Project for P-spline and Multilevel

Choi TaeYoung

2020-08-06

Contents

1 필요한 패키지

```r
#' generate simulated response for multilevel splines
#'
#' Generates simulated response for multilevel splines
#'
#' @author YD Hwang \email{yhwang@@g.skku.edu} and ER Lee \email{erlee@@skku.edu}
#' @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{
#'   \item{x_list}{the length-J list of design matrices. The nrow of each element is between N_s and N_
#'   \item{y_list}{the length-J list of response vectors. The length of each element is between N_s and
#'   \item{e_list}{the length-J list of error vectors. The length of each element is between N_s and N_
#'   \item{true_mu}{the true mu vector of length J}
#'   \item{z}{the grid vector of length J}
#' }
#' @export

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
#' @author YD Hwang \email{yhwang@@g.skku.edu} and ER Lee \email{erlee@@skku.edu}
#' @param x_k design matrix
#' @param y_k response vector
#' @param mod underlying model; either `lm` or `glm`
#' @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
}
```

```r
#' 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 \email{yhwang@@g.skku.edu} and ER Lee \email{erlee@@skku.edu}
#' @param z Mid-interval value vector, it is safe to assume this to be equi-distant, but in principle i
#' @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 \email{yhwang@@g.skku.edu} and ER Lee \email{erlee@@skku.edu}
#' @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 \email{yhwang@@g.skku.edu} and ER Lee \email{erlee@@skku.edu}
#' @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 \email{yhwang@@g.skku.edu} and ER Lee \email{erlee@@skku.edu}
```

```r
#' @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{
#' This function is supposed to be combined with the other generation function.. but later.
#'   \item{x_list}{the length-J list of design matrices. The nrow of each element is between N_s and N_
#'   \item{y_list}{the length-J list of response vectors. The length of each element is between N_s and
#'   \item{e_list}{the length-J list of error vectors. The length of each element is between N_s and N_
#'   \item{true_mu}{the true mu vector of length J}
#'   \item{z}{the grid vector of length J}
#' }
#' @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)
}

# GetDiffMatrix() ----------------------------------------------------------
```

```r
#' Get Difference Matrix
#'
#' Calculates the difference matrix of order 2 (ISSUE -- Only gets difference
#'   matrix of order 2 for now).
#'
#' @param nr Number of rows the difference matrix will have.
#' @param ord Order of the difference matrix.
#'
#' @return A difference matrix of order \code{ord} with dimensions \code{nr}
#'   by \eqn{\code{nr} + \code{ord}}
#'
#' @export
#'
```

```r
GetDiffMatrix <- function(nr, ord = 2) {
  # Set up difference matrix
  D <- matrix(0, nrow = nr, ncol = nr + ord)

  # Determine first row
  FRow <- c(c(1, -2, 1), rep(0, nr + ord - 3))

  # Fill in diff. matrix
  for (i in 1:nr) D[i, ] <- CPerm(FRow, i - 1)

  return(D)
}
# CPerm() -----------------------------------------------------------------
#' Cyclic Permuatation
#'
#' Get cyclic permutation of vector. Can can the function multiple times
#'   through the use of the \code{i} argument.
#'
#' @param x Vector to be cycled through
#' @param i Number of cyclic permutations
#'
#' @return Vector with all elements moved \code{i} spaces to the left with
#'   end elements wrapping around.
#'
#' @export
#'
CPerm <- function(x, i = 1) {
  if(i == 0) {
    return(x)
  } else {
    LastElement <- utils::tail(x,1)
    CycledX <- c(LastElement, utils::head(x,-1))
    return(CPerm(CycledX, i - 1))
  }
}
# GetBSpline() ------------------------------------------------------------
#' Get B-Spline Matrix
#'
#' Generates B-Spline functions over some parameters and places such functions
#'   into columns of a \eqn{n} by \eqn{(m + deg + 1)} matrix.
```

```r
#'
#' @param x Range of values to define the function over.
#' @param deg Degree of the desired B-Spline.
#' @param IntKnots Interior knots that partially define the B-Spline.
#' @param ExtKnots Exterior knots, often \code{ExtKnots = c(min(x),max(x))}.
#'
#' @return Matrix where \eqn{i,j}-th entry corresponds to \eqn{j}-th basis
#'   function evaluated at \eqn{i}-th data point.
#'
#' @export
#'
GetBSpline <- function(x, deg = 3, IntKnots, ExtKnots) {
  # Augment exterior knots around interior knots
  AugKnots <- c(rep(ExtKnots[1], deg + 1), IntKnots, rep(ExtKnots[2], deg + 1))

  # Expect m+k basis functions, call this integer "NumF"
  NumF <- length(IntKnots) + (deg + 1)

  # Fill matrix columns with basis functions
  B <- matrix(0, length(x), NumF)
  for (i in 1:NumF) B[,i] <- PSplinesR::GetBasis(x, deg, AugKnots, i)

  # Manually add in boundary to final basis function
  if(any(x == ExtKnots[2])) B[x == ExtKnots[2], NumF] <- 1
  return(B)
}
#library(devtools)
#devtools::install_github("nclJoshCowley/PSplinesR", force = T)

#' Main EM function using pspline
#'
#'
```

```r
main_EM_p <- function(beta_hat_vec, V, B, D, lambda, maxit = 5000) {

  # parameter initilization
  eps <- 10  # convergence tracker
  tol <- 1e-07  # 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 <- B%*%solve(t(B)%*%D_m%*%B + lambda * t(D)%*%D,tol=1e-30) %*%t(B)%*% D_m %*% beta_hat_vec

    S_lambda <- B%*%solve(t(B)%*%D_m%*%B + lambda * t(D)%*%D,tol=1e-30) %*%t(B)%*% 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 using p-spline
#'
#'
naive_ss_p <- function(beta_hat_vec, lambda, B, D) {

  J <- length(beta_hat_vec)
  I <- diag(J)
  Sn_lambda <- B%*%solve(t(B)%*%B + lambda*t(D)%*%D) %*% t(B)
  f_hat <- Sn_lambda %*% beta_hat_vec

  eff_df <- sum(diag(Sn_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, Sn_lambda = Sn_lambda, BIC = BIC, GCV = GCV)
  out
}
```

