Bayesian_regression_professor

September 30, 2024

1 Bayesian Parametric Regression

```
Notebook version: 1.6 (Sep 29, 2024)

Author: Jerónimo Arenas García (jarenas@tsc.uc3m.es)
        Jesús Cid-Sueiro (jesus.cid@uc3m.es)

Changes: v.1.0 - First version
         v.1.1 - ML Model selection included
         v.1.2 - Some typos corrected
         v.1.3 - Rewriting text, reorganizing content, some exercises.
         v.1.4 - Revised introduction
         v.1.5 - Revised notation. Solved exercise 5
         v.1.6 - Eq. citation and numbering, revised Secs. 5-6, simplified code, predictive mea

Pending changes: * Include regression on the stock data
```

[17]:
```python
# Import some libraries that will be necessary for working with data and␣
 ↪displaying plots

# To visualize plots in the notebook
%matplotlib inline
from IPython import display

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import scipy.io       # To read matlab files
import pylab
import time

from sklearn.preprocessing import PolynomialFeatures
```

1.1 A quick note on the mathematical notation

In this notebook we will make extensive use of probability distributions. In general, we will use capital letters $\mathbf{X}$, S, E ..., to denote random variables, and lower-case letters $\mathbf{x}$, s, ϵ ..., to denote the values they can take.

In general, we will use letter p for probability density functions (pdf). When necessary, we will use, capital subindices to make the random variable explicit. For instance, $p_{\mathbf{X},S}(\mathbf{x}, s)$ would be the joint pdf of random variables $\mathbf{X}$ and S at values $\mathbf{x}$ and s, respectively.

However, to avoid a notation overload, we will omit subindices when they are clear from the context. For instance, we will use $p(\mathbf{x}, s)$ instead of $p_{\mathbf{X},S}(\mathbf{x}, s)$.

Finally, we will use notation $\mathbf{U} \sim N(\mathbf{m}, \mathbf{V})$ to express that $\mathbf{U}$ is a Gaussian random variable with mean $\mathbf{m}$ and variance matrix $\mathbf{V}$

1.2 1. Model-based parametric regression

1.2.1 1.1. The regression problem.

Given an observation vector $\mathbf{x}$, the goal of the regression problem is to find a function $f(\mathbf{x})$ providing *good* predictions about some unknown variable s. To do so, we assume that a set of *labelled* training examples, $\{\mathbf{x}_k, s_k\}_{k=0}^{K-1}$ is available.

The predictor function should make good predictions for new observations $\mathbf{x}$ not used during training. In practice, this is tested using a second set (the *test set*) of labelled samples.

1.2.2 1.2. Model-based parametric regression

Model-based regression methods assume that all data in the training and test dataset have been generated by some stochastic process. In **parametric regression**, we assume that the probability distribution generating the data has a known parametric form, but the values of some parameters are unknown.

In particular, in this notebook we will assume the target variables in all pairs $(\mathbf{x}_k, s_k)$ from the training and test sets have been generated independently from some posterior distribution $p(s|\mathbf{x}, \mathbf{w})$, were $\mathbf{w}$ is some unknown parameter. The training dataset is used to estimate $\mathbf{w}$.

1.2.3 1.3. Model assumptions

In order to estimate $\mathbf{w}$ from the training data in a mathematicaly rigorous and compact form let us group the target variables into a vector

$$\mathbf{s} = (s_0, \dots, s_{K-1})^\top$$

and the input vectors into a matrix

$$\mathbf{X} = (\mathbf{x}_0, \dots, \mathbf{x}_{K-1})^\top$$

We will make the following **assumptions**:

- A1. All samples in D have been generated by the same distribution, $p(\mathbf{x}, s \mid \mathbf{w})$
- A2. Input variables $\mathbf{x}$ do not depend on $\mathbf{w}$. This implies that

$$p(\mathbf{X} \mid \mathbf{w}) = p(\mathbf{X})$$

- A3. Targets $s_0, \dots, s_{K-1}$ are statistically independent, given $\mathbf{w}$ and the inputs $\mathbf{x}_0, \dots, \mathbf{x}_{K-1}$, that is:

$$p(\mathbf{s} \mid \mathbf{X}, \mathbf{w}) = \prod_{k=0}^{K-1} p(s_k \mid \mathbf{x}_k, \mathbf{w})$$

1.3 2. Bayesian inference.

1.3.1 2.1. The Bayesian approach

The main idea of **Bayesian inference** is the following: assume we want to estimate some unknown variable U given an observed variable O. If U and O are random variables, we can describe the relation between U and O through the following functions:

- **Prior distribution**: $p_U(u)$. It describes our uncertainty on the true value of U before observing O.
- **Likelihood function**: $p_{O|U}(o \mid u)$. It describes how the value of the observation is generated for a given value of U.
- **Posterior distribution**: $p_{U|O}(u \mid o)$. It describes our uncertainty on the true value of U once the true value of O is observed.

The major component of the Bayesian inference is the posterior distribution. All Bayesian estimates are computed as some of its central statistics (e.g. the mean, the median or the mode), for instance

- **Maximum A Posteriori (MAP) estimate**: $\hat{u}_{\text{MAP}} = \arg\max_u p_{U|O}(u \mid o)$
- **Minimum Mean Square Error (MSE) estimate**: $\hat{u}_{\text{MSE}} = \mathbb{E}\{U \mid O = o\}$

The choice between the MAP or the MSE estimate may depend on practical or computational considerations. From a theoretical point of view, $\hat{u}_{\text{MSE}}$ has some nice properties: it minimizes $\mathbb{E}\{(U - \hat{u})^2\}$ among all possible estimates, $\hat{u}$, so it is a natural choice. However, it involves the computation of an integral, which may not have a closed-form solution. In such cases, the MAP estimate can be a better choice.

The prior and the likelihood function are auxiliary distributions: if the posterior distribution is unknown, it can be computed from them using the Bayes rule:

$$p_{U|O}(u \mid o) = \frac{p_{O|U}(o \mid u) \cdot p_U(u)}{p_O(o)}$$

In the next two sections we show that the Bayesian approach can be applied to both the prediction and the estimation problems.

1.3.2 2.2. Bayesian prediction under a known model

Assuming that the model parameters $\mathbf{w}$ are known, we can apply the Bayesian approach to predict $\mathbf{s}$ for an input $\mathbf{x}$. In that case, we can take

- Unknown variable: $\mathbf{s}$, and
- Observations: $\mathbf{x}$

an the Bayesian estimates become:

- Maximum A Posterior (MAP):

$$\hat{s}_{\text{MAP}} = \arg\max_s p(s|\mathbf{x}, \mathbf{w}) \tag{1}$$

- Minimum Mean Square Error (MSE):

$$\hat{s}_{\text{MSE}} = \mathbb{E}\{S|\mathbf{x}, \mathbf{w}\} \tag{2}$$

Exercise 1: Assuming

$$p(s \mid x, w) = \frac{s}{wx^2} \exp\left(-\frac{s^2}{2wx^2}\right), \qquad s \geq 0,$$

compute the MAP and MSE predictions of s given x and w.

Solution:

$$\begin{aligned}
\hat{s}_{\text{MAP}} &= \arg\max_s \left\{ \frac{s}{wx^2} \exp\left(-\frac{s^2}{2wx^2}\right) \right\} \\
&= \arg\max_s \left\{ \log(s) - \log(wx^2) - \frac{s^2}{2wx^2} \right\} \\
&= \sqrt{w}x
\end{aligned}$$

where the last step results from maximizing by differentiation.

$$\begin{aligned}
\hat{s}_{\text{MSE}} &= \mathbb{E}\{s|x, w\} \\
&= \int_0^\infty \frac{s^2}{wx^2} \exp\left(-\frac{s^2}{2wx^2}\right) \\
&= \frac{1}{2} \int_{-\infty}^\infty \frac{s^2}{wx^2} \exp\left(-\frac{s^2}{2wx^2}\right) \\
&= \frac{\sqrt{2\pi}}{2\sqrt{wx^2}} \int_{-\infty}^\infty \frac{s^2}{\sqrt{2\pi wx^2}} \exp\left(-\frac{s^2}{2wx^2}\right)
\end{aligned}$$

Noting that the last integral corresponds to the variance of a zero-mean Gaussian distribution, we get

$$\begin{aligned}
\hat{s}_{\text{MSE}} &= \frac{\sqrt{2\pi}}{2\sqrt{wx^2}} wx^2 \\
&= \sqrt{\frac{\pi w}{2}} x
\end{aligned}$$

2.2.1. The Gaussian case The Gaussian model is one of the most widely used in Bayesian regression, mainly because it can be applied to arbitrarily non-linear and multidimensional regression problems. It is based on the assumption that the target variable and the observation are related through the stochastic equation

$$s = \mathbf{w}^\top \mathbf{z} + \varepsilon \tag{3}$$

where $\mathbf{z} = T(\mathbf{x})$ is a known, possibly nonlinear transformation of the input variables, and ε is a zero-mean Gaussian random variable

$$\varepsilon \sim N(0, \sigma_\varepsilon^2)$$

that is independent of $\mathbf{w}$ and $\mathbf{x}$. This is equivalent to claim that

$$p(s|\mathbf{x}, \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma_\varepsilon} \exp\left(-\frac{(s - \mathbf{w}^\top \mathbf{z})^2}{2\sigma_\varepsilon^2}\right) \tag{4}$$

For a Gaussian distribution (and for any unimodal symetric distributions) the mean and the mode are the same and, thus, using Eqs. (1) and (2),

$$\hat{s}_{\text{MAP}} = \hat{s}_{\text{MSE}} = \mathbf{w}^\top \mathbf{z} \tag{5}$$

Such expression includes a linear regression model, where $\mathbf{z} = [1; \mathbf{x}]$, as well as any other non-linear model as long as it is "linear in the parameters".

