Statistical Programming Assignment 3

Teng Wei Yeo

```
knitr::opts_chunk$set(fig.width=8, fig.height=5)
```

```
# Teng Wei Yeo, S2566430

###########################################
#### Smoothing with Thin Plate Splines ####
###########################################

# This code generates a fitted thin plate spline (TPS) to some data. A TPS is
# an estimation technique of estimating a smooth 3-dimensional surface based
# on some 2-dimensional observations (x1, x2) and their effect on some
# outcome variable (y).

# The goal is to find a model (i.e. find parameters of some fitted function)
# which (i) fits the data well, while (ii) having some degree of smoothness.

# For some TPS model f(x), the goal is to minimise the following objective
# function: the weighted average of the sum of squared residuals (y-f(x)),
# and a measure of un-smoothness (the square of the second derivative).
# The relative weights in this average is governed by parameter lambda.

# One way of choosing lambda is to choose a lambda that minimises the
# generalised cross validation (GCV) score. The GCV is a measure of the
# model's fit, penalised by the size of its Effective Degrees of Freedom
# (EDF)).

# This code can be separated into 3 distinct sections or functions:
#
## 1) getTPS(), which takes as input a matrix of 2-dimensional points
##    (x1, x2), samples some of these points to act as control points, and
##    computes the matrices X and S needed for the computation of the
##    parameters in the model which minimise the weighted sum of lack of fit
##    and the un-smoothness. Notably, getTPS() considers a re-parametrised
##    version of the objective function which is easier to minimise.
##
##    Helper functions eta_j() and eta() are used to apply a radial basis
##    function (by finding the Euclidean distance of a given point to the
##    control points, then applying a function on this norm).
##
## 2) fitTPS(), which calls getTPS(), and finds the lambda value which
##    minimises the GCV score. The use of a QR decomposition and an Eigen
##    decomposition of a symmetric matrix speeds up the computation
##    of the GCV score and the EDF.
##
```

```r
## 3) plot.tps(), a plot method function for objects with the class "tps"
##    (i.e. the object returned from fit.TPS), which plots a perspective
##    plot of the TPS model using the lambda value which minimises the GCV
##    score. Helper function pred_y() is used to predict the expected values
##    of y using the model's parameters (using the GCV-minimising lambda).


#######################
#### Brief Outline ####
#######################
# 1) eta() function
# 2) eta_j() function
# 3) getTPS() function
# 4) fitTPS() function
# 5) pred_y() function
# 6) plot.tps() function
# Appendix (A) testing() function
# Appendix (B) check_run_time() function
```

```r
#######################
#### Start of Code ####
#######################

eta <- function(r) {
  #' @description This function takes a real value, or vector, or matrix, and
  #' applies a piecewise function to each element in the input.
  #' The function eta(r) is r^2 * log(r) if r > 0, and 0 otherwise.
  #' This function uses the '> 0' condition to create a Boolean
  #' vector, which is used for sub-setting.
  #'
  #' @param r: real value, or vector, or matrix of real values.
  #'
  #' @returns r: real value, or vector, or matrix of real values after
  #' applying the function r^2 * log(r) if r > 0, 0 otherwise, to each
  #' element of r.

  # Set all non-positive values to 0.
  r[r <= 0] <- 0

  # Apply the function r^2 * log(r) to only the positive values.
  r[r > 0] <- r[r > 0]^2 * log(r[r>0])
  r # return r
}





eta_j <- function (x, xs){
  #' @description This function takes a (n by 2) matrix 'x' and a (k by 2)
  #' matrix 'xs', and finds the Euclidean distance between every point
  #' (i.e. every row) in 'x' with every other point (row) in 'xs'.
  #' This results in an (n by k) matrix. This function then calls function
  #' eta() on this resulting matrix.
