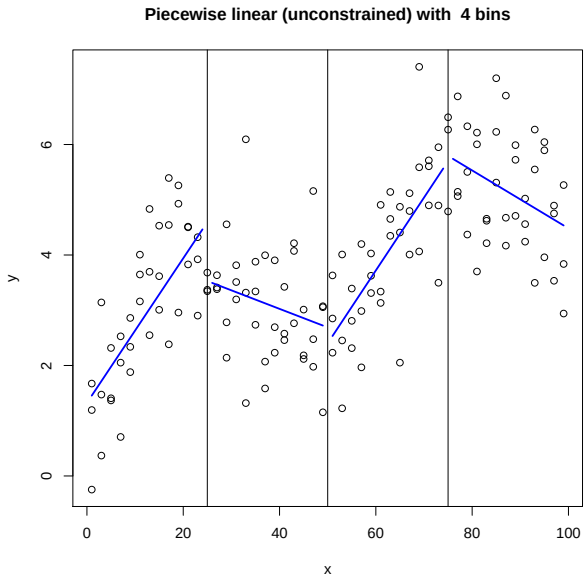
Machine Learning II
Week 3 Lecture – 24th October 2022
Non-linear regression methods 3

Contents

- Local Regression
- General Additive Models

Local Regression

Weeks 1 & 2: One of the non-linear function prediction methods was to piecewise linear regression. A regression line is fitted to each intervall



$$y_i = f(x_i) = f_0(x_i)C_0(x_i) + f_0(x_i)C_0(x_i) + \cdots f_K(x_i)C_K(x_i) + \epsilon_i$$

where each $f_k(x_i) = \beta_{0k} + \beta_{1k}x$ is a linear function

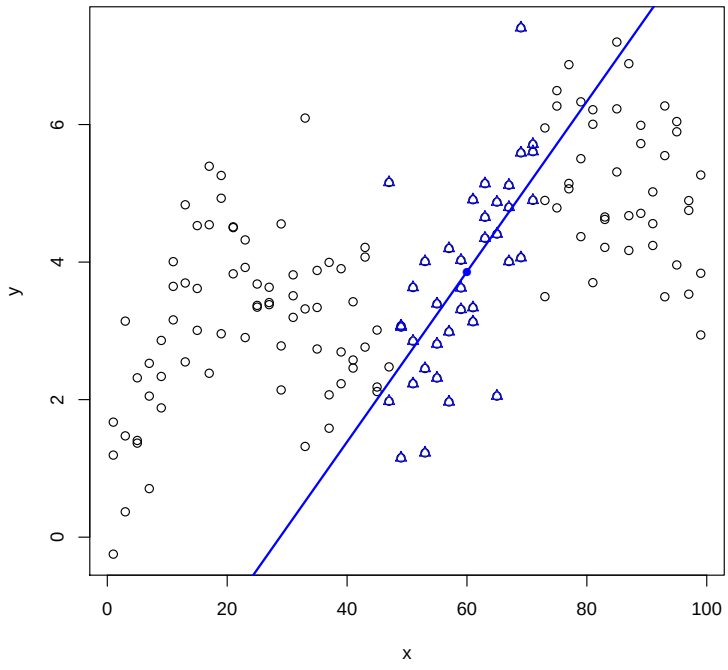
We quickly moved on to adding constraints such as the piecewise function should be continuous and smooth.

This week we will take this idea in another direction: Instead of four disjoint intervals each with a quarter of the points, we will use many overlapping intervals. We consider a *window* containing a quarter of the data points passing through the range of x.

To estimate $f(x_0)$, instead of finding the disjoint interval which contains x_0, we define a window containing take the $\frac{1}{4}n$ points nearest to x_0.

