Chapter 4

Randy

7/10/2021

Covariance Functions

- similarity
- covariance function

4.1 Preliminaries

- stationary
- isotropy
- dot product covariance
- kernel
- Gram matrix
- covariance matrix
- positive semi-definite

$$(T_k f)(x) = \int_{\mathcal{X}} k(x,\ x')f(x')d\mu(x') \quad (4.1)$$

$$Q(v) \ = \ v^\top K v \ \geq 0,\ \forall v \in \mathbb{R}^n \int k(x,x')f(x)f(x')d\mathfrak{z}(x)d\mathfrak{z}(x') \geq 0,\ \forall f \in L_2(\mathcal{X},\mu) \quad (4.2)$$

$$\mathbb{E}[N_u] = \frac{1}{2\pi}\sqrt{\frac{-k''(0)}{k(0)}} exp\Big(-\frac{u^2}{2k(0)}\Big) \quad (4.3)$$

4.1.1 Mean Square Continuity and Differentiability

- Let $\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ \ldots$ be a sequence of points and x_* be a fixed point in $\mathbb{R}^D$ such that $|\boldsymbol{x}_k - \boldsymbol{x}_*| \to 0$ as $k \to \infty$. Then a process $f(\boldsymbol{x})$ is continuous in mean square at $\boldsymbol{x}_*$ if $E[|f(\boldsymbol{x}_k) - f(\boldsymbol{x}_*)|^2] \to 0$ as mean square continuity as $k \to \infty$.
- If this holds for all $x_* \in \mathcal{A}$ where $\mathcal{A}$ is a subset of $\mathbb{R}^D$ then $f(\boldsymbol{x})$ is said to be continuous in mean square (MS) over $\mathcal{A}$.
- A random field is continuous in mean square at x^* if and only if its covariance function $k(x,x')$ is continuous at the point $x = x' = x^*$
- For stationary covariance functions this reduces to checking continuity at $k(\mathbf{0})$. Note that MS continuity does not necessarily imply sample function continuity

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} = \lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{e}_i) - f(\boldsymbol{x})}{h} \quad (4.4)$$

Notice that it is the properties of the kernel k around $\mathbf{0}$ that determine the smoothness properties (MS differentiability) of a stationary process.

4.2 Examples of Covariance Functions

4.2.1 Stationary Covariance Functions

Theorem 4.1 (Bochner's theorem) A complex-valued function k on $\mathbb{R}^D$ is the covariance function of a weakly stationary mean square continuous complex valued random process on $\mathbb{R}^D$ if and only if it can be represented as

$$k(\tau) = \int_{\mathbb{R}^D} e^{2\pi i \boldsymbol{s}\cdot\boldsymbol{\tau}} d\mu(s) \quad (4.5)$$

where μ is a positive finite measure

Wiener-Khintchine theorem The covariance function and the spectral density are Fourier duals

$$k(\tau) = \int S(s) e^{2\pi i s\cdot\tau} ds \quad (4.6a) S(s) = \int k(\tau) e^{-2\pi i s\cdot\tau} d\tau \quad (4.6b) k(\mathbf{0}) = \int S(s) ds$$

use spherical polar coordinates and integrating out the angular variables

$$S(\boldsymbol{s}): \ s \triangleq |\boldsymbol{s}| k(r) = \frac{2\pi}{r^{D/2-1}} \int_0^\infty S(s) J_{D/2-1}(2\pi r s) s^{D/2} ds \quad (4.7) S(s) = \frac{2\pi}{s^{D/2-1}} \int_0^\infty k(r) J_{D/2-1}(2\pi r s) r^{D/2} dr \quad (4.8)$$

Squared Exponential Covariance Function

$$k_{SE}(r) = exp\Big(-\frac{r^2}{2l^2}\Big) \quad (4.9a) S(s) = (2\pi l^2)^{D/2} exp(-2\pi^2 l^2 s^2) \quad (4.9b)$$

$$\phi_c(x) = exp\Big(-\frac{(x-c)^2}{2l^2}\Big) \quad (4.10)$$

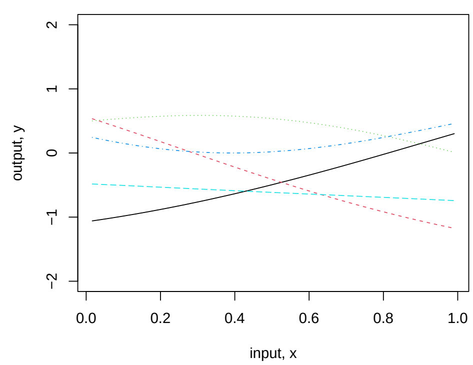
$$k(x_p, x_q) = \sigma_p^2 \sum_{c=1} N \phi_c(x_p)\phi_c(x_q) \quad (4.11)$$

$$\lim_{N\to\infty} \frac{\sigma_p^2}{N} \sum_{c=1}^{N} \phi_c(x_p)\phi_c(x_q) = \sigma_p^2 \int_{c_{min}}^{c_{max}} \phi_c(x_p)\phi_c(x_q) dc \quad (4.12)$$

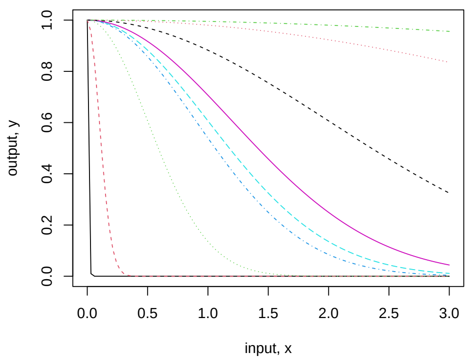
$$k(x_p, x_q) = \sigma_p^2 \int_{-\infty}^{\infty} exp\Big(-\frac{(x_p-c)^2}{2l^2}\Big) exp\Big(-\frac{(x_q-c)^2}{2l^2}\Big) dc = \sqrt{\pi} l \sigma_p^2 exp\Bigg(-\frac{(x_p-x_q)^2}{2(\sqrt{2}l)^2}\Bigg) \quad (4.13)$$

```
get_symm <- function(M) {
  M[upper.tri(M)] <- t(M)[upper.tri(M)]
  return(M)
}
```

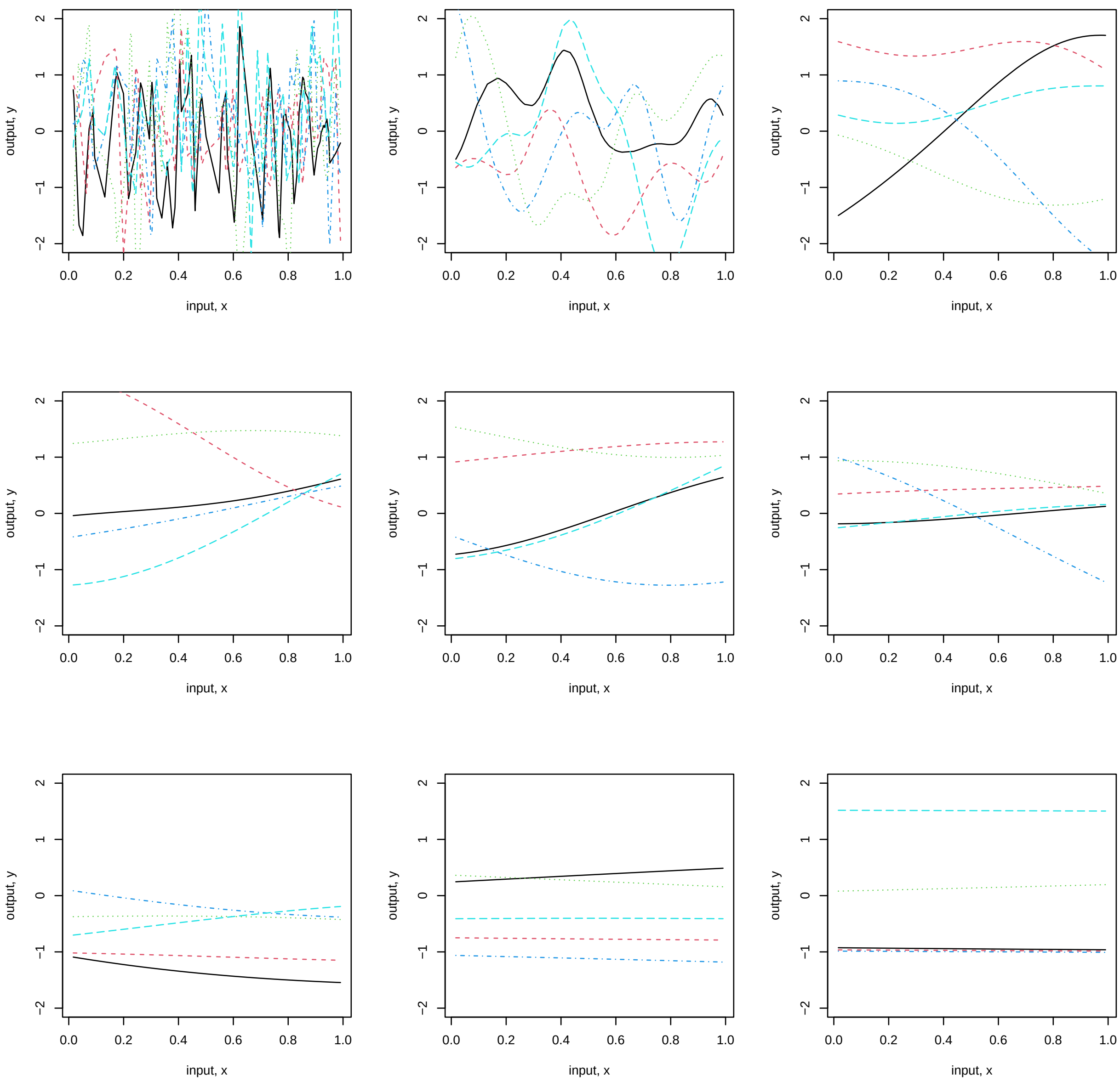
```r
set.seed(55)
## for one squared exponential kernel ----------------------------------------------
## simulate 100 points
x <- runif(100)
## compute distance matrix
d <- abs(outer(x, x, FUN = "-"))
## the characteristic length scale
l0 <- 1
# squared exponential kernel
kernel_se <- exp(-d^2 / (2 * l0^2))
sim <- mvtnorm::rmvnorm(5, sigma = kernel_se)
data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
matplot(data[, 1], data[, -1], "l",
        xlab = "input, x", ylab = "output, y",
        ylim = c(-2, 2))
```




```r
r = seq(0, 3, len = 100)
l1 <- c(0.01, 0.1,  0.5,
         0.9,   1,  1.2,
           2,   5,  10)

kernel_se <- matrix(data = NA, nrow = length(l1), ncol = length(r))
for (i in 1:9) {
  for (j in 1:length(r)) {
    kernel_se[i, j] <- exp(-r[j]^2 / (2 * l1[i]^2))
  }
}
matplot(r, t(kernel_se),
        xlab = "input, x", ylab = "output, y",
        "l")
```




```
## change the characteristic length ---------------------------------------------

l1 <- c(0.01, 0.1,  0.5,
         0.9,   1,  1.2,
           2,   5,  10)
op <- par(mfrow = c(3, 3))
for (i in 1:9) {
  kernel_se <- exp(-d^2 / (2 * l1[i]^2))
  sim <- mvtnorm::rmvnorm(5, sigma = kernel_se)
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1],
          xlab = "input, x", ylab = "output, y",
          "l", ylim = c(-2, 2))
}
```




```
par(op)
```

```
get_se_Sigma <- function(X1, X2, l = 1) {
  Sigma <- matrix(rep(0, length(X1) * length(X2)),
                  nrow = length(X1))

  for (i in 1:nrow(Sigma)) {
    for (j in 1:ncol(Sigma)) {
      Sigma[i, j] <- exp(-0.5 * (abs(X1[i] - X2[j]) / l)^2)
    }
  }
  return(Sigma)
}
```

The Matern Class of Covariance Functions

$$k_{Matern}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\Big(\frac{\sqrt{2\nu}r}{l}\Big)^{\nu}K_{\nu}\Big(\frac{\sqrt{2\nu}r}{l}\Big) \qquad (4.14)$$

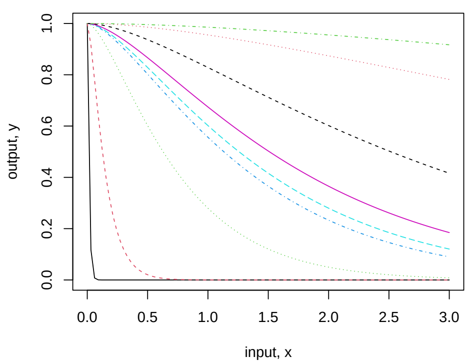
K_{ν} is a modified Bessel function!! Check what is a Bessel function

$$S(s) = \frac{2^{D}\pi^{D/2}\Gamma(\nu + D/2)(2\nu)^{\nu}}{\Gamma(\nu)l^{2\nu}}\Big(\frac{2\nu}{l^2} + 4\pi^2 s^2\Big)^{-(\nu+D/2)} \qquad (4.15)$$

$$k_{\nu=p+1/2}(r) = exp\Big(-\frac{\sqrt{2\nu}r}{l}\Big)\frac{\Gamma(p+1)}{\Gamma(2p+1)}\sum_{i=0}^{p}\frac{(p+i)!}{i!(p-i)!}\Big(\frac{\sqrt{8\nu}r}{l}\Big)^{p-i} \qquad (4.16)$$

