SDS 383D: Exercises 3 – Linear smoothing and Gaussian processes

February 22, 2017

Professor Scott

Spencer Woody

Problem 1

Basic Concepts

(A)

Problem 2

Curve fitting by linear smoothing

In this problem, consider a general nonlinear regression with one predictor and one response, $y_i = f(x_i) + \epsilon_i$, where ϵ_i are mean-zero random variables.

(A) For now, consider a linear regression on a response y_i with one predictor x_i, and both y_i and x_i have had their means subtracted, so the $y_i = \beta x_i + \epsilon_i$. Define $S_x := \sum_{i=1}^n x_i^2$. The least squares estimate for the coefficient, from Exercises 1, is

$$\begin{aligned}\hat{\beta} &= (X^T X)^{-1} X^T y \\ &= (x^T x)^{-1} x^T y \\ &= \frac{\sum_{i=1}^n x_i \cdot y_i}{\sum_{i=1}^n x_i^2} \\ &= \frac{\sum_{i=1}^n x_i \cdot y_i}{S_x} \\ &= \sum_{i=1}^n \frac{x_i}{S_x} \cdot y_i.\end{aligned}$$

So now our prediction $y^\star | x^\star$ is,

$$\begin{aligned}\hat{y}^\star &= \hat{f}(x^\star) \\ &= \hat{\beta} x^\star \\ &= \left(\sum_{i=1}^n \frac{x_i}{S_x} \cdot y_i \right) \cdot x^\star \\ &= \sum_{i=1}^n \left(\frac{x_i}{S_x} \cdot x^\star \right) \cdot y_i,\end{aligned}$$

which we recognize as being in the form of the general *linear smoother*

$$\hat{f}\,(x^\star) = \sum_{i=1}^n w(x_i, x^\star) \cdot y_i$$

for some weight function $w(x_i, x^\star)$. In particular, the weight function for linear regression gives weight to each y_i proportional to the value of x_i. Contrast this with the k-nearest neighbors smoothing weight function,

$$w_K(x_i, x^\star) = \begin{cases} 1/K & \text{if } x_i \text{ is one of the K closest sample points to } x^\star \\ 0 & \text{otherwise} \end{cases},$$

which gives *equal* weight to y_is but *only* to the k-nearest neighbors of $x^\star$.

(B) Now we have the very general weight function

$$w(x_i, x^\star) = \frac{1}{h} \cdot K\left(\frac{x_i - x^\star}{h}, \right)$$

where $K(\bullet)$ is some kernel function. The script `myfuns03.R` in the appendix shows an R function for linear smoothing, as well functions for the uniform and Gaussian kernels. Figure 1 shows an example of smoothing with a bandwidth of 0.75 for a cubic function $f(x)$ with iid residuals from the $\mathcal{N}(0, 15^2)$ distribution.

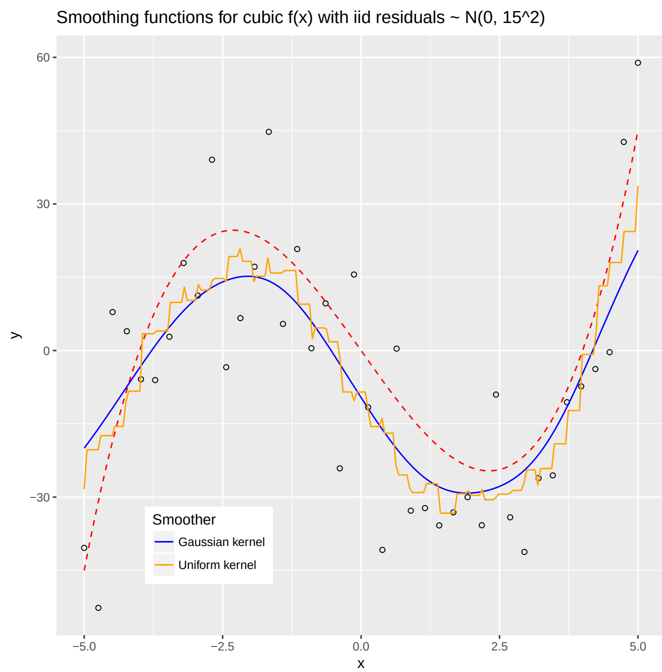

Figure 1: Uniform and Gaussian kernel smoothing for $y = f(x) + e$, $f(x) = x(x-4)(x+4)$, $h = 0.75$

Problem 3

Cross validation

(A) See attached R code for a script to return prediction error estimates for smoothing given a specified choice of bandwidth, h.

