

Project 1

FYS-STK4155

Håvard Skåli, Erik Røset & Oskar Idland
University of Oslo, Department of Physics
(Dated: October 6, 2024)

We have explored various regression techniques and resampling methods within the context of machine learning, motivated by the need to develop robust models that can accurately predict and generalize from complex datasets. The main objective was to analyze and compare the performance of Ordinary Least Squares (OLS), Ridge, and Lasso regression in fitting synthetic and real-world data, focusing on the bias-variance tradeoff and model generalizability. We applied these regression techniques to the Franke function, a synthetic benchmark used in numerical analysis, and extended our analysis to cosmological N-body simulation data generated using the public GASOLINE2 code SPH code. To assess model performance and generalization, we employed resampling methods such as bootstrap and k-fold cross-validation, examining how they help to evaluate model accuracy under different training and test data conditions. Our study provided insights into how model complexity and regularization parameters affect the bias-variance tradeoff and prediction error, enabling us to critically evaluate each regression technique's performance using statistical and resampling approaches.

rewrite /remove last part when results are gathered and discussion/conclusion is finished

<https://github.com/Oskar-Idland/FYS-STK4155-Projects>

I. INTRODUCTION

In many scientific and practical applications, developing models that can accurately predict outcomes and generalize to new data is a critical challenge. Complex datasets often contain noise and exhibit nonlinear relationships, making it essential to employ robust regression techniques that balance model flexibility with the risk of overfitting. This work is motivated by the need to understand how different regression methods perform when applied to synthetic and real-world data, and how resampling techniques can aid in evaluating model accuracy and generalizability. The primary goal is to develop a solid understanding of the Ordinary Least Squares (OLS), Ridge and Lasso regression techniques, and their application to both synthetic and real-world data. We also intend to investigate how resampling methods such as bootstrap and cross-validation can help assess model performance.

We begin by applying the abovementioned methods to the Franke function, a widely used test function in numerical analysis, and extend the analysis to cosmological N-body simulation data made with the public version of the GASOLINE2 Smoothed Particle Hydrodynamics (SPH) code [1]. Our approach involves fitting polynomial regression models of varying complexity to the Franke function as well as the simulation data, and studying the bias-variance tradeoff, an essential concept in machine learning, in both cases. This allows us to evaluate the impact of model complexity, regularization parameters and the size of training data on the accuracy and generalizability of the models. The overarching aim is to gain insights into how different regression methods handle overfitting, model complexity and data variability, and to develop a framework for critically evaluating

model performance using statistical and resampling techniques.

In section II we present relevant background theory, including central concepts such as OLS, Ridge and Lasso regression, model bias, variance and the bias-variance tradeoff, as well as two crucial resampling methods; bootstrapping and cross-validation, both of which will be implemented in this work. A few of the most important expressions introduced in this section are derived in appendix B. Our methodology is explained in section III, specifically how we define the essential design matrix, scale our data and implement the regression and resampling techniques. We also give an overview of our code structure and present the cosmological simulation data. In section IV we present, interpret and discuss the results of our analyses in light of expectations based on our previous knowledge of the implemented regression and resampling methods. In appendix C we include some additional figures that are not essential to our discussions, yet still referred to in the aforementioned section because they are necessary in reasoning our choice of presented results. Lastly, we summarize and conclude the main findings of our work in section V, and provide some open questions for further exploration. maybe shorten/edit

II. THEORY

A. Regression Analysis

Regression analysis is a fundamental statistical technique used to model the relationship between one or more independent variables (also known as predictors or features) and a dependent variable (or target). The goal of regression analysis is to find the mathematical relation-

ship that best explains the variation in the dependent variable based on the values of the independent variables.

Given a dataset consisting of n data points $\{(\mathbf{x}_i, y_i)\}_{i=0}^{n-1}$, where \mathbf{x}_i represents the input features (which is a vector in the case of multiple features), and y_i is the corresponding target value, the goal is to find a model that predicts \tilde{y}_i based on x_i such that it is as close to y_i as possible. For a linear model, the relationship between x_i and \tilde{y}_i can be expressed as

$$\tilde{y}_i = \mathbf{x}_i^\top \boldsymbol{\beta} = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}, \quad (1)$$

where \mathbf{x}_i^\top is the vector of input features for the i -th data point and $\boldsymbol{\beta} = [\beta_0, \beta_1, \dots, \beta_p]^\top$ is the vector of regression coefficients that we want to estimate. This can be written as a full-fledged matrix equation by using the so-called design or feature matrix $\mathbf{X} \in \mathbb{R}^{n \times (p+1)}$, which contains all input features and the bias term (so that the intercept β_0 is included):

$$\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}. \quad (2)$$

write differently?

At its core, regression analysis seeks to find a mathematical function that relates the independent variables to the dependent variable. In its most basic form, the relationship between a dependent variable \mathbf{y} and an independent variable \mathbf{x} is modeled as

$$\mathbf{y} = f(\mathbf{x}) + \boldsymbol{\epsilon}, \quad (3)$$

where $f(\mathbf{x})$ is the function we are trying to estimate, which represents the relationship between \mathbf{x} and \mathbf{y} . This is then what we approximate with our model $\tilde{\mathbf{y}}$. Furthermore, $\boldsymbol{\epsilon}$ is the error term, representing the part of the variation in \mathbf{y} that is not explained by the model (due to noise or other unobserved factors).

1. Ordinary Least Squares

The Ordinary Least Squares (OLS) method is one of the most fundamental and widely used techniques in regression analysis. Its objective is to find the best-fitting line or curve for a given set of data by minimizing the sum of the squared differences between the observed values and the predicted values. These squared differences are called errors, hence the sum of squared errors (SSE):

$$\text{SSE} = \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2. \quad (4)$$

Minimizing this sum is equivalent to minimizing the cost function

$$C(\mathbf{X}, \boldsymbol{\beta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}), \quad (5)$$

since the solution to this minimization problem gives the optimal values of $\boldsymbol{\beta}$.

To minimize $C(\boldsymbol{\beta})$, we take the derivative of the cost function with respect to $\boldsymbol{\beta}$ and set it to zero:

$$\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -2\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = 0. \quad (6)$$

This gives the normal equation

$$\mathbf{X}^\top \mathbf{X}\boldsymbol{\beta} = \mathbf{X}^\top \mathbf{y}, \quad (7)$$

which, by solving for $\boldsymbol{\beta}$ yields the OLS estimator:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \quad (8)$$

This formula provides the best-fitting coefficients $\boldsymbol{\beta}$ that minimize the sum of squared errors. In appendix B 1 we derive the following important results:
edit? maybe move up here?

$$\mathbb{E}[\hat{\boldsymbol{\beta}}] = \boldsymbol{\beta}, \quad (9)$$

$$\text{Var}[\hat{\boldsymbol{\beta}}] = \sigma^2 (\mathbf{X}^\top \mathbf{X})^{-1}. \quad (10)$$

Here $\mathbb{E}[\hat{\boldsymbol{\beta}}]$ and $\text{Var}[\hat{\boldsymbol{\beta}}]$ are the expectation value and variance of the OLS estimator, respectively, while σ^2 is the variance of the error term $\boldsymbol{\epsilon}$.

2. Ridge

While OLS provides a foundational method for regression, it can lead to problems when the data has high multicollinearity or when there are more features than data points, leading to overfitting. OLS attempts to minimize the sum of squared errors, but it does not impose any restrictions on the model complexity. This often results in high variance when the model learns to fit noise in the training data, especially when the design matrix \mathbf{X} is poorly conditioned (i.e., when columns of \mathbf{X} are nearly linearly dependent). This is because the matrix $\mathbf{X}^\top \mathbf{X}$ becomes close to singular, making its inverse highly sensitive to small changes in the data.
remove last sentences?

To mitigate the issue with OLS, Ridge regression introduces a regularization term to the cost function (5), penalizing large coefficients and preventing the model from becoming overly complex. The modified cost function is

$$C_{\text{Ridge}}(\mathbf{X}, \boldsymbol{\beta}) = \frac{1}{n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_2^2, \quad (11)$$

where $\lambda \geq 0$ is a hyperparameter that controls the strength of the regularization, and the subscripts 2 simply mean that we are taking the L^2 norm. This is defined as

$$\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}.$$

We see that when $\lambda = 0$, Ridge regression reduces to OLS, as the regularization term disappears. On the

other hand, when λ is large, the coefficients β must shrink to minimize the cost function, potentially reducing overfitting. It is important to note, however, that if λ becomes too large the coefficients tend to zero, and we may end up with underfitting instead. We therefore need to determine the hyperparameter carefully in order to find the perfect balance between the two extremes. formulate differently?

To minimize this altered cost function, we again take the derivative with respect to β and set it to zero:

$$\frac{\partial C_{\text{Ridge}}(\beta)}{\partial \beta} = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\beta) + 2\lambda\beta = 0. \quad (12)$$

Rearranging this equation gives us the Ridge regression normal equation:

$$(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})\beta = \mathbf{X}^\top\mathbf{y}, \quad (13)$$

where \mathbf{I} is the identity matrix. Notice the term $\lambda\mathbf{I}$ added to $\mathbf{X}^\top\mathbf{X}$, which makes the matrix invertible even when $\mathbf{X}^\top\mathbf{X}$ is poorly conditioned. This leads to the Ridge regression estimator:

$$\hat{\beta}_{\text{Ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}. \quad (14)$$

Here, the regularization term λ shrinks the coefficients and prevents them from becoming too large, thereby controlling the model's variance. remove this paragraph?

3. Lasso

Like Ridge regression, Lasso regression (Least Absolute Shrinkage and Selection Operator) is a regularization technique designed to improve the generalizability of the model by introducing a penalty term to the cost function. While Ridge regression uses the L^2 norm for regularization, Lasso regression employs the L^1 norm, defined as

$$\|\mathbf{x}\|_1 = \sum_i |x_i|.$$

This leads to a different form of regularization, since the cost function now takes the form

$$C_{\text{Lasso}}(\mathbf{X}, \beta) = \frac{1}{n} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \|\beta\|_1. \quad (15)$$

The difference from Ridge regression has important implications, mainly because the L^2 norm makes it impossible for any of the coefficients to vanish completely. correct? This can be problematic in high-dimensional datasets where we might expect many features to be irrelevant. By using the L^1 norm, Lasso regression has the unique ability to drive some coefficients to exactly zero, effectively performing both regularization and feature selection. Thus, Lasso regression not only addresses overfitting but also simplifies the model by automatically excluding irrelevant features, making it particularly useful for sparse models where only a subset of features are truly important. double check, introduce expressions, cite?

B. Properties of Predictive Models

maybe change structure, avoid repetitiveness

1. Predicted Values

In the context of predictive modeling, $\tilde{\mathbf{y}}$ represents the model's predicted values of the target variable \mathbf{y} . These predictions are made based on the features and parameters learned from the training data. For new or unseen data, the true target values \mathbf{y} are often unknown, but the model generates an estimate $\tilde{\mathbf{y}}$ that approximates these values.

The performance of the model's predictions can be analyzed by decomposing the error into several components. The error of $\tilde{\mathbf{y}}$ stems from three main sources: noise variance σ^2 , model bias and model variance. The latter two are expressed as

$$\text{Bias}[\tilde{y}] = \mathbb{E}[(\mathbf{y} - \mathbb{E}[\tilde{\mathbf{y}}])^2], \quad (16)$$

$$\text{Var}[\tilde{y}] = \mathbb{E}[(\tilde{\mathbf{y}} - \mathbb{E}[\tilde{\mathbf{y}}])^2]. \quad (17)$$

Each of these represents a different aspect of the total error that affects the model's performance.

