Stat 427/627 Statistical Machine Learning

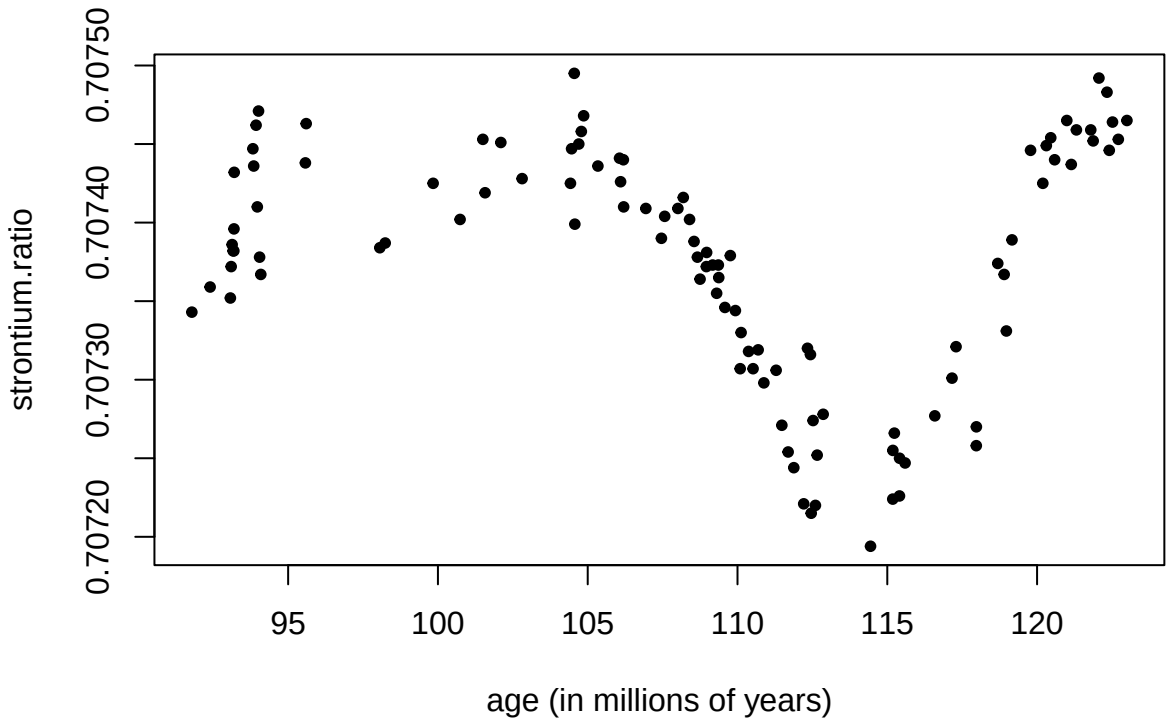
In-class Lab (Class 7, Topic 1): Introduction to Splines and Smoothing Methods

Contents

1 The fossil data

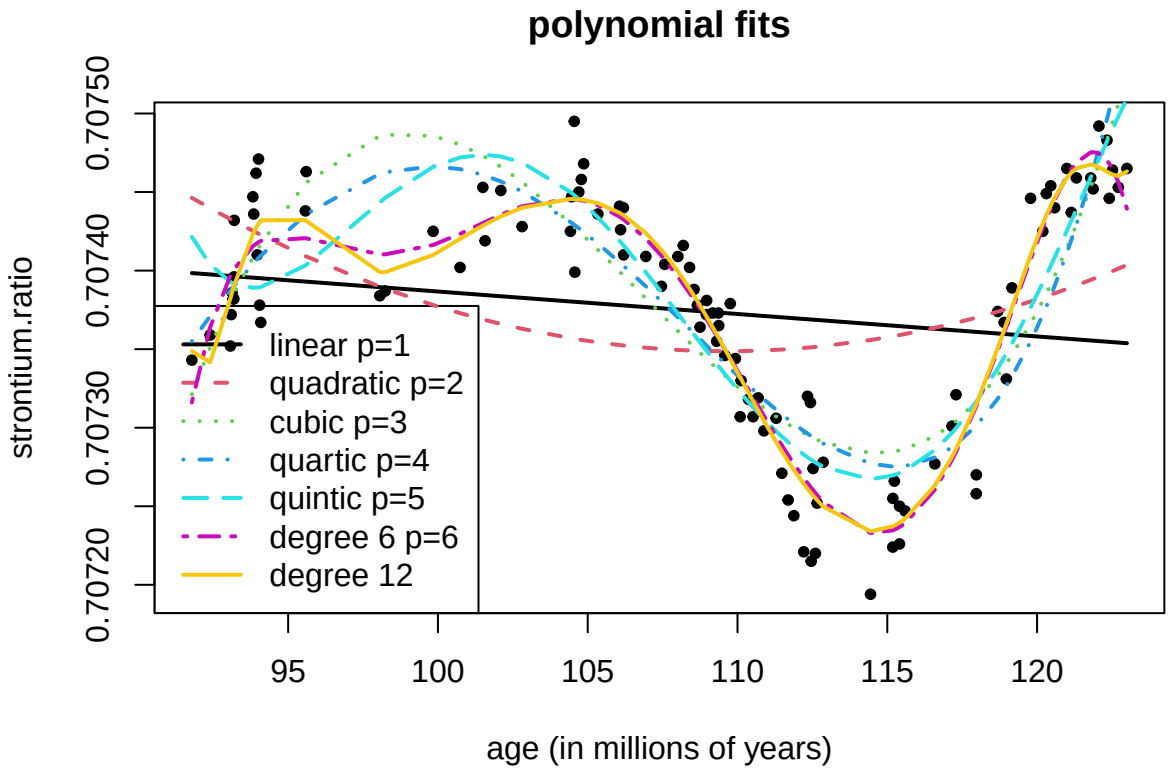
The `fossil.dat` data set in Canvas has data collected by Dr. Timothy Bralower of the Penn State Department of Geosciences. It consists of two variables measured on 106 fossil shells collected from cores drilled in the floor of the ocean: the age of shell dated by the surrounding material and the ratio of the appearance of isotopes of Strontium (in strontium.ratio). It is believed that this ratio is driven by sea level and is related to climate change. A scatter plot of the data is below.

```
fossil <- read.table("../Data/fossil.dat", header=T)
sx <- sort(fossil$age, index.return=T) # sorting here makes it easier to plot later
fossil <- fossil[sx$ix,]
attach(fossil)

## scatter plot of the data
plot(age, strontium.ratio, pch=20, xlab="age (in millions of years)")
```



