Hwang’s mlspline

Choi TaeYoung

2019 11 30

#’ generate simulated response for multilevel splines #’ #’ Generates simulated response for multilevel splines #’ #’ @importFrom stats coef glm lm rbinom rnorm vcov #’ @param J number of ‘data’ intervals #’ @param mod underlying model; either lm or glm #’ @param x\_sigma design matrix sigma #’ @param e\_sigma error variance - around the mean function; data level. #’ @param z\_sigma error variance around my surface; structural level. #’ @param N\_s the minimum sample size for each interval. #’ @param N\_m the maximum sample size for each interval; default = 200. #’ @return returns a list described above. #’ @format list(x\_list = x\_list, y\_list = y\_list, e\_list = e\_list, true\_mu = mu, z = z) #’ \describe{ #’

{x\_list}{the length-J list of design matrices. The nrow of each element is between N\_s and N\_m} #’

{y\_list}{the length-J list of response vectors. The length of each element is between N\_s and N\_m.} #’

{e\_list}{the length-J list of error vectors. The length of each element is between N\_s and N\_m.} #’

{true\_mu}{the true mu vector of length J} #’

{z}{the grid vector of length J} #’

generate\_response <- function(J, mod, e\_sigma = 1, x\_sigma = 1, z\_sigma = 0.5, N\_s, N\_m = 200) {  
  
 # currently the data interval (z interval) is set to be between -3 and 3.  
  
 n <- sample(N\_s:N\_m, J, replace = TRUE)  
  
 # smooth surface: z is the grid sequence and mu is the generated smooth function.  
 z <- seq(from = -3, to = 3, length.out = J)  
 mu <- z^2 - 10 \* cos(2 \* pi \* z) # "true" surface.  
  
 beta\_1 <- mu + rnorm(J, 0, z\_sigma) # slope  
 beta\_0 <- 0 # intercept  
  
 x\_list <- lapply(n, rnorm, mean = 0, sd = x\_sigma)  
 e\_list <- lapply(n, rnorm, mean = 0, sd = e\_sigma)  
  
 # outcome generation function; gives 'y' list given e, beta\_0, beta\_1, and  
 # x (design matrix)  
 # for glm: logit link binary p(y = 1) = 1/(1 + exp(-beta\_0 - beta\_1 \* x - e)  
 # for lm: ordinary linear model structure y = xb + e  
 if (mod == "glm") {  
 y\_list <- mapply(function(x, e, b, beta\_0 = 0)  
 rbinom(length(x), 1, 1/(1 + exp(-beta\_0 - b \* x - e))),  
 x = x\_list, e = e\_list, b = beta\_1)  
 }  
 if (mod == "lm") {  
 y\_list <- mapply(function(x, e, b, beta\_0 = 0)  
 beta\_0 + b \* x + e, x = x\_list, e = e\_list, b = beta\_1)  
 }  
 list(x\_list = x\_list, y\_list = y\_list, e\_list = e\_list, true\_mu = mu, z = z)  
}

#’ Builds `granular'' data #' #' obtains the regression slope and its variance #' certainly not optimal but this step shouldn't take long regardless #' @param x\_k design matrix #' @param y\_k response vector #' @param mod underlying model; eitherlmorglm` #’ @export

granular <- function(x\_k, y\_k, mod) {  
 # summarizing the regression part  
 if (mod == "glm")  
 fit\_lm <- glm(y\_k ~ x\_k, family = "binomial")  
 if (mod == "lm")  
 fit\_lm <- lm(y\_k ~ x\_k)  
  
 kth\_beta\_hat <- coef(fit\_lm)[2]  
 kth\_var <- diag(vcov(fit\_lm))[2]  
 grain\_out <- list(kth\_beta\_hat, kth\_var)  
 grain\_out  
}

