Ch. 7 - Moving Beyond Linearity

Chris Hayduk

March 5, 2021

1 Polynomial Regression and Step Functions

Polynomial regression and step functions both exist to give a more flexible fit than standard linear regression will allow. Polynomial regression works by creating new variables out of powers of the initial prediction variable. For example, if the initial predictor is x, we may use x, x^2, and x^3 as our three predictors if we wish to use a cubic fit. On the other than, step functions work by cutitng the data up into bins and then fitting constant functions on each of these bins.

We start here by fitting a degree 4 polynomial with the `lm()` function.

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

```
##                 Estimate Std. Error    t value     Pr(>|t|)
## (Intercept)    111.70361  0.7287409 153.283015 0.000000e+00
## poly(age, 4)1  447.06785 39.9147851  11.200558 1.484604e-28
## poly(age, 4)2 -478.31581 39.9147851 -11.983424 2.355831e-32
## poly(age, 4)3  125.52169 39.9147851   3.144742 1.678622e-03
## poly(age, 4)4  -77.91118 39.9147851  -1.951938 5.103865e-02
```

The `poly()` function used above allows us to avoid writing out a long equation by hand. However, by default it returns a matrix whose columns are a basis of orthogonal polynomials. That is, each column is a linear combination of the variables age, age^2, age^3, and age^4. While the choice of basis will not affect the fitted values, we may want to use age, age^2, age^3, and age^4 directly. We can do this by setting `raw=TRUE` in the arguments for the function `poly()`.

```
fit2=lm(wage~poly(age,4,raw=T),data=Wage)
coef(summary(fit2))
```

```
##                             Estimate   Std. Error   t value     Pr(>|t|)
## (Intercept)           -1.841542e+02 6.004038e+01 -3.067172 0.0021802539
```

```
## poly(age, 4, raw = T)1  2.124552e+01 5.886748e+00  3.609042 0.0003123618
## poly(age, 4, raw = T)2 -5.638593e-01 2.061083e-01 -2.735743 0.0062606446
## poly(age, 4, raw = T)3  6.810688e-03 3.065931e-03  2.221409 0.0263977518
## poly(age, 4, raw = T)4 -3.203830e-05 1.641359e-05 -1.951938 0.0510386498
```

There are several other equivalent was of fitting this model in R. For example,

```
fit2a = lm(wage~age+I(age^2)+I(age^3)+I(age^4), data=Wage)
coef(fit2a)

##   (Intercept)           age      I(age^2)      I(age^3)      I(age^4)
## -1.841542e+02  2.124552e+01 -5.638593e-01  6.810688e-03 -3.203830e-05
```

and,

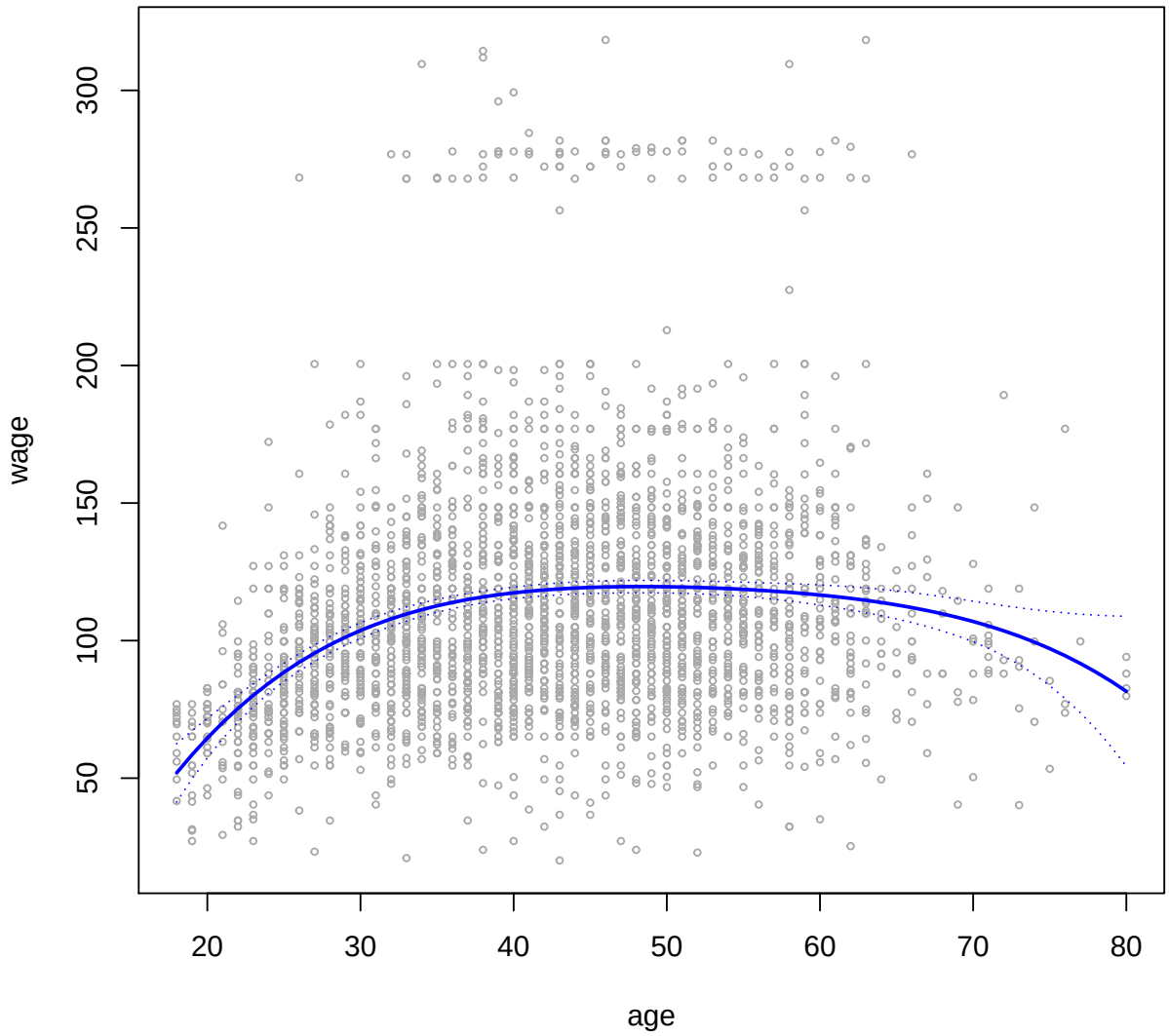
```
fit2b = lm(wage~cbind(age, age^2, age^3, age^4), data = Wage)
```

We now create a grid of values for `age` at which we want predictions, and then call the generic `predict()` function, specifying that we want standard errors as well.

```
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.fit, preds$fit-2*preds$se.fit)
```

The above code creates our grid of age values, gets predictions for each value in the grid, and then sets the standard error bars at 95% confidence (two times the standard error plus/minus the prediction). Finally, let us plot the data and add the fit from the degree-4 polynomial.

```
par(mfrow=c(1,1), mar=c(4.5,4.5,1,1), oma=c(0,0,4,0))
plot(age, wage, xlim=agelims, cex=.5, col="darkgrey")
title("Degree-4 Polynomial", outer=T)
lines(age.grid, preds$fit, lwd=2, col="blue")
matlines(age.grid, se.bands, lwd=1, col="blue", lty=3)
```



As we mentioned above, the choice of basis will not affect the fitted values of the function. We can verify this by checking the predicted values on our age grid from both types of our fitted polynomial functions:

```
preds2 = predict(fit2, newdata=list(age=age.grid), se=TRUE)
max(abs(preds$fit-preds2$fit))

## [1] 7.81597e-11
```

As we can see, the maximum of the absolute value of the difference of these two fits is extremely close to 0.

