Natural Cubic Spline, Random Forest, and Gaussian Kernel Density Estimator

Yizhou Zhang

April 2018

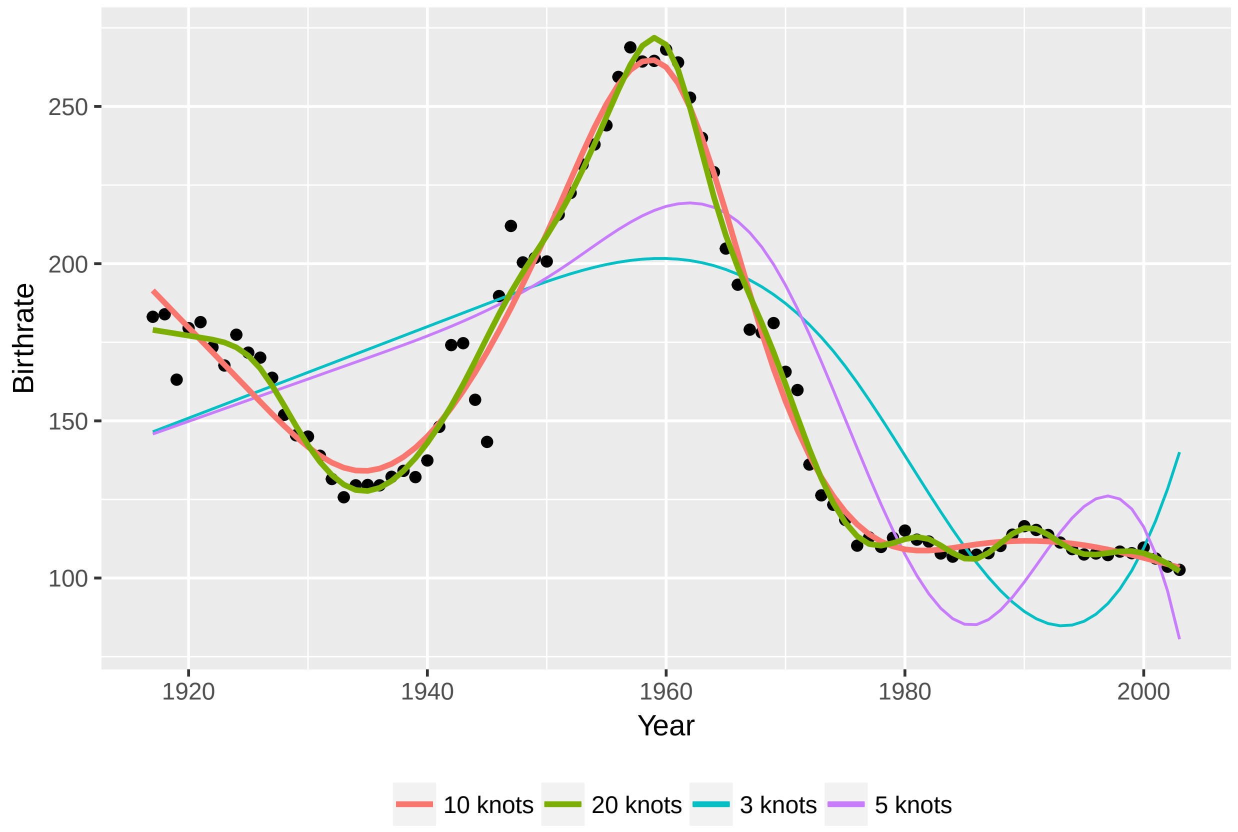
Q2, Implement the NCS estimator

Define a function named `ncs` with a single argument `n` as the number of knots. The kth kont is at the `k/n` quantile of running variable. Then a natural cubic spline is estimated as:

$$f(x) = \beta_0 + \beta_1 + \sum_{k=1}^{K} \theta_k (X - \xi_k)_+^3$$

The following code first constructs the ξ_k terms, then perform linear regression.

```r
ncs = function(n){
k = quantile(x = df$Year, probs = (1:n)*(1/n))
for (i in 1:length(k)){
  temp = paste("K", i, sep = "_")
  df[,temp] = ifelse(df$Year - k[i] >=0, (df$Year - k[i])^3 , 0)
}
ncs_fit = lm(Birthrate ~ ., data = df)
return(fitted(ncs_fit))
}
```