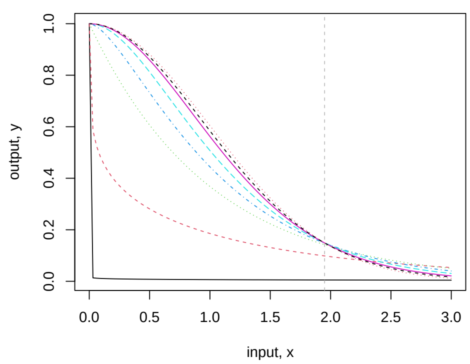
$$k_{\nu=1/2}(r) = exp(-\frac{r}{l}) \quad (4.17a) k_{\nu=3/2}(r) = \Big(1+\frac{\sqrt{3}r}{l}\Big)exp\Big(-\frac{\sqrt{3}r}{l}\Big) \quad (4.17b) k_{\nu=5/2}(r) = \Big(1+\frac{\sqrt{5}r}{l}+\frac{5r^2}{3l^2}\Big)exp\Big(-\frac{\sqrt{5}r}{l}\Big) \quad (4.17c)$$

```r
## change the scale length -------------------------------------------
r = seq(0, 3, len = 100)
l1 <- c(0.01, 0.1,  0.5,
         0.9,   1,  1.2,
           2,   5,  10)
kernel_mat <- matrix(data = NA, nrow = length(l1), ncol = length(r))
for (i in 1:9) {
  for (j in 1:length(r)) {
    kernel_mat[i, j] <- geoR::matern(r[j], phi = l1[i], kappa = 1)
  }
}
matplot(r, t(kernel_mat), "l",
        xlab = "input, x", ylab = "output, y")
```

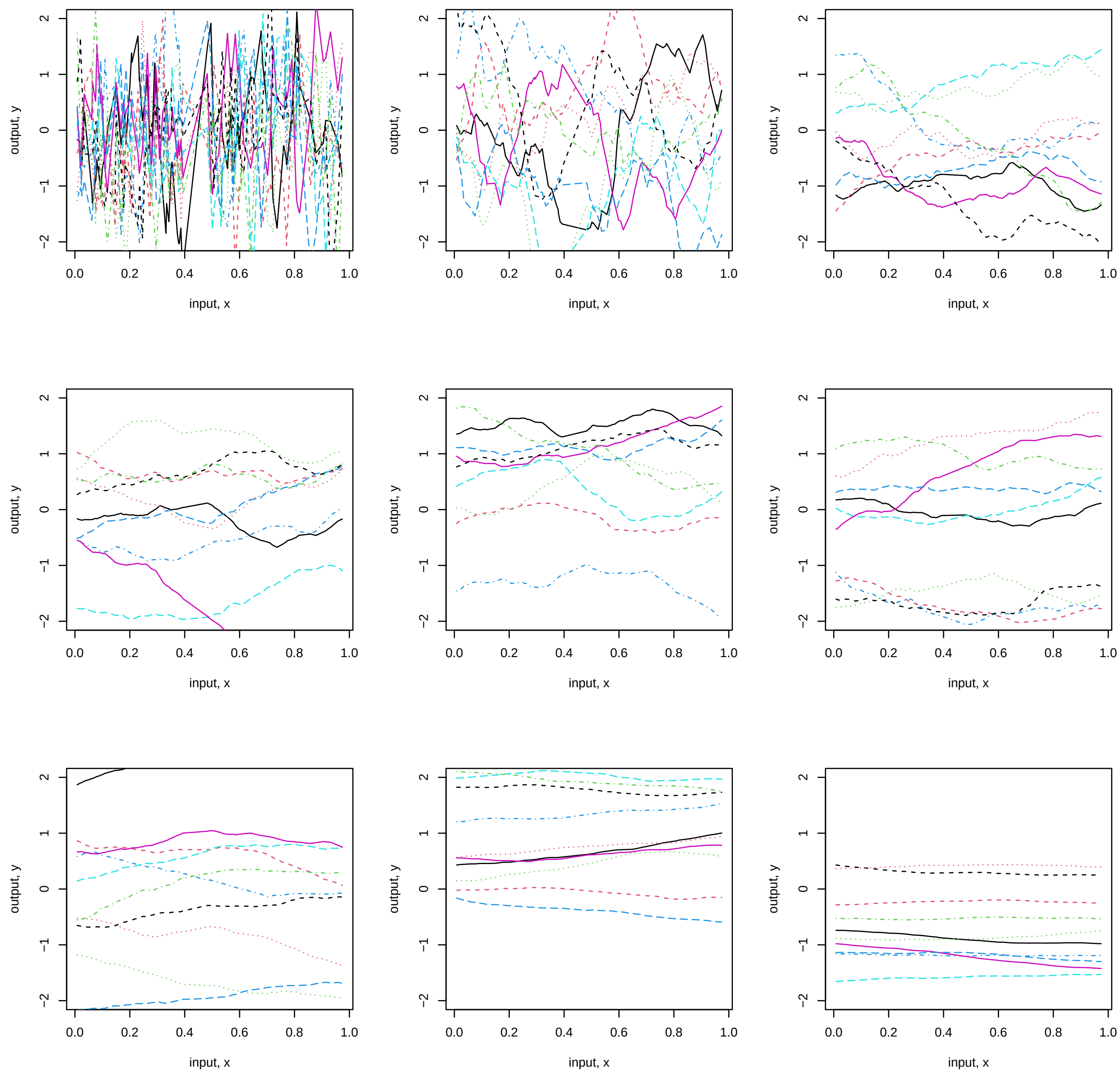



```r
## change the degree of freedom ---------------------------------------------

nu <- c(0.001, 0.1, 0.5,
        1, 2, 5,
        10, 100, 1000)
kernel_mat_df <- matrix(data = NA, nrow = length(l1), ncol = length(r))
for (i in 1:9) {
  for (j in 1:length(r)) {
```

```
    kernel_mat_df[i, j] <- geoR::matern(r[j],
                                        phi = 1/(sqrt(2 * nu[i])),
                                        kappa = nu[i])
  }
}
matplot(r, t(kernel_mat_df),
        xlab = "input, x", ylab = "output, y",
        "l")
abline(v = 1.95, lty = "dashed", col = "grey")
```



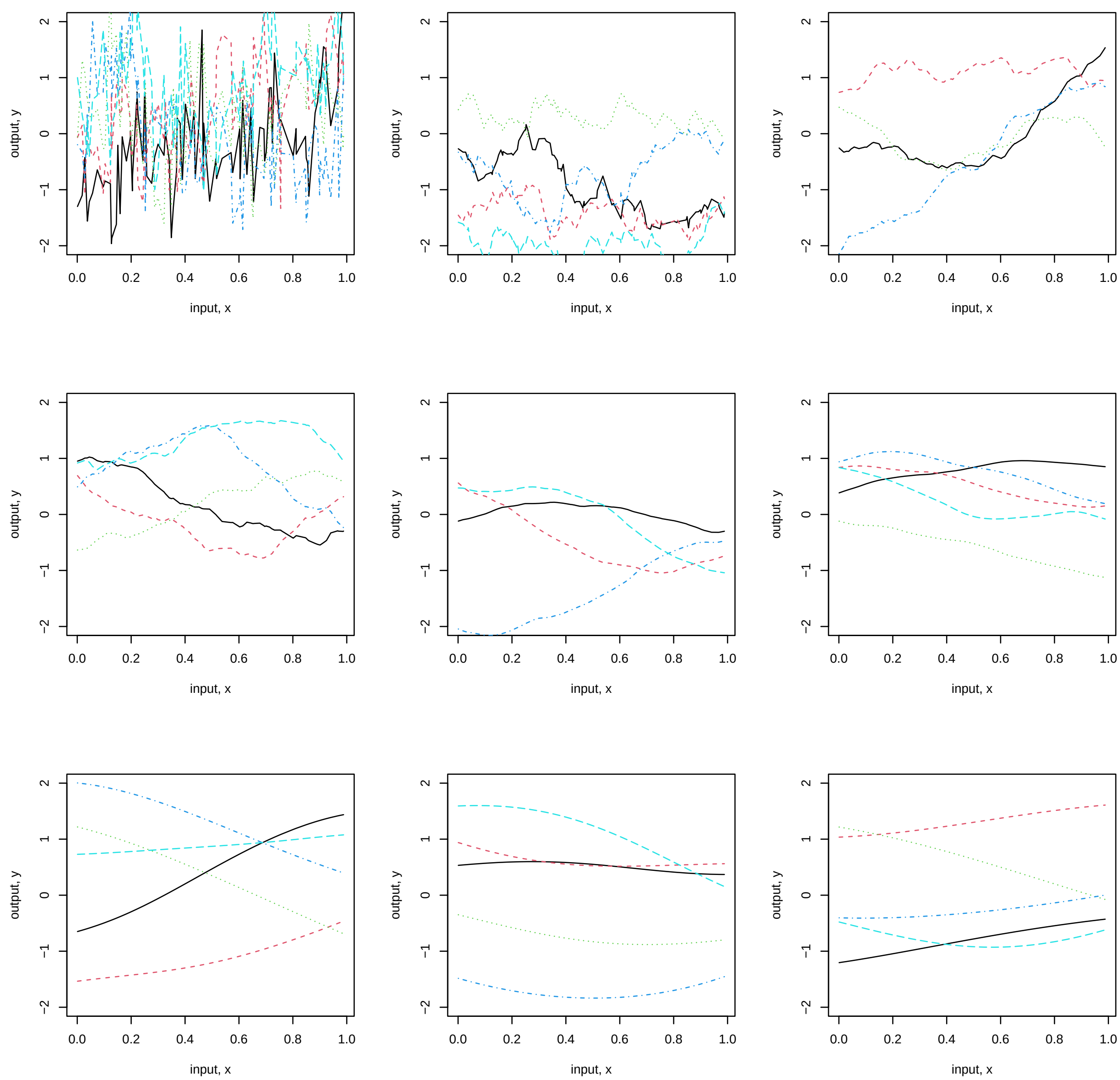

```
l1 <- c(0.01, 0.1,  0.5,
         0.9,   1,  1.2,
           2,   5,  10)
op <- par(mfrow = c(3, 3))
x <- runif(100)
d <- abs(outer(x, x, FUN = "-"))
for (i in 1:9) {
  ## \phi is the l scale in GP book
  ## \kappa is the \mu in GP book
  kernel_mat <- geoR::matern(d, phi = l1[i], kappa = 1)
  sim <- mvtnorm::rmvnorm(10, sigma = kernel_mat)
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1],
          xlab = "input, x", ylab = "output, y",
          "l", ylim = c(-2, 2))
}
```




```
par(op)
```

```
set.seed(555)
nu <- c(0.1, 0.5, 0.9, 1,
        1.5, 2, 5, 10, 20)
op <- par(mfrow = c(3, 3))
x <- runif(100)
d <- abs(outer(x, x, FUN = "-"))
for (i in 1:9) {
  ## \phi is the l / (sqrt(2*\kappa))  scale in GP book
  ## \kappa is the \nu in GP book
  kernel_mat <- geoR::matern(d, phi = 1/(sqrt(2 * nu[i])), kappa = nu[i])
  kernel_mat <- get_symm(t(kernel_mat))
  kernel_mat[is.na(kernel_mat)] <- 0
```

