Imperial College London

MACHINE LEARNING & GLOBAL HEALTH NETWORK

Practical Gaussian Process Regression

Shozen Dan and Oliver Ratmann
Imperial College London

Feb. 22, 2024

Table of contents

Gaussian process basics

Bayesian linear regression

Back to foundations

Let $Y_i, x_i,\ (i = 1, \ldots, N)$ be the outcome and input, respectively. Consider the following simple linear regression model.

$$Y_i \sim \mathcal{N}\left(f_\theta(x_i), \sigma_\varepsilon^2\right)$$
$$f_\theta(x_i) = \beta_0 + \beta_1 x_i.$$

In the Bayesian framework, the parameters of $f_\theta(x_i)$, $\theta = (\beta_0, \beta_1)$, are given the following priors

$$\beta_0 \sim \mathcal{N}\left(\mu_0, \sigma_0^2\right), \quad \beta_1 \sim \mathcal{N}\left(\mu_1, \sigma_1^2\right)$$

Every time we draw β_0, β_1 from the prior, f_θ is a different line.

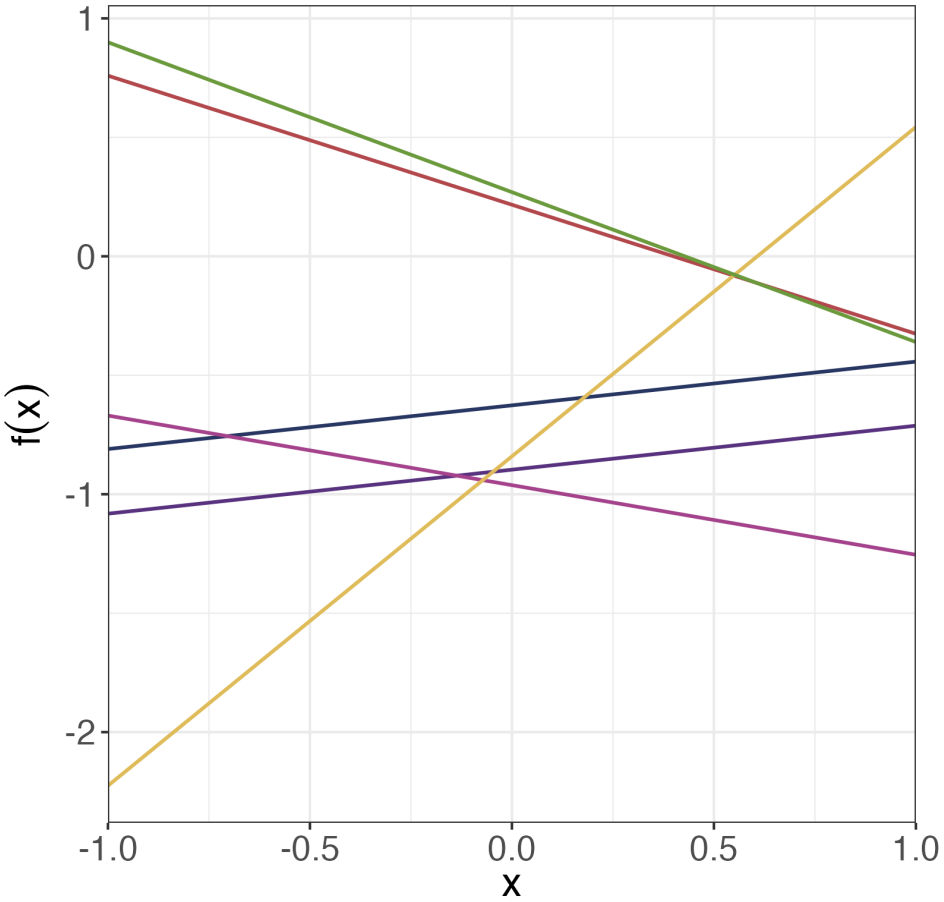
Key point

We can interpret f_θ as a stochastic process **which can be used as a prior over the space of linear functions**.

$$f_\theta \sim \mathcal{LP}(\mu_0, \mu_1, \sigma_0, \sigma_1)$$

※ $\mathcal{LP}$ stands for Linear Process. This is not a conventional notation.

Samples from the prior



Bayesian linear regression

Bayesian inference

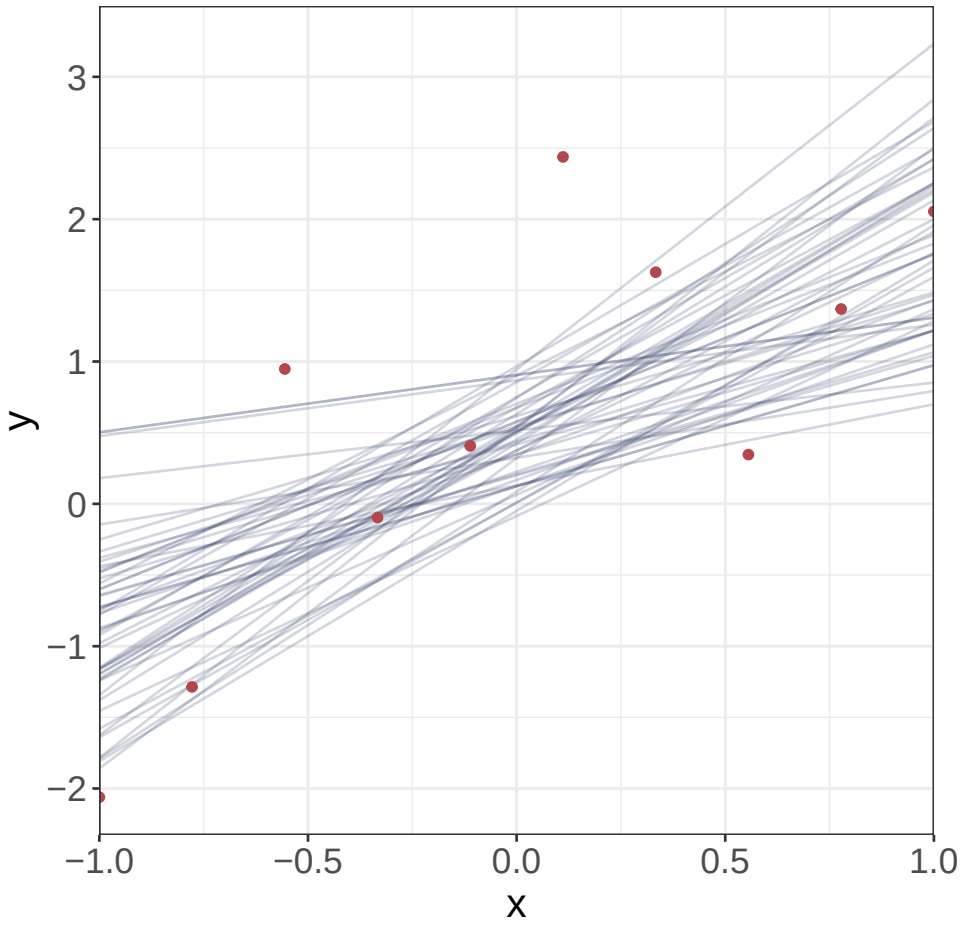
In Bayesian inference, we apply the Bayes rule

$$p(\theta|\boldsymbol{y}) \propto p(\boldsymbol{y}|\theta)p(\theta)$$

to remove lines in the prior that does not fit the observed data. Here, $\boldsymbol{y} = (y_1, \ldots, y_N)^\top$, $\theta = (\beta_0, \beta_1)$.

Issues

Linear functions can only model linear relationships. We would like to model complex non-linear relationships as well.



Bayesian polynomial regression

A simple extension

The polynomial regression model is a simple extension of linear regression:

$$Y_i \sim \mathcal{N}\left(f_\theta(x_i), \sigma_\varepsilon^2\right)$$
$$f_\theta(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_M x_i^M.$$

We often call $\phi_1(x) = x, \phi_2(x) = x^2, \ldots, \phi_M(x) = x^M$ **basis functions**. We may give $\theta = (\beta_0, \beta_1, \ldots, \beta_M)$ the following priors:

$$\beta_0 \sim \mathcal{N}(0, 10^2), \ \beta_m \overset{i.i.d.}{\sim} \mathcal{N}(\mu, \sigma^2) \ (m = 1, \ldots, M).$$

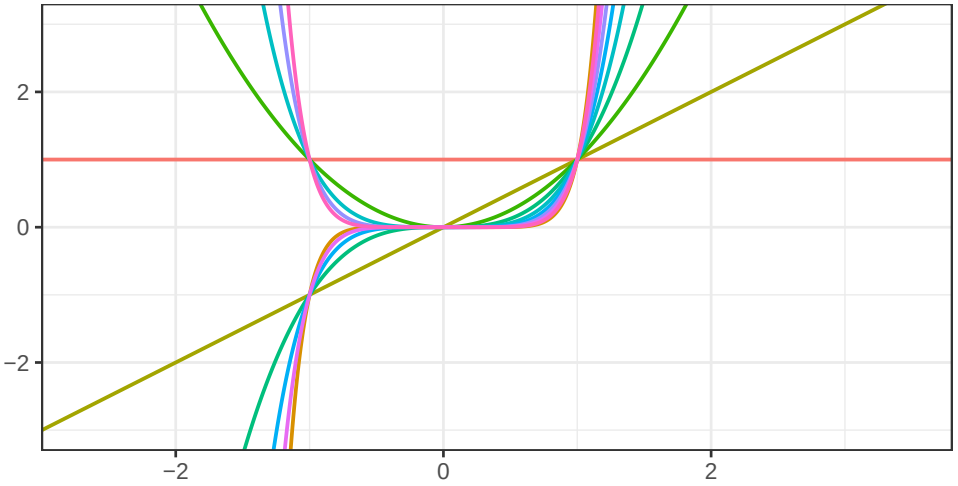
Key point

We can view f_θ as a stochastic process comprised of basis functions which can be used as the **prior for non-linear functions**:

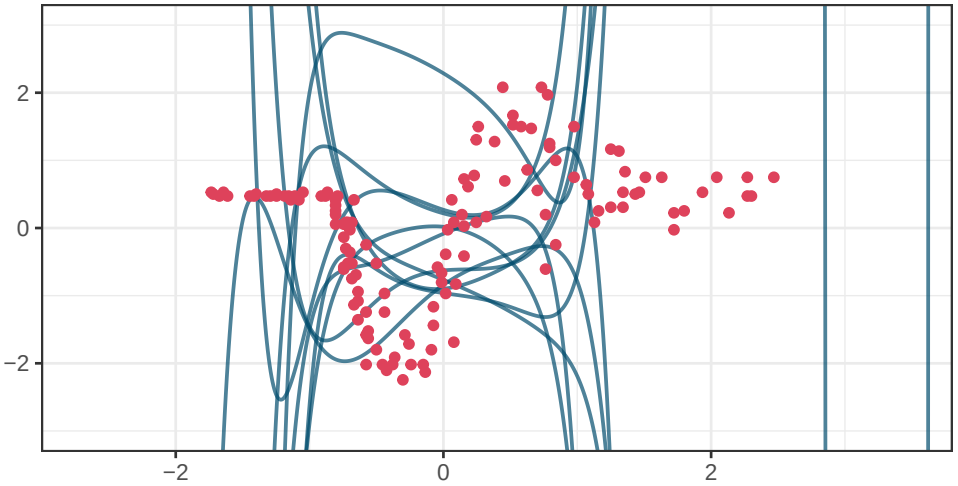
$$f_\theta \sim \mathcal{PP}(\mu, \sigma).$$

Note: $\mathcal{PP}$ stands for Polynomial Process. Not a conventional notation.

Basis functions



Prior samples



Bayesian polynomial regression

Bayesian inference

We apply the Bayes rule

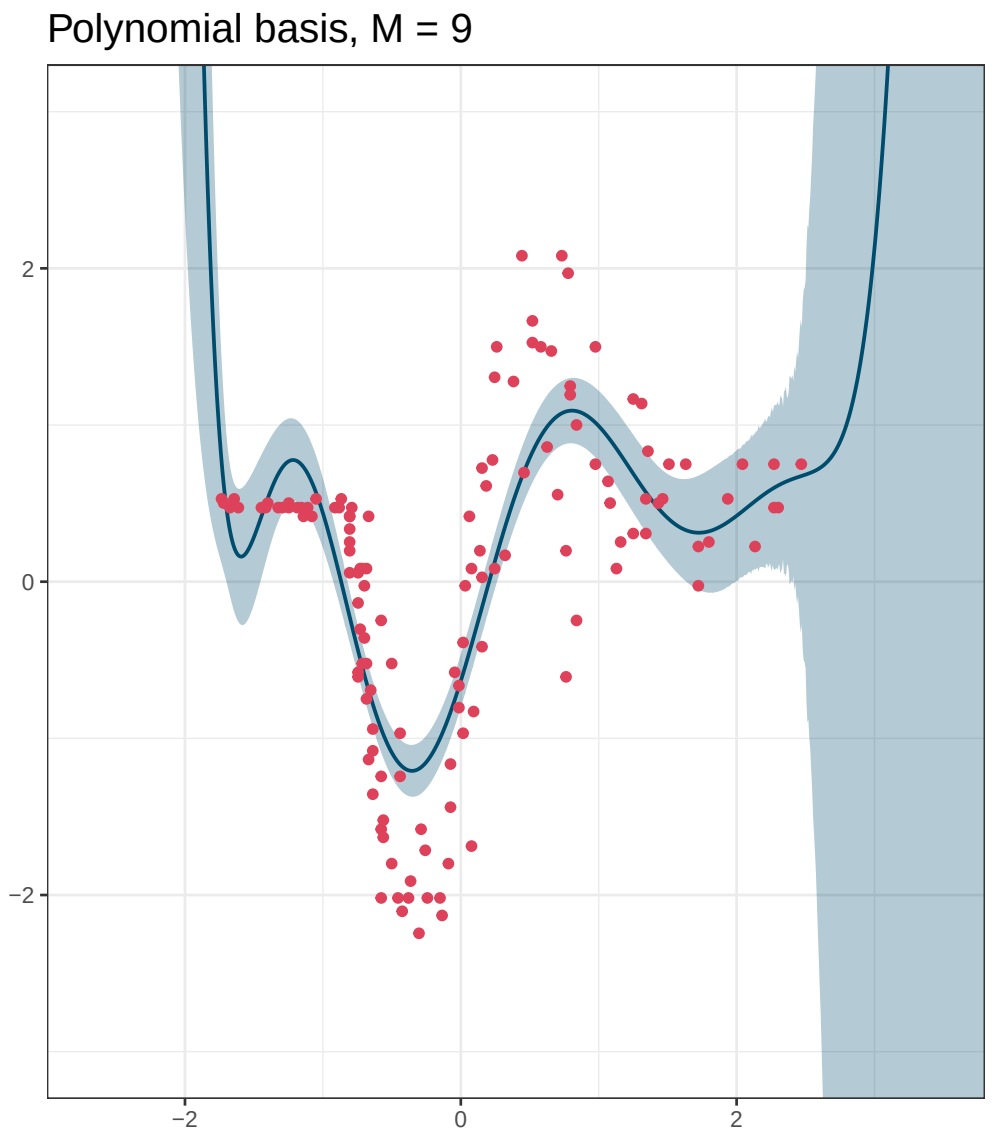
$$p(\theta|\boldsymbol{y}) \propto p(\boldsymbol{y}|\theta)p(\theta)$$

where $\theta = (\beta_0, \beta_1, \ldots, \beta_M)$ to remove curves that does not fit the observed data.

Issues

Polynomial regression is flexible enough to most non-linear functions but...

- Prone to over-fitting
- Extremely unrealistic predictions when extrapolating



Bayesian polynomial regression

Overfitting

As we increase the number of basis functions M, the better the fit to the training data. This leads to good training accuracy but terrible testing accuracy *a.k.a.*, over-fitting.

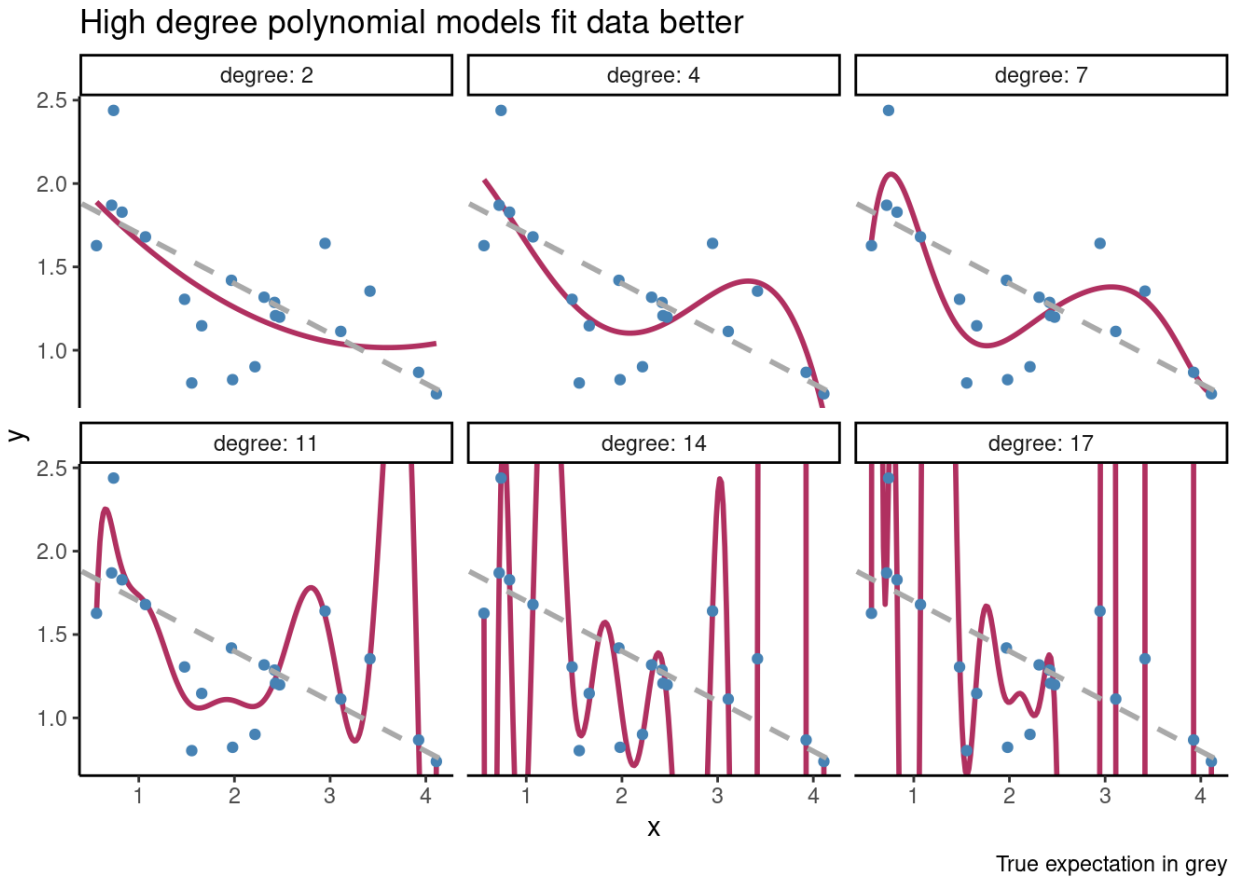


Figure by Alex Hayes. https://www.alexpghayes.com/post/2020-01-06_overfitting-a-guide-tour/

Other basis functions

There are other options

There are many alternatives to polynomials. 2 common examples are:

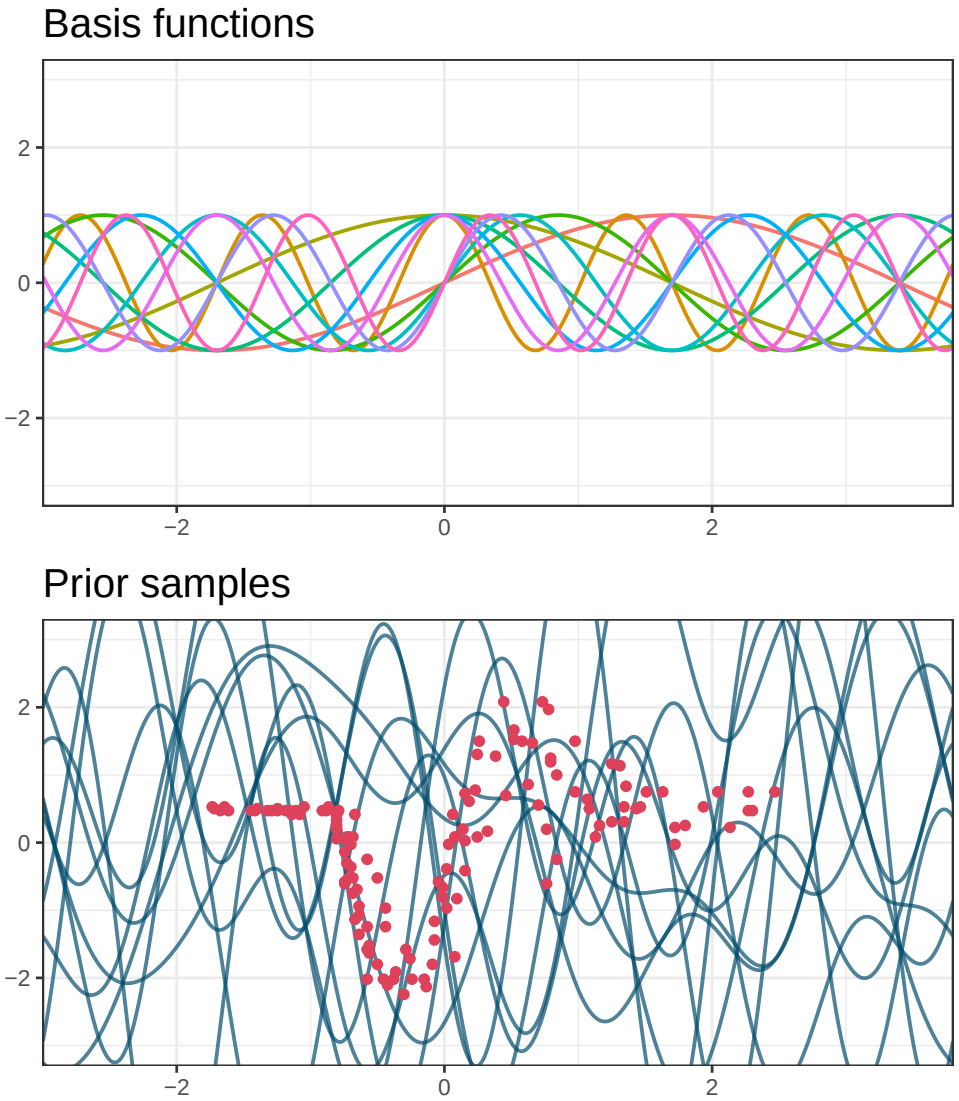
- Fourier basis
- B-Spline basis

Using different kinds of basis can have different effects on the estimates and the uncertainty.

The figure shows the basis functions and prior samples when using Fourier bases,

$$\sum_{m=1}^{M} (\alpha_m \sin(2\pi m x/L) + \beta_m \cos(2\pi m x/L))$$

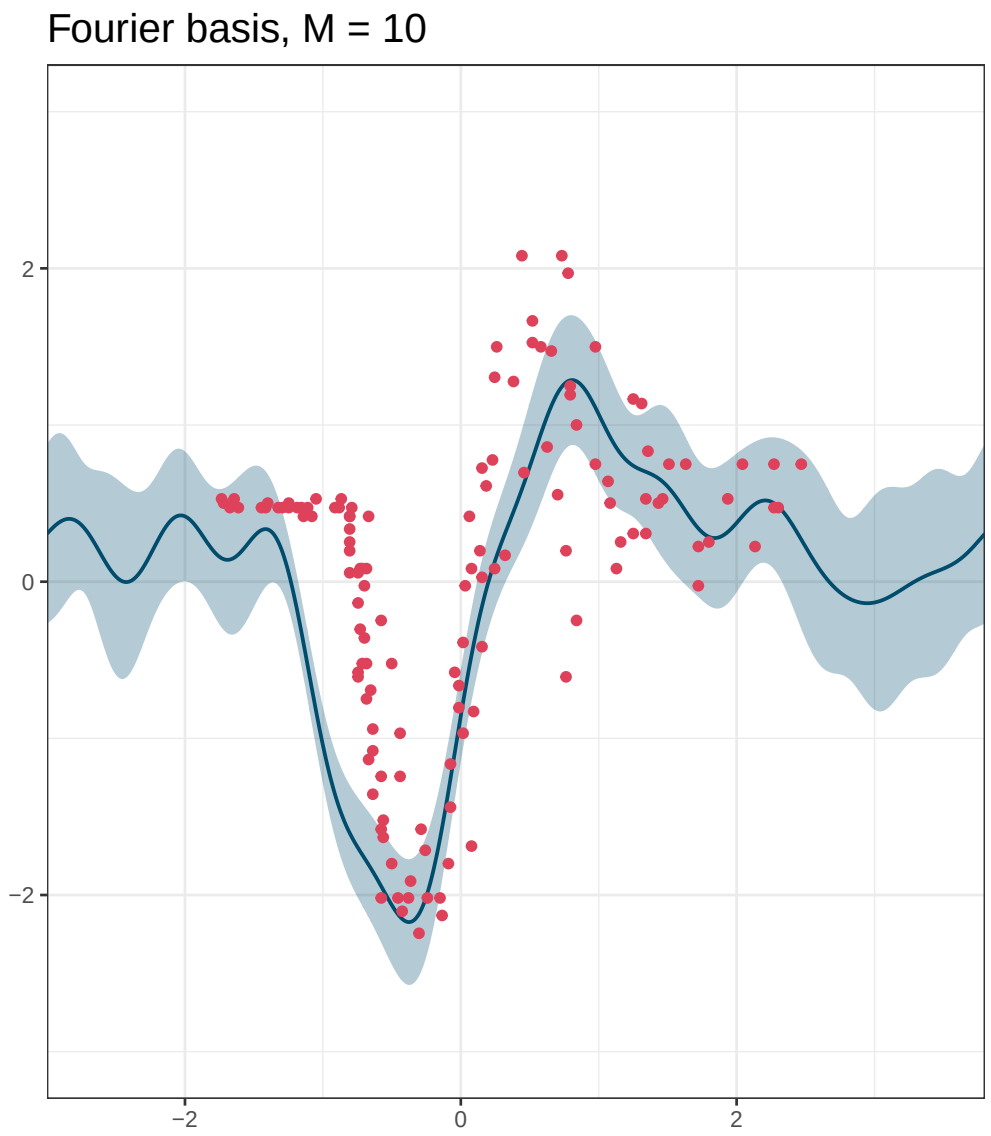
where L is the length of a period.



Other basis functions

An example: Fourier basis

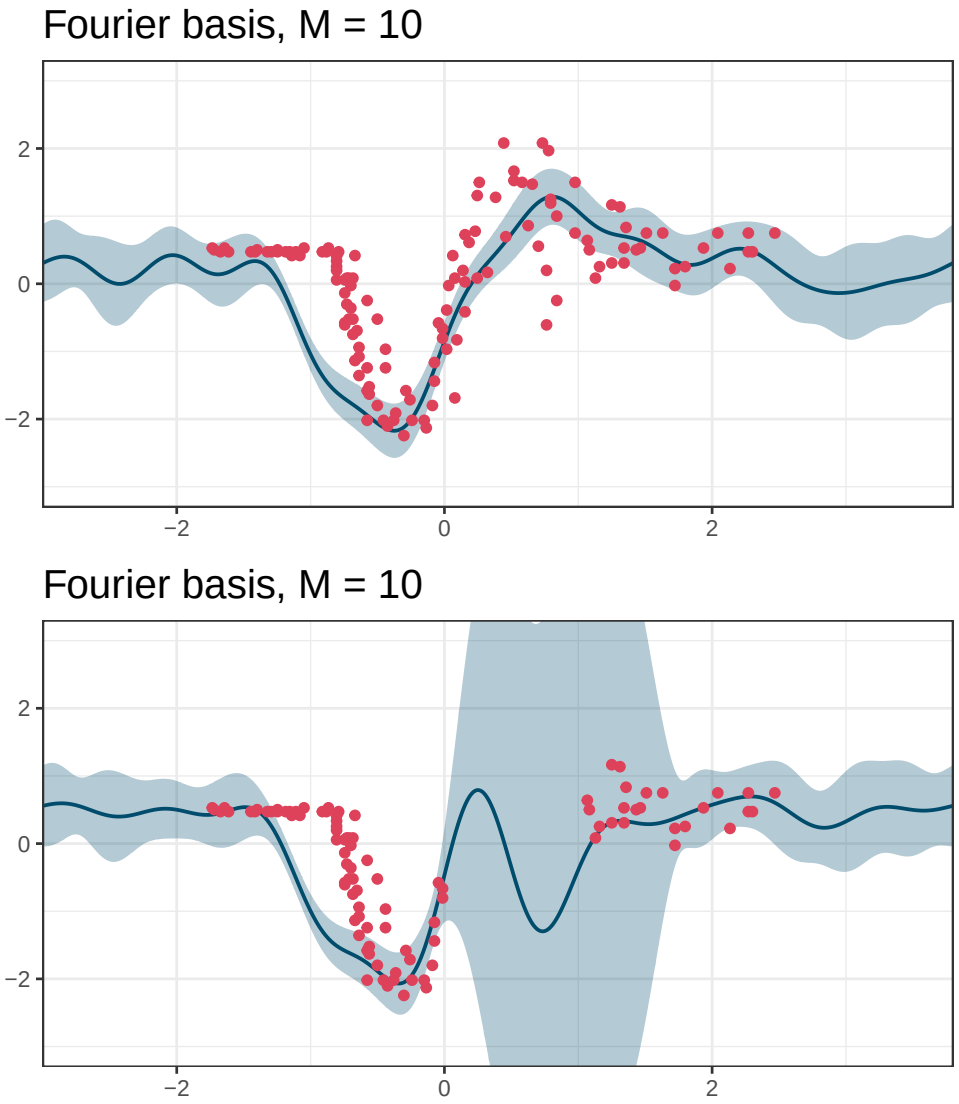
Fourier bases may be more appealing than polynomial bases because they are bounded in output *i.e.*, $|\phi(x)| < \infty$. This **prevents extreme output values**.