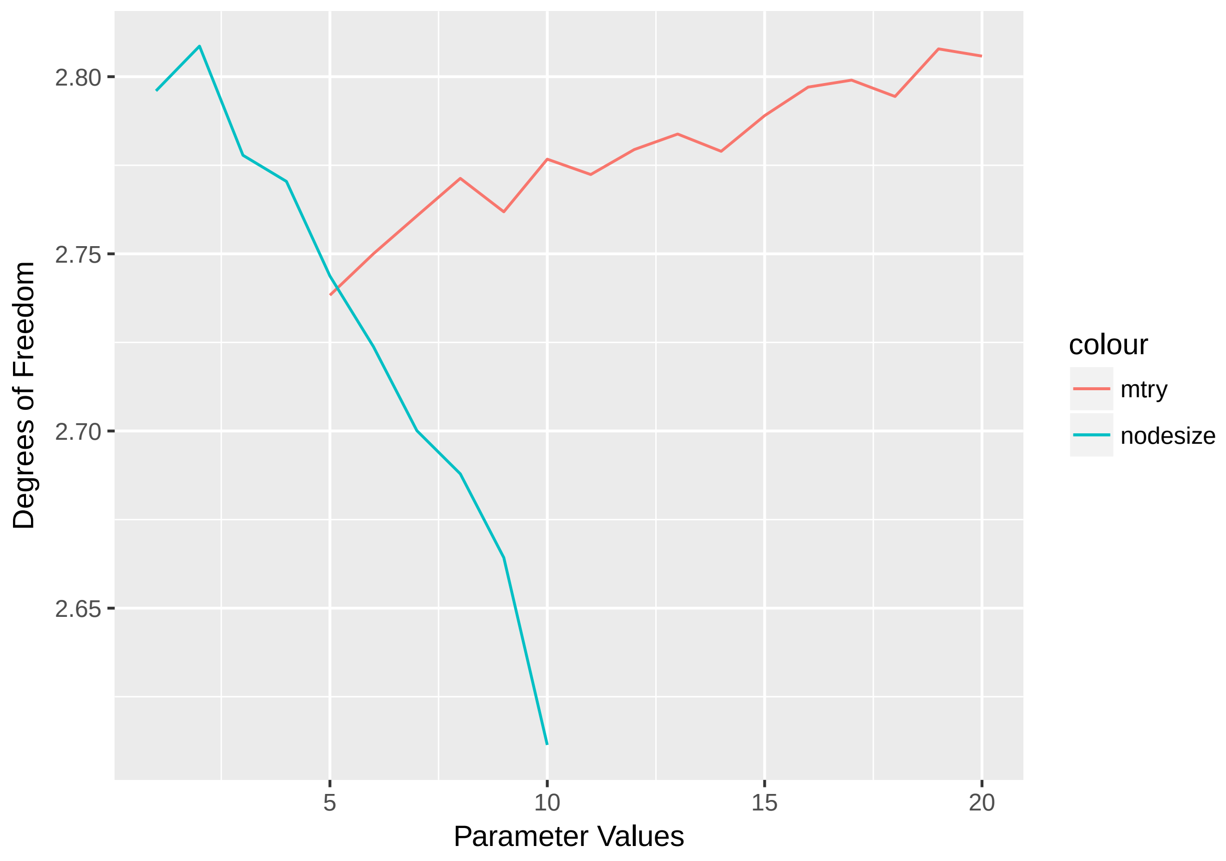
The plot above shows that having 10 knots performs the best among the four knot choices of 3,5, 10,20. The former two knot choices underfit the data, while a knot choice of 20 overfits the data.

