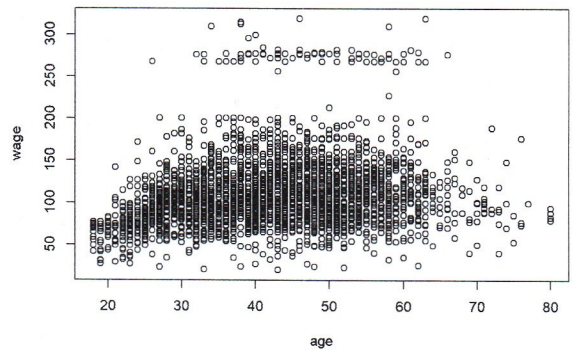
```
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### NONPARAMETRIC REGRESSION TUTORIAL
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
# Let's familiarise with the dataset we will be using.
# It's something that you should remember quite well.
load('nlr_data.rda')
# Wage is the Wage dataset by ISL.
# Prestige is the occupational prestige in Canada by Blishen and McRoberts.
# The idea is to carry on the two examples, and see how different methods perform.
# We start with Wage, and inspect the relationship between age and wage.
attach(Wage)

plot(age,wage)
```

(polynomial regression) — global approach



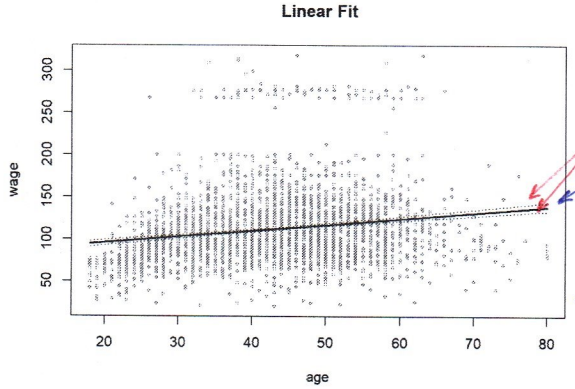

```
# Lets inspect the relationship between wage and age.
# We have a "weird" cluster of people with a VERY high wage (C-Level?),
# and we observe some kind of reverse bathtub behaviour.
# What happens if I try to model it linearly?
m_linear = lm(wage ~ age)
summary(m_linear)
```

lavori tipo capi/managers ..

```
##
## Call:
## lm(formula = wage ~ age)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -100.265  -25.115   -6.063   16.601  205.748
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 81.70474    2.84624   28.71   <2e-16 ***
## age          0.70728    0.06475   10.92   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 40.93 on 2998 degrees of freedom
## Multiple R-squared:  0.03827,    Adjusted R-squared:  0.03795
## F-statistic: 119.3 on 1 and 2998 DF,  p-value: < 2.2e-16
```

```
age.grid = seq(range(age)[1],range(age)[2],by=0.5)
preds    = predict(m_linear,list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey ", main='Linear Fit')
lines(age.grid,preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```

pointwise bounds for the prediction (to have a general idea of the uncertainty of the predictions)



uncertainty

regression function

← We have high bias and very low variance (we've over-simplifying)

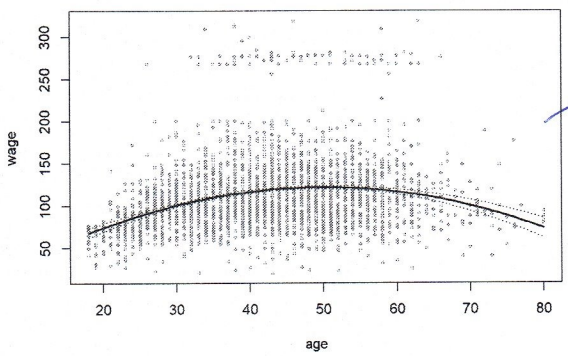
```
# Not very good.

# Let's try to add terms...
m_quad = lm(wage ~ age + I(age^2))
summary(m_quad)
```

The model is still linear because the β's are linearly multiplying the age and age^2 but we're adding some curvature

```
##
## Call:
## lm(formula = wage ~ age + I(age^2))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -99.126 -24.309  -5.017  15.494 205.621
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -10.425224   8.189780  -1.273    0.203
## age           5.294030   0.388689  13.620   <2e-16 ***
## I(age^2)     -0.053005   0.004432 -11.960   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.99 on 2997 degrees of freedom
## Multiple R-squared:  0.08209,    Adjusted R-squared:  0.08147
## F-statistic:   134 on 2 and 2997 DF,  p-value: < 2.2e-16
```

```
preds    = predict(m_quad,list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
print(plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey "), main='Quadratic Fit')
```

```
lines(age.grid, preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```



This is nicer!
However, do we have to do it by hand? Realizing that we need an other term, then one more and so on? No. (↓)

function: "poly"

```
# There's actually a more efficient way to do so
m_list = lapply(1:10,function(degree){lm(wage ~ poly(age,degree=degree))})
anova(m_list[[1]], m_list[[2]])
```

We're generating a list: we're applying the function on each term 1:10
(we're generating a list of 10 models where each one of them is fitted on a polynomial expansion of a different degree)

```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, degree = degree)
## Model 2: wage ~ poly(age, degree = degree)
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1   2998 5022216
## 2   2997 4793430  1    228786 143.04 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

in this way we're testing:
H_0: adding a term of higher degree makes no changes
H_1: the two models are different (degree 1 and degree 2 in this case)

```
do.call(anova,m_list)
```

→ "automatization" of (↑)

Note: if we call it like that we get an expansion in orthogonal polynomials. This is very effective from a model selection point of view.

```
## Analysis of Variance Table
##
## Model  1: wage ~ poly(age, degree = degree)
## Model  2: wage ~ poly(age, degree = degree)
## Model  3: wage ~ poly(age, degree = degree)
## Model  4: wage ~ poly(age, degree = degree)
## Model  5: wage ~ poly(age, degree = degree)
## Model  6: wage ~ poly(age, degree = degree)
## Model  7: wage ~ poly(age, degree = degree)
## Model  8: wage ~ poly(age, degree = degree)
## Model  9: wage ~ poly(age, degree = degree)
## Model 10: wage ~ poly(age, degree = degree)
##    Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1    2998 5022216
## 2    2997 4793430  1    228786 143.7638 < 2.2e-16 ***
## 3    2996 4777674  1     15756   9.9005  0.001669 **
## 4    2995 4771604  1      6070   3.8143  0.050909 .
## 5    2994 4770322  1      1283   0.8059  0.369398
## 6    2993 4766389  1      3932   2.4709  0.116074
## 7    2992 4763834  1      2555   1.6057  0.205199
## 8    2991 4763707  1       127   0.0796  0.777865
## 9    2990 4756703  1      7004   4.4014  0.035994 *
## 10   2989 4756701  1         3   0.0017  0.967529
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

a ogni passo questo è il p-value per:
H_0: non ha senso aggiungere un degree
H_1: ha senso aggiungere un degree
qui in particolare (H_1: ha senso avere degree = 2

the optimal model is a 3 or 4 degree polynomial (we'll check both of them)

!

This can be considered more as a noise, it's better to stop after degree = 5 (for example) to avoid this kind of problems

```
# I get the very same results with non-orthogonal poly
m_list_raw = lapply(1:10,function(degree){lm(wage ~ poly(age,degree=degree,raw=T))})
do.call(anova,m_list_raw)
```

(because we're doing a global test) but *

```
## Analysis of Variance Table
##
## Model  1: wage ~ poly(age, degree = degree, raw = T)
## Model  2: wage ~ poly(age, degree = degree, raw = T)
## Model  3: wage ~ poly(age, degree = degree, raw = T)
## Model  4: wage ~ poly(age, degree = degree, raw = T)
## Model  5: wage ~ poly(age, degree = degree, raw = T)
## Model  6: wage ~ poly(age, degree = degree, raw = T)
## Model  7: wage ~ poly(age, degree = degree, raw = T)
## Model  8: wage ~ poly(age, degree = degree, raw = T)
## Model  9: wage ~ poly(age, degree = degree, raw = T)
## Model 10: wage ~ poly(age, degree = degree, raw = T)
##    Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1    2998 5022216
## 2    2997 4793430  1    228786 143.7638 < 2.2e-16 ***
## 3    2996 4777674  1     15756   9.9005  0.001669 **
## 4    2995 4771604  1      6070   3.8143  0.050909 .
## 5    2994 4770322  1      1283   0.8059  0.369398
## 6    2993 4766389  1      3932   2.4709  0.116074
## 7    2992 4763834  1      2555   1.6057  0.205199
## 8    2991 4763707  1       127   0.0796  0.777865
## 9    2990 4756703  1      7004   4.4014  0.035994 *
## 10   2989 4756701  1         3   0.0017  0.967529
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

*

```
# BUT, the situation gets badly messed up if I try classic, t-test based model selection:
summary(m_list[[5]])
```

```
##
## Call:
## lm(formula = wage ~ poly(age, degree = degree))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -99.049 -24.386  -5.028  15.344 202.886
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  111.7036     0.7288 153.278  < 2e-16 ***
## poly(age, degree = degree)1  447.0679    39.9161  11.200  < 2e-16 ***
## poly(age, degree = degree)2 -478.3158    39.9161 -11.983  < 2e-16 ***
## poly(age, degree = degree)3  125.5217    39.9161   3.145  0.00168 **
## poly(age, degree = degree)4  -77.9112    39.9161  -1.952  0.05105 .
## poly(age, degree = degree)5  -35.8129    39.9161  -0.897  0.36968
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.92 on 2994 degrees of freedom
## Multiple R-squared:  0.08651,    Adjusted R-squared:  0.08498
## F-statistic: 56.71 on 5 and 2994 DF,  p-value: < 2.2e-16
```

These remain the same ✓
(because the elements of the polynomial expansion are ⊥, so the betas are ⊥ so the tests are ⊥ (check the other two too))
(and so the p-values)

```
summary(m_list[[4]])
```

```
##
## Call:
## lm(formula = wage ~ poly(age, degree = degree))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -98.707 -24.626  -4.993  15.217 203.693
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  111.7036     0.7287 153.283  < 2e-16 ***
## poly(age, degree = degree)1  447.0679    39.9148  11.201  < 2e-16 ***
## poly(age, degree = degree)2 -478.3158    39.9148 -11.983  < 2e-16 ***
## poly(age, degree = degree)3  125.5217    39.9148   3.145  0.00168 **
## poly(age, degree = degree)4  -77.9112    39.9148  -1.952  0.05104 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.91 on 2995 degrees of freedom
## Multiple R-squared:  0.08626,    Adjusted R-squared:  0.08504
## F-statistic: 70.69 on 4 and 2995 DF,  p-value: < 2.2e-16
```

These remain the same ✓

```
summary(m_list[[3]])
```

```
##
## Call:
## lm(formula = wage ~ poly(age, degree = degree))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -99.693 -24.562  -5.222  15.096 206.119
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  111.7036     0.7291 153.211  < 2e-16 ***
## poly(age, degree = degree)1  447.0679    39.9335  11.195  < 2e-16 ***
## poly(age, degree = degree)2 -478.3158    39.9335 -11.978  < 2e-16 ***
## poly(age, degree = degree)3  125.5217    39.9335   3.143  0.00169 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.93 on 2996 degrees of freedom
## Multiple R-squared:  0.0851, Adjusted R-squared:  0.08419
## F-statistic: 92.89 on 3 and 2996 DF,  p-value: < 2.2e-16
```

These remain the same ✓

```
# vs
summary(m_list_raw[[5]])
```

```
##
## Call:
## lm(formula = wage ~ poly(age, degree = degree, raw = T))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -99.049 -24.386  -5.028  15.344 202.886
##
## Coefficients:
##                                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)                         -4.970e+01  1.614e+02  -0.308    0.758
## poly(age, degree = degree, raw = T)1  3.993e+00  2.011e+01   0.199    0.843
## poly(age, degree = degree, raw = T)2  2.760e-01  9.585e-01   0.288    0.773
## poly(age, degree = degree, raw = T)3 -1.265e-02  2.191e-02  -0.577    0.564
## poly(age, degree = degree, raw = T)4  1.835e-04  2.408e-04   0.762    0.446
## poly(age, degree = degree, raw = T)5 -9.157e-07  1.021e-06  -0.897    0.370
##
## Residual standard error: 39.92 on 2994 degrees of freedom
## Multiple R-squared:  0.08651,    Adjusted R-squared:  0.08498
## F-statistic: 56.71 on 5 and 2994 DF,  p-value: < 2.2e-16
```

These change (based on how many models we're testing (check also the other two)) ✗

```
summary(m_list_raw[[4]])
```

because of this, it's hard to make some decisions on which model is the best (better to leave **raw = F**)

```
##
## Call:
## lm(formula = wage ~ poly(age, degree = degree, raw = T))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -98.707 -24.626  -4.993  15.217 203.693
##
## Coefficients:
##                                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)                         -1.842e+02  6.004e+01  -3.067 0.002180 **
## poly(age, degree = degree, raw = T)1  2.125e+01  5.887e+00   3.609 0.000312 ***
## poly(age, degree = degree, raw = T)2 -5.639e-01  2.061e-01  -2.736 0.006261 **
## poly(age, degree = degree, raw = T)3  6.811e-03  3.066e-03   2.221 0.026398 *
## poly(age, degree = degree, raw = T)4 -3.204e-05  1.641e-05  -1.952 0.051039 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.91 on 2995 degrees of freedom
## Multiple R-squared:  0.08626,    Adjusted R-squared:  0.08504
## F-statistic: 70.69 on 4 and 2995 DF,  p-value: < 2.2e-16
```

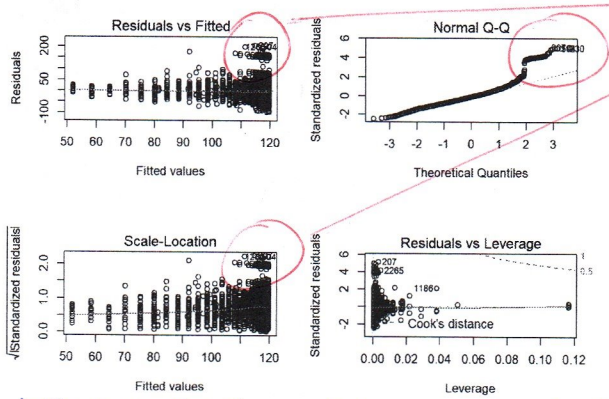
These change ✗

```
summary(m_list_raw[[3]])
```

```
##
## Call:
## lm(formula = wage ~ poly(age, degree = degree, raw = T))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -99.693 -24.562  -5.222  15.096 206.119
##
## Coefficients:
##                                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)                         -7.524e+01  2.218e+01  -3.392 0.000703 ***
## poly(age, degree = degree, raw = T)1  1.019e+01  1.605e+00   6.348 2.51e-10 ***
## poly(age, degree = degree, raw = T)2 -1.680e-01  3.686e-02  -4.559 5.36e-06 ***
## poly(age, degree = degree, raw = T)3  8.495e-04  2.702e-04   3.143 0.001687 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.93 on 2996 degrees of freedom
## Multiple R-squared:  0.0851, Adjusted R-squared:  0.08419
## F-statistic: 92.89 on 3 and 2996 DF,  p-value: < 2.2e-16
```

These change ✗

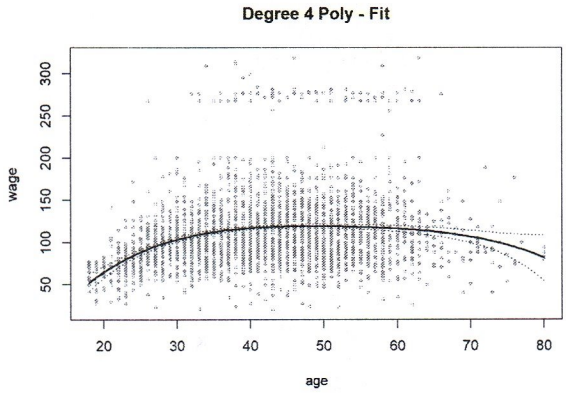
```
# Look at the p-values

# Bottomline, the correct model seems to be the degree 4 one.
par(mfrow=c(2,2))
plot(m_list[[4]])
```



These are probably because of the group of outlier (the people that are working as boss / managers.. (the C-level of a company)

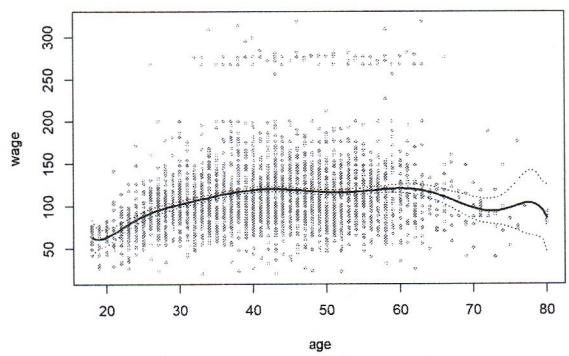
Let's check the shape of the regression function:

```
preds    = predict(m_list[[4]],list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
par(mfrow=c(1,1))
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey ", main='Degree 4 Poly - Fit')
lines(age.grid, preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```



Better job than the degree 2 that we tried by hand

```
# What happens, instead, if I try to overfit?
preds    = predict(m_list[[10]],list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey ", main='Degree 10 Poly - Fit')
lines(age.grid, preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```



The function becomes more wiggly and the variance of the predictions increases (less biased but with high variance)

```
m_list_logit = lapply(1:5,function(degree){
  glm(I(wage>250) ~poly(age,degree=degree),family='binomial')})
```

← we try to fit a logistic regression with different degree of polynomials (we're modelling a dummy variable with a linear model: dummy_var = 1{wage>250})

```
do.call(anova,m_list_logit)
```

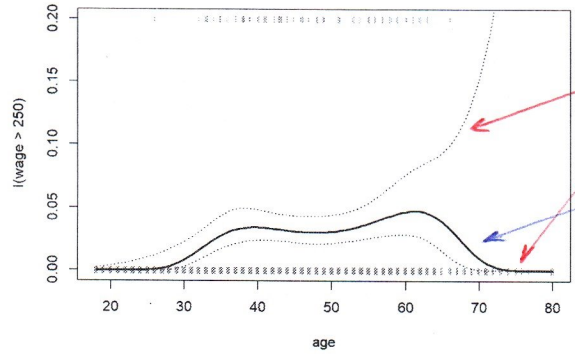
So we've modelling the probability of a person to get more than 250: we've modelling the logit actually, to get the probability we need to ri-convert it.

