Project 1

FYS-STK4155 — Applied data analysis and machine learning

Anders Johansson

8th October 2018

Abstract

This project uses fitting of Franke's function and geographical data to compare ordinary least squares, Ridge and LASSO regression. Resampling is done in order to get accurate estimates for variances and means. A bias-variance decomposition is derived analytically and confirmed numerically. The regression methods are implemented from scratch in Fortran and show good performance, although Newton's method proves to be suboptimal as a minimisation method for the LASSO cost function, due to its non-differentiability in its minimum for large values of the regularisation parameter. Ridge and LASSO regression are found to outperform ordinary least squares on test data.

All files for this project are available at `https://github.com/anjohan/ml1`.

Contents

1 Introduction

Fitting of data is an important tool in most sciences. The goal can be to interpolate between already measured data or predict values outside the measured range when extra experiments are infeasible, or to discover relations between quantities. By assuming that the data points can be modelled by a linear combination of some set of functions and minimising the error, one ends up with linear regression. This report explores three different linear regression methods — ordinary least squares, which simply minimises the error, and Ridge and LASSO, which make up for their worse performance on training data by returning better predictors.

The main goal of this project is to compare these three regression methods empirically by applying it to two sets of data. First, the regression methods are applied to a data set generated from a known function, specifically the Franke's function, which is a sum of bivariate Gaussian functions. Having verified the implementation of the methods, they are then applied to geographic data, namely the altitude as a function of geographic coordinates.

The first part of this report goes through the basic theory of these regression methods and their implementation, as well as resampling techniques and the bias-variance decomposition. Then, the regression methods are applied to the Franke's function and the results are analysed with resampling. Finally, geographical data is fitted with all three methods.

2 Theory and methods

2.1 Test case

The function to be fitted as verification of the regression methods is the famous Franke's function $f : [0,1]^2 \to \mathbb{R}$ given by

$$\begin{aligned} f(x_1, x_2) = \frac{3}{4}\exp\left(-\frac{(9x_1-2)^2}{4} - \frac{(9x_2-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x_1+1)^2}{49} - \frac{(9x_2+1)}{10}\right) \\ + \frac{1}{2}\exp\left(-\frac{(9x_1-7)^2}{4} - \frac{(9x_2-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x_1-4)^2 - (9x_2-7)^2\right) \end{aligned} \tag{2.1}$$

and shown in figure 1 on the following page.

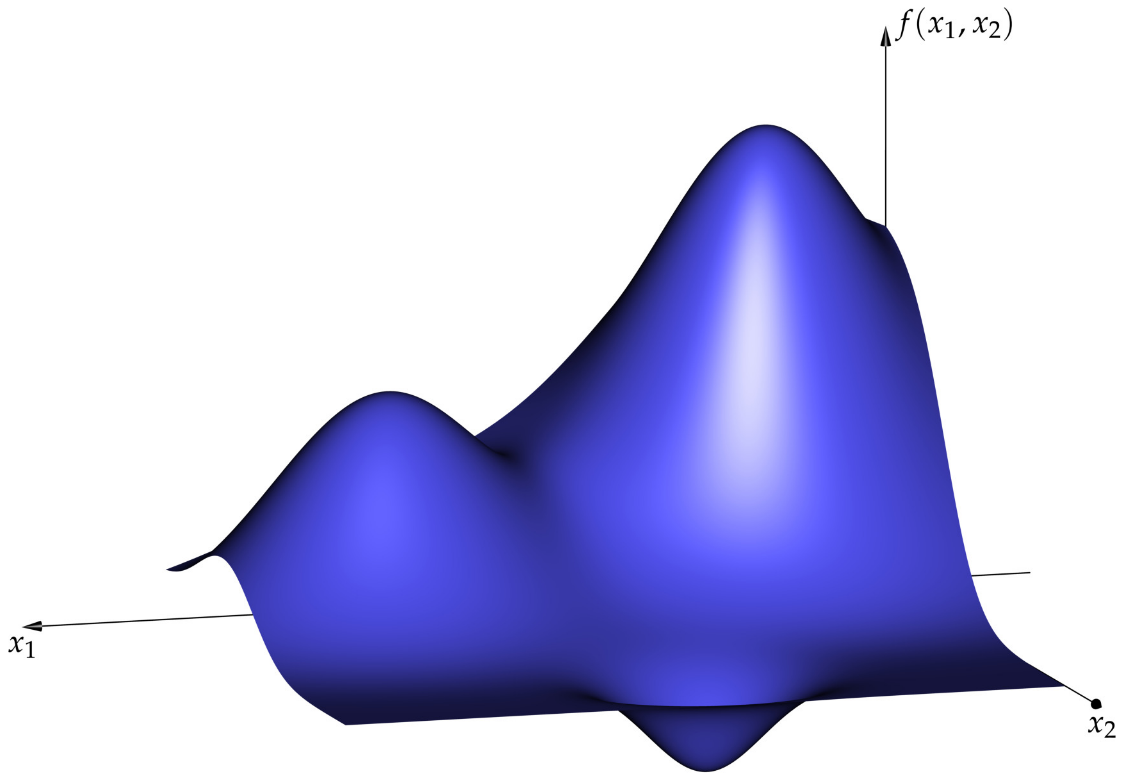


Figure 1: The Franke's function to be fitted, given by equation (2.1) on the preceding page.

2.2 Fitting problem statement

The general problem is to fit a given set of data $D = \{(\vec{x}_i, y_i)\}_{i=1}^{N}$, where $\vec{x}_i$ are inputs of some dimension (2 in this project) and y_i are scalar outputs. A set of basis functions $\{\phi_j\}_{j=1}^{p}$ is chosen, and the goal is to approximate the input data with the linear combination $\beta_j \phi_j(\vec{x})$. If the data set were generated by such a linear combination, there would exist parameters $\{\beta_j\}$ such that $\beta_j \phi_j(\vec{x}_i) = y_i$, which can be rewritten as the matrix equation

$$X\vec{\beta} = \vec{y}, \tag{2.2}$$

where $X \in \mathbb{R}^{N \times p}$ is the matrix with elements $x_{ij} = \phi_j(x_i)$, and $\vec{\beta}$ and $\vec{y}$ are the vectors with elements β_j and y_i.

In general, it will not be possible to find parameters β_j which fulfill this, since the data set will not be generated from the simple basis functions. Consequently, one must find the β_js which in some sense make $X\vec{\beta}$ as close to $\vec{y}$ as possible. The way in which to measure deviation from the perfect solution is called the *cost function*, frequently written $Q(\vec{\beta}; D)$. Regression methods differ in the choice of cost function, while their aim is always to find the parameters β_j which minimise the cost function.

When a regression method has been used to find $\vec{\beta}$, predicted values are denoted $\tilde{y}_i = X_{ij}\beta_j$, while the original, exact values are denoted y_i.

2.3 Ordinary Least Squares

The ordinary least squares method is the simplest and most intuitive, since its cost function is simply the square of the error made by the fit,

$$Q(\beta; D) = \left\|\vec{y} - X\vec{\beta}\right\|_2^2, \tag{2.3}$$

where $\|\cdot\|_p$ denotes the usual p-norm.

2.3.1 Geometric view

The geometric view of the equation $X\vec{\beta} = \vec{y}$ is to find the linear combination of the columns of X equal to $\vec{y}$. This is not necessarily possible if the columns of X do not span $\mathbb{R}^N$, which is not possible if $p < N$. Ordinary least squares seeks to find the linear combination of the columns of X as close to $\vec{y}$ as possible. This is achieved when $X\vec{\beta}$ is equal to the projection of $\vec{y}$ onto the column space of X, i.e.

$$X\vec{\beta} = \mathrm{Proj}_{\mathrm{Col}\,X}\,\vec{y}. \tag{2.4}$$

By construction, this equation will always have a solution, although it may not be unique if the columns of X are not linearly independent. Since $X\vec{\beta}$ is the projection of $\vec{y}$ onto the column space of X, the error, $\vec{y} - X\vec{\beta}$, is orthogonal to the columns of X, i.e. the rows of X^T. Consequently,

$$X^T\left(\vec{y} - X\vec{\beta}\right) = \vec{0} \implies X^T X\vec{\beta} = X^T\vec{y}, \tag{2.5}$$

which is a simple linear set of equations which can be solved for $\vec{\beta}$. This set of equations is called the *normal equations*. If X is non-singular, the minimisation problem in ordinary least squares has the closed form solution

$$\vec{\beta}_{\mathrm{OLS}} = \left(X^T X\right)^{-1} X^T\vec{y}. \tag{2.6}$$

2.3.2 Minimisation view

The cost function for ordinary least squares can be written as

$$Q(\beta; D) = \left\|\vec{y} - X\vec{\beta}\right\|_2^2 = \sum_{i=1}^{N}\left(y_i - X_{ij}\beta_j\right)^2. \tag{2.7}$$

When this is minimised, $\nabla Q = \vec{0}$, where the gradient denotes differentiation with respect to the parameters β_j. The components of the gradient can straightforwardly be calculated from the cost function,

$$\nabla_k Q = \frac{\partial}{\partial\beta_k}\left(\left(y_i - X_{ij}\beta_j\right)\left(y_i - X_{ij}\beta_j\right)\right) = 2\left(y_i - X_{ij}\beta_j\right)\frac{\partial}{\partial\beta_k}\left(y_i - X_{ij}\beta_j\right) = -2X_{ik}\left(y_i - X_{ij}\beta_j\right), \tag{2.8}$$

which can be rewritten on vector form as

$$\nabla Q = -2X^T\left(\vec{y} - X\vec{\beta}\right). \tag{2.9}$$

Setting $\nabla Q = \vec{0}$ gives equation (2.5).

