Project-IV - Parametric/Nonparametric Nonlinear Regression

Md Al Masum Bhuiyan

November 05, 2018

Contents

This project is based on a data set jaws.txt, which is concerned about the association between jaw bone length (y = bone) and age in deer (x = age). I am going try out several parametric/nonparametric nonlinear regression models in this low-dimensional (p = 1) setting.

1 Problem 1: Data Import and Analysis

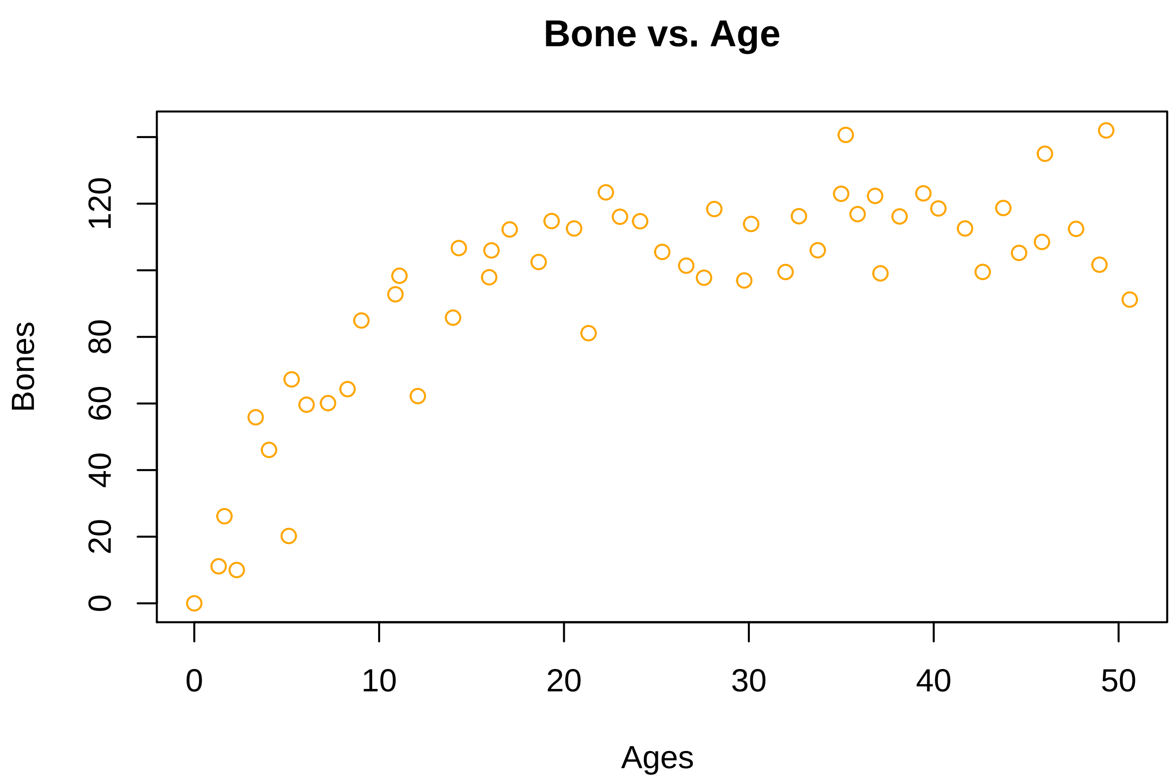
```
data <- read.table(file="/Users/masum/Desktop/Fall_2018/Data_Mining/Project/Project_04/jaws.txt
dat <- data
dim(dat)
```

```
## [1] 54  2
```

```
head(dat)
```

```
##          age      bone
## 1  0.000000   0.00000
## 2  5.112000  20.22000
## 3  1.320000  11.11130
## 4 35.240000 140.65000
## 5  1.632931  26.15218
## 6  2.297635  10.00100
```

```
plot(dat$age, dat$bone, main="Bone vs. Age", xlab="Ages ", ylab="Bones", pch=1, col="orange")
```



Remarks: The association between bones vs. ages (deer jaws) does not look linear.

2 Problem 2: Data Partition

I randomly partition the data into training and test with a ratio 2:1.

```
set.seed(123)
n <- nrow(dat)
split_data <- sample(x=1:2, size = n, replace =TRUE, prob=c(0.67, 0.33))
train <- dat[split_data ==1, ]
test <- dat[split_data ==2, ]
```