#’ Generates kerel matrix #’ #’ Generates kernel matrix of J by J, where J = length(z) for multilevel splines #’ certainly not optimal but this step shouldn’t take long regardless. #’ Used the formulation from Reinsch (1967). #’ @param z Mid-interval value vector, it is safe to assume this to be equi-distant, but in principle it doesn’t have to be. it’s not tested though. #’ @export

make\_K <- function(z) {  
 J <- length(z)  
 Del <- matrix(0, nrow = J - 2, ncol = J)  
 W <- matrix(0, nrow = J - 2, ncol = J - 2)  
 h <- diff(z)  
 for (l in 1:(J - 2)) {  
 Del[l, l] <- 1/h[l]  
 Del[l, (l + 1)] <- -1/h[l] - 1/h[(l + 1)]  
 Del[l, (l + 2)] <- 1/h[(l + 1)]  
 W[(l - 1), l] <- W[l, (l - 1)] <- h[l]/  
 6  
 W[l, l] <- (h[l] + h[l + 1])/3  
 }  
 K <- t(Del) %\*% solve(W) %\*% Del  
 K  
}

#’ Main EM function #’ #’ Running EM for multilevel splines #’ certainly not optimal… #’ @param beta\_hat\_vec data vector of length J #’ @param V covariance matrix of size J by J #’ @param K kernel matrix from make\_K #’ @param lambda tuning parameter #’ @param maxit maximum iteration number #’ @export

main\_EM <- function(beta\_hat\_vec, V, K, lambda, maxit = 500) {  
  
 # parameter initilization  
 eps <- 1000 # convergence tracker  
 tol <- 1e-05 # convergence threshold  
 sigma2\_m <- mean(diag(V))  
 J <- length(beta\_hat\_vec)  
 mu\_m <- rep(mean(beta\_hat\_vec), J)  
 I <- diag(J)  
 iter <- 1  
  
 while (eps > tol & iter <= maxit) {  
 # .. EM starts here  
 mu\_m\_old <- mu\_m  
 sigma2\_m\_old <- sigma2\_m # current sigma^2  
  
 Vst <- solve(solve(V) + (1/sigma2\_m) \* diag(J)) # Vst  
 D\_m <- Vst %\*% solve(V) #D\_m <- part\_cov %\*% V  
 mu\_m <- solve(D\_m + lambda \* K) %\*% D\_m %\*% beta\_hat\_vec  
  
 S\_lambda <- solve(I %\*% D\_m %\*% I + lambda \* K) %\*% I %\*% D\_m  
 effective\_df <- sum(diag(S\_lambda))  
  
 sigma2\_m <- mean((beta\_hat\_vec - mu\_m)^2)  
 eps <- sum(abs(mu\_m - mu\_m\_old)) + abs(sigma2\_m\_old - sigma2\_m)  
 iter <- iter + 1  
 if (iter == maxit) {  
 cat("for lambda =", lambda, "max iteration reached; may need to double check \n")  
 }  
 } # end of EM .. convergence reached.  
  
 BIC <- sum((beta\_hat\_vec - mu\_m)^2)/(J^(1 - effective\_df/J))  
 GCV <- sum((beta\_hat\_vec - mu\_m)^2)/(J - effective\_df)^2 \* J  
  
 EM\_out <- list(mu = mu\_m, S\_lambda = S\_lambda, sigma2 = sigma2\_m, BIC = BIC, GCV = GCV)  
 EM\_out  
}

#’ Naive strawman #’ #’ Running naive splines #’ @param beta\_hat\_vec data vector of length J #’ @param K kernel matrix from make\_K #’ @param lambda tuning parameter #’ @export

naive\_ss <- function(beta\_hat\_vec, lambda, K) {  
  
 J <- length(beta\_hat\_vec)  
 I <- diag(J)  
 S\_lambda <- solve(I + lambda \* K)  
 f\_hat <- S\_lambda %\*% beta\_hat\_vec  
  
 eff\_df <- sum(diag(S\_lambda))  
  