Compared to B-splines, the Natural Cubic Splines impose linearity constraints on the extreme values of the covariates. Therefore, it generates more reasonable predictions within these neighborhoods. However, NCS has lower degrees of freedoms than B-spline(Recall that the DF of B-spline is #Knots +4, while that of NCS is #Knots). Therefore, B-spline may perform better than NCS on highly nonlinear data.

Q3 a):Calculate the DF of random forest

After generating the dataset, fit the `randomForest` model on different parameter combinations. `mtry` is iterated over 5 to 20, while `nodesize` is iterated through 1 to 10. The degrees of freedom of each iteration(model) is calculated by `sum(diag(cov(y, fitted)))`

```
a=5:20
deg_mtry = rep(0,length(a))
for (i in seq_along(a)){
  rf = randomForest(y~., mtry =a[i], data = df2)
  fitted = predict(rf,X)
  deg_mtry[i] =  sum(diag(cov(y, fitted)))
}

b= 1:10
deg_ns = rep(0,length(b))
for (i in seq_along(b)){
  rf = randomForest(y~., nodesize =b[i], data = df2)
  fitted = predict(rf,X)
  deg_ns[i] =  sum(diag(cov(y, fitted)))
}
```



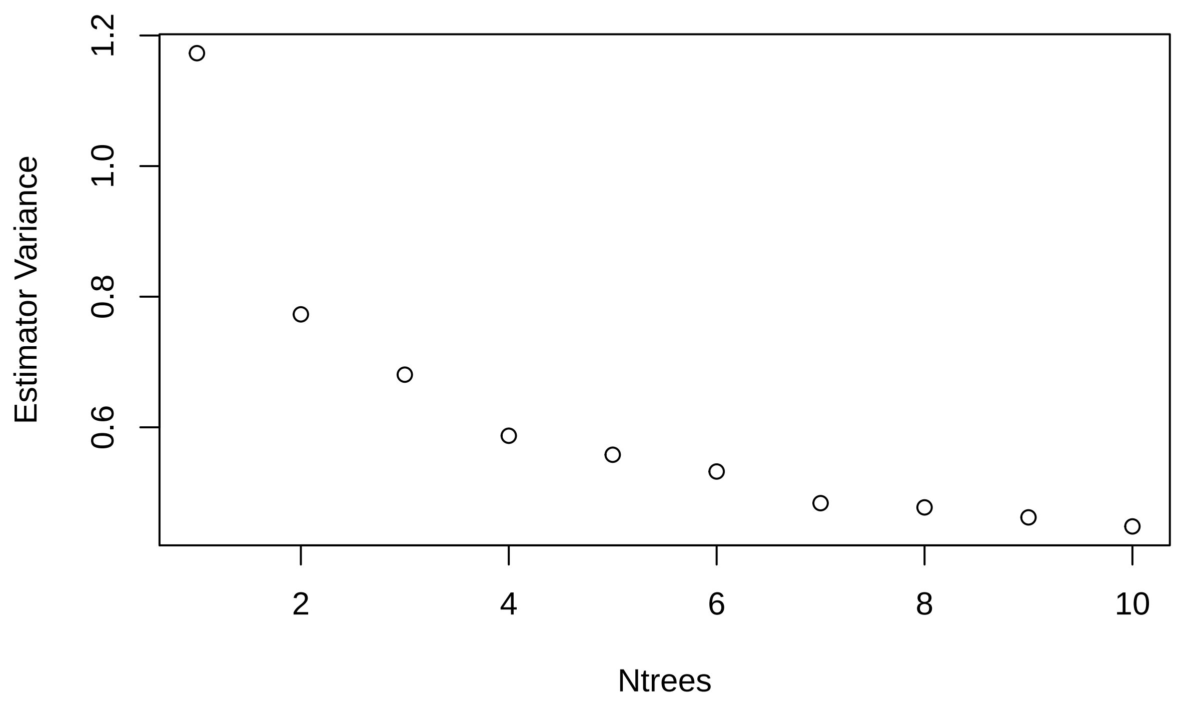
The two lines above show that degrees of freedom increases with `mtry` but decreases with `nodesize`. In other words, lower `mtry` and bigger `nodesize` makes the model less flexible, reduces model variance at the cost of greater bias.

