# Package 'FKSUM'

April 29, 2020

**Type** Package

**Title** Fast Kernel Sums

**Version** 0.1.4

**Author** David P. Hofmeyr

**Maintainer** David P. Hofmeyr <dhofmeyr@sun.ac.za>

**Description**

Implements the method of Hofmeyr, D.P. (2019) <DOI:10.1109/TPAMI.2019.2930501> for fast evaluation of univariate kernel smoothers based on recursive computations.
Applications to the basic problems of density and regression function estimation are provided, as well as some projection pursuit methods
for which the objective is based on non-parametric functionals of the projected density, or conditional density of a response given projected
covariates.

**License** GPL

**Encoding** UTF-8

**Depends** Rcpp (>= 0.12.16)

**Imports** rARPACK, MASS

**LinkingTo** Rcpp, RcppArmadillo

**LazyData** true

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2020-04-29 06:30:02 UTC

## R topics documented:

---

`FKSUM-package` *Fast Exact Kernel Smoothing*

---

## Description

Uses recursive expressions to compute exact univariate kernel smoothers in log-linear time based on the method described by Hofmeyr (2019). The main general purpose function is fk_sum() which computes exact (or binned approximations of) weighted sums of kernels, or their derivatives. Standard smoothing problems such as density estimation and regression can be addressed directly using this function, or using the purpose-built functions fk_density() and fk_regression. Projection pursuit algorithms based on minimum entropy (ICA, Hyvarinen and Oja 2000), minimum density cluster separation (MDH, Pavlidis et al. 2016) and regression-type losses (PPR, Friedman and Stuetzle 1981) are implemented in the functions fk_ICA(), fk_ppr() and fk_mdh() respectively.

## Details

Package: FKSUM

Type: Package

Title: Fast Kernel Sums

Version: 0.1.3

Depends: Rcpp (>= 0.12.16)

License: GPL-3

LazyData: yes

Imports: rARPACK, MASS

LinkingTo: Rcpp, RcppArmadillo

## Author(s)

David Hofmeyr[aut, cre]

Maintainer: David P. Hofmeyr <dhofmeyr@sun.ac.za>

## References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

Hyvarinen, A., and Oja, E. (2000) "Independent component analysis: algorithms and applications." *Neural networks*, 13(4), 411–430.

Friedman, J.H., and Stuetzle, W. (1981) "Projection pursuit regression." *Journal of the American statistical Association*, 76(376), 817–823.

Pavlidis, N.G., Hofmeyr, D.P., and Tasoulis, S.K. (2016) "Minimum density hyperplanes." *Journal of Machine Learning Research*, 17(156), 1–33.

---

`fk_density` *Fast univariate kernel density estimation*

---

## Description

Uses recursive formulation for kernel sums as described in Hofmeyr (2019) to evaluate kernel estimate of the density exactly. Binning approximation also available for faster computation if needed. Default is exact evaluation on a grid, but evaluation at an arbitrary collection of points is possible.

## Usage

```
fk_density(x, h = 'Silverman', h_adjust = 1, beta = NULL, from = NULL,
              to = NULL, ngrid = 1000, nbin = NULL, x_eval = NULL)
```

## Arguments

| | |
|---|---|
| `x` | vector of sample points. |
| `h` | (optional) bandwidth to be used in estimate. Can be either positive numeric, or one of "Silverman" for Silverman's rule of thumb (Silverman, 1986) or "mlcv" for maximum pseudo-likelihood cross validation bandwidth. Default is Silverman's heuristic. Exact "mlcv" will be time consuming for samples of more than millions of points. Binning approximation for "mlcv" bandwidth needs at least 10000 bins for reasonable accuracy, and even more if density has very sharp features. |
| `h_adjust` | (optional) positive numeric. Final bandwidth will be h*h_adjust. Default value is 1. R's base function density uses Silverman's heuristic with h_adjust approximately 0.85. |

| | |
|---|---|
| beta | (optional) numeric vector of kernel coefficients. See Hofmeyr (2019) for details. The default is the smooth order one kernel described in the paper. |
| from | (optional) lower end of evaluation interval if evaluation on a grid is desired. Default is min(x)-6*h |
| to | (optional) upper end of evaluation interval if evaluation on a grid is desired. Default is max(x)+6*h |
| ngrid | (optional) integer number of grid points for evaluation. Default is 1000. |
| nbin | (optional) integer number of bins if binning estimator is to be used. The default is to compute the exact density on a grid of 1000 points. |
| x_eval | (optional) vector of evaluation points. The default if both ngrid and nbin are set to NULL is evaluation at the sample points themselves. If another specific set of points is required then ngrid must be set to null and x_eval supplied. Evaluation at arbitrary x_eval using binned approximation is also possible, in which case nbin and x_eval must both be supplied. |

## Value

A named list with fields

| | |
|---|---|
| $x | the vector of points at which the density is estimated. |
| $y | the estimated density values. |
| $h | the value of the bandwidth. |

## References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

Silverman, B. (1986) *Density estimation for statistics and data analysis*, volume 26. CRC press.

