Splines and Smoothing

Alexander McLain

September 27, 2023

Example 1: Varying coefficients for the bodyfat data

This example will use the bodyfat data, but we'll only consider abdomen, height and age as our X variables.

```
## Registered S3 method overwritten by 'printr':
##   method                from
##   knit_print.data.frame rmarkdown
```

```
bf_dat <- read.csv("bodyfat2.csv")
bf_df <- data.frame(bf_dat)
bf_df_subset <- subset(bf_df,select = c(bodyfat,abdomen,height,age))
head(bf_df_subset)
```

bodyfat	abdomen	height	age
12.3	85.2	67.75	23
6.1	83.0	72.25	22
25.3	87.9	66.25	22
10.4	86.4	72.25	26
28.7	100.0	71.25	24
20.9	94.4	74.75	24

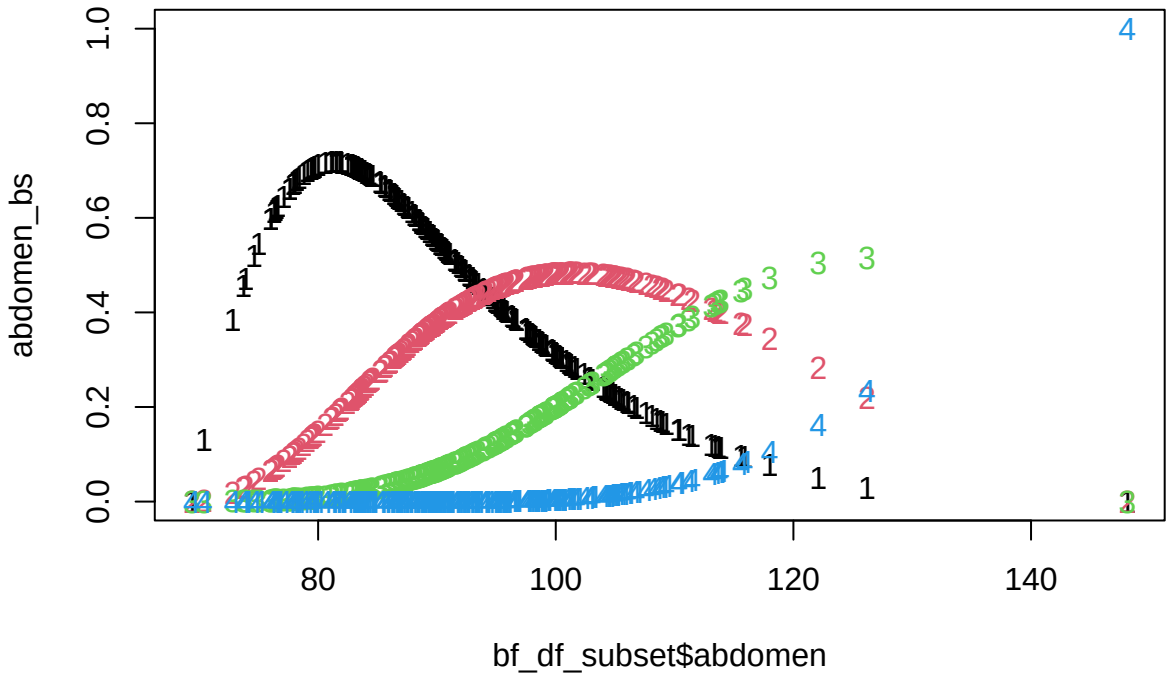
First we'll create the basis function for each of our variables

```
library(splines)
?bs
```

```
B-Spline Basis for Polynomial Splines

Description:

     Generate the B-spline basis matrix for a polynomial spline.

Usage:

     bs(x, df = NULL, knots = NULL, degree = 3, intercept = FALSE,
        Boundary.knots = range(x), warn.outside = TRUE)

Arguments:

       x: the predictor variable.  Missing values are allowed.

      df: degrees of freedom; one can specify 'df' rather than 'knots';
          'bs()' then chooses 'df-degree' (minus one if there is an
```

```
          intercept) knots at suitable quantiles of 'x' (which will
          ignore missing values).  The default, 'NULL', takes the
          number of inner knots as 'length(knots)'.  If that is zero as
          per default, that corresponds to 'df = degree - intercept'.

   knots: the _internal_ breakpoints that define the spline.  The
          default is 'NULL', which results in a basis for ordinary
          polynomial regression.  Typical values are the mean or median
          for one knot, quantiles for more knots.  See also
          'Boundary.knots'.

  degree: degree of the piecewise polynomial-default is '3' for cubic
          splines.

intercept: if 'TRUE', an intercept is included in the basis; default is
          'FALSE'.

Boundary.knots: boundary points at which to anchor the B-spline basis
          (default the range of the non-'NA' data).  If both 'knots'
          and 'Boundary.knots' are supplied, the basis parameters do
          not depend on 'x'.  Data can extend beyond 'Boundary.knots'.

warn.outside: 'logical' indicating if a 'warning' should be signalled
          in case some 'x' values are outside the boundary knots.

Examples:

     require(stats); require(graphics)
     bs(women$height, df = 5)
     summary(fm1 <- lm(weight ~ bs(height, df = 5), data = women))

     ## example of safe prediction
     plot(women, xlab = "Height (in)", ylab = "Weight (lb)")
     ht <- seq(57, 73, length.out = 200)
     lines(ht, predict(fm1, data.frame(height = ht)))
```

```r
abdomen_bs <- bs(bf_df_subset$abdomen,df=4,degree = 3)
height_bs <- bs(bf_df_subset$height,df=4,degree = 3)
age_bs <- bs(bf_df_subset$age,df=4,degree = 3)
matplot(bf_df_subset$abdomen,abdomen_bs,type = "p")
```



Now we'll fit the model

$$Y_i = \beta_0 + \sum_{j=1}^{K} \beta_{1j} h_{1j}(Abd_i) + \sum_{j=1}^{K} \beta_{2j} h_{2j}(Height_i) + \sum_{j=1}^{K} \beta_{3j} h_{3j}(Age_i) + \epsilon_i$$

where the $h_{kj}(x)$ correspond to cubic B-spline functions

```
B_spline_mod <- lm(bf_df_subset$bodyfat~abdomen_bs+height_bs+age_bs)
summary(B_spline_mod)
```

```
##
## Call:
## lm(formula = bf_df_subset$bodyfat ~ abdomen_bs + height_bs +
##     age_bs)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.9325  -3.3717  -0.1278   3.0088   9.7443
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    9.741      3.860   2.523 0.012270 *
## abdomen_bs1    3.695      3.581   1.032 0.303134
## abdomen_bs2   27.600      3.095   8.917  < 2e-16 ***
## abdomen_bs3   37.783      5.611   6.733 1.22e-10 ***
## abdomen_bs4   33.240      5.046   6.588 2.82e-10 ***
## height_bs1    -7.345      3.860  -1.903 0.058242 .
## height_bs2    -4.070      2.819  -1.444 0.150131
## height_bs3   -12.820      3.807  -3.368 0.000884 ***
## height_bs4   -11.093      3.814  -2.908 0.003974 **
## age_bs1       -1.305      2.847  -0.458 0.647248
## age_bs2        3.537      2.226   1.589 0.113380
## age_bs3       -2.264      3.540  -0.640 0.523077
```

```
## age_bs4        3.611      3.615   0.999 0.318960
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.437 on 239 degrees of freedom
## Multiple R-squared:  0.7323, Adjusted R-squared:  0.7189
## F-statistic: 54.48 on 12 and 239 DF,  p-value: < 2.2e-16
```

