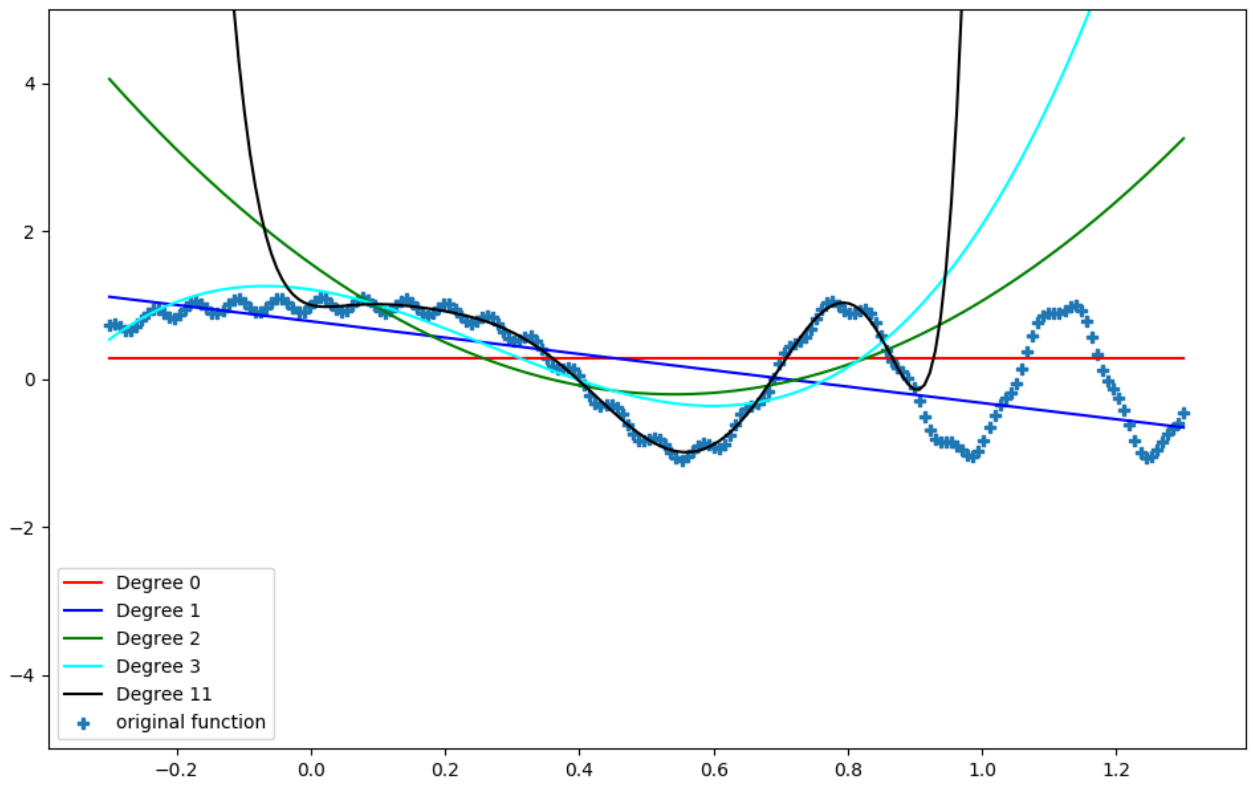
496 - Mathematics for Machine Learning CW 2

