CUED - Engineering Tripos Part IIB 2021-2022 **Module Coursework**

Module	4F13	Title of report	Gaussian Processes

Date submitted: 02/11/2021	Assessment for this module is [✓ 100%] / ☐ 25% coursework of which this assignment forms 33 %

UNDERGRADUATE STUDENTS ONLY		POST GRADUATE STUDENTS ONLY			
Candidate number:	5585G	Name:		College:	

Feedback to the student ☐ **See also comments in the text**		Very good	**Good**	Needs improvmt
C O N T E N T	**Completeness, quantity of content:** Has the report covered all aspects of the lab? Has the analysis been carried out thoroughly?			
	Correctness, quality of content Is the data correct? Is the analysis of the data correct? Are the conclusions correct?			
	Depth of understanding, quality of discussion Does the report show a good technical understanding? Have all the relevant conclusions been drawn?			
	Comments:			
P R E S E N T A T I O N	**Attention to detail, typesetting and typographical errors** Is the report free of typographical errors? Are the figures/tables/references presented professionally?			
	Comments:			

Indicative grades are not provided for the FINAL piece of coursework in a module

Assessment (circle one or two grades)		A*	A	B	C	D
Indicative grade guideline		>75%	65-75%	55-65%	40-55%	<40%
Penalty for lateness:		*20% of maximum achievable marks per week or part week that the work is late.*				

Marker: Date:

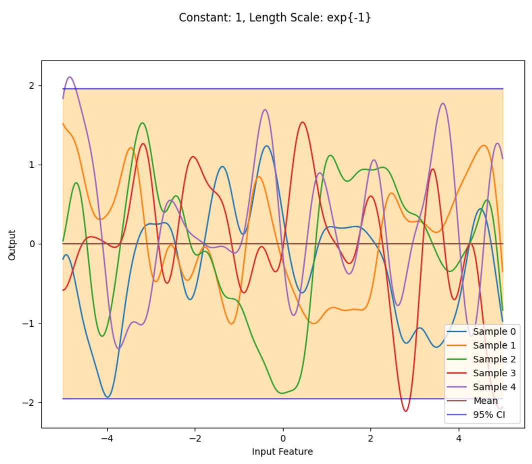
1 Gaussian Processes

1.1 Question 1:

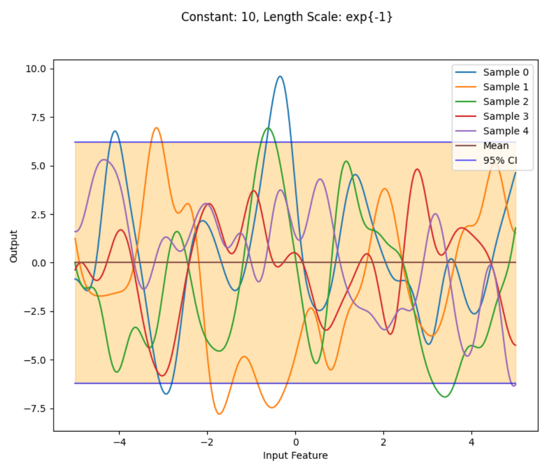
The first Gaussian Process Regressor (GPR) has a simple kernel corresponding to the covariance function:

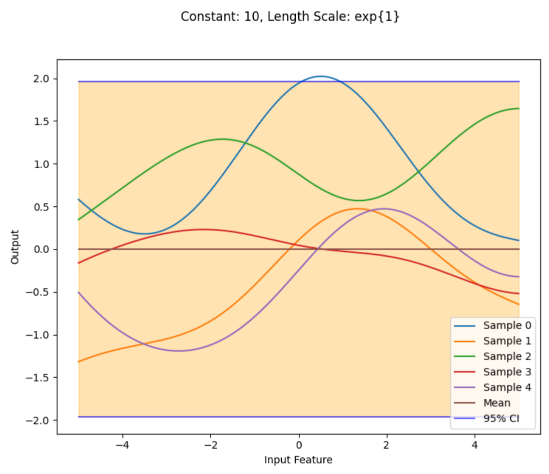
$$k(x_1, x_2) = v^2 \exp\{-\frac{1}{2l^2}(x_1 - x_2)^2\} + {\sigma_n}^2 \delta_{x_1 x_2}$$

v^2 controls the the function variance, l controls the breadth of the peaks in the samples. Some illustrative examples are:



(a) Base example. $v^2 = 1, l = e^{-1}$





(b) Larger constant. $v^2 = 10, l = e^{-1}$

(c) Longer length scale. $v^2 = 1, l = e^1$

Figure 1: Noiseless samples from Squared Exponential Covariance

A larger v^2 gives functions with a larger amplitude, whilst a longer scale gives samples with broader peaks.