2. Model Bias

From the expression (16) we see that the model bias measures how far the average model prediction $\mathbb{E}[\tilde{\mathbf{y}}]$ is from the true target value \mathbf{y} . More specifically, it refers to the systematic error introduced by the model's inability to capture the true underlying relationship between the independent and dependent variables. This error arises when the model makes incorrect assumptions about the data or oversimplifies the relationship. For example, a linear model trying to fit highly non-linear data will result in high bias because the model cannot flexibly represent the complexity of the data.

3. Model Variance

We see from the expression (17) for the model variance that it represents the variability in the model's predictions across different training sets. A high model variance indicates that the model is overly sensitive to the specific training data, leading to overfitting, where the model performs well on the training set but poorly on new, unseen data such as the test set. This happens because the model has "memorized" the noise in the training data rather than capturing the true underlying pattern. High variance typically arises in overly complex models that are capable of capturing minute details, which may not generalize well to new data.

4. Bias-Variance Tradeoff

Measures of a predictive model’s total error and how well it is likely to predict future samples are commonly expressed in terms of the mean-squared error (MSE) and the score function, respectively. These are given by

$$\text{MSE}(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2, \quad (18)$$

$$R^2(\mathbf{y}, \tilde{\mathbf{y}}) = 1 - \frac{\sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1} (y_i - \bar{y})^2}, \quad (19)$$

where \bar{y} is the mean value of \mathbf{y} and n is the number of data points. In appendix B 2 [maybe move up here?] we show that the MSE alternatively can be expressed in terms of the model bias, model variance and noise variance as

$$\mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] = \text{Bias}[\tilde{y}] + \text{Var}[\tilde{y}] + \sigma^2. \quad (20)$$

Since the noise variance adds an irreducible contribution to the total model error, an ideal model strikes a balance between bias and variance, where it is flexible enough to capture the underlying patterns in the data but not so flexible that it fits the noise. This balance is referred to as the bias-variance tradeoff. It is a fundamental concept in machine learning, specifically when building predictive models, and will be studied in great detail throughout this work.

C. Resampling Methods

Resampling methods are statistical techniques used to generate additional data samples from the available data. These methods are particularly useful in machine learning and data analysis when the dataset is limited, and we want to better assess the performance of a model. The primary goal of resampling is to estimate the accuracy of a model by splitting the data into different subsets or generating new samples of the data, repeatedly fitting the model, and evaluating its performance on different sets of data. In this work we implement two commonly used resampling techniques: bootstrap and cross-validation.

1. Bootstrap

The bootstrap method is a powerful resampling technique used to estimate the uncertainty and variability of a model by repeatedly drawing random samples, with replacement, from the original dataset. This resampling creates multiple “bootstrapped” datasets, each the same size as the original, but with some samples appearing multiple times and others potentially omitted. By computing the average performance of the multiple models created with the bootstrapped datasets it becomes easier to estimate the model’s bias and variance.

The process of bootstrap resampling approximates the underlying distribution of a statistic, whether it’s model performance, a parameter estimate or prediction error, without requiring strong parametric assumptions about the data. This makes it particularly useful when the theoretical distribution of a statistic is unknown or difficult to calculate. Bootstrap is especially convenient when the dataset is small, or when no explicit train-test split is available, as it makes the most out of the available data. It is also good for understanding the bias-variance tradeoff since it can be used to determine a model’s variability, and it allows for the estimation of confidence intervals for predictions and model parameters. The downside is that it involves training a model multiple times, which can be computationally expensive, especially with large datasets or complex models. [shorten or cite?]

2. Cross-Validation

Cross-validation is another resampling technique that involves partitioning the dataset into several distinct subsets (or “folds”), and then systematically training the model on one subset while testing it on another. A common form is k -fold cross-validation, where the dataset is divided into k equally-sized folds. The model is trained on $k - 1$ folds and tested on the remaining fold. This process is repeated k times, with each fold serving as the test set exactly once.

Since all observations are used for both training and validation, cross-validation maximizes the use of available data, hence it is especially useful in the case of limited datasets. It also helps in managing the bias-variance tradeoff by evaluating the model’s performance on different subsets of data. By averaging the results across multiple folds, cross-validation reduces the variance of the model’s predictions, as it minimizes overfitting to any particular subset of the data. Additionally, by testing the model on unseen data, it ensures that the bias is not too high, making it a good indicator of model generalization. It does require multiple model trainings however (e.g., 10-fold cross-validation requires 10 models to be trained), making it computationally expensive for large datasets or complex models. [shorten or cite?]

D. The Franke Function

The Franke function is a widely used two-dimensional synthetic function in numerical analysis and computational mathematics, particularly in the fields of interpolation, regression analysis, and surface fitting. It is defined

over the unit square $[0, 1] \times [0, 1]$ and expressed as

$$\begin{aligned} f(x, y) = & \frac{3}{4} \exp \left\{ \left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4} \right) \right\} \\ & + \frac{3}{4} \exp \left\{ \left(-\frac{(9x+1)^2}{49} - \frac{9y+1}{10} \right) \right\} \\ & + \frac{1}{2} \exp \left\{ \left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4} \right) \right\} \\ & - \frac{1}{5} \exp \{ -(9x-4)^2 - (9y-7)^2 \}, \quad (21) \end{aligned}$$

We will use the function multiple times throughout this work, as it produces a surface with both smooth and non-smooth regions, making it an excellent benchmark for testing regression and resampling algorithms. It simulates real-world data that may contain both complex variations and noise, mimicking scenarios that we will encounter as we move on to implementing the algorithms on actual simulation data.

III. METHODS & IMPLEMENTATION

write in past tense from here on!

A. Generating the Data

1. Synthetic Data

The Franke function generated data used in this study was created by sampling the function on a grid of $N \times N$ points, where $N = 50$. A visualization of the function as a surface is shown in fig. 1. Furthermore, to emulate inherent variability and uncertainty in real-world data we added a gaussian noise component with a zero mean and standard deviation of $\sigma = 0.1$. The noise was added to simulate the inherent variability and uncertainty in real-world data.

2. Cosmological Simulation Data

After implementing regression analysis and resampling methods using the Franke function we repeated the process using data from an N-body simulation of dark matter structure formation made with the public GASOLINE2 SPH code [1]. The initial conditions for the simulation were generated with MUSIC (MULTI-Scale Initial Conditions) [2], and standard Planck cosmology parameters $\Omega_m = 0.3077$, $\Omega_\Lambda = 0.6923$ and $H_0 = 67.81 \text{ km/s/Mpc}$ were used. The simulation box has length, width and height $L = 20 \text{ Mpc}$, corresponding to a $500 \times 500 \times 500$ grid, and contains 64^3 dark matter particles.

The output data includes 128 “snapshots” of the box through time, where the time parameter in the simula-

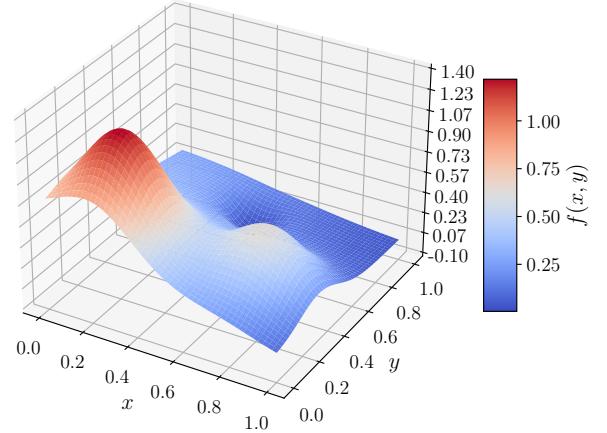


FIG. 1: Visualization of the two-dimensional Franke function expressed in (21).

tion is the scale factor, defined as

$$a = \frac{1}{1+z}.$$

Here z is the cosmological redshift, which is 0 today and thus in the 128th snapshot. We focused on the dark matter density ρ , and studied a snapshot at redshift $z \approx 12.88$ because the distribution of ρ is smoother at larger z . To get a function of two variables we averaged over one of the three axes so that we get a 2D grid, and because the density is so large and vastly different at different points in the box we took $\ln \rho$ to be our dependent variable instead of ρ . The resulting data is visualized as a contour plot and a surface plot in figure 2. We did not add a synthetic noise component to this data, i.e. $\sigma = 0$ in this case, since the simulation output can be interpreted as the true data if we were to perform real observations of dark matter density.

B. Regression Analysis

explain implementation on both Franke and data

1. Creating the Design Matrix

Our first step when performing regression analyses on our data was to extend the design matrix \mathbf{X} to the situation when we have two independent variables \mathbf{x} and \mathbf{y} instead of one \mathbf{x} . We then use \mathbf{z} to denote our dependent variable. In the case of polynomial regression of degree p with one independent variable vector $\mathbf{x} = (x_0 \dots x_n)^\top$

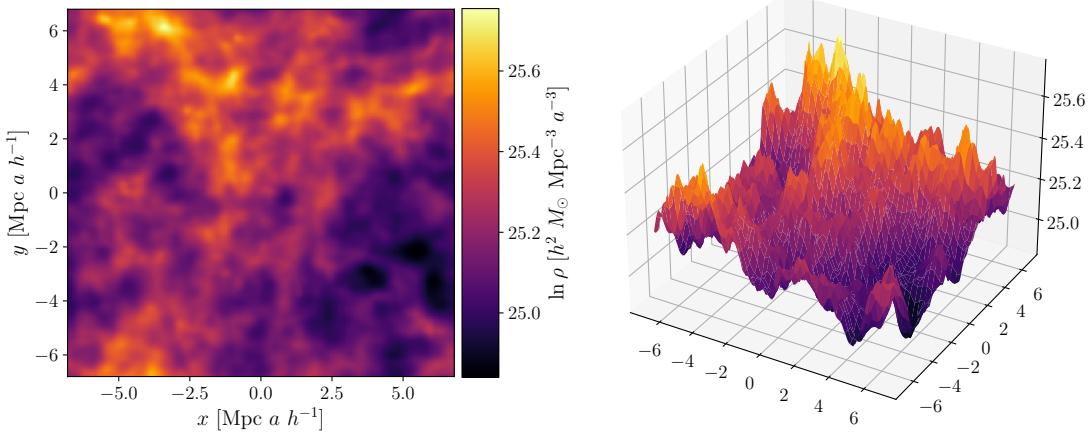


FIG. 2: Logarithm of dark matter density in the simulation box averaged over one of the three axes at redshift $z \approx 12.88$. Plotted as contour plot on the left and as surface plot on the right. Because the Universe expands as time goes on the box does as well, and we therefore scale the coordinates of the box by multiplying with a . Similarly, space between the dark matter particles also increases with time, which is why the density is scaled by multiplying with a^{-3} . GASOLINE2 uses the rest of the scaling factors by default.

the design matrix has the form

$$\mathbf{X} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^p \\ 1 & x_1 & x_1^2 & \dots & x_1^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^p \end{bmatrix}.$$

The feature vector in this case is $\beta = (\beta_0 \dots \beta_p)^\top$, hence it is a column vector of length $p + 1$, while our design matrix has dimensions $(n + 1) \times (p + 1)$

When we have two independent variables representing physical coordinates in a grid each combination of x_i and y_j corresponds to a unique component z_{ij} in \mathbf{z} . If \mathbf{x} and \mathbf{y} contain $n_x + 1$ and $n_y + 1$ components, respectively, the design matrix for polynomial degree p then becomes

$$\mathbf{X} = \begin{bmatrix} 1 & x_0 & y_0 & \dots & x_0^p & x_0^{p-1}y_0 & \dots & x_0y_0^{p-1} & y_0^p \\ 1 & x_1 & y_0 & \dots & x_1^p & x_1^{p-1}y_0 & \dots & x_1y_0^{p-1} & y_0^p \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n_x} & y_0 & \dots & x_{n_x}^p & x_{n_x}^{p-1}y_0 & \dots & x_{n_x}y_0^{p-1} & y_0^p \\ 1 & x_0 & y_1 & \dots & x_0^p & x_0^{p-1}y_1 & \dots & x_0y_1^{p-1} & y_1^p \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n_x} & y_{n_y} & \dots & x_{n_x}^p & x_{n_x}^{p-1}y_{n_y} & \dots & x_{n_x}y_{n_y}^{p-1} & y_{n_y}^p \end{bmatrix}.$$

We thus see that for polynomial degree 0 we have one column, three columns for first degree, six for second, ten for third, etc. This is a well known series of numbers, and we can express the number of features l in terms of the polynomial degree p as

$$l = \frac{1}{2}(p + 1)(p + 2).$$

Our design matrix thus has dimensions $N \times l$ where $N = (n_x + 1) \times (n_y + 1)$. move to theory?

