Projection Pursuit

Natashia Benjamin

2/20/2022

Cubic Splines and Natural Cubic Splines

The cubic spline uses the third-degree polynomial. To derive the solutions for the cubic spline, we assume the second derivation 0 at endpoints, which in turn provides a boundary condition that adds two equations to m-2 equations to make them solvable.

$$S(X) = \beta_0 X^3 + \beta_1 X^2 + \beta_2 X + \beta_3$$
$$S'(X) = 3\beta_0 X^2 + 2\beta_1 X + \beta_2$$
$$S''(X) = 6\beta_0 X + 2\beta_1$$

The cubic spline equations should not only continuous and differentiable but also have defined first and second derivatives that are also continuous on control points.

$$S'_i(X) = S'_{i+1}(X)$$
$$S''_i(X) = S''_{i+1}(X)$$

In Natural cubic spline, we assume that the second derivative of the spline at boundary points is 0, that is:

$$S''(X_0) = 0$$
$$S''(X_n) = 0$$

Assuming

$$t_i = x_i$$

for i= 0, 1,. . . n, and

$$z_i = S''(X_i), i = 0, 1, 2, ..., n$$

Fitting the PPR Model

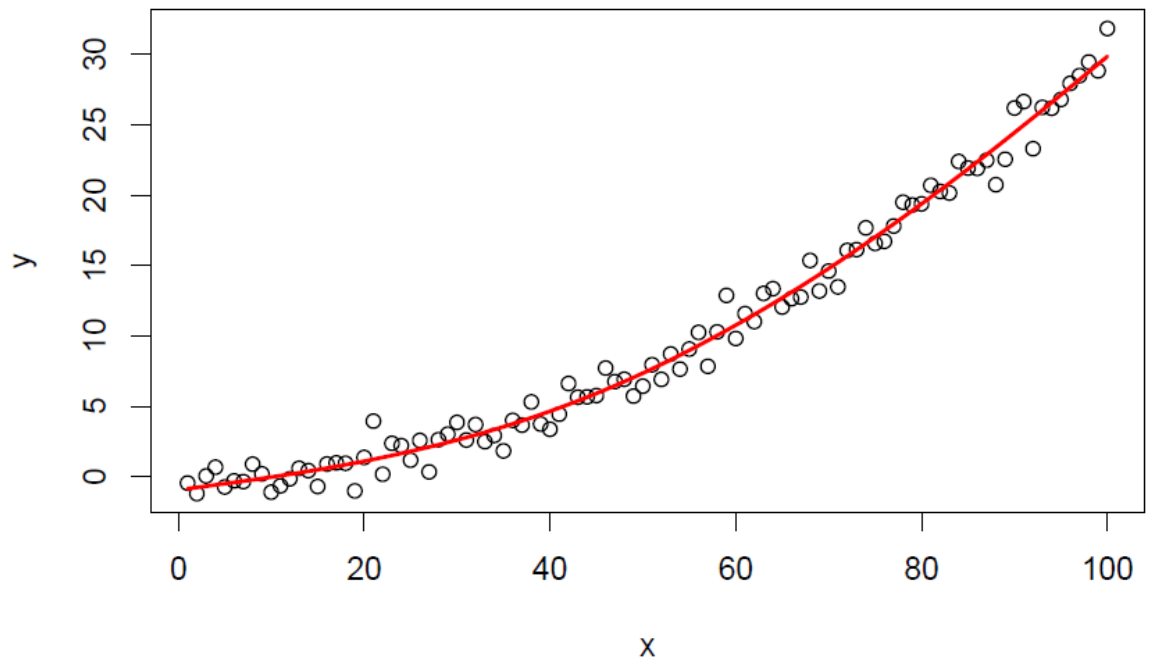
```
library("dplyr")
library("foreach")
```

```
set.seed(1000)
x <- seq(1:100)
y <- (3*(x^2)-2*x)/1000 + rnorm(100)

plot(x,y)
```

```r
x_y.ppr <- ppr(y~x, data=data.frame(x,y), nterms=1, max.terms = 1, sm.method = "spline")

points(x,x_y.ppr$fitted.values, type = 'l', lwd=2, col="red")
```




```r
summary(x_y.ppr)
```

```
## Call:
## ppr(formula = y ~ x, data = data.frame(x, y), nterms = 1, max.terms = 1,
##     sm.method = "spline")
##
## Goodness of fit:
##  1 terms
## 97.31162
##
## Projection direction vectors ('alpha'):
## [1] 1
##
## Coefficients of ridge terms ('beta'):
##   term 1
## 9.164612
##
## Equivalent df for ridge terms:
## term 1
##   5.04
```

```
sum((x_y.ppr$fitted.values-y)^2)
```

```
## [1] 97.31162
```

