FYS-STK 3155 Project 1 Erlend Abrahamsen

October 7, 2019

Abstract

On noise generated franke data we find that Ridge regression with parameter $\lambda = 10^{-3}$ and polynomial degree $n = 4$ gives the best fit. On our real terrain data we got a plausible best result at $(n, \alpha) = (6, 10^{-2})$ with Ridge.

1 Introduction

For the source code (folder Source_code) and the results (folder Results) go to my Github: `https://github.com/ErlendAbrahamsen/FYS-STK3155/tree/master/Project1`

In this project we will study various regression methods such as ordinary least squares (OLS), Ridge regression and Lasso regression. To score our models we will use K-Folds Cross-validation which is a well known resampling technique in machine learning. We will also study the MSE (mean square error) in depth analytically. We start off with data from the Franke function and later move on to real world data taken from near Stavanger in Norway.

Franke function:
$f(x,y) = 0.75\exp(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}) + 0.75\exp(-\frac{(9x+1)^2}{49} - \frac{9y+1}{10})$

$+0.5\exp(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}) - 0.2\exp(-(9x-4)^2 - (9y-7)^2).\ x, y \in [0,1]$

Outline [Introduction, Methods, Results and discussion, Conclusion]
I will reference to source code/project part where relevant!

2 Methods

2.0.1 Regression intro/notation

With k measured datapoints $\mathbf{z} = (z_0, ..., z_{k-1})$ we can find a best fit continuous function $p(\mathbf{x})$ with respect to a cost function.
Different regression methods calls for different cost functions with different merits.
In this project we want to fit polynomials up to 5'th degree or higher.
I.e. find coefficients for $p : R^2 \to R$ with basis $\{1, x, y, x^2, xy, y^2, ..., y^5\}$ (5'th degree case, $x^n y^m$ with $n + m \leq 5$)

With respect to a cost function $C(X, \beta)$, we find the closest solution to $X\beta = \mathbf{z}$ and get the prediction $\tilde{\mathbf{z}} = X\beta$.
$\beta = (\beta_0, ..., \beta_{m-1})^T \in R^m$ is our coefficients.
X is the design matrix (n'th degree $p(\mathbf{x})$ case):

$$X = \begin{bmatrix} 1 & x_0 & y_0 & x_0^2 & x_0 y_0 & y_0^2 & \cdots & x_0^n & x_0^{n-1} y_0 & \cdots & y_0^n \\ 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 & \cdots & x_1^n & x_1^{n-1} y_1 & \cdots & y_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{k-1} & y_{k-1} & x_{k-1}^2 & x_{k-1} y_{k-1} & y_{k-1}^2 & \cdots & x_{k-1}^n & x_{k-1}^{n-1} y_{k-1} & \cdots & y_{k-1}^n \end{bmatrix}$$

(See Designmatrix method in (Part_a.py) for implementation.)

2.1 OLS regression

We start by minimizing MSE or the cost $C(X, \beta) = \frac{1}{k} \sum_{i=0}^{k-1} (z_i - \tilde{z}_i)^2 = \frac{1}{k} (\mathbf{z} - X\beta)^T (\mathbf{z} - X\beta)$

Setting partial derivatives to zero:

$\frac{\partial C}{\partial \beta} = \frac{1}{k} (2X^T X\beta - 2X^T \mathbf{z}) = 0$

So $X^T X\beta = X^T \mathbf{z}$ and we finally solve for β and get

$\beta = (X^T X)^{-1} X^T \mathbf{z}$, given that $X^T X$ is invertible.
(See OLS method in (Part_a.py) for implementation.)

2.2 MSE analysis

Assume that the true data is generated from a noisy model $z_i = f(\mathbf{x}) + \epsilon_i$, where ϵ has mean 0 and standard deviation σ^2.
Rewriting our MSE (also OLS cost):

$C(X, \beta) = E[(\mathbf{z} - \tilde{\mathbf{z}})^2] = \frac{1}{k}\sum_{i=0}^{k-1}(z_i - \tilde{z}_i)^2$

$= E\Big[\Big((f + \epsilon - \tilde{\mathbf{z}}) + (E[\tilde{\mathbf{z}}] - E[\tilde{\mathbf{z}}])\Big)^2\Big]$ (adding 0 "trick")

$= E\Big[(f - E[\tilde{\mathbf{z}}])^2 + (-\tilde{\mathbf{z}} + E[\tilde{\mathbf{z}}])^2 + \epsilon^2$
$+ 2(f - E[\tilde{\mathbf{z}}])(-\tilde{\mathbf{z}} + E[\tilde{\mathbf{z}}]) + 2\epsilon(f - E[\tilde{\mathbf{z}}])) + 2\epsilon(-\tilde{\mathbf{z}} + E[\tilde{\mathbf{z}}])\Big]$

$= E[(f - E[\tilde{\mathbf{z}}])^2] + E[(\tilde{\mathbf{z}} - E[\tilde{\mathbf{z}}])^2] + E[\epsilon^2] + 2E[(f - E[\tilde{\mathbf{z}}])(-\tilde{\mathbf{z}} + E[\tilde{\mathbf{z}}])] +$
$2E[\epsilon]E[(f - E[\tilde{\mathbf{z}}])] + 2E[\epsilon]E[(-\tilde{\mathbf{z}} + E[\tilde{\mathbf{z}}])]$ **

$= E[(f - E[\tilde{\mathbf{z}}])^2] + E[(\tilde{\mathbf{z}} - E[\tilde{\mathbf{z}}])^2] + E[\epsilon^2]$

$= \frac{1}{k}\sum_{i=0}^{k-1}(f_i - E[\tilde{\mathbf{z}}])^2 + \frac{1}{k}\sum_{i=0}^{k-1}(\tilde{z}_i - E[\tilde{\mathbf{z}}])^2 + \sigma^2$

Note that $E[\epsilon] = 0$ and
$2E[(f - E[\tilde{\mathbf{z}}])(-\tilde{\mathbf{z}} + E[\tilde{\mathbf{z}}])] = 2E[(f - E[\tilde{\mathbf{z}}])](-E[\tilde{\mathbf{z}}] + E[\tilde{\mathbf{z}}]) = 0$
so the three last terms in ** are zero.

So the expected square error between collected data $\mathbf{z}$ and predicted value $\tilde{\mathbf{z}}$ consists of three error terms.

(Chronological order)
Term 1: This is MSE between the true values f_i and the expected prediction $E[\tilde{\mathbf{z}}]$, called the bias.
Term 2: This is MSE between predictions $\tilde{z}_i$ and expected prediction $E[\tilde{\mathbf{z}}]$. I.e prediction variance.
Term 3: This is simply collected data variance, which is $\sigma^2 = 1$ (Noise is $\mathcal{N}(0, 1)$).

