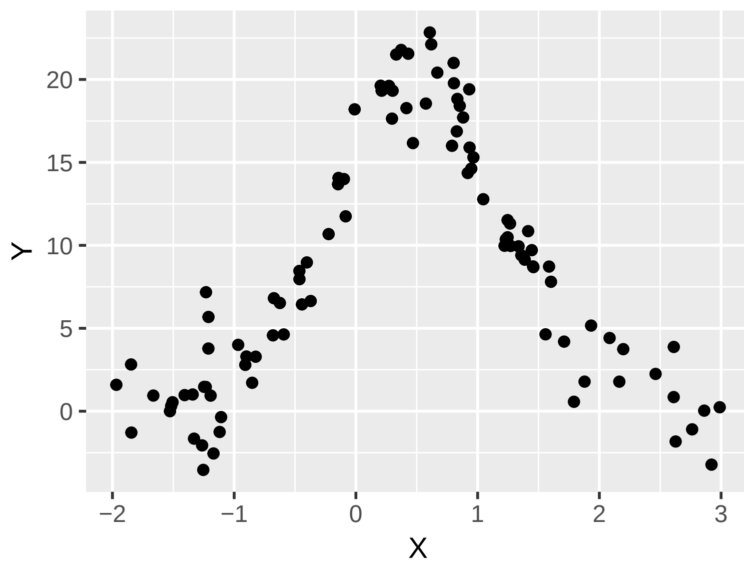
STATS 415 hw8 solution

Weijing Tang

Mar 20, 2018

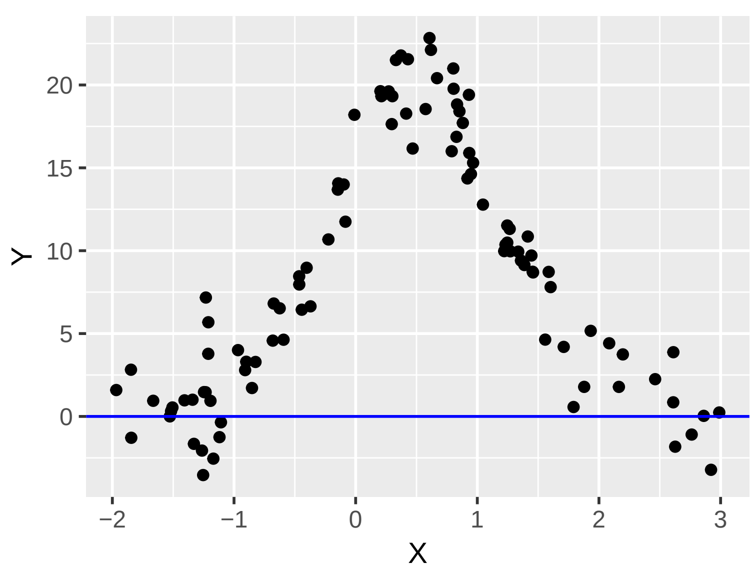
Q1: Provide an example sketch of $\hat{g}$ in the following scenarios:

Given the true function $f(x) = 20e^{-(x-0.5)^2}$ on [-2,3], we generate 100 observations.



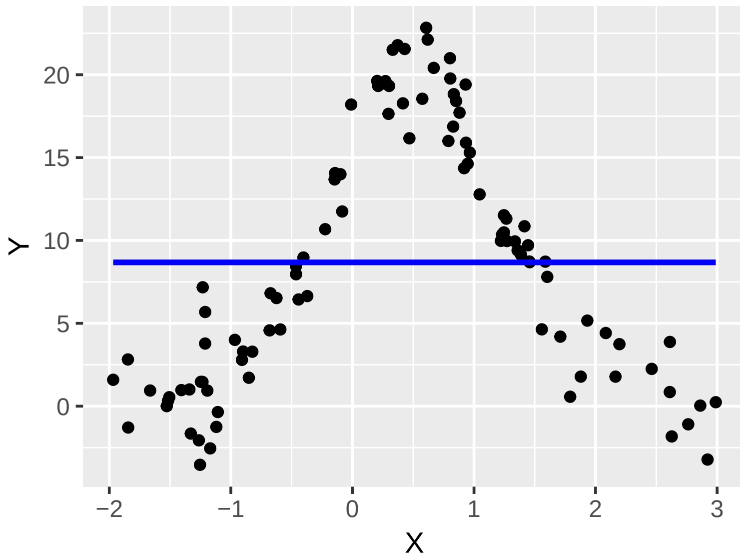
(a)(2 pts) $\lambda = \infty$, $m = 0$.