## Examples

```
op <- par(no.readonly = TRUE)

set.seed(1)

### generate a bimodal Gaussian mixture sample with 100000 points

n1 <- rbinom(1, 100000, .5)

x <- c(rnorm(n1), rnorm(100000-n1)/4+2)



# --------- Example 1: Grid evaluation --------------#
### evaluate exact and binned approximation on a
### grid and plot along with true density. We can
### use both Silverman's heuristic and the "mlcv" estimate.
```

```
xs <- seq(-5, 3.5, length = 1000)

ftrue <- dnorm(xs)/2 + dnorm(xs, 2, 1/4)/2

fhat <- fk_density(x, from = -5, to = 3.5)

fhat_bin <- fk_density(x, nbin = 1000, from = -5, to = 3.5)

par(mfrow = c(2, 2))
plot(xs, ftrue, type = 'l', lwd = 4, col = rgb(.7, .7, .7),
    xlab = 'x', ylab = 'f(x)',
    main = 'Exact evaluation, Silverman bandwidth')
lines(fhat, lty = 2)

plot(xs, ftrue, type = 'l', lwd = 4, col = rgb(.7, .7, .7),
    xlab = 'x', ylab = 'f(x)',
    main = 'Binned approximation, Silverman bandwidth')
lines(fhat_bin, lty = 2)




fhat <- fk_density(x, from = -5, to = 3.5, h = 'mlcv')

fhat_bin <- fk_density(x, nbin = 1000, from = -5,
    to = 3.5, h = 'mlcv')

plot(xs, ftrue, type = 'l', lwd = 4, col = rgb(.7, .7, .7),
    xlab = 'x', ylab = 'f(x)',
    main = 'Exact evaluation, MLCV bandwidth')
lines(fhat, lty = 2)

plot(xs, ftrue, type = 'l', lwd = 4, col = rgb(.7, .7, .7),
    xlab = 'x', ylab = 'f(x)',
    main = 'Binned approximation, MLCV bandwidth')
lines(fhat_bin, lty = 2)


par(op)


# --------- Example 2: Evaluation at sample --------------#
### evaluate exact and binned approximation at the sample.
### Note that the output will be in the order of the original
### sample, and also the number of points will be large. It is
### not advisable, therefore, to simply plot these. We instead
### compute the mean squared deviations from the true density

ftrue <- sapply(x, function(xi) dnorm(xi)/2 + dnorm(xi, 2, 1/4)/2)

fhat <- fk_density(x, ngrid = NULL)

fhat_bin <- fk_density(x, nbin = 1000, x_eval = x)
```

```
mean((ftrue-fhat$y)^2)

mean((ftrue-fhat_bin$y)^2)


### now for MLCV bandwidth

fhat <- fk_density(x, h = 'mlcv', ngrid = NULL)

fhat_bin <- fk_density(x, nbin = 1000,
    h = 'mlcv', x_eval = x)


mean((ftrue-fhat$y)^2)

mean((ftrue-fhat_bin$y)^2)
```

---

`fk_ICA` *Independent component analysis with sample entropy estimated via kernel density*

---

## Description

Performs minimum entropy projection pursuit

## Usage

```
fk_ICA(X, ncomp = 1, beta = c(.25, .25), hmult = 1.5, it = 20, nbin = NULL)
```

## Arguments

| | |
|---|---|
| `X` | numeric data matrix (num_data x num_dimensions). |
| `ncomp` | integer number of indpenedent components to extract. |
| `beta` | numeric vector of kernel coefficients. See Hofmeyr (2019) for details. The default is the smooth order one kernel described in the paper. |
| `hmult` | positive numeric. The bandwidth in the kernel density is set to hmult multiplied by Silverman's rule of thumb value, which is based on the AMISE minimiser when the underlying distribution is Gaussian. |
| `it` | integer maximum number of iterations. The default is 20. |
| `nbin` | integer number of bins if binning estimator is to be used. The default is to compute the exact entropy estimate from the kde. |

