

# Package ‘cctools’

March 24, 2025

**Type** Package

**Title** Tools for the Continuous Convolution Trick in Nonparametric Estimation

**Version** 0.1.3

**Description** Implements the uniform scaled beta distribution and the continuous convolution kernel density estimator.

**License** GPL-3

**Encoding** UTF-8

**Imports** stats, Rcpp (>= 0.12.5), qrng

**LinkingTo** Rcpp, RcppArmadillo

**RoxygenNote** 7.3.2

**Suggests** testthat

**NeedsCompilation** yes

**Author** Thomas Nagler [aut, cre]

**Maintainer** Thomas Nagler <mail@tnagler.com>

**Repository** CRAN

**Date/Publication** 2025-03-24 19:30:05 UTC

## Contents

cctools-package . . . . .	2
cckde . . . . .	2
cont_conv . . . . .	3
dusb . . . . .	4
expand_as_numeric . . . . .	5
expand_names . . . . .	6
expand_vec . . . . .	7

<b>Index</b>	<b>8</b>
--------------	----------

---

 cctools-package

*Tools for the continuous convolution trick in nonparametric estimation*


---

### Description

Implements the uniform scaled beta distribution `dusb()`, a generic function for continuous convolution `cont_conv()`, and the continuous convolution kernel density estimator `cckde()`.

### Author(s)

Thomas Nagler

### References

Nagler, T. (2017). *A generic approach to nonparametric function estimation with mixed data*. [arXiv:1704.07457](https://arxiv.org/abs/1704.07457)

---

 cckde

*Continuous convolution density estimator*


---

### Description

The continuous convolution kernel density estimator is defined as the classical kernel density estimator based on continuously convoluted data (see `cont_conv()`). `cckde()` fits the estimator (including bandwidth selection), `dcckde()` and `predict.cckde()` can be used to evaluate the estimator.

### Usage

```
cckde(x, bw = NULL, mult = 1, theta = 0, nu = 5, ...)
```

```
dcckde(x, object)
```

```
## S3 method for class 'cckde'
predict(object, newdata, ...)
```

### Arguments

<code>x</code>	a matrix or data frame containing the data (or evaluation points).
<code>bw</code>	vector of bandwidth parameter; if NULL, the bandwidths are selected automatically by likelihood cross validation.
<code>mult</code>	bandwidth multiplier; either a positive number or a vector of such. Each bandwidth parameter is multiplied with the corresponding multiplier.
<code>theta</code>	scale parameter of the USB distribution (see, <code>dusb()</code> ).

nu	smoothness parameter of the USB distribution (see, <code>dusb()</code> ). The estimator uses the Epanechnikov kernel for smoothing and the USB distribution for continuous convolution (default parameters correspond to the uniform distribution on $[-0.5, 0.5]$ ).
...	unused.
object	cckde object.
newdata	matrix or data frame containing evaluation points.

### Details

If a variable should be treated as ordered discrete, declare it as `ordered()`, factors are expanded into discrete dummy codings.

### References

Nagler, T. (2017). *A generic approach to nonparametric function estimation with mixed data*. [arXiv:1704.07457](https://arxiv.org/abs/1704.07457)

### Examples

```
# dummy data with discrete variables
dat <- data.frame(
  F1 = factor(rbinom(10, 4, 0.1), 0:4),
  Z1 = ordered(rbinom(10, 5, 0.5), 0:5),
  Z2 = ordered(rpois(10, 1), 0:10),
  X1 = rnorm(10),
  X2 = rexp(10)
)

fit <- cckde(dat) # fit estimator
dcckde(dat, fit) # evaluate density
predict(fit, dat) # equivalent
```

---

cont\_conv

*Continuous convolution*

---

### Description

Applies the continuous convolution trick, i.e. adding continuous noise to all discrete variables. If a variable should be treated as discrete, declare it as `ordered()` (passed to `expand_as_numeric()`).

### Usage

```
cont_conv(x, theta = 0, nu = 5, quasi = TRUE)
```

**Arguments**

x	data; numeric matrix or data frame.
theta	scale parameter of the USB distribution (see, <code>dusb()</code> ).
nu	smoothness parameter of the USB distribution (see, <code>dusb()</code> ). The estimator uses the Epanechnikov kernel for smoothing and the USB for continuous convolution (default parameters correspond to the $U[-0.5, 0.5]$ distribution).
quasi	logical indicating whether quasi random numbers should be used ( <code>qrng::ghalton()</code> ); only works for $\theta = 0$ .

**Details**

The UPSB distribution (`dusb()`) is used as the noise distribution. Discrete variables are assumed to be integer-valued.

