Package ‘gplm’

February 20, 2015

Type Package
Title Generalized partial linear models (GPLM)
Version 0.7-2
Date 2014-05-07
Author Marlene Mueller
Maintainer Marlene Mueller <marlene.mueller@gmx.de>
Description Functions for estimating generalized partial linear models
Depends AER
License GPL (>= 2)
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-05-07 22:38:05

R topics documented:

bandwidth.scott ................................................. 2
convol .......................................................... 3
create.grid ...................................................... 4
glm.inverse.link ............................................... 5
glm.link ......................................................... 6
glm.ll .......................................................... 7
glm.1ld ......................................................... 8
kbackfit ......................................................... 9
kde ............................................................. 10
kernel.constants ............................................... 11
kernel.function ............................................... 12
kgplm ......................................................... 13
kreg .......................................................... 16
sgplm1 ......................................................... 17

Index 21
bandwidth.scott

Scott’s rule of thumb

Description

Calculates Scott’s rule of thumb bandwidth vector.

Usage

bandwidth.scott(x, kernel = "biweight", product = TRUE)

Arguments

x n x d matrix, data
kernel text string, see kernel.function
product (if d>1) product or spherical kernel

Details

The default bandwidth vector is computed by Scott’s rule of thumb for the Gaussian kernel and adapted to the chosen kernel function.

Value

d x 1 bandwidth vector used for calculation

Author(s)

Marlene Mueller

References


See Also

kernel.function, kde

Examples

```r
## two-dimensional data
n <- 1000
u <- runif(n)
thresh <- 0.4
x1 <- rnorm(n)*(u<thresh) + rnorm(n,mean=3)*(u>=thresh)
x2 <- rnorm(n)*(u<thresh) + rnorm(n,mean=9)*(u>=thresh)
bandwidth.scott( cbind(x1,x2) )
```
**convol**

*Kernel convolution*

**Description**

Calculates the convolution of data with a kernel function.

**Usage**

```r
convol(x, h = 1, grid = NULL, y = 1, w = 1, p = 2, q = 2,
       product = TRUE, sort = TRUE)
```

**Arguments**

- `x`: n x d matrix, data
- `h`: scalar or 1 x d, bandwidth(s)
- `grid`: m x d matrix, where to calculate the convolution (default = x)
- `y`: n x c matrix, optional responses
- `w`: scalar or n x 1 or 1 x m or n x m, optional weights
- `p`: integer or text, see `kernel.function`
- `q`: integer, see `kernel.function`
- `product`: (if d>1) product or spherical kernel
- `sort`: logical, TRUE if data need to be sorted

**Details**

The kernel convolution which is calculated is \( \sum_i K_h(x_i - grid_j) y_i w_{ij} \) for \( i = 1,...,n \) and \( j = 1,...,m \). The kernel function is determined by the kernel parameters \( p \) and \( q \), see `kernel.function`. The default kernel is the biweight (quartic) kernel function. Note that the DLL requires the data matrix to be sorted by its first column.

**Value**

m x c matrix

**Author(s)**

Marlene Mueller

**See Also**

`kernel.function`, `kde`, `kreg`

**Examples**

```r
n <- 100
x <- rnorm(n)
convol(x,h=0.8,grid=-3:3)/n  # estimates density of x at points -3:3
```
create.grid

Create a grid for kernel estimation

Description

Helps to define a grid for kernel density or regression estimates (univariate or multivariate).

Usage

create.grid(grid.list, sort=TRUE)

Arguments

- **grid.list**: list of 1-dimensional vectors containing the grid values for each dimension
- **sort**: sort the vectors (can be set to FALSE if vectors are already sorted in ascending order)

Details

This function allows easily to define grids for the "gplm" package. If the data are d-dimensional and the grid vector lengths are n1, ... nd, then the output is a (n1*...*nd) x d matrix with each row corresponding to one d-dimensional data point at which the function estimate is to be calculated.