## Value

A named list with fields

| | |
|---|---|
| $X | the data matrix given as argument. |
| $K | the pre-whitening matrix. |
| $W | the estimated un-mixing matrix. |
| $S | the estimated source matrix, S = sweep(X,2,colMeans(X),'-')%*%K%*%W. |

## References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

## Examples

```
op <- par(no.readonly = TRUE)

set.seed(1)

### Generate a set of data with 5 non-Gaussian components
### and 20 noise (Gaussian) components. The non-Gaussian
### components are cubed Gaussian, exponential, uniform,
### t3 and bimodal Gaussian mixture.

errdim = 10
ndat = 2000
X = cbind(rnorm(ndat)^3/5, rexp(ndat), runif(ndat)*sqrt(12)*2, rt(ndat, 3),
    c(rnorm(ndat/2), rexp(ndat/2)+2), matrix(rnorm(errdim*ndat), ncol = errdim))

### Generate a random mixing matrix and mixed data XX

R = matrix(rnorm((5+errdim)*(5+errdim)), ncol = 5+errdim)
XX = X%*%R

### Apply fk_ICA to XX to extract components.
### Note that ordering of extracted components is
### in some sense arbitrary w.r.t. their generation

model <- fk_ICA(XX, ncol(XX))

par(mfrow = c(1, 5))

for(i in 1:5) plot(density(model$S[,i]))

par(op)
```

fk_mdh *Minimum density hyperplanes*

## Description

Estimates minimum density hyperplanes for clustering using projection pursuit, based on the method of Pavlidis et al. (2016)

## Usage

```
fk_mdh(X, v0 = NULL, hmult = 1, beta = c(.25,.25), alphamax = 1)
```

## Arguments

| | |
|---|---|
| X | numeric data matrix (num_data x num_dimensions). |
| v0 | (optional) vector representing the initial projection direction. Default is the first principal direction. |
| hmult | (optional) positive numeric. The bandwidth in the kernel density is set to hmult multiplied by Silverman's rule of thumb value, which is based on the AMISE minimiser when the underlying distribution is Gaussian. |
| beta | (optional) numeric vector of kernel coefficients. The default is the smooth order one kernel described by Hofmeyr (2019). |
| alphamax | (optional) maximum/final (scaled) distance of the optimal hyperplane from the mean of the data. The default is alphamax = 1. |

## Value

A named list with fields

| | |
|---|---|
| $v | the optimal projection vector. |
| $b | the location of the minimum density hyperplane orthogonal to v. |

## References

Pavlidis N.G., Hofmeyr D.P., Tasoulis S.K. (2016) "Minimum Density Hyperplanes", *Journal of Machine Learning Research*, 17(156), 1–33.

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

## Examples

```
op <- par(no.readonly = TRUE)

set.seed(1)

### Generate data from a simple 10 component mixture model in 10 dimensions:
```

```
### Determine means of components

mu <- matrix(runif(100), ncol = 10)

### Determine scales of components (diagonal elements of covariance)

covs <- matrix(rexp(100), ncol = 10)/10

### Determine cluster indicator matrix

I <- t(rmultinom(2000, 1, 1:10))

### Determine mean and residual matrix

M <- I%*%mu

R <- matrix(rnorm(20000), 2000, 10)*(I%*%covs)

### Data is given by the sum of these

X <- M + R

### Find the minimum density hyperplane separator and plot
### the projected data as well as the PCA plot for comparison

mdh <- fk_mdh(X)

par(mfrow = c(2, 2))

plot(X%*%mdh$v, X%*%eigen(cov(X))$vectors[,2], xlab = 'mdh vector',
    ylab = '2nd principal component', main = 'MDH solution')
abline(v = mdh$b, col = 2)
plot(X%*%eigen(cov(X))$vectors[,1:2], xlab = '1st principal component',
    ylab = '2nd principal component', main = 'PCA plot')

plot(fk_density(X%*%mdh$v))
abline(v = mdh$b, col = 2)
plot(fk_density(X%*%eigen(cov(X))$vectors[,1]))

par(op)
```

---

fk_ppr *Projection pursuit regression with local linear kernel smoother*

---

## Description

Generates a projection pursuit regression model for covariates X and response y

## Usage

```
fk_ppr(X, y, nterms=1, hmult=1, betas=NULL, loss=NULL,
    dloss=NULL, initialisation="lm", type = "loc-lin")
```

