Report coursework 1

4F13 - Probabilistic Machine Learning

November 2, 2021

Candidate no.: J902C

Word count: 999

UNIVERSITY OF CAMBRIDGE
Department of Engineering

Table of contents

1. Question a)

A GP model using SE covariance function is trained using as initial hyperparameters: $l = e^{-1}$, $\nu = 1$ and $\sigma = 1$, where $l :=$ lengthscale, $\nu :=$ amplitude and $\sigma :=$ likelihood function noise:

```
hyp = struct('mean', [], 'cov', [-1 0], 'lik', 0);
```

Hyperparameters can be set by optimizing the negative log marginal likelihood (NLML) using the function `minimize`:

```
minimize(hyp, @gp, -100, @infGaussLik, meanfunc, covfunc, likfunc, x, y);
```

The optimized hyperparameters are: $l = e^{-2.0540} = 0.1282$, $\nu = e^{-0.1087} = 0.8970$ and $\sigma = e^{-2.1385} = 0.1178$, that describe the found local minimum of the NLML.

To make predictions using the optimized hyperparameters the `gp` function can be used, getting as a result the predictive mean and standard deviation at the test points:

```
gp(hyp2, @infGaussLik, meanfunc, covfunc, likfunc, x, y, xs);
```

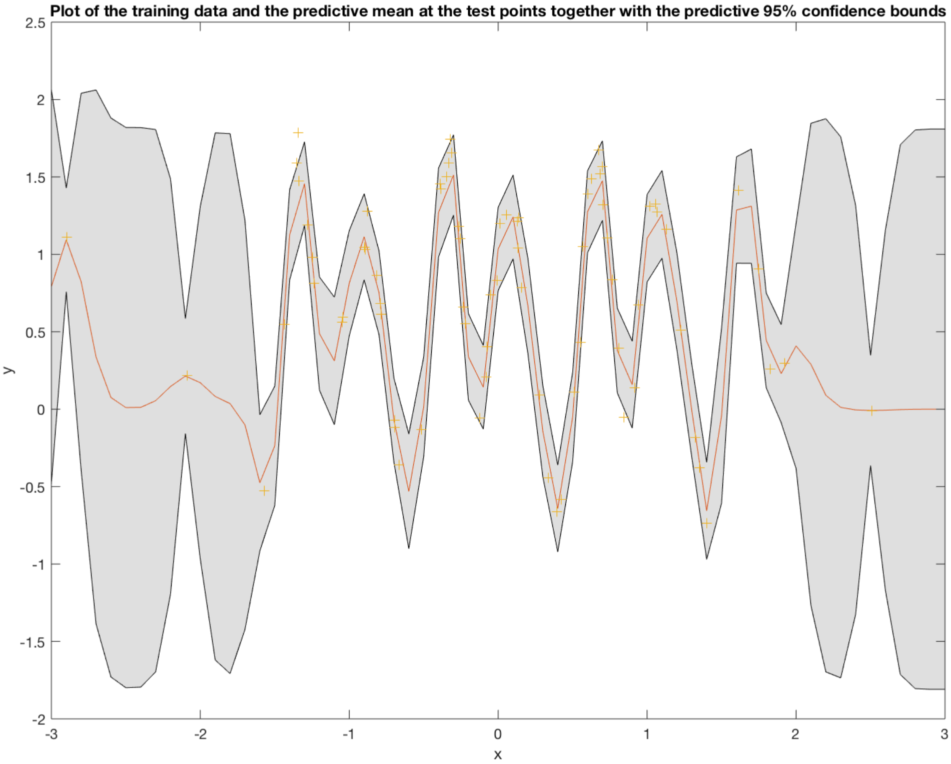

Figure 1.1: Training data, predictive mean and 95% confidence bounds at test points

Plotting the results as it's been done in the figure 1.1 allows us to visualise the data fit, using a evenly spaced test set between in $[-3, 3]$. The GP model fits the training data quite well since the predictive mean is really close to all training points. However, the uncertainty increases when there are no training data nearby.

2. Question b)

Different local optimum for the hyperparameters can be found just by printing the optimized hyperparameters after the `minimize` function. For example, after setting the initial hyperparameters to $l = e^{-1}$, $\nu = 0.7$ and $\sigma = 1$, the optimized hyperparameters are: $l = e^{2.0844} = 8.0398$, $\nu = e^{-0.3625} = 0.6959$, $\sigma = e^{-0.4109} = 0.6631$.
This can also be checked by plotting different data fit for different initial hyperparameters, as shown in figure 2.1.