Ordinary least squares has been implemented by calling `dgelss`, which uses a singular value decomposition to prevent numerical instabilities.

2.4 Ridge regression

The cost function used in Ridge regression is

$$Q_\lambda(\vec{\beta}; D) = \left\| \vec{y} - X\vec{\beta} \right\|_2^2 + \lambda \left\| \vec{\beta} \right\|_2^2 = \sum_{i=1}^{N} \left(y_i - X_{ij}\beta_j \right)^2 + \lambda \sum_{i=1}^{p} \beta_i^2, \tag{2.10}$$

which penalises solution vectors $\vec{\beta}$ where some coefficients are large. λ is a parameter which should be chosen with great care. The error is as small as possible for $\lambda = 0$, as this will reproduce the solution found by ordinary least squares, but a non-zero value of λ will give β_js which yield more reasonable predictions for other values of $\vec{x}$ than those contained in the training data set D[2].

Using the derivative of the first term from equation (2.9) on the preceding page, the gradient of the cost function is

$$\nabla Q_\lambda = -2X^T \left(\vec{y} - X\vec{\beta} \right) + 2\lambda\vec{\beta} = -2X^T\vec{y} + 2X^TX\vec{\beta} + 2\lambda\vec{\beta} = -2X^T\vec{y} + 2\left(X^TX + \lambda I \right)\vec{\beta}. \tag{2.11}$$

Minimisation requires $\nabla Q_\lambda = \vec{0}$, which gives

$$\left(X^TX + \lambda I \right)\vec{\beta} = X^T\vec{y}. \tag{2.12}$$

This can be solved for $\vec{\beta}$, and the closed-form solution to the minimisation of the Ridge cost function is

$$\vec{\beta}_{\text{Ridge}} = \left(X^TX + \lambda I \right)^{-1} X^T\vec{y}. \tag{2.13}$$

Having a non-zero λ clearly reduces problems with linearly dependent columns of X and singularity of X^TX.

Ridge regression has been implemented by calculating X^TX and adding λ on the diagonal, calculating X^Ty and using `dposv` to find $\vec{\beta}$ using a Cholesky-decomposition, since $X^TX + \lambda I$ is positive definite.

2.5 LASSO regression

The cost function used in LASSO regression is

$$Q_\lambda(\vec{\beta}; D) = \left\| \vec{y} - X\vec{\beta} \right\|_2^2 + \lambda \left\| \vec{\beta} \right\|_1 = \sum_{i=1}^{N} \left(y_i - X_{ij}\beta_j \right)^2 + \lambda \sum_{i=1}^{p} |\beta_i|, \tag{2.14}$$

which also penalises solution vectors $\vec{\beta}$ where some coefficients are large. LASSO regression has the advantage that certain choices of λ will give a sparse solution, i.e. $\beta_j = 0$ for some values of j[3].

As with the other regression methods, the gradient of the cost function should now be differentiated and set to zero. Mathematicians will now point out that the absolute value is not differentiable at zero and start deriving ingenious minimisation methods to circumvent the problem. This is a non-issue in a physics course — I will now *define* the derivative to be

$$\frac{\mathrm{d}|x|}{\mathrm{d}x} := \operatorname{sgn} x := \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \tag{2.15}$$

and proceed happily.

The gradient of the cost function is thus

$$\nabla Q_\lambda = -2X^T\left(\vec{y} - X\vec{\beta}\right) + \lambda \operatorname{sgn}\left(\vec{\beta}\right), \tag{2.16}$$

and $\nabla Q_\lambda = \vec{0}$ does unfortunately not have a closed-form solution. A separate minimisation algorithm must therefore be used to find the parameters β_j which minimise the cost function. The calculated gradient can be put straight into the gradient descent algorithm with a suitable step length. Alternatively, Newton's method can be used with the second derivative (Hessian) matrix of the cost function. The second derivative of the absolute value is even more problematic, which is solved by setting it equal to zero.

The Hessian matrix of the LASSO cost function is thus the same as the second derivative of the ordinary least squares cost function,

$$H_{kl} = H_{lk} = \frac{\partial^2 Q_\lambda}{\partial \beta_l \partial \beta_k} = \frac{\partial}{\partial \beta_l}(\nabla_k Q) \overset{\text{equation (2.8)}}{=} \frac{\partial}{\partial \beta_l}\left(-2X_{ik}\left(y_i - X_{ij}\beta_j\right)\right) = 2X_{ik}X_{il}, \tag{2.17}$$

which is the component form of the matrix equation

$$H = 2X^T X. \tag{2.18}$$

The gradient can now be rewritten as

$$\nabla Q_\lambda = -2X^T\vec{y} + H\vec{\beta} + \lambda \operatorname{sgn}\left(\vec{\beta}\right). \tag{2.19}$$

Equation (2.23) on the next page in Newton's method can then transformed into

$$H\left(\vec{\beta}_{i+1} - \vec{\beta}_i\right) = 2X^T\vec{y} - H\vec{\beta}_i - \lambda \operatorname{sgn}\left(\vec{\beta}_i\right) \implies H\vec{\beta}_{i+1} = 2X^T\vec{y} - \lambda \operatorname{sgn}\left(\vec{\beta}_i\right). \tag{2.20}$$

Since $H = 2X^T X$, this reduces to the normal equations of ordinary least squares (equation (2.5) on page 4) in the case of $\lambda = 0$, as it should.

Minimisation of the LASSO cost function has the benefit that the second derivative is independent of $\vec{\beta}$. H can therefore be Cholesky-decomposed once, an $\mathcal{O}\left(p^3\right)$ operation, and the result can be used to solve the linear set of equations for each iteration, which is $\mathcal{O}\left(p^2\right)$ with a pre-Cholesky-decomposed matrix. The Cholesky-decomposition can be used instead of an LU-decomposition, reducing the number of floating point operations by a factor of two, because $H = 2X^T X$ is positive definite when X is non-singular.

Evaluation of the gradient only involves matrix-vector products, which is also an $\mathcal{O}\left(p^2\right)$ operation. The minimisation can, therefore, be done with one initial $\mathcal{O}\left(p^3\right)$ and then only $\mathcal{O}\left(p^2\right)$ operations per iteration, while the more general problem requires both the construction and the Cholesky-decomposition of H, an $\mathcal{O}\left(p^3\right)$ operation, for every single iteration.

LASSO regression has been implemented by calculating the Cholesky-decomposition of H using `dpptrf`, guessing $\beta_j = 1$ and then repeatedly solving equation (2.20) using `dpptrs` until convergence, as illustrated by the following snippet.

```fortran
beta(:) = 1
minimisation: do i = 1, max_iterations
    ! calculate right-hand side
    beta(:) = two_X_T_y - lambda*sgn(beta)

    ! solve H beta_new = 2 X^T y - lambda sgn(beta_old)
    call dpptrs('L', p, 1, H_cholesky, beta, p, info)
    call check_info(info, "dpptrs")

    ! calculate gradient and its norm
    grad(:) = - two_X_T_y + matmul(H, beta) + lambda*sgn(beta)
    grad_norm = norm2(grad)

    ! check for convergence
    if (grad_norm < grad_tolerance) exit minimisation
end do minimisation
```

This implementation gives both good performance and accurate results for small values of λ. Larger values, however, give rise to issues. The theory suggests that a large λ should force some of the parameters β_j to become zero, but the cost function is not differentiable, and certainly not double differentiable, when β_j is zero. Consequently, Newton's method struggles to converge when some β_js are zero in the true minimum of the LASSO cost function. Smarter methods such as coordinate descent[1] should therefore be used, as in e.g. scikit-learn.

2.6 Minimisation methods

The important step in a regression method, or indeed in any statistical learning method, is to minimise the cost function. Certain cost functions, such as those of ordinary least squares and Ridge regression, admit analytical closed-form solutions for the parameters β_j which minimise $Q(\vec{\beta}; D)$, while most methods, including LASSO regression and logistic regression, require usage of some general minimisation algorithm.