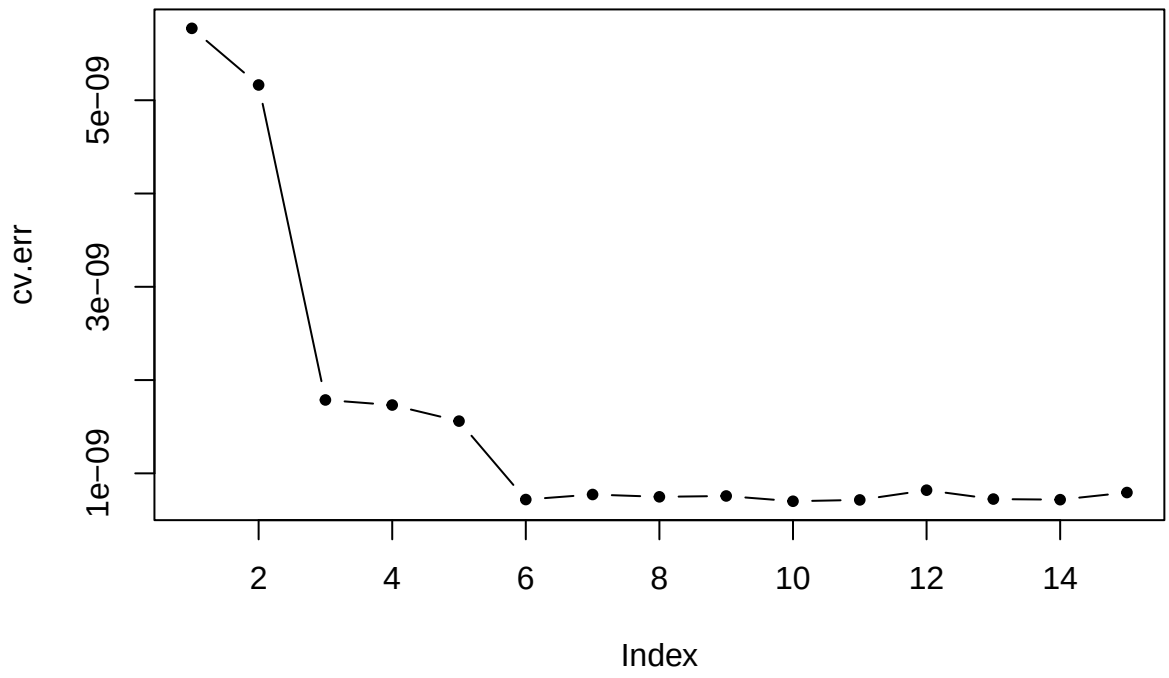
2 Polynomial Regression

```
poly.deg1 <- lm(strontium.ratio~age)
poly.deg2 <- lm(strontium.ratio~age+I(age^2))
poly.deg3 <- lm(strontium.ratio~age+I(age^2)+I(age^3))
poly.deg4 <- lm(strontium.ratio~age+I(age^2)+I(age^3)+I(age^4))
poly.deg5 <- lm(strontium.ratio~age+I(age^2)+I(age^3)+I(age^4)+I(age^5))
poly.deg6 <- lm(strontium.ratio~age+I(age^2)+I(age^3)+I(age^4)+I(age^5)+
    I(age^6))
poly.deg12 <- lm(strontium.ratio~poly(age, 12))

# plots of polynomial fits
plot(age, strontium.ratio, pch=20,xlab="age (in millions of years)",
    main="polynomial fits")
legtxt <- c("linear p=1","quadratic p=2","cubic p=3",
    "quartic p=4","quintic p=5","degree 6 p=6", "degree 12")
legend('bottomleft',legtxt, col=1:7,lty=1:7,lwd=2)
lines(age, poly.deg1$fitted,lwd=2)
lines(age, poly.deg2$fitted,lty=2,col=2,lwd=2)
lines(age, poly.deg3$fitted,lty=3,col=3,lwd=2)
lines(age, poly.deg4$fitted,lty=4,col=4,lwd=2)
lines(age, poly.deg5$fitted,lty=5,col=5,lwd=2)
lines(age, poly.deg6$fitted,lty=6,col=6,lwd=2)
lines(age, poly.deg12$fitted,lty=7,col=7,lwd=2)
```



Recall the `cv.glm()` function in `boot` package.

```
library(boot)
cv.err <- rep(0, 15)
set.seed(2023)
for (p in 1:15){
  poly.fit <- glm(strontium.ratio ~ poly(age, p), family="gaussian", data=fossil)
  cv.err[p] <- cv.glm(poly.fit, data=fossil)$delta[1]
}
plot(cv.err, type="b", pch=20)
```



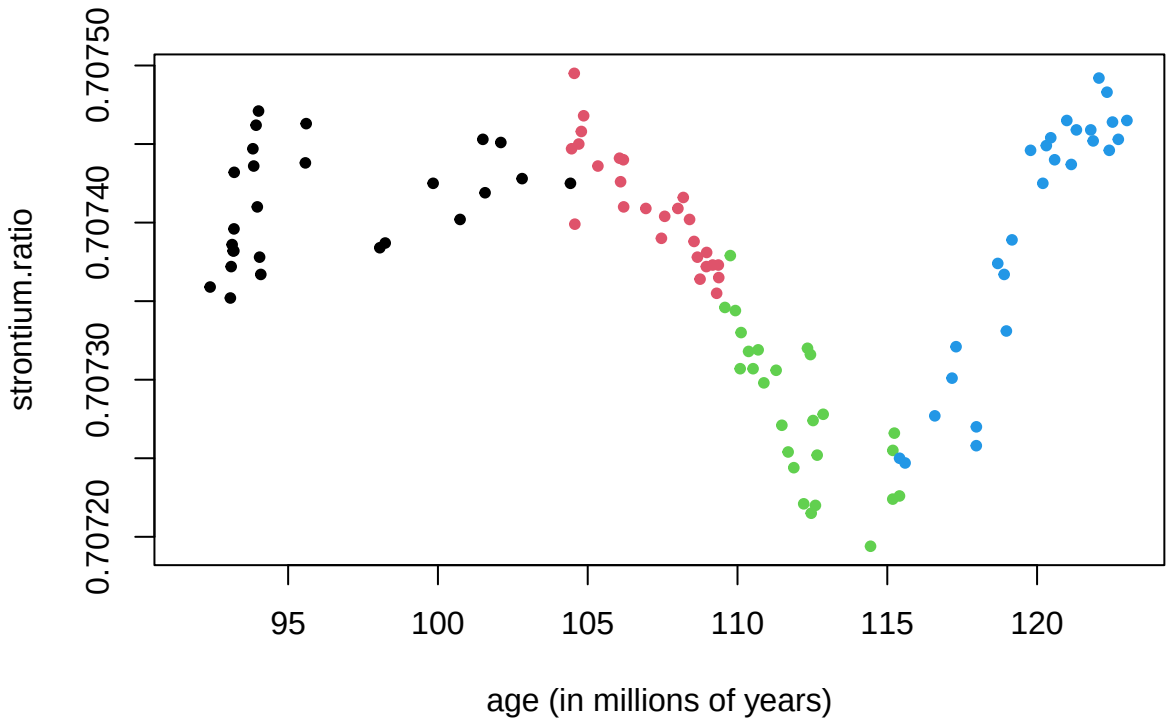
3 Piece-wise Regression and Splines

Let's cut the data into 4 segments according to the quarantines of `age`.

```
age.quan <- quantile(age)
age.group <- cut(age, breaks = age.quan)
table(age.group)
```