The kernel was initialised to: $\ln l = -1$, $\ln C = 0$ and $\ln \sigma^2 = 0$. The model was trained on data from `cw1a.mat` by tuning the hyperparameters to minimise the negative log-marginal-likelihood (LML).

```
# python libraries for GPR
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import WhiteKernel, RBF, ExpSineSquared

# prefactor is the signal amplitude, length_scale is the RBF scale
# WhiteKernel includes noise_level to measure the noise variance
kernel = 1.0 * RBF(length_scale=np.exp(-1), length_scale_bounds=(1e-5,1e5)) + WhiteKernel
    (noise_level=np.exp(0), noise_level_bounds=(1e-10, 1e3))

# build the regressor model
gpr = GaussianProcessRegressor(kernel=kernel)

# fit the regressor where x is the input feature and y is the output
gpr.fit(x, y)

# use the GPR model to perform prediction over the x axis
y_mean, y_std = gpr.predict(x_plot, return_std=True)
```

Figure 2: Python code for Q1

This trained model defines a GP with covariance shown below, from which posterior sample functions can be drawn:

$$k(x_1, x_2) = 0.897^2 \exp\{-\frac{1}{2 \times 0.128^2}(x_1 - x_2)^2\} + 0.118^2 \delta_{x_1,x_2}$$

The posterior model is used to form confidence intervals (CIs) over the x-axis. The results are shown below with the black line representing the expected value of a sample, and the blue lines defining 95% CIs (mean $\pm 1.96 \times$standard deviations).

The plots show that areas with several training data points are characterised by tight confidence bands (confident predictions) and the outer regions where training data are sparse have broad bands (very unconfident predictions). The short length scale ($l = 0.128$) allows for tight oscillatory patterns of the sample functions through the training data and the signal variance, v^2, is scaled appropriately to pass through the training points. The confidence intervals tighten up around the single datapoints in the sparse regions but do not completely narrow to zero since noise with variance ${\sigma_n}^2 = 0.118^2$ is incorporated in the generative model.

