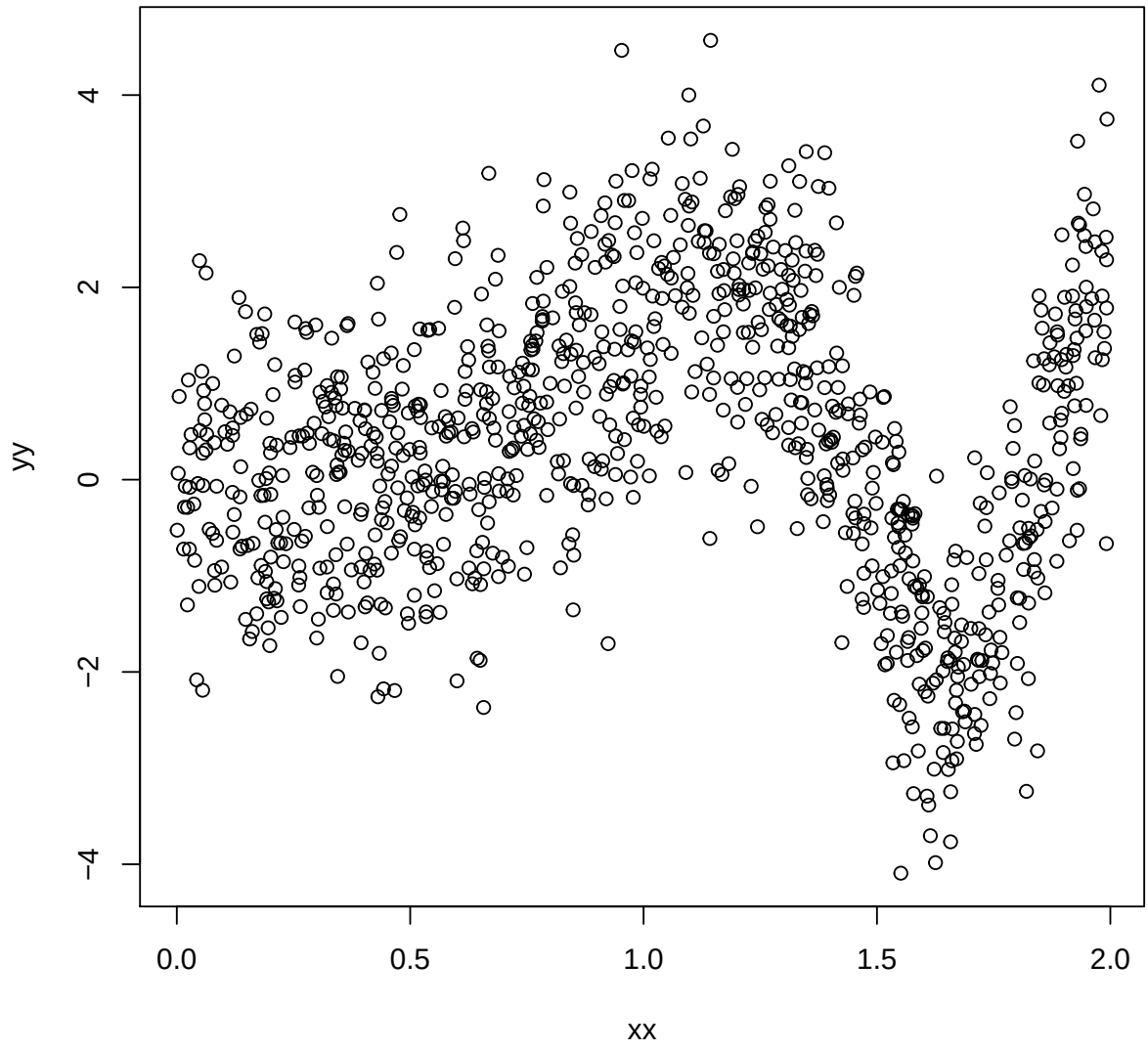
Part 1

Import data

```
d5 = read.csv(file ='/Users/liuxin/Documents/graduate/297B/dat-20160413.csv', header = T)
plot(d5)
```



