4F13: Gaussian Processes
Coursework 1

Raghavendra Narayan Rao

November 7, 2025

Abstract

This report investigated the effect of squared exponential and periodic kernels, individually and combined. Surprisingly, when added, the resulting model achieves low complexity.

1 Introduction

Fitting gaussian processes involves finding the set of hyperparameters that produce the maximum log marginal likelihood (LML). This report investigates how the LML is maximised under different kernels.

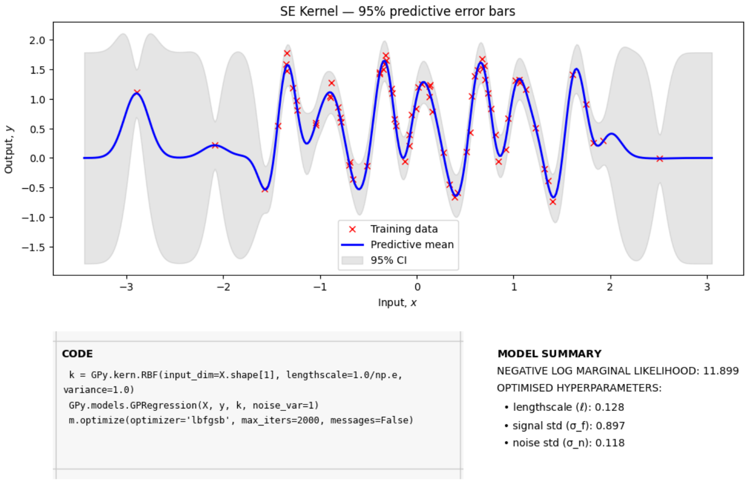
$$\log Z_{|y} = \underbrace{-\tfrac{1}{2}\mathbf{y}^\top[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}\mathbf{y}}_{\text{Data fit term}} - \underbrace{\tfrac{1}{2}\log|\mathbf{K}+\sigma_n^2\mathbf{I}|}_{\text{Complexity penalty term}} - \tfrac{n}{2}\log(2\pi) \tag{1}$$

2 Task a - Squared Exponential Kernel

In this task, we use a squared exponential kernel (RBF) to train a Gaussian process. The initialisations and corresponding Python code are provided in Figure 1.

$$k(\mathbf{x},\mathbf{x}') = \sigma_f^2 \exp\left(-\tfrac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x_d')^2}{l_d^2}\right)$$

The points appear to only be correlated with those that belong to the same peak so the length scale, 0.13, is lower than the width of the peaks, 0.5. The noise standard deviation is low as the model is able to fit the data well (between -1 and 2).



CODE

```
 k = GPy.kern.RBF(input_dim=X.shape[1], lengthscale=1.0/np.e,
variance=1.0)
 GPy.models.GPRegression(X, y, k, noise_var=1)
 m.optimize(optimizer='lbfgsb', max_iters=2000, messages=False)
```

MODEL SUMMARY

NEGATIVE LOG MARGINAL LIKELIHOOD: 11.899

OPTIMISED HYPERPARAMETERS:

- lengthscale (ℓ): 0.128
- signal std (σ_f): 0.897
- noise std (σ_n): 0.118

Figure 1: Fit Gaussian Process with squared exponential kernel

2.1 Error Bars

The variance of the predictive distribution will be reduced when more data points are provided. The squared exponential kernel amplifies this effect for predictions near data points.

$$p(\mathbf{y}_* \mid \mathbf{x}_*, \mathbf{x}, \mathbf{y}) \sim \mathcal{N}\left(\mathbf{k}(\mathbf{x}_*, \mathbf{x})^\top \left[\mathbf{K} + \sigma^2_{\text{noise}}\mathbf{I}\right]^{-1}\mathbf{y},\ k(\mathbf{x}_*, \mathbf{x}_*) + \sigma^2_{\text{noise}} - \mathbf{k}(\mathbf{x}_*, \mathbf{x})^\top \left[\mathbf{K} + \sigma^2_{\text{noise}}\mathbf{I}\right]^{-1}\mathbf{k}(\mathbf{x}_*, \mathbf{x})\right)$$

The majority of x values are between -1 and 2. Therefore, the variance in predictive y values are reduced and the error bars are much thinner in this region. At the tails of the data distribution ($x < -1.5$ and $x > 2$), there are way fewer data points to reduce the variance so the error bars are large.