## Arguments

| | |
|---|---|
| X | a numeric matrix (num_data x num_dimensions) of covariates. |
| y | a numeric vector of responses. |
| nterms | The number of terms in the additive regression model. The default is a single term. |
| betas | The coefficients in the expression of the kernel. See Hofmeyr (2019) for details. The default is the smooth order one kernel described in the paper. |
| hmult | The bandwidth in the kernel smoother is set to eigen(cov(X))values[1]^.5/nrow(X)^.2*hmult during projection pursuit. The final value is based on leave-one-out cross validation on the optimal projection. |
| loss | The (additive) loss function to be used. Allows for generalised responses by specifying an appropriate likelihood/deviance function for the loss. Note: loss is to be minimised, so deviance or negative log-likelihood would be appropriate. If specified then must be a function of y and hy (the fitted values, yhat), returning a vector of "errors"" which are added as the total loss. The default is the squared error. |
| dloss | The derivative of the loss function. Also takes arguments y and hy, and returns the vector of partial derivatives of the loss w.r.t. hy. |
| initialisation | Method use to initialise the projection vectors. Must be one of "lm" and "random", or a function taking only arguments X and y, and returning a vector of length ncol(X). The default is "lm", which is a simple linear model with a small ridge to ensure a solution. "random" uses random initialisation. |
| type | The type of regression estimator. Must be one of "loc-lin" for local linear regression, or "NW" for the Nadaray Watson, or local constant regression. The default is local linear regression. |

## Value

A named list with class fk_ppr containing

| | |
|---|---|
| $mu | The estimated (global) mean of the response. |
| $mu_X | The vector of estimated means of the covariates. |
| $y | The responses given as argument to the function. |
| $X | The covariates given as argument to the function. |
| $hs | A vector of bandwidths used for each term in the model. |
| $vs | A matrix whose rows are the projection vectors. |
| $fitted | The fitted values on each projection. Note that these are based on the residuals used to fit that component of the model, and not the original y values. $fitted is used for prediction on test data. |
| $beta | The beta coefficients in the kernel formulation. |

## References

Friedman, J., and Stuetzle, W. (1981) "Projection pursuit regression." *Journal of the American statistical Association* **76**.376.

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

## Examples

```
op <- par(no.readonly = TRUE)

set.seed(2)

### Generate a set of data

X = matrix(rnorm(10000), ncol = 10)

### Generate some "true" projection vectors

beta1 = (runif(10)>.5)*rnorm(10)
beta2 = (runif(10)>.5)*rnorm(10)

### Generate responses, dependent on X%*%beta1 and X%*%beta2

y = 1 + X%*%beta1 + ((X%*%beta2)>2)*(X%*%beta2-2)*10
y = y + (X%*%beta1)*(X%*%beta2)/5 + rnorm(1000)

### Fit a PPR model with 2 terms on a sample of the data

train_ids = sample(1:1000, 500)

model = fk_ppr(X[train_ids,], y[train_ids], nterms = 2)

### Predict on left out data, and compute
### estimated coefficient of determination

yhat = predict(model, X[-train_ids,])

MSE = mean((yhat-y[-train_ids])^2)
Var = mean((y[-train_ids]-mean(y[-train_ids]))^2)

1-MSE/Var


################################################

### Add some "outliers" in the training data and fit
### the model again, as well as one with an absolute loss

y[train_ids] = y[train_ids] + (runif(500)<.05)*(rnorm(500)*100)

model1 <- fk_ppr(X[train_ids,], y[train_ids], nterms = 2)
```

```
model2 <- fk_ppr(X[train_ids,], y[train_ids], nterms = 2,
    loss = function(y, hy) abs(y-hy),
    dloss = function(y, hy) sign(hy-y))

### Plot the resulting components in the model on the test data

par(mar = c(2, 2, 2, 2))
par(mfrow = c(2, 2))

plot(X[-train_ids,]%*%model1$vs[1,], y[-train_ids])
plot(X[-train_ids,]%*%model1$vs[2,], y[-train_ids])
plot(X[-train_ids,]%*%model2$vs[1,], y[-train_ids])
plot(X[-train_ids,]%*%model2$vs[2,], y[-train_ids])

par(op)

### Estimate comparative estimated coefficients of determination

MSE1 = mean((predict(model1, X[-train_ids,])-y[-train_ids])^2)
MSE2 = mean((predict(model2, X[-train_ids,])-y[-train_ids])^2)
Var = mean((y[-train_ids]-mean(y[-train_ids]))^2)


1-MSE1/Var
1-MSE2/Var
```

---

`fk_regression` *Fast univariate kernel regression*

---

## Description

Uses recursive formulation for kernel sums as described in Hofmeyr (2019) for exact evaluation of kernel regression estimate. Both local linear and Nadaraya-Watson (Watson, 1964; Nadaraya, 1964) estimators are available. Binning approximation also available for faster computation if needed. Default is exact evaluation on a grid, but evaluation at sample points is also possible.