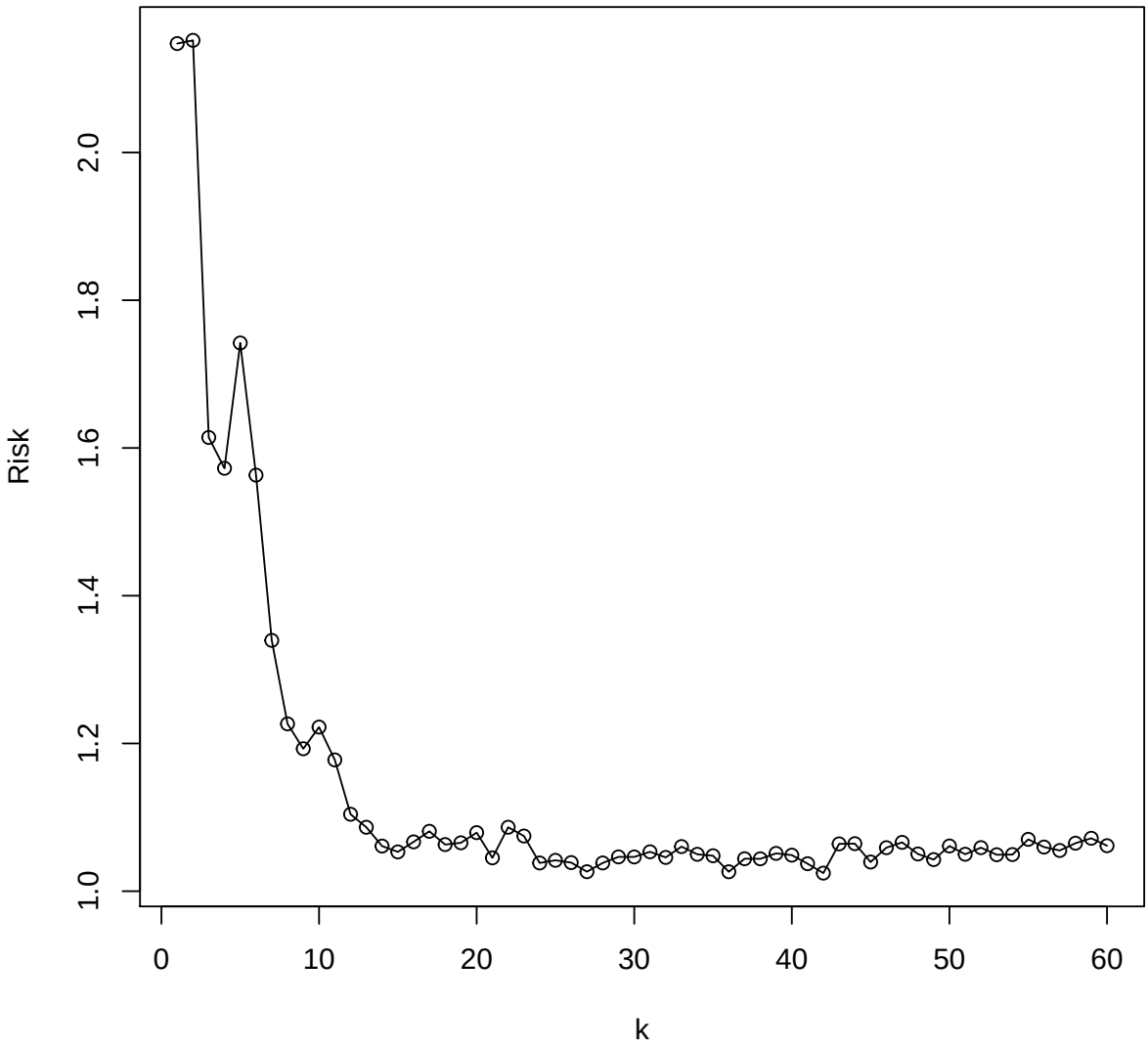

```
attach(d5)
```

Define the regressogram function

```
regressogram = function(x,y,left,right,k){
### assumes the data are on the interval [left,right]
n = length(x)
B = seq(left,right,length=k+1)
WhichBin = findInterval(x,B)
N = tabulate(WhichBin)
m.hat = rep(0,k+1)
for(j in 1:k){
if(N[j]>0)m.hat[j] = mean(y[WhichBin == j])
}
RSS=0
for (i in 1:n){
  RSS=RSS+(y[i]-m.hat[WhichBin[i]])^2
}
variance_estimate=RSS/(n-k)
risk=RSS/n+2*variance_estimate*k/n
return(risk)
}
```

Plot the estimated risk when k varies from 1 to 60

```
Risk=rep(0,60)
for (k in 1:60){
  Risk[k]=regressogram(xx,yy,min(xx),max(xx),k)
}
k=seq(from=1,to=60)
r=cbind(k,Risk)
plot(r)
lines(r)
```



From the plot, we can observe that the k around 10 will perform a good trade-off.