Now we'll estimate the impact (and the standard errors) as a function of our three explanatory variables. For example, for abdomen we'll estimate

$$\sum_{j=1}^{K} \beta_{1j} h_{1j}(X)$$

which we can write as

$$H_1(X)\beta_1$$

where $\beta_1 = (\beta_{11}, \ldots, \beta_{14})'$ and $H_1(x)$ is a $N \times 4$ matrix where N is the length of X and the jth column of H_1 is $h_{1j}(X)$.

If we let τ_l denote the left boundary knot, we have $h_{1j}(\tau_l) = 0$ for all $j = 1, \ldots, 4$ (this is by definition). Thus, τ_l serves at the "baseline"' ' " value (similar to if we use dummy coding) for the effect of X. All β values will be relative the the impact at τ_l.

The variance the total coefficient for abdomen at when abdomen$= X$ is given by

$$H_1(X)' Cov(\beta_1) H_1(X),$$

which we'll transform to standard error then plot $\pm 2SE$.

First, let's get $\beta_1 = (\beta_{11}, \ldots, \beta_{14})'$

```
abd_coef <- coef(B_spline_mod)[2:5]
```

Now, we'll get the X values we want to get the coefficient and SE for:

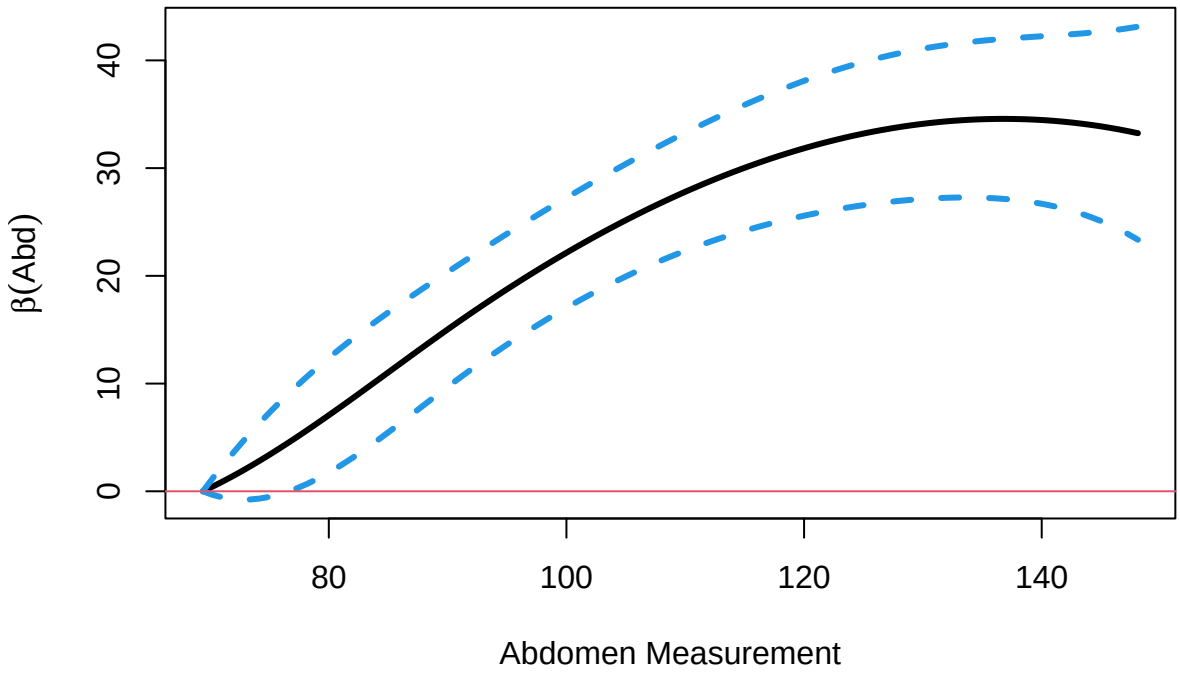
```
# Values on the abdomen scale
abd_vec <- seq(min(bf_dat$abdomen),max(bf_dat$abdomen),length.out = 100)
# Values on the expansion scale (note the knots and Boundary.knots)
H1_mat <- bs(abd_vec,knots = attr(abdomen_bs,"knots"), Boundary.knots =
                   attr(abdomen_bs,"Boundary.knots"))
```

Now, we'll calculate the coefficient and SE for all X values

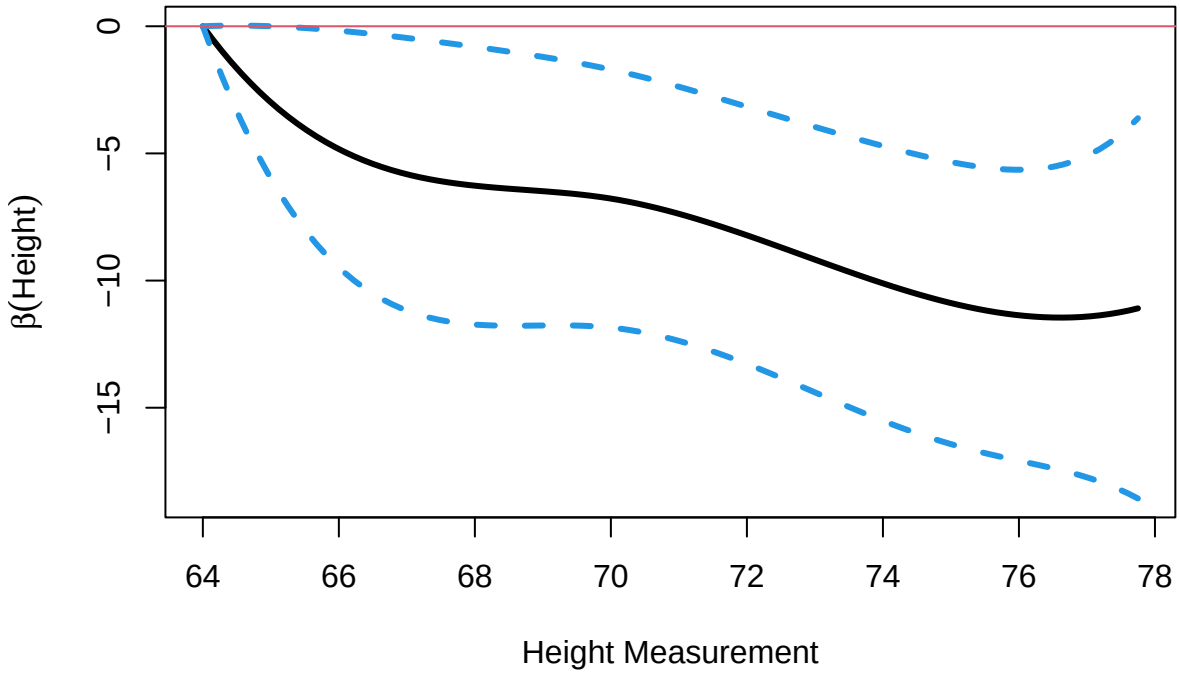
```
abd_beta <- H1_mat%*%abd_coef
cov_b1 <- vcov(B_spline_mod)[2:5,2:5]
abd_beta_se <- sqrt(diag(H1_mat%*%cov_b1%*%t(H1_mat)))
```