(B) For this exercise, I produced 500 data points on the x-space $[0, 1]$ from a sine function $f(x)$ with a given period and set the amplitude, and added Gaussian noise with a given standard deviation. Then I used 5-fold cross validation to select the optimal bandwidth for that given period and standard deviation of noise term. Figure 2 shows the optimal bandwidths for period ranging from 0.1 to 1, and standard deviation ranging from 0.001 to 0.5, and Figure 3 shows four example The highest bandwidths are chosen for functions with high "wigglyness" and high noise, and the smallest bandwidths are chosen for functions with low "wigglyness" and low noise. This makes sense. As the frequency increases (i.e., period decreases) then we need a tighter bandwidth because the value of the function is fluctuating at a greater rate. As noise increases, we need a greater bandwidth to smooth out the noise. Furthermore, we can see that in all cases we recover the underlying function pretty well.

Figure 2: Optimal bandwidths for varying periods and standard deviations

(C) I'll get around to this eventually. . . .

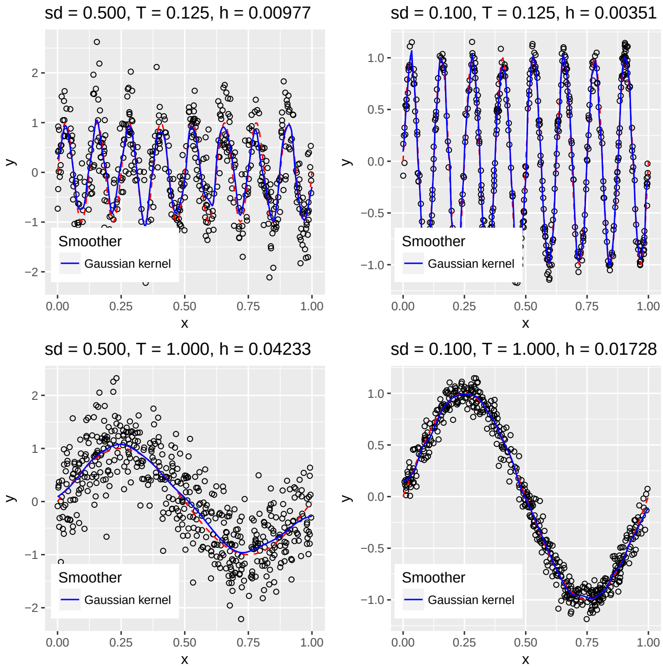

Figure 3: 2×2 example with fitted curves

Problem 4

Local polynomial regression

(A) Define

$$g_x(u; a) = a_0 + \sum_{k=1}^{D} a_k (u - x)^k \tag{1}$$

$$= \begin{cases} \sum_{j=0}^{D+1} a_j (u-x)^j & \text{if } u \neq x \\ a_0 & \text{if } u = x \end{cases}. \tag{2}$$

The coefficients of a will come from the weighted least squares problem

$$\hat{a} = \arg \min_{a \in \mathcal{R}^{D+1}} \sum_{i=1}^{n} w_i \left[y_i - g_x(x_i, a) \right]^2 \tag{3}$$

Furthermore, define the matrix R_x whose (i, j) element is $(x_i - x)^2$. Then the estimate $\hat{f}(x)$ will be $R_x \hat{a}$. The solution of $\hat{a}$ may be found with

$$\begin{aligned} \hat{a} &= \arg \min_{a \in \mathcal{R}^{D+1}} (y - R_x a)^T W (y - R_x a) \\ &= (R_x^T W R_x)^{-1} R_x^T W y \end{aligned}$$

where $W = \text{diag}(w_1, \ldots, w_n)$ following the same argument to find the WLS estimate of linear regression from Exercises 1.