Part 2

Define the function of computing the estimated covariances in Efron's procedure.

```
bootstrap_cov = function(x, y_fitted, residual, B, func, parameter){
  N = length(x)
  y_hat = c()
  miu_hat = c()
  cov = rep(0, N)
```

```
for(i in 1:B){
    newindex=sample(1:N,N,replace = TRUE)
    newy = y_fitted[newindex]
    y_hat = cbind(y_hat, newy)
    y_new_fitted = func(x, newy, parameter)
    miu_hat = cbind(miu_hat, y_new_fitted)
  }
  mean_y_hat = apply(y_hat, 1, mean)
  for(i in 1:N){
    cov[i] = (miu_hat[i,] %*% (y_hat[i,] - mean_y_hat[i]))/(B-1)
  }
  return(cov)
}
```

For linear splines,define the fit function.

```
library(splines)
fit_linear_spline = function(x, y, nknots){
  seq_knots = seq(0,1,1/(nknots+1))
  seq_knots = seq_knots[2:(nknots+1)]
  K = quantile(x, seq_knots, type=1)
  model = lm(y ~ bs(x, degree=1, knots=K))
  return(model$fitted.values)
}
```

use the linear spline function to fit our model

```
y_fitted_linear_spline = fit_linear_spline(xx,yy,50)
residual_linear_spline = yy - y_fitted_linear_spline
```

We know from the paper that the Err estimate is

$$\hat{Err} = \text{err} + 2\sum_{i=1}^{n} \hat{cov}_i$$

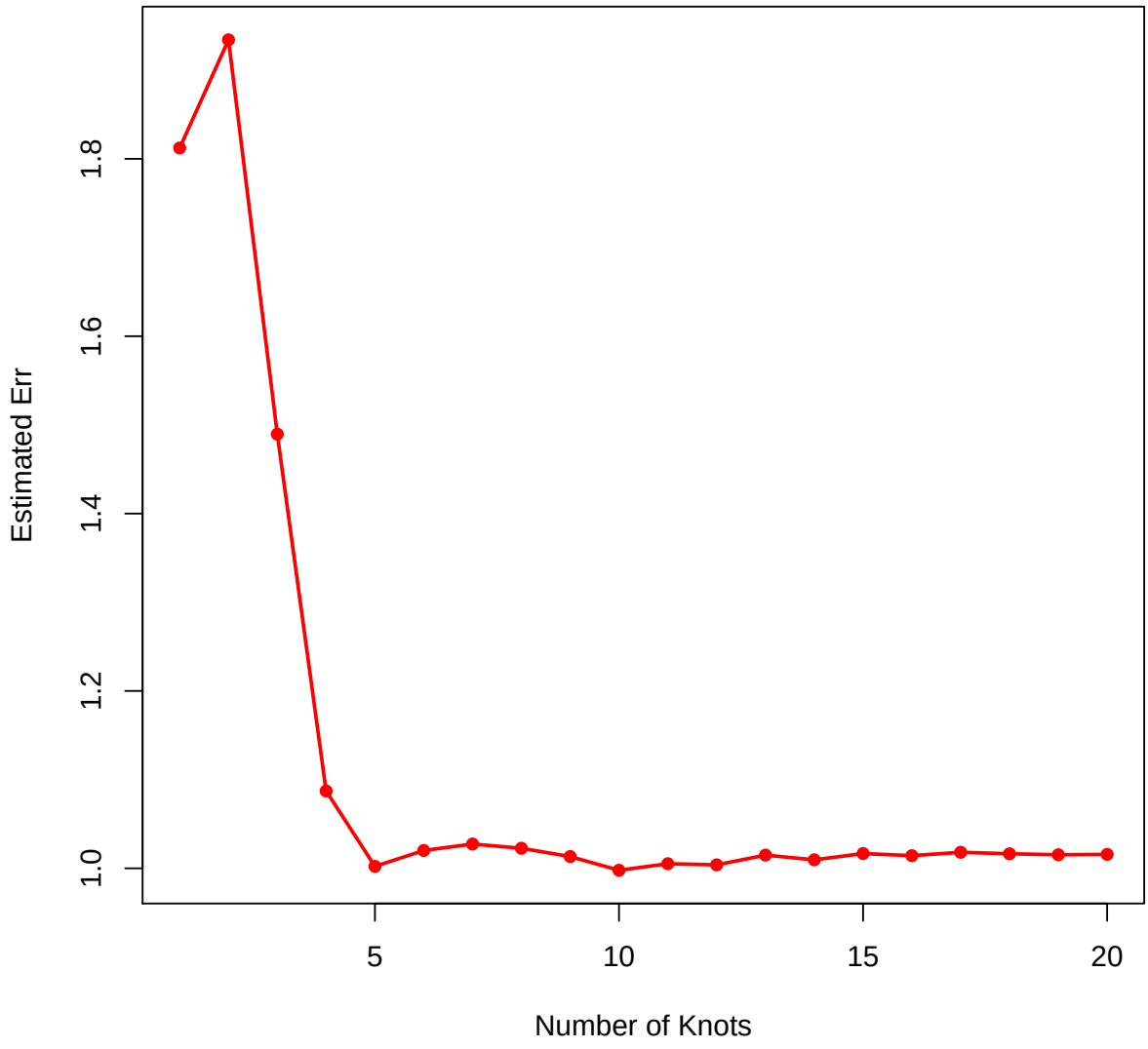
We computed the Err estimate for linear spline.

```
risk_linear_spline = function(x, y, number_knots){
  y_fitted = fit_linear_spline(x, y, number_knots)
  residual = y - y_fitted
  bootcovs = bootstrap_cov(x, y_fitted_linear_spline, residual_linear_spline,
                       100, fit_linear_spline, number_knots)
  risk = sum((residual)^2) + 2*sum(bootcovs)
  return(risk/length(x))
}
```

