Michaelmas Term 2024

Data, Estimation and Inference Lab Report

J Rosser

1 Introduction

A Gaussian process (GP) is a stochastic process where any finite collection of random variables follows a multivariate normal distribution. For a function $f(x)$ and points $x_1, x_2, x_3, \ldots$, we can define a Gaussian Process as $p(f) = \mathcal{GP}(f; \mu, K)$ if

$$p\left(\begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \end{bmatrix}; \begin{bmatrix} \mu(x_1) \\ \mu(x_2) \\ \mu(x_3) \\ \vdots \end{bmatrix}, \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & k(x_1,x_3) & \cdots \\ k(x_2,x_1) & k(x_2,x_2) & k(x_2,x_3) & \cdots \\ k(x_3,x_1) & k(x_3,x_2) & k(x_3,x_3) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}\right) \tag{1}$$

where $f(x)$ is a function sampled from the multivariate Gaussian, μ is the mean function, which gives the expected value of the function at any point x and $k(x, x')$ is the covariance function or "kernel", which defines the covariance between the function at x and x'.

We assume then that the joint distribution of the observed values f_d and the predicted values f_* follow a multivariate Gaussian:

$$p\left(\begin{bmatrix} f_* \\ f_d \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} f_* \\ f_d \end{bmatrix}; \begin{bmatrix} \mu(x_*) \\ \mu(x_d) \end{bmatrix}, \begin{bmatrix} K(x_*,x_*) & K(x_*,x_d) \\ K(x_d,x_*) & K(x_d,x_d) \end{bmatrix}\right) \tag{2}$$

From this joint Gaussian, we can define the probability of our predictions given the observed data like so

$$p(f_*|f_d) = \mathcal{N}\left(f_*; m_{*|d}, C_{*|d}\right), \tag{3}$$

where:

$$m_{*|d} = \mu(x_*) + K(x_*, x_d)K(x_d, x_d)^{-1}(f_d - \mu(x_d)) \tag{4}$$

$$C_{*|d} = K(x_*, x_*) - K(x_*, x_d)K(x_d, x_d)^{-1}K(x_d, x_*) \tag{5}$$

$m_{*|d}$ is the predicted mean and $C_{*|d}$ is the predicted covariance aka the uncertainty in the prediction.

The hyperparameters θ of the kernel in a Gaussian Process can be optimized by maximising the log

marginal likelihood, which is given by:

$$\log p(f|X,\theta) = -\frac{1}{2} f^\top K^{-1} f - \frac{1}{2} \log \det K - \frac{n}{2} \log 2\pi \tag{6}$$

2 Experiments and Results

2.1 Question 1

In the lab, we explore data from SotonMet, a weather monitoring system that provides real-time and historical weather data for Southampton Water. Our dataset contains readings at 5 minute intervals for metrics such as air temperature, wind speed and tide height in the harbour as well as their true values. Given noisy and missing tide height sensor readings, we are given the task of predicting the missing values using Gaussian Processes, which in the real world would aid ships in navigating safely.

In Question 1, we are tasked with simply loading the data, a plot of which is shown (along with the results of Gaussian Process Regression) in Figure 3. For data pre-processing, I removed duplicates, centered the data and added 100 steps of padding at the start and end such that the variance of the Gaussian Process increasing in these regions would be visible in the plots.

2.2 Question 2

In Question 2, we were tasked to write our own Gaussian Process code and create predictions for the missing readings. I decided to challenge myself to write the Gaussian process code in PyTorch so that I could explore different optimizers such as Adam and SGD with Momentum in Question 5.

I started by exploring the Squared Exponential kernel as shown in Figure 1.

To ensure the code is working as expected, we can plot draws from the Gaussian Process as shown in Figure 2:

In this question, I struggled with finding the correct hyperparameters. Diving deeper into what they fundamentally represent and making better estimates was a great help! The result of Gaussian Process Regression is shown in Figure 3

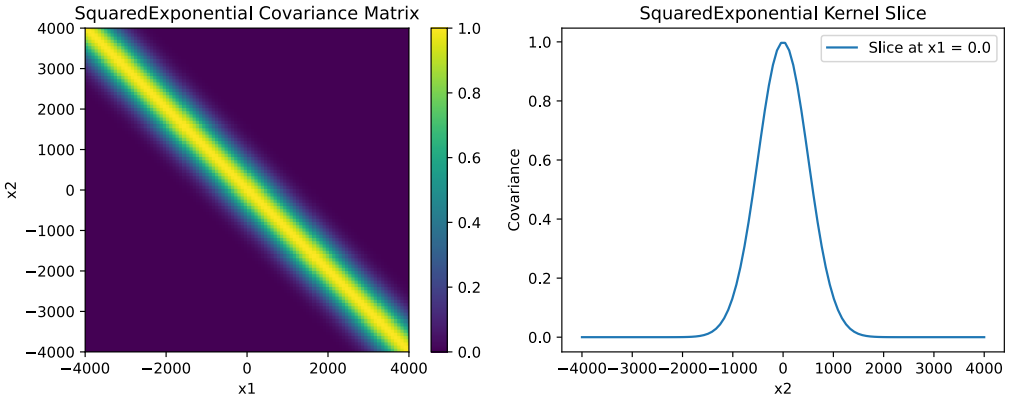

Figure 1: Figures depicting the squared exponential kernel.