2. Preprocessing

For this study, data scaling was essential to ensure that the features were on a similar scale, which is critical for many machine learning algorithms. By utilizing the `StandardScaler` function from the scikit-learn library [cite](#), we standardized our datasets by subtracting the mean and dividing the standard deviation over the data set, for each feature. Although the data generated by the Franke function and the cosmological simulation data used in subsequent analyses are inherently two-dimensional and may not exhibit significant variation in scale between features, scaling still provides consistent benefits.

The primary motivation for applying this scaling function is to enhance the performance and stability of our algorithms. Scaling prevents features with larger magnitudes from disproportionately influencing the objective function and helps balance minor discrepancies between features that might affect the learning process. The performance of our regression models is significantly improved when the data is scaled due to ensuring faster convergence and reducing the risk of numerical instability.

To illustrate the impact of scaling, we plotted the beta coefficients for both a non-scaled and a scaled design matrix as a function of polynomial degree in our OLS model fits on the Franke function-generated data. This visualization allowed us to observe how scaling influenced the coefficients and affected the model's performance.

Another important preprocessing step was splitting the data into training and test sets. This step enables the evaluation of the model's performance on unseen data and helps in assessing its generalization capabilities. We used the `train_test_split` function from the scikit-

learn library to randomly split the data into training and test sets, with a test size of 0.2. This ensured that 20% of the data was reserved for testing, while the remaining 80% was used to train the model. By splitting the data in this manner, we could evaluate the model's performance on new data and determine how well it generalized to unseen samples.

3. Parameter Dependencies

Investigating the dependency on polynomial degree was a critical aspect of understanding how the regression models performed under varying conditions. By increasing the polynomial order, we enhanced the models' capacity to fit more complex patterns in the data. However, this also raised the risk of overfitting, where the model might capture noise along with the underlying signal. To explore this, we evaluated the performance of the models across different polynomial degrees using metrics such as the MSE and R^2 scores mentioned in the previous section. These evaluations allowed us to quantify how well each model balanced bias and variance.

For the Franke function-generated data, we systematically varied the polynomial degree from 1 to 6 for OLS. We then assessed the model's performance using the MSE and R^2 scores, plotting these metrics as a function of model complexity to determine the optimal polynomial degree for the regression method.

For the cosmological simulation data, we followed a similar approach. We varied the polynomial degree from 1 to 40, using only every other step, for OLS. The model's performance was evaluated using the same metrics as for the Franke function-generated data. We also plotted the predicted surfaces to visually assess the models' fit to the data and identify any potential overfitting or underfitting issues. Specifically, we analyzed the fits for polynomial degrees 5, 10, 20, 30, and 50. Additionally, a 3D surface plot of a 30-degree polynomial fit was created alongside the raw data to visualize the model's performance in capturing the underlying patterns.

For both Ridge and Lasso regression on the Franke function-generated data, we explored polynomial degrees ranging from 1 to 41, using every second polynomial degree for Ridge and every fourth polynomial degree for Lasso. These choices were motivated by the runtime constraints of the algorithms. For the cosmological simulation data, the polynomial degree range was from 1 to 42 for Ridge and from 1 to 21 for Lasso, with a step of every fourth polynomial degree for both methods.

In addition to polynomial degree, we examined the impact of the hyperparameter λ on the performance of our Ridge and Lasso regression models. Visualizing the MSE and R^2 scores for Ridge and Lasso regression involved not only evaluating polynomial complexity but also varying λ values.

To thoroughly evaluate the models, we systematically tested a range of λ values, from 1e-05 to 1. For the Franke

function-generated data, this range included 30 values, evenly spaced on a logarithmic scale, whereas 11 values were tested for the cosmological simulation data. Each λ value was tested for every polynomial degree within the ranges mentioned earlier.

The resulting MSE and R^2 scores were plotted as functions of both λ and polynomial degree using contour plots. This dual-parameter visualization allowed us to identify the optimal combination of polynomial degree and λ for each model, thereby providing a comprehensive view of how model performance changes with varying complexity and regularization strength.

C. Bias-Variance Tradeoff

To showcase the bias-variance trade-off and validate our implementation, we attempted to imitate the analysis illustrated in fig. 3, which is fig. 2.11 in Hastie et al. [cite](#). Specifically, we implemented OLS regression for polynomial degrees $\in [1, 81]$ in steps of 4, and computed the MSE evaluated on both the training and test portions of the Franke data each time. The generation and splitting of the Franke data were conducted as described previously.

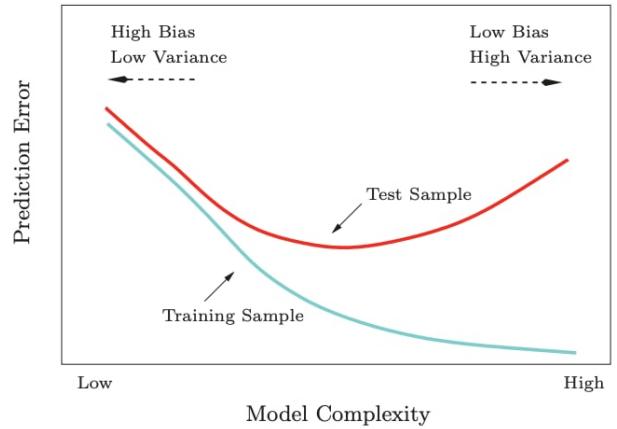


FIG. 3: Illustration of bias-variance trade-off as a function of model complexity, taken from fig. 2.11 in Hastie, Tibshirani, and Friedman [cite](#)

1. Bootstrap With OLS

Our implementation of the bootstrap technique involved resampling the training data multiple times and fitting an OLS model to each resampled dataset. This was accomplished using the resample function from the scikit-learn library. For each resampled dataset, we computed the MSE, bias, and variance to evaluate the model's performance.

For the Franke function-generated data we designed four different cases, each varying a specific parameter to investigate the influence of the number of bootstraps, data points, model complexity, and test size separately. The default values we used were 100 bootstraps, 50 data points, polynomial degree of 5, and a test size of 20%. When varying the parameters in each respective scenario we used the following:

- Polynomial degree: from 1 to 41 with a step of 2
- Number of data points (elements in \mathbf{x} and \mathbf{y}): [10, 20, 30, 40, 50, 100, 150, 200, 300, 400, 500, 1000]
- Number of bootstraps steps: [1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000]
- Test size: ranging from 5% of the entire set to 95% with a step of 5%

For each case we computed the model evaluation metrics as a function of the varying parameter to analyze how the model's performance changed with different settings.

For the cosmological simulation data, we only did a single scenario with 100 bootstraps, 50 data points, test size of 20%, and a varying polynomial degree from 1 to 31 with a step of 3. We only computed the MSE as a function of polynomial degree for this scenario.

Explain why, mostly to save time

2. Cross-Validation

go into detail how we implement

compare with bootstrap include Ridge and Lasso

For our cross-validation analysis, we used the `KFold` function from the scikit-learn library to split the data into k folds, with the integer $k \in [5, 15]$. We iterated over a

D. The Program

1. Code Structure

All the source code that we used to produce our results is available on our GitHub repository, linked in appendix A. The `README.md` file contains the entire project structure. Our code is divided into the following files and notebooks:

`franke.ipynb`: Jupyter notebook containing the analysis of the Franke function data, which we used to validate our implementation.

`data_analysis.ipynb`: Jupyter notebook containing the analysis of the dark matter density data. Here we have applied and further tested our regression models.

`functions.py`: Python script containing all the functions used in the notebooks, all properly documented.

2. Tools

All our code is written in Python [3], and we used scikit-learn [4] to create most of our models. To vectorize our code we used numpy [5], and for visualization we used matplotlib.pyplot [6]. Code completion and debugging was done in Visual Studio Code [7] with additional assistance of GitHub Copilot [8]. We used git [9] for version control, and GitHub [10] for remote storage of our code.

IV. RESULTS & DISCUSSION

A. Effects of Data Scaling

In the upper part of fig. 4 we have plotted the estimated MSE and R^2 scores evaluated on the Franke function test for polynomial degrees $\in [1, 6]$. The left and right columns show the scores computed for the raw and scaled data, respectively. As expected there looks to be a constant factor between the raw and scaled MSE's, while the R^2 scores are equal in both cases. This is because y_i , \tilde{y}_i and \bar{y} (or in our case, z_i , \tilde{z}_i and \bar{z}) are scaled equally, and from eq. (19) we thus see that the scaling cancels out from the numerator and denominator in the second term.

One might expect to see the same effects of scaling for the cosmological simulation data, but in the lower part of fig. 4 we see that this is not the case. Here we have plotted the MSE and R^2 scores for 21 evenly spaced polynomial degrees in the interval $[1, 41]$, and we see that for degrees $\gtrsim 9$ the raw scores diverge from the expected trend. Instead of a decreasing MSE and increasing R^2 as in the scaled case we get MSE's over 400 and very large, negative R^2 scores, a metric that should never be negative in the first place. This clearly shows why it is beneficial to scale the data, since it is safe to assume that the raw data catastrophe occurs due to numerical instability caused by loss of floating point precision. correct? Moreover, we see that we never reach a turning point in the scores where they start to worsen with polynomial degree, which makes sense considering the complexity of our data set.