2 데이터

- Y data : Y데이터의 경우 134주(2018년 1월 ~ 2020년 7월)동안의 카카오로 “여행”을 검색한 횟수를 지역별로 나타냄
- X data : X데이터의 경우 17개의 지역별 인구수

3 데이터 정리 및 Goodness of fit test를 통한 적절한 모델 찾기

- X, Y 데이터 모두 리스트화를 거쳤다. Y데이터가 허들모델이라는 가정으로 각 행마다 0이 얼마나 포함되어 있는지 알아보았다.
- 그 결과 2, 14, 48행 이외에는 0을 포함하지 않아서 허들모델이나 zero inflated 방법을 이용하여 모델을 적합할 수 없었다.
- 그래서 우리는 Goodness of Fit(GoF)를 이용하여 일반화 선형모형의 적합도를 검정해보았다.

```
x_list <- x_pop %>% as.data.frame() %>% unlist() %>% as.list()

#How many zero in y_list?
y_list <- obs_y[-1,-1] %>% t() %>% as.data.frame()
y_zero <- NULL
for(m in 1:134){
  zero <- NULL
  zero <- length(which(y_list[m] == 0))/17
  y_zero <- rbind(y_zero,zero)

  zero_count <- length(which(y_zero > 0))
  zero_where <- which(y_zero > 0)
  zero_count
  zero_where
}
zero_count
```

```
## [1] 15
```

3.1 GoF 결과

- 그 결과 포아송 GoF는 모두 0으로 나왔으며, 음이항분포 GoF는 낮은 값을 보였다. 즉, 포아송분포를 사용하였을 때 과대산포가 발생하므로, 음이항분포를 이용하여 모형적합을 시도했다.

```r
#GOF Calculate
goodness <- NULL
  for(m in 1:134){
    result1_out <- NULL
    result2_out <- NULL
    results1 <- glm(unlist(y_list[m]) ~ unlist(x_list), family = poisson, maxit=1000)
    results2 <- glm.nb(unlist(y_list[m]) ~ unlist(x_list), maxit=1000)


    poi_GOF <- 1 - pchisq(summary(results1)$deviance,
           summary(results1)$df.residual
           )
    nb_GOF <- 1 - pchisq(summary(results2)$deviance,
           summary(results2)$df.residual
           )
    out <- cbind(poi_GOF,nb_GOF)
    goodness <- rbind(goodness, out)
  }

tail(goodness)
```

```
##              poi_GOF     nb_GOF
## [129,] 8.205658e-12 0.08855233
## [130,] 2.016209e-03 0.29961296
## [131,] 1.309578e-05 0.26798184
## [132,] 1.143654e-07 0.27205837
## [133,] 3.545785e-10 0.26672456
## [134,] 1.316112e-05 0.22556052
```