Value

m x d grid matrix

Author(s)

Marlene Mueller

See Also

expand.grid, kde, kreg

Examples

```r
v1 <- 1:5
v2 <- 3:1
grid <- create.grid(list(v1,v2))
x <- matrix(rnorm(60),30,2)
v1 <- seq(min(x[,1]),max(x[,1]),length=10)
v2 <- seq(min(x[,2]),max(x[,2]),length=5)
grid <- create.grid(list(v1,v2))
```
 glm.inverse.link

---

**glm.inverse.link**  
*Link function for GLM*

### Description

Defines the link function for a GLM.

### Usage

```r
glm.inverse.link(mu, family="gaussian", link="identity", k=1)
```

### Arguments

- `mu`: n x 1, linear predictors
- `family`: text string, family of distributions (e.g. "gaussian" or "bernoulli", see details for `glm.ll`)
- `link`: text string, link function (depending on family, see details for `glm.ll`)
- `k`: integer > 0, parameter for the negative binomial

### Value

n x 1, vector eta (predictors)

### Author(s)

Marlene Mueller

### See Also

- `glm.ll`, `glm.1ld`, `glm.link`

### Examples

```r
glm.inverse.link(c(0.25,0.5), family="bernoulli", link="logit")
```
glm.link

Inverse Link function for GLM

Description

Defines the inverse link function for a GLM.

Usage

glm.link(eta, family="gaussian", link="identity", k=1)

Arguments

- **eta**: n x 1, linear predictors
- **family**: text string, family of distributions (e.g. "gaussian" or "bernoulli", see details for glm.11)
- **link**: text string, link function (depending on family, see details for glm.11)
- **k**: integer > 0, parameter for the negative binomial

Value

n x 1, vector mu (responses)

Author(s)

Marlene Mueller

See Also

glm.11, glm.1ld, glm.inverse.link

Examples

glm.link(c(-1,2), family="bernoulli", link="logit")
glm.ll 

Log-likelihood for GLM

Description

Calculates the log-likelihood function of a GLM. Currently only the gaussian and the bernoulli family are implemented.

Usage

glm.ll(mu, y, phi=1, family="gaussian", k=1)

Arguments

mu    n x 1, predicted regression function
y     n x 1, responses
phi   scalar, nuisance parameter (sigma^2 for the gaussian and inverse gaussian families, nu for the gamma family)
family text string, family of distributions (e.g. "gaussian" or "bernoulli", see details below)
k     integer > 0, parameter for the negative binomial

Details

Implemented are the "gaussian" family (with links "identity" and "log"), the "bernoulli" family (with links "logit" and "probit"), the "gamma" family (with link "inverse"), the "poisson" family (with link "log"), the "inverse.gaussian" family (with link "inverse.squared") and the "negative.binomial" (with its canonical "log" type link).

The default value k=1 leads to the geometric distribution (as a special case of the negative binomial).

Value

log-likelihood value

Author(s)

Marlene Mueller

See Also

glm.lld, glm.link

Examples

glm.ll(rep(0.4,2), c(0,1), family="bernoulli")
glm.lld

Log-likelihood derivatives for GLM

Description

Computes first and second derivatives of the individual log-likelihood with respect to the linear predictor. Currently only the gaussian (with identity link) and the bernoulli family (with logit and probit links) are implemented.

Usage

\[ \text{glm.lld(eta, y, family="gaussian", link="identity", k=1)} \]

Arguments

- `eta`: n x 1, linear predictors
- `y` : n x 1, responses
- `family`: text string, family of distributions (e.g. "gaussian" or "bernoulli", see details for glm.lld)
- `link` : text string, link function (depending on family, see details for glm.lld)
- `k`: integer > 0, parameter for the negative binomial

Details

See details for glm.lld.

Value

List with components:

- `ll1`: n x 1, vector of first derivatives
- `ll2`: n x 1, vector of second derivatives
- `ll1ll2`: n x 1, ratio ll1/ll2

Author(s)

Marlene Mueller

See Also

glm.l1, glm.link

Examples

\[ \text{glm.lld(c(-1,2), c(0,1), family="bernoulli", link="logit")} \]
kbackfit

Backfitting for an additive model using kernel regression

Description

Implements kernel-based backfitting in an additive model, optional with a partial linear term.

