GAUSSIAN PROCESS REGRESSION

Mancini Lorenzo & Vicentini Giulio

14/9/2020

THE REGRESSION PROBLEM

Regression is the problem of learning input-output mappings from empirical data (the training dataset). In general we denote the input as x, and the output (or target) as y. Given this training data we wish to make predictions for new inputs x_* that we have not seen in the training set. Thus it is clear that the problem at hand is inductive; we need to move from the finite training data to a unction f that makes predictions for all possible input values. To do this we must make assumptions about the characteristics of the underlying function, as otherwise any function which is consistent with the training data would be equally valid. There are two common approaches. The first is to restrict the class of functions that we consider, for example by only considering linear functions of the input. The second approach is to give a prior probability to every possible function, where higher probabilities are given to functions that we consider to be more likely, for example because they are smoother than other functions.

The first approach has an obvious problem in that we have to decide upon the richness of the class of functions considered; if we are using a model based on a certain class of functions (e.g. linear functions) and the target function is not well modelled by this class, then the predictions will be poor. One may be tempted to increase the flexibility of the class of functions, but this runs into the danger of overfitting, where we can obtain a good fit to the training data, but perform badly when making test predictions.

The second approach appears to have a serious problem, in that surely there are an uncountably infinite set of possible functions, and how are we going to compute with this set in finite time? This is where the Gaussian process comes to our rescue. A Gaussian process is a generalization of the Gaussian probability distribution. Whereas a probability distribution describes random variables which are scalars or vectors (for multivariate distributions), a stochastic process governs the properties of functions. It turns out, that although this idea is a little naive, it is surprisingly close what we need. Indeed, the question of how we deal computationally with these infinite dimensional objects has the most pleasant resolution imaginable: if you ask only for the properties of the function at a finite number of points, then inference in the Gaussian process will give you the same answer if you ignore the infinitely many other points, as if you would have taken them all into account! And these answers are consistent with answers to any other finite queries you may have. One of the main attractions of the Gaussian process framework is precisely that it unites a sophisticated and consistent view with computational tractability.

LINEAR REGRESSION

Recall that in the simple linear regression setting, we have a dependent variable y that we assume can be modeled as a function of an independent variable x, i.e. $y = f(x) + \epsilon$ (where ϵ is the irreducible error) but we assume further that the function f defines a linear relationship and so we are trying to find the parameters θ_0 and θ_1 which define the intercept and slope of the line respectively, i.e. $y = \theta_0 + \theta_1 x + \epsilon$. Bayesian linear regression provides a probabilistic approach to this by finding a distribution over the parameters that gets updated whenever new data points are observed. The GP approach, in contrast, is a non-parametric approach, in that it finds a distribution over the possible functions $f(x)$ that are consistent with the observed data. As with all Bayesian methods it begins with a prior distribution and updates this as data points are observed, producing the posterior distribution over functions.

But, if we don't want to specify upfront how many parameters are involved, we'd like to consider every possible function that matches our data, with however many parameters are involved. That's what non-parametric means: it's not that there aren't parameters, it's that there are infinitely many parameters.

But of course we need a prior before we've seen any data. In fact, given that we don't really want all the functions, we have to put some constraints on it.

GAUSSIAN PROCESSES

DEF: A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

Roughly speaking a stochastic process is a generalization of a probability distribution (which describes a finite-dimensional random variable) to functions. The mathematical framework of GPs is the multivariate Gaussian distribution, for this reason a Gaussian process is completely specified by its mean function and co-variance function. We define mean function $m(x)$ and the covariance function $k(x, x_0)$ of a real process $f(x)$ as

$$m(x) = E[f(x)],$$
$$k(x, x_0) = E[(f(x) - m(x))(f(x_0) - m(x_0))].$$

The covariance matrix ensure that values that are close together in input space will produce output values that are close together, and in this way we can specify the smoothness of the functions. This choice, toghether with the specification of the domain, allow us to reduce the functions set.

Operationally, a GP defines a prior over functions, which can be converted into a posterior over functions once we have seen some data.

