$$C(x) = \sum_{i=1}^{I} a_i B_{i,d}(x), \quad \text{where } I = K+d+1,$$

K is number of knots

d is degree of piecewise poly

a_i is control point.

No penalty

B-spline power basis $B_{i,d}(x)$

$$1 \quad x \quad \cdots \quad x^d, \quad (x-\varepsilon_1)^d, \quad \cdots, \quad (x-\varepsilon_K)^d$$

$$\hat{a}_i = \underset{a_i}{\arg\min} \sum_{j=1}^{J} \left\{ y_j - \sum_{i=1}^{I} a_i B_{i,d}(x_j) \right\}^2 = \arg\min \sum_{j=1}^{J} \left\{ y_j - C(x_j) \right\}^2$$

$$\hat{C}(z) = \sum_{i=1}^{I} \hat{a}_i B_{i,d}(z)$$

$y_j = \mu(z_j) + \varepsilon_j$, z_j is sparse time.

$$C(x) = \sum_{i=1}^{I} a_i B_{i,d}(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_{2+d}(x-\varepsilon_1)^d + \cdots + a_{K+d+1}(x-\varepsilon_K)^d$$

$$\mu(z) \approx \sum_{j=1}^{I} a_j B_j(z)$$

$\underset{n\times 1}{y} = (y_1, \cdots, y_n)^T$, $n \times I$ matrix $\mathbb{B}$, $\alpha = (a_1 \quad \cdots \quad a_I)^T$

$$SSE(\alpha^*) = (y - \mathbb{B}\alpha)^T (y - \mathbb{B}\alpha)$$

$$\hat{\alpha}^* = (\mathbb{B}^T \mathbb{B})^{-1} \mathbb{B}^T y$$

$$\hat{\mu}(z) = B(z)\hat{\alpha}^* = B(z)(\mathbb{B}^T\mathbb{B})^{-1}\mathbb{B}^T y$$

P-spline $= \sum (y_i - \mu(z_i))^2 + \lambda \int [D^2 \mu(z)]^2 dz$

$$\int (D^2 \mu(z))^2 dz = \int \alpha^T [D^2 B(z)][D^2 B(z)]^T \alpha \, dz = \alpha^T R_2 \alpha$$

$[R_2]_{jk} = \int [D^2 B(z)][D^2 B(z)] dz$ is penalty matrix.

$$\hat{\alpha} = [\mathbb{B}^T\mathbb{B} + \lambda R_2]^{-1} \mathbb{B}^T y$$

$\hat{y} = \mathbb{B}[\mathbb{B}^T\mathbb{B} + \lambda R_2]^{-1}\mathbb{B}^T y = \ell(\lambda) y \longrightarrow$ Linear Smoother.

$$\hat{a}_i = \underset{a_i}{\arg\min} \sum_{j=1}^{n} \left\{ y_j - \sum_{i=1}^{I} a_i B_{i,d}(x_j) \right\}^2 + \lambda \int_{x_{min}}^{x_{max}} \left\{ \sum_{i=1}^{I} a_i B''_{i,d}(x) \right\}^2 dx$$

EM Algorithm

$h(\hat{\beta} | \beta) \sim AN(\beta_j, V_j) \longrightarrow V_j$ is fisher information.

$(\beta | \alpha, \sigma^2) \sim N(N\alpha, \Sigma)$

$(\alpha | \sigma^2) \sim N(0, \sigma^2 \lambda^{-1} W^{-1})$

$$\left(\boldsymbol{\beta} \mid \hat{\boldsymbol{\beta}}; \boldsymbol{\theta}, \sigma\right) \sim N\left(\boldsymbol{\beta}^*, \boldsymbol{V}^*\right),$$

$$\boldsymbol{\beta}^* = \left(\boldsymbol{V}^{-1} + \boldsymbol{\Sigma}^{-1}\right)^{-1} \left(\boldsymbol{V}^{-1}\hat{\boldsymbol{\beta}} + \boldsymbol{\Sigma}^{-1}\boldsymbol{N}\boldsymbol{\theta}\right) \text{ and } \boldsymbol{V}^* = \left(\boldsymbol{V}^{-1} + \boldsymbol{\Sigma}^{-1}\right)^{-1}$$

$$\underset{n \times I}{B_{i,d}(x)} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d & (x_1 - \varepsilon_1)^d & \cdots & (x_1 - \varepsilon_K)^d \\ \vdots & \vdots & \vdots & & & \ddots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^d & (x_n - \varepsilon_1)^d & \cdots & (x_n - \varepsilon_K)^d \end{pmatrix}$$

$\hat{\alpha}$ is coefficient estimators matrix of penalized B-spline B

$$\hat{\alpha}^{(m+1)} = (N^T N + \lambda W)^{-1} N^T (V^{-1} + \Sigma^{-1})^{-1} (V^{-1} \hat{y} + \Sigma^{-1} N \hat{\theta}^{(m)})$$

$$\hat{\alpha} = \left[N^T N + \lambda W - N^T (V^{-1} + \Sigma^{-1})^{-1} \Sigma^{-1} N \right]^{-1} N^T \underbrace{(V^{-1} + \Sigma^{-1})^{-1}}_{V^*} V^{-1} \hat{y}$$

$$= \left[N^T \left(\underline{I - (V^{-1} + \Sigma^{-1})^{-1} \Sigma^{-1}} \right) N + \lambda W \right]^{-1} N^T \underline{(V^{-1} + \Sigma^{-1})^{-1} V^{-1}} \hat{y}$$

$$\because D = I - V^* \Sigma^{-1} = V^* (V^{*-1} - \Sigma^{-1}) = V^* V^{-1}$$

$$= \left[N^T D N + \lambda W \right]^{-1} N^T D \hat{y}$$

$$\hat{\mu}(z) = \sum_{i=1}^{I} l_{\lambda,i}(z) \cdot \hat{y}_i$$

#' Builds `granular'' data

#' 기울기, 기울기의 분산 구하기

#' 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
}
```

$D^T W^{-1} D$

커널 fnc 생성