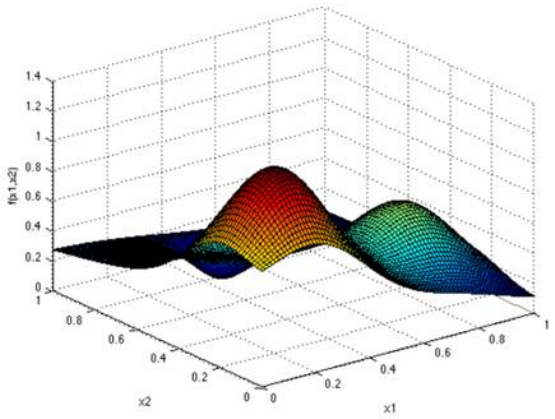

Figure 1: Shape of Franke's function which will be used as a interpolating goal. Note to self: Might be better to plot the graph instead of using a figure from the web. Remember to add the source for the figure by using for example bib. https://www.sfu.ca/ ssurjano/franke2d.html

1 Theory

1.1 Franke's function

Franke's function is a function which is often used to test different regression and interpolation methods. The function has two Gaussian peaks of differing heights, and a smaller dip. It's expression is

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

and we define it for the interval $x, y \in [0,1]$. An illustration of Franke's function can be seen in figure 1

1.1.1 The Vandermonde matrix applied to Franke's function

When performing the different regression methods, it is important to sort the data into a well designed system for easy access and easier calculations. The data which is to be used during the calculations is sorted into a so called Vandermonde matrix, which is also known as a design matrix. A Vandermonde matrix is a matrix where the rows is built up by geometric progression

Franke's function is a two dimensional function defined by the components x and y. Then it is instinctive to have a design matrix X also built up by x and

y components into a np matrix, the following way

$$\texttt{X=}\begin{pmatrix} 1 & x_0 & y_0 & x_0^2 & x_0y_0 & y_0^2 & \dots & x_0^d y_0^{p-d} \\ 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 & \dots & x_1^d y_1^{p-d} \\ & & & & & & \vdots & \\ 1 & x_{n-1} & y_{n-1} & x_{n-1}^2 & x_{n-1}y_{n-1} & y_{n-1}^2 & \dots & x_{n-1}^d y_{n-1}^{p-d} \end{pmatrix}$$

Where the d is the degree of polynomial. Making the degree higher leads to more complicated equations which naturally gives a more precise regression. Having a design matrix built up by too many terms per polynomial increases the possibility of eventually getting an over fitting model, which will be discussed later in the article.

By having different weights in front of each term in the design matrix, it is possible to calculate how large each term in the polynomials should be in order to make the best approximation. This is the clue of having a design matrix built up by x and y components, making it possible to approximate the polynomials which will fit the Franke's function best. Then the approximation can be written as the following

$$\tilde{y} = X\beta + \epsilon \tag{1}$$

Where ϵ is the error. This can be written as

$$\texttt{\textbackslash tilde\{y\}=}\begin{pmatrix} \epsilon_0 & 1 & \beta_0 x_0 & \beta_1 y_0 & \beta_2 x_0^2 & \beta_3 x_0 y_0 & \beta_4 y_0^2 & \dots & \beta_5 x_0^d y_0^{p-d} \\ \epsilon_1 & 1 & \beta_0 x_1 & \beta_1 y_1 & \beta_2 x_1^2 & \beta_3 x_1 y_1 & \beta_4 y_1^2 & \dots & \beta_5 x_1^d y_1^{p-d} \\ & & & & & & \vdots & & \\ \epsilon_{n-1} & 1 & \beta_0 x_{n-1} & \beta_1 y_{n-1} & \beta_2 x_{n-1}^2 & \beta_3 x_{n-1} y_{n-1} & \beta_4 y_{n-1}^2 & \dots & \beta_5 x_{n-1}^d y_{n-1}^{p-d} \end{pmatrix}$$

1.2 Linear regression

The goal of a linear regression model is to find the coefficients $\hat{\beta}$ best suited for predicting new data via this expression:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta},$$

where $\mathbf{X}$ is the design matrix constructed from the input data, and $\hat{\mathbf{y}}$ is the resulting prediction. To achieve this, we define a "cost function" of sorts, that evaluates each coefficients ability to predict the initial training dataset. We then find the $\hat{\beta}$ that minimizes this cost function. More formally put, we want to find

$$\hat{\beta} = \arg\min_{\beta} C(\beta), \tag{2}$$

where $C(\beta)$ is the cost function.

1.2.1 Ordinary least squares regression (L_0)

Ordinary least squares (OLS) regression uses the residual sum of squares (RSS) function as the cost function. Given N datapoints and the predicted output $\mathbf{y}$, it reads

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2. \tag{3}$$

As found in appendix **??**, a cost function like (3) gives the following matrix equation for $\hat{\beta}$

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{4}$$

OLS regression has very low variance, but high bias - as a consequence of the bias-variance tradeoff. This makes OLS regression an "accurate" predictor of its own training data, but susceptible to overfitting, which the following two models are better suited to handle.

1.2.2 Lasso regression (L_1)

Lasso regression expands upon the above cost function by adding a term that penalizes the size of each coefficient. This is done by a factor of λ, as shown here:

$$\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}. \tag{5}$$

The first sum is from the RSS function, while the second sum is the imposed penalty. The Lasso decreases bias from the OLS model, by decreasing the size of the coefficients, and thus making the variables more equally weighted.

Lasso regression has no closed form expression for $\hat{\beta}$, which means it must be calculated programatically.

1.2.3 Ridge regression (L_2)

Ridge regression has a penalty corresponding to the coefficient's squared sizes, further decreasing the bias from The Lasso, but obviously also increases variance. It defines $\hat{\beta}$ like this

$$\hat{\beta}^{\mathrm{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\},$$

which - by the same proceedure as shown in appenix **??**, has this closed form expression

$$\hat{\beta}^{\mathrm{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}. \tag{6}$$

1.3 Assessment

1.3.1 K-fold cross-validation

When faced with the issue of a small dataset, a good solution to circumvent issues is to utilize whats called K-fold cross-validation. It involves splitting the dataset in to K equally sized "folds", and using each fold sequentially as the validation set, whilst training on the others. Say, for example, that we set $K = 6$. In this example we would first train our model on the last 5 datasets and evaluate the skill of our model by trying to predict the first. The evaluation is done by calculating the R^2 score, MSE or similar. Thereafter, we use the second fold as the testing set, train on the others, and so forth. The final cross-validation value is the mean of all the calculated statistics.
This is especially a great tool when trying to find the best parameter α present in a model to minimize a certain error/statistic. Cross-validation will then serve as a function we want to minimize to achieve the best result, namely

$$\mathrm{CV}(\hat{f}, \alpha) = \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, \hat{f}^{-\kappa(i)}(x_i, \alpha)\right) \tag{7}$$

where L is the error function we want to optimize, y_i are the true values, $\hat{f}^{-k}$ is the fitted function excluding the fold k and $\kappa(i)$ is the function taking in the index of the original dataset and returning the fold containing it. Blabla