Gaussian process regression exercise

Miguel Lázaro Gredilla

December 6, 2013

Introduction

In this exercise the student will review several key concepts of Gaussian process regression. In particular, weight-space regression, function-space regression and the model selection problem will be considered

For the purpose of this exercise, the regression model is

$$y(\mathbf{x}_i) = f(\mathbf{x}_i) + \varepsilon_i$$

where $y(\mathbf{x}_i)$ is the output corresponding to input $\mathbf{x}_i$, $f(\mathbf{x}_i)$ is the unobservable latent function and ε_i is white zero-mean Gaussian noise, i.e., $\varepsilon_i \sim \mathcal{N}(0, \sigma_n^2)$.

Latent function $f(\mathbf{x})$ will be modeled a priori as a Gaussian process.

$$f(\mathbf{x}) \sim \mathcal{GP}(m_f, k_f(\mathbf{x}_i, \mathbf{x}_j))$$

with constant mean m_f and covariance function (kernel) $k_f(\mathbf{x}_i, \mathbf{x}_j)$. Since ε_i is white Gaussian noise, this will imply a Gaussian process model on the observed outputs

$$y(\mathbf{x}) \sim \mathcal{GP}(m_f, k_f(\mathbf{x}_i, \mathbf{x}_j) + \sigma_n^2 \delta_{ij})$$

where δ_{ij} is a Kronecker delta, i.e., has value 1 if $i = j$ and 0 otherwise.

Some practical considerations regarding efficient computation in MatLAB that can be useful for the solution of this and other exercises follow.

- Though sometimes unavoidable, it is recommended not to use explicit matrix inversion. if an operation like $\mathbf{A}^{-1}\mathbf{B}$ must be performed, it is preferable to code it using the backslash operator `A\B`, which computes the result of the whole operation (inversion and product) as a single step without explicitly computing the inverse. If matrix $\mathbf{A}$ is positive definite, it is even better to use `L\(L'\B)`, where $\mathbf{L}$ is Cholesky's decomposition of $\mathbf{A}$ (i.e., $\mathbf{A} = \mathbf{L}^\top\mathbf{L}$).

- Sometimes, the computation of $\log|\mathbf{A}|$ (where $\mathbf{A}$ is a positive definite matrix) can overflow available precision, producing incorrect results. A numerically more stable alternative, providing the same result is $2\sum_i \log([\mathbf{L}]_{ii})$, where $\mathbf{L}$ is the Cholesky decomposition of $\mathbf{A}$ (i.e., $\mathbf{A} = \mathbf{L}^\top\mathbf{L}$).

- Non-degenerate covariance matrices, such as the ones in this exercise, are always positive definite.

- It may happen, as a consequence of chained rounding errors that a matrix, which was mathematically expected to be positive definite, turns out not to be so numerically in MatLAB. This implies its Cholesky decomposition won't be available. A quick way to palliate this problem is by adding a small number (such as 10^{-6}) to the diagonal of such matrix. This is equivalent to adding white noise of negligible power (known as "jitter noise") to the associated random vector. It should be reasoned, on a case per case basis,

the interpretation of such a step in our computations and decide whether it is appropriate to do it, or instead some of the previous steps should be stabilized to avoid losing the positive definiteness of the matrix. In this exercise, jitter noise can be needed in the random function generation from part 2.c), but should be avoided in all remaining cases.

- To guarantee the exact reproducibility of the experiments, it may be useful to start your code with the line `randn('seed',0);rand('seed',0);`, so that you can compare your results with the ones presented here.
- To compute all the pairwise distances between row vectors of matrix `X`, `pdist2(X,X)` can be used.
- Unless otherwise specified, every vector is a column.
- Notation $[\mathbf{v}]_i$ referes to the i-th element of a vector, whereas $[\mathbf{A}]_{ij}$ refers to the element at row i, column j within matrix $\mathbf{A}$.
- For convenience $\mathbf{y}$ is often used to refer to the vector that contains all training outputs, $\mathbf{X}$ to the matrix that contains all training inputs (one per row) and $\mathbf{X}_*$ to the matrix containing all test inputs (one per row).