THE COVARIANCE FUNCTION

In this case we use the Squared Exponential, or Radial Basis Function. It calculates the squared distance between points and converts it into a measure of similarity, controlled by a tuning parameter.

$$cov\left(f(x_p), f(x_q)\right) = k(x_p, x_q) = exp(-\frac{1}{2}|x_p - x_q|^2).$$

Note that the covariance between the outputs is written as a function of the inputs. For this particular covariance function, we see that the covariance is almost unity between variables whose corresponding inputs are very close, and decreases as their distance in the input space increases. This make the function infinitely differentiable, leading to the process being infinitely mean-square differentiable.

JOINT PROBABILITY

The main advantage of using GP model is that we can turn from the joint probability distribution to the conditional probability distribution that we need in order to estimate the missing outcomes. By focussing on processes which are Gaussian, it turns out that the computations required for inference and learning become relatively easy. If we have the joint probability x_1 and x_2

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{pmatrix}\right),$$

it's possible to get the conditional probability of a variable given the other. This is how, in a Gaussian Process, we can derive the posterior from the prior and our observations. In general, we're not just talking about the joint probability of two variables, as in the bivariate case, but the joint probability of the values of

$f(x)$ for all the x we're looking at. Our posterior is the joint probability of the values corresponding to the data we've observed and the missing data:

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu \\ \mu_* \end{pmatrix}, \begin{pmatrix} K & K_* \\ K_*^T & K_{**} \end{pmatrix}\right).$$

Since there are some missing data f_* we would like to get the probability distribution $p(f_*|x, x_*, f)$. With some algebra it can be shown that from the joint distribution $p(f, f_*)$, we can get the conditional probability $p(f_*|f)$. Now we can sample from this distribution to express our multivariate normal distribution in terms of standard normals:

$$f_* \sim \mu + L\mathcal{N}(0, I),$$

where L is the matrix such that $LL^T = \Sigma_*$. We'll see later that sometimes it is necessary to add a small term to the covariance matrix in order to make the Cholesky decomposition work.

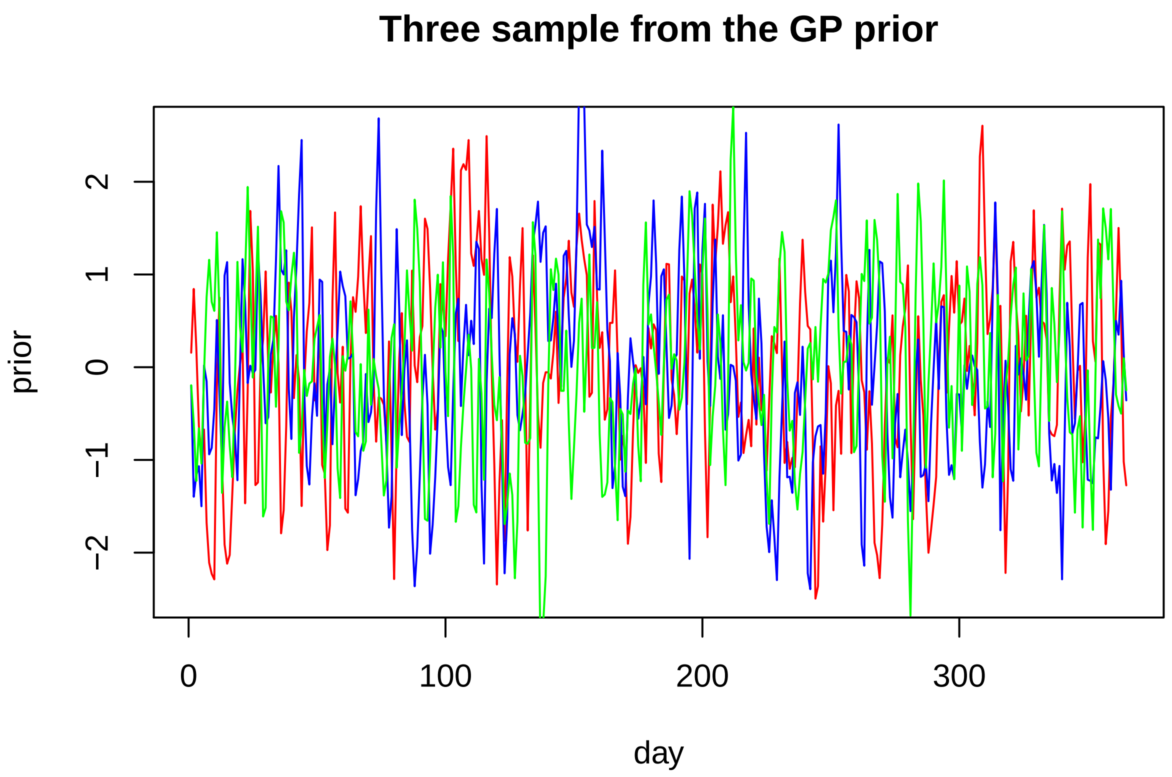
```r
#COVARIANCE FUNCTION
sqdist <- function(a, b){
return((a-b)^2)
}

kernel <- function(a, b, param, sf){
 matr <- matrix(0, nrow = length(a), ncol = length(b))
 for(i in 1:length(a)){
   for(j in 1:length(b)){
   d <- sqdist(a[i], b[j])
   elem <- sf*exp(-0.5* (1/param) *d)
   matr[i, j] = elem
   }
 }
 return(matr)
}
```