2.6.1 Newton's method

Since the gradient of the cost function, ∇Q, is a perfectly normal function from $\mathbb{R}^p$ to $\mathbb{R}^p$, it can be Taylor expanded around some point $\vec{\beta}_0$. This Taylor expansion can then be evaluated at a point $\vec{\beta}_0 + \delta\vec{\beta}$. To first order in $\delta\vec{\beta}$,

$$\nabla Q\left(\vec{\beta}_0 + \delta\vec{\beta}\right) = \nabla Q\left(\vec{\beta}_0\right) + H\left(\vec{\beta}_0\right)\delta\vec{\beta}, \tag{2.21}$$

where $H\left(\vec{\beta}_0\right)$ is the derivative of ∇Q, i.e. the Hessian of Q, evaluated at β_0. Given a guess β_0 somewhere near the minimum of Q, a better estimate can be found by solving for the $\delta\vec{\beta}$ which makes $\nabla Q(\vec{\beta}_0 + \delta\vec{\beta}) = \vec{0}$, i.e.

$$H\left(\vec{\beta}_0\right)\delta\vec{\beta} = -\nabla Q\left(\vec{\beta}_0\right), \tag{2.22}$$

and, in general, solving

$$H\left(\vec{\beta}_i\right)\delta\vec{\beta}_i = -\nabla Q\left(\vec{\beta}_i\right), \tag{2.23}$$

letting $\vec{\beta}_{i+1} = \vec{\beta}_i + \delta\vec{\beta}_i$ and continuing the process until the method has converged sufficiently. The above equation is a simple linear set of equations.

2.6.2 Gradient descent

Evaluating the Hessian matrix and inverting it can sometimes be infeasible, for example due to poor time complexity or being woefully undefined. In such cases, one can replace the Hessian H with a number $1/\alpha$. Equation (2.23) on the preceding page is then rewritten as

$$\vec{\beta}_{i+1} = \vec{\beta}_i - \alpha\nabla Q\left(\vec{\beta}_i\right), \tag{2.24}$$

with a simple geometric interpretation: By going a small step in the opposite direction of the gradient, the value of the cost function will decrease and $\vec{\beta}$ will approach the true minimum. A small step will guarantee convergence for a convex function at the cost of slow convergence, while a larger value may give faster convergence or not converge at all.

2.7 Performance of regression methods

While the cost function is minimised by all regression methods, its value is not particularly meaningful. In particular, the cost functions in Ridge and LASSO regression depend on the parameter λ, and a measure independent of λ should be used to determine the optimal λ. Other functions are therefore introduced to measure the performance of a given regression method and/or a choice of parameters.

The simplest measure is the mean square error,

$$\text{MSE}\left(\vec{\beta}, D\right) = \frac{1}{N}\|\vec{y} - \vec{\tilde{y}}\|_2^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2, \tag{2.25}$$

which should be as small as possible. Another measure is the R^2 score, defined as

$$R^2(\vec{\beta}, D) = 1 - \frac{\|\vec{y} - \vec{\tilde{y}}\|_2^2}{\|\vec{y} - \bar{y}\|_2^2} = 1 - \frac{\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \tag{2.26}$$

where $\bar{y}$ is the mean of the measured values. The R^2 score should be as close to 1 as possible.

Prediction and these measures of performance can be applied to two main types of data. Firstly, it can be applied to the data from which $\vec{\beta}$ was derived, which is called the training data. Ordinary least squares, corresponding to $\lambda = 0$ for the other methods, will, by definition, give the best results for this data set. Secondly, the performance can be measured for values not among the training data, called test data. Ridge and LASSO are expected to outperform ordinary least squares for small, non-zero values of λ for this category of data.

2.8 Resampling methods

Resampling methods are techniques to improve the prediction accuracy and obtain estimates for quantities such as the variance of $\vec{\beta}$ by repeatedly dividing the data set into training and test data.

Two simple examples are k-fold cross-validation and bootstrapping. The former partitions the data set into k partitions. One of these subsets is chosen as test data on which the performance is measured, while the rest are used as training data. This process is the repeated k times, so that each subset is used once as test data. Bootstrapping, on the other hand, repeatedly creates training data sets by randomly selecting N values from the data set with replacement, while the same, separate data set is used as test data each time, as illustrated by the snippet below.

```fortran
!$omp parallel do private(fitter, tmp_real, indices, &
!$omp&                    y_selection, X_selection)
bootstraps: do i = 1, num_bootstraps
    if (.not. allocated(fitter)) fitter = self%fitter

    ! choose size(y_train) random indices
    call random_number(tmp_real)
    indices(:) = nint((N_train-1)*tmp_real) + 1

    ! select from training data
    y_selection(:)   = y_train(indices)
    X_selection(:,:) = X_train(indices,:)

    fitter%X = X_selection

    ! fit to selection
    call fitter%fit(y_values=y_selection)

    ! apply to test data, evaluate performance
    call fitter%predict(x_test, y_predictions(:,i), &
                        y_test, MSEs(i), R2s(i))

    betas(:, i) = fitter%beta
end do bootstraps
!$omp end parallel do
```

2.9 Bias and variance

The choice of the number of basis functions, p, determines the complexity of the model to which the data set is fitted. A higher complexity will mean that the model will be a better approximation to the training data, which is said to reduce the *bias* of the model. On the other hand, a higher complexity may decrease the model's ability to predict reasonable values for test data, since a more complex model will be more affected by noise in the model. This is said to increase the model's *variance*. The best prediction performance is therefore achieved for a complexity which balances bias and variance, which is called the bias-variance trade-off[2].

Following [2], a mathematical manifestation of the bias-variance trade-off can be derived by assuming that the measured data set $D = \{(\vec{x}_i, y_i)\}_{i=1}^{N}$ is generated by a combination of some model $f(\vec{x})$ (for example Franke's function) and random noise, specifically

$$y_i = f(\vec{x}_i) + \varepsilon_i =: f_i + \varepsilon_i, \tag{2.27}$$

where the variables ε_i are independent and normally distributed with variance σ^2 centred in zero. Given a data set D, a prediction $\tilde{y}_i^D$ can be made by any of the regression methods discussed above.

An expected mean square error can be found by averaging over all datasets D and all noises $\vec{\varepsilon}$,

$$E_{D,\varepsilon}\left[\|\vec{y}-\vec{\tilde{y}}_D\|_2^2\right] = E_{D,\varepsilon}\left[\left\|\vec{y}-\vec{f}+\vec{f}-\vec{\tilde{y}}_D\right\|_2^2\right] = E_{D,\varepsilon}\left[\left\|\vec{y}-\vec{f}\right\|_2^2+\left\|\vec{f}-\vec{\tilde{y}}_D\right\|_2^2+2\left(\vec{y}-\vec{f}\right)\cdot\left(\vec{f}-\vec{\tilde{y}}_D\right)\right], \tag{2.28}$$

and using the linearity of the expectation value,

$$= E_{D,\varepsilon}\left[\left\|\vec{y}-\vec{f}\right\|_2^2\right] + E_{D,\varepsilon}\left[\left\|\vec{f}-\vec{\tilde{y}}_D\right\|_2^2\right] + 2E_{D,\varepsilon}\left[\left(\vec{y}-\vec{f}\right)\cdot\left(\vec{f}-\vec{\tilde{y}}_D\right)\right] \tag{2.29}$$

$$= E_{D,\varepsilon}\left[\|\vec{\varepsilon}\|_2^2\right] + E_{D,\varepsilon}\left[\left\|\vec{f}-\vec{\tilde{y}}_D\right\|_2^2\right] + 2E_{D,\varepsilon}\left[\vec{\varepsilon}\cdot\left(\vec{f}-\vec{\tilde{y}}_D\right)\right] \tag{2.30}$$

$$= \sigma^2 + E_{D,\varepsilon}\left[\left\|\vec{f}-\vec{\tilde{y}}_D\right\|_2^2\right] + 2\cancel{E_\varepsilon[\vec{\varepsilon}]}\cdot E_D\left[\left(\vec{f}-\vec{\tilde{y}}_D\right)\right] \tag{2.31}$$

$$= \sigma^2 + E_D\left[\left\|\vec{f}-\vec{\tilde{y}}_D\right\|_2^2\right]. \tag{2.32}$$

This expression can be further decomposed by adding and subtracting the average prediction, $E_D\left[\vec{\tilde{y}}_D\right]$,

$$E_D\left[\|\vec{y}-\vec{\tilde{y}}_D\|_2^2\right] = \sigma^2 + E_D\left[\left\|\vec{f}-E_D\left[\vec{\tilde{y}}_D\right]+E_D\left[\vec{\tilde{y}}_D\right]-\vec{\tilde{y}}_D\right\|_2^2\right] \tag{2.33}$$

$$= \sigma^2 + \left\|\vec{f}-E_D\left[\vec{\tilde{y}}_D\right]\right\|_2^2 + E_D\left[\|E_D\left[\vec{\tilde{y}}_D\right]-\vec{\tilde{y}}_D\|_2^2\right] \tag{2.34}$$

$$+2E_D\left[\left(\vec{f}-E_D\left[\vec{\tilde{y}}_D\right]\right)\cdot\cancel{\left(E_D\left[\vec{\tilde{y}}_D\right]-\vec{\tilde{y}}_D\right)}\right] \tag{2.35}$$

$$= \sigma^2 + \left\|\vec{f}-E_D\left[\vec{\tilde{y}}_D\right]\right\|_2^2 + E_D\left[\|E_D\left[\vec{\tilde{y}}_D\right]-\vec{\tilde{y}}_D\|_2^2\right]. \tag{2.36}$$

The first term, σ^2, represents the noise in the generated data, which no model can overcome. The second term measures the deviation of the average prediction from the true, noise-free model, which is the (squared) bias. Lastly, the third term measures how much the predictions vary and is called the variance. A complex model (large p) will minimise the second term, since a complex model will be better suited to fit the true model, f, while the third term will increase with model complexity since the fitting procedure will be more sensitive to noise in the different data sets.