Adrian Catană

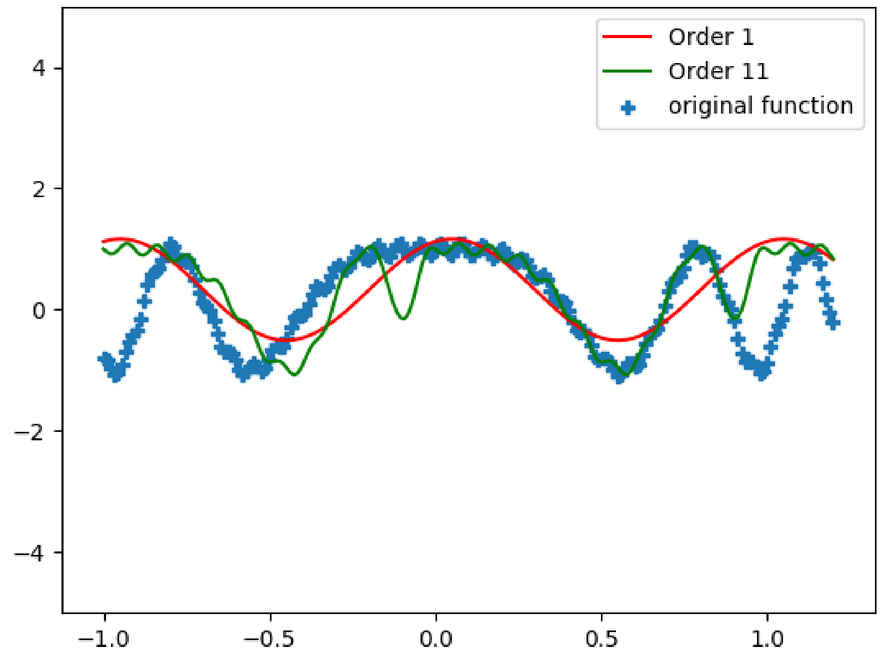
November 6, 2018

Exercise 1a. Since y is Gaussian, by using the same techniques in the book, chapter 9, we get $w_{\text{opt}} = (\Phi^T \cdot \Phi)^{-1} \cdot \Phi^T \cdot y$ and $\sigma^2_{\text{opt}} = \frac{1}{N} \sum (y_i - w^T \cdot \phi(x_i))^2$. Note that y, x are from the training set and $\Phi_{ij} = x_i^j$. The 200 points were generated using:

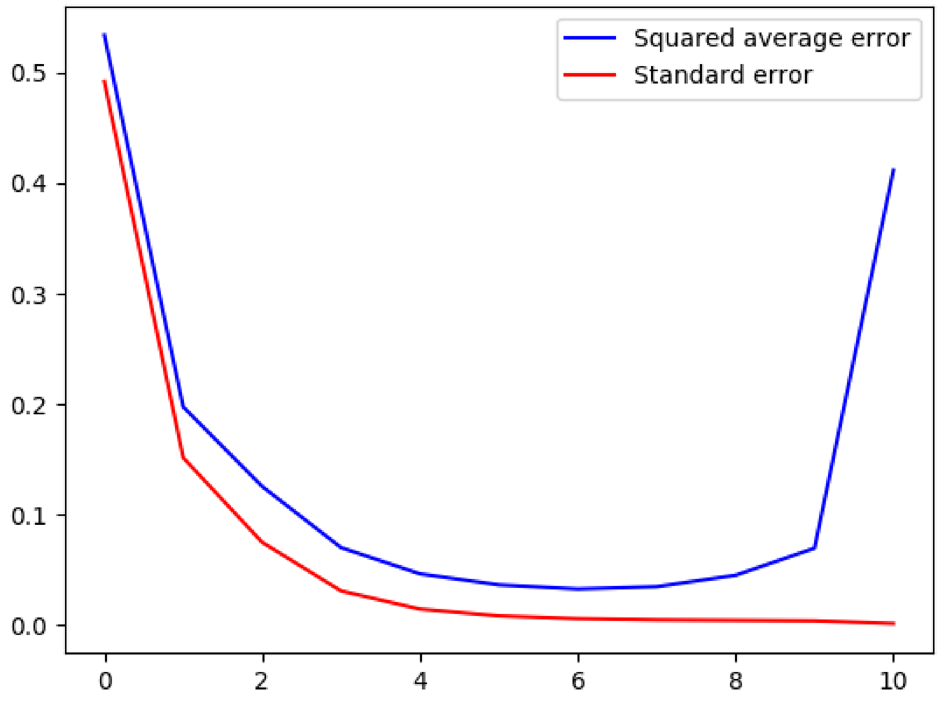
```
mockPoints = np.reshape(np.linspace(-1, 1.2, 200), (200, 1))
```



Exercise 1b. Same formulas as above, excepting the trigonometric basis functions.



Exercise 1c. For the purpose of this exercise, I have implemented a cross validation algorithm that is easily configurable for the number of points and the number of segments that we are going to use for training and testing (K segments, $K-1$ for training, 1 for testing). The first graph below is built for 25 points, with segments of length 1, i.e. for 25 steps, every point becomes testing data, while the other 24 are used for training. The blue curve is the average of squared errors when using cross validation, while the red one represents the standard error, computed using the whole 25 datapoints, after the formula described in 1a.



Exercise 1d. Overfitting occurs when the model used behaves too well on the training set, without a good generalization on the unseen, testing datapoints. The basic intuition is that complex models turn out to be less probable and more error prone than simpler ones. The goal of our model is to see how influential each feature is, based on previous experience (the training set) and predict what will happen in the future for some certain inputs.

For our model selection we use MLE to compare and determine the best order of our chosen basis functions (polynomial or trigonometric) by finding the function that best minimizes the objective.

We notice that linear polynomials, i.e. of degree 0 and 1, fit the data poorly and, hence, are bad representations for our objective function. For degrees M $= 2, 3$ the graphs look a bit more plausible, even though they only intersect the training points in only a few points.