$\hat{g} = 0$.



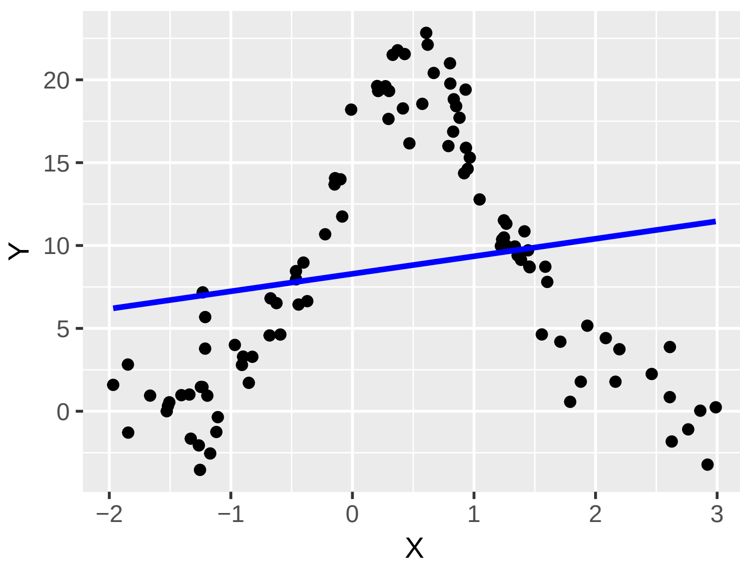
(b)(2 pts) $\lambda = \infty$, $m = 1$.

$\hat{g}$ is a constant which minimizes RSS.



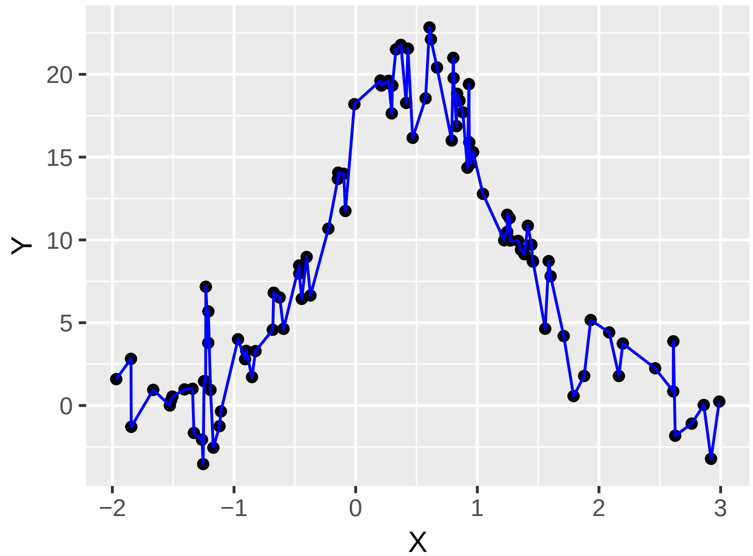
(c)(2 pts) $\lambda = \infty$, $m = 3$.

$\hat{g}$ is a line which minimizes RSS.



(d)(2 pts) $\lambda = 0$, $m = 3$.

$\hat{g}$ is any function which satisfies $g(x_i) = y_i$ for $i = 1, \cdots, n$.



Q2: Predicting the air quality variable `nox`.

