14. Suppose that with \( p = 1 \),
\[
y|x \sim N \left( \frac{\sin(12(x + .2))}{x + .2}, 1 \right)
\]
and \( N = 101 \) training data pairs are available with \( x_i = (i - 1)/100 \) for \( i = 1, 2, \ldots, 101 \). Vardeman will send you a text file with a data set like this in it. Use it in the following.

(a) Fit all of the following using first 5 and then 9 effective degrees of freedom

(i) a cubic smoothing spline,

(ii) a locally weighted linear regression smoother based on a normal density kernel, and

(iii) a locally weighted linear regression smoother based on a tricube kernel.

Plot for 5 effective degrees of freedom all of \( y_i \) and the 3 sets of smoothed values against \( x_i \). Connect the consecutive \((x_i, \hat{y}_i)\) for each fit with line segments so that they plot as "functions." Then redo the plotting for 9 effective degrees of freedom.

Neither the R function `smooth.spline()` nor the package `pspline` returns the coefficient matrix, so I decide to implement the whole function from scratch based on the Section 4.1 of the lecture outline. (Yes, my code is really messy where using global variable is not a good idea, and I would not recommend this way except for computing efficiency.) Please let me know if you find an easy way to get \( c_i \) of the part b) from any cubic smoothing spline function or JMP. As HTF5.2, I use a truncated-power basis set easily obtained second derivatives, and let \( a = x_1 = 0, b = x_{101} = 1, \) and knots \( \xi_l = x_{l+1} \) for \( l = 1, \ldots, K \) and \( K = 99 \). Also, the basis functions for a cubic spline \( M = 4 \) are

\[
h_j(x) = x^{j-1} \quad j = 1, \ldots, M,
\]
\[
h_{M+l}(x) = (x - \xi_l)^{M-1} \quad l = 1, \ldots, K.
\]

Then, \( H = (h_j(x_i))_{N,M+K} \) where \( h_j(x_i) \) is for the \( i \)-th row and the \( j \)-th column. Let \( \Omega = (\omega_{i,j})_{M+K,M+K} \) be a symmetric matrix and the upper triangular \( \omega_{i,j} = \int_a^b h_i''(t)h_j''(t)dt \) is

\[
\omega_{i,j} = 0 \quad \text{for } i < M,
\]
\[
\omega_{M,j} = \frac{1}{3}b^3 - \frac{1}{2}b^2\xi_j + \frac{1}{6}\xi_j^3 \quad \text{for } j > M, \text{and}
\]
\[
\omega_{i,j} = \frac{1}{3}(b^3 - \xi_0^3) - \frac{1}{2}(b^2 - \xi_0^2)(\xi_{i-M} + \xi_{j-M}) + (b - \xi_0)\xi_{i-M}\xi_{j-M} \quad \text{for } j \geq i > M,
\]
where \( \xi_0 = \max\{\xi_{i-M}, \xi_{j-M}\} \).

Given a \( \lambda \),
\[
\hat{Y}_\lambda = H(H'\lambda + \lambda\Omega)^{-1}H'Y
\]
and

\[ S_\lambda \equiv H(H'H + \lambda \Omega)^{-1}H' \]

can be used to determine the degrees of freedom by \( tr(S_\lambda) \). Also, each row of \( S_\lambda \) contains the \( c_i \) for a given \( x_i \).

Use the same procedure described in the Section 5.1 of the lecture outline to perform kernel smoothers as the followings. The bandwidths \( \lambda \) for the corresponding degrees of freedom are shown in the plots.