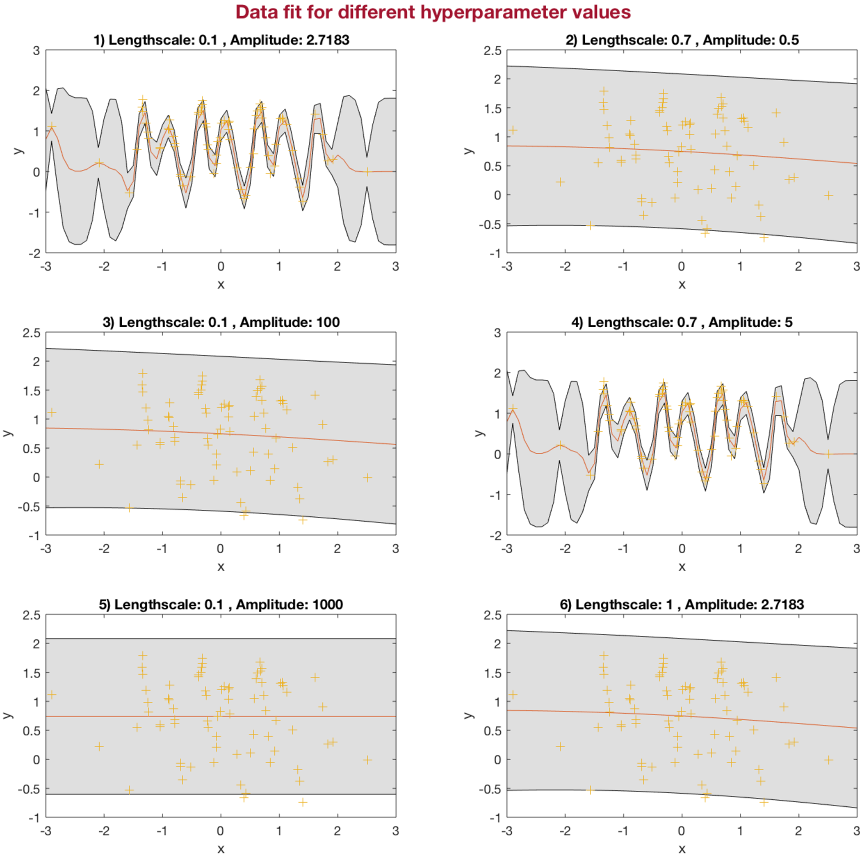

Figure 2.1: Data fit for different initial hyperparameter values

Several models have reached barely the same local optimum for hyperparameters. This can be seen in plots 2.1.1 and 2.1.4, where the initial lengthscale and amplitude are low enough to also find small optimized values, hence the wiggly shape of the data fit. Remember that the lengthscale determines the covariance between near points, so greater the lengthscale, smoother the GP functions, and if the optimized lengthscale is smaller, the GP functions are going to be more wiggly. Similar regarding the amplitude: it determines the average distance away from the function's mean.

We can see in figure 2.1 that if any of the initial hyperparameters is large enough, the optimized hyperparamters tend to be large. When the amplitude is huge (figure 2.1.5), the GP model increases its uncertainty ending up being useless.

Figures 2.1.1 and 2.1.2 can be compared. The first model seems to have better data fit, since the uncertainty has been reduced near data points. However, we have to be careful and avoid overfitting. This could be checked by using cross-validation over a larger dataset.