**Value**

A data frame with noise added to each discrete variable (ordered columns).

**References**

Nagler, T. (2017). *A generic approach to nonparametric function estimation with mixed data*. [arXiv:1704.07457](https://arxiv.org/abs/1704.07457)

**Examples**

```
# dummy data with discrete variables
dat <- data.frame(
  F1 = factor(rbinom(10, 4, 0.1), 0:4),
  Z1 = ordered(rbinom(10, 5, 0.5), 0:5),
  Z2 = ordered(rpois(10, 1), 0:10),
  X1 = rnorm(10),
  X2 = rexp(10)
)

pairs(dat)
pairs(expand_as_numeric(dat)) # expanded variables without noise
pairs(cont_conv(dat))        # continuously convoluted data
```

---

dusb

*Uniform scaled beta distribution*


---

**Description**

The uniform scaled beta (USB) distribution describes the distribution of the random variable

$$U_{b,\nu} = U + \theta(B - 0.5),$$

where  $U$  is a  $U[-0.5, 0.5]$  random variable,  $B$  is a  $Beta(\nu, \nu)$  random variable, and  $\theta > 0, \nu \geq 1$ .

**Usage**

```
dusb(x, theta = 0, nu = 5)

rusb(n, theta = 0, nu = 5, quasi = FALSE)
```

**Arguments**

x	vector of quantiles.
theta	scale parameter of the USB distribution.
nu	smoothness parameter of the USB distribution.
n	number of observations.
quasi	logical indicating whether quasi random numbers ( <code>qrng::ghalton()</code> ) should be used for generating uniforms (which are then transformed by the quantile function)

**References**

Nagler, T. (2017). *A generic approach to nonparametric function estimation with mixed data*. [arXiv:1704.07457](https://arxiv.org/abs/1704.07457)

**Examples**

```
# plot distribution
sq <- seq(-0.8, 0.8, by = 0.01)
plot(sq, dusb(sq), type = "l")
lines(sq, dusb(sq, theta = 0.25), col = 2)
lines(sq, dusb(sq, theta = 0.25, nu = 10), col = 3)

# simulate from the distribution
x <- rusb(100, theta = 0.3, nu = 0)
```

---

expand\_as\_numeric      *Numeric model matrix for continuous convolution*

---

**Description**

Turns ordered variables into integers and expands factors as binary dummy codes. `cont_conv()` additionally adds noise to discrete variables, but this is only useful for estimation. `[cc_prepare()]` can be used to evaluate an already fitted estimate.

**Usage**

```
expand_as_numeric(x)
```

**Arguments**

x	a vector or data frame with numeric, ordered, or factor columns.
---	--

**Value**

A numeric matrix containing the expanded variables. It has additional type `expanded_as_numeric` and `attr("i_disc")` contains the indices of discrete variables.

**Examples**

```
# dummy data with discrete variables
dat <- data.frame(
  F1 = factor(rbinom(100, 4, 0.1), 0:4),
  Z1 = as.ordered(rbinom(100, 5, 0.5)),
  Z2 = as.ordered(rpois(100, 1)),
  X1 = rnorm(100),
  X2 = rexp(100)
)

pairs(dat)
pairs(expand_as_numeric(dat)) # expanded variables without noise
pairs(cont_conv(dat))        # continuously convoluted data
```

---

expand_names	<i>Expands names for expand_as_numeric</i>
--------------	--

---

**Description**

Expands each element according to the factor expansions of columns in `expand_as_numeric()`.

**Usage**

```
expand_names(x)
```

**Arguments**

`x` as in `expand_as_numeric()`.

**Value**

A vector of size `ncol(expand_as_numeric(x))`.

---

`expand_vec`*Expand a vector like `expand_as_numeric`*

---

**Description**

Expands each element according to the factor expansions of columns in `expand_as_numeric()`.

**Usage**

```
expand_vec(y, x)
```

**Arguments**

`y` a vector of length 1 or `ncol(x)`.  
`x` as in `expand_as_numeric()`.

**Value**

A vector of size `ncol(expand_as_numeric(x))`.

# Index

## \* package

cctools-package, 2

cckde, 2

cckde(), 2

cctools (cctools-package), 2

cctools-package, 2

cont\_conv, 3

cont\_conv(), 2, 5

dcckde (cckde), 2

dcckde(), 2

dusb, 4

dusb(), 2–4

expand\_as\_numeric, 5

expand\_as\_numeric(), 3, 6, 7

expand\_names, 6

expand\_vec, 7

ordered(), 3

predict.cckde (cckde), 2

predict.cckde(), 2

qrng::ghalton(), 4, 5

rusb (dusb), 4