Linear regression as an example of machine learning, with an analysis of the Franke function and the territory of Møsvatn Austfjell.

Ivar Lønning, Gianmarco Puleo, Arne Elias Tidemand Ruud
University of Oslo
(Dated: October 2022)

Linear regression is known in the literature as one of the lowest level machine learning (ML) techniques. Proper manipulation of the raw data make this method suitable to fit polynomials of any degree. For these reasons, and because of its simplicity, it embodies the ideal cornerstone to investigate many of the overarching topics emerging in ML. In this project, we develop a python code which fits 2D polynomials to any given set of points (x, y, z). We implement ordinary least squares regression, adding then Ridge and LASSO regularization, together with some of the most popular resampling techniques: bootstrap and cross-validation. In particular, these are respectively employed to study the bias-variance tradeoff and to discuss model selection. The codes are first tested on vanilla data generated using the Franke function. We later look for the best-fit polynomials describing the altitude profile of a region near Møsvatn Austfjell in Norway. We are able to fit polynomials of degree $d \sim 20$ to two different parts of the territory, with average accuracies of order 0.1% and 1%.

I. INTRODUCTION

The problem of fitting a function to an available set of data points is well-known to be a recurring theme in physics, where we deal with experimental, noisy data, and we want to compare them with theoretical models. The most elementary method which allows to perform such a fit is ordinary least squares (OLS), which was first described by Legendre [1] in 1805 and is often referred to as "linear regression". Suppose we have a set of N data points

$$\mathcal{D} = \{(\mathbf{X}_{i,\star}, z_i) \in \mathbb{R}^p \times \mathbb{R}, \quad i \in \mathbb{N}, \quad 0 \leq i \leq N\}, \quad (1)$$

where each $\mathbf{X}_{i,\star}$ is a row vector corresponding to a data point in the feature space $\mathbb{R}^p$, whereas the y_i's are the corresponding target values. The matrix $\mathbf{X}$ whose rows are X_i is often termed *design matrix*. As the name suggests, OLS consists in finding the linear function

$$\tilde{z}_i(X_i, \boldsymbol{\beta}) = \sum_{j=0}^{p-1} \beta_j X_{ij} \quad (2)$$

which minimizes the mean square error (MSE)

$$C^{\mathrm{OLS}}(\boldsymbol{\beta}) = \frac{1}{N} \sum_{i=1}^{N} (\tilde{\mathbf{z}}(X_i, \boldsymbol{\beta}) - z_i)^2. \quad (3)$$

It is particularly convenient to express equations (2) and (3) using matrix notation, so that they become, respectively:

$$\tilde{\mathbf{z}} = \mathbf{X}\boldsymbol{\beta} \quad (4)$$

$$C^{\mathrm{OLS}}(\boldsymbol{\beta}) = \frac{1}{N} \|\tilde{\mathbf{z}} - \mathbf{z}\|_2^2 \quad (5)$$

The minimization of cost functions like this lies at the core of all machine learning (ML) techniques, and is therefore suitable to get a grasp on some common topics which characterize this subject. In particular, it is known that trying to train complex models on noisy datasets leads to the risk of overfitting. To deal with this issue it is customary to add penalty terms to the cost function. This leads to the so-called Ridge and LASSO regression, which are based on the minimization of the following functions:

$$C^{\mathrm{Ridge}}(\boldsymbol{\beta}) = C^{\mathrm{OLS}}(\boldsymbol{\beta}) + \lambda \|\boldsymbol{\beta}\|_2^2, \quad (6)$$

$$C^{\mathrm{LASSO}}(\boldsymbol{\beta}) = C^{\mathrm{OLS}}(\boldsymbol{\beta}) + \lambda \|\boldsymbol{\beta}\|_1. \quad (7)$$

Here, λ is a regularization parameter, often called the hyperparameter.

The goal of this project is to fit polynomials of increasing complexity to a given dataset, motivated by Weierstrass' theorem (and generalisations of it) that any continuous function (defined on a compact set) can be approximated to arbitrary precision by a polynomial. We will first do the fitting with OLS regression. We then implement some of the most common resampling methods: bootstrap and cross validation, which help us in understanding the bias-variance tradeoff. Afterwards we improve our code so that it can perform also Ridge and LASSO regression. All these algorithms are first tested on vanilla datasets and later on real data, namely terrain data which describe the altitude profile of Møsvatn Austfjell in Norway. The report is structured as follows: in section II we discuss the theory of the various regression and resampling techniques we use in our code, which is then described in detail. In section III we present a critical discussion of our results. Finally, in section IV we draw the conclusions.

II. METHODS

II.1. Theory

We describe here the most relevant theoretical tools which are necessary for understanding our analysis. More about these topics can be found in [2].

Train and test

When we obtain a model by minimizing a cost function like (3), we usually split our dataset into a *train* and a *test* set. The former is what we actually use to find the optimal parameters, or in other words, to *train* the model. The latter is instead used to assess the model's performance on new data, which have never been used in the optimization procedure. It is really important that the test set does not play any role in the training procedure at all. Throughout this project, we often compute the values of the MSE on the both the train set and the test set. They are sometimes respectively termed *in-sample* and *out-of-sample* error.

OLS, Ridge and LASSO

It can be shown [2] that the optimal set of parameters minimizing the cost function (3) is

$$\hat{\boldsymbol{\beta}}^{\mathrm{OLS}} = (\mathbf{X}^\intercal \mathbf{X})^{-1}\mathbf{X}^\intercal \mathbf{z}. \tag{8}$$

Observe that this expression can be computed without knowing anything about how the dataset is generated; strictly speaking we only need to ensure that the matrix $\mathbf{X}^\intercal\mathbf{X}$ is invertible. However, linear regression acquires statistical meaningfulness when we know something about how the data have been generated. Namely, we now make the assumption that they are described by

$$z_i = f(\mathbf{X}_{i,\star}) + \varepsilon_i, \tag{9}$$

where ε_i are normally distributed, uncorrelated random variables with mean 0 and variance σ^2. We are now going to turn our attention to the statistical properties of the optimal parameters found in equation (8). In order to do it, we first take the expectation value of the random variables (9), which yields:

$$\mathbb{E}(z_i) = \mathbb{E}\left[f(\mathbf{X}_{i,\star})\right] + \mathbb{E}\left[\epsilon_i\right] = f(\mathbf{X}_{i,\star}). \tag{10}$$

We then assume that there exists a linear function of the kind (2), which is a good approximation to the function $f(\mathbf{X}_{i,\star})$. So, equation (10) reads

$$\mathbb{E}(z_i) = \sum_{j=0}^{p-1} X_{ij}\beta_j = \mathbf{X}_{i\star}\boldsymbol{\beta}. \tag{11}$$

We can also evaluate the variance of each variable z_i as

$$\mathbb{E}\left[(z_i - \mathbb{E}(z_i))^2\right] = \mathbb{E}\left[(\varepsilon_i)^2\right] = \mathrm{Var}(\varepsilon_i) = \sigma^2, \tag{12}$$

where we used (9), (10) and the fact that ε_i has mean 0. With this in mind, we can compute the expectation value of the random vector $\hat{\boldsymbol{\beta}}^{\mathrm{OLS}}$ given by equation (8). Because of the linearity of the operator $\mathbb{E}(\cdot)$, it is simply:

$$\begin{aligned}\mathbb{E}(\hat{\boldsymbol{\beta}}^{\mathrm{OLS}}) &= (\mathbf{X}^\intercal\mathbf{X})^{-1}\mathbf{X}^\intercal\mathbb{E}(\mathbf{z}) = \\ &= (\mathbf{X}^\intercal\mathbf{X})^{-1}\mathbf{X}^\intercal\mathbf{X}\boldsymbol{\beta} = \boldsymbol{\beta},\end{aligned} \tag{13}$$

where the second equality relies on equation (11). We are now interested in the covariance matrix of the random vector $\hat{\boldsymbol{\beta}}^{\mathrm{OLS}}$. For the sake of clarity, we define