(B)

(C)

(D) With H a smoothing matrix (or "hat matrix"), let $r = y - Hy$ be the vector of residuals. If the random vector x with mean vector μ and covariance matrix Σ, then $E(x^T Qx) = \mathrm{tr}(Q\Sigma) + \mu^T Q\mu$. Then,

$$E\left(||r^2||\right) = E\left((y - Hy)^T (y - Hy)\right)$$

(E)

(F)

(G)

Problem 5

Gaussian processes

(A)

Problem 6

In nonparametric regression and spacial smoothing

(A)

R code for `myfuns03.R`

```r
##########################################################
######### Created by Spencer Woody on 11 Feb 2017 #########
##########################################################

# ===========================================================================
# Linear smoothing ==========================================================
# ===========================================================================

lin.smooth <- function(x.new, x, y, kern.fun, h) {
    # ————————————————————————————————————————————————————————————————————
    # Linear smoother for some kernel function
    # ————————————————————————————————————————————————————————————————————
    # INPUTS:
    # x.new — a new point for which to estimate f(x.new)
    # x — a vector of covariates from previous observations
    # y — a vector of responses from previous observations
    # kern.fun — some kernel function (e.g. Gaussian)
    #            *** takes 2 arguments: distance (dist) and bandwidth (h)
    # h is the bandwidth for the kernel function
    # ————————————————————————————————————————————————————————————————————
    # OUTPUT:
    # weights — a vector of weights for a new observation
    # ————————————————————————————————————————————————————————————————————

    weights <- kern.fun(dist = x - x.new, h = h) / h
    weights <- weights / sum(weights)

    fit <- crossprod(weights, y)

    return(fit)
}

kern.unif <- function(dist, h) {
    # ————————————————————————————————————————————————————————————————————
    # Uniform kernel function
    # ————————————————————————————————————————————————————————————————————
    # INPUTS:
    # dist
    # h is
    # Sigma is the covariance matrix
    # ————————————————————————————————————————————————————————————————————
    # OUTPUT:
    # kern — the value of the uniform kernel function
    # ————————————————————————————————————————————————————————————————————

    kern <- ( abs(dist / h) <= 1) / 2

    return(kern)
}

kern.norm <- function(dist, h) {
    # ————————————————————————————————————————————————————————————————————
```

```
    # Gaussian (normal) kernel function
    # ---------------------------------------------------------------------
    # INPUTS:
    # dist the distance
    # h is the bandwidth
    # Sigma is the covariance matrix
    # ---------------------------------------------------------------------
    # OUTPUT:
    # kern is the value of the Gaussian kernel function
    # ---------------------------------------------------------------------

    kern <- 1 / sqrt(2 * pi) * exp(-(dist / h)^2 / 2)

    return(kern)
}

make.noise <- function(x, f, res.dist, sd = NA, scale = NA, df = NA) {
    # ---------------------------------------------------------------------
    # Simulate noisy response from some non-linear function
    # ---------------------------------------------------------------------
    # INPUTS:
    # x - the predictor values
    # f - a function for the expected value, E(y | x) = f(x)
    # res.dist - a string for distribution of errors, either
    #             "normal" for normal errors, or
    #             "cauchy" for cauchy error
    # sd - the standard deviation for residuals (for normal errors only)
    # scale - the scale paramater for residuals (for cauchy errors only)
    # ---------------------------------------------------------------------
    # OUTPUT:
    # noise - the simulated noisy responses
    # ---------------------------------------------------------------------

    if (res.dist == "normal" & !is.na(sd)) {
