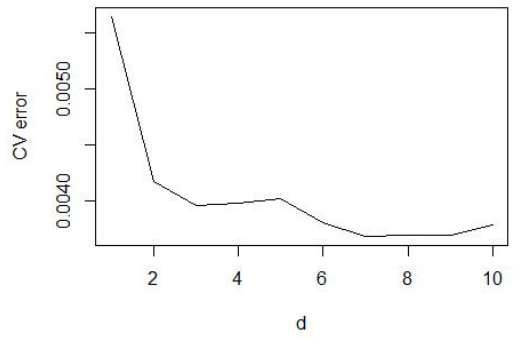
Stat415homework8

(a) Set the random seed to 34567 and randomly split the data into 80% training and 20% test data.The test data is not used until the last question.

```
set.seed(34567)
library(MASS)
train = sample(1:nrow(Boston),trunc(nrow(Boston)*0.8))
```

(b) Fit a smooth nonlinear function on the training data to predict nox from dis (one predictor).

```
# polynomial regression
# use 10-fold crossvalidation by setting K=10 argument
set.seed(34567)
library(boot)
cv.error_poly = rep(0,10)
for (i in 1:10){
  fit=glm(nox~poly(dis,i),data=Boston[train,])
  cv.error_poly[i]=cv.glm(Boston[train,], fit, K=10)$delta[1]
}
plot(1:10,cv.error_poly,xlab = "d",ylab = "CV error",type = "l")
```



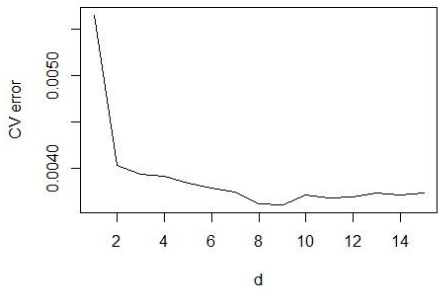

```
which(cv.error_poly==min(cv.error_poly))

## [1] 7

lm.fit = lm(nox~poly(dis, 7), data=Boston[train,])
summary(lm.fit)
## Call:
## lm(formula = nox ~ poly(dis, 7), data = Boston[train, ])
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.553415   0.003026 182.907  < 2e-16 ***
## poly(dis, 7)1 -1.770681   0.060815 -29.116  < 2e-16 ***
## poly(dis, 7)2  0.770450   0.060815  12.669  < 2e-16 ***
## poly(dis, 7)3 -0.294733   0.060815  -4.846 1.81e-06 ***
## poly(dis, 7)4  0.035747   0.060815   0.588 0.557000
```

```
## poly(dis, 7)5  0.140129   0.060815   2.304 0.021729 *
## poly(dis, 7)6 -0.212277   0.060815  -3.491 0.000536 ***
## poly(dis, 7)7  0.206528   0.060815   3.396 0.000753 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06081 on 396 degrees of freedom
## Multiple R-squared:  0.7282, Adjusted R-squared:  0.7234
## F-statistic: 151.6 on 7 and 396 DF,  p-value: < 2.2e-16
```

```r
# natural spline
set.seed(34567)
library(splines)
cv.error_ns = rep(0,15)
for (i in 1:15){
fit=glm(nox~ns(dis,df = i),data=Boston[train,])
cv.error_ns[i]=cv.glm(Boston[train,], fit, K=10)$delta[1]
}
proper_d = which.min(cv.error_ns)
proper_d
```

```
## [1] 9
```

```r
plot(1:15,cv.error_ns,xlab = "d",ylab = "CV error",type = "l")
```




```r
ns.fit = lm(nox~ns(dis,df = proper_d),data=Boston[train,])
summary(ns.fit)
```

```
##
## Call:
## lm(formula = nox ~ ns(dis, df = proper_d), data = Boston[train,
##     ])
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                0.6422470  0.0234222  27.420  < 2e-16 ***
## ns(dis, df = proper_d)1 -0.0337040  0.0242530  -1.390   0.1654
## ns(dis, df = proper_d)2 -0.0001696  0.0310175  -0.005   0.9956
## ns(dis, df = proper_d)3 -0.1308300  0.0293379  -4.459 1.07e-05 ***
## ns(dis, df = proper_d)4 -0.1410184  0.0306684  -4.598 5.75e-06 ***
## ns(dis, df = proper_d)5 -0.1329604  0.0301146  -4.415 1.31e-05 ***
## ns(dis, df = proper_d)6 -0.1998153  0.0283047  -7.059 7.59e-12 ***
## ns(dis, df = proper_d)7 -0.2654380  0.0237953 -11.155  < 2e-16 ***
```

```
## ns(dis, df = proper_d)8 -0.1036093  0.0570493  -1.816   0.0701 .
## ns(dis, df = proper_d)9 -0.3088906  0.0286526 -10.781  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05975 on 394 degrees of freedom
## Multiple R-squared:  0.739,  Adjusted R-squared:  0.733
## F-statistic: 123.9 on 9 and 394 DF,  p-value: < 2.2e-16
```