(a)(1 pt)

```
library(MASS)
data(Boston)

# split training and test data
set.seed(34567)
train = sample(1:nrow(Boston),trunc(nrow(Boston)*0.8))
```

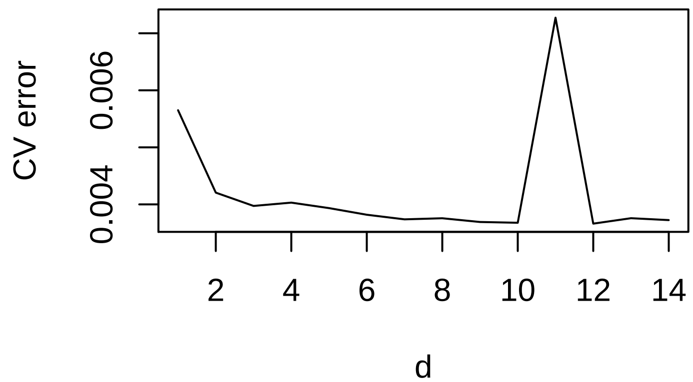
(b)(4 pts)

- For polynomial regression,

```
library(boot)
set.seed(1)
cv.error_poly = rep(0,14)
for (i in 1:14){
  fit=glm(nox~poly(dis,i),data=Boston[train,])
  cv.error_poly[i]=cv.glm(Boston[train,], fit, K=10)$delta[1]
}
which.min(cv.error_poly)
```

```
## [1] 12
```

```
plot(1:14,cv.error_poly,xlab = "d",ylab = "CV error",type = "l")
```



When d = 12, it minimizes CV error.

```
fit.poly = glm(nox~poly(dis,12),data=Boston[train,])
summary(fit.poly)
```

```
##
## Call:
## glm(formula = nox ~ poly(dis, 12), data = Boston[train, ])
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -0.131337  -0.038574  -0.009979   0.031460   0.204793
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.553415   0.003021 183.193  < 2e-16 ***
## poly(dis, 12)1  -1.770681   0.060720 -29.161  < 2e-16 ***
## poly(dis, 12)2   0.770450   0.060720  12.689  < 2e-16 ***
## poly(dis, 12)3  -0.294733   0.060720  -4.854 1.75e-06 ***
## poly(dis, 12)4   0.035747   0.060720   0.589 0.556390
## poly(dis, 12)5   0.140129   0.060720   2.308 0.021533 *
## poly(dis, 12)6  -0.212277   0.060720  -3.496 0.000526 ***
## poly(dis, 12)7   0.206528   0.060720   3.401 0.000740 ***
## poly(dis, 12)8  -0.101131   0.060720  -1.666 0.096610 .
## poly(dis, 12)9   0.031644   0.060720   0.521 0.602565
## poly(dis, 12)10  0.009847   0.060720   0.162 0.871252
## poly(dis, 12)11 -0.018736   0.060720  -0.309 0.757818
## poly(dis, 12)12 -0.106375   0.060720  -1.752 0.080577 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.003686947)
##
##     Null deviance: 5.3890  on 403  degrees of freedom
## Residual deviance: 1.4416  on 391  degrees of freedom
## AIC: -1102.3
```

```
##
## Number of Fisher Scoring iterations: 2
```

```
mean((fit.poly$residuals)^2)
```

```
## [1] 0.003568308
```

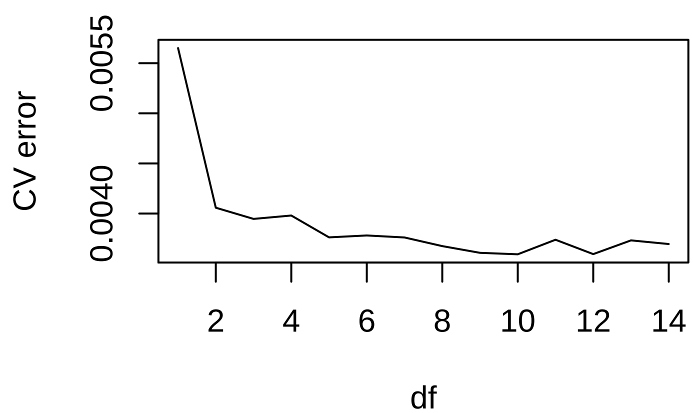
We can see from the regression output that the terms with degree higher than 7 are not statistically significant even though we choose d=12 based on cross-validation. The CV plot also support this since the CV errors are pretty similar or even higher after degree 7. The training MSE of polynomial regression is 0.003568308.(cv error is 0.003662715.)