Now I indicate the observations that are from train or test obs.

```
dat$partition <- sample(c("train","test"), size=nrow(dat), replace=TRUE, prob=c(.67, .33))
dat
```

```
##          age      bone partition
## 1   0.000000   0.00000     train
## 2   5.112000  20.22000     train
## 3   1.320000  11.11130     train
## 4  35.240000 140.65000      test
## 5   1.632931  26.15218      test
## 6   2.297635  10.00100     train
## 7   3.322125  55.85634     train
## 8   4.043794  46.06754     train
## 9   5.266018  67.24174     train
## 10  6.075060  59.63458     train
## 11  7.236330  60.10914      test
## 12  8.291007  64.31393     train
## 13  9.040889  84.92216      test
## 14 10.881032  92.78716      test
## 15 11.098745  98.35159      test
## 16 12.093001  62.22000     train
## 17 13.998177  85.77307      test
## 18 14.310943 106.66963     train
## 19 15.954420  97.90825      test
## 20 16.076023 105.96718     train
## 21 17.062700 112.25996     train
## 22 18.629606 102.48663     train
## 23 19.330897 114.78824     train
## 24 20.546239 112.55459     train
## 25 21.329421  81.11000     train
## 26 22.264398 123.40000     train
## 27 23.028779 116.07627     train
## 28 24.118884 114.72701     train
## 29 25.314858 105.51583     train
## 30 26.612978 101.38623      test
## 31 27.574067  97.77144     train
## 32 28.130495 118.38879     train
## 33 29.751232  96.94697      test
```

```
## 34 30.123247 113.91214      test
## 35 31.986346  99.48091      test
## 36 32.707306 116.23068     train
## 37 33.726616 106.01806     train
## 38 34.993040 122.96777     train
## 39 35.882134 116.86115     train
## 40 36.829509 122.32905     train
## 41 37.117894  99.07155     train
## 42 38.153487 116.16566     train
## 43 39.439631 123.10436      test
## 44 40.251855 118.56645     train
## 45 41.693744 112.55648     train
## 46 42.649101  99.50362     train
## 47 43.764632 118.69918     train
## 48 44.615721 105.21623     train
## 49 45.855321 108.51228     train
## 50 46.015332 135.00000      test
## 51 47.700156 112.42955     train
## 52 48.964349 101.67611      test
## 53 49.329557 142.00000      test
## 54 50.604097  91.20000     train
```

3 Problem-3: Parametric Nonlinear Models

3.1 Problem-3(a) - Model Fitting

I fit an asymptotic exponential model from given problem with training data set.

```
control0 <- nls.control(maxiter = 5000, tol = 1e-05, minFactor = 1/10)
full.mod <- nls(bone ~ (beta1-beta2*exp(-beta3*age)), data=train,
               start=list(beta1 = 115, beta2 = 120, beta3 = 0.9), trace=T,
               control=control0)
```

```
## 28453.36 :  115.0 120.0   0.9
## 15816.13 :  112.0642268 114.8791089   0.4614787
## 4221.839 :  109.3227563 108.8996564   0.1746354
## 3741.965 :  110.6666282 116.1033177   0.1457199
## 3725.994 :  110.7395722 116.6675732   0.1509562
## 3725.985 :  110.7602931 116.6630297   0.1508762
## 3725.985 :  110.7601249 116.6634934   0.1508789
```

```
summary(full.mod)
```

```
##
## Formula: bone ~ (beta1 - beta2 * exp(-beta3 * age))
##
## Parameters:
##        Estimate Std. Error t value Pr(>|t|)
```

```
## beta1 110.76012    2.50589  44.200  < 2e-16 ***
## beta2 116.66349    7.96750  14.642 9.78e-16 ***
## beta3   0.15088    0.02114   7.138 4.22e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.79 on 32 degrees of freedom
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 7.28e-07
```

Remarks: The p-values of each case (β_1, β_2, and β_3) are smaller than the significance level (0.05), so the estimates of $\beta_1.\beta_2$, and β_3 are highly significant.

3.2 Problem-3(b): Hypothesis Testing

```
red.mod <- nls(bone ~ (beta1-beta1*exp(-beta3*age)), data=train,
               start=list(beta1 = 140, beta3 = .1), trace=T,
               control=control0)
```

```
## 18604.11 :  140.0   0.1
## 4107.508 :  109.2341511   0.1297439
## 3791.072 :  110.9584509   0.1418749
## 3790.713 :  111.0949813   0.1415447
## 3790.713 :  111.0934313   0.1415632
## 3790.713 :  111.0935209   0.1415622
```

The aim of model fitting is to explain as much of the variation in the dependent variable as possible from information contained in the independent variables. The contributions of the independent variables to the model are measured by partitions of the total sum of squares of the difference of data and model ("analysis of variance", ANOVA).

```
anova(full.mod, red.mod)
```

```
## Analysis of Variance Table
##
## Model 1: bone ~ (beta1 - beta2 * exp(-beta3 * age))
## Model 2: bone ~ (beta1 - beta1 * exp(-beta3 * age))
##   Res.Df Res.Sum Sq Df  Sum Sq F value Pr(>F)
## 1     32     3726.0
## 2     33     3790.7 -1 -64.729  0.5559 0.4614
```

Remarks: As expected, the residual sums of squares of reduced model is bigger than the full model. By definition, the null hypothesis of the test says that the reduced model is correct. On the other hand, the alternative hypothesis says that the reduced model is too simple and that the more complex full model is more appropriate. From ANOVA table, the p-value > 0.05 indicates that:

