Documentation to *ts* library

Piotr Z. Jelonek*

April 15, 2019

Abstract

Generation of pseudo-random numbers from a (Classical) Tempered Stable distribution. This library consists of a selection of the codes originally written for my PhD that I re-coded to Python. Out of the three featured randomisation methods, the mixture representation relies on earlier results by Baeumer and Meerschaert (2010), which utilise rejection sampling. The cdf inversion requires an approximation of a cdf using a pdf, which is computed via an inverse Fourier transform (specifically: an implementation of a Slow Fourier Transform which utilises FFT). The implementation of Devroye's algorithm requires an approximation of the integrals via (Gauss-Lobatto) quadratures and cubic spline forms.

*e-mail address: piotr.z.jelonek@gmail.com.

1 Global constants

A class *gc_ts()* defines the global parameters of a simulation as follows.

PARAMETERS

small	-	definition of a *'small'* number
large	-	definition of a *'large'* number
maxiter	-	maximal number of rejections in BM algorithm
maxorder	-	maximal order of a quadrature in Devroye algorithm
normalised	-	True if the standard deviation is supposed to be one, False otherwise
outpt	-	output (on/off)
N	-	a number of points for the pdf (FFT works best for powers of 2)

2 Random number generation

2.1 Mixture representation

rand_stab(alpha,beta,n):

Returns pseudo-random draws from a stable random variable $S_\alpha(\beta, 1, 0)$ with parameters alpha, beta, delta=1, and mu=0.

INPUT

alpha	-	constant, in (0,2) interval
beta	-	constant, in [-1,1] interval
n	-	integer, a required number of draws, default value is 1

RETURN

x	-	numpy.ndarray of generated draws, its shape: (n,1)

rand_ts_bm(alpha,theta,c,n):

Returns pseudo-random draws from a tempered stable random variable $TS_\alpha(1, 1, 0, \theta)$ with parameters alpha, beta=1, delta=1, mu=0, and theta. Reference: Baeumer and Meerschaert (2010).

INPUT

alpha	-	constant, in (0,2) interval
theta	-	constant, positive
c	-	constant, a cut-off value from the underlying stable distribution (here: 0.01% quantile)
n	-	integer, a required number of draws, default value is 1

RETURN

v - numpy.ndarray of generated draws, its shape: (n,1)

NOTE
The resulting random numbers are centred.

rand_ts_mixture(alpha,beta,delta,mu,theta,n):

Returns pseudo-random draws from a tempered stable random variable $TS_\alpha(\beta, \delta, \mu, \theta)$ via a mixture algorithm. Reference: Jelonek (2014), p. 4.

INPUT

alpha	-	constant, in (0,2) interval
beta	-	constant, in [-1.1]
delta	-	constant, positive
mu	-	constant, real
theta	-	constant, positive
n	-	integer, a required number of draws, default value is 1

RETURN

v - numpy.ndarray of generated draws, its shape: (n,1)

2.2 Cdf inversion (via inverse Fourier transform)

phi0(u):

Returns the values of a characteristic function of a tempered stable random variable $TS_\alpha(\beta, \delta, \mu, \theta)$, evaluated on vector u.

INPUT

u - real numpy.ndarray

RETURN

v - complex numpy.ndarray (with a size matching the input)

pdf(alpha,beta,delta,mu,theta):

Identifies the domain in which the pdf of a tempered stable random variable $TS_\alpha(\beta, \delta, \mu, \theta)$ exceeds the required value (gc.small). Next it evaluates its pdf (via inverse Fourier transform) at the (gc.N) points evenly spaced across this domain. References: Mittnik et al. (1999), Jelonek (2014), p. 11.

INPUT

alpha	-	constant, in (0,2) interval
beta	-	constant, in [-1.1]
delta	-	constant, positive
mu	-	constant, real
theta	-	constant, positive

RETURN

x	-	`numpy.ndarray` of points evenly spaced across the domain (with `gc.N` elements)
y	-	`numpy.ndarray` of the corresponding pdf values

cdf(alpha,beta,delta,mu,theta):

Calculates a cdf of a tempered stable random variable $TS_\alpha(\beta, \delta, \mu, \theta)$ using its (Fourier transformed) pdf. Next it corrects possible numerical errors.

INPUT

alpha	-	constant, in (0,2) interval
beta	-	constant, in [-1.1]
delta	-	constant, positive
mu	-	constant, real
theta	-	constant, positive

RETURN

x	-	`numpy.ndarray` of points evenly spaced across the domain (with `gc.N` elements)
z	-	`numpy.ndarray` of the corresponding cdf values

rand_ts_inv(alpha,beta,delta,mu,theta,n):

Returns pseudo-random draws from a tempered stable random variable $TS_\alpha(\beta, \delta, \mu, \theta)$ via a cdf inversion algorithm (requires an inverse Fourier transform to obtain the pdf). Reference: Jelonek (2014), p. 5.