In practice, we will want to check several possible polynomial degrees in order to find the simplest model that provides the best fit. We will now fit models ranging from linear to degree-5 polynomials and use the `anova()` function to perform an analysis of variance in order to determine the best model. Analysis of variance (ANOVA) works by testing the null hypothesis that a model M_1 is sufficient to explain the data again the alternative hypothesis that a more complex model M_2 is required. In order to use the `anova()` function, M_1 and M_2 must be nested models. In this case, we are fitting five different models and wll sequentially compare the simpler model to the more complex model.

```
fit.1 = lm(wage~age, data=Wage)
fit.2 = lm(wage~poly(age,2), data=Wage)
fit.3 = lm(wage~poly(age,3), data=Wage)
fit.4 = lm(wage~poly(age,4), data=Wage)
fit.5 = lm(wage~poly(age,5), data=Wage)
anova(fit.1, fit.2, fit.3, fit.4, fit.5)

## Analysis of Variance Table
##
## Model 1: wage ~ age
## Model 2: wage ~ poly(age, 2)
## Model 3: wage ~ poly(age, 3)
## Model 4: wage ~ poly(age, 4)
## Model 5: wage ~ poly(age, 5)
##   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1   2998 5022216
## 2   2997 4793430  1    228786 143.5931 < 2.2e-16 ***
## 3   2996 4777674  1     15756   9.8888  0.001679 **
## 4   2995 4771604  1      6070   3.8098  0.051046 .
## 5   2994 4770322  1      1283   0.8050  0.369682
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The p-value comparing the linear Model 1 to the quadratic Model 2 is essentially 0, indicating that a linear fit is not sufficient. Similarly, the p-value comparing the quadratic Model 2 to the cubic Model 3 is very low, so the quadratic fit is also insufficient. The p-value comparing the cubic model and the degree-4 polynomial is approximately 5%, while the degree-5 polynomial seems unnecessary because its p-value is 0.37. Hence, either a cubic or quartic polynomial appears to provide a reasonable fit to the data, but lower- or higher-order models are not justified.

We can actually avoid using the `anova()` function since `poly()` creates orthogonal polynomials:

```
coef(summary(fit.5))
```

```
##                 Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)    111.70361  0.7287647 153.2780243 0.000000e+00
## poly(age, 5)1  447.06785 39.9160847  11.2001930 1.491111e-28
## poly(age, 5)2 -478.31581 39.9160847 -11.9830341 2.367734e-32
## poly(age, 5)3  125.52169 39.9160847   3.1446392 1.679213e-03
## poly(age, 5)4  -77.91118 39.9160847  -1.9518743 5.104623e-02
## poly(age, 5)5  -35.81289 39.9160847  -0.8972045 3.696820e-01
```

As you can see, the p-values aree exactly the same, and in fact the square of the t-statistics are equal to the F-statistics from the `anova()` function. However, the ANOVA method works whether or not we used orthogonal polynomials; it also works when we have other terms in the model as well. For example, we can use `anova()` to compare these three models:

```
fit.1 = lm(wage~education+age, data=Wage)
fit.2 = lm(wage~education + poly(age,2), data=Wage)
fit.3 = lm(wage~education+poly(age,3), data=Wage)
anova(fit.1, fit.2, fit.3)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ education + age
## Model 2: wage ~ education + poly(age, 2)
## Model 3: wage ~ education + poly(age, 3)
##   Res.Df     RSS Df Sum of Sq        F Pr(>F)
## 1   2994 3867992
## 2   2993 3725395  1    142597 114.6969 <2e-16 ***
## 3   2992 3719809  1      5587   4.4936 0.0341 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As an alternative to using ANOVA, we could also choose the polynomial degree using cross-validation.

Now let us consider the task of predicting whether an individual earns more than $250,000 per year. We proceed much as before, except that first we create the appropriate response vector, and then apply the `glm()` function using `family="binomial"` in order to fit a polynomial logistic regression model.

```
fit = glm(I(wage>250) ~ poly(age,4), data = Wage, family = binomial)
```

We are using the wrapper `I()` to create the binary response variable on the fly. The expression `wage>250` evaluates to a logical vairable containing `TRUEs` and `FALSEs`, which

`glm()` coerces to binary by setting `TRUEs` to 1 and `FALSEs` to 0.

We now use the `predict()` function to make our predictions,

```
preds=predict(fit, newdata=list(age=age.grid),se=T)
```

Confidence intervals in this scenario are a bit more involved than they were in the linear regression case. Our predictions are for the *logit*. That is, we have fit a model of the form

$$\log\left(\frac{\Pr(Y=1|X)}{1-\Pr(Y=1|X)}\right)=X\beta$$

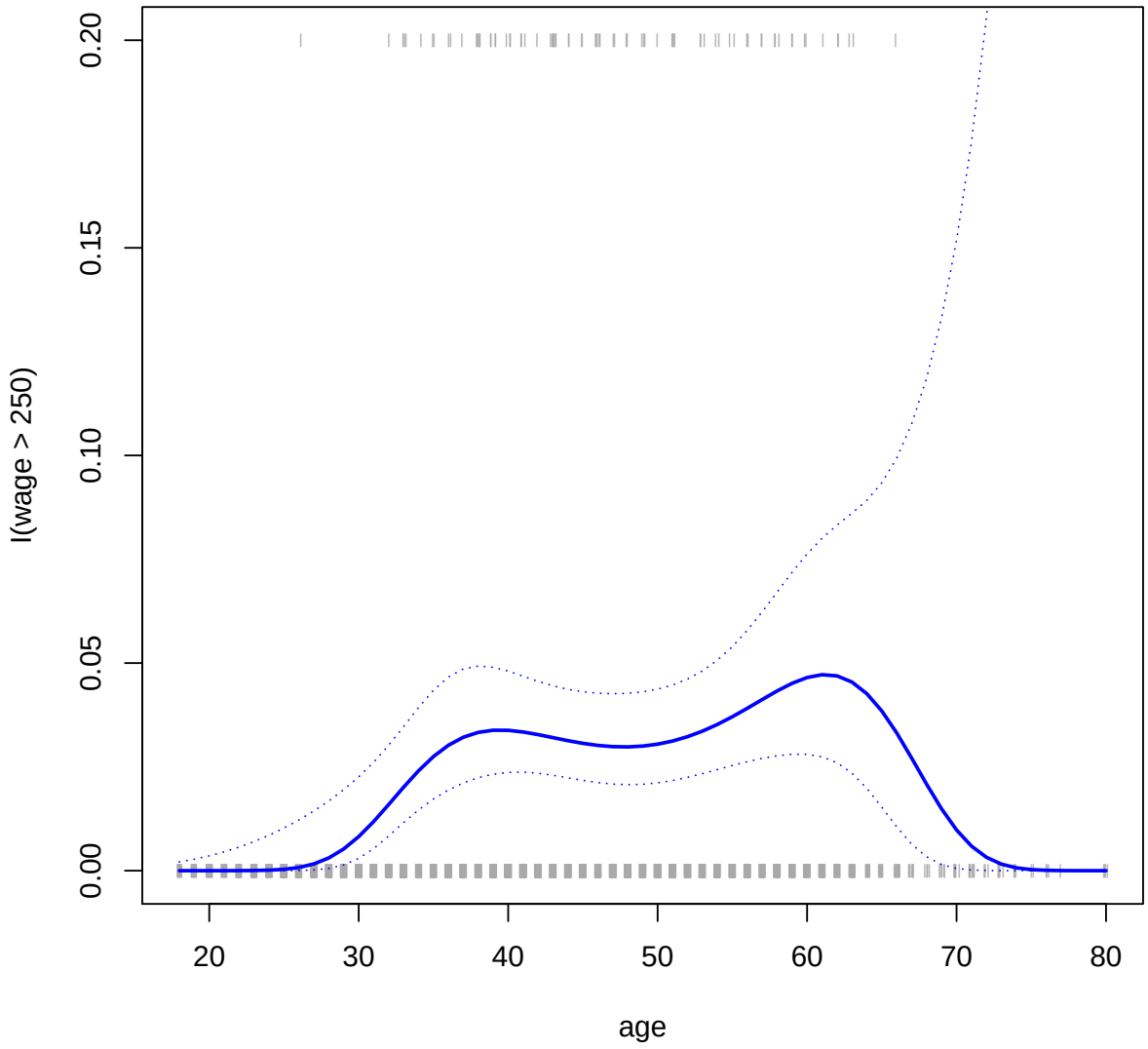
, and the predictions given are of the form $X\hat{\beta}$. The standard errors given are also of this form. In order to obtain confidence intervals for $\Pr(Y=1|X)$, we use the transformation

$$\Pr(Y=1|X)=\frac{\exp(X\beta)}{1+\exp(X\beta)}$$

```
pfit = exp(preds$fit)/(1+exp(preds$fit))
se.bands.logit = cbind(preds$fit+2*preds$se.fit, preds$fit-2*preds$se.fit)
se.bands = exp(se.bands.logit)/(1+exp(se.bands.logit))
```

However, the corresponding confidence intervals would not have been sensible because we would end up with negative probabilities! Finally, the right-hand plot from Figure 7.1 was made as follows:

```
plot(age,I(wage>250), xlim=agelims, type="n", ylim=c(0,0.2))
points(jitter(age), I((wage>250)/5), cex=.5, pch="|", col="darkgrey")
lines(age.grid, pfit, lwd=2, col="blue")
matlines(age.grid, se.bands, lwd=1, col="blue", lty=3)
```



In order to fit a step function, as discussed in Section 7.2, we use the `cut()` function.