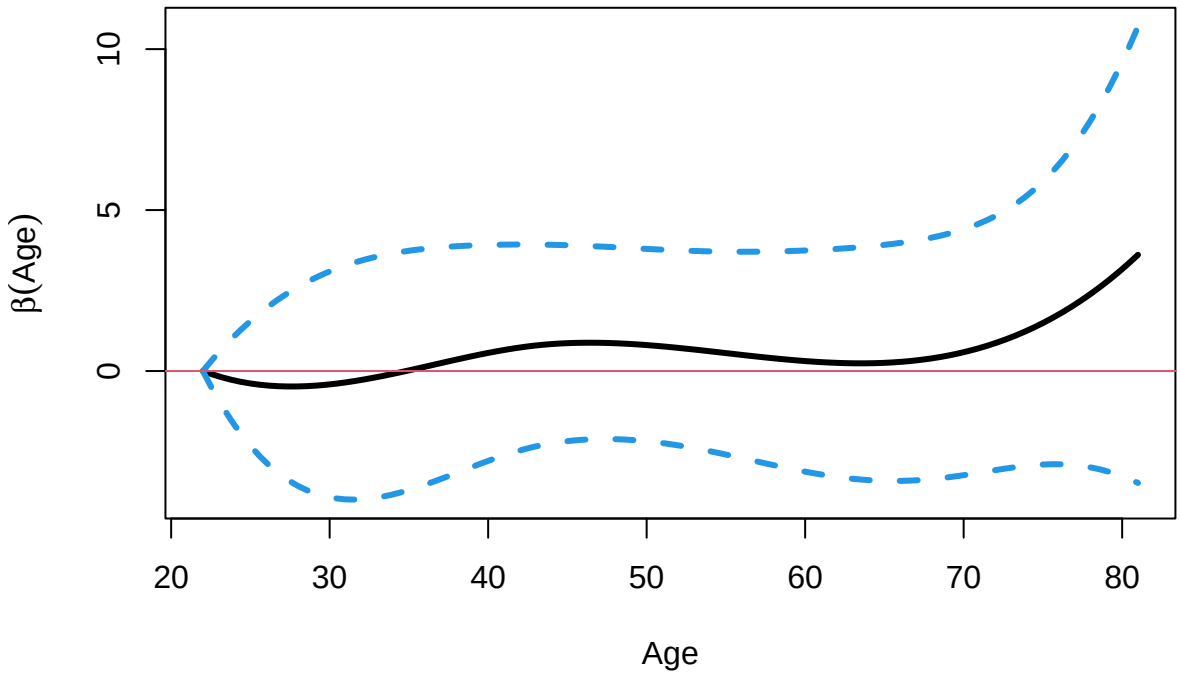
We'll add $\pm 2SE$ to the coefficient and then plot

```
l_limit_abd <- abd_beta - 1.96*abd_beta_se
u_limit_abd <- abd_beta + 1.96*abd_beta_se
abd_res <- data.frame(abd_beta,l_limit_abd,u_limit_abd)
matplot(abd_vec,abd_res,type = "l",col=c(1,4,4), lwd=3, lty=c(1,2,2),
        xlab=c("Abdomen Measurement"),ylab = expression(beta(Abd)))
abline(h=0,col=2)
```



We'll do the same for height and age, but not show the code.





Example 1 with smothing splines

Let's return to the first example, but this time we'll fit the model using penalized splines. Recall from the notes that the penalized RSS can be written as

$$RSS(f, \lambda) = (y - N\theta)'(y - N\theta) + \lambda\theta'\Omega_N\theta$$

where $N_{ij} = N_j(x_i)$ and $\{\Omega_N\}_{jk} = \int N_j''(t)N_k''(t)dt$. The solution is given by

$$\hat{\theta} = (N'N + \lambda\Omega_N)^{-1}N'y$$

We could program this ourselves, but that would require estimating $\{\Omega_N\}_{jk} = \int N_j''(t)N_k''(t)dt$.

Instead we'll use the `gam` package to fit the data. This is an extremely popular package in `R`. Let's look at the main function of this package.

```
library(gam)
```

```
Loading required package: foreach
```

```
Loaded gam 1.22-2
```

```
?gam
```

```
Fitting Generalized Additive Models

Description:

     'gam' is used to fit generalized additive models, specified by
     giving a symbolic description of the additive predictor and a
     description of the error distribution. 'gam' uses the _backfitting
     algorithm_ to combine different smoothing or fitting methods. The
     methods currently supported are local regression and smoothing
     splines.
```

Usage:

```
     gam(
       formula,
       family = gaussian,
       data,
       weights,
       subset,
       na.action,
       start = NULL,
       etastart,
       mustart,
       control = gam.control(...),
       model = TRUE,
       method = "glm.fit",
       x = FALSE,
       y = TRUE,
       ...
     )

     gam.fit(
       x,
       y,
       smooth.frame,
       weights = rep(1, nobs),
       start = NULL,
       etastart = NULL,
       mustart = NULL,
       offset = rep(0, nobs),
       family = gaussian(),
       control = gam.control()
     )
```

Arguments:

```
 formula: a formula expression as for other regression models, of the
          form 'response ~ predictors'. See the documentation of 'lm'
          and 'formula' for details.  Built-in nonparametric smoothing
          terms are indicated by 's' for smoothing splines or 'lo' for
          'loess' smooth terms.  See the documentation for 's' and 'lo'
          for their arguments. Additional smoothers can be added by
          creating the appropriate interface functions. Interactions
          with nonparametric smooth terms are not fully supported, but
          will not produce errors; they will simply produce the usual
          parametric interaction.

  family: a description of the error distribution and link function to
          be used in the model. This can be a character string naming a
          family function, a family function or the result of a call to
          a family function. (See 'family' for details of family
          functions.)

    data: an optional data frame containing the variables in the model.
          If not found in 'data', the variables are taken from
```