The error dilemma we get is that we want to minimize all the errors and variances.
Usually as the model complexity increases the bias reduces but the prediction variance increases. So we have to decide how much bias we want to trade off for variance. High bias and low variance leads to an underfitted model, while low bias and high variance leads to an overfitted model. It is essential in machine learning to balance underfitting vs overfitting.

2.3 Ridge regression

Cost function: $C(X, \beta) = ||\mathbf{z} - X\beta||_2^2 + \lambda||\beta||_2^2$, for some $\lambda \in R^+$.
Here the first term is the same as in an OLS regression, but then we add an penalty term for overfitting prevention.
Lin. alg. solution used in method Ridge (Part_e);

$\frac{\partial}{\partial\beta}[(\mathbf{z} - X\beta)^T(\mathbf{z} - X\beta) + \lambda\beta^T\beta] = 0$

$2X^TX\beta - 2X^T\mathbf{z} + 2\lambda\mathrm{I}\beta = 0$

$(X^TX + \lambda\mathrm{I})\beta = X^T\mathbf{z}$

$\beta = (X^TX + \lambda\mathrm{I})^{-1}X^T\mathbf{z}$, assuming $(X^TX + \lambda\mathrm{I})$ is invertible.

Further for finding most optimal β we will use KFolds-crossvalidation for several λ values. (See section 2.0.6)

2.4 Lasso regression

Cost function: $C(X, \beta) = ||\mathbf{z} - X\beta||_2^2 + \alpha||\beta||_1$, for some $\alpha \in R^+$
Again add an extra penalty for defeating OLS overfitting. This is a more simple model than Ridge as we tend to set irrelevant β terms to zero, i.e. regularization. We will use KFolds-crossvalidation (section 2.0.6) on several α values to determine optimal β.

2.5 KFolds-Crossvalidation

For evaluating how good our model is on predicting new values we can split the data into a training set and a testing set.

Now we can train the model on the training set and evaluate it on the outside testing set.
We want to have a lot of data for training so it's common to use 60-90% of the data as training set.
Measuring accuracy with only one set could be misleading as the sample might be to small, so it's standard to use crossvalidation techniques.

KFolds-Crossvalidation general Algorithm;
For simplicity let's use $k = 5$ folds (sets) which is also common.
The idea is to shuffle all the data randomly and then divide the data into 5 folds of equal size.
Then we set fold 1 as testing data and folds 2 trough 5 as testing data. (Group 1)
We repeat this process, set fold 2 as testing data and folds $\{1, 3, 4, 5\}$ as training data (Group 2), etc. We are finished when testing data has cycled trough the last fold.

Now we can get a good accuracy measure of our model by e.g. computing average MSE over all 5 groups.

The implementation of KFoldCross method in part_b.py is explained in the first doc-string and comments.
We basically use np.random.shuffle([,indices]) as randomizer and append all k groups into lists for each x, y, z, f variable.

Two groups for $\mathbf{x}$ as example (Each fold has length p)
Group 1 is $\mathbf{x}$_test = $\mathbf{x}$[0:p] (fold 1). $\mathbf{x}$_train = $\mathbf{x}$[p:]. (fold 2-5)
Group 2 is $\mathbf{x}$_test = $\mathbf{x}$[p:2p] (fold2). $\mathbf{x}$_train = [$\mathbf{x}$[:p], $\mathbf{x}$[2p:]] (fold 1,3,4,5) etc.

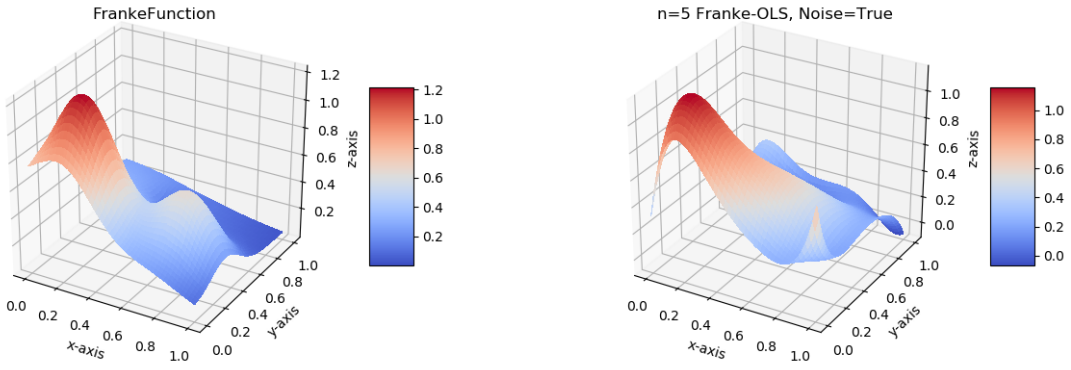
3 Results and discussion

3.1 OLS on Noised Franke

3.1.1 Part a

With a [50**x**50] grid of points in $[0, 1]$ we evalute f (Franke) and our generated data measures $\mathbf{z} = f + \mathcal{N}(0, 1)$.
We do OLS on $\mathbf{z}$ and see how well it reproduces the real Franke function.

Plots:



We can see that the red left top does a good job and the right white/blue top does a bad one.

Confidenceinterval for β when $n = 5$:

β 95% CI
0 (-0.88, 1.53)
1 (7.74, 10.16)
2 (2.58, 4.99)
3 (-44.89, -42.47)
4 (-13.06, -10.64))
5 (-9.55, -7.14)
6 (83.7, 86.12)
7 (32.52, 34.93)

8 (3.2, 5.61)
9 (-3.83, -1.42)
10 (-79.97, -77.56)
11 (-32.26, -29.84)
12 (-2.2, 0.22)
13 (-5.77, -3.36)
14 (13.23, 15.65)
15 (27.79, 30.21)
16 (-0.85, 1.57)
17 (21.53, 23.94)
18 (-23.0, -20.59)
19 (7.25, 9.67)
20 (-8.71, -6.29)

All intervals have a length or spread ≤ 2.5. The best possible polynom at $n = 5$ is the 5'th degree Taylor expansion of f. We would then expect $\geq 95\%$ of the Taylor coefficients to fall within our CI.

MSE and R2 score for $n = 1, ..., 5$:

n=1: MSE=0.02508 , R2=0.6947
n=2: MSE=0.01924 , R2=0.7658
n=3: MSE=0.01029 , R2=0.8747
n=4: MSE=0.009581, R2=0.8834
n=5: MSE=0.01083 , R2=0.8682

From the MSE and $R2$ score we notice that OLS performs best at $n = 4$.

3.1.2 Part b

Now we resample the data with KFoldCross method.