$$\mathbf{A} = (\mathbf{X}^\intercal\mathbf{X})^{-1}\mathbf{X}^\intercal$$

so that $\hat{\boldsymbol{\beta}}^{\mathrm{OLS}} = \mathbf{A}\mathbf{z}$. Moreover, we observe that

$$\mathbf{A}\mathbf{A}^\intercal = (\mathbf{X}^\intercal\mathbf{X})^{-1}.$$

Then we get:

$$\begin{aligned}\mathrm{Var}(\hat{\boldsymbol{\beta}}^{\mathrm{OLS}}) &= \mathbb{E}\left[(\mathbf{A}\mathbf{z} - \mathbf{A}\mathbb{E}(\mathbf{z}))(\mathbf{A}\mathbf{z} - \mathbf{A}\mathbb{E}(\mathbf{z}))^\intercal\right] \\ &= \mathbf{A}\,\mathrm{Var}(\mathbf{z})\mathbf{A}^\intercal \\ &= \sigma^2(\mathbf{X}^\intercal\mathbf{X})^{-1},\end{aligned} \tag{14}$$

where in the last equality we reminded that the random variables ε_i are uncorrelated, meaning that the covariance matrix of $\mathbf{z}$ is diagonal, with its elements given by (12).

We now briefly discuss the minimization of the cost functions (6) and (7), for Ridge and Lasso regression respectively. As regards the former, the solution to the optimization problem has a closed form expression:

$$\hat{\boldsymbol{\beta}}^{\mathrm{Ridge}} = (\mathbf{X}^\intercal\mathbf{X} + \lambda\mathbf{Id})^{-1}\mathbf{X}^\intercal\mathbf{z}, \tag{15}$$

which is very similar to the one for the OLS regression. The cost function for LASSO, however, is clearly non-differentiable at the origin. This makes it impossible to find an analytical expression for its minimum, so that iterative methods are needed to find it. One example is the coordinate descent, which is implemented in the built-in `sklearn.linear_model.Lasso` class [3].

The bias-variance tradeoff

Now that we outlined the three main regression methods we will use throughout the project, we take a closer look at the mean square error (3). The analysis described hereafter is mainly inspired by [4]. The MSE is a random variable, of which we want to compute the expectation value. It is important to remark that the stochasticity here is introduced at two different levels: (i) when the points $\mathbf{X}_{i,\star}$ of the dataset (1) are chosen and (ii) when the noise ε_i corresponding to $f(X_{i,\star})$ in equation (9) is generated in a Gaussian fashion. Thus, we can compute expectation values both over all possible datasets and over all possible instances of the noise. We denote the former as $\mathbb{E}_{\mathcal{D}}$ and the latter as $\mathbb{E}_{\boldsymbol{\varepsilon}}$, and

then write

$$\mathbb{E}_{\boldsymbol{\varepsilon}}\left\{\mathbb{E}_{\mathcal{D}}\left[(\tilde{z}_i - z_i)^2\right]\right\} = \mathbb{E}_{\boldsymbol{\varepsilon}}\left\{\mathbb{E}_{\mathcal{D}}\left[(f(\mathbf{X}_{i,\star}) + \varepsilon_i - \tilde{z}_i)^2\right]\right\}$$
$$= \mathbb{E}_{\boldsymbol{\varepsilon}}\left\{\mathbb{E}_{\mathcal{D}}\left[\left(f(\mathbf{X}_{i,\star}) + \mathbb{E}_{\mathcal{D}}(\tilde{z}_i) - \mathbb{E}_{\mathcal{D}}(\tilde{z}_i) + \varepsilon_i - \tilde{z}_i\right)^2\right]\right\}$$
$$= \mathbb{E}_{\boldsymbol{\varepsilon}}\left\{\mathbb{E}_{\mathcal{D}}\left[\left(f(\mathbf{X}_{i,\star}) - \mathbb{E}_{\mathcal{D}}(\tilde{z}_i)\right)^2 + \left(\mathbb{E}_{\mathcal{D}}(\tilde{z}_i) - \tilde{z}_i\right)^2\right.\right.$$
$$+ \varepsilon_i^2 + 2\left(\left(f(\mathbf{X}_{i,\star}) - \mathbb{E}_{\mathcal{D}}(\tilde{z}_i)\right)\left(\mathbb{E}_{\mathcal{D}}(\tilde{z}_i) - \tilde{z}_i\right)\right.$$
$$+ \varepsilon_i\left(\mathbb{E}_{\mathcal{D}}(\tilde{z}_i) - \tilde{z}_i\right)$$
$$\left.\left.+\varepsilon_i\left(f(\mathbf{X}_{i,\star}) - \mathbb{E}_{\mathcal{D}}(\tilde{z}_i)\right)\right)\right]\right\}.$$

If we note that

$$\mathbb{E}_{\mathcal{D}}\left[\mathbb{E}_{\mathcal{D}}(\tilde{z}_i) - \tilde{z}_i\right] = 0,$$
$$\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\varepsilon_i\right] = 0,$$

then the cross terms in the previous expression all cancel. Moreover, if we define:

$$\text{bias}^2 = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\mathbb{E}_{\mathcal{D}}\left(f(\mathbf{X}_{i,\star}) - \mathbb{E}_{\mathcal{D}}(\tilde{z}_i)\right)^2\right], \quad (16)$$

$$\text{variance} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\mathbb{E}_{\mathcal{D}}\left(\mathbb{E}_{\mathcal{D}}(\tilde{z}_i) - \tilde{z}_i\right)^2\right], \quad (17)$$

$$\text{error} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\mathbb{E}_{\mathcal{D}}\varepsilon_i\right] = \sigma^2, \quad (18)$$

we can rewrite the expectation value of the mean square error C in the following way:

$$\mathbb{E}_{\boldsymbol{\varepsilon}}\left[\mathbb{E}_{\mathcal{D}}\left(C\right)\right] = \text{bias}^2 + \text{variance} + \text{error} \quad (19)$$

The three quantities appearing in equations (16-18) are purposefully defined with their names, and the reason is apparent: the bias measures, on average, how much the prediction given by the model would differ from the value of the function we would like to learn; the variance measures the fluctuations in our prediction and finally the error is a measure of the noise in our data. In real life, we always have a limited amount of data available, and for this reason using too much a complex model could lead to a very high variance. This is the case when we come across overfitting: to avoid it is a common choice to restrict ourselves to models with a higher bias using regularization terms, which in turn help limit the variance. Finally, we emphasize three facts about the bias-variance-error derivation above.

1. We did not specify the cost function which is optimized in the training.
2. We did not say anything about the *true* function f which generates the data.
3. We did not utilize any assumption about the model which is fitted to the data.
4. We did not even specify if the dataset $\mathcal{D}$ is the train or the test set.

For these reasons, the result is valid in many machine learning problems, which go far beyond the elementary regression techniques on which we focus during this project.