- For natural spline,

```
library(splines)
set.seed(1)
cv.error_ns = rep(0,14)
for (i in 1:14){
  fit=glm(nox~ns(dis,df = i),data=Boston[train,])
  cv.error_ns[i]=cv.glm(Boston[train,], fit, K=10)$delta[1]
}
which.min(cv.error_ns)
```

```
## [1] 10
```

```
plot(1:14,cv.error_ns,xlab = "df",ylab = "CV error",type = "l")
```



When d = 10, it minimizes CV error.

```
fit.ns = glm(nox~ns(dis,df = 10),data=Boston[train,])
summary(fit.ns)
```

```
##
## Call:
## glm(formula = nox ~ ns(dis, df = 10), data = Boston[train, ])
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.13598  -0.03689  -0.01173   0.02889   0.19522
```

```
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          0.6353882  0.0246540  25.772  < 2e-16 ***
## ns(dis, df = 10)1  -0.0043391  0.0258654  -0.168 0.866860
## ns(dis, df = 10)2  -0.0003568  0.0327463  -0.011 0.991313
## ns(dis, df = 10)3  -0.0567744  0.0310123  -1.831 0.067902 .
## ns(dis, df = 10)4  -0.1571810  0.0332475  -4.728 3.17e-06 ***
## ns(dis, df = 10)5  -0.1083910  0.0313034  -3.463 0.000594 ***
## ns(dis, df = 10)6  -0.1565244  0.0321393  -4.870 1.62e-06 ***
## ns(dis, df = 10)7  -0.1867140  0.0296636  -6.294 8.25e-10 ***
## ns(dis, df = 10)8  -0.2687669  0.0251047 -10.706  < 2e-16 ***
## ns(dis, df = 10)9  -0.0912977  0.0598371  -1.526 0.127871
## ns(dis, df = 10)10 -0.3026695  0.0292817 -10.336  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.003571911)
##
##     Null deviance: 5.3890  on 403  degrees of freedom
## Residual deviance: 1.4038  on 393  degrees of freedom
## AIC: -1117.1
##
## Number of Fisher Scoring iterations: 2
```

```
mean((fit.ns$residuals)^2)
```

```
## [1] 0.003474656
```

7 out of 11 basis functions within natural spline model are statistically significant. The training MSE of natural spline is 0.003474656.(cv error is 0.003593527.)

- For smoothing spline,

```
set.seed(1)
fit.ss=smooth.spline(Boston[train,"dis"],Boston[train,"nox"],cv=TRUE)
fitvalue = predict(fit.ss,Boston[train,"dis"])$y
fit.ss
```

```
## Call:
## smooth.spline(x = Boston[train, "dis"], y = Boston[train, "nox"],
##     cv = TRUE)
##
## Smoothing Parameter  spar= 0.8273621  lambda= 7.566141e-05 (11 iterations)
## Equivalent Degrees of Freedom (Df): 15.98821
## Penalized Criterion: 1.396404
## PRESS: 0.003646622
```

```
cat("training error:",mean((Boston[train,"nox"]-fitvalue)^2),"\n")
```