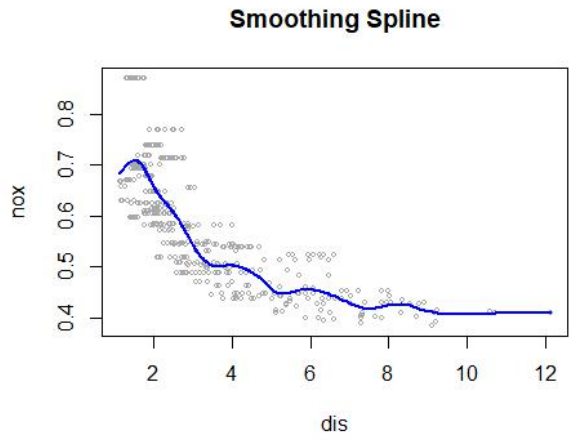
```
# smoothing spline
attach(Boston)
fit.ss=smooth.spline(dis,nox,cv=TRUE)
```

```
## Warning in smooth.spline(dis, nox, cv = TRUE): cross-validation with
 non-
## unique 'x' values seems doubtful
```

```
fit.ss
```

```
## Call:
## smooth.spline(x = dis, y = nox, cv = TRUE)
##
## Smoothing Parameter  spar= 0.8684939  lambda= 9.029534e-05 (11 itera
tions)
## Equivalent Degrees of Freedom (Df): 15.42984
## Penalized Criterion (RSS): 1.785015
## PRESS(l.o.o. CV): 0.003676223
```

```
dislims=range(dis)
plot(dis,nox,xlim=dislims,cex=.5,col="darkgrey")
title("Smoothing Spline")
lines(fit.ss,col="blue",lwd=2)
```

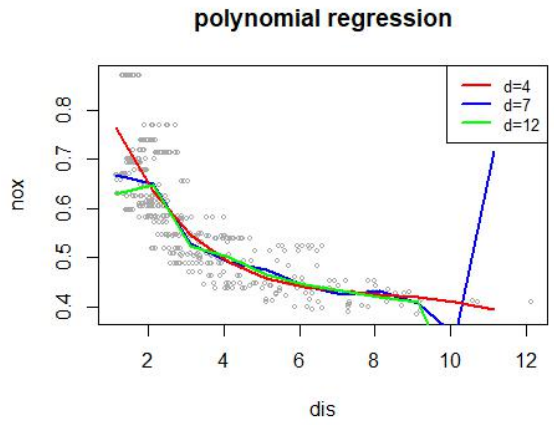


Comment: for polynomial regression, df=7 is suitable for the model since the CV error at df=7 is small and the model is simpler relatively.For natural splines, df=9 is suitable for the model since the CV error at df=9 is the smallest. For smoothing splines, the proper df should be 15.42984 and corresponding lambda is 9.029534e-

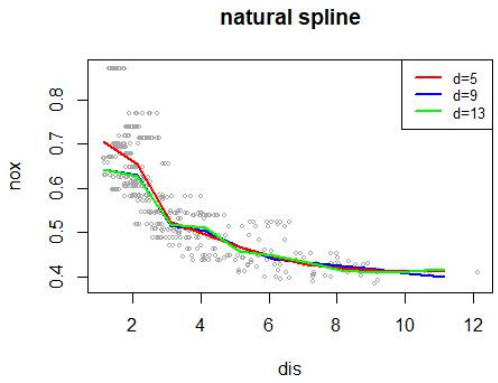
05. According to the regression output, the Adjusted R-squared of natural splines is a little better than the one of polynomial regression.