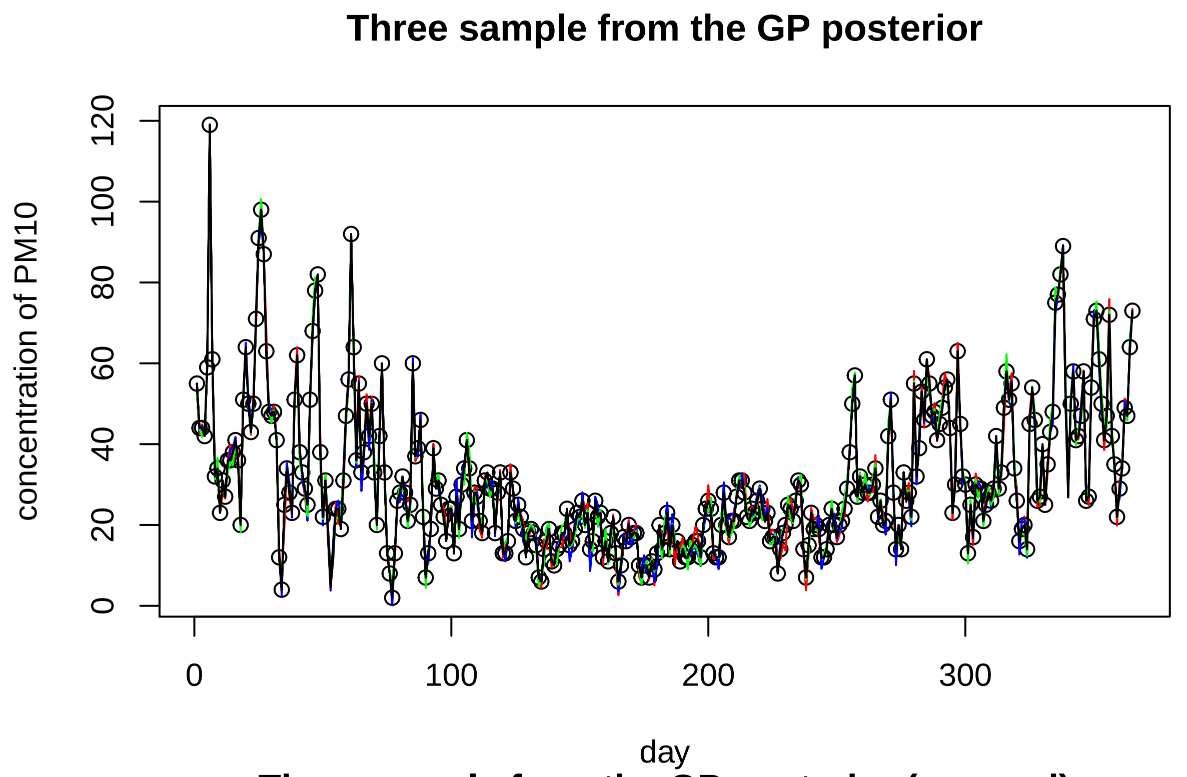
```r
# l = 8
# s1 = 1e-15
# s2 = 0.00005
# s3 = 3

year <- function(y0, l, s1, s2, s3, sf){

 #IMPORT THE DATA
 library('readxl')
 pol <- read_excel(sprintf('data/20%s PM 10 RO-CENTRO.xls', y0),
                   skip=8)
 y <- pol$`µg/m3`
 x <- seq(1, length(pol$Giorno), 1)
 data <- data.frame(x, y)
 p <- x[which(is.na(data$y))]
 data <- na.omit(data)

 #Test data
 Xtest <- x

 #covarianc matrix
 K_ss <- kernel(Xtest, Xtest, l, sf)
 # Get cholesky decomposition (square root) of the covariance matrix
 L1 <- chol(K_ss + s1*diag(length(Xtest)))
```