In fact, we see from the bottom part of fig. 5 that the scaled coefficients for the fits to the cosmological data (plotted in orange) grow in orders of magnitude with complexity (cirka up until degree 21, where stay around 10^4), while the raw coefficients (plotted in purple) fall to zero around polynomial degree 9. The reason for the strong increase in the β_{scaled} 's is that \mathbf{x} and \mathbf{y} (both raw and scaled) are centered at 0, and in the scaled case these are divided by their standard deviations, hence most of the highest order polynomial terms decrease with the degree. We thus need very large coefficients for their contributions to the fit to be noticeable. The β_{raw} 's must be much smaller since the highest order terms grow rapidly with degree at the edges of the box, and eventually we

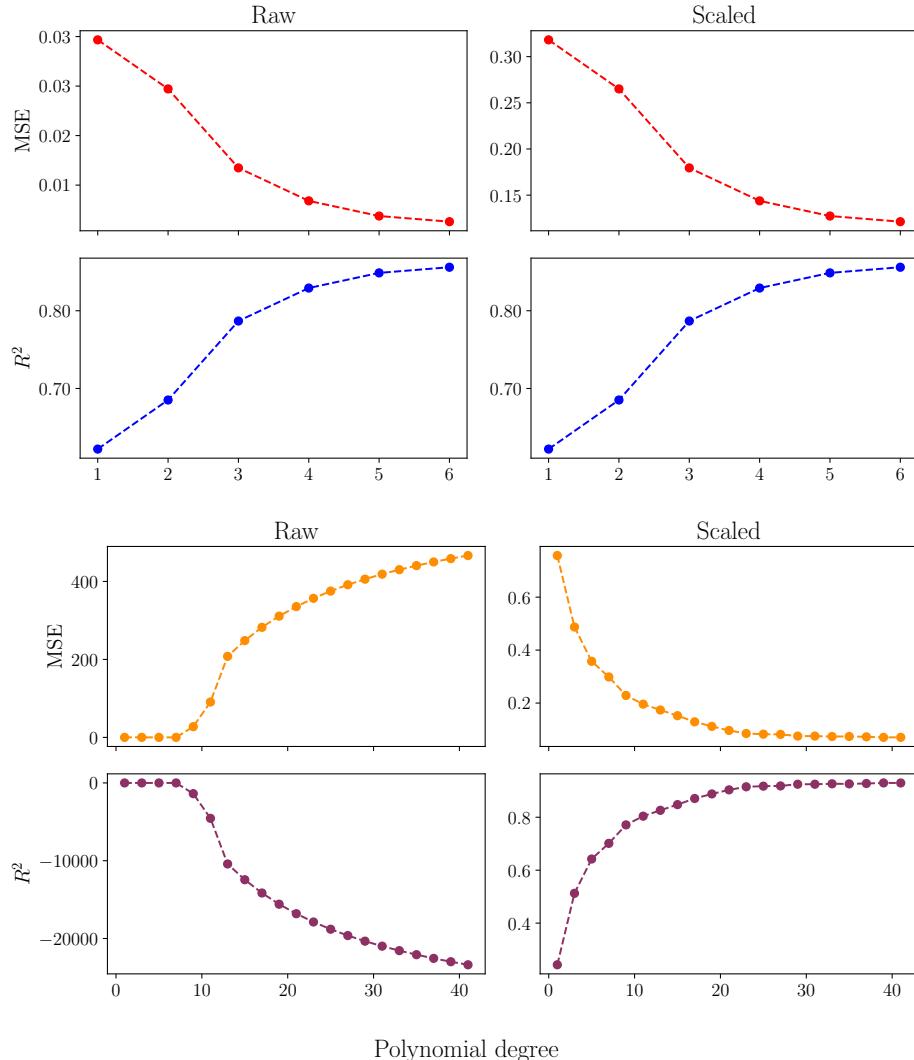


FIG. 4: MSE (upper rows) and R^2 (lower rows) evaluated on the Franke function test set (upper section) and the cosmological simulation test set (lower section) using unscaled (left column) and scaled (right column) values. For the Franke function we have used polynomial degrees in the interval $[1, 6]$, while for the cosmological data we have used every other degree in the interval $[1, 41]$.

encounter floating point errors. This is contrary to the coefficients estimated for the Franke function, where generally $\beta_{\text{raw}} \geq \beta_{\text{scaled}}$, as seen in the top part of the figure. This is because the Franke function is defined over the unit square, hence the raw coefficients grow with polynomial degree as well. rewrite?

B. Hyperparameter Dependency

In fig. 6 we have contour plotted MSE and R^2 scores evaluated on the test portion of the Franke function data for Ridge (top section) and Lasso (bottom section) regression. Here we see that in both cases it is generally favorable to opt for smaller hyperparameters, although Lasso is much more sensitive to changes in λ above a

threshold of around 7 than what Ridge is. This is because Lasso regression employs the L^1 norm, which means that the coefficients can be driven completely to zero if the hyperparameter is large enough, contrary to the L^2 norm used in Ridge regression. Thus, as we continue increasing the value of λ in Lasso above a certain value our fits will practically be zero at all points, and the scores will then stay constant.

For Ridge we see that the increased error associated with a large λ is smaller as we move to the highest complexities, which makes sense considering that we already had a good fit (low MSE and high R^2) for polynomial degrees around 5-6, hence the higher order terms are not as necessary. Since the L^2 norm ensures that the coefficients are shrunk uniformly, a large hyperparameter will thus have more severe consequences for the quality of the lower complexity fits. Moreover, we found that the

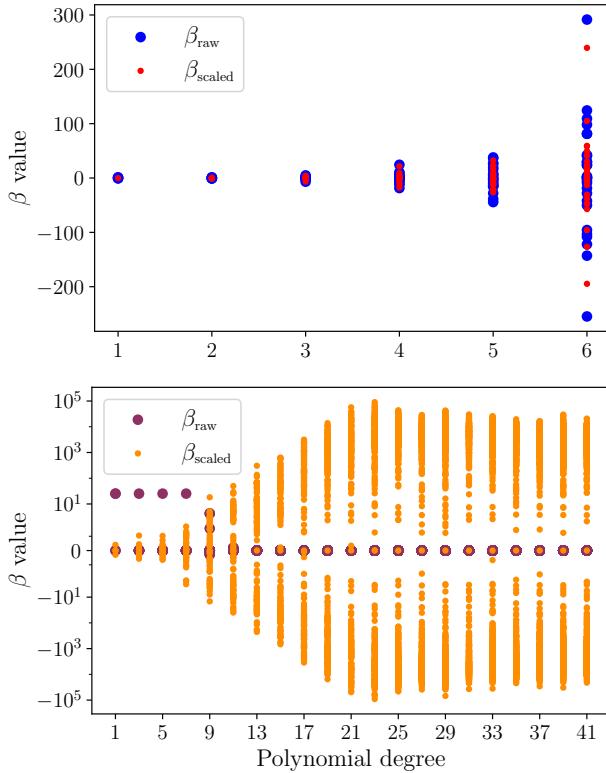


FIG. 5: The coefficients β for the Franke (top) and cosmological (bottom) fits estimated using unscaled and scaled data, plotted for polynomial degrees in the intervals $[1, 6]$ and $[1, 41]$ for Franke and dark matter, respectively. We have used a logarithmic scale for the β 's in the latter case.

scores' overall trend of the dependencies on polynomial degree and hyperparameter were similar for the cosmological data, for both Ridge and Lasso regression, as can be seen in fig. 13 in appendix C. [discuss more?](#)

In the right columns of figs. 6 and 13 we have cropped out the smallest polynomial degrees and largest hyperparameters. Here we can see that the scores solely improve as we move towards larger degree and smaller λ for the dark matter data, while for the Franke data some combinations of degree and λ lead to slightly better scores than others. Although these variations seem to follow some sort of pattern, at least for Ridge, looking at the values on the colorbar axes we see that they are very small, and they might just be random. For future studies this would be worth looking into. [formulate differently?](#)

C. Bias-Variance Tradeoff

1. Training and Test Errors

In fig. 7 we have plotted MSE scores evaluated on the training and test portions of the Franke data as functions

of polynomial degrees up to 81. Although our curves are not as smooth as those in fig. 2.11 from Hastie et al., the overall trend is similar. The errors fall much faster with complexity for low degrees than the test set error increases for high degrees, in contrast to their figure, where the decrease is not as abrupt and the test set error follows more of a “U-shape”. It is worth noting that their figure is an illustration, though, and we can not know how complex the data is assumed to be relative to the Franke function, nor what exactly is meant by a “high” model complexity. The former affects how large the error is for the lowest polynomial degrees, and the magnitude of the test error at high complexity depends on both.

When studying the complexity and hyperparameter dependence of the Franke function MSE and R^2 scores, we saw that they continued to improve with complexity up to degree 41 and that vanishing λ was the best option. We found this rather peculiar, as we would expect overfitting and thus a need for regularization at this level, considering the relatively low complexity of the Franke function and that the fits we obtained with polynomial degrees between 5 and 10 already seemed satisfactory score-wise. However, fig. 7 offers an explanation to this result, since the test set MSE for OLS regression starts increasing after we surpass polynomial degree 41. Also, since we used 50×50 data points for the regression the chance of fitting to the noise is smaller than if we were to use e.g. 10×10 points, since the expectation value of true data plus noise converges towards the expectation value of the true data plus the expectation value of the noise as the number of data points increases, and the expectation value of the noise is zero in our case. [fact check, rewrite](#)

2. Parameter Dependencies of MSE, Bias and Variance

HIGHER ERROR DUE TO BOOTSTRAPPING BECAUSE THE STATISTICAL DISTRIBUTION OF THE NOISE IS IGNORED, I.E. IT IS A HIGHER CHANCE OF FITTING TO THE NOISE?

D. Resampling Methods

1. *OLS: Cross-Validation VS Bootstrap*
2. *Cross-Validation With Ridge and Lasso*

E. Fits to the Cosmological Data

1. Complexity Dependency in OLS

In fig. 11 we have plotted OLS fits to the cosmological data using polynomial degrees 5, 10, 20, 30 and 50 to further examine the effects of increasing model complexity. From degree 30 to 50 the fit actually seems to worsen, looking more smudged in the center, while

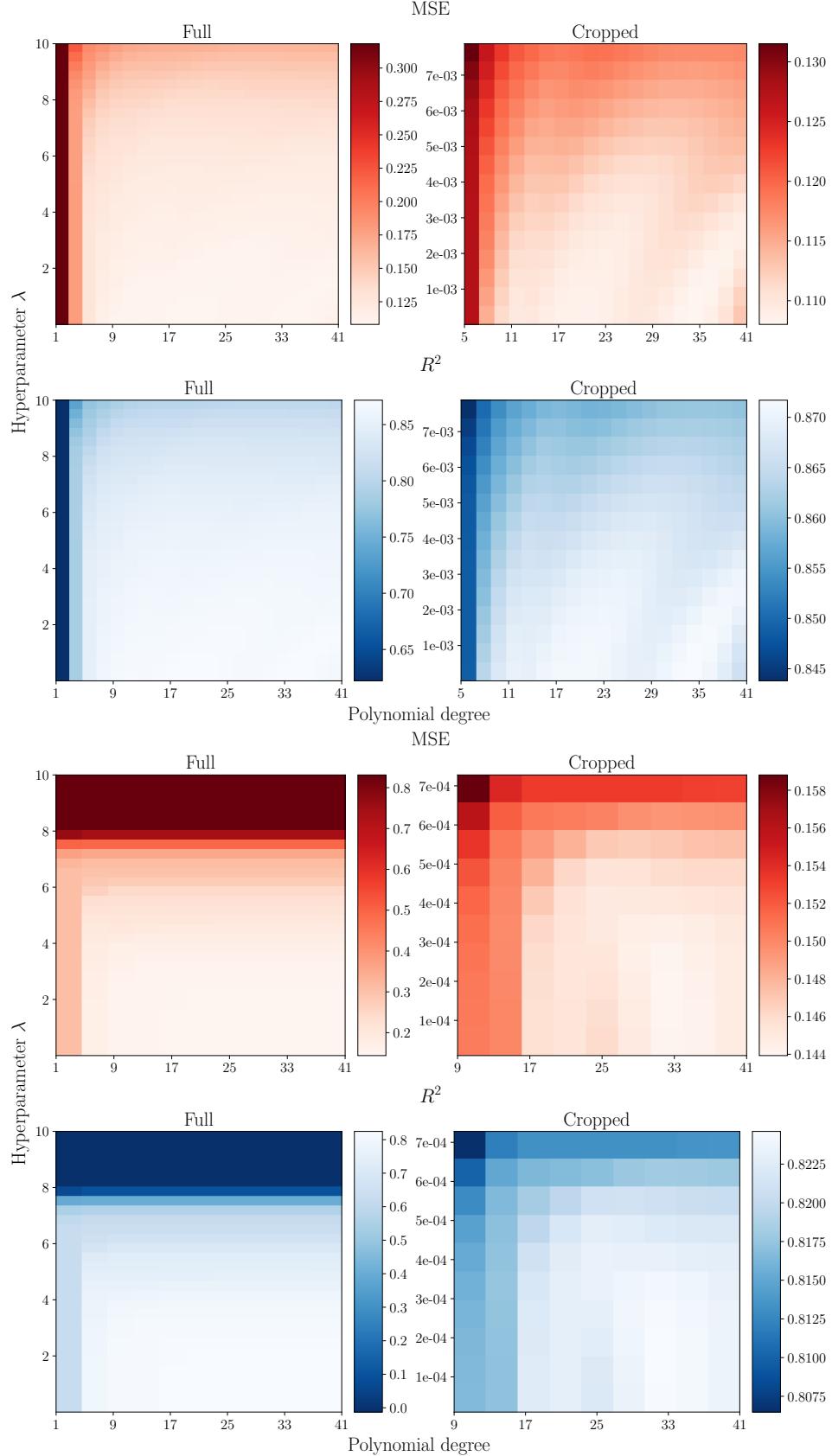


FIG. 6: MSE (upper rows) and R^2 (lower rows) for Ridge (top section) and Lasso (bottom section) regression evaluated on the Franke test data. The scores vary with polynomial degree horizontally and hyperparameter λ vertically. In the right columns we have cropped away degrees 1 and 3 (1 and 5) and the 15 (20) largest λ values for Ridge (Lasso), purely for visualization purposes.

