Gaussian Process Regression

Sam Bowyer

2023-04-21

Task 1

Question 1

Given some function $g : \mathcal{X} \to \mathbb{R}$ we can think of the function $k(x, x') = g(x)g(x')$ as an inner product in $\mathbb{R}$ between $g(x) \in \mathbb{R}$ and $g(x') \in \mathbb{R}$ (also note in particular that clearly k is symmetric: $k(x, x') = k(x', x)$ $\forall x, x' \in \mathcal{X}$), hence k is indeed a kernel.

Question 2

For some constant $a \geq 0$, we may define the function $g : \mathcal{X} \to \mathbb{R}$ by $g(x) = \sqrt{a}\ \forall x \in \mathcal{X}$. Hence by the previous question, the kernel $k(x, x') = a$ can be written as $k(x, x') = g(x)g(x')$, meaning that k is indeed a kernel.

Question 3

Let $n \in \mathbb{N} \setminus \{0\}$, and suppose we have a set of kernels $\{k_j\}_{j=1}^m$ and non-negative real numbers $\{c_j\}_{j=1}^m$. For an arbitrary collection of points $\{x_i\}_{i=1}^n$ let us define the following Gram matrices:

$$\mathbf{K}_j = (k_j(x_i, x_l))_{i,l=1}^n$$

for each $j \in \{1, 2, ..., m\}$. Note that by definition of each k_j being a kernel, we also have that each matrix $\mathbf{K}_j$ is symmetric and positive semidefinite.

Next, let us define the Gram matrix $\mathbf{K} = (k(x_i, x_l))_{i,l=1}^n$ using the function $k = \sum_{j=1}^m c_j k_j$, which we intend to prove is in fact a kernel. Importantly, note that we can write this Gram matrix as $\mathbf{K} = \sum_{j=1}^m c_j \mathbf{K}_j$.

In order to prove that k is a kernel, it suffices to show that $\mathbf{K}$ is symmetric and positive semidefinite. Firstly, since a linear combination of symmetric matrices is also symmetric, and each $\mathbf{K}_j$ is symmetric, it then follows that $\mathbf{K}$ is symmetric. Secondly, observe that for any vector $a \in \mathbb{R}^n$ we have

$$\begin{aligned} a^T \mathbf{K} a &= a^T \left(\sum_{j=1}^m c_j \mathbf{K}_j \right) a \\ &= \sum_{j=1}^m (c_j a^T \mathbf{K}_j a) \\ &\geq 0, \end{aligned}$$

since each $\mathbf{K}_j$ being positive semidefinite means that $a^T \mathbf{K}_j a \geq 0$. Hence $\mathbf{K}$ is also positive semidefinite and thus k is a kernel.

Question 4

Since k is a kernel on $\mathbb{R}^p$ then for any $n \in \mathbb{N} \setminus \{0\}$ and any data points $x_1, ..., x_n \in \mathbb{R}^p$, the Gram matrix $\mathbf{K} = (k(x_i, x_l))_{i,l=0}^n$ is symmetric and positive semidefinite, properties that clearly still hold if we restrict our data points to $\mathcal{X}$, i.e. if $x_1, ..., x_n \in \mathcal{X} \subset \mathbb{R}^p$. Because we know that these properties still hold on the Gram matrix of the kernel restricted to $\mathcal{X} \times \mathcal{X}$, it must then be true that this restriction of the kernel is a valid kernel on $\mathcal{X}$.

Task 2

In this task we will fit a Gaussian process regression model on the Bone Mineral Density dataset[1]—containing 485 recordings of spinal bone mineral density for subjects of varying ages.

```r
dataset = read.table("boneMineralDensity.dat", header=T)
head(dataset)
```

```
##   idnum   age gender      spnbmd
## 1     1 11.70   male 0.018080670
## 2     1 12.70   male 0.060109290
## 3     1 13.75   male 0.005857545
## 4     2 13.25   male 0.010263930
## 5     2 14.30   male 0.210526300
## 6     2 15.30   male 0.040843210
```

```r
dim(dataset)
```

```
## [1] 485   4
```

First we split our dataset into our inputs X and outputs Y.

```r
X = as.matrix(dataset$age)
Y = as.matrix(dataset$spnbmd)
```

