Nonlinear Models

Here we explore the use of nonlinear models using some tools in R

```
require(ISLR)
```

```
## Loading required package: ISLR
```

```
attach(Wage)
```

Polynomials

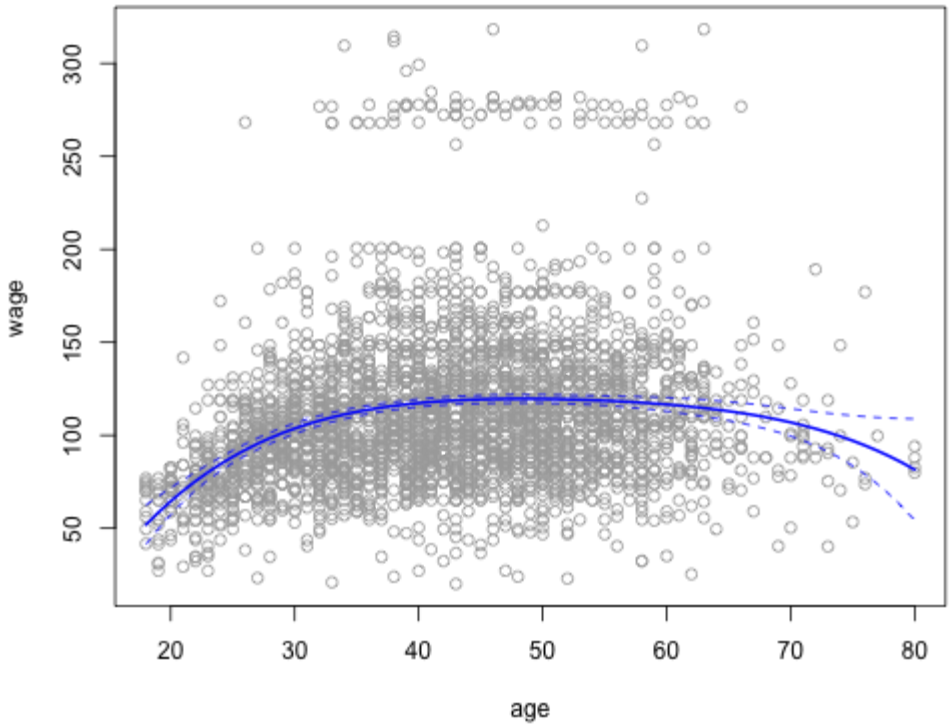
First we will use polynomials, and focus on a single predictor age:

```
fit = lm(wage ~ poly(age, 4), data = Wage)
summary(fit)
```

```
##
## Call:
## lm(formula = wage ~ poly(age, 4), data = Wage)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -98.71 -24.63  -4.99  15.22 203.69
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    111.704      0.729  153.28   <2e-16 ***
## poly(age, 4)1  447.068     39.915   11.20   <2e-16 ***
## poly(age, 4)2 -478.316     39.915  -11.98   <2e-16 ***
## poly(age, 4)3  125.522     39.915    3.14   0.0017 **
## poly(age, 4)4  -77.911     39.915   -1.95   0.0510 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.9 on 2995 degrees of freedom
## Multiple R-squared:  0.0863, Adjusted R-squared:  0.085
## F-statistic: 70.7 on 4 and 2995 DF,  p-value: <2e-16
```

The poly() function generates a basis of *orthogonal polynomials*. Lets make a plot of the fitted function, along with the standard errors of the fit.

```
agelims = range(age)
age.grid = seq(from = agelims[1], to = agelims[2])
preds = predict(fit, newdata = list(age = age.grid), se = TRUE)
se.bands = cbind(preds$fit + 2 * preds$se, preds$fit - 2 *
preds$se)
plot(age, wage, col = "darkgrey")
lines(age.grid, preds$fit, lwd = 2, col = "blue")
matlines(age.grid, se.bands, col = "blue", lty = 2)
```