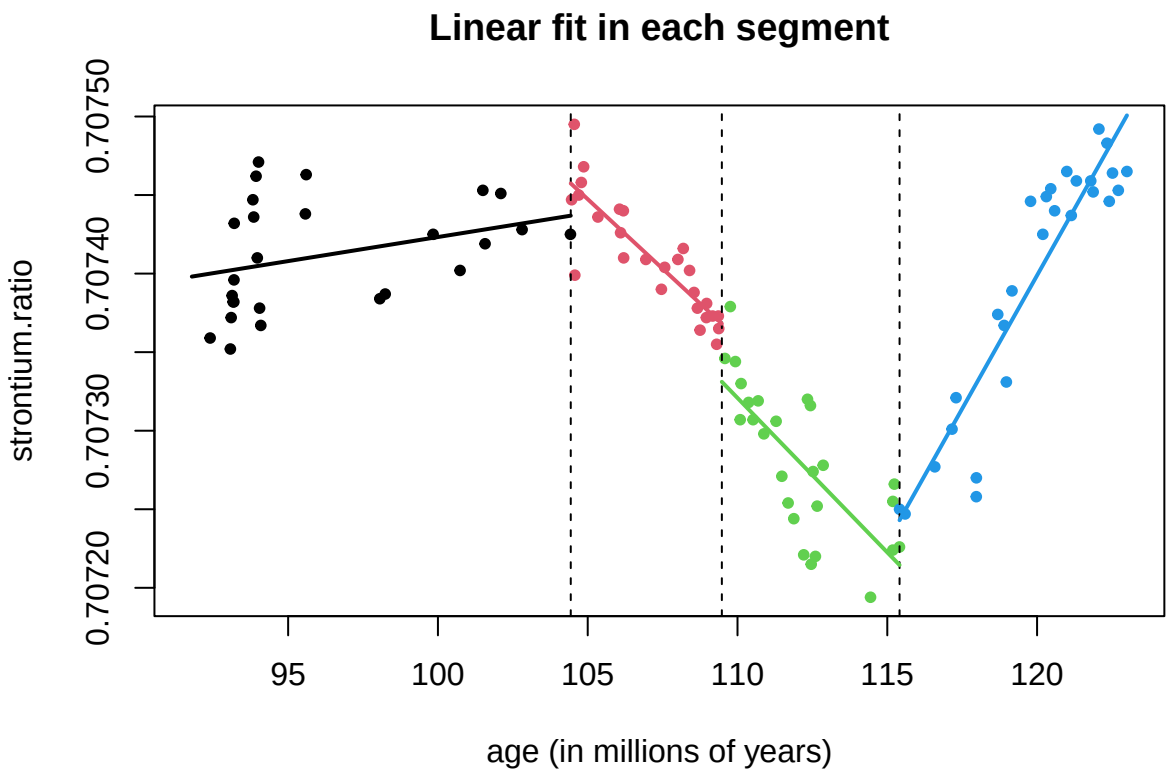
```
## age.group
## (91.8,104]  (104,109]  (109,115]  (115,123]
##         26         26         26         27
```

```
plot(age, strontium.ratio, col=age.group, pch=20,
     xlab="age (in millions of years)")
```

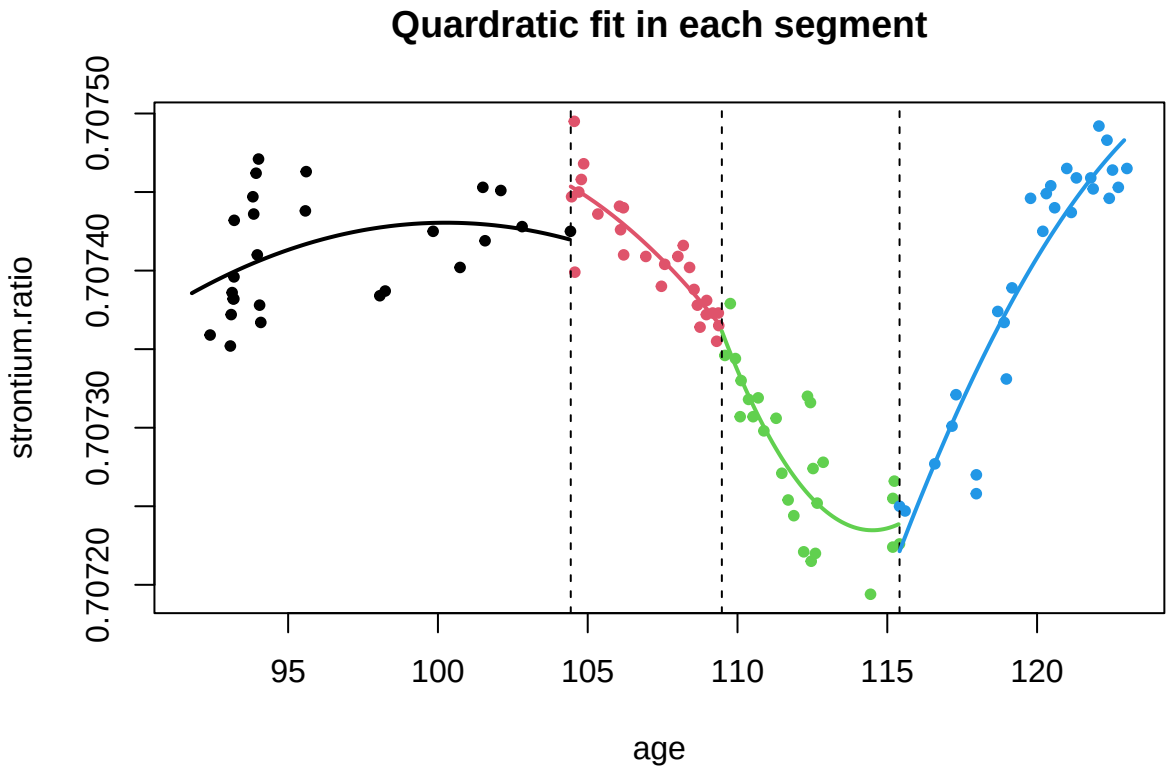


Broken lines with linear or quardratic function regression

```
plot(age, strontium.ratio, col=age.group, pch=20,
     xlab="age (in millions of years)",
     main = "Linear fit in each segment")
abline(v=age.quan[2:4], lty=2)
for (i in 1:4){
  broken.lin <- lm(strontium.ratio ~ age, data=fossil, subset = (age.group == levels(age.group)[i]))
  x <- c(age.quan[i], age.quan[i+1])
  lines(x, predict(broken.lin, newdata=data.frame(age=x)), col=i, lwd=2)
}
```

