Imperial College London

COURSEWORK # 3

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Mathematics for Machine Learning

Author:
Alexandre Allani (CID: 01797836)

Date: November 8, 2019

1 Question 1

1.1 Question a

We want to compute maximum of likelihood based on :

$$P(\boldsymbol{y}|X) = \prod_{i=1}^{N} p(y_i|x_i) \text{ and } y_i \ \mathcal{N}(\boldsymbol{w}^T\Phi(x_i), \sigma^2)$$

The log-likelihood would be then:

$$\mathcal{L}(\boldsymbol{w}) = log(\prod_{i=1}^{N} \mathcal{N}(\boldsymbol{w}^T\Phi(x_i), \sigma^2)) = \sum_{i=1}^{N} log(\mathcal{N}(\boldsymbol{w}^T\Phi(x_i), \sigma^2)) \tag{1}$$

$$= constant - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \boldsymbol{w}^T\Phi(x_i))^2 \tag{2}$$

We can recognize the L_2 norm, hence :

$$L(\boldsymbol{w}) = constant - \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{\Phi w})^T(\boldsymbol{y} - \boldsymbol{\Phi w}) \tag{3}$$

$$= constant - -\frac{1}{2\sigma^2}(\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{\Phi w} - \boldsymbol{w}^T\boldsymbol{\Phi y} + \boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi w}) \tag{4}$$

The maximum of likelihood is also the maximum of log likelihood, since log is a concave function. We are going to compute the gradient to find the maximum:

$$\frac{d\mathcal{L}}{d\boldsymbol{w}}(\boldsymbol{w}) = -\boldsymbol{y}^T\boldsymbol{\Phi} - \boldsymbol{y}^T\boldsymbol{\Phi} + \boldsymbol{w}^T(\boldsymbol{\Phi\Phi}^T + \boldsymbol{\Phi}^T\boldsymbol{\Phi}) \tag{5}$$

$$= -2\boldsymbol{y}^T\boldsymbol{\Phi} + 2\boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi} \tag{6}$$

The maximum if found for :

$$\frac{d\mathcal{L}}{d\boldsymbol{w}}(\boldsymbol{w}) = \boldsymbol{0}^T \tag{7}$$

$$\iff -2\boldsymbol{y}^T\boldsymbol{\Phi} + 2\boldsymbol{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi} = \boldsymbol{0}^T \tag{8}$$

$$\iff \boldsymbol{w}^T = \boldsymbol{y}^T\boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1} \tag{9}$$

$$\iff \boldsymbol{w} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{y} \tag{10}$$

The vectore $\boldsymbol{w} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{y}$ is the parameter maximising the likelihood

Using the result the code python joined to this file I obtained the graph on **Figure 1**

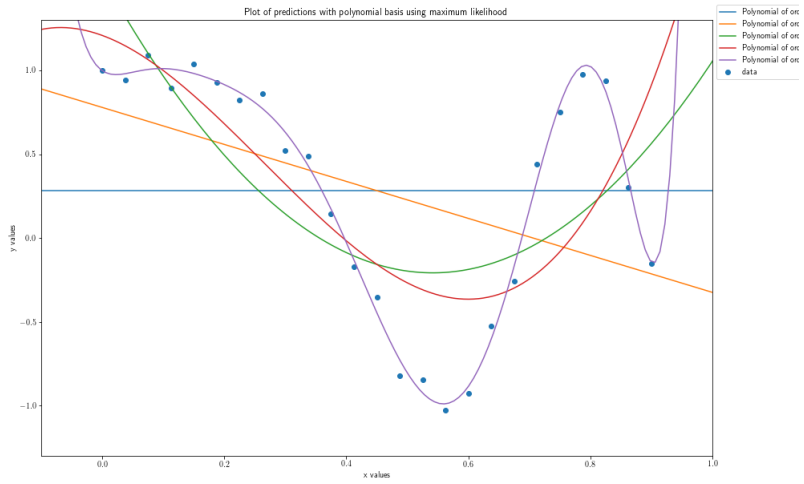

Figure 1: Shape prediction according to a polynomial basis

1.2 Question b

Using the maximum likelihood with a different basis, we obtain the graph on **Figure 2**