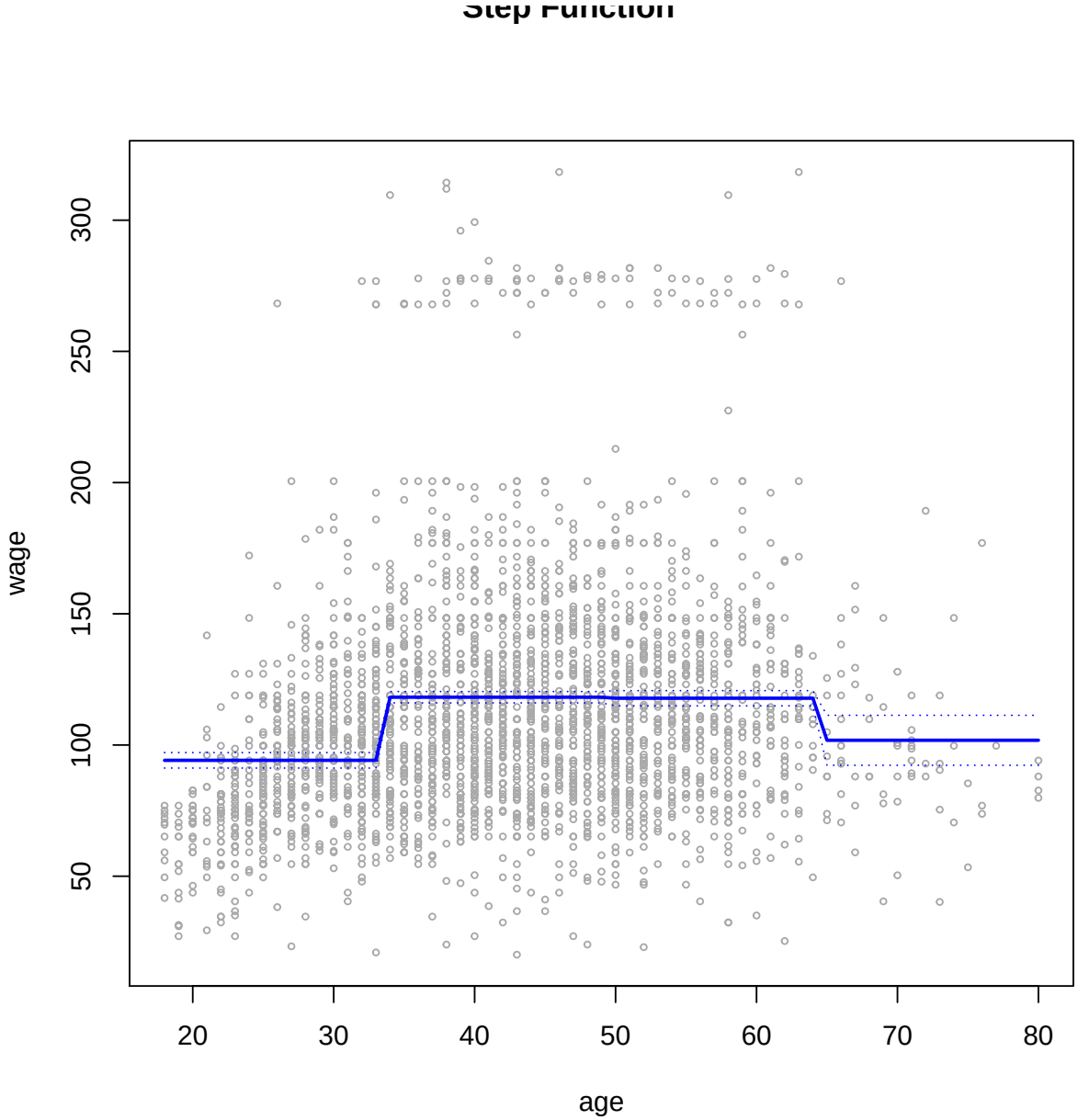
```
table(cut(age,4))

##
## (17.9,33.5]   (33.5,49]   (49,64.5] (64.5,80.1]
##         750        1399         779          72

fit=lm(wage~cut(age,4), data = Wage)
coef(summary(fit))

##                         Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)            94.158392   1.476069 63.789970 0.000000e+00
```

```
## cut(age, 4)(33.5,49]   24.053491   1.829431 13.148074 1.982315e-38
## cut(age, 4)(49,64.5]   23.664559   2.067958 11.443444 1.040750e-29
## cut(age, 4)(64.5,80.1]  7.640592   4.987424  1.531972 1.256350e-01
```

Here `cut()` automatically picked the cutpoints at 33.5, 49, and 64.5 years of age. The function `cut()` returns an ordered categorical variable; the `lm()` function then creates a set of dummy variables for use in the regression. The `age<33.5` category is left out, so the intercept coefficient of \$94,160 can be interpreted as the average salary for those under 33.5 years of age, and the other coefficients can be interpreted as the average additional salary for those in the other age groups.

We can produce predictions and plots just as we did in the case of the polynomial fit:

```
preds=predict(fit, newdata=list(age=age.grid), se=TRUE)
se.bands = cbind(preds$fit+2*preds$se.fit, preds$fit-2*preds$se.fit)

plot(age, wage, xlim=agelims, cex=.5, col="darkgrey")
title("Step Function", outer=T)
lines(age.grid, preds$fit, lwd=2, col="blue")
matlines(age.grid, se.bands, lwd=1, col="blue", lty=3)
```