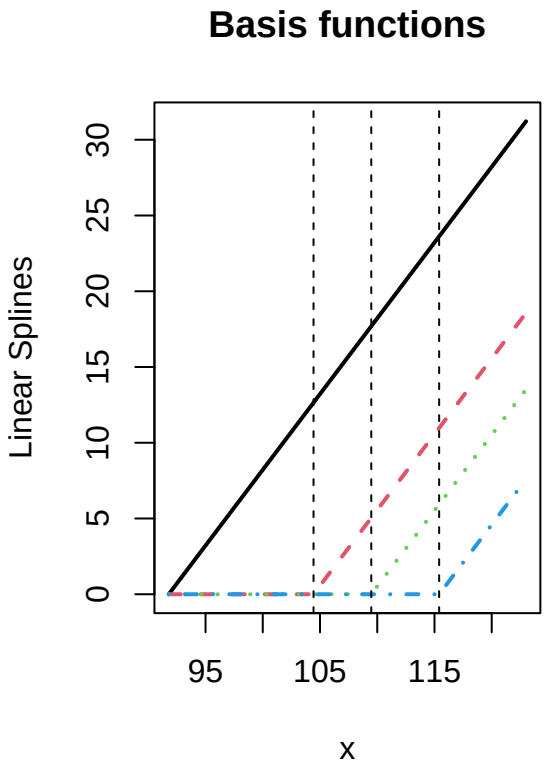


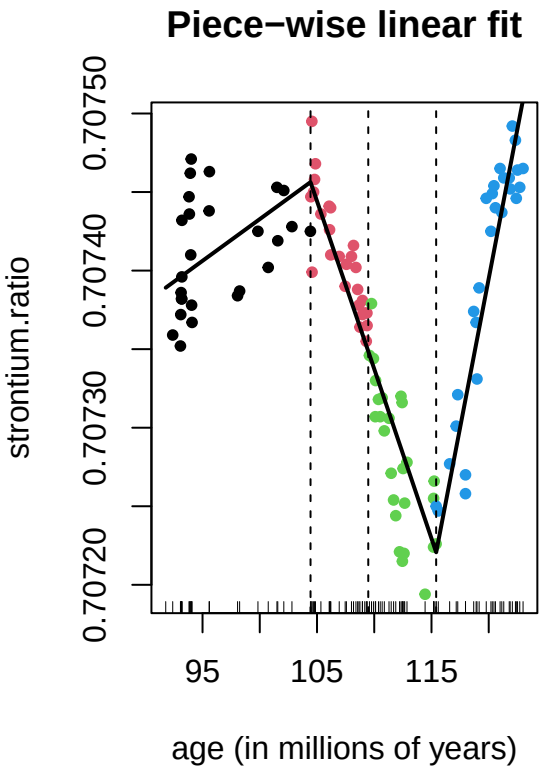

```
plot(age, strontium.ratio, col=age.group, pch=20,
        main = "Quardratic fit in each segment")
abline(v=age.quan[2:4], lty=2)
for (i in 1:4){
  broken.lin <- lm(strontium.ratio ~ age + I(age^2), data=fossil, subset = age.group == levels(age.grou
  x <- seq(age.quan[i], age.quan[i+1], by=0.1)
  lines(x, predict(broken.lin, newdata=data.frame(age=x)), col=i, lwd=2)
}
```



Continuous lines (splines)

```
n <- nrow(fossil)
basis1 <- matrix(NA, ncol=4, nrow=n)

par(mfrow=c(1, 2))

for (i in 1:4){
basis1[, i] <- ifelse((age > age.quan[i]), yes=age-age.quan[i], no=0)
}
colnames(basis1) <- c("b1", "b2", "b3", "b4")
matplot(age, basis1, type='l', lwd=2, xlab="x", ylab="Linear Splines",
    main="Basis functions")
abline(v=age.quan[2:4], lty=2)

pw.lin <- lm(strontium.ratio ~ basis1)
plot(age, strontium.ratio, pch=20, col=age.group,
     xlab="age (in millions of years)",
     main="Piece-wise linear fit")
lines(age, pw.lin$fitted, lwd=2)
rug(age)
abline(v=age.quan[2:4], lty=2)
```





4 B-spline (basis-spline)

The exact form of the B-spline (basis-spline) is derived in a recursive way and may be hard to write out analytically. It can be considered as piece-wise polynomial splines at various degrees.

Functions `bs()` in package `splines` creates the basis function of the B-splines.

- Argument "`knots =`" defines the knots.
- Argument "`degree =`" defines the (highest) order of the splines. (Cubic, by default.)
- After creating the splines, use `lm()` or `glm()` functions to fit the regression model.

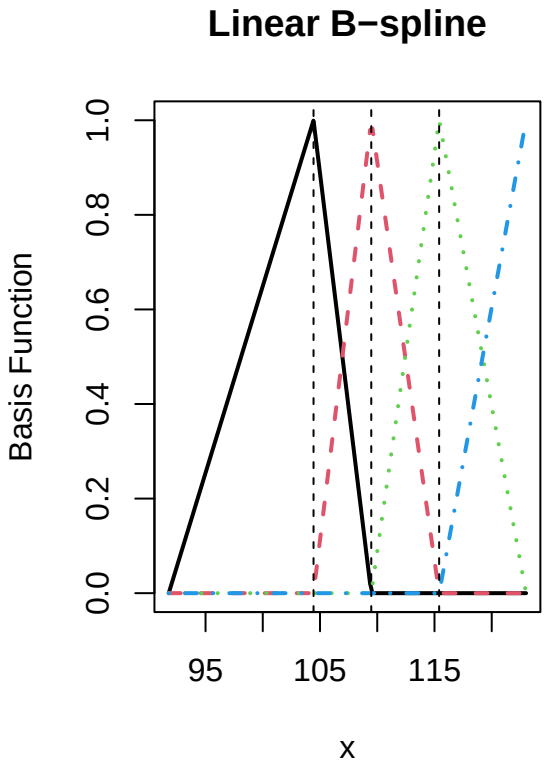
```
# install.packages("splines")
library(splines)
```

