Project 3

Robin Lin

2022-11-03

## Code to smooth with basis expansions and penalties.  
  
## The y values consist of a smoothing function of x, as well as the error terms.  
  
## The smoothing function is a linear combination of k evenly-spaced B-splined  
## basis functions, and the coefficients are to be estimated.  
  
## In order to avoid over-fitting, some smoothing penalty is imposed, making each   
## coefficient to vary smoothly from its neighbouring ones.  
  
## The model is hence estimated by penalised least squares, and the smoothing  
## parameter is chosen in order to minimise the generalised cross validation  
## (GCV) criterion.  
  
## This project aims at fitting P-splines to the x, y data, and selecting  
## the smoothing parameter based on GCV criterion. Also, some details of the   
## method function are needed to print out. Predictions are to be made, and   
## some plots are to be sketched.

pspline <- function(x, y, k = 20, logsp = c(-5, 5), bord = 3, pord = 2, ngrid = 100){  
   
 ## This function aims at performing the best fit to the smoothing function, and  
 ## print out the details of the model.  
   
 ## x, y are data points, and k is the number of evenly-spaced B-splined basis   
 ## functions. 'logsp' represents the boundaries of smoothing functions in log  
 ## scale. 'bord' is the B-spline order, and 'pord' is the penalty order of   
 ## difference. 'ngrid' is the number of smoothing parameters to try.  
   
 dk <- diff(range(x)) / (k - bord) # Knot spacing.  
 knots <- seq(min(x) - dk \* bord, by = dk, length = k + bord + 1) # Knots   
 # generating.  
 X <- splines::splineDesign(knots, x, ord = bord + 1, outer.ok = TRUE) # Data   
 # matrix.  
 D <- diff(diag(k), differences = pord) # Penalisation matrix.  
   
 qrX <- qr(X) # Obtains QR-factorisation of X, i.e., X = QR.  
 Q <- qr.Q(qrX) # Matrix Q.  
 R <- qr.R(qrX) # Matrix R.  
   
 eig <- eigen(t(solve(R)) %\*% t(D) %\*% D %\*% solve(R)) # Obtains eigen-  
 # decomposition of (D \* R ^ (-1)) ^ T \* (D \* R ^ (-1)), i.e.,  
 # (D \* R ^ (-1)) ^ T \* (D \* R ^ (-1)) = U \* \Lambda \* (U ^ T).  
 U <- eig$vectors # Matrix U.  
 Lam <- diag(eig$values) # Matrix \Lambda.  
   
 logSP <- seq(from = logsp[1], to = logsp[2], length.out = ngrid) # Equally   
 # separated smoothing parameters in log scale.  
 SP <- exp(logSP) # Smoothing parameters.  
   
 gcv <- function(lambda){  
   
 ## This function aims at reckoning GCV criterion given a smoothing parameter.  
   
 revised\_Lam <- diag(diag(1 + lambda \* Lam)) # Matrix (I + smoothing\_par \*   
 # \Lambda).  
 A <- solve(R) %\*% U %\*% solve(revised\_Lam) %\*% t(U) # Matrix (R ^ (-1) \* U \*   
 # revised\_Lam ^ (-1) \* U ^ T).  
 coef <- A %\*% (t(Q) %\*% y) # Estimates of coefficients.  
   
 fitted <- X %\*% coef # Fitted Values.  
 edf <- sum(diag(solve(revised\_Lam))) # Effective degrees of freedom, taking   
 # the trace on the inverse of 'revised\_Lam'.  
 sig2 <- sum((y - fitted) ^ 2) / (nrow(X) - edf) # Residual Variance.  
 gcv <- sig2 / (nrow(X) - edf) # GCV criterion.  
 return(gcv)  
 }  
   
 GCV <- as.numeric(lapply(X = SP, FUN = gcv)) # Stores all the GCV values.  
 sp <- SP[which(GCV == min(GCV))] # Selects the smoothing parameter with the   
 # smallest GCV criterion.  
  