1 Linear regression (weight space view)

First, we are going to generate synthetic data (so that we have the ground-truth model) and use them to make sure everything works correctly and our estimation are sensible. We will work with real data (whose ground-truth is unknown) afterwards.

1.a) Set parameters $\sigma_0^2 = 2$ and $\sigma_n^2 = 0.2$. Generate the weight vector `true_w` (two elements) from pdf $\mathcal{N}(\mathbf{0}, \sigma_0^2\mathbf{I})$. This vector determines the regression line that we want to find.

Generate an input matrix `X` containing the constant term 1 in all elements of the first column and values between 0 and 2 (included), with a 0.1 step, in the second column.

Finally, generate the output vector `y` as the product `X*true_w` plus Gaussian noise of pdf $\mathcal{N}(0, \sigma_n^2)$ at each element.

Plot generated data. You will notice a linear behavior, but the presence of noise makes it hard to estimate precisely the original straight line that generated them (which is stored in `true_w`).

1.b) Let's see to which extent it is possible to determine the original straight line from observed data. Knowing that the generative model is linear, i.e., the latent function is $f(\mathbf{x}) = \mathbf{x}^\top\mathbf{w}$ and knowing the prior pdf of weights $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \sigma_0^2\mathbf{I})$ and noise $p(\varepsilon_i) = \mathcal{N}(0, \sigma_n^2)$, compute the posterior pdf of the weights, $p(\mathbf{w}|\mathbf{X}, \mathbf{y})$. The result is:

```
true_w =
    1.6475
    0.8865

mean_w =
    1.8689
    0.6501

cov_w =
    0.0346   -0.0252
   -0.0252    0.0253
```

1.c) Compute the confidence ranges of your estimations of the elements of $\mathbf{w}$, at 95.45%. I.e., find the shorter ranges for $[\mathbf{w}]_1$ and $[\mathbf{w}]_2$ that contain the true weight with 95.45% probability. The result is:

```
range_w1 =
    1.4971    2.2407

range_w2 =
    0.3319    0.9684
```

As you can see, the true value of $\mathbf{w}$ is contained within the range. If you increase the number of available data (now only 21 are available), for instance reducing the sampling step from 0.1 to 0.05, you will see how the confidence ranges are reduced, thus increasing accuracy.

1.d) Plot now samples from the posterior distribution of the function space. To this end, generate random vectors $\mathbf{w}_s$ with $s = 1, \ldots, 50$ from the posterior density of the weights, $p(\mathbf{w}|\mathbf{X}, \mathbf{y})$, and use them to generate 50 straight lines, $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}_s$ with x between -1 and 3, with step 0.1 (remember that $\mathbf{x} = [1 \;\; x]$).

Plot the original ground-truth straight line, corresponding to `true_w`, along with the 50 generated straight lines and the original samples, all in the same plot, as depicted in Fig. 1.

The Bayesian model isn't providing a single answer, but instead a density over them, from which we have extracted 50 options.

1.e) On top of the previous figure and in the same range (-1 to 3, step 0.1) plot functions $\mathbb{E}[f(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*]$ and $\mathbb{E}[f(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*] \pm 2\sqrt{\mathbb{V}[f(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*]}$ (i.e., the posterior mean of $f(\mathbf{x})$, as well as two standard deviations above and below). You will notice that the random lines from previous step fall inside the designated region with 95.45% probability.

Plot now $\mathbb{E}[y(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*] \pm 2\sqrt{\mathbb{V}[y(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*]}$ ($\mathbb{E}[y(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*] = \mathbb{E}[f(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*]$, so there is no need to plot it again) and notice that 95.45% of observed data lie now within the newly designated region. These new limits establish a confidence range for our predictions. See how the uncertainty grows as we move away from the interpolation region to the extrapolation areas (see Fig. 1).