Let we plot the estimated Err with number of plots and choose the optimal knots using Efron's procedure.

```
knots = seq(1,20)
risks = rep(0,length(knots))
for(i in 1:length(knots)){
```

```
  risks[i] = risk_linear_spline(xx, yy, knots[i])
}
plot(knots, risks, type='o', pch=16, col='red', lwd=2, xlab="Number of Knots", ylab="Estimated Err")
```




```
optimal_knots = knots[which.min(risks)]
```

The number of knots that minimizes the estimated Err is 10, but when we observe the plot, the Err doesn't show a significantly decrease from number 5. As a result, to aviod overfitting, 5 knots is more appropriate.

Part 3

Define the kernel regression function

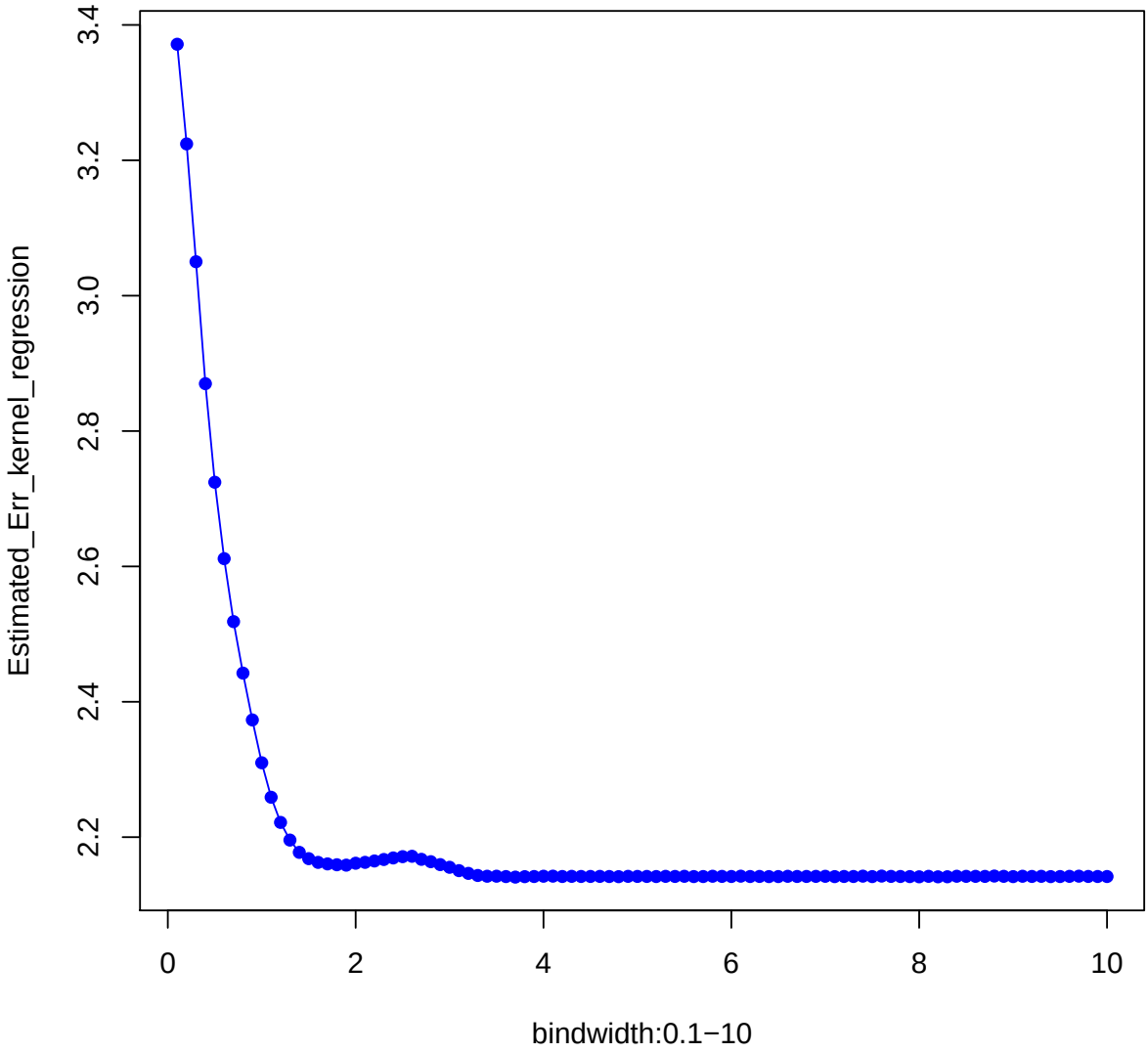
```
fit_kernel_regression=function(x,y,bindwidth){
    model=ksmooth(x, y, kernel = c("box", "normal"), bandwidth=bindwidth)
    fitted_values=model$y
    return (fitted_values)
}
y_fitted_kernel_regression = fit_kernel_regression(xx,yy,0.3)
residual_kernel_regression = yy - y_fitted_kernel_regression
```

Define the function to return the estimated Err for kernel regression.

```
Err_kernel_regression = function(x, y, bw){
  y_fitted = fit_kernel_regression(x, y, bw)