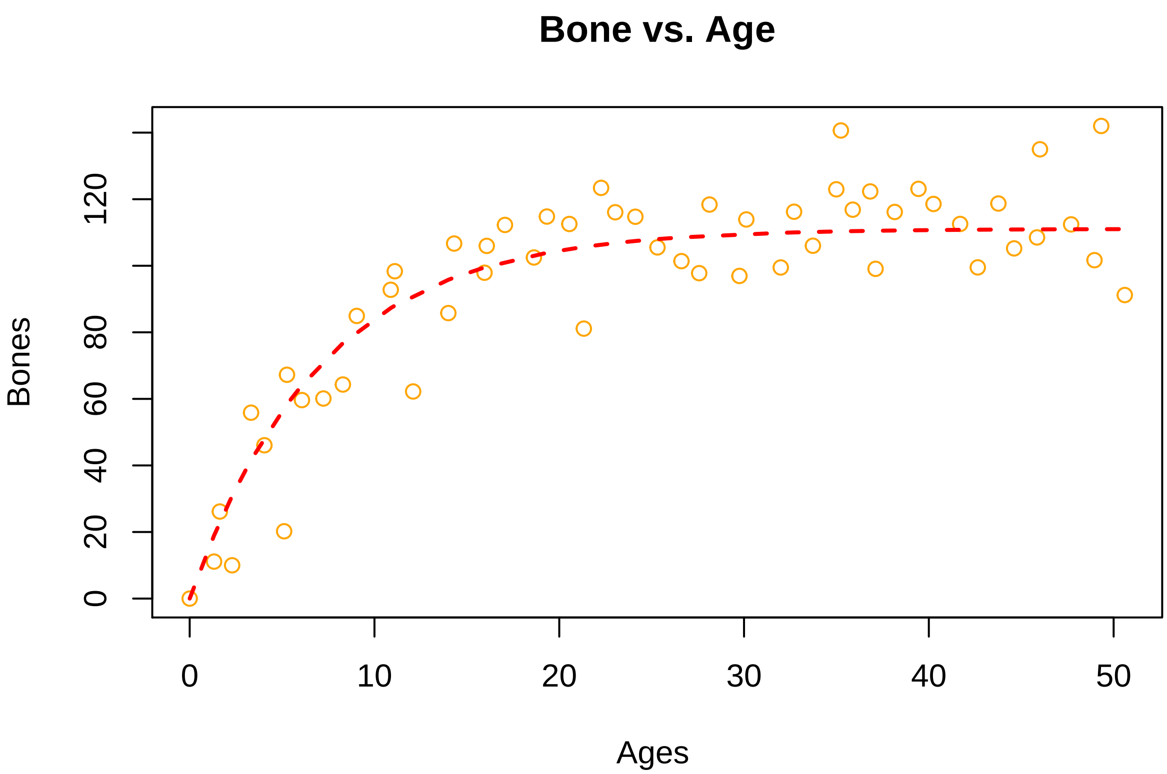
1. We accept the null hypothesis and conclude that the reduced model is better;

2. There is no lack of fit. The reason is that the pure error sd. $\sqrt(3790.7/33) = 10.71$ is less than the regression standard error of 10.79.

3.3 Problem-3(c) Fitted curve

I add a fitted curve of reduced model to the scatterplot.

```
plot(dat$age, dat$bone, main="Bone vs. Age", xlab="Ages ", ylab="Bones", col="orange", pch=1)
lines(sort(train$age), fitted(red.mod)[order(train$age)], lty = 2, col = "red", lwd = 2)
```



3.4 Problem-3(d) MSE

Now I apply the reduced model to the test dataset and compute the prediction mean square error (MSE).

```
pred <- predict(red.mod, test)
pmse <- function(y, yhat) mean((y-yhat)^2)
(a <- pmse(test$bone, pred))
```

```
## [1] 301.7998
```