1.f) Once our code has been tested on synthetic data, we will use it with real data. Load and properly normalize data corresponding to the evolution of the stocks of 10 airline companies[1].

```
load DatosLabReg Xtrain Ytrain Xtest Ytest
[N, D] = size(Xtrain); D = D + 1; Ntest = size(Xtest,1);
mx = mean(Xtrain,1); stdx = std(Xtrain,1,1);
X = [ones(N,1) (Xtrain - ones(N,1)*mx)./(ones(N,1)*stdx)];
Xtest = [ones(Ntest,1) (Xtest - ones(Ntest,1)*mx)./(ones(Ntest,1)*stdx)];
y = Ytrain;
```

After running this code, you will have inside matrix `X` an initial column of ones and the evolution of (normalized) price for 9 airlines, whereas vector `y` will contain a single column with the price evolution of the tenth airline. The objective of the regression task is to estimate the price of the tenth airline from the prices of the other nine.

[1]Data are an adaptation of the *Stock dataset* from `http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html`, which in turn was taken from the StatLib Repository, `http://lib.stat.cmu.edu/`.

1.g) To this end, we are going to use the linear model that we already got working with synthetic data. Since the values of hyperparameters σ_0^2 and σ_n^2 are no longer known, use a gross estimation (we will soon see how to estimate these values in a principled way)

```
s02 = mean((X\y).^2);   % Gross estimation of s02 and sn2
sn2 = 2*mean((y-X*(X\y)).^2);
```

and repeat parts 1.b) and 1.c) with these synthetic data. Instead of two weights there will now be 10. The resulting posterior is:

```
mean_w =
   47.0582
    5.0041
    2.2381
    0.1528
   -1.2132
    1.3502
   -3.1205
    1.0843
    0.8576
    2.2421

cov_w =
  0.01 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00  0.00  0.00
 -0.00  0.16  0.01  0.03 -0.00  0.00  0.03  0.01 -0.12 -0.02
 -0.00  0.01  0.21 -0.05 -0.01 -0.07 -0.07 -0.04  0.05 -0.08
 -0.00  0.03 -0.05  0.06  0.01  0.00 -0.02  0.01 -0.03  0.02
 -0.00 -0.00 -0.01  0.01  0.10  0.02 -0.03 -0.07 -0.02 -0.01
 -0.00  0.00 -0.07  0.01  0.02  0.06  0.00 -0.00 -0.01  0.02
 -0.00  0.03 -0.01 -0.02 -0.03  0.00  0.06  0.01 -0.01  0.00
 -0.00  0.00 -0.04  0.01 -0.07 -0.00  0.01  0.09 -0.03  0.03
  0.00 -0.12  0.05 -0.03 -0.02 -0.01 -0.01 -0.03  0.15  0.00
  0.00 -0.02 -0.09  0.02 -0.01  0.02  0.00  0.03  0.00  0.05
```

and the 95.45% confidence intervals are:

```
range_w0 =
   46.8235   47.2929
range_w1 =
    4.2012    5.8071
range_w2 =
    1.3177    3.1584
range_w3 =
   -0.3389    0.6446
range_w4 =
   -1.8345   -0.5919
range_w5 =
    0.8618    1.8386
range_w6 =
   -3.5970   -2.6441
range_w7 =
    0.4808    1.6879
range_w8 =
    0.0943    1.6208
range_w9 =
    1.7964    2.6877
```

Recall that these confidence intervals are just as good as the hypotheses they come from. In this case, the fundamental hypothesis is that the price of the tenth airline is a linear

combination of the prices of the remaining airlines plus a bias a term and a certain amount of zero-mean Gaussian noise. This linear model is probably too simple to be realistic.

1.h) In order to verify the performance of the resulting model, compute the posterior mean and variance of each of the test outputs from the posterior over $\mathbf{w}$. I.e, compute $\mathbb{E}[y(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*]$ and $\mathbb{V}[y(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \mathbf{x}_*]$ for each test sample $\mathbf{x}_*$ contained in each row of `Xtest`. Be sure not to use the outputs `Ytest` at any point during this process.