        noisy <- f(x) + rnorm(n = length(x), mean = 0, sd = sd)
    } else if (res.dist == "cauchy" & !is.na(scale)) {
        noisy <- f(x) + rcauchy(n = length(x), location = 0, scale = scale)
    } else {
        stop(paste("Must give res.dist argument as ",
              "either \"normal\" or \"cauchy\", AND ",
              "also specify sd (for normal) or ",
              "scale (for cauchy)", sep = ""))
    }

    return(noisy)
}

# ===========================================================================
# Cross-validation ==========================================================
# ===========================================================================

cv <- function(x.tr, y.tr, x.te, y.te, KERN.FUN, h) {
    # ---------------------------------------------------------------------
```

```
    # Give cross validation mean squared prediction error
    # ---------------------------------------------------------------------
    # INPUTS:
    # x.tr — vector of predictors in * training * set
    # y.tr — vector of responses in * training * set
    # x.tr — vector of predictors in * testing * set
    # y.tr — vector of responses in * testing * set
    # kern.fun — some kernel function (e.g. Gaussian)
    #            *** takes 2 arguments: distance (dist) and bandwidth (h)
    # h — vector or bandwidths
    # ---------------------------------------------------------------------
    # OUTPUT:
    # mse — mean square predictive error
    # ---------------------------------------------------------------------

    numbands <- length(h)
    mse <- rep(NA, numbands)

    for (i in 1:numbands) {
        y.pr <- sapply(
            x.te,
            lin.smooth,
            x = x.tr,
            y = y.tr,
            kern.fun = KERN.FUN,
            h = h[i]
            )

        mse[i] <- mean((y.te - y.pr)^2)
    }

    return(mse)
}

# =========================================================================
# Local polynomial regression =============================================
# =========================================================================

loc.lin <- function() {
    # ---------------------------------------------------------------------
    # Give local linear estimator at some new point
    # ---------------------------------------------------------------------
    # INPUTS:
    # x is some new point on x
    # x.vec — vector of x in sample
    # y.vec — vector of y in sample
    # kern.fun — some kernel function (e.g. Gaussian)
    #            *** takes 2 arguments: distance (dist) and bandwidth (h)
    # h — vector or bandwidths
    # ---------------------------------------------------------------------
    # OUTPUT:
    # mse — mean square predictive error
    # ---------------------------------------------------------------------
```

```
    kern.x <- kern.norm(x - x.vec, h)

    w.x <-
    s1 <- sum(kern.x * (x.vec - x))

}

# ============================================================================
# Gaussian process ===========================================================
# ============================================================================

my.mvn <- function(n, mu, Sigma) {
    #  Simulate n draws from MVN(mu, Sigma)
    #
    #  Note: this function assumes that X already has an intercept term
    # (or doesn't, if we want to force OLS through the origin)
    #
    # INPUTS:
    # n is the number of draws
    # mu is the mean vector
    # Sigma is the covariance matrix
    #
    # OUTPUT:
    # x is matrix of n draws from MVN(mu, Sigma) [with n rows, p columns]
    #

    # dimension of MVN
    p <- length(mu)

    # Check if inputs are valid (dimensions match, Sigma is square and p.s.d.)
    cond<- (ncol(Sigma) != p) |
           (nrow(Sigma) != p) |
           (max(eigen(Sigma)$values) <= 0)

    if (cond) {
        return("Try again...")
    }

    # Generate n*p univariate standard normal variables
    z     <- matrix(rnorm(n*p), nrow = p)