1.3.3 2.3. Bayesian Inference for Parameter Estimation

In a similar way, we can apply Bayesian inference to estimate the model parameters $\mathbf{w}$ from a given dataset, D. In that case

- the unknown variable is $\mathbf{w}$, and
- the observation is $D \equiv \{\mathbf{X}, \mathbf{s}\}$

so that

- Maximum A Posterior (MAP):

$$\widehat{\mathbf{w}}_{\text{MAP}} = \arg\max_{\mathbf{w}} p(\mathbf{w}|D) \tag{6}$$

- Minimum Mean Square Error (MSE):

$$\widehat{\mathbf{w}}_{\text{MSE}} = \mathbb{E}\{\mathbf{W}|D\} \tag{7}$$

1.4 3. Bayesian parameter estimation

NOTE: Since the training data inputs are known, all probability density functions and expectations in the remainder of this notebook will be conditioned on the data matrix, $\mathbf{X}$. To simplify the mathematical notation, from now on we will remove $\mathbf{X}$ from all conditions. For instance, we will write $p(\mathbf{s}|\mathbf{w})$ instead of $p(\mathbf{s}|\mathbf{w}, \mathbf{X})$, etc. Keep in mind that, in any case, all probabilities and expectations may depend on $\mathbf{X}$ implicitely.

Summarizing, the steps to design a Bayesian parametric regresion algorithm are the following:

1. Assume a parametric data model $p(s|\mathbf{x}, \mathbf{w})$ and a prior distribution $p(\mathbf{w})$.
2. Using the data model and the i.i.d. assumption, compute $p(\mathbf{s}|\mathbf{w})$.
3. Applying the bayes rule, compute the posterior distribution $p(\mathbf{w}|\mathbf{s})$.
4. Compute the MAP or the MSE estimate of $\mathbf{w}$ given $\mathbf{x}$.
5. Compute predictions using the selected estimate.

1.4.1 3.1. Bayesian Inference and Maximum Likelihood.

Applying the Bayes rule the MAP estimate in Eq. (6) can be alternatively expressed as

$$\begin{aligned} \widehat{\mathbf{w}}_{\text{MAP}} &= \arg\max_{\mathbf{w}} \frac{p(D|\mathbf{w}) \cdot p(\mathbf{w})}{p(D)} \\ &= \arg\max_{\mathbf{w}} \{p(D|\mathbf{w}) \cdot p(\mathbf{w})\} \end{aligned} \tag{8}$$

By comparison, the ML (Maximum Likelihood) estimate has the form:

$$\widehat{\mathbf{w}}_{\text{ML}} = \arg\max_{\mathbf{w}} p(\mathcal{D}|\mathbf{w})$$

This shows that the MAP estimate takes into account the prior distribution on the unknown parameter.

Another advantage of the Bayesian approach is that it provides not only a point estimate of the unknown parameter, but a whole funtion, the posterior distribution, which encompasses our belief on the unknown parameter given the data. For instance, we can take second order statistics like the variance of the posterior distributions to measure the uncertainty on the true value of the parameter around the mean.

The MAP estimate can also be expressed as a function of the negative-log likelihood. Applying the minus logarithm over Eq. (8), we get

$$\begin{aligned}\widehat{\mathbf{w}}_{\text{MAP}} &= \arg\min_{\mathbf{w}} \left\{\text{NLL}(w) - \log\left(p(\mathbf{w})\right)\right\} \\ &= \arg\min_{\mathbf{w}} \left\{ -\sum_{k=0}^{K-1} \log(p(s_k|x_k, \mathbf{w})) - \log\left(p(\mathbf{w})\right) \right\}\end{aligned}$$

1.4.2 3.2. The prior distribution

Since each value of $\mathbf{w}$ determines a regression function, by stating a prior distribution over the weights we state also a prior distribution over the space of regression functions.

For instance, assume that the data likelihood follows the Gaussian model in Eq. (3), with $T(x) = (1, x, x^2, x^3)$, i.e. the regression functions have the form

$$w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

Each value of $\mathbf{w}$ determines a specific polynomial of degree 3. Thus, the prior distribution over $\mathbf{w}$ describes which polynomials are more likely before observing the data.

For instance, assume a Gaussian prior with zero mean and variance $\mathbf{V}_p$, i.e.,

$$p(\mathbf{w}) = \frac{1}{(2\pi)^{D/2}|\mathbf{V}_p|^{1/2}} \exp\left(-\frac{1}{2}\mathbf{w}^\top \mathbf{V}_p^{-1}\mathbf{w}\right)$$

where D is the dimension of $\mathbf{w}$. To abbreviate, we will also express this as

$$\mathbf{w} \sim N\left(\mathbf{0}, \mathbf{V}_p\right)$$

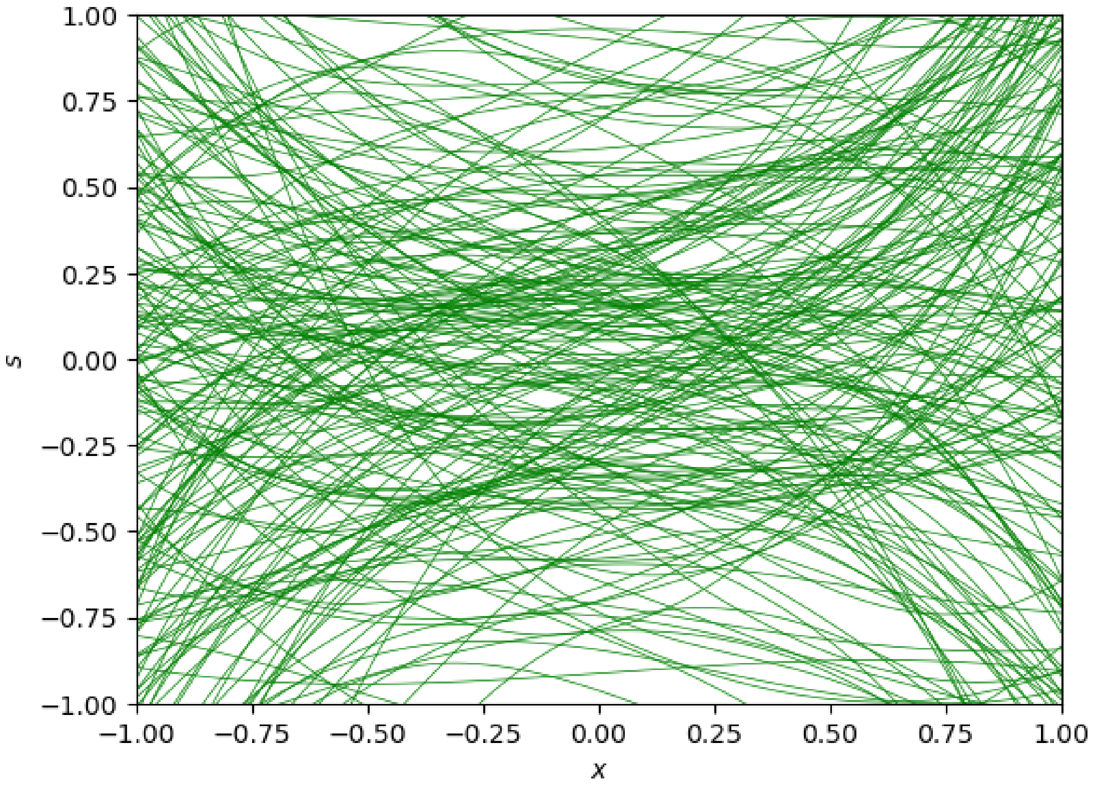
The following code samples $\mathbf{w}$ according to this distribution for $\mathbf{V}_p = 0.002\,\mathbf{I}$, and plots the resulting polynomial over the scatter plot of an arbitrary dataset.

You can check the effect of modifying the variance of the prior distribution.

[18]:

```
n_grid = 200
degree = 3
nplots = 200

# Prior distribution parameters
mean_w = np.zeros((degree + 1,))
v_p = 0.2      ### Try increasing this value
var_w = v_p * np.eye(degree+1)
```

```
xmin = -1
xmax = 1
X_grid = np.linspace(xmin, xmax, n_grid)

fig = plt.figure()
ax = fig.add_subplot(111)

for k in range(nplots):

    #Draw weigths fromt the prior distribution
    w_iter = np.random.multivariate_normal(mean_w, var_w)
    S_grid_iter = np.polyval(w_iter, X_grid)
    ax.plot(X_grid, S_grid_iter, 'g-', lw=0.5)

ax.set_xlim(xmin, xmax)
ax.set_ylim(-1, 1)
ax.set_xlabel('$x$')
ax.set_ylabel('$s$')
plt.show()
```



The data observation will modify our belief about the true data model according to the posterior distribution. In the following we will analyze this in a Gaussian case.