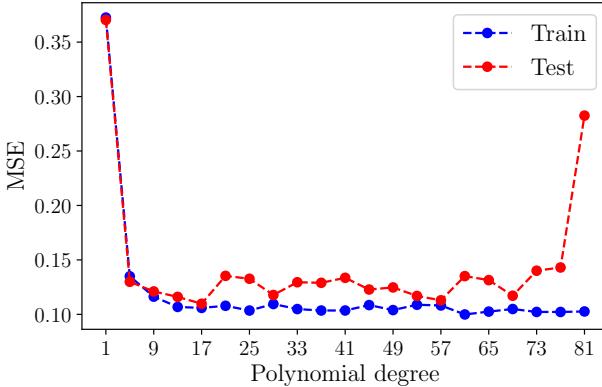


FIG. 7: MSE evaluated on the Franke function training set (blue) and test set (red) for 21 evenly spaced polynomial degrees in the interval [1, 81].

ripple-like structures near the edges become more prominent. These are already visible for degree 30, especially in fig. 14 in appendix C, where we have surface plotted the fit together with the actual data. However, they arguably don't make the fit look less accurate than that obtained with polynomial degree 20.

To explain the reason for the ripple effect, and why the smaller scale structures in the data don't show up more clearly as we continue increasing the model complexity we need to consider the columns of the scaled design matrix. As mentioned in section IV A most of the scaled box coordinates are less than 1. They are in fact close to zero in the center region and slightly larger than 1 in magnitude near the edges. Thus, as we move to higher order polynomials the elements of the rightmost columns in $\mathbf{X}_{\text{scaled}}$ (corresponding to the highest order terms) diverge, some tend to zero while others grow in magnitude. To minimize the model error the coefficients cannot continue increasing because then the fit worsens in the edges, but they cannot decrease too much either as this leads to a less accurate fit in the center. As a result we see the fits become increasingly smeared in the center and more complex at the edges as we increase the complexity, hence the ripple effect. explain better?

In an attempt to confirm this interpretation we decided to split our simulation data into equally sized pieces, scale each piece and perform fits on them separately, then stitch them back together. That way, we hypothesized that the smearing and ripple effects would be minimized and more evenly distributed through the grid since the

complex structure of the data is simpler on smaller, local scales. We did this with 4 pieces and a 100 pieces, using OLS and polynomial degree 30 in both cases, and the result is shown in fig. 15 in appendix C. Indeed, we see that the resulting fit with 100 pieces looks nearly identical to the actual data. For the fit with 4 pieces it is very obvious that we have stitched separate fits back together, since we clearly see the ripple effects from each fit meeting at the boundaries.

2. Comparing Regression Variants

In fig. 12 we have plotted the OLS, Ridge and Lasso fits using polynomial degree 30, hyperparameter $\lambda = 0.001$ and $k = 5$ folds together with the actual data to get a feel for the implications of the estimated MSE scores. Here we see that the Lasso fit is significantly worse than both OLS and Ridge, although Ridge is lacking in comparison to OLS as well. This agrees with figs. fig. 9 and fig. 10, since the MSE's for the largest polynomial degrees lie somewhere below 0.1 for OLS, in the interval [0.1, 0.2] for Ridge and [0.2, 0.3] for Lasso. It also makes sense considering the scores plotted in fig. 13 in appendix C, where we saw that Lasso generally performed poorly compared to Ridge.

It is not surprising that OLS worked best with our chosen polynomial degree, since we are far from reaching the complexity of the actual data. Thus, introducing a penalizing regularization term when computing the estimators $\hat{\beta}$ to account for overfitting is of little use, and the result is instead that the fits become too simplistic. In order to understand why the Lasso fit is so poor compared to the Ridge fit we need to review their cost functions. The L^1 norm used in Lasso regression creates a sparser model by shrinking some coefficients entirely to zero, which tends to be more pronounced when dealing with higher polynomial degrees where many coefficients could have minimal or redundant contributions to the model. The L^2 norm in Ridge regression penalizes large coefficients as well, but does not drive them to zero. While it still reduces the impact of large polynomial coefficients, it tends to be less severe in its penalization than Lasso, particularly for coefficients that contribute significantly to the model.

V. CONCLUSION

open questions

reflections?

REFERENCES

- [1] T. R. Q. James W. Wadsley, Benjamin W. Keller, "Gasoline2: A Modern SPH Code," <https://arxiv.org/abs/1707.03824> (Accessed: September 2024).
- [2] O. Hahn, "MUSIC," <https://www-n.oca.eu/ohahn/MUSIC/> (Accessed: September 2024).
- [3] Python Software Foundation, "Python 3.11.4 documentation," <https://docs.python.org/3.11/> (Accessed: October 2023).

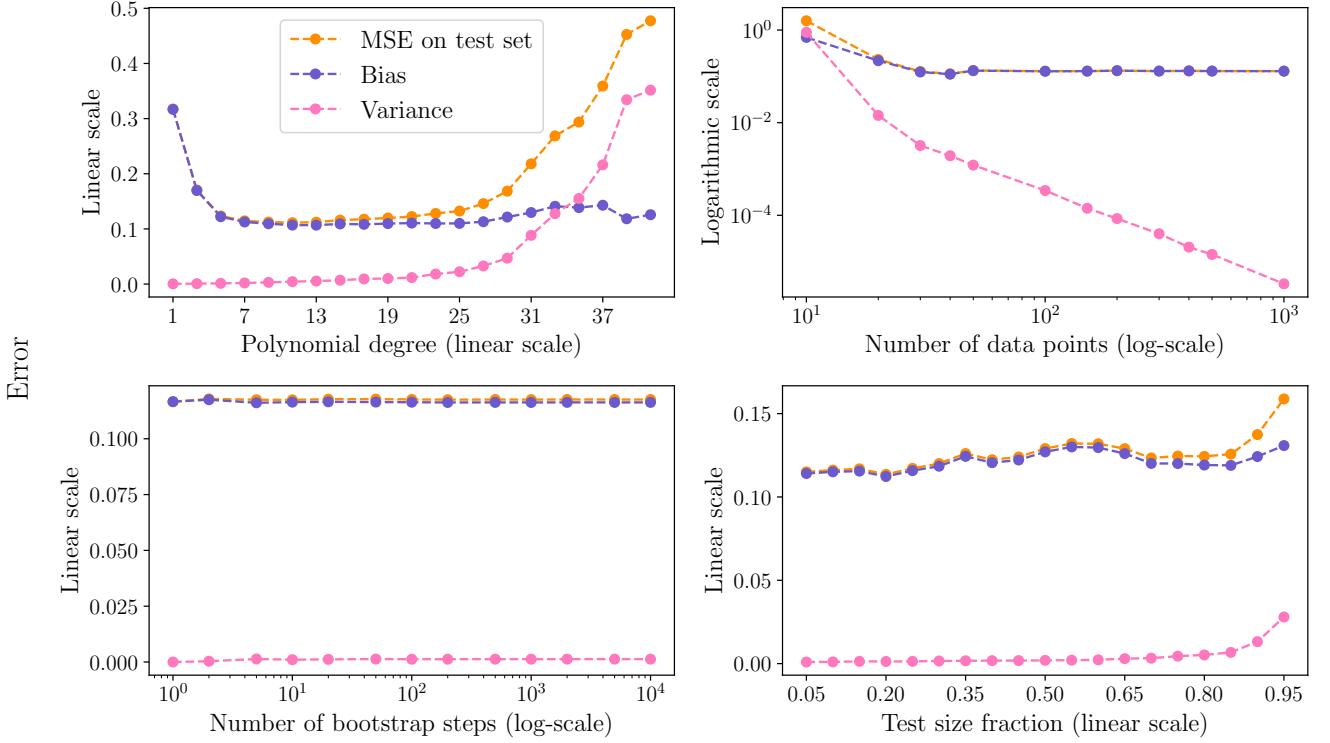


FIG. 8: MSE (orange), bias (blue) and variance (pink) for OLS regression evaluated on Franke test data and plotted as functions of polynomial degree (upper left), number of data points (upper right), number of bootstrap steps (lower left) and test size fraction (lower right). When these are not varied we have used degree 5, 50 data points in one direction (i.e. 2 500 in the grid), 100 bootstrap steps and a test size fraction of 0.2. Note that some axes are scaled logarithmically and some have a linear scale.

- [4] 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, Journal of Machine Learning Research **12**, 2825 (2011).
- [5] NumPy Developers, “NumPy v1.22 Manual,” <https://numpy.org/doc/stable/> (Accessed: October 2023).
- [6] Matplotlib Development Team, “Matplotlib: Visualization with Python,” <https://matplotlib.org/stable/index.html> (Accessed: October 2023).
- [7] Microsoft, “Visual Studio Code,” <https://code.visualstudio.com/docs> (Accessed: October 2023).
- [8] Microsoft, “GitHub Copilot,” <https://copilot.github.com/> (Accessed: October 2023).
- [9] J. Hamano, “Git,” <https://git-scm.com/doc> (Accessed: October 2023).
- [10] Microsoft, “Github,” <https://docs.github.com/en> (Accessed: October 2023).