```

```r
#'
#' #### Brief Outline ####
#' Step 1: construct a (1 by n) matrix with just the x1 coordinates x[,1]:
#' |x11 x21 x31 ... xn1|
#'
#' Step 2: Using step 1, construct a (k by n) matrix, where each row is an
#' identical repeat of the first row. Call this matrix x1temp:
#' |x11 x21 x31 ... xn1|
#' |x11 x21 x31 ... xn1|
#' |       ...         |
#' |x11 x21 x31 ... xn1|
#'
#' Step 3: Subtract xs[,1] from the matrix above. Because of the recycling
#' rule, every i,j element of the matrix will be the x[j,1] subtracted by
#' xs[i,1].
#'
#' Step 4: take the transpose of that resulting matrix, and find the square
#' of every term.
#'
#' Step 5: repeat steps 1-4 for the x2 coordinates for matrix x2temp.
#'
#' Step 6: take sqrt(x1temp + x2temp). The result is a pairwise Euclidean
#' norm for every term in x with every other term in xs.
#'
#' @param x: an (n by 2) matrix, or a vector of length 2 which will be
#' coerced into a (1 by 2) matrix.
#' @param xs: a (k by 2) matrix, or a vector of length 2 which will be
#' coerced into a (1 by 2) matrix.
#'
#' @returns an (n by k) matrix, where the (i,j) element is the eta() of the
#' distance (Euclidean norm) between the i-th row in 'x' and the j-th
#' row in 'xs'.

# Make x into a (1 by 2) matrix if it is a vector of length 2
if (!is.matrix(x)) {
  if (length(x) != 2){stop("x needs to be a vector of length 2, or
                           a matrix of size n by 2")}
  else x <- t(matrix(x))
}

# Make xs into a (1 by 2) matrix if it is a vector of length 2
if (!is.matrix(xs)) {
  if (length(xs) != 2){stop("xs needs to be a vector of length 2, or
                           a matrix of size n by 2")}
  else xs <- t(matrix(xs))
}

# Construct matrix for first coordinate.
x1temp <- matrix(rep(x[, 1], each = nrow(xs)), nrow = nrow(xs),
                 ncol = nrow(x))
x1temp <- t(x1temp - xs[,1]) # note the use of recycling rule

# Repeat for the second coordinate.
```

```r
  x2temp <- matrix(rep(x[, 2], each = nrow(xs)), nrow = nrow(xs),
                   ncol = nrow(x))
  x2temp <- t(x2temp - xs[,2])

  # Find the norm, then call function eta.
  eta(sqrt((x1temp^2 + x2temp^2)))
}
```

```r
getTPS <- function(x, k = 100){
  #' @description This function first chooses a set of k points in x to use as
  #' control points. If k >= n, then all x points are used as the control
  #' points.
  #'
  #' This function then computes the necessary matrices used in the re-
  #' parametrisation of the objective function to-be-minimised (the weighted
  #' sum of the lack of fit and the un-smoothness of the TPS).
  #'
  #' These matrices are X and S. X and S will be used by fitTPS() to compute
  #' beta-hat (the coefficients in the re-parametrised model) and mu-hat
  #' (the predicted y-values). These are used to compute the Effective Degrees
  #' of Freedom (EDF) and the Generalised Cross Validation (GCV) score for
  #' a given value of the smoothness parameter lambda.
  #'
  #' This function also finds the QR decomposition of Z, which is needed to
  #' compute X and S, but without explicitly forming Z (to be computationally
  #' more efficient). This decomposition is also used by plot.tps() for
  #' model prediction.
  #'
  #' @param x: matrix of size (n by 2). This matrix represents a set of
  #' n points, where each point is a row, and has two coordinates x1 and
  #' x2.
  #' @param k: number of basis functions to use for the TPS model
  #'
  #' @returns output: a list containing the following named items:
  #' xk: size (k by 2) matrix containing the selected x* points which
  #' are used as the TPS's 'control points' (i.e. basis functions).
  #'
  #' X: size (n by k) matrix, where X = [E %*% Z, T]. X is the model matrix
  #' used in the re-parametrisation.
  #'
  #' S: size (k by k) matrix, where S is the matrix used in the
  #' re-parametrisation. S is in the term multiplied by lambda in the
  #' objective function, and will be used to find the parameters of the
  #' TPS.
  #'
  #' TsQR: the QR decomposition of Ts, from which computations using Z
  #' (where Z is the last k-3 columns of Q) can be made without explicitly
  #' forming Z.