We calculate average MSE and R2 for $k = 3$ and $k = 5$ for $1 \leq n \leq 10$.
($k = 3$ and $k = 5$ sets 67% and 80% of the data as training sets)

(n,k)=(1,3): avg_MSE=0.02543, avg_R2=0.6898
(n,k)=(2,3): avg_MSE=0.02035, avg_R2=0.7514
(n,k)=(3,3): avg_MSE=0.01201, avg_R2=0.8532
(n,k)=(4,3): avg_MSE=0.01183, avg_R2=0.8554
(n,k)=(5,3): avg_MSE=0.01526, avg_R2=0.8132
(n,k)=(6,3): avg_MSE=0.01682, avg_R2=0.7935
(n,k)=(7,3): avg_MSE=0.01986, avg_R2=0.7562
(n,k)=(8,3): avg_MSE=0.02465, avg_R2=0.697
(n,k)=(9,3): avg_MSE=0.03092, avg_R2=0.6197
(n,k)=(10,3): avg_MSE=0.03966, avg_R2=0.5132

(n,k)=(1,5): avg_MSE=0.02548, avg_R2=0.6887
(n,k)=(2,5): avg_MSE=0.02002, avg_R2=0.7553
(n,k)=(3,5): avg_MSE=0.01105, avg_R2=0.865
(n,k)=(4,5): avg_MSE=0.01095, avg_R2=0.8661
(n,k)=(5,5): avg_MSE=0.01305, avg_R2=0.8405
(n,k)=(6,5): avg_MSE=0.01483, avg_R2=0.8184
(n,k)=(7,5): avg_MSE=0.01748, avg_R2=0.7856
(n,k)=(8,5): avg_MSE=0.02101, avg_R2=0.7423
(n,k)=(9,5): avg_MSE=0.02666, avg_R2=0.6722
(n,k)=(10,5): avg_MSE=0.03054, avg_R2=0.6256

Notice that the R2 decreases for higher values of n. This is because of overfitting, we will get more into this in Part c section.

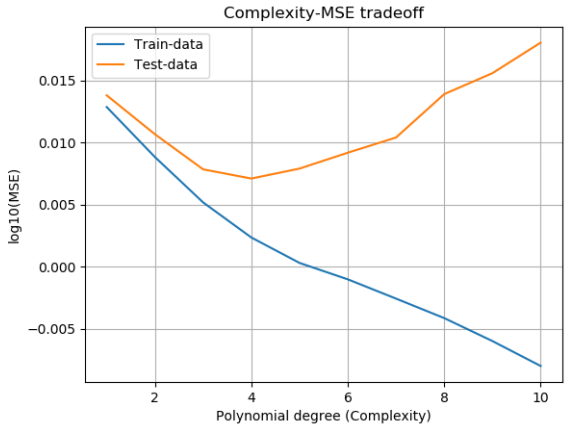
We again see that $n = 4$ performs the best.
Note that R2-scores without resampling (Part a) are a bit higher than both $k = 3$ and $k = 5$.
(n,k)=(4,3) gives top R2 score of 0.8554 and (n,k)=(4,5) gives top score 0.8661 (also lowest MSE) which is about 1.1% difference. To get even more trusted results, one could do Kfold crossvalidation e.g. 100 times and again average these results.
This can be time consuming and don't seem necessary in our case as we get consistent R2 peak for $n = 4$ here and even in part a.

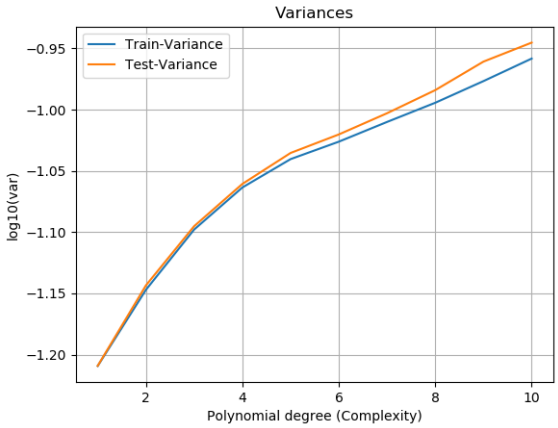
3.1.3 Part c

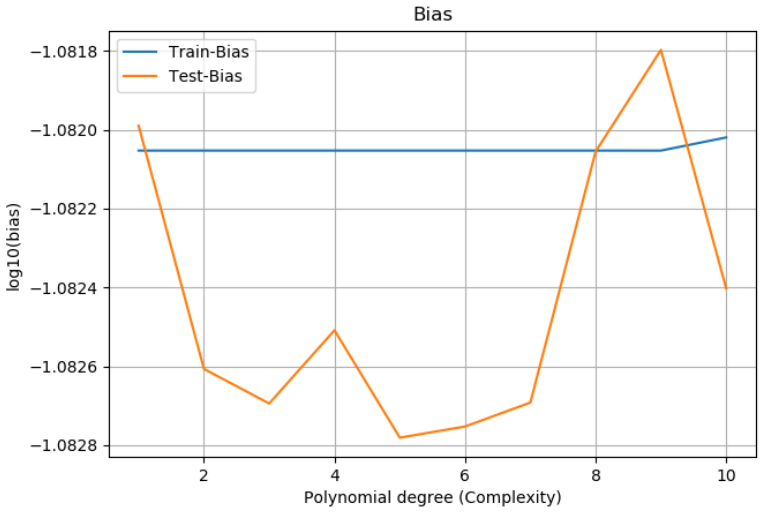
In the MSE analysis we found that;

$E[(\mathbf{z}-\tilde{\mathbf{z}})^2] = \frac{1}{k}\sum_{i=0}^{k-1}(f_i - E[\tilde{\mathbf{z}}])^2 + \frac{1}{k}\sum_{i=0}^{k-1}(\tilde{z}_i - E[\tilde{\mathbf{z}}])^2 + \sigma^2$
We do predictions on both training sets and testing sets for various n values and plot the resulting errors.