Bootstrap and cross-validation

Last, but not least, we describe two resampling techniques which we are going to use to improve the estimate of the error of our model. The first one is the bootstrap: suppose we have a dataset $\mathcal{D}$ of the form (1) made up of n observations. We can generate a *bootstrapped* set $\mathcal{D}_i^\star$ by drawing n samples from $\mathcal{D}$ *with replacement*. This means that the same point can occur multiple times in $\mathcal{D}_i^\star$, which is now going to be used as a train set, yielding one estimate of the optimal parameters $\beta_i^\star$. The procedure is repeated k times: this allows us to have k different estimates of any function that can be evaluated using the parameters $\beta_i^\star$, for example the mean square error. In particular we can investigate the bias-variance tradeoff in this way: for every point X_i of the test set, we store the value of the prediction $z_{ij}^{(\text{pred})}$ given by the j-th model gotten using the bootstrap. We also denote as $z_i^{(\text{test})}$ the target value in the test corresponding to X_i. The following approximations hold:

$$\mathbb{E}_{\mathcal{D}}(\tilde{z}_i) \approx \frac{1}{k}\sum_{j=1}^{k} z_{ij}^{(\text{pred})} \equiv \mu_i,$$
$$f(\mathbf{X}_{i,\star}) \approx z_i^{(\text{test})},$$
$$\mathbb{E}_{\mathcal{D}}\left(\mathbb{E}_{\mathcal{D}}(\tilde{z}_i) - \tilde{z}_i\right)^2 \approx \frac{1}{k}\sum_{j=1}^{k}(\mu_i - z_{ij}^{(\text{pred})})^2.$$

With such expressions in mind, we can compute numerical approximations of the bias and the variance term appearing in equations (16) and (17):

$$\text{bias}^2 \approx \frac{1}{N}\sum_{i=1}^{N}(y_i^{(\text{test})} - \mu_i)^2, \quad (20)$$

$$\text{variance} \approx \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{k}\sum_{j=1}^{k}(\mu_i - z_{ij}^{(\text{pred})})^2\right]. \quad (21)$$

Hence, in our results we are able to study how the bias and variance change when we increase the complexity of the model and when we employ regularization techniques, for various values of the hyperparameter λ.

The second method we aim our attention to is cross validation, which helps us select the best λ in the training procedure, when doing Ridge and LASSO regression. In particular we would like to find the value of λ which minimizes the function $\text{MSE}(\lambda)$. The train set is split into K different disjoint subsets, or *folds*, in a way that they have all roughly the same size. We label them as S_i, with $1 \leq i \leq K$. For this reason this is also called

k-fold cross validation. Note that we are have used only the train set, because we do not want any data from the test set to leak during the procedure of the optimal hyperparameter λ. We fix the value of λ and for each value of i we proceed as follows: we perform the optimization step using $\{S_j,\ 1 \leq j \leq K \wedge j \neq i\}$ as train set and we then asses the MSE on the set S_i. We define the *validation error* as the average of all the values of the MSE which we got in this way, and we use it as our final estimate of the function MSE(λ). The procedure is repeated for different choices of λ, and we finally select the one that has yielded the minimum MSE.

II.2. Code structure

First of all, we define the Franke function as

$$\begin{aligned} f(x,y) = &\frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) \\ &+ \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ &+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) \\ &- \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right)\ . \end{aligned}$$

Using this expression, it is possible to generate a vanilla dataset of the kind described by equation (1), by the following procedure. We first generate two sequences $\{x_j\}$ and $\{y_j\}$, each consisting in M random numbers drawn from a uniform distribution in the interval $[0,1]$, and then construct all of the possible $N = M^2$ pairs (x_i, y_i). Next, we generate a sample of target, noisy data:

$$z_i = f(x_i, y_i) + \varepsilon_i, \tag{22}$$

where each ε_i is picked from a Gaussian distribution with mean 0 and variance σ^2. Our goal is now to fit a two variable polynomial of degree d to this dataset, which in the case $d = 2$ looks like this:

$$\tilde{z}(x,y) = \beta_0 + \beta_1 y + \beta_2 x + \beta_3 y^2 + \beta_4 xy + \beta_5 x^2\ . \tag{23}$$

Clearly enough, the following row vector gives exactly what we want when we multiply it by the column vector $\boldsymbol{\beta}$:

$$X_i = \begin{pmatrix} x_i^0 y_i^0 & x_i^0 y_i^1 & x_i^1 y_i^0 & x_i^0 y_i^2 & x_i^1 y_i^1 & x_i^2 y_i^0 \end{pmatrix}, \tag{24}$$

where we redundantly wrote every exponent to highlight a pattern which can be generalized to any degree d, and is summarized in algorithm 1. By looking at the `for` loop in the algorithm, we can calculate the number of columns in the design matrix. It is

$$1 + 2 + 3 + \cdots + (d+1) = \frac{(d+1)(d+2)}{2}.$$

Algorithm 1 Set up of the design matrix

```
j=0
for S = 0, 1, ..., d do
    for k = 0, 1, ... S do
        X_ij ← x_i^k y_i^{S-k};
        j ← j + 1;
```

Once we have our design matrix set up, we are almost ready to perform OLS regression. Before doing it, we split our data set using the built-in Scikit-learn function `train_test_split`. We implement linear regression, Ridge and LASSO in three different functions which make use respectively of equations (8), (15) and the library function `sklearn.linear_model.LassoCV`. Observe that the Ridge optimization procedure reduces to OLS when setting $\lambda = 0$, and we can take advantage of this to simplify our code. These three methods can be called inside a solver function, which also takes as input some flags, which enable the user to decide whether to use a resampling techinque or not. The whole code we developed is available at the following GitHub repository: `https://github.com/giammy00/FYS-STK4155.git`. Finally we describe how we decide to scale our dataset. It is customary in fact to shift and scale the design matrix as follows:

$$X^\star_{ij} = \frac{X_{ij} - \mu_j}{\sigma_j}, \tag{25}$$

where μ_j and σ_j are the mean and standard deviation of the j-th column. From this definition, it immediately follows that

$$\sum_{i=1}^{N} X^\star_{ij} = 0. \tag{26}$$

We now discuss why this turns out to be particularly useful when doing Ridge and LASSO regression. Remember that our model is a polynomial of the form:

$$\tilde{z}(x,y) = \beta_0 + \beta_1 y + \beta_2 x + \text{ higher order terms.} \tag{27}$$

Since the value of β_0 just shifts the whole prediction by a constant offset, it does not make very much sense to put a penalty on it when using regularization. To avoid this, when optimizing the cost functions (6) and (7), we compute the L_1 and L_2 norms as:

$$\|\boldsymbol{\beta}\|_p = \left(\sum_{j=1}^{p-1} |\beta_j|\right)^{1/p}$$

where the sum starts from $j = 1$ on purpose, and p is the number of parameters β, for a given polynomial degree. Now let us optimize the cost function obtained in this way, by differentiating it with respect to the parameter β_0:

$$\frac{\partial}{\partial \beta_0}\left[\sum_{i=1}^{N}\left(\beta_0 + \sum_{j=1}^{p-1}\beta_j X^\star_{ij} - z_i\right)^2 + \lambda\|\boldsymbol{\beta}\|_p^p\right] = 0\ .$$

The solution is seen to be

$$\begin{aligned}\beta_0 &= \frac{1}{N}\sum_{i=1}^{N} z_i - \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{j=1}^{p-1}\left(\beta_j X^\star_{ij}\right)\right] \\ &= \bar{z} - \frac{1}{N}\left[\sum_{j=1}^{p-1}\beta_j\left(\sum_{i=1}^{N} X^\star_{ij}\right)\right] \\ &= \bar{z},\end{aligned}$$

where we have let $\bar{z}$ be the average of all z_i, and reminded (26). Clearly, if we replace z_i with

$$z_i^\star = z_i - \bar{z}, \tag{28}$$

we get $\beta_0 = 0$. So, provided that we shift both the target values z_i and the columns of the design matrix by their mean value, a fit with no intercept seems to be a good candidate to remove the penalty term on β_0.