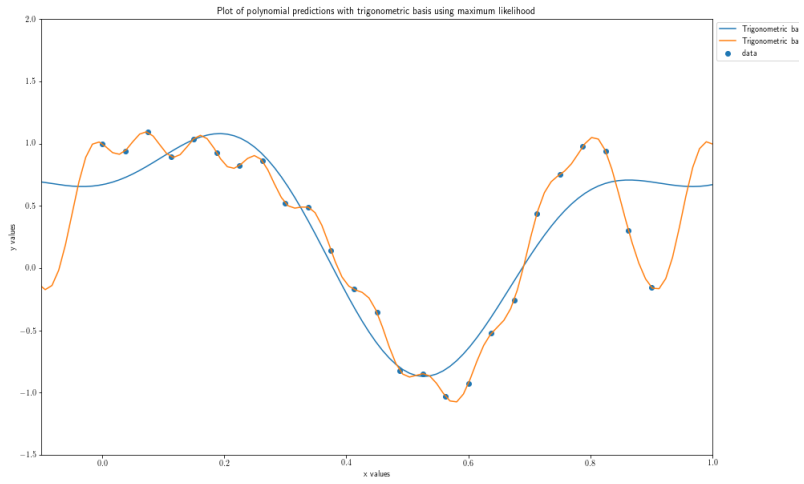

Figure 2: Shape prediction according to a trigonometric basis

1.3 Question c

Using the code joined to this document, I implemented a class model that does a leave one out cross validation and compute the average test error and train error. With this we obtain the plot on **Figure 3**

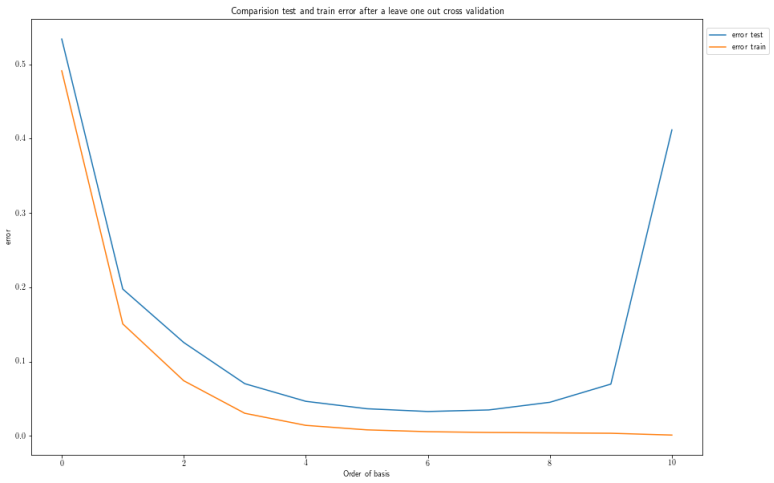

Figure 3: Train and test error with polynomial basis

1.4 Question d

Over-fitting is a concept related to how well a trained model is capable to generalize on new data. When the model predicts with high precision on data he has trained on, but poorly predict on data he has never seen, this means that the model over-fits the data he trained on. It doesn't generalize, it learns "by heart" the data.
Our dataset was constructed so that: $\forall i \in [1, 25]\ y_i = cos(10x_i^2) + 0.1sin(100x_i)$. Clearly, this is not a polynomial relationship. With the polynomial basis, we want the model to approximate this function. As we can see from the graph, σ_{ML} is decreasing when the order is increasing. On the other hand, the test error is decreasing until order 6 and the is increasing again.
This is explained by the fact that maximum likelihood estimation tends to over-fit the data. Indeed, we are "guessing" the best parameter that will fit the data we have. And with a degree high enough, we can indeed find a parameter $\boldsymbol{w}$ that will make the prediction go through each point (since for a set of N points, a polynomial of degree N can interpolate each point). However this is not what we want, because first, the data are usually noisy, and second, this poorly generalizes. **Figure 3** gives us the hint that we should train only with a polynomial basis of order 5 to 6.
The **Figure 2** shows also a good example of over-fitting with the orange curve. It goes indeed through all the points, but won't generalize well. In **Figure 1**, we have an example of under-fitting (with low-degree polynomial basis) and over-fitting with the purple line (order 11).

This explains why the test error is actually increasing when the order is too high when the training error keeps getting lower. This is why we study then Bayesian linear regression which can't over-fit because we average out the parameter thanks to Bayes Theorem.