### Generate data
```r
set.seed(1234)
N <- 101
x <- (1:N - 1) / 100
y <- rnorm(N, sin(12 * (x + .2)) / (x + .2), 1)
```

### Functions and global variables for i), I assume (N-2) interior points
```r
## as knots.
xi <- x[c(-1, -N)]
b <- x[N]
K <- length(xi)
M <- 4
H <- matrix(0, nrow = N, ncol = M + K)
for(i in 1:N){
  h.x <- x[i] - xi
  H[i,] <- c(x[i]^((1:M - 1), (h.x * (h.x > 0))^((M - 1))
}
```
Omega <- matrix(0, nrow = M + K, ncol = M + K)
tmp <- matrix(0, nrow = K, ncol = K)
for(i in 1:K){
  for(j in i:K){
    xi.0 <- max(xi[i], xi[j])
    tmp[i, j] <- (b^3 - xi.0^3) / 3 -
                 (b^2 - xi.0^2) * (xi[i] + xi[j]) / 2 +
                 (b - xi.0) * xi[i] * xi[j]
  }
}
Omega[(M + 1):(M + K), (M + 1):(M + K)] <- tmp
Omega[lower.tri(Omega)] <- t(Omega)[lower.tri(Omega)]
t.H <- t(H)
get.S.lambda <- function(lambda){
  H %*% solve(t.H %*% H + lambda * Omega) %*% t.H
}
find.lambda.spline <- function(df, range.df){
  f.df <- function(lambda, df) sum(diag(get.S.lambda(lambda))) - df
  uniroot(f.df, range.df, df)$root
}

### Functions and global variables for ii) and iii), find lambda
### corresponding to the df and D(.).
B <- cbind(1, x)
t.B <- t(B)
W.x.0 <- matrix(0, nrow = N, ncol = N)
get.K.lambda <- function(x.new, x.0, lambda, D) D((x.new - x.0) / lambda)
get.L.lambda <- function(lambda, D){
  L.lambda <- matrix(0, nrow = N, ncol = N)
  for(i in 1:N){
    for(j in 1:N){
      W.x.0[j, j] <- get.K.lambda(x[j], x[i], lambda, D)
    }
    L.lambda[i,] <- t(B[i,]) %*% solve(t.B %*% W.x.0 %*% B) %*% t.B %*% W.x.0
  }
}
find.lambda.lm <- function(df, D, range.df){
  f.df <- function(lambda, df) sum(diag(get.L.lambda(lambda, D))) - df
  uniroot(f.df, range.df, df)$root
}
D.2 <- function(t) dnorm(t)
D.3 <- function(t) (1 - abs(t)^3)^3 * (abs(t) <= 1)

### Start here. Note that partial global variables are initialized above.
df <- 5
(lambda.1 <- find.lambda.spline(df, c(0.01, 0.5)))
C.1 <- get.S.lambda(lambda.1)
y.1 <- C.1 %*% y
(lambda.2 <- find.lambda.lm(df, D.2, c(0.01, 0.5)))
C.2 <- get.L.lambda(lambda.2, D.2)
y.2 <- C.2 %*% y
(lambda.3 <- find.lambda.lm(df, D.3, c(0.1, 0.5)))
C.3 <- get.L.lambda(lambda.3, D.3)
y.3 <- C.3 %*% y
df <- 9
(lambda.1.9 <- find.lambda.spline(df, c(0.01, 0.4)))
C.1.9 <- get.S.lambda(lambda.1.9)
y.1.9 <- C.1.9 %*% y
(lambda.2.9 <- find.lambda.lm(df, D.2, c(0.01, 0.4)))
C.2.9 <- get.L.lambda(lambda.2.9, D.2)
y.2.9 <- C.2.9 %*% y
(lambda.3.9 <- find.lambda.lm(df, D.3, c(0.1, 0.4))
C.3.9 <- get.L.lambda(lambda.3.9, D.3)
y.3.9 <- C.3.9 %*% y

b) For all of the fits in a) plot as a function of i the coefficients $c_i$ applied to the observed $y_i$ in order to produce $\hat{f}(x) = \sum_{i=1}^{101} c_i y_i$ for $x = 0.05, 0.1, 0.2, 0.3$. (Make a different plot of three curves for 5 degrees of freedom and each of the values x (four in all). Then redo the plotting for 9 degrees of freedom.)

The $c_i$ can be obtained from the previous code, and the plots are in the followings.