 revised\_Lam <- diag(diag(1 + sp \* Lam)) # Matrix (I + smoothing\_par \*   
 # \Lambda).  
 A <- solve(R) %\*% U %\*% solve(revised\_Lam) %\*% t(U) # Matrix (R ^ (-1) \* U \*   
 # revised\_Lam ^ (-1) \* U ^ T).  
 coef <- A %\*% (t(Q) %\*% y) # Estimates of coefficients.  
   
 fitted <- X %\*% coef # Fitted Values.  
 edf <- sum(diag(solve(revised\_Lam))) # Effective degrees of freedom, taking   
 # the trace on the inverse of 'revised\_Lam'.  
 sig2 <- sum((y - fitted) ^ 2) / (nrow(X) - edf) # Residual variance.  
 V <- sig2 \* A %\*% t(solve(R)) # Covariance matrix for the coefficients.  
 r2 <- 1 - (nrow(X) - 1) \* sig2 / sum((y - mean(y)) ^ 2) # Model R-squared.  
 gcv <- sig2 / (nrow(X) - edf) # GCV criterion.  
   
 newlist <- list('x' = x, 'y' = y, 'B\_spline\_Order' = bord, 'Penalty\_Order\_of\_Difference' = pord, 'Number\_of\_Basis\_Functions' = k, 'Knots' = knots, 'Smoothing\_Parameter' = sp, 'Coefficients' = coef, 'Fitted\_Values' = fitted, 'Effective\_Degrees\_of\_Freedom' = edf, 'Residual\_Variance' = sig2, 'Residual\_Std' = sqrt(sig2), 'Cov\_for\_Coefficients' = V, 'R\_Squared' = r2, 'Generalised\_Cross\_Validation' = gcv)   
 # Creates a list containing a bunch of details of the model.  
   
 return(newlist)  
}

print.pspline <- function(m){  
   
 ## This function aims at showing the EDF, k, residual std, r ^ 2, and GCV of the  
 ## model, given the method function 'pspline'.  
   
 cat('Order', m$B\_spline\_Order, 'p-spline with order', m$Penalty\_Order\_of\_Difference, 'penalty', '\n')  
   
 cat('Effective degrees of freedom:', m$Effective\_Degrees\_of\_Freedom, 'Coefficients:', m$Number\_of\_Basis\_Functions, '\n')  
   
 cat('residual std dev:', m$Residual\_Std, 'r-squared:', m$R\_Squared, 'GCV:', m$Generalised\_Cross\_Validation, '\n')  
   
 newlist <- list('gcv' = m$Generalised\_Cross\_Validation, 'edf' = m$Effective\_Degrees\_of\_Freedom, 'r2' = m$R\_Squared)   
 # Stores the values of gcv, edf, and r2.  
  
 invisible(newlist) # Silently returns the new list.  
}

predict.pspline <- function(m, x, se = TRUE){  
   
 ## This function aims at making predictions from the smooth fit given new x   
 ## values within the range of the original data, as well as the method function   
 ## 'pspline'.  
   
 Xp <- splines::splineDesign(m$Knots, x, ord = m$B\_spline\_Order + 1, outer.ok = TRUE)   
 # Data matrix.  
   
 coef <- m$Coefficients # Estimates of coefficients.  
 fitted <- Xp %\*% coef # Fitted Values.  
   
 if(se){ # Standard errors and fitted values are to be returned.  
 V <- m$Cov\_for\_Coefficients # Covariance matrix for the coefficients.  
 std\_error <- rowSums(Xp \* (Xp %\*% V)) ^ .5 # Standard Errors.  
 newlist <- list('fit' = fitted, 'se' = std\_error) # Stores the values of   
 # fitted values and the standard errors.  
 }else{ # Only fitted values are to be returned.  
 newlist <- list('fit' = fitted) # Stores only the values of fitted values.  
 }  
   
 return(newlist)  
}

plot.pspline <- function(m){  
  
 ## This function   
   
 CL <- .95 # Credible Level.  
 CV <- qnorm((1 - CL) / 2, lower.tail = FALSE) # Critical Value.  
 ll <- m$Fitted - CV \* predict.pspline(m, m$x)$se # Lower Confidence Limit.  
 ul <- m$Fitted + CV \* predict.pspline(m, m$x)$se # Upper Confidence Limit.  
   
