Testing up-to-date machine learning techniques for the modelling of meridional heat transport in the Northern Hemisphere.

Christina Kappatou,
Department of Geosciences, UiO,
christina.kappatou@geo.uio.no

May 2023

Abstract

Inspired by the work of [Stone, 1978], I performed a polynomial fit analysis on the non-dimensional meridional heat transport equation, using a series of machine learning techniques. After applying plain Ordinary Least Squares (OLS) and Ridge regression, as well as their Bootstrap and Cross-Validation resampling variations, I concluded that the polynomial degree should be at least 6, in order to achieve a satisfactory bias-variance trade-off. Moving into choosing the optimal polynomial coefficients, I used plain Gradient Descent (GD) and Stochastic Gradient Descent (SGD), both with and without momentum, on OLS and Ridge regression. The learning rate was tuned by varying its constant values and exponentially but I also tested the effects of Adaptive Gradient Algorithm (AdaGrad), Root Mean Square propagation (RMS-prop) and ADAptive Moment estimation (ADAM) tuning techniques. Using the simplest variation, GD for OLS with a constant learning rate, resulted in a polynomial exceptionally close to the one acquired from the theoretical solution, that is the inversion of the Hessian matrix, for a version of Stone's equation that includes random noise.

1 Introduction

1.1 Theoretical background: [Stone, 1978]

"How is the total meridional heat transport dependent on the particular dynamics and structure of the atmosphere-ocean system?"

This is one of the questions Stone addressed in his 1978 paper ([Stone, 1978]), following the experiments carried out at the Geophysical Fluid Dynamics Laboratory (GFDL) as well as one-dimensional heat balance climate models, both of which suggesting that, in fact, the meridional heat transport is insensitive to the latitudinal structure of the Earth. To confirm these results, Stone used the one-dimensional heat balance equation, which, assuming a state of equilibrium and planetary scales, is exact:

$$\frac{dF}{d\phi} = 2\pi \cdot R^2 \cdot cos\phi \cdot [Q \cdot (1-\alpha) - I], \tag{1}$$

where F is the total flux across a latitude belt, $R = 6.37 \cdot 10^6$ m is the radius of the Earth, Q is the mean *incident* solar radiation per unit area at latitude ϕ, α is the mean albedo at latitude ϕ and I is the mean *emitted* thermal radiation per unit area at latitude ϕ.

He then proceeded into calculating F, using two models: one without and one with the axial tilt of the Earth, while setting α and I as constants, independent of ϕ. His results can be seen in Fig. **1**.

The curve he gets for constant α and I, axial tilt considered, is remarkably close to the values of F he gets from the calculations of [Oort and Haar, 1976]. At the same time, the results for the Northern Hemisphere (N.H.) are very similar to the ones for the Southern Hemisphere (S.H.), in spite of the considerable differences

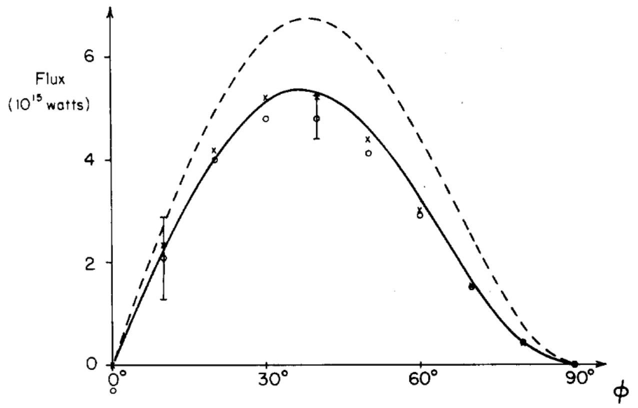


Figure 1: Stone's curve for total energy flux across a latitude circle in the Northern Hemisphere against the latitude, ϕ. The solid line comes from his model that includes the axial tilt but assumes α and *I* to be independent of latitude. The circles indicate the values of heat flux, assuming axial tilt but this time using the latitudinal dependency of α and *I*, calculated by [Oort and Haar, 1976]. The crosses indicate the values of heat flux, calculated by eq.(4).

in topography and ocean cover between the two. In this way he confirmed that the total meridional heat flux is indeed independent of the latitudinal structure of its parameters.