    # Create a matrix containing copies of mu
    mumat <- matrix(rep(mu, n), nrow = p)

    # Decompose Sigma into Sigma = L %*% Lt
    Lt <- chol(Sigma)

    # Generate sample with affine transformation of z
    x <- crossprod(Lt, z) + mumat

    return(t(x))
}
```

```
ell2 <- function(x) {
    # Compute the ell2 norm of x, a vector in Euclidean space

    return(sqrt(sum(x^2)))
}

C.SE <- function(x.i, x.j, params = NA) {
    # ------------------------------------------------------------------------
    #  Compute the (i, j) element of a squared exp. covariance matrix
    #
    # ------------------------------------------------------------------------
    # INPUTS:
    # x.i and x.j are two vectors in same space (need not be [0, 1])
    # params should be a vector of three hyperparameters
    #      1) b
    #      2) tau1.sq
    #      3) tau2.sq
    #
    # ------------------------------------------------------------------------
    # OUTPUT:
    # c.se is the value of the Matern-5/2 covariance matrix for x.i and x.j
    # ------------------------------------------------------------------------

    if (prod(is.na(params))) {
        return("Must have three valid parameters.")
    }

    if (length(params) != 3) {
        return("Must have three valid parameters.")
    }

    b       <- params[1]
    tau1.sq <- params[2]
    tau2.sq <- params[3]


    b       <- params[1]
    tau1.sq <- params[2]
    tau2.sq <- params[3]

    # Euclidean distance between x.i and x.j
    d <- ell2(x.i - x.j)

    c.se <- tau1.sq * exp(-0.5 * (d / b)^2) + tau2.sq * (x.i == x.j)

    return(c.se)
}

C.M52 <- function(x.i, x.j, params = NA) {
    # ------------------------------------------------------------------------
    #  Compute the (i, j) element of a Matern-5/2 covariance matrix
    #
```

```
    # ------------------------------------------------------------------------
    # INPUTS:
    # x.i and x.j are two vectors in same space (need not be [0, 1])
    # params should be a vector of three hyperparameters
    #       1) b
    #       2) tau1.sq
    #       3) tau2.sq
    #
    # ------------------------------------------------------------------------
    # OUTPUT:
    # c.m52 is the value of the Matern-5/2 covariance matrix for x.i and x.j
    # ------------------------------------------------------------------------

    if (prod(is.na(params))) {
        return("Must have three valid parameters.")
    }

    if (length(params) != 3) {
        return("Must have three valid parameters.")
    }

    b       <- params[1]
    tau1.sq <- params[2]
    tau2.sq <- params[3]

    # Euclidean distance between x.i and x.j
    d <- ell2(x.i - x.j)

    c.m52 <- tau1.sq * ( 1 + (5^0.5 * d / b) + (5 / 3 * (d / b)^2) ) *
             exp(-5^0.5 * d / b)  + tau2.sq * (x.i == x.j)

    return(c.m52)
}

make.covmat <- function(x, cov.fun, params = NA) {
    # ------------------------------------------------------------------------
    #  Compute the covariance matrix for a GP, given some cov. function
    #
    # ------------------------------------------------------------------------
    # INPUTS:
    # x is a vector of N values in [0, 1]
    # params should be a vector of three hyperparameters
    #       1) b
    #       2) tau1.sq
    #       3) tau2.sq
    #
    # ------------------------------------------------------------------------
    # OUTPUT:
    # covmat is the covariance matrix of GP
    # ------------------------------------------------------------------------

    if (prod(is.na(params))) {
        return("Must have three valid parameters.")
```

```
    }

    if (length(params) != 3) {
        return("Must have three valid parameters.")
    }

    N <- length(x)

    covmat <- matrix(nrow = N, ncol = N)

    for (j in 1:N) {
        for (i in j:N) {
            covmat[i, j] <- cov.fun(x[i], x[j], params = params)
            covmat[j, i] <- covmat[i, j]
        }
    }

    return(covmat)
}
```