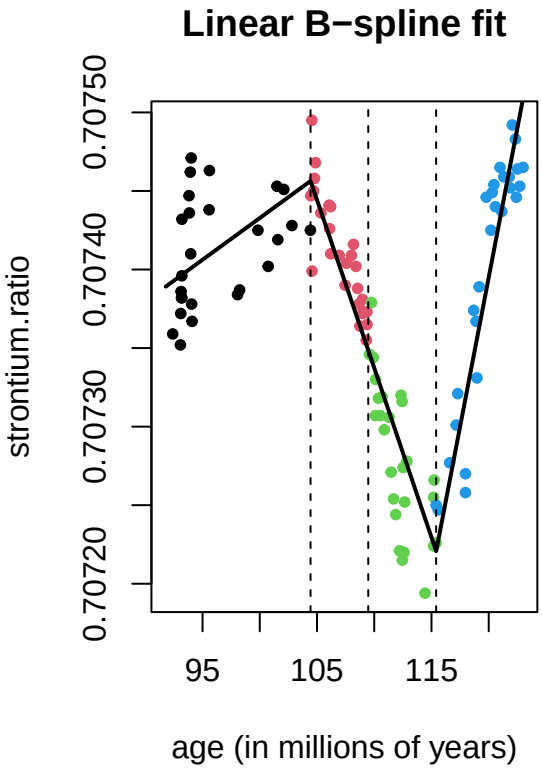
4.1 Linear B-spline (not smooth)

```
par(mfrow=c(1, 2))

bsage1 <- bs(age, knots=age.quan[2:4], degree=1)
matplot(age, bsage1, type='l', lwd=2, xlab="x", ylab="Basis Function",
    main="Linear B-spline")
abline(v=age.quan[2:4], lty=2)

bspline.fit1 <- lm(strontium.ratio~bsage1)
bspline.fit1$coefficients
```

```
##   (Intercept)       bsage11       bsage12       bsage13       bsage14
##  7.073891e-01  6.719856e-05 -4.048016e-05 -1.684489e-04  1.197341e-04
```

```r
plot(age,strontium.ratio,pch=20,xlab="age (in millions of years)",
    main="Linear B-spline fit", col=age.group)
lines(age, bspline.fit1$fitted,lwd=2)
abline(v=age.quan[2:4], lty=2)
```





- Note that the basis functions are not unique. Any one-to-one transformation of a set a basis functions can be used as basis functions and will lead to the same fitted regression line.

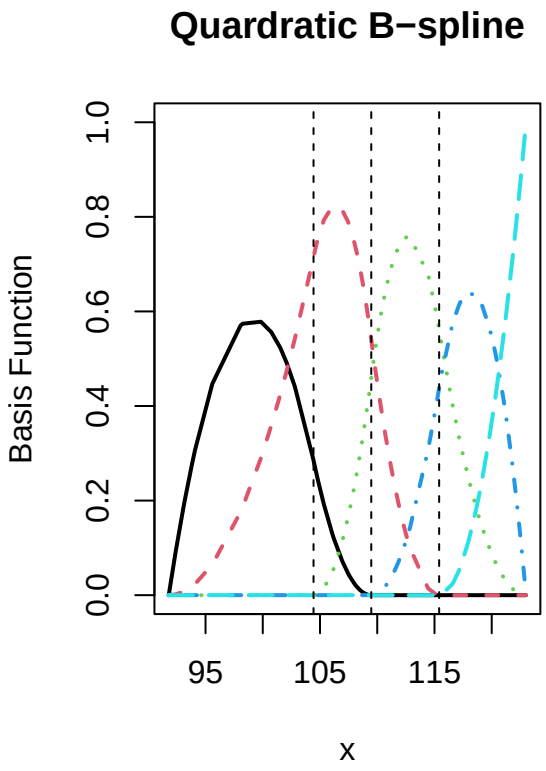
4.2 Quardratic

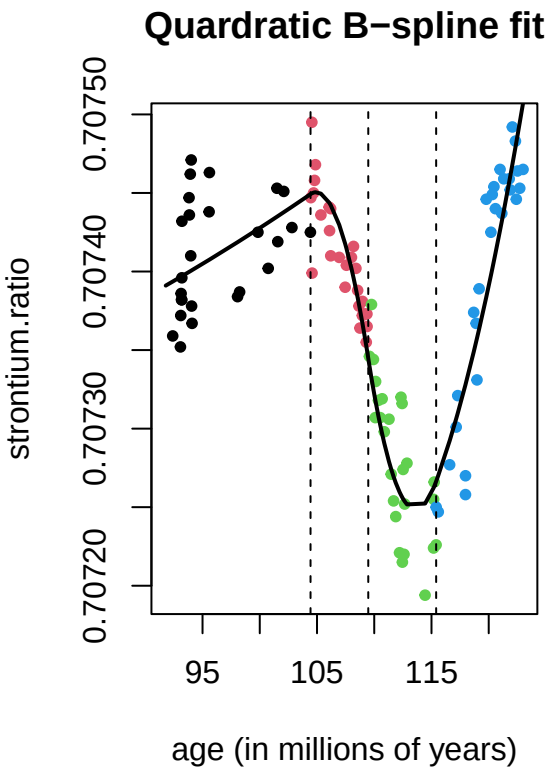
```r
knots <- age.quan[2:4]

par(mfrow=c(1, 2))

bsage2 <- bs(age, knots=knots, degree=2)
matplot(age, bsage2, type='l', lwd=2, xlab="x", ylab="Basis Function",
    main="Quardratic B-spline")
abline(v=knots, lty=2)

bspline.fit2 <- lm(strontium.ratio~bsage2)
bspline.fit2$coefficients
```

```
##   (Intercept)       bsage21       bsage22       bsage23       bsage24
##  7.073910e-01  2.662466e-05  7.080513e-05 -1.840711e-04 -4.837951e-05
```

```
##       bsage25
##  1.161913e-04
```

```
plot(age,strontium.ratio,pch=20,xlab="age (in millions of years)",
    main="Quardratic B-spline fit", col=age.group)
lines(age, bspline.fit2$fitted,lwd=2)
abline(v=knots, lty=2)
```