```

```r
n <- nrow(x) # count the number of rows in matrix x

# If the number of basis functions exceeds n, then set k to n.
# Otherwise, pick k of the n points (without replacement) to be used as
# the control points.
if (k >= n) {
  k <- n
  xk <- x # set all x points as xk points (control points)
} else {
  # Sample k of the n points in x to use as control points
  xk <- x[sample(n, k, replace = FALSE), ]
}


# Create matrix E, a size (n by k) matrix where the i,j-th element
# is the eta() function applied to the Euclidean distance between
# the i-th point in x and the j-th point in xk.
E <- eta_j(x, xk)


# Create matrix Es, a size (k by k) matrix where the i,j-th element
# is the eta() function applied to the Euclidean distance between
# the i-th point in xk and the j-th point in xk.
Es <- eta_j(xk, xk)


# Create matrix T of size (n by 3). The first column is a column of 1.
# The second and third column are the values of x.
T <- cbind(rep(1,nrow(x)), x)


# Create matrix Ts of size (k by 3). The first column is a column of 1.
# The second and third column are the values of xk.
Ts <- cbind(rep(1,nrow(xk)), xk)


# QR decomposition of Ts, without forming Q and R explicitly.
TsQR <- qr(Ts)


# Constructing X matrix of size (n by k). The first (n by k-3) columns are
# from E %*% Z. The last 3 columns are from T.
# Z is the last k-3 columns of Q (Q is from the QR decomposition of Ts).
# EZ is computed by taking the last k-3 columns of t(t(Q) %*% t(E)).
# qr.qty used to speed up computation instead of explicitly forming Q
X <- cbind(t(qr.qty(TsQR, t(E)))[, -(1:3)], T)


# Construct S matrix, by first computing B = last k-3 rows of (t(Q) %*% Es)
# then finding the last k-3 columns of t(t(Z) %*% t(B))
# qr.qty was used to speed up computation.
# Extra zeros were then added such that S is of size (k by k).
S <- rbind(cbind(t(qr.qty(TsQR , t(qr.qty(TsQR, Es)[-(1:3), ])))[, -(1:3)],
```

```r
                    matrix(0, k-3, 3)),
             matrix(0, 3, k))
  # S is symmetric; the eigen decomposition will yield a matrix of
  # eigenvectors which is orthogonal.


  # Return 'output', which is a list of items needed by subsequent functions
  # to compute the EDF and GCV, and to predict fitted values.
  output <- list(xk = xk, X = X, S = S, TsQR = TsQR)
}
```

```r
fitTPS <- function(x, y, k=100, lsp=c(-5,5)){
  #' @description The goal is now to search for a lambda that minimises
  #' the GCV score. The function speeds up computation by using a
  #' transformation of the problem by finding the QR decomposition of the
  #' model matrix X. It also finds the eigen decomposition of a symmetric
  #' matrix R^(-T) S R^(-1).
  #'
  #' These transformations aid in the computation of:
  #' 1) the GCV score, because explicit inverses need not be computed. The
  #'    QR decomposition leads to a triangular system which allows the use
  #'    of backsolves.
  #'
  #' 2) the EDF. Because of the properties of trace, the orthogonality of
  #'    the eigenvectors (since the matrix was symmetric), and the
  #'    orthogonality of Q, the computation simplified to the sum of
  #'    (1/(1 + lambda * eigenvalues)) for all eigenvalues.
  #'
  #' X and S are matrices obtained from getTPS().
  #'
  #' The lambdas searched over are the exponential of the 100 values of log
  #' lambda evenly spaced between the limits lsp[1] and lsp[2].
  #'
  #' @param x: size (n by 2) matrix of points.
  #' @param y: vector of size n, which is the response/outcome for each point
  #' in x. The TPS then finds the function f(x) which smooths over these y
  #' values.
  #' @param k: number of basis functions to use.
  #' @param lsp: log lambda limits which the function fitTPS() searches over
  #' to find the lambda which minimises the GCV score.
  #'
  #' @returns output: a list object of class 'tps' containing the following
  #' named items:
  #'
  #' beta: the beta-hat parameters which minimise the objective function using
  #' the value of lambda which minimises the GCV score.