'environment(formula)', typically the environment from which 'gam' is called.

weights: an optional vector of weights to be used in the fitting process.

subset: an optional vector specifying a subset of observations to be used in the fitting process.

na.action: a function which indicates what should happen when the data contain 'NA's. The default is set by the 'na.action' setting of 'options', and is 'na.fail' if that is unset. The "factory-fresh" default is 'na.omit'. A special method 'na.gam.replace' allows for mean-imputation of missing values (assumes missing at random), and works gracefully with 'gam'

start: starting values for the parameters in the additive predictor.

etastart: starting values for the additive predictor.

mustart: starting values for the vector of means.

control: a list of parameters for controlling the fitting process. See the documentation for 'gam.control' for details. These can also be set as arguments to 'gam()' itself.

model: a logical value indicating whether _model frame_ should be included as a component of the returned value. Needed if 'gam' is called and predicted from inside a user function. Default is 'TRUE'.

method: the method to be used in fitting the parametric part of the model. The default method '"glm.fit"' uses iteratively reweighted least squares (IWLS). The only current alternative is '"model.frame"' which returns the model frame and does no fitting.

x, y: For 'gam': logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value.

For 'gam.fit': 'x' is a model matrix of dimension 'n * p', and 'y' is a vector of observations of length 'n'.

...: further arguments passed to or from other methods.

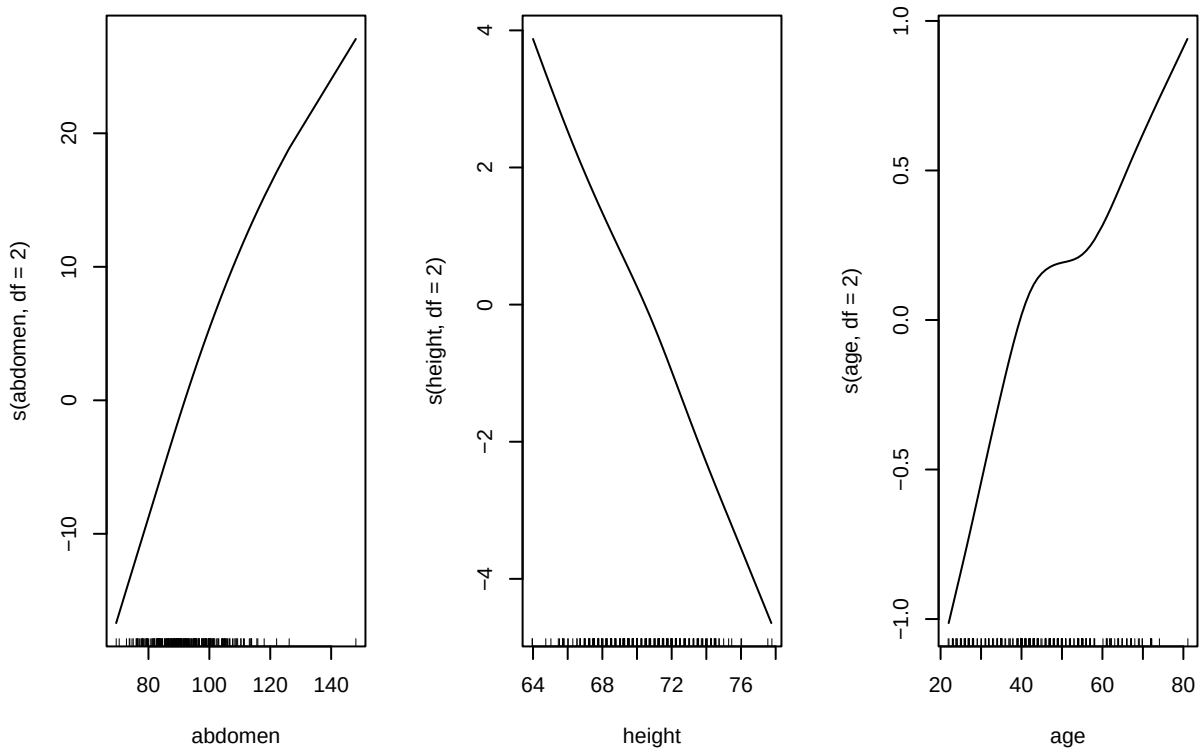
smooth.frame: for 'gam.fit' only. This is essentially a subset of the model frame corresponding to the smooth terms, and has the ingredients needed for smoothing each variable in the backfitting algorithm. The elements of this frame are produced by the formula functions 'lo' and 's'.

offset: this can be used to specify an _a priori_ known component to be included in the additive predictor during fitting.