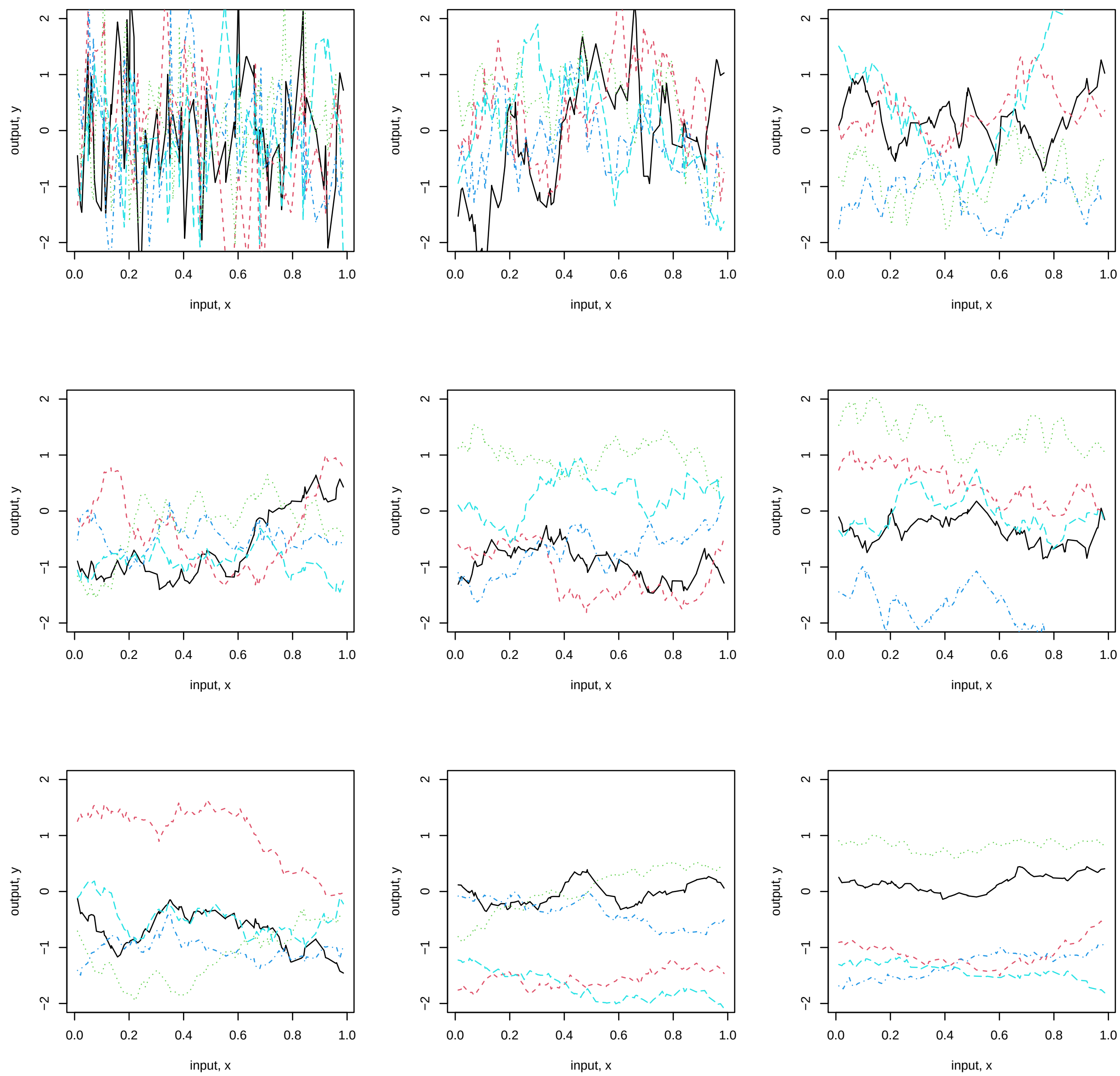
```
  sim <- mvtnorm::rmvnorm(5, sigma = kernel_mat)
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1],
          xlab = "input, x", ylab = "output, y",
          "l", ylim = c(-2, 2))
}
```




```
par(op)

## use besselK() for the Bessel functions
```

Ornstein-Uhlenbeck Process and Exponential Covariance Function

```
l1 <- c(0.01, 0.1,  0.5,
         0.9,   1,  1.2,
           2,   5,  10)

op <- par(mfrow = c(3, 3))
x <- runif(100)
d <- abs(outer(x, x, FUN = "-"))

for (i in 1:9) {
  kernel_ou <- exp(-d / l1[i])
  sim <- mvtnorm::rmvnorm(5, sigma = kernel_ou)
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1],
          xlab = "input, x", ylab = "output, y",
          "l", ylim = c(-2, 2))
}
```



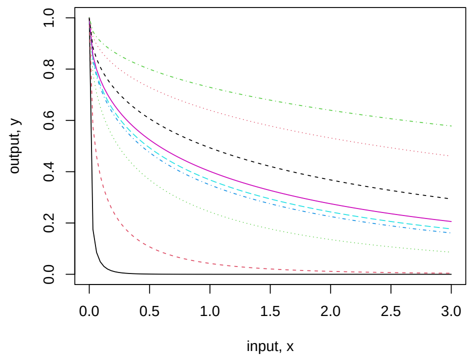

```r
par(op)
```

The γ-exponential Covariance Function

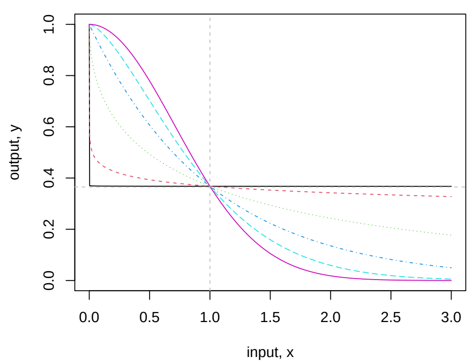
$$k(r) = exp\Big(-\big(\frac{r}{l}\big)\Big)^{\gamma},\ for\ 0 < \gamma \neq 2 \quad (4.18)$$

```r
## change the scale length ---------------------------------------------
l1 <- c(0.01, 0.1,  0.5,
        0.9,   1,  1.2,
          2,   5,  10)
kernel_gamma <- matrix(data = NA, nrow = length(l1), ncol = length(r))
for (i in 1:9) {
  for (j in 1:length(r)) {
    kernel_gamma[i, j] <- exp(- (r[j] / l1[i])^0.5)
```

```
  }
}
matplot(r, t(kernel_gamma),
        xlab = "input, x", ylab = "output, y", "l")
```

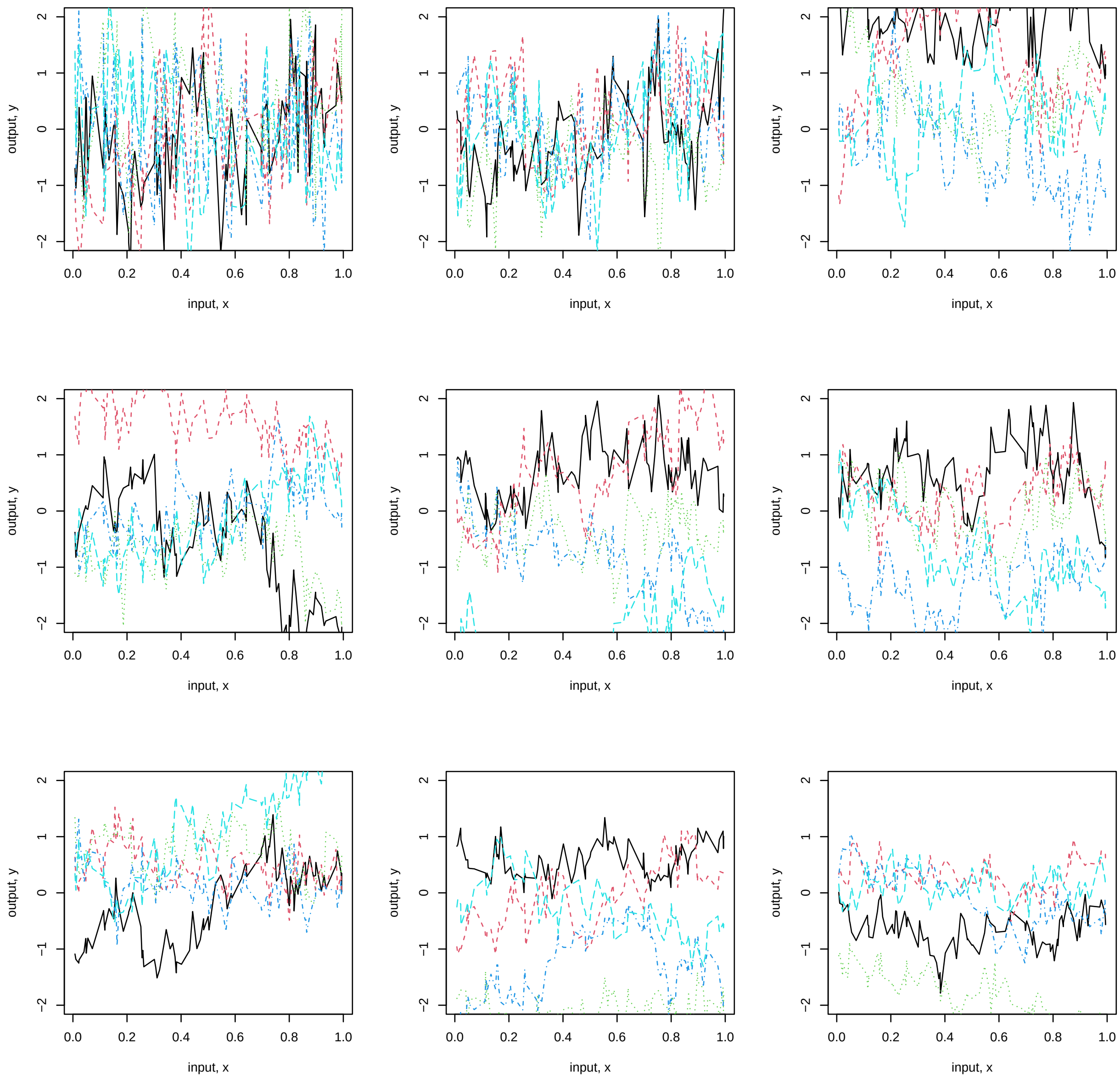



```
## change the degree of freedom --------------------------------------------
r = seq(0, 3, len = 1000)
gamma <- c(0.001, 0.1, 0.5,
          1, 1.5, 2)
kernel_gammaex <- matrix(data = NA, nrow = 6, ncol = length(r))
for (i in 1:6) {
  for (j in 1:length(r)) {
    kernel_gammaex[i, j] <- exp(-(r[j])^gamma[i])
  }
}
matplot(r, t(kernel_gammaex),
        xlab = "input, x", ylab = "output, y",
        "l")
abline(v = 1, h = 0.365, lty = "dashed", col = "grey")
```




```
l1 <- c(0.01, 0.1,  0.5,
         0.9,   1,  1.2,
           2,   5,  10)

op <- par(mfrow = c(3, 3))
```

```r
x <- runif(100)
d <- abs(outer(x, x, FUN = "-"))

for (i in 1:9) {
  kernel_gammaex <- exp(- (d / l1[i])^0.5)
  sim <- mvtnorm::rmvnorm(5, sigma = kernel_gammaex)
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1],
          xlab = "input, x", ylab = "output, y",
          "l", ylim = c(-2, 2))
}
```




```
par(op)
```