  residual = y - y_fitted
  bootcovs = bootstrap_cov(x, y_fitted_kernel_regression, residual_kernel_regression,
                      100, fit_kernel_regression, bw)
  Err = sum(residual^2) + 2*sum(bootcovs)
  return(Err/length(x))
}
```

To find the optimal kernel regression model.

```
bindwidth = seq(0,10,by=0.1)
Estimated_Err_kernel_regression = rep(0, length(bindwidth))
for(i in 1:length(bindwidth)){
  Estimated_Err_kernel_regression[i] = Err_kernel_regression(xx, yy, bindwidth[i])
}
plot(bindwidth, Estimated_Err_kernel_regression, type='o', pch=16, col='blue', xlim=c(0.1,10),
     xlab="bindwidth:0.1-10", ylab="Estimated_Err_kernel_regression")
```




```
optimal_kernel_bindwidth = bindwidth[which.min(Estimated_Err_kernel_regression)]
optimal_kernel_bindwidth

## [1] 3.7
```

The optimal bindwidth is 3.7 From the plot, the bindwidth around 1.7 perform a good trade-off.

Part 4

For local linear model, there is a function named locfit in R, the paratmeter 'nn' which refers to the proportion of data that is closest to the given postion to fit the curve.It varies from 0 to 1. Define a function to diretly

return the prediction.