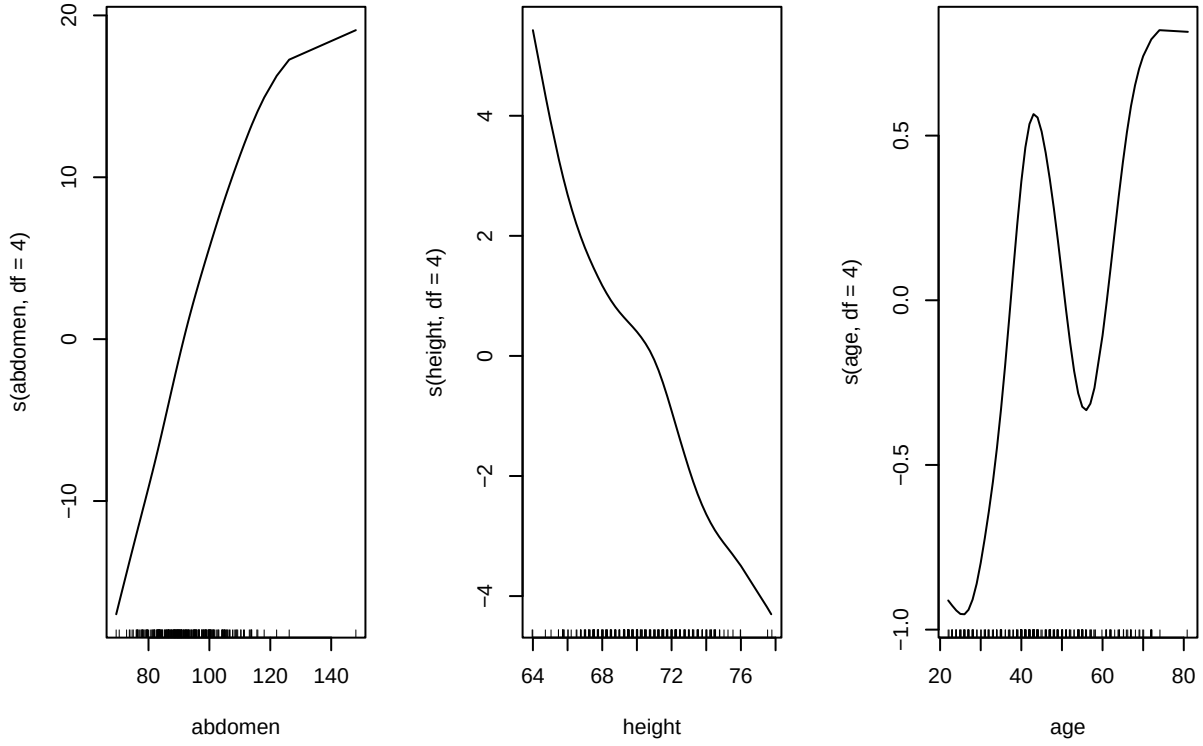
```
?s
```

```
Specify a Smoothing Spline Fit in a GAM Formula

Description:

     A symbolic wrapper to indicate a smooth term in a formala argument
     to gam

Usage:

     gam.s(x, y, w = rep(1, length(x)), df = 4, spar = 1, xeval)

     s(x, df = 4, spar = 1)

Arguments:

       x: the univariate predictor, or expression, that evaluates to a
          numeric vector.

       y: a response variable passed to 'gam.s' during backfitting

       w: weights

      df: the target equivalent degrees of freedom, used as a smoothing
          parameter. The real smoothing parameter ('spar' below) is
          found such that 'df=tr(S)-1', where 'S' is the implicit
          smoother matrix. Values for 'df' should be greater than '1',
          with 'df=1' implying a linear fit. If both 'df' and 'spar'
          are supplied, the former takes precedence. Note that 'df' is
          not necessarily an integer.

    spar: can be used as smoothing parameter, with values typically in
          '(0,1]'. See 'smooth.spline' for more details.

   xeval: If this argument is present, then 'gam.s' produces a
          prediction at 'xeval'.
```

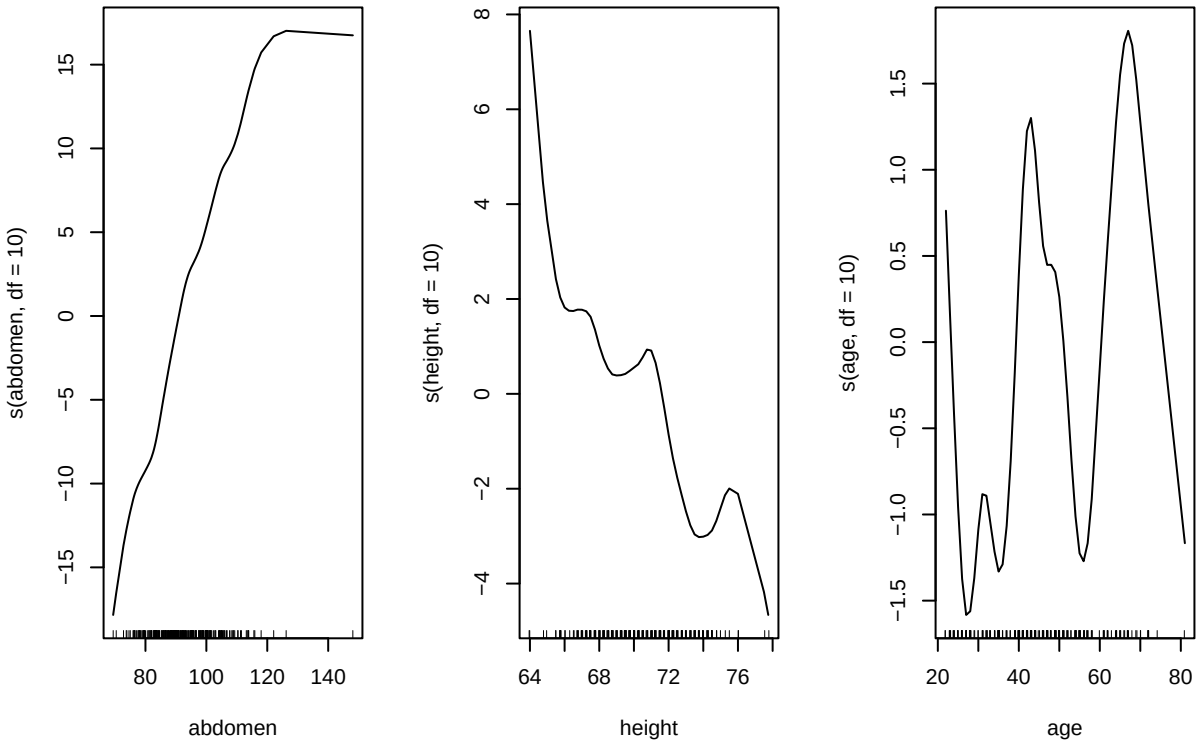
```
b <- gam(bodyfat~s(abdomen,df=2)+
           s(height,df=2)+
           s(age,df=2),
         data=bf_df_subset)
par(mfrow=c(1,3))
plot(b)
```



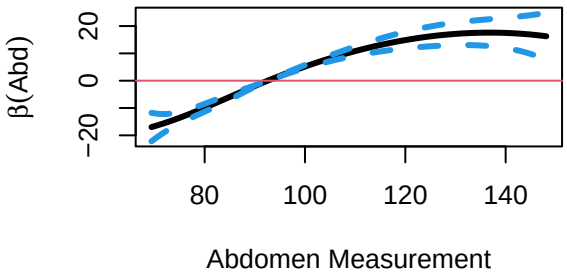

```
b <- gam(bodyfat~s(abdomen,df=4)+
           s(height,df=4)+
           s(age,df=4),
         data=bf_df_subset)
plot(b)
```

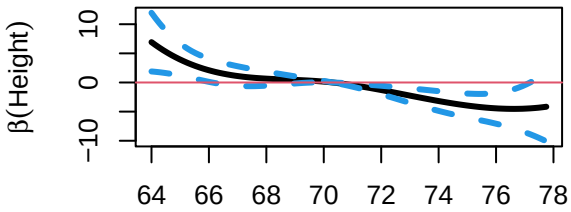


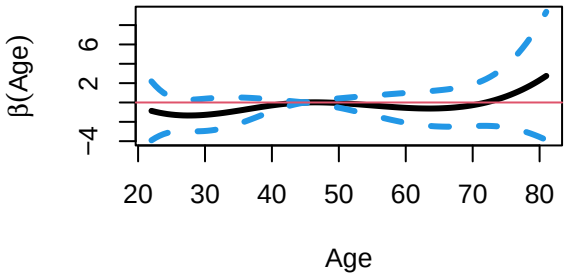
```
b <- gam(bodyfat~s(abdomen,df=10)+
           s(height,df=10)+
           s(age,df=10),
         data=bf_df_subset)
plot(b)
```