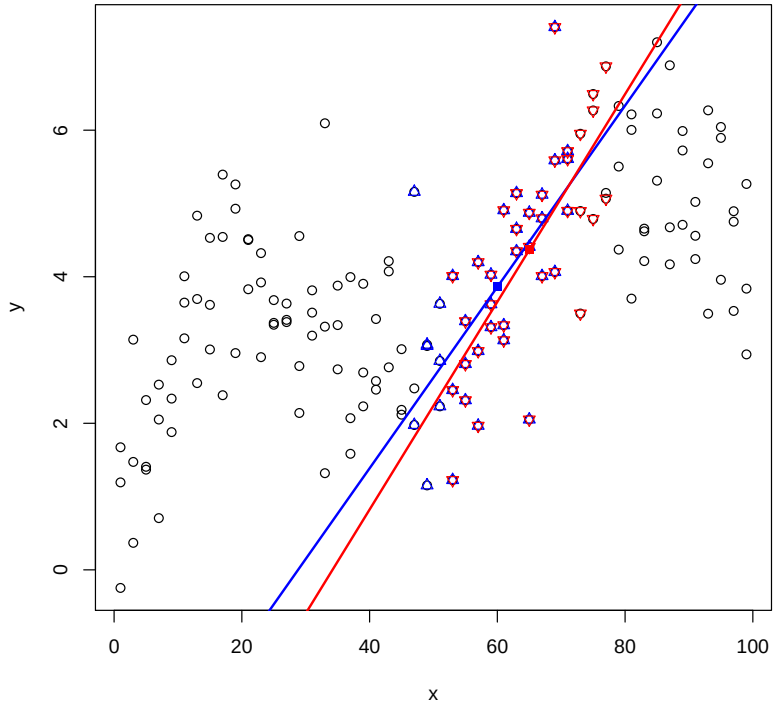
Because we are using $\frac{1}{4}n$ points, the *span* of the window is equal to $\frac{1}{4}$.

Example: $x_0 = 60$ and $n = 150$, so $\lceil \frac{n}{4} \rceil = 38$[1].
The regression line using the 38 points nearest to x_0 is $f(x) = -3.6 + 0.12x$, giving $f(x_0) = f(60) = 3.6$.



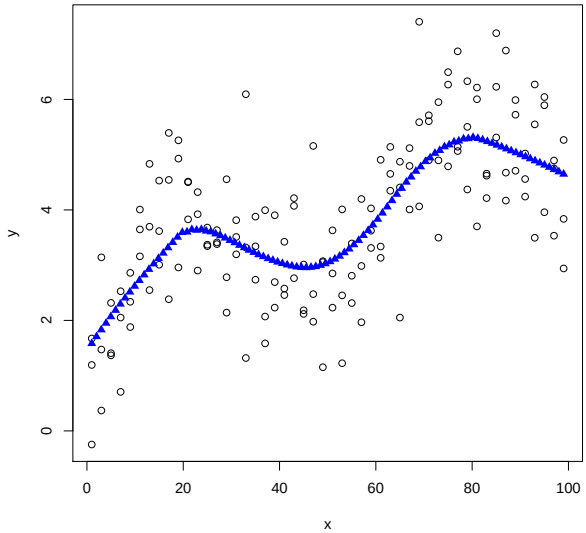
[1] non-integers are rounded up.

To predict $f(65)$ we use the 38 nearest points to $x_0 = 65$ which includes 31 of the blue points on the previous slide.



Blue: $x_0 = 60$ Red: $x_0 = 65$

We then repeat for all the points on a grid and plot them to give the predictor function $f(x)$.



Note that the method does not require that x_0 is a value in our data, we just need to identify which points are nearest to x_0.

The local regression algorithm, called *loess* (locally estimated least squares) or *lowess*, has a couple of important difference to the above explanation.

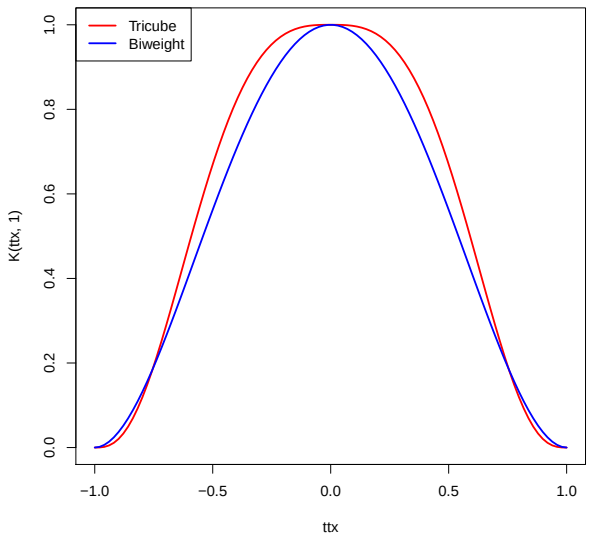
In the above example we computed the regression line giving equal weight to all points in the window.