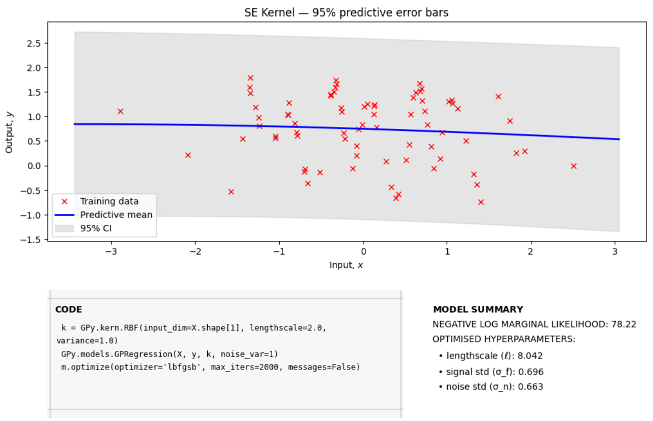
3 Task b - Local Optima

The goal is to determine if the optimum found in task a is unique (and hence a global optimum). By performing the following grid search on initial hyperparameters, 2 local optimums were found.

```python
lengthscale, variance, noise = np.linspace(0.01, 2, 10), np.linspace(0.01, 2, 10), np.
    linspace(0.01, 2, 10)
L, V, N = np.meshgrid(lengthscale, variance, noise, indexing='ij')
for l,v,n in np.stack([L.ravel(), V.ravel(), N.ravel()], axis=-1):
    # ... create kernel with l, v, n hyperparameters ...
    m.optimize(optimizer='lbfgsb', max_iters=2000, messages=False)
```

Listing 1: Hyperparmeter Grid Search

The grid search revealed that there are only 2 optimums, the one found in task a and the other shown in Figure 2. This optimum has a much larger length scale than that in task a. A very large length scale results in the model treating every data point as noise so it merely plots the mean (DC) value. Therefore, task a, with the much lower NLML and error bars, is the better optimum.



CODE

```python
 k = GPy.kern.RBF(input_dim=X.shape[1], lengthscale=2.0,
variance=1.0)
 GPy.models.GPRegression(X, y, k, noise_var=1)
 m.optimize(optimizer='lbfgsb', max_iters=2000, messages=False)
```

MODEL SUMMARY

NEGATIVE LOG MARGINAL LIKELIHOOD: 78.22

OPTIMISED HYPERPARAMETERS:

- lengthscale (ℓ): 8.042
- signal std (σ_f): 0.696
- noise std (σ_n): 0.663

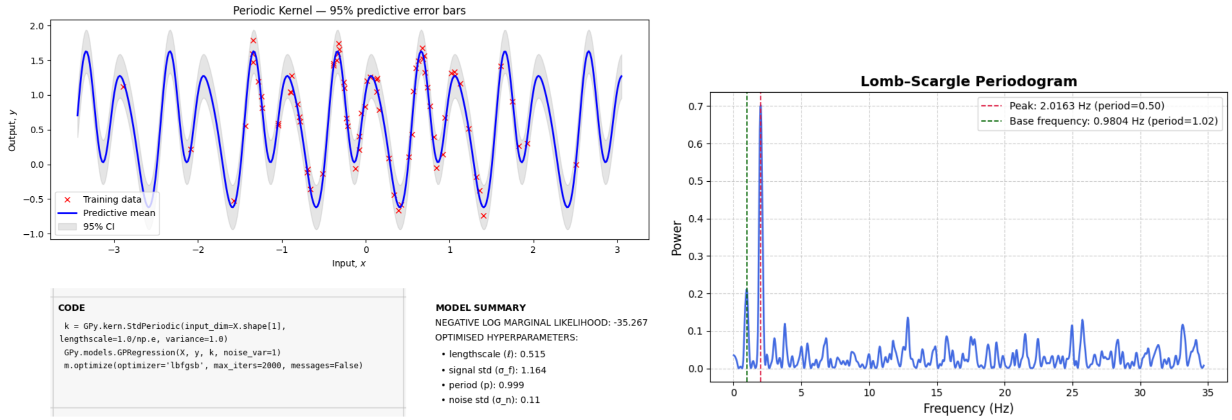
Figure 2: Fit Wide squared exponential kernel

4 Task c - Periodic Kernel

We use the following GPy implementation of a periodic kernel.

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left[-\frac{2}{l^2}\sin^2\left(\frac{\pi\|\mathbf{x} - \mathbf{x}'\|}{p}\right)\right]$$

The error bars produced by the periodic kernel (left of Figure 3) are a lot thinner on the tails of the distribution compared to the square kernel as the periodic kernel effectively has "infinite lengthscale". Local information can provide period estimates that affect global predictions of the model.



