***Gaussian processes*** are a widely employed statistical tool because of their flexibility and computational tractability. (For instance, one recent area where Gaussian processes are used is in machine learning for [hyperparameter optimization](https://en.wikipedia.org/wiki/Hyperparameter_optimization).)

A stochastic process \{ X_t \}_{t \in \mathbb{I}}is a Gaussian process if (and only if) any finite subcollection of random variables (X_{t_1}, \dots, X_{t_n})has a multivariate Gaussian distribution. Here, \mathbb{I}is the index set for the Gaussian process; most often we have \mathbb{I} = [0, \infty)(to index time) or \mathbb{I} = \mathbb{R}^d(to index space).

The stochastic nature of Gaussian processes also allows it to be thought of as a distribution over functions. One draw from a Gaussian process over corresponds to choosing a function f: \mathbb{I} \mapsto \mathbb{R}according to some probability distribution over these functions.

Gaussian processes are defined by their mean and covariance functions. The covariance (or kernel) function K: \mathbb{I} \times \mathbb{I} \mapsto \mathbb{R}is what characterizes the shapes of the functions which are drawn from the Gaussian process. ***In this post, we will demonstrate how the choice of covariance function affects the shape of functions it produces.*** For simplicity, we will assume \mathbb{I} = \mathbb{R}.

**Overall set-up**

Let’s say we have a zero-centered Gaussian process denoted by GP(m(\cdot), K(\cdot, \cdot)), and that fis a function drawn from this Gaussian process. For a vector (x_1, \dots, x_n), the function values (f(x_1), \dots, f(x_n))must have a multivariate Gaussian distribution with mean (m(x_1), \dots, m(x_n))and covariance matrix \Sigmawith entries \Sigma_{ij} = K(x_i, x_j). We make use of this property to draw this function: we select a fine grid of x-coordinates, use mvrnorm() from the MASS package to draw the function values at these points, then connect them with straight lines.

Assume that we have already written an R function kernel\_fn for the kernel. The first function below generates a covariance matrix from this kernel, while the second takes N draws from this kernel (using the first function as a subroutine):

library(MASS)

# generate covariance matrix for points in `x` using given kernel function

cov\_matrix <- function(x, kernel\_fn, ...) {

outer(x, x, function(a, b) kernel\_fn(a, b, ...))

}

# given x coordinates, take N draws from kernel function at those points

draw\_samples <- function(x, N, seed = 1, kernel\_fn, ...) {

Y <- matrix(NA, nrow = length(x), ncol = N)

set.seed(seed)

for (n in 1:N) {

K <- cov\_matrix(x, kernel\_fn, ...)

Y[, n] <- mvrnorm(1, mu = rep(0, times = length(x)), Sigma = K)

}

Y

}

The ... argument for the draw\_samples() function allows us to pass arguments into the kernel function kernel\_fn.

We will use the following parameters for the rest of the post:

x <- seq(0, 2, length.out = 201) # x-coordinates

N <- 3 # no. of draws

col\_list <- c("red", "blue", "black") # for line colors

**Squared exponential (SE) kernel**

The squared exponential (SE) kernel, also known as the ***radial basis function (RBF) kernel*** or the ***Gaussian kernel*** has the form

\begin{aligned} K(x, x') = \sigma^2 \exp \left[ -\frac{\| x - x' \|^2}{2l^2} \right], \end{aligned}

where \sigma^2 \geq 0and l > 0are hyperparameters. (All kernels have a \sigma^2parameter which determines how variable the function is overall; for simplicity we will assume it to be equal to 1 for the rest of this post.) It is possibly the most commonly used kernel because of its computational tractability. The following code generates 3 draws from the SE kernel with $latex l = 0.2:

se\_kernel <- function(x, y, sigma = 1, length = 1) {

sigma^2 \* exp(- (x - y)^2 / (2 \* length^2))

}

Y <- draw\_samples(x, N, kernel\_fn = se\_kernel, length = 0.2)

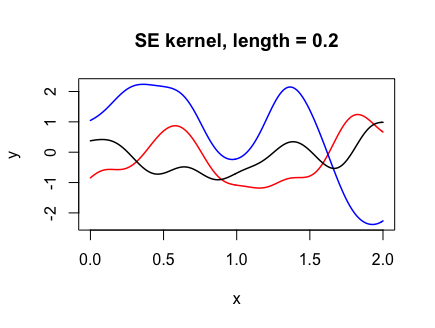
plot(range(x), range(Y), xlab = "x", ylab = "y", type = "n",

main = "SE kernel, length = 0.2")

for (n in 1:N) {

lines(x, Y[, n], col = col\_list[n], lwd = 1.5)

}



The following code shows how changing the “length-scale” parameter l affects the functions drawn. The smaller l is, the more wiggly the functions drawn.