Loess uses weighted regression, so that the points close to x_0 have a large influence on f.
Points further away, but still within the window, have less influence.
Points outside the window have zero influence.

The weight function is defined using the distance of each x_i from x_0.
The function used to calculate the weights is called a Kernel $K(\delta)$.
In loess $\delta \propto x_i - x_0$, so the further away a data point is from x_0, the smaller $K(\delta)$ is.
There are lots of possibilities for the Kernel. They are all symmetric about the y axis, have a maximum at δ=0 and are monotonic decreasing in $|\delta|$. There is a computational advantage when $K(\delta) = 0$ when δ is outside the window.

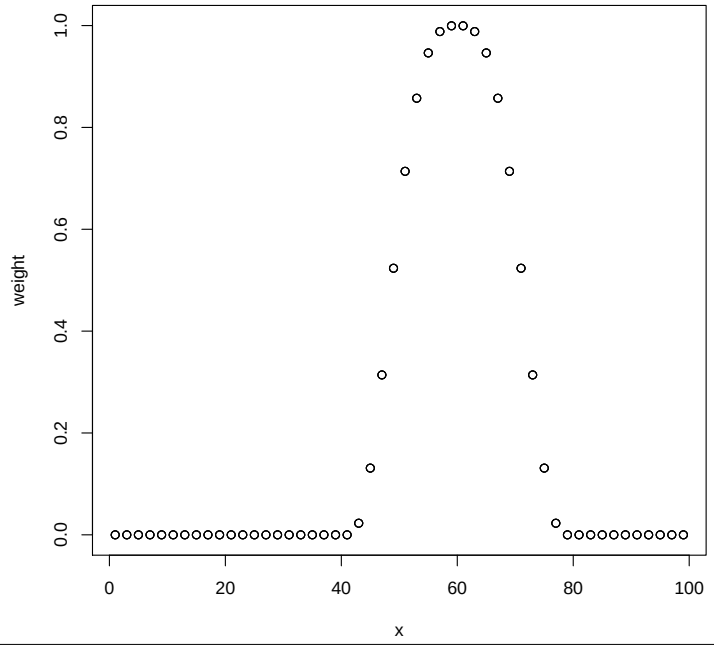
The two most common are Kernel functions for loess fitting are
Biweight $K(\delta) = (1 - \delta^2)^2$ for $|\delta| \leqslant 1$ and
Tricube $K(\delta) = (1 - |\delta|^3)^3$ for $|\delta| \leqslant 1$ (the default in R)

The unweighted example corresponds to using a *uniform* or *boxcar* kernel.
$K(\delta) = 1$ for $|\delta| \leqslant 1$



For a given x_0 the weight of the i-th point is $w_i = K\left(\frac{x_i - x_0}{s}\right)$ where s is the distance $s = \max_j |x_j - x_0|$ using the furthest x_j still inside the window.

Here are the weights for each data point with a span of 0.4 and $x_0 = 60$.As the weight is more localised than when using the uniform kernel it makes sense to use a wider span.