1.5 4. Bayesian regression for a Gaussian model.

We will apply the above steps to derive a Bayesian regression algorithm for a Gaussian model.

1.5.1 4.1. Step 1: The Gaussian model.

Let us assume that the likelihood function is given by the Gaussian model in Eq. (3)

$$s \mid \mathbf{w} \sim N\left(\mathbf{z}^\top \mathbf{w}, \sigma_\varepsilon^2\right)$$

Assume, also, that the prior is Gaussian

$$\mathbf{w} \sim N\left(\mathbf{0}, \mathbf{V}_p\right)$$

1.5.2 4.2. Step 2: Complete data likelihood

Using the assumptions A1, A2 and A3 (see Notebook about ML regression), it can be shown that

$$\mathbf{s} \mid \mathbf{w} \sim N\left(\mathbf{Z}\mathbf{w}, \sigma_\varepsilon^2 \mathbf{I}\right)$$

that is

$$p(\mathbf{s}|\mathbf{w}) = \frac{1}{\left(\sqrt{2\pi}\sigma_\varepsilon\right)^K} \exp\left(-\frac{1}{2\sigma_\varepsilon^2}\|\mathbf{s} - \mathbf{Z}\mathbf{w}\|^2\right)$$

1.5.3 4.3. Step 3: Posterior weight distribution

The posterior distribution of the weights can be computed using the Bayes rule

$$p(\mathbf{w}|\mathbf{s}) = \frac{p(\mathbf{s}|\mathbf{w})\ p(\mathbf{w})}{p(\mathbf{s})}$$

Since both $p(\mathbf{s}|\mathbf{w})$ and $p(\mathbf{w})$ follow a Gaussian distribution, we know also that the joint distribution and the posterior distribution of $\mathbf{w}$ given $\mathbf{s}$ are also Gaussian. Therefore,

$$\mathbf{w} \mid \mathbf{s} \sim N\left(\mathbf{w}_{\text{MSE}}, \mathbf{V}_{\mathbf{w}}\right)$$

It can be shown that mean and the covariance matrix of the distribution are:

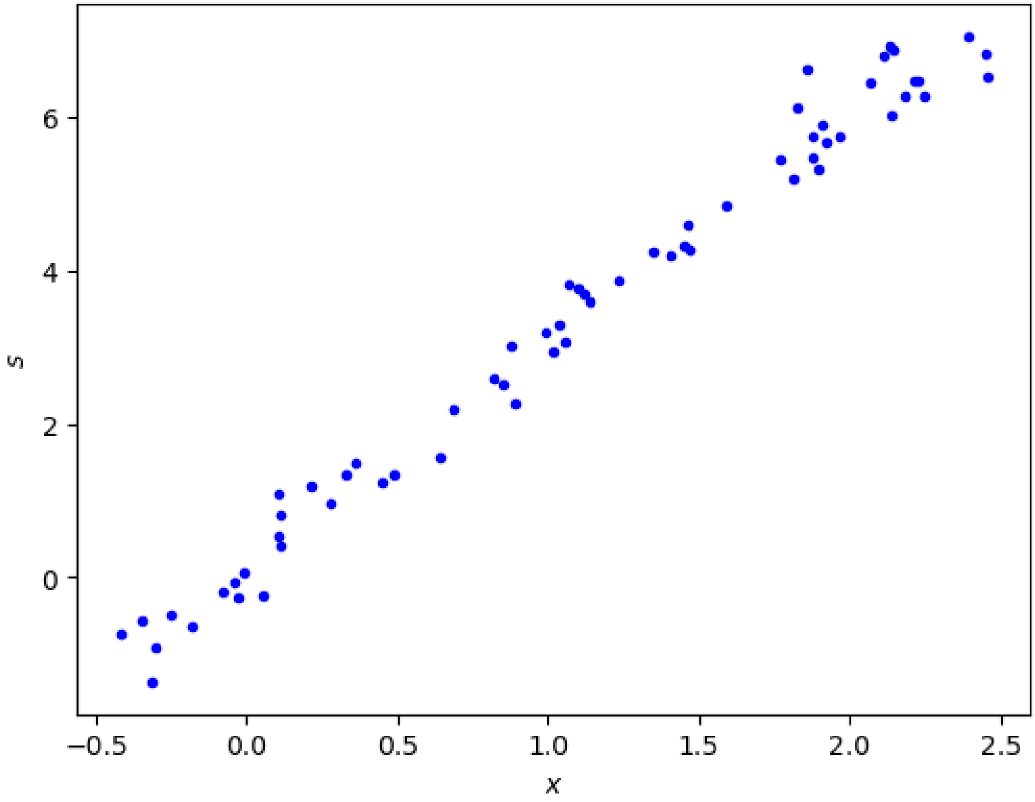
$$\boxed{\mathbf{V}_{\mathbf{w}} = \left[\frac{1}{\sigma_\varepsilon^2}\mathbf{Z}^\top \mathbf{Z} + \mathbf{V}_p^{-1}\right]^{-1}} \tag{9}$$

$$\boxed{\mathbf{w}_{\text{MSE}} = \sigma_\varepsilon^{-2}\mathbf{V}_{\mathbf{w}}\mathbf{Z}^\top \mathbf{s}} \tag{10}$$

Exercise 2: Consider the dataset with one-dimensional inputs given by

[19]:

```
# True data parameters
w_true = 3
std_n = 0.4

# Generate the whole dataset
n_max = 64
X_tr = 3 * np.random.random((n_max,1)) - 0.5
S_tr =  w_true * X_tr + std_n * np.random.randn(n_max,1)

# Plot data
plt.figure()
plt.plot(X_tr, S_tr, 'b.')
plt.xlabel('$x$')
plt.ylabel('$s$')
plt.show()
```



Fit a Bayesian linear regression model assuming $z = x$ and

[20]:

```
# Model parameters
sigma_eps = 0.4
mean_w = 1              # Since w is a scalar, mean_w is a scalar, too
```

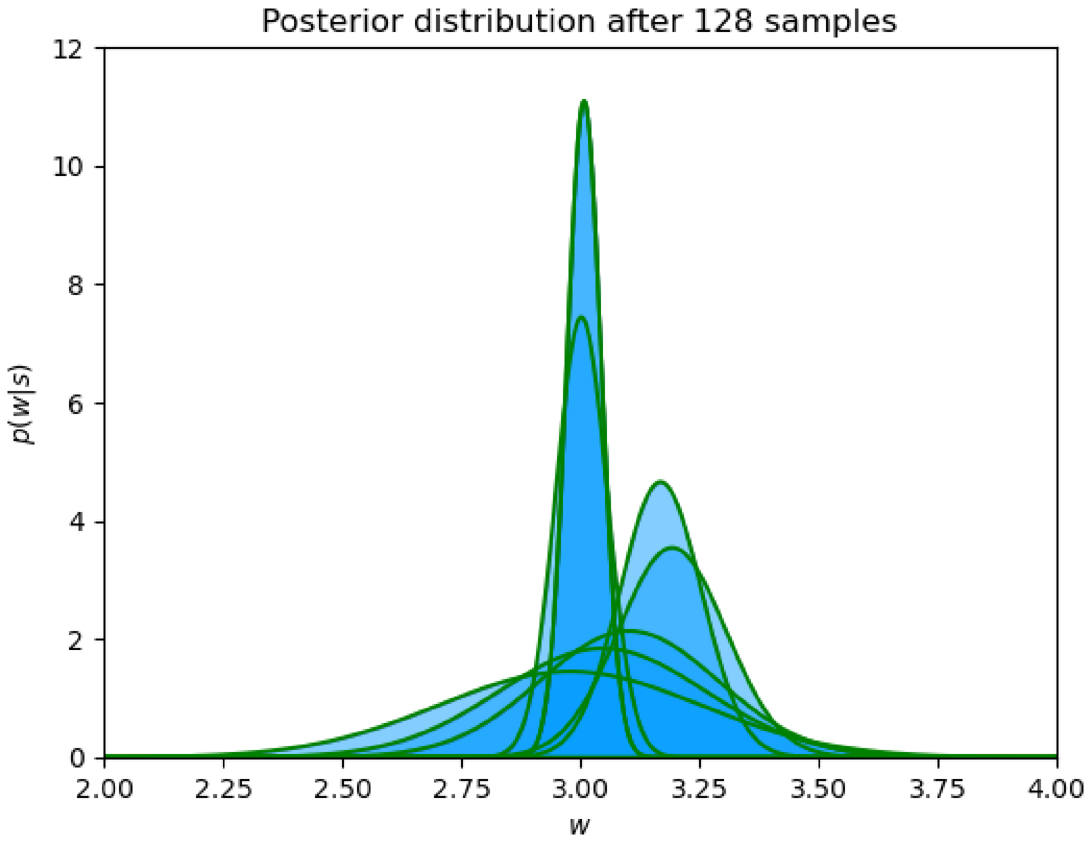
```
sigma_p = 1e6
Var_p = sigma_p**2   # Since w is a scalar, Var_p is a scalar, too
```

(note that parameter w is a scalar).

To do so, compute the posterior weight distribution using the first k samples in the complete dataset, for $k = 1, 2, 4, 8, \ldots 128$. Draw all these posteriors along with the prior distribution in the same plot.