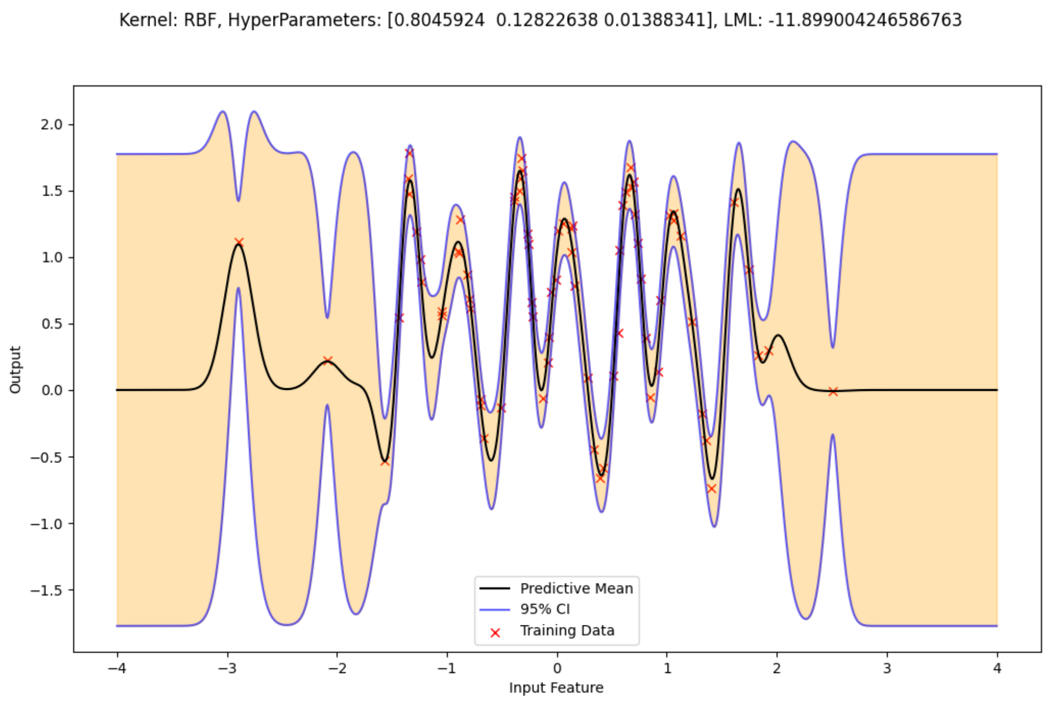


Figure 3: Full predictive distribution for *cw1a.mat* using RBF

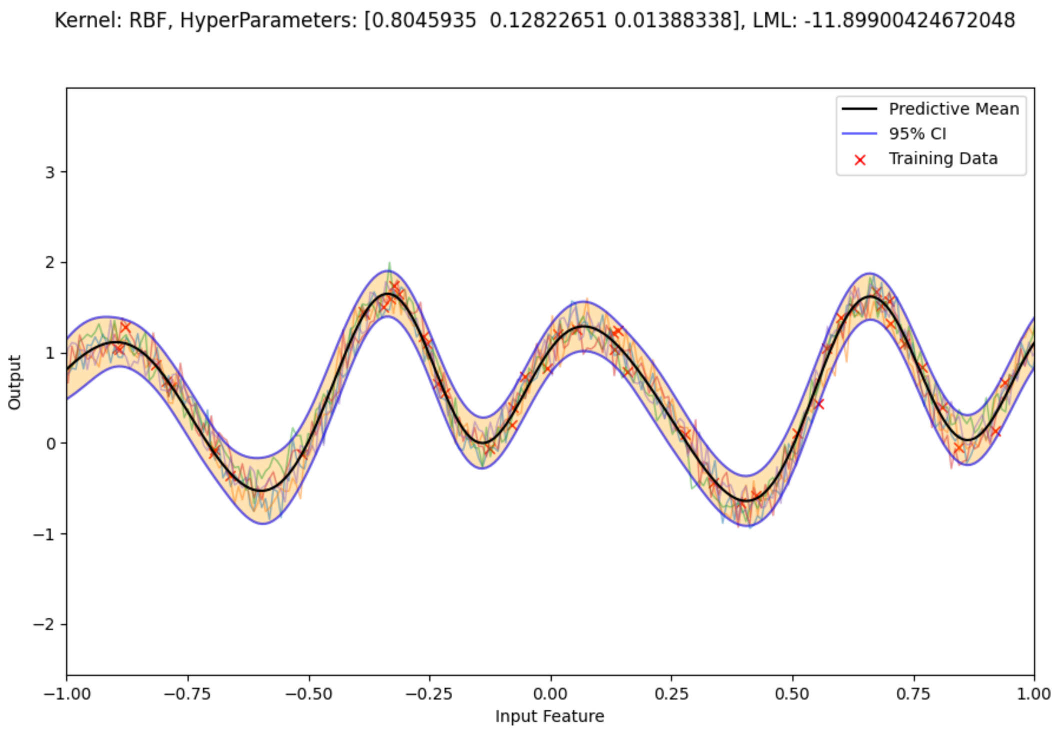


Figure 4: Zoomed in view including some posterior samples

1.2 Question 2

The LML may have multiple local optima which means that the optimum found depends on the initialisation of hyperparameters.

I found several different local optima. If we examine the form of the covariance function under limits:

$$l \to \infty \quad \text{gives} \quad k(x_1, x_2) = v^2 + {\sigma_n}^2 \delta_{x_1, x_2}$$

$$l \to 0 \quad \text{gives} \quad k(x_1, x_2) = {\sigma_n}^2 \delta_{x_1, x_2}$$

Which both correspond to a constant level plus white noise. Therefore we expect our sample functions to consist of independent random samples and to have no covariance structures (i.e flat mean, flat confidence intervals). These models are modelling the variation in the input as solely noise. Some example plots are shown below:

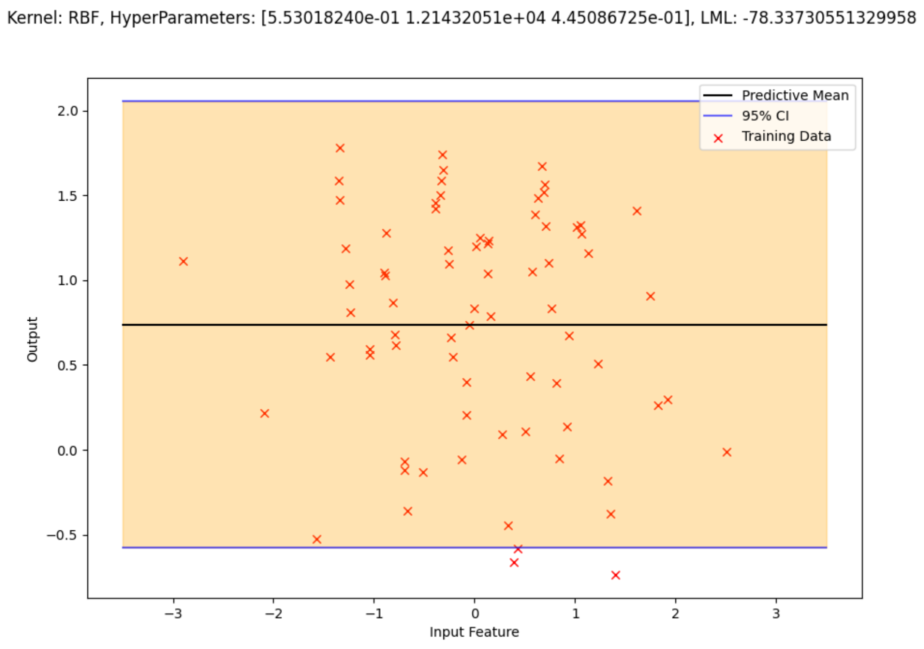


Figure 5: Large length scale

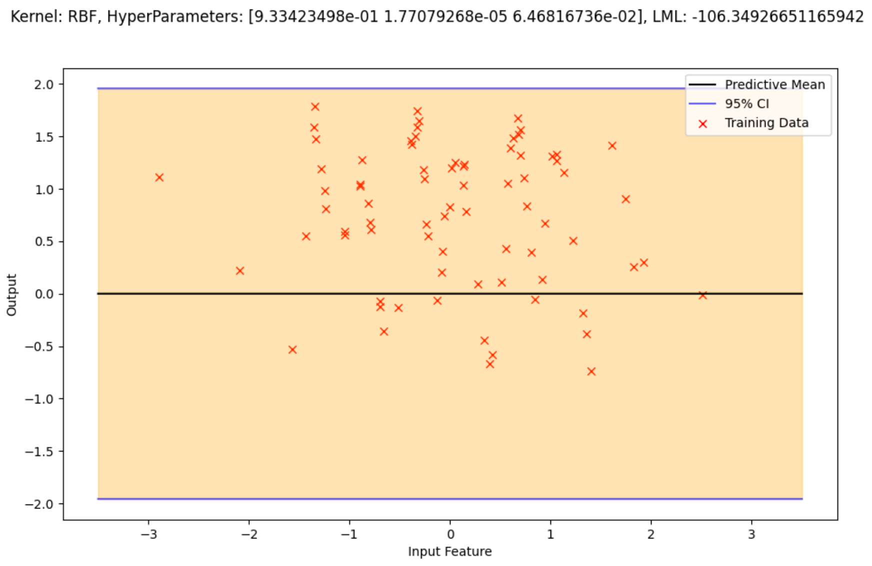


Figure 6: Very short length scale

Figure (5) demonstrates how a large l incorporates a mean level (around $\sqrt{0.553} = 0.746$). Figure (6) demonstrates a very short length scale. In this case, the confidence interval is broader than the long length scale case - since the model cannot include the mean.Hence the fit is poorer.

Other local optima existed, including one with a moderately large length scale. This model captured a large-scale trend that is more complex than a linear trend, but still appears very general when plotted over the training data:

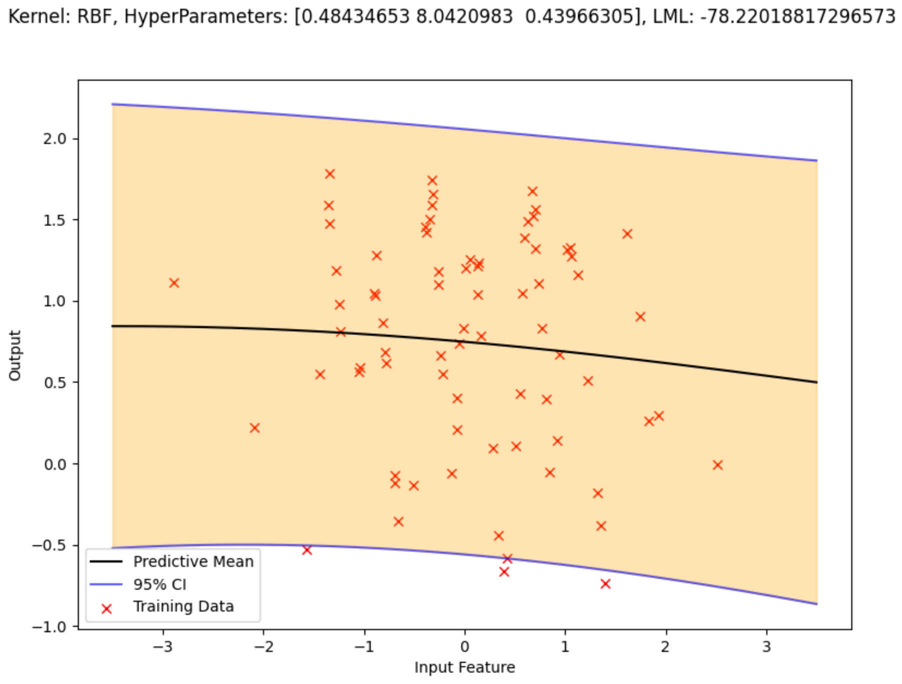


Figure 7: Very short length scale

These models have significantly poorer LML than the model in part (a), thus are not global optima for the LML. The model in part (a) is the best combination of data fit and complexity penalty for the models considered, however this doesn't mean it is necessarily the best model for this data since there are infinitely many other models we could consider.

1.3 Question 3

This question explores a GP with a periodic covariance function:

$$k(x_1, x_2) = v^2 \exp\{-\frac{2}{l^2} \sin^2 \frac{\pi |x_1 - x_2|}{p}\} + {\sigma_n}^2 \delta_{x_1 x_2}$$

The hyperparameter p controls the *periodicity* of the model.

```
# Periodic Kernel
kernel = 1.0*ExpSineSquared(length_scale=np.exp(0), periodicity=np.exp(0)) + WhiteKernel(
    noise_level=np.exp(0), noise_level_bounds=(1e-10, 1e3))
```