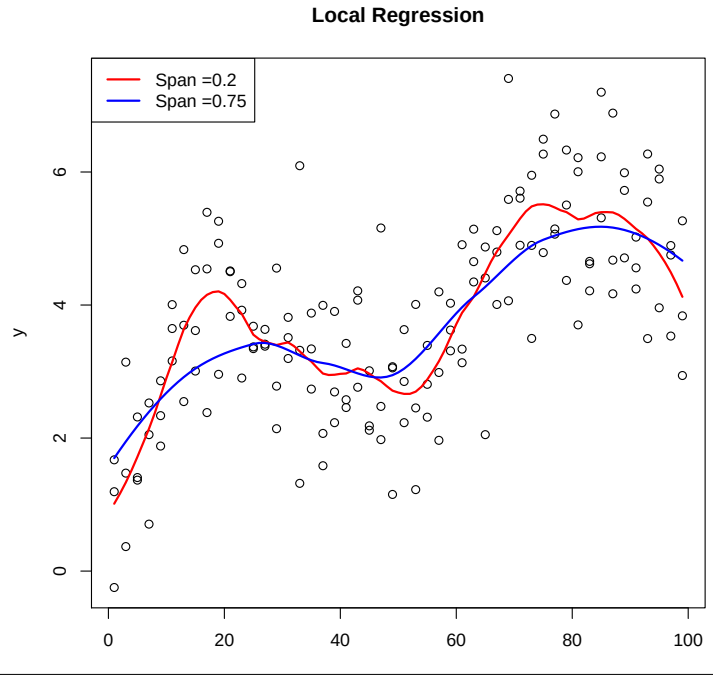
Weighted least squares: In principle the regression is carried out using standard least squares estimation, except that each residual is multiplied by the weight. Minimise $RSS = \sum_{i=1}^{n} w_i(y_i - f_\beta(x_i))^2$

The other difference in the loess implementation is that usually the regression is a locally quadratic regression instead of locally linear regression. The `loess` function in R takes an argument `degree=` to specify locally constant, linear or quadratic.

N.B. If a uniform kernel is used and a constant estimated (`degree=0`) rather than estimating a linear/quadratic regression line, then the method is equivalent to K-Nearest Neighbours regression, which was covered in ML1.

Comments on local regression

As with spline smoothing we need to specify the level of smoothing. This is done by defining the span. As the span is usually within the range $[0, 1]$, we have a reasonable starting point for the parameter unlike specifying λ in spline smoothing. The default in R is 0.75.



Loess is a good all-round method which can be used to gain a first impression of the relationship between *x* and *y*.

In R some plot methods add a loess curve by default, or make it easy to specify as an option. Example: The diagnostic plots for an R object of class `lm` add a red loess curve as a visual guide

Generalised additive Models

All the non-linear regression methods considered so far have used just one predictor variable X. We will now consider how we can adapt these methods to multiple predictor variables X_1, X_2, X_p.

A multiple linear regression model has the form.

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon_i.$$

Non-linear equivalent is the generalised additive Model (GAM)

$$Y_i = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \ldots + f_p(x_{ip}) + \epsilon_i.$$

Note: there is another class of statistical models called generalised linear models (GLMs), which include logistic regression. These are not the same!

There is no constraint that each f_j has to be the same type of function.
So f_1 could be a quadratic function, f_2 a smoothing spline function and f_3 a loess function.

As with linear models the predictor variables can be either numeric or factor variables.

For convenience, we will assume for the rest of this lecture that each component function is a smoothing spline. The following methods generalise easily to any univariate linear or non-linear regression method.

How do we estimate each of the functions in the GAM?

We can easily estimate $f_1(x_{i1})$ fitting univariate spline smoothing on (x_{i1}, y_i).

To estimate $f_2(x_{i2})$ we need to remove the fitted effect that $f_1(x_{i1})$ has on y_i before using spline smoothing on X_2.
This means we calculate $\eta_{i2} = y_i - f_1(x_{i1})$, and fit univariate spline smoothing on (x_{i2}, η_{i2}).

To estimate $f_3(x_{i3})$ we need to remove the fitted effect that $f_1(x_{i1})$ and $f_2(x_{i2})$ before using spline smoothing has on y_i before using spline smoothing on X_2.
This means we calculate $\eta_{i3} = y_i - f_1(x_{i1}) - f_2(x_{i2})$, and fit univariate spline smoothing on (x_{i3}, η_{i3}).

The η values obtained by subtracting out the other components are called partial residuals.

The *k-th partial residuals* for a GAM model is defined as

$$\eta_{ik} = y_i - \beta_0 - f_1(x_{i1}) - \ldots - f_{(k-1)}(x_{i(k-1)}) - f_{(k+1)}(x_{i(k+1)}) - \ldots f_p(x_{ip})$$

for $i = 1, \ldots, n$.