```
## Analysis of Deviance Table
##
## Model 1: I(wage > 250) ~ poly(age, degree = degree)
## Model 2: I(wage > 250) ~ poly(age, degree = degree)
## Model 3: I(wage > 250) ~ poly(age, degree = degree)
## Model 4: I(wage > 250) ~ poly(age, degree = degree)
## Model 5: I(wage > 250) ~ poly(age, degree = degree)
##   Resid. Df Resid. Dev Df Deviance
## 1      2998     719.23
## 2      2997     709.02  1  10.2048
## 3      2996     707.92  1   1.1022
## 4      2995     701.22  1   6.7017
## 5      2994     698.87  1   2.3499
```

Let's try to stick with degree = 4

```
# We can probably be content, here, with a smaller model than before

preds          = predict(m_list_logit[[4]],list(age=age.grid),se=T)
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))

plot(age, I(wage >250), xlim=range(age.grid), type ="n", ylim=c(0,.2))
points (jitter(age), I((wage >250)/5), cex =.5, pch ="|", col =" darkgrey ",
        main='Poly 4 Fit - Logistic')
lines(age.grid, pfit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```

conversion in probabilities

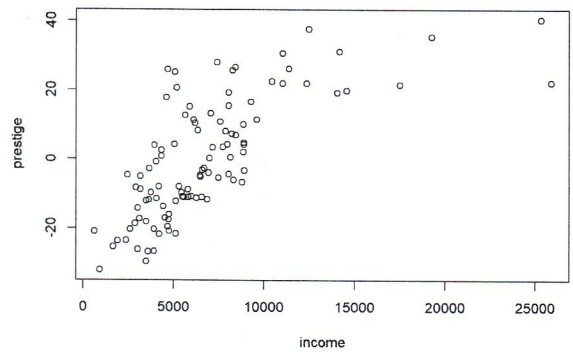


uncertainty around the value

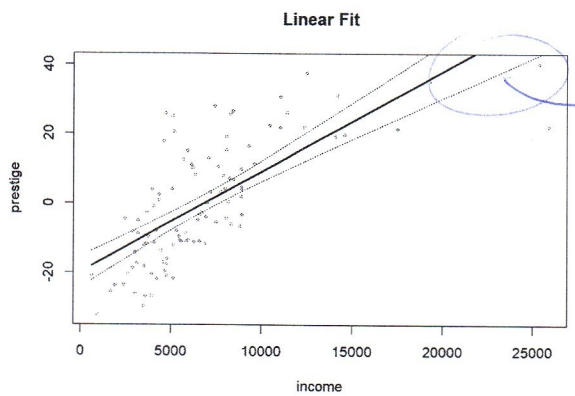
probability

```
### ---------------------------------------------------------------------------
### Let's see what happens with our other dataset!
### ---------------------------------------------------------------------------
detach(Wage)
attach(Prestige)
```

```
plot(income,prestige)
```



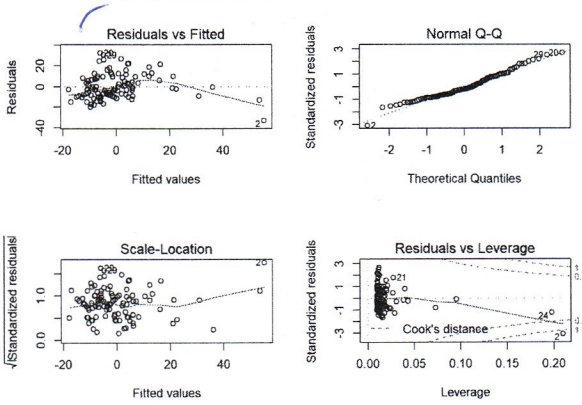

```
# Evidently nonlinear relationship...

m_linear    = lm(prestige ~ income)
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_linear,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",main='Linear Fit')
lines(income.grid,preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



wider because here we have fewer observations

```
par(mfrow=c(2,2))
plot(m_linear)
```

heteroschedastic

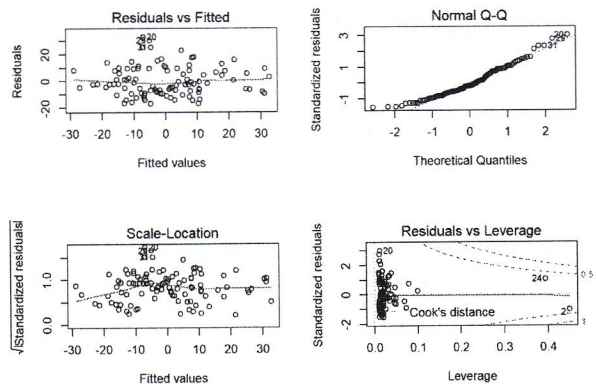



```
# Of course it doesn't make much sense... let's try a poly fit
m_list = lapply(1:10,function(degree){lm(prestige ~ poly(income,degree=degree))})
do.call(anova,m_list)
```

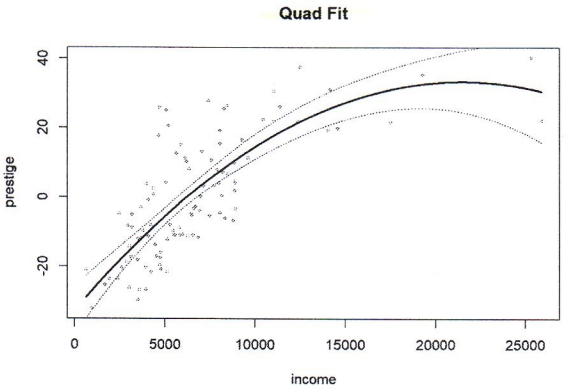
```
## Analysis of Variance Table
##
## Model  1: prestige ~ poly(income, degree = degree)
## Model  2: prestige ~ poly(income, degree = degree)
## Model  3: prestige ~ poly(income, degree = degree)
## Model  4: prestige ~ poly(income, degree = degree)
## Model  5: prestige ~ poly(income, degree = degree)
## Model  6: prestige ~ poly(income, degree = degree)
## Model  7: prestige ~ poly(income, degree = degree)
## Model  8: prestige ~ poly(income, degree = degree)
## Model  9: prestige ~ poly(income, degree = degree)
## Model 10: prestige ~ poly(income, degree = degree)
##    Res.Df   RSS Df Sum of Sq       F    Pr(>F)
## 1     100 14616
## 2      99 12077  1   2539.27 20.2010 2.047e-05 ***
## 3      98 12070  1      7.37  0.0586    0.8093
## 4      97 12062  1      7.11  0.0566    0.8125
## 5      96 12062  1      0.29  0.0023    0.9616
## 6      95 11892  1    169.83  1.3510    0.2481
## 7      94 11722  1    170.72  1.3582    0.2469
## 8      93 11497  1    224.80  1.7884    0.1845
## 9      92 11454  1     42.94  0.3416    0.5603
## 10     91 11439  1     15.14  0.1205    0.7293
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

it seems that degree = 2 will be the best choice for it

```
plot(m_list[[2]])
```



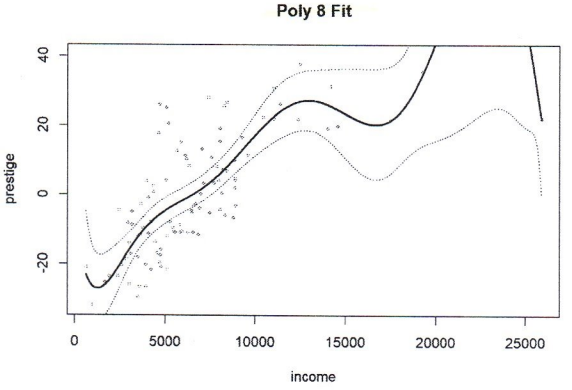

```
preds    = predict(m_list[[2]],list(income=income.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
par(mfrow=c(1,1))
plot(income, prestige, xlim=range(income.grid) ,cex =.5, col =" darkgrey ",main='Quad Fit')
lines(income.grid,preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



We're doing a better job (but we still have a lot of uncertainty at the end)

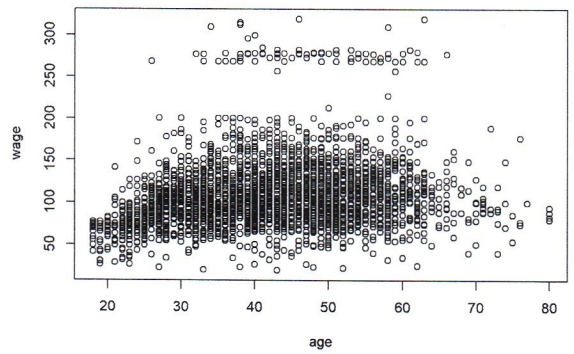
Let's try an overfit:

```
preds    = predict(m_list[[8]],list(income=income.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)

plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",main='Poly 8 Fit')
lines(income.grid,preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



Here we're also fitting the noise, not only the signal

```
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### LOCAL REGRESSION
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
load('nlr_data.rda')
attach(Wage)
plot(age,wage)
```



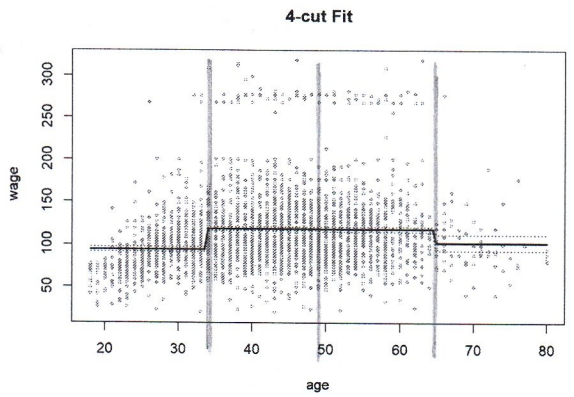

```
# Let's try to fit a stepwise constant regression
table(cut(age,4))
```

if we don't add anything else then it'll cut in 4 equal-sized bins (equal sized in length, not in # elements)

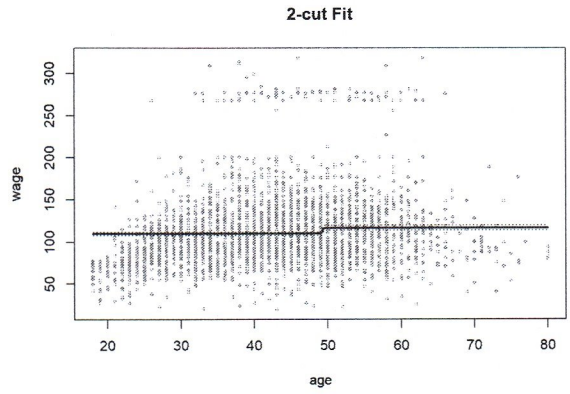
```
##
## (17.9,33.5]   (33.5,49]   (49,64.5] (64.5,80.1]
##         750        1399         779          72
```

```
m_cut     = lm(wage ~ cut(age,4))
age.grid = seq(range(age)[1],range(age)[2],by=0.5)
preds     = predict(m_cut,list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey ",main='4-cut Fit')
lines(age.grid,preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```

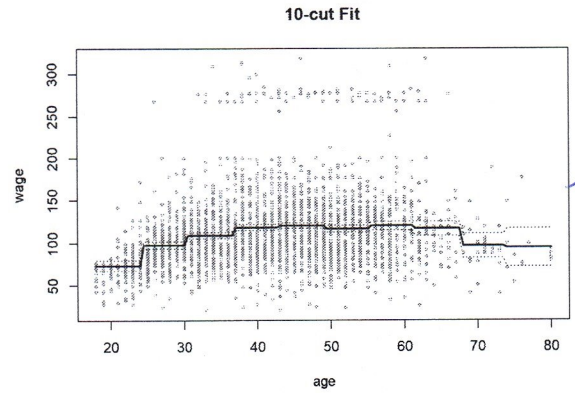
regression in each bin




```
m_cut     = lm(wage ~ cut(age,2))
age.grid = seq(range(age)[1],range(age)[2],by=0.5)
preds     = predict(m_cut,list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey ",main='2-cut Fit')
lines(age.grid, preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```

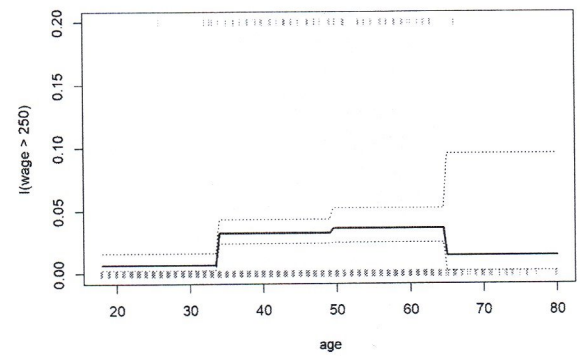



```
m_cut     = lm(wage ~ cut(age,10))
age.grid = seq(range(age)[1],range(age)[2],by=0.5)
preds     = predict(m_cut,list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey ",main='10-cut Fit')
lines(age.grid, preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```

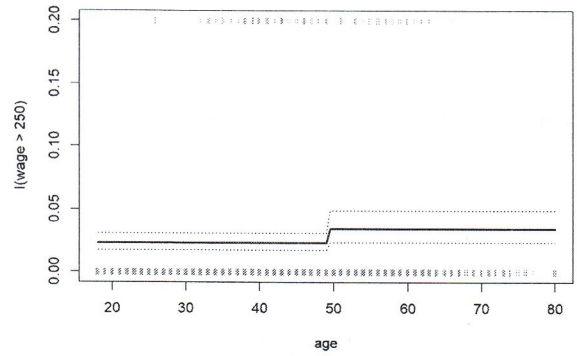


reduced bias at cost of an higher variance

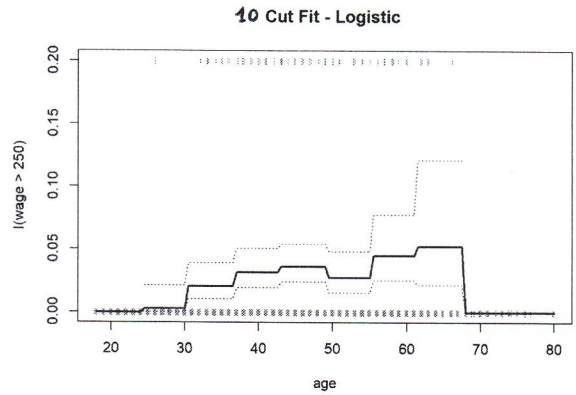
```
# The same thing can be done also in a glm setting
m_logit        = glm(I(wage>250) ~ cut(age,4),family='binomial')
preds          = predict(m_logit,list(age=age.grid),se=T)
pfit           = exp(preds$fit )/(1+ exp( preds$fit ))
se.bands.logit = cbind(preds$fit +2* preds$se.fit , preds$fit -2*preds$se.fit)
se.bands       = exp(se.bands.logit)/(1+ exp(se.bands.logit))
plot(age, I(wage >250), xlim=range(age.grid), type ="n", ylim=c(0 ,.2) )
points(jitter (age), I((wage >250) /5) , cex =.5, pch ="|", col =" darkgrey ")
lines(age.grid, pfit, lwd =2, col =" blue", main='4 Cut Fit - Logistic')
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```




```
m_logit        = glm(I(wage>250) ~ cut(age,2),family='binomial')
# We can probably be content, here, with a smaller model than before
preds          = predict(m_logit,list(age=age.grid),se=T)
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))
plot(age, I(wage >250), xlim=range(age.grid), type ="n", ylim=c(0 ,.2) )
points(jitter (age), I((wage >250) /5), cex =.5, pch ="|", col =" darkgrey ")
lines(age.grid, pfit, lwd =2, col =" blue",main='2 Cut Fit - Logistic')
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```

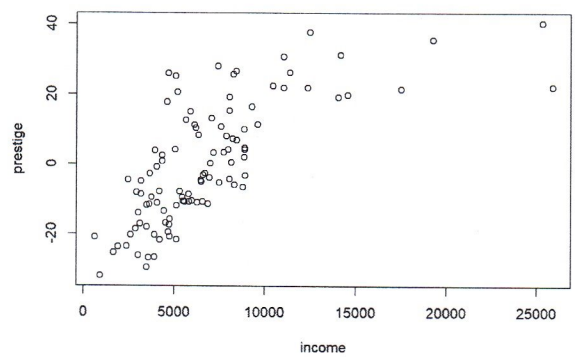