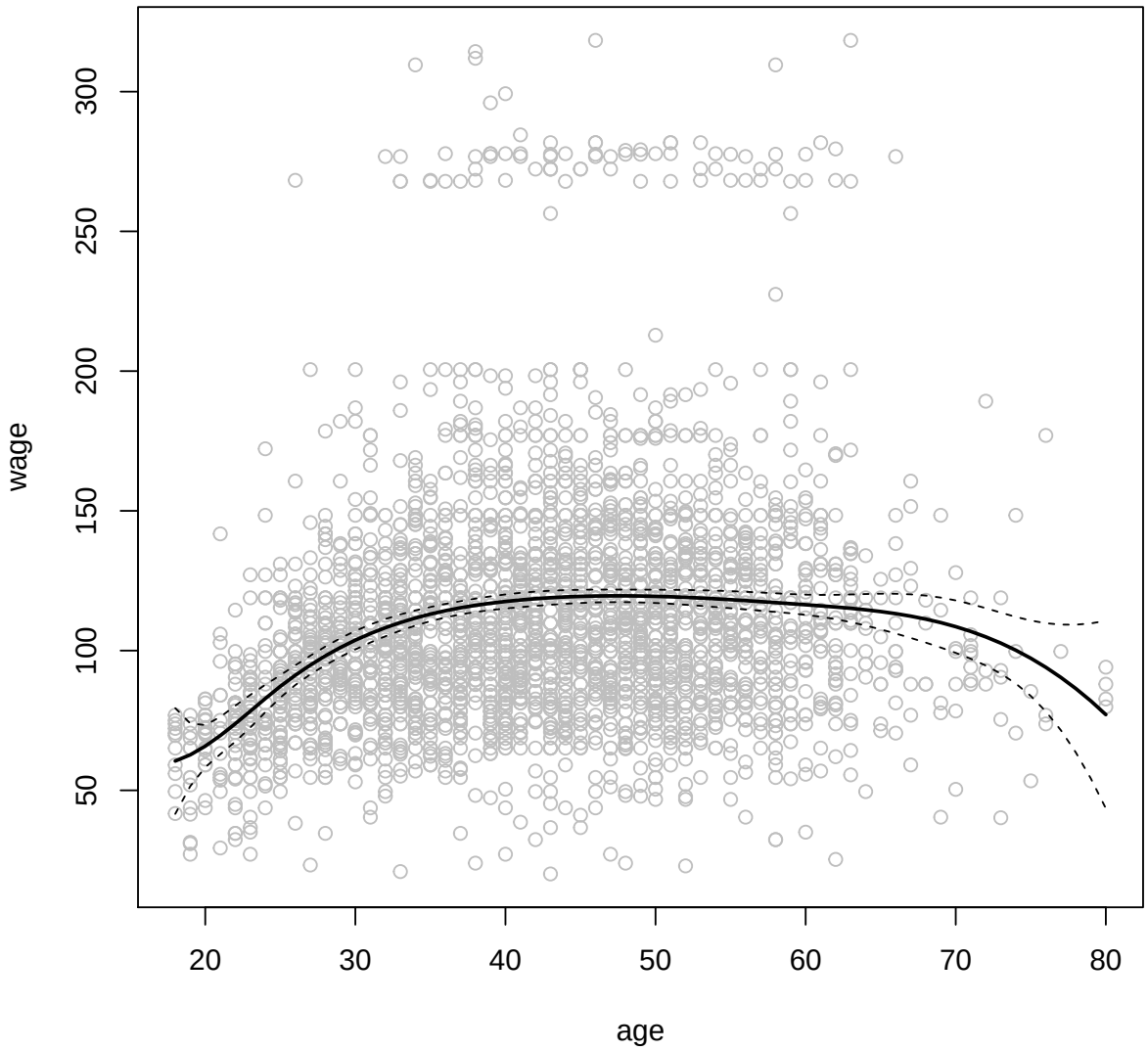

Step Function
wage
age
50
100
150
200
250
300
20
30
40
50
60
70
80

2 Splines

This section covers regression splines, which is a generalization of the step function we computed in the previous section. Instead of fitting constant functions to each cut of the data, we will now fit polynomials. We generally require that these degree n polynomials have continuous $n-1$st derivative at the boundaries of each cut, giving us a very smooth fit to the data.

In order to fit regression splines in `R`, we use the `splines` library. In Section 7.4, we saw that regression splines can be fit by constructing an appropriate matrix of basis functions. The `bs()` function generates the entire matrix of basis functions for splines with the specified set of knots. By default, cubic splines are produced (ie. the polynomials are degree 3). Fitting `wage` to `age` using a regression spline is simple:

```
fit=lm(wage~bs(age, knots=c(25,40,60)),data = Wage)
pred = predict(fit, newdata=list(age=age.grid),se=T)
plot(age, wage, col="gray")
lines(age.grid, pred$fit, lwd=2)
lines(age.grid, pred$fit+2*pred$se, lty="dashed")
lines(age.grid, pred$fit-2*pred$se, lty="dashed")
```



Here we have prespecified knots at ages 25, 40, and 60. This produces a spline with six basis functions. (Recall that a cubic spline with three knots has seven degrees of freedom; these degrees of freedom are used.) We could also use the `df` option to produce a spline with knots at uniform quantiles of the data.

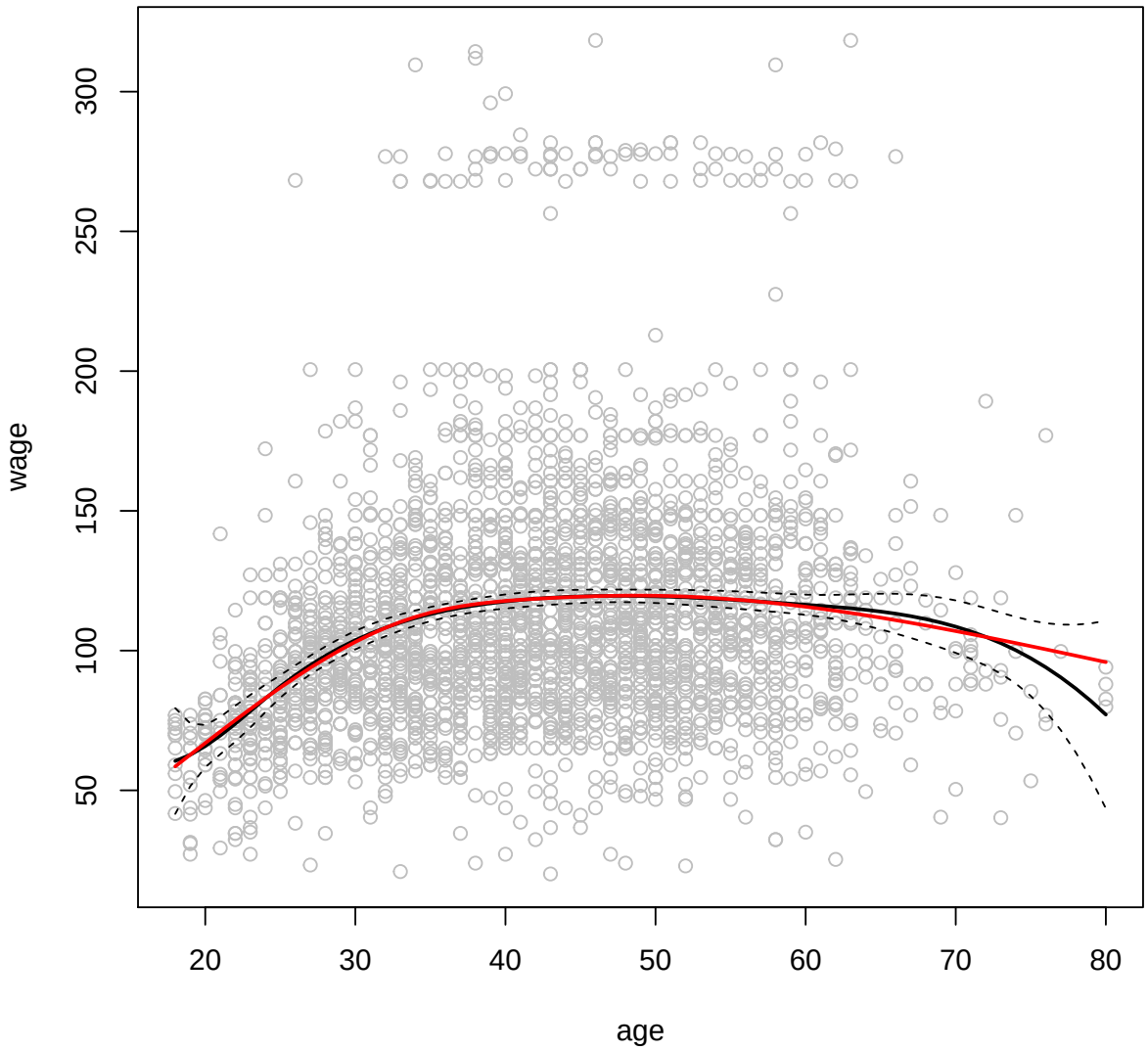
```
dim(bs(age, knots=c(25,40,60)))

## [1] 3000    6

dim(bs(age,df=6))

## [1] 3000    6
```

```
attr(bs(age, df=6),"knots")

##   25%   50%   75%
## 33.75 42.00 51.00
```

In this case R chooses knots at ages 33.8, 42.0, and 51.0, which correspond to the 25th, 50th, and 75th percentiles of `age`. The function `bs()` also has a `degree` argument, so we can fit splines of any degree, rather than the default degree of 3.