par(mfrow = c(1, 3))

for (l in c(0.2, 0.7, 1.5)) {

Y <- draw\_samples(x, N, kernel\_fn = se\_kernel, length = l)

plot(range(x), range(Y), xlab = "x", ylab = "y", type = "n",

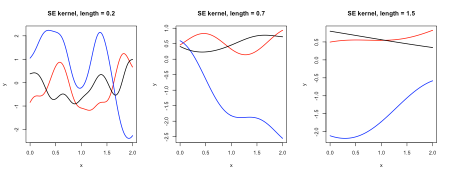
main = paste("SE kernel, length =", l))

for (n in 1:N) {

lines(x, Y[, n], col = col\_list[n], lwd = 1.5)

}

}



**Rational quadratic (RQ) kernel**

The rational quadratic (RQ) kernel has the form

\begin{aligned} K(x, x') = \sigma^2 \left( 1 + \frac{\| x - x' \|^2}{2 \alpha l^2}\right)^{-\alpha}, \end{aligned}

where \sigma \geq 0, l > 0and \alpha > 0are hyperparameters. Below we create a function for the kernel and demonstrate how l affects the functions drawn:

rq\_kernel <- function(x, y, alpha = 1, sigma = 1, length = 1) {

sigma^2 \* (1 + (x - y)^2 / (2 \* alpha \* length^2))^(-alpha)

}

par(mfrow = c(1, 3))

for (a in c(0.01, 0.5, 50)) {

Y <- draw\_samples(x, N, kernel\_fn = rq\_kernel, alpha = a)

plot(range(x), range(Y), xlab = "x", ylab = "y", type = "n",

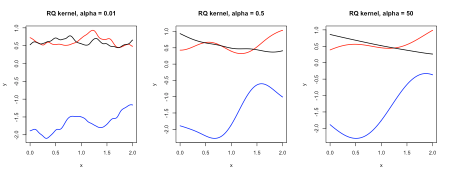
main = paste("RQ kernel, alpha =", a))

for (n in 1:N) {

lines(x, Y[, n], col = col\_list[n], lwd = 1.5)

}

}



**Matérn covariance functions**

The Matérn covariance function has the form

\begin{aligned} K(x, x') = \sigma^2 \frac{2^{1-\nu}}{\Gamma (\nu)} \left( \frac{\sqrt{2\nu} \|x-x'\|}{l}\right)^\nu K_\nu \left( \frac{\sqrt{2\nu} \|x-x'\|}{l} \right), \end{aligned}

where \Gammais the [gamma function](https://en.wikipedia.org/wiki/Gamma_function) and K_\nuis the [modified Bessel function of the second kind](http://mathworld.wolfram.com/ModifiedBesselFunctionoftheSecondKind.html). The hyperparameters are \sigma \geq 0, l > 0and \nu \geq 0. For \nu = p + 1/2for some integer p, the expression on the RHS becomes a little nicer. In practice the values of \nu = 3/2and \nu = 5/2are used much more often than anything else, so that is all we code up here.

matern\_kernel <- function(x, y, nu = 1.5, sigma = 1, l = 1) {

if (!(nu %in% c(0.5, 1.5, 2.5))) {

stop("p must be equal to 0.5, 1.5 or 2.5")

}

p <- nu - 0.5

d <- abs(x - y)

if (p == 0) {

sigma^2 \* exp(- d / l)

} else if (p == 1) {

sigma^2 \* (1 + sqrt(3)\*d/l) \* exp(- sqrt(3)\*d/l)

} else {

sigma^2 \* (1 + sqrt(5)\*d/l + 5\*d^2 / (3\*l^2)) \* exp(-sqrt(5)\*d/l)

}

}

par(mfrow = c(1, 3))

for (nu in c(0.5, 1.5, 2.5)) {

Y <- draw\_samples(x, N, kernel\_fn = matern\_kernel, nu = nu)

plot(range(x), range(Y), xlab = "x", ylab = "y", type = "n",

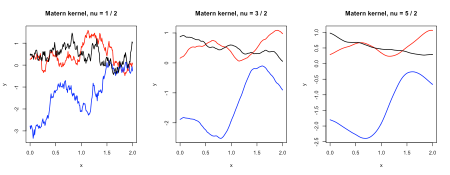
main = paste("Matern kernel, nu =", nu \* 2, "/ 2"))

for (n in 1:N) {

lines(x, Y[, n], col = col\_list[n], lwd = 1.5)

}

}



The paths from the Matérn-1/2 kernel are often deemed too rough to be used in practice.