[21]:

```
# No. of points to analyze
n_points = [1, 2, 4, 8, 16, 32, 64, 128]

# Prepare plots
w_grid = np.linspace(2, 4, 5000)   # Sample the w axis
plt.figure()

# Compute the prior distribution over the grid points in w_grid
# p = <FILL IN>
p = 1.0/(sigma_p * np.sqrt(2*np.pi)) * np.exp(-(w_grid**2) / (2*sigma_p**2))
plt.plot(w_grid, p,'g-')

for k in n_points:

    # Select the first k samples
    Zk = X_tr[:k, :]
    Sk = S_tr[:k]

    # Parameters of the posterior distribution
    # 1. Compute the posterior variance.
    #    (Make sure that the resulting variable, Var_w, is a scalar.)
    # Var_w = <FILL IN>
    Var_w = 1/ ((Zk.T @ Zk)[0][0]/(sigma_eps**2) + 1 / Var_p)

    # 2. Compute the posterior mean.
    #    (Make sure that the resulting variable, w_MSE, is a scalar)
    # w_MSE = <FILL IN>
    w_MSE = (Var_w * (Zk.T @ Sk)[0][0] / sigma_eps**2)

    # Compute the posterior distribution over the grid points in w_grid
    sigma_w = np.sqrt(Var_w)     # First we take a scalar standard deviation
    # p = <FILL IN>
    p = 1.0/(sigma_w*np.sqrt(2*np.pi)) * np.exp(-((w_grid-w_MSE)**2)/
 ↪(2*sigma_w**2))

    plt.plot(w_grid, p,'g-')
    plt.fill_between(w_grid, 0, p, alpha=0.5, edgecolor='#1B2ACC',␣
 ↪facecolor='#089FFF',
                     linewidth=0.8, antialiased=True)
    plt.title(f'Posterior distribution after {k} samples')
    plt.xlim(w_grid[0], w_grid[-1])
    plt.ylim(0, 12)
    plt.xlabel('$w$')
    plt.ylabel('$p(w|s)$')

    display.clear_output(wait=True)
    display.display(plt.gcf())
    time.sleep(2.0)

# Remove the temporary plots and fix the last one
display.clear_output(wait=True)
plt.show()
```



Exercise 3: Note that, in the example above, the model assumptions are correct: the target variables have been generated by a linear model with noise standard deviation `sigma_n` which is exactly equal to the value assumed by the model, stored in variable `sigma_eps`. Check what happens if we take `sigma_eps=4*sigma_n` or `sigma_eps=sigma_n/4`.

- Does the algorithm fail in that cases?

- What differences can you observe with respect to the ideal case `sigma_eps=sigma_n`?

[ ]:

```

```

1.5.4 4.4. Step 4: Weight estimation.

Since the posterior weight distribution is Gaussian, both the MAP and the MSE estimates are equal to the posterior mean in Eq. (10):

$$\widehat{\mathbf{w}}_{\text{MAP}} = \widehat{\mathbf{w}}_{\text{MSE}} = \sigma_\varepsilon^{-2} \mathbf{V}_{\mathbf{w}} \mathbf{Z}^\top \mathbf{s}$$

1.5.5 4.5. Step 5: Prediction

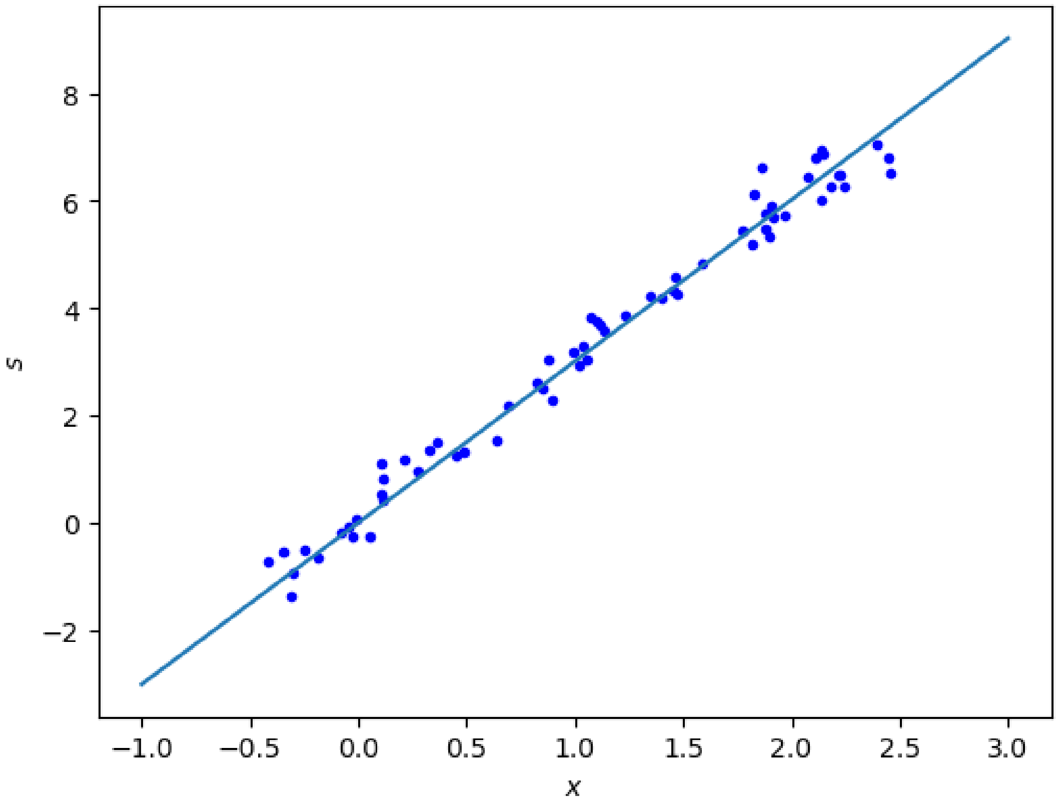
Using the MSE estimate, as in Eq. (5), the final predictions are given by

$$\hat{s}_{\text{MSE}} = \widehat{\mathbf{w}}_{\text{MSE}}^\top \mathbf{z}$$

Exercise 4: Plot the minimum MSE predictions of $s$ for inputs $x$ in the interval [-1, 3].

[22]:

```
# Make grid on points (two points suffices to plot a line)
# x = <FILL IN>
x = np.array([-1.0, 3.0])
# Compute the predicted values for the points in the grid
# s_pred = <FILL IN>
s_pred = w_MSE * x

plt.figure()
plt.plot(X_tr, S_tr,'b.')
plt.plot(x, s_pred)
plt.xlabel('$x$')
plt.ylabel('$s$')
plt.show()
```



1.6 5. The predictive mean.

Up to here, we have followed a Bayesian approach in two steps:

1. **Learning**: use $p(\mathbf{w} \mid D)$ to compute the MSE estimate, $\mathbf{w}_{\text{MSE}}$.
2. **Prediction**: use $p(s \mid \mathbf{x}, \mathbf{w})$ to compute prediction $\widehat{s}_{\text{MSE}} = \mathbb{E}\{s \mid \mathbf{x}, \mathbf{w}_{\text{MSE}}\}$

From a purely Bayesian perspective, if the goal is to predict s (the unknown variable) for an input $\mathbf{x}$ and from the data in D (the observarions) we can use $p(s \mid \mathbf{x}, D)$ to compute estimate

$$\begin{aligned}\widehat{s}^* &= \mathbb{E}\{s|\mathbf{x}, D\} \\ &= \int \mathbb{E}\{s|\mathbf{x}, \mathbf{w}, D\} p(\mathbf{w} \mid D) d\mathbf{w} \\ &= \int \mathbb{E}\{s|\mathbf{x}, \mathbf{w}\} p(\mathbf{w} \mid D) d\mathbf{w}\end{aligned} \tag{10}$$

where we have applied the total expectation theorem in the second equality, and the independence assumption in the third one. The Bayesian estimate $\widehat{s}^*$ is known as the **predictive mean**.

From Eq. (10), we can interpret the predictive mean as the aggregation of all possible estimates of the target variable (one per each $\mathbf{w}$), weighted by the posterior parameter distribution. That is, the purely Bayesian estimate integrates all possible predictors for a given model into a single one.

In general, $\widehat{s}_{\text{MSE}}$ and $\widehat{s}^*$ are different estimations, and $\widehat{s}^*$ may be hard to compute. Fortunatelly, for the Gaussian model, they are equivalent, which can be shown by noting that

$$\begin{aligned}
\widehat{s}^* &= \int_{\mathbf{w}} \mathbf{w}^\top \mathbf{z} p(\mathbf{w} \mid D) d\mathbf{w} \\
&= \left(\int_{\mathbf{w}} \mathbf{w} p(\mathbf{w} \mid D) d\mathbf{w} \right)^\top \mathbf{z} \\
&= \mathbf{w}_{\text{MSE}}^\top \mathbf{z} = \widehat{s}_{\text{MSE}}
\end{aligned}$$

1.7 6. Posterior distribution of the target variable

One of the powerfull advantages of the Bayesian regression models is that they provide no only a prediction of the target variable, but a posterior distribution, which may be very informative of the quality of the predictions.