Q3 b): Estimate Estimator Variance.

In order to calculate the estimator variance, I will run the data generating process 20 times to get 20 "batches" of data. This will also generate the same batches of fitted values, and I will calculate the fitted values' variance. For example , I will run the `randomforest` on 1:10 `ntrees`. For each `ntrees` value, there will be a 200*20 matrix. Then the sum of the matrix's by-row variance will be the estimator variance.

```
set.seed(1)
V = matrix(0.5, P, P)
diag(V) = 1
X = as.matrix(mvrnorm(N, mu = rep(0, P), Sigma = V))

ntree = (1:10)
Est_var = rep(0, length(ntree))
for (i in seq_along(ntree)){
  fit_matrix = matrix(0,nrow = 200, ncol = 20)

  for (j in 1:20){
```

```
  y2 = 1 + X %*% Beta + rnorm(N)
  rf = randomForest(y2~., ntree =ntree[i], data = data.frame(y2,X))
  fitted = predict(rf,data.frame(X))
  fit_matrix[,j] =  fitted
  Est_var[i] = sum(rowVars(fit_matrix ))/200
  }}
plot(1:10, Est_var,xlab = "Ntrees", ylab = "Estimator Variance")
```

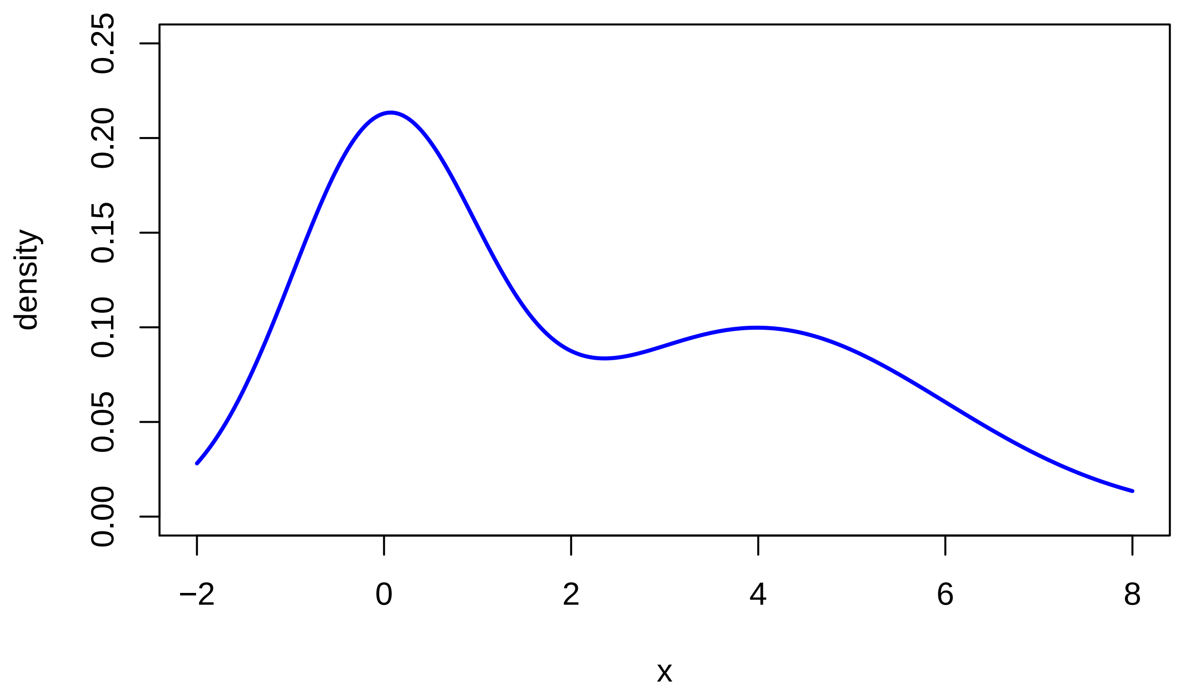


The above plot shows that as the number of trees increase from one to ten, the estimator variance decreases.

