Project 3

Robin Lin

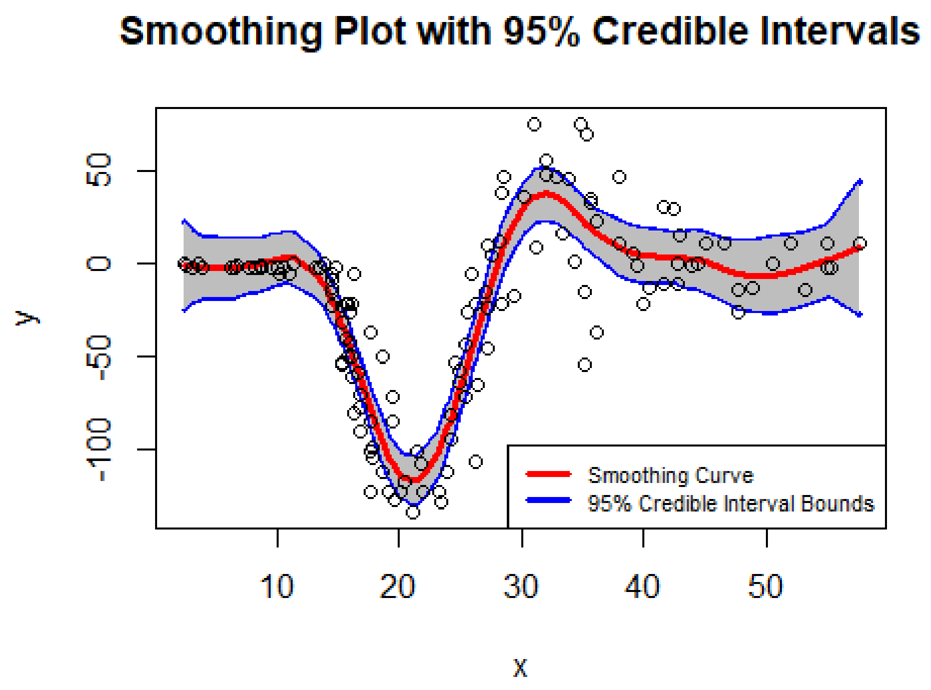
2022-11-04

```
## The author of this project is Sihong Lin, which can also be referred to as
## Robin Lin.

## The student number of the author is s2435943.

## The author declares that everything in the project is his own work, and he
## promises that he did not plagiarise from others.

## The url of the GitHub repository is listed as follows.
## https://github.com/RobinLam2435943/Statistical-Programming-Individual-
Projects.git

## ------------------------------------------------------------------------------
---

## 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 vary smoothly from its neighbouring ones.

## The model is hence estimated by the 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 optimal smoothing parameter based on GCV criterion. Also, some details
of
## the output of the method function are needed. Predictions are to be made,
## and some diagnostic 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.
The
  ## output is a bunch of lists containing detailed information 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(sm_par){

    ## This function aims at reckoning GCV criterion given a smoothing
parameter.

    revised_Lam <- diag(diag(1 + sm_par * 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,
reckoned
```

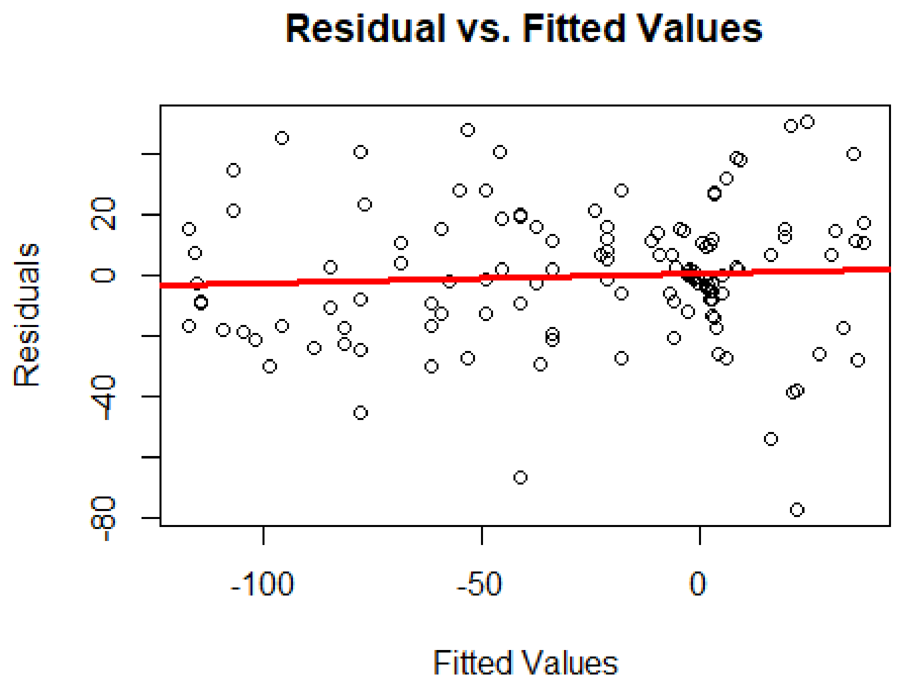
```
    # by taking the trace on the inverse of 'revised_Lam'.
    sig2 <- sum((y - fitted) ^ 2) / (nrow(X) - edf) # Residual Variance.
    gcv <- sig2 / (nrow(X) - edf) # GCV criterion.
    return(gcv)
  }

  GCV <- as.numeric(lapply(X = SP, FUN = gcv)) # Stores all the GCV values.
  sp <- SP[which.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,
reckoned
  # by 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')
```

```r
  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 model given new x
values
  ## within the range of the original data, as well as providing the standard
  ## errors of the model.

  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 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 aims at sketching 3 diagnostic plots of the model.

  ## The first one is the plot of the original data set, with the smoothing
  ## line being overlaid, along with the 95% credible intervals for the
smooth.
  ## In order to obtain the 95% credible intervals, both the lower and the
  ## upper limits should be reckoned. These two limits, along with their
  ## corresponding x values, should be returned silently.
  ## The second one is the plot of the residuals against fitted values.

  ## The third one is the Normal Q-Q plot.

  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 grey-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$Fitted_Values, xlab = 'Fitted Values',
ylab = 'Residuals', main = 'Residual vs. Fitted Values')
  # sketches the 'Residuals vs. Fitted' plot.
  abline(lm((m$y - m$Fitted_Values) ~ m$Fitted_Values), col = 'red', lwd = 3)
  # Adds a line representing the trend.

  ## The Third Plot

  qqnorm(m$y - m$Fitted_Values) # Normal Q-Q plot.
  qqline(m$y - m$Fitted_Values, col = 'red', lwd = 3) # Normal Q-Q line.

  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.
}
```