```
m_logit        = glm(I(wage>250) ~ cut(age,10),family='binomial')
preds          = predict(m_logit,list(age=age.grid),se=T)
pfit           = exp(preds$fit )/(1+ exp( preds$fit ))
se.bands.logit = cbind(preds$fit +2* preds$se.fit , preds$fit -2*preds$se.fit)
se.bands       = exp(se.bands.logit)/(1+ exp(se.bands.logit))
plot(age ,I(wage >250), xlim=range(age.grid), type ="n", ylim=c(0 ,.2), main='10 Cut Fit - Logistic')
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)
```



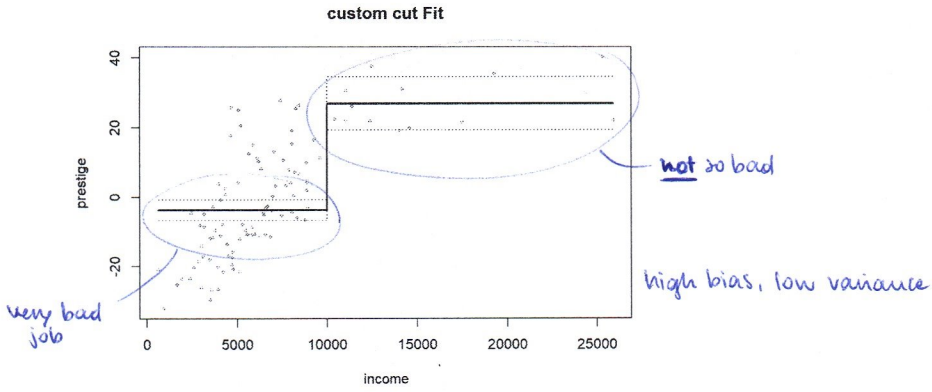

```
detach(Wage)
attach(Prestige)
```

```
plot(income,prestige)
```




```
# Cutpoint at around 10000? how?
m_cut       = lm(prestige ~ cut(income,breaks = c(min(income),10000,max(income))))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_cut,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",main='custom cut Fit')
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```

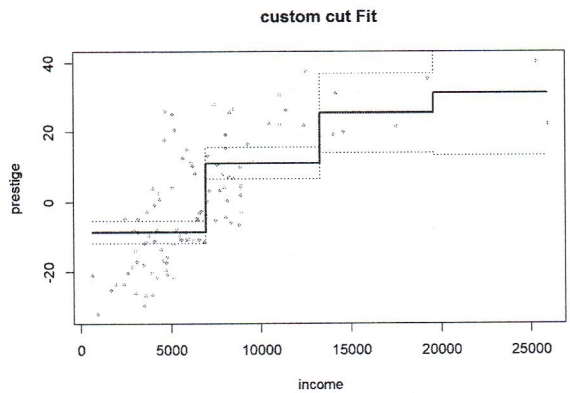
— we just need to specify it inside the "cut" function




```
# We can go fancier...
table(cut(income,4))
```

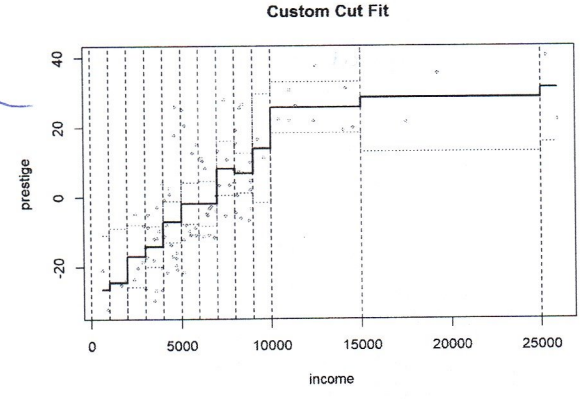
```
##
##      (586,6.93e+03] (6.93e+03,1.32e+04] (1.32e+04,1.96e+04] (1.96e+04,2.59e+04]
##                  63                  32                   5                   2
```

```
m_cut       = lm(prestige ~ cut(income,4))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_cut,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",main='custom cut Fit')
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```




```
# Uneven bins
br          = c(seq(0,10000,by=1000),seq(15000,35000,by=10000))
m_cut       = lm(prestige ~ cut(income,breaks = br ))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_cut,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",main='Custom Cut Fit')
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
abline(v=br,lty=2)

detach(Prestige)

# WEIGHTED LOCAL AVERAGING
library(np)
```

we want smaller bins where there are more data (since this procedure is a specific case of splines, having more and smaller bins where we have more data is equivalent to have "more basis" where we have more data and where we expect the most of variation to happen)

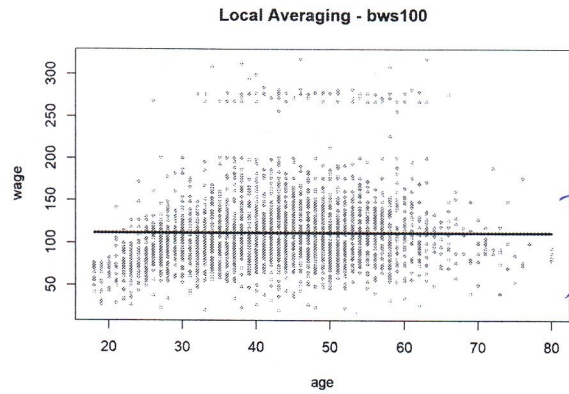


```
attach(Wage)

# Let's implement local averaging
m_loc     = npreg(age,wage,ckertype='uniform',bws=100)
```

```
age.grid = seq(range(age)[1],range(age)[2],by=0.5)
preds    = predict(m_loc,list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey ",main='Local Averaging - bws100')
lines(age.grid, preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```

This code is the implementation of local averaging: the implement. uses the package for the kernel regression. (local averaging is a special case of kernel regression when we have a uniform kernel of fix grid)

We take a neighborhood of each point, we take the average and obtain a piece of function.

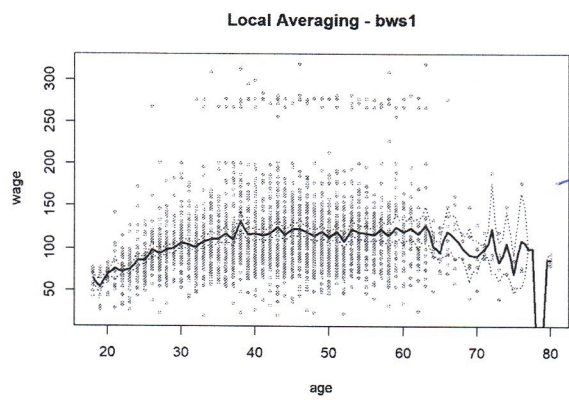


very biased, low variance

averaging on 100 (the amplitude goes outside the range of the data "$\bar{\sigma}$") ⇒ we're obtaining a constant prediction

```
# Too much

m_loc     = npreg(age,wage,ckertype='uniform',bws=1)
```

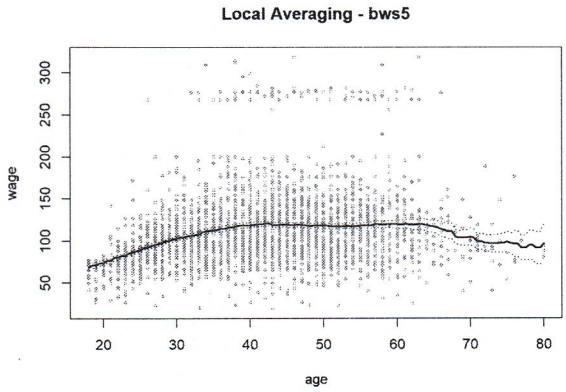
```
age.grid = seq(range(age)[1],range(age)[2],by=0.5)
preds    = predict(m_loc,list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey ", main='Local Averaging - bws1')
lines(age.grid,preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```



this is basically the average of every age (since bws = 1)

```
# Not enough

m_loc     = npreg(age,wage,ckertype='uniform',bws=5)
```

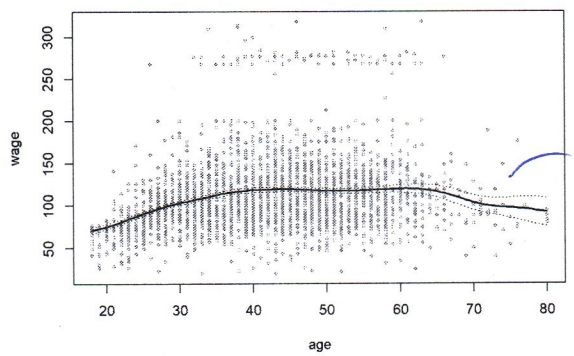
```
age.grid = seq(range(age)[1],range(age)[2],by=0.5)
preds    = predict(m_loc,list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey ",main='Local Averaging - bws5')
lines(age.grid, preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```




```
# Good!
m_loc     = npreg(age,wage,ckertype='gauss',bws=3)
age.grid = seq(range(age)[1],range(age)[2],by=0.5)
preds     = predict(m_loc,list(age=age.grid),se=T)
se.bands = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey ",
     main='Local Averaging, Gauss Kernel - bws3')
lines(age.grid, preds$fit, lwd =2, col =" blue")
matlines(age.grid, se.bands,lwd =1, col =" blue",lty =3)
```

Let's change the kernel! → Gaussian kernel

variance of the gaussian distr.

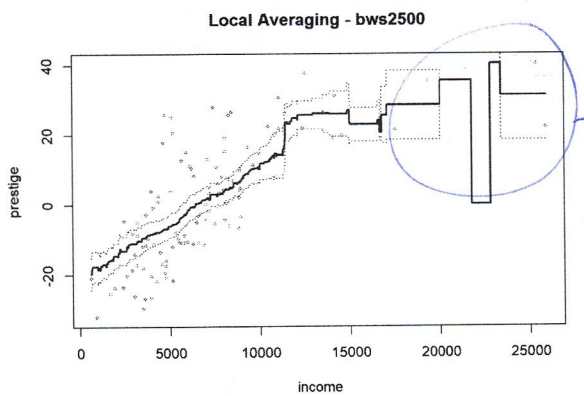


smooth kernel → smooth function

```
detach(Wage)
attach(Prestige)
```

```
### ------------------------------------------------------------------------------
### Tests on prestige dataset
### ------------------------------------------------------------------------------
m_loc        = npreg( income,as.numeric(prestige),ckertype='uniform',bws=2500)
```

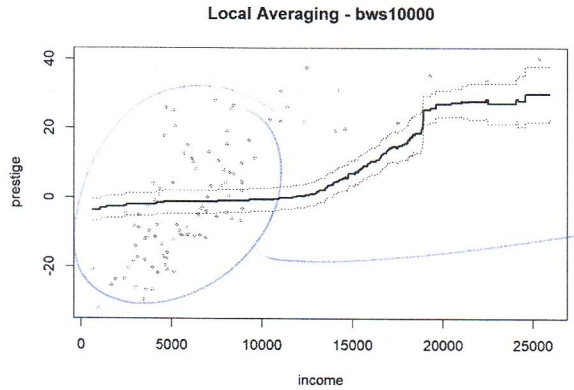
```
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_loc,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",
     main='Local Averaging - bws2500')
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



Here it goes crazy (because we have extremely few data → we don't even have the uncertainty (because we have just one point in a while))

```
m_loc        = npreg( income,as.numeric(prestige),ckertype='uniform',bws=10000)
```

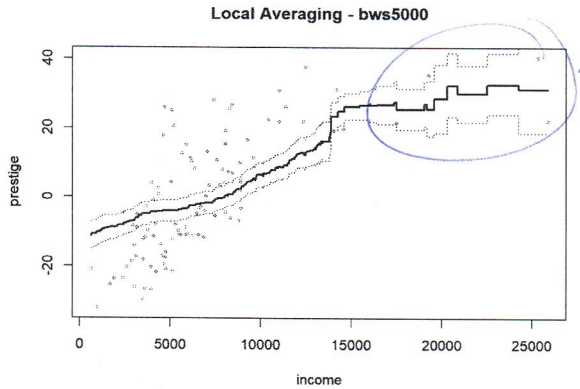
```
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_loc,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",
     main='Local Averaging - bws10000')
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



If, instead, we increase too much we have problems estimating the first part (while the ending part works)

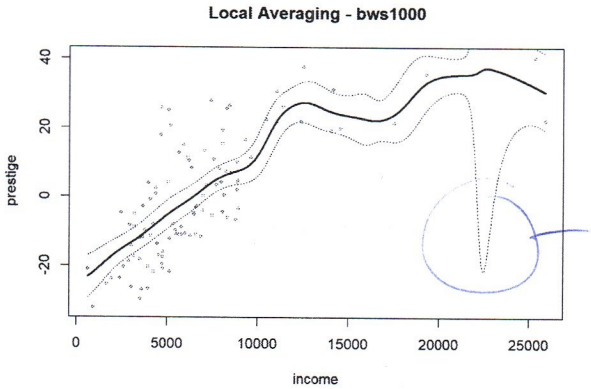
```
m_loc       = npreg( income,as.numeric(prestige),ckertype='uniform',bws=5000)
```

```
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_loc,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",
     main='Local Averaging - bws5000')
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



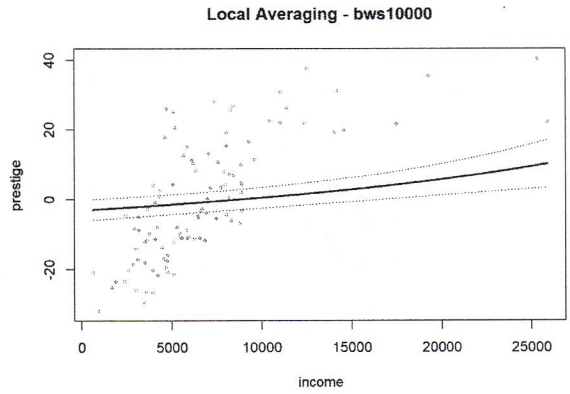
We still have some problems

```
# Let's change the kernel
m_loc       = npreg( income,as.numeric(prestige),ckertype='gaussian',bws=1000)
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_loc,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",
     main='Local Averaging - bws1000')
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```

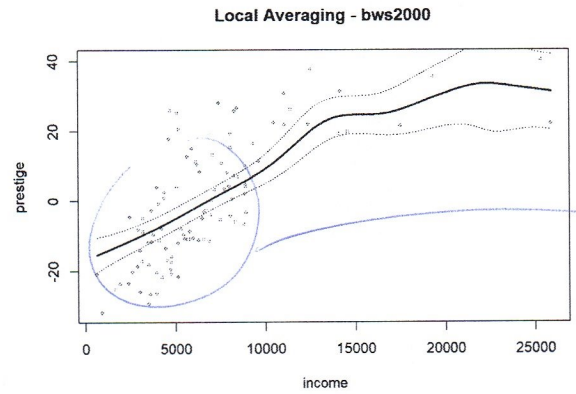


The uncertainty about these predictions is enormous

```
m_loc       = npreg( income,as.numeric(prestige),ckertype='gaussian',bws=10000)
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_loc,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",
     main='Local Averaging - bws10000')
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



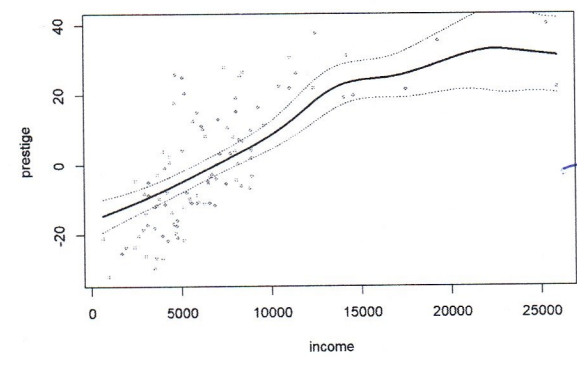

```
m_loc       = npreg( income,as.numeric(prestige),ckertype='gaussian',bws=2000)
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_loc,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",
     main='Local Averaging - bws2000')
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



still does **not** work here

```
# Let's try to use an adaptive kernel
m_loc       = npreg( income,as.numeric(prestige),ckertype='gaussian',bws=1.8,bwscaling=T)
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(m_loc,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey ",
     main='Local Averaging - Adaptive Kernel')
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```

adaptive-part

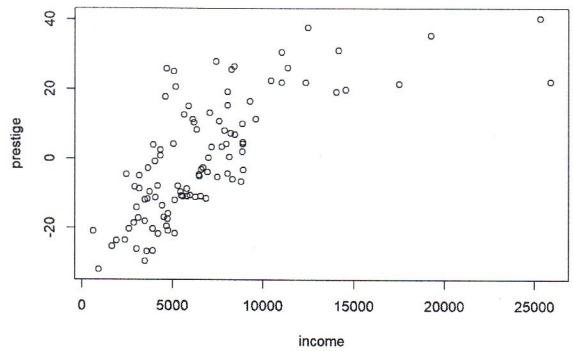


not in this case but generally it helps to have an adaptive kernel