  #'
  #' mu: the predicted y-hat values using the fitted TPS for each point in x,
```

```
#' using the value of lambda which minimises the GCV score.
#'
#' medf: the effective degrees of freedom based on the value of lambda which
#' minimises the GCV score.
#'
#' lambda: a vector of size 100 of all the values of lambda tried by
#' fitTPS()
#'
#' gcv: a vector of size 100 of the GCV scores for each lambda searched over
#'
#' edf: a vector of size 100 of the EDF for each lambda searched over
#'
#' TsQR: the QR decomposition of the Ts matrix needed to compute with Z
#' without explicitly forming Z. TsQR will be used by plot.tps() to compute
#' the parameters needed for predicting with the model.
#'
#' xk: the set of control points, used by plot.tps() to predict with the
#' model.

# Set up the lambdas using lsp, the log lambda limits:
# Create a sequence of 100 evenly spaced values from lsp[1] to lsp[2]
sequence <- seq(from = lsp[1], to = lsp[2], length.out = 100)

# Convert from log lambda to lambda
lambda <- exp(sequence)

# Set up the TPS
vals <- getTPS(x = x, k = k)
xk <- vals$xk
X <- vals$X
S <- vals$S
TsQR <- vals$TsQR

n <- nrow(x) # number of observations in x

# The rest of this code optimises the computation of GCV and EDF for each
# value of lambda tried. Each GCV computation requires computing:
### mu-hat: the model's predicted value of E(y)
### EDF: Effective Degrees of Freedom
# Simplifying the computation of GCV requires a transformation of beta-hat.
# This transformation relies on a QR decomposition of X, and a eigen
# decomposition of R^{-T} %*% S %*% R^{-1}

temp <- qr(X) # QR decomposition of X
Q <- qr.Q(temp, complete = FALSE) # Q is orthogonal of size (n by k)
R <- qr.R(temp, complete = FALSE) # R is upper-triangular, size (k by k)

# Need to find matrix A = R^{-T} %*% S %*% R^{-1}.
# Finding the inverse of R explicitly is computationally slow.
# Instead, use forward solve, since R is an upper-triangular matrix.

# Use forward solve to find B in R^{T} B = S.
# We then need to find A in AR = B.
```

```r
# Notice that ((R^T)(A^T)) = B^T
# Hence, A = t(forwardsolve(t(R), t(B)))

A <- t(forwardsolve(t(R), t(forwardsolve(t(R), S))))

# Notice that A is theoretically symmetric, but numerically it is not
# due to numeric approximations by the computer. Hence, we force symmetry:
A <- (t(A)+A)*.5

# Can then find the eigen decomposition:
ed <- eigen(A)
U <- ed$vectors # eigenvectors, which is orthogonal since A is symmetric.

# Create vector of length 100 to store the GCV score of each lambda
gcv <- rep(0, 100)
# Create vector of length 100 to store the EDF value for each lambda
edf <- rep(0, 100)


# Now we need to find beta-hat for each lambda where beta-hat is defined by
# R beta-hat = U (I+lambda LAMBDA)^-1 U^T Q^T y,
# where lambda is the smoothing parameter, and LAMBDA is the diagonal matrix
# of eigenvalues.

# Since (I + lambda LAMBDA) is diagonal, its inverse is the reciprocal
# of its elements.