R code for exercises03.R

```
##########################################################
######### Created by Spencer Woody on 11 Feb 2017 #########
##########################################################

library(ggplot2)
library(reshape2)
library(gridExtra)
library(wesanderson) # nice palettes

# Prep color palette
# pal <- wes_palette("Zissou", 5)
# col1 <- pal[5]
# col2 <- pal[4]
# col3 <- pal[1]

col1 <- "red"
col2 <- "orange"
col3 <- "blue"

source("myfuns03.R")

# ==========================================================================
# Linear smoothers (part one) ==============================================
# ==========================================================================

# nonlinear function f(x)
f1 <- function(x){
    return(x * (x - 4) * (x + 4))
}

# Predictor vector
x1 <- seq(-5, 5, length.out = 40)

# Create sequence along x-space
x.seq <- seq(min(x1), max(x1), length.out = 200)

# Response vector
y1 <- make.noise(x1, f1, "normal", sd = 15)

# Bin width
h1 <- 0.75

# Gaussian kernel smoothing
y.norm <- sapply(
    x.seq,
    lin.smooth,
    x = x1,
    y = y1,
    kern.fun = kern.norm,
    h = h1
    )
```

```
# Uniform kernel smoothing
y.unif <- sapply(
    x.seq,
    lin.smooth,
    x = x1,
    y = y1,
    kern.fun = kern.unif,
    h = h1
    )

# Make a nice plot
h <- qplot(x.seq, geom = "blank") +
xlab("x") +
ylab("y") +
ggtitle(sprintf("Smoothing of cubic function")) +
geom_point(aes(x = x1, y = y1), pch = 1) +
stat_function(fun = f1, col = col1, linetype = "dashed") +
geom_line(aes(y = y.norm, colour = "Gaussian kernel")) +
geom_line(aes(y = y.unif, colour = "Uniform kernel")) +
scale_colour_manual(name = "Smoother", values = c(col3, col2)) +
theme(legend.position = c(0.25, 0.15),
    text = element_text(family="Helvetica"))

h

pdf("firstexample.pdf")
h
dev.off()

# ===========================================================================
# Linear smoothers (cross validation) ==== make big heat map ================
# ===========================================================================

# Sample size
N = 500

# Set limits of x-space
xlo = 0
xhi = 1

# Number of bins for cross-validation
numbins <- 5

# Vector for bandwidths to search over
h.vec <- seq(0.001, 0.125, length.out = 100)

# Vector for standard deviations
s.vec <- seq(0.001, 0.5, length.out = 100)
s.vec <- rev(s.vec)

# Vector for periods
p.vec <- seq(0.1, 1, length.out = 100)

