

# Package ‘KernGPLM’

June 24, 2009

**Type** Package

**Title** Kernel-based GPLM and GAM

**Version** 0.65

**Date** 2009-06-24

**Author** Marlene Mueller

**Maintainer** Marlene Mueller <marlene.mueller@gmx.de>

**Description** Functions for estimating generalized partial linear models (GPLM)

**License** GPL version 2 or later

## R topics documented:

backfit . . . . .	1
convol . . . . .	3
create.grid . . . . .	4
glm.link . . . . .	5
glm.ll . . . . .	5
glm.lld . . . . .	6
kde . . . . .	7
kernel.constants . . . . .	8
kernel.function . . . . .	9
kgplm . . . . .	10
kreg . . . . .	12
sgplm1 . . . . .	14
<b>Index</b>	<b>17</b>

backfit

*Backfitting for an additive model***Description**

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

**Usage**

```
backfit(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.p = 2, kernel.q = 2)
```

**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", "linit" or "modified"
max.iter	maximal number of iterations
eps.conv	convergence criterion
m.start	n x q matrix, start values for m
kernel.p	integer or text, see <a href="#">kernel.function</a>
kernel.q	integer, see <a href="#">kernel.function</a>

**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](#)

---

convol	<i>Kernel convolution</i>
--------	---------------------------

---

**Description**

Calculates the convolution of data with a kernel function.

**Usage**

```
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 <a href="#">kernel.function</a>
q	integer, see <a href="#">kernel.function</a>
product	(if d>1) product or spherical kernel
sort	sort the data, necessary to use the DLL code

**Details**

The kernel convolution which is calculated is  $\sum_i K_h(x_i - grid_j) y_i w_{ij}$  for  $i = 1, \dots, n$  and  $j = 1, \dots, m$ .

**Value**

m x c matrix

**Author(s)**

Marlene Mueller

**See Also**

[kernel.function](#), [kde](#), [kreg](#)

**Examples**

```
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 KernGPLM 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

```
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.link *(Inverse) Link function for GLM*

---

**Description**

Defines the inverse link function for a GLM. (Currently only the LM as well as Logit and Probit are implemented.)

**Usage**

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

**Arguments**

eta	n x 1, linear predictors
family	text string (currently "gaussian", "bernoulli")
link	text string (currently "identity" for "gaussian", "logit" or "probit" for "bernoulli")

**Value**

n x 1, vector mu = glm.link( eta )

**Author(s)**

Marlene Mueller

**See Also**

[glm.ll](#), [glm.lld](#)

**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 LM as well as Logit and Probit are implemented.)

**Usage**

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

**Arguments**

mu	n x 1, predicted regression function
y	n x 1, responses
phi	scalar, nuisance parameter (sigma for "lm")
family	text string (currently "gaussian", "bernoulli")

**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 LM as well as Logit and Probit are implemented.)

**Usage**

```
glm.lld(eta, y, family="gaussian", link="identity")
```

**Arguments**

eta	n x 1, linear predictors
y	n x 1, responses
family	text string (currently "gaussian", "bernoulli")
link	text string (currently "identity" for "gaussian", "logit" or "probit" for "bernoulli")

**Value**

List with components:

ll1	n x 1, vector of first derivatives
ll2	n x 1, vector of second derivatives

**Author(s)**

Marlene Mueller

**See Also**

[glm.ll](#), [glm.link](#)

**Examples**

```
glm.lld(c(-1,2), c(0,1), family="bernoulli", link="logit")
```

---

kde

*Kernel density estimation*


---

**Description**

Calculates a kernel density estimate (univariate or multivariate).

**Usage**

```
kde(x, bandwidth = NULL, grid = TRUE, p = 2, q = 2,
    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)
p	integer or text, see <a href="#">kernel.function</a>
q	integer, see <a href="#">kernel.function</a>
product	(if d>1) product or spherical kernel
sort	sort the data, necessary to use the DLL code

**Details**

The kernel density estimator is calculated as  $\frac{1}{n} \sum_i K_h(x_i - grid_j)$  for  $i = 1, \dots, n$  and  $j = 1, \dots, m$ .

**Value**

List with components:

x	m x d matrix, where density has been calculated
y	m x 1 vector, density estimates
bandwidth	bandwidth 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**

```

n <- 1000
x <- rnorm(n)
plot(kde(x))

## mixed normals
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
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)
grid2 <- seq(min(x2),max(x2),length=25)
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))

```

---

kernel.constants     *Kernel constants*

---

**Description**

Calculates several constants of a (product) kernel function.

**Usage**

```
kernel.constants(p = 2, q = 2, d = 1, product = TRUE)
```

**Arguments**

**p**                    integer or text, see [kernel.function](#)  
**q**                    integer, see [kernel.function](#)

d                    integer (dimension of the kernel)  
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 depend on the dimension d).