**Periodic kernel**

The periodic kernel has the form

\begin{aligned} K(x, x') = \sigma^2 \exp \left[ - \frac{2 \sin^2 (\pi \| x - x'\| / p) }{l^2} \right], \end{aligned}

where \sigma \geq 0, l > 0and p > 0are hyperparameters. It is good for modeling functions which repeat themselves exactly. pis the period of the function, as can be seen from the paths below:

period\_kernel <- function(x, y, p = 1, sigma = 1, length = 1) {

sigma^2 \* exp(-2 \* sin(pi \* abs(x - y) / p)^2 / length^2)

}

par(mfrow = c(1, 3))

for (p in c(0.5, 1, 2)) {

Y <- draw\_samples(x, N, kernel\_fn = period\_kernel, p = p)

plot(range(x), range(Y), xlab = "x", ylab = "y", type = "n",

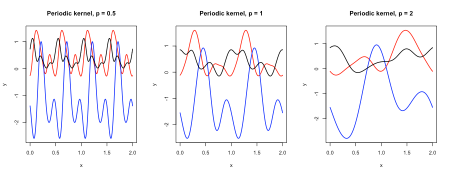
main = paste("Periodic kernel, p =", p))

for (n in 1:N) {

lines(x, Y[, n], col = col\_list[n], lwd = 1.5)

}

}



**Linear/polynomial kernel**

The polynomial kernel has the form

\begin{aligned} K(x, x') = (x^T x' + \sigma^2)^d, \end{aligned}

where d \in \mathbb{N}is the degree of the polynomial and \sigma \geq 0is a hyperparameter. The first plot shows how the value of \sigmacan affect the linear kernel, while the second plot shows how the dimension affects the shape of the polynomial kernel.

poly\_kernel <- function(x, y, sigma = 1, d = 1) {

(sigma^2 + x \* y)^d

}

# linear kernel w different sigma

par(mfrow = c(1, 3))

for (s in c(0.5, 1, 5)) {

Y <- draw\_samples(x, N, kernel\_fn = poly\_kernel, sigma = s)

plot(range(x), range(Y), xlab = "x", ylab = "y", type = "n",

main = paste("Linear kernel, sigma =", s))

for (n in 1:N) {

lines(x, Y[, n], col = col\_list[n], lwd = 1.5)

}

}

# poly kernel of different dimensions

par(mfrow = c(1, 3))

for (d in c(1, 2, 3)) {

Y <- draw\_samples(x, N, kernel\_fn = poly\_kernel, d = d)

plot(range(x), range(Y), xlab = "x", ylab = "y", type = "n",

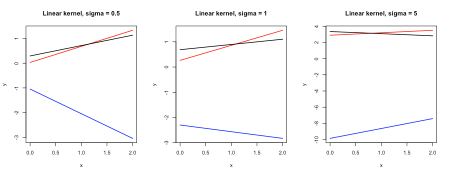
main = paste("Polynomial kernel, d =", d))

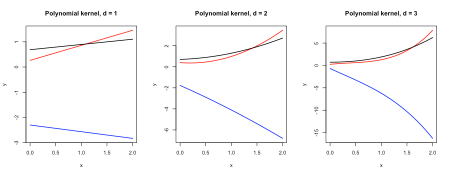
for (n in 1:N) {

lines(x, Y[, n], col = col\_list[n], lwd = 1.5)

}

}





**Brownian motion**

[Brownian motion](https://en.wikipedia.org/wiki/Wiener_process), the most studied object in stochastic processes, is a one-dimensional Gaussian process with mean zero and covariance function K(x, x') = \min (x, x'). Its paths are extremely rough.

bm\_kernel <- function(x, y) {

pmin(x, y)

}

Y <- draw\_samples(x, N, kernel\_fn = bm\_kernel)

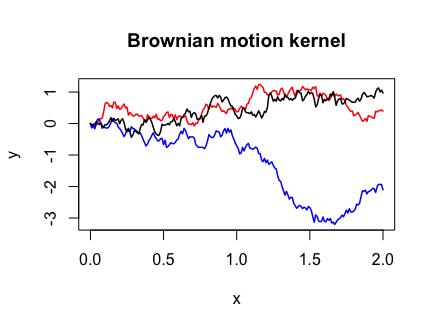
plot(range(x), range(Y), xlab = "x", ylab = "y", type = "n",

main = "Brownian motion kernel")

for (n in 1:N) {

lines(x, Y[, n], col = col\_list[n], lwd = 1.5)

}



Note that I had to use the pmin() function instead of min() in the Brownian motion covariance function. This is because the outer(X, Y, FUN, ...) function we called in cov\_matrix()

“…[extends X and Y] by rep to length the products of the lengths of X and Y before FUN is called.” (from outer() documentation)

pmin() would perform the operation we want, while min() would simply return the minimum value present in the two arguments, not what we want.