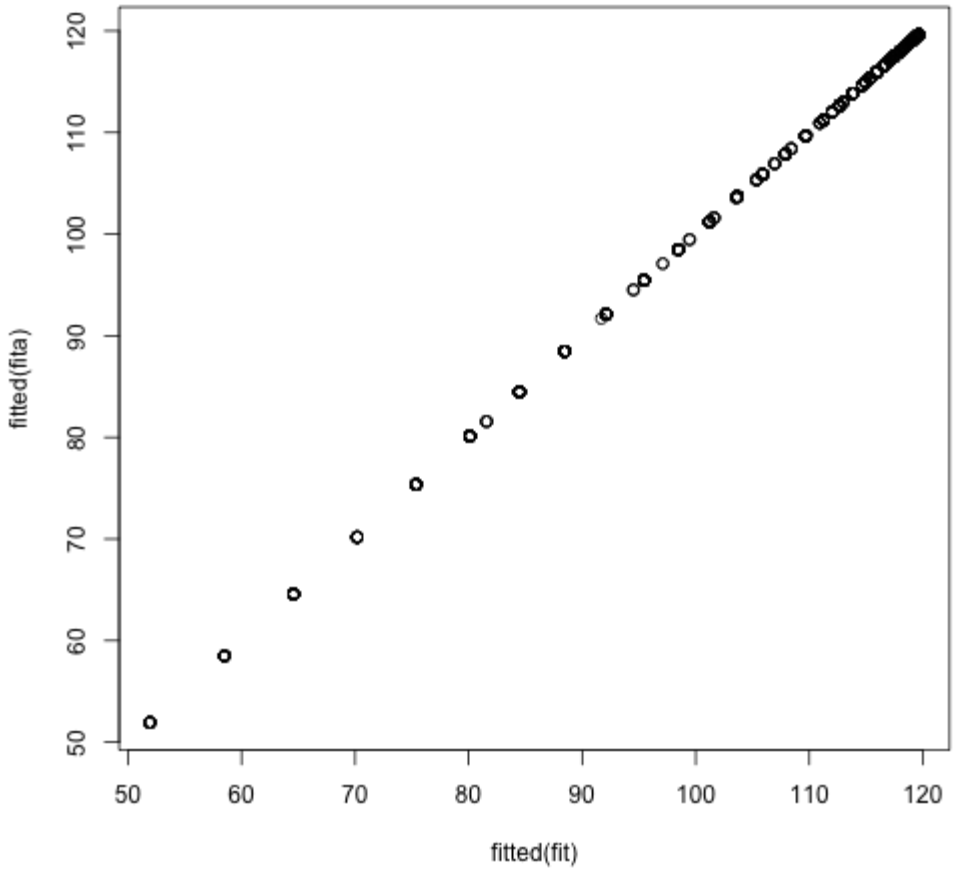
There are other more direct ways of doing this in R. For example

```
fita = lm(wage ~ age + I(age^2) + I(age^3) + I(age^4), data =
Wage)
summary(fita)
```

```
##
## Call:
## lm(formula = wage ~ age + I(age^2) + I(age^3) + I(age^4), data
= Wage)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -98.71 -24.63  -4.99  15.22 203.69
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.84e+02   6.00e+01   -3.07  0.00218 **
## age          2.12e+01   5.89e+00    3.61  0.00031 ***
## I(age^2)    -5.64e-01   2.06e-01   -2.74  0.00626 **
## I(age^3)     6.81e-03   3.07e-03    2.22  0.02640 *
## I(age^4)    -3.20e-05   1.64e-05   -1.95  0.05104 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.9 on 2995 degrees of freedom
## Multiple R-squared:  0.0863, Adjusted R-squared:  0.085
## F-statistic: 70.7 on 4 and 2995 DF,  p-value: <2e-16
```

Here `I()` is a *wrapper* function; we need it because `age^2` means something to the formula language, while `I(age^2)` is protected. The coefficients are different to those we got before! However, the fits are the same:

```
plot(fitted(fit), fitted(fita))
```



By using orthogonal polynomials in this simple way, it turns out that we can separately test for each coefficient. So if we look at the summary again, we can see that the linear, quadratic and cubic terms are significant, but not the quartic.

```
summary(fit)
```

```
##
## Call:
## lm(formula = wage ~ poly(age, 4), data = Wage)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -98.71 -24.63  -4.99  15.22 203.69
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    111.704      0.729  153.28   <2e-16 ***
## poly(age, 4)1  447.068     39.915   11.20   <2e-16 ***
## poly(age, 4)2 -478.316     39.915  -11.98   <2e-16 ***
## poly(age, 4)3  125.522     39.915    3.14   0.0017 **
## poly(age, 4)4  -77.911     39.915   -1.95   0.0510 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.9 on 2995 degrees of freedom
## Multiple R-squared:  0.0863, Adjusted R-squared:  0.085
## F-statistic: 70.7 on 4 and 2995 DF,  p-value: <2e-16
```