In order to instead fit a natural spline, we use the `ns()` function. Recall that natural splines impose the additional condition that the function be linear on the boundaries (that is, in the first cut and last cut of the data).

```
#Plot commands from previous chunk
fit=lm(wage~bs(age, knots=c(25,40,60)),data = Wage)
pred = predict(fit, newdata=list(age=age.grid),se=T)
plot(age, wage, col="gray")
lines(age.grid, pred$fit, lwd=2)
lines(age.grid, pred$fit+2*pred$se, lty="dashed")
lines(age.grid, pred$fit-2*pred$se, lty="dashed")

fit2 = lm(wage~ns(age, df=4), data = Wage)
pred2 = predict(fit2, newdata=list(age=age.grid), se=T)
lines(age.grid, pred2$fit, col="red",lwd=2)
```

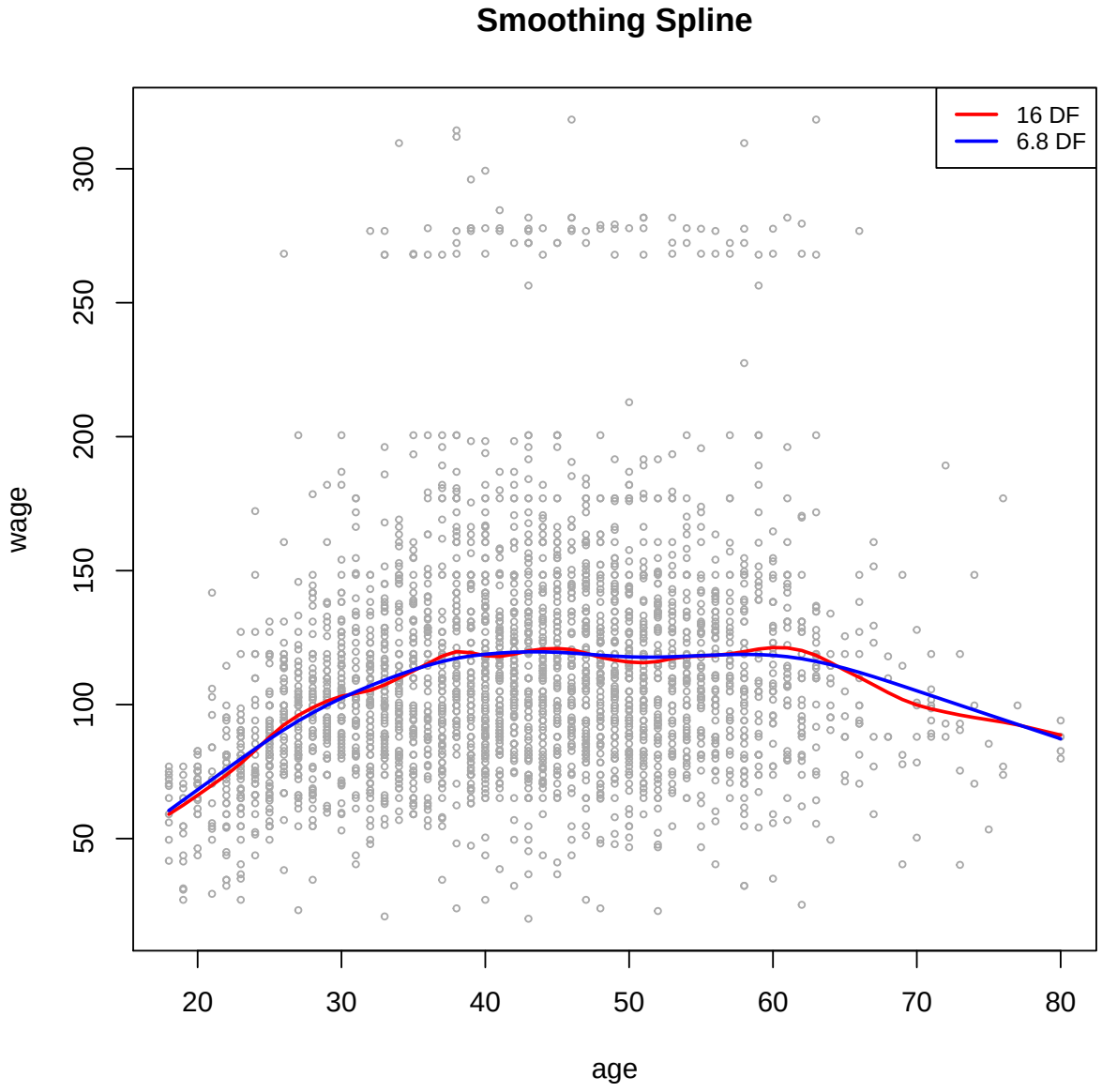


As with the `bs()` function, we could instead specify the knots directly using the `knots` option.

In order to fit a smoothing spline, we use the `smooth.spline()` function. Recall that the smoothing spline function is minimized by a natural cubic spline with knots at the data points $x_1, \ldots, x_n$. Figure 7.8 was produced with the following code:

```
plot(age, wage, xlim=agelims, cex=.5, col="darkgrey")
title("Smoothing Spline")
fit=smooth.spline(age, wage, df=16)
fit2=smooth.spline(age, wage, cv=TRUE)
fit2$df
```

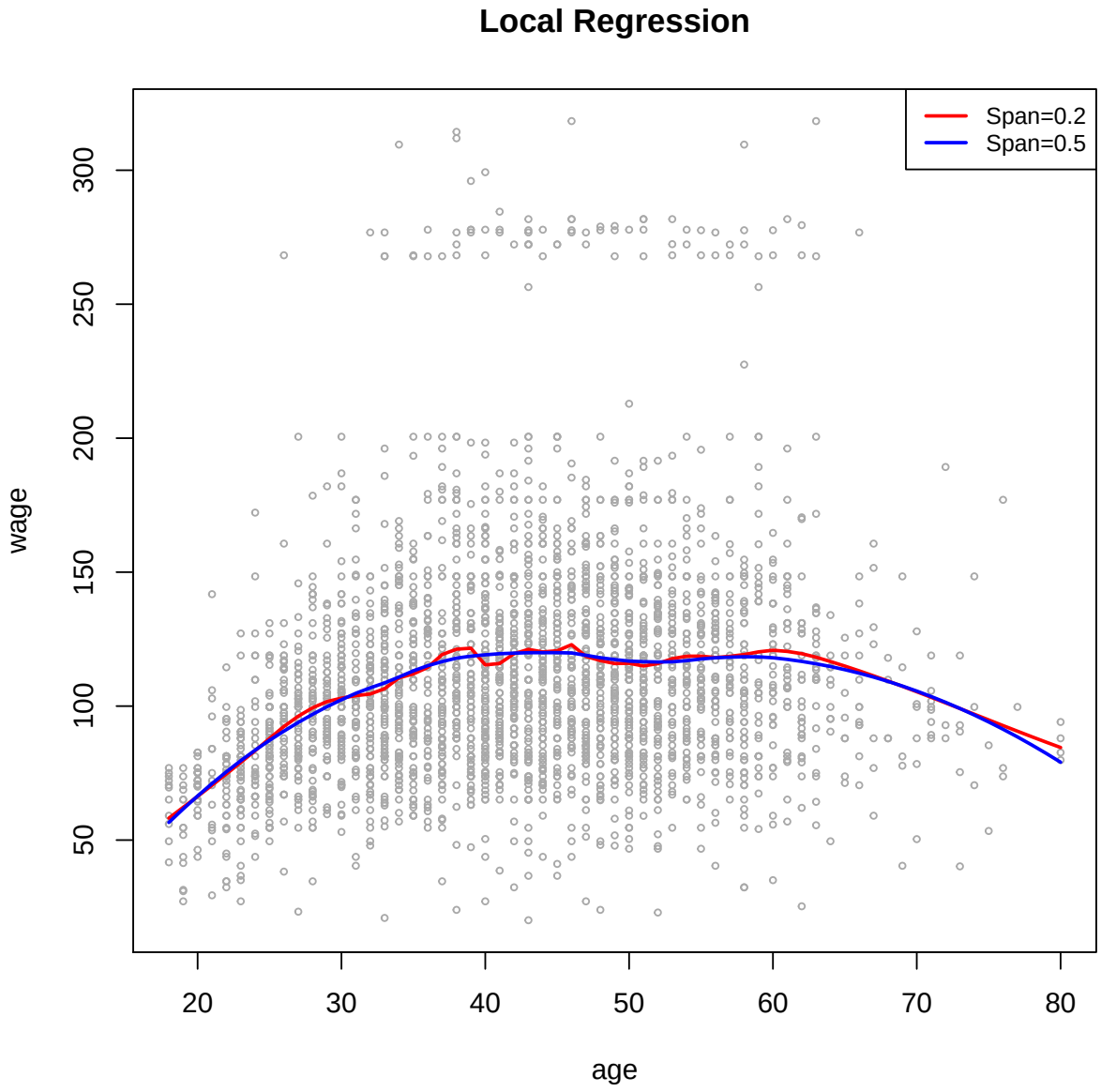
```
## [1] 6.794596

lines(fit, col="red",lwd=2)
lines(fit2,col="blue",lwd=2)
legend("topright",legend=c("16 DF", "6.8 DF"), col=c("red","blue"), lty=1, lwd=2, cex=0.
```



Notice in the first call to `smooth.spline()`, we specified `df=16`. The function then determines which value of λ leads to 16 degrees of freedom. In the second call to `smooth.spline()`, we select the smoothness level by cross-validation; this results in a value of λ that yields 6.8 degrees of freedom.