However, we need to ensure that such translation does not alter the optimal values of the other parameters. In order to check this, we add a generic constant term $c \in \mathbb{R}$ to every z_i and write down the optimization equation corresponding to β_j, with $j \neq 0$:

$$\frac{\partial}{\partial\beta_j}\left[\sum_{i=1}^{N}\left(\beta_0 + \sum_{j=1}^{p-1}\beta_j X^\star_{ij} - (z_i + c)\right)^2 + \lambda\|\boldsymbol{\beta}\|_p^p\right] = 0\,.$$

This translates into:

$$\begin{aligned}\sum_{i=1}^{N} X^\star_{ij}\left(\beta_0 + \sum_{j=1}^{p-1}\beta_j X^\star_{ij} - (z_i + c)\right) + \lambda\frac{\partial\|\boldsymbol{\beta}\|_p^p}{\partial\beta_j} = 0, \\ \sum_{i=1}^{N} X^\star_{ij}\left(\beta_0 + \sum_{j=1}^{p-1}\beta_j X^\star_{ij} - z_i\right) + \lambda\frac{\partial\|\boldsymbol{\beta}\|_p^p}{\partial\beta_j} = 0,\end{aligned}$$

where again equation (26) has been used. This is the same equation that we would get if $c = 0$, so we conclude that shifting the target values by any constant parameter c does not affect the optimization equations for any of the parameters β_j, the only exception is β_0 which is set to be zero by choosing $c = -\bar{z}$. Note that in these arguments we still have not used the fact that we scaled the columns of the design matrix in equation (25), so that their variance is 1. Why did we care about doing it, then? An intuitive argument follows. Suppose the variances of columns $X_{\star,\ell}$ and $X_{\star,j}$ of the design matrix are such that

$$\mathrm{Var}(X_{\star,\ell}) \gg \mathrm{Var}(X_{\star,j}),$$

then, for the corresponding optimal parameters we expect

$$|\beta_\ell| \ll |\beta_j|\,,$$

which would result in affecting some coefficients more than others during the regularization procedure. Scaling the columns of the design matrix prevents this from happening, so that all β_i's are treated on equal footing. To sum up, in algorithm 2 we outline a general procedure which can be used to perform linear regression:

Algorithm 2

generate the dataset;
build the design matrix using algorithm 1;
split it into $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{test}}$;
scale and shift the columns of the design matrix using (25) (see footnote 1);
shift the target using (28);
carry out the training procedure (using either OLS, Ridge or LASSO), with no intercept, to find optimal parameters β_i, with $i \neq 0$;
Set $\beta_0 = \bar{z}$;
compute prediction on $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{test}}$, using (2).

This is the core of our code, which can be re-used multiple times to carry out analyses of what we discussed earlier, e. g. the bias-variance tradeoff and the different resampling techniques. This is the topic of next section, where we first present a series of test runs which we do using the vanilla data generated by Franke function. The code is implemented in such a way that different kind of datasets can be given as inputs, so we later move on to the analysis of real data, which describe the altitude profile of a geographical region.

III. RESULTS AND DISCUSSION

III.1. Tests of code using the Franke function

In this section we present some code tests that we are able to perform on a dataset of $16 \times 16 = 256$ points (x, y), generated using the Franke function. We always split them into train and test as described in section II. Namely, we decide to use the 80% of our data as train set, and the remaining part as test set. We use the input parameter `random_state` of Scikitlearn's `train_test_split` function, so as to guarantee reproducibility of the split across multiple function calls. We choose a standard deviation of 0.1 for the Gaussian noise appearing in equation (22). Our first goal is to study the quality of the prediction on both the train and test set, when doing OLS regression. To do so, we can calculate the MSE (defined in equation (3)) and the R^2 score, in order to study their behaviour when the degree d of the polynomial changes. The R^2 score is defined as

$$R^2(\mathbf{z}, \tilde{\mathbf{z}}) = 1 - \frac{\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2}{\sum_{i=0}^{n-1}(z_i - \bar{z})^2}, \tag{29}$$

where a value of $R^2 = 1$ means a perfect fit. The result is plotted and commented in figure 1.

[1] Here, it is important that we compute μ_j and σ_j using only the train data, because we do not want the train set to be influenced in any way by the test set.

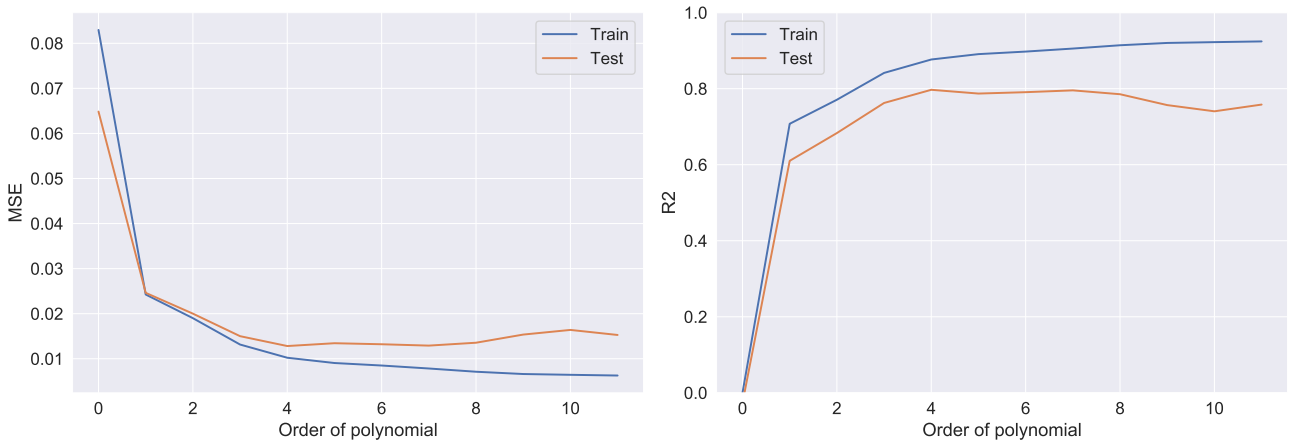

Figure 1: MSE and R^2 score as function of the model complexity. Recall that a perfect prediction gives MSE= 0 and $R^2 = 1$. Models have been fitted using OLS regression. We clearly see two trends: when the complexity is very low, increasing it improves the scores both on the test and on the train set. However, when it becomes too large, the prediction keeps improving on the train data, while worsening on the test set, which is a clear indicator of overfitting.

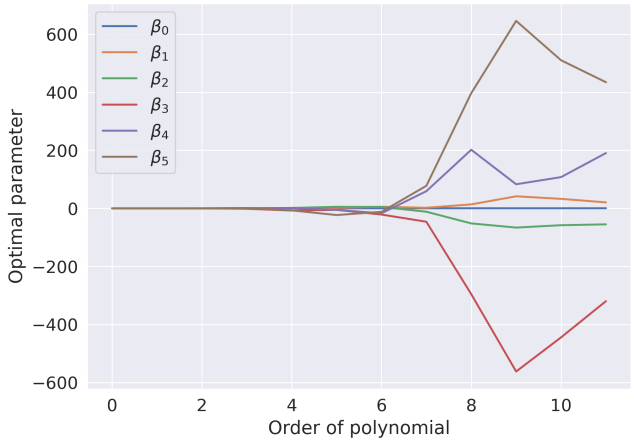

Figure 2: Behaviour of the optimal parameters corresponding to the 6 lowest order term in the polynomial. β_0 is the intercept, and which term corresponds to each β_i can be inferred from agorithm 1. Apparently, the values blow up when the degree of the polynomial reaches the value $d \sim 7$. This behaviour is particularly evident when looking at β_j with $3 \leq j \leq 5$, which correspond to the second degree terms in the polynomial.

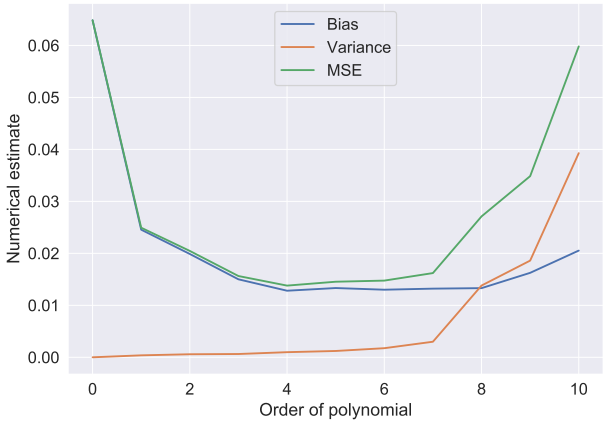

Figure 3: The bias variance tradeoff is investigated using the bootstrap resampling. We performed as many bootstraps as the number of elements in the train set. The trend is clear: when the complexity is low, we have a high bias, but low variance. Conversely, as we increase the polynomial degree, we increase the variance, which is a sign of overfitting, while the bias goes down. The MSE plot is obtained by assessing the mean

As we can see by looking at the scores in the figure, overfitting occurs when a model is so complex that it can also reproduce the noise in the data. When increasing the polynomial degree, we know that large fluctuations in the parameters β_i can be another telltale sign that overfitting is happening. We portray this behaviour for some of these coefficients in figure 2.