CODE

```
 k = GPy.kern.StdPeriodic(input_dim=X.shape[1],
lengthscale=1.0/np.e, variance=1.0)
 GPy.models.GPRegression(X, y, k, noise_var=1)
 m.optimize(optimizer='lbfgsb', max_iters=2000, messages=False)
```

MODEL SUMMARY

NEGATIVE LOG MARGINAL LIKELIHOOD: -35.267

OPTIMISED HYPERPARAMETERS:

- lengthscale (ℓ): 0.515
- signal std (σ_f): 1.164
- period (p): 0.999
- noise std (σ_n): 0.11

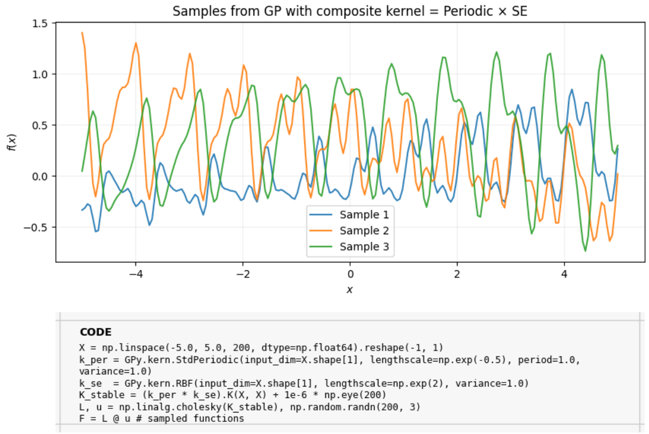
Figure 3: Fit Gaussian Process with periodic kernel

4.1 Periodicity of Data generating function

The frequency spectrum of the unevenly spaced data is produced as a Lomb-Scargle periodogram on the right of Figure 3. The base period is 1.02, which agrees with the model's predicted period. The peak frequency explains the half period fluctuations in the data. The rest of the frequencies' amplitudes are much smaller and clearly indicate noise in the data. Therefore, the data is periodic (with period 1) ignoring noise.

5 Task d - Product of Squared exponential and Periodic Kernels

By taking a product of both kernels, we have designed a hybrid model in Figure 4. There is a need to add a small diagonal matrix to improve the numerical stability of the cholesky algorithm. This arises due to miniscule eigenvalues of the covariance matrix (and hence has a high condition number). The squared exponential's length scale can control how well the model fits the data (overfit by reducing length scale). The period provides low error bars. Figure 5 contrasts the two.



CODE

```
X = np.linspace(-5.0, 5.0, 200, dtype=np.float64).reshape(-1, 1)
k_per = GPy.kern.StdPeriodic(input_dim=X.shape[1], lengthscale=np.exp(-0.5), period=1.0,
variance=1.0)
k_se  = GPy.kern.RBF(input_dim=X.shape[1], lengthscale=np.exp(2), variance=1.0)
K_stable = (k_per * k_se).K(X, X) + 1e-6 * np.eye(200)
L, u = np.linalg.cholesky(K_stable), np.random.randn(200, 3)
F = L @ u # sampled functions
```

Figure 4: Product of Squared exponential and Periodic Kernels

6 Task e - Data Fit vs Complexity

In this task, we compare a Single RBF with a sum of 2 different RBF. (shown in Figure 6). Although the summed model achieves a higher LML than the single model, this alone does not imply that it is better. A fair comparison requires examining the components (1) contributing to the LML. Both models have identical data fit terms but the summed model has a more negative complexity penalty term. This means that the summed