(c) For each of the methods in the previous question, make three plots: the fitted curve with your optimally selected degrees of freedom (df), one with less df, one with more df. Comment on your results.

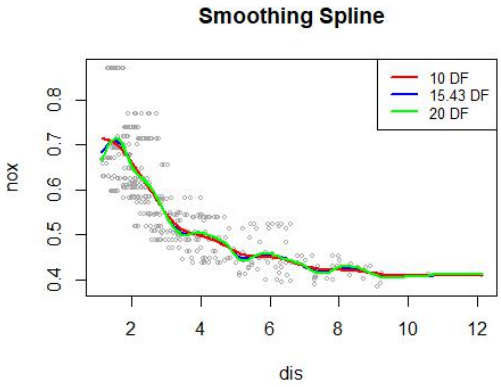
```
# polynomial regression
dislims=range(dis)
dis.grid=seq(from=dislims[1],to=dislims[2])
plot(dis,nox,xlim=dislims,cex=.5,col="darkgrey")
title("polynomial regression")
preds_pr=predict(lm.fit,newdata=data.frame(dis=dis.grid))
lm.fit1 = lm(nox~poly(dis, 4), data=Boston[train,])
lm.fit2 = lm(nox~poly(dis, 12), data=Boston[train,])
preds_pr1=predict(lm.fit1,newdata=data.frame(dis=dis.grid))
preds_pr2=predict(lm.fit2,newdata=data.frame(dis=dis.grid))
lines(dis.grid,preds_pr,col="blue",lwd=2)
lines(dis.grid,preds_pr1,col="red",lwd=2)
lines(dis.grid,preds_pr2,col="green",lwd=2)
legend("topright",legend=c("d=4","d=7","d=12"),col=c("red","blue","gree
n"),lty=1,lwd=2,cex=.8)
```




```
# natural spline
agelims=range(dis)
plot(dis,nox,xlim=agelims,cex=.5,col="darkgrey")
title("natural spline")
preds_ns=predict(ns.fit,newdata=data.frame(dis=dis.grid))
ns.fit1 = lm(nox~ns(dis,df = 5),data=Boston[train,])
ns.fit2 = lm(nox~ns(dis,df = 13),data=Boston[train,])
preds_ns1=predict(ns.fit1,newdata=data.frame(dis=dis.grid))
preds_ns2=predict(ns.fit2,newdata=data.frame(dis=dis.grid))
lines(dis.grid,preds_ns,col="blue",lwd=2)
lines(dis.grid,preds_ns1,col="red",lwd=2)
lines(dis.grid,preds_ns2,col="green",lwd=2)
legend("topright",legend=c("d=5","d=9","d=13"),col=c("red","blue","gree
n"),lty=1,lwd=2,cex=.8)
```




```
# smoothing spline
agelims=range(dis)
plot(dis,nox,xlim=agelims,cex=.5,col="darkgrey")
title("Smoothing Spline")
fit.20=smooth.spline(dis,nox,df=20)
fit.10=smooth.spline(dis,nox,df=10)
lines(fit.ss,col="blue",lwd=2)
lines(fit.10,col="red",lwd=2)
lines(fit.20,col="green",lwd=2)
legend("topright",legend=c("10 DF","15.43 DF","20 DF"),col=c("red","blu
e","green"),lty=1,lwd=2,cex=.8)
```



Comment: the plot has been shown above. According to the plots, the plot with smaller degree of freedom is always smooth compared with those with larger degree of freedom. But the plot with larger df should be more accurate in fitting data. Thus, the plot with optimal df is a kind of balance between accuracy and smooth.