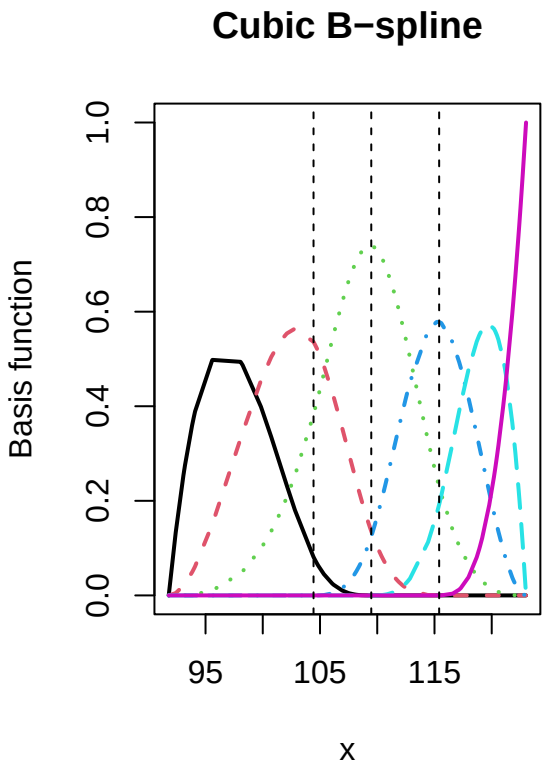


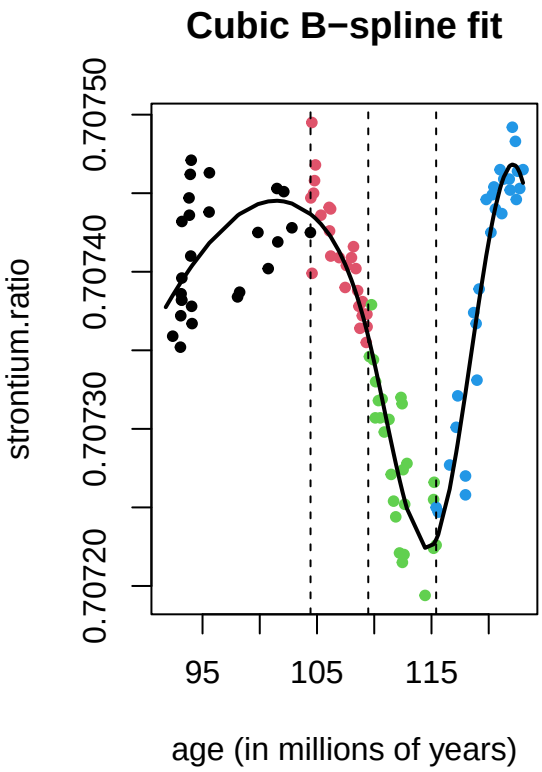
4.3 Cubic B-Spline

```
knots <- age.quan[2:4]

par(mfrow=c(1, 2))

bsage3 <- bs(age, knots=knots, degree=3)
matplot(age, bsage3, type='l', lwd=2, xlab="x", ylab="Basis function",
    main="Cubic B-spline")
abline(v=knots, lty=2)

bspline.fit3 <- lm(strontium.ratio~bsage3)
bspline.fit3$coefficients
```

```
##   (Intercept)       bsage31       bsage32       bsage33       bsage34
##  7.073775e-01  5.292379e-05  9.588816e-05  9.179065e-06 -3.088446e-04
##       bsage35       bsage36
##  1.481660e-04  7.909088e-05
```

```r
plot(age,strontium.ratio,pch=20,xlab="age (in millions of years)",
    main="Cubic B-spline fit", col=age.group)
lines(age, bspline.fit3$fitted,lwd=2)
abline(v=knots, lty=2)
```





4.4 Tuning using cross-validation.

We can conduct cross-validation to assess different choice of knots and/or how many knots.

Note that, the more knots we use, the more basis functions we need. Hence the model becomes more flexible.

```r
library(boot)

poly.deg6 <- glm(strontium.ratio ~ age)

bs3knots.fit <- glm(strontium.ratio ~ bs(age, knots = age.quan[2:4], degree=3),
                    family="gaussian")

bs6knots.fit <- glm(strontium.ratio ~ bs(age, knots = c(94.08, 104.7, 108.6,
                                                        110.6, 115.2, 119.7),
                                         degree=3),
                    family="gaussian")

cv.glm(data=fossil, poly.deg6, K=10)$delta[1]
```

```
## [1] 5.903382e-09
```

```
cv.glm(data=fossil, bs3knots.fit, K=10)$delta[1]
```

```
## [1] 8.458514e-10
```

```
cv.glm(data=fossil, bs6knots.fit, K=10)$delta[1]
```

```
## [1] 9.38517e-10
```

- Recall that cross-validation is a common approach to obtain reliable estimates of model accuracy measures. The estimates of the model accuracy measures can then be used for model comparison and tune the parameters of the machine learning algorithm. It can be applied to other spline methods.

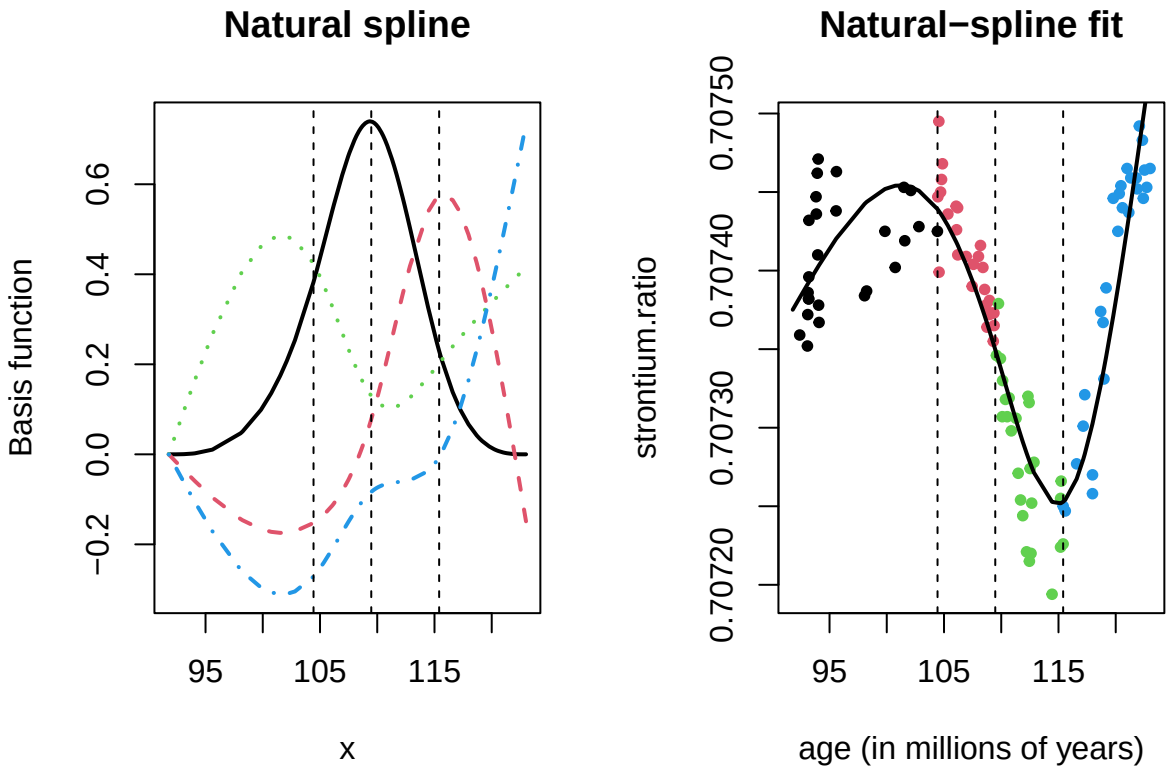
5 Natural (Cubic) Spline and Smoothing Spline