## Usage

```
fk_regression(x, y, h = 'amise', beta = NULL, from = NULL,
    to = NULL, ngrid = 1000, nbin = NULL, type = 'loc-lin')
```

## Arguments

| | |
|---|---|
| x | vector of covariates. |
| y | vector of response values. |
| h | (optional) bandwidth to be used in estimate. Can be either positive numeric, or one of "amise" for a rough estimate of the asymptotic mean integrated square error minimiser, or "cv" for leave-one-out cross validation error minimiser based |

| | |
|---|---|
| | on squared error. Default is "amise". Cross validation not available for binning approximation. |
| beta | (optional) numeric vector of kernel coefficients. See Hofmeyr (2019) for details. The default is the smooth order one kernel described in the paper. |
| from | (optional) lower end of evaluation interval if evaluation on a grid is desired. Default is min(x) |
| to | (optional) upper end of evaluation interval if evaluation on a grid is desired. Default is max(x) |
| ngrid | (optional) integer number of grid points for evaluation. Default is 1000. |
| nbin | (optional) integer number of bins if binning estimator is to be used. The default is to compute the exact density on a grid of 1000 points. |
| type | (optional) one of "loc-lin" and "NW" if local linear or Nadaraya-Watson is desired, respectively. Default is local linear estimator. |

## Value

A named list with fields

| | |
|---|---|
| $x | the vector of points at which the regression function is estimated. |
| $y | the estimated function values. |
| $h | the value of the bandwidth. |

## References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

Nadaraya, E.A. (1964) "On estimating regression." *Theory of Probability \& Its Applications*,9(1), 141–142.

Watson, G.S. (1964) "Smooth regression analysis." *Sankhya: The Indian Journal of Statistics*, Series A, pp. 359–372.

## Examples

```
op <- par(no.readonly = TRUE)

set.seed(1)

n <- 2000
x <- rbeta(n, 2, 2) * 10
y <- 3 * sin(2 * x) + 10 * (x > 5) * (x - 5)
y <- y + rnorm(n) + (rgamma(n, 2, 2) - 1) * (abs(x - 5) + 3)

xs <- seq(min(x), max(x), length = 1000)
ftrue <- 3 * sin(2 * xs) + 10 * (xs > 5) * (xs - 5)

fhat_loc_lin <- fk_regression(x, y)
fhat_NW <- fk_regression(x, y, type = 'NW')
```

```
par(mfrow = c(2, 2))
plot(fhat_loc_lin, main = 'Local linear estimator with amise bandwidth')

plot(fhat_NW, main = 'NW estimator with amise bandwidth')


fhat_loc_lin <- fk_regression(x, y, h = 'cv')
fhat_NW <- fk_regression(x, y, type = 'NW', h = 'cv')

plot(fhat_loc_lin, main = 'Local linear estimator with cv bandwidth')

plot(fhat_NW, main = 'NW estimator with cv bandwidth')

par(op)
```

---

fk_sum *Fast Exact Kernel Sum Evaluation*

---

## Description

Computes exact (and binned approximations of) kernel and kernel derivative sums with arbitrary weights/coefficients. Computation is based on the method of Hofmeyr (2019).

## Usage

```
fk_sum(x, omega, h, x_eval = NULL, beta = c(.25,.25),
    nbin = NULL, type = "ksum")
```

## Arguments

| | |
|---|---|
| x | numeric vector of sample points. |
| omega | numeric vector of weights. |
| h | numeric bandwidth (must be strictly positive). |
| x_eval | vector of evaluation points. Default is to evaluate at the sample points themselves. |
| beta | numeric vector of kernel coefficients. Default is c(.25, .25); the smooth order 1 kernel. |
| nbin | integer number of bins for binning approximation. Default is to compute the exact sums. |
| type | one of "ksum": returns the kernel sums, "dksum": returns the kernel derivative sums and "both": returns a matrix cbind(ksum, dksum). |

## Value

A vector (if type%in%c("ksum","dksum")) of kernel sums, or kernel derivative sums. A matrix (if type == "both") with kernel sums in its first column and kernel derivative sums in its second column.