Q4. Write own code for Gaussian Kernel Density Estimator. The code should include a tuning parameter for the bandwidth.

Use the following code to generate a toy data, with two modes at 0 and 4 and two variances at 1 and 2.

```
#toy data generation
set.seed(1)
n=1000
x = c(rnorm(n), rnorm(n, 4, 2))
grid = seq(-2, 8, 0.01)
plot(grid, 0.5*dnorm(grid, 0, 1) + 0.5* dnorm(grid, 4, 2) , type = "l", lwd = 2, col = "blue", xlab = "
```

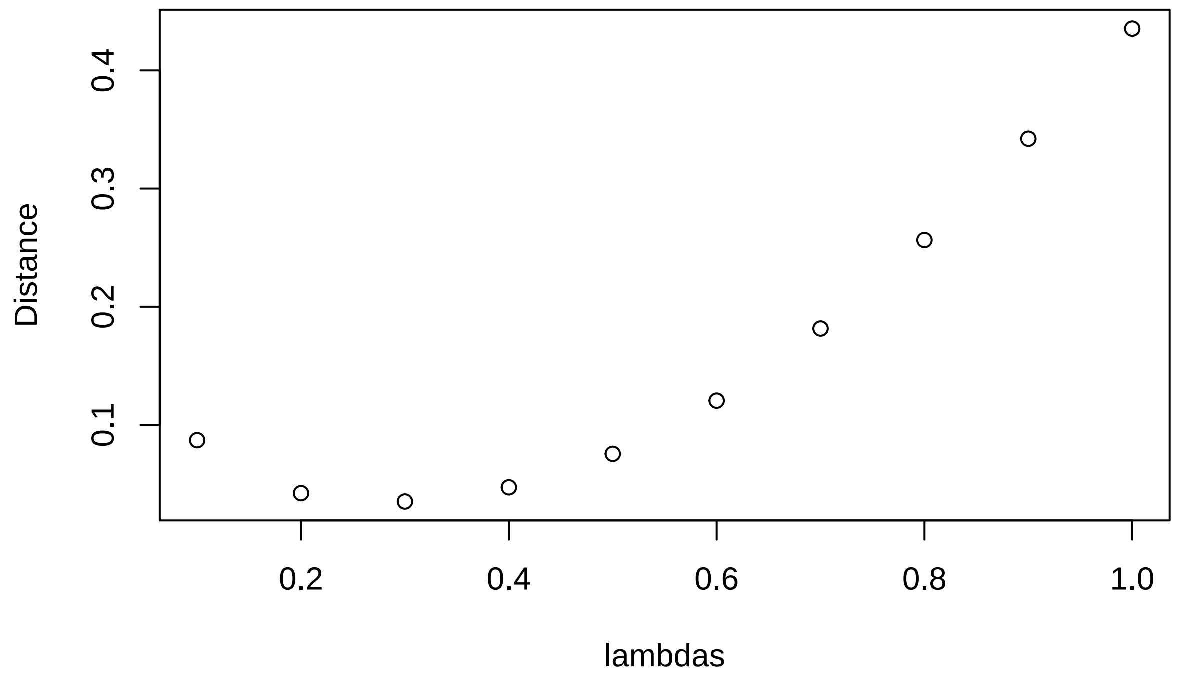


Define a function with two parameters `x`, and `lambda`. The parameter `x` is one observed sample of the underlying density function. `lambda` is a chosen bandwidth.

```
KernelEst = function(x,lambda){

den = matrix(NA, length(x), length(grid))
for (i in 1:length(x))
{
  den[i, ] =   dnorm((grid - x[i])/lambda)/length(x)/lambda
}
fit_den = colSums(den)
return(fit_den)
}
```