For the Gaussian model, the computation of the posterior distribution is straightforward. Eq. (3) shows that the target variable is the sum of two components:

$$s = f + \varepsilon$$

were $f = \mathbf{w}^\top \mathbf{z}$. The prediction error is a consequence of the randomness of the noise (which cannot be predicted, since it is independent on the date) and the uncertainty about $\mathbf{w}$, which can be reduced using data.

Since f is a linear combination of random variables, it is a Gaussian random variable, with mean and variance

$$\begin{aligned}
\mathbb{E}\{f \mid \mathbf{x}, D\} &= \mathbb{E}\{s \mid \mathbf{x}, D\} = \widehat{\mathbf{w}}_{\text{MSE}}^\top \mathbf{z} \\
\text{var}(f \mid \mathbf{x}, D) &= \mathbb{E}\{(f - \mathbf{w}_{\text{MSE}}^\top \mathbf{z})^2 \mid \mathbf{x}, D\} \\
&= \mathbb{E}\{((\mathbf{w} - \mathbf{w}_{\text{MSE}})^\top \mathbf{z})^2 \mid \mathbf{x}, D\} \\
&= \mathbf{z}^\top \mathbf{V}_{\mathbf{w}} \mathbf{z}
\end{aligned}$$

Therefore,

$$f^* \mid \mathbf{s}, \mathbf{x} \sim N\left(\widehat{\mathbf{w}}_{\text{MSE}}^\top \mathbf{z}, \quad \mathbf{z}^\top \mathbf{V}_{\mathbf{w}} \mathbf{z}\right)$$

and, since $s = f + \varepsilon$, which is the sum of independent variables the posterior distribution of the target variable

$$s \mid \mathbf{s}, \mathbf{z}^* \sim N\left(\widehat{\mathbf{w}}_{\text{MSE}}^\top \mathbf{z}, \quad \mathbf{z}^\top \mathbf{V}_{\mathbf{w}} \mathbf{z} + \sigma_\varepsilon^2\right)$$

The variance of the posterior distribution,

$$v_{s|\mathbf{x}} = \mathbf{z}^\top \mathbf{V}_{\mathbf{w}} \mathbf{z} + \sigma_\varepsilon^2$$

is known as the **predictive variance**, and it accounts for the variance of the target variable for each observation. Note that the noise variance cannot be reduced, but more data reduce the variance of the parameter estimates, and thus the predictive variance, too.

Example: random sinusoid For the remainder of this notebook, we will illustrate the behavior of the Bayesian regression models by using a dataset of noisy samples from a sinusoidal signal.

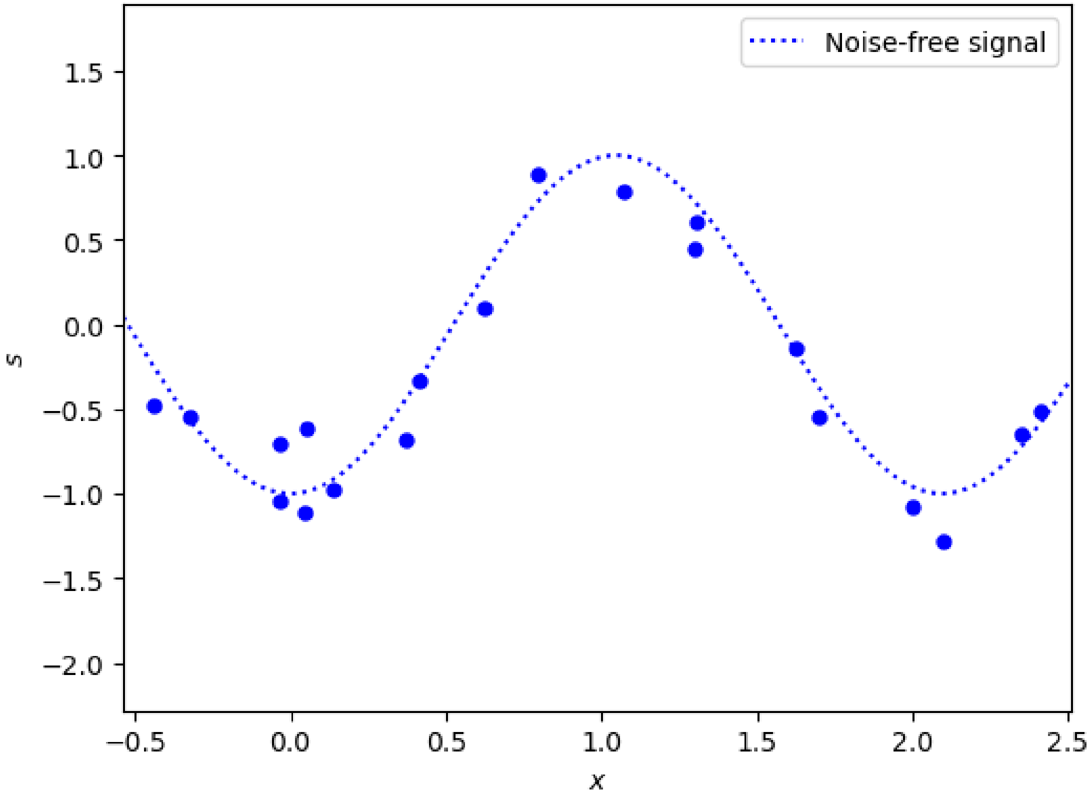
[23]:

```
# Configurable parameters
n_points = 20
frec = 3
std_n = 0.2      # Noise standard deviation
n_grid = 200

# Initialize random generator, for reproducibility
np.random.seed(42)

# Data generation
X_tr = 3 * np.random.random((n_points,1)) - 0.5
S_tr = - np.cos(frec*X_tr) + std_n * np.random.randn(n_points, 1)
```

The figure shows the sampled points, and the underlying function (in the dotted curve).

[24]:

```
# Signal
xmin = np.min(X_tr) - 0.1
xmax = np.max(X_tr) + 0.1
X_grid = np.linspace(xmin, xmax, n_grid)
S_grid = - np.cos(frec*X_grid) #Noise free for the true model

# Plot data
plt.figure()
plt.plot(X_tr, S_tr,'b.',markersize=10)
# Plot noise-free function
plt.plot(X_grid, S_grid, 'b:', label='Noise-free signal')

# we set some limits for the plot
ymin = np.min(S_tr) - 1
ymax = np.max(S_tr) + 1

# Set axes
plt.xlim(xmin, xmax), plt.xlabel('$x$')
plt.ylim(ymin, ymax), plt.ylabel('$s$')
plt.legend(loc='best')
plt.show()
```

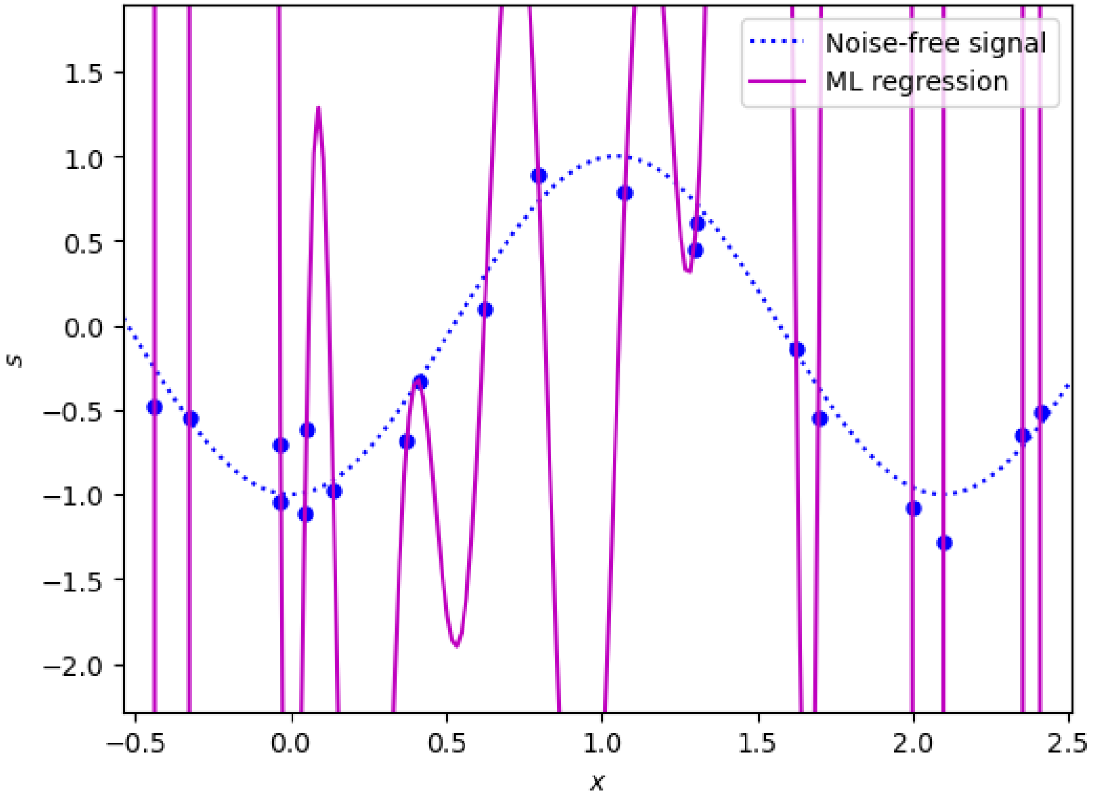


Let us assume that the cosine form of the noise-free signal is unknown, and we assume a polynomial model with a high degree. The following code plots the ML estimate