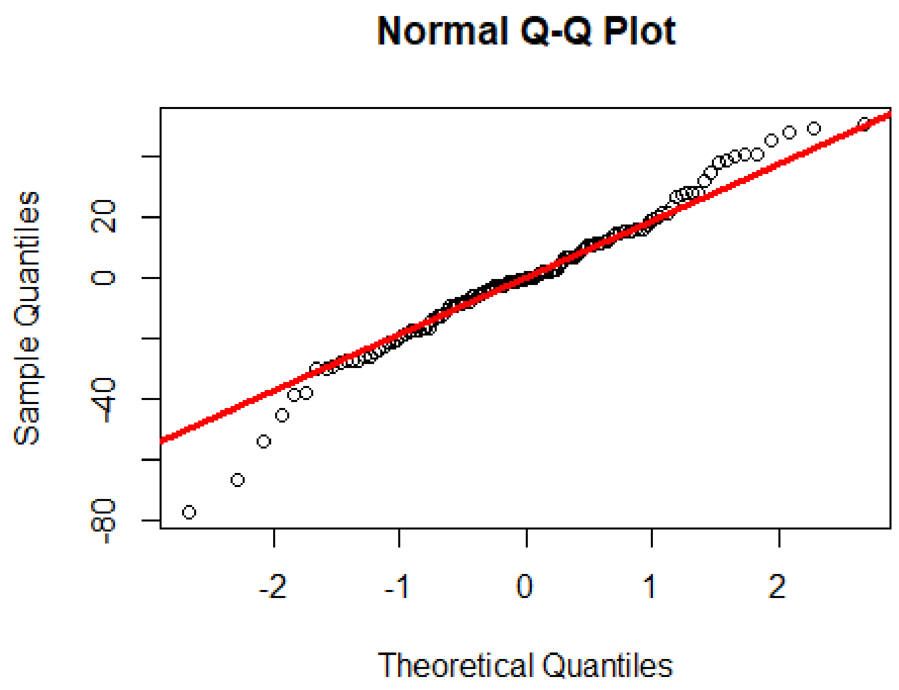
```
## Load the data set.
library(MASS)
x <- mcycle$times
y <- mcycle$accel

## Apply the 'pspline' function.
m <- pspline(x, y)

## 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

## Make predictions.
new_x<- x[1 : 10]
predict.pspline(m, new_x)

## $fit
##            [,1]
##  [1,] -1.143021
##  [2,] -1.318180
##  [3,] -1.819277
##  [4,] -2.122588
##  [5,] -2.391554
##  [6,] -2.885941
##  [7,] -2.736742
##  [8,] -2.630237
##  [9,] -1.760593
## [10,] -1.246979
##
## $se
##  [1] 12.369838 11.501348  9.580341  8.831780  8.449088  8.298085  8.019304
##  [8]  7.864015  7.222621  7.092230

## Sketch the diagnostic plots.
plot.pspline(m)
```



Smoothing Plot with 95% Credible Intervals
y
x
Smoothing Curve
95% Credible Interval Bounds
-100
-50
0
50
10
20
30
40
50



Residual vs. Fitted Values
Residuals
Fitted Values
-80
-40
0
20
-100
-50
0



Normal Q-Q Plot
Sample Quantiles
Theoretical Quantiles
-80
-40
0
20
-2
-1
0
1
2