15. The ”point and click” JMP software will fit neural nets (with logistic sigmoidal function, $\sigma(\cdot)$). It has pretty good online help files, from which you should be able to figure out how to use the
software. (If I can do it, you can do it!) Use that software to produce a neural net approximation for the data set in problem 14, with an error sum of squares about like those for the 9 degrees of freedom fits. Provide appropriate JMP reports/summaries. You’ll be allowed to vary the number of hidden nodes for a single-hidden-layer architecture and to vary a weight for a penalty made from a sum of squares of coefficients. Each run of the routine makes several random starts of an optimization algorithm. Extract the coefficients from the JMP run and use them to plot the fitted function of x that you settle on. How does this compare to the plotted fits produced in 14?

As the 9 degrees of freedom fits, the error sum of squares are 79.5594, 83.1913, 82.1246 for spline, normal, and tricube in the problem 14. I pick 80 for neural net approximation.

Save the previous JMP output in a file named "Q15a.txt" and use the following code to generate the predictor of y and draw a plot for comparison that the neural net approximation with similar error sum of squares performs similarly to other methods. The other way is to draw the plot is to utilize the JMP to produce the overlay plot for predicted values. Note that
the JMP performs neural nets on the standardized $x$ and $y$, so it has to be adjusted accordingly for plotting the predicted values.

```
R code
coef.nn <- read.table("Q15.txt", sep = "\t", header = FALSE, quote = "")
coef.x <- matrix(coef.nn[1:8, 2], ncol = 2)
coef.y <- coef.nn[9:13, 2]
x.std <- (x - mean(x)) / sqrt(var(x))
y.nn <- y
for(i in 1:N){
  for(j in 1:N){
    y.nn[i] <- sum(coef.y * c(1, 1 / (1 + exp(-coef.x %*% c(1, x.std[i])))))
  }
}y.nn <- y.nn * sqrt(var(y)) + mean(y)
```

16. A $p = 2$ data set consists of $N = 441$ training vectors $(x_{1i}, x_{2i}, y_i)$ for the distinct pairs $(x_{1i}, x_{2i})$ in the set $\{-1.0, -.9, \ldots, 9, 1.0\}^2$ where the $y_i$ were generated as

$$y_i = \frac{\sin(10(x_{1i} + x_{2i}))}{10(x_{1i} + x_{2i})} + \epsilon_i$$

for iid $N(0, (.02)^2)$ variables $\epsilon_i$. Vardeman will send you such a data file. Use it in the following.
a) Why should you expect MARS to be ineffective in producing a predictor in this context? (You may want to experiment with the `earth` package in R trying out MARS.)

Data are generated by the following code. The function shown below wiggle along two directions.

```r
### Generate data
set.seed(1234)
N <- 441
p <- 2
x.s <- seq(-1, 1, by = 0.1)
x.1 <- rep(x.s, rep(21, 21))
x.2 <- rep(x.s, 21)
y.true <- sin(10 * (x.1 + x.2)) / (10 * (x.1 + x.2))
y <- y.true + rnorm(N, sd = 0.02)
da.org <- cbind(x.1, x.2, y, y.true)
da <- as.data.frame(da.org[!is.nan(da.org[, 3]), ])
library(lattice)
contour(x.s, x.s, matrix(y.true, nrow = 21, byrow = TRUE),
       xlab = "x.1", ylab = "x.2", main = "true")
contour(x.s, x.s, matrix(y, nrow = 21, byrow = TRUE),
       xlab = "x.1", ylab = "x.2", main = "data")
wireframe(y.true ~ x.1 * x.2, data = da, shade = TRUE, scales = list(arrows = FALSE),
          aspect = c(61/87, 0.4), light.source = c(0, 0, 1),
          screen = list(z = -40, x = -60), main = "true")
wireframe(y ~ x.1 * x.2, data = da, shade = TRUE, scales = list(arrows = FALSE),
          aspect = c(61/87, 0.4), light.source = c(0, 0, 1),
          screen = list(z = -40, x = -60), main = "data")
```