It would be interesting, however, to understand *why* this is so. For this, he rewrote eq.**(1)** in non-dimensional form and then Fourier-expanded its components, concluding to the following expression:

$$\frac{df}{dx} = s\alpha - i, \tag{2}$$

where:

- x= $\sin(\phi)$,
- $f(x) = \frac{2F}{\pi \cdot R^2 \cdot S_0}$, S_0 being the solar constant
- $s(x) = \frac{4Q}{S_0}$
- $\alpha(x) = 1 - \alpha$
- $i(x) = \frac{4I}{S_0}$

After expanding the s, α and i coefficients in Legendre Polynomials, a common technique for spherical coordinates, while assuming zero heat transport at the North Pole as a boundary condition, he ended up with the following expression:

$$f = (s_0 \cdot a_0 + \frac{s_2 \cdot a_2}{5} + ... - i_0)(x-1) + (s_0 \cdot a_2 + s_2 \cdot a_0 + \frac{2}{7}s_2 \cdot a_2 + ... - i_2)(\frac{x^3 - x}{2}) + ... \tag{3}$$

Yet, assuming that the system is in equilibrium, that is, that there is not heat flux in the equator, and considering the values for the s, α and i coefficients, tabulated by [Ellis and Haar, 1976], he simplified eq.**(3)** into:

$$f = \frac{1}{2}(s_0 \cdot a_2 + s_2 \cdot a_0 + \frac{2}{7}s_2 \cdot a_2 - i_2)(x^3 - x) \tag{4}$$

From this equation one can gather that, although the s, α and i coefficients affect the *magnitude* of the heat flux, they do not affect its *distribution* and therefore the maximum of the curve always lies around 35°, even with abrupt changes of these coefficients.

1.2 Present research

The accuracy of the produced curve, compared to the heat flux values calculated by [Oort and Haar, 1976], is fascinating, especially given the simplicity of Stone's reasoning. As an exercise to study optimization, it would be interesting to approach Stone's eq.**(4)** by trying out various machine learning techniques. For this, eq.**(4)** was modelled as a noisy dataset with x being the values of latitude (ϕ) for the N.H. (x $\in$ [0,$\pi/2$]) and y being the value of f with added noise, that is, each value of f, f_i, gets replaced by a value, randomly picked, from a normal distribution with $\mu = f_i$, $\sigma = |0.05 \cdot f_i|$. This leads to a different dataset for each run. But if the use of many different datasets is later needed, although their points will be fixed, swifting from one dataset to another, will be like running the code again and getting a slightly different f for each run. Needing an approximation immune to such swifts, random noise was added to f though all the phases of this project.

At this point, it is important to remind to the reader that f is a *non-dimensional* expression for the meridional heat flux, as a component of eq.**(2)**, receiving very small values as will later be seen (i.e., Fig.**15*b***). Plus, the calculations were performed for ϕ expressed in radians: [0, $\pi/2$] or [0, 1.5708]. Therefore, the domains of both x and y are of a small order of magnitude and thus, no scaling was performed to the dataset.

The dataset having been established, it can now be approached by a simple polynomial fit. In order to settle on the polynomial degree, OLS and Ridge regression were applied to the dataset, along with bootstrap and cross-validation resampling. A theoretical layout of these methods can be found in Secs. 2.1-2.3, while the final decision on the polynomial degree and the best method to find it is given in Sec. 2.4.

To optimize the polynomial coefficients, GD and SGD with and without momentum, coupled with 5 methods of tuning the learning rate were used. In Secs. 3.1- 3.2 a brief overview of their theoretical basis is given, while the results of this investigation, along with the best technique for this dataset are laid in Sec. 3.3.

A brief discussion follows in Sec. 4, regarding the results and suggestions for future work on modelling the meridional heat transport follow right up, in Sec 5.

For the interested reader, there are some results in the Appendices part of the report, that are not included in the main text, while all of the codes can be found in my GitHub page, where all of the computations performed during this research can be reproduced.