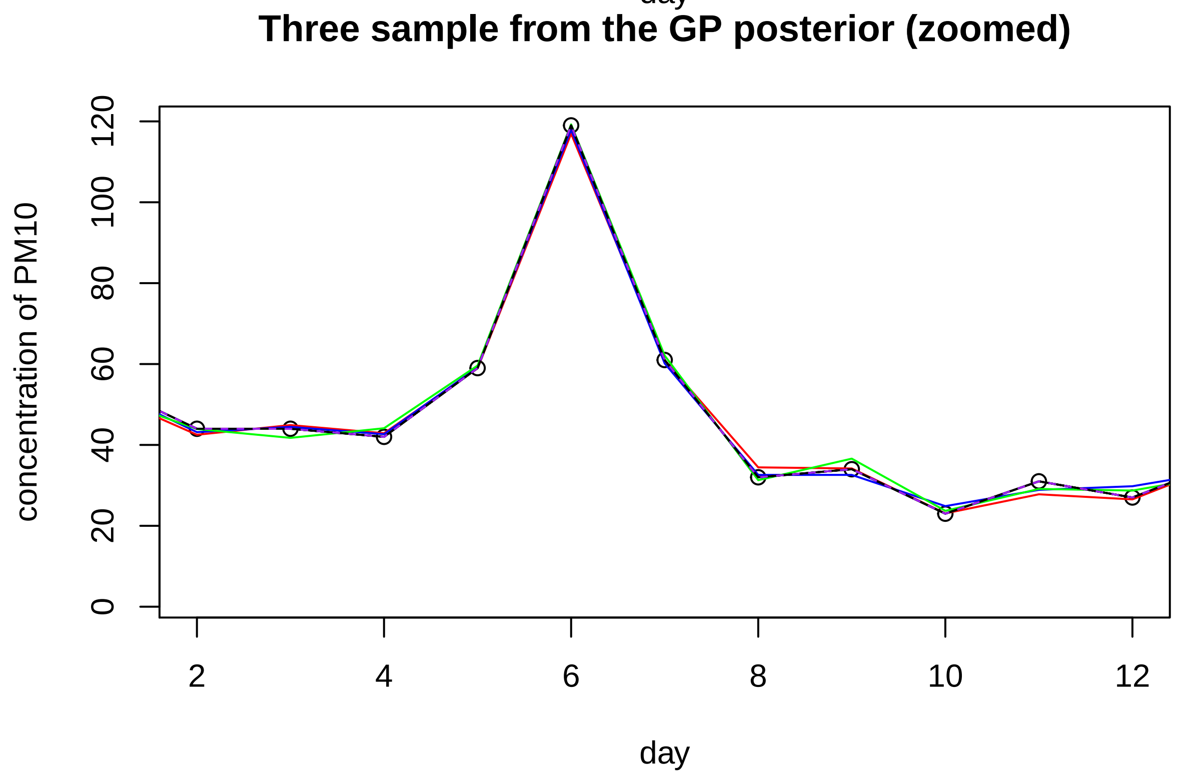
```r
# Sample 3 sets of standard normals for our test points,
#multiply them by the square root of the covariance matrix
f_prior1 <- t(L1) %*% rnorm(length(Xtest))
f_prior2 <- t(L1) %*% rnorm(length(Xtest))
f_prior3 <- t(L1) %*% rnorm(length(Xtest))

#plot the 3 sampled functions.
plot(Xtest, f_prior1, type='l', col='red',
     main = 'Three sample from the GP prior',
     xlab = "day",
     ylab = "prior")
lines(Xtest, f_prior2, col='blue')
lines(Xtest, f_prior3, col='green')


#training data
Xtrain <- data$x
Ytrain <- data$y

# Apply the kernel function to our training points
K <- kernel(Xtrain, Xtrain, l, sf)
L2 <- chol(K + s2*diag(length(Xtrain)))

# Compute the mean at our test points.
K_s <- kernel(Xtrain, Xtest, l, sf)
Lk <- solve(t(L2), K_s)
mu <- t(Lk) %*% solve(t(L2), Ytrain)

# Compute the standard deviation so we can plot it
s2 <- diag(K_ss) - colSums(Lk^2)
stdv <- sqrt(s2)

# Draw samples from the posterior at our test points.
L3 <- chol(K_ss + s3*diag(length(Xtest)) - t(Lk) %*% Lk)
f_post1 <- mu + t(L3) %*% rnorm(length(Xtest))
f_post2 <- mu + t(L3) %*% rnorm(length(Xtest))
f_post3 <- mu + t(L3) %*% rnorm(length(Xtest))

plot(Xtrain, Ytrain, xlim=c(1,365),
     main = "Three sample from the GP posterior",
     xlab = "day",
     ylab = "concentration of PM10")
lines(Xtest, f_post1, col='red')
lines(Xtest, f_post2, col='blue')
lines(Xtest, f_post3, col='green')
lines(Xtest, mu, lty=2)
polygon(c(Xtest, rev(Xtest)), c(mu+2*stdv, rev(mu-2*stdv)),
     col = "grey")

plot(Xtrain, Ytrain, xlim=c(2, 12),
     main = "Three sample from the GP posterior (zoomed)",
     xlab = "day",
     ylab = "concentration of PM10")
```

```
 lines(Xtest, f_post1, col='red')
 lines(Xtest, f_post2, col='blue')
 lines(Xtest, f_post3, col='green')
 lines(Xtest, mu, lty=2)
 polygon(c(Xtest, rev(Xtest)), c(mu+2*stdv, rev(mu-2*stdv)),
      col = "grey")
 lines(Xtest, mu+2*stdv, col='purple', lty=2)

 miss <- mu[p]
 miss
}
```

```
year(y0='18', l=1, s1=1e-15, s2=5e-15, s3=3, sf=1)
```