Notice all the other functions are removed **except** $f_k(x_{ik})$

The full residuals are the error terms when the full model is fitted, the *k-th partial residuals* is the error term when everything except for $f_k($ is fitted.

The method for obtaining all p estimating functions is called *backfitting*. An insight to the method is below, but the technical details are omitted.

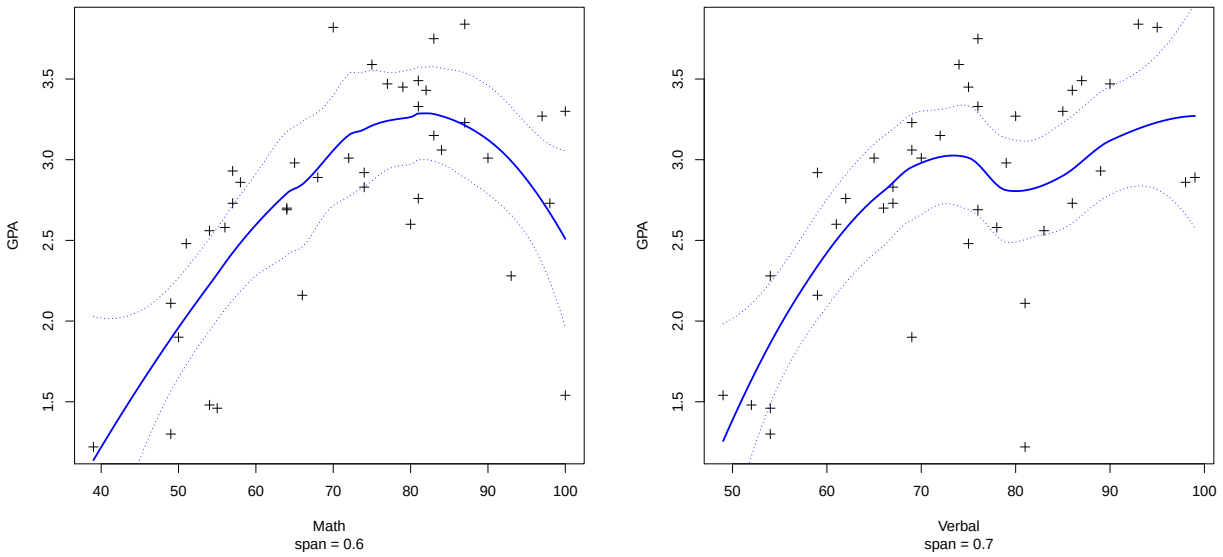
Backfitting algorithm for GAMs

1) Set the constant parameter $\widehat{\beta}_0 = \overline{y}$ this remains fixed throughout the algorithm.
2) Initialise all the component functions to $f_k(x) = 0$
3) Iterate over $k \in \{1, \ldots, p\}$
   a) Obtain the k-th partial residuals η_{ik}.
   b) Obtain the spline smoothing function $f_k(x)$, which fits the η_{ik} partial residuals.
4) Repeat Step 3 until convergence, i.e. the change in the fitted values for each the component $f_k(x)$ is less than a specified tolerance.

Example

The GPA (grade point average) for matriculating freshmen in US universities is the grade attained when completing the 1st year at university. GPA is a continuous number on a scale from 1 (bad) to 4 (very good). The data set `Freshmen` contains the GPA score along with the percentage scores for a mathematics test and a verbal test, as part of their entrance exam.