```

```
# Matrix for optimal bandwidth values
opt.h <- matrix(nrow = length(p.vec), ncol = length(s.vec))

for (i in 1:length(p.vec)) {
    # Select what the period is for this iteration
    p.i <- p.vec[i]

    # Create a new function with this current period
    mysin.i <- function(x) {
        return(sin(x * 2*pi / p.i))
    }

    for (j in 1:length(s.vec)) {
        # Select what the residual sd is for this iteration
        s.j <- s.vec[j]

        # Generate data
        x.ij <- (xhi - xlo) * runif(N) + xlo
        y.ij <- make.noise(x.ij, mysin.i, "normal", sd = s.j)

        # Prepare mse matrix for this current iteration
        mse.ij <- matrix(nrow = numbins, ncol = length(h.vec))

        # Create random partition into five bins
        jumble <- sample(1:N, N, replace = F)
        bin.indices <- split(jumble, cut(1:N, numbins))

        for (bin in 1:numbins) {
            my.indices <- bin.indices[[bin]]
            x.tr <- x.ij[-(my.indices)]
            y.tr <- y.ij[-(my.indices)]
            x.te <- x.ij[my.indices]
            y.te <- y.ij[my.indices]
            mse.ij[bin, ] <- cv(x.tr, y.tr, x.te, y.te, kern.norm, h.vec)
        }

        # Average out MSE over all bins
        mse.vec <- colMeans(mse.ij)

        # Choose bandwidth with lowest average MSE
        opt.h[i, j] <- h.vec[which.min(mse.vec)]

    }
    print(i)
}

# Check to make sure optimal bandwidths look OK
opt.h / max(opt.h)

# Plot these things
ohm <- melt(opt.h)

w <- ggplot(ohm, aes(rev(Var1), Var2, z = value)) +
```

```
ggtitle("Picking Optimal Bandwidth for Gaussian Kernel Smoothing (5-fold CV)") +
xlab("Decreasing standard deviation (0.5:0.001) (less noisy)") +
ylab("Decreasing period (1:0.1) (more wiggly)") +
geom_tile(aes(fill = value)) +
scale_fill_distiller("Bandwidth", palette = "Spectral")
w

# Save this to PDF
pdf("opth3.pdf", width = 7, height = 6)
w
dev.off()



# ===========================================================================
# Linear smoothers (cross validation) ==== make 2 x 2 plot ==================
# ===========================================================================

# Sample size
N = 500

# Set limits of x-space
xlo = 0
xhi = 1

# Number of bins for cross-validation
numbins <- 5

# Vector for bandwidths to search over
h.vec <- seq(0.001, 0.125, length.out = 100)

# Vector for standard deviations
# s.vec <- seq(0.001, 0.5, length.out = 64)
s.vec <- c(0.1, 0.5)

# Vector for periods
# p.vec <- seq(0.1, 1, length.out = 64)
p.vec <- c(0.125, 1)

# Matrix for optimal bandwidth values
opt.h <- matrix(nrow = length(p.vec), ncol = length(s.vec))

# Matrix for random x-values and y-values
x.mat <- matrix(nrow = nrow(opt.h) * ncol(opt.h), ncol = N)
y.mat <- matrix(nrow = nrow(opt.h) * ncol(opt.h), ncol = N)

# Matrix for values of f at x
x.seq <- seq(xlo, xhi, length.out = 200)
fx.mat <- matrix(nrow = nrow(opt.h) * ncol(opt.h), ncol = length(x.seq))
smooth <- matrix(nrow = nrow(opt.h) * ncol(opt.h), ncol = length(x.seq))

count <- 1
```

```r
for (i in 1:length(p.vec)) {
    # Select what the period is for this iteration
    p.i <- p.vec[i]

    # Create a new function with this current period
    mysin.i <- function(x) {
        return(sin(x * 2*pi / p.i))
    }

    for (j in 1:length(s.vec)) {
        # Select what the residual sd is for this iteration
        s.j <- s.vec[j]

        x.ij <- (xhi - xlo) * runif(N) + xlo
        y.ij <- make.noise(x.ij, mysin.i, "normal", sd = s.j)

        fx.mat[count, ] <- mysin.i(x.seq)

        x.mat[count, ] <- x.ij
        y.mat[count, ] <- y.ij

        mse.ij <- matrix(nrow = numbins, ncol = length(h.vec))

        jumble <- sample(1:N, N, replace = F)
        bin.indices <- split(jumble, cut(1:N, numbins))

        for (bin in 1:numbins) {
            my.indices <- bin.indices[[bin]]
            x.tr <- x.ij[-(my.indices)]
            y.tr <- y.ij[-(my.indices)]
            x.te <- x.ij[my.indices]
            y.te <- y.ij[my.indices]
            mse.ij[bin, ] <- cv(x.tr, y.tr, x.te, y.te, kern.norm, h.vec)
        }

        mse.vec <- colMeans(mse.ij)

        # Choose first 4/5 of x's and y's to be training, the rest of testing
        # x.tr <- x.ij[1:round(N*4/5)]
        # y.tr <- y.ij[1:round(N*4/5)]
        # x.te <- x.ij[-(1:round(N*4/5))]
        # y.te <- y.ij[-(1:round(N*4/5))]

        # mse.vec <- cv(x.tr, y.tr, x.te, y.te, kern.norm, h.vec)