### Value

List with components:

m2                  second moment  
c2                    square norm  
d0                    canonical bandwidth

### Author(s)

Marlene Mueller

### See Also

[kernel.function](#)

### Examples

```
kernel.constants()            ## default (biweight)
kernel.constants(2,1)        ## epanechnikov
kernel.constants(2,1,2)      ## product epanechnikov (d=2)
```

---

kernel.function     *Kernel function*

---

### Description

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

### Usage

```
kernel.function(u, p = 2, q = 2, product = TRUE)
```

### Arguments

u                    n x d matrix  
p                    integer or text  
q                    integer  
product             (if d>1) product or spherical kernel

**Details**

The parameters  $p > 0$  and  $q$  define the univariate kernel functions proportional to  $(1 - |u|^p)^q$ . The multivariate kernels are obtained by a product of univariate kernels  $K(u_1) \dots K(u_d)$  or by a spherical (radial symmetric) kernel proportional to  $K(\|u\|)$ . (Proportional means that the resulting kernel is a density, i.e. integrates to 1.) If  $p=0$  is set, the resulting kernel function is the gaussian (normal) kernel.

Alternatively, the  $p$  parameter may be a text string. Possible strings (and their related parameters) are "triangle" ( $p=q=1$ ), "uniform" ( $p$  arbitrary,  $q=0$ ), "epanechnikov" ( $p=2$ ,  $q=1$ ), "biweight" or "quartic" ( $p=q=2$ ), "triweight" ( $p=2$ ,  $q=3$ ), "gaussian" or "normal" ( $p=0$ ,  $q$  arbitrary).

**Value**

$n \times 1$  vector of kernel weights

**Author(s)**

Marlene Mueller

**Examples**

```
kernel.function(0)           ## default (biweight)
kernel.function(0,p=2,q=1)  ## epanechnikov
kernel.function(0,p=2,q=0)  ## uniform
```

---

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. (Note that currently only the PLM as well as partial linear Logit and Probit models are implemented.)

**Usage**

```
kgplm(x, t, y, h, family, link,
      b.start=NULL, m.start=NULL, grid = NULL, m.grid.start = NULL,
      offset = 0, method = "speckman", sort = TRUE, weights = 1,
      weights.trim = 1, weights.conv = 1, max.iter = 25, eps.conv = 1e-8,
      kernel.p = 2, kernel.q = 2, kernel.product = TRUE, verbose = FALSE)
```

**Arguments**

<code>x</code>	$n \times p$ matrix, data for linear part
<code>y</code>	$n \times 1$ vector, responses
<code>t</code>	$n \times q$ matrix, data for nonparametric part
<code>h</code>	scalar or $1 \times q$ , bandwidth(s)
<code>family</code>	text string (currently "gaussian", "bernoulli")
<code>link</code>	text string (currently "identity" for "gaussian", "logit" or "probit" for "bernoulli")

<code>b.start</code>	<code>p x 1</code> vector, start values for linear part
<code>m.start</code>	<code>n x 1</code> vector, start values for nonparametric part
<code>grid</code>	<code>m x q</code> matrix, where to calculate the nonparametric function (default = <code>t</code> )
<code>m.grid.start</code>	<code>m x 1</code> vector, start values for nonparametric part on grid
<code>offset</code>	offset
<code>method</code>	"speckman" or "backfit"
<code>sort</code>	sort data (default=TRUE)
<code>weights</code>	binomial weights
<code>weights.trim</code>	trimming weights for fitting the linear part
<code>weights.conv</code>	weights for convergence criterion
<code>max.iter</code>	maximal number of iterations
<code>eps.conv</code>	convergence criterion
<code>kernel.p</code>	integer or text, see <a href="#">kernel.function</a>
<code>kernel.q</code>	integer, see <a href="#">kernel.function</a>
<code>kernel.product</code>	(if <code>p&gt;1</code> ) product or spherical kernel
<code>verbose</code>	print additional convergence information

### Value

List with components:

<code>b</code>	<code>p x 1</code> vector, linear coefficients
<code>b.cov</code>	<code>p x p</code> matrix, linear coefficients
<code>m</code>	<code>n x 1</code> vector, nonparametric function estimate
<code>m.grid</code>	<code>m x 1</code> vector, nonparametric function estimate on grid
<code>it</code>	number of iterations
<code>deviance</code>	deviance
<code>df.residual</code>	approximate degrees of freedom (residuals)
<code>aic</code>	Akaike's information criterion

### Author(s)

Marlene Mueller

### References

- Mueller, M. (2001) Estimation and testing in generalized partial linear models – A comparative study. *Statistics and Computing*, 11:299–309.
- Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models*. London: Chapman and Hall.