Unfortunately, the above bias-variance decomposition requires knowledge of the exact model behind the data, f. A more computationally practical expression could have been derived by adding and subtracting $E_D\left[\vec{\tilde{y}}_D\right]$ in equation (2.28), which gives the expression

$$E_D\left[\|\vec{y}-\vec{\tilde{y}}_D\|_2^2\right] = \|\vec{y}-E_D\left[\vec{\tilde{y}}_D\right]\|_2^2 + E_D\left[\|E_D\left[\vec{\tilde{y}}_D\right]-\vec{\tilde{y}}_D\|_2^2\right] \tag{2.37}$$

by manipulations analogous to the ones used above. This is another formulation of the bias-variance decomposition which can be used without any information about the underlying model. Dividing by N gives the average MSE on the left-hand side.

2.10 Implementation

The three regression methods and bootstrapping have been implemented in a polymorphic class hierarchy in Fortran. An object-oriented approach ensures ease of reuse for later projects, while also simplifying certain parts of the code and organisation. For instance, the bootstrapping algorithm can take in a regression method object, which is guaranteed to have methods for prediction and fitting, and not care whether ordinary least squares, Ridge or LASSO regression is used. Fortran was chosen because it combines excellent performance with object-oriented capabilities and a nice syntax for numerical work.

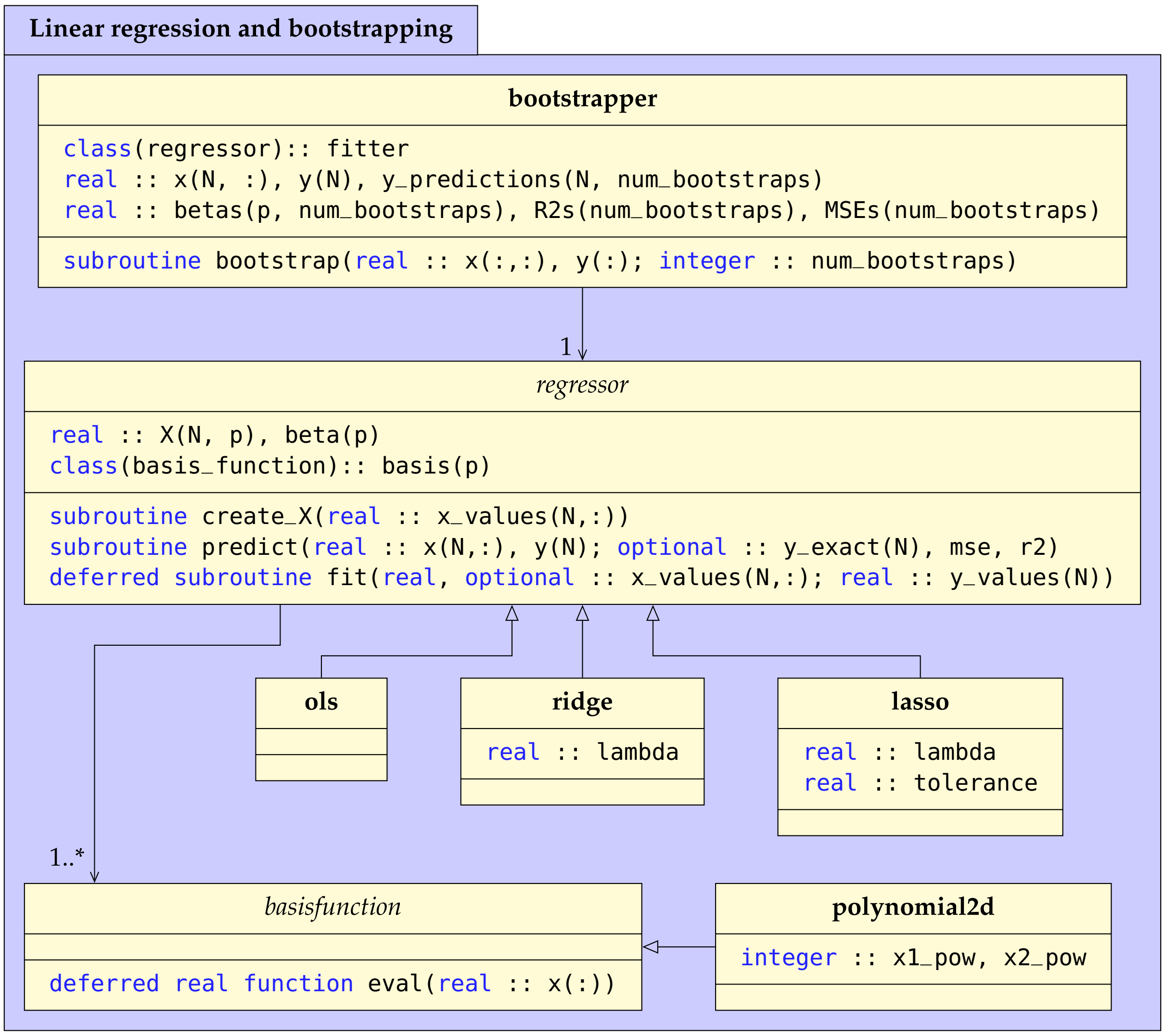
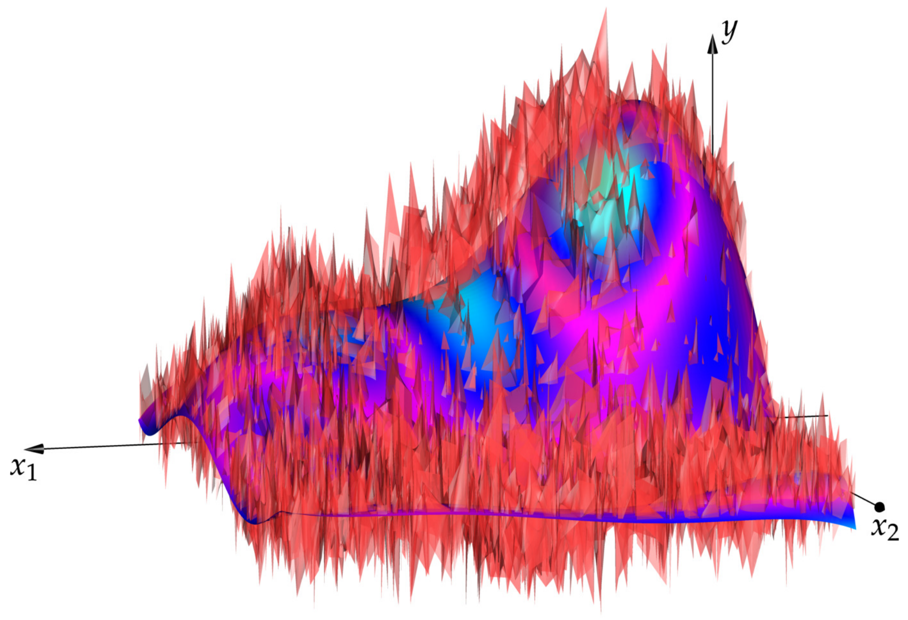

Linear regression and bootstrapping
bootstrapper
class(regressor):: fitter
real :: x(N, :), y(N), y_predictions(N, num_bootstraps)
real :: betas(p, num_bootstraps), R2s(num_bootstraps), MSEs(num_bootstraps)
subroutine bootstrap(real :: x(:,:), y(:); integer :: num_bootstraps)
1
regressor
real :: X(N, p), beta(p)
class(basis_function):: basis(p)
subroutine create_X(real :: x_values(N,:))
subroutine predict(real :: x(N,:), y(N); optional :: y_exact(N), mse, r2)
deferred subroutine fit(real, optional :: x_values(N,:); real :: y_values(N))
ols
ridge
real :: lambda
lasso
real :: lambda
real :: tolerance
1..*
basisfunction
deferred real function eval(real :: x(:))
polynomial2d
integer :: x1_pow, x2_pow

Figure 2: Class hierarchy of implementation. All `real` variables are of kind `real64` from `iso_fortran_env`.