In order to perform local regression, we use the `loess()` function.

```
plot(age, wage, xlim=agelims, cex=.5, col="darkgrey")
title("Local Regression")
fit=loess(wage~age, span=.2, data=Wage)
fit2=loess(wage~age, span=.5, data=Wage)

lines(age.grid, predict(fit, data.frame(age=age.grid)), col="red", lwd=2)
lines(age.grid, predict(fit2,data.frame(age=age.grid)), col="blue", lwd=2)

legend("topright", legend=c("Span=0.2", "Span=0.5"),
       col=c("red","blue"), lty=1, lwd=2, cex=.8)
```



Here we have performed local linear regression using spans of 0.2 and 0.5. That is, each

neighborhood consists of 20% or 50% of the observations. The larger the span, the smoother the fit. The `locfit` library can also be used for fitting local regression models in `R`.

3 GAMs

We now fit a GAM to predict `wage` using natural spline functions of `year` and `age`, treating `education` as a qualitative predictor as in (7.16). Since this is just a big linear regression model using an appropriate choice of basis functions, we can simply do this using the `lm()` function.

```
gam1 = lm(wage~ns(year,4) + ns(age,5) + education, data = Wage)
```

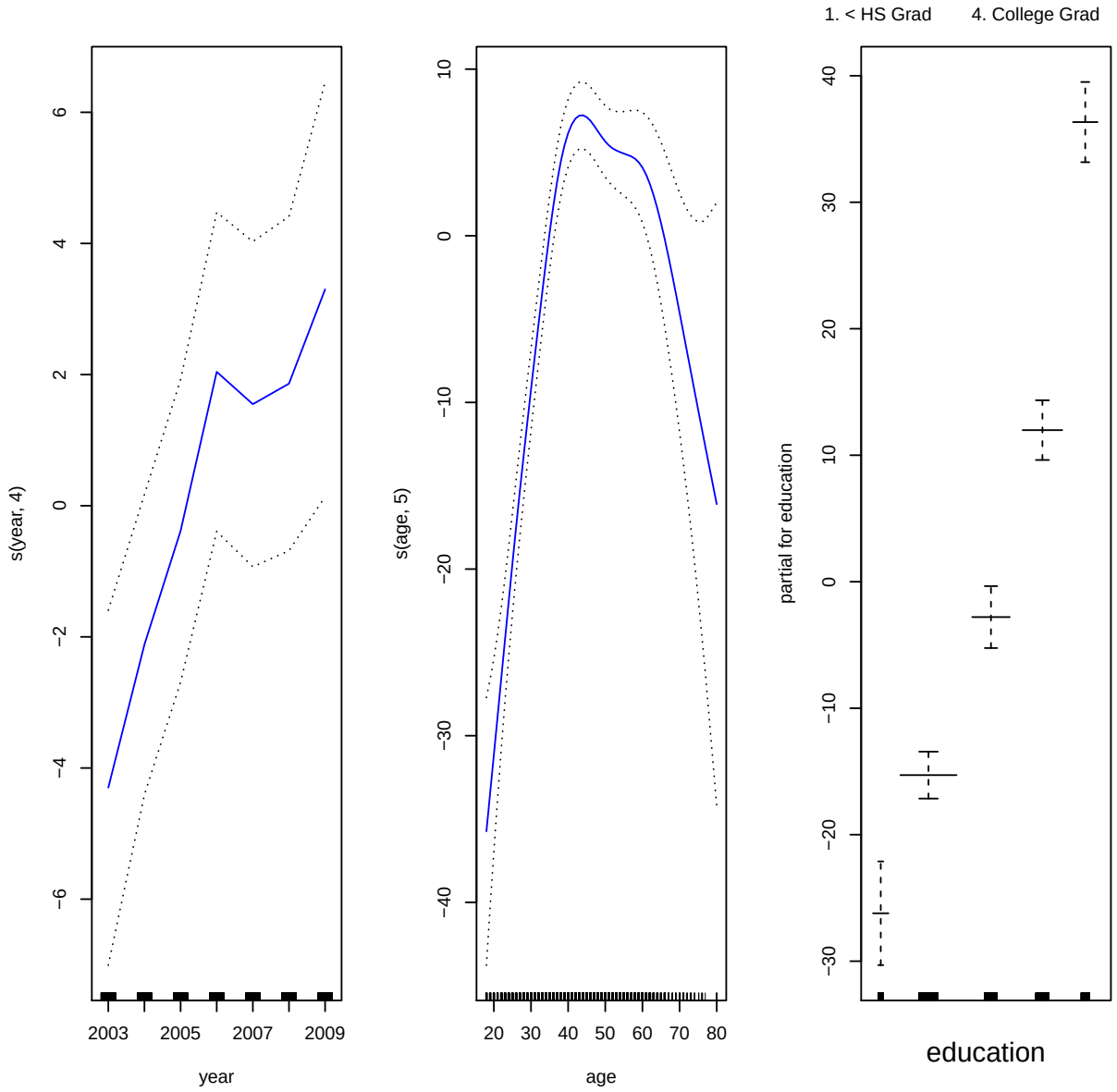
We now fit the model (7.16) using smoothing splines rather than natural splines. In order to fit more general sorts of GAMs, using smoothing splines or other components that cannot be expressed in terms of basis functions and then fit using least squares regression, we will need to use the `gam` library in `R`.

The `s()` function, which is part of the `gam` library, is used to indicate that we would like to use a smoothing spline. We specify that the function of `year` should have 4 degrees of freedom, and that the function of `age` will have 5 degrees of freedom. Since `education` is qualitative, we leave it as is and it is converted into four dummy variables. We use the `gam()` function in order to fit a GAM using these components. All of the terms in (7.16) are fit simultaneously, taking each other into account to explain the response.

```
gam.m3 = gam(wage~s(year,4) + s(age,5) + education, data=Wage)
```

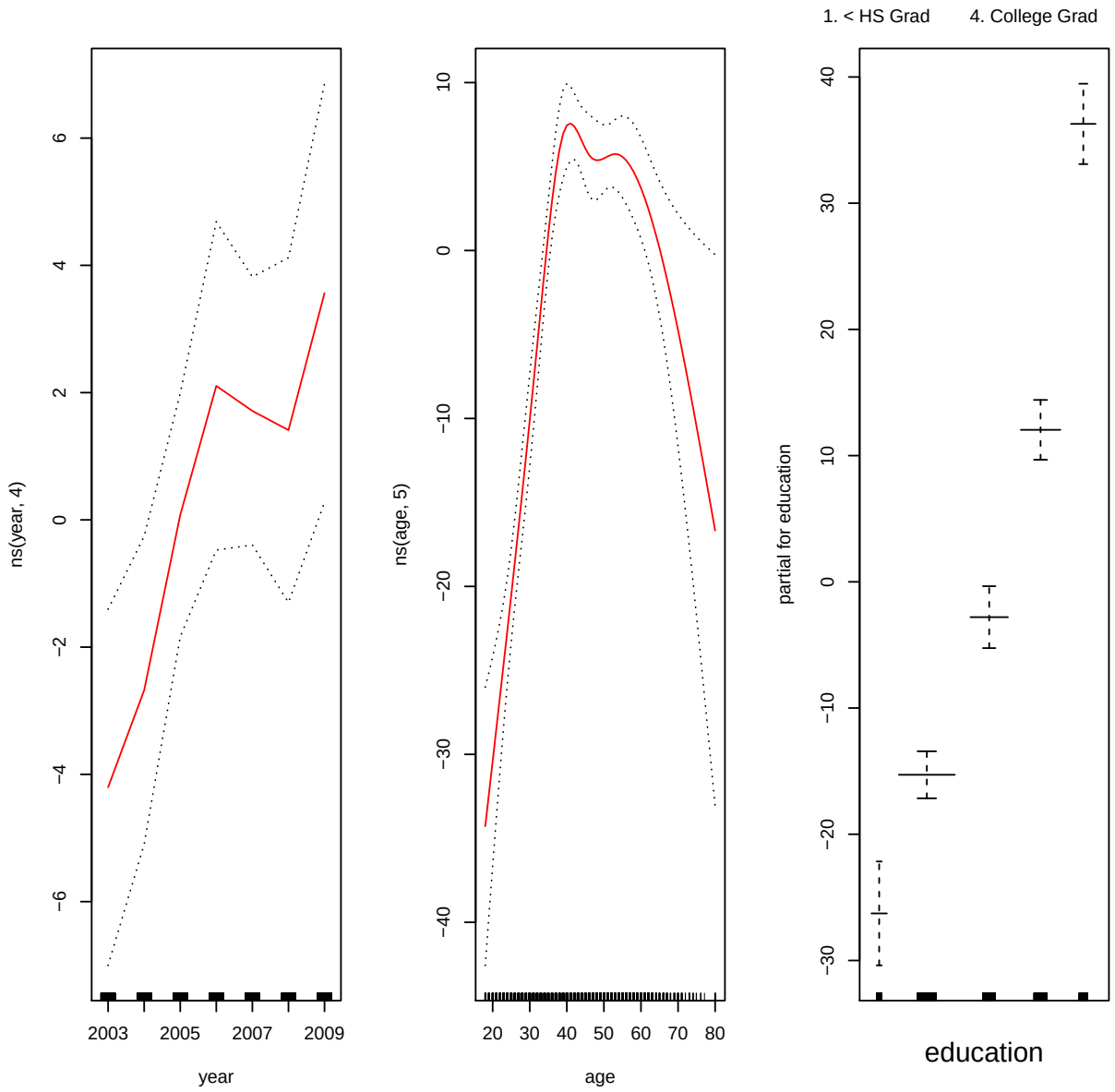
In order to produce Figure 7.12, we simply call the `plot()` function:

```
par(mfrow=c(1,3))
plot(gam.m3, se=TRUE, col="blue")
```