This only works with linear regression, and if there is a single predictor. In general we would use `anova()` as this next example demonstrates.

```
fita = lm(wage ~ education, data = Wage)
fitb = lm(wage ~ education + age, data = Wage)
fitc = lm(wage ~ education + poly(age, 2), data = Wage)
fitd = lm(wage ~ education + poly(age, 3), data = Wage)
anova(fita, fitb, fitc, fitd)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ education
## Model 2: wage ~ education + age
## Model 3: wage ~ education + poly(age, 2)
## Model 4: wage ~ education + poly(age, 3)
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1   2995 3995721
## 2   2994 3867992  1    127729 102.74 <2e-16 ***
## 3   2993 3725395  1    142597 114.70 <2e-16 ***
## 4   2992 3719809  1      5587   4.49  0.034 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Polynomial logistic regression

Now we fit a logistic regression model to a binary response variable, constructed from wage. We code the big earners (>250K) as 1, else 0.

```
fit = glm(I(wage > 250) ~ poly(age, 3), data = Wage, family =
binomial)
summary(fit)
```

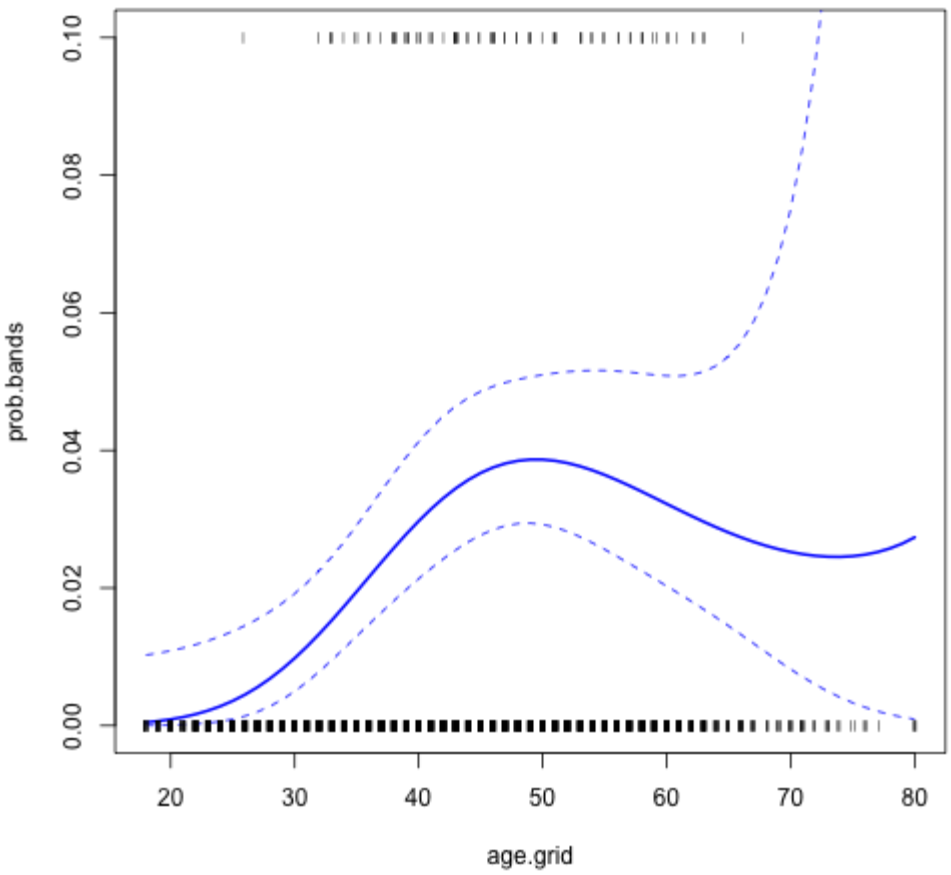
```
##
## Call:
## glm(formula = I(wage > 250) ~ poly(age, 3), family = binomial,
##     data = Wage)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -0.281  -0.274  -0.249  -0.176   3.287
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -3.85       0.16  -24.10  < 2e-16 ***
## poly(age, 3)1    37.88      11.48    3.30  0.00097 ***
## poly(age, 3)2   -29.51      10.56   -2.79  0.00520 **
## poly(age, 3)3     9.80       9.00    1.09  0.27632
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 730.53  on 2999  degrees of freedom
## Residual deviance: 707.92  on 2996  degrees of freedom
## AIC: 715.9
##
## Number of Fisher Scoring iterations: 8
```