Usage

kbackfit(t, y, h, x = NULL, grid = NULL, weights.conv = 1,
    offset = 0, method = "generic",
    max.iter = 50, eps.conv = 1e-04, m.start = NULL,
    kernel = "biweight")

Arguments

y n x 1 vector, responses
t n x q matrix, data for nonparametric part
h scalar or 1 x q, bandwidth(s)
x optional, n x p matrix, data for linear part
grid m x q matrix, where to calculate the nonparametric function (default = t)
weights.conv weights for convergence criterion
offset offset
method one of "generic", "lininit" or "modified"
max.iter maximal number of iterations
eps.conv convergence criterion
m.start n x q matrix, start values for m
kernel text string, see kernel.function

Value

List with components:
c constant
b p x 1 vector, linear coefficients
m n x q matrix, nonparametric marginal function estimates
m.grid m x q matrix, nonparametric marginal function estimates on grid
rss residual sum of squares

Author(s)

Marlene Mueller

See Also

kernel.function, kreg
Kernel density estimation

Description

Calculates a kernel density estimate (univariate or multivariate).

Usage

kde(x, bandwidth = NULL, grid = TRUE, kernel = "biweight",
     product = TRUE, sort = TRUE)

Arguments

x     n x d matrix, data
bandwidth scalar or 1 x d, bandwidth(s)
grid   logical or m x d matrix (where to calculate the density)
kernel text string, see kernel.function
product (if d>1) product or spherical kernel
sort   logical, TRUE if data need to be sorted

Details

The kernel density estimator is calculated as \( \frac{1}{n} \sum_{i} K_{h}(x_{i} - grid_{j}) \) for \( i = 1, ..., n \) and \( j = 1, ..., m \). The default bandwidth vector is computed by Scott’s rule of thumb (adapted to the chosen kernel function).

Value

List with components:

x     m x d matrix, where density has been calculated
y     m x 1 vector, density estimates
bandwidth bandwidth vector used for calculation
rearrange if sort=TRUE, index to rearrange x and y to its original order

Author(s)

Marlene Mueller

See Also

kernel.function, convol, kreg
### Examples

```r
n <- 1000
x <- rnorm(n)
plot(kde(x), type="l")

## mixed normal data
n <- 1000
u <- runif(n)
thresh <- 0.4
x <- rnorm(n)*(u<thresh) + rnorm(n,mean=3)*(u>=thresh)
h <- 1
fh <- kde(x, bandwidth=h)
plot(kde(x, bandwidth=h), type="l", lwd=2); rug(x)
lines(kde(x, bandwidth=h*1.2), col="red")
lines(kde(x, bandwidth=h*1.4), col="orange")
lines(kde(x, bandwidth=h/1.2), col="blue")
lines(kde(x, bandwidth=h/1.4), col="cyan")

## two-dimensional data
n <- 1000
u <- runif(n)
thresh <- 0.4
x1 <- rnorm(n)*(u<thresh) + rnorm(n,mean=3)*(u>=thresh)
x2 <- rnorm(n)*(u<thresh) + rnorm(n,mean=9)*(u>=thresh)
grid1 <- seq(min(x1), max(x1), length=20)  ## grid for x1
grid2 <- seq(min(x2), max(x2), length=25)  ## grid for x2

fh <- kde( cbind(x1,x2), grid=create.grid(list(grid1, grid2)) )
o <- order(fh$x[,2], fh$x[,1])
density <- matrix(fh$y[o], length(grid1), length(grid2))

par(mfrow=c(2,2))
plot(kde(x1), type="l", main="x1"); rug(x1)
plot(kde(x2), type="l", main="x2"); rug(x2)
persp(grid1,grid2,density,main="KDE",
theta=30, phi=30, expand=0.5, col="lightblue", shade=0.5)
contour(grid1,grid2,density, main="KDE Contours")
points(x1,x2,col="red",pch=18,cex=0.5)
par(mfrow=c(1,1))
```

### Description

Calculates several constants of a (product) kernel function.