We will fit this model using a Gaussian kernel $k(x, x') = \exp(-\psi||x - x'||^2)$ for some $\psi > 0$ that will be chosen using the empirical Bayes approach alongside the choice of model variance $\lambda = \sigma^2$. To do this, we'll use the `optim` function to find the values

$$\begin{aligned}(\lambda^*, \psi^*) &\in \underset{\lambda,\psi}{\operatorname{argmax}} \log p(y^0_{1:n}|\lambda, k) \\ &= \underset{\lambda,\psi}{\operatorname{argmax}} \left(-\frac{1}{2}\log|\mathbf{K}_n + \lambda\mathbf{I}_n| - \frac{1}{2}(y^0_{1:n})^T(\mathbf{K}_n + \lambda\mathbf{I}_n)^{-1}y^0_{1:n}\right).\end{aligned}$$

That is, we're choosing the parameters λ, ψ of our prior so as to maximise the marginal likelihood of our observations $y^0_{1:n}$, which we calculate in the following function for a given λ and ψ.

```r
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:purrr':
##
##     cross
```

```r
marg_lik <- function(Xs, ys, lambda, psi){
  kernel = rbfdot(psi)
  K_n = kernelMatrix(kernel, Xs)

  return(-0.5*log(det(K_n + lambda*diag(nrow(Xs)))) -
           0.5*t(ys) %*% solve(K_n + lambda*diag(nrow(Xs))) %*% ys)
}
```

[1]https://hastie.su.domains/ElemStatLearn/

Now we can optimise this with a variety of initialisations for λ and ψ (which is important because this is not a convex optimisation problem, and we want to avoid local maxima). In particular, we'll use all combinations of $\lambda, \psi \in \{0.5, 1, 3\}$.

```
lambdas = c(0.5, 1, 3)
psis    = c(0.5, 1, 3)
log_psis = log(psis)

empiricalBayes <- function(Xs, ys, lambdas, log_psis){

  opts = list(vals = c(), ls = c(), ps = c())
  max_marg_lik = -Inf
  opt_l = NA
  opt_p = NA

  # optimise params for all combinations of initialisations
  for (l in lambdas){
    for (l_p in log_psis){
      # print(par[1], exp(par[2]))
      opt = optim(c(l, l_p), function(par) - marg_lik(Xs, ys, par[1], exp(par[2])))
      if (opt$val > max_marg_lik){
        max_marg_lik = opt$val
        opt_l = opt$par[1]
        opt_p = exp(opt$par[2])
      }
    }
  }
  return(c(opt_l, opt_p))
}
```

Using this we obtain the optimum parameters $\lambda^* = 0.1583235$ and $\psi^* = 3.1791619$.

```
params = empiricalBayes(X, Y, lambdas, log_psis)
params
```