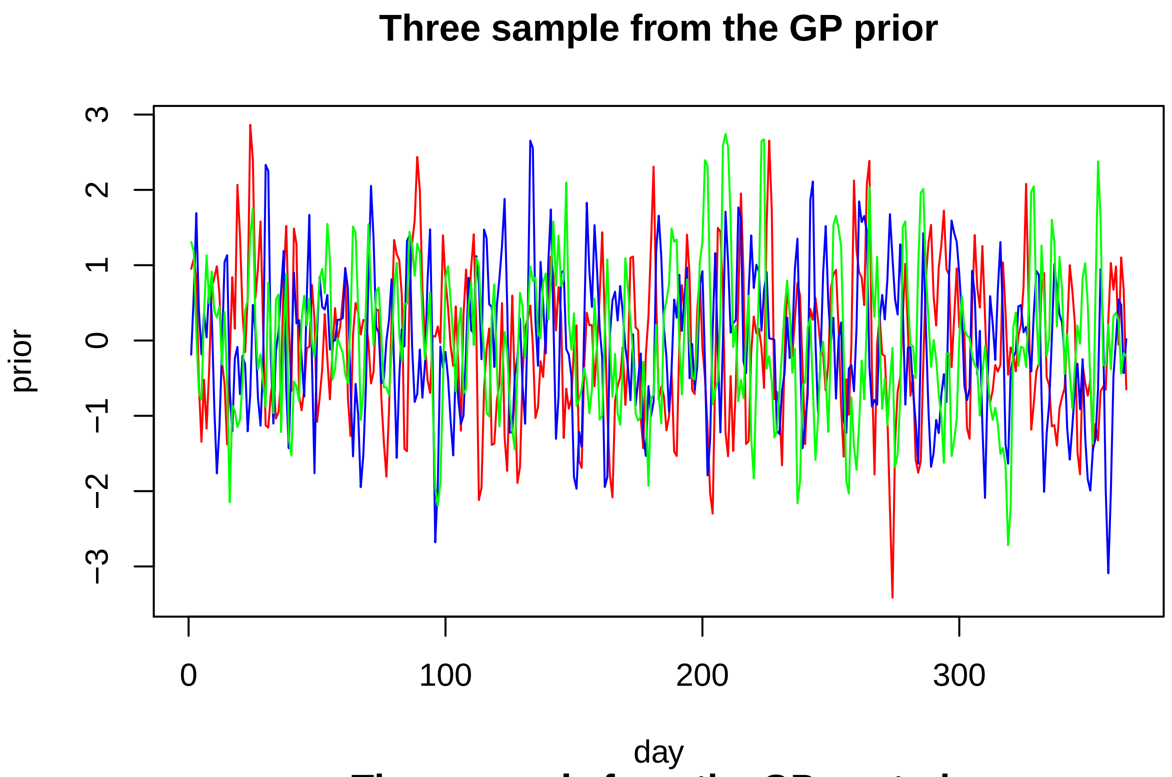


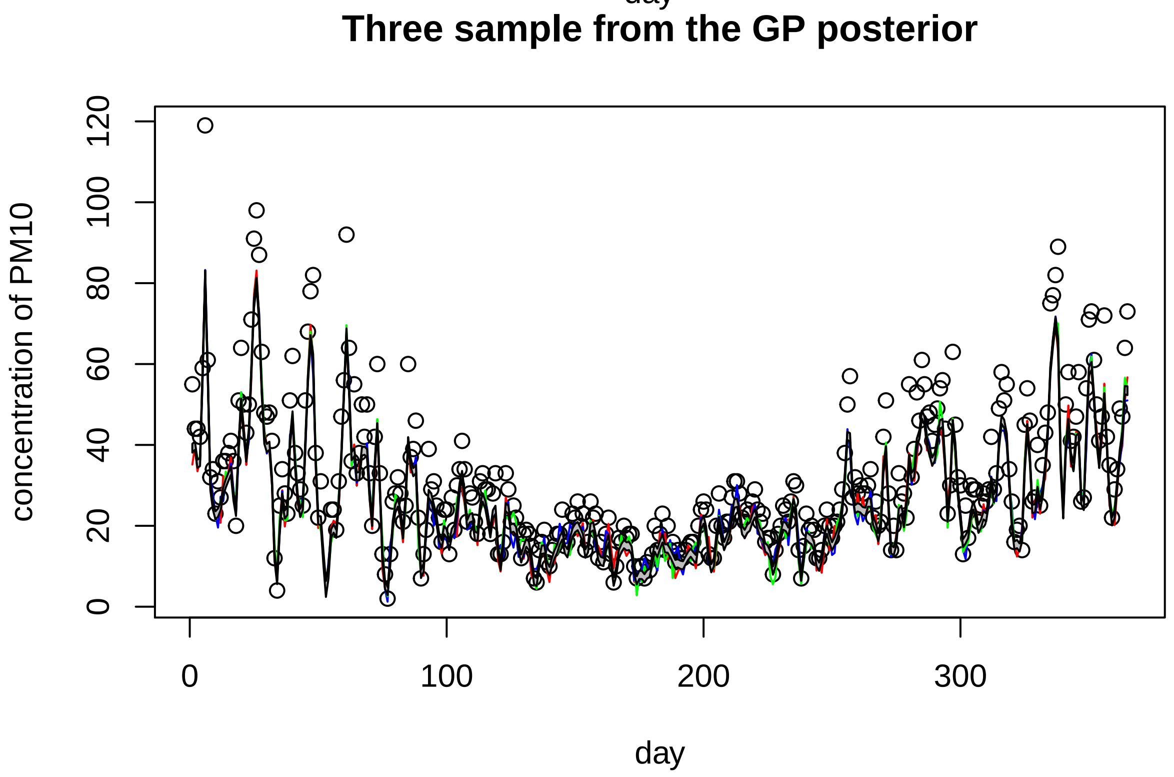


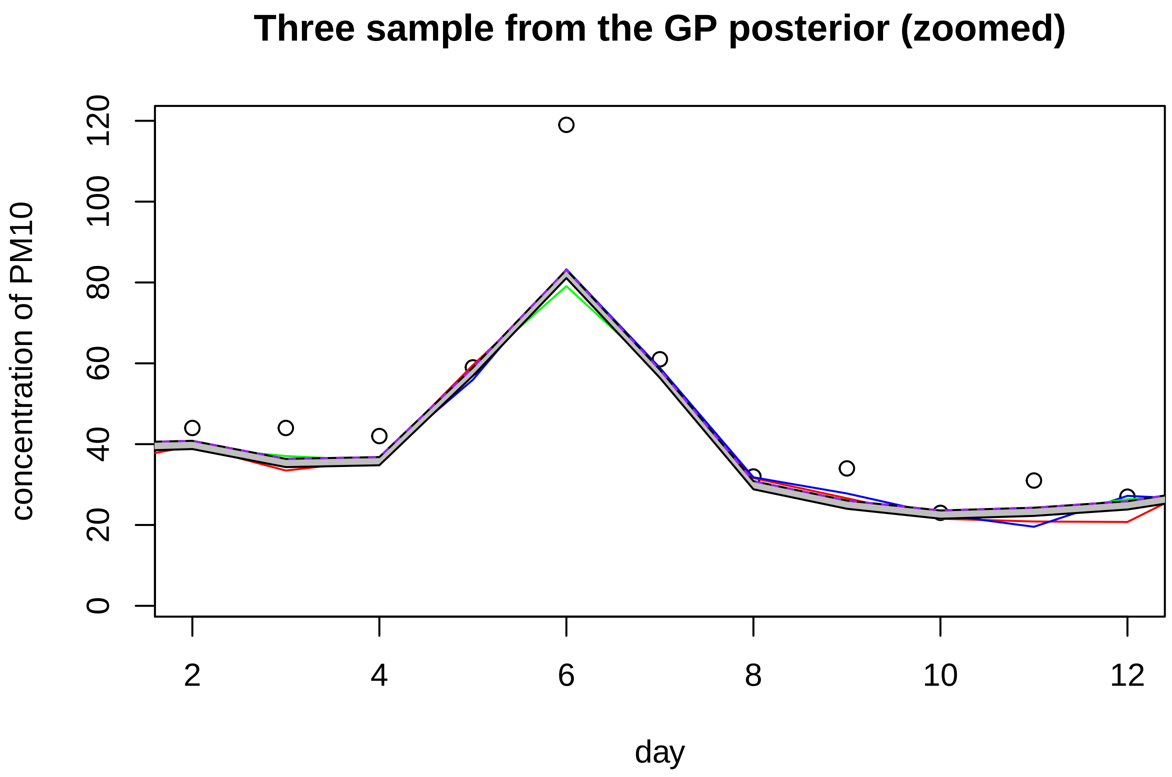


```
## [1] 18.115255  6.468548 12.640635 51.362951 28.189663
```

```
year(y0='18', l=1, s1=1e-15, s2=5e-1, s3=3, sf=1)
```


Three sample from the GP prior
prior
3
2
1
0
−1
−2
−3
0
100
200
300
day


Three sample from the GP posterior
concentration of PM10
120
100
80
60
40
20
0
0
100
200
300
day



```
## [1] 12.178913  4.376567  8.764934 35.463966 23.526809
```

THE HYPERPARAMETERS

Typically the covariance functions that we use will have some free parameters. For example, the squared-exponential covariance function in one dimension has the following form

$$k_y(x_p, x_q) = \sigma_f^2 * exp(-\frac{1}{2l^2}(x_p - x_q)^2) + \sigma_n^2 \delta_{pq}.$$

Observe that the length-scale l, the signal variance σ_f^2 and the noise variance σ_n^2 can be varied. In general we call them the free hyperparameters. The characteristic length-scale can be thought of as roughly the distance you have to move in input space before the function value can change significantly. The σ_n parameter is necessary to ensure that the Cholesky decomposition works. We can see that the smaller σ_2 is, the more the graph of posteriors passes through the training points. Conversely, as σ_2 increases, the posterior graph does not adequately represent our observations. At the end σ_f can be seen as an amplitude of the squared exponential. In our case, for simplicity we took $\sigma_f = 1$.