For this function, MARS may not be effective at the region near the origin, and not select appropriate "hockey-stick functions" since the function self has a similar pattern.

```r
library(earth)
fit.earth <- earth(y ~ x.1 + x.2, data = da)
summary(fit.earth)
Call: earth(formula = y ~ x.1 + x.2, data = da)

coefficients
(Intercept) 0.1313312

Selected 1 of 7 terms, and 0 of 2 predictors
Importance: x.1-unused, x.2-unused
Number of terms at each degree of interaction: 1 (intercept only model)
GCV 0.1076228 RSS 46.49332 GRSq 0 RSq 0
```
b) Try 2-D locally weighted regression smoothing on this data set using the `loess` function in R. Contour plot your fits for 2 different choices of smoothing parameters.

```
R Code

y.1 <- loess(y ~ x.1 * x.2, data = da)$fitted
y.2 <- loess(y ~ x.1 * x.2, data = da, span = 0.2)$fitted
y.loess.1 <- y
y.loess.1[!is.nan(y.loess.1)] <- y.1
y.loess.2 <- y
y.loess.2[!is.nan(y.loess.2)] <- y.2
da.loess.1 <- da
da.loess.1$y <- y.1
da.loess.2 <- da
da.loess.2$y <- y.2
```
c) Try fitting a thin plate spline to these data. There is an old tutorial at http://www.stat.wisc.edu/~xie/thin_plate_spline_tutorial.html for using the Tps function in the fields package that might make this simple.
d) Use the neural network routines in JMP to fit these data. How complicated does the network architecture have to be in order to do a good job fitting these data? Contour plot your fit.

```R
library(fields)
y.Tps <- Tps(da[c("x.1", "x.2")], da$y)
out.Tps <- predict.surface(y.Tps, grid = list(x.s, x.s))
y.Tps <- out.Tps$z
da.Tps <- da
tmp.y <- as.vector(t(out.Tps$z))
da.Tps$y <- tmp.y[!is.nan(y)]
contour(x.s, x.s, matrix(y.Tps, nrow = 21, byrow = TRUE),
xlab = "x.1", ylab = "x.2", main = "thin plate spline")
wireframe(y ~ x.1 * x.2, data = da.Tps, shade = TRUE, scales = list(arrows = FALSE),
aspect = c(61/87, 0.4), light.source = c(0, 0, 1),
screen = list(z = -40, x = -60), main = "thin plate spline")
```

R Output

```R
R> sum((y.loess.1 - y)^2, na.rm = TRUE)
[1] 34.31899
R> sum((y.loess.2 - y)^2, na.rm = TRUE)
[1] 15.14250
R> sum((out.Tps$z - y)^2, na.rm = TRUE)
[1] 0.3749738
```

Base on the above calculation, I pick 0.37 as the SSE and force JMP to fit a model with similar level of SSE, and I get about 10 hidden nodes to reach the level. The result is shown in the following. I use JMP’s default option to save predicted values in tables and load into R to draw a contour plot.
e) If you were going to use a structured kernel and 1-D smoothing to produce a predictor here, what form for the matrix $A$ would work best? What would be a completely ineffective choice of a matrix $A$? Use the good choice of $A$ and produce a corresponding predictor.

Let

$$A = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}.$$
Based on the Section 5.2 of the lecture outline, for $A$, I will consider a matrix has a positive correlation of $x_1$ and $x_2$ as a better choice, and a negative correlation as an ineffective choice. Given the same SSE, 0.37, then I apply similar code for the question 14 and have the following table. The larger the positive $\rho$, the larger the $\lambda$, the smaller the degrees of freedom $df(\lambda)$, and the less coefficients involved to obtain the same level of SSE. So, the positive correlation gives a better performance. (i.e. $\rho \to 1$ gives the most efficient estimations, and $\rho \to -1$ gives the worst.) However, as $\rho \to \pm 1$, my code yield errors by inverting a singular matrix or an ill-condition.

$$\begin{array}{|c|c|}
\hline
\rho & \lambda \\
0.9 & 0.05911 \\
0.0 & 0.05193 \\
-0.9 & 0.01972 \\
\hline
\end{array}$$