Store the predictive mean and variance of all test samples in two vectors called `m_y` and `v_y`, respectively. Then compute the mean square error (MSE) and the negative log-predictive density (NLPD) with the following code.

```
MSE = mean((Ytest-m_y).^2)
NLPD = 0.5*mean((Ytest-m_y).^2./v_y+0.5*log(2*pi*v_y))
```

Result should be:

```
MSE =
    6.1135
NLPD =
    1.3376
```

These two measures reveal the quality of our predictor (with lower values revealing higher quality). The first measure (MSE) only compares the predictive mean with the actual value and always has a positive value (if zero was reached, it would mean a perfect prediction). It doesn't take into account predictive variance. The second measure (NLPD) takes into account both the deviation and the predictive variance (uncertainty) to measure the quality of the probabilistic prediction (a high error in a prediction that was already known to have high variance has a smaller penalty, but also, announcing a high variance when the prediction error is small won't award such a good score).

2 Linear and non-linear regression (function space view)

2.a) Instead of considering the explicit model $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ as we did in the previous section, let's now consider that $f(\mathbf{x}) \sim \mathcal{GP}(\mathbf{0}, k_f(\mathbf{x}_i, \mathbf{x}_j))$, with linear covariance function $k_f(\mathbf{x}_i, \mathbf{x}_j) = \sigma_0^2 \mathbf{x}^\top \mathbf{x}$.

Using this view and the predictive equations based on the kernel matrices, repeat part 1.h). Note that there is no need to use the weight vector or infer its posterior pdf at any point. The obtained results in terms of MSE and NLPD must be identical to the ones you got in part 1.h). Since results are identical, is there any advantage or disadvantage to obtaining them this way, as opposed to the process followed in part 1.h)?

2.b) Repeat the previous process but using now a non-linear covariance (kernel). I.e., use as model of the latent function $f(\mathbf{x}) \sim \mathcal{GP}(m_f, k_f(\mathbf{x}_i, \mathbf{x}_j))$, where m_f is the sample mean of the training outputs and the covariance function is Ornstein-Uhlenbeck, defined as

$$k_f(\mathbf{x}_i, \mathbf{x}_j)) = \sigma_0^2 \exp(-||\mathbf{x}_i - \mathbf{x}_j||_2/\ell)$$

Use the following gross estimation for the hyperparameters:

```
s02 = var(y,1);   % Gross estimation of s02, ell, and sn2
sn2 = s02/10;
ell = 8;
```