We now use the bootstrap resampling technique in order to study the bias-variance tradeoff, which will give a further insight into the issue of overfitting. We stick to ordinary least squares regression, and we assess estimates of the bias and variance using equations (20) and (21). The results are shown in figure 3. Now we remind that resampling techniques are particularly useful when we want to investigate about the probability distributions of estimators that can be computed as functions of the optimal parameters. In particular, both the cross-validation and the bootstrap can be employed to obtain estimates of the out-of-sample MSE. Observe that here we are not using cross validation to tune a regularization parameter, but we limit ourselves to look at the statistics of the test error estimator which is evaluated on the different test folds. A comparison between the two cases is portrayed in figure 4. We now focus on Ridge and LASSO regression, and we aim to study how the test error behaves for different complexities and values of the regularization parameter λ. We

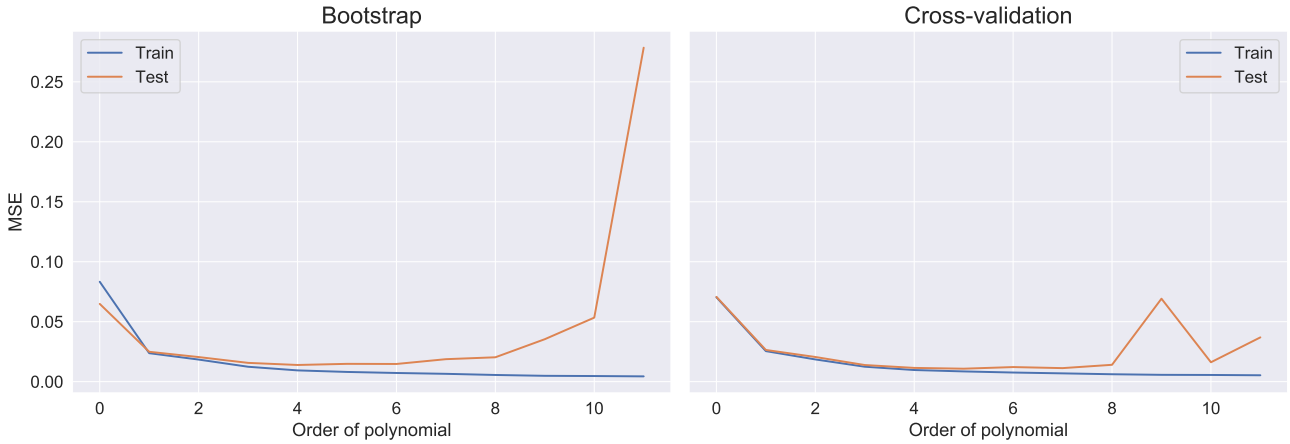

Figure 4: We here compare the estimates of the test and train MSE for both the bootstrap and the cross-valiation resampling. As expected, the trend is very similar to the one portrayed in figure 1. We notice that for polynomial of high degree, the bootstrap estimates of the test error are way larger than the ones gotten by cross-validation.

do the same thing (i) for the test errors of single regressions which use the whole train set once, and (ii) for the cross-validation test errors, in the exact same way as we did in the right side of figure 4. In figures 5 and 6 we portray the results of this analysis with the aid of 2D colormaps. The effect of the regularization is quite clear in all figures: when the degree d of the polynomial is high, a large value of λ helps reduce the MSE. We notice, however, that the MSEs look a bit more noisy in figure 6 than in figure 5, when a single regression is carried out using all the train set. We believe this is due to the stochasticity which is introduced in the training procedure, at the level of splitting the train set into different folds. In fact, when we change d and λ, we repeat the k-fold cross validation with a different splitting of the train set into different folds. It is possible to avoid this by setting the `random_state` parameter of the Scikit's `Kfold` function so that the randomness is reproducible across multiple function calls. Despite this qualitative difference, the overall trend is always roughly the same: when the complexity is fixed, there is some intermediate value $\lambda = \lambda_{\min}$ for which we find a minimum MSE. Choices of larger and smaller λ result in an increase of the MSE, which is not so unreasonable: when the regularization is too strong, it prevents our model to capture the main trends in the data, whereas if it is too weak, it might not succeed in preventing overfitting from happening. We can see this in the left-most columns of the figures. Such kind of grids turn out to be really useful when we need to tune the hyperparameter in regularized regressions, and for this reason we portrayed them here.

Our last test consists in an analysis of how the bias variance tradeoff is influenced by the regularization hyperparameter. In particular, figure 7 shows the behaviour of the bias and the variance, together with the MSE, for three different values of λ. For any given polynomial degree, we see that the variance is dragged down when we increase the regularization parameter. This is expected both in Ridge and LASSO regression, because we know that the penalty in the cost function translates into a constraint on the magnitude of the vector $\boldsymbol{\beta}$. The higher the penalty, the stricter the constraint[2]. A higher bias is therefore expected, which is also what we see, even if the change is a bit less dramatic than the one we see for the variance. Despite these quantitative changes, which make the figure look different from the plot in figure 3, we observe the same trend: increasing the complexity of the model leads to a higher variance and a lower bias. In the plot corresponding to LASSO regression, note that for high levels of regularization, the three quantities seem to not to change too much when turning to polynomials of higher degrees. A possible explanation for this, which would need to be verified, is that LASSO drags some of the coefficients down to 0 (a more detailed discussion of this can be found for example in section 3.4.3 of [2]). Finally, we note that the MSE is roughly the sum of the two other quantities, which is what we expect by looking at equation (19) and neglecting the error term.

III.2. Analysis of terrain data

All the codes we have developed can be used to find the polynomials which best fit the altitude profile of a geographical region. In particular, the dataset we have available describes a territory near the lake of Møsvatn Austfjell (Norway), consisting in a grid of points (x, y), together with the corresponding altitude relative to the lake (in meters), which we label as $h(x, y)$. It is possible to visualize the dataset in the colorscale plots represented in figure 8.

We select two ($\mathcal{N}$ pixels) $\times$ ($\mathcal{N}$ pixels) regions, with $\mathcal{N} = 50$, in such a way that their orographical features look qualitatively different. They are shown in the left-most and center plots in figure 8; for the sake of clarity, we refer to them respectively as territory A and B. We consider them as two different, independent datasets, and we now discuss how we can employ all of the re-

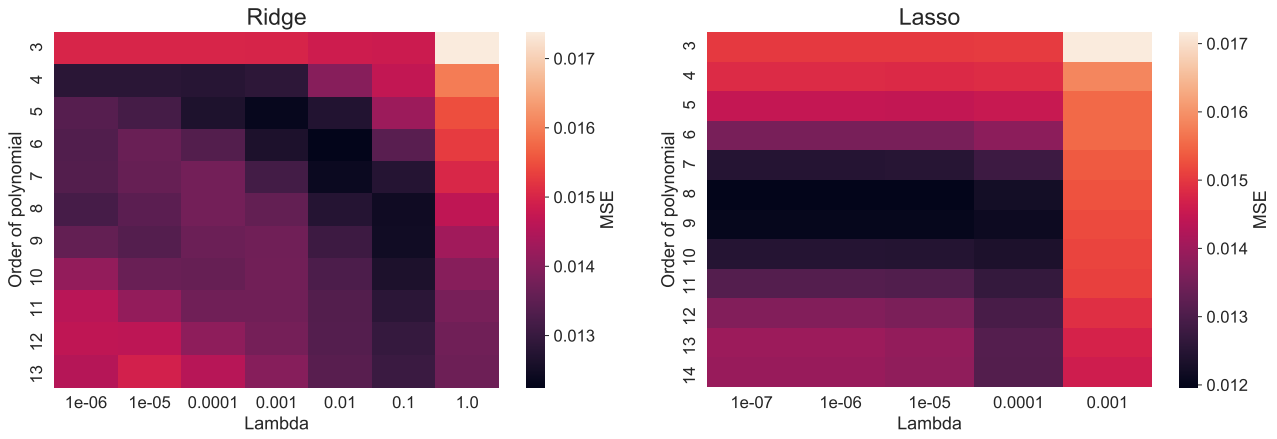

Figure 5: Evaluation of the test mean square error for both Ridge and LASSO single regression. The effect of the regularization parameter seems to be more evident in the Ridge regression, while remaining a bit more disguised in LASSO: along the rows of the corresponding plot, in fact, the validation error stays rather constant.

