```
require("LaplacesDemon")
require("extrafont")
par(family = 'Times New Roman')
setwd("/Users/mikhailgaerlan/Box Sync/Education/UC Davis/2016-2017 Spring/STA 243 Computational
Statistics/Assignments/Assignment 5")
rm(list=ls())
set.seed(0518)

# Parameter Values
p = 3
n = 200
k = 30
export = T

phi = function(u){
  return(exp(-u^2/2)/sqrt(2*pi))
}

f = function(x){
  return(1.5*phi((x-0.35)/0.15)-phi((x-0.8)/0.04))
}

modelf = function(x,knots,betas){
  sums = 0
  for(i in 1:(p+1)){
    sums = sums + betas[i]*x^(i-1)
  }
  for(i in 1:k){
    sums = sums + betas[p+1+i]*(max((x-knots[i]),0))^p
  }
  return(sums)
}

designmatrix = function(x,p,k){
  n = length(x)
  X = array(0,c(n,p+1+k))
  for(i in 1:(p+1)){
    for(j in 1:n){
      X[j,i] = x[j]^(i-1)
    }
  }
  for(i in 1:k){
    for(j in 1:n){
      X[j,p+1+i] = (max(c((x[j]-knots[i]),0)))^p
    }
  }
  return(X)
}

for(o in 1:4){
  if (o == 1){
    print("Cross Validation")
    method = "CV"
  } else if(o == 2){
    print("Generalized Cross Validation")
    method = "GCV"
  } else if(o == 3){
    print("AICc")
    method = "AICc"
  } else if(o == 4){
    print("Risk")
    method = "Risk"
  }
  for(l in 1:4){
    if (l == 1){
      print("Noise Level")
      functions = "Noise Level"
    } else if(l == 2){
      print("Design Density")
      functions = "Design Density"
    } else if(l == 3){
      print("Spatial Variation")
      functions = "Spatial Variation"
    } else if(l == 4){
      print("Variance Function")
      functions = "Variance Function"
    }
    for(m in 1:6){
      if (export) file_name =
paste("Graphs/",toString(o),"/",toString(l),"/assignment5_a_",toString(o),"_",toString(l),"_",toString(m),".pdf",sep=""
```

```
      if (export) pdf(file = file_name,width = 4,height=3.5,family="Times New Roman")
      # Data generation
      xtest = seq(0,1,0.001)
      if(l==1){
        # Noise Level
        f = function(x){
          return(1.5*phi((x-0.35)/0.15)-phi((x-0.8)/0.04))
        }
        x = (1:200-0.5)/n
        knots = min(x) + 1:k*(max(x)-min(x))/(k+1)
        sigma = 0.02 + 0.04*(m-1)^2
        epsilon = rnorm(n,0,1)
        truef = f(x)
        y = truef + sigma*epsilon
      }else if(l==2){
        # Design Density
        f = function(x){
          return(1.5*phi((x-0.35)/0.15)-phi((x-0.8)/0.04))
        }
        x = qbeta(runif(n,0,1),(m+4)/5,(11-m)/5)
        knots = min(x) + 1:k*(max(x)-min(x))/(k+1)
        sigma = 0.1
        epsilon = rnorm(n,0,1)
        truef = f(x)
        y = truef + sigma*epsilon
        data = cbind(x,y,truef)
        sorteddata = data[order(data[,1]),]
        x = sorteddata[,1]
        y = sorteddata[,2]
        truef = sorteddata[,3]
      }else if(l==3){
        # Spatial Variation
        f = function(x){
          return(sqrt(x*(1-x))*sin((2*pi*(1+2^((9-4*m)/5)))/(x+2^((9-4*m)/5))))
        }
        x = (1:200-0.5)/n
        knots = min(x) + 1:k*(max(x)-min(x))/(k+1)
        sigma = 0.2
        epsilon = rnorm(n,0,1)
        truef = f(x)
        y = truef + sigma*epsilon
      }else if(l==4){
        # Variance Function
        f = function(x){
          return(1.5*phi((x-0.35)/0.15)-phi((x-0.8)/0.04))