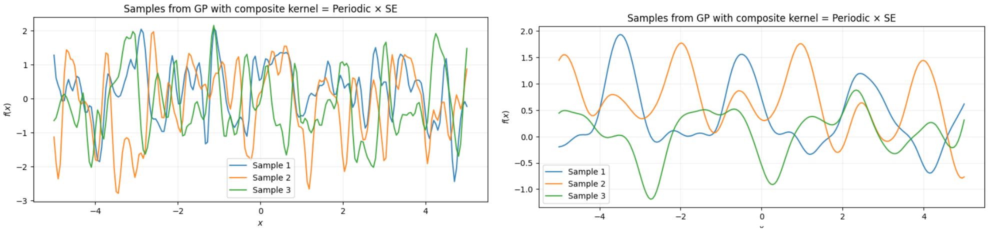
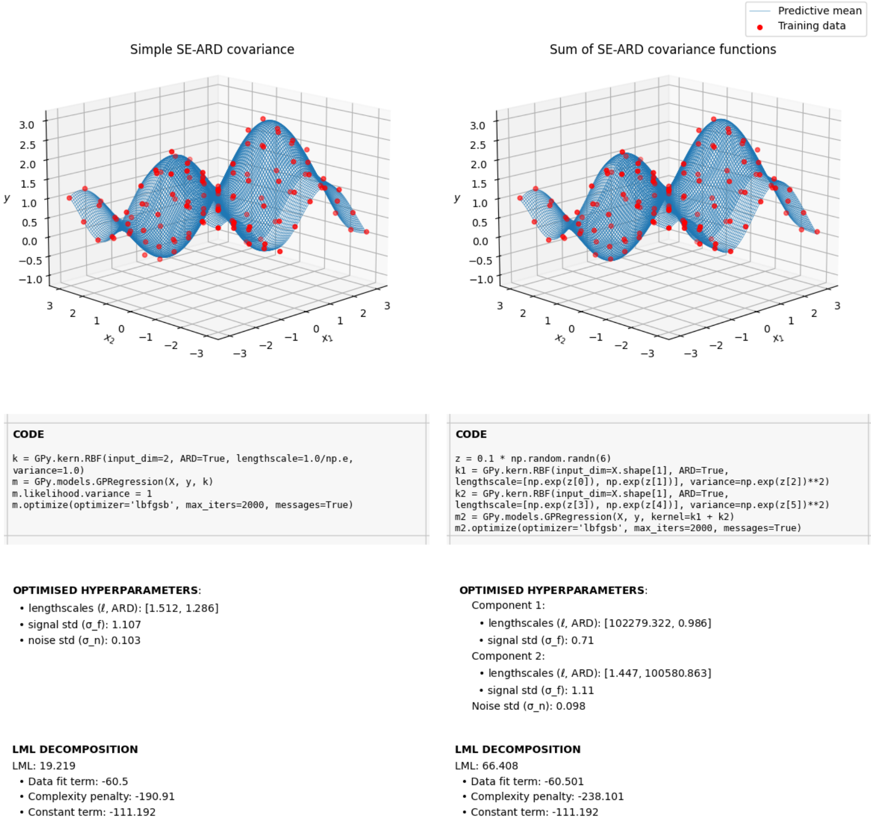


Figure 5: Reduce length scale of RBF (left) vs Increase period (right)

model type has managed to converge to a simpler model despite having more initial hyperparameters. It has achieved this by pushing 2 of the 4 length scale parameters to "effectively" infinity, making them redundant. Therefore, by Occam's Razor, the summed model is the better choice.



CODE

```
k = GPy.kern.RBF(input_dim=2, ARD=True, lengthscale=1.0/np.e,
variance=1.0)
m = GPy.models.GPRegression(X, y, k)
m.likelihood.variance = 1
m.optimize(optimizer='lbfgsb', max_iters=2000, messages=True)
```

CODE

```
z = 0.1 * np.random.randn(6)
k1 = GPy.kern.RBF(input_dim=X.shape[1], ARD=True,
lengthscale=[np.exp(z[0]), np.exp(z[1])], variance=np.exp(z[2])**2)
k2 = GPy.kern.RBF(input_dim=X.shape[1], ARD=True,
lengthscale=[np.exp(z[3]), np.exp(z[4])], variance=np.exp(z[5])**2)
m2 = GPy.models.GPRegression(X, y, kernel=k1 + k2)
m2.optimize(optimizer='lbfgsb', max_iters=2000, messages=True)
```

OPTIMISED HYPERPARAMETERS:
- lengthscales (ℓ, ARD): [1.512, 1.286]
- signal std (σ_f): 1.107
- noise std (σ_n): 0.103

OPTIMISED HYPERPARAMETERS:

Component 1:
- lengthscales (ℓ, ARD): [102279.322, 0.986]
- signal std (σ_f): 0.71

Component 2:
- lengthscales (ℓ, ARD): [1.447, 100580.863]
- signal std (σ_f): 1.11

Noise std (σ_n): 0.098

LML DECOMPOSITION

LML: 19.219
- Data fit term: -60.5
- Complexity penalty: -190.91
- Constant term: -111.192

LML DECOMPOSITION

LML: 66.408
- Data fit term: -60.501
- Complexity penalty: -238.101
- Constant term: -111.192

Figure 6: Single RBF (left) vs Sum of RBF (right)

References

[1] *GPy.kern.src Package* SheffieldML, 2020 (Accessed 03 Oct 2025). Available at https://gpy.readthedocs.io.