### Usage

```r
kernel.constants(kernel = "biweight", d = 1, product = TRUE)
```
Arguments

kernel function

Arguments

kernel: text string, see kernel.function

Arguments

d: integer (dimension of the kernel, d <= 4)

Arguments

product: (if d>1) product or spherical kernel

Details

The constants which are calculated are the second moment, the square norm and the canonical bandwidth of the kernel (only the two latter terms depend on the dimension d).

Value

List with components:

m2: second moment

m2: second moment

c2: square norm

d0: canonical bandwidth

Author(s)

Marlene Mueller

See Also

kernel.function

Examples

kernel.constants()  # default (biweight), d=1

kernel.constants("epanechnikov",1)  # epanechnikov, d=1

kernel.constants("epanechnikov",2)  # product epanechnikov, d=2

Description

Calculates several kernel functions (uniform, triangle, epanechnikov, biweight, triweight, gaussian).

Usage

kernel.function(u, kernel = "biweight", product = TRUE)

Arguments

u: n x d matrix

Arguments

kernel: text string

Arguments

product: (if d>1) product or spherical kernel
Details

The kernel parameter is a text string specifying the univariate kernel function which is either the
Gaussian pdf or proportional to \((1 - |u|^p)^q\). Possible text strings are "triangle" \((p=q=1)\), "uniform"
\((p=1, q=0)\), "epanechnikov" \((p=2, q=1)\), "biweight" or "quartic" \((p=q=2)\), "triweight" \((p=2, q=3)\),
"gaussian" or "normal" (Gaussian pdf).

The multivariate kernels are obtained by a product of univariate kernels \(K(u_1) \ldots K(u_d)\) or by a
spherical (radially symmetric) kernel proportional to \(K(||u||)\). (The resulting kernel is a density,
i.e. integrates to 1.)

Value

n x 1 vector of kernel weights

Author(s)

Marlene Mueller

Examples

- `kernel.function(0)` ## default (biweight)
- `kernel.function(0, kernel="epanechnikov")` ## epanechnikov
- `kernel.function(0, kernel="gaussian")` ## equals dnorm(0)

kgplm

Generalized partial linear model

Description

Fits a generalized partial linear model (kernel-based) using the (generalized) Speckman estimator
or backfitting (in the generalized case combined with local scoring) for two additive component
functions.

Usage

kgplm(x, t, y, h, family, link,
  b.start=NULL, m.start=NULL, grid = NULL,
  offset = 0, method = "speckman", sort = TRUE, weights = 1,
  weights.trim = 1, weights.conv = 1, max.iter = 25, eps.conv = 1e-8,
  kernel = "biweight", kernel.product = TRUE, verbose = FALSE)

Arguments

- x n x p matrix, data for linear part
- y n x 1 vector, responses
- t n x q matrix, data for nonparametric part
- h scalar or 1 x q, bandwidth(s)
family text string, family of distributions (e.g. "gaussian" or "bernoulli", see details for \texttt{glm.11})
link text string, link function (depending on family, see details for \texttt{glm.11})
b.start p x 1 vector, start values for linear part
m.start n x 1 vector, start values for nonparametric part
grid m x q matrix, where to calculate the nonparametric function (default = t)
offset offset
method "speckman" or "backfit"
sort logical, TRUE if data need to be sorted
weights binomial weights
weights.trim trimming weights for fitting the linear part
weights.conv weights for convergence criterion
max.iter maximal number of iterations
eps.conv convergence criterion
kernel text string, see \texttt{kernel.function}
kernel.product (if p>1) product or spherical kernel
verbose print additional convergence information