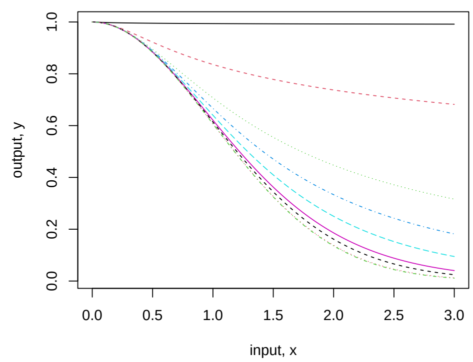
Rational Quadratic Covariance Function

$$k_{RQ}(r) = \Big(1 + \frac{r^2}{2\alpha l^2}\Big)^{-\alpha} \qquad (4.19)$$

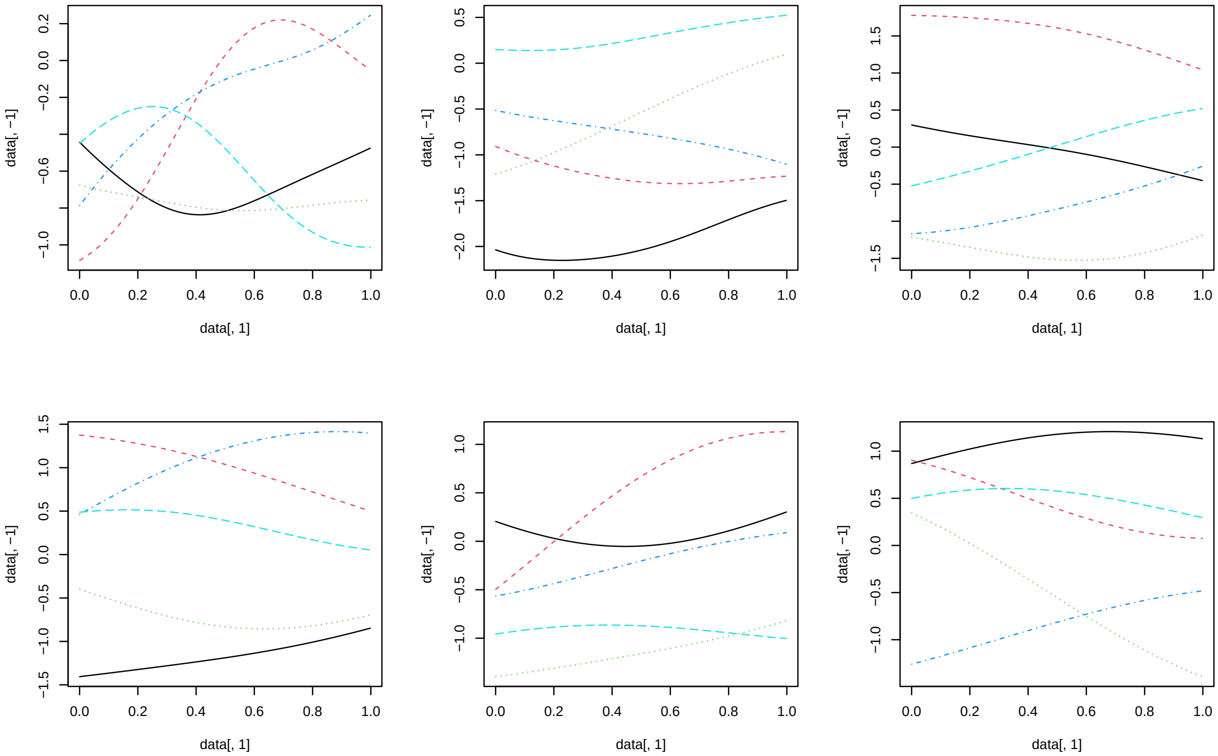
$$k_{RQ}(r) = \int p(\tau|\alpha,\ \beta)k_{SE}(r|\tau)d\tau \propto \int \tau^{\alpha-1}exp\Big(-\frac{\alpha\tau}{\beta}\Big)exp\Big(-\frac{\tau r^2}{2}\Big)d\tau \propto \Big(1 + \frac{r^2}{2\alpha l^2}\Big)^{-\alpha} \qquad (4.20)$$

```
## change the alpha ---------------------------------------------
alpha <- c(0.001, 0.1, 0.5,
           1, 2, 5,
           10, 100, 1000)
```

```r
kernel_rq <- matrix(data = NA, nrow = length(alpha), ncol = length(r))
for (i in 1:9) {
  for (j in 1:length(r)) {
    kernel_rq[i, j] <- (1 + r[j]^2 / (2 * alpha[i]))^(-alpha[i])
  }
}
matplot(r, t(kernel_rq),
        xlab = "input, x", ylab = "output, y",
        "l")
```




```r
alpha <- c(0.5, 1, 2,
           10, 100, 1000)
op <- par(mfrow = c(3, 3))
x <- runif(1000)
d <- abs(outer(x, x, FUN = "-"))

for (i in 1:6) {
  kernel_rq_alpha <- (1 + d^2 / (2 * alpha[i]))^(-alpha[i])
  sim <- mvtnorm::rmvnorm(5, sigma = kernel_rq_alpha)
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1], "l")
}

par(op)
```



Piecewise Polynomial Covariance Functions with Compact Support

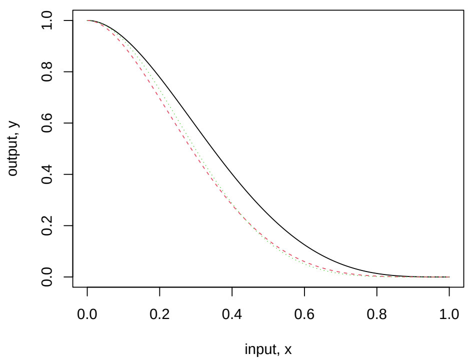
$$k_{ppD,0}(r) = (1-r)_+^j,\ \ where\ j = \lfloor\frac{D}{2}\rfloor+q+1 \quad (4.21a) k_{ppD,1}(r) = (1-r)_+^{j+1}\big((j+1)r+1\big) \quad (4.21b) k_{ppD,2}(r) = (1-r)_+^{j+2}\big((j^2+4j+$$

Further Properties of Stationary Covariance Functions

$$r^2(x,\ x') = (x-x')^\top M(x-x')M = \Lambda\Lambda^\top + \Psi \qquad (4.22)$$

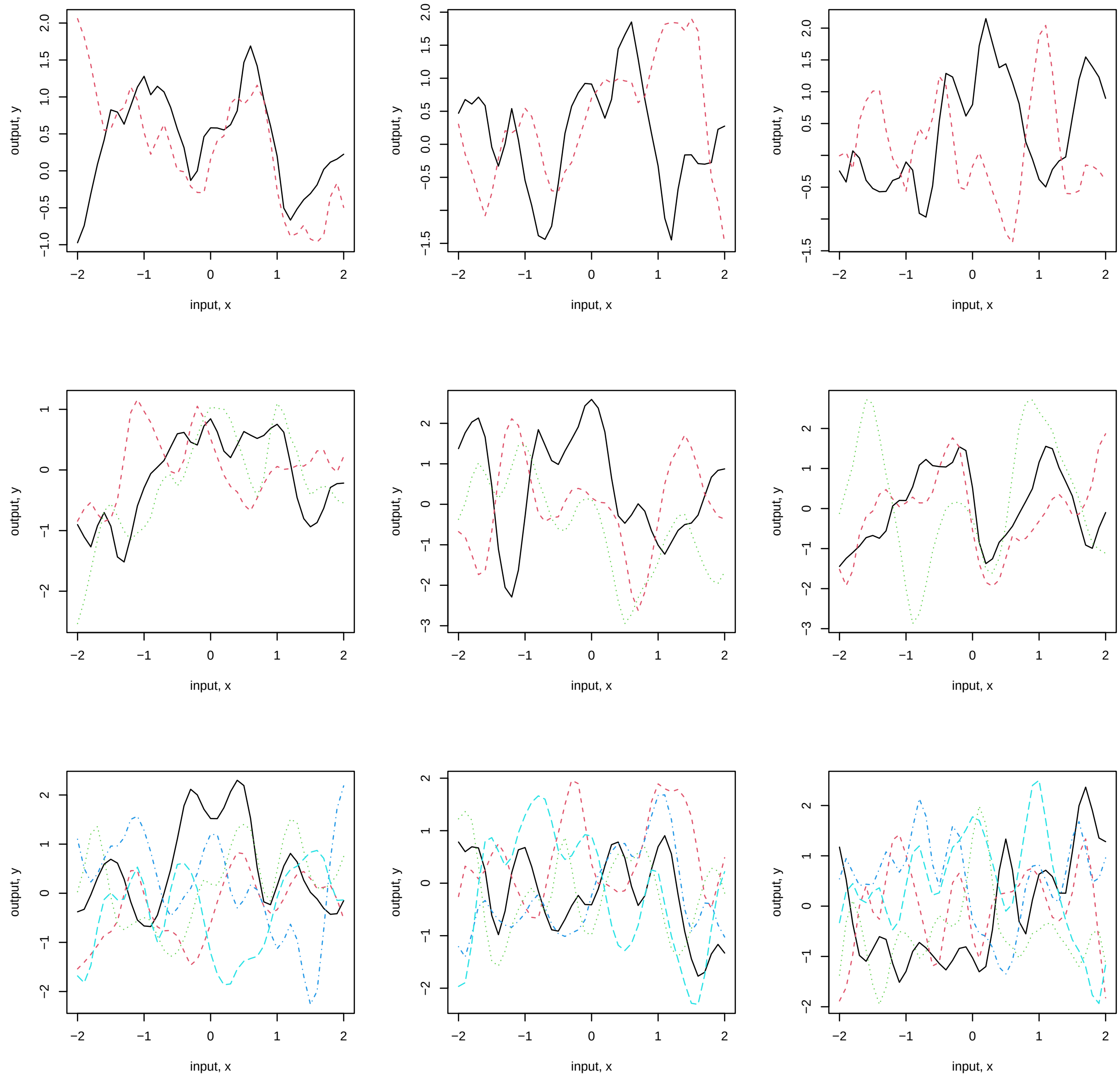
```
r = seq(0, 1, len = 1000)
kernel_pp <- matrix(data = NA, nrow = 3, ncol = length(r))
D = c(1, 3, 1)
q = c(1, 1, 2)
```

```r
j <- D/2 + q + 1

for (n in 1:length(r)) {
  ## D =1 q =1
  kernel_pp[1, n] <- max(c((1 - r[n])^(j[1] + 1), 0)) * ((j[1] + 1) * r[n] + 1)
  kernel_pp[2, n] <- max(c((1 - r[n])^(j[2] + 1), 0)) * ((j[2] + 1) * r[n] + 1)
  kernel_pp[3, n] <- max(c((1 - r[n])^(j[3] + 2), 0)) * ((j[3]^2 + 4 * j[3] + 3) * r[n]^2 +(3 * j[3] + 
}
# View(kernel_pp)
matplot(r, t(kernel_pp),
        xlab = "input, x", ylab = "output, y", "l")
```




```r
x <- seq(-2, 2, by = 0.1)
r <- abs(outer(x, x, FUN = "-"))
r[r > 1] = 1

op <- par(mfrow = c(3, 3))
## q=1 --------------------------------------------
D <- c(1, 2, 3)
q <- 1
j <- D/2 + q + 1
for (i in 1:3) {
  kernel_pp_q1 <- (1 - r)^{j[i] + 1} *
    ((j[i] + 1) * r + 1) %>% get_symm()
  # fields::image.plot(x, x, kernel_pp_q1)
  # chol(kernel_pp_q1)
  # View(kernel_pp_q1)
  # solve(kernel_pp_q1) %>% View()
  sim <- mvtnorm::rmvnorm(2, sigma = kernel_pp_q1)
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1], xlab = "input, x", ylab = "output, y", "l")
}


## q=2 --------------------------------------------
D <- c(1, 2, 3)
q <- 2
```

```r
j <- D/2 + q + 1
for (i in 1:3) {
  kernel_pp_q2 <- (1 - r)^(j[i] + 2) *
    ((j[i]^2 + 4 * j[i] + 3) * r^2 +
       (3 * j[i] + 6) * r + 3) / 3 %>% get_symm()
  sim <- mvtnorm::rmvnorm(3, sigma = kernel_pp_q2)
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1], xlab = "input, x", ylab = "output, y", "l")
}

## q=3 ---------------------------------------------
D <- c(1, 2, 3)
q <- 3
j <- D/2 + q + 1

for (i in 1:3) {
  kernel_pp_q3 <- ((1 - r)^(j[i] + 3) *
    ((j[i]^3 + 9 * j[i]^2 + 23 * j[i] + 15) * r^3 +
       (6 * j[i]^2 + 36 * j[i] + 45) * r^2 +
       (15 * j[i] + 45) * r +
       15) / 15 ) %>% get_symm()
  # View(kernel_pp_q3)
  # fields::image.plot(kernel_pp_q3)
  # isSymmetric.matrix(kernel_pp_q3)
  sim <- mvtnorm::rmvnorm(5, sigma = kernel_pp_q3)
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1], xlab = "input, x", ylab = "output, y", "l")
}
```




```
par(op)
```

Stationary kernels can also be defined on a periodic domain, and can be readily constructed from stationary kernels on $\mathbb{R}$. Given a stationary kernel $k(x)$, the kernel $k_{\mathbb{T}}(x) = \sum_{m \in Z} k(x + ml)$ is periodic with period l.

4.2.2 Dot Product Covariance Functions

```
x <- seq(0, 1, length = 100)
input <- cbind(x, x^2) %>%
  data.frame() %>%
  select(x = 1, xseq = 2)
sigma <- 0.1
kernel_dot <- geometry::dot(input, input, d = T)
kernel_dot
```

```
       x     xseq
33.50168 20.30337
```

$$k(x,\ x') = \sigma_0^2 + x \cdot x' k(x,\ x') = \sigma_0^2 + x^\top \Sigma_p x' k(x,\ x') = (\sigma_0^2 + x^\top \Sigma_p x')^p$$

$$k(x,\ x') = (x \cdot x')^p = \Big(\sum_{d=1}^{D} x_d x'_d\Big)^p = \Big(\sum_{d_1=1}^{D} x_{d1} x'_{d1}\Big) \dots \Big(\sum_{d_p=1}^{D} x_{dp} x'_{dp}\Big) = \sum_{d_1=1}^{D} \dots \sum_{d_p=1}^{D} (x_{d_1} \dots x_{d_p})(x'_{d_1} \dots x'_{d_p}) \triangleq \phi(x) \cdot \phi(x') \quad (4.23)$$

$$\phi_m(x) = \sqrt{\frac{p!}{m_1! \dots ! \ m_D!}} x_1^{m1} \dots x_D^{m_D} \quad (4.24) for\ p = 2\ in\ D = 2,\ \phi(x) = (x_1^2,\ x_2^2,\ \sqrt{2} x_1 x_2)^\top$$

4.2.3 Other Non-stationary Covariance Functions

Neural network kernel by *Neal (1996)*

$$f(x) = b + \sum_{j=1}^{N_H} \nu_j h(x;\ u_j) \quad (4.25)$$

$$\mathbb{E}_w[f(x)] = 0 \quad (4.26) \mathbb{E}_w[f(x) f(x')] = \sigma_b^2 + \sum_j \sigma_\nu^2 \mathbb{E}_u[h(x;\ u_j) h(x';\ u_j)] \quad (4.27) = \sigma_b^2 + N_H \sigma_\nu^2 \mathbb{E}_u[h(x;\ u) h(x';\ u)] \quad (4.28)$$