[25]:

```
# Configurable parameters
degree = 18

# Compute matrix with training input data using sklearn
poly = PolynomialFeatures(degree=degree)

# Transform the training data
Z_tr = poly.fit_transform(X_tr)
# Compute the ML parameters
wML = np.linalg.lstsq(Z_tr, S_tr, rcond=1e-20)[0]

# Transform the samples in the grid.
X_grid_matrix = np.array([X_grid]).T      # Convert to 1-column matrix
Z_grid = poly.transform(X_grid_matrix)    # Transform the matrix

# Make predictions for the samples in the grid
# (and convert to a 1D array, as X_grid, for plotting)
S_grid_ML = (Z_grid @ wML).T[0]
```

```
# Plot data
fig = plt.figure()
plt.plot(X_tr, S_tr, 'b.', markersize=10)

# Plot noise-free function
plt.plot(X_grid, S_grid, 'b:', label='Noise-free signal')
# Plot ML regression function
plt.plot(X_grid, S_grid_ML, 'm-', label='ML regression')

# Set axis
plt.xlim(xmin, xmax), plt.xlabel('$x$')
plt.ylim(ymin, ymax), plt.ylabel('$s$')
plt.legend(loc='best')
plt.show()
```

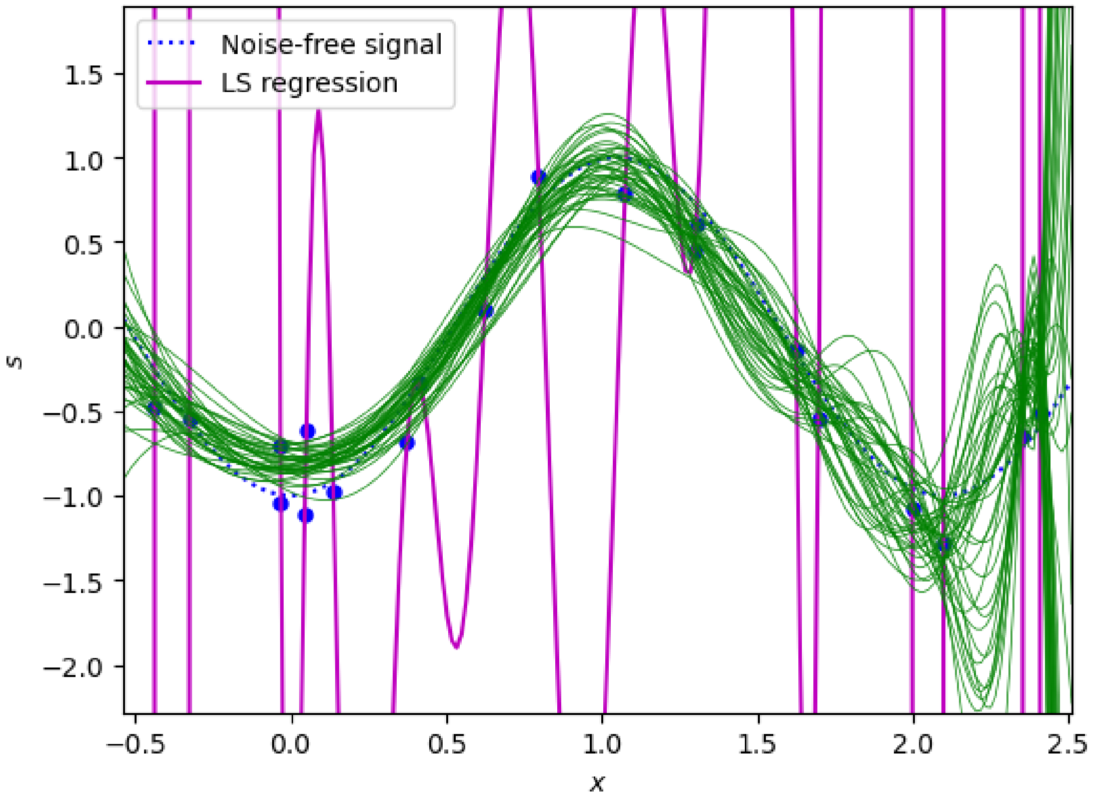


When the degree of the polynomial is close to the size of the dataset, the ML regresion overfits the training data, and would produce a large prediction errors over an independent test set.

The following fragment of code computes the posterior weight distribution, draws random vectors from $p(\mathbf{w}|\mathbf{s})$, and plots the corresponding regression curves along with the training points. Compare these curves with those extracted from the prior distribution of $\mathbf{w}$ and with the ML solution.

[26]:

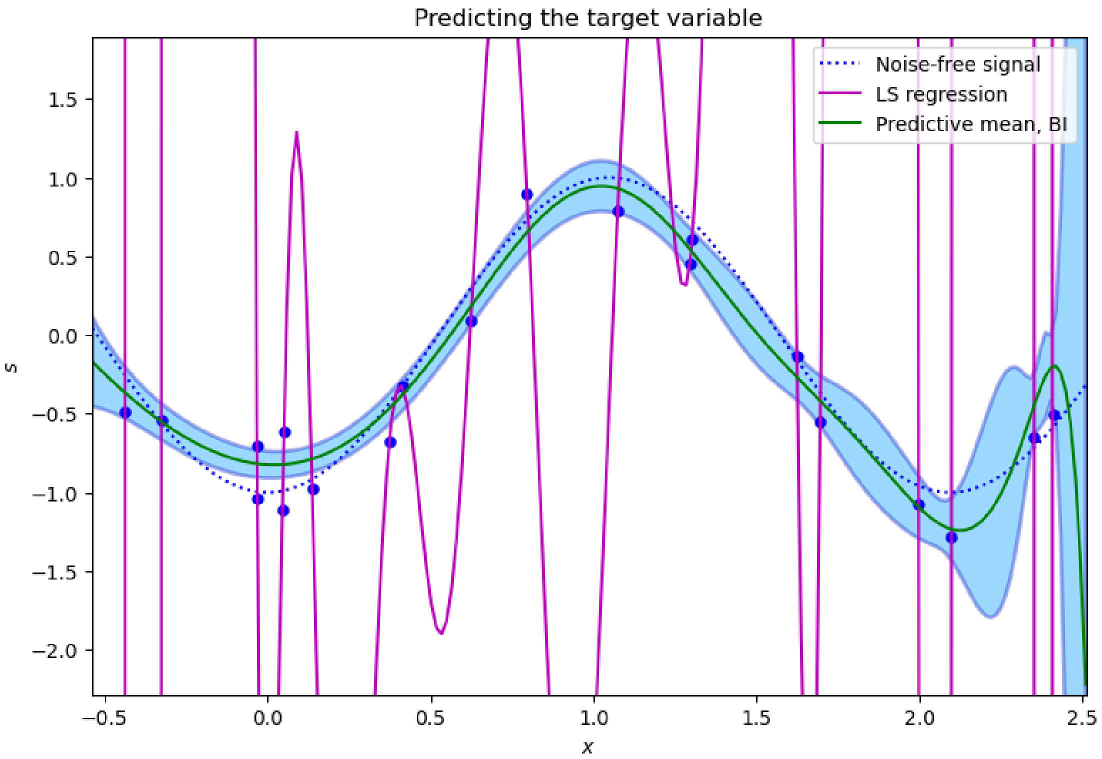
```
# Configurable parameters
nplots = 40
# Noise standard deviation
sigma_eps = 0.2
# Prior distribution parameters
mean_w = np.zeros((degree + 1,))
sigma_p = 2

# Compute posterior distribution parameters
Vp = sigma_p**2 * np.eye(degree + 1)
inv_Vp = 1 / sigma_p**2 * np.eye(degree + 1)
Vw = np.linalg.inv(inv_Vp + Z_tr.T @ Z_tr / (sigma_eps**2))
# This is to ensure that Var_w is symmetric, and does not deviate because of
# numerical errors
Vw = (Vw + Vw.T) / 2
# Compute the posterior mean, and convert to 1D array
wMSE = Vw @ Z_tr.T @ S_tr / (sigma_eps**2)
wMSE = wMSE.T[0]

# Plot data
plt.figure()
plt.plot(X_tr, S_tr,'b.',markersize=10)
# Plot noise-free function
plt.plot(X_grid, S_grid, 'b:', label='Noise-free signal')
# Plot ML regression function
plt.plot(X_grid, S_grid_ML, 'm-', label='LS regression')

for k in range(nplots):
    # Draw weights from the posterior distribution
    w_iter = np.random.multivariate_normal(wMSE, Vw)

    # Compute and plot predictions for the samples in the grid
    S_grid_iter = Z_grid @ w_iter
    plt.plot(X_grid, S_grid_iter, 'g-', lw=0.5)

# Set axis
plt.xlim(xmin, xmax), plt.xlabel('$x$')
plt.ylim(ymin, ymax), plt.ylabel('$s$')
plt.legend(loc='best')
plt.show()
```



As an alternative to the The following fragment of code computes the posterior weight distribution, draws random vectors from $p(\mathbf{w}|\mathbf{s})$, and plots the corresponding regression curves along with the training points. Compare these curves with those extracted from the prior distribution of $\mathbf{w}$ and with the LS solution.