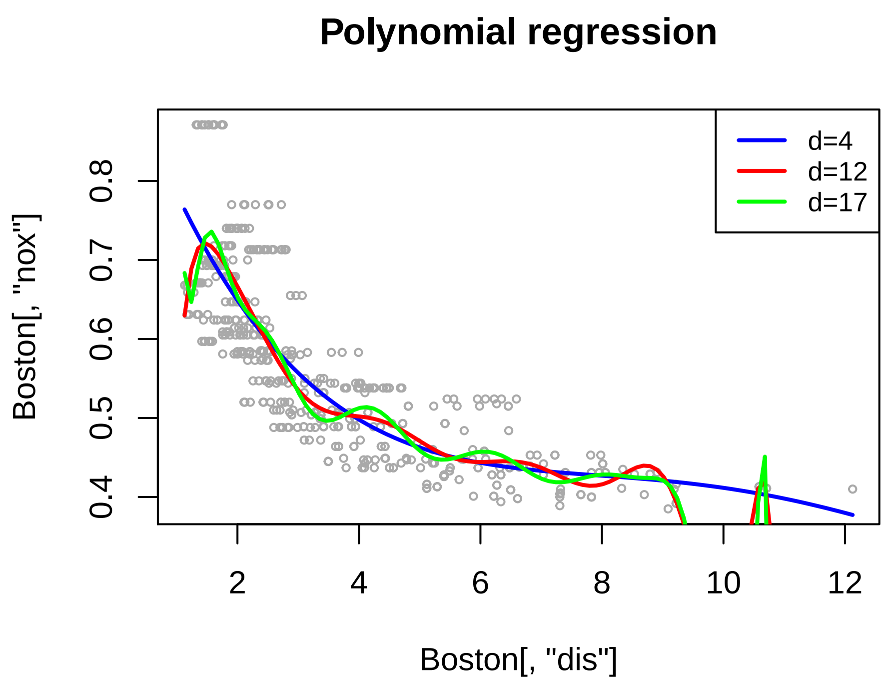
```
## training error: 0.003459933
```

```
cat("cv error:",fit.ss$cv.crit)
```

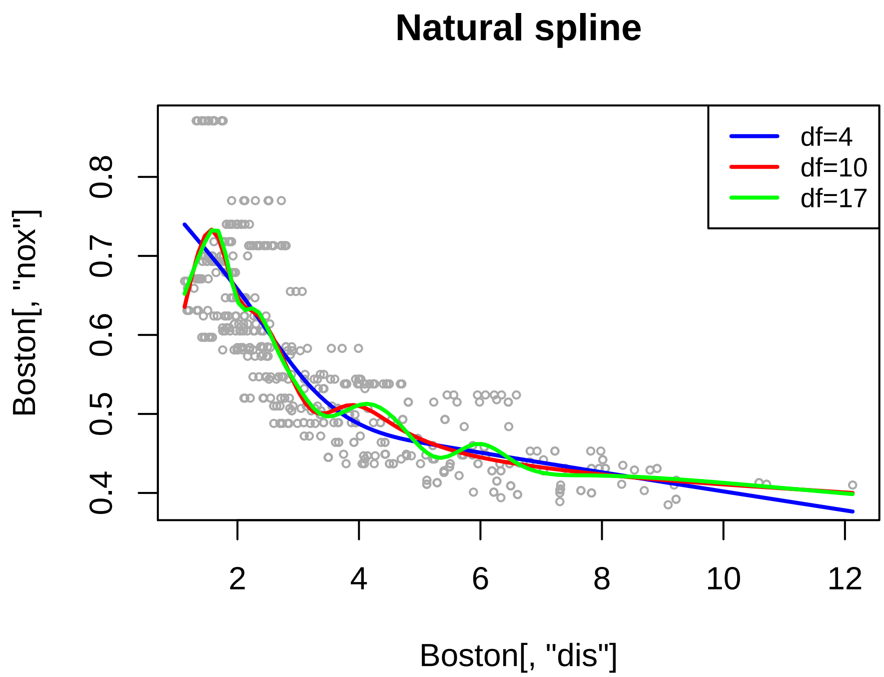
```
## cv error: 0.003646622
```

The optimized Df tuning parameter is around 16 which corresponds to $\lambda = 7.566141e - 05$. The training MSE of smoothing spline is 0.003459933.(cv error is 0.003646622) (c)(4 pts)