2 Question 2

2.1 Question a

We have the following joint distribution:

$$P(\boldsymbol{y}, \boldsymbol{w}|\boldsymbol{x}, \alpha, \beta) = \left(\prod_{i=1}^{N} \mathcal{N}(\boldsymbol{w}^T \phi(x_i), \beta)\right) \mathcal{N}(\boldsymbol{w}|0, \alpha \boldsymbol{I})$$

$$\boldsymbol{y} = \boldsymbol{w}^T \boldsymbol{\Phi} + \epsilon \text{ where } \epsilon \text{ is a noice with variance } \beta$$

With Bayes Theorem:

$$P(\boldsymbol{y}, \boldsymbol{w}|\boldsymbol{x}, \alpha, \beta) P(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{x}, \alpha, \beta) * p(\boldsymbol{w}|\alpha)$$

Hence the marginal likelihood is given by:

$$P(\boldsymbol{y}|\boldsymbol{x}, \alpha, \beta) = \int P(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{x}, \alpha, \beta) * p(\boldsymbol{w}|\alpha) d\boldsymbol{w}$$

We know, that the product of two Gaussians stays Gaussian, and the same rules apply for the sum of two Gaussians. A Gaussian is characterized by its mean and variance. Hence:

$$\begin{aligned}
E(\boldsymbol{y}) =& E(vec w^T \boldsymbol{\Phi} + \epsilon) && (11)\\
=& \boldsymbol{\Phi} E(\boldsymbol{w}) + E(\epsilon) && (12)\\
=& 0 && (13)\\
V(\boldsymbol{y}) =& V(\boldsymbol{w}^T \boldsymbol{\Phi} + \epsilon) && (14)\\
=& \boldsymbol{\Phi} Var(\boldsymbol{w}) \boldsymbol{\Phi}^T + \beta \boldsymbol{I} && (15)\\
=& \alpha \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \beta \boldsymbol{I} && (16)
\end{aligned}$$

The marginal likelihood is a gaussian:

$$\mathcal{N}(0, \alpha \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \beta \boldsymbol{I})$$

The log marginal likelihood is:

$$\mathcal{L}_m(\alpha, \beta)) - 1/2 \left[N log(2\pi) + log(det(\alpha \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \beta \boldsymbol{I})) + y^T (\alpha \boldsymbol{\Phi} \boldsymbol{\Phi}^T + \beta \boldsymbol{I})^{-1} y \right]$$

We know then that the gradient with respect to $[\alpha, \beta]$is of dimension 2x1:

$$\underset{[\alpha,\beta]}{\text{grad}}\mathcal{L}_m = \begin{bmatrix} \frac{d\mathcal{L}_m}{d\alpha} & \frac{d\mathcal{L}_m}{d\beta} \end{bmatrix}$$

Let: $C = \alpha \mathbf{\Phi}\mathbf{\Phi}^T + \beta \boldsymbol{I}$

We have:

$$\frac{d\mathcal{L}_m}{d\alpha} = -\frac{1}{2}\left(tr(C^{-1}\mathbf{\Phi}\mathbf{\Phi}^T) - \boldsymbol{y}^T C^{-1}\mathbf{\Phi}\mathbf{\Phi}^T C^{-1}\boldsymbol{y}\right) \tag{17}$$

$$\frac{d\mathcal{L}_m}{d\beta} = -\frac{1}{2}\left(tr(C^{-1}) - \boldsymbol{y}^T C^{-1} C^{-1}\boldsymbol{y}\right) \tag{18}$$

The code joined to this document gives the implementation of the log marginal likelihood and its gradient.

2.2 Question b

For a gradient descent of 2000 iterations, a learning rate of 0.01 and a starting point $[0.5, 0.57]$, **Figure 4** is obtained:

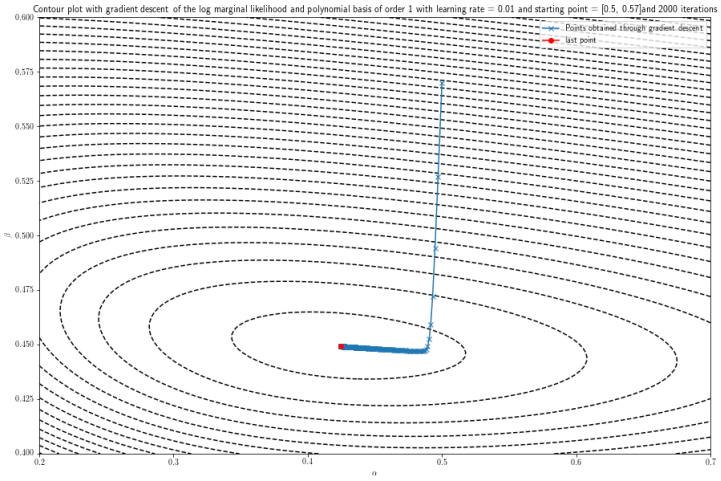

Figure 4: Gradient descent on marginal likelihood with polynomial basis of order 1