 GCV <- sum((beta\_hat\_vec - f\_hat)^2)/(J - eff\_df)^2 \* J  
 BIC <- log(mean((beta\_hat\_vec - f\_hat)^2)) + eff\_df \* log(J)/J  
  
 out <- list(mu = f\_hat, S\_lambda = S\_lambda, BIC = BIC, GCV = GCV)  
 out  
}

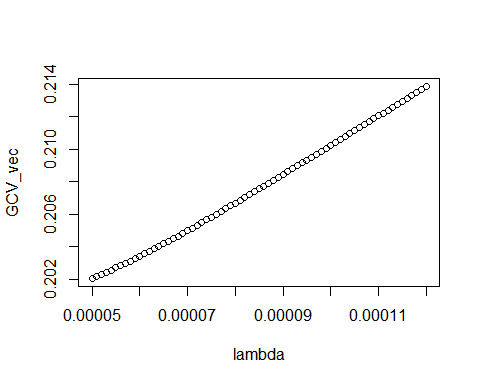
#’ Generates simulated response for multilevel splines – test function #2 #’ #’ @importFrom stats coef glm lm rbinom rnorm vcov #’ @param J number of ‘data’ intervals #’ @param mod underlying model; either lm or glm #’ @param x\_sigma design matrix sigma #’ @param e\_sigma error variance - around the mean function; data level. #’ @param z\_sigma error variance around my surface; structural level. #’ @param N\_s the minimum sample size for each interval. #’ @param N\_m the maximum sample size for each interval; default = 200. #’ @return returns a list described above. #’ @format list(x\_list = x\_list, y\_list = y\_list, e\_list = e\_list, true\_mu = mu, z = z) #’ #’ @export

generate\_response\_smooth <- function(J, mod, e\_sigma = 1, x\_sigma = 1, z\_sigma = 0.5, N\_s, N\_m = 200) {  
  
 # currently the data interval (z interval) is set to be between 0 and 1  
  
 n <- sample(N\_s:N\_m, J, replace = TRUE)  
  
 # smooth surface: z is the grid sequence and mu is the generated smooth function.  
 z <- seq(from = 0, to = 1, length.out = J)  
 mu <- sin(12\*(z + 0.2)) / (z + 0.2) # "true" surface.  
  
 beta\_1 <- mu + rnorm(J, 0, z\_sigma) # slope  
 beta\_0 <- 0 # intercept  
  
 x\_list <- lapply(n, rnorm, mean = 0, sd = x\_sigma)  
 e\_list <- lapply(n, rnorm, mean = 0, sd = e\_sigma)  
  
 # outcome generation function; gives 'y' list given e, beta\_0, beta\_1, and  
 # x (design matrix)  
 # for glm: logit link binary p(y = 1) = 1/(1 + exp(-beta\_0 - beta\_1 \* x - e)  
 # for lm: ordinary linear model structure y = xb + e  
 if (mod == "glm") {  
 y\_list <- mapply(function(x, e, b, beta\_0 = 0)  
 rbinom(length(x), 1, 1/(1 + exp(-beta\_0 - b \* x - e))),  
 x = x\_list, e = e\_list, b = beta\_1)  
 }  
 if (mod == "lm") {  
 y\_list <- mapply(function(x, e, b, beta\_0 = 0)  
 beta\_0 + b \* x + e, x = x\_list, e = e\_list, b = beta\_1)  
 }  
 list(x\_list = x\_list, y\_list = y\_list, e\_list = e\_list, true\_mu = mu, z = z)  
}

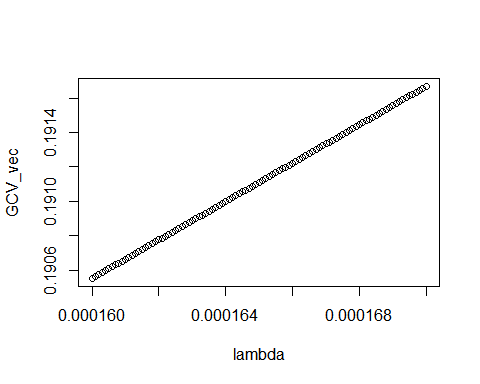
## simulation model

ag2 <- generate\_response\_smooth(J=50, mod="glm", e\_sigma=2, N\_s=50)  
ag22 <- generate\_response\_smooth(J=50, mod="glm", e\_sigma=2, N\_s=100)  
ag4 <- generate\_response\_smooth(J=50, mod="glm", e\_sigma=4, N\_s=50)  
ag44 <- generate\_response\_smooth(J=50, mod="glm", e\_sigma=4, N\_s=100)  
ag8 <- generate\_response\_smooth(J=50, mod="glm", e\_sigma=8, N\_s=50)  
ag88 <- generate\_response\_smooth(J=50, mod="glm", e\_sigma=8, N\_s=100)  
  