3. Question c)

Changing the form of the covariance function allows us to fit the data in different ways. In this case, a periodic covariance function is used, that turns out to reduce significantly the uncertainty near training data. This can be seen in figure 3.1, where the GP was initialized with the values $l = p = \nu = \sigma = 1$, and trained to get the optimized hyperparameters $l = e^{0.044} = 1.045, p = e^{-0.0012} = 0.9988, \nu = e^{0.2126} = 1.2369, \sigma = e^{-2.2123} = 0.1094$. The obtained NLML is the lowest one so far for this dataset. This can be explained by the fact that the training points seem to have a periodic behaviour since there are point accumulations around $y = 1.5$ and $y = -0.5$ with period $= 1$.

$$\log(p(\mathbf{y}|\mathbf{x}, \mathcal{M})) = \underbrace{-\frac{1}{2}\mathbf{y}^T \left[\mathbf{K} + \sigma_n^2 \mathbf{I}\right]^{-1} \mathbf{y}}_{\text{data fit term}} \underbrace{- \frac{1}{2}\log(\left|\mathbf{K} + \sigma_n^2 \mathbf{I}\right|)}_{\text{complexity penalty}} - \frac{n}{2}\log(2\pi). \tag{3.1}$$

If our prior assumptions are that the data is truly periodic, then the GP is not overfitting since the expression of the (log) marginal likelihood (3.1) has a complexity penalty term that avoids it. However, this assumption can be wrong and we cannot be 100% sure that the underlying function is truly periodic since we only have 75 training points and we've no information outside the $[-3, 3]$ interval.

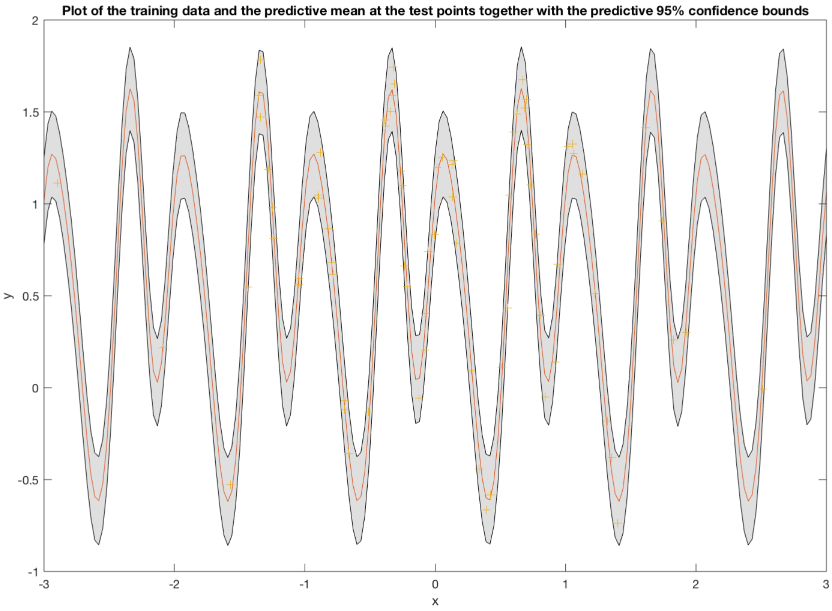

Figure 3.1: Data fit of a GP model using a periodic covariance function (1)

4. Question d)

A composite covariance function is used, formed by the product of a periodic function and the squared exponential (SE) function. The initial hyperparameters are: $l_1 = e^{-0.5} = 0.6065$, $p = \nu_1 = e^0 = 1$, $l_2 = e^2 = 7.3891$ and $\nu = e^0 = 1$. The resulting covariance matrix evaluated at the train data is obtained by using the `feval` function:

```
K = feval(covfunc:, hyp.cov, x);
```

To sample some functions of this GP model we can use the Cholesky decomposition, after adding a small diagonal matrix in order to force K to be positive definite due to rounding errors:

```
y = chol(K+1e-6*eye(nPts))' * gpml_randn(seed, nPts, nSampledFunc);
```

In figure 4.1 we can see a single sampled function, and in the figure 4.2 three sampled functions are plotted.

The periodic covariance function is known for generating periodic functions with a constant amplitude. The SE covariance function tends to generate smoother functions but with no periodicity. When the product of theese two is used the sampled functions have a certain periodicity behaviour but with changing amplitude. This can be easily seen in the figure 4.1. For more details, visit the post [3].

In the 4.2 several sampled functions are plotted, but all of them follow the same pattern described.