        opt.h[i, j] <- h.vec[which.min(mse.vec)]

        smooth[count, ] <- y.norm <- sapply(
                            x.seq,
                            lin.smooth,
                            x = x.ij,
                            y = y.ij,
                            kern.fun = kern.norm,
```

```
                                h = opt.h[i, j]
                                )


                count <- count + 1
            }
            print(i)
        }

ess <- qplot(x.seq, geom = "blank") +
xlab("x") +
ylab("y") +
ggtitle(sprintf("sd = %5.3f, T = %5.3f, h = %5.5f", s.vec[1], p.vec[1], opt.h[1, 1])) +
geom_point(aes(x = x.mat[1, ], y = y.mat[1, ]), pch = 1) +
geom_line(aes(y = fx.mat[1, ]), col = "red", linetype = "dashed") +
geom_line(aes(y = smooth[1, ], colour = "Gaussian kernel")) +
scale_colour_manual(name = "Smoother", values = "blue") +
theme(legend.position = c(0.25, 0.15), text = element_text(family="Helvetica"))

tee <- qplot(x.seq, geom = "blank") +
xlab("x") +
ylab("y") +
ggtitle(sprintf("sd = %5.3f, T = %5.3f, h = %5.5f", s.vec[2], p.vec[1], opt.h[1, 2])) +
geom_point(aes(x = x.mat[2, ], y = y.mat[2, ]), pch = 1) +
geom_line(aes(y = fx.mat[2, ]), col = "red", linetype = "dashed") +
geom_line(aes(y = smooth[2, ], colour = "Gaussian kernel")) +
scale_colour_manual(name = "Smoother", values = "blue") +
theme(legend.position = c(0.25, 0.15), text = element_text(family="Helvetica"))

you <- qplot(x.seq, geom = "blank") +
xlab("x") +
ylab("y") +
ggtitle(sprintf("sd = %5.3f, T = %5.3f, h = %5.5f", s.vec[1], p.vec[2], opt.h[2, 1])) +
geom_point(aes(x = x.mat[3, ], y = y.mat[3, ]), pch = 1) +
geom_line(aes(y = fx.mat[3, ]), col = "red", linetype = "dashed") +
geom_line(aes(y = smooth[3, ], colour = "Gaussian kernel")) +
scale_colour_manual(name = "Smoother", values = "blue") +
theme(legend.position = c(0.25, 0.15), text = element_text(family="Helvetica"))

vee <- qplot(x.seq, geom = "blank") +
xlab("x") +
ylab("y") +
ggtitle(sprintf("sd = %5.3f, T = %5.3f, h = %5.5f", s.vec[2], p.vec[2], opt.h[2, 2])) +
geom_point(aes(x = x.mat[4, ], y = y.mat[4, ]), pch = 1) +
geom_line(aes(y = fx.mat[4, ]), col = "red", linetype = "dashed") +
geom_line(aes(y = smooth[4, ], colour = "Gaussian kernel")) +
scale_colour_manual(name = "Smoother", values = "blue") +
theme(legend.position = c(0.25, 0.15), text = element_text(family="Helvetica"))


pdf("2x2.pdf")
grid.arrange(tee, ess, vee, you)
dev.off()
```

```
# ==============================================================================
# Gaussian process =============================================================
# ==============================================================================

x.seq <- seq(0, 1, length.out = 100)

b <- 1
tau1.sq <- 1e-6
tau2.sq <- 1e-5

myparams <- c(b, tau1.sq, tau2.sq)

xCM52 <- make.covmat(x.seq, C.M52, params = myparams)
xSE <- make.covmat(x.seq, C.SE, params = myparams)


# ==============================================================================
# Extra code for CV plotting.... ===============================================
# ==============================================================================

# Cross-validation with error bars

RSS.vec <- apply(RSS.mat, 2, mean)
RSS.SE <- apply(RSS.mat, 2, sd) / sqrt(numbins)

lower = RSS.vec - RSS.SE
upper = RSS.vec + RSS.SE

# Make a plot!
pdf("perror.pdf", width = 12 / 1.25, height = 8 / 1.25)
h <- qplot(log(fit1$lambda), RSS.vec, geom = "path")
h + xlab(expression(paste("Penalization term, log(", lambda, ")"))) +
ylab("Expected prediction error") +
labs(title = sprintf("%i-fold Cross-validation, Mallow's CP, and In-sample MSE", numbins)) +
geom_ribbon(aes(ymin = lower, ymax = upper), fill = "grey80") +
geom_line(aes(y = RSS.vec, colour = "CV"), show.legend = TRUE) +
geom_line(aes(y = MSE.lasso, colour = "In-sample MSE"), show.legend = TRUE) +
geom_line(aes(y = MCP, colour = "CP"), show.legend = TRUE) +
geom_vline(xintercept = log(minlambdaCV), linetype = 3, col = "black", size = 0.75, show.legend = TRU
geom_vline(xintercept = log(minlambdaMCP), linetype = 3, col = "red", size = 0.75, show.legend = TRUE
scale_colour_manual(name = "", values = c("CV" = "black", "In-sample MSE" = "blue", "CP" = "red"))
dev.off()


numbins <- 10
jumble <- sample(1:N2, N2, replace = F)
bin.indices <- split(jumble, cut(1:N2, numbins))
```