The B-spline is the fundation of other popular spline methods

5.1 Natural Spline

Function `ns()` fits Natural (Cubic) Splines to the data. The Natural (Cubic) Spline uses (cubic) B-splines, but with more constrains for boundary segments. E.g., linear spline for the first and the last segments.

```
par(mfrow=c(1, 2))

ns3knots <- ns(age, knots = age.quan[2:4])
matplot(age, ns3knots, type='l', lwd=2, xlab="x", ylab="Basis function",
    main="Natural spline")
abline(v=knots, lty=2)

ns3knots.fit <- glm(strontium.ratio ~ ns(age, knots = c(104.43, 109.48, 115.41)),
                    family="gaussian")
plot(age, strontium.ratio, pch=20,xlab="age (in millions of years)",
    main="Natural-spline fit", col=age.group)
lines(age, ns3knots.fit$fitted,lwd=2)
abline(v=knots, lty=2)
```



- We can tune the knots for Natural-Spline too.

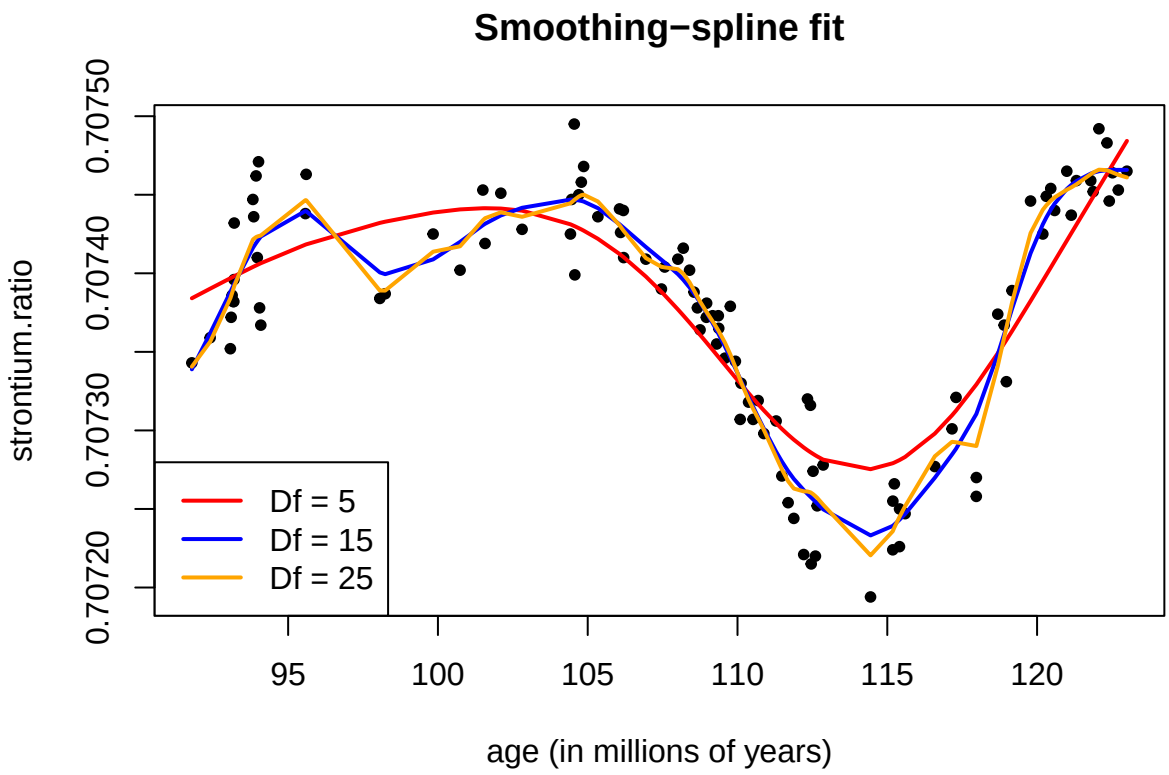
5.2 Smoothing Splines

Function `smooth.spline()` fits smoothing splines. You may consider smoothing splines as B-splines with many knots and a penalty on the "roughness" (i.e., the "bumps" in the plots, or, more precisely, $g''(x) = d^2g(x)/dx^2$).

- Use one of the following argument to set the model flexibility.
  - `df =`: degrees of freedom between 1 and (number of unique X-values). Large `df =` value makes the splines less smooth but more flexible. (Recall flexible models tend to have smaller bias, larger variance, and higher risk of overfitting.)
  - `spar =`: smooth parameter, typically between 0 and 1. Large `spar =` value makes the splines smoother, but less flexible. (Recall inflexible or restrictive models tend to have larger bias, smaller variance, and higher risk of underfitting.)
  - `lambda =`: penalty parameter. It depends on the scale of the predictor. Large `lambda =` value makes the splines smoother, but less flexible.