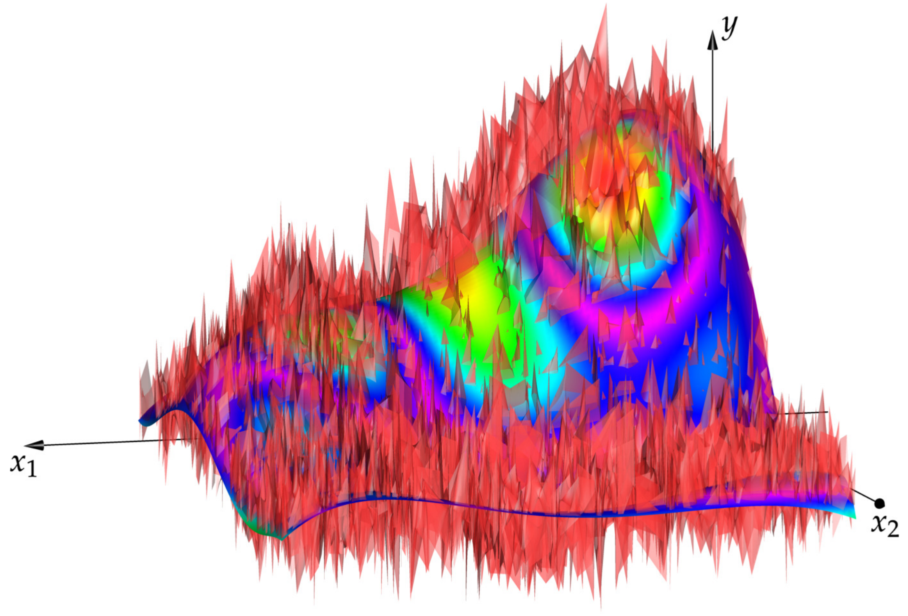
3 Analysis of methods

The accuracy and viability of the methods can be studied on a test case where the underlying function is known. Such an example is Franke's function shown in figure 1 on page 3.

3.1 Verification of theory and fitting



(a) Ordinary least squares regression.



(b) Ridge regression.

Figure 3: Comparison of Franke's function with noise (red) and a fifth degree polynomial approximation (multi-coloured). The colour of the polynomial approximation shows its deviation from Franke's function. Figure 4 on the following page shows that the parameters are significantly different for the different methods, yet the resulting approximations are approximately equal.

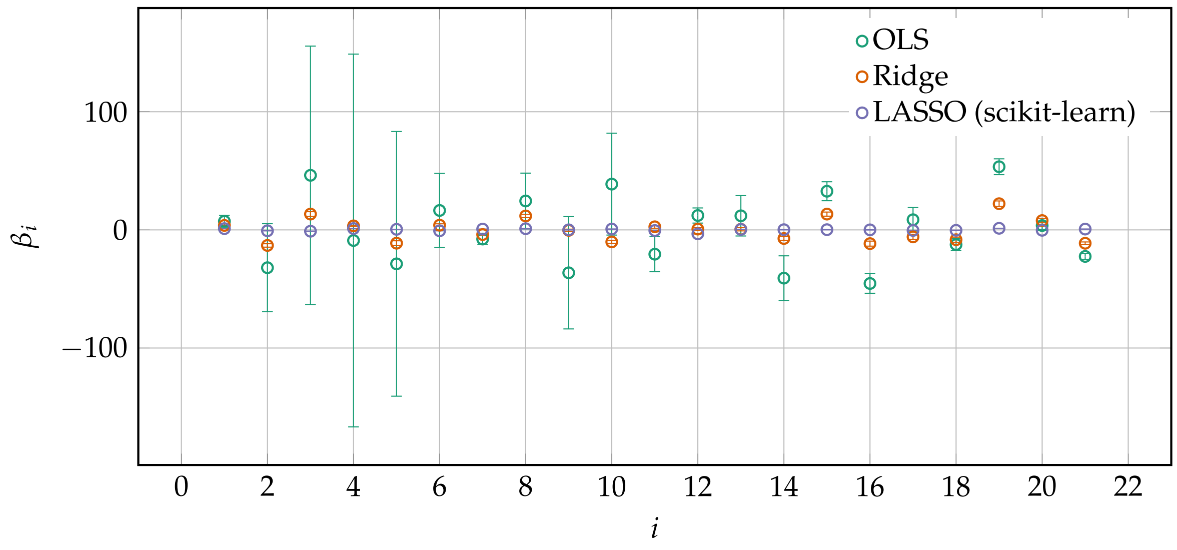


Figure 4: Comparison of parameters for the different regression methods when applied to Franke's function with noise and using polynomials up to fifth order. The Ridge coefficients are severely suppressed, yet table 1 shows that the fitting quality is virtually the same. LASSO regression (using scikit-learn, since my Newton minimiser won't f**king converge) gives a sparse solution, i.e. some β_js are exactly zero. Confidence intervals are estimated as $\beta_j \pm 2\sigma_{\beta_j}$, where σ_{β_j} is the standard deviation of β_j calculated from bootstrapping.

Table 1: Measure of performance on training and test data sampled from Franke's function with noise for the different methods. Ordinary least squares should, by design, perform best on the training data, while Ridge and LASSO are expected to perform better on test data not used in the fitting due to their regularisation parameter. The advantage of regularised methods is amplified when a smaller number of points is used (10 000 points were used here for plotting purposes). See also table 2 on the following page, which shows the average error on test data for many bootstrap samples.

Method	MSE (train)	R^2 (train)	MSE (test)	R^2 (test)
Ridge	0.0120	0.8502	0.0146	0.8159
OLS	0.0117	0.8538	0.0140	0.8238
LASSO	0.0171	0.8112	0.0178	0.8083

3.2 Bias and variance

According to the bias-variance decomposition, the mean squared error should be equal to the sum of the bias (squared) and the variance. The bias and variance are calculated from bootstrapping, and the results show good agreement with the theoretical prediction.

Table 2: Verification of the bias-variance decomposition via bootstrapping for fitting of Franke's function with noise. As expected, the sum of the bias and the variance is equal to the mean squared error, and the regularised methods perform better on test data than ordinary least squares.

Method	MSE	Bias+Variance	Bias	Variance
Ridge	$2.473 \cdot 10^{-1}$	$2.473 \cdot 10^{-1}$	$2.447 \cdot 10^{-1}$	$2.636 \cdot 10^{-3}$
OLS	$1.642 \cdot 10^{0}$	$1.642 \cdot 10^{0}$	$1.236 \cdot 10^{0}$	$4.058 \cdot 10^{-1}$
LASSO	$1.695 \cdot 10^{-2}$	$1.695 \cdot 10^{-2}$	$1.691 \cdot 10^{-2}$	$3.300 \cdot 10^{-5}$

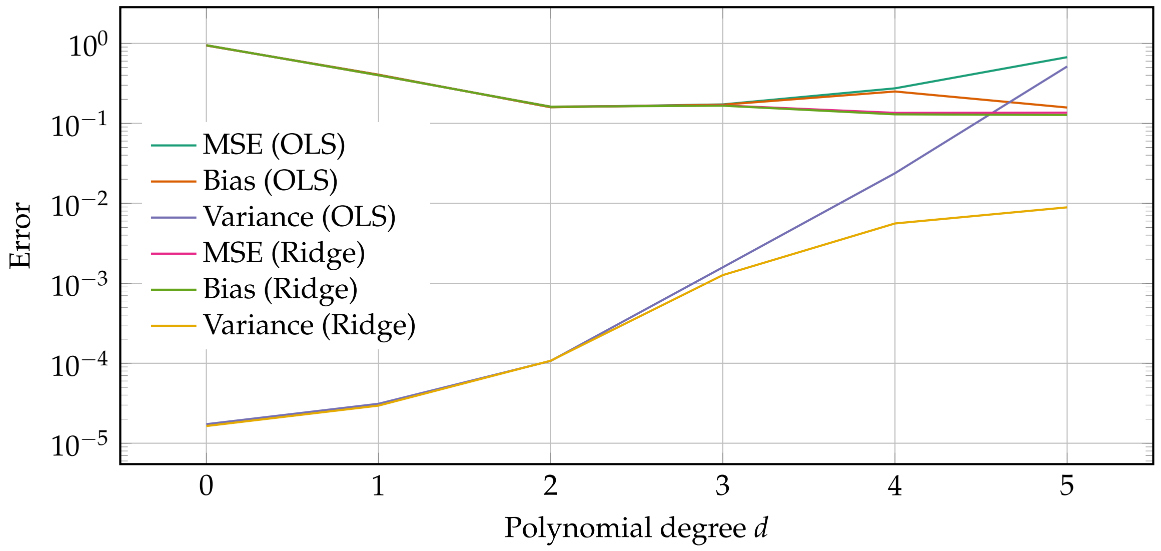


Figure 5: Visualisation of bias-variance trade-off. The mean squared error for test data is expected to first decrease with increasing complexity as the bias decreases towards σ^2, and then increase as the variance becomes dominant. Additionally, the regularisation of the Ridge regressor is intended to keep the variance moderate even with a high polynomial degree.