4 Multilevel 모델에 적용

- 논문의 방법인 EM알고리즘을 통해 multilevel spline 방법으로 최적의 μ 벡터를 찾았다.

```r
x_list <- x_pop %>% as.data.frame() %>% unlist() %>% as.list()
y_list <- obs_y[-1,-1] %>% t() %>% as.data.frame()
#multilevel

  #beta_hat_vector 구하기

  grain_out <- NULL
  J=134
  beta_hat <- NULL
  for(m in 1:134){
    result2_out <- NULL
    results2 <- glm.nb(unlist(y_list[m]) ~ unlist(x_list), maxit=1000)
      kth_beta_hat <- coef(results2)[2]
      kth_var <- diag(vcov(results2))[2]
      grain_out <- list(kth_beta_hat, kth_var)
      grain_out
    beta_hat <- rbind(beta_hat,grain_out)
  }
```

- p-spline 기법을 활용하여 새롭게 짠 코드로 리얼데이터에 적용

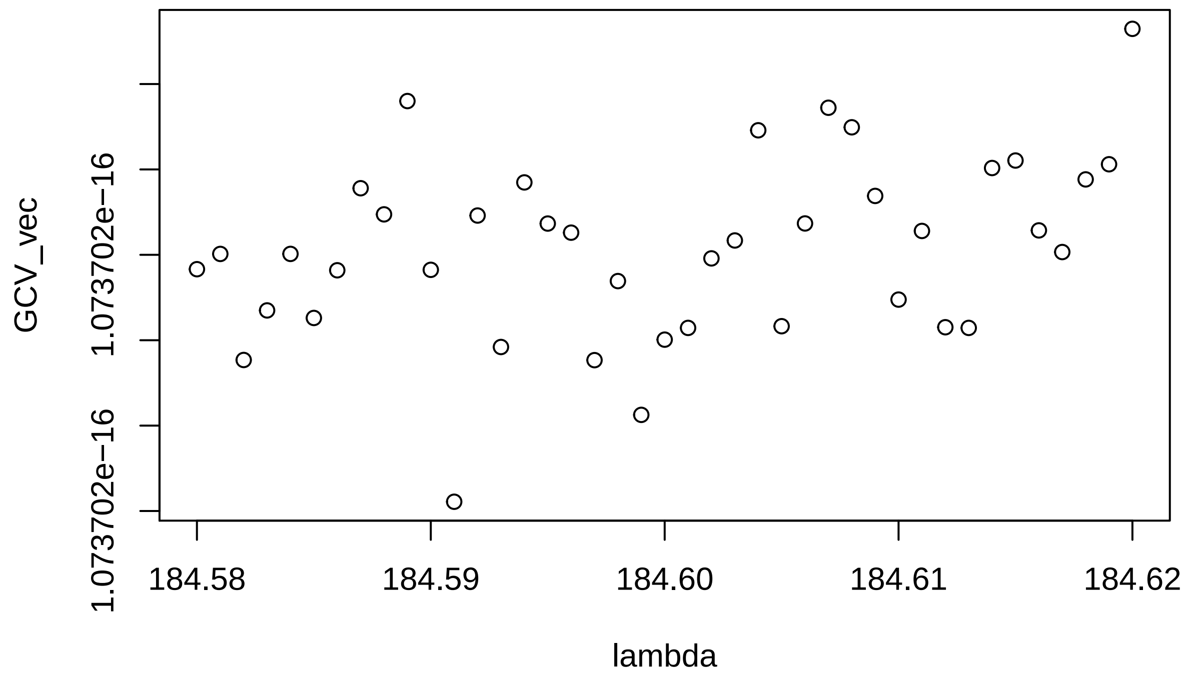
```
# lambda <- c(1e-09,1e-08,1e-07,1e-06,1e-05,1e-04,1e-03,1e-02,0.1,1, 10,100,1000,10000)
# GCV_vec <- NULL
#
n <- length(unlist(beta_hat[,1]))
IK <- unlist(beta_hat[,1])[-c(1, n)]
EK <- unlist(beta_hat[,1])[c(1, n)]
B = GetBSpline(unlist(beta_hat[,1]), deg = 3, IK, EK)
#
# for(i in 1:length(lambda)){
# EM_out <- main_EM_p(beta_hat_vec = unlist(beta_hat[,1]), V = diag(unlist(beta_hat[,2])),
#                     B=GetBSpline(unlist(beta_hat[,1]), deg = 3, IK, EK), D=GetDiffMatrix(dim(B)[1], 2
# GCV_vec <- rbind(GCV_vec,EM_out$GCV)
# }
#
# lambda[which.min(GCV_vec)]
```