```
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
### SPLINES TUTORIAL
### ---------------------------------------------------------------------------
### ---------------------------------------------------------------------------
load('nlr_data.rda')
library(splines)
attach(Prestige)
```

```
# Piecewise polynomial
plot(income,prestige)
```

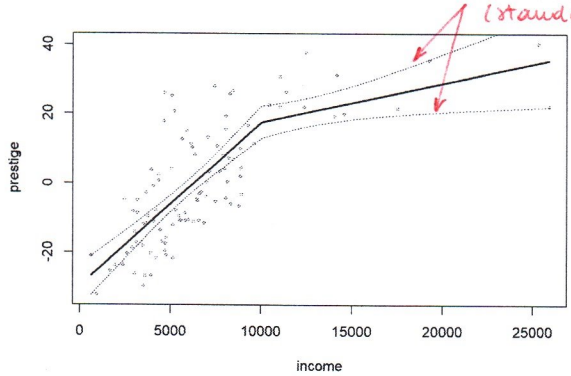
we want to grow splines from piece-wise polynomial regression
local polynomial regression (i.e. piecewise polynomial)
(another local regression method)



Let's begin with a **piecewise linear** function:

```
# There is no easy way to do it...
cutoff           = 10000
income_cut       = income>cutoff
income_cut_model = (income-cutoff)*income_cut

model_cut    = lm(prestige ~ income + income_cut_model)
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds        = predict(model_cut,list(income=income.grid,
                                      income_cut_model=(income.grid-cutoff)*(income.grid>cutoff)), se=T)
se.bands     = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```

where we want to change slope
this is a kind of dummy variable



uncertainty in the estimation (standard error of the fitted values calculated with a normality hypothesis)

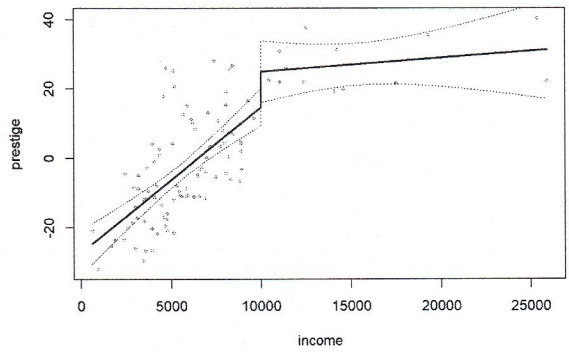
By simply putting a cutoff we've able to catch more of this weird behaviour both at the beginning and at the end.

```
summary(model_cut)
```

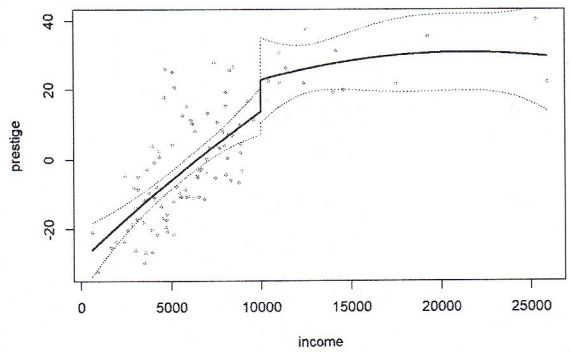
```
##
## Call:
## lm(formula = prestige ~ income + income_cut_model)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.376  -7.847  -2.887   7.689  33.467
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -2.953e+01  3.070e+00  -9.618 7.45e-16 ***
## income            4.700e-03  4.883e-04   9.625 7.19e-16 ***
## income_cut_model -3.566e-03  8.164e-04  -4.368 3.09e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.13 on 99 degrees of freedom
## Multiple R-squared:  0.5901, Adjusted R-squared:  0.5818
## F-statistic: 71.26 on 2 and 99 DF,  p-value: < 2.2e-16
```

this is the change of slope that we have at income = 10000

```
# Allow for discontinuity
model_cut    = lm(prestige ~ income + income_cut_model + I(income>cutoff))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds        = predict(model_cut,list(income=income.grid,
                                      income_cut_model=(income.grid-cutoff)*(income.grid>cutoff)), se=T)
se.bands     = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```




```
# Allow for different functional forms
dummy_disc   = I(income>cutoff)
model_cut    = lm(prestige ~ poly(income,degree = 2) + income_cut_model + dummy_disc)
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds        = predict(model_cut,list(income=income.grid,
                                      income_cut_model=(income.grid-cutoff)*(income.grid>cutoff),
                                      dummy_disc=income.grid>cutoff), se=T)
se.bands     = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```

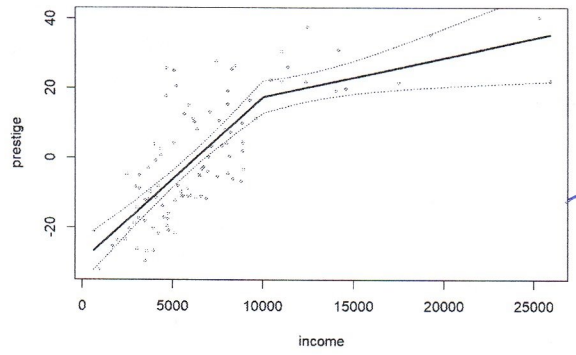



```
# How to move forward in this direction? SPLINES!
# We use the built-in "splines" package.
library(splines)

model_cut    = lm(prestige ~ bs(income, knots=(c(0,10000,30000)),degree=1))
# bs generates a design matrix that allows splines to be calculated using the lm function
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds        = predict(model_cut,list(income=income.grid,
                                      income_cut_model=(income.grid-cutoff)*(income.grid>cutoff),
                                      dummy_disc=income.grid>cutoff), se=T)
```

so we're using least-squares

"basis splines"

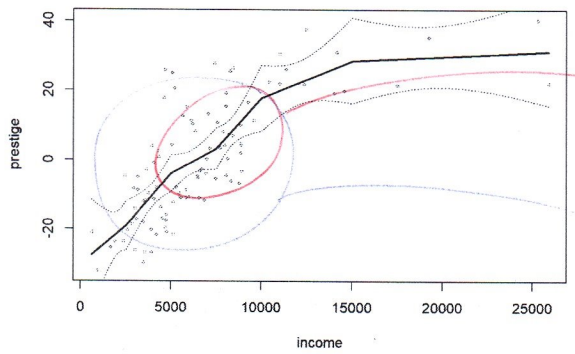
```
se.bands     = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



← This recreates the same results that we had before without all the dummy variables or weird things in the code

```
# Piecewise linear
br          = c(seq(0,10000,by=2500),15000,30000)
model_cut   = lm(prestige ~ bs(income, knots=br,degree=1))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(model_cut,list(income=income.grid) ,se=T)
```

```
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



We may want everywhere the same concavity: How can we change it? We'll see this at the end

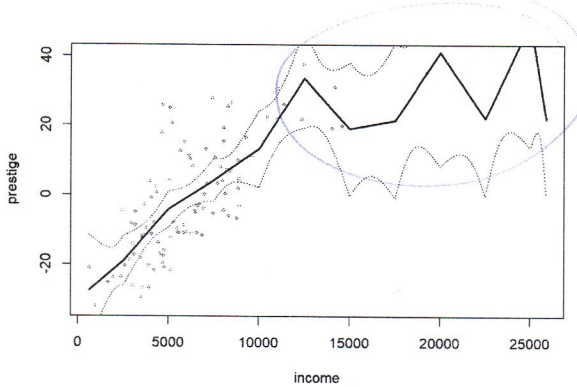
We want to select more knots where we have more data (to capture more variation)

Note: putting too many knots where we shouldn't may cause some problems (↓)

```
model_cut   = lm(prestige ~ bs(income, knots=seq(0,30000, by=2500),degree=1))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(model_cut,list(income=income.grid) ,se=T)
```

← perfectly equally-spaced knots

```
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



This is extremely bad. We have one data per bin and so we're basically interpolating

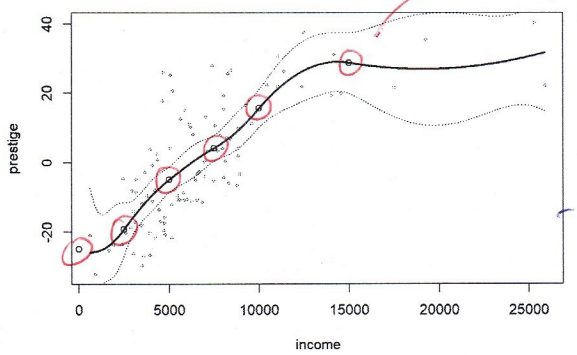
Selecting where we want our knots to be is a very serious problem!

Let's change the **degree** of the splines!

```
model_cut   = lm(prestige ~ bs(income, knots=br ,degree=2))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(model_cut,list(income=income.grid),se=T)
```

```
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
knots_pred  = predict(model_cut, list(income=br))
```

```
points(br, knots_pred)
lines(income.grid,preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```

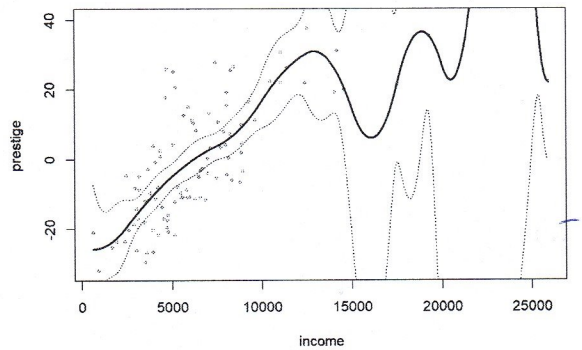


We can also see where are the points that separate bins (notice that we have smooth connections)

Again, the unequally spaced strategy seems to be good

```
model_cut    = lm(prestige ~ bs(income, knots=seq(0,30000, by=2500),degree=2))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds        = predict(model_cut,list(income=income.grid) ,se=T)
```

```
se.bands     = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```

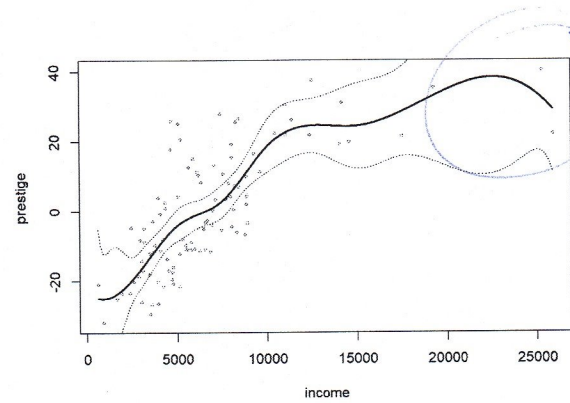


Again, equally spaced generates some problems where we have fewer data (we're interpolating again)

```
model_cut    = lm(prestige ~ bs(income, knots=br ,degree=3))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds        = predict(model_cut,list(income=income.grid) ,se=T)
```

(CUBIC SPLINES)

```
se.bands     = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```



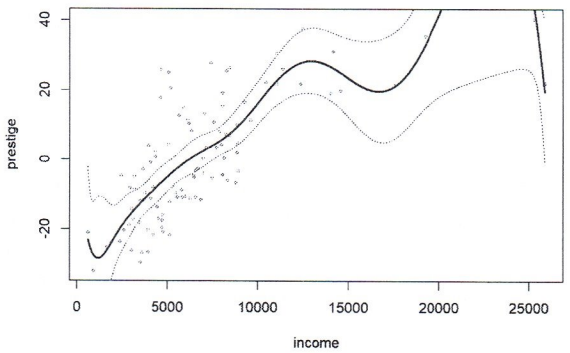
One of the problems of these splines is the weird behaviour at the beginning/end of the x domain. For instance here it makes no sense for the line at the begin to go up (✓), it would make more sense to: ✓ ✗

Same case at the end (*).

```
model_cut    = lm(prestige ~ bs(income, knots=br ,degree=4))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds        = predict(model_cut,list(income=income.grid) ,se=T)
```

Suppose that we want to inspect the first derivative of the relation with a cubic spline ⇒ we must go with degree = 4 at the beginning

```
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```




```
# I can either specify the knots, or have R select automatically the spacing.
# I just need to specify their number knowing that:
# dof = #knots + degree
# to have 4 breaks, with degree 3, we need 7 dofs

model_cut   = lm(prestige ~ bs(income, degree=3,df=7))
summary(model_cut)
```

this creates the knots in the more relevant quantities (it's a smart-knots-placement)

equally spaced empirical quantiles

```
##
## Call:
## lm(formula = prestige ~ bs(income, degree = 3, df = 7))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.434  -8.707  -1.536   8.482  31.538
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     -26.463      9.353  -2.829   0.0057 **
## bs(income, degree = 3, df = 7)1   4.026     18.499   0.218   0.8282
## bs(income, degree = 3, df = 7)2   9.301     10.804   0.861   0.3915
## bs(income, degree = 3, df = 7)3  25.743     11.013   2.338   0.0215 *
## bs(income, degree = 3, df = 7)4  25.262      9.953   2.538   0.0128 *
## bs(income, degree = 3, df = 7)5  70.124     16.679   4.204 5.97e-05 ***
## bs(income, degree = 3, df = 7)6  49.945     23.862   2.093   0.0390 *
## bs(income, degree = 3, df = 7)7  57.670     12.703   4.540 1.67e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.25 on 94 degrees of freedom
## Multiple R-squared:  0.6023, Adjusted R-squared:  0.5727
## F-statistic: 20.34 on 7 and 94 DF,  p-value: < 2.2e-16
```

estimate of the parameters (this is not like with the anova thing where we were testing if we should add a degree or not)

```
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(model_cut,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
knots       = attr(bs(income, degree=3,df=7),'knots')
knots
```

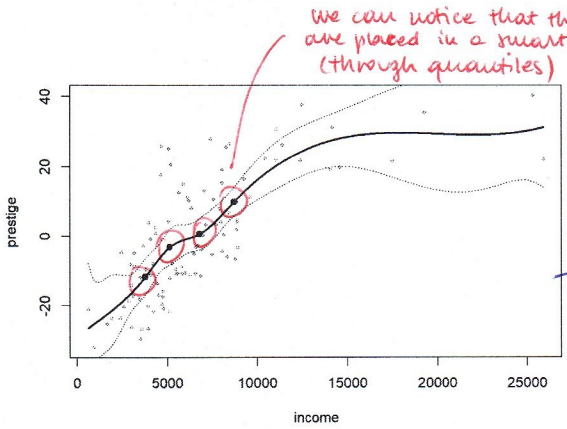
extraction of the knots

We can also extract "degree", "intercept" (= if we're fitting an intercept or not), and so on

```
##    20%    40%    60%    80%
## 3773.2 5134.0 6790.4 8709.0
```

with this division every bin has 25% of the data

```
knots_pred  = predict(model_cut,data.frame(income=knots))
points(knots, knots_pred, col='blue',pch=19)
matlines(income.grid, se.bands ,lwd =1, col =" blue",lty =3)
```



We can notice that the knots are placed in a smart way (through quantiles)

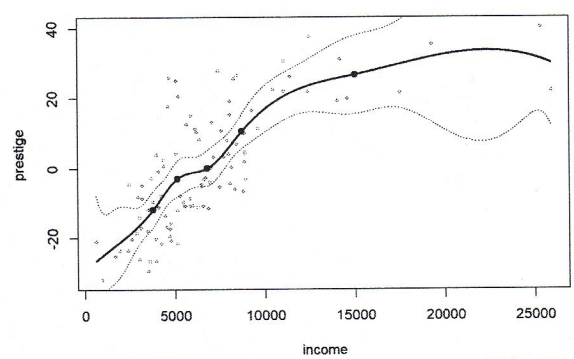
With very few degrees of freedom and very few knots we have a pretty good result

```
# Merge the two approaches
br          = c(quantile(income, probs = c(0.2, 0.4, 0.6, 0.8)), 15000)
model_cut   = lm(prestige ~ bs(income, degree=3, knots=br))
summary(model_cut)
```

we specify the knots at quantiles of the data and we add manually another knot (at 15.000)

```
##
## Call:
## lm(formula = prestige ~ bs(income, degree = 3, knots = br))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.661  -8.323  -1.597   8.425  31.344
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       -26.600      9.398  -2.830 0.005699 **
## bs(income, degree = 3, knots = br)1   4.841     18.668   0.259 0.795980
## bs(income, degree = 3, knots = br)2   8.725     10.927   0.798 0.426627
## bs(income, degree = 3, knots = br)3  26.544     11.205   2.369 0.019906 *
## bs(income, degree = 3, knots = br)4  24.172     10.291   2.349 0.020947 *
## bs(income, degree = 3, knots = br)5  49.586     12.691   3.907 0.000177 ***
## bs(income, degree = 3, knots = br)6  53.103     21.863   2.429 0.017069 *
## bs(income, degree = 3, knots = br)7  64.667     31.290   2.067 0.041541 *
## bs(income, degree = 3, knots = br)8  56.732     12.931   4.387 3.02e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.29 on 93 degrees of freedom
## Multiple R-squared:  0.6031, Adjusted R-squared:  0.569
## F-statistic: 17.67 on 8 and 93 DF,  p-value: 9.451e-16
```