The generic `plot()` function recognizes that `gam.m3` is an object of class `gam`, and invokes the appropriate `plot.Gam()` method. Conveniently, even though `gam1` is not of class `gam` but rather of class `lm`, we can still use `plot.Gam()` on it. Figure 7.11 was produced using the following expression:

```
par(mfrow=c(1,3))

plot.Gam(gam1, se=TRUE, col="red")
```



Notice here we had to use `plot.Gam()` rather than the generic `plot()` function.

In these plots, the function of `year` looks rather linear. We can perform a series of ANOVA tests in order to determine which of these three models is best: a GAM that excludes `year` (M_1), a GAM that uses a linear function of `year` (M_2), or a GAM that uses a spline function of year (M_3).

```
gam.m1=gam(wage~s(age,5)+education, data=Wage)
gam.m2=gam(wage~year+s(age,5)+education, data=Wage)
anova(gam.m1, gam.m2, gam.m3, test="F")

## Analysis of Deviance Table
```

```
##
## Model 1: wage ~ s(age, 5) + education
## Model 2: wage ~ year + s(age, 5) + education
## Model 3: wage ~ s(year, 4) + s(age, 5) + education
##   Resid. Df Resid. Dev Df Deviance       F    Pr(>F)
## 1      2990    3711731
## 2      2989    3693842  1  17889.2 14.4771 0.0001447 ***
## 3      2986    3689770  3   4071.1  1.0982 0.3485661
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We find that there is compelling evidence that a GAM with a linear function of `year` is better than a GAM that does not include `year` at all. However, there is no evidence that a non-linear function of `year` is needed. In other words, based on the results of this ANOVA, M_2 is preferred.

The `summary()` function produces a summary of the gam fit.

```
summary(gam.m3)
```

```
##
## Call: gam(formula = wage ~ s(year, 4) + s(age, 5) + education, data = Wage)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -119.43  -19.70   -3.33   14.17  213.48
##
## (Dispersion Parameter for gaussian family taken to be 1235.69)
##
##     Null Deviance: 5222086 on 2999 degrees of freedom
## Residual Deviance: 3689770 on 2986 degrees of freedom
## AIC: 29887.75
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##              Df  Sum Sq Mean Sq F value    Pr(>F)
## s(year, 4)    1   27162   27162  21.981 2.877e-06 ***
## s(age, 5)     1  195338  195338 158.081 < 2.2e-16 ***
## education     4 1069726  267432 216.423 < 2.2e-16 ***
## Residuals  2986 3689770    1236
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df Npar F  Pr(F)
```

```
## (Intercept)
## s(year, 4)         3  1.086 0.3537
## s(age, 5)          4 32.380 <2e-16 ***
## education
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

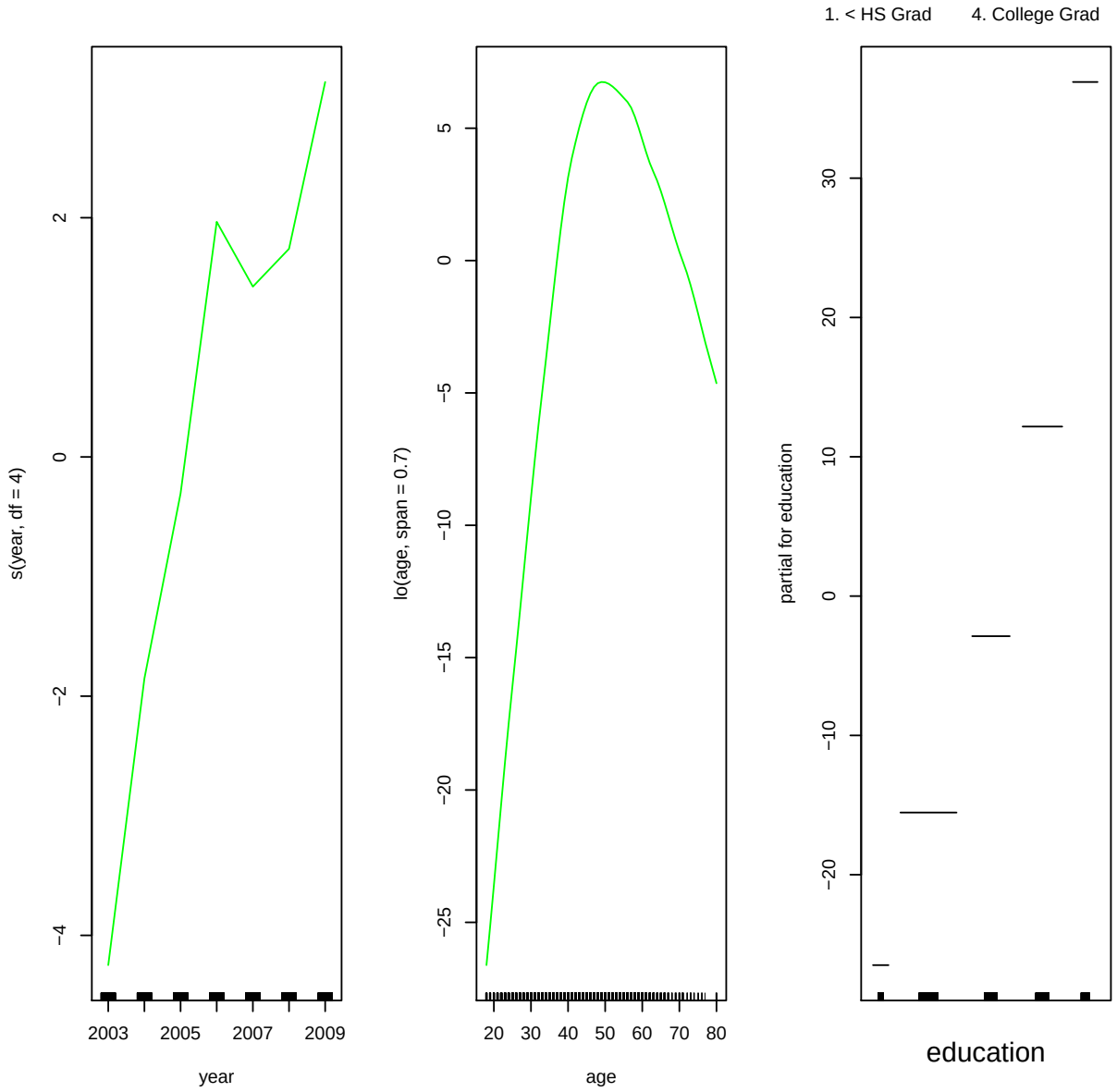
The p-values for `year` and `age` correspond to a null hypothesis of a linear relationship versus the alternative of a non-linear relationship. The large p-value for `year` reinforces our conclusion from the ANOVA test that a linear function is adequate for this term. However, there is very clear evidence that a non-linear term is required for `age`.