al2 <- generate\_response\_smooth(J=50, mod="lm", e\_sigma=2, N\_s=50)  
al22 <- generate\_response\_smooth(J=50, mod="lm", e\_sigma=2, N\_s=100)  
al4 <- generate\_response\_smooth(J=50, mod="lm", e\_sigma=4, N\_s=50)  
al44 <- generate\_response\_smooth(J=50, mod="lm", e\_sigma=4, N\_s=100)  
al8 <- generate\_response\_smooth(J=50, mod="lm", e\_sigma=8, N\_s=50)  
al88 <- generate\_response\_smooth(J=50, mod="lm", e\_sigma=8, N\_s=100)  
  
  
bg2 <- generate\_response(J=50, mod="glm", e\_sigma=2, N\_s=50)  
bg22 <- generate\_response(J=50, mod="glm", e\_sigma=2, N\_s=100)  
bg4 <- generate\_response(J=50, mod="glm", e\_sigma=4, N\_s=50)  
bg44 <- generate\_response(J=50, mod="glm", e\_sigma=4, N\_s=100)  
bg8 <- generate\_response(J=50, mod="glm", e\_sigma=8, N\_s=50)  
bg88 <- generate\_response(J=50, mod="glm", e\_sigma=8, N\_s=100)  
  
bl2 <- generate\_response(J=50, mod="lm", e\_sigma=2, N\_s=50)  
bl22 <- generate\_response(J=50, mod="lm", e\_sigma=2, N\_s=100)  
bl4 <- generate\_response(J=50, mod="lm", e\_sigma=4, N\_s=50)  
bl44 <- generate\_response(J=50, mod="lm", e\_sigma=4, N\_s=100)  
bl8 <- generate\_response(J=50, mod="lm", e\_sigma=8, N\_s=50)  
bl88 <- generate\_response(J=50, mod="lm", e\_sigma=8, N\_s=100)

### Example 1  
  
## GLM\_2\_50  
ag2 <- generate\_response\_smooth(J=50, mod="glm", e\_sigma=2, N\_s=50)  
# generation  
beta\_hat <- NULL  
for(i in 1:50){  
 results <- granular(unlist(ag2$x\_list[i]), unlist(ag2$y[i]), mod = "glm")  
 beta\_hat <- rbind(beta\_hat,results)  
}  
  
K <- make\_K(ag2$z)  
  
# multilevel  
  
GCV\_vec <- NULL  
lambda <- seq(0.00005, 0.00012, by = 1e-6)  
for(i in 1:length(lambda)){  
 EM\_out <- main\_EM(beta\_hat\_vec = unlist(beta\_hat[,1]), V = diag(unlist(beta\_hat[,2])), K = K, lambda[i])  
 GCV\_vec <- rbind(GCV\_vec,EM\_out$GCV)  
}  
  
plot(lambda, GCV\_vec)



EM\_out <- main\_EM(beta\_hat\_vec = unlist(beta\_hat[,1]), V = diag(unlist(beta\_hat[,2])), K = K, lambda = 0.0001)  
  
RMSE\_Multilevel <- sqrt((1/50)\*t(ag2$true\_mu-EM\_out$mu)%\*%(ag2$true\_mu-EM\_out$mu))  
  
# naive  
  
GCV\_vec <- NULL  
lambda <- seq(0.00016, 0.00017, by = 1e-7)  
for(i in 1:length(lambda)){  
 naive\_out <- naive\_ss(beta\_hat\_vec = unlist(beta\_hat[,1]), lambda = lambda[i], K = K)  
 GCV\_vec <- rbind(GCV\_vec,naive\_out$GCV)  
}  
  
plot(lambda, GCV\_vec)