From Dr. Vardeman’s suggestion,

$\rho = 1$ corresponds to using $|(x_1 + x_2) - (x_{01} + x_{02})|$ where $x = (x_1, x_2)$ and $x_0 = (x_{01}, x_{02})$ in the distance function. That is, essentially reducing $(x_1, x_2)$ to $x_1+x_2$ and doing 1-D smoothing on $(y, x_1+x_2)$ that should work perfectly because of the original form of the function generating the data. On the other hand, $\rho = -1$ corresponds to using $|(x_1 - x_2) - (x_{01} - x_{02})|$ in the distance function. That is reducing $(x_1, x_2)$ to $x_1-x_2$ and doing 1-D smoothing on $(y, x_1-x_2)$ that should be essentially ineffective.

After I change my code to 1-D problem that replacing $B = (1, x_a)$ where $x_a = x_1 \pm x_2$ depending on $\rho = \pm 1$. The inverse problem of $(B'W(x_0)B)^{-1}$ is still unsolved.
y.nonan <- da$y
x <- as.matrix(da[, c("x.1", "x.2")])
N <- length(y.nonan)
B <- cbind(1, x)
t.B <- t(B)
W.x.0 <- matrix(0, nrow = N, ncol = N)
get.K.lambda <- function(x.new, x.0, lambda, D, A){
  D(sqrt(t(x.new - x.0) %*% A %*% (x.new - x.0)) / lambda)
}
g.L.lambda <- function(lambda, D, A){
  for(i in 1:N){
    for(j in 1:N){
      W.x.0[j, j] <<- get.K.lambda(x[j,], x[i,], lambda, D, A)
    }
  }
  L.lambda[i,] <- t(B[i,]) %*% solve(t(B) %*% W.x.0 %*% B) %*% t.B %*% W.x.0
}
find.lambda <- function(SSE, D, A, range.df){
  f.df <- function(lambda) sum((g.L.lambda(lambda, D, A) %*% y.nonan - y.nonan)^2) - SSE
  uniroot(f.df, range.df)$root
}
D <- function(t) dnorm(t)

### Start here. Note that partial global variables are initialized above.
### These take a while.

R Code

SSE <- 0.37
rho <- 0.9
A <- matrix(c(1, rho, rho, 1), nrow = 2)
lambda.1 <- find.lambda(SSE, D, A, c(0.05, 0.08))
rho <- 0
A <- matrix(c(1, rho, rho, 1), nrow = 2)
lambda.2 <- find.lambda(SSE, D, A, c(0.015, 0.08))
rho <- -0.9
A <- matrix(c(1, rho, rho, 1), nrow = 2)
lambda.3 <- find.lambda(SSE, D, A, c(0.015, 0.03))
17. Center the outputs for the data set of problem 14. Then derive sets of predictions \( \hat{y}_i \) based on \( \mu \equiv 0 \) Gaussian process priors for \( f(x) \). Plot several of those as functions on the same set of axes (along with centered original data pairs) as follows:

a) Make one plot for cases with \( \sigma^2 = 1, \rho(\delta) = \exp(-c\delta^2), \tau^2 = 1, 4 \) and \( c = 1, 4 \).

Further readings for these questions, CFZ p338-344, and an on-line document http://www.cs.berkeley.edu/~jordan/courses/281B-spring04/lectures/gaussian3.pdf. The left plot is for given settings, and the right plot increases \( c \) to make a better look. i.e. the performance may be dependent on \( \sigma^2, \tau^2, \) and \( c \).