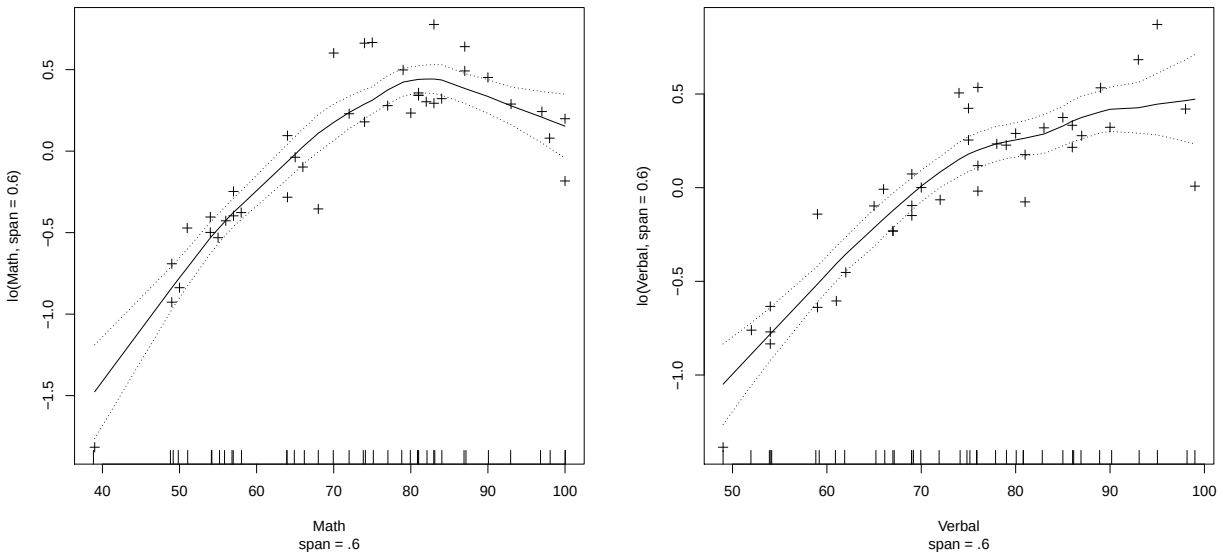
Loess curves for one variable models



The local regression *GPA dependent on math score* ignoring the verbal score seems to be non-linear, as does *GPA dependent on verbal score* ignoring the math score.

GAM with loess curve for Math and Verbal

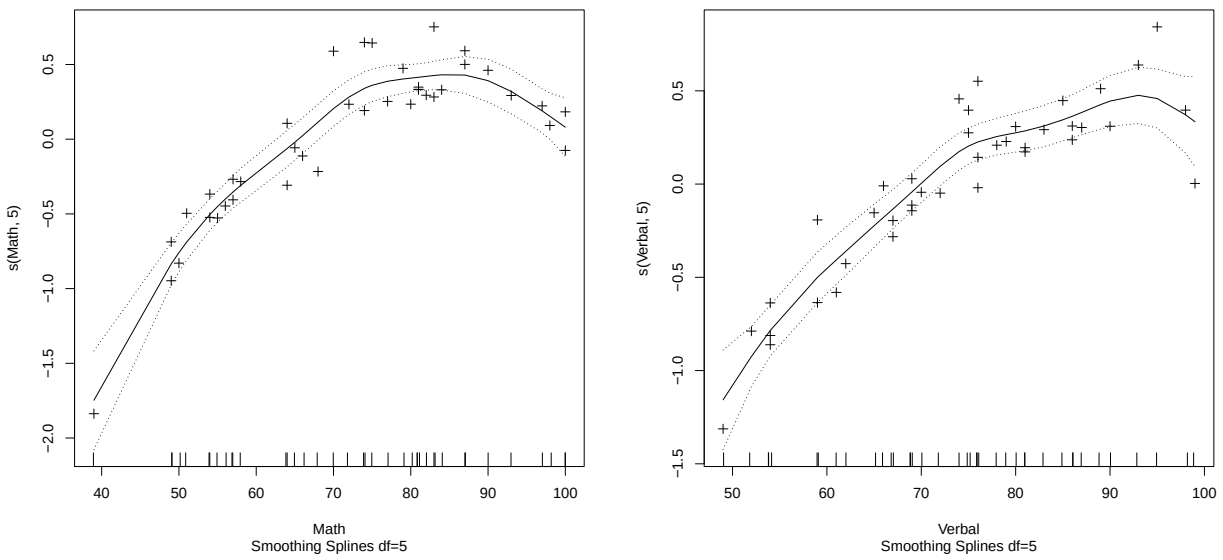
```
gam.mod=gam(GPA~lo(Math,span=0.6)+lo(Verbal,span=0.6),
        data=Freshmen)
```