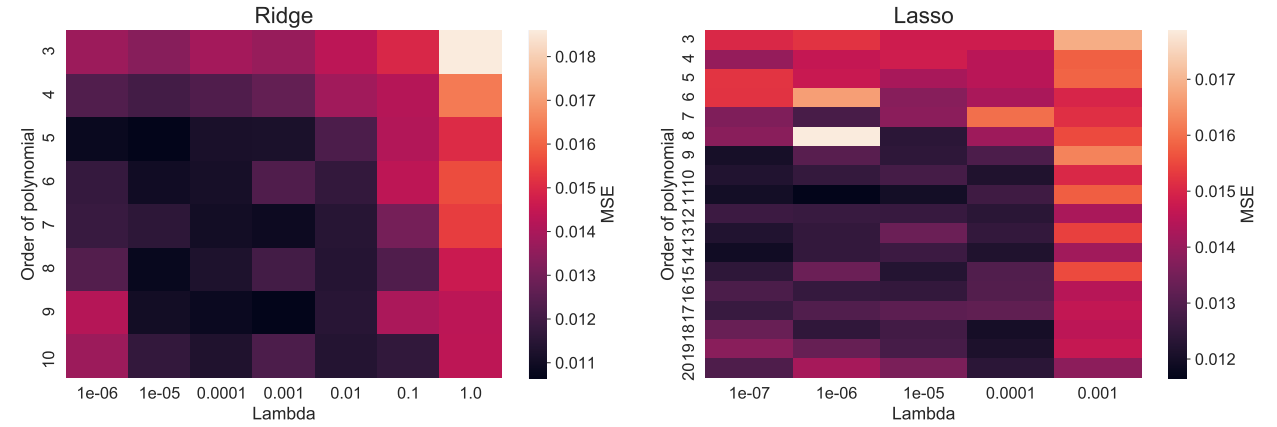

Figure 6: The plots in this figure are analog to the ones in figure 5. The difference is that the MSE here is each time assessed by performing a 5-fold cross validation, by averaging over all the test fold errors. The trend we observe is roughly the same, although here we see some noise, probably due to the randomness of the k-fold split.

gression techniques we have discussed and tested so far, in order to fit polynomials of increasing complexity to such altitude profiles.

First of all, we need to say something about how we scale the inputs (x, y). In principle, they are integers describing each pixel location, so that they can be much larger than unity. However, we need to remind that when setting up the design matrix we compute powers of x and y. Now suppose $x \sim 40$, then we might encounter numerical instabilities for polynomials of high degree: for example if $d = 18$ then x^d would be insanely huge. It is therefore more convenient to have our x and y scaled down to be evenly spaced in the interval $[-1, 1]$. Aftwerwards we can set up the design matrix and finally scale it as we described in section II. We experienced worse results when doing the regression with the original pixel values (x, y) than with (x, y) of order unity. Because pixel values are arbitrary units, this change of inputs is not a big deal.

With the inputs (x, y) of proper size, we start with a plain, single OLS regression, and in this case we select the polynomial degree which yields the minimum MSE evaluated on the test set. This can be visualized in the two plots at the top of figure 12.

Ridge and LASSO regression were utilized to perform polynomial fits of the same kind, but with the added hyperparameter λ. We already mentioned how we can use resampling methods in order to tune the λ parameter. Essentially, we perform a 5-fold cross validation, and construct a 2D matrix containing the resulting test MSE, exactly like the ones portrayed in figures 5 and 6. In this case, we scan the range $3 \leq d \leq 25$ for the polynomial degree, and 20 evenly spaced values of λ in the ranges $\left[10^{-14}, 10^{-5}\right]$ and $\left[10^{-10}, 10^{1}\right]$ for LASSO and Ridge respectively. It is important that we use only the train set to construct such grid, and that we keep apart a test set which does not enter this procedure at all. Once the best λ and model degree d have been identified, we use them to carry out the optimization procedure on the whole train set to find the best parameters $\boldsymbol{\beta}$. Finally, we evaluate the performance of our model on the test set, which we had put aside at

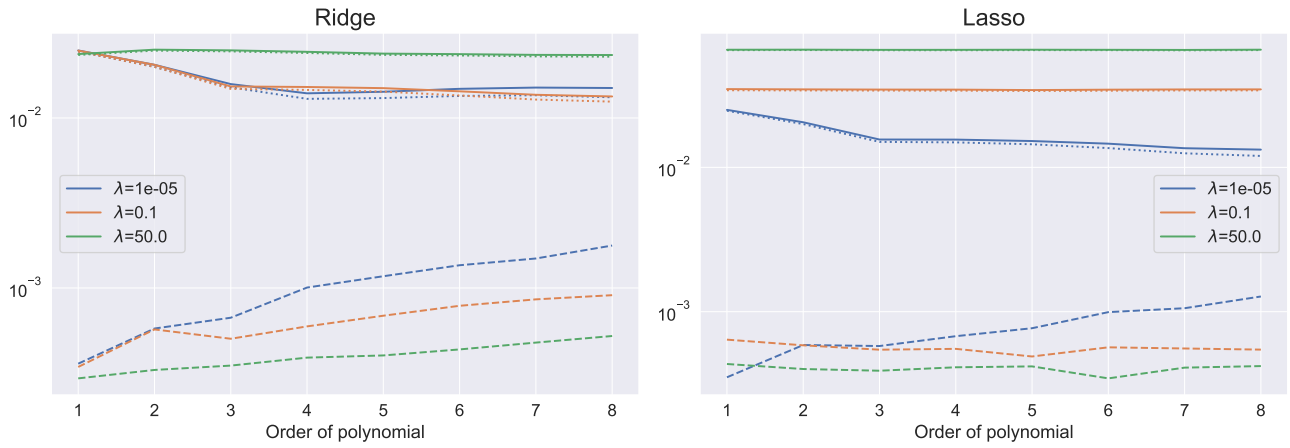

Figure 7: Bias-variance tradeoff in Ridge and LASSO regression, for different values of λ. For each of them, we display a dotted, a dashed and a solid line, which respectively depict the bias, the variance, and the mean square error on the test set. Observe that the scale is logarithmic.