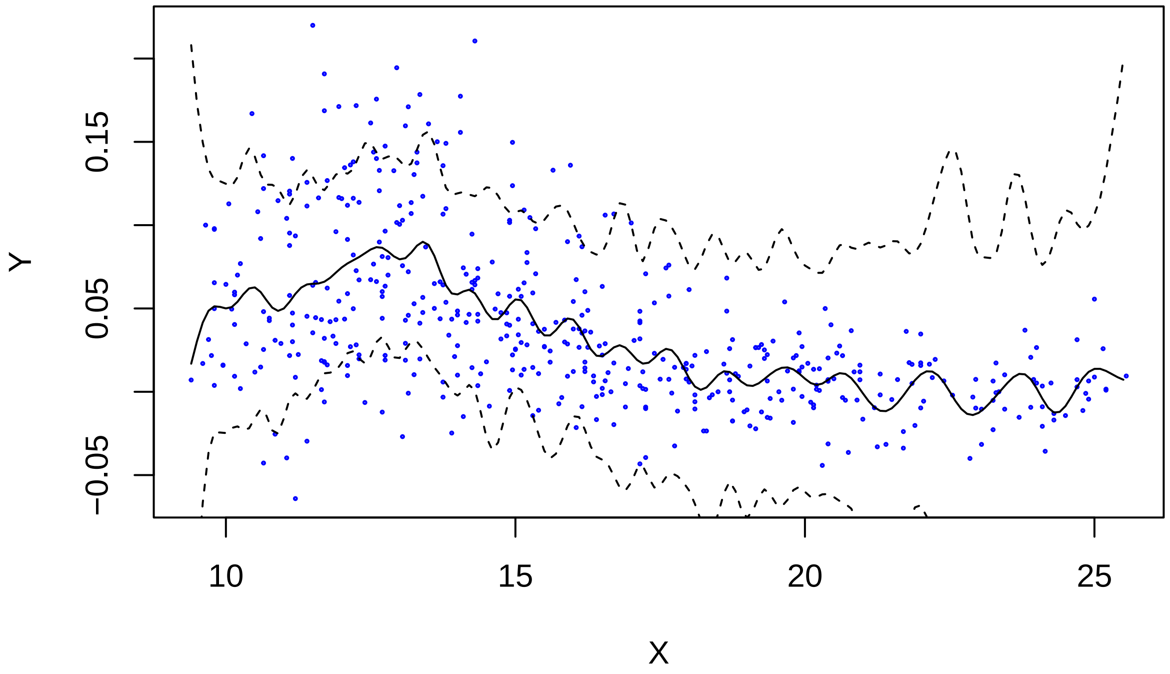
```
## [1] 0.1583235 3.1791619
```

Now we can fit our models $f|y^0_{1:n} \sim \text{GP}(f_n, k_n)$ where $f_n : \mathcal{X} \to \mathbb{R}$ and $k_n : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ are given by

$$f_n(x) = k_n(x)^T(\mathbf{K}_n + \lambda\mathbf{I}_n)^{-1}y^0_{1:n}$$
$$k_n(x, x') = k(x, x') - k_n(x)^T(\mathbf{K}_n + \lambda\mathbf{I}_n)^{-1}k_n(x'),$$

for $x, x' \in \mathcal{X}$, with $\mathbf{K}_n = (k(x_i, x_l))^n_{i,l=1} \in \mathbb{R}^{n\times n}$ and $k_n(x) = (k(x^0_1, x), ..., k(x^0_n, x)) \in \mathbb{R}^n$.

```
kernelVec <- function(Xs, x_new, kernel){
  k_n = rep(0, length(Xs))
  for (i in 1:length(Xs)){
    k_n[i] = kernel(Xs[i], x_new)
  }
  k_n
}

fitGP <- function(Xs, ys, kernel, lambda){
  K_n = kernelMatrix(kernel, Xs)
  invertedMatrix = solve(K_n + lambda*diag(nrow(Xs)))
  force(invertedMatrix)
  f_n = function(new_x){
    k_n = kernelVec(Xs, new_x, kernel)
    t(k_n) %*% invertedMatrix %*% ys
  }

  k_n = function(new_x1, new_x2){
    k_n1 = kernelVec(Xs, new_x1, kernel)
    k_n2 = kernelVec(Xs, new_x2, kernel)
    kernel(new_x1, new_x2) - t(k_n1) %*% invertedMatrix %*% k_n2
  }

  return(c(f_n, k_n))
}

gp = fitGP(X, Y, rbfdot(params[2]), params[1])
age_range = seq(min(dataset$age), max(dataset$age), 0.1)
library(purrr)
mean_vals = as.numeric(map(age_range, gp[[1]]))
var_vals  = as.numeric(map(age_range, function(x1) gp[[2]](x1, x1)))
conf_intv = (1-dnorm(-0.5/2))*sqrt(var_vals)

plot(X, Y, col="blue", cex=0.2)
lines(age_range, mean_vals)
lines(age_range, mean_vals+conf_intv,lty=2)
lines(age_range, mean_vals-conf_intv,lty=2)
```



In the plot above, the solid line represents the mean function $f_n(x)$ and the dashed lines show a 95% confidence interval with width $2z_{0.975}\sqrt{k_n(x,x)}$, where $z_{0.975} = 1 - \Phi(-0.025)$ with Φ the c.d.f. of $\mathcal{N}(0,1)$.

Low-Rank Approximation

We can also implement a low-rank approximation for the posterior distribution by defining a kernel $\tilde{k}^{(d)}$ using $d < n$ data points, which we will chose uniformly at random. Let us suppose, without loss of generality, that we choose the first d datapoints, i.e. $\{x_i^0\}_{i=1}^d$, and therefore we define $\mathbf{K}_d \in \mathbb{R}^{d\times d}$ to be the upper-left $d \times d$ submatrix of $\mathbf{K}$ and $k_d(x) = (k(x_1^0, x), ..., k(x_d^0, x)) \in \mathbb{R}^d$.