Now, let's look at a re-fit of the previous results. This time the B-spline basis matrices are centered at the mean value of X, thus making all results relative to the mean.








```
library(mgcv)
```

```
Loading required package: nlme
This is mgcv 1.8-42. For overview type 'help("mgcv-package")'.

Attaching package: 'mgcv'
The following objects are masked from 'package:gam':

    gam, gam.control, gam.fit, s
```

```
?smooth.construct.bs.smooth.spec
```

```
Penalized B-splines in GAMs

Description:

     'gam' can use smoothing splines based on univariate B-spline bases
     with derivative based penalties, specified via terms like
     's(x,bs="bs",m=c(3,2))'. 'm[1]' controls the spline order, with
     'm[1]=3' being a cubic spline, 'm[1]=2' being quadratic, and so
     on. The integrated square of the 'm[2]'th derivative is used as
     the penalty. So 'm=c(3,2)' is a conventional cubic spline. Any
     further elements of 'm', after the first 2, define the order of
     derivative in further penalties. If 'm' is supplied as a single
     number, then it is taken to be 'm[1]' and 'm[2]=m[1]-1', which is
     only a conventional smoothing spline in the 'm=3', cubic spline
     case. Notice that the definition of the spline order in terms of
     'm[1]' is intuitive, but differs to that used with the 'tprs' and
     'p.spline' bases. See details for options for controlling the
     interval over which the penalty is evaluated (which can matter if
     it is necessary to extrapolate).
```

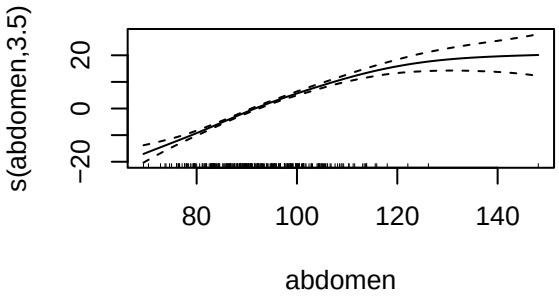
Now let's fit `gam` to the bodyfat example using the `mgvc` package. The `gam` function in the `mgvc` package does not let the user specify the λ parameter. Here the λ parameter is selected using REML. For more details on the `gam` function in the `mgvc` package see https://stat.ethz.ch/R-manual/R-devel/library/mgcv/html/gam.html.

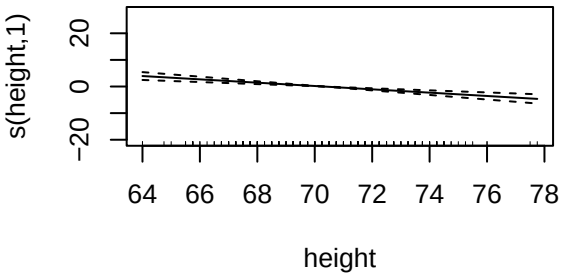
The nice thing about these plots it that they come with confidence intervals.

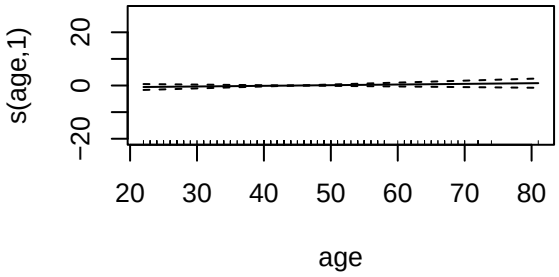
```
b <- gam(bodyfat~s(abdomen, bs="bs", m=c(3,2)) +
           s(height,bs="bs",m=c(3,2)) +
           s(age,bs="bs",m=c(3,2)),
         data=bf_df_subset, method="REML")
summary(b)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## bodyfat ~ s(abdomen, bs = "bs", m = c(3, 2)) + s(height, bs = "bs",
##     m = c(3, 2)) + s(age, bs = "bs", m = c(3, 2))
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  19.1508     0.2785   68.78   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##               edf Ref.df       F  p-value
## s(abdomen) 3.505  4.273 133.469  < 2e-16 ***
## s(height)  1.000  1.000  28.445 3.05e-07 ***
## s(age)     1.001  1.002   1.032    0.311
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.721   Deviance explained = 72.7%
## -REML = 727.88  Scale est. = 19.54     n = 252
```

```
plot(b,pages=1)
```







Example 2: Nonparametric Logistic Regression

Here we'll do some analysis on the wpbc data set, where we're trying to predict R = recurrent, N = nonrecurrent. Here we'll only use the first predictor, which was significant when regular logistic regression was completed.

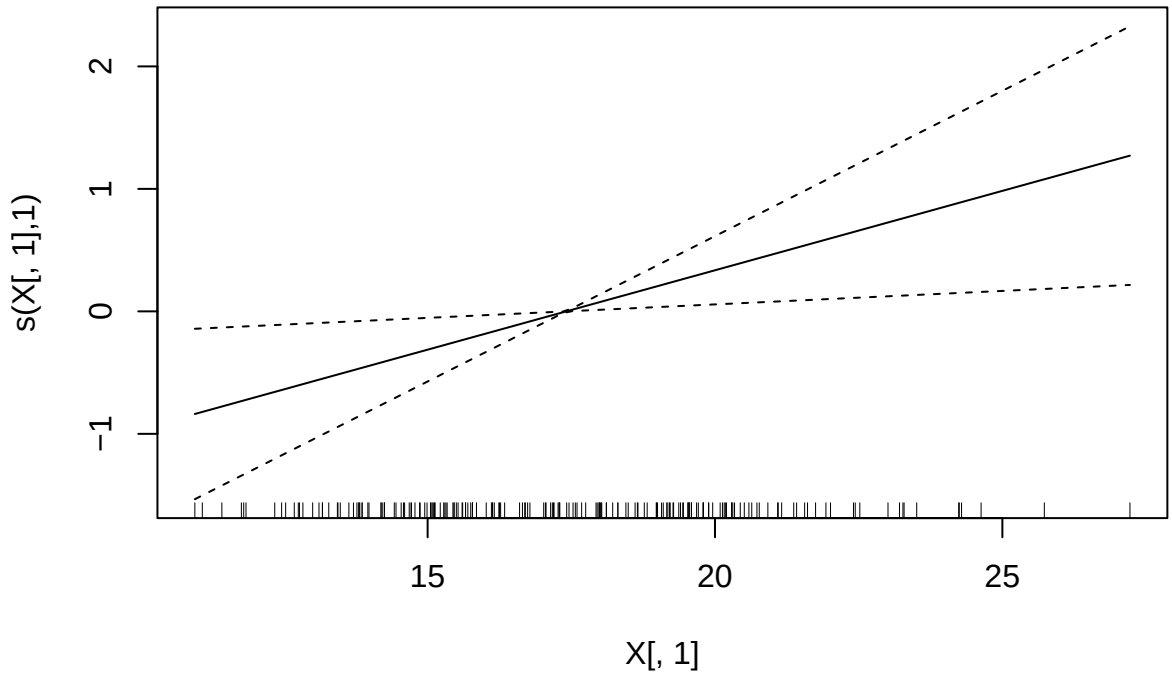
```
wpbc <- read.csv("wpbc.csv")
X <- matrix(as.numeric(unlist(wpbc[,4:32])),198,29)
K <- 1*I( wpbc[,2] == "R")
library(mgcv)
wpbc_gam <- gam(K~s(X[,1],bs="bs",m=c(4,2)),
                family=binomial(),
                method="ML")
summary(wpbc_gam)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## K ~ s(X[, 1], bs = "bs", m = c(4, 2))
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.2096     0.1731  -6.988 2.78e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##           edf Ref.df Chi.sq p-value
```

```
## s(X[, 1])   1      1  5.802   0.016 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0268   Deviance explained = 2.75%
## -ML = 105.53  Scale est. = 1         n = 198
```