- lambda[which.min(GCV_vec)]을 실행할 때, 100이 나온다.
- 그래서 100근처에서 GCV벡터를 더 찾아보기로 한다.

```
lambda <- seq(184.58, 184.62, by=.001)
GCV_vec <- NULL

for(i in 1:length(lambda)){
EM_out <- main_EM_p(beta_hat_vec = unlist(beta_hat[,1]), V = diag(unlist(beta_hat[,2])),
                    B=GetBSpline(unlist(beta_hat[,1]), deg = 3, IK, EK), D=GetDiffMatrix(dim(B)[1], 2),
GCV_vec <- rbind(GCV_vec,EM_out$GCV)
}


plot(lambda, GCV_vec)
```



- 최적의 GCV_vec로 EM_out구하기 <- mu_hat 구함

```
lambda[which.min(GCV_vec)]
```

```
## [1] 184.591
```

```
EM_out <- main_EM_p(beta_hat_vec = unlist(beta_hat[,1]), V = diag(unlist(beta_hat[,2])),
                    B=GetBSpline(unlist(beta_hat[,1]), deg = 3, IK, EK), D=GetDiffMatrix(dim(B)[1], 2),
tail(EM_out$mu)
```

```
##                [,1]
## [129,] 3.882265e-08
## [130,] 1.591944e-08
## [131,] 5.279677e-08
## [132,] 2.697011e-08
## [133,] 3.877374e-08
## [134,] 2.594444e-08
```

- Multilevel과 성능을 비교하기위해서 Naive한 방법으로 구해보자.
- Naive기법 역시 P-spline으로 코드를 짠 후 실행했다.

```
#naive
GCV_vec <- NULL
lambda <- c(0.1,1,10,100,1000, 10^4, 10^5, 10^6)
for(i in 1:length(lambda)){
  naive_out <- naive_ss_p(beta_hat_vec = unlist(beta_hat[,1]), B=GetBSpline(unlist(beta_hat[,1]), deg =
  GCV_vec <- rbind(GCV_vec,naive_out$GCV)
}

lambda[which.min(GCV_vec)]
```

```
## [1] 1000
```

4.1 Naive's GCV vector 찾기

- Naive 역시 비슷한 방법으로 풀어나간다.

```
# lambda <- seq(3190, 3210, by=1)
# GCV_vec <- NULL
# for(i in 1:length(lambda)){
#   naive_out <- naive_ss_p(beta_hat_vec = unlist(beta_hat[,1]), lambda = lambda[i],
#                           B=GetBSpline(unlist(beta_hat[,1]), deg = 3, IK, EK), D=GetDiffMatrix(dim(B)[
#   GCV_vec <- rbind(GCV_vec,naive_out$GCV)
# }
# plot(lambda, GCV_vec)

lambda[which.min(GCV_vec)]
```

```
## [1] 1000
```

```
naive_out <- naive_ss_p(beta_hat_vec = unlist(beta_hat[,1]), B=GetBSpline(unlist(beta_hat[,1]), deg = 3


tail(naive_out$mu)
```

```
##                [,1]
## [129,] 4.242683e-09
## [130,] 1.558140e-08
## [131,] 6.497480e-09
## [132,] 3.664972e-08
## [133,] 1.097519e-08
## [134,] 3.869234e-08
```

5 그래프

```
# hat_all
single_beta <- unlist(beta_hat[,1]) %>% as.vector()
mu_z_naive <- naive_out$mu %>% as.vector()
mu_z_multi <- EM_out$mu %>% as.vector()

hat_all <- cbind(mu_z_naive,mu_z_multi,single_beta) %>% as.data.frame

test_mon <- fread("S:\\Teo\\OneDrive - 성균관대학교\\tr_search_y.csv")
```