4 Problem-4: Local Regression Methods

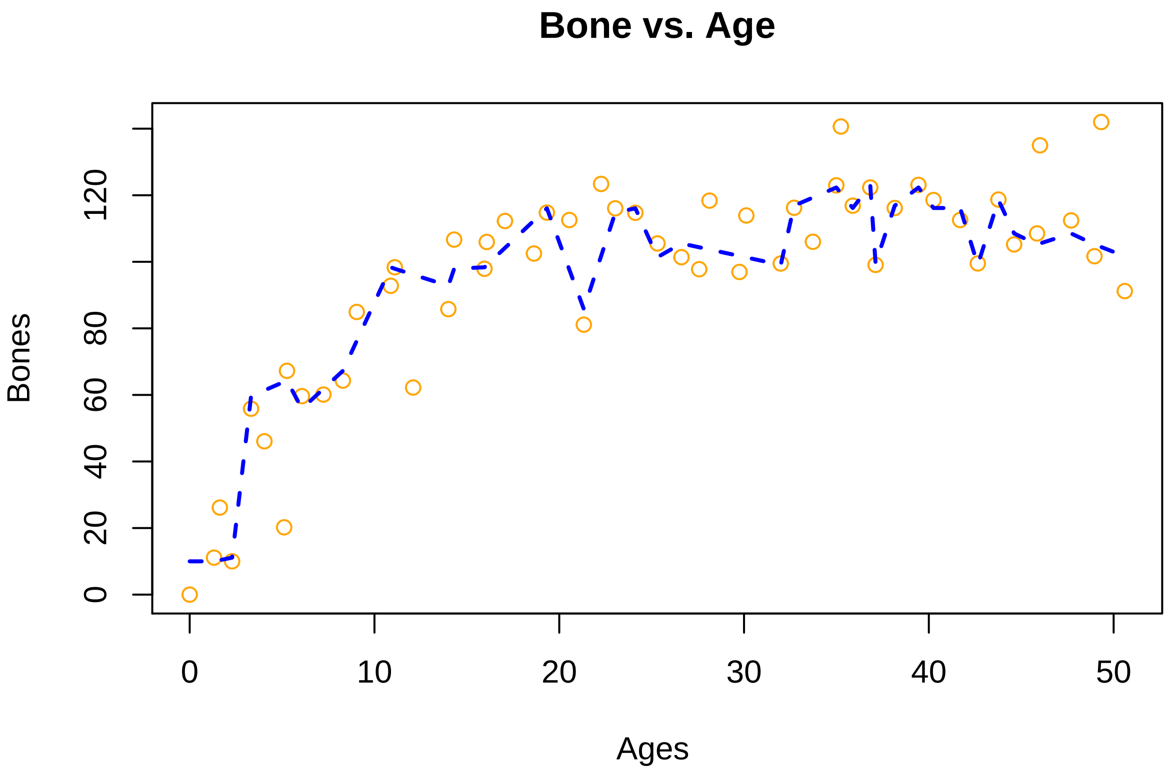
4.1 Problem 4(a): KNN regression

Given the data set, I want to find a regression function such that it is the best match of our data. For the KNN regression, the algorithm uses 'feature similarity' to predict values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set. I considered the number of neighbours as k=1, since it gives minimum SSE in the cross validation.

```
library("FNN")
set.seed(123)
# Fina a optimal K via 10-flog CV
SSEP <- function(yobs, yhat) sum((yobs-yhat)^2)
K <- 1:10
V <- 3
id.fold <- sample(1:V, size = NROW(train), replace=T)
SSE <- rep(0, length(K))
for(k in 1:length(K)){
  for(v in 1:V){
    train1<- train[id.fold!=v, ];
    train2<- train[id.fold==v, ];
    yhat2 <- knn.reg(train=train1, y=train1$bone, test=train2, k=K[k], algorithm="kd_tree")$pre
    SSE[k] <- (SSE[k] + SSEP(train2$bone, yhat2))
  }
}
cbind(K, SSE)
```

```
##        K        SSE
##  [1,]  1   776.9259
##  [2,]  2  1107.7884
##  [3,]  3  1576.6320
##  [4,]  4  3230.0434
##  [5,]  5  5251.5361
##  [6,]  6  7535.4341
##  [7,]  7  9258.5638
##  [8,]  8 11310.4174
##  [9,]  9 13415.5427
## [10,] 10 14768.3572
```

```
fit.knn <- knn.reg(train=train, y=train$bone, k=1, algorithm=("kd_tree"))
plot(dat$age, dat$bone, main="Bone vs. Age", xlab="Ages ", ylab="Bones", col="orange", pch=1)
lines(train$age, fit.knn$pred, col="blue", lwd=2, lty = 2)
```



I then applied the fitted model to test data and obtain the prediction MSE.

```
pred2 = knn.reg(train = train, test = test, y=train$bone, k = 1)
(b = pmse(test$bone, pred2$pred))
```

```
## [1] 72.64694
```

4.2 Problem 4(b): Kernel Regression

The kernel Regression is a non parametric technique to fit our data. It does not assume any underlying distribution to estimate the regression function. That is why, kernel regression is categorized as non paramteric technique. To improve the smoothness in KNN, kernel regression assigns the weights through a smooth kernel function, that is a pdf symmetric to 0.