3.3 Effect of noise

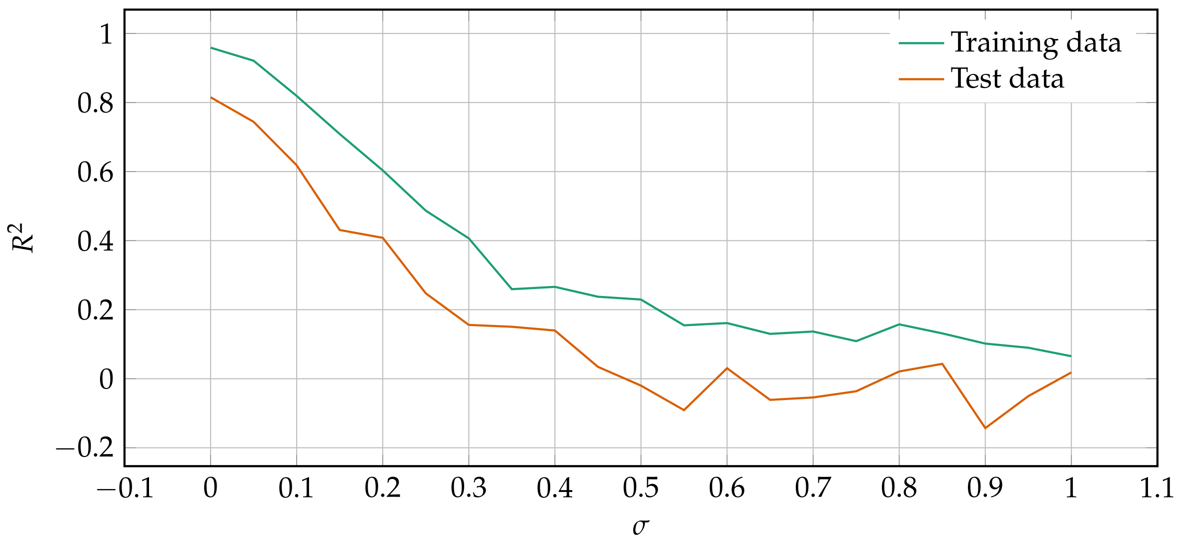


Figure 6: R^2 score of Ridge regression with $\lambda = 0.001$ when normally distributed noise with a standard deviation of σ is added to Franke's function. 400 data points are used, and the performance is clearly better for the training data to which the model was fitted than novel test data. A higher number of data points reduces the overfitting, so that the prediction and its performance are less affected by noise.

3.4 Effect of regularisation

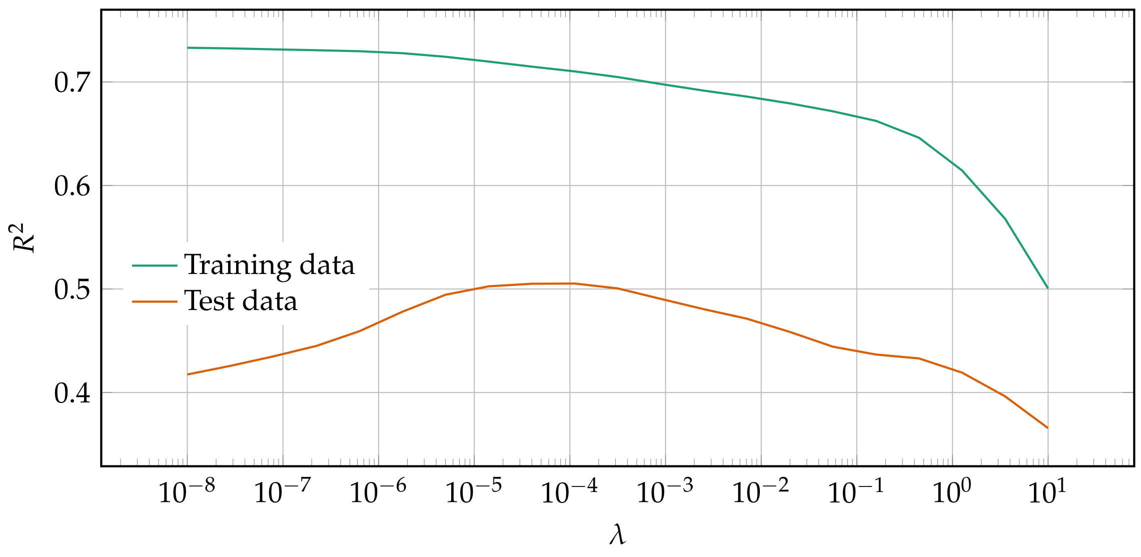


Figure 7: R^2-score as a function of the regularisation parameter λ for Ridge regression. $\lambda = 0$ would correspond to ordinary least squares. While ordinary least squares by construction give the best performance on training data, the results show that a smart choice of λ can give better predicting performance since the regularisation reduces overfitting and variance.

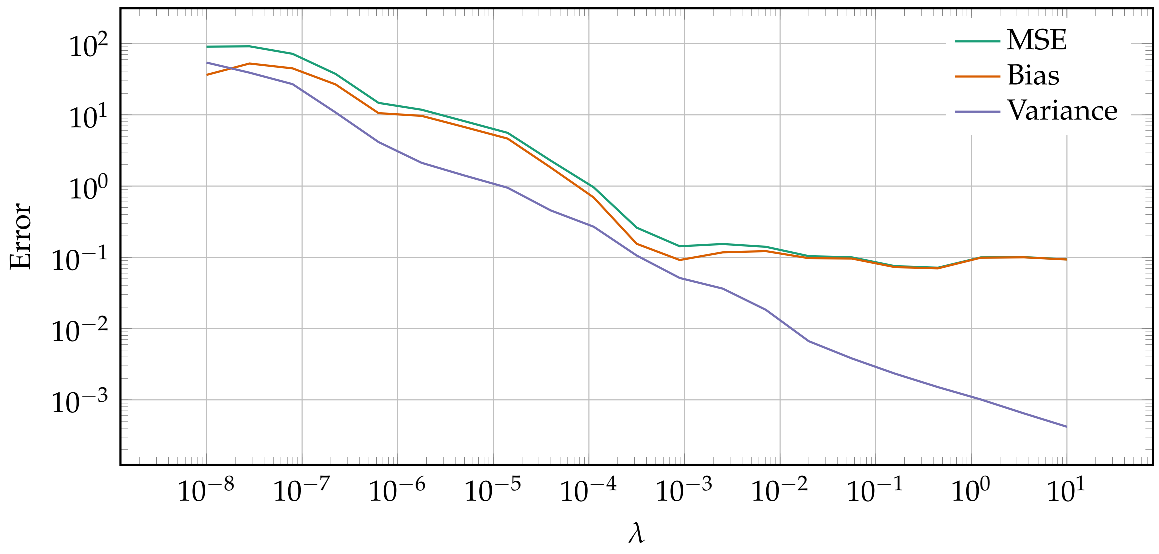


Figure 8: Error, bias and variance as a function of the regularisation parameter λ for Ridge regression, calculated from bootstrapping. The variance, which is the error due to overfitting of the noise, gradually decreases as λ increases, while a too large λ causes the bias to increase because the severe suppression of the β_j coefficients makes it impossible for the polynomial expansion to approximate the model function (Franke's function).

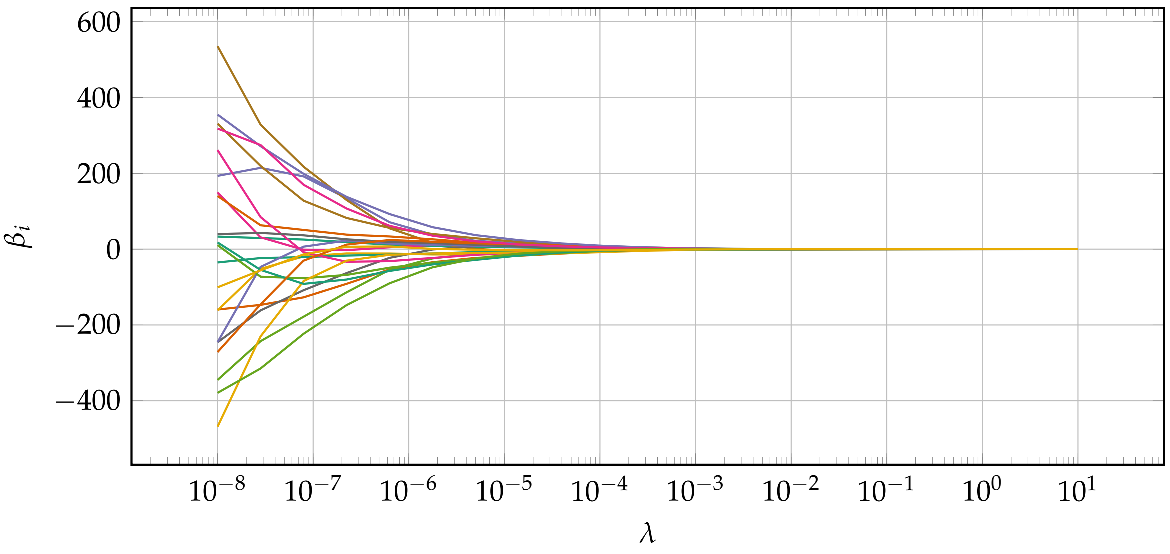


Figure 9: Coefficients β_j as a function of the regularisation parameter λ for Ridge regression. Regularisation suppresses the coefficients due to the penalising $\lambda\|\vec{\beta}\|_2^2$ term in the cost function, in order to achieve better prediction abilities due to lower variance and effect of noise. LASSO regression would set many of the coefficients to exactly zero, ref. figure 4 on page 13.