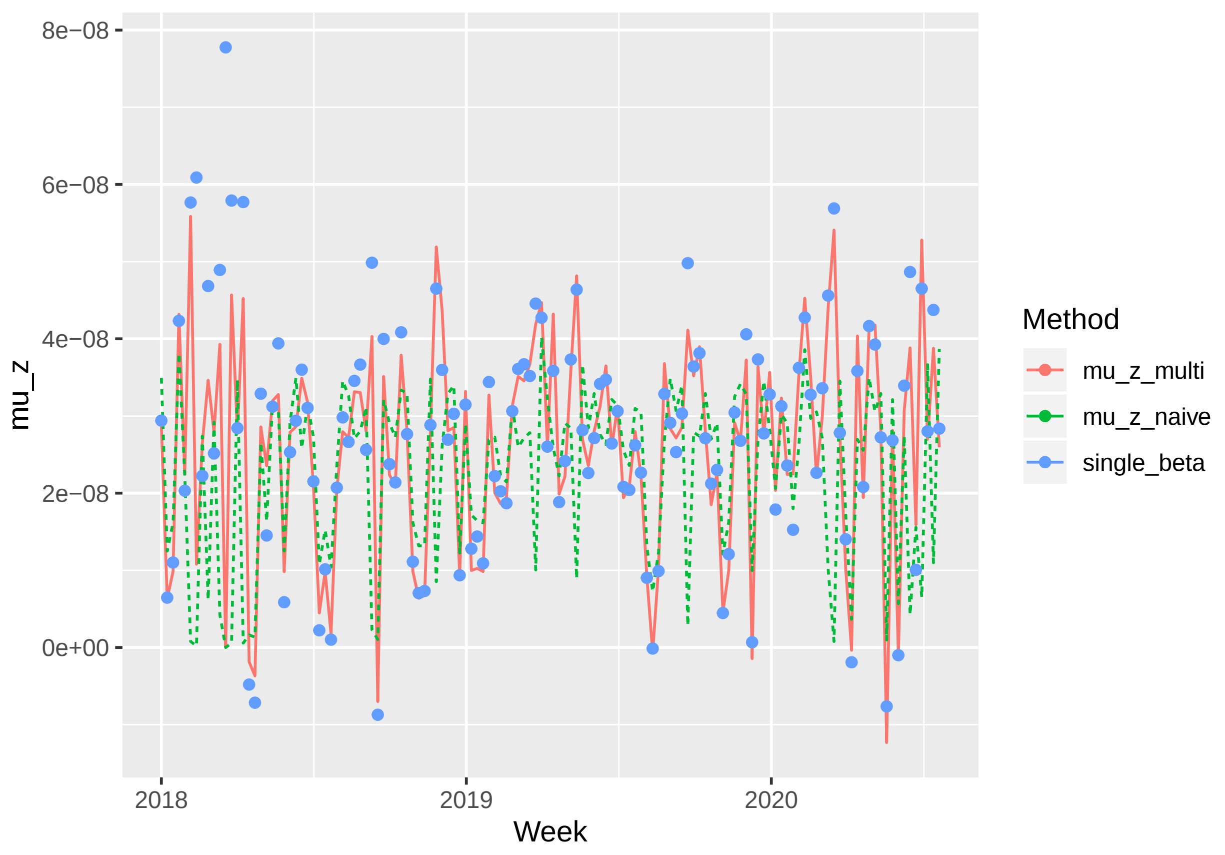
```r
test_mon <- test_mon[-1,1]
hat_all <- cbind(test_mon,hat_all)
hat_all <- rename(hat_all, Week = V1)

hat_all$Week <- parse_date_time(hat_all$Week, "ymd")
hat_all$Week <- as.Date(hat_all$Week, format="%Y-%m-%d")
hat_all <- as.data.frame(hat_all)
hat_all <- hat_all %>% mutate_if(is.character,parse_number)

# gather 사용
df1 <- gather(hat_all[, c("Week", "mu_z_naive", "mu_z_multi")],
             key = "Method", value = "mu_z", -Week)

df2 <- cbind(test_mon,single_beta)
df2 <- rename(df2, Week =V1)
df2$Week <- parse_date_time(df2$Week, "ymd")
df2$Week <- as.Date(df2$Week, format="%Y-%m-%d")
df2 <- as.data.frame(df2)
df2 <- df2 %>% mutate_if(is.character,parse_number)


g <- ggplot(df1) +
  geom_path(aes(x = Week, y = mu_z, color = Method, linetype = Method)) +
  geom_point(data=df2, aes(x = Week, y = single_beta, color = "single_beta")) +
  guides(linetype = "none") +
  scale_color_discrete(name = "Method")
g
```


8e−08
6e−08
4e−08
2e−08
0e+00
mu_z
2018
2019
2020
Week
Method
mu_z_multi
mu_z_naive
single_beta