The gradient descent returns the following values:

$$\alpha \approx 0.4246$$
$$\beta \approx 0.4492$$

2.3 Question c

In order to choose a good starting point, a good learning rate and a good number of iterations, I plotted some of them for the degree 1 and degree 12 to see if it converges. **Figure 5** and **Figure 6** shows the gradient descent for the best setup that I chose.
However, I have seen during the runs, that for degree 11 and 12, the learning rate was still too big, making the β value negative. After this point, the gradient descent makes no sense, since β can't be negative (otherwise the log marginal likelihood is not defined). I added some code to the gradient descent as you can see below, so that if β is negative, I divide the learning rate by a factor 100, and "restart" from the previous point.

```
def grad_desc(grad, step, e, start, *argv):
    mn = np.asmatrix(start)
    record = [np.copy(mn)]
    for _ in range(step):
        mn += e * grad(mn[0, 0], mn[0, 1], *argv)
        if mn[0, 1] < 0:
            mn[0, 1] = record[-1][0, 1]
            e /= 100
        record.append(np.copy(mn))
    return mn, record
```

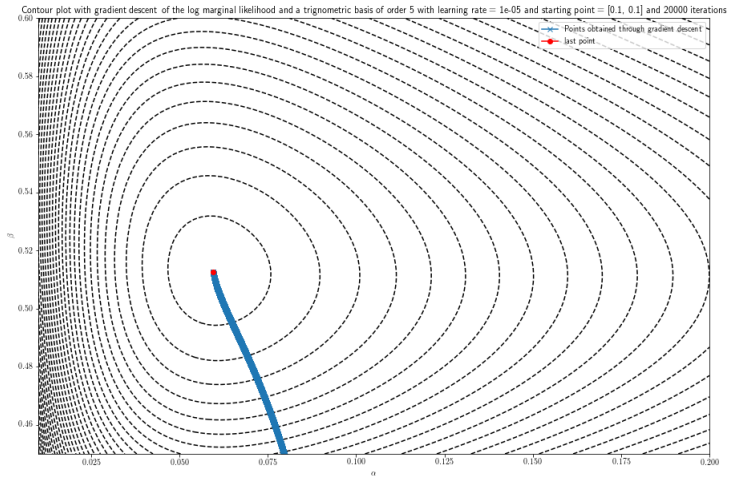

Figure 5: Gradient descent on marginal likelihood with trigonometric basis of order 0

I got the following plot (**Figure 7**) after computing the maximum for the twelve orders:
As expected with cross-validation, the maximum likelihood is bigger for orders 4 to 6. Then it goes down since the degree of the basis is too high. The approach of cross-validation gives us a good way to see when we tend to over-fit as explained in

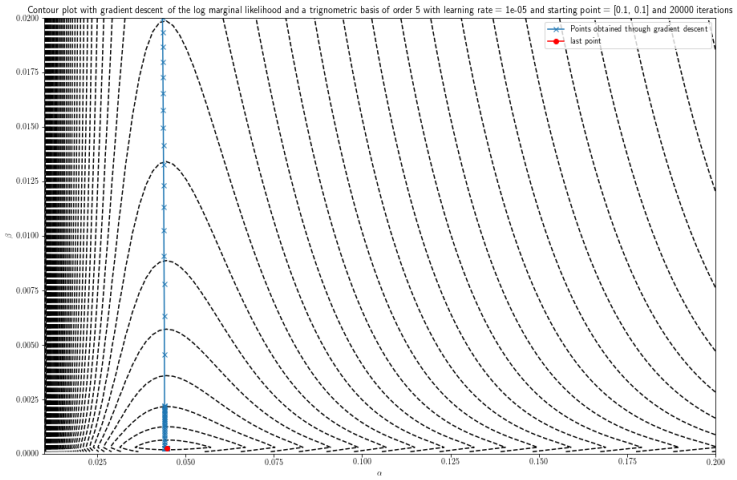

Figure 6: Gradient descent on marginal likelihood with trigonometric basis of order 0