We can make predictions from `gam` objects, just like from `lm` objects, using the `predict()` method for the class `gam`. Here we make predictions on the training set.

```
preds = predict(gam.m2, newdata=Wage)
```

We can also use local regression fits as building blocks in a GAM, using the `lo()` function.

```
par(mfrow=c(1,3))

gam.lo = gam(wage~s(year, df =4)+ lo(age, span=0.7) + education,
             data=Wage)
plot.Gam(gam.lo, set=TRUE, col="green")
```

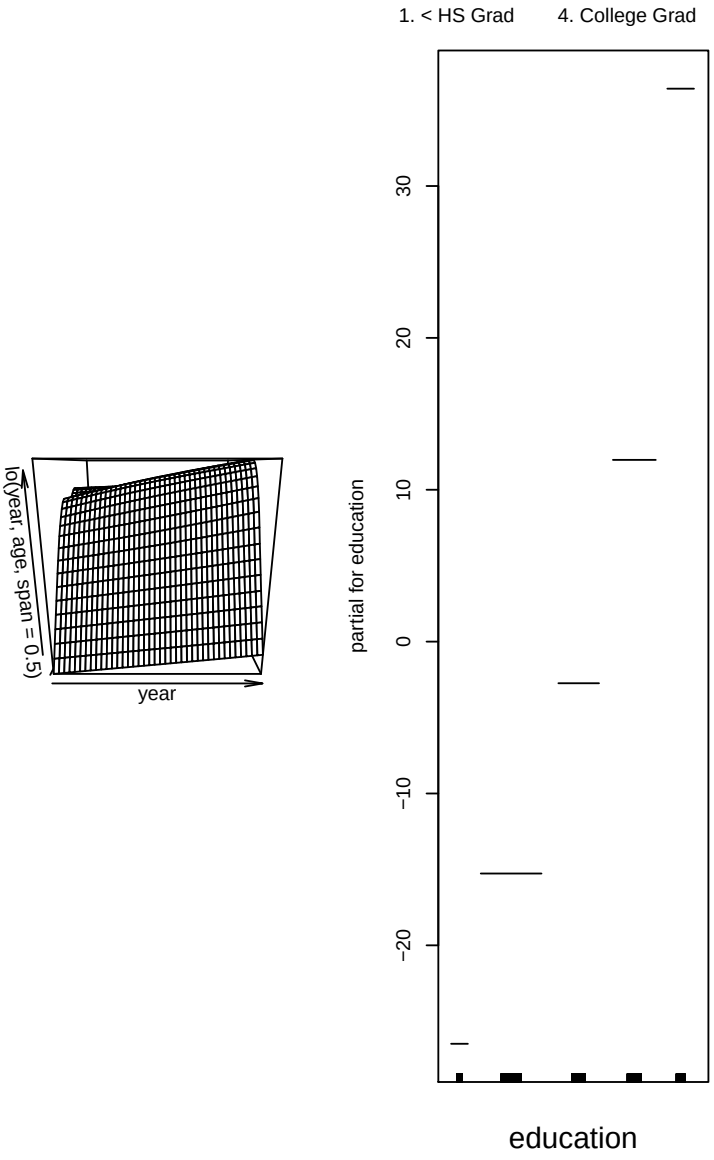


Here we have used local regression for the `age` term, with a span of 0.7. We can also use the `lo()` function to create interactions before calling the `gam()` function. For example,

```
gam.lo.i = gam(wage~lo(year, age, span=0.5)+education, data=Wage)
```

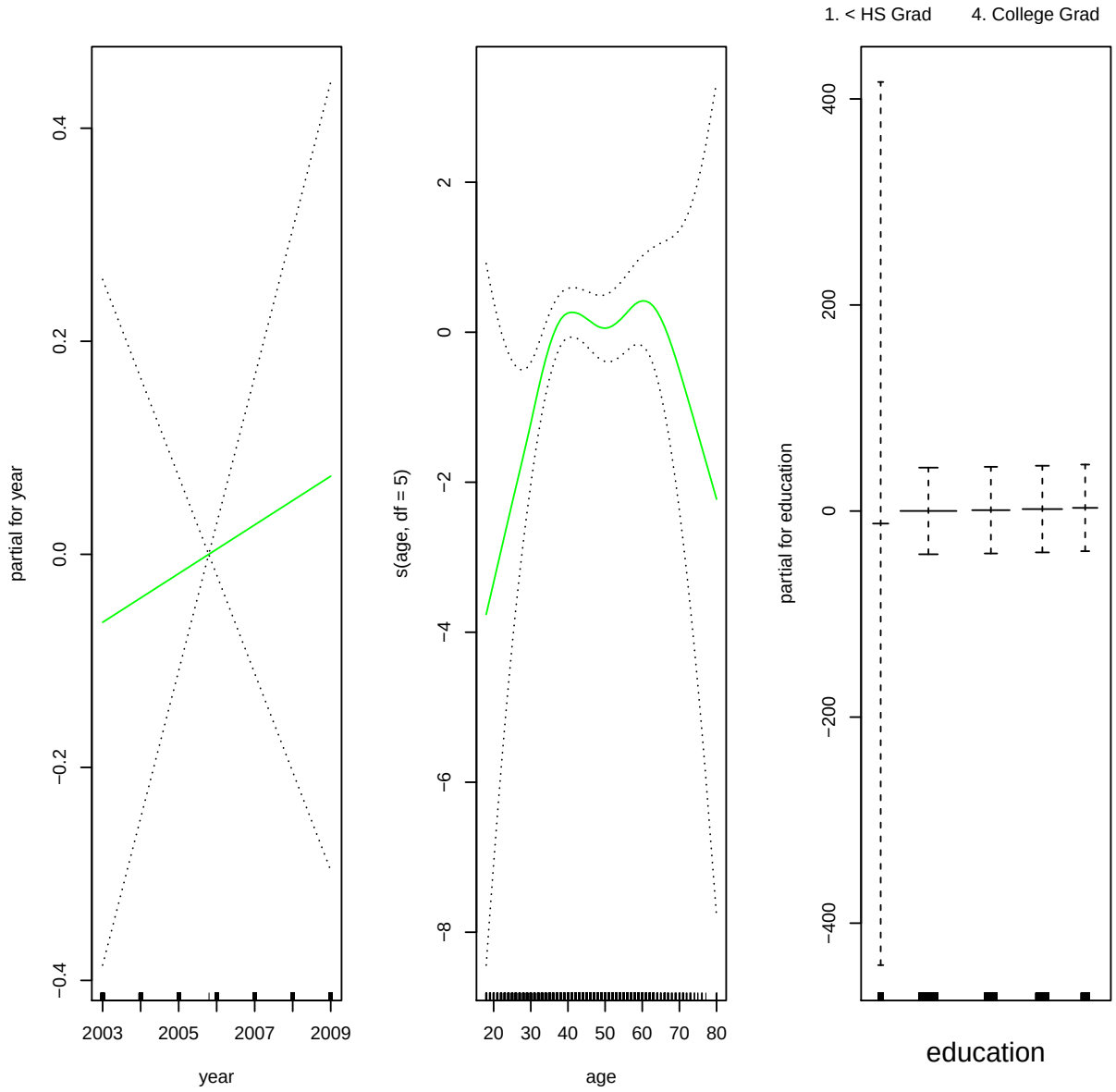
fits a two-term model, in which the first term is an interaction between `year` and `age`, fit by a local regression surface. We can plot the resulting two-dimensional surface if we first install the `akima` package.

```
par(mfrow=c(1,3))

plot(gam.lo.i)
```



In order to fit a logistic regression GAM, we once again use the `I()` function in constructing the binary response variable, and set `family = binomial`.

```
par(mfrow=c(1,3))

gam.lr=gam(I(wage>250)~year+s(age,df=5)+education,
           family = binomial, data = Wage)
par(mfrow=c(1,3))
plot(gam.lr, se=T, col="green")
```



It is easy to see that there are no high earners in the <HS category:

```
table(education, I(wage>250))

##
## education            FALSE TRUE
##   1. < HS Grad         268    0
##   2. HS Grad           966    5
##   3. Some College      643    7
##   4. College Grad      663   22
##   5. Advanced Degree   381   45
```

Hence, we git a logistic regression GAM using all but this category. This provides more sensible results.

```
par(mfrow=c(1,3))

gam.lr.s = gam(I(wage>250)~year+s(age, df=5)+education,
               family=binomial, data=Wage,
               subset=(education!="1. < HS Grad"))
plot(gam.lr.s, se=T, col="green")
```