## References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

## Examples

```
### Compute density estimates directly with
### kernel sums and constant normalising weights

set.seed(1)
n <- 150000
num_Gauss <- rbinom(1, n, 2 / 3)
x <- c(rnorm(num_Gauss), rexp(n - num_Gauss) + 1)

hs <- seq(.025, .1, length = 5)
xeval <- seq(-4, 8, length = 1000)
ftrue <- 2 / 3 * dnorm(xeval) + 1 / 3 * dexp(xeval - 1)
plot(xeval, ftrue, lwd = 6, col = rgb(.8, .8, .8), xlab = "x",
  ylab = "f(x)", type = "l")

for(i in 1:5) lines(xeval, fk_sum(x, rep(1 / hs[i] / n, n), hs[i],
    x_eval = xeval), lty = i)
```

---

# h_Gauss_to_K *Bandwidth conversion from Gaussian*

---

## Description

A naive and simple, but useful way of transforming a bandwidth for use with the Gaussian kernel to an appropriate value to be used with a kernel implemented in the FKSUM package. The transformation if based on the relative values for the kernels for the AMISE minimal bandwidth for the purpose of density estimation. The transformation has been useful for other problems as well.

## Usage

```
h_Gauss_to_K(h, beta)
```

## Arguments

| | |
|---|---|
| h | positive numeric bandwidth value appropriate for use with the Gaussian kernel. |
| beta | numeric vector of kernel coefficients. |

## Value

positive numeric value, the bandwidth to be used for estimation using the kernel with coefficients beta.

## References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

## Examples

```
### Use existing bandwidth selection with the package

### generate sample from bimodal Gaussian mixture with varying scale

x <- c(rnorm(100000), rnorm(100000)/4 + 2)

### estimate bandwidth using bw.SJ

bwsj <- h_Gauss_to_K(bw.SJ(x), c(.25,.25))

xs <- seq(-3, 4, length = 1000)
plot(xs, dnorm(xs)/2+dnorm(xs,2,1/4)/2, type = 'l',
    col = rgb(.7, .7, .7), lwd = 4)

lines(fk_density(x, h = bwsj), lty = 2, lwd = 2)
```

---

# h_K_to_Gauss *Bandwidth conversion to Gaussian*

---

## Description

A naive and simple, but useful way of transforming a bandwidth for use with a kernel implemented in FKSUM to an appropriate value to be used with the Gaussian kernel. The transformation if based on the relative values for the kernels for the AMISE minimal bandwidth for the purpose of density estimation. The transformation has been useful for other problems as well.

## Usage

```
h_K_to_Gauss(h, beta)
```

## Arguments

| | |
|---|---|
| h | positive numeric bandwidth value appropriate for use with the FKSUM kernel with coefficients beta to an appropriate value for use with the Gaussian kernel. |
| beta | numeric vector of kernel coefficients. |

## Value

positive numeric value, the bandwidth to be used for estimation using the Gaussian kernel.

## References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

## Examples

```
### Use the package for data driven bandwidth for use with
### the Gaussian kernel in existing implementations

### generate sample from bimodal Gaussian mixture with varying scale

x <- c(rnorm(100000), rnorm(100000)/4 + 2)

### estimate bandwidth with the package using MLCV and convert

bwml <- h_K_to_Gauss(fk_density(x, h = 'mlcv',
    beta = c(.25,.25))$h, c(.25,.25))

bwml_binned <- h_K_to_Gauss(fk_density(x, h = 'mlcv',
    beta = c(.25,.25), nbin = 10000)$h, c(.25,.25))

xs <- seq(-3, 4, length = 1000)
plot(xs, dnorm(xs)/2+dnorm(xs,2,1/4)/2, type = 'l',
    lwd = 4, col = rgb(.7, .7, .7))

lines(density(x, bw = bwml), lty = 2, lwd = 2)

lines(density(x, bw = bwml_binned), lty = 3, lwd = 2)
```

---

# norm_const_K *Normalising constant for kernels in FKSUM*

---

## Description

Computes the normalising constant for kernels implemented in the FKSUM package, to convert kernel with arbitrary beta coefficients to one which is a probability density. That is the kernel with coefficients equal to beta/norm_const_beta(beta) has unit integral

## Usage

```
norm_const_K(beta)
```

## Arguments

| | |
|---|---|
| beta | numeric vector of positive coefficients. |

## Value

positive numeric normalising constant

## References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

---

`norm_K` *The L2 norm of a kernel*

---

## Description

Computes the L2 norm of a kernel implemented in FKSUM, based on its coefficients. NB: the coefficients will first be normalised so that the kernel represents a density function. Equivalent to sqrt(roughness_K())

## Usage

```
norm_K(beta)
```

## Arguments

| | |
|---|---|
| `beta` | numeric vector of positive coefficients. |

## Value

positive numeric representing the L2 norm of the kernel with coefficients beta/norm_const(beta).