---

### R Code

```r
### Global variables and functions.
rho.1 <- function(delta, c) exp(-c * delta^2)
rho.2 <- function(delta, c) exp(-c * abs(delta))
get.Rho <- function(x, rho, c){
  Rho <- matrix(0, nrow = N, ncol = N)
  for(i in 1:N){
    for(j in i:N){
      Rho[i, j] <- rho(x[i] - x[j], c)
    }
  }
  Rho[lower.tri(Rho)] <- t(Rho)[lower.tri(Rho)]
  Rho
}
get.Rho.new <- function(x.new, rho, c){
  Rho.new <- matrix(0, nrow = length(x.new), ncol = N)
  for(i in 1:length(x.new)){
    for(j in 1:N){
      Rho.new[i, j] <- rho(x.new[i] - x[j], c)
    }
  }
  Rho.new
}
get.y.new <- function(sigma.2, rho, tau.2, c){
  get.Rho.new(x, rho, c) %*% solve(tau.2 * get.Rho(x, rho, c) + diag(sigma.2, N)) %*% y
}
```
### Start here.

```r
y.hat.1 <- get.y.new(1, rho.1, 1, 1)
y.hat.2 <- get.y.new(1, rho.1, 1, 4)
y.hat.3 <- get.y.new(1, rho.1, 4, 1)
y.hat.4 <- get.y.new(1, rho.1, 4, 4)
plot(x, y, xlim = range(x), cex = 0.8,
     ylim = range(c(y, y.hat.1, y.hat.2, y.hat.3, y.hat.4)),
     main = expression(list(rho(Delta) == exp(-c * Delta^2), sigma^2 == 1)),
     xlab = "x", ylab = "y")
lines(list(x = x, y = y.hat.1), lty = 2, col = 2, lwd = 2)
lines(list(x = x, y = y.hat.2), lty = 3, col = 3, lwd = 2)
lines(list(x = x, y = y.hat.3), lty = 4, col = 4, lwd = 2)
lines(list(x = x, y = y.hat.4), lty = 5, col = 5, lwd = 2)
legend(0.35, -1.7, c(
    expression(list(tau^2 == 1, c == 1)),
    expression(list(tau^2 == 1, c == 4)),
    expression(list(tau^2 == 4, c == 1)),
    expression(list(tau^2 == 4, c == 4))),
    lty = 2:5, col = 2:5, lwd = 2, cex = 0.8)
```

b) Make one plot for cases with $\sigma^2 = 1$, $\rho(\delta) = \exp(-c|\delta|)$, $\tau^2 = 1, 4$ and $c = 1, 4$.

The left plot is for given settings, and the right plot increases $c$ to make a better look. i.e. the performance may be dependent on $\sigma^2$, $\tau^2$, and $c$.

![Plot 1](image1.png) ![Plot 2](image2.png)

c) Make one plot for cases with $\sigma^2 = .25$, but otherwise the parameters of a) are used.

The left plot is for given settings, and the right plot increases $c$ to make a better look. i.e. the performance may be dependent on $\sigma^2$, $\tau^2$, and $c$. 
18. Center the outputs for the data set of problem 16. Then derive a set of predictions $\hat{y}_i$ based on a $\mu(x) \equiv 0$ prior for $f(x)$. Use $\sigma^2 = (.02)^2$, $\rho(x - z) = \exp(-2\|x - z\|^2)$, and $\tau^2 = .25$. How do these compare to the ones you made in problem 16?
19. Consider radial basis functions built from kernels. In particular, consider the choice \( D(t) = \phi(t) \), the standard normal pdf.

a) For \( p = 1 \), plot on the same set of axes the 11 functions

\[
K_\lambda(x, \xi_j) = D \left( \frac{|x - \xi_j|}{\lambda} \right) \quad \text{for} \quad \xi_j = \frac{j - 1}{10} \quad j = 1, 2, \ldots, 11
\]

first for \( \lambda = .1 \) and then (in a separate plot) for \( \lambda = .01 \). Then make plots on the a single set of axes the 11 normalized functions

\[
N_{\lambda_j}(x) = \frac{K_\lambda(x, \xi_j)}{\sum_{i=1}^{11} K_\lambda(x, \xi_i)}
\]

first for \( \lambda = .1 \), then in a separate plot for \( \lambda = .01 \).
$K_{\lambda}(x, \xi_j)$