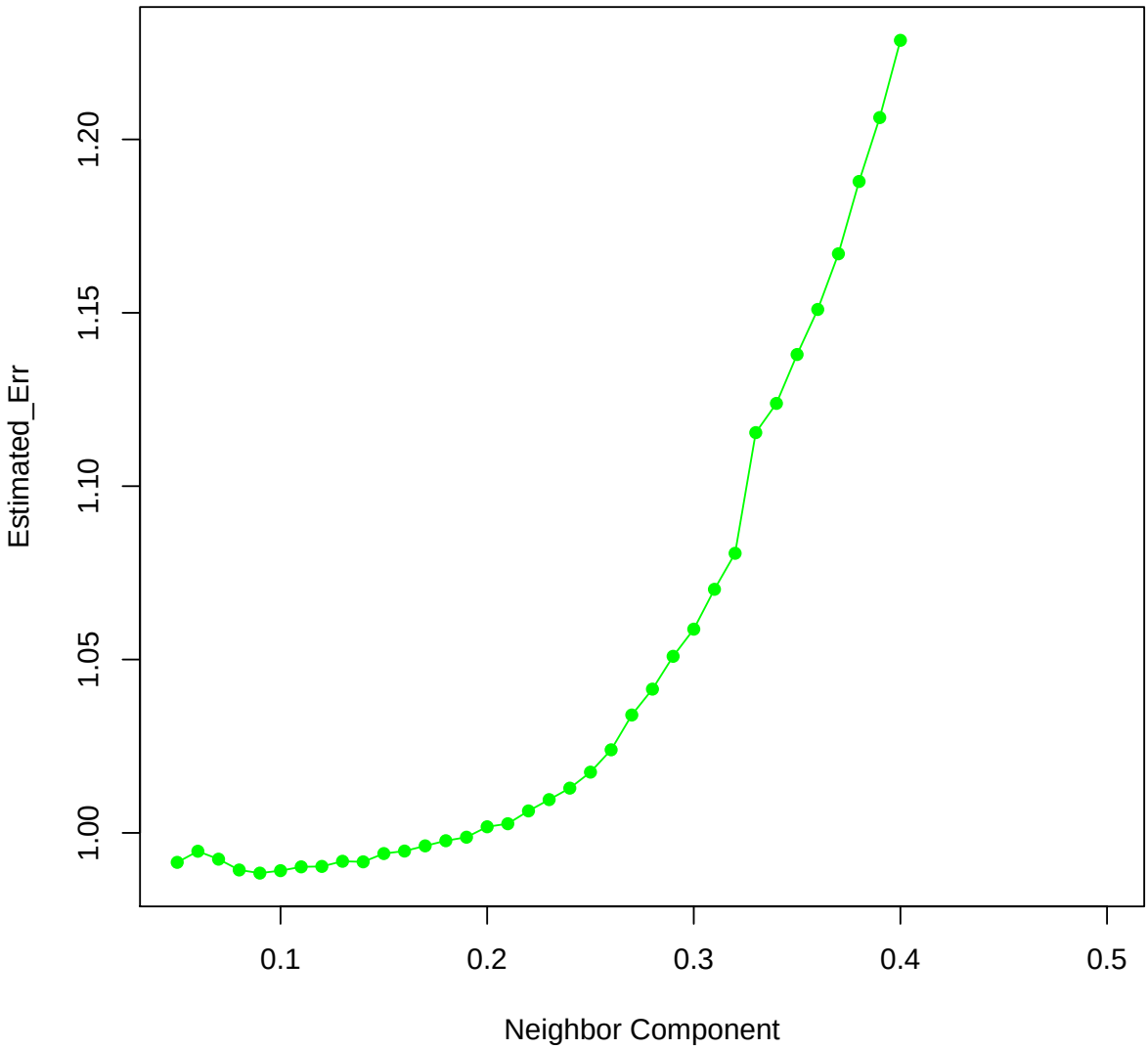
```
library(locfit)

## locfit 1.5-9.1 2013-03-22

# Local linear
fit_local_linear = function(x, y, neighbor){
  model = locfit(y~lp(x,nn=neighbor,deg=1))
  fitted_values = predict(model,newdata=x)
  return(fitted_values)
}
y_fitted_local_linear = fit_local_linear(xx,yy,0.3)
residual_local_linear = yy - y_fitted_local_linear
```

Define the function to return the estimated Err for local linear model.

```
Err_local_linear = function(x, y, propo){
  y_fitted = fit_local_linear(x, y, propo)
  residual = y - y_fitted
  bootcovs = bootstrap_cov(x, y_fitted_local_linear, residual_local_linear,
                      100, fit_local_linear, propo)
  Err = sum(residual^2) + 2*sum(bootcovs)
  return(Err/length(x))
}
```