Value
List with components:
\texttt{b} p x 1 vector, linear coefficients
\texttt{b.cov} p x p matrix, linear coefficients
\texttt{m} n x 1 vector, nonparametric function estimate
\texttt{m.grid} m x 1 vector, nonparametric function estimate on grid
\texttt{it} number of iterations
\texttt{deviance} deviance
\texttt{df.residual} approximate degrees of freedom (residuals)
\texttt{aic} Akaike’s information criterion

Author(s)
Marlene Mueller

References
kgplm

See Also

kernel.function, kreg

Examples

```r
## data
n <- 1000; b <- c(1,-1); rho <- 0.7
m <- function(t){ 1.5*sin(pi*t) }
x1 <- runif(n,min=-1,max=1); u <- runif(n,min=-1,max=1)
t <- runif(n,min=-1,max=1); x2 <- round(m(rho*t + (1-rho)*u))
x <- cbind(x1,x2)
y <- x %*% b + m(t) + rnorm(n)

## partial linear model (PLM)
gh <- kgplm(x,t,y,h=0.25,family="gaussian",link="identity")
o <- order(t)
plot(t[o],m(t[o]),type="l",col="green")
lines(t[o],gh$m[o]); rug(t)

## partial linear probit model (GPLM)
y <- (y>0)
gh <- kgplm(x,t,y,h=0.25,family="bernoulli",link="probit")
o <- order(t)
plot(t[o],m(t[o]),type="l",col="green")
lines(t[o],gh$m[o]); rug(t)

## data with two-dimensional m-function
n <- 1000; b <- c(1,-1); rho <- 0.7
m <- function(t1,t2){ 1.5*sin(pi*t1)+t2 }
x1 <- runif(n,min=-1,max=1); u <- runif(n,min=-1,max=1)
t1 <- runif(n,min=-1,max=1); t2 <- runif(n,min=-1,max=1)
x2 <- round( m( rho*t1 + (1-rho)*u , t2 ) )
x <- cbind(x1,x2); t <- cbind(t1,t2)
y <- x %*% b + m(t1,t2) + rnorm(n)

## partial linear model (PLM)
grid1 <- seq(min(t[,1]),max(t[,1]),length=20)
grid2 <- seq(min(t[,2]),max(t[,2]),length=25)
grid <- create.grid(list(grid1,grid2))
gh <- kgplm(x,t,y,h=0.5,grid=grid,family="gaussian",link="identity")
o <- order(grid[2],grid[1])
est.m <- matrix(gh$m,grid[0],length(grid1),length(grid2)))
orig.m <- outer(grid1,grid2,m)
par(mfrow=c(1,2))
persp(grid1,grid2,orig.m,main="Original Function",
theta=30,phi=30,expand=0.5,col="lightblue",shade=0.5)
persp(grid1,grid2,est.m,main="Estimated Function",
theta=30,phi=30,expand=0.5,col="lightblue",shade=0.5)
par(mfrow=c(1,1))
```
**kreg**

*Kernel regression*

**Description**

Calculates a kernel regression estimate (univariate or multivariate).

**Usage**

```r
kreg(x, y, bandwidth = NULL, grid = TRUE, kernel = "biweight",
     product = TRUE, sort = TRUE)
```

**Arguments**

- `x` n x d matrix, data
- `y` n x 1 vector, responses
- `bandwidth` scalar or 1 x d, bandwidth(s)
- `grid` logical or m x d matrix (where to calculate the regression)
- `kernel` text string, see `kernel.function`
- `product` (if d>1) product or spherical kernel
- `sort` logical, TRUE if data need to be sorted

**Details**

The estimator is calculated by Nadaraya-Watson kernel regression. Future extension to local linear (d>1) or polynomial (d=1) estimates is planned. The default bandwidth is computed by Scott’s rule of thumb for kde (adapted to the chosen kernel function).

**Value**

List with components:

- `x` m x d matrix, where regression has been calculated
- `y` m x 1 vector, regression estimates
- `bandwidth` bandwidth used for calculation
- `df.residual` approximate degrees of freedom (residuals)
- `rearrange` if sort=TRUE, index to rearrange x and y to its original order.