```
preds = predict(fit, list(age = age.grid), se = T)
se.bands = preds$fit + cbind(fit = 0, lower = -2 * preds$se, upper
= 2 * preds$se)
se.bands[1:5, ]
```

```
##      fit   lower  upper
## 1 -7.665 -10.760 -4.570
## 2 -7.325 -10.107 -4.543
## 3 -7.002  -9.493 -4.511
## 4 -6.695  -8.917 -4.473
## 5 -6.405  -8.379 -4.431
```

We have done the computations on the logit scale. To transform we need to apply the inverse logit mapping \[p=\frac{e^\eta}{1+e^\eta}. \] (Here we have used the ability of MarkDown to interpret TeX expressions.) We can do this simultaneously for all three columns of `se.bands`:

```
prob.bands = exp(se.bands)/(1 + exp(se.bands))
matplot(age.grid, prob.bands, col = "blue", lwd = c(2, 1, 1), lty
= c(1, 2,
    2), type = "l", ylim = c(0, 0.1))
points(jitter(age), I(wage > 250)/10, pch = "|", cex = 0.5)
```



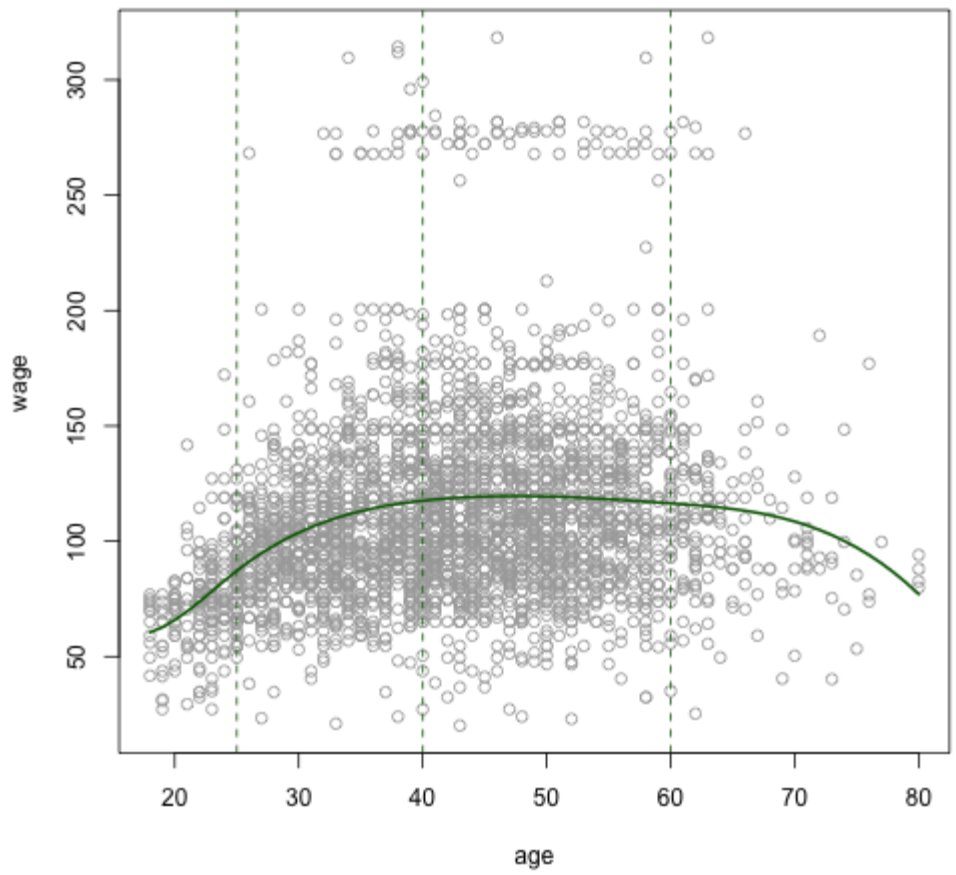
Splines

Splines are more flexible than polynomials, but the idea is rather similar. Here we will explore cubic splines.