When we go to higher-degree polynomials, i.e. degree 11, we notice that the function will go through more and more data points in the $[0, 0.9]$ interval. That is also the case for 1b, where the green, higher-order trigonometric curve will overlap with the original function in many more points than the order 1, red curve. This is a sign of overfitting, as the models become more complex and the "weights" or features of the training data points become larger and larger.

By looking at the test error, which is how we measure the generalization of the corresponding polynomial, we notice that initially the test error decreases. For 4th-order polynomials the test error is relatively low and stays relatively constant up to degree 7. However, from degree 7 onward the test error increases significantly, higher-order polynomials having very bad generalization properties.

Note that the training error (red curve) never increases when the degree of the polynomial increases. In our example, the best generalization (the point of the smallest test error) is obtained for a polynomial of degree 6. Also, another sign of when the overfitting starts is when the gap in between the training and test errors grows wider and wider. That is the moment when we start learning too specific about our training data, while negatively influencing the results on testing data.

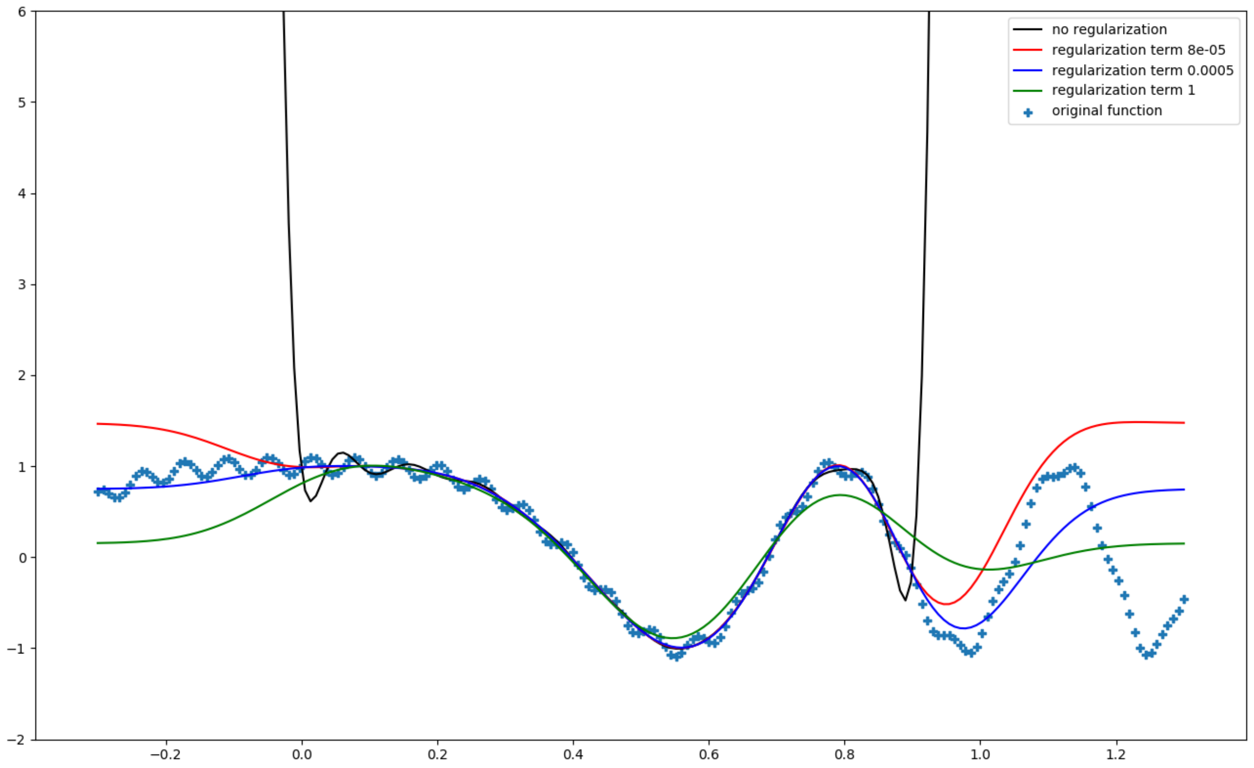
Exercise 2a. The MAP approach gives us the parameter w_{MAP} that maximizes the posterior $p(w|X, y)$. Why we need this? Because we want the highest probability for the parameter we are seeking, w, for given training data X, y. Now, applying Bayes' theorem, we get that $p(w|X, y) = \frac{p(y|X,w)\cdot p(w)}{p(y|X)}$. Now, taking log and negating gives us $-\log p(w|X, y) = -\log p(y|X, w) - \log p(w) + \log p(y|X)$ and we seek the minimum of this. Note that $\log p(y|X)$ is constant, as training data is known a-priori, so what we really wish to mimimize is $-\log p(y|X, w) - \log p(w)$.