**Author(s)**

Marlene Mueller

**See Also**

`kernel.function`, `convol`, `kde`
Examples

```r
n <- 1000
x <- rnorm(n)
m <- sin(x)
y <- m + rnorm(n)
plot(x,y,col="gray")
o <- order(x); lines(x[o],m[o],col="green")
lines(kreg(x,y),lwd=2)

## two-dimensional
n <- 100
x <- 6*cbind(runif(n),runif(n))
m <- function(x1,x2){ 4*sin(x1) + x2 }
y <- m(x[,1],x[,2]) + rnorm(n)
mh <- kreg(x,y)#,bandwidth=1)

grid1 <- unique(mh$x[,1])
grid2 <- unique(mh$x[,2])
est.m <- t(matrix(mh$y,length(grid1),length(grid2)))
orig.m <- outer(grid1,grid2,m)
par(mfrow=c(1,2))
persp(grid1,grid2,orig.m,main="Original Function",
     theta=30,phi=30,expand=0.5,col="lightblue",shade=0.5)
persp(grid1,grid2,est.m,main="Estimated Function",
     theta=30,phi=30,expand=0.5,col="lightblue",shade=0.5)
par(mfrow=c(1,1))

## now with normal x, note the boundary problem,
## which can be somewhat reduced by a gaussian kernel
n <- 1000
x <- cbind(rnorm(n), rnorm(n))
m <- function(x1,x2){ 4*sin(x1) + x2 }
y <- m(x[,1],x[,2]) + rnorm(n)
mh <- kreg(x,y)#,p="gaussian")

grid1 <- unique(mh$x[,1])
grid2 <- unique(mh$x[,2])
est.m <- t(matrix(mh$y,length(grid1),length(grid2)))
orig.m <- outer(grid1,grid2,m)
par(mfrow=c(1,2))
persp(grid1,grid2,orig.m,main="Original Function",
     theta=30,phi=30,expand=0.5,col="lightblue",shade=0.5)
persp(grid1,grid2,est.m,main="Estimated Function",
     theta=30,phi=30,expand=0.5,col="lightblue",shade=0.5)
par(mfrow=c(1,1))
```

**Generalized partial linear model**
Description

Fits a generalized partial linear model (based on smoothing spline) using the (generalized) Speckman estimator or backfitting (in the generalized case combined with local scoring) for two additive component functions. In contrast to \texttt{kgplm}, this function can be used only for a 1-dimensional nonparametric function.

Usage

\begin{verbatim}
sgplm1(x, t, y, spar, df=4, family, link,
        b.start=NULL, m.start=NULL, grid = NULL, offset = 0,
        method = "speckman", weights = 1, weights.trim = 1,
        weights.conv = 1, max.iter = 25, eps.conv = 1e-8,
        verbose = FALSE, ...)
\end{verbatim}

Arguments

- \texttt{x}          n x p matrix, data for linear part
- \texttt{y}          n x 1 vector, responses
- \texttt{t}          n x 1 matrix, data for nonparametric part
- \texttt{spar}       scalar smoothing parameter, as in \texttt{smooth.spline}
- \texttt{df}         scalar equivalent number of degrees of freedom (trace of the smoother matrix), as in \texttt{smooth.spline}
- \texttt{family}     text string, family of distributions (e.g. "gaussian" or "bernoulli", see details for \texttt{glm})
- \texttt{link}       text string, link function (depending on family, see details for \texttt{glm})
- \texttt{b.start}    p x 1 vector, start values for linear part
- \texttt{m.start}    n x 1 vector, start values for nonparametric part
- \texttt{grid}       m x q matrix, where to calculate the nonparametric function (default = \texttt{t})
- \texttt{offset}     offset
- \texttt{method}     "speckman" or "backfit"
- \texttt{weights}    binomial weights
- \texttt{weights.trim} trimming weights for fitting the linear part
- \texttt{weights.conv} weights for convergence criterion
- \texttt{max.iter}   maximal number of iterations
- \texttt{eps.conv}   convergence criterion
- \texttt{verbose}   print additional convergence information
- \texttt{...}       further parameters to be passed to \texttt{smooth.spline}
**Value**