# We hence find:
# [U^T R beta-hat]_i = [U^T Q^T * y]_i / (1 + lambda Lambda_{i,i}) = C
# Then use U^T = U^-1 to make:
# R beta_hat = U %*% C
# Then back solve.

# Before looping over lambda, compute [U^T Q^T * y] first, since these
# values are always the same for each lambda. Notice the placement of
# brackets to speed up the computation of the matrix multiplication.
inter <- t(U) %*% (t(Q) %*% y)

# Loop over all lambda values within the lambda limits found earlier
for (i in 1:length(lambda)){
  # Element-wise division of two vectors of the same length
  rhs <- inter / (1 + lambda[i] * ed$values)

  full_rhs <- U %*% rhs
  beta_hat <- backsolve(R, full_rhs) # solving R beta_hat = full_rhs


  mu_hat <- X %*% beta_hat # predicted y values

  # Now, find EDF: the trace of (1 + lambda LAMBDA)^-1,
  # which is equivalent to the sum of 1/(1+lambda LAMBDA{ii})
  edf[i] <- sum(1 / (1+lambda[i]*ed$values))
```

```r
    # And now, compute the GCV score.
    gcv[i] <- sqrt(sum((y - mu_hat)^2)) / (n - edf[i])^2
  }

  opt_i <- which.min(gcv) # index of minimum GCV

  # Compute beta-hat and mu-hat of the GCV-minimising lambda
  rhs <- inter / (1 + lambda[opt_i] * ed$values)
  full_rhs <- U %*% rhs
  beta <- backsolve(R, full_rhs)
  mu <- X %*% beta

  medf <- edf[opt_i] # edf corresponding to GCV-minimising lambda

  output <- list(beta = beta, mu = mu, medf = medf, lambda = lambda,
                 gcv = gcv, edf = edf, TsQR = TsQR, xk = xk)
  class(output) <- "tps" # set output to class "tps"
  output # return output
}




pred_y <- function(output, xp){
  #' @description This is a helper function which uses the parameters (which
  #' minimise the objective function for the optimal lambda which minimises
  #' the GCV score) from fitTPS() to set up the TPS model for prediction.
  #'
  #' eta_j() is also called to find the eta function of the new points xp,
  #' using the control points xk (where xk was chosen in getTPS() and
  #' returned in the output of fitTPS()).
  #'
  #' @param output: an object of class "tps" which is returned from fitTPS().
  #' @param xp: a matrix of size (m by 2) points, to have their y-values
  #' predicted.
  #'
  #' @returns a vector of the predicted y values of size m (nrow(xp)).

  # We first recover the delta coefficients (the coefficients used to
  # multiply the eta_j() values of x in the original f(x) model) from beta
  # (the coefficients in the re-parametrised objective function).
  # The first k-3 elements in beta are delta_z.
  # Then, delta = Z %*% delta_z.

  # This can be computed without explicitly forming Z.
  # Because Q is of size k by k, but delta_z is of size (k-3 by 1), the
  # matrices are non-conformable. Hence, a dummy_deltaz vector is created
  # with additional zeroes for its first 3 terms.

  # This construction ensures that Q %*% dummy_deltaz is equivalent
  # to Z %*% delta_z, since the first 3 columns of Q (which are not in Z)
  # will be multiplied by the zeroes in dummy_deltaz.
```

```r
  dummy_deltaz = c(rep(0,3), output$beta[1:(length(output$beta)-3)])
  delta <- qr.qy(output$TsQR, dummy_deltaz)

  # The last 3 elements in beta are the alpha coefficients in f(x).
  alpha <- output$beta[(length(output$beta)-2):length(output$beta)]

  # Compute vector 'inter' of size (nrow(xp)), where the i-th element
  # corresponds to the value of {sum (eta_j(xp)[i,j] * delta_j) for all j},
  # where j = 1, 2, ..., k
  # i = 1, 2, ..., nrow(xp)

  inter <- eta_j(xp, output$xk) %*% delta

  # compute the predicted y
  T <- cbind(rep(1,nrow(xp)), xp)
  y_hat <- T %*% alpha + inter
}




plot.tps <- function(output, m = 50, theta = 30, phi = 30, ...) {
  #' @description This function is a plot method function for objects of
  #' the class 'tps', as returned from fitTPS(). It generates a (m by m)
  #' grid of x values for the grid visualisation of the TPS. It uses the
  #' helper function pred(y) to predict the y-value of each of these grid
  #' values. It then plots a perspective plot.
  #'
  #' @param output: an object of class 'tps' returned from fitTPS.
  #' @param m: the number of grid lines on each axis to plot
  #' @param theta: azimuthal direction of the viewing angle of the plot
  #' @param phi: colatitude of the viewing angle of the plot
  #' @param ...: any other argument to be passed into persp()
  #'
  #' @returns a perspective plot of the TPS.