[27]:
```
# Compute the posterior mean
S_grid_MSE = Z_grid @ wMSE

# Compute standard deviations of the posterior distribution for the samples in
# the grid
Vsx = np.sum(Z_grid * (Z_grid @ Vw), axis=1)
std_x = np.sqrt(Vsx)

# Plot data
plt.figure(figsize=(9, 6))
plt.plot(X_tr, S_tr, 'b.', markersize=10)
# Plot true function
plt.plot(X_grid, S_grid, 'b:', label='Noise-free signal')
# Plot LS regression function
plt.plot(X_grid, S_grid_ML, 'm-', label='LS regression')
```

```python
# Plot predictive mean for the Bayesian Inference
plt.plot(X_grid, S_grid_MSE,'g-',label='Predictive mean, BI')
# Plot confidence intervals for the Bayesian Inference
plt.fill_between(X_grid, S_grid_MSE-std_x, S_grid_MSE+std_x,
    alpha=0.4, edgecolor='#1B2ACC', facecolor='#089FFF',
    linewidth=2, antialiased=True)

# Set axis
plt.xlim(xmin, xmax), plt.xlabel('$x$')
plt.ylim(ymin, ymax), plt.ylabel('$s$')
plt.title('Predicting the target variable')
plt.legend(loc='best')
plt.show()
```



Not only do we obtain a better predictive model, but we also have confidence intervals (error bars) for the predictions.

Exercise 5: Assume the dataset $D = \{x_k, s_k\}_{k=0}^{K-1}$ containing K i.i.d. samples from a distribution

$$p(s|x, w) = wx \exp(-wxs), \qquad s > 0, \quad x > 0, \quad w > 0$$

We model also our uncertainty about the value of w assuming a prior distribution for w following a Gamma distribution with parameters $\alpha > 0$ and $\beta > 0$.

$$w \ \sim \text{Gamma}\left(\alpha, \beta\right) = \frac{\beta^\alpha}{\Gamma(\alpha)} w^{\alpha-1} \exp\left(-\beta w\right), \qquad w > 0$$

Note that the mean and the mode of a Gamma distribution can be calculated in closed-form as

$$\mathbb{E}\left\{w\right\} = \frac{\alpha}{\beta};$$

$$\text{mode}\{w\} = \arg\max_w p(w) = \frac{\alpha - 1}{\beta}$$

1. Determine an expression for the likelihood function.

Solution:

$$\begin{aligned} p(\mathbf{s}|w) &= \prod_{k=0}^{K-1} p(s_k|w, x_k) = \prod_{k=0}^{K-1} \left(w x_k \exp(-w x_k s_k)\right) \\ &= w^K \cdot \left(\prod_{k=0}^{K-1} x_k\right) \exp\left(-w \sum_{k=0}^{K-1} x_k s_k\right) \end{aligned} \tag{1}$$

2. Determine the maximum likelihood coefficient, $\widehat{w}_{\text{ML}}$.

Solution:

$$\widehat{w}_{\text{ML}} = \arg\max_w w^K \cdot \left(\prod_{k=0}^{K-1} x_k\right) \exp\left(-w \sum_{k=0}^{K-1} x_k s_k\right) \tag{2}$$

$$= \arg\max_w \left(w^K \cdot \exp\left(-w \sum_{k=0}^{K-1} x_k s_k\right)\right) \tag{3}$$

$$= \arg\max_w \left(K \log(w) - w \sum_{k=0}^{K-1} x_k s_k\right) \tag{4}$$

$$= \frac{K}{\sum_{k=0}^{K-1} x_k s_k} \tag{5}$$

3. Obtain the posterior distribution $p(w|\mathbf{s})$. Note that you do not need to calculate $p(\mathbf{s})$ since the posterior distribution can be readily identified as another Gamma distribution.

Solution:

$$p(w|\mathbf{s}) = \frac{p(\mathbf{s}|w)p(w)}{p(s)} \tag{6}$$

$$= \frac{1}{p(s)} \left(w^K \cdot \left(\prod_{k=0}^{K-1} x_k\right) \exp\left(-w \sum_{k=0}^{K-1} x_k s_k\right)\right) \left(\frac{\beta^\alpha}{\Gamma(\alpha)} w^{\alpha-1} \exp\left(-\beta w\right)\right) \tag{7}$$

$$= \frac{1}{p(s)} \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\prod_{k=0}^{K-1} x_k\right) \left(w^{K+\alpha-1} \cdot \exp\left(-w \left(\beta + \sum_{k=0}^{K-1} x_k s_k\right)\right)\right) \tag{8}$$

that is

$$w \mid \mathbf{s} \sim Gamma\left(K+\alpha, \beta + \sum_{k=0}^{K-1} x_k s_k\right)$$

4. Determine the MSE and MAP a posteriori estimators of w: $w_{\mathrm{MSE}} = \mathbb{E}\{w|\mathbf{s}\}$ and $w_{\mathrm{MAP}} = \max_w p(w|\mathbf{s})$.

Solution:

$$w_{\mathrm{MSE}} = \mathbb{E}\{w \mid \mathbf{s}\} = \frac{K+\alpha}{\beta + \sum_{k=0}^{K-1} x_k s_k}$$

$$w_{\mathrm{MAP}} = \mathrm{mode}\{w\} = \arg\max_w p(w) = \frac{K+\alpha-1}{\beta + \sum_{k=0}^{K-1} x_k s_k}$$

5. Compute the following estimators of S:

$\hat{s}_1 = \mathbb{E}\{s|w_{\mathrm{ML}}, x\}$

$\hat{s}_2 = \mathbb{E}\{s|w_{\mathrm{MSE}}, x\}$

$\hat{s}_3 = \mathbb{E}\{s|w_{\mathrm{MAP}}, x\}$

Solution:

$$\hat{s}_1 = \mathbb{E}\{s|w_{\mathrm{ML}}, x\} = w_{\mathrm{ML}} x$$

$$\hat{s}_2 = \mathbb{E}\{s|w_{\mathrm{MSE}}, x\} = w_{\mathrm{MSE}} x$$

$$\hat{s}_3 = \mathbb{E}\{s|w_{\mathrm{MAP}}, x\} = w_{\mathrm{MAP}} x$$

1.8 7. Maximum evidence model selection

We have already addressed with Bayesian Inference the following two issues:

- For a given degree, how do we choose the weights?
- Should we focus on just one model, or can we use several models at once?

However, we still needed some assumptions: a parametric model (i.e., polynomial function and a priori degree selection) and several parameters needed to be adjusted.

Though we can recur to cross-validation, Bayesian inference opens the door to other strategies.

- We could argue that rather than keeping single selections of these parameters, we could use simultaneously several sets of parameters (and/or several parametric forms), and average them in a probabilistic way … (like we did with the models)
- We will follow a simpler strategy, selecting just the most likely set of parameters according to an ML criterion

1.8.1 7.1 Model evidence

The evidence of a model is defined as

$$L = p(\mathbf{s} \mid M)$$

where M denotes the model itself and any free parameters it may have. For instance, for the polynomial model we have assumed so far, M would represent the degree of the polynomia, the variance of the additive noise, and the a priori covariance matrix of the weights

Applying the Theorem of Total probability, we can compute the evidence of the model as

$$L = \int p(\mathbf{s} \mid \mathbf{f}, M) p(\mathbf{f} \mid M) d\mathbf{f}$$

For the linear model $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{z}$, the evidence can be computed as

$$L = \int p(\mathbf{s} \mid \mathbf{w}, M) p(\mathbf{w} \mid M) d\mathbf{w}$$

It is important to notice that these probability density functions are exactly the ones we computed on the previous section. We are just making explicit that they depend on a particular model and the selection of its parameters. Therefore:

- $p(\mathbf{s} \mid \mathbf{w}, M)$ is the likelihood of $\mathbf{w}$
- $p(\mathbf{w} \mid M)$ is the a priori distribution of the weights

1.8.2 7.2 Model selection via evidence maximization

- As we have already mentioned, we could propose a prior distribution for the model parameters, $p(M)$, and use it to infer the posterior. However, this can be very involved (usually no closed-form expressions can be derived)
- Alternatively, maximizing the evidence is normally good enough

$$M_{\text{ML}} = \arg\max_M p(s \mid M)$$

Note that we are using the subscript 'ML' because the evidence can also be referred to as the likelihood of the model

1.8.3 7.3 Example: Selection of the degree of the polynomia

For the previous example we had (we consider a spherical Gaussian for the weights):

- $\mathbf{s} \mid \mathbf{w}, M \ \sim \ N\left(\mathbf{Z}\mathbf{w},\ \sigma_\varepsilon^2 \mathbf{I}\right)$
- $\mathbf{w} \mid M \ \sim \ N\left(\mathbf{0},\ \sigma_p^2 \mathbf{I}\right)$

In this case, $p(\mathbf{s} \mid M)$ follows also a Gaussian distribution, and it can be shown that

- $L = p(\mathbf{s} \mid M) = N\left(\mathbf{0}, \sigma_p^2 \mathbf{Z}\mathbf{Z}^\top + \sigma_\varepsilon^2 \mathbf{I}\right)$

If we just pursue the maximization of L, this is equivalent to maximizing the log of the evidence