Other basis functions

Non-local update of uncertainty and induced bias

When new data is added to or removed from the training set, the **posterior estimates and uncertainty change non-locally**, even though we only acquired / lost data in a specific region. This is because the functions themselves are non local.



Squared Exponential basis functions

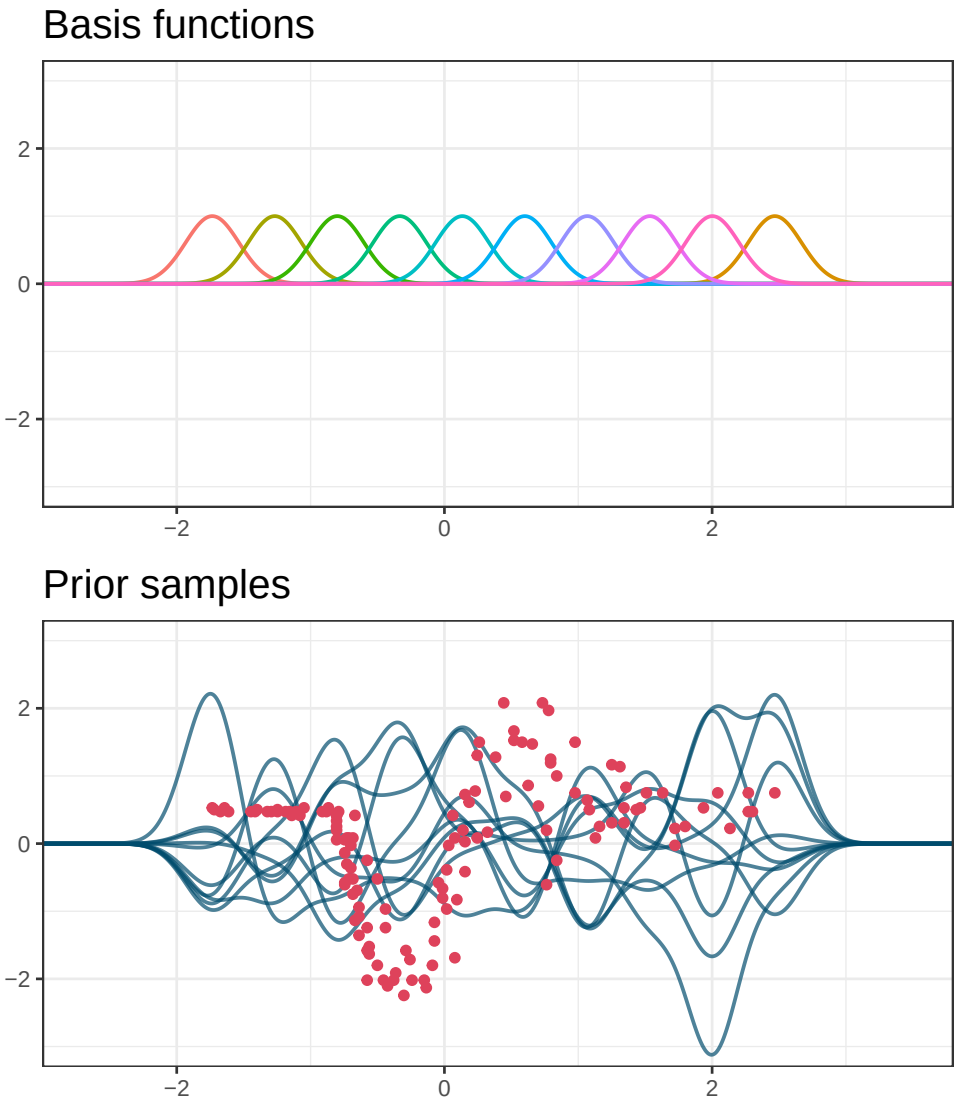
Extending the basis function idea

We would like our basis function to satisfy the following conditions:

- To prevent wild extrapolation, we chose basis functions $\phi(x)$ bounded in output value *i.e.*, $|\phi(x)| < \infty$.
- To prevent sensitivity on distant values, we chose a basis with a bounded input range.

One choice that satisfies the conditions above is the Squared Exponential basis function.

$$\phi_c(x) = \exp\left(-(x-c)^2\right)$$



Squared Exponential basis functions

Still not quite there...

The good

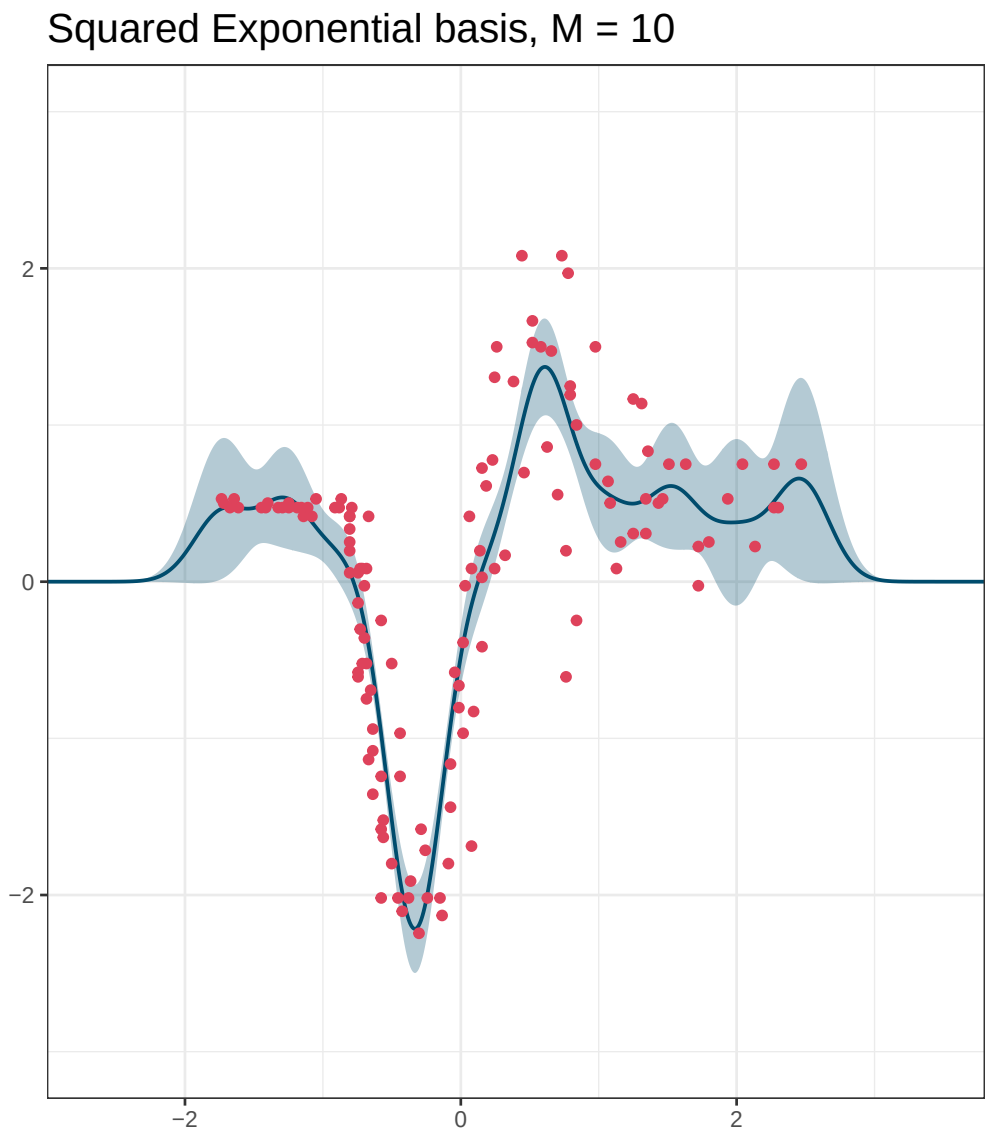
- More sensible posterior
- Interpolation is much better

The bad

- The model is super certain that nothing happens outside of certain range
- No good justification for the choice of where to place the basis functions

Key point

Using the SE basis leads to a much more sensible posterior but it is still not realistic especially for extrapolation. **What if we placed the SE basis function everywhere?**



Computing the Posterior

The Gaussian case

For models with Gaussian noise, computing the posterior is a straightforward procedure that involves finding the joint distribution between the outcome and parameters, and then applying the **Gaussian conditioning rule**. Let us define the following:

$$\boldsymbol{\beta} \sim \mathcal{N}(0, \boldsymbol{I}_M), \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \boldsymbol{I}_N \sigma_\varepsilon^2), \quad [\boldsymbol{\Phi}(\boldsymbol{x})]_{nm} = \phi_m(x_n)$$

$$\boldsymbol{\beta} \in \mathbb{R}^M, \quad \boldsymbol{y} \in \mathbb{R}^N, \quad \boldsymbol{\varepsilon} \in \mathbb{R}^N, \quad \boldsymbol{\Phi}(\boldsymbol{x}) \in \mathbb{R}^{N \times M}, \quad \boldsymbol{x} \in \mathbb{R}^N$$

The Gaussian conditioning rule

Finding the conditional distribution

The Gaussian conditioning rule is as follows. When $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ are random vectors that follow a multivariate normal distribution, *i.e.*

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}), \quad \boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22})$$

then the joint distribution can be written as

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} \right)$$

and the conditional distribution of $\boldsymbol{x}_2$ given $\boldsymbol{x}_1$ is

$$\boldsymbol{x}_2 | \boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1), \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}).$$

Key point

The conditional expectation and covariance of $\boldsymbol{x}_2$ given $\boldsymbol{x}_1$ is

$$\begin{aligned} \mathbb{E}[\boldsymbol{x}_2 | \boldsymbol{x}_1] &= \boldsymbol{\mu} + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1) \\ \mathrm{Cov}[\boldsymbol{x}_2 | \boldsymbol{x}_1] &= \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12} \end{aligned}$$

The conditionals for $\boldsymbol{x}_1$ given $\boldsymbol{x}_2$ can be obtained by simple reordering.