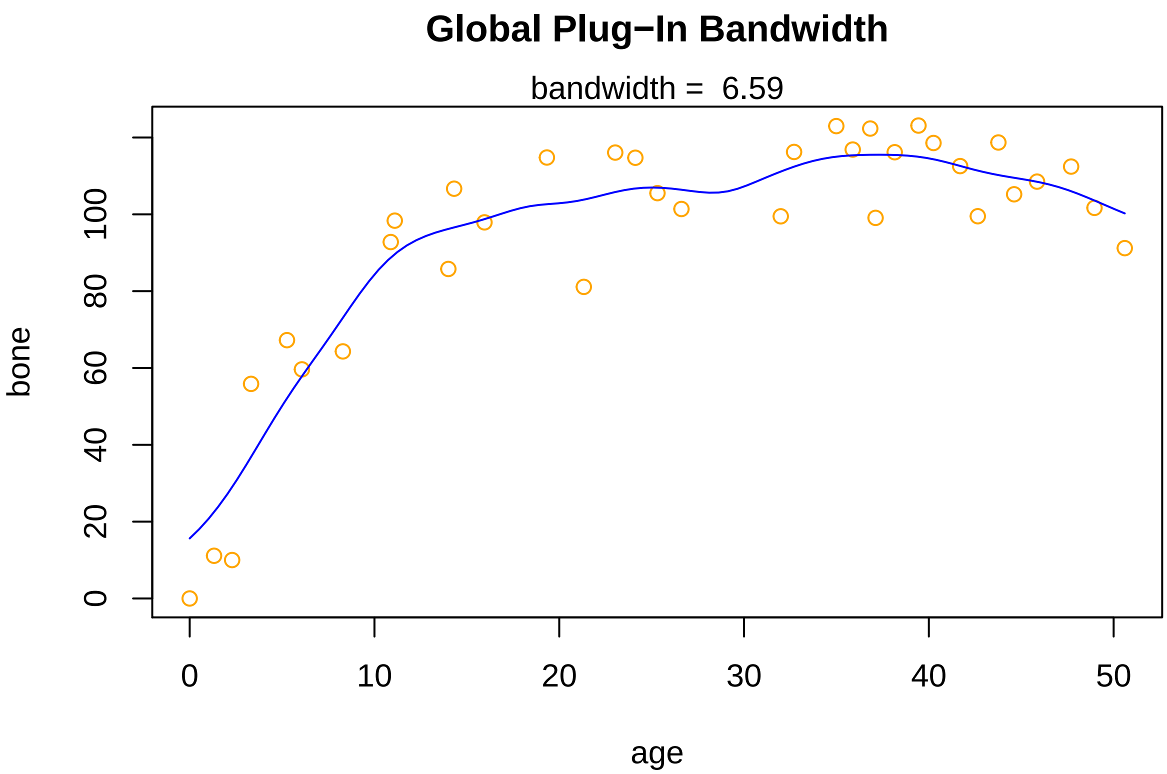
The kernel bandwidth works as a smoothing parameter. All kernels are scaled, so the upper and lower quartiles of the kernel (viewed as a probability density) are +/- 0.25. Larger values of bandwidth make smoother estimates; smaller values of bandwidth make less smooth estimates.

But too small bandwidths will lead to a wiggly curve, and too large ones will smooth away important details. The function glkerns calculates an estimator of the regression function or derivatives of the regression function with an automatically chosen global plugin bandwidth (6.59).

I use the optimal bandwidth in ksmooth, for which predict() works.

```
library(lokern)
with(train, {
  par(mfrow=c(1,1))
  plot(bone ~ age, data = train, main = "Global Plug-In Bandwidth", col="orange")
```

```
  fit <- glkerns(train$age, train$bone)
  lines(ksmooth(train$age, train$bone, "normal", bandwidth = fit$bandwidth), col = "blue")
   mtext(paste("bandwidth = ", format(fit$bandwidth, dig = 4)))
})
```



Now I use the test dataset to predict by kernel regression technique with optimal bandwidth, and compute the prediction mean square error (MSE).

```
fit <- glkerns(train$age, train$bone)
pred3 <- predict(fit, newdata=test)
```

```
## using first column of data.frame as 'x'
```

```
(c <- pmse(test$bone, pred3$y))
```

```
## [1] 317.5978
```

4.3 Problem 4(c): Local (Cubic) Polynomial Regression

The bias problems in kernel regression can be alleviated by local polynomial regression. Here, we use Epak kernels in the argument list. To predict the data, we use test data into the argument of xeval which is the vector of evaluation points. From lpFit we find the fitted value of bones and we estimate the MSE.

```
library(locpol)
r <- locpol(bone~age, data=train, deg=3, kernel=EpaK, bw=5)
pred4 <- locpol(bone~age, data=train, xeval=test$age, deg=3, kernel=EpaK, bw=5)
```

```
(d <-pmse(test$bone, pred4$lpFit$bone))
```

```
## [1] 870.8387
```

5 Problem 5: Regression/ Smoothing Splines

5.1 Problem 5(a): Regression Splines (Natural Cubic)

Splines fit the data very smoothly in most of the cases where polynomials would become wiggly and overfit the training data due to high variance at high degrees of polynomial.

Here, we use a Regression spline (non linear regression technique), where instead of building one model for the entire dataset, we divide the dataset into multiple bins and fit each bin with a separate model. The points where the division occurs are called Knots. The functions which we can use for modelling each piece/bin are known as Piecewise functions. There are various piecewise functions that we can use to fit these individual bins.

One problem with regression splines is that the estimates tend to display erractic behavior, i.e., they have high variance, at the boundaries of the domain of $x_1 \cdots x_n$. Even this gets worse as the order k gets larger. We can igonre this issue using natural splines, i.e., to force the piecewise polynomial function to have a lower degree to the left of the leftmost knot, and to the right of the rightmost knot. The cubic splines have continious 1st Derivative and continious 2nd derivative. The locations of the knots are typically quantiles of X, and the number of knots, K, is chosen by cross validation.

In the following code, we use *ns* function which generates a basis matrix for representing the family of piecewise-cubic splines with the specified sequence of interior knots, and the natural boundary conditions. These enforce the constraint that the function is linear beyond the boundary knots, which can either be supplied or default to the extremes of the data.

We then use *bs* function, which generates a basis matrix for representing the family of piecewise polynomials with the specified interior knots and degree, evaluated at the values of age variable.