        }
        x = (1:200-0.5)/n
        knots = min(x) + 1:k*(max(x)-min(x))/(k+1)
        sigma = sqrt((0.15*(1+0.4*(2*m-7)*(x-0.5)))^2)
        epsilon = rnorm(n,0,1)
        truef = f(x)
        y = truef + sigma*epsilon
      }
      plot_name = bquote(paste(.(method),", ",.(functions),",  ",italic(j)," = ",.(m)))

plot(x,y,pch=20,main=plot_name,cex=0.15,cex.main=0.8,cex.lab=0.8,cex.axis=0.8,xlab=expression(italic(x)),ylab=expressio

      lines(xtest,f(xtest),col="black")

      # Spline Regression
      X = designmatrix(x,p,k)
    betas = solve(t(X) %*% X,tol=1e-50) %*% t(X) %*% y
      fhat = X %*% betas
      ytest = 1:(length(xtest))
      for(nn in 1:(length(xtest))){
        ytest[nn] = modelf(xtest[nn],knots,betas)
      }
      lines(xtest,ytest,col="blue")

      # Penalized Spline Regression
      if(o==1){
        # Cross Validation
        minimizer = function(lambda,x,y){
          X = designmatrix(x,p,k)

          D = diag(c(0*(1:(p+1)),0*(1:k)+1))
          hlambda = X %*% (solve((t(X) %*% X)+lambda*D,tol=1e-50) %*% t(X))
          fhatlambda = hlambda %*% y
          hii = diag(hlambda)

          return(sum(((y-fhatlambda)/(1-hii))^2))
```

```
        }
      } else if(o == 2){
        # Generalized Cross Validation
        minimizer = function(lambda,x,y){
          X = designmatrix(x,p,k)

          n = length(x)
          D = diag(c(0*(1:(p+1)),0*(1:k)+1))
          hlambda = X %*% (solve(t(X) %*% X+lambda*D,tol=1e-50) %*% t(X))
          fhatlambda = hlambda %*% y
          hii = sum(diag(hlambda))/n

          return(sum(((y-fhatlambda)/(1-hii))^2))
        }
      } else if(o == 3){
        # AICc
        minimizer = function(lambda,x,y){
          X = designmatrix(x,p,k)

          D = diag(c(0*(1:(p+1)),0*(1:k)+1))
          hlambda = X %*% (solve(t(X) %*% X + lambda*D,tol=1e-50) %*% t(X))
          fhatlambda = hlambda %*% y

          n = length(x)
          tr = sum(diag(hlambda))
          norm = sum((y-fhatlambda)^2)
          return(log(norm)+2*(tr+1)/(n-tr-2))
        }
      }else if(o == 4){
        # Risk
        minimizer = function(lambda,x,y){
          X = designmatrix(x,p,k)

          D = diag(c(0*(1:(p+1)),0*(1:k)+1))
          hlambda = X %*% (solve(t(X) %*% X + lambda*D,tol=1e-50) %*% t(X))
          fhatlambda = hlambda %*% y

          n = length(x)
          norm = sum((fhatlambda-(hlambda %*% fhatlambda))^2)
          sigma2 = sum((y-fhatlambda)^2)/(n-1)

          return((norm+sigma2*(sum(diag(hlambda %*% t(hlambda)))-2*sum(diag(hlambda))+n)))
        }
      }

      lambda = 1e-7
      minlambda = nlm(minimizer,c(lambda),x = x, y = y,steptol = 1e-20)
      lambda = minlambda$estimate
      D = diag(c(0*(1:(p+1)),0*(1:k)+1))
      betalambda = (solve(t(X) %*% X + lambda*D,tol=1e-50) %*% t(X)) %*% y
      hlambda = X %*% betalambda
      fhatlambda = hlambda %*% y
      ytest = 1:(length(xtest))
      for(nn in 1:(length(xtest))){
        ytest[nn] = modelf(xtest[nn],knots,betalambda)
      }
      print(paste("j =",m,", lambda =",lambda, ", min =",minlambda$minimum))
      lines(xtest,ytest,col="red")
      if (export) dev.off()
    }
  }
}
```