With this, we can write our low-rank kernel as

$$\tilde{k}^{(d)}(x, x') = k_d(x)^T \mathbf{K}_d^{-1} k_d(x').$$

After first defining the matrix $\mathbf{K}_{nd} = [k_d(x_1^0), ..., k_d(x_d^0)]^T \in \mathbb{R}^{n\times d}$ we can define our GP's mean and covariance functions:

$$f_n^{(d)}(x) = k_d(x)^T(\lambda \mathbf{K}_d + \mathbf{K}_{nd}^T \mathbf{K}_{nd})^{-1} \mathbf{K}_{nd}^T y_{1:n}^0$$
$$k_n^{(d)}(x, x') = \lambda k_d(x)^T(\lambda \mathbf{K}_d + \mathbf{K}_{nd}^T \mathbf{K}_{nd})^{-1} k_d(x').$$

Importantly, note that we don't have to actually compute $\tilde{k}^{(d)}$!

As we first go through, we'll use $d = 25$.

```
d = 25
d_idx = sample(1:nrow(X), d)

fitGPApprox <- function(Xs, ys, kernel, lambda, d_idx){
  d = length(d_idx)

  K_n  = kernelMatrix(kernel, Xs)
  K_nd = K_n[,d_idx]
  K_d  = K_n[d_idx,d_idx]

  invertedMatrix = solve(lambda*K_d + t(K_nd)%*%K_nd + 1e-5*diag(d))
  force(invertedMatrix)

  f_n = function(new_x){
    k_d = kernelVec(Xs[d_idx,], new_x, kernel)
    t(k_d) %*% invertedMatrix %*% t(K_nd) %*% ys
  }

  k_n = function(new_x1, new_x2){
    k_d1 = kernelVec(Xs[d_idx,], new_x1, kernel)
    k_d2 = kernelVec(Xs[d_idx,], new_x2, kernel)
    lambda*t(k_d1) %*% invertedMatrix %*% k_d2
  }

  return(c(f_n, k_n))
}
```

However, our low-rank approximation requires us to use a different marginal likelihood when using the empirical Bayes method to choose parameters. In particular, we now want to maximise

$$\log \tilde{p}^{(d)}(y_{1:n}^0|\lambda, k) = -\frac{1}{2}\left(\log|\Sigma_d| - \log|\mathbf{K}_d| + (n-d)\log\lambda\right)$$
$$- \frac{||y_{1:n}^0||^2 - ||\Sigma_d^{-1/2}\mathbf{K}_{nd}^T y_{1:n}^0||^2}{2\lambda} - \frac{n}{2}\log 2\pi$$

where $\Sigma_d = \lambda \mathbf{K}_d + \mathbf{K}_{nd}^T \mathbf{K}_{nd}$.

```
library(expm)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'expm'

## The following object is masked from 'package:Matrix':
##
##     expm
```

```
marg_lik_approx <- function(Xs, ys, lambda, psi, d_idx){
  d = length(d_idx)
  n = nrow(Xs)

  kernel = rbfdot(psi)
  K_n = kernelMatrix(kernel, Xs)
  K_nd = K_n[,d_idx]
  K_d  = K_n[d_idx,d_idx]

  Sig_d = lambda*K_d + t(K_nd) %*% K_nd

  vec = solve(sqrtm(Sig_d)) %*% t(K_nd) %*% ys

  return(-0.5*(log(det(Sig_d)) + log(det(K_d)) + (n- d) * log(lambda)) -
           (t(ys) %*% ys - t(vec) %*% vec)/(2*lambda) - (n * log(2 * pi))/2)
}
```