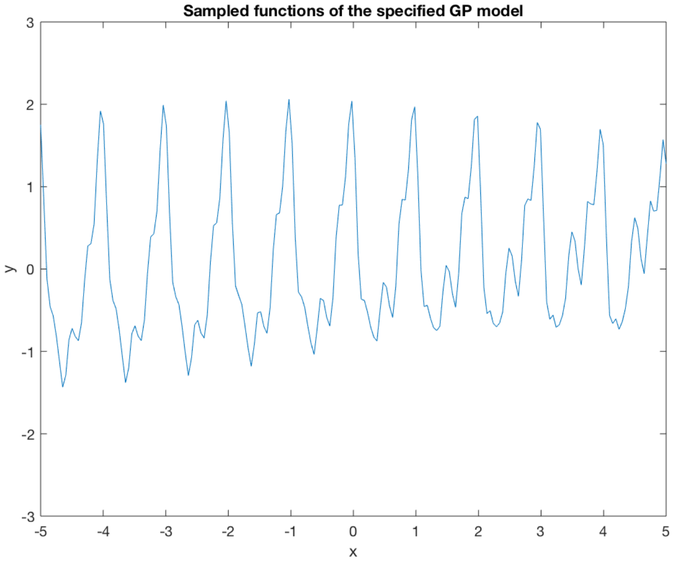

Figure 4.1: One sampled function from the specified GP model

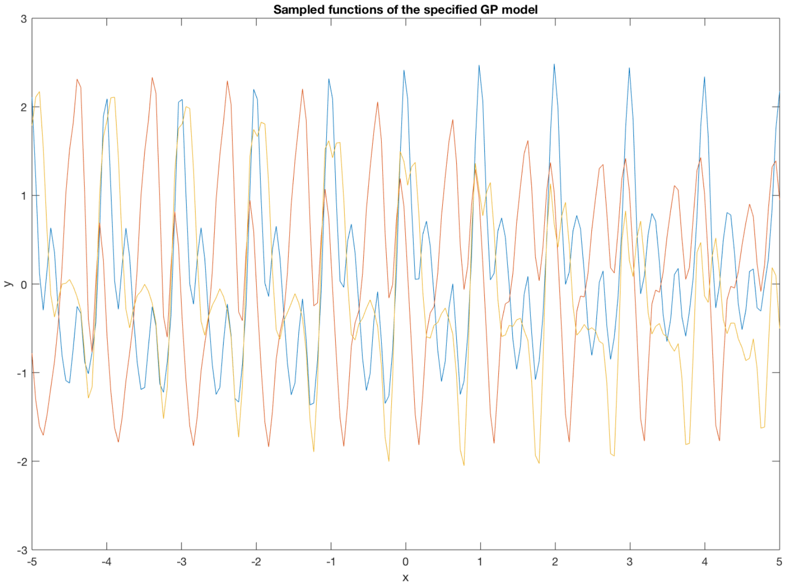

Figure 4.2: Three sampled functions from the specified GP model

5. Question e)

A different dataset is used in this question. Let's visualise the training data (figure 5.1), using the `mesh` function.

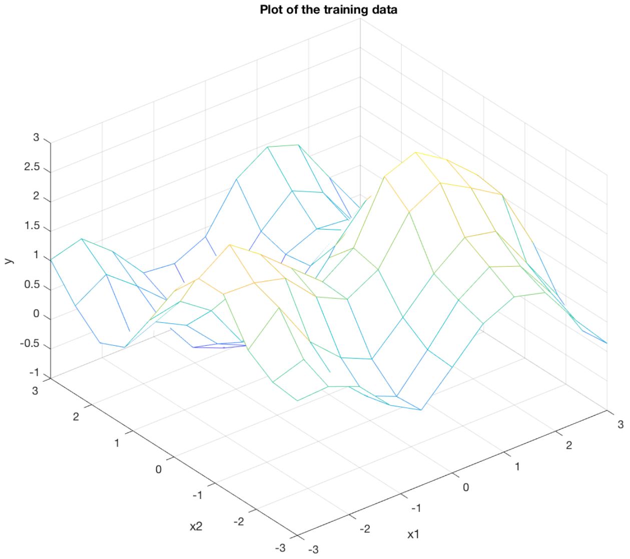

Figure 5.1: Visualising the training data to get a good feel for it

Now, two different GP models are trained, using different covariance functions to define them. The first one uses the automatic relevance determination (ARD) covariance function, and the second one uses the sum of two ARD functions. The first GP model is initialized with all hyperparameters set to 1, and for the second one we make sure to break symmetry (using `hyp.cov = 0.1*randn(6,1);`).