```
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(model_cut,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
knots_pred  = predict(model_cut,data.frame(income=br))
points(br,knots_pred, col='blue',pch=19)
matlines(income.grid ,se.bands ,lwd =1, col =" blue",lty =3)
```

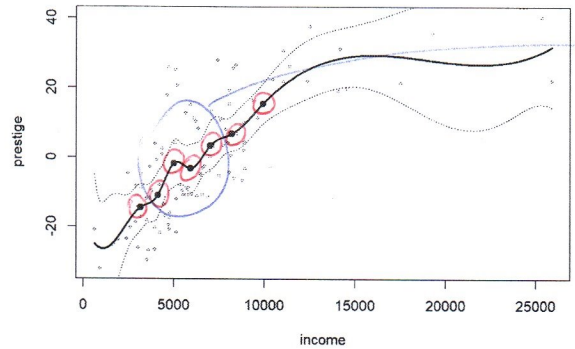


let's have **7 knots** (with degree = 3) ← still we're using the equally-spaced quantiles

```
model_cut   = lm(prestige ~ bs(income, degree=3,df=10))
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(model_cut,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
knots       = attr(bs(income, degree=3,df=10),'knots')
knots
```

```
##     12.5%       25%     37.5%       50%     62.5%       75%     87.5%
## 3156.125 4106.000 5015.625 5930.500 7000.375 8187.250 9907.625
```

```
knots_pred  = predict(model_cut,list(income=knots))
points(knots, knots_pred, col='blue',pch=19)
matlines(income.grid, se.bands, lwd =1, col =" blue",lty =3)
```

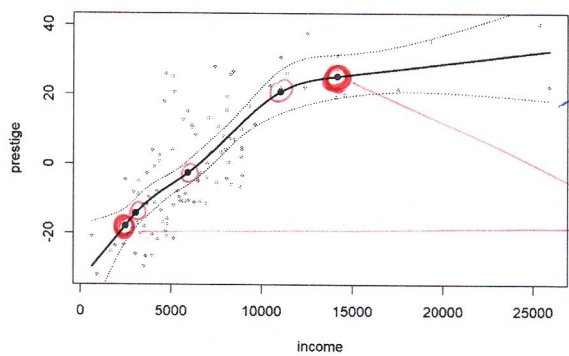


It seems like here we've little overfitting

we select two boundary knots outside of which we'll build just a linear predictor, while inside we build the standard cubic splines

notice that **ns** has **CUBIC SPLINES** and we cannot change it

```
# How to avoid weird behaviour at the boundaries of the domain? NATURAL SPLINES:
knots          = quantile(income,probs=c(0.1,0.5,0.9))
boundary.knots = quantile(income,probs=c(0.05,0.95))
model_cut      = lm(prestige ~ ns(income,knots=knots,Boundary.knots=boundary.knots)) #defaults to 3 knots
income.grid = seq(range(income)[1],range(income)[2],by=0.5)
preds       = predict(model_cut,list(income=income.grid),se=T)
se.bands    = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(income, prestige, xlim=range(income.grid), cex =.5, col =" darkgrey " )
lines(income.grid, preds$fit, lwd =2, col =" blue")
knots_pred  = predict(model_cut,list(income=knots))
points(knots, knots_pred, col='blue',pch=19)
boundary_pred = predict(model_cut,list(income=boundary.knots))
points(boundary.knots, boundary_pred, col='red',pch=19)
matlines(income.grid ,se.bands ,lwd =1, col =" blue",lty =3)
```



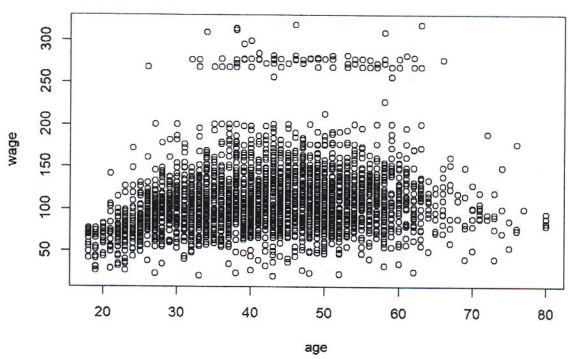
The fitting seems pretty good →

boundary knots

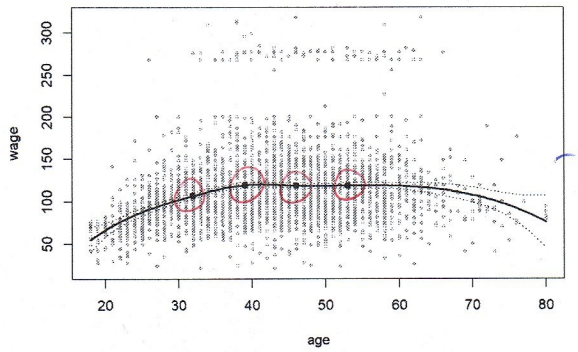
we've avoiding to give degrees of freedom where they're not needed (we're not having the previous weird behaviours) moreover in the middle the behaviour is still pretty good (the uncertainty is under control)

If we care a lot about what happens in the boundary **NATURAL SPLINES** is the way (However the problems which remain are : where to put boundaries knots and how many knots to put inside)

```
# The specification of knots can of course be non-quantile driven...
# Exercise: let's try to work with our other dataset
detach(Prestige)
attach(Wage)

plot(age,wage)
```

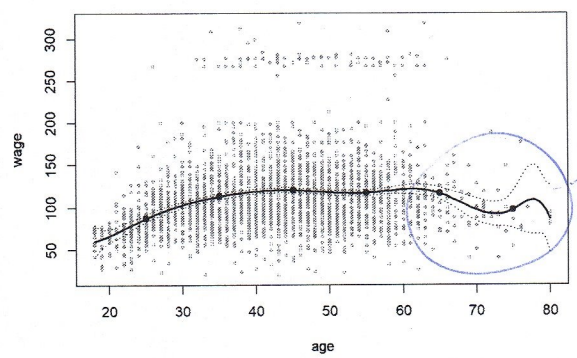


```
model_cut  = lm(wage ~ bs(age, degree=3,df=7)) # 4 knots
age.grid   = seq(range(age)[1],range(age)[2],by=0.5)
preds      = predict(model_cut,list(age=age.grid),se=T)
se.bands   = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey " )
lines(age.grid, preds$fit, lwd =2, col =" blue")
knots      = attr(bs(age, degree=3,df=7),'knots')
knots_pred = predict(model_cut,list(age=knots))
points(knots, knots_pred, col='blue',pch=19)
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```



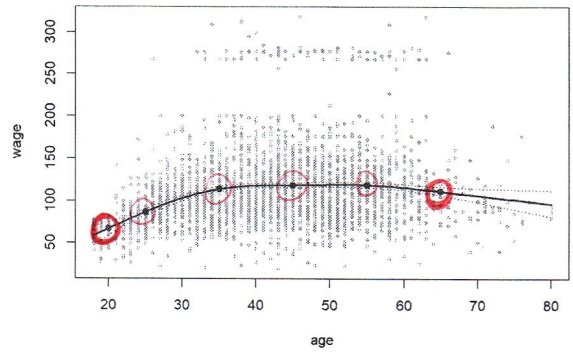
Quite good

```
# Let's try equispaced knots
knots      = seq(25,75,by=10)
model_cut  = lm(wage ~ bs(age, degree=3,knots=knots))
age.grid   = seq(range(age)[1],range(age)[2],by=0.5)
preds      = predict(model_cut,list(age=age.grid),se=T)
se.bands   = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey " )
lines(age.grid, preds$fit,lwd =2, col =" blue")
knots_pred = predict(model_cut,list(age=knots))
points(knots, knots_pred, col='blue',pch=19)
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```



overfitting because we don't have enough data in this region

```
# We can clearly see some wigglyness on the right side... natural spline?
knots          = seq(25,60,by=10)
boundary.knots = c(20,65)
model_cut      = lm(wage ~ ns(age, knots=knots,Boundary.knots = boundary.knots ))
age.grid       = seq(range(age)[1],range(age)[2],by=0.5)
preds          = predict(model_cut,list(age=age.grid),se=T)
se.bands       = cbind(preds$fit +2* preds$se.fit ,preds$fit -2* preds$se.fit)
plot(age, wage, xlim=range(age.grid), cex =.5, col =" darkgrey " )
lines(age.grid, preds$fit, lwd =2, col =" blue")
knots_pred     = predict(model_cut,list(age=knots))
points(knots, knots_pred, col='blue',pch=19)
boundary_pred  = predict(model_cut,list(age=boundary.knots))
points(boundary.knots, boundary_pred, col='red',pch=19)
matlines(age.grid, se.bands, lwd =1, col =" blue",lty =3)
```




```
# waaay better

# Smoothing splines
# You can either specify the equivalent DoF
length(table(age))
```

based on the idea of minimizing a penalization (the standard smoothing spline setting is the minimization of the sum of square error + penalty term (= the integral of the second derivative of the function)). The idea behind is that we want a spline that is smooth.

(we don't want many changes in the concavity (represented by the second derivative))

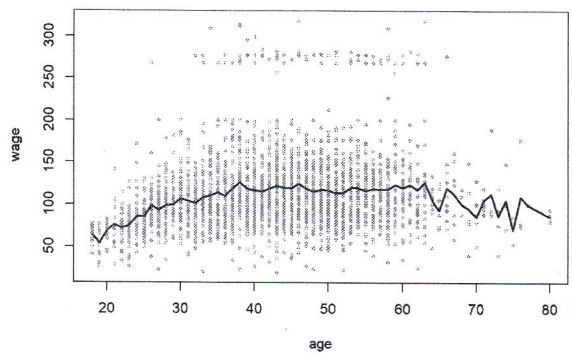
```
## [1] 61
```

```
fit = smooth.spline(age,wage,df=61)
plot(age, wage, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```

There are 2 ways to specify the amount of smoothness:

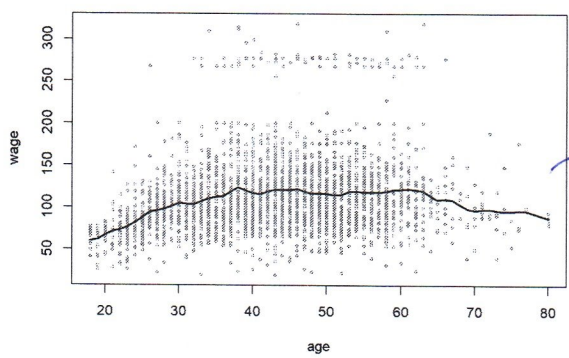
- degrees of freedom
- value of the smoothing parameter

the max number of degrees of freedom we can use is the number of different values in the x axes (in our case = 61)



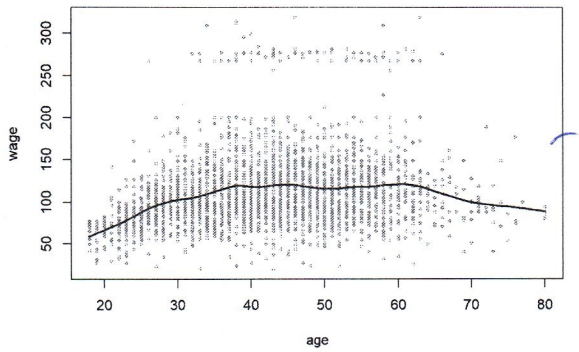
Using degree of freedom = # different values of x (= degrees of freedom max) is equivalent to interpolating the data and trying to smooth as much as possible the interpolation. The result however may be extremely rough.

```
fit = smooth.spline(age,wage,df=30)
plot(age, wage, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```



way smoother, still a little bit rough

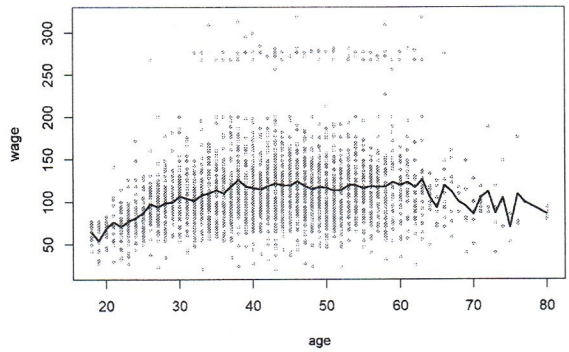
```
fit = smooth.spline(age,wage,df=16)
plot(age, wage, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```



good

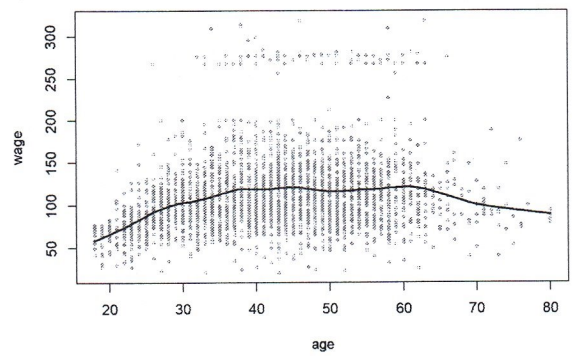
```
# or specify the value of the smoothing parameter...
fit = smooth.spline(age,wage,lambda=1e-10)
plot(age, wage, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```

we're not giving a lot of weight to the penalty term

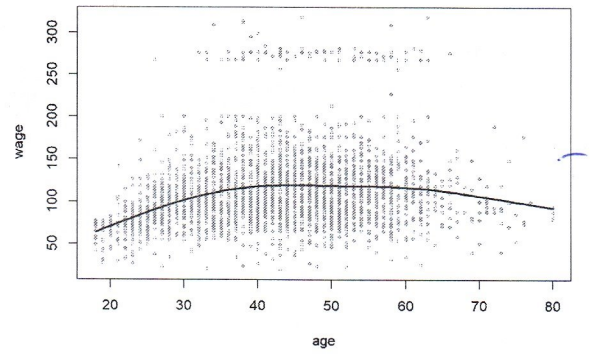



```
fit = smooth.spline(age,wage,lambda=1e-3)
plot(age, wage, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```

more penalization




```
fit = smooth.spline(age,wage,lambda=1e-1)
plot(age, wage, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```



The fitting in this case is very good

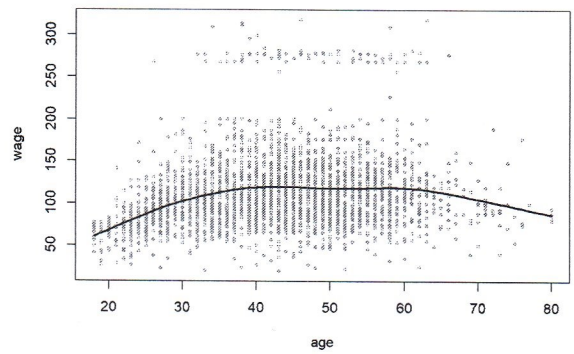
generalized cross-validation

```
# In general, what you want to do, is to optimize the CV error, or the GCV error
fit = smooth.spline(age,wage,cv=T)
```

we should choose this because it's more reliable (GCV is used when CV is computationally intensive)

```
plot(age, wage, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```

in this way **lambda** and **df** are choosen in an automatic way (to minimize the **CV** error)




```
fit$lambda
```

```
## [1] 0.02792303
```

```
fit$df
```

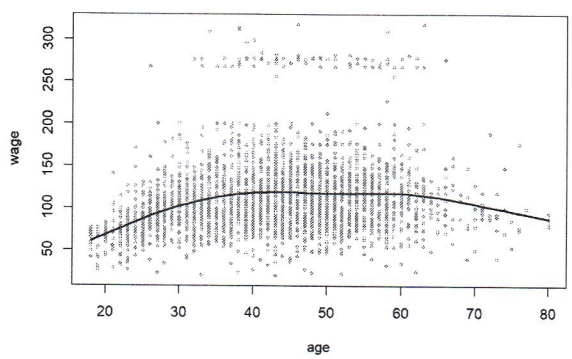
```
## [1] 6.794596
```

```
fit = smooth.spline(age,wage,cv=F)
plot(age, wage, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```

if we put it **F** then we use **GCV** error (instead of CV error)

(it gives slightly different results but not so much)

The idea is: use CV error when it's computationally feasible and GCV error when it's not




```
fit$lambda
```

```
## [1] 0.0348627
```

```
fit$df
```

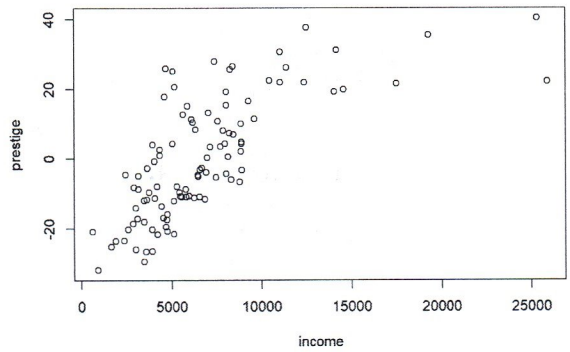
```
## [1] 6.468299
```

```
# like with the previous methods, it is very easy to forecast...
predict(fit, c(50.5, 60.5, 90.5))
```