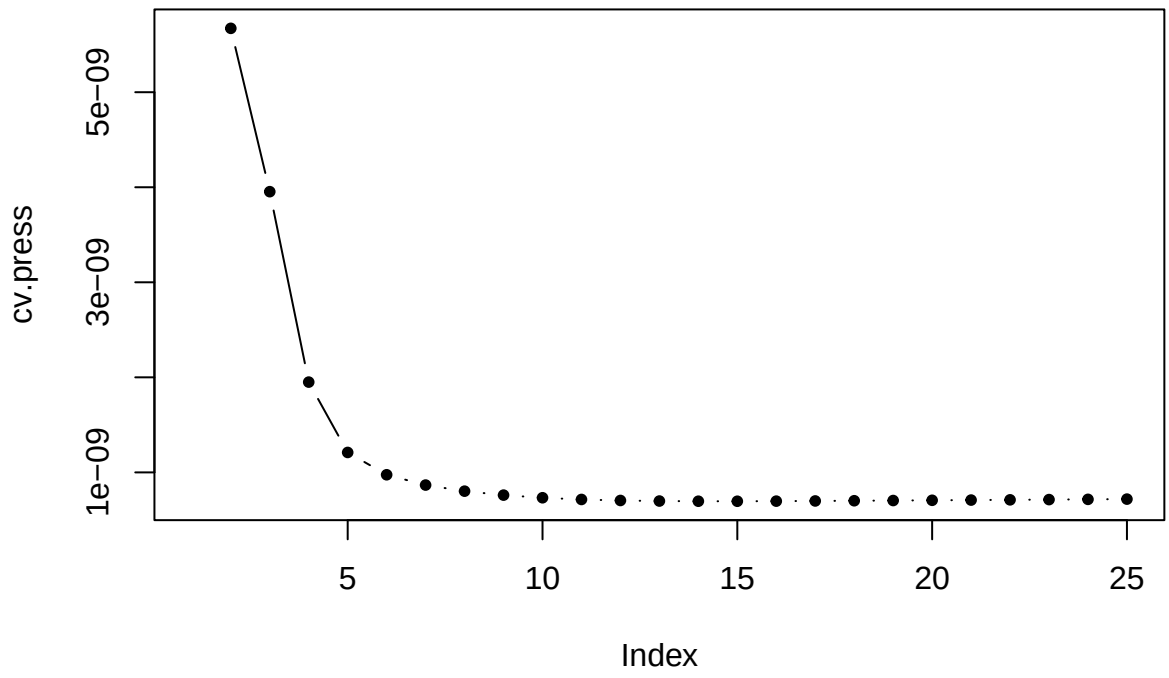
```
ss.df5 <- smooth.spline(x=age, y=strontium.ratio, df=5)
ss.df15 <- smooth.spline(x=age, y=strontium.ratio, df=15)
ss.df25 <- smooth.spline(x=age, y=strontium.ratio, df=25)

plot(age, strontium.ratio, pch=20,xlab="age (in millions of years)",
    main="Smoothing-spline fit")
lines(ss.df5, col="red", lwd=2)
lines(ss.df15, col="blue", lwd=2)
lines(ss.df25, col="orange", lwd=2)
```

```
legend('bottomleft', legend = c("Df = 5", "Df = 15", "Df = 25"), col=c("red", "blue", "orange"),
       lwd=2)
```



- Argument `cv = TRUE` computes LOOCV. For regression, the prediction Sum of Squared Errors (PRESS) will be calculated.

```
cv.press <- rep(NA, 25)
for (i in 2:length(cv.press)){
  cv.press[i] <- smooth.spline(x=age, y=strontium.ratio, df=i, cv = TRUE)$cv.crit
}

plot(cv.press, type="b", pch=20)
```

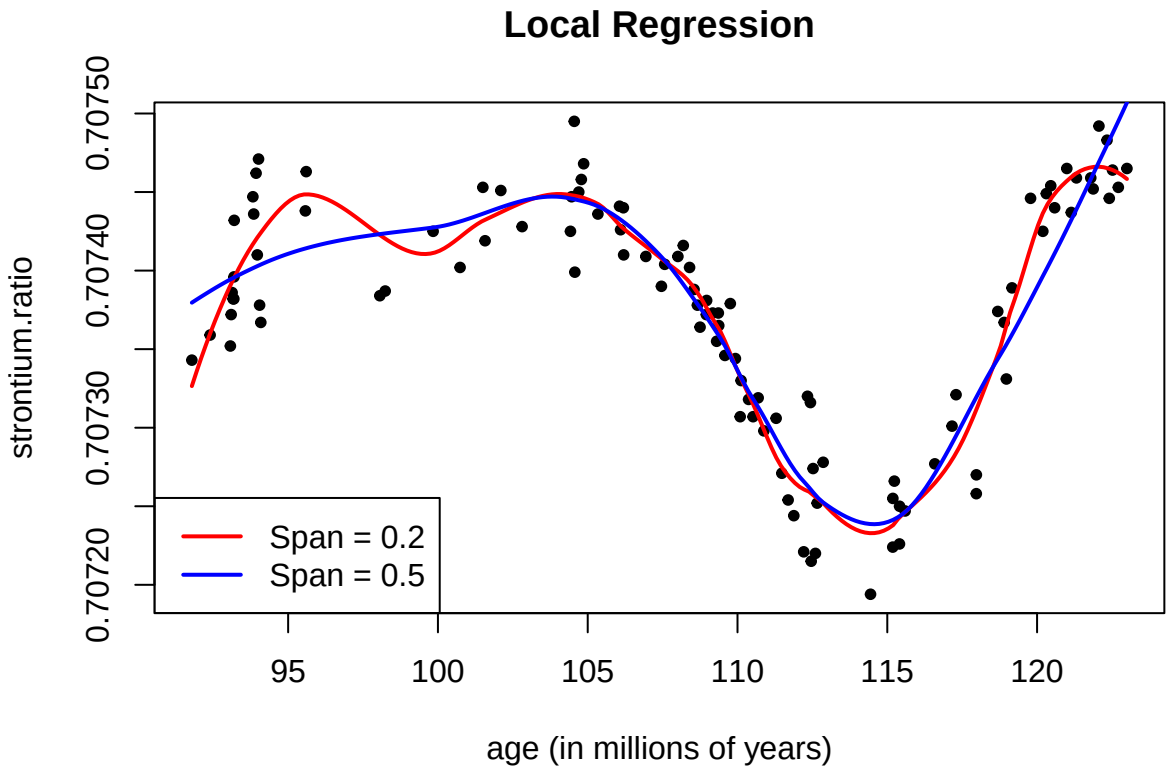



```
which.min(cv.press)
```

```
## [1] 15
```

6 Local Regression

```
plot(age , strontium.ratio, pch=20, xlab="age (in millions of years)",
     main= "Local Regression")
fit <- loess(strontium.ratio ~ age , span = .2, data = fossil)
fit2 <- loess(strontium.ratio ~ age , span = .5, data = fossil)
age.grid <- seq(min(age), max(age), by=diff(range(age))/200)
lines(age.grid, predict(fit , data.frame(age = age.grid)), col = "red", lwd = 2)
lines(age.grid, predict(fit2 , data.frame(age = age.grid)), col = "blue", lwd = 2)
legend("bottomleft", legend = c("Span = 0.2", "Span = 0.5"),
       col = c("red", "blue"), lty = 1, lwd = 2)
```



7 Generalized Additive Model (GAM)

```
# install.packages("gam")
library(gam)
? gam()
```