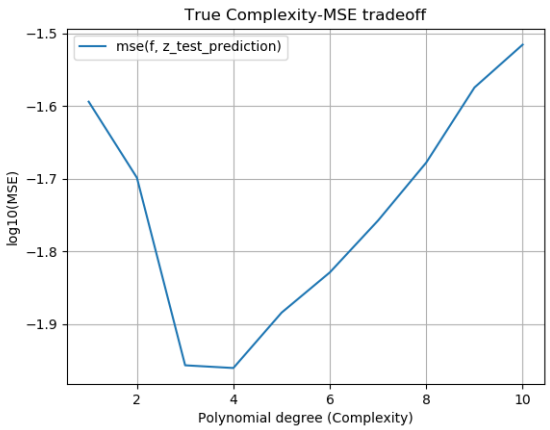
MSE, bias and variance as function of polynomial degree:







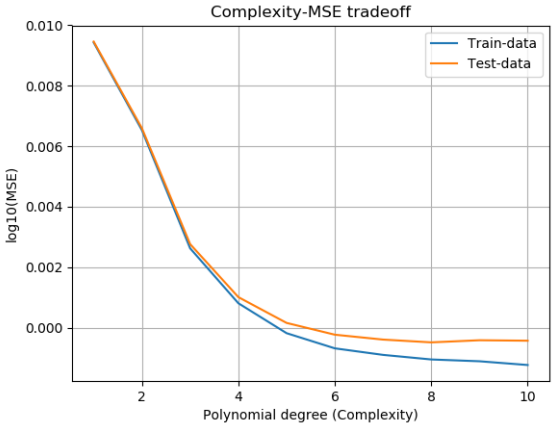
From the MSE-tradeoff plot we see that as n increases the model gets better at prediction on the training data. We also find the best outside test data prediction with $E[MSE(\mathbf{z}, \tilde{\mathbf{z}})] = 1.01649$ at $n = 4$. ($n = 4$ Test-data graph)
The real best MSE $min[MSE(f, \tilde{\mathbf{z}})]$ should be close to minimized at $n = 4$. Plotting $E[MSE(f, \tilde{\mathbf{z}})]$:

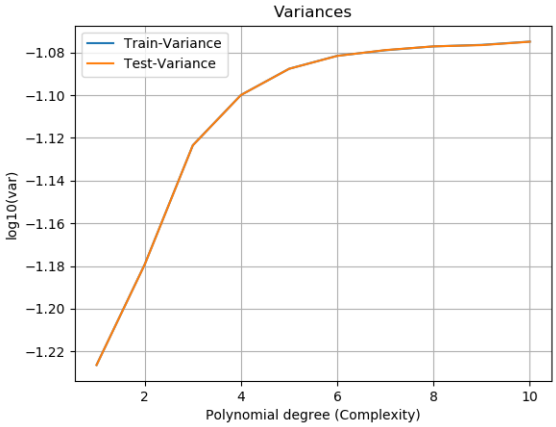


We find that $n = 4$ gives the best result with $E[MSE(f, \tilde{\mathbf{z}})] = 0.01095$

In the n end ranges we see big error on the testing set prediction, i.e overfitting.
The model starts to fit the noise in the training set and will make bad predictions on data outside the training sets.

We see some big spikes in the test-bias graph, we expect it to decrease in general. Otherwise the MSE and variance graph seems reasonable. Lets try to increase number of data points from 50x50 to 300x300 and see what happens:



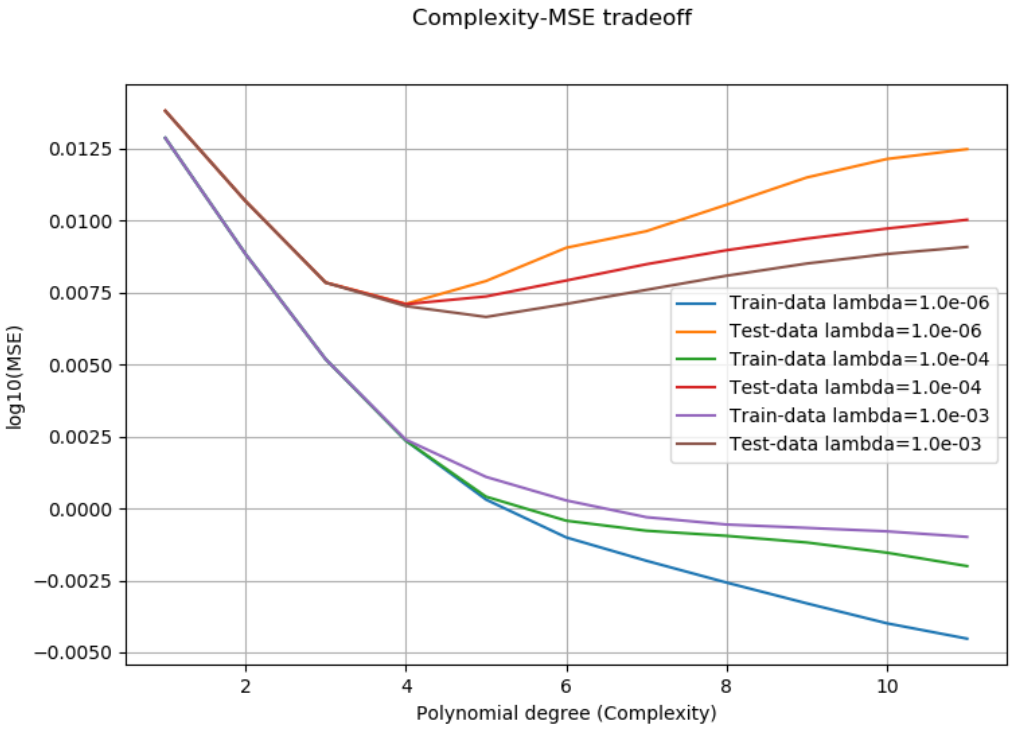


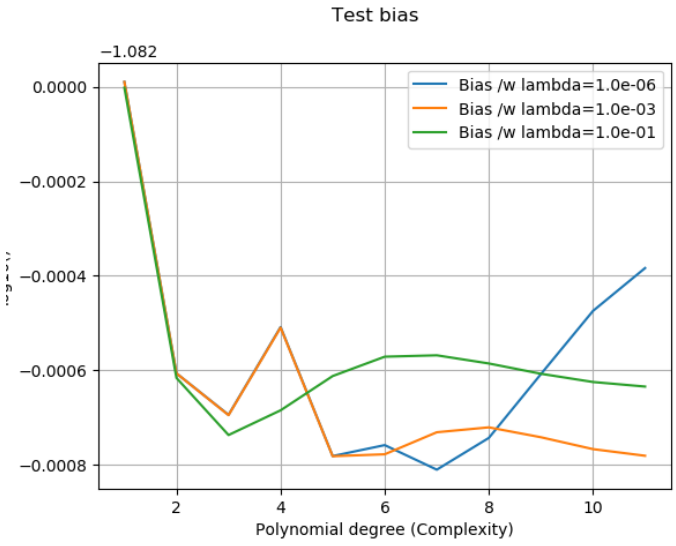
Now the Test-Bias seems to be smallest for the bigger n values which is what we are expecting. From the new MSE Test-data we see that the best performance is shifted towards bigger n values. Looks like higher model complexity is better for bigger data samples in this case.