```
## $x
## [1] 50.5 60.5 90.5
##
## $y
## [1] 117.92704 117.90889  69.67348
```

```
# let's apply it to our other dataset...
detach(Wage)
attach(Prestige)
```

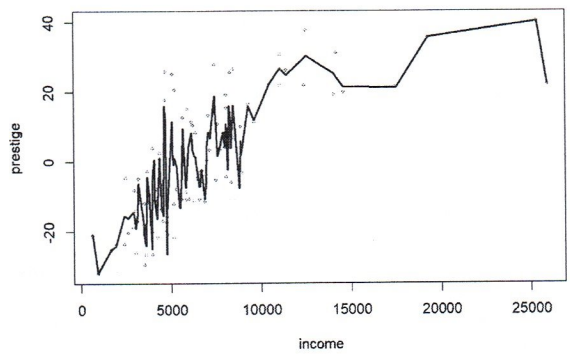
```
plot(income, prestige)
```




```
length(table(income))
```

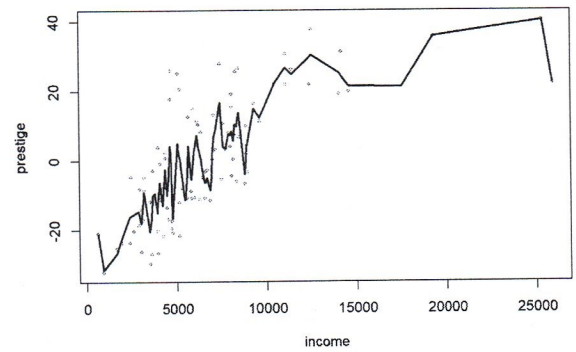
```
## [1] 100
```

```
fit = smooth.spline(income,prestige,df=100)
plot(income, prestige, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```



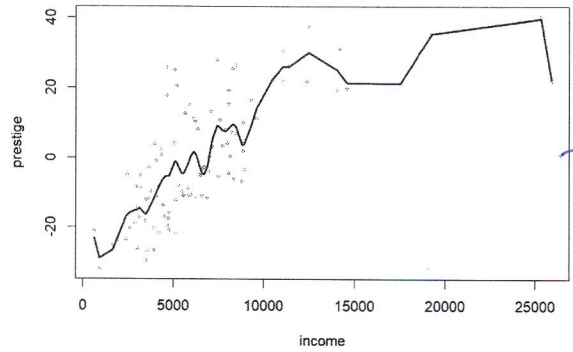
Very very rough
(not at the end because it's basically connecting dots)

```
fit = smooth.spline(income,prestige,df=50)
plot(income, prestige, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```



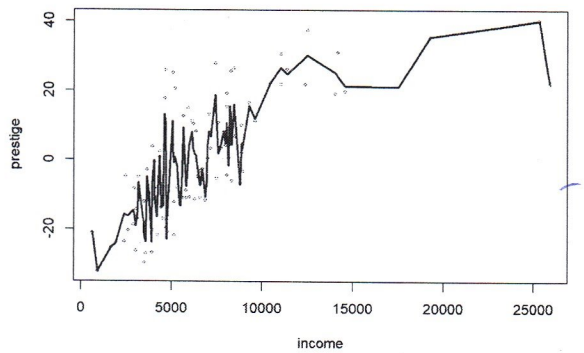
Doesn't change much
(still very bad)

```
fit = smooth.spline(income,prestige,df=25)
plot(income, prestige, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```



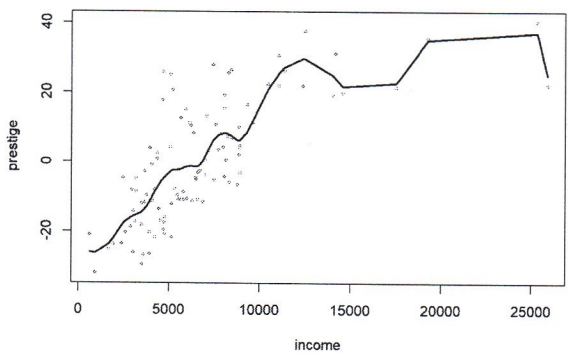
← slightly getting better

```
fit = smooth.spline(income,prestige,lambda=1e-10)
plot(income, prestige, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```

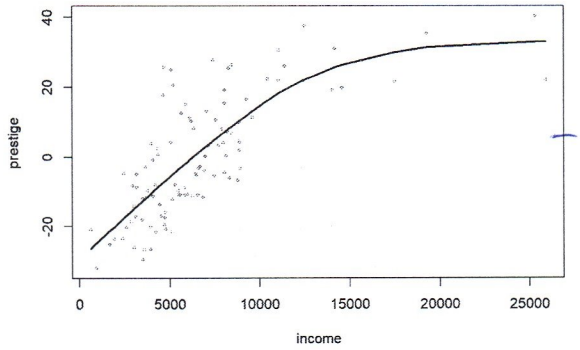


← Not so much regularization

```
fit = smooth.spline(income,prestige,lambda=1e-5)
plot(income, prestige, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```



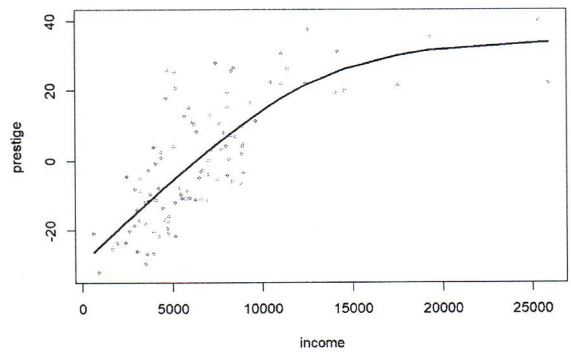

```
fit = smooth.spline(income,prestige,lambda=1e-2)
plot(income, prestige, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```



— Very good estimate!

```
fit = smooth.spline(income,prestige,cv=T)
```

```
plot(income, prestige, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```



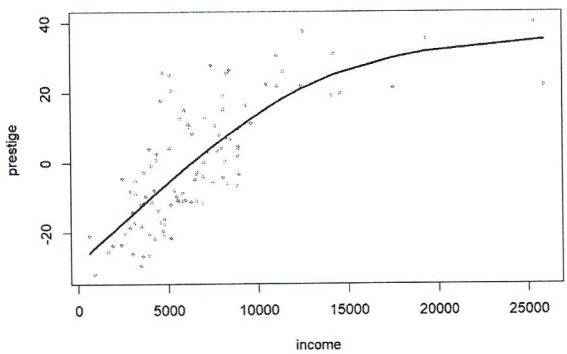

```
fit$df
```

```
## [1] 3.720986
```

```
fit$lambda
```

```
## [1] 0.01474169
```

```
fit = smooth.spline(income,prestige,cv=F)
plot(income, prestige, cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```




```
fit$df
```

```
## [1] 3.46843
```

```
fit$lambda
```

```
## [1] 0.02190045
```

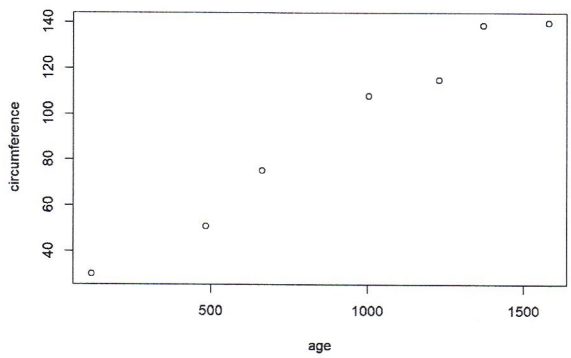
```
predict(fit, c(10000, 5000, 1000 ))
```

```
## $x
## [1] 10000  5000  1000
##
## $y
## [1]  14.127492  -5.646666 -23.958414
```

```
# What about putting constraints to my splines?
# It can actually be done pretty easily... periodic splines, monotone.

# Let's see a monotone case:
data('Orange')
head(Orange)
```

```
##   Tree  age circumference
## 1    1  118            30
## 2    1  484            58
## 3    1  664            87
## 4    1 1004           115
## 5    1 1231           120
## 6    1 1372           142
```

```
O_3 = subset(Orange,Orange$Tree==3)
attach(O_3)
plot(age,circumference)

library(fda)
```



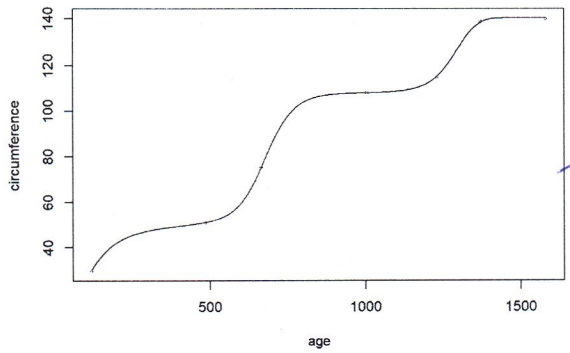
We already tried a logistic.
Let's try a monotone increasing spline.
We fit a B-spline basis with a monotonicity constraint.

```
basis = create.bspline.basis(range(age),breaks = age,norder=4)
# Starting values
cvec0 <- matrix(0,basis$nbasis,1)
Wfd0  <- fd(cvec0, basis)

start_fd = fdPar(Wfd0,Lfdobj = 2,lambda=1e4)
res      = smooth.monotone(age,circumference,start_fd)
```

→ level of derivative that we want to penalize
→ how much we want to penalize it

```
##
##  Results
##
## Iter.   PENSSE   Grad Length Intercept   Slope
## 0        37.2217      37.3879      28.7747      0.0811
## 1        13.7272      38.1167      32.0294      0.0669
## 2        2.5675      11.6761      31.8132      0.0625
## 3        0.6047      4.2934      30.4473      0.0605
## 4        0.5086      0.8978      30.1598      0.0607
## 5        0.499      0.4152      30.0503      0.0608
## 6        0.498      0.093      30.0177      0.0608
## 7        0.4979      0.0418      30.0067      0.0608
```

```
Wfd      = res$Wfdobj
beta     = res$beta
age.grid = seq(range(age)[1],range(age)[2],by=1)
pred     <- beta[1] + beta[2]*eval.monfd(age.grid, Wfd, 0)

plot(age ,circumference,cex =.5, col =" darkgrey ")
lines(age.grid,pred)
```



The fit is probably not the best one, we already saw that the logistic function was working well
(However, the point is: there are ways in which we can impose limitations and constraints to our splines)

```
# What happens instead with multivariate data?
detach(O_3)
attach(Wage)
```

GAM: GENERALIZED ADDITIVE MODELS

```
library(gam)
```

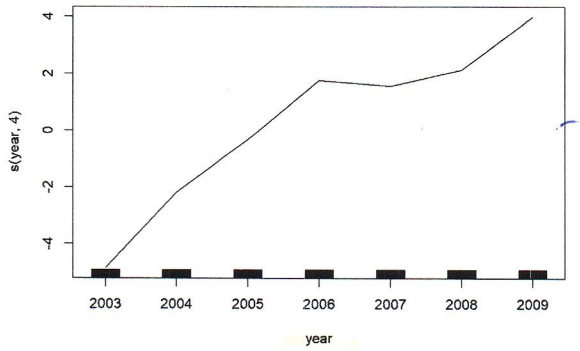
smooth term with 4 degrees of freedom for "year"
smooth term with 5 degrees of freedom for age

```
gam.m3      = gam(wage ~ s(year ,4)+s(age ,5))
age.grid    = seq(range(age)[1],range(age)[2],by=1)
year.grid   = seq(range(year)[1],range(year)[2],by=1)
summary(gam.m3)
```

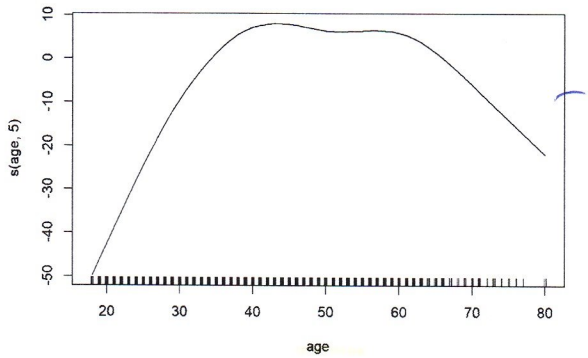
we actually don't look at the details because this won't be the library we'll use for GAMs

```
##
## Call: gam(formula = wage ~ s(year, 4) + s(age, 5))
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -99.379 -24.919  -4.895  15.563 201.905
##
## (Dispersion Parameter for gaussian family taken to be 1584.779)
##
##     Null Deviance: 5222086 on 2999 degrees of freedom
## Residual Deviance: 4738489 on 2990 degrees of freedom
## AIC: 30630.21
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##              Df  Sum Sq Mean Sq F value    Pr(>F)
## s(year, 4)    1   27732   27732  17.499 2.957e-05 ***
## s(age, 5)     1  195226  195226 123.188 < 2.2e-16 ***
## Residuals  2990 4738489    1585
## ---
## 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  0.688 0.5594
## s(age, 5)          4 41.345 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
plot(gam.m3)
```

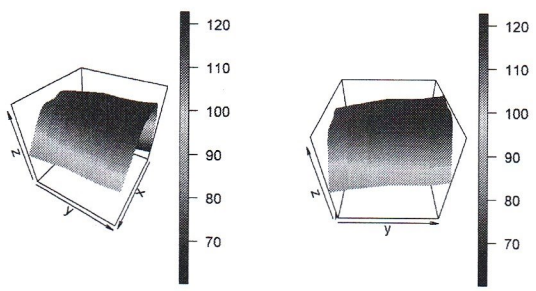


It seems like the year has a linear impact (relatively linear)

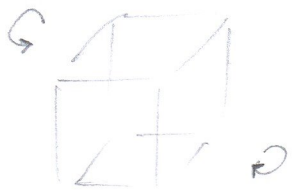


reverse bath-tub shape (we already saw it in the data, good that's pretty well respected)

```
grid        = expand.grid(age.grid,year.grid)
names(grid) = c('age','year')
pred        = predict(gam.m3,newdata = grid)

library(plot3D)
par(mfrow=c(1,2))
persp3D(z = pred, theta = 120)
persp3D(z = pred, theta = 90)
```




```
library(rgl)
persp3d(age.grid,year.grid,pred,col='blue')
```



```
options(rgl.useNULL = TRUE)
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
### GAMS
### ------------------------------------------------------------------------
### ------------------------------------------------------------------------
load('nlr_data.rda')
#install.packages('mgcv')
library(car)
```

The idea is to model the response function as the sum of smooth terms

```
attach(Prestige)
```

```
scatterplotMatrix(data.frame(Prestige$prestige, Prestige$education, Prestige$income))

# The relationship between prestige and education seems fairly linear.
# As we know, prestige and income is quite nonlinear instead.