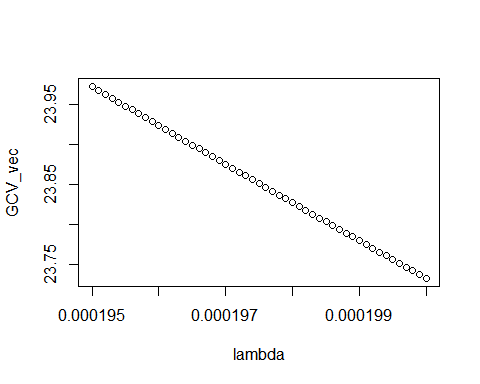


naive\_out <- naive\_ss(beta\_hat\_vec = unlist(beta\_hat[,1]), lambda = 0.000168, K = K)  
  
RMSE\_Naive <- sqrt((1/50)\*t(ag2$true\_mu-naive\_out$mu)%\*%(ag2$true\_mu-naive\_out$mu))

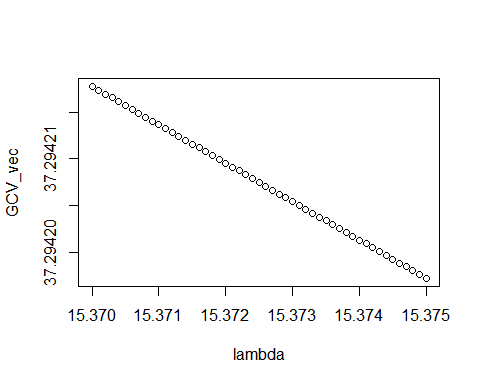
### Example 2  
  
## GLM\_2\_50  
bg2 <- generate\_response(J=50, mod="glm", e\_sigma=2, N\_s=50)  
  
# generation  
  
beta\_hat <- NULL  
for(i in 1:50){  
 results <- granular(unlist(bg2$x\_list[i]), unlist(bg2$y[i]), mod = "glm")  
 beta\_hat <- rbind(beta\_hat,results)  
}

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred  
  
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

K <- make\_K(bg2$z)  
  
# multilevel  
  
GCV\_vec <- NULL  
lambda <- seq(0.000195, 0.0002, by = 1e-7)  
for(i in 1:length(lambda)){  
 EM\_out <- main\_EM(beta\_hat\_vec = unlist(beta\_hat[,1]), V = diag(unlist(beta\_hat[,2])), K = K, lambda[i])  
 GCV\_vec <- rbind(GCV\_vec,EM\_out$GCV)  
}  
  
plot(lambda, GCV\_vec)



EM\_out <- main\_EM(beta\_hat\_vec = unlist(beta\_hat[,1]), V = diag(unlist(beta\_hat[,2])), K = K, lambda = 0.000197)  
  
RMSE\_Multilevel <- sqrt((1/50)\*t(bg2$true\_mu-EM\_out$mu)%\*%(bg2$true\_mu-EM\_out$mu))  
  
# naive  
  
GCV\_vec <- NULL  
lambda <- seq(15.37, 15.375, by = 0.0001)  
for(i in 1:length(lambda)){  
 naive\_out <- naive\_ss(beta\_hat\_vec = unlist(beta\_hat[,1]), lambda = lambda[i], K = K)  
 GCV\_vec <- rbind(GCV\_vec,naive\_out$GCV)  
}  
  
plot(lambda, GCV\_vec)



naive\_out <- naive\_ss(beta\_hat\_vec = unlist(beta\_hat[,1]), lambda = 15.374, K = K)  
  
RMSE\_Naive <- sqrt((1/50)\*t(bg2$true\_mu-naive\_out$mu)%\*%(bg2$true\_mu-naive\_out$mu))

## problem

Setting : Model GLM,

We can get s multilevel approach suffers under GLM.

We can get s naive approach suffers under GLM.