4 Geographical data

The previous section verified the implementation of ordinary least squares and Ridge regression (as well as scikit-learn's LASSO regression), as well as some theoretical predictions. It is now time to apply them to real-world data, such as the geographical data in the figure below.

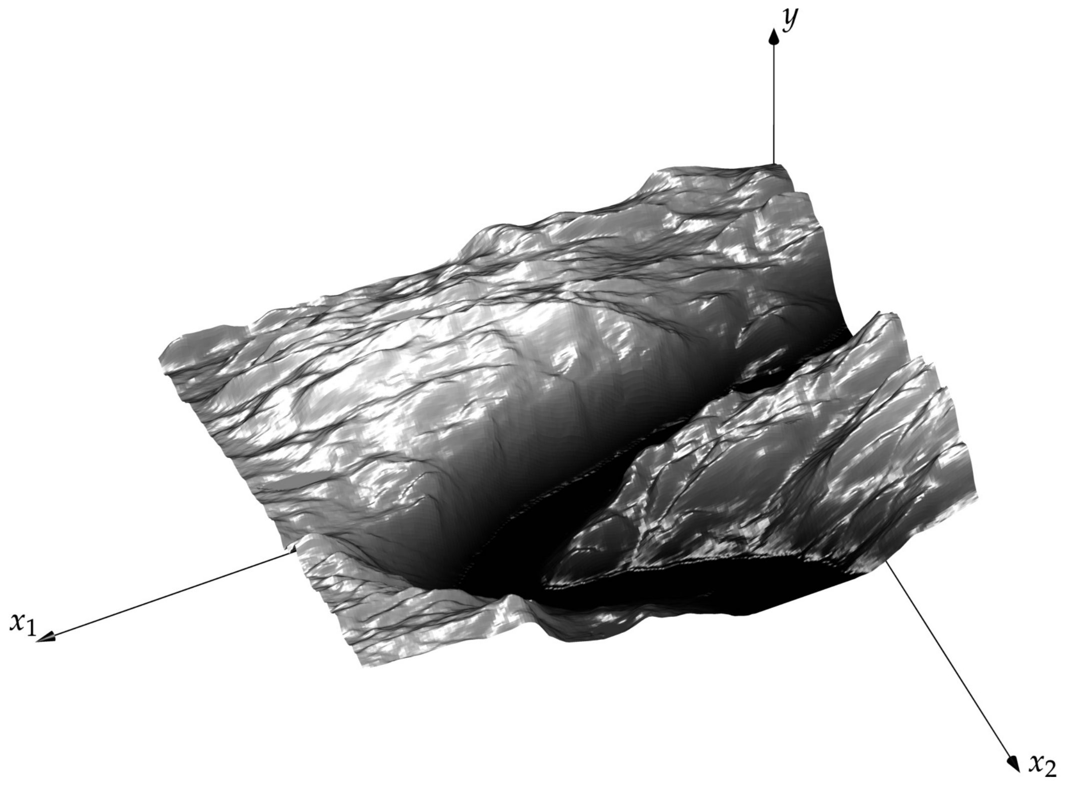


Figure 10: Some Norwegian geography, complete with a fjord.

To determine the best possible model for the terrain, I have chosen to iterate over different polynomial degrees and compare ordinary least squares with Ridge for a selection of λs for each degree. The best λ is determined for each degree, and the overall best approximation (measured by MSE for test data in bootstrap) is visualised in figure 12 on page 19.

Table 3: Errors for different approximations to figure 10 on the previous page. Several values of λ for Ridge regression is tried for each degree, and the one resulting in the smallest test MSE during bootstrapping is reported.

d	MSE (OLS)	Bias (OLS)	Variance (OLS)	MSE (Ridge)	Bias (Ridge)	Variance (Ridge)	λ
1	$1.1 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$4.7 \cdot 10^{-7}$	$1.0 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$5.4 \cdot 10^{-7}$	$1.0 \cdot 10^{-2}$
2	$1.1 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$1.9 \cdot 10^{-6}$	$1.1 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$1.3 \cdot 10^{-6}$	$1.0 \cdot 10^{1}$
3	$6.1 \cdot 10^{-3}$	$6.1 \cdot 10^{-3}$	$4.9 \cdot 10^{-6}$	$5.9 \cdot 10^{-3}$	$5.9 \cdot 10^{-3}$	$3.4 \cdot 10^{-6}$	$6.3 \cdot 10^{-1}$
4	$9.4 \cdot 10^{-3}$	$9.4 \cdot 10^{-3}$	$1.6 \cdot 10^{-5}$	$5.0 \cdot 10^{-3}$	$5.0 \cdot 10^{-3}$	$3.6 \cdot 10^{-6}$	$1.6 \cdot 10^{-1}$
5	$7.1 \cdot 10^{-3}$	$7.1 \cdot 10^{-3}$	$6.1 \cdot 10^{-5}$	$4.9 \cdot 10^{-3}$	$4.9 \cdot 10^{-3}$	$3.4 \cdot 10^{-6}$	$1.6 \cdot 10^{-1}$
6	$2.7 \cdot 10^{-2}$	$2.7 \cdot 10^{-2}$	$2.2 \cdot 10^{-4}$	$3.6 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$2.9 \cdot 10^{-5}$	$4.0 \cdot 10^{-5}$
7	$8.4 \cdot 10^{-3}$	$7.4 \cdot 10^{-3}$	$9.7 \cdot 10^{-4}$	$3.7 \cdot 10^{-3}$	$3.7 \cdot 10^{-3}$	$1.4 \cdot 10^{-5}$	$6.3 \cdot 10^{-4}$
8	$7.8 \cdot 10^{-2}$	$7.2 \cdot 10^{-2}$	$6.1 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$1.1 \cdot 10^{-5}$	$2.5 \cdot 10^{-3}$
9	$9.2 \cdot 10^{-1}$	$9.0 \cdot 10^{-1}$	$2.0 \cdot 10^{-2}$	$3.4 \cdot 10^{-3}$	$3.4 \cdot 10^{-3}$	$6.2 \cdot 10^{-6}$	$1.0 \cdot 10^{-2}$
10	$3.3 \cdot 10^{0}$	$3.1 \cdot 10^{0}$	$1.2 \cdot 10^{-1}$	$2.8 \cdot 10^{-3}$	$2.8 \cdot 10^{-3}$	$3.9 \cdot 10^{-5}$	$1.6 \cdot 10^{-4}$
11	$1.7 \cdot 10^{1}$	$1.6 \cdot 10^{1}$	$3.4 \cdot 10^{-1}$	$2.4 \cdot 10^{-3}$	$2.4 \cdot 10^{-3}$	$3.8 \cdot 10^{-5}$	$1.6 \cdot 10^{-4}$
12	$1.9 \cdot 10^{1}$	$1.6 \cdot 10^{1}$	$2.6 \cdot 10^{0}$	$2.4 \cdot 10^{-3}$	$2.4 \cdot 10^{-3}$	$2.1 \cdot 10^{-5}$	$6.3 \cdot 10^{-4}$
13	$2.6 \cdot 10^{2}$	$2.6 \cdot 10^{2}$	$9.7 \cdot 10^{0}$	$2.2 \cdot 10^{-3}$	$2.2 \cdot 10^{-3}$	$2.0 \cdot 10^{-5}$	$6.3 \cdot 10^{-4}$
14	$9.9 \cdot 10^{2}$	$9.3 \cdot 10^{2}$	$5.5 \cdot 10^{1}$	$2.3 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	$1.7 \cdot 10^{-5}$	$6.3 \cdot 10^{-4}$
15	$1.8 \cdot 10^{4}$	$1.8 \cdot 10^{4}$	$3.2 \cdot 10^{2}$	$2.4 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	$1.0 \cdot 10^{-5}$	$2.5 \cdot 10^{-3}$
16	$7.4 \cdot 10^{3}$	$5.4 \cdot 10^{3}$	$2.0 \cdot 10^{3}$	$2.3 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	$1.2 \cdot 10^{-5}$	$2.5 \cdot 10^{-3}$
17	$5.3 \cdot 10^{4}$	$4.9 \cdot 10^{4}$	$3.9 \cdot 10^{3}$	$2.4 \cdot 10^{-3}$	$2.4 \cdot 10^{-3}$	$1.3 \cdot 10^{-5}$	$2.5 \cdot 10^{-3}$
18	$3.8 \cdot 10^{5}$	$3.6 \cdot 10^{5}$	$1.8 \cdot 10^{4}$	$2.4 \cdot 10^{-3}$	$2.4 \cdot 10^{-3}$	$1.2 \cdot 10^{-5}$	$2.5 \cdot 10^{-3}$
19	$1.8 \cdot 10^{5}$	$9.5 \cdot 10^{4}$	$8.1 \cdot 10^{4}$	$2.4 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	$8.2 \cdot 10^{-5}$	$1.0 \cdot 10^{-5}$
20	$4.6 \cdot 10^{5}$	$3.0 \cdot 10^{5}$	$1.6 \cdot 10^{5}$	$2.3 \cdot 10^{-3}$	$2.2 \cdot 10^{-3}$	$9.5 \cdot 10^{-5}$	$1.0 \cdot 10^{-5}$
21	$7.5 \cdot 10^{5}$	$4.1 \cdot 10^{5}$	$3.5 \cdot 10^{5}$	$2.4 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	$1.1 \cdot 10^{-4}$	$1.0 \cdot 10^{-5}$
22	$9.0 \cdot 10^{5}$	$4.5 \cdot 10^{5}$	$4.5 \cdot 10^{5}$	$2.5 \cdot 10^{-3}$	$2.4 \cdot 10^{-3}$	$5.8 \cdot 10^{-5}$	$4.0 \cdot 10^{-5}$