2 Methods for assessing the polynomial degree

Needing to settle on the polynomial degree that fits best Stone's equation, I calculated the MSE of the assessed heat flux function, against the noisy version of Stone's equation, using plain OLS and Ridge regression, as well as their resampling variations with bootstrap and cross validation. A small introduction to the theoretical basis of these methods is considered necessary at this point for the understanding of my reasoning and conclusions further on.

2.1 Regression methods

Through regression methods we aim to find a dependence of the mean value of some random variable on another variable or on several variables (Encyclopedia of Mathematics). Extending this concept, having an input and an output dataset, we aim to find a continuous function that connects the two.

One of the most common regression methods is the Linear Regression (Hjorth-Jensen), according to which we try to fit to our output dataset, $\mathbf{y}$ a model that follows the relationship:

$$\tilde{\boldsymbol{y}} = \boldsymbol{X} \cdot \boldsymbol{\beta} + \boldsymbol{\epsilon}, \qquad \textbf{(5)}$$

where:

- $\tilde{\boldsymbol{y}}$ is the new model we use to approach the output dataset, $\mathbf{y}$.
- $\boldsymbol{X}$ is the *design matrix* of the model. It is constructed by receiving the input values, x_i of the data and then arranging them in a polynomial form (Vardemonde matrix) or any other orthogonal function

(sinusoidal, cosinusoidal, Fourier expansion, etc). In the present study $\boldsymbol{X}$ will have a simple polynomial expression.

- $\boldsymbol{\beta}$ is the vector containing coefficients of the linear relationship.
- $\boldsymbol{\epsilon}$ some noise we add to the model, usually following a normal distribution: $\boldsymbol{\epsilon} \sim N(0, \sigma^2)$

The main challenge in all regression methods is finding the optimum value of the vector $\boldsymbol{\beta}$, the one that minimizes the difference between the actual output, $\mathbf{y}$ and the model's output, $\tilde{\boldsymbol{y}}$. In the present research, three methods were studied in order to obtain these coefficients. In the sections that follow, the reader can find the basic elements of Ordinary Least Squares (Sec. 2.1.1), Ridge (Sec. 2.1.2) and LASSO (Sec. 2.1.3) regressions. Although some additional optimization techniques will later be added to these basic regression methods, they are enough at this point to at least conclude on the degree of the polynomial that will fit the dataset best.

2.1.1 Ordinary Least Squares (OLS)

Set as $\boldsymbol{y}$ the noisy version of Stone's heat flux function, as described in Sec. 1.2 above and $\tilde{\boldsymbol{y}} = \mathbf{X} \cdot \boldsymbol{\beta}$ the polynomial to fit $\mathbf{y}$. The noise in included in $\tilde{\boldsymbol{y}}$ itself.

The aim of every machine learning technique is to minimize the error between the values of $\mathbf{y}$ and $\tilde{\boldsymbol{y}}$.

The Mean Squared Error is used to represent the difference of these two values. In Ordinary Least Squares, the MSE is defined as:

$$\mathbf{MSE} = \frac{1}{n} \cdot (\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta}) \tag{6}$$

Therefore, we are now looking for the polynomial coefficients that will minimize the MSE, or, in other words, the $\boldsymbol{\beta}$ that gives:

$$\frac{\partial \boldsymbol{MSE}}{\partial \boldsymbol{\beta}} = 0 \tag{7}$$

Following the reasoning of [Hastie et al., 2009], Sec.3.2: from eqs. **(6)** and **(7)**, setting $\boldsymbol{\omega} = \mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta}$, we get:

$$\frac{1}{n} \cdot \frac{\partial(\boldsymbol{\omega^T} \cdot \boldsymbol{\omega})}{\partial \boldsymbol{\beta}} = 0 \Rightarrow \frac{\partial(\boldsymbol{\omega^T} \cdot \boldsymbol{\omega})}{\partial \boldsymbol{\beta}} = 0 \Rightarrow \boldsymbol{\omega^T} \cdot \frac{\partial \boldsymbol{\omega}}{\partial \boldsymbol{\beta}} + \boldsymbol{\omega^T} \cdot \frac{\partial \boldsymbol{\omega}}{\partial \boldsymbol{\beta}} = 0 \Rightarrow$$