Now we take an important step, that is making an assumption about our parameter prior $p(w)$, that is normally distributed with mean 0 and variance σ_w^2. Nicely, we get $-\log p(w) = \log \frac{1}{\sqrt{2\pi\sigma_w^2}} - \log e^{-\frac{(w-0)^2}{2\sigma_w^2}} = \text{const} + \frac{1}{2\sigma_w^2} w^2 = \text{const} + \frac{1}{2\sigma_w^2} w^T w = \text{const} + \frac{1}{2\sigma_w^2} ||w||_2^2$. Thus, we get $-\log p(w|X, y) = -\log p(y|X, w) + \frac{1}{2\sigma_w^2} ||w||_2^2 + \text{const}$. However, from 1a, $-\log p(y|X, w) = \frac{1}{2\sigma^2}(y - \Phi w)^T (y - \Phi w)$, so we get $-\log p(w|X, y) = \frac{1}{2\sigma^2}(y - \Phi w)^T (y - \Phi w) + \frac{1}{2\sigma_w^2} ||w||_2^2 + \text{const}$, which we want to minimize. This is equivalent to minimizing $(y-\Phi w)^T(y-\Phi w)+\frac{2\sigma^2}{2\sigma_w^2}||w||_2^2$. However, expanding, this is $\sum(y_i - w^T\phi(x_i))^2 + \frac{\sigma^2}{\sigma_w^2}\sum w_j^2$, which looks pretty close to our $L(w)$ that we try to minimize. Taking $\sigma_w^2 = \frac{\sigma^2}{\lambda}$, we get exactly $L(w)$.

Thus the regularization term we added for MLE to $p(y|X,w)$ is $\lambda = \frac{\sigma^2}{\sigma_w^2}$, hence for quadratic regularization the parameter λ as defined above is the proportion between the variance of the Gaussian likelihood and the variance of the Gaussian prior $p(w)$. The only difference is the constant term, which, however, will completely disappear when we differentiate with respect to w, i.e. $\frac{\partial L(w)}{\partial w} = 0$ is equivalent, after multiplying by σ^2, to $\frac{\partial(-\log p(y|X,w)-\log p(w)+\log p(y|X))}{\partial w} = 0$.

Therefore, the log-prior, as defined above, reflects the impact of the regularizer that penalizes implausible values, i.e., values that are unlikely under the prior. This was exactly what we wanted in regularization: by adding a term to the log-likelihood $(-logp(y|X,w))$, we penalize the magnitude of the parameters w. Thus, the parameters we seek for are not going to grow as fast as previously did for MLE, meaning that some specific features found during training are not going to influence the generalization to the point where we overfit.

Exercise 2b.



For the general case, when we'd also have a $\frac{1}{\sigma}$ factor for the first term of the expression that we try to minimize, when using MAP we get optimal σ^2 and w on in therms of the other. Thus, a decision about σ^2 would need to be made (or, having a program implement an iterative approach by defining recurrences $\sigma_{i+1} = g(w_i)$ and $w_{i+1} = \sigma_i$. Firstly, I missed the subtle difference, i.e. that the variance term does not appear in the $L(w)$ expression, so I went for the option of computing the variance in terms of w_{MLE}.

However, for this exercise, the expression is so nice, that the ratio between σ^2 and σ_w^2 is exactly λ, so there was no need for tricks like the ones described above.

That being said, I have plotted the graphs for regularization terms 0.00001 (overfitting), 0.0005 (just right) and 1 (underfitting), together with the initial function and with the non-regularized approximation. The interesting thing to see here is that on the trained interval, $[0, 0.9]$, all functions behave really well (not so well for regularizatiom term 1 though) and close to the true value. However, what we seek for is to generalize from our small interval to wider intervals, e.g. $[-0.3, 1.3]$ and see what happens there. In the case of non-regularized approximation, we can see that as soon as our graph reaches the left neighborhood of 0 and the right one of 0.9 it starts fluctuating, a sign that our function does not generalize well. This is also a hint for us regarding the regularized functions: a really small $\lambda = 0.00001$ term will still overfit (see red curve - this is actually expected, as the values get closer to the function with no regularization, i.e. $\lim_{\lambda \to 0} f(\lambda) = 0$), giving worst approximations for the objective function outside the boundaries of the training interval. However, for a bigger value, $\lambda = 0.0005$, we see that our function generalizes better and is extremely close to the truth on the left of 0 and pretty close (anyways, much closer than all the others) to the right of 0.9. The third choice, $\lambda = 1$, behaves worse on both training and testing sets, so it clearly underfits, offering a really bad model.