## References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

---

`plot.fk_density` *Plot method for class fk_density*

---

## Description

Plot method for kernel density estimate obtained with function fk_density()

## Usage

```
## S3 method for class 'fk_density'
plot(x, main = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | an object of class fk_density, output from the function of the same name. |
| `main` | (optional) main title for plot. |
| `...` | (optional) further plotting parameters. |

## plot.fk_ICA *Plot method for class fk_ICA*

### Description

Plot method for Independent Component Analysis model obtained with function fk_ICA()

### Usage

```
## S3 method for class 'fk_ICA'
plot(x, ...)
```

### Arguments

| | |
|---|---|
| x | an object of class fk_ICA, output from the function of the same name. |
| ... | (optional) further plotting parameters. |

## plot.fk_mdh *Plot method for class fk_mdh*

### Description

Plot method for minimum density hyperplane solution obtained with function fk_mdh()

### Usage

```
## S3 method for class 'fk_mdh'
plot(x, ...)
```

### Arguments

| | |
|---|---|
| x | an object of class fk_mdh, output from the function of the same name. |
| ... | (optional) further plotting parameters. |

## plot.fk_ppr *Plot method for class fk_ppr*

### Description

Plot method for projection pursuit regression model fit with function fk_ppr()

### Usage

```
## S3 method for class 'fk_ppr'
plot(x, term = NULL, ...)
```

### Arguments

| | |
|---|---|
| x | an object of class fk_ppr, output from the function of the same name. |
| term | (optional) the term in the additive regression model to be visualised. If omitted then the residuals and fitted values for the full model are shown. |
| ... | (optional) further arguments passed to or from other methods. |

## plot.fk_regression *Plot method for class fk_regression*

### Description

Plot method for kernel regression estimate obtained with function fk_regression()

### Usage

```
## S3 method for class 'fk_regression'
plot(x, main = NULL, ...)
```

### Arguments

| | |
|---|---|
| x | an object of class fk_regression, output from the function of the same name. |
| main | (optional) main title for plot. |
| ... | (optional) further plotting parameters. |

## plot_kernel — *Plot the shape of a kernel function implemented in FKSUM based on its vector of beta coefficients*

### Description

Plots the kernel function for a given set of beta coefficients. NB: coefficients will be normalised so that the kernel describes a probability density.

### Usage

```
plot_kernel(beta, type = 'l', ...)
```

### Arguments

| | |
|---|---|
| beta | positive numeric vector of kernel coefficients. |
| type | (optional) plot type, as in base plot() function. The default is a line plot. |
| ... | (optional) any additional arguments accepted by base plot() function. |

### References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

### Examples

```
### Plot order 4 smooth kernel

plot_kernel(1/factorial(0:4))

### Use a different line style

plot_kernel(1/factorial(0:4), lty = 2)
```

## predict.fk_ppr — *Predict method for class fk_ppr*

### Description

Standard prediction method for regression models, specific to outputs from the fk_ppr() function. See help(fk_ppr) for more details on the model.

### Usage

```
## S3 method for class 'fk_ppr'
predict(object, Xtest = NULL, ...)
```

## Arguments

| | |
|---|---|
| object | an object of class fk_ppr, output from the function of the same name. |
| Xtest | (optional) matrix of test data on which predictions are to be made. If omitted then fitted values from training data are returned. |
| ... | (optional) further arguments passed to or from other methods. |

## Value

A vector of predictions for Xtest.

## Examples

```
op <- par(no.readonly = TRUE)

set.seed(2)

### Generate a set of data

X = matrix(rnorm(10000), ncol = 10)

### Generate some "true" projection vectors

beta1 = (runif(10)>.5)*rnorm(10)
beta2 = (runif(10)>.5)*rnorm(10)

### Generate responses, dependent on X%*%beta1 and X%*%beta2

y = 1 + X%*%beta1 + ((X%*%beta2)>2)*(X%*%beta2-2)*10
y = y + (X%*%beta1)*(X%*%beta2)/5 + rnorm(1000)

### Fit a PPR model with 2 terms on a sample of the data

train_ids = sample(1:1000, 500)

model = fk_ppr(X[train_ids,], y[train_ids], nterms = 2)

### Predict on left out data, and compute
### estimated coefficient of determination

yhat = predict(model, X[-train_ids,])

MSE = mean((yhat-y[-train_ids])^2)
Var = mean((y[-train_ids]-mean(y[-train_ids]))^2)