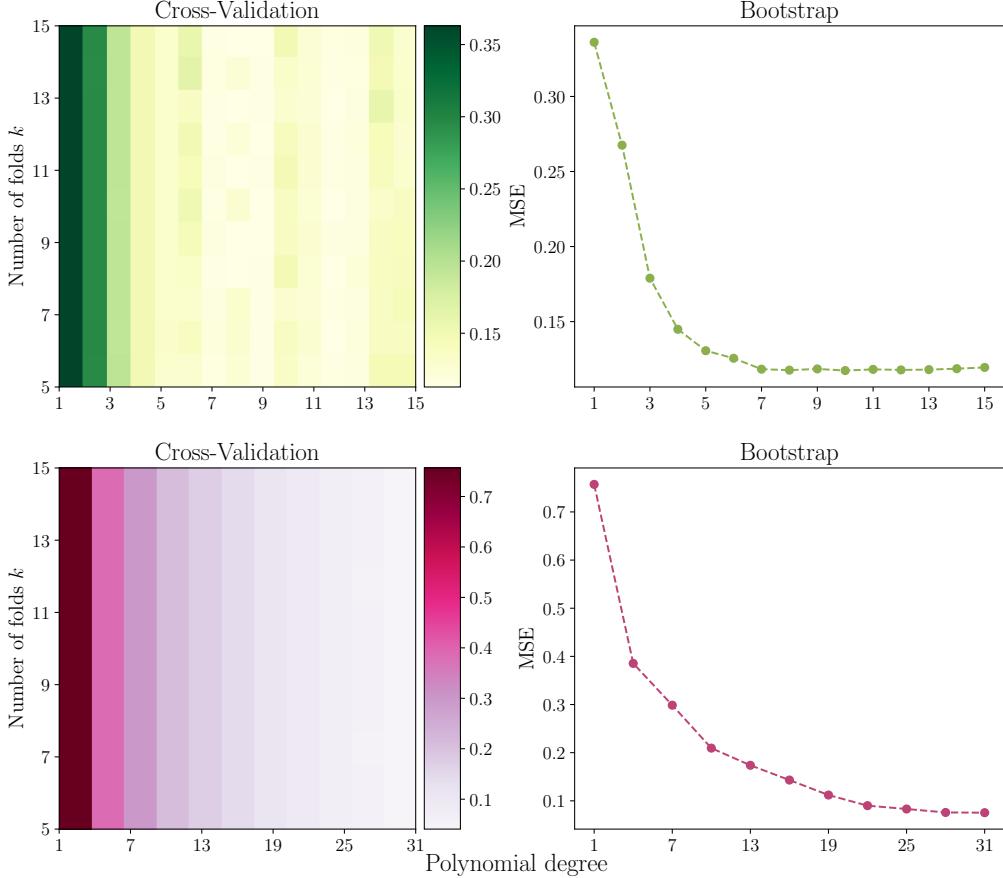


FIG. 9: MSE evaluated on Franke (upper row) and cosmological (lower row) test data as function of polynomial degrees in the intervals [1, 15] (Franke) and [1, 31] (cosmological) and number of folds $k \in [5, 15]$ used in cross-validation (contour plots on the left). On the right we have plotted the MSE's purely as functions of complexity when using the bootstrap resampling method with 100 steps.

Appendix A: Code

<https://github.com/Oskar-Idland/FYS-STK4155-Projects>

Appendix B: Derivations

1. Expectation value and variance of the OLS estimator $\hat{\beta}$

rewrite to fit report?

Since the error in \mathbf{y} is normal distributed as $\boldsymbol{\varepsilon} \sim N(0, \sigma^2)$ we know that the expectation value and variance of the i 'th element of $\boldsymbol{\varepsilon}$ is $\mathbb{E}(\varepsilon_i) = 0$ and $\text{Var}(\varepsilon_i) = \sigma^2$. Thus, since we approximate $f(\mathbf{x})$ with $\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}$ the expectation value of \mathbf{y} becomes

$$\mathbb{E}(\mathbf{y}) = \mathbb{E}(\mathbf{X}\boldsymbol{\beta}) + \mathbb{E}(\boldsymbol{\varepsilon}) = \mathbb{E}(\mathbf{X}\boldsymbol{\beta}), \quad (\text{B1})$$

which gives us the expectation value of \mathbf{y} for a given element i :

$$\mathbb{E}(y_i) = \mathbb{E} \left(\sum_j X_{ij} \beta_j \right) = \sum_j X_{ij} \beta_j = \mathbf{X}_{i,*} \boldsymbol{\beta}. \quad (\text{B2})$$

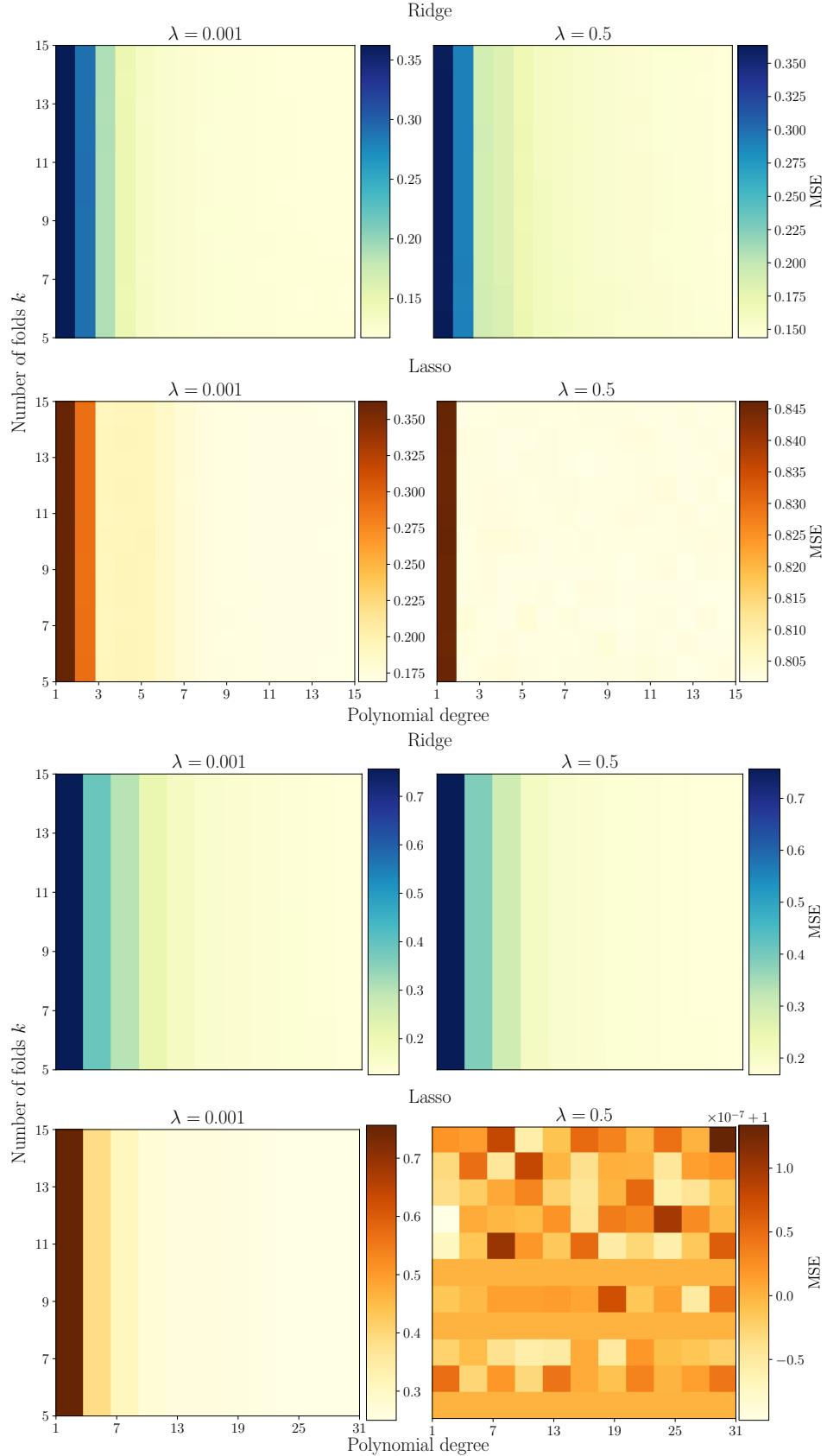


FIG. 10: MSE evaluated on the Franke (top section) and cosmological (bottom section) test sets as function of polynomial degrees and number of k -folds used in cross-validation, for Ridge (upper rows) and Lasso (lower rows) regression. We have used $\lambda = 0.001$ and $\lambda = 0.5$ in the left and right columns, respectively.

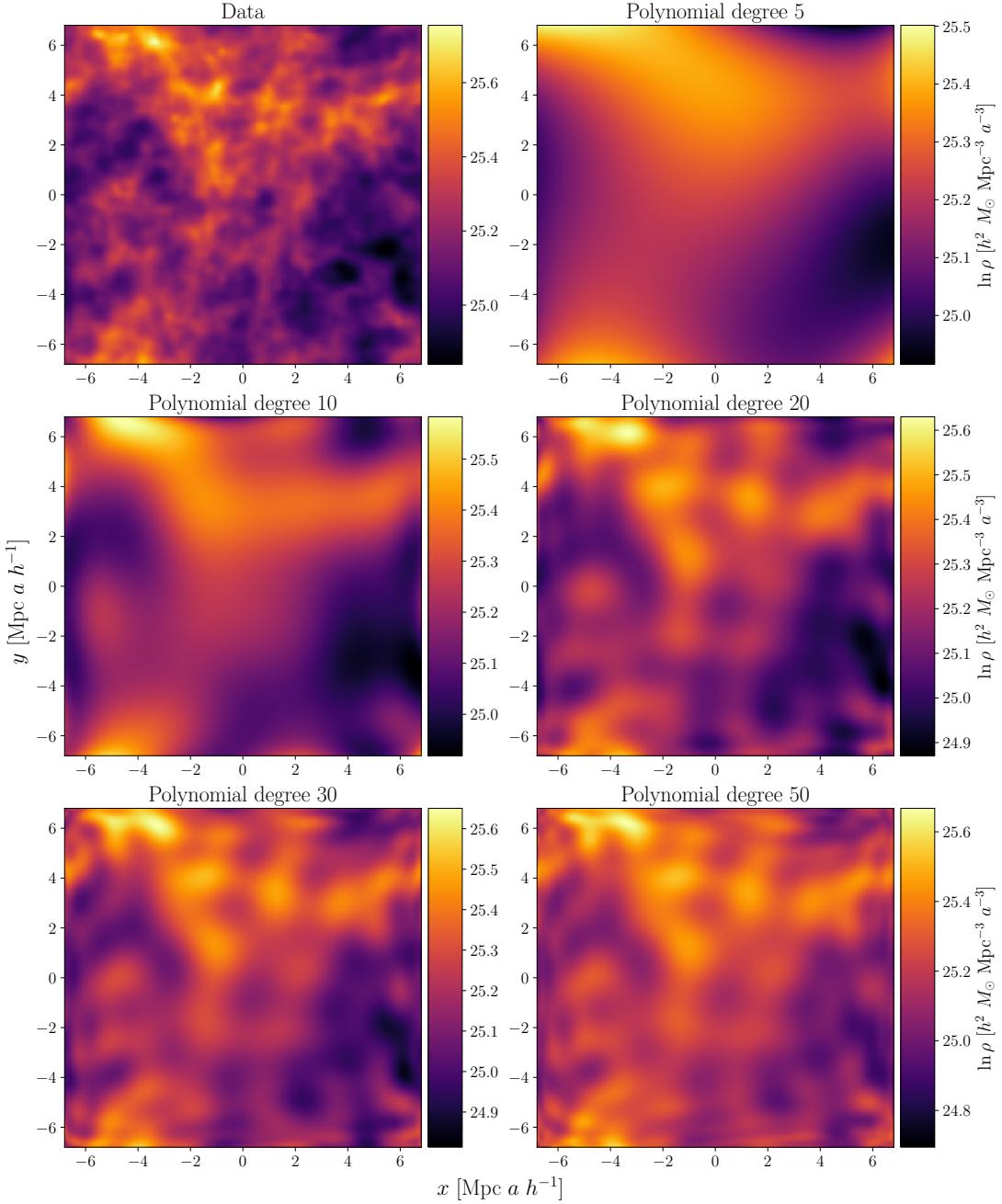


FIG. 11: Contour plots of the cosmological simulation data (upper left) and the OLS fits for polynomial degrees 5 (upper right), 10 (middle left), 20 (middle right), 30 (lower left) and 50 (lower right).