List with components:

- \( b \)  
  \( \mathbf{p} \times 1 \) vector, linear coefficients
- \( b.\text{cov} \)  
  \( \mathbf{p} \times \mathbf{p} \) matrix, linear coefficients
- \( m \)  
  \( \mathbf{n} \times 1 \) vector, nonparametric function estimate
- \( m.\text{grid} \)  
  \( \mathbf{m} \times 1 \) vector, nonparametric function estimate on grid
- \( it \)  
  number of iterations
- \( \text{deviance} \)  
  deviance
- \( \text{df.residual} \)  
  approximate degrees of freedom (residuals)
- \( \text{aic} \)  
  Akaike’s information criterion

**Note**

This function is mainly implemented for comparison. It is not really optimized for performance, however since it is spline-based, it should be sufficiently fast. Nevertheless, there might be several possibilities to improve for speed, in particular I guess that the sorting that \texttt{smooth.spline} performs in every iteration is slowing down the procedure quite a bit.

**Author(s)**

Marlene Mueller

**References**


**See Also**

\texttt{kgplm}

**Examples**

```r
## generate data
n <- 1000; b <- c(1,-1); rho <- 0.7
mm <- function(t){ 1.5*sin(pi*t) }
x1 <- runif(n,min=-1,max=1); u <- runif(n,min=-1,max=1)
t <- runif(n,min=-1,max=1); x2 <- round(mm(rho*t + (1-rho)*u))
x <- cbind(x1,x2)
y <- x %*% b + mm(t) + rnorm(n)

## fit partial linear model (PLM)
k.plm <- kgplm(x,t,y,h=0.35,family="gaussian",link="identity")
s.plm <- sgplm1(x,t,y,spar=0.95,family="gaussian",link="identity")
o <- order(t)
```
ylim <- range(c(mm(t[o]), k.plm$m, s.plm$m), na.rm=TRUE)
plot(t[o], mm(t[o]), type="l", ylim=ylim)
lines(t[o], k.plm$m[0], col="green")
lines(t[o], s.plm$m[0], col="blue")
rug(t); title("Kernel PLM vs. Spline PLM")

## fit partial linear probit model (GPLM)
y <- (y>0)
k.gplm <- kgplm(x, t, y, h=0.35, family="bernoulli", link="probit")
s.gplm <- sgplm(x, t, y, spar=0.95, family="bernoulli", link="probit")

o <- order(t)
ylim <- range(c(mm(t[o]), k.gplm$m, s.gplm$m), na.rm=TRUE)
plot(t[o], mm(t[o]), type="l", ylim=ylim)
lines(t[o], k.gplm$m[0], col="green")
lines(t[o], s.gplm$m[0], col="blue")
rug(t); title("Kernel GPLM vs. Spline GPLM (Probit)")
Index

*Topic smooth
  bandwidth.scott, 2
  convol, 3
  create.grid, 4
  glm.inverse.link, 5
  glm.link, 6
  glm.1l, 7
  glm.1ld, 8
  kbackfit, 9
  kde, 10
  kernel.constants, 11
  kernel.function, 12
  kgplm, 13
  kreg, 16
  sgplm1, 17

bandwidth.scott, 2

convol, 3, 10, 16
create.grid, 4

expand.grid, 4

glm.inverse.link, 5, 6
glm.link, 5, 6, 7, 8
glm.1l, 5, 6, 7, 8, 14, 18
glm.1ld, 5–7, 8

kbackfit, 9
kde, 2–4, 10, 16
kernel.constants, 11
kernel.function, 2, 3, 9, 10, 12, 12, 14–16
kgplm, 13, 18, 19
kreg, 3, 4, 9, 10, 15, 16

sgplm1, 17
smooth.spline, 18, 19