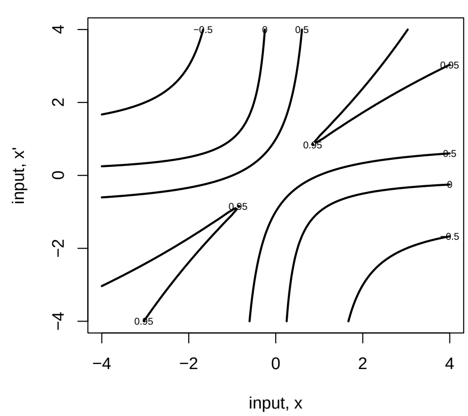
$$h(z) = erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt h(x;\ u) = erf(u_0 + \sum_{j=1}^{D} u_j x_j), \quad u \sim \mathcal{N}(0,\ \Sigma) k_{NN}(x,\ x') = \frac{2}{\pi} \sin^{-1} \Bigg(\frac{2\tilde{x}^\top \Sigma \tilde{x}'}{\sqrt{(1 + 2\tilde{x}^\top \Sigma \tilde{x})(1 + 2\tilde{x}'^\top \Sigma$$

```r
## X as \tilde x two dimensional vector augmented with 1s
x0 <- 1
x1 <- seq(-4, 4, length = 100)

## check the X is a column vector
## each value in X is a two dimension vector
X <- cbind(x0, x1)
# View(X %*% Sigma %*% t(X))

Sigma <- matrix(c(100, 0, 0, 100), nrow = 2)
insin <- matrix(NA, nrow = nrow(X), ncol = nrow(X))
for (i in seq_along(1:nrow(X))) {
  for (j in seq_along(1:nrow(X))) {
    insin[i, j] <- 2 * X[i, ] %*% Sigma %*% X[j, ] /
      sqrt((1 + 2 * X[i, ] %*% Sigma %*% X[i, ]) * (1 + 2 * X[j, ] %*% Sigma %*% X[j, ]))
  }
}

knn <- 2 / pi * asin(insin)
```

```r
contour(x1, x1, knn,
        ylim = c(-4, 4),
        levels = c(-0.5, 0, 0.5, 0.95),
```

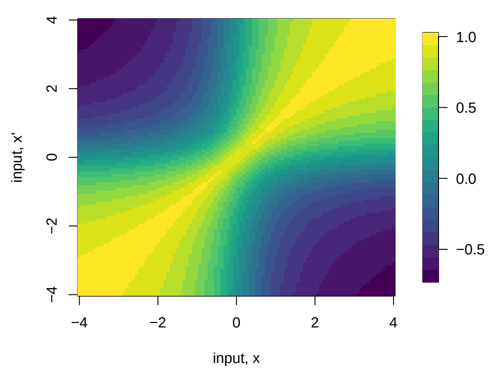
```
        lwd = 2,
        method = "simple",
        xlab = "input, x",
        ylab = "input, x'",
        main = "Figure 4.5(a)")
```

Figure 4.5(a)




```
fields::image.plot(x1, x1, knn,
                   xlab = "input, x",
                   ylab = "input, x'",
                   main = "Figure 4.5(a)",
                   nlevel = 20,
                   col = viridis::viridis(20))
```

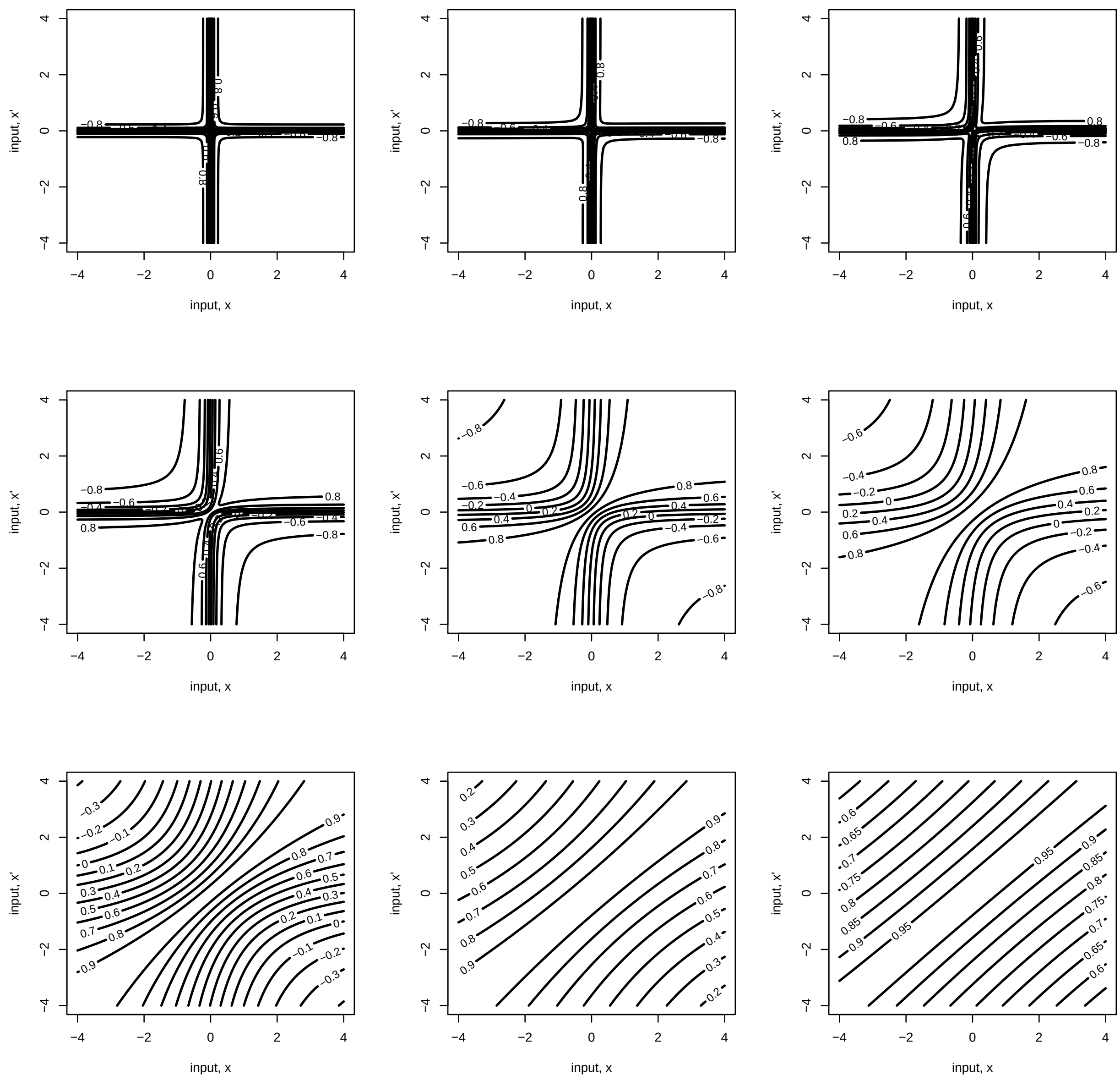
Figure 4.5(a)




```
kernel_nn <- function(sigma_e = 10,
                      sigma_f = 10,
                      xmin = -4,
                      xmax = 4,
                      length = 100) {
  x0 <- 1
  x1 <- seq(xmin, xmax, length = length)
  X <- cbind(x0, x1)
  Sigma <- matrix(c(sigma_e^2, 0, 0, sigma_f^2), nrow = 2)
  insin <- matrix(NA, nrow = nrow(X), ncol = nrow(X))
```

```r
  for (i in seq_along(1:nrow(X))) {
    for (j in seq_along(1:nrow(X))) {
      insin[i, j] <- 2 * X[i, ] %*% Sigma %*% X[j, ] /
        sqrt((1 + 2 * X[i, ] %*% Sigma %*% X[i, ]) * (1 + 2 * X[j, ] %*% Sigma %*% X[j, ]))
    }
  }
  knn <- 2 / pi * asin(insin)
}
```

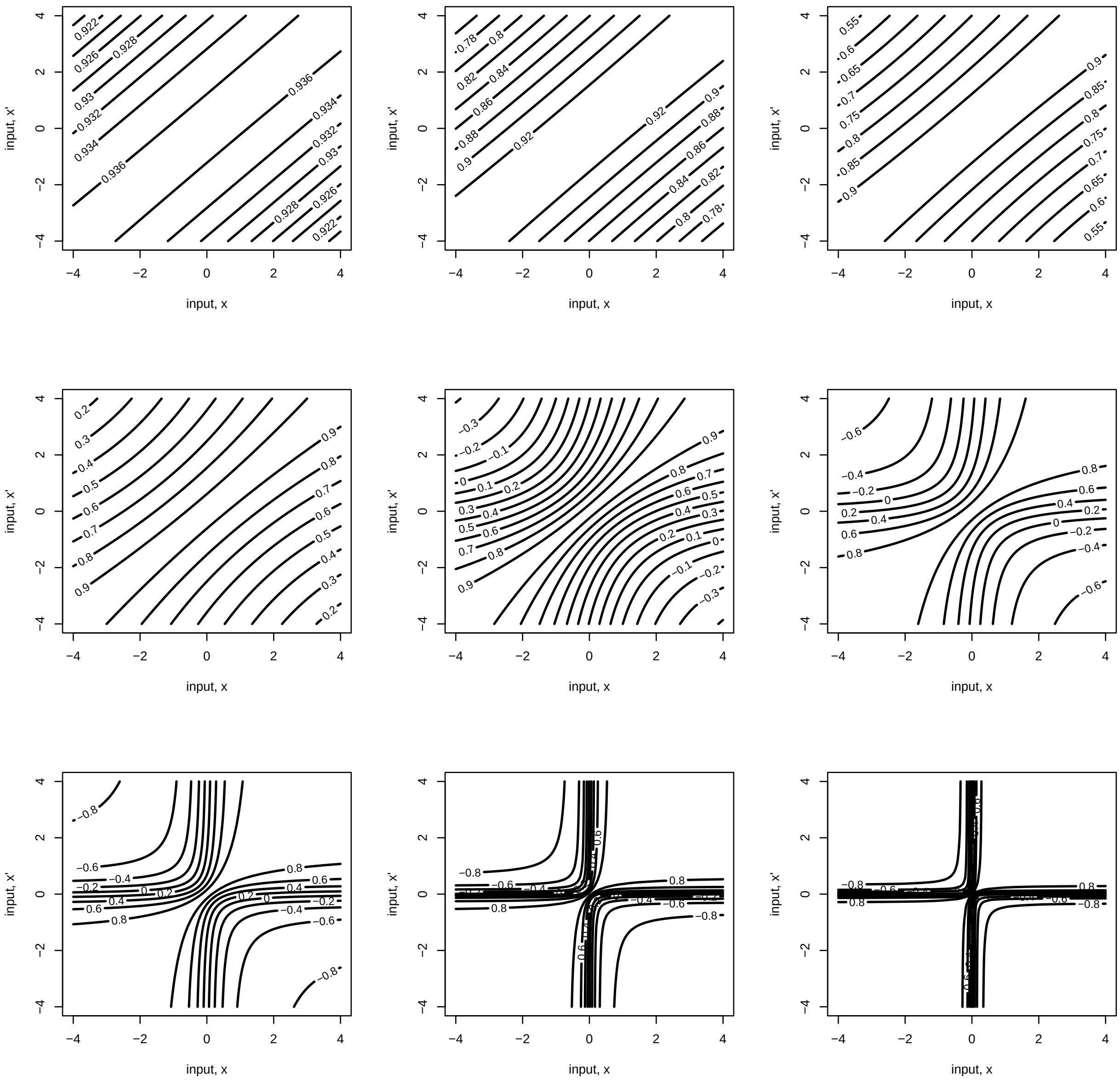
```r
op <- par(mfrow = c(3, 3))
## changing the sigma_e
walk(c(0.1, 0.5, 1,
      2, 5, 10,
      20, 50, 100),
    ~ kernel_nn(sigma_e = .x,
                sigma_f = 10,
                xmin = -4,
                xmax = 4,
                length = 100) %>%
      contour(x1, x1, .,
              lwd = 2,
              xlab = "input, x",
              ylab = "input, x'"))
```



```r
par(op)
```

```r
op <- par(mfrow = c(3, 3))
## changing the sigma_e
walk(c(0.1, 0.5, 1,
       2, 5, 10,
       20, 50, 100),
     ~ kernel_nn(sigma_e = 10,
                 sigma_f = .x,
                 xmin = -4,
                 xmax = 4,
                 length = 100) %>%
       contour(x1, x1, .,
               lwd = 2,
```