Computing the Posterior

Using the Gaussian conditioning rule, we can obtain the following. The joint distribution is,

$$p\left(\begin{bmatrix}\boldsymbol{\beta}\\ \boldsymbol{y}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\beta}\\ \boldsymbol{y}\end{bmatrix} \mid 0, \begin{bmatrix}\boldsymbol{I}_M & \boldsymbol{\Phi}(\boldsymbol{x})^\top \\ \boldsymbol{\Phi}(\boldsymbol{x}) & \boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2 \boldsymbol{I}_N\end{bmatrix}\right).$$

The posterior distribution of $\boldsymbol{\beta}$ works out to be,

$$p(\boldsymbol{\beta} \mid \boldsymbol{y}) = \mathcal{N}(\boldsymbol{\beta} \mid \boldsymbol{\Phi}(\boldsymbol{x})^\top[\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1}\boldsymbol{y}, \\ \boldsymbol{I}_M - \boldsymbol{\Phi}(\boldsymbol{x})^\top[\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1}\boldsymbol{\Phi}(\boldsymbol{x}))$$

Key point

The posterior expectation and covariance matrix of $\boldsymbol{\beta}$ is

$$\mathbb{E}[\boldsymbol{\beta}|\boldsymbol{y}] = \boldsymbol{\Phi}(\boldsymbol{x})^\top[\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1}\boldsymbol{y}$$
$$\mathrm{Cov}[\boldsymbol{\beta}|\boldsymbol{y}] = \boldsymbol{I}_M - \boldsymbol{\Phi}(\boldsymbol{x})^\top[\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1}\boldsymbol{\Phi}(\boldsymbol{x})$$

Computational cost

Whats the cost of computing the posterior

$$p(\boldsymbol{\beta}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\beta} \mid \boldsymbol{\Phi}(\boldsymbol{x})^\top[\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2\boldsymbol{I}_N]^{-1}\boldsymbol{y}, \\ \boldsymbol{I}_M - \boldsymbol{\Phi}(\boldsymbol{x})^\top[\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2\boldsymbol{I}_N]^{-1}\boldsymbol{\Phi}(\boldsymbol{x}))$$

If we assume the cost of simple linear algebra algorithms, the computational cost for each component of the posterior is as follows:

- $\boldsymbol{\Phi}(\boldsymbol{x})$: $\mathcal{O}(NM)$ Compute M basis functions for N data points
- $\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top$: $\mathcal{O}(N^2M)$: Matrix multiplication
- $[\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2\boldsymbol{I}_N]^{-1}$: $\mathcal{O}(N^3)$ Matrix inversion or Cholesky decomposition

Key point

In total, the computational cost of computing the posterior is $\mathcal{O}(NM + N^2M + N^3)$

Posterior predictive distribution

Regardless of whether the objective is inference or prediction, in almost every case the quantity of interest is not the parameters but the posterior predictive $p(y_*|\boldsymbol{y})$.

We can find the posterior predictive distribution without finding the posterior of the parameters using the fact that

$$p(\boldsymbol{y}, y_*) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{y}\\ y_*\end{bmatrix} \,\middle|\, 0, \begin{bmatrix}\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2 \boldsymbol{I}_N & \boldsymbol{\phi}(x_*) \\ \boldsymbol{\phi}(x_*)^\top \boldsymbol{\Phi}(\boldsymbol{x})^\top & \boldsymbol{\phi}(x_*)^\top \boldsymbol{\phi}(x_*) + \sigma_\varepsilon^2\end{bmatrix}\right).$$

where $y_* \in \mathbb{R}$, $\boldsymbol{\phi}(x_*) \in \mathbb{R}^M$. Using the Gaussian conditioning rule we can derive,

$$\begin{aligned} p(y_*|\boldsymbol{y}) = \mathcal{N}(y_* \mid & \boldsymbol{\phi}(x_*)^\top \boldsymbol{\Phi}(\boldsymbol{x})^\top [\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1} \boldsymbol{y}, \\ & \boldsymbol{\phi}(x_*)^\top \boldsymbol{\phi}(x_*) + \sigma_\varepsilon^2 - \boldsymbol{\phi}(x_*)\boldsymbol{\Phi}(\boldsymbol{X})^\top [\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1} \boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\phi}(x_*)). \end{aligned}$$

The cost of computing the posterior predictive in this way is

$$\mathcal{O}(N^3 + N^2 M + NM)$$

Infinite basis functions

Recall that we would like to place basis functions everywhere *i.e.*, have infinitely many basis functions. However it is impossible to compute the posterior predictive when $M \to \infty$ as the computational cost will also be infinite.

But notice that the components we need are

$$\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top \in \mathbb{R}^{N\times N}, \quad \boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\phi}(x_*) \in \mathbb{R}^{N\times 1}$$

which means we only need the **inner products** between feature vectors:

$$[\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top]_{ij} = \boldsymbol{\phi}(x_i)^\top \boldsymbol{\phi}(x_j)$$

What if we could compute the inner products directly without computing the basis functions? This is where the famous **kernel trick** comes into play.

Kernel trick

An example: Polynomial kernel

If we can compute the matrices $\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\Phi}(\boldsymbol{x})^\top \in \mathbb{R}^{N\times N}$ and $\boldsymbol{\Phi}(\boldsymbol{x})\boldsymbol{\phi}(x_*) \in \mathbb{R}^{N\times 1}$ directly, we could do computations without incurring cost for a large number of basis functions.

For example the Polynomial kernel is

$$k(x,y) = (xy+1)^{M-1} = \sum_{m=0}^{M} \binom{M-1}{m} x^m y^m = \boldsymbol{\phi}(x)^\top \boldsymbol{\phi}(y)$$

where $\boldsymbol{\phi}(x) = (1, \sqrt{2}x, x^2)^\top$ if $M = 3$.

Kernel trick

Infinite dimensional feature spaces

If the limit of the inner product exist, we can even consider infinite dimensional feature spaces.

$$\phi_m(x) = \exp\left(-\frac{(x - c_m)^2}{2\ell^2}\right), \quad c_m = \frac{m}{M}(c_{\max} - c_{\min})$$

$$k(x, x') = \frac{1}{M}\sum_{p=1}^{M} \phi_m(x)\phi_m(x')$$

$$\begin{aligned}\lim_{M\to\infty} \frac{1}{M}\sum_{m=1}^{M} \phi_m(x)\phi_m(x') &= \int_{c_{\min}}^{c_{\max}} \exp\left(-\frac{(x - c)^2}{2\ell^2}\right) \exp\left(-\frac{(x - c)^2}{2\ell^2}\right) dc \\ &= \sqrt{\pi}\ell \exp\left(-\frac{(x - x')^2}{4\ell^2}\right)\end{aligned}$$

which is called the Squared Exponential (SE) kernel and it is equivalent to placing SE basis functions everywhere.

Posterior predictions

Posterior predictive distribution of a Gaussian process

To obtain posterior predictions, all we need to do is to replace the inner products $\boldsymbol{\phi}(x)^\top \boldsymbol{\phi}(x')$ with $k(x, x')$.

$$p(y_*|\boldsymbol{y}) = \mathcal{N}(y_* \mid k(x_*, \boldsymbol{x})[k(\boldsymbol{x}, \boldsymbol{x}) + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1}\boldsymbol{y}, \\ k(x_*, x_*) + \sigma_\varepsilon^2 - k(x_*, \boldsymbol{x})[k(\boldsymbol{x}, \boldsymbol{x}) + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1} k(\boldsymbol{x}, x_*)).$$

Now the cost is $\mathcal{O}(N^3 + N^2) = \mathcal{O}(N^3)$

The definition of a Gaussian process

Gaussian process can be thought of as a probability distribution for functions. In simpler terms, it is a box from which you can draw random samples of functions.

Definition

A Gaussian process is a infinite set of random variables, any finite subset of which follows a multivariate normal distribution.

The definition of a Gaussian process

Some standard notation and terminology

Let $x, x' \in \mathbb{R}$ be two inputs. The standard notation for a GP is

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')).$$

where,

$$\begin{aligned} m(x) &= \mathbb{E}[f(x)] \\ k(x, x') &= \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] \end{aligned}$$

are called the **mean function** and **covariance kernel**, respectively.