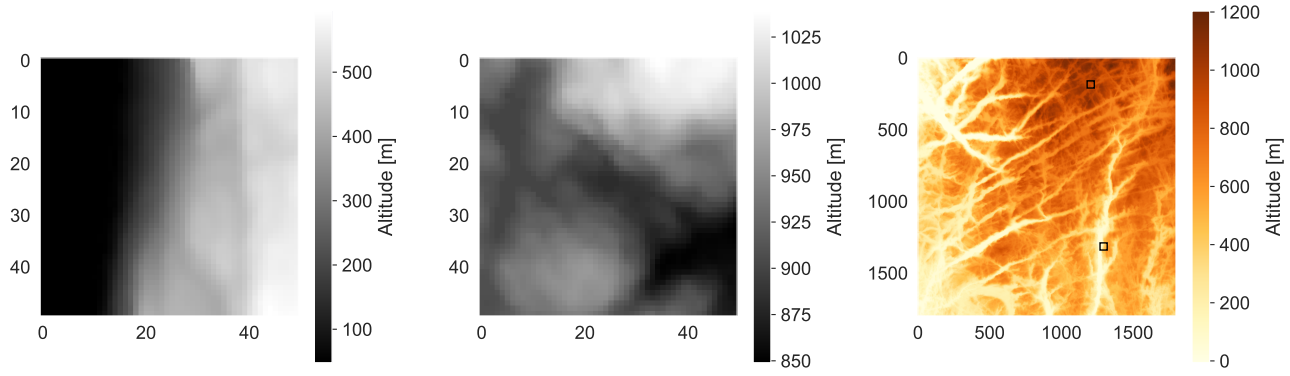

Figure 8: The rightmost image depicts the map of the region from which we import the datasets. We focus on two regions, each consisting in a 50×50 grid of pixels. The other two figures correspond to the small black squares which are visible in the map. We are going to fit polynomials to the altitude profile of such regions.

the beginning. This is shown in figure 12, as function of the polynomial degree. We check that the test error gets smaller for more complex models.

Had this set influenced the choice of the best parameters $\boldsymbol{\beta}$ and/or hyperparameter λ in any way, this estimate of the test error would not have been reliable and we would have fallen into a so-called data leakage. This is a very important issue to keep in mind when developing any machine learning algorithm. In figures 9 and 10, we summarize the hyperparameter tuning in the case of the two selected regions and Ridge regression. In both cases, we show only the behaviour of MSE(λ) for the polynomial complexities which yielded the best validation error. Interestingly, the LASSO validation error seeems not to be influenced very much by the value of λ in the scanned range. This negligible effect of the hyperparameter suggests that LASSO may be a redundant method for fitting this kind of terrain data, and OLS or Ridge may instead be used. The resulting prediction on all the grid points (*i.e.* encompassing both the training and the test prediction, for aesthetical reasons) is depicted in figure 11 for all three methods and for both regions of interest, region A and region B.

In all cases, the similarity between the predictions and the target (figure 8) is apparent. This indicates the well functioning of our regression algorithms. It seems that the LASSO produces a model which looks a bit smoother than the other two, which in turn look way more complex and more similar to the true altitude profiles. This is a demonstration of LASSO's characteristic effect of forcing the $\boldsymbol{\beta}$ coefficients to zero, leaving us with a flatter looking surface. Note also the high polynomial degrees of the fits, with $d \sim 20$. There apparently was no overfitting even at these high degrees. Keep in mind a degree 20 polynomial has 231 coefficients, so we have 231 features, which is quite a large amount. That said, the terrain we fitted was bumpy. We know that a polynomial of degree d has at most $d-1$ local extrema, corresponding to bumps in the terrain. Hence with a lot of bumps, the degree of the fit must perforce be high. But it will not get arbitrarily high, because our model is selected based on its performance on test data, and therefore we preclude overfitting.

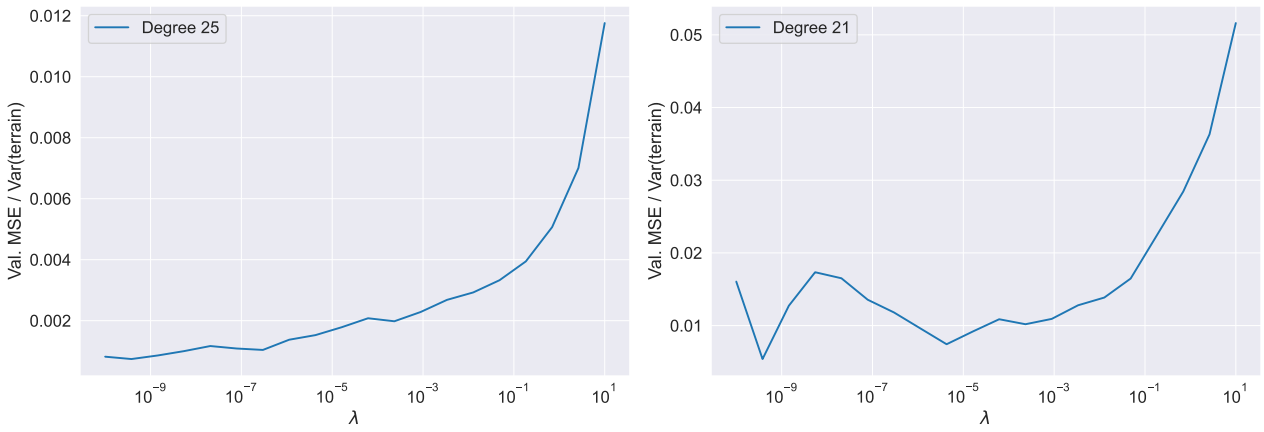

Figure 9: Hyperparameter tuning in Ridge regression, using both regions A (left) and B (right). Only the scores for the indicated polynomial degrees are represented.

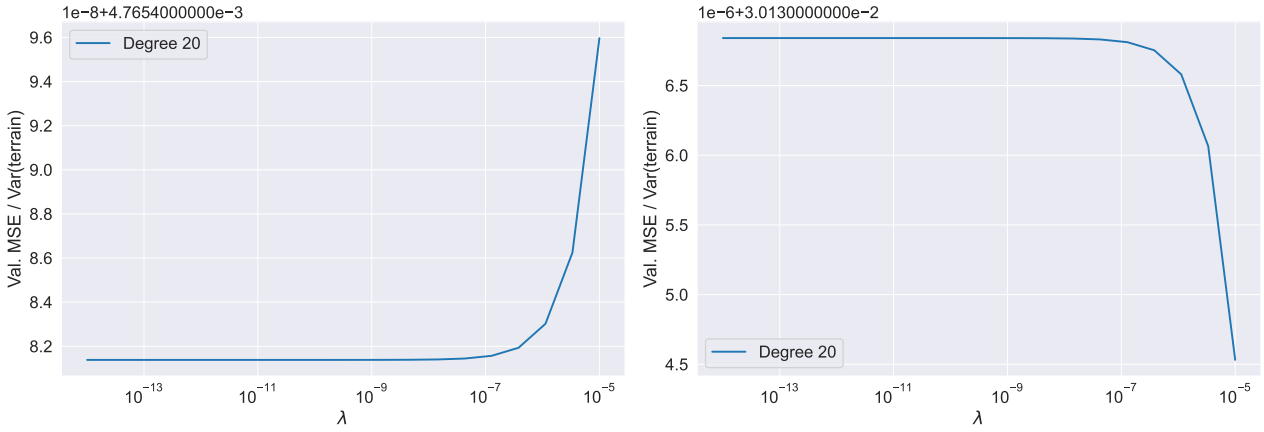

Figure 10: The plot here is the same as in figure 9, now for LASSO regression. Observe that on top of each graph there is a number of the form $m + q$, written in the standard scientific programming notation. The y-axis in these plots has been shifted and scaled, so that $y^{\text{displayed}} = m \times y^{\text{original}} + q$, in order to better visualize the results.

We can quantitatively assess the quality of our fits by computing the mean relative error, defined as

$$\epsilon = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{z_i - \tilde{z}_i}{z_i} \right|, \tag{30}$$

where the sum here runs over all the data points.[2] The values we find for each fit are displayed in table I. We observe that we get errors that are approximately 10 times larger in territory A than in territory B. This might be due to the step-like profile of region A, which is probably harder to fit using a polynomial. Higher complexities could be investigated together with regularization techniques, in order to improve the quality of these results. Also, we see that the model obtained via LASSO performs about 3 times worse than the Ridge and the OLS ones. This is consistent with what we glimpsed when looking at figure 11, noticing that the LASSO predictor looks smoother and less accurate than the other two. To sum up, the predictions have in all cases a satisfying degree of accuracy. This suggests that the regression methods are suitable for fitting terrain profiles, with maybe OLS and Ridge more so than LASSO.

method	ϵ_A	ϵ_B
OLS	1.8%	0.25%
Ridge	1.6%	0.25
LASSO	6.3%	0.62%

Table I: Relative errors for the different regression methods employed.