  x2 <- x1 <- seq(0,1,length=m) # for grid lines
  xp <- cbind(rep(x1,m),rep(x2,each=m)) # size ((m^2) by 2) matrix of points

  # Perspective plot of fitted thin plate spline by calling pred_y().
  # Theta and Phi are the viewing angle.
  persp(x1,x2,matrix(pred_y(output = output, xp = xp),m,m),theta=theta,
        phi=phi, zlab = "y-hat", main="Fitted Thin Plate Spline", ...)
}


########################################
#### Appendix A: Testing function ####
########################################

testing <- function() {
  #' @description This function runs a test on the TPS functions written in
```

```
  #' this code.
  #'
  #' @returns A 4-panel (2x2) plot:
  #' In the top row, the first plot shows the fitted thin plate spline, where
  #' the fitted model uses the lambda that minimises the GCV score. The second
  #' plot shows the true function.
  #' In the bottom row, the first plot shows the GCV score against
  #' log(lambda). The second plot shows the GCV score against EDF.

  ff <- function(x) exp(-(x[,1]-.3)^2/.2^2-(x[,2] - .3)^2/.3^2)*.5 +
    exp(-(x[,1]-.7)^2/.25^2 - (x[,2] - .8 )^2/.3^2) # test function

  # Next, simulate the data.
  n <- 500 # number of observations

  # Generate n pairs of observations from the uniform distribution
  x <- matrix(runif(n*2),n,2)

  y <- ff(x) + rnorm(n)*.1 # generate data with some noise for fitting

  output <- fitTPS(x, y, k = 100) # call the TPS fitting function

  par(mar = c(0, 2, 2, 2)) # set up the margins to better display the graphs
  par(mfrow=c(2,2)) # plot window: two rows, two columns
  plot(output) # plot the fitted thin plate spline

  # Next, set up the plot for the true function:
  m <- 50;x2 <- x1 <- seq(0,1,length=m) # for grid lines
  xp <- cbind(rep(x1,m), rep(x2,each=m)) # ((m^2) by 2) matrix of x values
  persp(x1, x2, matrix(ff(xp), m, m), theta=30, phi=30, zlab = "True y",
        main = "True Function") # perspective plot of true function


  par(mar = c(5, 4, 4, 2) + 0.1) # new margins to better display the next row

  plot(log(output$lambda), output$gcv, # plot of GCV against log lambda
       type = "l", # line graph
       xlab = expression(paste(log(lambda))), # x label
       ylab = "gcv",  # y label
       main = "GCV score against log lambda") # title
  plot(output$edf, output$gcv,  # plot of GCV against EDF
       type = "l", # line graph
       xlab = "edf", # x label
       ylab = "gcv", # y label
       main = "GCV score against EDF") # title
}



##################################################
#### Appendix B: Test Code, and Check Profile ####
##################################################
```

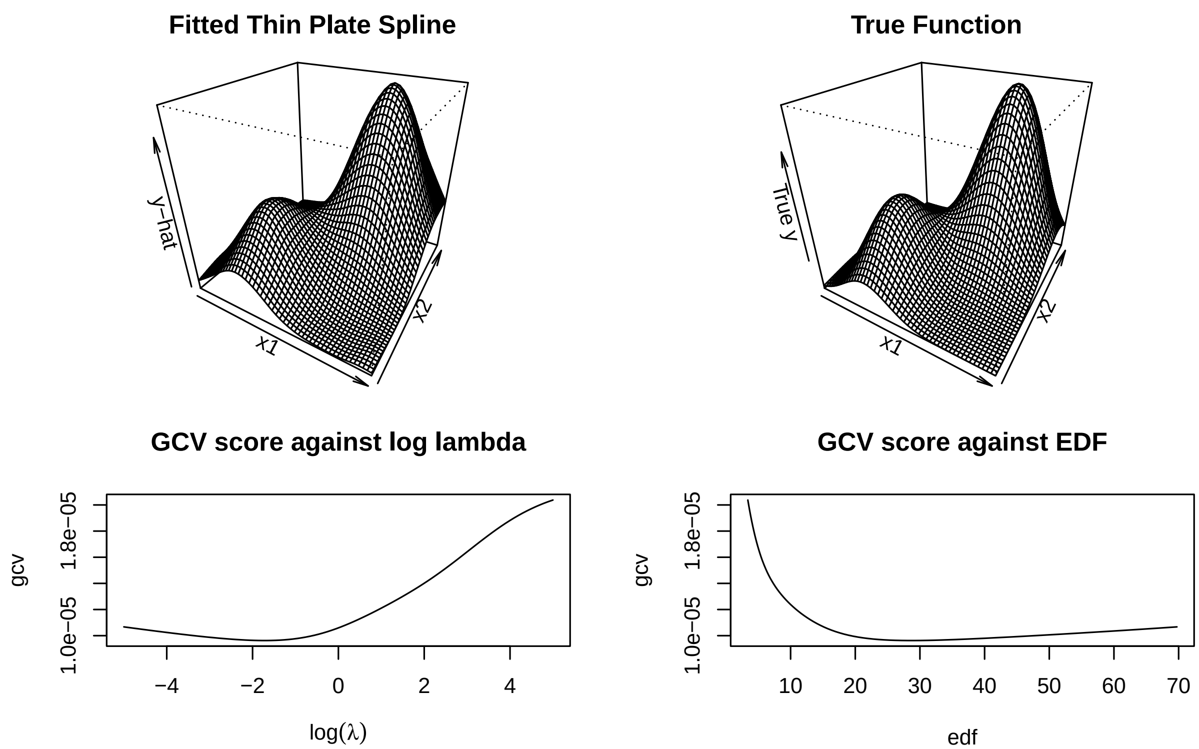
```r
check_run_time <- function() {
  #' @description This function runs testing(), and checks the profile of
  #' the program.
  #'
  #' @returns the 2x2 plots from testing(), and the profile of the code.

  Rprof() # starts profiling
  testing() # run testing code
  Rprof(NULL) # finish profiling
  summaryRprof() # show profiling
}

####################
#### END of Code ####
####################
```

```r
### Run a test on the code, display the output, and show the time taken

check_run_time()