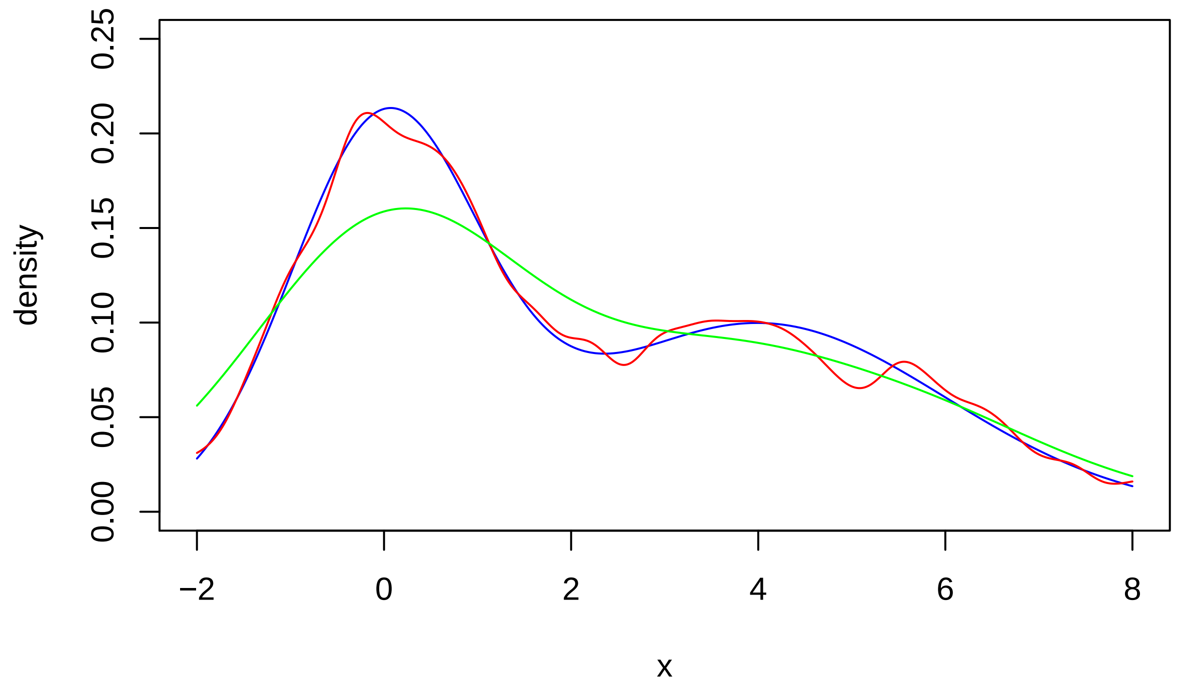
Use the following code to tune over bandwidths 0.1 to 1. The best bandwidth is the one that generates a density distribution with the smallest RSS to the true density.

```
#Use a tuning process to choose the bandwidth

True_den= 0.5*dnorm(grid, 0, 1) + 0.5* dnorm(grid, 4, 2)
lambdas = (1:10)/10
Distance = rep(0, length(lambdas))
for (i in seq_along(lambdas)){
dist = sum((True_den - KernelEst(x,lambdas[i]))^2)
Distance[i] = dist
}
plot(lambdas, Distance)
```



The tuning process shows that the optimal bandwidth is `0.3`, where the fitted kernel has the lowest RSS to the true underlying density. Additionally, the plot below explores the boundary effect of the choice of bandwidth `lambda`.The blue line is the true underlying density, the green line corresponds to `lambda = 1`, while the red line corresponds to `lambda = 0.2`. The plot shows that bigger bandwidth increases the bias of the kernel estimator on boundary values of x.

```
plot(grid, 0.5*dnorm(grid, 0, 1) + 0.5* dnorm(grid, 4, 2) , type = "l", lwd = 1, col = "blue", xlab = "
lines(grid, KernelEst(x,0.2),type = "l", lwd = 1, col = "red")
lines(grid, KernelEst(x,1),type = "l", lwd = 1, col = "green")
```