Here we have used that the sum $\sum_j X_{ij}\beta_j$ is known to be the value of \tilde{y}_i , hence its expectation value is itself. Moreover, we can similarly find the variance of \mathbf{y} for a given element i by using that the terms in the aforementioned sum are known to be \tilde{y}_i for all i , i.e. $\text{Var}(\mathbf{X}_{i,*}\boldsymbol{\beta}) = 0$. Thus, we have

$$\text{Var}(y_i) = \text{Var}(\mathbf{X}_{i,*}\boldsymbol{\beta}) + \text{Var}(\varepsilon_i) = \sigma^2, \quad (\text{B3})$$

and consequently

$$y_i \sim N(\mathbf{X}_{i,*}\boldsymbol{\beta}, \sigma^2). \quad (\text{B4})$$

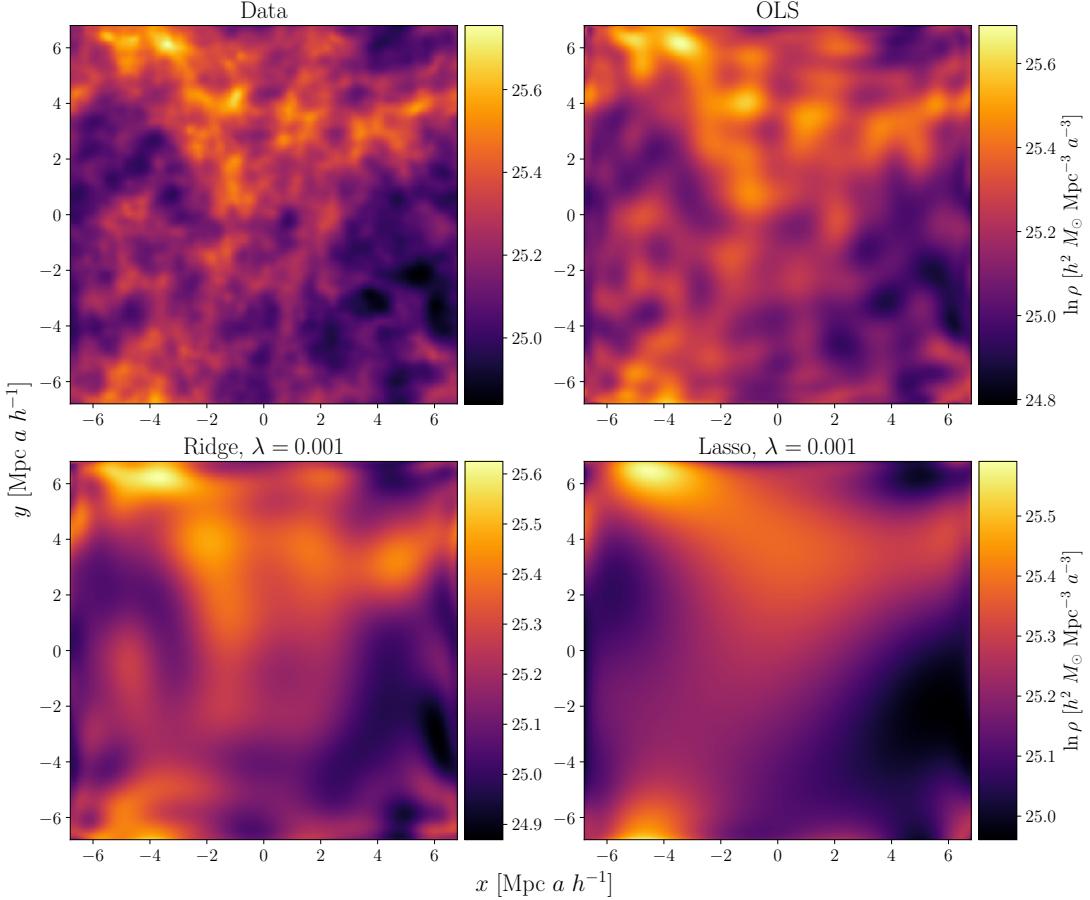


FIG. 12: Contour plots of the cosmological simulation data (upper left) and OLS (upper right), Ridge (lower left) and Lasso (lower right) regression fits using polynomial degree 30, $\lambda = 0.001$ for both Ridge and Lasso, and $k = 5$ folds in cross-validation.

Now, using that the OLS estimator is given by $\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$, its expectation values become

$$\begin{aligned}
\mathbb{E}(\hat{\beta}) &= \mathbb{E}\left\{(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}\right\} \\
&= \mathbb{E}\left\{(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top\right\} \mathbb{E}(\mathbf{y}) \\
&= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{X} \beta \\
&= \beta.
\end{aligned} \tag{B5}$$

Furthermore, if x and y are two independent variables, the variance of their product is given by

$$\begin{aligned}
\text{Var}(xy) &= \mathbb{E}(x^2 y^2) - (\mathbb{E}(xy))^2, \\
&= \mathbb{E}(x^2) \mathbb{E}(y^2) - (\mathbb{E}(x))^2 (\mathbb{E}(y))^2, \\
&= [\mathbb{E}(x^2) - (\mathbb{E}(x))^2 + (\mathbb{E}(x))^2] [\mathbb{E}(y^2) - (\mathbb{E}(y))^2 + (\mathbb{E}(y))^2] - \mathbb{E}(x^2) \mathbb{E}(y^2), \\
&= [\text{Var}(x) + (\mathbb{E}(x))^2] [\text{Var}(y) + (\mathbb{E}(y))^2] - \mathbb{E}(x^2) \mathbb{E}(y^2), \\
&= \text{Var}(x) \text{Var}(y) + \text{Var}(x)(\mathbb{E}(y))^2 + \text{Var}(y)(\mathbb{E}(x))^2 + \mathbb{E}(x^2) \mathbb{E}(y^2) - \mathbb{E}(x^2) \mathbb{E}(y^2), \\
&= \text{Var}(x) \text{Var}(y) + \text{Var}(x)(\mathbb{E}(y))^2 + \text{Var}(y)(\mathbb{E}(x))^2,
\end{aligned}$$

so if we now set $x = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ and $y = \mathbf{y}$ we find that the variance of $\hat{\beta}$ is

$$\begin{aligned}
\text{Var}(\hat{\beta}) &= \underbrace{\text{Var}\left[(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top\right] \text{Var}(\mathbf{y})}_{0} + \underbrace{\text{Var}\left[(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top\right] (\mathbb{E}(\mathbf{y}))^2}_{0} + \text{Var}(\mathbf{y}) \left(\mathbb{E}\left[(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top\right]\right)^2, \\
&= \sigma^2 \left[(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top\right]^2, \\
&= \sigma^2 \left[(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top\right] \left[(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top\right]^\top, \\
&= \sigma^2 \left[(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top\right] \left[\mathbf{X} (\mathbf{X}^\top \mathbf{X})^{-1}\right], \\
&= \sigma^2 (\mathbf{X}^\top \mathbf{X})^{-1}.
\end{aligned} \tag{B6}$$

Here we have used that the transpose of $(\mathbf{X}^\top \mathbf{X})^{-1}$ is itself since it is square and symmetric. mention that sigma is zero in cosmo case, so variance is zero

2. MSE expressed in terms of model bias, model variance and noise variance

rewrite to fit report?

Substituting \mathbf{y} with $f(\mathbf{x}) + \epsilon$, and adding and subtracting $\mathbb{E}[\tilde{\mathbf{y}}]$, we find that

$$\begin{aligned}
\mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] &= \mathbb{E}\left[\underbrace{(f(\mathbf{x}) + \epsilon - \tilde{\mathbf{y}})^2}_\mathbf{f}\right], \\
&= \mathbb{E}\left[(\mathbf{f} + \epsilon - \tilde{\mathbf{y}} + \mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}[\tilde{\mathbf{y}}])^2\right], \\
&= \mathbb{E}\left[\mathbf{f}^2 + \mathbf{f}\epsilon - \mathbf{f}\tilde{\mathbf{y}} + \mathbf{f}\mathbb{E}[\tilde{\mathbf{y}}] - \mathbf{f}\mathbb{E}[\tilde{\mathbf{y}}] + \epsilon\mathbf{f} + \epsilon^2 - \epsilon\tilde{\mathbf{y}} + \epsilon\mathbb{E}[\tilde{\mathbf{y}}] - \epsilon\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}}\mathbf{f} - \tilde{\mathbf{y}}\epsilon + \tilde{\mathbf{y}}^2 - \tilde{\mathbf{y}}\mathbb{E}[\tilde{\mathbf{y}}] + \tilde{\mathbf{y}}\mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}]\mathbf{f} + \mathbb{E}[\tilde{\mathbf{y}}]\epsilon - \mathbb{E}[\tilde{\mathbf{y}}]\tilde{\mathbf{y}} + (\mathbb{E}[\tilde{\mathbf{y}}])^2 - (\mathbb{E}[\tilde{\mathbf{y}}])^2 - \mathbb{E}[\tilde{\mathbf{y}}]\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}]\epsilon + \mathbb{E}[\tilde{\mathbf{y}}]\tilde{\mathbf{y}} - (\mathbb{E}[\tilde{\mathbf{y}}])^2 + (\mathbb{E}[\tilde{\mathbf{y}}])^2\right], \\
&= \mathbb{E}\left[\mathbf{f}^2 + \mathbf{f}\epsilon + \epsilon\mathbf{f} + \epsilon^2 - \mathbf{f}\mathbb{E}[\tilde{\mathbf{y}}] - \epsilon\mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}[\tilde{\mathbf{y}}]\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}]\epsilon + (\mathbb{E}[\tilde{\mathbf{y}}])^2\right] \\
&\quad + \mathbb{E}\left[\tilde{\mathbf{y}}^2 - \tilde{\mathbf{y}}\mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}[\tilde{\mathbf{y}}]\tilde{\mathbf{y}} + (\mathbb{E}[\tilde{\mathbf{y}}])^2\right] \\
&\quad + \mathbb{E}\left[-\mathbf{f}\tilde{\mathbf{y}} - \epsilon\tilde{\mathbf{y}} + \mathbf{f}\mathbb{E}[\tilde{\mathbf{y}}] + \epsilon\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}}\mathbf{f} - \tilde{\mathbf{y}}\epsilon + \tilde{\mathbf{y}}\mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}]\mathbf{f} + \mathbb{E}[\tilde{\mathbf{y}}]\epsilon + \mathbb{E}[\tilde{\mathbf{y}}]\tilde{\mathbf{y}} - 2(\mathbb{E}[\tilde{\mathbf{y}}])^2\right].
\end{aligned}$$