Figure 8: Code for Q3

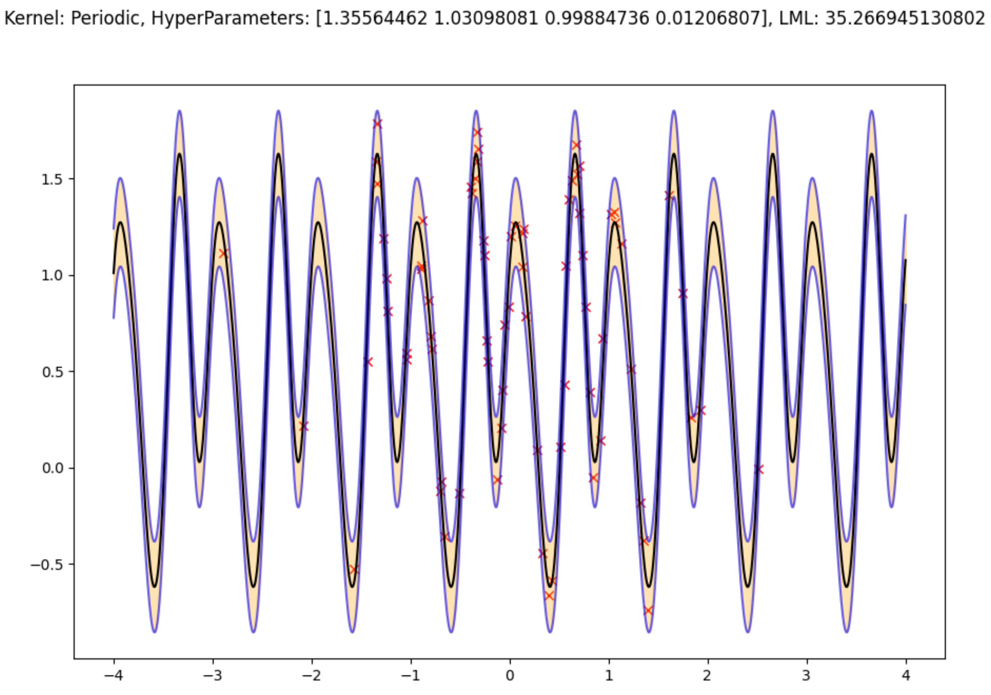


Figure 9: Predictive distribution extending beyond the training data

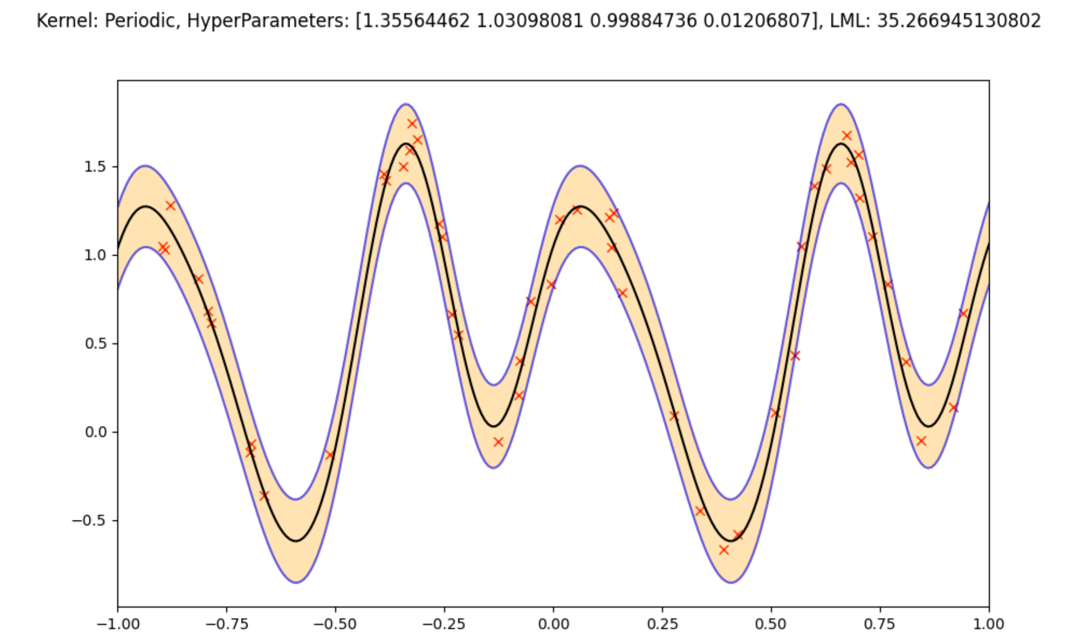


Figure 10: Fit in a data rich area

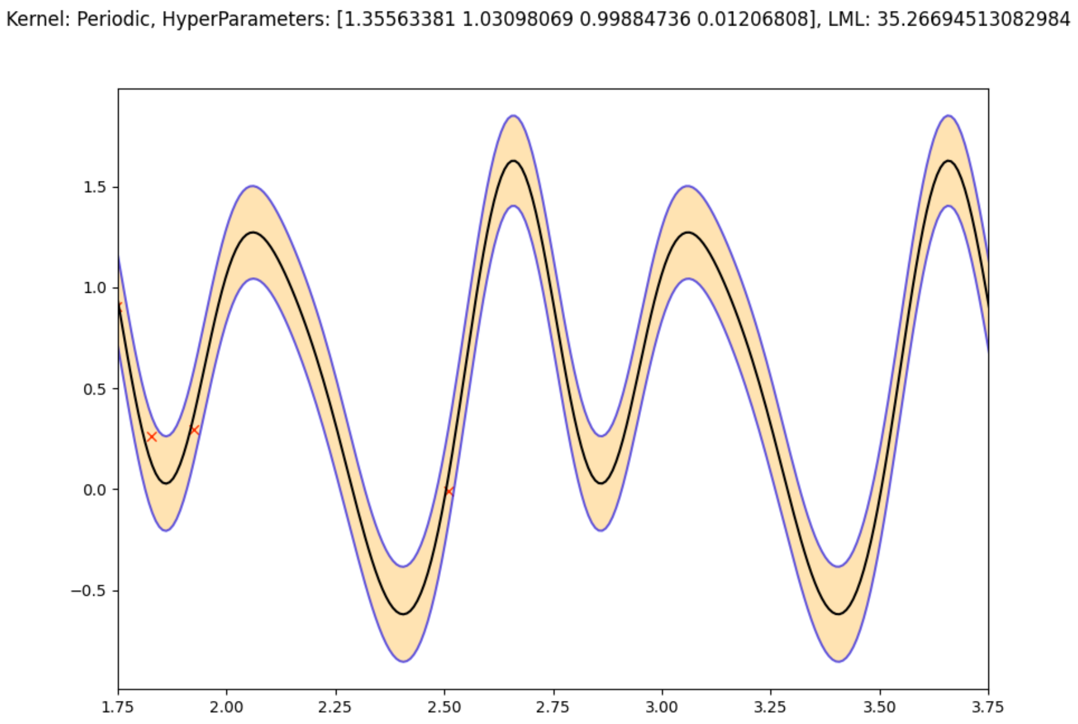


Figure 11: Fit in an area with little data

Interestingly, beyond the data, predictions maintain the periodic structure and the confidence bands remain very tight as opposed to becoming flat. The high LML indicates that this is a better model than using RBFs after accounting for the improved data fit and extra complexity of the parameter p. It gives good confidence that the generating model was periodic, though it would be worth checking the models performance on test data, or checking that the residuals of the fit are Gaussian. The model suggests a kernel of the form:

$$k(x_1, x_2) = 1.356^2 \exp\{-\frac{2}{1.031^2} \sin^2 \frac{\pi|x_1 - x_2|}{0.999}\} + 0.012^2 \delta_{x_1 x_2}$$

Training the model with two separate periodic kernels offered no improvement. The model removed one component by setting its periodicity to infinity. Training the model to include a periodic plus an RBF similarly removes the RBF component.

1.4 Question 4

We wish to sample from a composite kernel:

$$k(x_1, x_2) = v^2 \exp\{-\frac{2}{l_p^2} \sin^2 \frac{\pi|x_1 - x_2|}{p}\} \exp\{-\frac{1}{2l_{RBF}^2}(x_1 - x_2)^2\}$$

Where $v^2 = 1$, $l_{RBF} = e^2$, $l_P = e^{-0.5}$, $p = 1$. Two samples are shown below:

```
# axis to define samples on
x_axis = np.linspace(-5, 5, 200).reshape(-1, 1)

# composite kernel
kernel = 1.0*ExpSineSquared(length_scale=np.exp(-0.5), periodicity=np.exp(0))*RBF(
    length_scale=np.exp(2))

# build and take a sample
gpr = GaussianProcessRegressor(kernel=kernel)
y_samp = gpr.sample_y(x_axis, random_state=None)
```