 ## The First Plot   
   
 plot(m$y ~ m$x, main = 'Smoothing Plot with 95% Credible Intervals', xlab = 'x', ylab = 'y', type = 'p', col = 'white')  
 # Sketches an empty graph.  
   
 lines(ll ~ m$x, col = 'blue', lwd = 3) # Sketches a line of lower confidence   
 # limit.  
 lines(ul ~ m$x, col = 'blue', lwd = 3) # Sketches a line of upper confidence   
 # limit.  
 polygon(c(m$x, rev(m$x)), c(ll, rev(ul)), col = 'grey', border = NA)  
 # Sketches a shaded area representing the 95% credible intervals.  
 lines(m$Fitted ~ m$x, col = 'red', lwd = 3) # Sketches the smoothing line.  
 points(m$y ~ m$x) # Sketches the data points.  
 legend('bottomright', legend = c('Smoothing Curve', '95% Credible Interval Bounds'), lwd = 3, col = c('red', 'blue'), cex = .7) # Introduces the legends.  
   
 ## The Second Plot  
 plot((m$y - m$Fitted\_Values) ~ m$Fitted\_Values, xlab = 'Fitted Values', ylab = 'Residuals', main = 'Residual vs. Fitted Values')  
   
 ## The Third Plot  
 qqnorm(m$y - m$Fitted\_Values)   
 qqline(m$y - m$Fitted\_Values)  
   
 newlist <- list('ll' = ll, 'ul' = ul, 'x' = m$x) # Stores the lower limits,  
 # the upper limits, as well as the corresponding x values to the new list.  
 invisible(newlist) # Silently returns the new list.  
  
}

## Load the data set.  
library(MASS)  
x <- mcycle$times  
y <- mcycle$accel  
  
## Apply the 'pspline' function.  
m <- pspline(x, y, k = 20, logsp = c(-5, 5), bord = 3, pord = 2, ngrid = 100)  
  
## Print EDF, k, residual std, r ^ 2, and GCV of the model.  
print.pspline(m)

## Order 3 p-spline with order 2 penalty   
## Effective degrees of freedom: 11.22137 Coefficients: 20   
## residual std dev: 22.67567 r-squared: 0.7797937 GCV: 4.222302

## Order 3 p-spline with order 2 penalty   
## Effective degrees of freedom: 11.22137 Coefficients: 20   
## residual std dev: 22.67567 r-squared: 0.7797937 GCV: 4.222302  
## Make predictions.  
new\_x<- 1 : 10  
predict.pspline(m, new\_x)

## $fit  
## [,1]  
## [1,] 0.07309776  
## [2,] -0.78716075  
## [3,] -1.65735217  
## [4,] -2.39155400  
## [5,] -2.85892111  
## [6,] -2.92953835  
## [7,] -2.50186050  
## [8,] -1.51607770  
## [9,] 0.08397340  
## [10,] 2.00730763  
##   
## $se  
## [1] 20.861461 14.451908 10.109967 8.449088 8.447217 8.404898 7.709292  
## [8] 7.148912 6.922900 6.670052

## $fit  
## [,1]  
## [1,] 0.07309776  
## [2,] -0.78716075  
## [3,] -1.65735217  
## [4,] -2.39155400  
## [5,] -2.85892111  
## [6,] -2.92953835  
## [7,] -2.50186050  
## [8,] -1.51607770  
## [9,] 0.08397340  
## [10,] 2.00730763  
##   
## $se  
## [1] 20.861461 14.451908 10.109967 8.449088 8.447217 8.404898 7.709292  
## [8] 7.148912 6.922900 6.670052  
## Sketch the diagnostic plots.  
plot.pspline(m)