```




```
## $by.self
##                     self.time self.pct total.time total.pct
## "cmpCall"                0.08    16.67       0.22     45.83
## "eigen"                  0.06    12.50       0.06     12.50
## "eta"                    0.06    12.50       0.06     12.50
## "constantFoldCall"       0.04     8.33       0.06     12.50
## "cb$putcode"             0.04     8.33       0.04      8.33
## "rnorm"                  0.04     8.33       0.04      8.33
## "persp.default"          0.02     4.17       0.08     16.67
```

```
## "matrix"               0.02     4.17       0.06     12.50
## "%in%"                 0.02     4.17       0.02      4.17
## ".Fortran"             0.02     4.17       0.02      4.17
## "as.list"              0.02     4.17       0.02      4.17
## "backsolve"            0.02     4.17       0.02      4.17
## "findCenvVar"          0.02     4.17       0.02      4.17
## "lazyLoadDBfetch"      0.02     4.17       0.02      4.17
##
## $by.total
##                           total.time total.pct self.time self.pct
## "block_exec"                    0.48    100.00      0.00     0.00
## "call_block"                    0.48    100.00      0.00     0.00
## "check_run_time"                0.48    100.00      0.00     0.00
## "eng_r"                         0.48    100.00      0.00     0.00
## "eval"                          0.48    100.00      0.00     0.00
## "eval_with_user_handlers"       0.48    100.00      0.00     0.00
## "evaluate"                      0.48    100.00      0.00     0.00
## "evaluate::evaluate"            0.48    100.00      0.00     0.00
## "evaluate_call"                 0.48    100.00      0.00     0.00
## "handle"                        0.48    100.00      0.00     0.00
## "in_dir"                        0.48    100.00      0.00     0.00
## "in_input_dir"                  0.48    100.00      0.00     0.00
## "knitr::knit"                   0.48    100.00      0.00     0.00
## "process_file"                  0.48    100.00      0.00     0.00
## "process_group"                 0.48    100.00      0.00     0.00
## "rmarkdown::render"             0.48    100.00      0.00     0.00
## "testing"                       0.48    100.00      0.00     0.00
## "timing_fn"                     0.48    100.00      0.00     0.00
## "withCallingHandlers"           0.48    100.00      0.00     0.00
## "withVisible"                   0.48    100.00      0.00     0.00
## "xfun:::handle_error"           0.48    100.00      0.00     0.00
## "fitTPS"                        0.26     54.17      0.00     0.00
## "cmpfun"                        0.24     50.00      0.00     0.00
## "compiler:::tryCmpfun"          0.24     50.00      0.00     0.00
## "doTryCatch"                    0.24     50.00      0.00     0.00
## "tryCatch"                      0.24     50.00      0.00     0.00
## "tryCatchList"                  0.24     50.00      0.00     0.00
## "tryCatchOne"                   0.24     50.00      0.00     0.00
## "cmpCall"                       0.22     45.83      0.08    16.67
## "cmp"                           0.22     45.83      0.00     