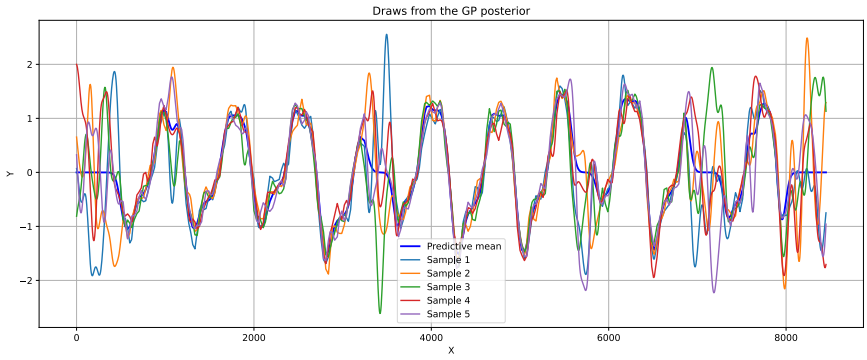

Figure 2: Figure depicting 5 random draws from the Gaussian Process.

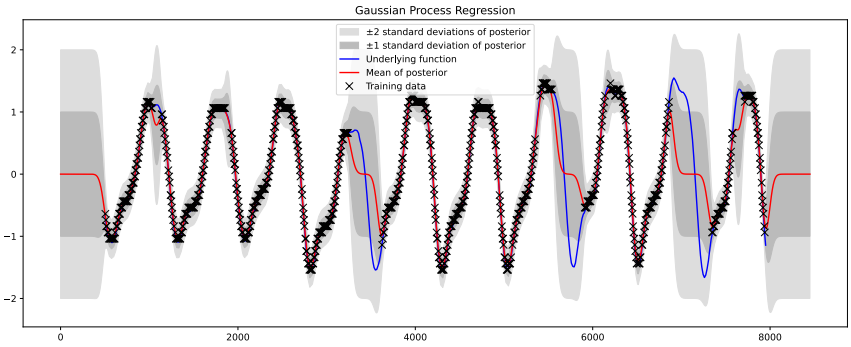

Figure 3: Figure depicting 5 random draws from the Gaussian Process.

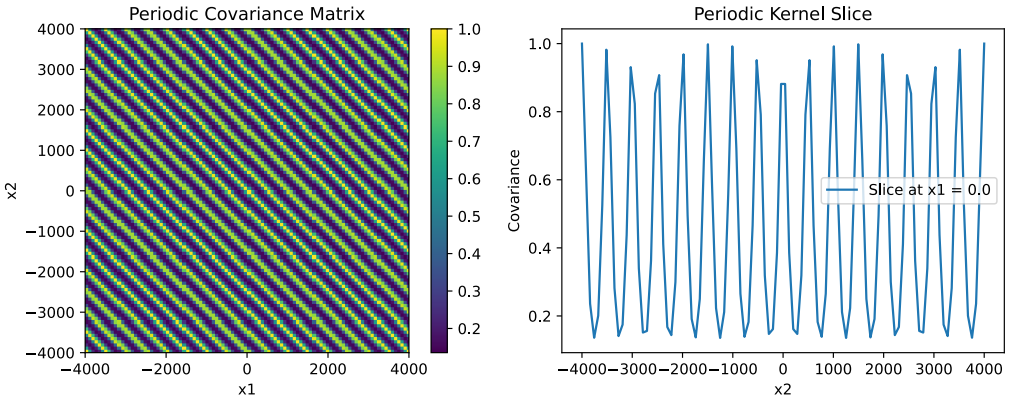

Figure 4: Figure depicting a periodic kernel.