Fitting the Spline Model

For the projection pursuit algorithm, it will let:

$$v = \omega^t x$$

We need to choose a a value for omega to begin the model

```
#Basis Functions
S <- function(x){
  return(x^3 + x^2 + x)
}
S_d1<- function(x){
  return(3*x^2 + 2*x)
}
S_d2<- function(x){
  return(6*x + 2)
}
func <- function(x){

  return(y0*S(x)+y1*S(x))
}
d1_func <- function(x){

  return(y0*S_d1(x)+y1*S_d1(x))
}
d2_func <- function(x){

  return(y0*S_d2(x)+y1*S_d2(x))
}
```

```{r}

#knot1 <- 5; knot2 <- 10;
w <- as.matrix(0.01)
x <- as.matrix(seq(1:100))
v <- as.numeric(w %*% t(x))
y <- (3*(x^2)+ 2*x +1)/1000 + rnorm(100)
#y <- x^3 + x^2 +x  + rnorm(100)
#cub_spline <- lm(y~x)
#ppr_cubic <- cub_spline$fitted.values
#points(x,ppr_cubic, type = 'l', col = "blue")
xknots <- c(20, 50,75)
y0 <- xknots[1]
y1 <- xknots[3]
```

The

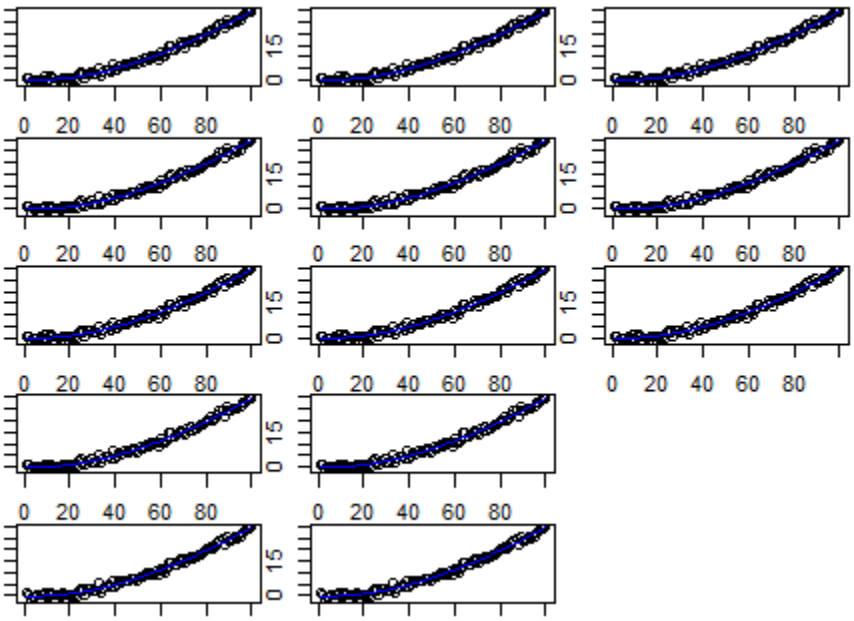
```{r}
num <- 3
while(num <= num){
  k1 <-v[xknots[1]]
  k2 <-v[xknots[2]]
  k3 <-v[xknots[3]]

  S1 <- as.matrix(func(v)-S_d1(v))
  S2 <- as.matrix(func(v)-S_d2(v))

  #lm.fit <- lm(y~ (!is.infinite(S1)+!is.infinite(S2)), data=data.frame(x,y))
  lm.fit <- lm(y~x+I(x^2)+I(x^3))
  ppr_cubic <- lm.fit$fitted.values
  deriv1 <- lm.fit$coefficients[1]*(d1_func(v)- d2_func(v))
  deriv1 <- deriv1 + lm.fit$coefficients[1]*(d1_func(v)-d2_func(v))
  plot(x,y)
  points(x,ppr_cubic, type = 'l', col = "blue")

  w <- w + sum(y-ppr_cubic/x*deriv1)
  v <- as.numeric(w %*% t(x))
  num <- num - 1
}

```



```
> summary(lm.fit)

Call:
lm(formula = y ~ x + I(x^2) + I(x^3))

Residuals:
     Min       1Q   Median       3Q      Max
-2.03497 -0.64205 -0.00037  0.55650  2.22905

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.876e-03  3.964e-01  -0.005   0.9962
x           -3.787e-02  3.382e-02  -1.119   0.2657
I(x^2)       4.332e-03  7.761e-04   5.582  2.2e-07 ***
I(x^3)      -1.000e-05  5.053e-06  -1.980   0.0506 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.9542 on 96 degrees of freedom
Multiple R-squared:  0.9897,    Adjusted R-squared:  0.9893
F-statistic:  3060 on 3 and 96 DF,  p-value: < 2.2e-16

> sum((lm.fit$fitted.values-y)^2)
[1] 87.40224
```

The values obtained by the ppr model, and the cubic spline were off by 10. It is possible to get values were much closer if the rstudio didn't constantly abort session. However, given how long it takes to converge, it is also possible that with the given code, and observing the graphs, the value would not have arrived that which was obtained by the ppr model.