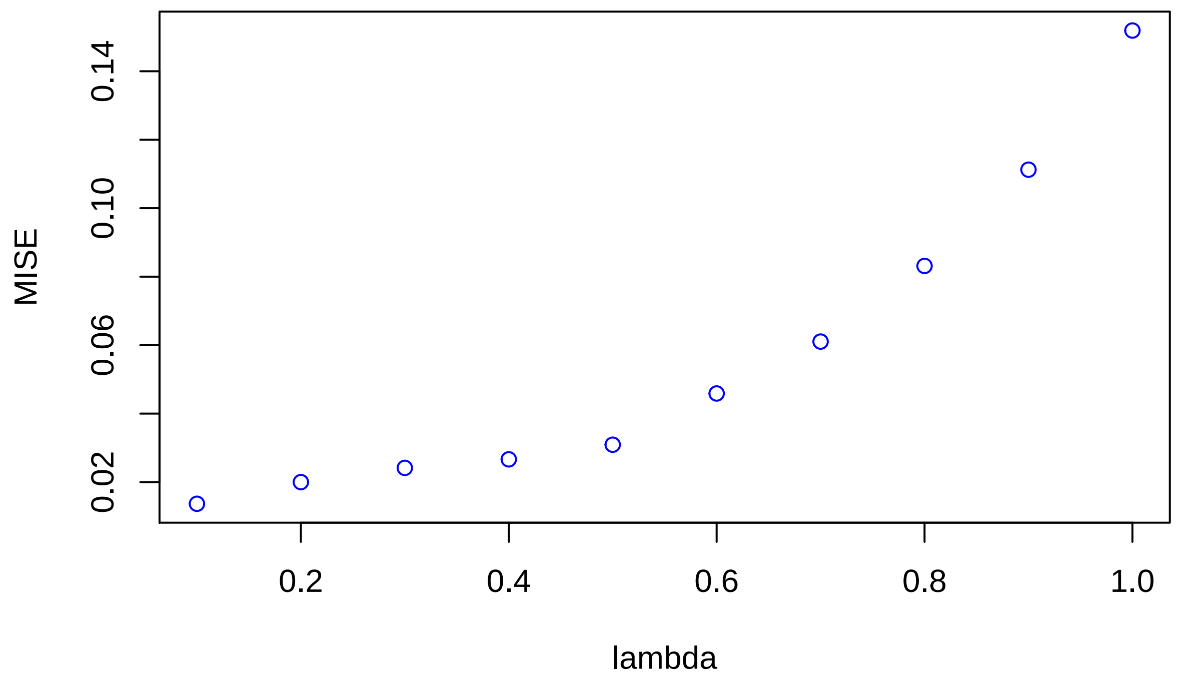


Q4 b: estimate the bias of kernel density estimator

Use the following codes to calculate MISE. For each choice of `lambda`, generate a new set of X and estimate the Gaussian density of the original grid using this new set of x and `lambda`. Then acquire MISE by the follwing formula:

$$\frac{1}{n}\sum_i E_{\hat{f}}(f(\hat{x}_i) - f(x_i))^2$$

```r
lambdas = (1:10)/10
mise = rep(0, length(lambdas))

#trueden = function(x){0.5*dnorm(x, 0, 1) + 0.5* dnorm(x, 4, 2)}

fit_matrix = matrix(0,nrow = 1001, ncol = length(lambdas))
for (i in seq_along(lambdas)){

  #for(j in 1:20){
  x = c(rnorm(n), rnorm(n, 4, 2))

  fit_matrix[,i]  = (KernelEst(x,lambdas[i]) - True_den)^2

  mise[i] = sum(rowmeans(fit_matrix))
}
plot(lambdas, mise,  col = "blue", xlab = "lambda", ylab = "MISE")
```



The result above shows that as lambda increases, the MISE increases too. This agrees with the findings in part a) of the question. As `lambda` increases, the density estimation takes points further into account. This would reduce the estimator variance at the cost of greater bias.