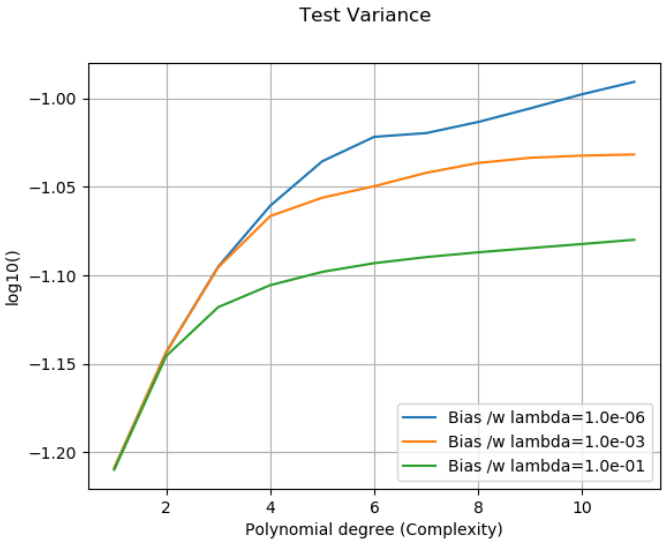
3.2 Ridge on Noised Franke

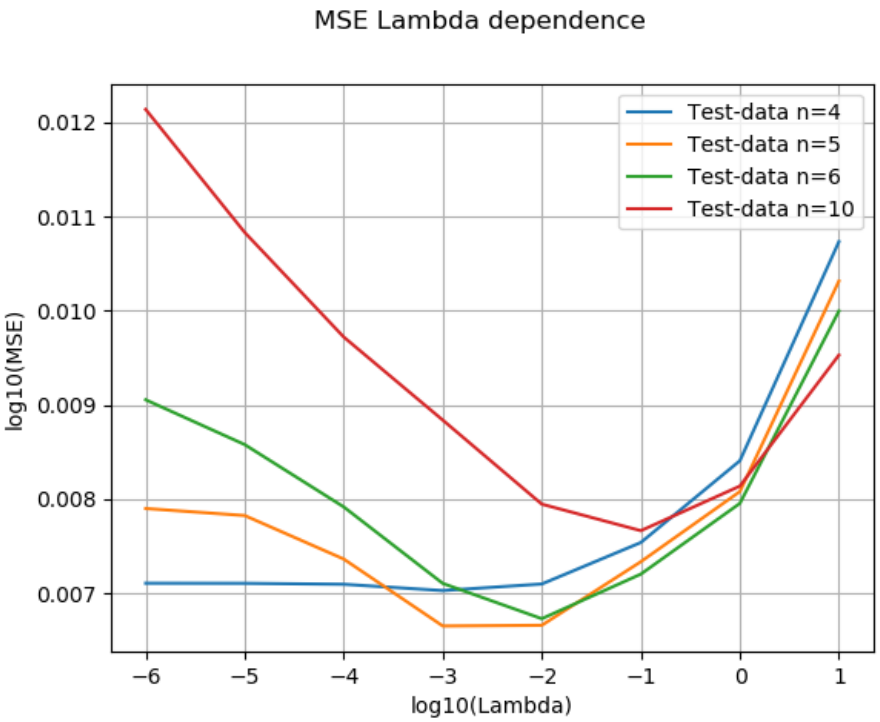
3.2.1 Part d

With $n = \{1, ..., 10\}$ and $\lambda = \{10^{-6}, ..., 10^{1}\}$ we use our own Ridge method and crossvalidate scoring metrics over a [len(n) **x** len(λ)] grid.


Complexity-MSE tradeoff
log10(MSE)
Polynomial degree (Complexity)
Train-data lambda=1.0e-06
Test-data lambda=1.0e-06
Train-data lambda=1.0e-04
Test-data lambda=1.0e-04
Train-data lambda=1.0e-03
Test-data lambda=1.0e-03

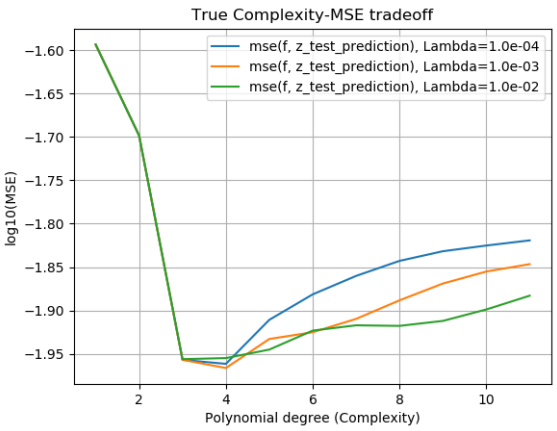

Test bias
−1.082
Bias /w lambda=1.0e-06
Bias /w lambda=1.0e-03
Bias /w lambda=1.0e-01
Polynomial degree (Complexity)


Test Variance
log10()
Bias /w lambda=1.0e-06
Bias /w lambda=1.0e-03
Bias /w lambda=1.0e-01
Polynomial degree (Complexity)



From the bottom figure we see that $(n, \lambda) = (5, 10^{-3})$ with $E[MSE(\mathbf{z}, \tilde{\mathbf{z}})] = 1.01543$ should be close to best real result.

Plotting the true mse between f and test predictions:



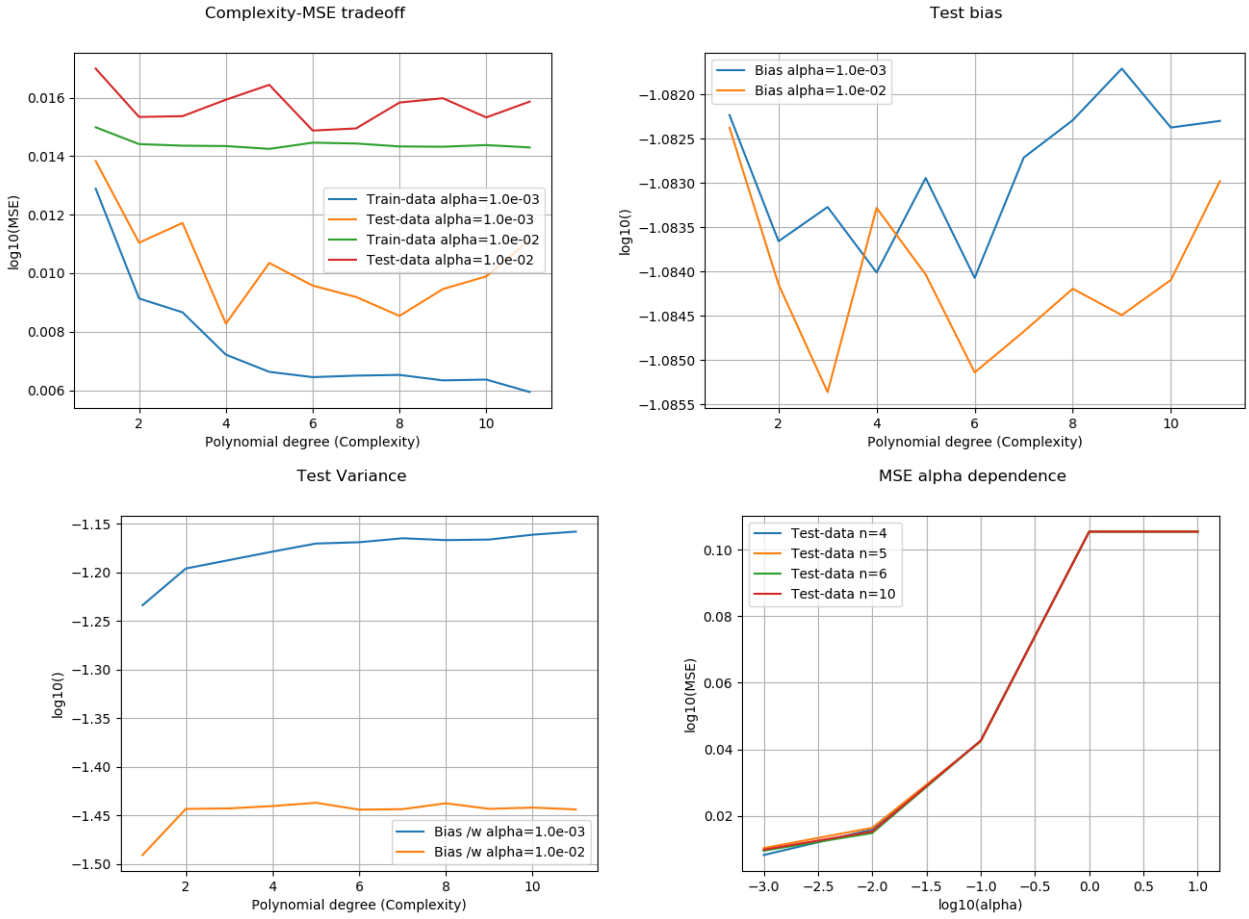
Since we have access to the true relationship f we find the best results with $(n, \lambda) = (4, 10^{-3})$ giving $E[MSE(f, \tilde{\mathbf{z}})] = 0.01081$.

3.3 Lasso on Noised Franke

3.3.1 Part e

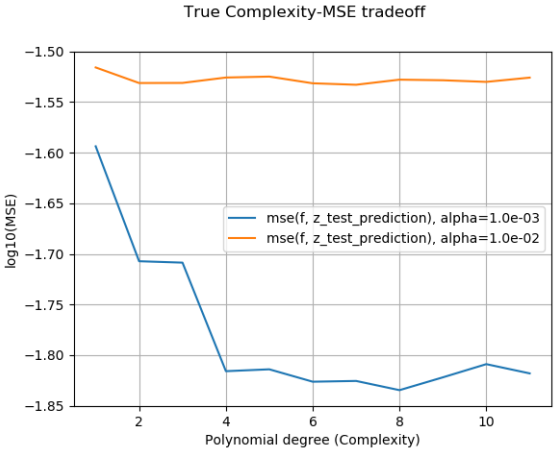
We essentially repeat Part d, but use a smaller grid because Lasso method from sklearn seems to run pretty slow for very small α values.

We crossvalidate over the grid [len(n) **x** len(α)], with $n = \{1, ..., 10\}$ and $\alpha = \{10^{-3}, ..., 10^{1}\}$



From MSE alpha dependence we see that $(n, \alpha) = (4, 10^{-3})$ with $E[MSE(\mathbf{z}, \tilde{\mathbf{z}})] = 1.01924$ should be close to the true best results.

Plotting true result ($E[MSE(f, \tilde{\mathbf{z}})]$):



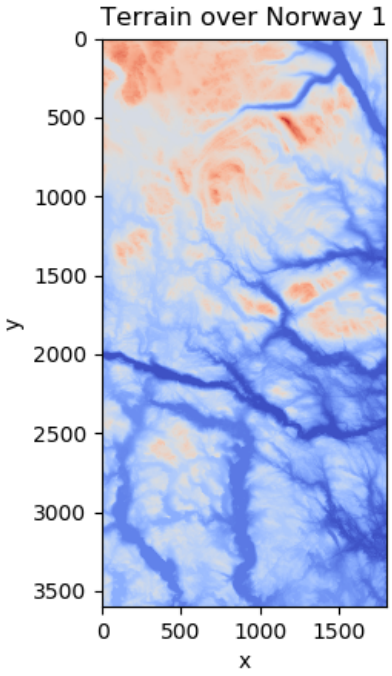
We find that $(n, \alpha) = (8, 10^{-3})$ with $E[MSE(f, \tilde{\mathbf{z}})] = 0.01464$ is the most optimal for this model.

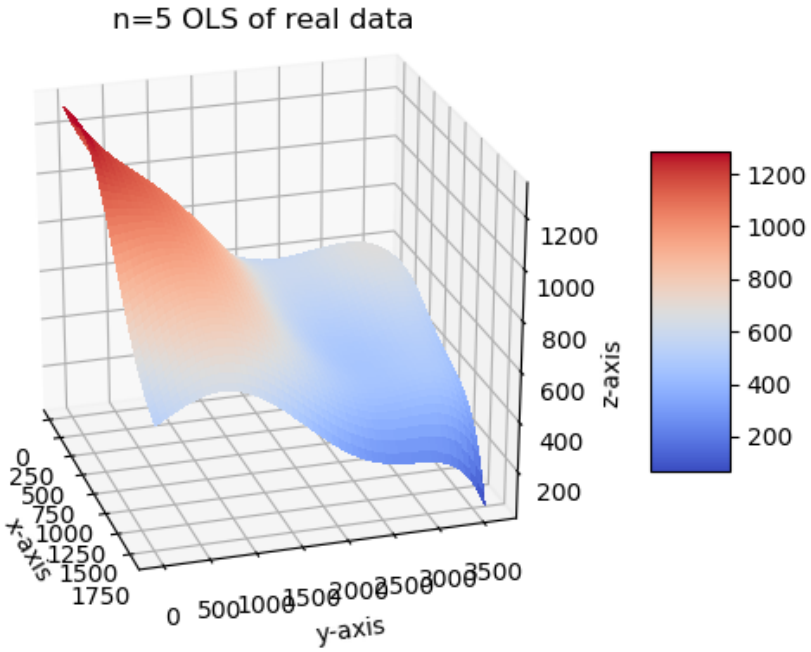
3.4 OLS/Ridge/Lasso on Real data

3.4.1 Part f: Loading our Data

For our real data analysis we use the GeoTIF file SRTM_data_Norway_1.tif, which is from a region close to Stavanger in Norway. (See 3. in section References).