INPUT

alpha	-	constant, in (0,2) interval
beta	-	constant, in [-1.1]
delta	-	constant, positive
mu	-	constant, real
theta	-	constant, positive
n	-	integer, a required number of draws, default value is 1

RETURN

v - `numpy.ndarray` of generated draws, its shape: (n,1)

2.3 Devroye (1981) algorithm

mod_phi2(u):

Returns modulus of a second order derivative of a characteristic function of a centred (mu=0) tempered stable random variable $TS_\alpha(\beta, \delta, 0, \theta)$.

INPUT

u - real `numpy.ndarray`

RETURN

abs(phi) - `numpy.ndarray` of absolute values of the cf (with a size matching the input)

getcform(x,y):

Outputs a cubic form interpolation of a function, values of which (y) are known only at given nodes given by (x). Reference: Burden and Faires (1997), p. 148.

INPUT

x - real `numpy.ndarray` (of size `gc.N`), arguments
y - real `numpy.ndarray` (of size `gc.N`), values of the function

RETURN

np.array([a,b,c,d]).T - real (`gc.N` -1 x 4) `numpy.ndarray`, parameters of the cubic form

rand_ts_devroye(alpha,beta,delta,mu,theta,n):

Returns pseudo-random draws from a tempered stable $TS_\alpha(\beta, \delta, \mu, \theta)$ random variable via an algorithm by Devroye (1981).

INPUT

alpha - constant, in (0,2) interval
beta - constant, in [-1.1]
delta - constant, positive
mu - constant, real
theta - constant, positive
n - integer, a required number of draws, default value is 1

RETURN

v	-	numpy.ndarray of generated draws, its shape: (n,1)

3 Estimation

estimate(v):

Estimates parameters of a tempered stable distribution from given sample.

INPUT

v	-	numpy.ndarray (with n elements) of i.i.d draws from a tempered stable distribution

RETURN

ahat	-	constant, estimate of alpha
bhat	-	constant, estimate of beta
dhat	-	constant, estimate of delta
mhat	-	constant, estimate of mu
that	-	constant, estimate of theta

WARNING
This procedure relies on sample moments of higher order. It requires a really large sample.

4 Additional procedures

params(alpha,beta,delta,mu,theta):

Verifies if the parameters of a tempered stable distribution have correct domains

INPUT

alpha	-	constant, real
beta	-	constant, real
delta	-	constant, real
mu	-	constant, real
theta	-	constant, real

RETURN

delta	-	constant, positive (if gc.normalised is True, this is the value of delta which yields a TS distribution with unit variance)
noerror	-	Boolean, equal to True if all the parameters lie in the correct range, equal to zero otherwise

moments_empiric(v):

Returns sample moments of independent tempered stable draws.

INPUT

v	-	numpy.ndarray (with n elements) of i.i.d draws from a tempered stable distribution

RETURN

m	-	numpy.ndarray of first 6 sample moments about the origin
k	-	numpy.ndarray, of first 6 sample cumulants
c	-	numpy.ndarray with, respectively, sample: mean, variance, skewness and excess kurtosis

moments_theoretic(alpha,beta,delta,mu,theta):

Returns theoretic moments of independent tempered stable draws. Reference: Jelonek (2014), p. 9-10.

INPUT

alpha	-	constant, in (0,2) interval
beta	-	constant, in [-1.1]
delta	-	constant, positive
mu	-	constant, real
theta	-	constant, positive

RETURN

m	-	numpy.ndarray of first 6 theoretic moments about the origin
k	-	numpy.ndarray, of first 6 theoretic cumulants
c	-	numpy.ndarray with, respectively, theoretic: mean, variance, skewness and excess kurtosis

gsf(mu,sigma,x):

This procedure returns the pdf of a normal distribution.

INPUT

mu	-	mean
sigma	-	standard deviation
x	-	real numpy.ndarray of values (n entries)

RETURN

y	-	numpy.ndarray of pdf values (with a size matching the input)

References

Baeumer, B., Meerschaert, M.M., 2010. Tempered stable Lévy motion and transient super–diffusion. Journal of Computational and Applied Mathematics 223, 2438–2448.

Burden, R.C., Faires, J.D., 1997. Numerical Analysis. Brooks/Cole Publishing Company, California. 6th edition edition.

Devroye, L., 1981. On the computer generation of random variables with a given characteristic function. International Journal of Computer Mathematics 7, 547–552.

Jelonek, P.Z., 2014. Generating tempered stable random variates from mixture representation. unpublished.

Mittnik, S., Doganoglu, T., Chenyao, D., 1999. Computing the probability density function of the stable Paretian distribution. Mathematical and Computer Modelling 29, 235–240.