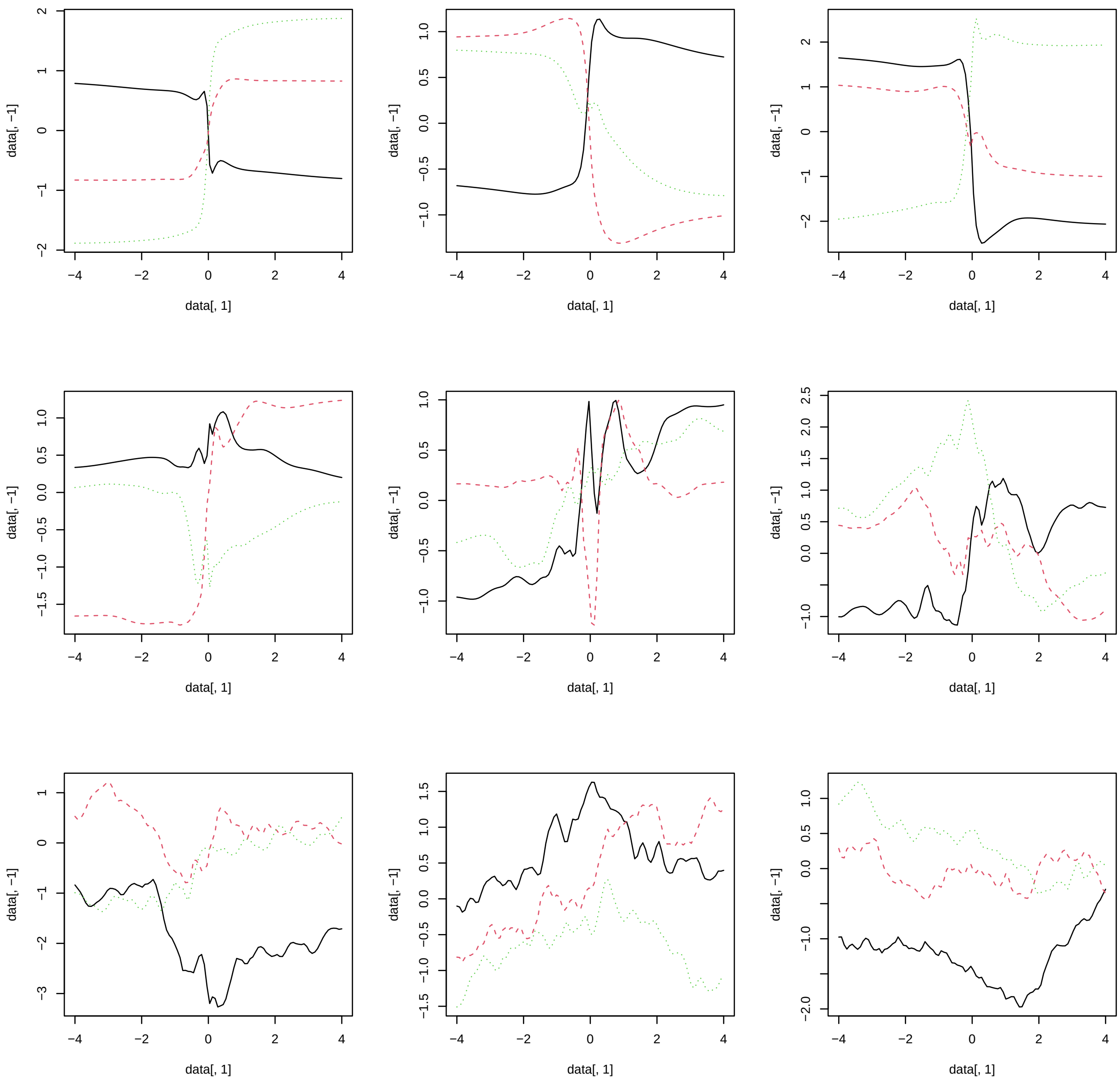
```
             xlab = "input, x",
             ylab = "input, x'"))
```




```
par(op)
```

```
op <- par(mfrow = c(3, 3))
x <- seq(-4, 4, length = 100)
sigma <- c(0.1, 0.5, 1,
           2, 5, 10,
           20, 50, 100)

for (i in 1:9) {
  sim <- mvtnorm::rmvnorm(3, sigma = kernel_nn(sigma_e = sigma[i]))
```

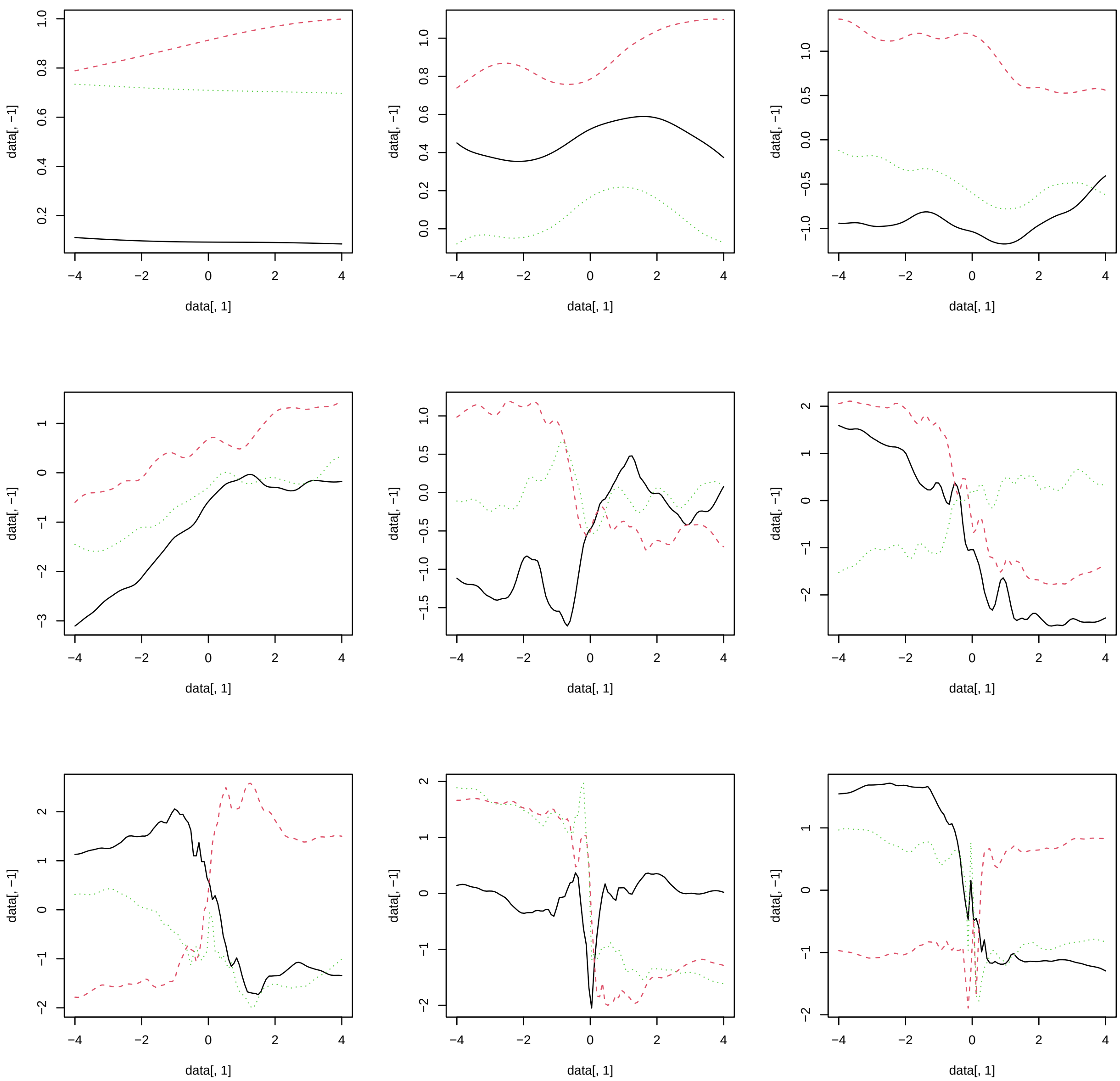
```
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1], "l")
}
```




```
par(op)
```

```
op <- par(mfrow = c(3, 3))
x <- seq(-4, 4, length = 100)
sigma <- c(0.1, 0.5, 1,
           2, 5, 10,
           20, 50, 100)
```

```r
for (i in 1:9) {
  sim <- mvtnorm::rmvnorm(3, sigma = kernel_nn(sigma_f = sigma[i]))
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1], "l")
}
```



```r
par(op)
```

Generalizednon stationary covariance function the squared exponential $k_G(\boldsymbol{x}, \boldsymbol{x}') \propto \exp(-|x - x'|^2/4\sigma_g^2)$.

For a finite value of σ_u^2, $k_G(\boldsymbol{x}, \boldsymbol{x}')$ comprises a squared exponential covariance function modulated by the Gaussian decay envelope function $\exp\left(-\frac{\boldsymbol{x}^\top \boldsymbol{x}}{2\sigma_m^2}\right) \exp\left(-\frac{\boldsymbol{x}'^\top \boldsymbol{x}'}{2\sigma_m^2}\right)$, cf. the vertical rescaling construction in **section 4.2.4**

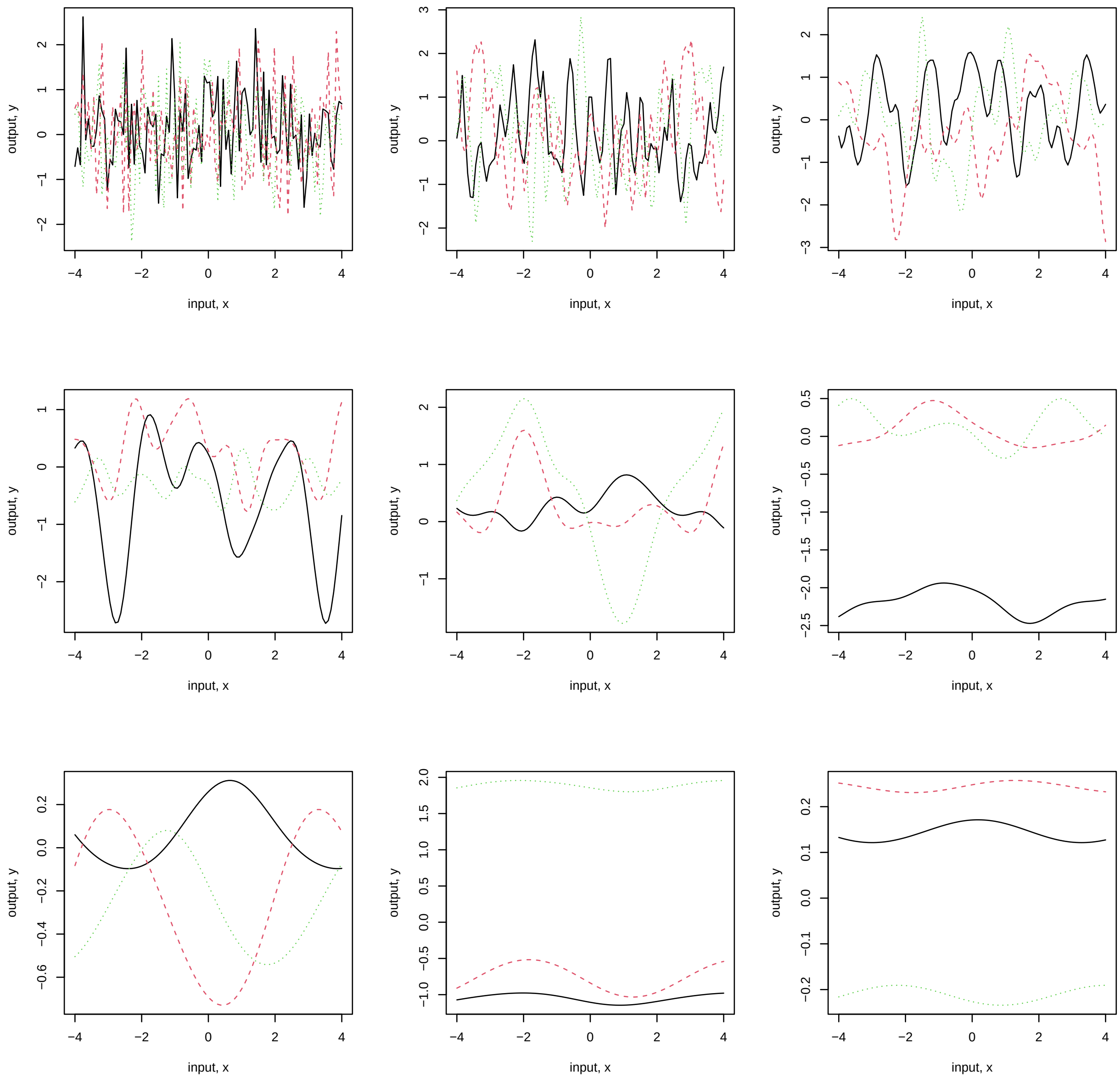
$$h(x;\ u) = exp\Big(-\frac{|x-u|^2}{2\sigma_g^2}\Big),\ \ u \sim \mathcal{N}(\mathbf{0},\ \sigma_u^2 I) k_G(x,\ x') = \frac{1}{(2\pi\sigma_u^2)^{d/2}} \int exp\Big(-\frac{|x-u|^2}{2\sigma_g^2} - \frac{|x'-u|^2}{2\sigma_g^2} - \frac{u^\top u}{2\sigma_u^2}\Big) du = \Big(\frac{\sigma_\epsilon}{\sigma_u}\Big)^d exp\Big(-\frac{x^\top}{2\sigma}$$