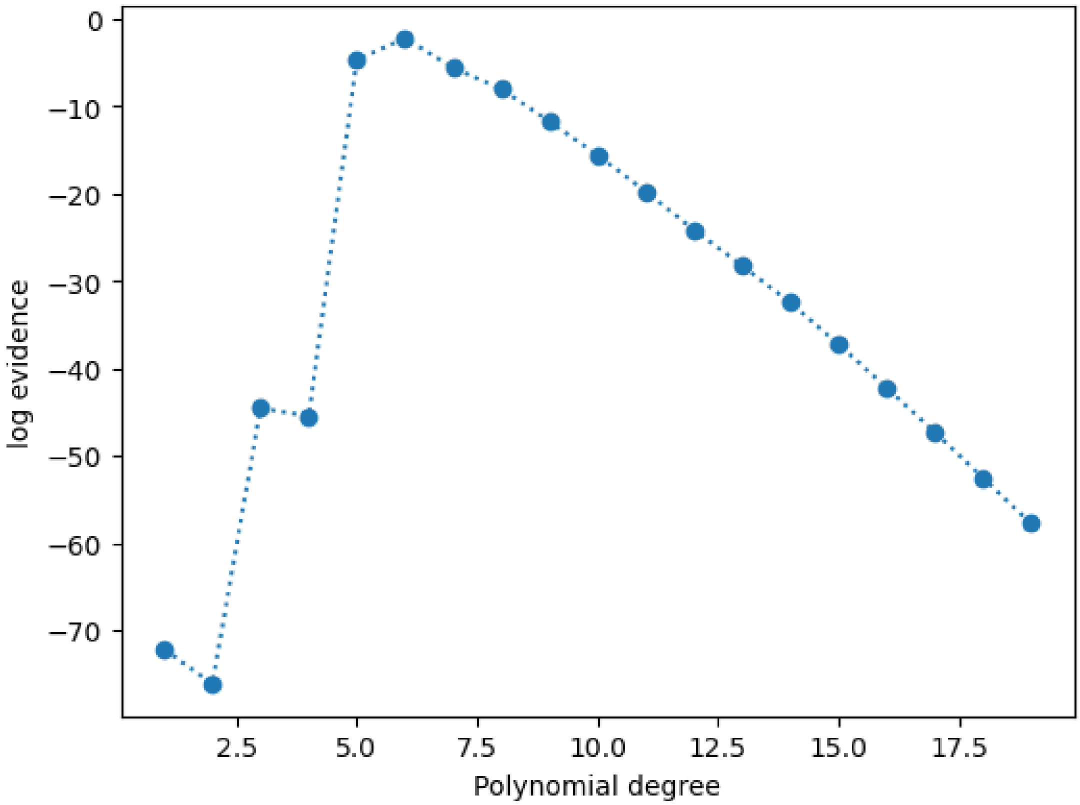
$$\log(L) = -\frac{M}{2}\log(2\pi) - \frac{1}{2}\log \mid \sigma_p^2\mathbf{Z}\mathbf{Z}^\top + \sigma_\varepsilon^2\mathbf{I} \mid -\frac{1}{2}\mathbf{s}^\top\left(\sigma_p^2\mathbf{Z}\mathbf{Z}^\top + \sigma_\varepsilon^2\mathbf{I}\right)^{-1}\mathbf{s}$$

where M denotes the length of vector $\mathbf{z}$ (the degree of the polynomia minus 1).

The following fragment of code evaluates the evidence of the model as a function of the degree of the polynomia

[28]:

```
# Prior distribution parameters
max_degree = n_points - 1

# Evaluate the posterior evidence
logE = []
for deg in range(max_degree):
    Z_iter = Z_tr[:,:deg+1]
    ZZ = Z_iter @ Z_iter.T
    logE_iter = - (deg + 1) * np.log(2 * np.pi) / 2   \
                - np.log(np.linalg.det((sigma_p**2) * ZZ + (sigma_eps**2)*np.
 ↪eye(n_points))) / 2   \
                - S_tr.T @ np.linalg.inv((sigma_p**2) * ZZ + (sigma_eps**2)*np.
 ↪eye(n_points)) @ S_tr / 2
    logE.append(logE_iter[0, 0])

plt.plot(np.array(range(max_degree))+1, logE, ':o')
plt.xlabel('Polynomial degree')
plt.ylabel('log evidence')
plt.show()
```



[29]:

```
degree = np.argmax(logE)
print(f'The degree of the polynomial is {degree}')
```

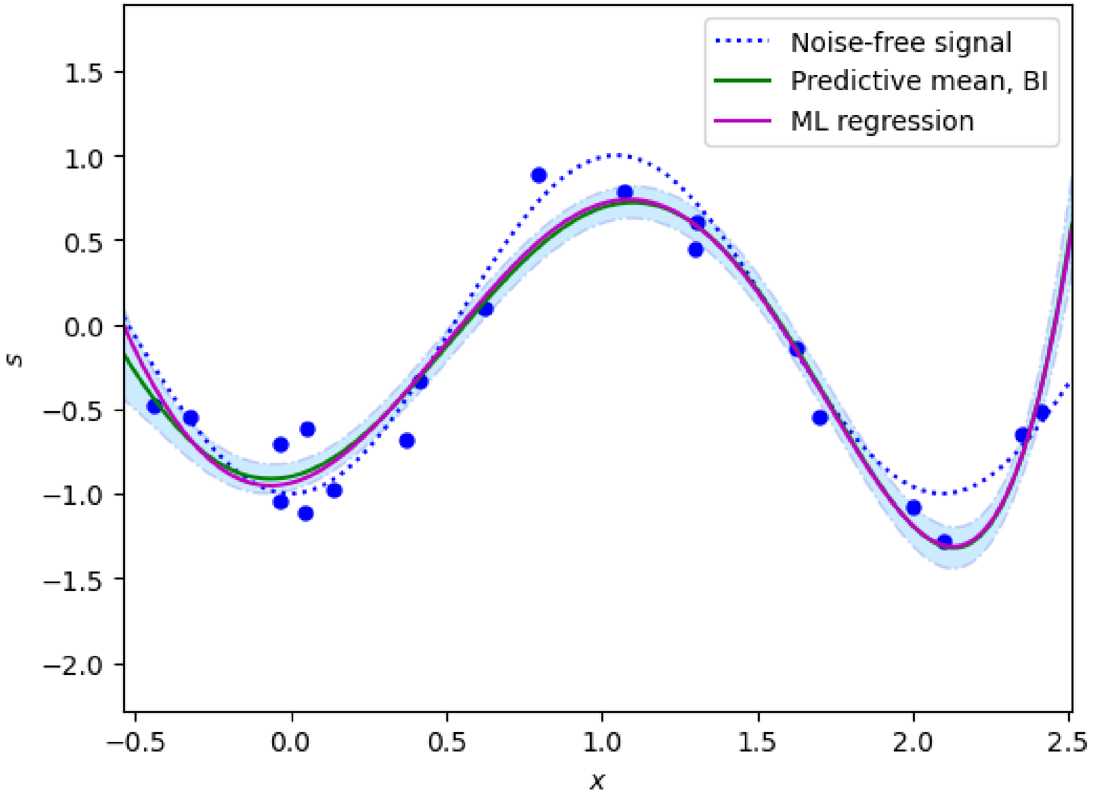
```
The degree of the polynomial is 5
```

Let's train the model for the maximum degree

[30]:

```
inv_Vp = np.eye(degree+1) / sigma_p**2

# Select transformed matrix up to the selected degree
Z = Z_tr[:, :(degree+1)]

# Compute posterior distribution parameters
Vw = np.linalg.inv(Z.T @ Z /(sigma_eps**2) + inv_Vp)
wMSE = (Vw @ Z.T @ S_tr / (sigma_eps**2)).T[0]

# Compute maximum likelihood solution for the selected degree
wML = np.linalg.lstsq(Z, S_tr, rcond=1e-20)[0]

# Compute ML and MSE predictions for the grid inputs
Z_grid_degree = Z_grid[:, :(degree+1)]
S_grid_ML = Z_grid_degree @ wML
```

```
S_grid_MSE = Z_grid_degree @ wMSE

# Compute standard deviations of the posterior distribution for the samples in
# the grid
Vsx = np.sum(Z_grid_degree * (Z_grid_degree @ Vw), axis=1)
std_x = np.sqrt(Vsx)

# Plot data
fig = plt.figure()
plt.plot(X_tr, S_tr,'b.',markersize=10)
# Plot true function
plt.plot(X_grid, S_grid, 'b:', label='Noise-free signal')
plt.plot(X_grid, S_grid_MSE,'g-',label='Predictive mean, BI')

plt.fill_between(X_grid, S_grid_MSE - std_x, S_grid_MSE + std_x,
    alpha=0.2, edgecolor='#1B2ACC', facecolor='#089FFF',
    linewidth=1, linestyle='dashdot', antialiased=True)

#We plot also the least square solution
plt.plot(X_grid, S_grid_ML,'m-',label='ML regression')

plt.xlim(xmin, xmax), plt.xlabel('$x$')
plt.ylim(ymin, ymax), plt.ylabel('$s$')
plt.legend(loc='best')
plt.show()
```



We can check, that now the model also seems quite appropriate for ML regression, but keep in mind that selection of such parameter was itself carried out using Bayesian inference.