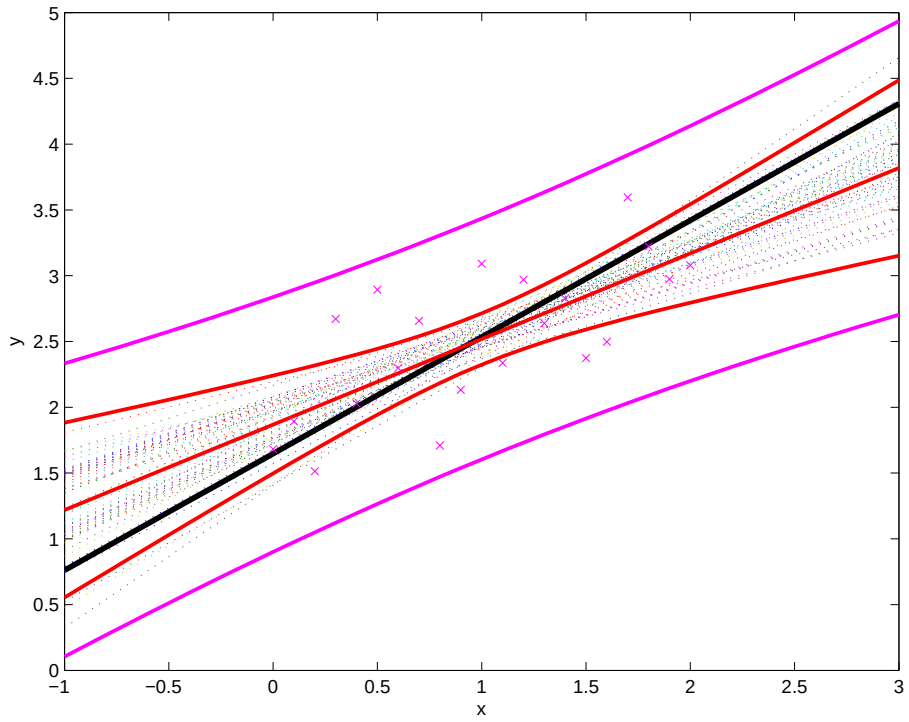

Figure 1: $\mathbb{E}[y(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*] \pm 2\sqrt{\mathbb{V}[y(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]}$ and data, in magenta. $\mathbb{E}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]$ and $\mathbb{E}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*] \pm 2\sqrt{\mathbb{V}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]}$, in red. The straight line that generated data, in black. With dashed line, 50 functions sampled from the posterior of $f(\mathbf{x})$.

so that you will obtain the following result:

```
MSE =
    0.4765
NLPD =
    0.9663
```

which is clearly better than the result obtained with linear regression.

2.c) Use now only the first company to compute the non-linear regression. Obtain the posterior distribution of $f(x_*)$ evaluated at the test values x_*, i.e, $p(f(\mathbf{X}_*)|\mathbf{X},\mathbf{y},\mathbf{X}*)$.

This distribution is Gaussian, with mean $\mathbb{E}[f(\mathbf{X}_*)|\mathbf{X},\mathbf{y},\mathbf{X}*)]$ and a covariance matrix $\text{Cov}[f(\mathbf{X}_*)|\mathbf{X},\mathbf{y},\mathbf{X}*)]$. Sample 50 random vectors from the distribution and plot them vs. vector $\mathbf{X}_*$, together with the test samples.

These 50 samples of the function space are analogous to the 50 straight lines that were generated in part 1.d). You can see the result in Fig. 2. Again, the Bayesian model doesn't provide a single function, but a pdf over function, from which we extracted 50 possible functions.

2.d) On top of the previous figure, and for the test data, plot functions $\mathbb{E}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]$ and $\mathbb{E}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*] \pm 2\sqrt{\mathbb{V}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]}$ (i.e., the posterior mean of $f(\mathbf{x})$), as well as of two standard deviation above and below).[2] The random function plotted in the previous step fall inside the designated area with 95.45% probability.

[2]Note that those mean and variances were already computed in the previous part, there is no need to make additional computations.