After that, we compute the MSE, by predicting y using test data.

```
library(splines)
ns(data$age, df = 5)
```

```
##                 1            2            3          4            5
##  [1,] 0.000000e+00 0.000000e+00  0.000000000 0.00000000  0.000000000
##  [2,] 2.598412e-02 0.000000e+00 -0.129418065 0.37483551 -0.245417446
##  [3,] 4.473601e-04 0.000000e+00 -0.036840523 0.10670177 -0.069861244
##  [4,] 1.521964e-02 4.657894e-01  0.455169213 0.12022528 -0.056403553
##  [5,] 8.469135e-04 0.000000e+00 -0.045413853 0.13153283 -0.086118980
##  [6,] 2.359275e-03 0.000000e+00 -0.063261915 0.18322645 -0.119964531
##  [7,] 7.131540e-03 0.000000e+00 -0.089436292 0.25903569 -0.169599402
##  [8,] 1.286183e-02 0.000000e+00 -0.106579388 0.30868750 -0.202108114
##  [9,] 2.840420e-02 0.000000e+00 -0.132422423 0.38353708 -0.251114657
## [10,] 4.361012e-02 0.000000e+00 -0.146841120 0.42529817 -0.278457050
## [11,] 7.370391e-02 0.000000e+00 -0.163018267 0.47215229 -0.309134021
```