### See Also

[kernel.function](#), [kreg](#)

**Examples**

```

## partial linear model (PLM)
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 <- x1*b[1]+x2*b[2] + m(t) + rnorm(n)
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)

## two-dimensional PLM
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 <- x1*b[1]+x2*b[2] + m(t1,t2) + rnorm(n)
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])
biv.m <- (matrix(gh$m.grid[o],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,biv.m,main="Estimated Function",
       theta=30,phi=30,expand=0.5,col="lightblue",shade=0.5)
par(mfrow=c(1,1))

```

**Description**

Calculates a kernel regression estimate (univariate or multivariate).

**Usage**

```
kreg(x, y, bandwidth = NULL, grid = TRUE, p = 2, q = 2,
     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)
p	integer or text, see <a href="#">kernel.function</a>
q	integer, see <a href="#">kernel.function</a>
product	(if d>1) product or spherical kernel
sort	sort the data, necessary to use the DLL code

**Details**

The estimator is calculated by Nadaraya-Watson kernel regression. Future extension to local linear (d>1) or polynomial (d=1) estimates is planned.

**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**

```
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))-3
```

```

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])
biv.reg <- 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,biv.reg,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])
biv.reg <- 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,biv.reg,main="Estimated Function",
      theta=30,phi=30,expand=0.5,col="lightblue",shade=0.5)
par(mfrow=c(1,1))

```

---

sgplm1

*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 [kgplm](#), this function can be used only for a 1-dimensional nonparametric function. (Note that as for [kgplm](#) currently only the PLM as well as partial linear Logit and Probit models are implemented.)

## Usage

```

sgplm1(x, t, y, spar, 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,
       verbose = FALSE, ...)

```

**Arguments**

<code>x</code>	<code>n x p</code> matrix, data for linear part
<code>y</code>	<code>n x 1</code> vector, responses
<code>t</code>	<code>n x 1</code> matrix, data for nonparametric part
<code>spar</code>	scalar smoothing parameter, as in <a href="#">smooth.spline</a>
<code>family</code>	text string (currently "gaussian", "bernoulli")
<code>link</code>	text string (currently "identity" for "gaussian", "logit" or "probit" for "bernoulli")
<code>b.start</code>	<code>p x 1</code> vector, start values for linear part
<code>m.start</code>	<code>n x 1</code> vector, start values for nonparametric part
<code>grid</code>	<code>m x q</code> matrix, where to calculate the nonparametric function (default = <code>t</code> )
<code>offset</code>	offset
<code>method</code>	"speckman" or "backfit"
<code>sort</code>	sort data (default=TRUE)
<code>weights</code>	binomial weights
<code>weights.trim</code>	trimming weights for fitting the linear part
<code>weights.conv</code>	weights for convergence criterion
<code>max.iter</code>	maximal number of iterations
<code>eps.conv</code>	convergence criterion
<code>verbose</code>	print additional convergence information
<code>...</code>	further parameters to be passed to <a href="#">smooth.spline</a>

**Value**

List with components:

<code>b</code>	<code>p x 1</code> vector, linear coefficients
<code>b.cov</code>	<code>p x p</code> matrix, linear coefficients
<code>m</code>	<code>n x 1</code> vector, nonparametric function estimate
<code>m.grid</code>	<code>m x 1</code> vector, nonparametric function estimate on grid
<code>it</code>	number of iterations
<code>deviance</code>	deviance
<code>df.residual</code>	approximate degrees of freedom (residuals)
<code>aic</code>	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 [smooth.spline](#) performs in every iteration is slowing down the procedure quite a bit.

**Author(s)**

Marlene Mueller

## References

- Mueller, M. (2001) Estimation and testing in generalized partial linear models – A comparative study. *Statistics and Computing*, 11:299–309.
- Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models*. London: Chapman and Hall.

## See Also

[kgplm](#)

## Examples

```
## partial linear model (PLM)
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 <- x1*b[1]+x2*b[2] + 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=0.5,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$m[o], col="green")
lines(t[o],s.plm$m[o], col="blue")
rug(t); title("Kern 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 <- sgplm1(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$m[o], col="green")
lines(t[o],s.gplm$m[o], col="blue")
rug(t); title("Kern GPLM vs. Spline GPLM (Probit)")
```

# Index

## \*Topic **smooth**

- backfit, 1
- convol, 2
- create.grid, 3
- glm.link, 4
- glm.ll, 5
- glm.lld, 5
- kde, 6
- kernel.constants, 8
- kernel.function, 9
- kgplm, 10
- kreg, 12
- sgplm1, 14

backfit, 1

convol, 2, 7, 13  
create.grid, 3

expand.grid, 4

glm.link, 4, 5, 6  
glm.ll, 4, 5, 6  
glm.lld, 4, 5, 5

kde, 3, 4, 6, 13  
kernel.constants, 8  
kernel.function, 2, 3, 6–8, 9, 10–13  
kgplm, 10, 14, 15  
kreg, 2–4, 7, 11, 12

sgplm1, 14  
smooth.spline, 14, 15