```
require(splines)
```

```
## Loading required package: splines
```

```
 fit = lm(wage ~ bs(age, knots = c(25, 40, 60)), data = Wage)
 plot(age, wage, col = "darkgrey")
 lines(age.grid, predict(fit, list(age = age.grid)), col =
"darkgreen", lwd = 2)
 abline(v = c(25, 40, 60), lty = 2, col = "darkgreen")
```



The smoothing splines does not require knot selection, but it does have a smoothing parameter, which can conveniently be specified via the effective degrees of freedom or `df`.

```
fit = smooth.spline(age, wage, df = 16)
lines(fit, col = "red", lwd = 2)
```

```
## Error: plot.new has not been called yet
```

Or we can use LOO cross-validation to select the smoothing parameter for us automatically:

```
fit = smooth.spline(age, wage, cv = TRUE)
```

```
## Warning: cross-validation with non-unique 'x' values seems
doubtful
```

```
lines(fit, col = "purple", lwd = 2)
```

```
## Error: plot.new has not been called yet
```

```
fit
```

```
## Call:
## smooth.spline(x = age, y = wage, cv = TRUE)
##
## Smoothing Parameter  spar= 0.6989  lambda= 0.02792 (12
iterations)
## Equivalent Degrees of Freedom (Df): 6.795
## Penalized Criterion: 75216
## PRESS: 1593
```

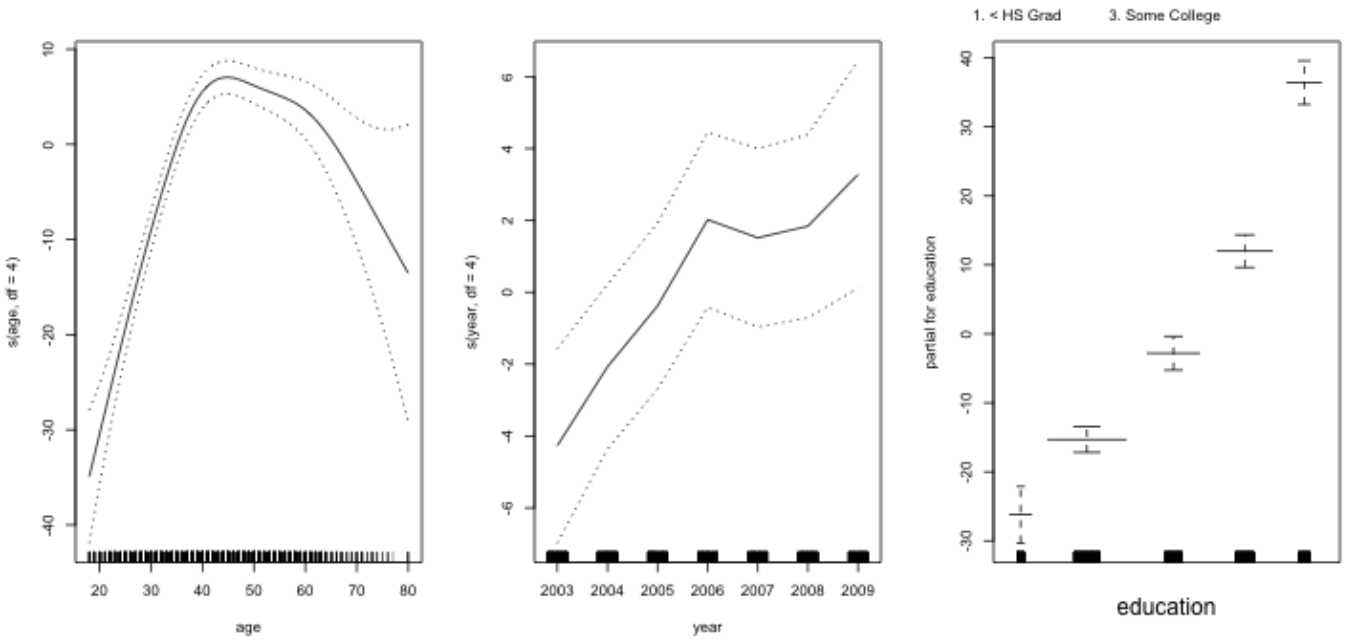
Generalized Additive Models