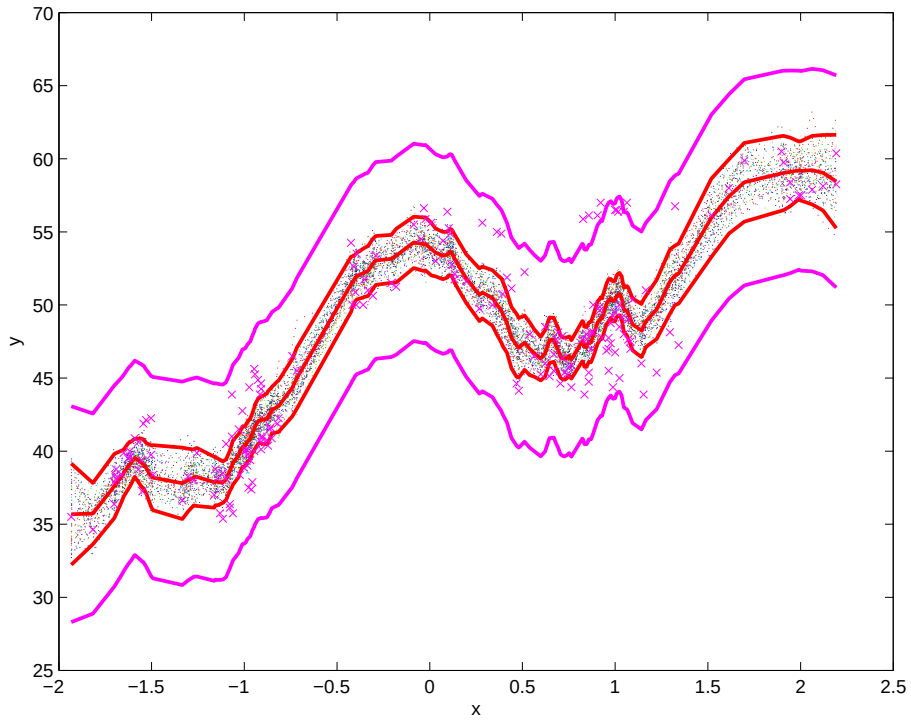

Figure 2: $\mathbb{E}[y(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*] \pm 2\sqrt{\mathbb{V}[y(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]}$ and data, in magenta. $\mathbb{E}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]$ and $\mathbb{E}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*] \pm 2\sqrt{\mathbb{V}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]}$, in red. The straight line that generated the data, in black. In dashed line, 50 functions sampled from the posterior of $f(\mathbf{x})$.

Plot now $\mathbb{E}[y(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*] \pm 2\sqrt{\mathbb{V}[y(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]}$ ($\mathbb{E}[y(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*] = \mathbb{E}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]$, so there is no need to plot it again) and observe how 95.45% of observed data fall within the designated area.[3] Those new limits set the confidence interval of our predictions, as shown in Fig. 2. In this case, regression is clearly non linear.

MSE and NLPD for this case are

```
MSE =
    6.9305
NLPD =
    1.6123
```

which, although worse, are close to the values obtained for the linear case, but now *using a single variable*.

3 The model selection model

3.a) In part 2.c) a gross estimation of the hyperparameters was used for the non-linear model. Now we are going to use principled method to find the values of hyperparameters σ_0^2, σ_n^2 and ℓ. The method boils down to computing the marginal likelihood (evidence) of the hyperparameters of the model for training data, and maximize it wrt the hyperparameters. To this end, build a MatLAB funcion with the following header:

[3] Again, no new computations are needed, sincee $\mathbb{V}[y(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*]$ is just $\mathbb{V}[f(\mathbf{x}_*)|\mathbf{X},\mathbf{y},\mathbf{x}_*] + \sigma_n^2$, as in the linear case.

```
function NLML = evidence(hyper, X, y)
s02 = hyper(1); ell = hyper(2); sn2 = hyper(3); N=size(X,1);
```

that computes the negative marginal log-likelihood (NLML):

$$-\log(p(\mathbf{y}|\mathbf{X})) = \frac{1}{2}(\mathbf{y} - m_f\mathbf{1})^\top(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}(\mathbf{y} - m_f\mathbf{1}) + \frac{1}{2}\log|\mathbf{K} + \sigma_n^2\mathbf{I}| + \frac{N}{2}\log 2\pi$$

where $\mathbf{K}$ is the covariance matrix $[\mathbf{K}]_{ij} = \sigma_0^2 \exp(-||\mathbf{x}_i - \mathbf{x}_j||_2/\ell)$.

and find the hyperparameters that minimize the NLML (i.e., maximize the evidence) using this code (we will use as a starting point the gross values previously suggested):[4]

```
obj = @(hyper) evidence(hyper, X, y);
hyper = fminsearch(obj, [s02, ell, sn2],optimset('Display','iter'));
s02 = hyper(1); ell = hyper(2); sn2 = hyper(3);
```

Finally, using the found hyperparameters instead of the gross estimation, repeat part 2.c).

You will get the following result, which is better than the one obtained in 2.c), for both quality measures, as a consequence of the rigorous model selection procedure

```
Iteration   Func-count     min f(x)         Procedure
     0            1          1940.77
                ...
   149          264          1871.58         contract inside

MSE =
    6.8841
NLPD =
    1.4270
```

[4]The provided code searches for the minimum of a function without needing its gradient. This makes it simpler (no need to compute the gradient of the NLML wrt to each hyperparameter), but also slower and less efficient finding the global optimum. In a realistic use case, both gradient and value are computed and the optimization is carried out using conjugate gradient.