0.00
## "genCode"                       0.22     45.83      0.00     0.00
## "tryInline"                     0.22     45.83      0.00     0.00
## "h"                             0.20     41.67      0.00     0.00
## "cmpSymbolAssign"               0.14     29.17      0.00     0.00
## "cb$putconst"                   0.12     25.00      0.00     0.00
## "cmpCallArgs"                   0.12     25.00      0.00     0.00
## "cmpCallSymFun"                 0.12     25.00      0.00     0.00
## "getTPS"                        0.12     25.00      0.00     0.00
## "eta_j"                         0.10     20.83      0.00     0.00
## "persp.default"                 0.08     16.67      0.02     4.17
## "persp"                         0.08     16.67      0.00     0.00
## "eigen"                         0.06     12.50      0.06    12.50
## "eta"                           0.06     12.50      0.06    12.50
## "constantFoldCall"              0.06     12.50      0.04     8.33
```

```
## "matrix"                  0.06     12.50      0.02     4.17
## "constantFold"            0.06     12.50      0.00     0.00
## "plot"                    0.06     12.50      0.00     0.00
## "plot.tps"                0.06     12.50      0.00     0.00
## "pred_y"                  0.06     12.50      0.00     0.00
## "cb$putcode"              0.04      8.33      0.04     8.33
## "rnorm"                   0.04      8.33      0.04     8.33
## "cmpPrim2"                0.04      8.33      0.00     0.00
## "getInlineInfo"           0.04      8.33      0.00     0.00
## "%in%"                    0.02      4.17      0.02     4.17
## ".Fortran"                0.02      4.17      0.02     4.17
## "as.list"                 0.02      4.17      0.02     4.17
## "backsolve"               0.02      4.17      0.02     4.17
## "findCenvVar"             0.02      4.17      0.02     4.17
## "lazyLoadDBfetch"         0.02      4.17      0.02     4.17
## "<Anonymous>"             0.02      4.17      0.00     0.00
## "cmpBuiltinArgs"          0.02      4.17      0.00     0.00
## "cmpForBody"              0.02      4.17      0.00     0.00
## "cmpIndices"              0.02      4.17      0.00     0.00
## "cmpPrim1"                0.02      4.17      0.00     0.00
## "cmpSubsetDispatch"       0.02      4.17      0.00     0.00
## "exists"                  0.02      4.17      0.00     0.00
## "findLocalsList"          0.02      4.17      0.00     0.00
## "findLocalsList1"         0.02      4.17      0.00     0.00
## "FUN"                     0.02      4.17      0.00     0.00
## "funEnv"                  0.02      4.17      0.00     0.00
## "getFoldFun"              0.02      4.17      0.00     0.00
## "getInlineHandler"        0.02      4.17      0.00     0.00
## "isBaseVar"               0.02      4.17      0.00     0.00
## "lapply"                  0.02      4.17      0.00     0.00
## "make.functionContext"    0.02      4.17      0.00     0.00
## "qr"                      0.02      4.17      0.00     0.00
## "qr.default"              0.02      4.17      0.00     0.00
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 0.48
```

— END OF DOCUMENT —