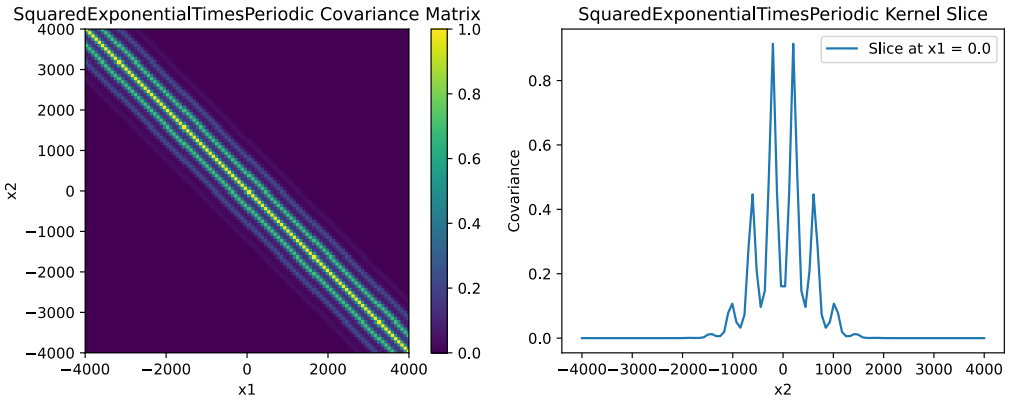

Figure 5: Figure depicting a periodic kernel multiplied by a squared exponential kernel.

2.3 Question 3

In Question 3, we are asked to compare against the ground truth tide heights using root-mean-square-error (RMSE), which when implemented resulted in a RMSE of 0.4474777281284332.

2.4 Question 4

In Question 4, we are tasked with exploring more sophisticated covariance functions. Having already explored Squared Exponential, I went on to explore Periodic, Squared Exponential plus Periodic and Squared Exponential times Periodic, as shown in Figure 5. For brevity's sake, I won't include plots of all the kernels and resulting Gaussian Processes here however you will find them all in the accompanying Python notebook.

2.5 Question 5

In Question 5, we were tasked with investigating alternate means of managing the hyperparameters. I opted for a different approach by implementing my kernels as subclasses of torch.nn.Module. Defining kernel parameters as torch.nn.Parameter allowed me to implement gradient-based hyperparameter optimization, minimising the negative log marginal likelihood via Adam. This approach effectively improves model performance by tuning parameters to maximise the likelihood of the observed data. I thought it would be of interest to include a snippet of code as show in Listing 1.

```
class SquaredExponential(CovarianceFunction):
    def __init__(self, l: float, sigma_f: float):
        super(SquaredExponential, self).__init__()
        # Register parameters so that they can be optimized
        self.l = nn.Parameter(torch.tensor(l))
        self.sigma_f = nn.Parameter(torch.tensor(sigma_f))

    def forward(self, x1, x2):
        dist = (x1.unsqueeze(1) - x2.unsqueeze(0)).pow(2).sum(2)
        return self.sigma_f**2 * torch.exp(-0.5 * dist / self.l**2)

...

class SquaredExponentialTimesPeriodic(CovarianceFunction):
    def __init__(self, l:float = 700.0, sigma_f:float = 1.0, p:float = 700.0, omega:float =
    1.0):
        super(SquaredExponentialTimesPeriodic, self).__init__()
        # Register the covariance functions as submodules
        self.squared_exponential = SquaredExponential(l=l, sigma_f=sigma_f)
        self.periodic = Periodic(omega=omega, sigma_f=sigma_f, p=p)

    def forward(self, x1: torch.Tensor, x2: torch.Tensor):
        return self.squared_exponential(x1, x2) * self.periodic(x1, x2)

class GaussianProcess:
    ...

    def optimize_hyperparameters(self, learning_rate=0.01, n_iters=100):

```

```
        optimizer = torch.optim.Adam(self.kernel.parameters(), lr=learning_rate)

        for i in tqdm(range(n_iters)):
            optimizer.zero_grad()
            loss = -self.log_marginal_likelihood()
            loss.backward()
            optimizer.step()
```

Listing 1: Implementation of Gradient-based Hyperparameter Optimization

3 Discussion and Analysis

As my first experience with Bayesian Nonparametrics, I had a lot of base level understanding to gain. In my implementation I assume zero mean and therefore center the data. In further implementations, I would explore removing this assumption.

In my initial implementation, I explicitly computed the matrix inverse which I found to be remarkably unstable and prone to conditioning errors. To rectify this I switched to the Cholesky inverse, which proved most helpful during hyperparameter optimization. In addition, I found that K was often singular, and therefore could not be inverted. To rectify this I included jitter and noise variance.

Through experimentation, I found that the Squared Exponential kernel performed remarkably well, although did have considerable trouble choosing the initial hyperparameters. The Periodic kernel performed very poorly and during hyperparameter optimization, Adam was never able to find a suitable set for the data. The highest performing kernels were the compositions of Periodic and Squared Exponential. This matches my intuition as the data does contain periodicity, with closer data points holding more influence.