# Let's try to fit a classical linear model
model_lm = lm(prestige ~ education + income)
summary(model_lm)
```

```
##
## Call:
## lm(formula = prestige ~ education + income)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.4040  -5.3308   0.0154   4.9803  17.6889
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.368e+01  3.219e+00 -16.676  < 2e-16 ***
## education    4.137e+00  3.489e-01  11.858  < 2e-16 ***
## income       1.361e-03  2.242e-04   6.071 2.36e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.81 on 99 degrees of freedom
## Multiple R-squared:  0.798,  Adjusted R-squared:  0.7939
## F-statistic: 195.6 on 2 and 99 DF,  p-value: < 2.2e-16
```

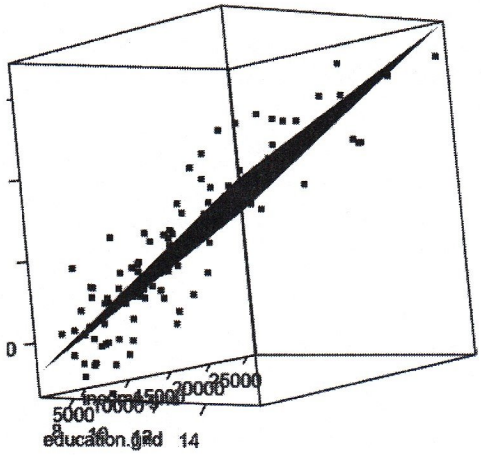
✓

✓

```
education.grid = seq(range(education)[1],range(education)[2],length.out = 100)
income.grid    = seq(range(income)[1],range(income)[2],length.out = 100)
grid           = expand.grid(education.grid,income.grid)
names(grid)    = c('education','income')
pred           = predict(model_lm,newdata=grid)
summary(model_lm)
```

```
##
## Call:
## lm(formula = prestige ~ education + income)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.4040  -5.3308   0.0154   4.9803  17.6889
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.368e+01  3.219e+00 -16.676  < 2e-16 ***
## education    4.137e+00  3.489e-01  11.858  < 2e-16 ***
## income       1.361e-03  2.242e-04   6.071 2.36e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.81 on 99 degrees of freedom
## Multiple R-squared:  0.798,  Adjusted R-squared:  0.7939
## F-statistic: 195.6 on 2 and 99 DF,  p-value: < 2.2e-16
```

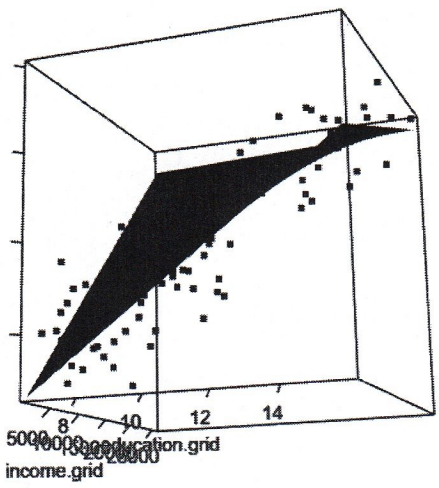
```
library(rgl)
persp3d(education.grid,income.grid,pred,col='grey30')
points3d(education,income,prestige,col='black',size=5)
rglwidget()
```




```
# What happens if I add the interaction?
model_lm = lm(prestige ~ education + income + education:income)
summary(model_lm)
```

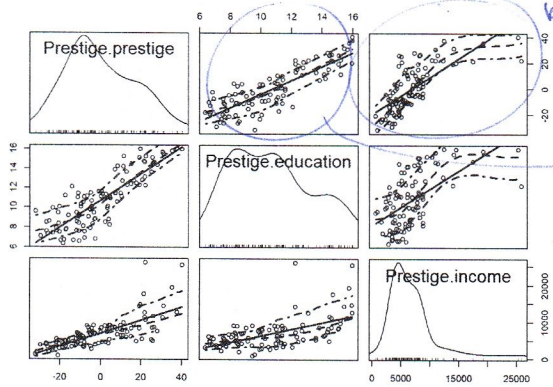
```
##
## Call:
## lm(formula = prestige ~ education + income + education:income)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.0026  -5.1918   0.3768   4.5915  19.9312
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -6.890e+01  6.581e+00 -10.469  < 2e-16 ***
## education         5.373e+00  5.797e-01   9.270 4.65e-15 ***
## income            3.944e-03  1.007e-03   3.918 0.000165 ***
## education:income -1.961e-04  7.460e-05  -2.628 0.009958 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.587 on 98 degrees of freedom
## Multiple R-squared:  0.8113, Adjusted R-squared:  0.8055
## F-statistic: 140.5 on 3 and 98 DF,  p-value: < 2.2e-16
```

```
# I manage to capture a bit of the nonlinearity with the interaction term, can I do better?
pred = predict(model_lm,newdata=grid)
persp3d(education.grid,income.grid,pred,col='grey30')
points3d(education,income,prestige,col='black',size=5)
rglwidget()
```




```
# How to go even further? mgcv package
# The main reference to follow (also, surprisingly easy and straightforward to use) is GAMs
library(mgcv)
```

library for GAMs: it automatically implements the selection of the correct amount of smoothness (it does it for all the smooth terms in the model)



nonlinear relation between the two variables

also here it is not linear (even if it's more difficult to capture)

```
model_gam  = gam(prestige ~ s(education,bs='cr') + s(income,bs='cr'))
model_gam1 = gam(prestige ~ education + s(income,bs='cr'))
summary(model_gam1)
```

s() = smooth
'cr' = cubic regression splines (for the basis, which is **bs**)

we're forcing education to be linear

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## prestige ~ education + s(income, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -42.6991     3.7355  -11.43   <2e-16 ***
## education     3.9764     0.3415   11.64   <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(income) 3.353  4.012 15.35  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.825   Deviance explained = 83.3%
## GCV = 54.563  Scale est. = 51.7       n = 102
```

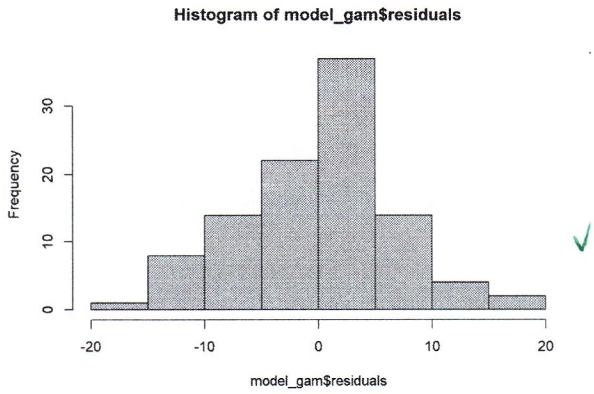
```
# What s() is doing is building a "smooth" term for each of the covariates I am putting in.
# The behaviour is very similar to what you've seen last time: no need to set the number of knots
# (they default to the unique values of the covariate), nor (like in the gam package)
# the equivalent degrees of freedom: mgcv will take care of everything for you. The fitting
# is based on penalised likelihood (which, in a gaussian setting, is equivalent to OLS),
# where the term to be penalised is of course the second derivatives of the smooths
summary(model_gam)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## prestige ~ s(education, bs = "cr") + s(income, bs = "cr")
##
## Parametric coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.277e-15  6.884e-01       0        1
##
## Approximate significance of smooth terms:
##                edf Ref.df     F p-value
## s(education) 3.154  3.913 39.22  <2e-16 ***
## s(income)    2.996  3.604 16.68  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.837   Deviance explained = 84.7%
## GCV = 51.984  Scale est. = 48.34      n = 102
```

```
# Let's look at the residuals
hist(model_gam$residuals)
```

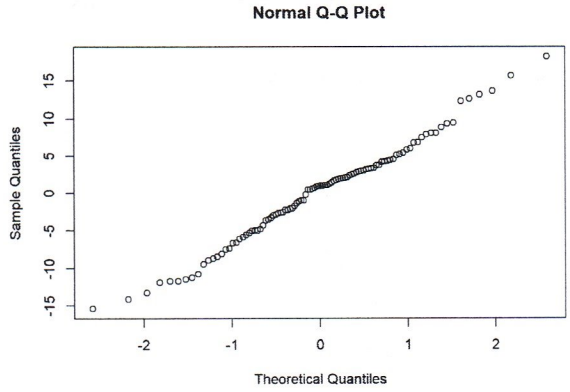
if the normality assumptions are **not** met it's better to do something else (to improve)

(The normality assumptions of the residuals are very important for **GAM**s)



✓

```
qqnorm(model_gam$residuals)
```



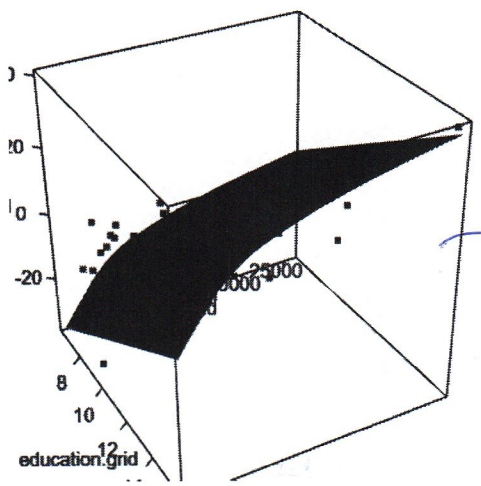
✓

```
# Normality test
shapiro.test(model_gam$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model_gam$residuals
## W = 0.98887, p-value = 0.5603
```

✓ ⟹ The inference automatically performed in the summary is quite reliable

```
pred = predict(model_gam1,newdata=grid)
# Adjusted R squared is better, without the interaction term.

persp3d(education.grid,income.grid,pred,col='grey30')
points3d(education,income,prestige,col='black',size=5)
rglwidget()
```



we've forcing "education" to be linear since in "gam1" we did not put S(·)

1.

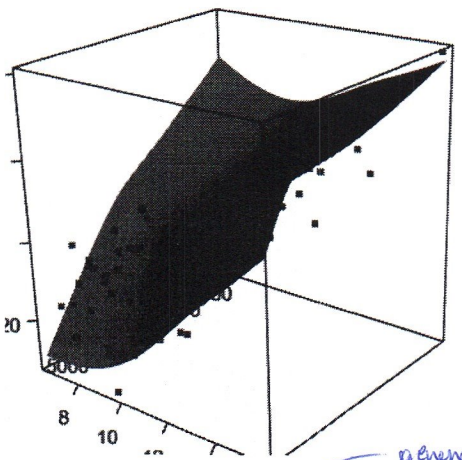
```
# How to put an interaction term?
# Either I add a smooth:
inter      = income*education
model_gam = gam(prestige ~ s(education,bs='cr') + s(income,bs='cr') + s(I(inter),bs='cr'))
summary(model_gam)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## prestige ~ s(education, bs = "cr") + s(income, bs = "cr") + s(I(inter),
##     bs = "cr")
##
## Parametric coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.277e-15  6.717e-01       0        1
##
## Approximate significance of smooth terms:
##               edf Ref.df     F  p-value
## s(education) 3.054  3.842 5.637 0.000756 ***
## s(income)    1.000  1.000 1.931 0.167938
## s(I(inter))  4.028  4.735 5.254 0.000369 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.845   Deviance explained = 85.7%
## GCV = 50.513  Scale est. = 46.015     n = 102
```

EQUIVALENT DEGREES OF FREEDOM
= degrees of freedom of the smooth term

The income becomes linear because of the presence of interaction (degree of freedom = 1)

```
pred = predict(model_gam,newdata=data.frame(grid,inter=grid$education*grid$income))
persp3d(education.grid,income.grid,pred,col='grey30')
points3d(education,income,prestige,col='black',size=5)
rglwidget()
```



2.

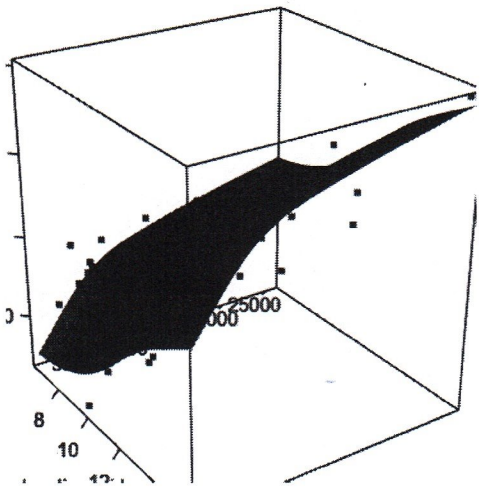
generalization of the concept of "smooth regression splines" in the 2 dimensional settings

```
# or, I use a thin plate spline (Wood, 2003)
model_gam = gam(prestige ~ s(education,bs='cr') + s(income,bs='cr') + s(education,income,bs='tp'))
summary(model_gam)
```

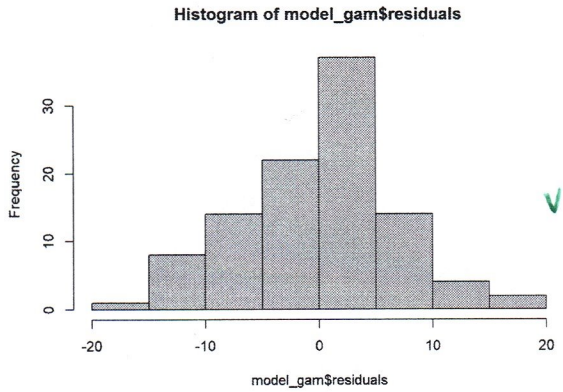
thin plate spline

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## prestige ~ s(education, bs = "cr") + s(income, bs = "cr") + s(education,
##     income, bs = "tp")
##
## Parametric coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.277e-15  6.884e-01       0        1
##
## Approximate significance of smooth terms:
##                            edf Ref.df     F p-value
## s(education)         3.154e+00  3.913 39.22  <2e-16 ***
## s(income)            2.996e+00  3.604 16.68  <2e-16 ***
## s(education,income) 4.240e-10 27.000  0.00   0.206
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.837   Deviance explained = 84.7%
## GCV = 51.984  Scale est. = 48.34      n = 102
```

```
pred = predict(model_gam,newdata=data.frame(grid,inter=grid$education*grid$income))
persp3d(education.grid,income.grid,pred,col='grey30')
points3d(education,income,prestige,col='black',size=5)
rglwidget()
```




```
hist(model_gam$residuals)
```



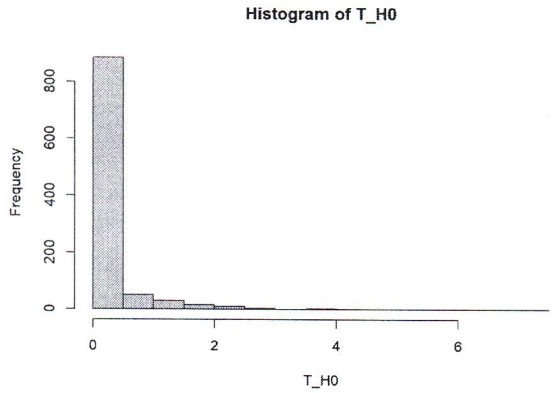

```
shapiro.test(model_gam$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model_gam$residuals
## W = 0.98887, p-value = 0.5603
```

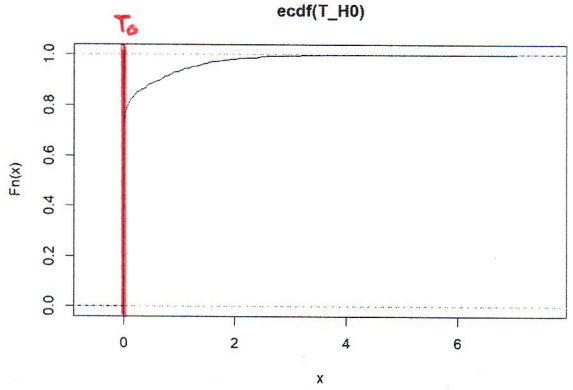
We can also trust the inference of the summary

```
# What if my residuals were not normal? PERMUTATION TESTS!
# Let's try to use the test statistic computed by the summary (Nychka 1988, Wood 2013).
# You can try by using the L^2 norm of the coefficient
T0          = abs(summary(model_gam)$s.table[3,3])
gam.H0      = gam(prestige ~ s(education,bs='cr') + s(income,bs='cr'))
residui.H0 = gam.H0$residuals
B           = 1000
n           = nrow(Prestige)

# To simulate the distribution under H0 of my test statistic, let's try another computational approach.
set.seed(27081991)
library(pbapply)

wrapper = function(){
  permutazione     = sample(nrow(Prestige))
  residui.H0.perm = residui.H0[permutazione]
  Y.perm.H0        = gam.H0$fitted + residui.H0.perm
  gam.perm         = gam(Y.perm.H0 ~ s(education,bs='cr')+s(income,bs='cr')+s(education,income,bs='tp'))
  return(abs(summary(gam.perm)$s.table[3,3]))
}

T_H0 = pbreplicate(B,wrapper(),simplify = 'vector')
hist(T_H0)
```

(alternative) PERMUTATION TEST




```
# I am using the fancy, pb version: you can of course use the vanilla, built in one.
# And... you can try to use it in a parallel fashion too!
plot(ecdf(T_H0))
abline(v=T0)
```




```
sum(T_H0>=T0)/B
```

```
## [1] 0.59
```

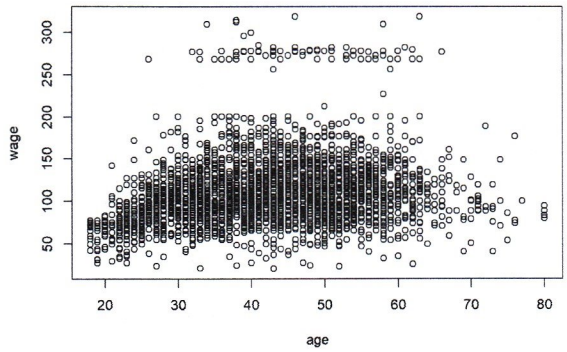
⟹ H_0: the interaction is not significant (agreement with the gam model summary)

valid if the residuals are normal

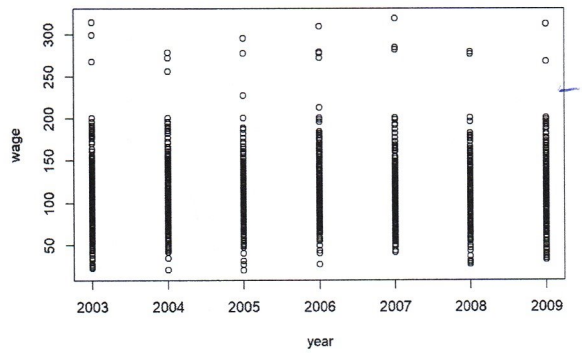
```
# How to generate predictions in this (normal) context for a new point?
newdata = data.frame(income=15000,education=15)
pred    = predict(model_gam,newdata=newdata,type='response',se.fit = T)

# Confidence intervals
alpha = 0.05
lwr   = pred$fit-pred$se.fit*qt(1-(alpha/2),nrow(Prestige))
lvl   = pred$fit
upr   = pred$fit+pred$se.fit*qt(1-(alpha/2),nrow(Prestige))
cbind(lwr,lvl,upr)
```