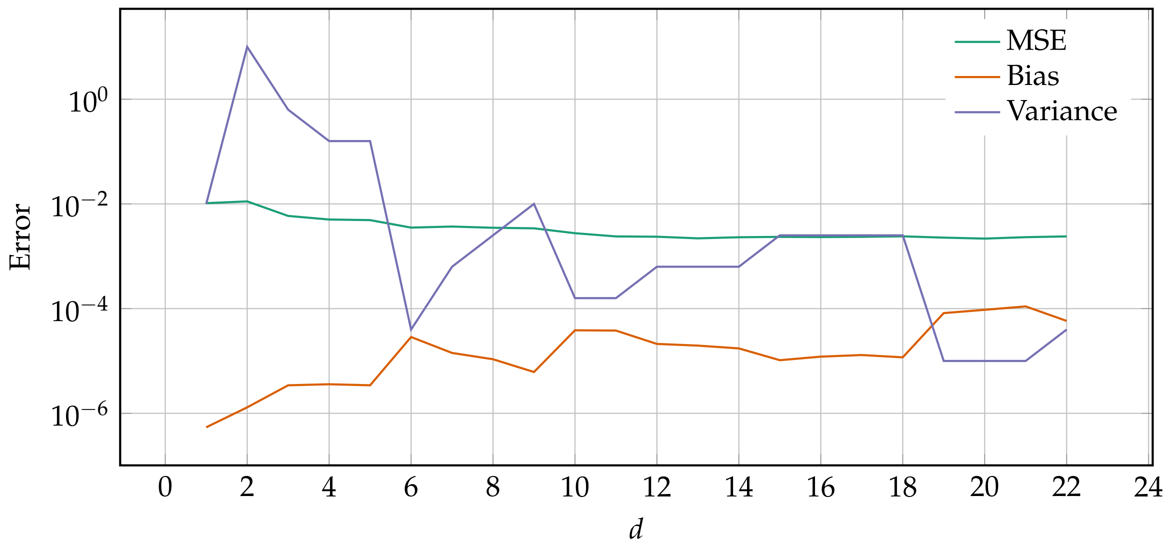


Figure 11: Error, bias and variance as a function of polynomial degree d for Ridge regression. The parameter λ is estimated for each d by bootstrapping for different λs and choosing the one which gives the smallest mean squared error for the test data.

Bias and variance no longer add up to the mean squared error with the same accuracy as when approximating Franke's function with noise. This is most likely due to the assumption of identically, independently and normally distributed noise in the derivation, which was added artificially. Geographical data is certainly not independent, and the other assumptions may also not hold.

Ideally, there should be some polynomial degree for which the error is minimised, since overfitting will occur and the variance will increase as the degree of the polynomial grows and the approximation becomes more sensitive to noise. Overfitting requires that the number of basis functions, $p = (d+1)(d+2)/2$, becomes non-neglibile compared to the number of data points. I have here used approximately 65 000 data points, meaning that a polynomial must be of a very high degree in order to overfit. Ridge regression scales as $\mathcal{O}(p^3) = \mathcal{O}(d^6)$ and ordinary least squares possibly even worse, so the runtime increases drastically when the polynomial degree becomes large. I plan to rerun my program after the deadline and upload an extended data set to github, although this report is intended to be readable as a complete answer (in case of point deductions due to late deliveries).

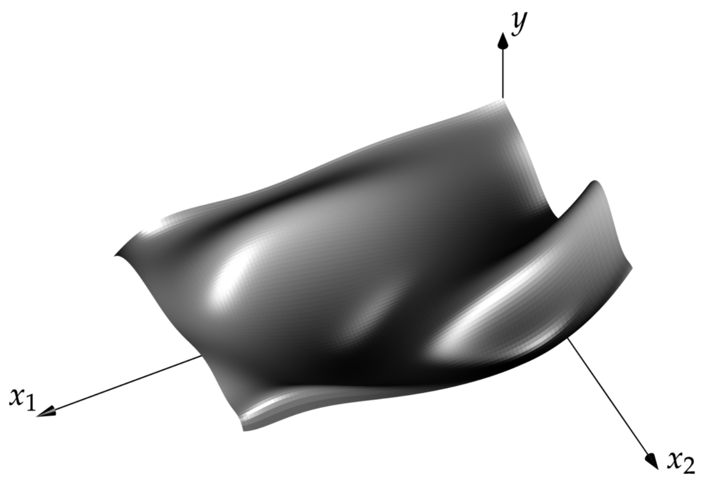


Figure 12: Polynomial approximation to the geographical data in figure 10 on page 17 using the best model from table 3 on the previous page. The model captures key elements such as the fjord, while missing the finer details of the mountaineous landscape. In particular, the steep climb from the fjord is almost step-like, which no polynomial of finite degree can approximate.

5 Summary and conclusion

This report started with a lengthy discussion of theory, culminating in three different regression methods, some error analysis and a few implementational tidbits. Ordinary least squares is derived from a minimisation of the training error, which has been proven correct. The bias-variance decomposition showed that the predicting error can be decomposed into bias, which measures the approximating function's ability to model the data's underlying function, and variance, which measures how much the approximation varies when applied to different parts of the data set, such as in bootstrapping. Predicting models obtained from ordinary least squares have been inferior to Ridge and LASSO regression when applied to test data not used in the fitting procedure, as these methods have a regularisation term in their cost function which prevents the regressor from being affected by noise and overfitting.

Application of the regression techniques to Franke's function showed that the regularised methods perform better than ordinary least squares when few data points with much noise is approximated, while a large number of data points reduces overfitting and makes ordinary least squares on par with Ridge and LASSO regression. Newton's method proved ineffective at minimising the LASSO cost function due to its non-differentiability, while the implementations of ordinary least squares, Ridge regression and bootstrapping performed well in terms of both accuracy and performance, the latter being crucial for large data sets with polynomials of a high degree.

Lastly, geographical data from a Norwegian fjord was fitted. Ordinary least squares proved to be utterly useless, while Ridge regression performed well for a variety of polynomial degrees when the optimal regularisation parameter was chosen for each polynomial degree. The main geographical features were reproduced, despite difficult features such as a Heaviside-like incline on the side of the fjord. With as many as 65 000 data points, bootstrapping and selection of λ, the poor time complexity of the regression algorithms meant that an overfitting polynomial was impossible to achieve before the deadline.

Future work therefore includes using a prediction method which scales better with data, such as neural networks. Additionally, a better version of the LASSO technique can be implemented by using a minimisation algorithm such as coordinate descent. A sparse solution with some β_js exactly equal to zero is then expected due to the use of a soft thresholding operator.

References

[1] Jerome Friedman, Trevor Hastie and Rob Tibshirani. "Regularization paths for generalized linear models via coordinate descent". In: *Journal of statistical software* 33.1 (2010), p. 1.

[2] Pankaj Mehta et al. "A high-bias, low-variance introduction to machine learning for physicists". In: *arXiv preprint arXiv:1803.08823* (2018).

[3] Wessel N van Wieringen. "Lecture notes on ridge regression". In: *arXiv preprint arXiv:1509.09169* (2015).