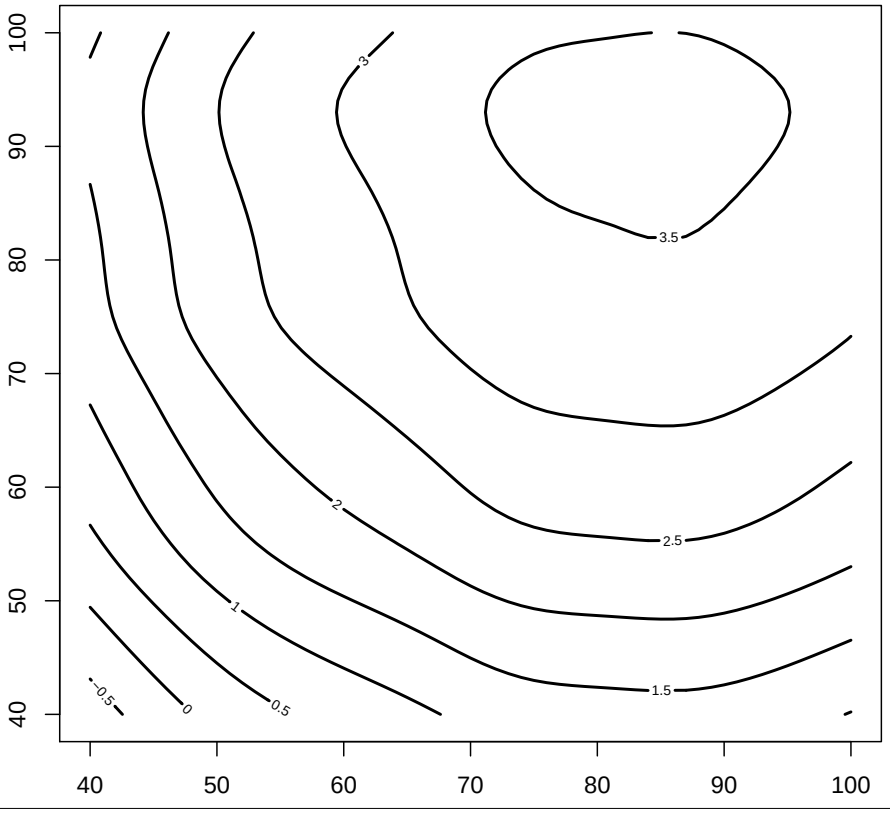
The dip in predicted GPA for verbal scores around 80 has disappeared when Math and Verbal are simultaneously fitted.
NB: the points plotted on the *y*-axis are the partial residuals and not GPA scores.

GAM with spline smoothing curve for Math and Verbal

```
gam.mod=gam(GPA~s(Math,5)+s(Verbal,5),data=Freshmen)
```



Contour plot of the predicted values using spline smoothing.



You can specify different combinations such as spline smoothing for `Math` and a quadratic polynomial for `Verbal`

```
gam.mod=gam(GPA~s(Math,5)+poly(Verbal,2),data=Freshmen)
```


Comments

- GAMs allow us to fit a non-linear predictor to each variable. We will find non-linear relationships that standard multiple linear regression will miss. We do not need to manually try out many different transformations on each variable individually.
- The non-linear fits can potentially make more accurate predictions for the response Y.
- Because the model is additive, we can examine the effect of each X_j on Y individually while holding all of the other variables fixed.
- The smoothness of the function f_j for the variable X_j can be controlled independently for each variable.

Comments ctd.

- Up to now we have assumed all of the variables are continuous, or at least numeric. Factor variables are usually modelled using one parameter for each factor level. These can be directly added to the GAM with no need for fitting a function. Eg. In the GPA data if there was a factor for type of high school: public vs private.
- A GAM is restricted to be additive. Important interactions might be missed. We can manually add an interaction term to the GAM model by adding a predictor for $X_j X_k$. This is normally only done if an interaction term between two variables is expected, because checking for all combinations of interactions adds a large computational overhead.