Before we move further we may note that the exact function $f(\mathbf{x})$ generally is not known, and we may therefore assume that our data is a good representation and replace \mathbf{f} with \mathbf{y} in the expression above. In practise this \mathbf{y} is then the part of the data set that we have chosen as test set, while the model is trained with the remaining data set (the training set). Thus, using that $\mathbb{E}[\mathbb{E}[\mathbf{x}]] = \mathbb{E}[\mathbf{x}]$, $\mathbb{E}[(\mathbb{E}[\mathbf{x}])^2] = (\mathbb{E}[\mathbf{x}])^2$ and $\mathbb{E}[\mathbf{x}, \mathbf{y}] = \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{y}]$ for any statistically independent \mathbf{x} and \mathbf{y} , and that $\mathbb{E}[\epsilon] = 0$ so that we can remove all first order terms in ϵ , we get

$$\begin{aligned}
\mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] &= \mathbb{E}\left[\mathbf{y}^2 + \epsilon^2 - \mathbf{y}\mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}[\tilde{\mathbf{y}}]\mathbf{y} + (\mathbb{E}[\tilde{\mathbf{y}}])^2\right] \\
&\quad + \mathbb{E}\left[\tilde{\mathbf{y}}^2 - \tilde{\mathbf{y}}\mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}[\tilde{\mathbf{y}}]\tilde{\mathbf{y}} + (\mathbb{E}[\tilde{\mathbf{y}}])^2\right] \\
&\quad + \mathbb{E}\left[-\mathbf{y}\tilde{\mathbf{y}} + \mathbf{y}\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}}\mathbf{y} + \tilde{\mathbf{y}}\mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}]\mathbf{y} + \mathbb{E}[\tilde{\mathbf{y}}]\tilde{\mathbf{y}} - 2(\mathbb{E}[\tilde{\mathbf{y}}])^2\right], \\
&= \mathbb{E}\left[(\mathbf{y} - \mathbb{E}[\tilde{\mathbf{y}}])^2\right] + \mathbb{E}[\epsilon^2] + \mathbb{E}\left[(\tilde{\mathbf{y}} - \mathbb{E}[\tilde{\mathbf{y}}])^2\right] \\
&\quad - \mathbb{E}[\mathbf{y}]\mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\mathbf{y}]\mathbb{E}[\tilde{\mathbf{y}}] - \mathbb{E}[\tilde{\mathbf{y}}]\mathbb{E}[\mathbf{y}] + \mathbb{E}[\tilde{\mathbf{y}}]\mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}]\mathbb{E}[\mathbf{y}] + \mathbb{E}[\mathbf{y}]\mathbb{E}[\tilde{\mathbf{y}}] - 2(\mathbb{E}[\tilde{\mathbf{y}}])^2, \\
&= \text{Bias}[\tilde{\mathbf{y}}] + \text{Var}[\tilde{\mathbf{y}}] + \sigma^2.
\end{aligned} \tag{B7}$$

Appendix C: Additional Figures

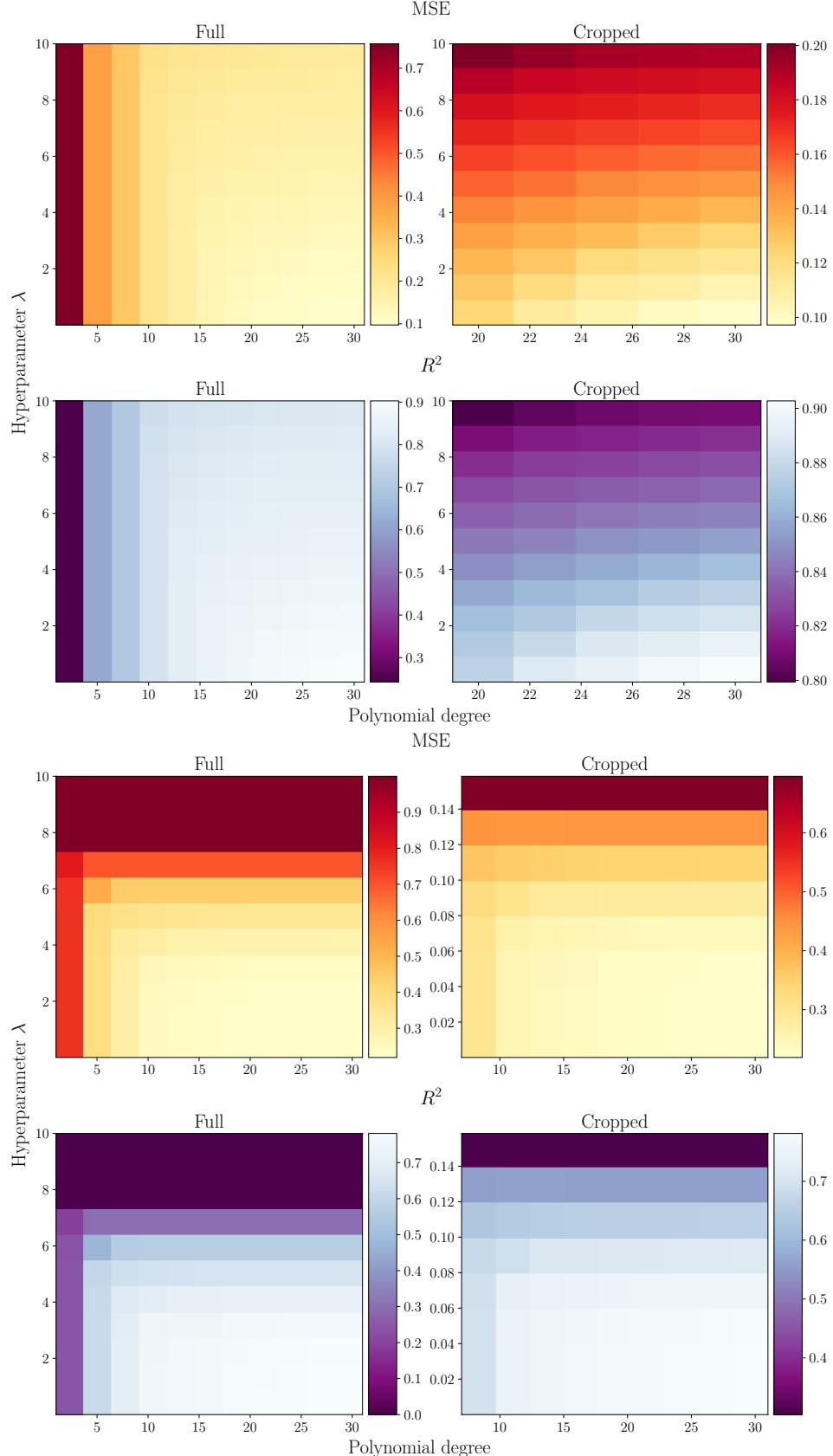


FIG. 13: MSE (upper rows) and R^2 (lower rows) for Ridge (top section) and Lasso (bottom section) regression evaluated on the cosmological test set. The scores vary with polynomial degree horizontally and hyperparameter λ vertically. In the right columns we have cropped away degrees smaller than 19, as well as the 3 largest λ 's only for Lasso, purely for visualization purposes.

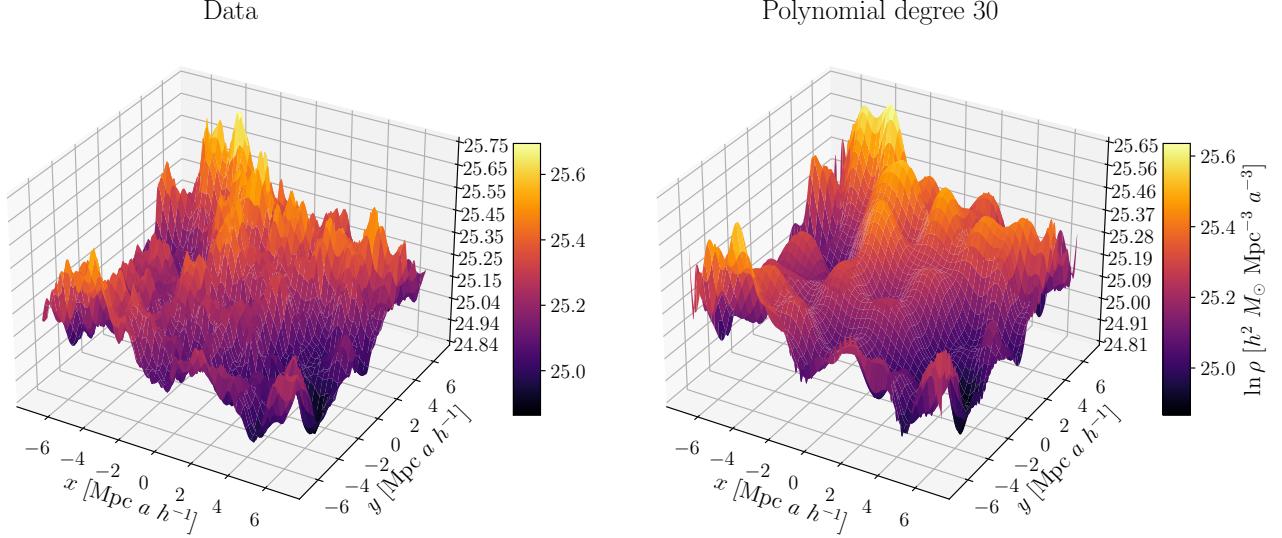


FIG. 14: Surface plots of the cosmological simulation data (left) and the OLS fit for polynomial degree 30 (right).

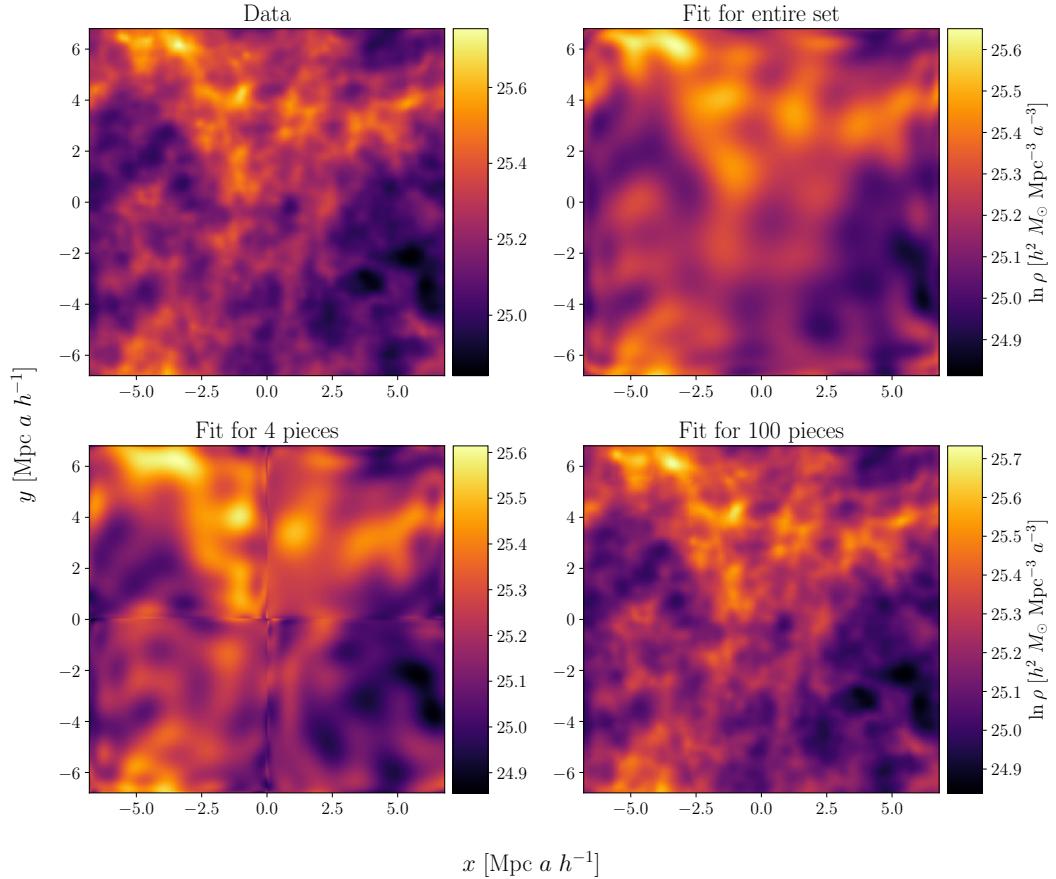


FIG. 15: Contour plots of the cosmological simulation data (upper left) and the OLS fits for the entire data (upper right), 4 separate pieces stitched back together (lower left) and 100 separate pieces stitched back together (lower right). We have used a polynomial degree of 30 for all fits.