So far we have focused on fitting models with mostly single nonlinear terms. The *gam* package makes it easier to work with multiple nonlinear terms. In addition it knows how to plot these functions and their standard errors.

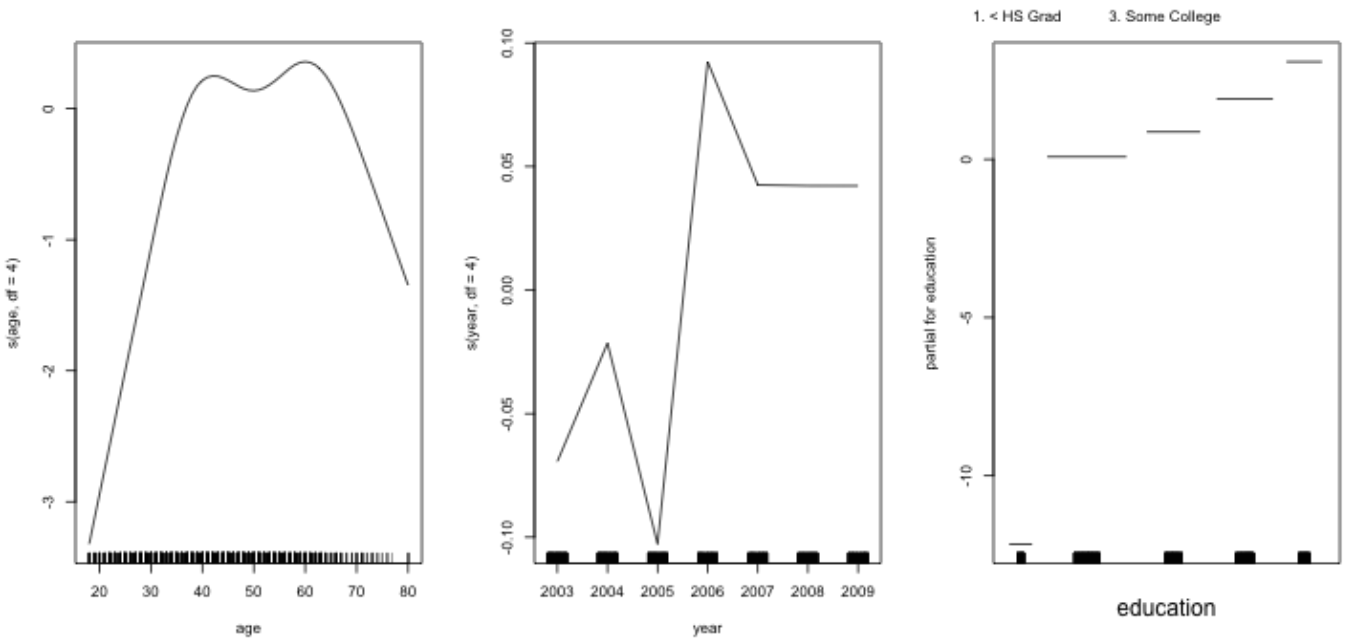
```
require(gam)
```

```
## Loading required package: gam Loaded gam 1.09
##
## Attaching package: 'gam'
##
## The following object is masked from 'package:hastie':
##
## ylim.scale
```

```
gam1 = gam(wage ~ s(age, df = 4) + s(year, df = 4) + education,
data = Wage)
par(mfrow = c(1, 3))
plot(gam1, se = T)
```




```
gam2 = gam(I(wage > 250) ~ s(age, df = 4) + s(year, df = 4) +
education, data = Wage,
    family = binomial)
plot(gam2)
```



Lets see if we need a nonlinear terms for year

```
gam2a = gam(I(wage > 250) ~ s(age, df = 4) + year + education,
data = Wage,
    family = binomial)
anova(gam2a, gam2, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: I(wage > 250) ~ s(age, df = 4) + year + education
## Model 2: I(wage > 250) ~ s(age, df = 4) + s(year, df = 4) +
education
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      2990        604
## 2      2987        603  3    0.905     0.82
```

One nice feature of the *gam* package is that it knows how to plot the functions nicely, even for models fit by `lm` and `glm`.

```
par(mfrow = c(1, 3))
lm1 = lm(wage ~ ns(age, df = 4) + ns(year, df = 4) + education,
data = Wage)
plot.gam(lm1, se = T)
```