Hwang’s mlspline

#’ generate simulated response for multilevel splines

#’

#’ Generates simulated response for multilevel splines

#’

#’ @author YD Hwang and ER Lee

#’ @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

```r
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).

#’ @author YD Hwang and ER Lee

#’ @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…

#’ @author YD Hwang and ER Lee

#’ @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

#' @author YD Hwang and ER Lee

#' @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

#’

#’ @author YD Hwang and ER Lee

#’ @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)
```

```r
  }
  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)
}
```