can be derived using change of variables (e.g., exchange X with the result) after which the remaining variable (Y in this case) is marginalized out through integration. This approach can also be applied to more complex operations and to dependent variables, see Hoffmann-Jørgensen (1994a, Chapter 8) or Springer (1979). The change of variables requires inverting the operation being performed and computing the Jacobian of the inverse. This is done by the `eval` method of the '`TwoVarsModel`' class. Symbolic computations are used to obtain the inverse and its Jacobian, and the elimination is done via numerical integration.

For operations on a large number of i.i.d. random variables one can minimize the computational cost using the algorithms typically applied to speed up the computation of integer powers (Knuth 1998, Chapter 4.6.3).

A.2. Minimum, maximum and order statistics

Let us keep the notation from the previous section. The cumulative distribution functions of the minimum and maximum of X and Y are given by:

$$\begin{aligned} CDF_{\max(X,Y)}(x) &= F(x)G(x), \\ CDF_{\min(X,Y)}(x) &= 1 - \big(1 - F(x)\big)\big(1 - G(x)\big), \end{aligned}$$

and (after differentiation) their probability density functions are:

$$\begin{aligned} PDF_{\max(X,Y)}(x) &= f(x) \int_{-\infty}^{x} g(x)\, dx + g(x) \int_{-\infty}^{x} f(x)\, dx, \\ PDF_{\min(X,Y)}(x) &= f(x) \int_{x}^{+\infty} g(x)\, dx + g(x) \int_{x}^{+\infty} f(x)\, dx. \end{aligned}$$

Those formulas can be generalized to arbitrary order statistics. Let $X_i, i = 1, \ldots, n$ be a set of i.i.d. random variables with PDF f and CDF F. Let $X_{(i)}$ be the ith order statistic of X_i's. Then we have (Hoffmann-Jørgensen 1994b, Chapter 2.24)

$$\begin{aligned} CDF_{X_{(i)}}(x) &= \sum_{k=i}^{n} \binom{n}{k} \big(F(x)\big)^k \big(1 - F(x)\big)^{n-k}, \\ PDF_{X_{(i)}}(x) &= \binom{n}{i-1, 1, n-i} \big(F(x)\big)^{i-1} \big(1 - F(x)\big)^{n-i} f(x), \end{aligned}$$

where the second equation is obtained by differentiating the first, and $\binom{n}{i-1,1,n-i}$ is the multinomial coefficient.

The joint density of the ith and jth order statistics is given by (Springer 1979, Chapter 9.7):

$$PDF_{X_{(i)},X_{(j)}}(x, y) = \begin{cases} \binom{n}{i-1,1,j-i-1,1,n-j} F(x)^{i-1} \big(F(y) - F(x)\big)^{j-i-1} (1 - F(y))^{n-j} f(x) f(y) & \text{if } x \leq y, \\ 0 & \text{otherwise.} \end{cases}$$

A.3. Copulas

Copulas are a way of defining joint distributions of several variables based on provided marginals. The joint CDF of variables X and Y is given by

$$CDF_{X,Y}(x, y) = C(F_X(x), F_Y(y)),$$

where F_X and F_Y are CDFs of X and Y respectively, and C is the copula function. A good introduction to copulas can be found in Nelsen (2006). In **PaCAL** we implement the product (independence) copula and three one-parameter families of Archimedean copulas defined as

$$C(u, v) = \varphi^{-1}(\varphi(u) + \varphi(v)),$$

where φ determines the family of the copula and is one of

Clayton $\varphi(t) = \frac{1}{\theta}(t^{-\theta} - 1)$, $\theta \in (0, \infty)$,

Gumbel $\varphi(t) = (-\log(t))^\theta$, $\theta \in (1, \infty)$,

Frank $\varphi(t) = -\log \frac{\exp(-\theta t)-1}{\exp(-\theta)-1}$, $\theta \in (-\infty, 0) \cup (0, \infty)$.

Clayton and Gumbel copulas only represent positive correlations, Frank copula can also represent negative correlations. The larger the value of θ, the more positively correlated X and Y are. Values close to zero correspond to independence. Additionally, for Frank copula, the closer θ is to $-\infty$, the stronger the negative correlation.

A.4. Numerical representation

We use piecewise continuous representation of density functions. The function is divided into smooth *segments*. Let $f(x) = \sum_{i=1}^{n} f_i(x)\chi_{[a_i,b_i]}(x)$, $g(x) = \sum_{j=1}^{m} g_j(x)\chi_{[c_j,d_j]}(x)$ be two piecewise smooth density functions of two independent random variables X and Y. f_i and g_j denote the smooth segments, and the characteristic functions χ select an appropriate segment. The density function of the sum $Z = X + Y$ can now be rewritten as

$$\begin{aligned} PDF_{X+Y}(t) &= \int_{-\infty}^{+\infty} f(x)g(t-x)\,dx \\ &= \int_{-\infty}^{+\infty} \sum_{i=1}^{n} f_i(x)\chi_{[a_i,b_i]}(x) \sum_{j=1}^{m} g_j(t-x)\chi_{[c_j,d_j]}(t-x)\,dx \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m} \int_{-\infty}^{+\infty} f_i(x)\chi_{[a_i,b_i]}(x)g_j(t-x)\chi_{[c_j,d_j]}(t-x)\,dx \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m} \int_{\max(a_i,t-d_j)}^{\min(b_i,t-c_j)} f_i(x)g_j(t-x)\,dx. \end{aligned}$$

The integrals are nonzero only if $\max(a_i, t-d_j) < \min(b_i, t-c_j)$ (equivalent to $a_i + c_j < t < b_i + d_j$) and ignored otherwise. The output function is a piecewise smooth function with segment endpoints at $a_i + b_j, c_i + d_j, i = 1, \ldots, n, j = 1, \ldots, m$.