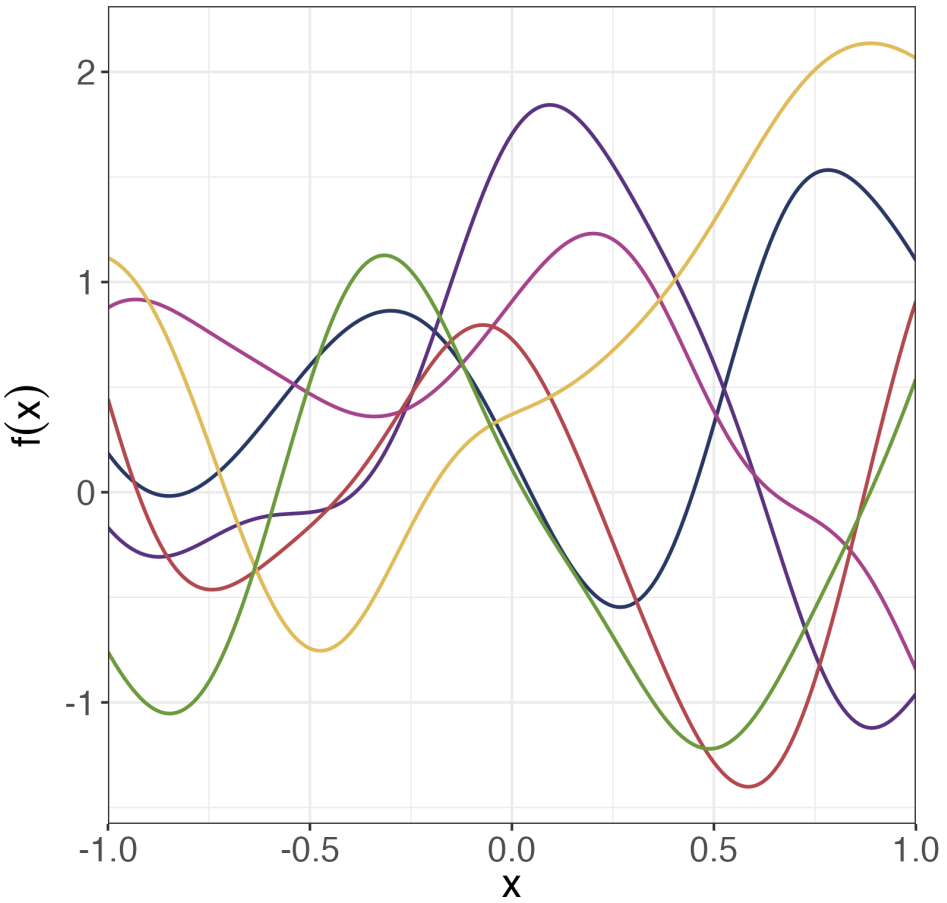
We set $m(x) = 0$ when we don't have any prior knowledge about the mean function, giving us

$$f(x) \sim \mathcal{GP}(0, k(x, x')).$$

Key point

GPs can be though of as **a prior over continuous functions**.

Samples from a GP prior



The definition of a Gaussian process

Looking under the hood

The definition and notation does not help us understand how GPs are constructed. At the end of the day GPs are simply multivariate Gaussians. Let $\boldsymbol{x} = (x_1, \ldots, x_N)^\top$ be a vector of inputs. Then,

$$f(\boldsymbol{x}) \sim \mathcal{GP}(\mathbf{0}, k(\boldsymbol{x}, \boldsymbol{x}')) \quad \Rightarrow \quad f(\boldsymbol{x}) \sim \mathcal{N}(\mathbf{0}, K).$$

where the covariance matrix K is,

$$K = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \ldots & k(x_1, x_N) \\ k(x_2, x_1) & k(x_2, x_2) & \ldots & k(x_2, x_N) \\ \vdots & & \ddots & \vdots \\ k(x_N, x_1) & k(x_N, x_2) & \ldots & k(x_N, x_N) \end{pmatrix}.$$

The function $k(\cdot, \cdot)$ that generates the elements of K is called the **covariance kernel** or **kernel function**. It encodes the strength of association between pairs of inputs.

Key point

Under the hood, a multivariate Gaussian with a special covariance matrix generated by a covariance kernel.

The covariance kernel

An example: Squared Exponential kernel

One of the most commonly used covariance kernel in machine learning is the Squared Exponential(SE) kernel.

Squared exponential kernel

$$k_{SE}(x, x') = \alpha \exp\left(-\frac{(x - x')^2}{2\ell^2}\right)$$

where

- α: Scale factor that dictates how far the function values can be from the mean
- ℓ: Referred to as the lengthscale and determines how "wiggly" the function is

These parameters are often unknown and are estimated during training.

If we fix α and ℓ, we see that as the distance between x and x' increase, $k(x, x')$ approaches 0, which imply that there are no associations between the two input points.

Key point

The covariance kernel determines the strength of relationship between pairs of input points.

Various covariance kernels

How do we modeling different kinds of functions

The kernel function characterise the behavior of the GP. There are many types of covariance kernels and they are chosen according to our beliefs about the kind of function we are modeling. Let $r = |x - x'|$. The following table is contains examples of different covariance kernels and the types of functions they can model

Name	Definition	Type of functions
Squared Exponential	$\alpha \exp\left(-\frac{r^2}{2\ell^2}\right)$	Infinitely differentiable functions
Matérn 3/2	$\alpha\left(1+\frac{\sqrt{3}r}{\ell}\right)\exp\left(-\frac{\sqrt{3}r}{\ell}\right)$	1 time differentiable functions
Matérn 5/2	$\alpha\left(1+\frac{\sqrt{5}r}{\ell}+\frac{5r^2}{3\ell^2}\right)\exp\left(-\frac{\sqrt{5}r}{\ell}\right)$	2 time differentiable functions
Linear Kernel	$\sigma_b^2 + \sigma_v^2(x-c)(x'-c)$	Linear function
Periodic Kernel	$\alpha \exp\left(-\frac{2\sin^2(\pi r/p)}{\ell^2}\right)$	Periodic functions
Locally Periodic Kernel	$\alpha \exp\left(-\frac{2\sin^2(\pi r/p)}{\ell^2}\right)\exp\left(-\frac{r}{2\ell^2}\right)$	Functions that are periodic at certain locations

https://www.cs.toronto.edu/~duvenaud/cookbook/

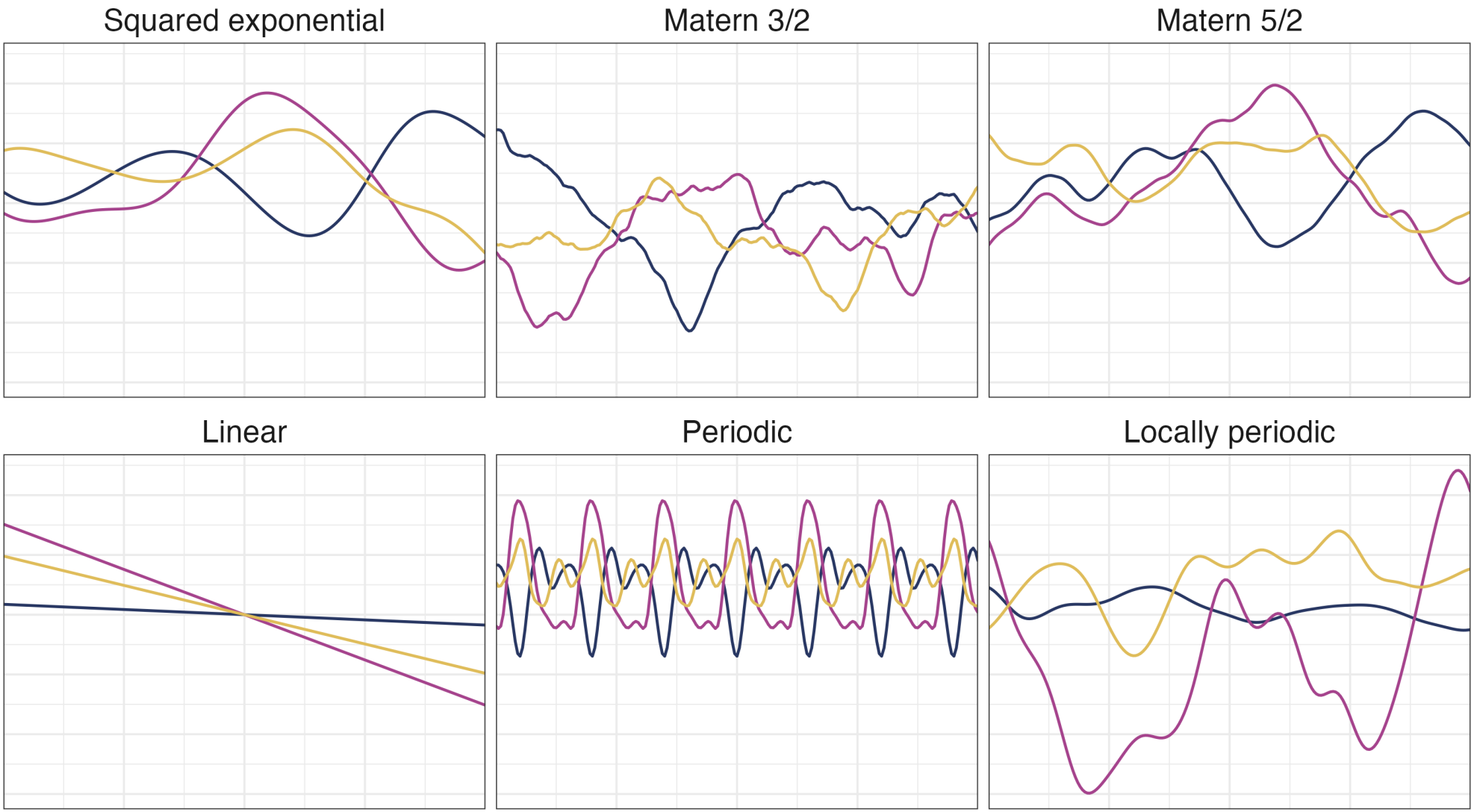
Key point

The covariance kernel determines the type of functions the GP can model. We can create new kernels by adding or multiplying them together.