MacKay's sin(x) cos(x) kernel

$$(\cos(x) - \cos(x'))^2 + (\sin(x) - \sin(x'))^2 = 4\sin^2(\frac{x-x'}{2}) k(x,\ x') = exp\Big(-\frac{2\sin^2(\frac{x-x'}{2})}{l^2}\Big) \tag{4.31}$$

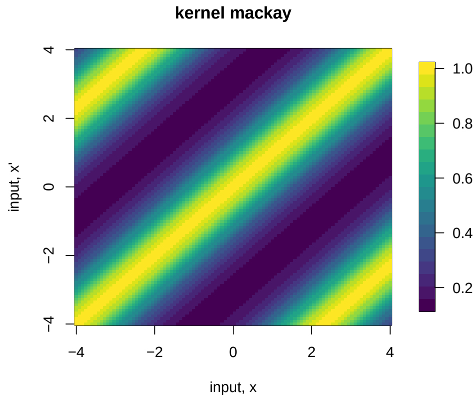
```
kernel_mackay <- function(xmin = -4,
                          xmax = 4,
                          length = 100,
                          scale = 1) {
  x <- seq(xmin, xmax, length = length)
  d <- abs(outer(x, x, FUN = "-"))
  kernel <- exp(- 2 * (sin(d / 2))^2 / scale^2)
  kernel <- get_symm(kernel)

  ## not suppose to do this,
  ## but working for current coding
  ## the first value is always NA?
  kernel[is.na(kernel)] <- 0
  return(kernel)
}
```

```
op <- par(mfrow = c(3, 3))
x <- seq(-4, 4, length = 100)
l <- c(0.01, 0.1, 0.2,
       0.5, 1, 2,
       5, 10, 50)
for (i in 1:9) {
  sim <- mvtnorm::rmvnorm(3, sigma = kernel_mackay(scale = l[i]))
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1],
          xlab = "input, x", ylab = "output, y",
          "l")
}
```




```
par(op)
```

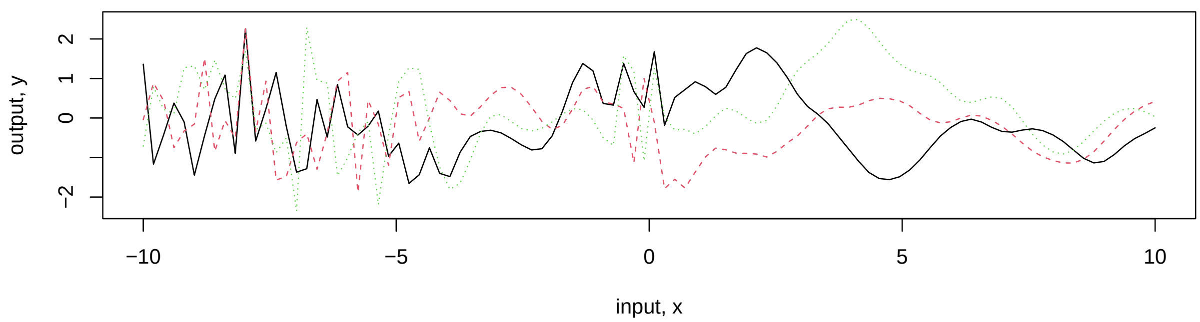
```
x1 <- seq(-4, 4, length = 100)
kmack <- kernel_mackay()
fields::image.plot(x1, x1, kmack,
xlab = "input, x",
ylab = "input, x'",
main = "kernel mackay",
nlevel = 20,
col = viridis::viridis(20))
```



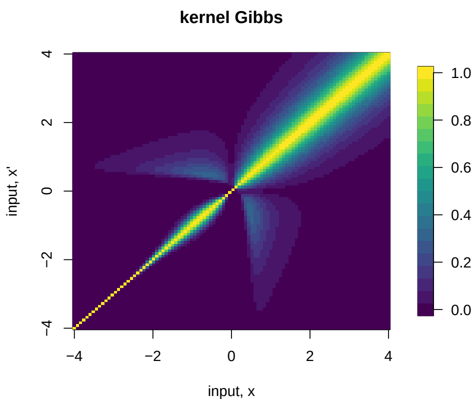
Gibbs Kernel

$$k(x,\ x') = \prod_{d=1}^{D} \Big(\frac{2l_d(x)l_d(x')}{l_d^2(x) + l_d^2(x')} \Big)^{1/2} exp\Big(- \sum_{d=1}^{D} \frac{(x_d - x_d')^2}{l_d^2(x) + l_d^2(x')} \Big) \qquad (4.32)$$

```
## Gibbs kernel for d = 1
x <- seq(-10, 10, length = 100)
# lfunction <- function(x) abs(sin(x))^2
lfunction <- function(x) x^2 * exp(x)
# lfunction <- function(x) x^2
# lfunction <- function(x) exp(x)
# lfunction <- function(x) abs(x)

kernel_gibbs <- matrix(NA, nrow = length(x), ncol = length(x))
for (i in seq_along(1:length(x))){
  for (j in seq_along(1:length(x))){
    kernel_gibbs[i, j] <- sqrt(2 * lfunction(x[i]) * lfunction(x[j]) /
                                 (lfunction(x[i])^2 + lfunction(x[j])^2)) *
      exp(-(x[i] - x[j])^2 / (lfunction(x[i])^2 + lfunction(x[j])^2))
  }
}

kernel_gibbs <- get_symm(kernel_gibbs)
kernel_gibbs[is.na(kernel_gibbs)] <- 0
sim <- mvtnorm::rmvnorm(3, sigma = kernel_gibbs)
data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
matplot(data[, 1], data[, -1],
        xlab = "input, x", ylab = "output, y",
        "l")
```




```
x1 <- seq(-4, 4, length = 100)
fields::image.plot(x1, x1, kernel_gibbs,
xlab = "input, x",
ylab = "input, x'",
main = "kernel Gibbs",
nlevel = 20,
col = viridis::viridis(20))
```



$$Q_{ij} = (x_i - x_j)^\top((\Sigma_i + \Sigma_j)/2)^{-1}(x_i - x_j) \quad (4.33)$$

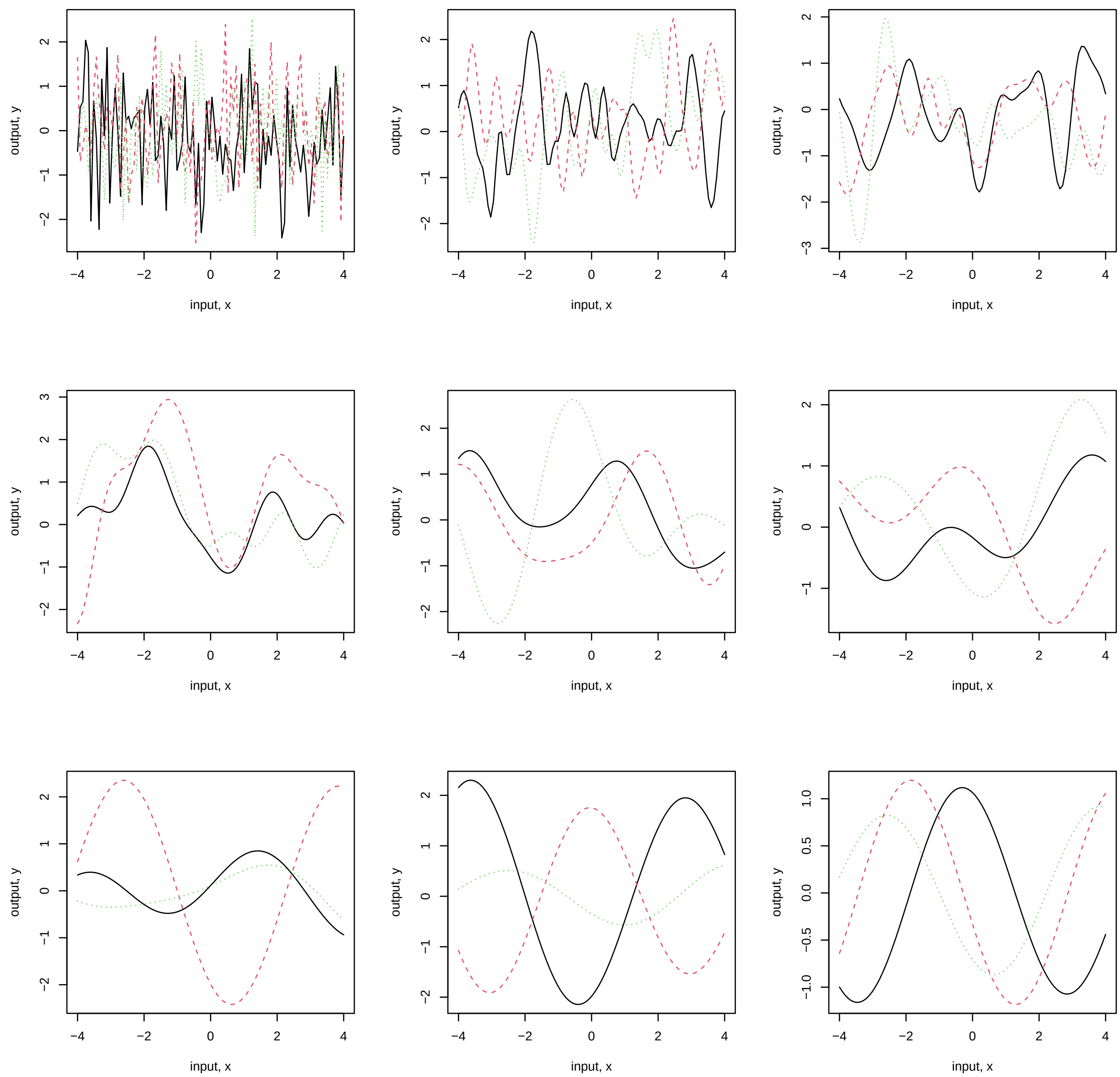
$$k_{NS}(x_i,\ x_j) = 2^{D/2}|\Sigma_i|^{1/4}|\Sigma_j|^{1/4}|\Sigma_i + \Sigma_j|^{-1/2}k_S(Q_{ij}) \quad (4.34)$$

Periodic (not in the book)

$$k_{PD}(x_i,\ x_j) = \sigma^2 \cos\big(\omega(x - x')\big) \exp\left(-\frac{1}{2l^2}(x - x')^2\right)$$

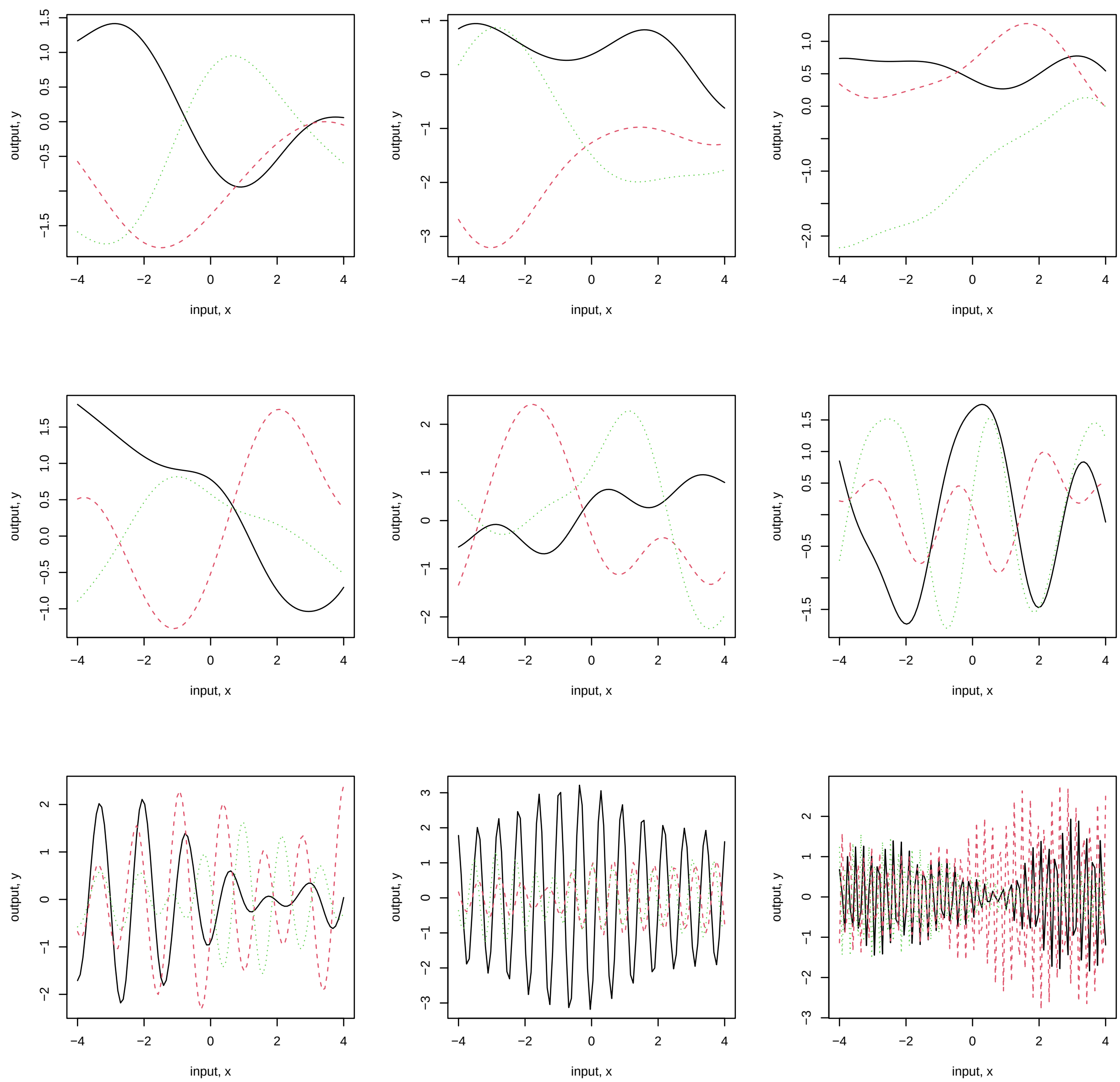
```
kernel_pd <- function(xmin = -4,
                      xmax = 4,
                      length = 100,
                      scale = 1,
                      sigma = 1,
                      omega = 1) {
  x <- seq(xmin, xmax, length = length)
```

```
  d <- abs(outer(x, x, FUN = "-"))
  kernel <- sigma^2 * cos(omega * d) *
    exp(-0.5 * (d / 2)^2 / scale^2)
  kernel <- get_symm(kernel)
  kernel[is.na(kernel)] <- 0
  return(kernel)
}
op <- par(mfrow = c(3, 3))
x <- seq(-4, 4, length = 100)
l <- c(0.01, 0.1, 0.2,
        0.5, 1, 2,
        5, 10, 50)
for (i in 1:9) {
  sim <- mvtnorm::rmvnorm(3, sigma = kernel_pd(scale = l[i]))
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1],
    xlab = "input, x",
    ylab = "output, y",
    "l")
}
```