Figure 12: Python code for Q4

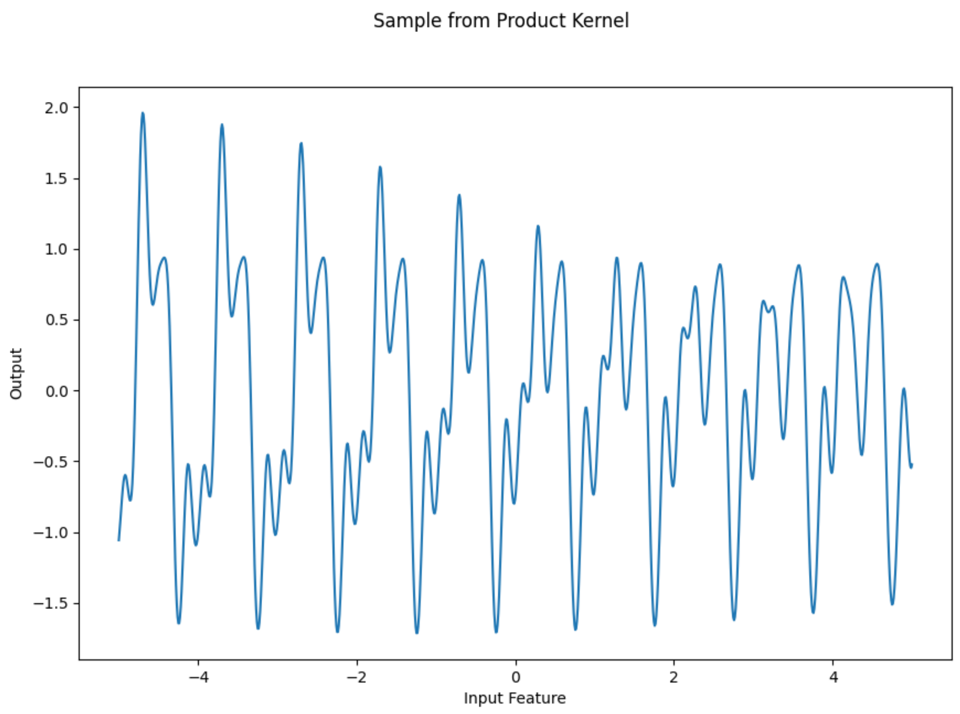


Figure 13: First sample

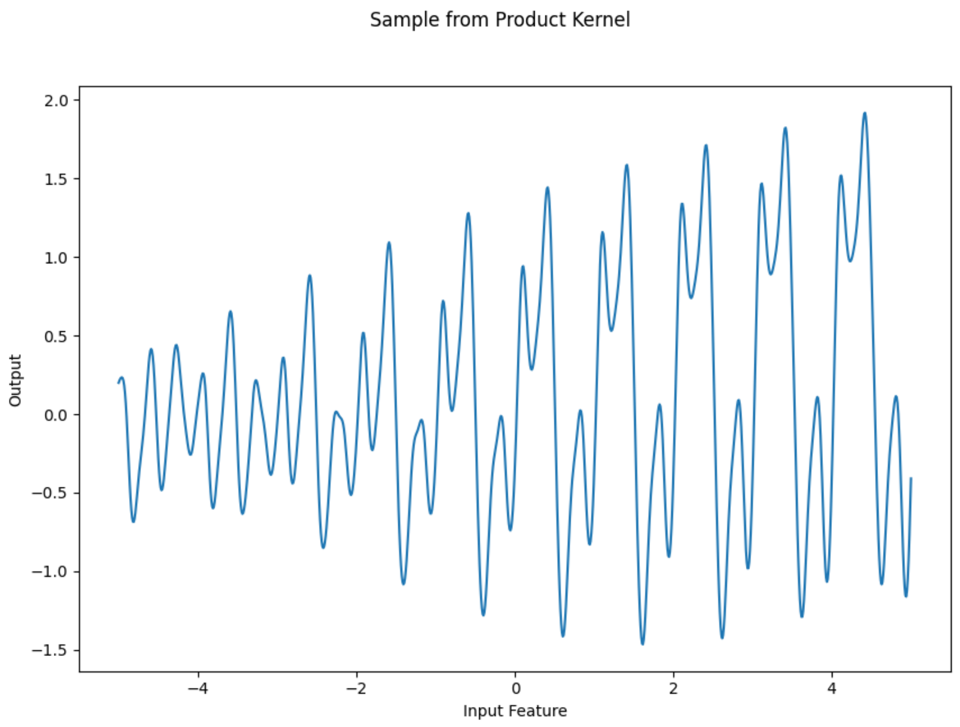


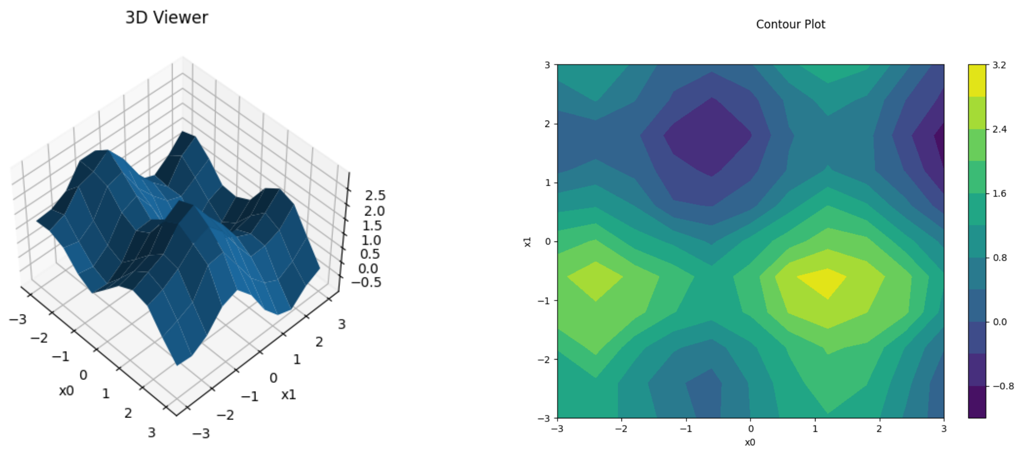
Figure 14: Second sample

The kernel consists of two competing factors - the RBF component tries to keep data points close together, whilst the periodic component tries to drive the oscillating structure. This gives sample functions with a periodic structure that can vary across input space. Here, the RBF length scale is quite long compared to the periodicity of the kernel hence the variations in the oscillation are slow. Shorter RBF length scales allow for faster variation of the periodic structure which lets sample functions jump quickly between different oscillation patterns.

The adding of a small diagonal matrix is handled by `alpha` in the `GaussianProcessRegressor` class, which improves robustness against round-off error in the Cholesky decomposition by promoting positive definiteness in the kernel.

1.5 Question 5

Question 5 examined the use of GPs for data with two input features.