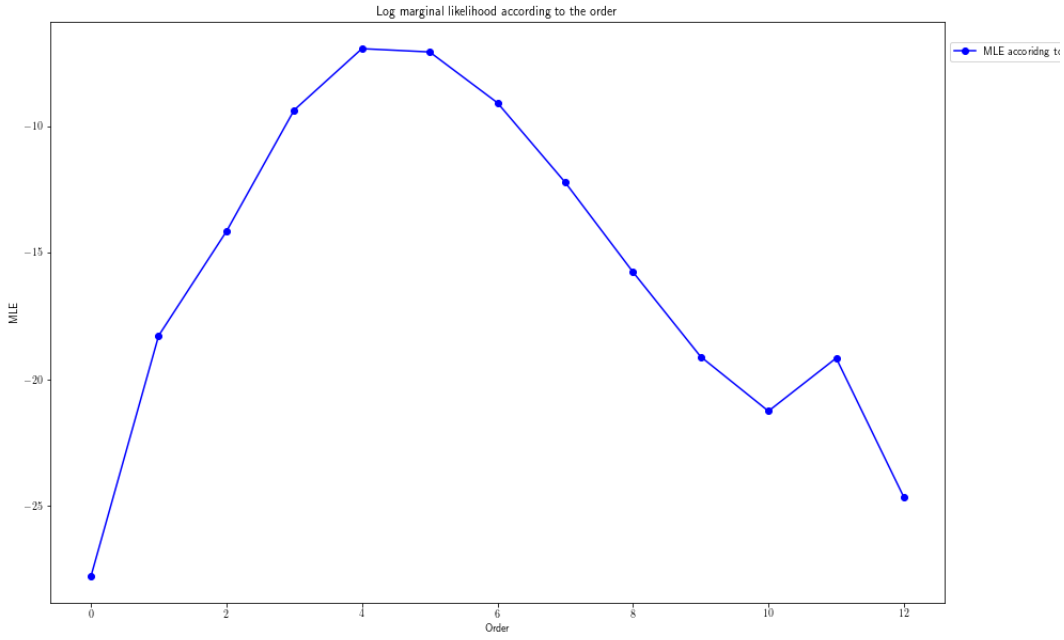

Figure 7: Maximum of log likelihood with respect to basis' order

Question 1.c. However, with the Bayesian approach, we can't over-fit since we are not optimizing a parameter.
Both approaches allow us to determine which basis is the most suited the problem.

2.4 question d

I obtain this plot when using a biased Gaussian basis:

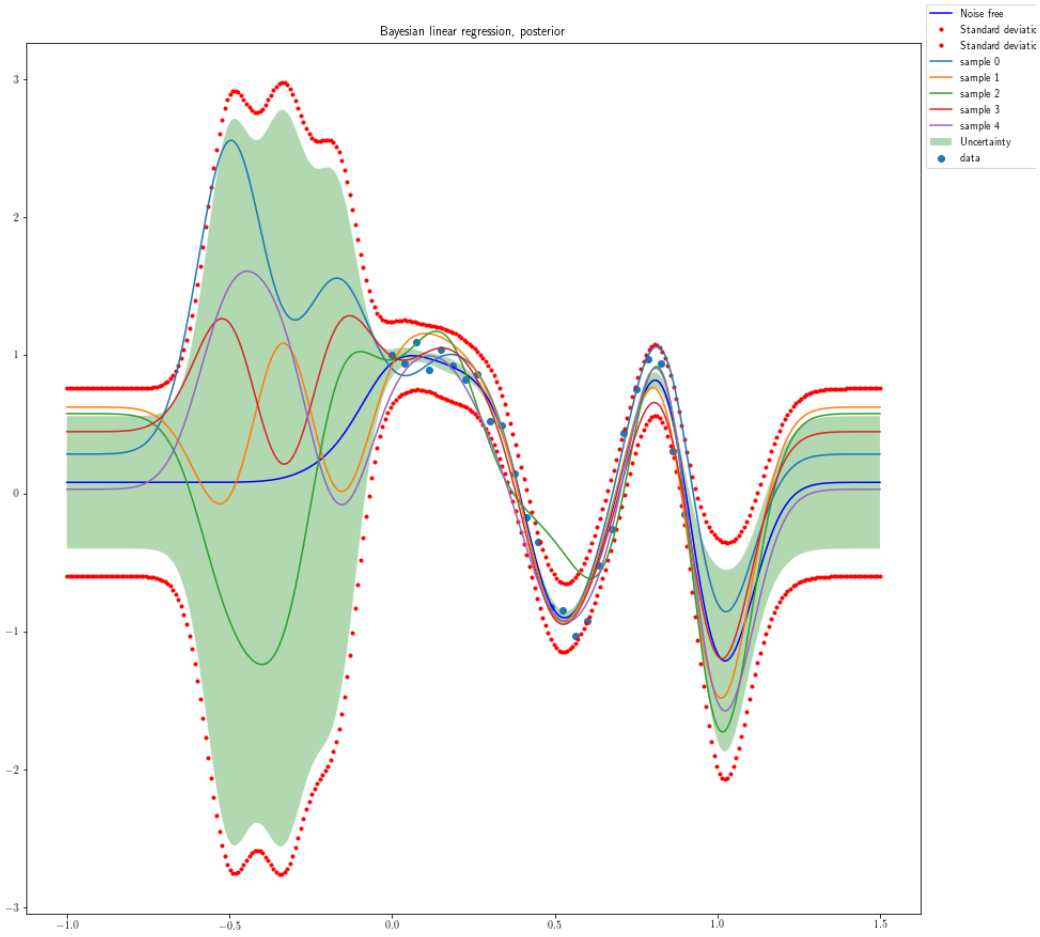

Figure 8: Maximum of log likelihood with respect to basis' order

The next figure is the same prediction, but with unbiased Gaussian basis:

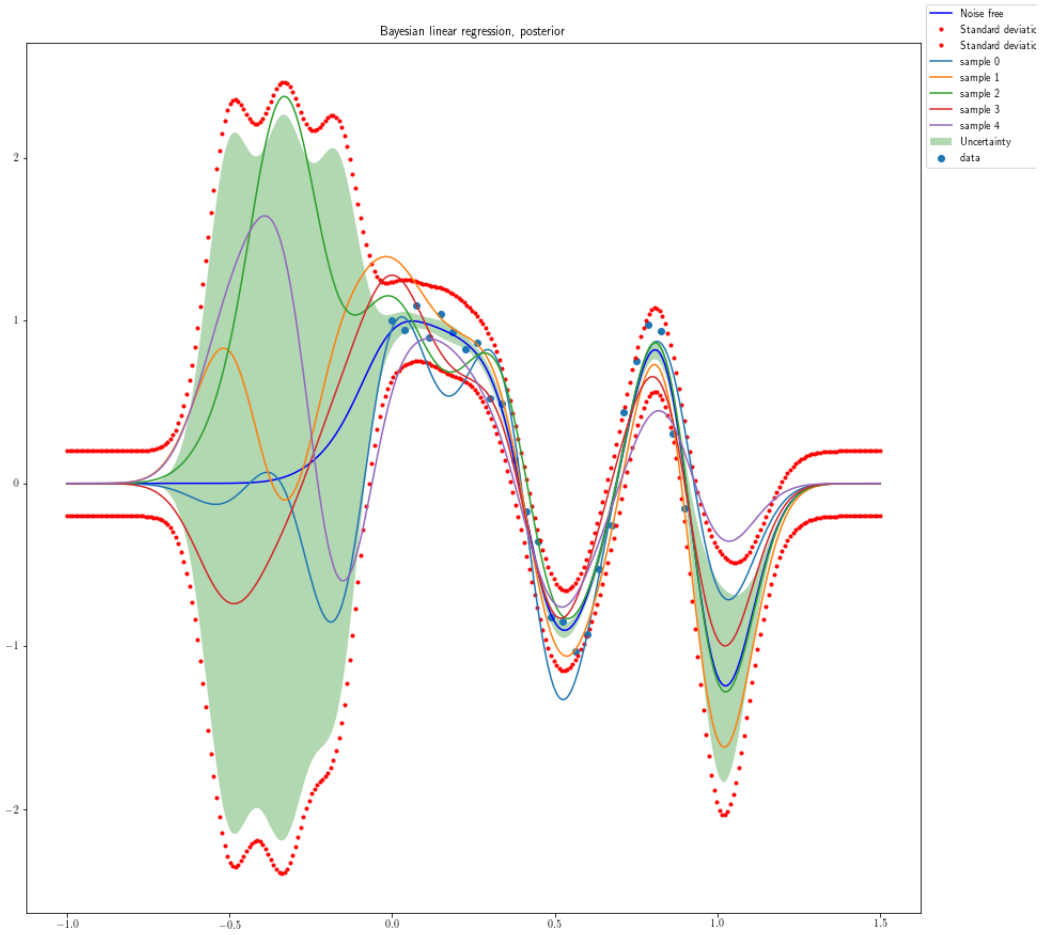

Figure 9: Maximum of log likelihood with respect to basis' order

As expected, when we are far from the basis of Gaussian and the data, the model doesn't know what to predict explaining this shape. Also when we compare the noise-free prediction, with the samples, it doesn't over-fit as expected