To find the optimal lowess model

```
propo = c(5:40)/100
Estimated_Err = rep(0, length(propo))
for(i in 1:length(propo)){
  Estimated_Err[i] = Err_local_linear(xx, yy, propo[i])
}
plot(propo, Estimated_Err, type='o', pch=16, col='green', xlim=c(0.05,0.5),
     xlab="Neighbor Component", ylab="Estimated_Err")
```




```
optimal_neighbors_component = propo[which.min(Estimated_Err)]
```

the optimal number is 0.09

Part 5

```
test_linear_spline = risk_linear_spline(xx, yy,optimal_knots)
test_kernel_regression = Err_kernel_regression(xx, yy,optimal_kernel_bindwidth)
test_lowess = Err_local_linear(xx, yy,optimal_neighbors_component)
```

Part 6

Applied cross validation:

```
set.seed(2020)
N = length(xx)
```

For Linear spline:

```
risk_linear_spline_cv = function(x, y, nknots){
  data=cbind(x,y)
  data=data[sample(nrow(data)),]
  folds=cut(seq(1,nrow(data)),breaks=7,labels=FALSE)
  err = rep(0,7)
  for(i in 1:7){
    testindexes=which(folds==i,arr.ind=TRUE)
    testdata=data[testindexes,]
    traindata=data[-testindexes,]
    qknots = seq(0,1,1/(nknots+1))
    qknots = qknots[2:(nknots+1)]
    K = quantile(x, qknots, type=1)
    model = lm(traindata[,2] ~ bs(traindata[,1], degree=1, knots=K))

    y_pred = predict(model, data.frame(testdata[,1]))
    err[i] = sum((testdata[,2]-y_pred)^2)/length(testdata[,1])
  }
  return(mean(err))
}

knots = c(1:30)
risks = rep(0,length(knots))
for(i in 1:length(knots)){
  risks[i] = risk_linear_spline_cv(xx, yy, knots[i])
}
optimal_knots_cv = knots[which.min(risks)]
```

the optimal number for both method is 10. For kernel:

```
risk_kernel_regression_cv = function(x, y, bindw){
  err = rep(0,7)
  data=cbind(x,y)
  data=data[sample(nrow(data)),]
  folds=cut(seq(1,nrow(data)),breaks=7,labels=FALSE)
  for(i in 1:7){
    testindexes=which(folds==i,arr.ind=TRUE)
    testdata=data[testindexes,]
    traindata=data[-testindexes,]

    model=ksmooth(traindata[,1], traindata[,2], kernel = c("box", "normal"), bandwidth=bindw)

    y_pred = model$y
    err[i] = sum((testdata[,2]-y_pred)^2)/length(testdata[,1])
```

```
  }
  return(mean(err))
}

bindwidth_cv = c(0.1:10)
risks_kernel_cv = rep(0,length(bindwidth_cv))
for(i in 1:length(bindwidth_cv)){
  risks_kernel_cv[i] = risk_kernel_regression_cv(xx, yy, bindwidth_cv[i])
}
optimal_bindwidth_cv = bindwidth_cv[which.min(risks_kernel_cv)]
```

the optimal number for Efron procedure is 7.1, for cv is 4.1

```
risk_local_linear_cv = function(x, y, neighbor){
  err = rep(0,7)
  data=cbind(x,y)
  data=data[sample(nrow(data)),]
  folds=cut(seq(1,nrow(data)),breaks=7,labels=FALSE)
  for(i in 1:7){
    testindexes=which(folds==i,arr.ind=TRUE)
    testdata=data[testindexes,]
    traindata=data[-testindexes,]

    model = locfit(traindata[,2]~lp(traindata[,1],nn=neighbor,deg=1))

    y_pred = predict(model, newdata=testdata[,1])
    err[i] = sum((testdata[,2]-y_pred)^2)/length(testdata[,1])
  }
  return(mean(err))
}


neighbors = c(5:40)/100
risks = rep(0,length(neighbors))
for(i in 1:length(neighbors)){
  risks[i] = risk_local_linear_cv(xx, yy, neighbors[i])
}
optimal_neighbors_component_cv = neighbors[which.min(risks)]
```

the optimal number using Efron is 0.13, using cv is 0.14