1-MSE/Var


################################################

### Add some "outliers" in the training data and fit
### the model again, as well as one with an absolute loss
```

```
y[train_ids] = y[train_ids] + (runif(500)<.05)*(rnorm(500)*100)

model1 <- fk_ppr(X[train_ids,], y[train_ids], nterms = 2)

model2 <- fk_ppr(X[train_ids,], y[train_ids], nterms = 2,
    loss = function(y, hy) abs(y-hy),
    dloss = function(y, hy) sign(hy-y))

### Plot the resulting components in the model on the test data

par(mar = c(2, 2, 2, 2))
par(mfrow = c(2, 2))

plot(X[-train_ids,]%*%model1$vs[1,], y[-train_ids])
plot(X[-train_ids,]%*%model1$vs[2,], y[-train_ids])
plot(X[-train_ids,]%*%model2$vs[1,], y[-train_ids])
plot(X[-train_ids,]%*%model2$vs[2,], y[-train_ids])

par(op)

### estimate comparative estimated coefficients of determination

MSE1 = mean((predict(model1, X[-train_ids,])-y[-train_ids])^2)
MSE2 = mean((predict(model2, X[-train_ids,])-y[-train_ids])^2)
Var = mean((y[-train_ids]-mean(y[-train_ids]))^2)


1-MSE1/Var
1-MSE2/Var
```

---

`predict.fk_regression` *Predict method for class fk_regression*

---

## Description

Predict method for kernel regression. Evaluates the fitted regression function at a set of evaluation/test points

## Usage

```
## S3 method for class 'fk_regression'
predict(object, xtest = NULL, ...)
```

## Arguments

| | |
|---|---|
| `object` | an object of class fk_regression, output from the function of the same name. |
| `xtest` | (optional) vector of evaluation/test points. If omitted then the fitted values at the original sample are returned. |
| `...` | (optional) further arguments passed to or from other methods. |

## Value

vector of fitted/predicted values of the regression function.

---

`print.fk_density` *Print method for class fk_density*

---

## Description

Print method for kernel density estimate obtained with function fk_density()

## Usage

```
## S3 method for class 'fk_density'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | an object of class fk_density, output from the function of the same name. |
| `...` | (optional) further arguments passed to or from other methods. |

---

`print.fk_ICA` *Print method for class fk_ICA*

---

## Description

Print method for Independent Component Analysis model fit with function fk_ICA()

## Usage

```
## S3 method for class 'fk_ICA'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | an object of class fk_ICA, output from the function of the same name. |
| `...` | (optional) further arguments passed to or from other methods. |

## print.fk_mdh *Print method for class fk_mdh*

### Description

Print method for minimum density hyperplane solution obtained with function fk_mdh()

### Usage

```
## S3 method for class 'fk_mdh'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | an object of class fk_mdh, output from the function of the same name. |
| ... | (optional) further arguments passed to or from other methods. |

## print.fk_ppr *Print method for class fk_ppr*

### Description

Print method for projection pursuit regression model fit with function fk_ppr()

### Usage

```
## S3 method for class 'fk_ppr'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | an object of class fk_ppr, output from the function of the same name. |
| ... | (optional) further arguments passed to or from other methods. |

## print.fk_regression *Print method for class fk_regression*

### Description

Print method for kernel regression estimate obtained with function fk_regression()

### Usage

```
## S3 method for class 'fk_regression'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | an object of class fk_regression, output from the function of the same name. |
| ... | (optional) further arguments passed to or from other methods. |

## roughness_K *Kernel roughness*

### Description

Computes the squared L2 norm, also known as the roughness, of a kernel implemented in FKSUM based on its beta coefficients. NB: the coefficients will first be normalised so that the kernel represents a density function. Equivalent to norm_K()^2

### Usage

```
roughness_K(beta)
```

### Arguments

| | |
|---|---|
| beta | numeric vector of positive coefficients. |

### Value

positive numeric representing the squared L2 norm, or roughness of the kernel with coefficients beta/norm_const(beta).

### References

Hofmeyr, D.P. (2019) "Fast exact evaluation of univariate kernel sums", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press.

var_K *Variance of a kernel*

## Description

Computes the variance of the random variable whose density is given by a kernel implemented in FKSUM, with coefficients beta. NB: coefficients will first be normalised so that the kernel is a density function.

## Usage

```
var_K(beta)
```

## Arguments

| | |
|---|---|
| beta | positive numeric vector of kernel coefficients. |

## Value

A positive numeric value representing the variance of the random variable with density given by the kernel.

# Index