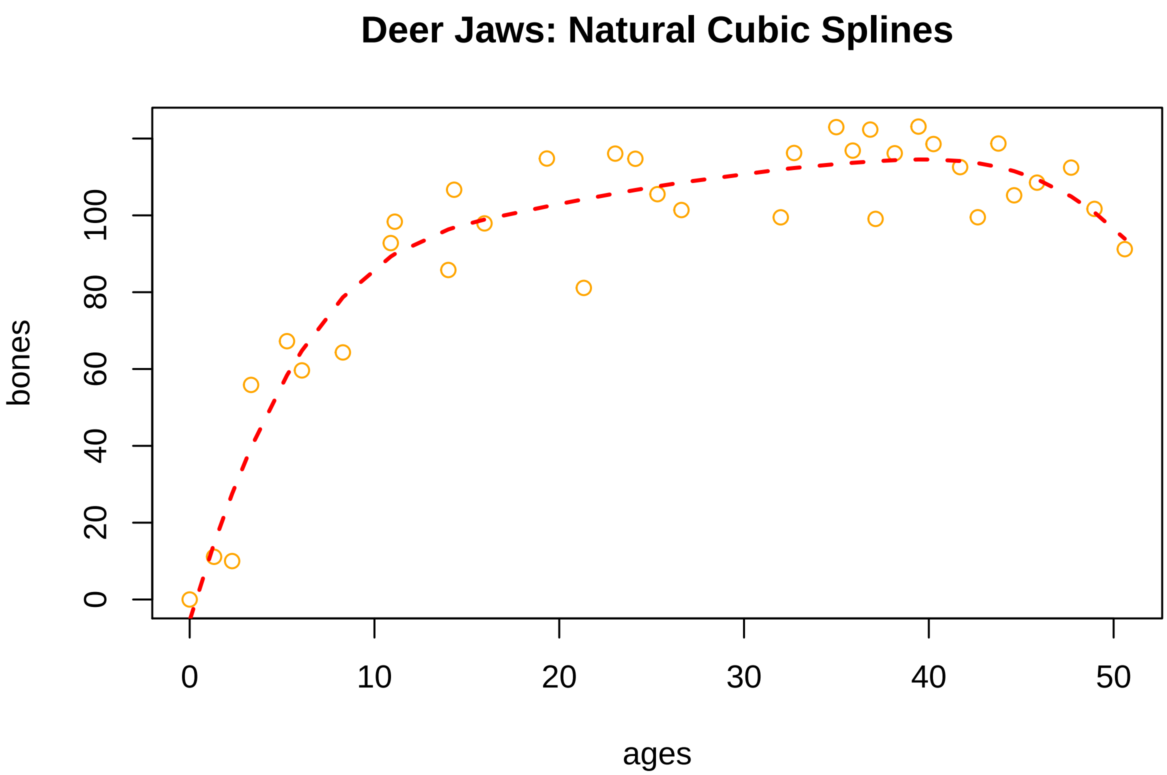
```
## [12,] 1.108555e-01 0.000000e+00 -0.172343138 0.49916005 -0.326816915
## [13,] 1.437199e-01 3.768559e-06 -0.175444957 0.50814390 -0.332698942
## [14,] 2.443530e-01 1.368727e-03 -0.170997564 0.49526285 -0.324265283
## [15,] 2.575969e-01 1.830388e-03 -0.169488948 0.49089342 -0.321404472
## [16,] 3.200564e-01 5.259625e-03 -0.160413109 0.46460693 -0.304193820
## [17,] 4.412133e-01 2.029041e-02 -0.135135305 0.39139444 -0.256259135
## [18,] 4.603060e-01 2.413151e-02 -0.130271438 0.37730715 -0.247035710
## [19,] 5.514655e-01 5.241463e-02 -0.102943401 0.29815654 -0.195213139
## [20,] 5.574020e-01 5.511034e-02 -0.100858505 0.29211802 -0.191259517
## [21,] 6.001247e-01 8.047545e-02 -0.083994680 0.24327507 -0.159280390
## [22,] 6.433281e-01 1.350302e-01 -0.058734745 0.17011433 -0.111379590
## [23,] 6.505857e-01 1.658442e-01 -0.048675998 0.14098103 -0.092305035
## [24,] 6.427136e-01 2.292052e-01 -0.033796453 0.09829081 -0.064354308
## [25,] 6.249461e-01 2.753867e-01 -0.025449008 0.07609580 -0.049822486
## [26,] 5.927989e-01 3.338996e-01 -0.015911296 0.05467884 -0.035800082
## [27,] 5.591125e-01 3.830212e-01 -0.007878587 0.04101751 -0.026855546
## [28,] 5.020618e-01 4.525417e-01  0.004959264 0.02706912 -0.017723068
## [29,] 4.310047e-01 5.243460e-01  0.022415958 0.01857956 -0.012164662
## [30,] 3.490284e-01 5.917798e-01  0.047447529 0.01649881 -0.010802327
## [31,] 2.884283e-01 6.312918e-01  0.071412075 0.01913850 -0.012530623
## [32,] 2.544397e-01 6.490497e-01  0.087827237 0.02210095 -0.014470238
## [33,] 1.649585e-01 6.744128e-01  0.148253390 0.03583147 -0.023460072
## [34,] 1.472617e-01 6.738477e-01  0.165113366 0.03990276 -0.026125615
## [35,] 7.732498e-02 6.347553e-01  0.264505736 0.06445437 -0.041040412
## [36,] 5.775462e-02 6.060475e-01  0.307102386 0.07559033 -0.046494881
## [37,] 3.616974e-02 5.556611e-01  0.368252580 0.09263117 -0.052714601
## [38,] 1.785032e-02 4.813338e-01  0.441638692 0.11556437 -0.056387168
## [39,] 9.615748e-03 4.243705e-01  0.488579238 0.13258263 -0.055148081
## [40,] 4.154474e-03 3.619771e-01  0.531895654 0.15132987 -0.049357118
## [41,] 3.046245e-03 3.430080e-01  0.543351978 0.15713138 -0.046537613
## [42,] 6.870692e-04 2.765064e-01  0.576371365 0.17820246 -0.031767332
## [43,] 5.634986e-06 2.012557e-01  0.596022456 0.20457166 -0.001855406
## [44,] 0.000000e+00 1.604666e-01  0.593814215 0.22112603  0.024593205
## [45,] 0.000000e+00 1.023213e-01  0.561624601 0.25015263  0.085901477
## [46,] 0.000000e+00 7.281170e-02  0.522448779 0.26915543  0.135584094
## [47,] 0.000000e+00 4.627512e-02  0.461025230 0.29114264  0.201557010
## [48,] 0.000000e+00 3.106049e-02  0.404202459 0.30778962  0.256947427
## [49,] 0.000000e+00 1.548910e-02  0.308398306 0.33186805  0.344244544
## [50,] 0.000000e+00 1.397554e-02  0.295048831 0.33496354  0.356012097
## [51,] 0.000000e+00 3.541967e-03  0.143592760 0.36741705  0.485448218
## [52,] 0.000000e+00 6.376943e-04  0.020200528 0.39164295  0.587518830
## [53,] 0.000000e+00 2.994617e-04 -0.016437144 0.39862873  0.617508947
## [54,] 0.000000e+00 0.000000e+00 -0.146042808 0.42298601  0.723056801
## attr(,"degree")
## [1] 3
## attr(,"knots")
##       20%       40%       60%       80%
##  8.740936 19.573965 30.048844 39.764521
```

```
## attr(,"Boundary.knots")
## [1]  0.0000 50.6041
## attr(,"intercept")
## [1] FALSE
## attr(,"class")
## [1] "ns"     "basis"  "matrix"
```

```
fm1 <- lm(bone ~ bs(age, df = 5), data = train)
summary(fm1)
```

```
##
## Call:
## lm(formula = bone ~ bs(age, df = 5), data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.081  -5.719   1.808   8.228  16.291
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)          -5.449      7.660  -0.711    0.483
## bs(age, df = 5)1     92.762     15.909   5.831 2.54e-06 ***
## bs(age, df = 5)2    107.881     12.434   8.676 1.49e-09 ***
## bs(age, df = 5)3    121.023     15.094   8.018 7.66e-09 ***
## bs(age, df = 5)4    120.777     11.478  10.522 2.05e-11 ***
## bs(age, df = 5)5     99.357     11.737   8.465 2.50e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.41 on 29 degrees of freedom
## Multiple R-squared:  0.9111, Adjusted R-squared:  0.8958
## F-statistic: 59.44 on 5 and 29 DF,  p-value: 2.371e-14
```

```
par(mfrow=c(1,1))
plot(bone~age, data=train, xlab = "ages", ylab = "bones", main="Deer Jaws: Natural Cubic Splin
#spd <- seq(min(train$age), max(train$age), len = 35)
lines(sort(train$age), fm1$fitted.values[order(train$age)], lty = 2, col = "red", lwd = 2)
```




```
pred5 <- predict(fm1, test)
(e <- pmse(test$bone, pred5))
```

```
## [1] 357.6741
```

5.2 Problem 5(b): Smoothing Splines

Smoothing splines are a regularized regression over the natural spline basis. In smoothing splines, we have a Knot at every unique value of $x_1, \cdots x_n$. It circumvents the problem of knot selection (as they just use the inputs as knots), and simultaneously, they control for overfitting by shrinking the coefficients of the estimated function (in its basis expansion). So it does not require the selection of the number of Knots, but require selection of only a Roughness Penalty (λ) which accounts for the wiggliness (fluctuations) and controls the roughness of the function and variance of the Model.