Various covariance kernels

Examples of draws from the GP prior

A comparison between functions sampled from GPs with different covariance kernels



Estimating a Gaussian processes

GPs: Posterior predictive inference

Some basic notations

Assume we have N training points and N_* test points.
We define the following:

Vector of training points

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_N)^\top$$

Vector of test points

$$\boldsymbol{x}_* = (x_{1*}, x_{2*}, \ldots, x_{N*})^\top$$

Values of f at inputs

$$\boldsymbol{f} := f(\boldsymbol{x}) = (f(x_1), f(x_2), \ldots, f(x_N))^\top,$$
$$\boldsymbol{f}_* := f(\boldsymbol{x}_*) = (f(x_{1*}), f(x_{2*}), \ldots, f(x_{N*}))^\top$$

Covariance matrix of the training points

$$K = \begin{pmatrix} k(x_1, x_1) & \ldots & k(x_1, x_N) \\ \vdots & \ddots & \vdots \\ k(x_N, x_1) & \ldots & k(x_N, x_N) \end{pmatrix}$$

Cov. matrix of training and test points

$$K_* = \begin{pmatrix} k(x_1, x_{1*}) & \ldots & k(x_1, x_{N*}) \\ \vdots & \ddots & \vdots \\ k(x_N, x_{1*}) & \ldots & k(x_n, x_{N*}) \end{pmatrix}$$

Covariance matrix of the test points

$$K_{**} = \begin{pmatrix} k(x_{1*}, x_{1*}) & \ldots & k(x_{1*}, x_{N*}) \\ \vdots & \ddots & \vdots \\ k(x_{N*}, x_{1*}) & \ldots & k(x_{N*}, x_{N*}) \end{pmatrix}$$

The Gaussian conditioning rule

Recap from the previous chapter

The Gaussian conditioning rule is as follows. When $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ are random vectors that follow a multivariate normal distribution, *i.e.*

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}), \quad \boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22})$$

then the joint distribution can be written as

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} \right)$$

and the conditional distribution of $\boldsymbol{x}_2$ given $\boldsymbol{x}_1$ is

$$\boldsymbol{x}_2 | \boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1), \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}).$$

Key point

The conditional expectation and variance of $\boldsymbol{x}_2$ given $\boldsymbol{x}_1$ is

$$\mathbb{E}[\boldsymbol{x}_2 | \boldsymbol{x}_1] = \boldsymbol{\mu} + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\boldsymbol{x}_1 - \boldsymbol{\mu}_1)$$
$$V[\boldsymbol{x}_2 | \boldsymbol{x}_1] = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}$$

The conditionals for $\boldsymbol{x}_1$ given $\boldsymbol{x}_2$ can be obtained by simple reordering.

GPs: Posterior predictive

Analytically deriving the posterior predictive distribution

Assuming Gaussian noise, the joint distribution of $\boldsymbol{f}$ and $\boldsymbol{f}_*$ can be written as:

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K + \sigma_\varepsilon^2 \boldsymbol{I}_N & K_* \\ K_*^\top & K_{**} \end{bmatrix}\right).$$

A simple application of the Gaussian conditioning rule gives us

$$\boldsymbol{f}_* | \boldsymbol{f} \sim \mathcal{N}(K_*^\top [K + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1} \boldsymbol{f}, K_{**} - K_*^\top [K + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1} K_*).$$

Key point

Given training data $\boldsymbol{f}$, the expectation and covariance of $\boldsymbol{f}_*$ are:

$$\mathbb{E}[\boldsymbol{f}_* | \boldsymbol{f}] = K_*^\top [K + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1} \boldsymbol{f}$$
$$\mathrm{Cov}[\boldsymbol{f}_* | \boldsymbol{f}] = K_{**} - K_*^\top [K + \sigma_\varepsilon^2 \boldsymbol{I}_N]^{-1} K_*$$

Approximate posterior inference in Stan

GPs in off-the-shelf Bayesian libraries

In `Stan`, we do not need to derive the posterior analytically. As an example, consider the following model:

$$Y \sim \mathcal{N}(\mu(x), \sigma_\varepsilon^2) \quad \mu(x) = \beta_0 + f(x).$$

to which we assign the following priors:

$$\begin{aligned} \sigma_\varepsilon^2 &\sim \text{InvGamma}(5, 5) \\ \beta_0 &\sim \mathcal{N}(0, 1) \\ f(x) &\sim \mathcal{GP}(0, K) \\ \alpha &\sim \text{InvGamma}(5, 1) \\ \ell &\sim \text{InvGamma}(5, 1) \end{aligned}$$

The ingredients we need are

1. Log density functions of the priors $\log p(\sigma_\varepsilon^2), \log p(\beta_0), \ldots$
2. Log likelihood function $\log p(y|\sigma_\varepsilon^2, \beta_0, f(x), \alpha, \ell) = \log \mathcal{N}(\beta_0 + f(x), \sigma_\varepsilon^2)$

Key point

In Stan, once we define the log-likelihood and priors, a sophisticated MCMC algorithm will take care of the rest.

Approximate posterior inference in Stan

What do we need to evaluate the log-likelihood

Assuming Gaussian noise, we need to evaluate

$$\log p(y|\sigma_\varepsilon^2, \beta_0, f(x), \alpha, \ell) = \log \mathcal{N}(\mu(x) = \beta_0 + f(x), \sigma_\varepsilon^2)$$

in each MCMC iteration. To do this we need realisations of β_0 and $f(x)$.

Sampling β_0 is easy. I will shift our attention to sampling from $f(x)$, *i.e.*, sampling from a GP.

Key point

To evaluate $\mu(x) = \beta_0 + f(x)$ we need samples for β_0 and $f(x)$.

Positive definite matrix

A review of basic concepts in linear algebra

Positive definite matrix

When a symmetric $N \times N$ matrix A satisfy the following conditions, A is called a **positive definite matrix**.

$$\boldsymbol{z}^\top A \boldsymbol{z} = \sum_{i=1}^{N} \sum_{j=1}^{N} z_i z_j A_{ij} > 0$$

here $\boldsymbol{z} \in \mathbb{R}^N$ is any non zero vector.

Commonly used kernels such as the SE kernel and Matérn family of kernels are **Positive definite kernels**, which generate positive definite matrices.

Cholesky decomposition

A review of basic concepts in linear algebra

Cholesky decomposition

A positive definite matrix can be decomposed in to a lower triangular matrix and its transpose.

$$LL^\top = A$$

Here, L is called the **Cholesky factor**.

Cholesky factors are very numerically stable requiring only $N^3/6$ to compute, hence it is the method of choice when it can be applied.

Covariance matrices and Cholesky decomposition

The multivariate version of standard deviation

The covariance matrices generated by SE and Matérn kernels are positive definite. Thus we can apply Cholesky decomposition.

$$K = LL^\top$$

Key point

In the case of covariance matrices, we can interpret their Cholesky factors to be the multivariate version of the standard deviation.

When $X \in \mathbb{R}$ and $X \sim \mathcal{N}(\mu, \sigma^2)$, X can be expressed as $X = \mu + \sigma Z$ where $Z \sim \mathcal{N}(0, 1)$.

$$\mathbb{E}[X] = \mathbb{E}[\mu + \sigma Z] = \mu, \quad V[X] = V[\mu + \sigma Z] = \sigma^2$$

Similarly, when $\boldsymbol{f} \in \mathbb{R}^N$ and $\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}, K)$, we can write $\boldsymbol{f} = \boldsymbol{\mu} + L\boldsymbol{\beta}$. Here, $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_N)$.

$$\mathbb{E}[\boldsymbol{f}] = \mathbb{E}[\boldsymbol{\mu} + L\boldsymbol{\beta}] = \boldsymbol{\mu}, \quad \mathrm{Cov}[\boldsymbol{f}] = \mathrm{Cov}[\boldsymbol{\mu} + L\boldsymbol{\beta}] = L\boldsymbol{I}_N L^\top = K$$

Sampling from a Gaussian process

A summary of the steps

In summary, the following steps can be taken to sample from a GP.

1. Sample the parameters of the covariance kernel k.
2. Use the kernel to generate a covariance matrix K.
3. Obtain the Cholesky factor L of K.
4. Sample $\boldsymbol{\beta} \in \mathbb{R}^N$ from a standard multivariate normal distribution $\mathcal{N}(0, \boldsymbol{I}_N)$
5. $\boldsymbol{f} = L\boldsymbol{\beta}$ will be sample from a GP with mea n0 and covariance kernel k.

The nugget effect

Some numerical gotchas

In `Stan` and other probabilistic programming languages, **if the distance between two inputs are too close, the covariance matrix may no longer be positive definite numerically**. To resolve this issue, we can add a small value called to the diagonal of the covariance matrix s.t. the matrix is positive definite. This is called the **nugget effect**.

Nugget

If K is a $n \times n$ covariance matrix and $\boldsymbol{I}_N$ is an identity matrix, the covariance matrix with the nugget $\tilde{K}$ is

$$\tilde{K} = K + \boldsymbol{I}_N \epsilon$$

where, ϵ is a "small enough" value (*e.g.*, 1.0×10^{-4}).

Computational cost of Gaussian processes

The computational bottleneck

When we have N data points,

1. the cost of computing a $N \times N$ covariance matrix is $\mathcal{O}(N^2)$.
2. The cost of computing the inverse* or Cholesky is $\mathcal{O}(N^3)$

In total the computational cost is approximately $\mathcal{O}(N^2 + N^3) = \mathcal{O}(N^3)$.

Key point

Inferring GPs with MCMC is feasible up to a few hundred data points. Computations becomes unbearably slow after surpassing $N > 1000$ and thus is not very practical.

* We can use techniques such as the Woodbury matrix identity to reduce computational cost.