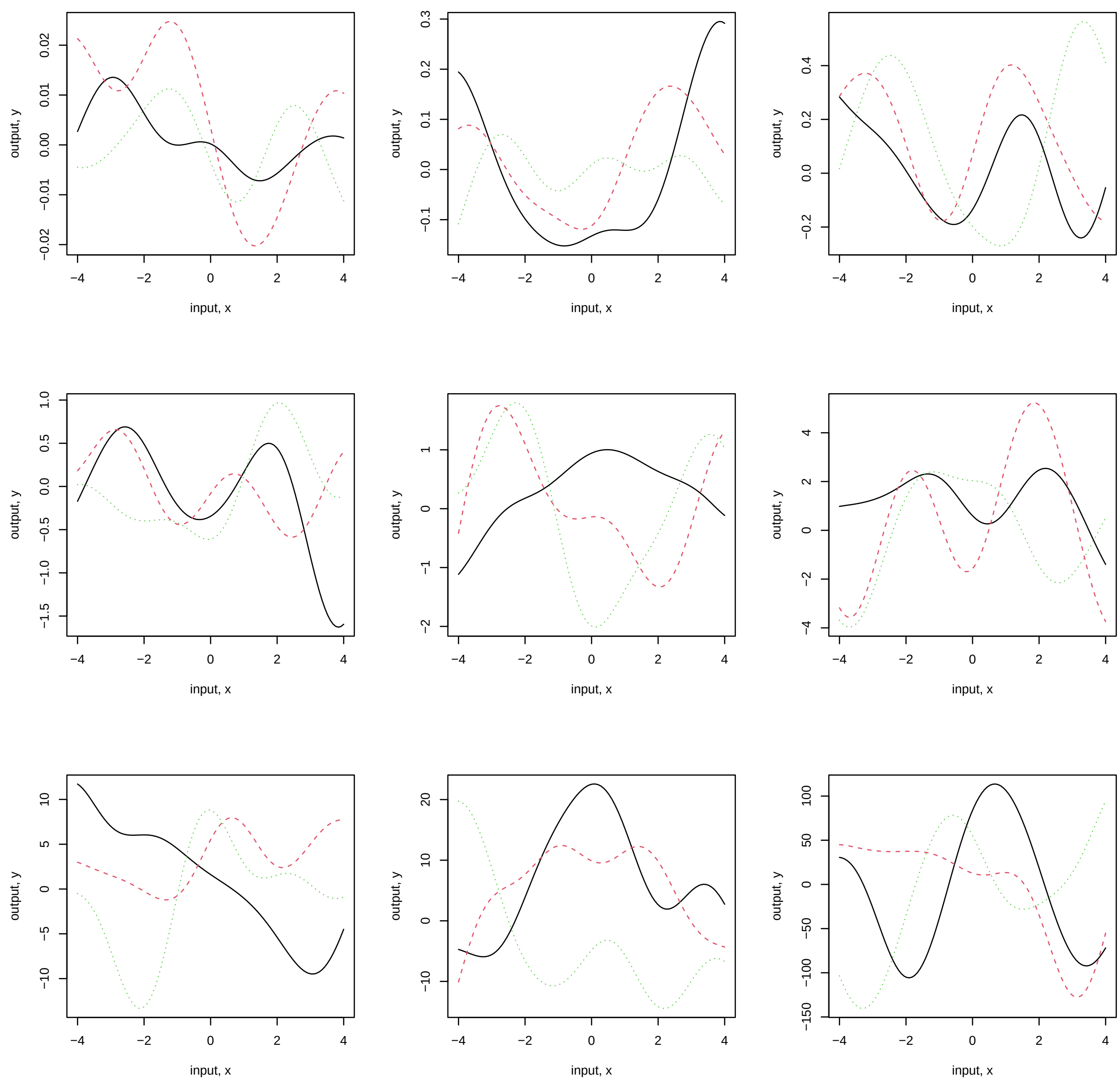

```r
par(op)

par(op)
op <- par(mfrow = c(3, 3))
w <- c(0.01, 0.1, 0.2,
       0.5, 1, 2,
       5, 10, 50)
for (i in 1:9) {
  sim <- mvtnorm::rmvnorm(3, sigma = kernel_pd(omega = w[i]))
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
    arrange(by = x)
  matplot(data[, 1], data[, -1],
    xlab = "input, x",
```

```
    ylab = "output, y",
    "l")
}
```




```
par(op)
op <- par(mfrow = c(3, 3))
s <- c(0.01, 0.1, 0.2,
       0.5, 1, 2,
       5, 10, 50)
for (i in 1:9) {
  sim <- mvtnorm::rmvnorm(3, sigma = kernel_pd(sigma = s[i]))
  data <- cbind(x, t(sim)) %>%
    data.frame() %>%
```

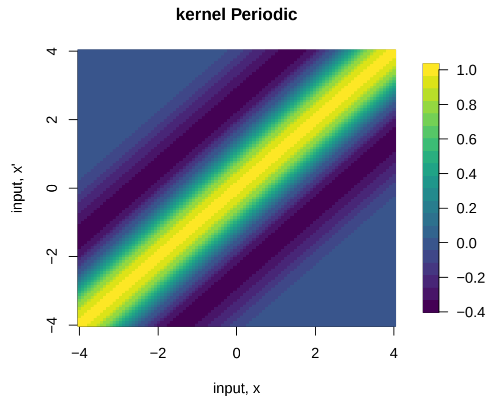
```
    arrange(by = x)
  matplot(data[, 1], data[, -1],
    xlab = "input, x",
    ylab = "output, y",
    "l")
}
```




```
par(op)
```

```
x1 <- seq(-4, 4, length = 100)
kpd <- kernel_pd()
fields::image.plot(x1, x1, kpd,
                   xlab = "input, x",
```

```
                   ylab = "input, x'",
                   main = "kernel Periodic",
                   nlevel = 20,
                   col = viridis::viridis(20))
```



4.2.4 Making New Kernels from Old

$$\tilde{k}(x,\ x') = \frac{k(x,\ x')}{\sqrt{k(x,\ x)}\sqrt{k(x', x')}} \qquad (4.35)$$

4.3 Eigenfunction Analysis of Kernels

an eigenfunction of kernel k with eigenvalue λ with respect to measure eigenfunction μ. The two measures of particular interest to us will be: * (i) Lebesgue measure over a compact subset $\mathcal{C}$ of $\mathbb{R}^D$ * (ii) when there is a density $p(x)$ so that $d\c{z}(x)$ can be written $p(x)dx$.

$$\int k(x,\ x')\phi(x)d\mu(x) = \lambda\phi(x') \qquad (4.36)$$

Theorem 4.2 (Mercer's theorem) Let (X, μ) be a finite measure space and $k \in L_\infty(\mathcal{X}^2,\ \mu^2)$ be a kernel such that $T_k : L_2(X,\ \mu) \to L_2(X,\ \mu)$ is positive definite *(see eq. (4.2))*. Let $\phi_i \in L_2(X,\ \mu)$ be the normalized eigenfunctions of T_k associated with the eigenvalues $\lambda_i > 0$. Then:

- 1. the eigenvalues $\{\lambda_i\}_{\infty}^{i=1}$ are absolutely summable
- 2. $k(x,\ x') = \sum_{i=1}^{\infty} \lambda_i \phi_i(x)\phi_i^*(x') \qquad (4.37)$

holds μ^2 almost everywhere, where the series converges absolutely and uniformly μ^2 almost everywhere

Definition 4.1 A degenerate kernel has only a finite number of non-zero eigenvalues

$$k(x - x') = \int_{\mathbb{R}^D} e^{2\pi i s\cdot(x-x')} d\mu(s) = \int_{\mathbb{R}^D} e^{2\pi i s \mathring{u} x}\left(e^{2\pi i s\cdot x'}\right)^* d\mu(s) \qquad (4.38)$$

4.3.1 An Analytic Example

$$\lambda_k = \sqrt{\frac{2a}{A}}B^k \qquad (4.39)\phi_k(x) = exp\big(-(c-a)x^2\big)H_k(\sqrt{2c}x) \qquad (4.40)$$

$$H_k(x) = (-1)^k exp(x^2)\int d^k dx^k exp(-x^2) \qquad (4.41a)a^{-1} = 4\sigma^2 b^{-1} = 2l^2 c = \sqrt{a^2 + 2ab}A = a + b + cB = b/A$$

4.3.2 Numerical Approximation of Eigenfunctions

$$\lambda_i \phi_i(x') = \int k(x, x') p(x) \phi_i(x) dx \simeq \frac{1}{n} \sum_{l=1}^{n} k(x_l, x') \phi_i(x_l) \quad (4.42)$$

$$K u_i = \lambda_i^{mat} u_i \quad (4.43)$$

$$\phi_i(x') \simeq \frac{\sqrt{n}}{\lambda_i^{mat}} k(x')^\top u_i \quad (4.44) k(x')^\top = \big(k(x1,\ x0),\ ...,\ k(x_n,\ x')\big)$$

4.4 Kernels for Non-vectorial Inputs

4.4.1 String Kernels

$$k(x,\ x') = \sum_{s \in \mathcal{A}^*} w_s \phi_s(x) \phi_s(x') \quad (4.45)$$

4.4.2 Fisher Kernels

$$k(x,\ x') = \phi_\theta^{(x)} M^{-1} \phi_\theta(x') \quad (4.46)$$

$$F = \mathbb{E}_x[\phi_\theta(x) \phi_\theta^\top(x)] \quad (4.47)$$

$$\nabla_\theta (log\ p(y = +1 | x,\ \theta) - log\ p(y = -1 | x,\ \theta))$$

Summary

- constant: $\sigma_0^2\ S$
- linear: $\sum_{d=1}^{D} \sigma_d^2 x_d x'_d$
- polynomial: $(x \cdot x' + \sigma_0^2)^p$
- squared exponential: $exp\Big(\frac{r^2}{2l^2}\Big)\ S,\ ND$
- Matern: $\frac{1}{2^{\nu-1}\Gamma(\nu)} \Big(\frac{\sqrt{2\nu}}{l} r\Big)^\nu K_\nu \Big(\frac{\sqrt{2\nu}}{l} r\Big)\ S,\ ND$
- exponential: $exp\Big(-\frac{r}{l}\Big)\ S,\ ND$
- γ-exponential: $exp\Big(-\big(\frac{r}{l}\big)^\gamma\Big)\ S,\ ND$
- rational quadratic: $(1 + \frac{r^2}{2\alpha l^2})^{-\alpha}\ S,\ ND$
- neural network: $\sin^{-1}\Big(2\bar{x}^\top \sum \bar{x}' \sqrt{(1 + 2\bar{x}^\top \Sigma \bar{x})(1 + 2\bar{x}'^\top \Sigma \bar{x}')}\Big)\ ND$

4.5 Exercises