(d) Fit a GAM on the training data to predict nox from dis and indus.Use what you learned in the previous question to select the best nonlinear function to model dis, and use the same type of function for indus. Plot the results and explain your findings.

```
# s() for smoothing, ns() for natural spline, poly() for polynomial
# fit.ss=smooth.spline(dis,nox,cv=TRUE)
# gam2=gam(wage~s(year,4)+s(age,6.79)+education,data=Wage[train,])
# smoothing spline
```

```r
library(splines)
fit.ss2 = smooth.spline(indus,nox,cv=TRUE)
```

```
## Warning in smooth.spline(indus, nox, cv = TRUE): cross-validation wi
th non-
## unique 'x' values seems doubtful
```

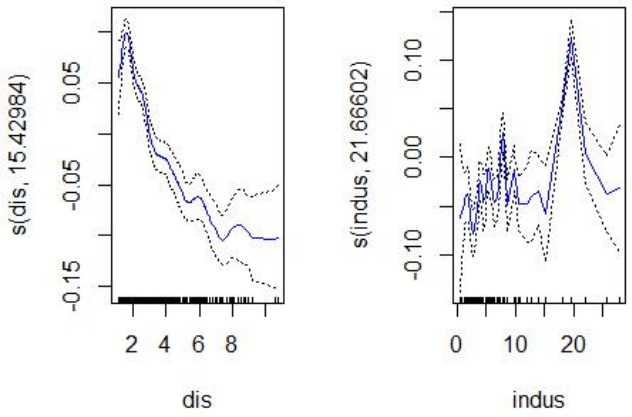
```r
df1 = fit.ss2$df
df1
```

```
## [1] 21.66602
```

```r
library(gam)

gam1=gam(nox~s(dis,15.42984)+s(indus,21.66602),data=Boston[train,])
summary(gam1)
```

```
##
## Call: gam(formula = nox ~ s(dis, 15.42984) + s(indus, 21.66602), dat
a = Boston[train,
##     ])
## Deviance Residuals:
##        Min         1Q     Median         3Q        Max
## -0.1653382 -0.0191646 -0.0009517  0.0169643  0.1333255
##
## (Dispersion Parameter for gaussian family taken to be 0.0022)
##
##     Null Deviance: 5.389 on 403 degrees of freedom
## Residual Deviance: 0.7887 on 365.9041 degrees of freedom
## AIC: -1295.791
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                         Df  Sum Sq Mean Sq F value    Pr(>F)
## s(dis, 15.42984)     1.0 3.11925 3.11925 1447.18 < 2.2e-16 ***
## s(indus, 21.66602)   1.0 0.33856 0.33856  157.08 < 2.2e-16 ***
## Residuals          365.9 0.78867 0.00216
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                     Npar Df  Npar F     Pr(F)
## (Intercept)
## s(dis, 15.42984)      14.4  6.1116 3.652e-11 ***
## s(indus, 21.66602)    20.7 11.5398 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
par(mfrow=c(1,2))
plot(gam1, se=TRUE,col="blue")
```



Comment: according to the plots, none of these plots is linear. It indicates that natural splines method is suitable for both predictors in this model.

(e) Report the test MSEs for all the methods you have implemented with relevant parameters suggested by cross-validation. Comment on your results.

```
# calculate the test MSE for polynomial regression
test.lm = predict(lm.fit,Boston[-train,])
test.ns = predict(ns.fit,Boston[-train,])
test.ss = predict(fit.ss,dis[-train])
test.gam = predict(gam1,Boston[-train,])
nox.test= nox[-train]
error.poly = mean((nox.test-test.lm)^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	0.0908248
natural spline	0.0040118
smoothing spline	0.0035419
GAM	0.0022855

Comment: according to the result shown above, the test error for GAM is the smallest and it should be the most suitable method to fit the data. In fact, natural splines, smoothing splines and GAM are approximately the same since their errors are similar, while for the polynomial regression, the test error is larger than the other three methods.