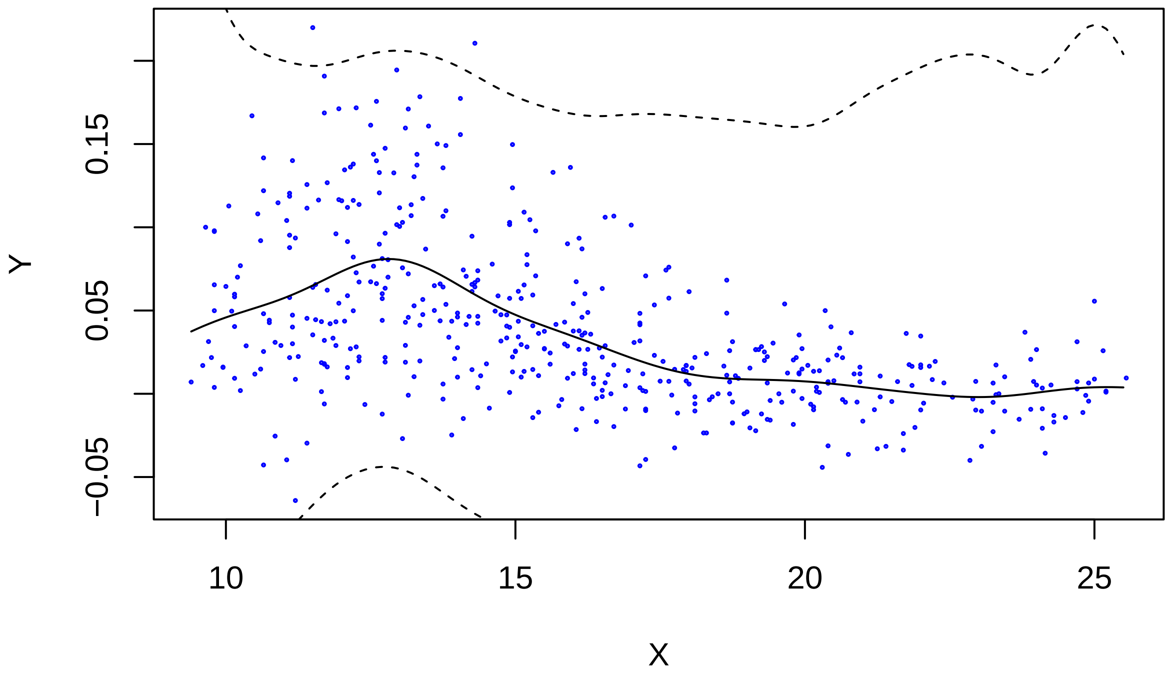
Now we can obtain parameters optimised for this likelihood.

```
lambdas = c(0.5, 1, 3)
psis    = c(0.5, 1, 3)
log_psis = log(psis)

empiricalBayesApprox <- function(Xs, ys, lambdas, log_psis, d_idx){

  opts = list(vals = c(), ls = c(), ps = c())
  max_marg_lik = -Inf
  opt_l = NA
  opt_p = NA

  # optimise params for all combinations of initialisations
  for (l in lambdas){
    for (l_p in log_psis){
      opt = optim(c(l, l_p), function(par) - marg_lik_approx(Xs, ys, par[1], exp(par[2]), d_idx))
      if (opt$val > max_marg_lik){
        max_marg_lik = opt$val
        opt_l = opt$par[1]
        opt_p = exp(opt$par[2])
      }
    }
  }
  return(c(opt_l, opt_p))
}

params = empiricalBayesApprox(X, Y, lambdas, log_psis, d_idx)
params
```

```
## [1] 3.3199219 0.2624543
```