$x <- \text{seq}(-0.3, 1.3, \text{by} = 0.01)$
$xi <- ((1:11) - 1) / 10$
$\lambda <- 0.1$
$\text{Replace by } \lambda \text{ by } 0.01$
$K <- \text{matrix}(0, \text{nrow} = \text{length}(x), \text{ncol} = \text{length}(xi))$
$\text{for}(j \text{ in } 1:\text{length}(xi))$
$\text{K[, j]} <- \text{dnorm}((x - xi[j]) / \lambda)$
$N <- K$
$\text{for}(i \text{ in } 1:\text{length}(x))$
$\text{N[i,]} <- K[i,] / \text{sum(K[i,])}$
$\text{colors} <- \text{c("orange", "green3", "blue")}$
$\text{plot(\text{NULL, NULL, xlim} = \text{range}(x), ylim = \text{range}(K),}$
$\text{main} = \text{substitute(K[lambda == a](x, xi[j]), list(a = lambda))},$
$\text{xlab} = "x", \text{ylab} = "")$
$\text{for(j in 1:\text{length}(xi))}$
$\text{lines(x, K[, j], lwd = 2, col = colors[j %% 3 + 1], lty = j %% 5 + 1)}$
legend(-0.35, 0.4, paste("j = ", 1:11, sep = ""), col = colors[1:11 %% 3 + 1],
cex = 0.7, lty = 1:11 %% 5 + 1, lwd = 2)

plot(NULL, NULL, xlim = range(x), ylim = range(N),
     main = substitute(N[list(lambda == a, j)](x), list(a = lambda)),
     xlab = "x", ylab = "")
ret <- lapply(1:length(xi),
    for(j in 1:length(xi)){
      lines(x, N[, j], lwd = 2, col = colors[j %% 3 + 1], lty = j %% 5 + 1)
    }
legend(-0.35, 0.8, paste("j = ", 1:11, sep = ""), col = colors[1:11 %% 3 + 1],
cex = 0.7, lty = 1:11 %% 5 + 1, lwd = 2)

b) For p = 2, consider the 121 basis functions

\[
K_\lambda(x, \xi_{ij}) = D \left( \frac{||x - \xi_{ij}||}{\lambda} \right) \quad \text{for} \quad \xi_{ij} = \left( \frac{i - 1}{10}, \frac{j - 1}{10} \right) \quad i = 1, 2, \ldots, 11 \quad \text{and} \quad j = 1, 2, \ldots, 11
\]

Make contour plots for \( K_{1,1}(x, \xi_{6,6}) \) and \( K_{0,1}(x, \xi_{6,6}) \). Then define

\[
N_{\lambda ij} = \frac{K_\lambda(x, \xi_{ij})}{\sum_{m=1}^{11} \sum_{l=1}^{11} K_\lambda(x, \xi_{lm})}
\]

Make contour plots for \( N_{1,6,6}(x) \) and \( N_{0,1,6,6}(x) \).
R Code

```r
x.1 <- seq(0.4, 0.6, by = 0.005)
x.2 <- seq(0.4, 0.6, by = 0.005)
xi <- ((1:11) - 1) / 10

x <- cbind(rep(x.1, length(x.2)), rep(x.2, rep(length(x.1), length(x.2))))
x.i.6 <- c((6 - 1) / 10, (6 - 1) / 10)
lambda <- 0.1  # Replace by lambda by 0.01

K.6 <- dnorm(sqrt(colSums((t(x) - xi.6)^2)) / lambda)
contour(x.1, x.2, matrix(K.6, nrow = length(x.1)), xlab = "x.1", ylab = "x.2",
        main = substitute(K[lambda == a](x, xi[6,6]), list(a = lambda)))

K.all <- matrix(0, nrow = nrow(x.all), ncol = nrow(xi.all))
for(j in 1:nrow(xi.all)){
  K.all[, j] <- dnorm(sqrt(colSums((t.x.all - xi.all[j,])^2)) / lambda)
}

N.6 <- K.all[, xi.all[1] == 0.5 & xi.all[2] == 0.5] / rowSums(K.all)
contour(x.1, x.2, matrix(N.6, nrow = length(x.1)), xlab = "x.1", ylab = "x.2",
            main = substitute(N[6,6](a, a), list(a = lambda)))
```

---

```