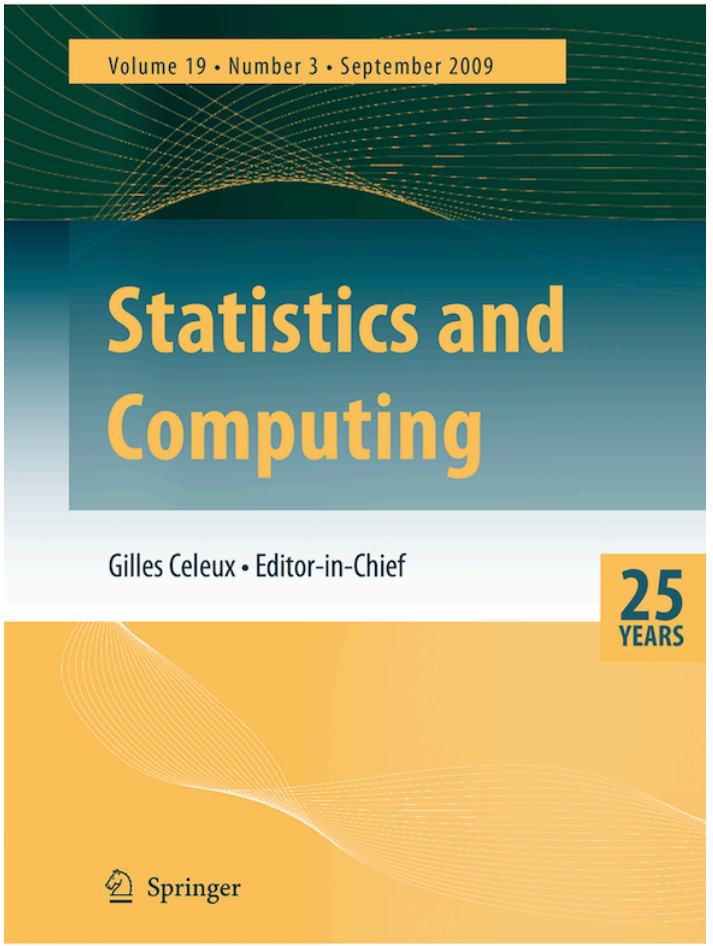
Gaussian process approximation

Gaussian process approximations

How do we reduce the computational cost of GPs

There are many methods for reducing the computational cost of GPs.

I will introduce a method called **Hilbert space approximate Gaussian process** (HSGP) proposed by Solin & Sarkka in 2019 for which Riutort-Mayol et al. wrote a tutorial paper for implementation in probabilistic programming languages.



1. https://link.springer.com/article/10.1007/s11222-019-09886-w
2. https://link.springer.com/article/10.1007/s11222-022-10167-2

Stationary covariance kernels

Invariant to translations

Stationary covariance kernels

Stationary covariance kernels is a function of $\boldsymbol{\tau} = \boldsymbol{x} - \boldsymbol{x}' \in \mathbb{R}^D$ *i.e.*, and can be written as,

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{\tau}).$$

Stationary covariance kernels are invariant to translations in the input space.

Spectral density functions

For every stationary covariance kernel, there is a spectral density

Every stationary covariance kernel has a corresponding spectral density function. For instance, the D-dimensional Matérn class covariance kernel has the following spectral density function

$$S_\nu(\omega) = \alpha \frac{2^D \pi^{D/2} \Gamma(\nu + D/2)(2\nu)^\nu}{\Gamma(\nu)\ell^{2\nu}} \left(\frac{2\nu}{\ell^2} + 4\pi^2 \omega^\top \omega \right)^{-(\nu + D/2)}$$

Here, $\omega \in \mathbb{R}^D$ is a vector in the frequency domain.

1-dimensional Matérn class covariance kernels and respective spectral densities

Name	Expression	Spectral density
Squared exponential	$\alpha^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$	$S_\infty(\omega) = \alpha\sqrt{2\pi}\ell \exp(-\frac{1}{2}\ell^2\omega^2)$
Matérn 3/2	$\alpha\left(1 + \frac{\sqrt{3}r}{\ell}\right)\exp\left(-\frac{\sqrt{3}r}{\ell}\right)$	$S_{3/2}(\omega) = 4\alpha\frac{3^{3/2}}{\ell^2}\left(\frac{3}{\ell^2} + \omega^2\right)^{-2}$
Matérn 5/2	$\alpha\left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right)\exp\left(-\frac{\sqrt{5}r}{\ell}\right)$	$S_{5/2}(\omega) = 32\alpha\frac{5^{5/2}}{3\ell^5}\left(\frac{5}{\ell^2} + \omega^2\right)^{-3}$

Representing kernels with spectral density functions

Spectral density + eigenvalues + eigenvectors

By using **Hilbert Space** methods,

Expressing stationary kernels using spectral density functions

In a compact range $\Omega = [-L, L] \subset \mathbb{R}$, stationary kernels can be written as the following infinite sum:

$$k(x, x') = \sum_{m=1}^{\infty} S_\theta(\sqrt{\lambda_m})\phi_m(x)\phi_m(x')$$

where, S_θ is the spectral density, and λ_m, $\phi_m(x)$ are given as,

$$\lambda_m = \left(\frac{m\pi}{2L}\right)^2, \quad \phi_m(x) = \sqrt{\frac{1}{L}} \sin\left(\sqrt{\lambda_m}(x + L)\right)$$

respectively. Note that the eigenvalues and eigenfunctions do not depend on the spectral density.

Approximating the kernel

Removing the high finer details

Notice that the eigenfunction $\phi_m(x)$ is a periodic function which increases its frequency with m. Most information about the kernel is contained within the low frequency components. Thus we may truncate the infinite sum

$$k(x, x') = \sum_{m=1}^{\infty} S_\theta(\sqrt{\lambda_m})\phi_m(x)\phi_m(x')$$

to the first m terms, and approximate the kernel as

$$k(x, x') \approx \sum_{m=1}^{M} S_\theta(\sqrt{\lambda_m})\phi_m(x)\phi_m(x')$$

Key point

Covariance kernels can be approximated using the spectral density and the first m terms of the infinite sum.

Gaussian process approximations

Rewriting in matrix notation

Rewriting the approximation using matrix notation, we obtain

Approximation of the covariance kernel

$$k(x, x') \approx \sum_{m=1}^{M} S_\theta(\sqrt{\lambda_m})\phi_m(x)\phi_m(x') = \boldsymbol{\phi}(x)^\top \Delta \boldsymbol{\phi}(x')$$

where $\boldsymbol{\phi}(x) = \{\phi_m(x)\}_{m=1}^m \in \mathbb{R}^M$ is a column vector of eigenfunction values and $\Delta \in \mathbb{R}^{M \times M}$ is a diagonal matrix consisting of spectral densities evaluated at the square root of the eigenvalues.

$$\Delta = \begin{bmatrix} S_\theta(\sqrt{\lambda_1}) & & \\ & \ddots & \\ & & S_\theta(\sqrt{\lambda_m}) \end{bmatrix}$$

Gaussian process approximation

The covariance matrix

When using this approximation, the covariance matrix becomes

$$K \approx \Phi \Delta \Phi^\top.$$

Here, $\Phi \in \mathbb{R}^{N \times M}$ is a matrix of eigenfunctions.

$$\Phi = \begin{bmatrix} \phi_1(x_1) & \dots & \phi_M(x_1) \\ \vdots & \ddots & \vdots \\ \phi_1(x_N) & \dots & \phi_M(x_N) \end{bmatrix}$$

From this we obtain,

$$\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}, \Phi \Delta \Phi).$$

This is equivalent to,

$$f(x) \approx \sum_{m=1}^{M} (S_\theta(\sqrt{\lambda_m}))^{1/2} \phi_m(x) \beta_m$$

where $\beta_m \sim \mathcal{N}(0, 1)$.

Reduction in the computational cost

How much did we gain

Upon examination of the approximation

$$f(x) \approx \sum_{m=1}^{M} (S_\theta(\sqrt{\lambda_m}))^{1/2} \phi_m(x) \beta_m$$

we notice the following:

1. λ_m and $\phi_m(x)$ does not depend on the parameters of the GP. Thus we only need to compute them once beforehand and reuse them
2. Only the m spectral density $S_\theta(\sqrt{\lambda_m})$ is dependent on the GP parameters

Key point

For each MCMC iteration we need to calculate

1. The value of M spectral densities $S_\theta(\sqrt{\lambda_m})$ and $(\mathcal{O}(M))$,
2. the M term sum of N data points $(\mathcal{O}(MN))$

Hence, the total computational cost works out to be $\mathcal{O}(MN + M)$.

In general $M \ll N$ thus compared to $\mathcal{O}(N^3)$, we significantly reduce the necessary computations.

Boundary condition L and no. of eigenfunctions M

Some hyperparameters

To use HSGP, we need to specify L **which determines the range** $\Omega = [-L, L]$ **for which the approximation is valid and the number of eigenfunctions** M.

Let $\{x_i\}_{i=1}^N$ be some input data with mean 0. If we let $S = \max_i |x_i|$, than for any given i, $x_i \in [-S, S]$. Hence, we can define the boundary condition L as follows:

Boundary condition

$$L = c \times S$$

where, $c \geq 1$ is a proportional scaling factor.

The relationship between c, M, and ℓ

How to set the hyper parameters

Riutort-Mayol et al. (2022) studied the accuracy of the approximation for different values of the proportional scaling factor c, the number of eigenfunctions M, and the lengthscale ℓ for the Squared Exponential, Matérn 3/2, Matérn 5/2 kernels. They provide recommendations for the value of c and m based on empirical results.

Covariance kernel	Conditions for the hyperparameters	
Squared exponential	$M = 1.75\frac{c}{\ell/S}$,	$c \geq 3.2\ell/S$ & $c \geq 1.2$
Matérn 5/2	$M = 2.65\frac{c}{\ell/S}$,	$c \geq 4.1\ell/S$ & $c \geq 1.2$
Matérn 3/2	$M = 3.42\frac{c}{\ell/S}$,	$c \geq 4.5\ell/S$ & $c \geq 1.2$

Key point

Riutort-Mayol et al. (2022) acts as a initial guideline, but because ℓ is unknown, the hyper-parameters may need to be tuned. In general, regardless of the kernel $c \geq 1.2$ in order for the approximation to be accurate.

References and Further reading

Some great resources on GPs

- Carl Edward Rasmussen and Christopher K. I. Williams. **Gaussian Process for Machine Learning**. The MIT Press, 2006.
- Mark van der Wilk's course on **Probilistic Inference**. GitHub repository.
- Arno Solin and Simo Särkkä. **Hilbert space methods for reduced-rank Gaussian process regression**. link.
- Riutort-Mayol et al. **Practical Hilbert space approximate Bayesian Gaussian processes for probabilistic programming**. link.

Imperial College London

MACHINE LEARNING & GLOBAL HEALTH NETWORK

Thank you.

Shozen Dan and Oliver Ratmann
Imperial College London

Feb. 22, 2024