(a) 3D Visual (b) Contour Plot

Figure 15: Two input training data

There is some correlation between input data points as the output surface has clear peaks and troughs, we postulate two models:

$$\mathcal{M}_1\colon\ k(x,x') = v^2 \exp\{-\tfrac{1}{2l_1^2}(x_1 - x_1')^2 - \tfrac{1}{2l_2^2}(x_2 - x_2')^2\} + {\sigma_n}^2\delta_{x_1x_2}$$

$$\theta_1\colon\ \{v^2, l_1, l_2, {\sigma_n}^2\}$$

$$\mathcal{M}_2\colon\ k(x,x') = v_1^2 \exp\{-\tfrac{1}{2l_1^2}(x_1 - x_1')^2\} + v_2^2 \exp\{-\tfrac{1}{2l_2^2}(x_2 - x_2')^2\} + {\sigma_n}^2\delta_{x_1x_2}$$

$$\theta_2\colon\ \{v_1^2, v_2^2, l_1, l_2, {\sigma_n}^2\}$$

Which represent the product and sum of two RBF kernels respectively.

```
# Product Kernel
kernel1 = 1.0*RBF(length_scale=np.random.rand(2)) + WhiteKernel(noise_level=np.exp(0),
    noise_level_bounds=(1e-10, 1e3))
# Sum Kernel
kernel2 = 1.0*RBF(length_scale=np.random.rand(2)) + 1.0*RBF(length_scale=np.random.rand
    (2)) + WhiteKernel(noise_level=np.exp(0), noise_level_bounds=(1e-10, 1e3))

# two separate models
gpr1 = GaussianProcessRegressor(kernel=kernel1)
gpr2 = GaussianProcessRegressor(kernel=kernel2)

# input feature array of shape (n_samples, n_features)
inputs = np.array([x1, x2]).T
= corresponding training outputs
outputs = y

# fit each model
gpr1.fit(inputs, outputs)
gpr2.fit(inputs, outputs)
```

Figure 16: Python code for Q5

The best resulting fits were:

Kernel	`1.11**2 * RBF(length_scale=[1.51, 1.29]) + WhiteKernel(noise_level=0.0105)`
LML	19.21874853620224

Table 1: Product kernel

Kernel	`0.71**2 * RBF(length_scale=[8.27e+04, 0.986]) + 1.11**2 *RBF(length_scale=[1.45, 1e+05]) + WhiteKernel(noise_level=0.00957)`
LML	66.40825185677517

Table 2: Sum kernel

This suggests through Occam's Razor that the sum kernel model is a better fit. The fits are shown as 3D surfaces;

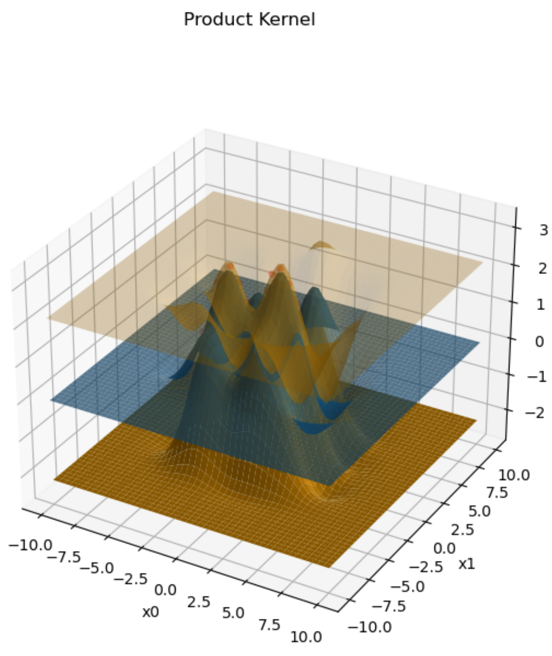


Figure 17: Red: Training Data, Blue: Mean Surface, Orange: Confidence Bands

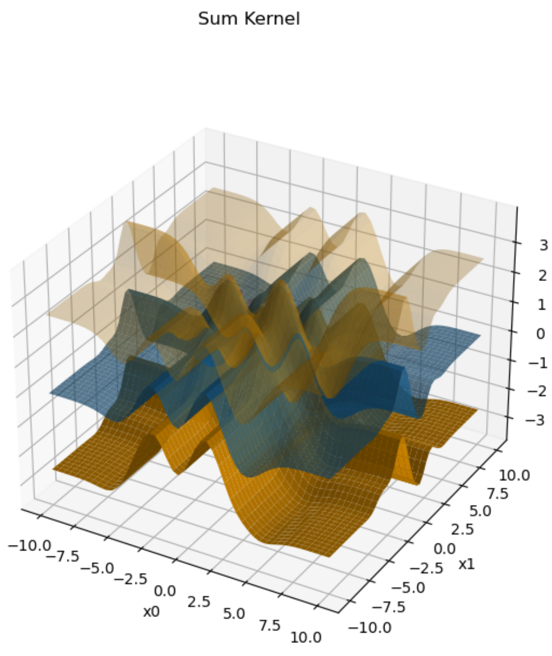


Figure 18: Red: Training Data, Blue: Mean Surface, Orange: Confidence Bands

The 3D surfaces show the different fits in regions beyond the data, and how the use of two RBF kernels has allowed the model to detect correlation in each feature separately (as shown by the CI structure outside the area with training data).

Examining each kernel we see that the noise levels are approximately the same suggesting there was some faint noise present in the generation process. Further, the sum kernel treated the two input features independently using one RBF for each feature (since a very large length scale implies a flat covariance structure for that feature) and benefitted from the ability to learn two separate function variance parameters. The product kernel was forced to compromise between both input features which lead to a poorer fit. In this case, the extra complexity of the parameter v_2^2 is justified by the improved fit in the LML, which quantifies the trade-off between data fit and model complexity.

Word Count: 958 not including code or section titles