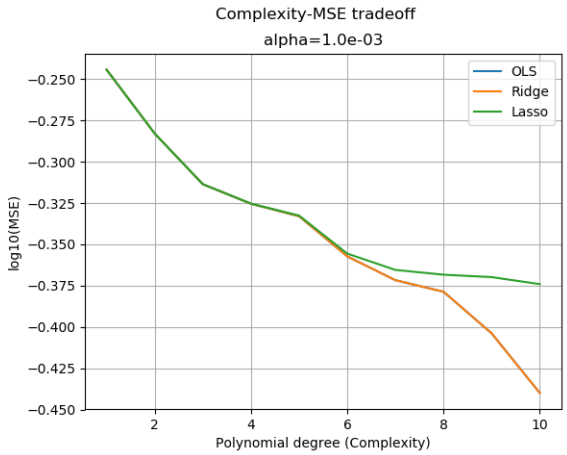
imshow plot and OLS surface plot of region (See Part.py):

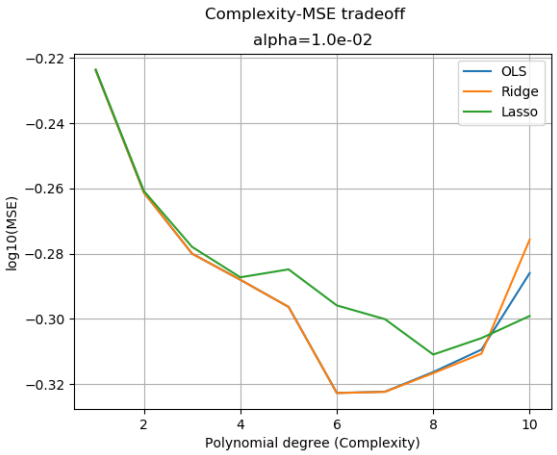


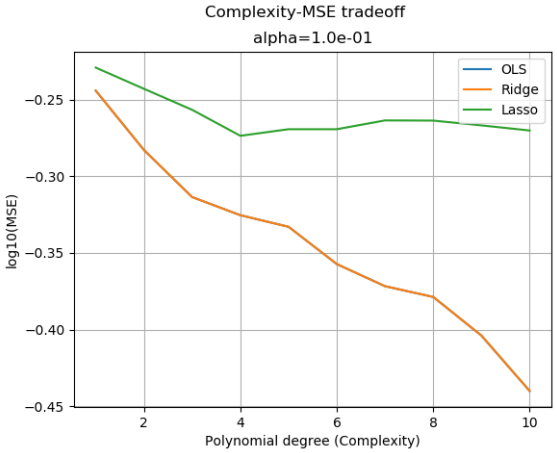


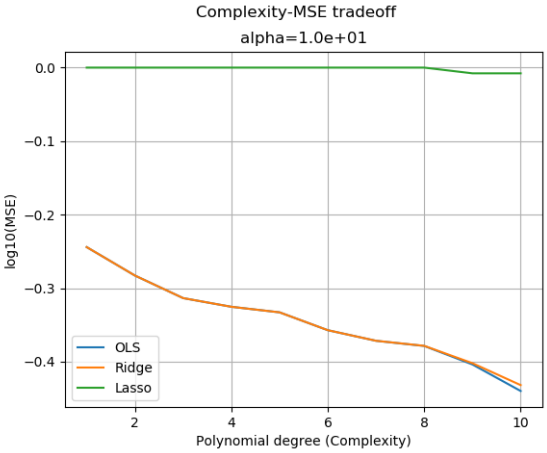
3.4.2 Part g

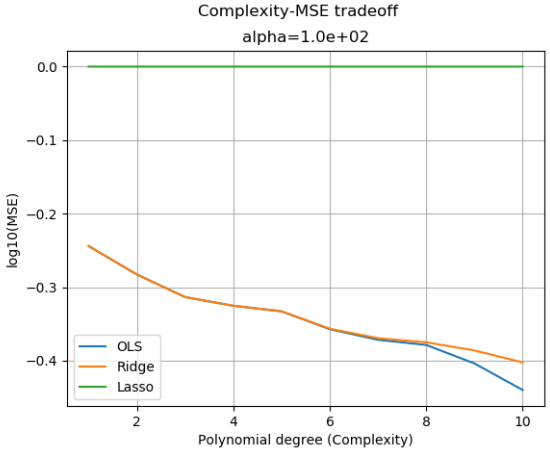
Since the real data set $\mathbf{z}$ is very big (approx $6.5 \cdot 10^6$), we decide to use a subset of 10^5 random points, which is only 1.5%. Else the computations will take to long. We also scale our real data to get better numerical stability. Now we try OLS, Ridge and Lasso over $n = \{1, 2, .., 10\}$ with $\alpha = \{10^{-3}, 10^{-2}, 10^{-1}, 10^1, 10^2\}$.
MSE plots:


Complexity-MSE tradeoff
alpha=1.0e-03
log10(MSE)
Polynomial degree (Complexity)
OLS
Ridge
Lasso

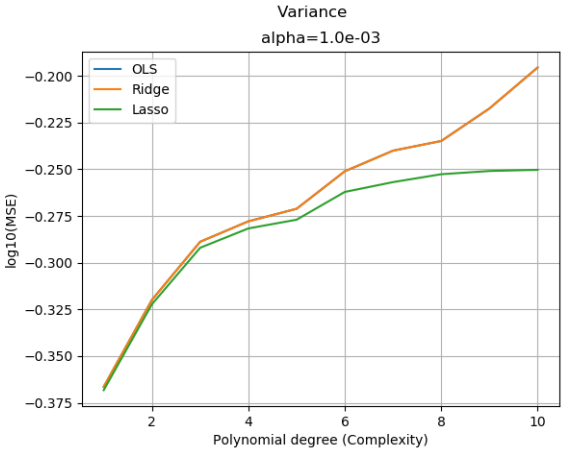

Complexity-MSE tradeoff
alpha=1.0e-02
log10(MSE)
Polynomial degree (Complexity)
OLS
Ridge
Lasso