Note that the edf=1, meaning that only a linear operator was used (with constant derivative).

```
plot(wpbc_gam)
```



Example 3: two-dimensional tensor regression

We'll now look at an example which fits a multidimensional smoothing spline using *tensor product basis*. This action will be performed using the `mgcv` package we discussed in the last example.

```
?te
```

```
Define tensor product smooths or tensor product interactions in GAM
formulae

Description:

     Functions used for the definition of tensor product smooths and
     interactions within 'gam' model formulae. 'te' produces a full
     tensor product smooth, while 'ti' produces a tensor product
     interaction, appropriate when the main effects (and any lower
     interactions) are also present.

     The functions do not evaluate the smooth - they exists purely to
     help set up a model using tensor product based smooths. Designed
     to construct tensor products from any marginal smooths with a
     basis-penalty representation (with the restriction that each
     marginal smooth must have only one penalty).
```

Usage:

```
     te(..., k=NA,bs="cr",m=NA,d=NA,by=NA,fx=FALSE,
                  np=TRUE,xt=NULL,id=NULL,sp=NULL,pc=NULL)
     ti(..., k=NA,bs="cr",m=NA,d=NA,by=NA,fx=FALSE,
                  np=TRUE,xt=NULL,id=NULL,sp=NULL,mc=NULL,pc=NULL)
```

Arguments:

```
     ...: a list of variables that are the covariates that this smooth
          is a function of. Transformations whose form depends on the
          values of the data are best avoided here: e.g. 'te(log(x),z)'
          is fine, but 'te(I(x/sd(x)),z)' is not (see 'predict.gam').

       k: the dimension(s) of the bases used to represent the smooth
          term.  If not supplied then set to '5^d'. If supplied as a
          single number then this basis dimension is used for each
          basis. If supplied as an array then the elements are the
          dimensions of the component (marginal) bases of the tensor
          product. See 'choose.k' for further information.

      bs: array (or single character string) specifying the type for
          each marginal basis. '"cr"' for cubic regression spline;
          '"cs"' for cubic regression spline with shrinkage; '"cc"' for
          periodic/cyclic cubic regression spline; '"tp"' for thin
          plate regression spline; '"ts"' for t.p.r.s. with extra
          shrinkage. See 'smooth.terms' for details and full list. User
          defined bases can also be used here (see 'smooth.construct'
          for an example). If only one basis code is given then this is
          used for all bases.

       m: The order of the spline and its penalty (for smooth classes
          that use this) for each term.  If a single number is given
          then it is used for all terms. A vector can be used to supply
          a different 'm' for each margin. For marginals that take
          vector 'm' (e.g. 'p.spline' and 'Duchon.spline'), then a list
          can be supplied, with a vector element for each margin. 'NA'
          autoinitializes.  'm' is ignored by some bases (e.g. '"cr"').

       d: array of marginal basis dimensions. For example if you want a
          smooth for 3 covariates made up of a tensor product of a 2
          dimensional t.p.r.s. basis and a 1-dimensional basis, then
          set 'd=c(2,1)'. Incompatibilities between built in basis
          types and dimension will be resolved by resetting the basis
          type.

      by: a numeric or factor variable of the same dimension as each
          covariate.  In the numeric vector case the elements multiply
          the smooth evaluated at the corresponding covariate values (a
          `varying coefficient model' results).  In the factor case
          causes a replicate of the smooth to be produced for each
          factor level. See 'gam.models' for further details. May also
          be a matrix if covariates are matrices: in this case
```

implements linear functional of a smooth (see 'gam.models' and 'linear.functional.terms' for details).

fx: indicates whether the term is a fixed d.f. regression spline ('TRUE') or a penalized regression spline ('FALSE').

np: 'TRUE' to use the `normal parameterization' for a tensor product smooth. This represents any 1-d marginal smooths via parameters that are function values at `knots', spread evenly through the data. The parameterization makes the penalties easily interpretable, however it can reduce numerical stability in some cases.

xt: Either a single object, providing any extra information to be passed to each marginal basis constructor, or a list of such objects, one for each marginal basis.

id: A label or integer identifying this term in order to link its smoothing parameters to others of the same type. If two or more smooth terms have the same 'id' then they will have the same smoothing paramsters, and, by default, the same bases (first occurance defines basis type, but data from all terms used in basis construction).

sp: any supplied smoothing parameters for this term. Must be an array of the same length as the number of penalties for this smooth. Positive or zero elements are taken as fixed smoothing parameters. Negative elements signal auto-initialization. Over-rides values supplied in 'sp' argument to 'gam'. Ignored by 'gamm'.

mc: For 'ti' smooths you can specify which marginals should have centering constraints applied, by supplying 0/1 or 'FALSE'/'TRUE' values for each marginal in this vector. By default all marginals are constrained, which is what is appropriate for, e.g., functional ANOVA models. Note that ''ti'' only applies constraints to the marginals, so if you turn off all marginal constraints the term will have no identifiability constraints. Only use this if you really understand how marginal constraints work.

pc: If not 'NULL', signals a point constraint: the smooth should pass through zero at the point given here (as a vector or list with names corresponding to the smooth names). Never ignored if supplied. See 'identifiability'.

Details:

Smooths of several covariates can be constructed from tensor products of the bases used to represent smooths of one (or sometimes more) of the covariates. To do this `marginal' bases are produced with associated model matrices and penalty matrices, and these are then combined in the manner described in 'tensor.prod.model.matrix' and 'tensor.prod.penalties', to produce

a single model matrix for the smooth, but multiple penalties (one for each marginal basis). The basis dimension of the whole smooth is the product of the basis dimensions of the marginal smooths.