```
##        lwr      lvl      upr
## 1 23.66254 28.26201 32.86148
```

```
### ------------------------------------------------------------------------
### Let's play around with our other dataset
### ------------------------------------------------------------------------
detach(Prestige)
attach(Wage)

# We know that the wage strongly depends on the year, and on the age... Let's see.
plot(age,wage)
```




```
plot(year,wage)
```



← slightly linear term (even if there is a lot of uncertainty in each time)

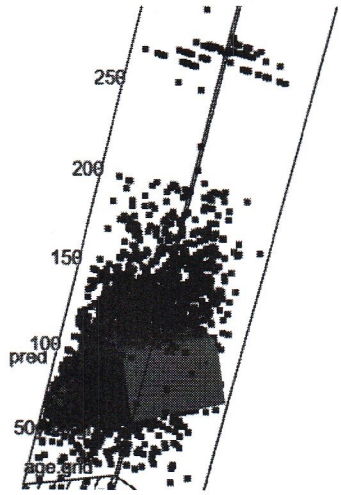
```
# If year is meaningful, it's probably linear, what happens if I add a smooth for it?
model_gam = gam(wage ~ s(age,bs='cr') + s(year,bs='cr',k=7))
summary(model_gam)
```

↳ we're deciding #knots

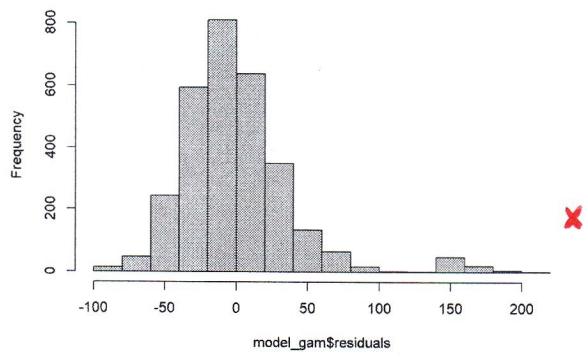
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## wage ~ s(age, bs = "cr") + s(year, bs = "cr", k = 7)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 111.7036     0.7266   153.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##           edf Ref.df     F  p-value
## s(age)  5.411   6.53 43.73  < 2e-16 ***
## s(year) 1.000   1.00 14.16 0.000172 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0904   Deviance explained = 9.24%
## GCV = 1587.7  Scale est. = 1583.8     n = 3000
```

as we said: **linear** (1 degree of freedom because the intercept is estimated in a standard (parametric) way)

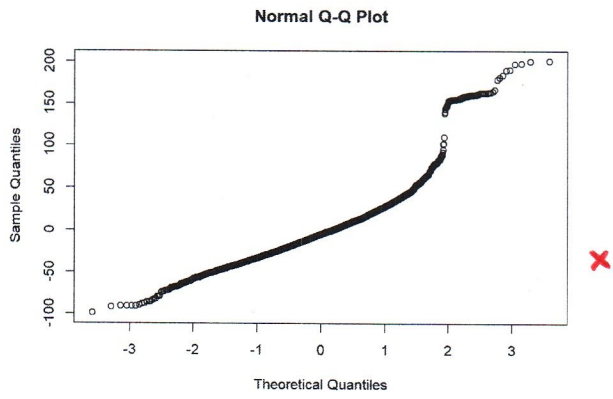
```
# Look at the degrees of freedom of the smooth term for year
age.grid     = seq(range(age)[1],range(age)[2],by=1)
year.grid    = seq(range(year)[1],range(year)[2],by=1)
grid         = expand.grid(age.grid,year.grid)
names(grid) = c('age','year')
pred         = predict(model_gam,newdata = grid)
persp3d(age.grid,year.grid,pred,col='blue')
points3d(age,year,wage,size=5)
rglwidget()
```




```
# Let's do it on probability of being high wage
model_gam_logit = gam(I(wage>250) ~ s(age,bs='cr') + s(year,bs='cr',k=7),family='binomial')
preds           = predict(model_gam_logit,newdata = grid)
pfit            = exp(preds )/(1+ exp( preds ))
persp3d(age.grid,year.grid,pfit,col='blue')
hist(model_gam$residuals)
```




```
qqnorm(model_gam$residuals)
```

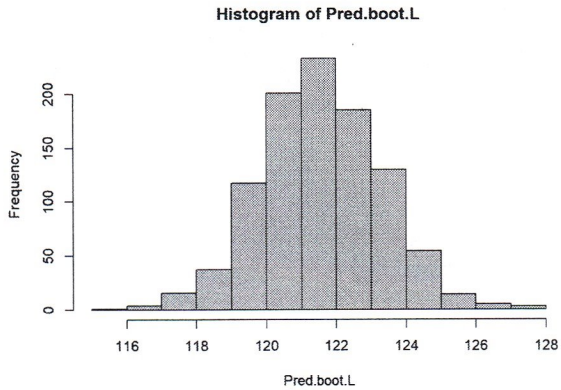



```
shapiro.test(model_gam$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model_gam$residuals
## W = 0.87536, p-value < 2.2e-16
```

Bootstrap **prediction** intervals: (when the residuals are **not** normal)

```
# How to generate predictions in this context?
# Let's generate predictions for the wage as a function of age and the year:
set.seed(27081991)
B          = 1000
newdata    = data.frame(age=40,year=2008)
fitted.obs = model_gam$fitted.values
res.obs    = model_gam$residuals
pred.obs   = predict(model_gam,newdata = newdata)

wrapper_boot = function(){
  response.b  = fitted.obs + sample(res.obs, replace = T)
  model_gam_b = gam(response.b ~ s(age,bs='cr') + s(year,bs='cr',k=7))
  pred.boot   = predict(model_gam_b,newdata=newdata)
}

Pred.boot.L = pbreplicate(B, wrapper_boot(),simplify = 'vector')
hist(Pred.boot.L)
```




```
# Reverse Percentile Interval of predicted value (prediction intervals (DH, chapter 6))
alpha             = 0.05
right.quantile.L = quantile(Pred.boot.L, 1 - alpha/2)
left.quantile.L  = quantile(Pred.boot.L, alpha/2)
right.quantile.L - pred.obs
```

```
##        1
## 2.724482
```

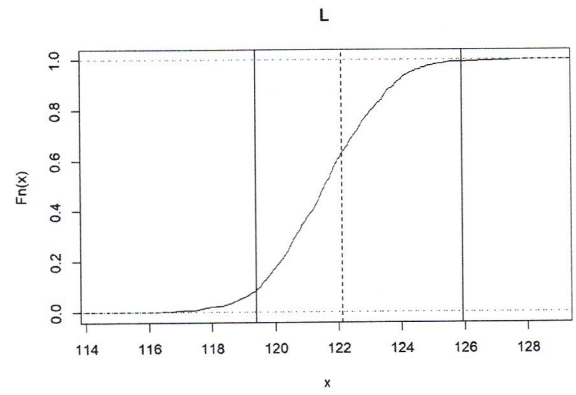
```
left.quantile.L  - pred.obs
```

```
##         1
## -3.816993
```

```
CI.RP.L = c(pred.obs - (right.quantile.L - pred.obs), pred.obs - (left.quantile.L - pred.obs))
CI.RP.L
```

```
##        1        1
## 119.3934 125.9349
```

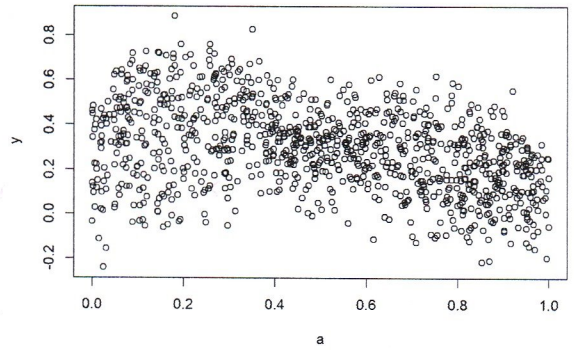
```
plot(ecdf(Pred.boot.L), main='L')
abline(v = pred.obs, lty=2)
abline(v = CI.RP.L)
```



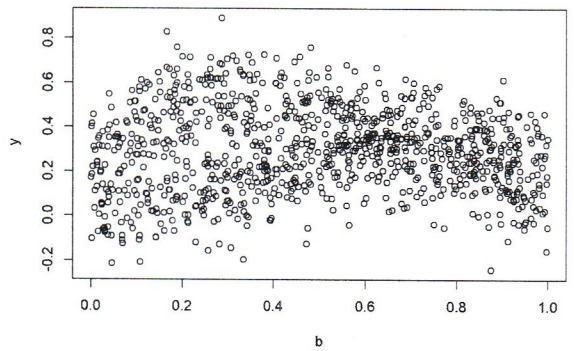

```
predict(model_gam,newdata = newdata)
```

```
##         1
## 122.1179
```

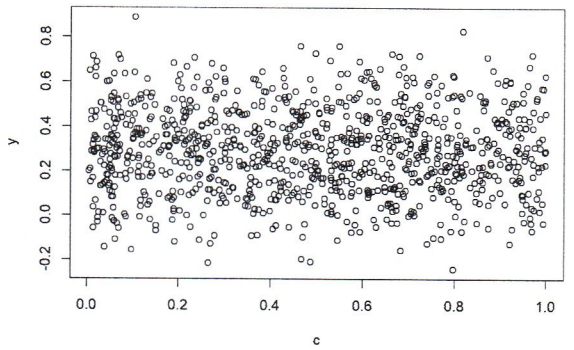
```
### ------------------------------------------------------------------------
### Exercise!
### ------------------------------------------------------------------------
# In gam_ex.rda there is a dataframe: y is the response variable, a, b, c, are the covariates.
# 1. Build a gam model complete up to second order interactions using cubic regression splines
#    also for the interaction terms
# 2. Build a reduced model performing model selection (choose a parametric or nonparametric
#    approach according to your situation)
# 3. Create an approximate confidence interval for a new point where a=3, b=4, c=10.
# 4. How you should've solved point 2. and 3. if the answer you got from your data was the opposite one?

# Solution!
load('gam_ex.rda')
attach(df)
# Let's inspect the data
plot(a,y)
```




```
plot(b,y)
```




```
plot(c,y)
```



1.

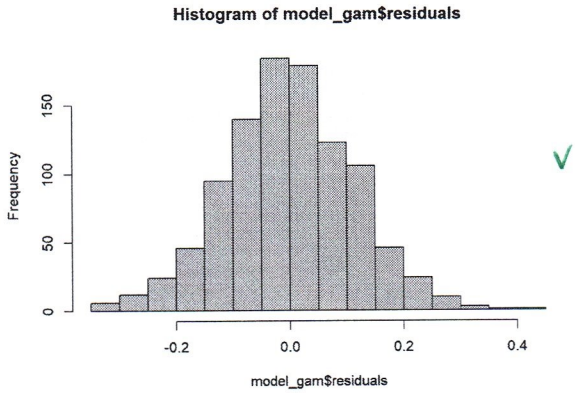
```
# Let's build a full model, then:
# we follow the (easier) manual interaction approach.
inter1 = a*b
inter2 = a*c
inter3 = b*c

model_gam = gam(y ~ s(a,bs='cr') + s(b,bs='cr') + s(c,bs='cr') +
                  s(inter1,bs='cr') + s(inter2,bs='cr') + s(inter3,bs='cr'))
summary(model_gam)
```

Since there are requested the cubic splines we cannot use the thin plate spline → add interactions by hand

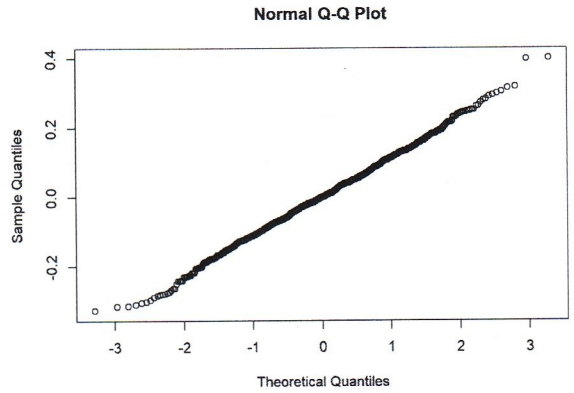
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(a, bs = "cr") + s(b, bs = "cr") + s(c, bs = "cr") + s(inter1,
##     bs = "cr") + s(inter2, bs = "cr") + s(inter3, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.293784   0.003621   81.13   <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(a)      8.249  8.839  89.669  <2e-16 ***
## s(b)      5.115  6.237 101.029  <2e-16 ***
## s(c)      1.000  1.000   1.849   0.174
## s(inter1) 5.756  6.608 157.094  <2e-16 ***
## s(inter2) 1.000  1.000   1.878   0.171
## s(inter3) 1.000  1.000   0.265   0.607
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.644   Deviance explained = 65.2%
## GCV = 0.013423  Scale est. = 0.013112  n = 1000
```

(Before we start deleting things)
The summary is based on the normality assumptions of the residuals. Check them first.
(If they're not met → permutations)

```
hist(model_gam$residuals)
```




```
qqnorm(model_gam$residuals)
```




```
# Normality test
shapiro.test(model_gam$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model_gam$residuals
## W = 0.99794, p-value = 0.2596
```

✓ → no need for permutational approach

2.

```
# Let's reduce the model, then
model_gam = gam(y ~ s(a,bs='cr') + s(b,bs='cr') + s(c,bs='cr') + s(inter1,bs='cr') + s(inter2,bs='cr'))
summary(model_gam)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(a, bs = "cr") + s(b, bs = "cr") + s(c, bs = "cr") + s(inter1,
##     bs = "cr") + s(inter2, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.29378    0.00362   81.16   <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(a)      8.252  8.840  89.699  <2e-16 ***
## s(b)      5.099  6.218 128.333  <2e-16 ***
## s(c)      1.000  1.000   1.867   0.172      ✗
## s(inter1) 5.747  6.599 157.664  <2e-16 ***
## s(inter2) 1.000  1.000   1.871   0.172      ✗
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.644   Deviance explained = 65.1%
## GCV = 0.0134  Scale est. = 0.013104  n = 1000
```

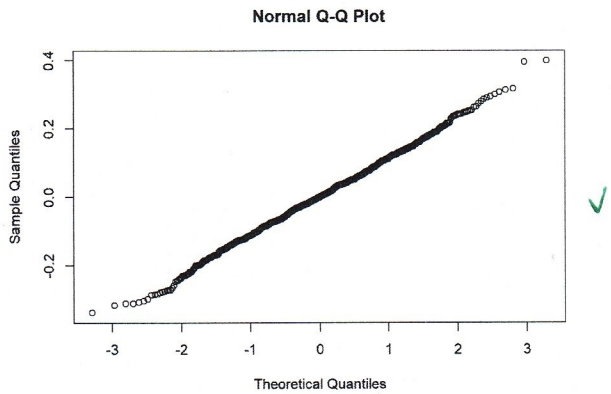
```
# slso inter2
model_gam = gam(y ~ s(a,bs='cr') + s(b,bs='cr') + s(c,bs='cr') + s(inter1,bs='cr'))
summary(model_gam)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(a, bs = "cr") + s(b, bs = "cr") + s(c, bs = "cr") + s(inter1,
##     bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.293784   0.003622   81.11   <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(a)      8.248  8.839 129.838  <2e-16 ***
## s(b)      5.044  6.159 129.744  <2e-16 ***
## s(c)      1.000  1.000   0.134   0.714      ✗
## s(inter1) 5.778  6.629 157.759  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.644   Deviance explained = 65.1%
## GCV = 0.0134  Scale est. = 0.013118  n = 1000
```

```
# let's drop also c
model_gam = gam(y ~ s(a,bs='cr') + s(b,bs='cr') + s(inter1,bs='cr'))
summary(model_gam)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(a, bs = "cr") + s(b, bs = "cr") + s(inter1, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.29378    0.00362   81.15   <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(a)      8.245  8.838 130.0  <2e-16 ***
## s(b)      5.032  6.144 130.2  <2e-16 ***
## s(inter1) 5.790  6.641 157.6  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.644   Deviance explained = 65.1%
## GCV = 0.013375  Scale est. = 0.013107  n = 1000
```

```
# good!
qqnorm(model_gam$residuals)
```




```
shapiro.test(model_gam$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model_gam$residuals
## W = 0.99808, p-value = 0.3178
```

3.

```
# Let's generate confidence interval for a new point
a_n      = 3
b_n      = 4
inter1_n = a_n*b_n
newdata  = data.frame(a=a_n,b=b_n,inter1=inter1_n)
pred     = predict(model_gam,newdata=newdata,type='response',se.fit = T)

# Confidence intervals
alpha = 0.05
lwr   = pred$fit-pred$se.fit*qt(1-(alpha/2),nrow(df))
lvl   = pred$fit
upr   = pred$fit+pred$se.fit*qt(1-(alpha/2),nrow(df))
cbind(lwr,lvl,upr)
```

```
##        lwr        lvl      upr
## 1 -4.41348 -0.3559127 3.701654
```

4. Normality of residuals is not met ⇒ permutation testing for the model selection and either bootstrap or permutation for the last part