After optimizing the hyperparameters, the value of the NLML is $nlZ = -19.2187$, and $nlZ = -66.3427$ for the second GP model, so it seems the second model is preferable. Anyway, let's keep analyzing both models.

In the figure 5.2, the surface representing the training data (green) and the predictive mean surface (purple) are plotted for both GP models. Both seem to generate almost the same solution since the surface described by the predictive mean is very close to the surface described by the training points.

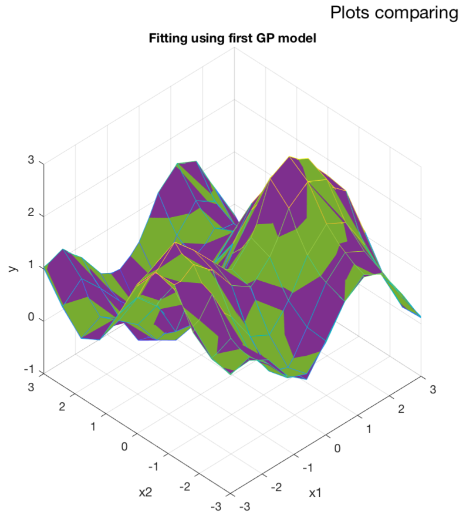

Figure 5.2: Comparing the data fit for both GP models

Finally, in figure 5.3 the model's uncertainty is represented by plotting two additional surfaces, one for the upper confidence bound, and another for the lower one. Additionally, test points are sampled in the 2-D interval $[-5, 5] \times [-5, 5]$ to try to capture the uncertainty far away from training points. It can be seen that despite reaching the same solution (in terms of predictive mean), the second GP has less uncertainty over the points far away from the training data. In summary, the second GP model is preferable since it's simpler (it explains well less datasets) and it has lower NLML.

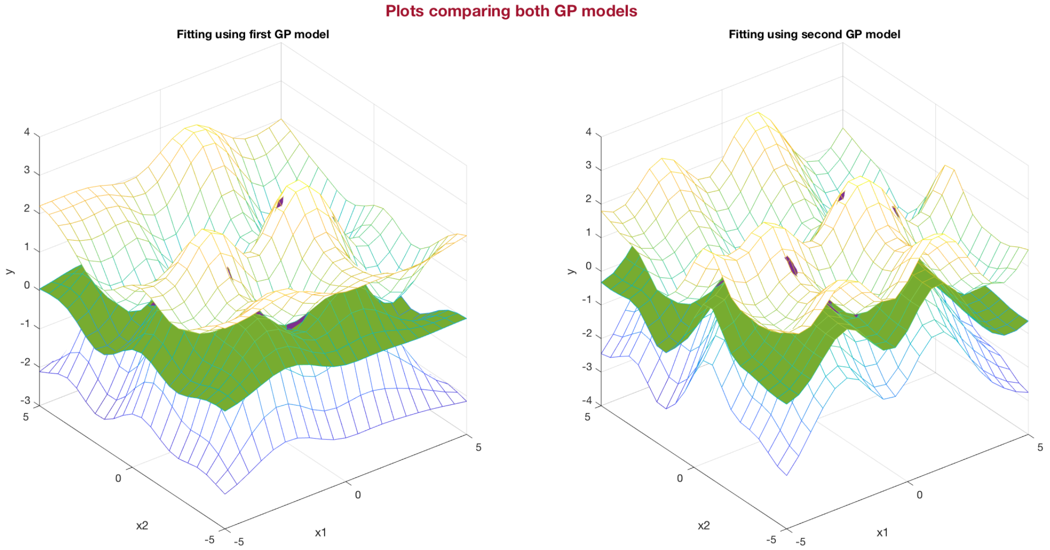

Figure 5.3: Comparing the data fit for both GP including 95% confidence surfaces

Bibliography

[1] Documentation for GPML Matlab Code (June 2018). Carl Edward Rasmussen, Hannes Nickisch, et.al. Version 4.2 http://www.gaussianprocess.org/gpml/code/matlab/doc/

[2] Carl Edward Rasmussen and Christopher K. I. Williams. 2005. *Gaussian Processes for Machine Learning*. The MIT Press.

[3] Peter Roelants. Gaussian processes (3/3) - exploring kernels. Github page. https://peterroelants.github.io/posts/gaussian-process-kernels/.