Integrals for subtraction, multiplication, and division can be rewritten in the same way. The idea is illustrated in Figure 14 where the integration paths for two of the segments are shown for each of the four operations.

Each smooth segment of the density function is interpolated using the barycentric formula on Chebyshev nodes of first or second kind, see Berrut and Trefethen (2004) for details. To handle singularities and infinite intervals we use variable transformations, see Jaroszewicz and Korzeń (2012). Integrals are computed using Clenshaw-Curtis quadrature (an integration analogue of Chebyshev interpolation), see Waldvogel (2006); Trefethen (2008). Again, variable transformations are used for singularities and infinite intervals. We have used modified versions of the variable transforms suggested in Boyd (1987, 2001) and Davis and Rabinowitz (1984, Chapter 2.13, 3.1); see Jaroszewicz and Korzeń (2012) for details.

The class '`params`' contains various configurable parameters of the package. They allow for changing the maximum number of interpolation points, variable transforms used for infinite intervals and singularities, convergence criteria for interpolation and integration, etc. See the source code for details.

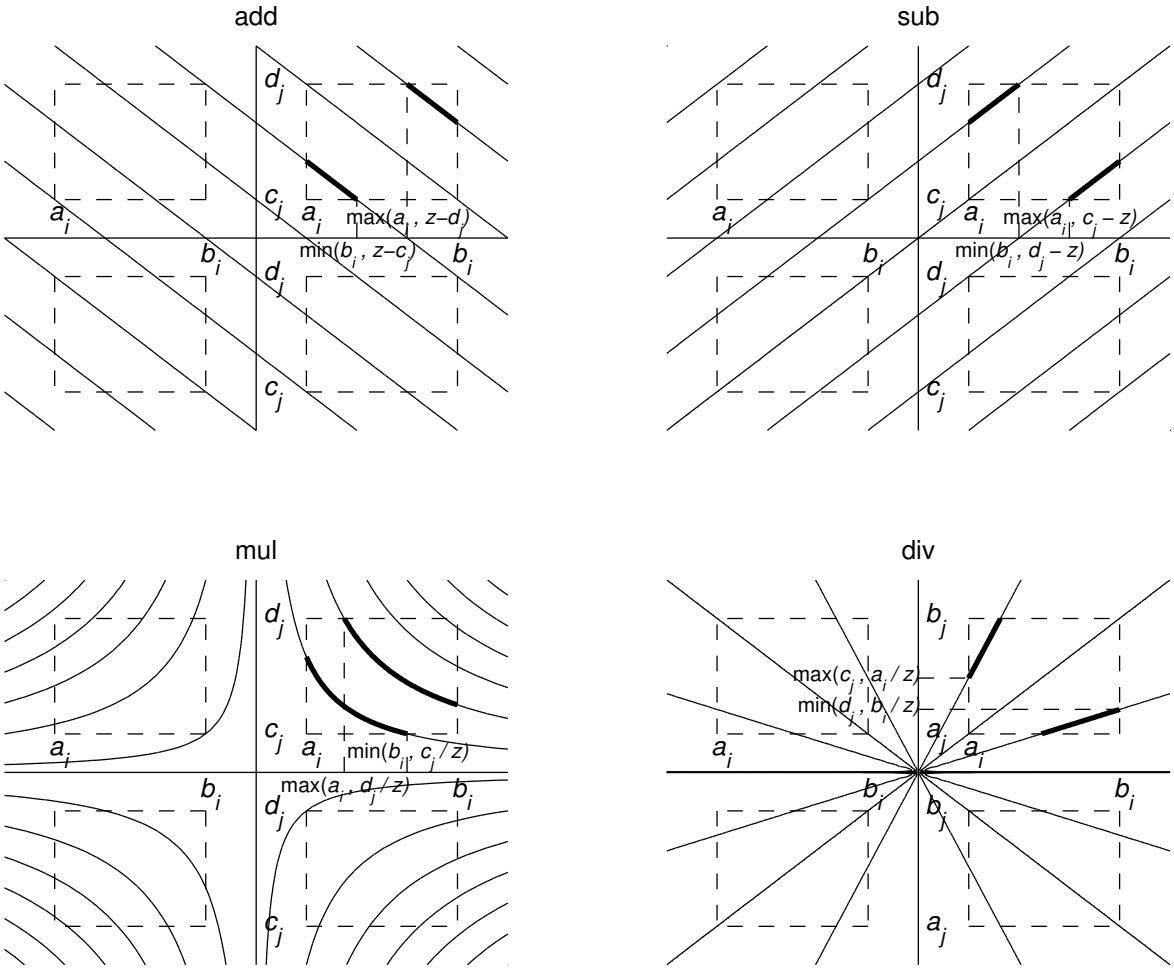

Figure 14: Integration paths used for computation of the four basic arithmetic operations on independent random variables: addition, subtraction, multiplication, and division. The figure illustrates, how the contours are truncated to the Cartesian product of two segments. Thin lines denote full integration paths, thick segments show parts of the path within a specific segment pair.

Affiliation:

Marcin Korzeń
Faculty of Computer Science and Information Technology
West Pomeranian University of Technology
ul. Żołnierska 49, 71–210, Szczecin, Poland
E-mail: mkorzen@wi.zut.edu.pl

Szymon Jaroszewicz
National Institute of Telecommunications
Warsaw, Poland
and
Institute of Computer Science
Polish Academy of Sciences
Warsaw, Poland
E-mail: s.jaroszewicz@ipipan.waw.pl

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 57, Issue 10 *Submitted:* 2012-02-14
April 2014 *Accepted:* 2013-07-21