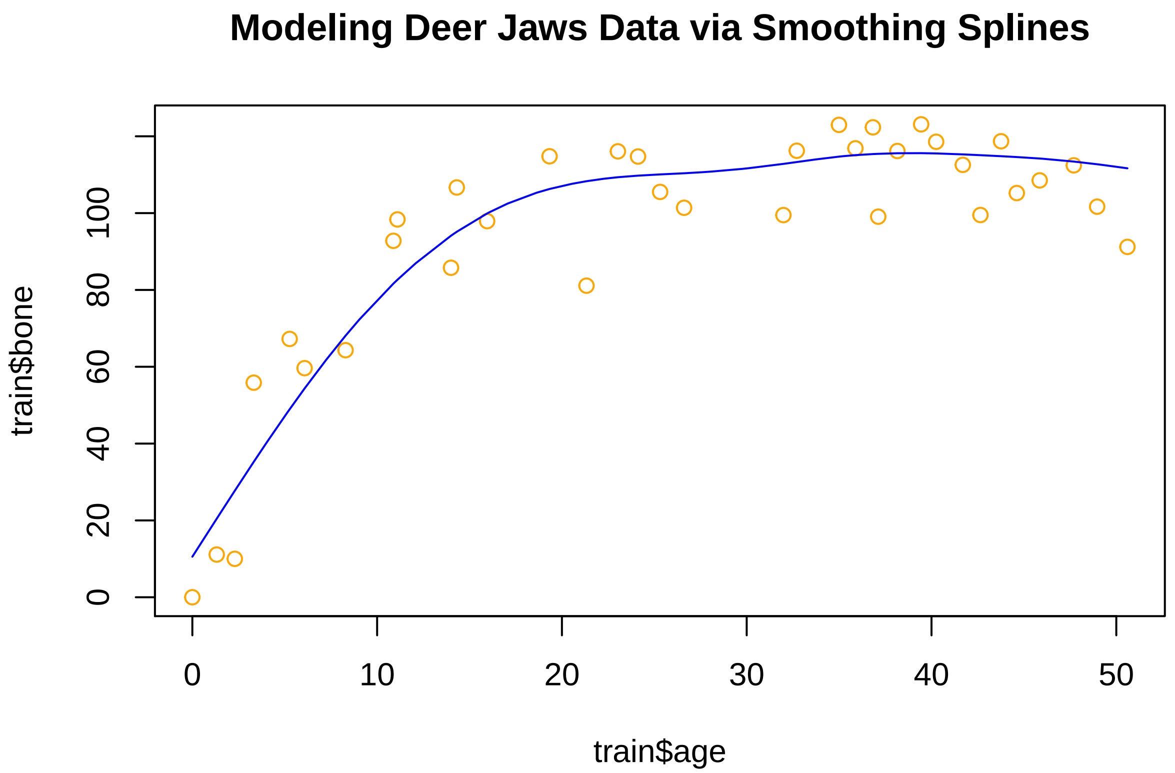
The smaller the λ, the more wiggly and fluctuating the function is. As λ, approcahes ∞, the function g(x) becomes linear.

```
library(splines)
plot(train$age, train$bone, main = "Modeling Deer Jaws Data via Smoothing Splines", col="orange
(jaws.spl <- smooth.spline(data$age, data$bone, cv=TRUE))
```

```
## Call:
## smooth.spline(x = data$age, y = data$bone, cv = TRUE)
##
## Smoothing Parameter  spar= 0.8004763  lambda= 0.002580874 (11 iterations)
## Equivalent Degrees of Freedom (Df): 5.285307
## Penalized Criterion (RSS): 8868.881
```

```
## PRESS(l.o.o. CV): 489.4116
```

```
lines(jaws.spl, col = "blue")
```




```
pred6 <- predict(jaws.spl, test)
(f <- pmse(test$bone, pred6$y$bone))
```

```
## [1] 3215.662
```

I determine the tuning parameter as lambda= 0.002580874 from the cross-validation. That is why, I use $CV = TRUE$ in the code.

6 Problem-6

```
(tabulate <- data.frame(
  Methods = c("NLS","KNN","Kernel Regression","local (cubic) Poly.", "Regression Spline", "Smoo
  MSE = c(a, b, c, d, e, f)
))
```

```
##                 Methods        MSE
## 1                   NLS  301.79976
## 2                   KNN   72.64694
## 3     Kernel Regression  317.59778
## 4  local (cubic) Poly.  870.83866
## 5     Regression Spline  357.67407
## 6         Smooth spline 3215.66187
```

The KNN mehtod gives minimum MSE and thus it gives favorable results.