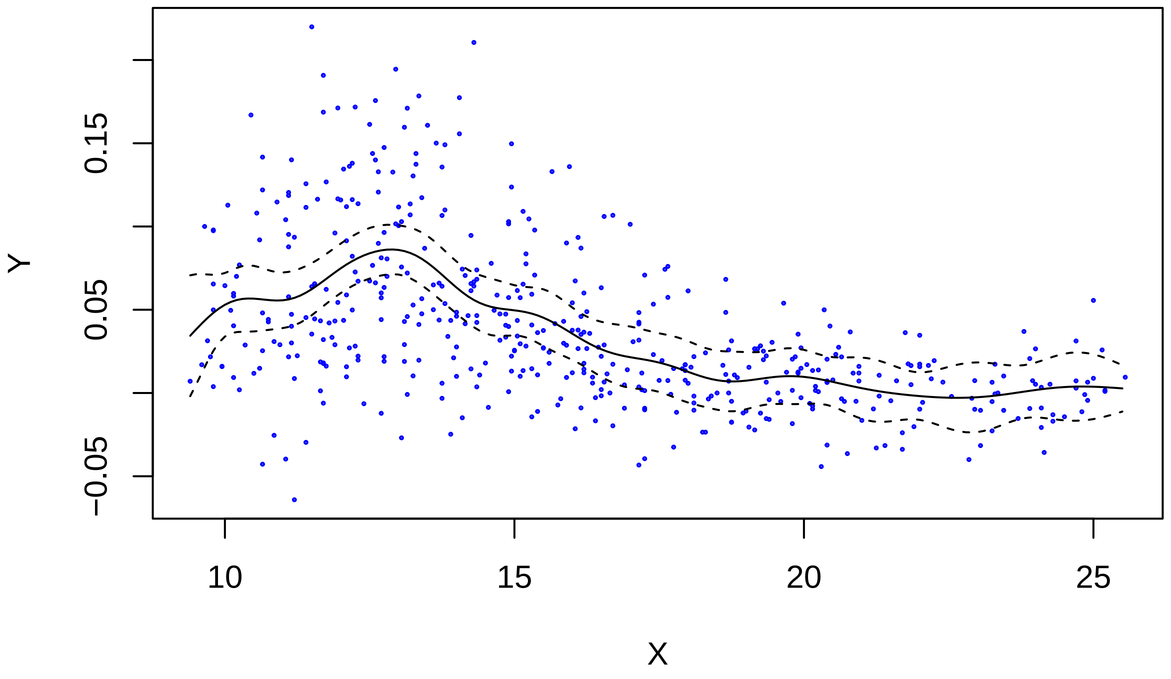
And finally we can fit our low-rank GP for $d = 25$.

```r
gp = fitGPApprox(X, Y, rbfdot(params[2]), params[1], d_idx)
mean_vals = as.numeric(map(age_range, gp[[1]]))
var_vals  = as.numeric(map(age_range, function(x1) gp[[2]](x1, x1)))
conf_intv = (1-pnorm(-0.5/2))*sqrt(var_vals)

plot(X, Y, col="blue", cex=0.2)
lines(age_range, mean_vals)
lines(age_range, mean_vals+conf_intv,lty=2)
lines(age_range, mean_vals-conf_intv,lty=2)
```



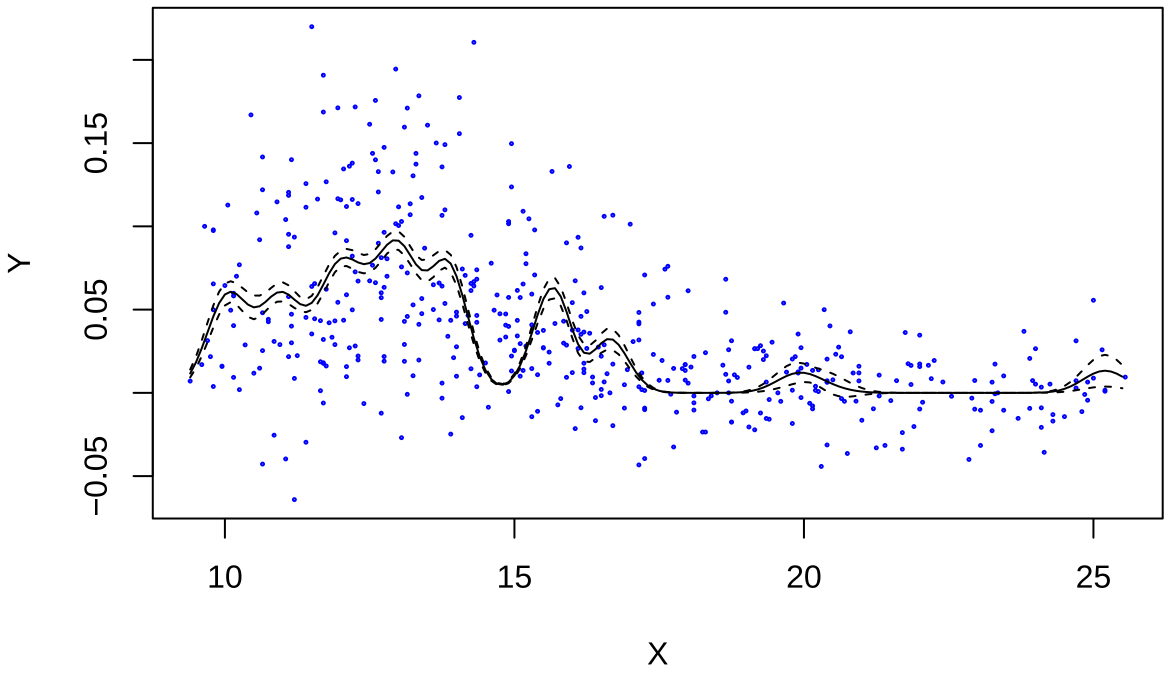
It is important to note that the resulting GP not only depends on the quality of the optimisation for our parameters, but also—in this low-rank approximation case—on the d data points chosen to form the approximation. Below we present another different GPs calculated based on a different random draw for d_idx (but still with $d = 25$) for which we'll obtain a significantly different GP.

```r
lambdas = c(0.5, 1, 3)
psis    = c(0.5, 1, 3)
log_psis = log(psis)

approxGPPlot <- function(Xs, Ys, lambdas, log_psis, d){
  n = length(Xs)
  d_idx = sample(1:n, d)
  params = empiricalBayesApprox(Xs, Ys, lambdas, log_psis, d_idx)

  gp = fitGPApprox(X, Y, rbfdot(params[2]), params[1], d_idx)
  mean_vals = as.numeric(map(age_range, gp[[1]]))
  var_vals  = as.numeric(map(age_range, function(x1) gp[[2]](x1, x1)))
  conf_intv = (1-pnorm(-0.5/2))*sqrt(var_vals)

  plot(X, Y, col="blue", cex=0.2)
  lines(age_range, mean_vals)
  lines(age_range, mean_vals+conf_intv,lty=2)
  lines(age_range, mean_vals-conf_intv,lty=2)
}

approxGPPlot(X, Y, lambdas, log_psis, 25)
```