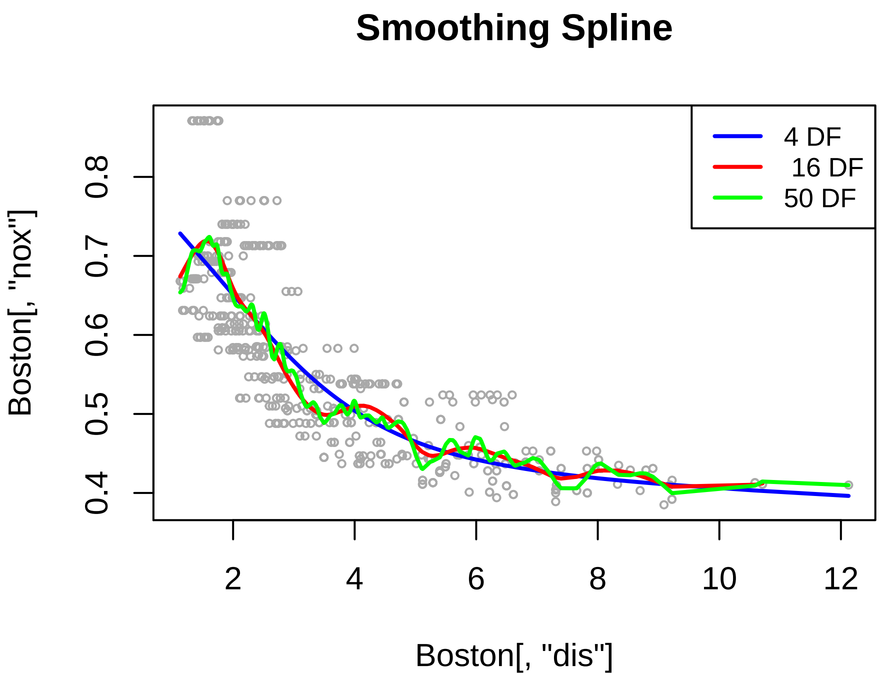
```
dislims=range(Boston[,"dis"])
dis.grid=seq(from=dislims[1],to=dislims[2],length.out = 100)
fit.poly17=lm(nox~poly(dis,17),data=Boston[train,])
fit.poly4 = lm(nox~poly(dis,4),data=Boston[train,])
preds1=predict(fit.poly4,newdata=data.frame(dis=dis.grid))
preds2=predict(fit.poly,newdata=data.frame(dis=dis.grid))
preds3=predict(fit.poly17,newdata=data.frame(dis=dis.grid))
plot(Boston[,"dis"],Boston[,"nox"],xlim=dislims,cex=.5,col="darkgrey")
title("Polynomial regression")
lines(dis.grid,preds1,lwd=2,col="blue")
lines(dis.grid,preds2,lwd=2,col="red")
lines(dis.grid,preds3,lwd=2,col="green")
legend("topright",legend=c("d=4","d=12","d=17"),col=c("blue","red","green"),lty=1,lwd=2,cex=.8)
```




```
fit.ns17=lm(nox~ns(dis,17),data=Boston[train,])
fit.ns4 = lm(nox~ns(dis,4),data=Boston[train,])
preds1=predict(fit.ns4,newdata=data.frame(dis=dis.grid))
preds2=predict(fit.ns,newdata=data.frame(dis=dis.grid))
preds3=predict(fit.ns17,newdata=data.frame(dis=dis.grid))
plot(Boston[,"dis"],Boston[,"nox"],xlim=dislims,cex=.5,col="darkgrey")
title("Natural spline")
lines(dis.grid,preds1,lwd=2,col="blue")
lines(dis.grid,preds2,lwd=2,col="red")
lines(dis.grid,preds3,lwd=2,col="green")
legend("topright",legend=c("df=4","df=10","df=17"),col=c("blue","red","green"),lty=1,lwd=2,cex=.8)
```




```
fit.50=smooth.spline(Boston[,"dis"],Boston[,"nox"],df=50)
fit.4=smooth.spline(Boston[,"dis"],Boston[,"nox"],df=4)
plot(Boston[,"dis"],Boston[,"nox"],xlim=dislims,cex=.5,col="darkgrey")
title("Smoothing Spline")
lines(fit.4,col="blue",lwd=2)
lines(fit.ss,col="red",lwd=2)
lines(fit.50,col="green",lwd=2)
legend("topright",legend=c("4 DF"," 16 DF","50 DF"),col=c("blue","red","green"),lty=1,lwd=2,cex=.8)
```



For each plot, the higher degree of freedom is, the less smoothness there is. Compared with natural spline and smoothing spline, we can see a huge jump at the boundary for polynomial regression when degree is large.