Tensor product smooths are especially useful for representing functions of covariates measured in different units, although they are typically not quite as nicely behaved as t.p.r.s. smooths for well scaled covariates.

It is sometimes useful to investigate smooth models with a main-effects + interactions structure, for example

```
f_1(x)  + f_2(z) + f_3(x,z)
```

This functional ANOVA decomposition is supported by 'ti' terms, which produce tensor product interactions from which the main effects have been excluded, under the assumption that they will be included separately. For example the '~ ti(x) + ti(z) + ti(x,z)' would produce the above main effects + interaction structure. This is much better than attempting the same thing with 's'or 'te' terms representing the interactions (although mgcv does not forbid it). Technically 'ti' terms are very simple: they simply construct tensor product bases from marginal smooths to which identifiability constraints (usually sum-to-zero) have already been applied: correct nesting is then automatic (as with all interactions in a GLM framework). See Wood (2017, section 5.6.3).

The `normal parameterization' ('np=TRUE') re-parameterizes the marginal smooths of a tensor product smooth so that the parameters are function values at a set of points spread evenly through the range of values of the covariate of the smooth. This means that the penalty of the tensor product associated with any particular covariate direction can be interpreted as the penalty of the appropriate marginal smooth applied in that direction and averaged over the smooth. Currently this is only done for marginals of a single variable. This parameterization can reduce numerical stability when used with marginal smooths other than '"cc"', '"cr"' and '"cs"': if this causes problems, set 'np=FALSE'.

Note that tensor product smooths should not be centred (have identifiability constraints imposed) if any marginals would not need centering. The constructor for tensor product smooths ensures that this happens.

The function does not evaluate the variable arguments.

```r
test1 <- function(x,z,sx=0.3,sz=0.4) {
  x <- x*20
  (pi**sx*sz)*(1.2*exp(-(x-0.2)^2/sx^2-(z-0.3)^2/sz^2)+
  0.8*exp(-(x-0.7)^2/sx^2-(z-0.8)^2/sz^2))
}
n <- 500

#Generate x, z, f and y
```

```r
x <- runif(n)/20
z <- runif(n);
f <- test1(x,z)
y <- f + rnorm(n)*0.2

#Generate a fine lattice of x and z, then evaluate the true f.
xs <- seq(0,1,length=30)/20
zs <- seq(0,1,length=30)
pr <- data.frame(x=rep(xs,30),z=rep(zs,each=30))
truth <- matrix(test1(pr$x,pr$z),30,30)


#Fit a full tensor product smooth
b2 <- gam(y~te(x,z),method = "REML")
summary(b2)
```

```
Family: gaussian
Link function: identity

Formula:
y ~ te(x, z)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.306166   0.008638   35.44   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
          edf Ref.df     F p-value
te(x,z) 16.37  19.78 19.07  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.429   Deviance explained = 44.8%
-REML = -89.677  Scale est. = 0.037309  n = 500
```

```r
#Estimated smoothing parameters for the model.
b2$sp
```

```
te(x,z)1 te(x,z)2
3.007385 4.204517
```

```r
#Fit an ANOVA type tensor product interaction
b3 <- gam(y~ ti(x) + ti(z) + ti(x,z),
          method = "REML")
summary(b3)
```

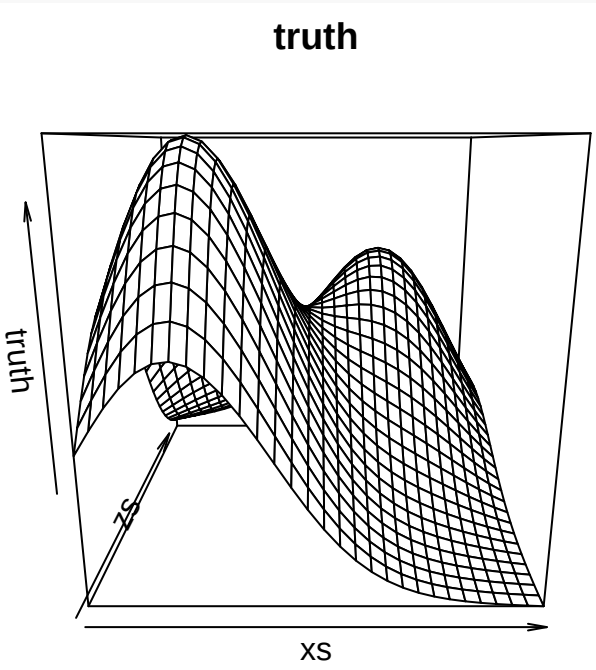
```
Family: gaussian
Link function: identity

Formula:
y ~ ti(x) + ti(z) + ti(x, z)
```

```
Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.313940   0.008774   35.78   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
          edf Ref.df     F p-value
ti(x)   3.632  3.916 30.71  <2e-16 ***
ti(z)   3.167  3.630 13.53  <2e-16 ***
ti(x,z) 9.399 12.012 17.49  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.429   Deviance explained = 44.8%
-REML = -88.059  Scale est. = 0.0373    n = 500
```

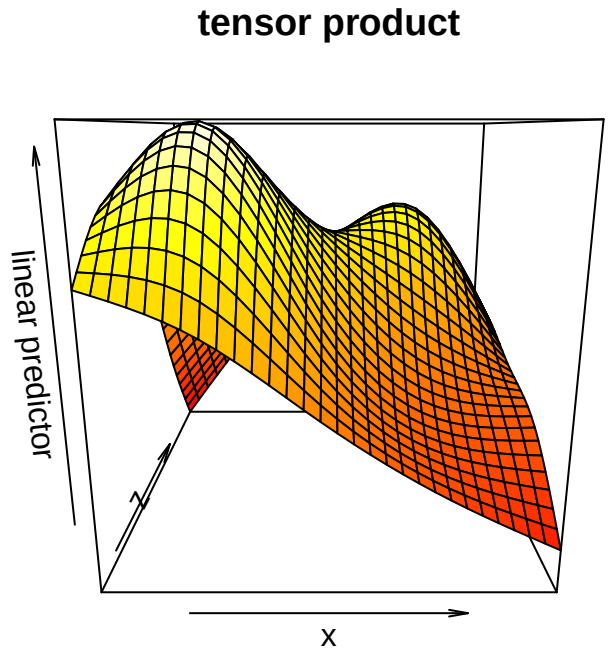
```
#Estimated smoothing parameters for the model.
b3$sp
```

```
    ti(x)     ti(z)  ti(x,z)1  ti(x,z)2
 7.471652 27.376337  2.290789  1.790958
```

```
#Plot the results.
persp(xs,zs,truth);title("truth")
```




```
vis.gam(b2);title("tensor product")
```




```
vis.gam(b3);title("tensor anova")
```