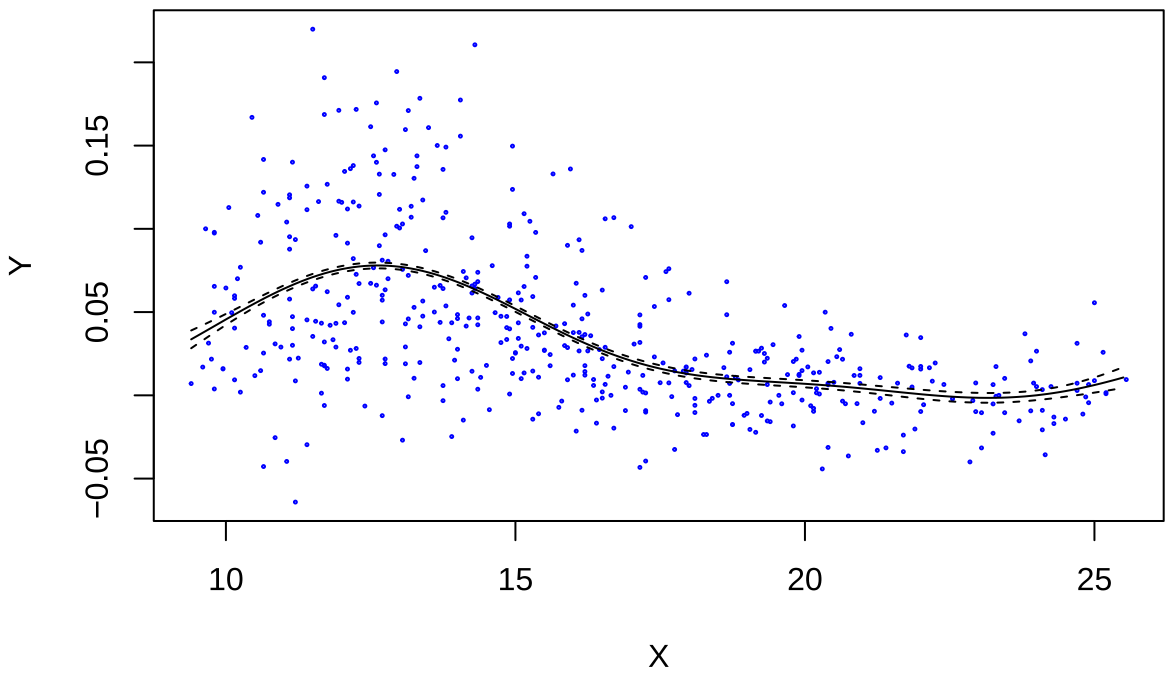


And we will also plot a low-rank GP with lower values of d, in particular we'll use $d = 15$ and $d = 10$, for which we'll see increasingly tighter confidence intervals.

```
approxGPPlot(X, Y, lambdas, log_psis, 15)
```



```
approxGPPlot(X, Y, lambdas, log_psis, 10)
```



Y
−0.05
0.05
0.15
10
15
20
25
X