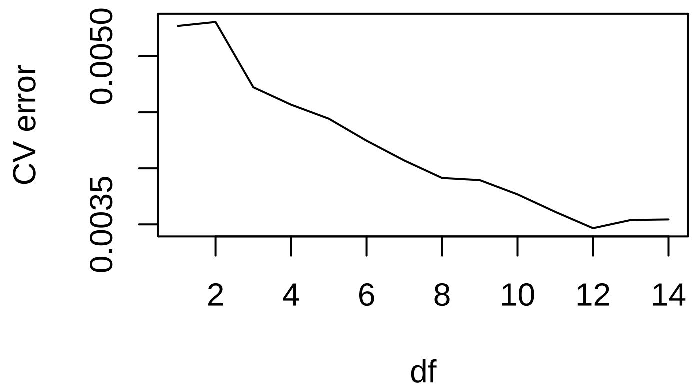
(d)(4 pts)

We choose natural spline to nonlinearly model the relationship between `nox` and `dis` based on cv errors. We also use natural spline to model `indus`. Here we use cross validation again to choose the degree of freedom.

```
set.seed(1)
cv.error_ns = rep(0,14)
for (i in 1:14){
  fit=glm(nox~ns(indus,df = i),data=Boston[train,])
  cv.error_ns[i]=cv.glm(Boston[train,], fit, K=10)$delta[1]
}
which.min(cv.error_ns)
```

```
## [1] 12
```

```
plot(1:14,cv.error_ns,xlab = "df",ylab = "CV error",type = "l")
```

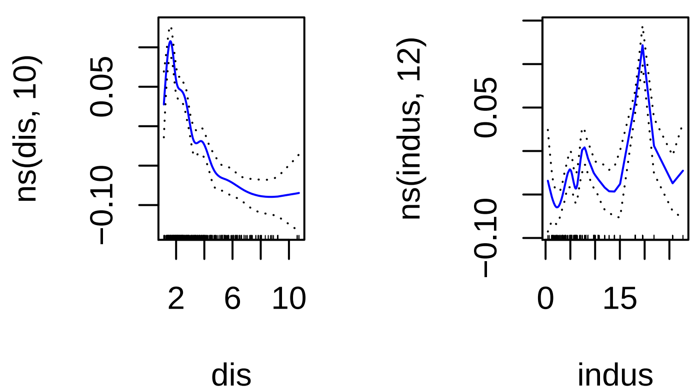


The degree of freedom that we choose is 12.

```
library(gam)
```

```
## Warning: package 'gam' was built under R version 3.3.3
## Loading required package: foreach
## Warning: package 'foreach' was built under R version 3.3.3
## Loaded gam 1.14
```

```
fit.gam=gam(nox~ns(dis,10)+ns(indus,12),data=Boston[train,])
par(mfrow=c(1,2))
plot(fit.gam, se=TRUE,col="blue")
```



The GAM shows us that there is a strong nonlinear relationship between distance(indus) and nitrogen oxides. Given `indus` fixed, 'nox' keeps decreasing as `dis` increases after a peak at 2. Compared with `dis`, `indus` has a more unstable relationship with `nox`.

(e)(2 pts)

```
test.poly = predict(fit.poly,Boston[-train,])
test.ns = predict(fit.ns,Boston[-train,])
test.ss = predict(fit.ss,Boston[-train,"dis"])
test.gam = predict(fit.gam,Boston[-train,])
nox.test= Boston[-train,"nox"]
error.poly = mean((nox.test-test.poly)^2)
error.ns = mean((nox.test-test.ns)^2)
error.ss = mean((nox.test-test.ss$y)^2)
error.gam = mean((nox.test-test.gam)^2)
d <- data.frame("TestMSE" = c(error.poly, error.ns, error.ss, error.gam))
rownames(d) <- c("poly regression", "natural spline", "smoothing spline","GAM")
knitr::kable(d)
```

	TestMSE
poly regression	93.2606133
natural spline	0.0039645
smoothing spline	0.0038431
GAM	0.0025572

```
summary(test.poly)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -97.1200   0.4653   0.5211  -0.3955   0.6704   0.7211
```

The reason for bad performance of polynomial regression is due to two test points which are located at the upper bound of `dis`. The extimated value of `nox` -97.12 is much lower than the true value. Based on test MSEs, we prefer GAM here.