[2] Analogous relative errors could be defined by limiting the summation either to the train or on the test set.

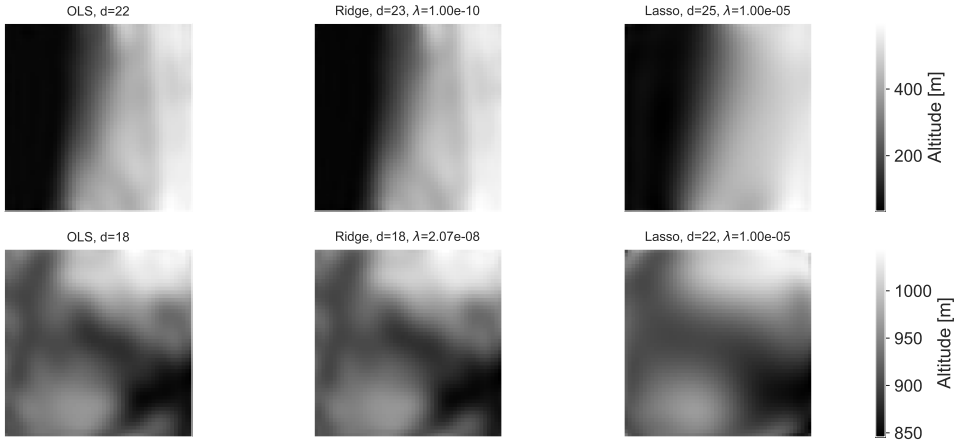

Figure 11: Prediction of the trained models on the two selected regions, using single OLS, Ridge and LASSO. In the latter two, the optimal degree d and regularization parameter λ have been chosen using a 5-fold cross validation. The first and second row correspond to region A and B, respectively. The corresponding targets are the ones in figure 8.

IV. CONCLUSION

In this project we studied linear regression as low level machine learning technique. We tested our code using the Franke function to generate our datasets, and reproduced some of the main results which are known in the literature. In particular, we were able to study the tradeoff between the bias and the variance of the model, which we verified to be the two main contributions to the mean square error. We then saw how this trend is affected by Ridge and LASSO regularization methods. Secondly, we looked for the best-fit polynomial to some real terrain data. The cross-validation resampling method was used to select the best hyperparameter λ and degree d in the training procedure. The models we get reproduce the terrain shape with average accuracies of order 0.1% - 1%.

What could have been done better?

We first note that we only explored a limited portion of the hyperparameter space, and also a restricted amount of polynomial degrees. It would be interesting to see how our results change if we bring these investigations further, especially while using the LASSO regression, where we observed that varying λ in the range of interest seemed to almost always have negligible impact on the MSE. We also realize that this project might be the starting point for a deeper research into the whole terrain dataset we had available, and in general to other geographic locations. Any grid representing the altitude of a geographical region can be in fact split into $\mathcal{N}$ small squares of suitable size. In principle, it is possible to use the regression techniques we have developed to fit $\mathcal{N}$ different polynomials, one for each square cell of territory. Thus it would become possible to render the whole territory in a 3D smooth fashion, using a suitable graphic sowtare. It would also be curious to investigate different families of models, such as linear combinations of trigonometric functions, and see how well they can capture the most relevant trends in the data.

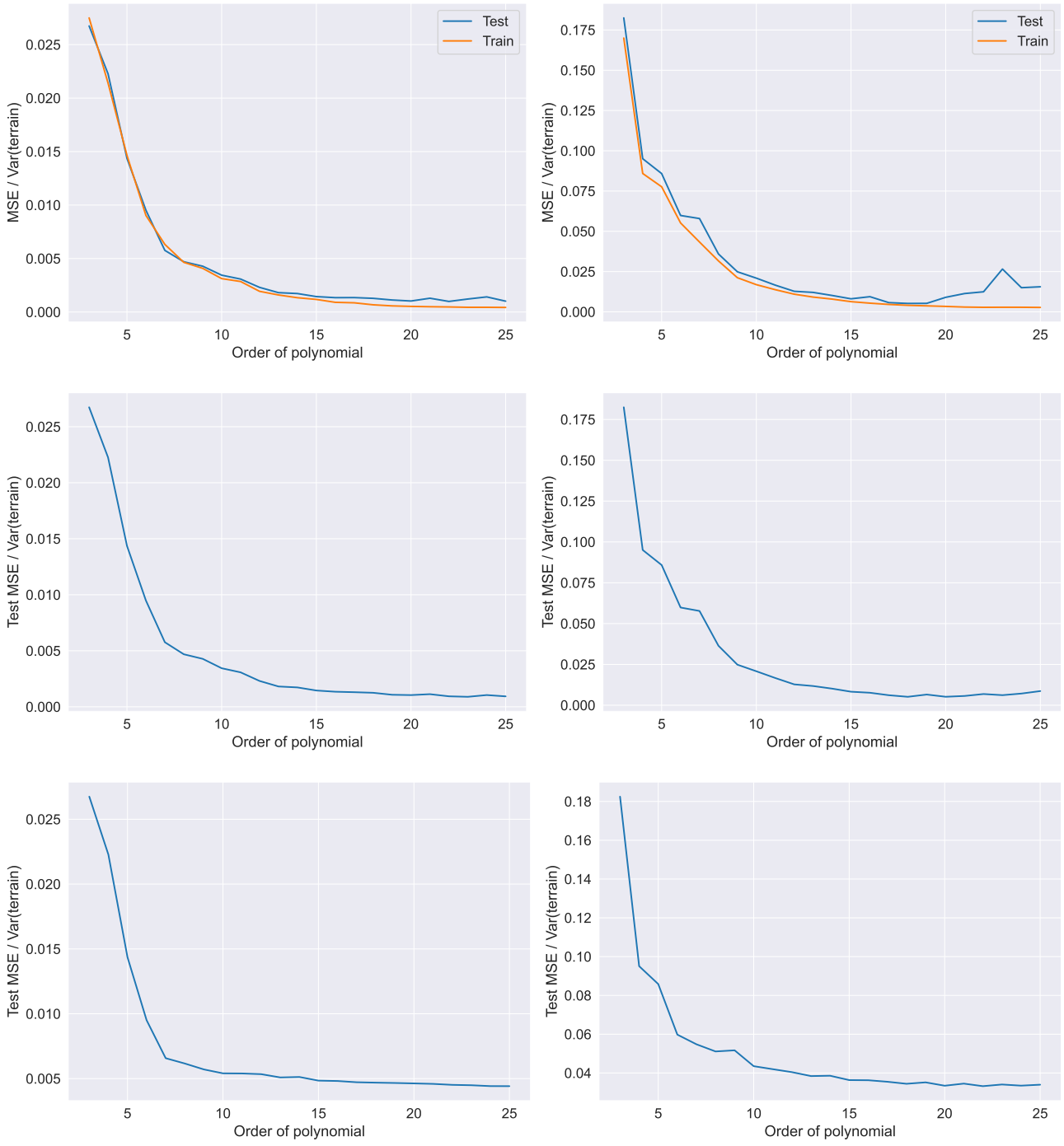

Figure 12: MSE evaluated both in the case of single OLS (first row) and regularized regressions (second row: Ridge, third row: LASSO). In the latter case we display the MSE which is obtained on the test set after tuning the parameter λ by performing cross-validation on the train set.

[1] A. Legendre, *Nouvelles methodes pour la determination des orbites des cometes*, 1805.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY: Springer New York, 2009. [Online]. Available: http://link.springer.com/10.1007/978-0-387-84858-7

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[4] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, "A high-bias, low-variance introduction to machine learning for physicists," *Physics Reports*, vol. 810, pp. 1–124, may 2019. [Online]. Available: https://doi.org/10.1016%2Fj.physrep.2019.03.001