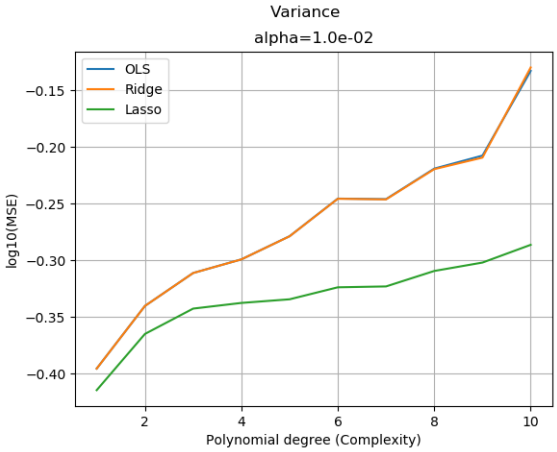

Complexity-MSE tradeoff
alpha=1.0e-01
log10(MSE)
Polynomial degree (Complexity)
OLS
Ridge
Lasso

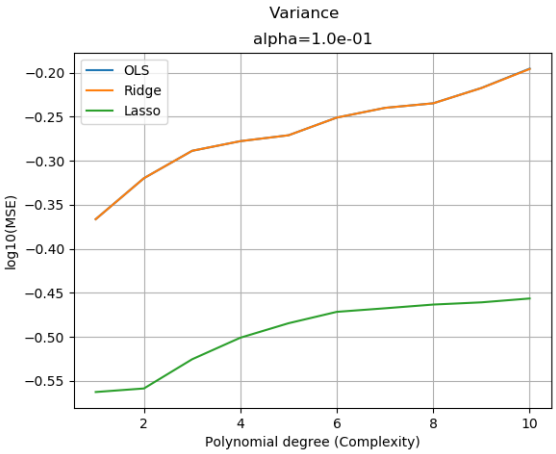

Complexity-MSE tradeoff
alpha=1.0e+01
log10(MSE)
Polynomial degree (Complexity)
OLS
Ridge
Lasso

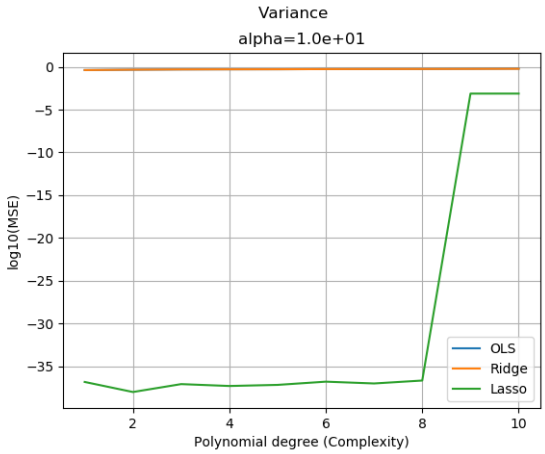

Complexity-MSE tradeoff
alpha=1.0e+02
log10(MSE)
Polynomial degree (Complexity)
OLS
Ridge
Lasso

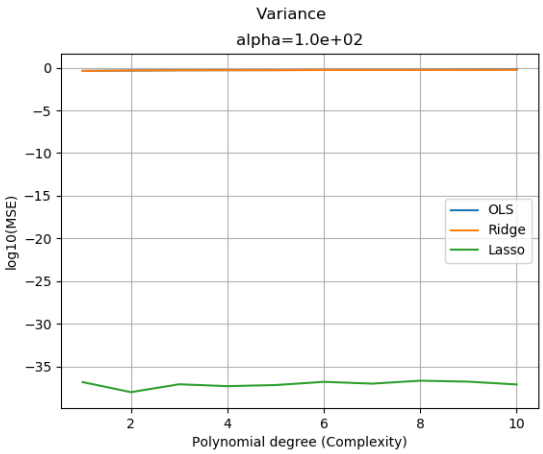
variance plots:











From our plots it looks like the best model we get is with Ridge with the parameters $(n, \alpha) = (6, 10^{-2})$ giving an $MSE = 0.48$ (scaled).

Disclaimer: I struggled a bit with convergence for $alpha = 10^{-3}$ and $alpha = 10^{-2}$.

I would actually expect Lasso to do best because of regularization. We are dealing with terrain that has big parts of water at $z = 0$ (height), and also big parts of high mountains.
It makes sense to push many coefficients to 0 because of water, and big coefficients elsewhere for the steep mountains. It seems as the coefficients should be polarized between 0 and big.

3.5 Testing and benchmarks of developed code

(For code see test_unit_tests.py)
I made some tests for various python methods i have written. The test functions are explained in the program itself with docstring and comments. We run the code and pass the tests with the following outputted benchmarks:

(m,m) = (1000,1000) DesignMatrix() benchmark: 7.412732 s.
(m,m) = (300,300) DesignMatrix() benchmark: 0.636992 s.
(m,m) = (300,300) Polynom() benchmark: 0.043882 s.
(m,m) = (300,300) MSE() benchmark VS Sklearn: 0.000998 s (Time diff).
(m,m) = (300,300) R2() benchmark VS Sklearn: 0.000998 s (Time diff).
(m,m) = (300,300) KFoldCross() benchmark: 0.358037 s.

Now we try test_Design_Matrix() with m=10 000. This results in an MemoryError.
Note that MSE, R2 is based on numpy, so time diff should be approx 0.

4 Summary and conclusions

4.0.1 OLS vs Ridge vs Lasso on Franke w/wo Noise

We did not compute results from regression on Franke without Noise. With basic math knowledge we already know from Taylor series computation that OLS will beat both Ridge and Lasso. This makes sense as we will not run into underfitting/overfitting problems. Note that infinite Taylor of Franke converge since $x, y \in [0, 1]$ and e^{-x^2} can be expressed as an convergent infinite Taylor series in our domain.

For regression on the generated noisy Franke data with 50**X**50 grid, we get the best machine learning model with Ridge regression. That is the lowest MSE on outside test data predictions.

The best results from worst to best:
Lasso: $(n, \alpha) = (8, 10^{-3})$ with $E[MSE(f, \tilde{\mathbf{z}})] = 0.01464$
OLS: $n = 4$ with $E[MSE(f, \tilde{\mathbf{z}})] = 0.01095$
Ridge: $(n, \lambda) = (4, 10^{-3})$ with $E[MSE(f, \tilde{\mathbf{z}})] = 0.01081$

4.0.2 OLS vs Ridge vs Lasso on Real data

From Part g we found the best model Ridge with the parameters $(n, \alpha) = (6, 10^{-2})$ giving an $MSE = 0.48$ (scaled).

Disclaimer: There was some convergence warnings for $alpha = 10^{-3}$ and $alpha = 10^{-2}$.

References

1. Chapter 3 and 7 of The Elements of Statistical Learning by Hastie, Tibshirani and Friedman. `https://www.springer.com/gp/book/9780387848570`
2. Bias variance analysis: `https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff`

3. Digital terrain: `https://earthexplorer.usgs.gov`