$$\Rightarrow 2 \cdot \boldsymbol{\omega^T} \cdot \tfrac{\partial \boldsymbol{\omega}}{\partial \boldsymbol{\beta}} = 0 \Rightarrow (\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta})^T \cdot \tfrac{\partial}{\partial \beta}(\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta}) \Rightarrow$$

$$\Rightarrow (\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta})^T \cdot [\tfrac{\partial \boldsymbol{y}}{\partial \beta} - \tfrac{\partial}{\partial \beta}(\mathbf{X} \cdot \boldsymbol{\beta})] = 0 \tag{8}$$

because $\mathbf{y} \neq \mathbf{y}(\boldsymbol{\beta})$ and $\mathbf{X} \neq \mathbf{X}(\boldsymbol{\beta})$, eq. **(8)** becomes:

$$(\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta})^T \cdot \mathbf{X} = 0 \Rightarrow \mathbf{y}^T \cdot \mathbf{X} - (\mathbf{X} \cdot \boldsymbol{\beta})^T \cdot \mathbf{X} = 0 \Rightarrow (\mathbf{X} \cdot \boldsymbol{\beta})^T \cdot \mathbf{X} = \mathbf{y}^T \cdot \mathbf{X} \Rightarrow$$

$$\Rightarrow (\mathbf{X} \cdot \boldsymbol{\beta})^T \cdot (\mathbf{X}^T)^T = \mathbf{y}^T \cdot (\mathbf{X}^T)^T \Rightarrow$$

$$\Rightarrow (\mathbf{X}^T \cdot \mathbf{X} \cdot \boldsymbol{\beta})^T = (\mathbf{X}^T \cdot \mathbf{y})^T \Rightarrow \mathbf{X}^T \cdot \mathbf{X} \cdot \boldsymbol{\beta} = \mathbf{X}^T \cdot \mathbf{y} \Rightarrow$$

$$\Rightarrow \boldsymbol{\beta} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y} \tag{9}$$

The second to last step can be taken because the transpose of a matrix is unique, therefore if two transposed matrices are equal with each other, so are the matrices themselves.

Finding the values of the vector $\boldsymbol{\beta}$ is therefore relatively simple. The only challenge from this point on would be to find the inverse of the Hessian matrix ($\boldsymbol{H} = \mathbf{X}^T \cdot \mathbf{X}$). LU, QR and Cholesky decomposition are some of the typical ways to process this. Yet, for large dimensionalities of $\boldsymbol{X}$, these methods could lead to singularities. In this case, one could invert $\boldsymbol{H}$, using Singular Value Decomposition (SVD). Because the present case turned out rather simple, I applied a plain matrix inversion in my code. However, if the reader is interested in applying OLS to more complicated cases, SVD would be a good alternative to matrix inversion and more details on the technique can be found in Appendix A.

2.1.2 Ridge regression

Another, rather cheap, way to avoid overflowing during the inversion of $\boldsymbol{H}$ is adding a small diagonal component to the cost function, here the **MSE**, as follows ([Hastie et al., 2009], Sec. 3.4.1):

$$\mathbf{MSE} = \frac{1}{n} \cdot [(\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta}) + \lambda \cdot (\boldsymbol{\beta}^T \cdot \boldsymbol{\beta})] \tag{10}$$

Following the same process as before, we end up with the optimum coefficients being expressed by:

$$\boldsymbol{\beta} = (\mathbf{X}^T \cdot \mathbf{X} + \lambda \cdot \mathbf{I})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y} \tag{11}$$

, with **I** being the identity matrix.

λ is a positive constant as small as we like but not zero (for $\lambda = 0$ we are back to OLS, so we are not talking about Ridge Regression anymore). We can minimize the **MSE** by tuning this hyperparameter λ.

2.1.3 LASSO regression

Another alternative to circumvent the calculation of the inverse of **H** is expressing the **MSE** as follows:

$$\mathbf{MSE} = \frac{1}{n} \cdot [(\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta}) + \lambda \cdot ||\boldsymbol{\beta}||_1] \tag{12}$$

, where $||\boldsymbol{\beta}||_1$ is the norm-1 value of $\boldsymbol{\beta}$, that is:

$$||\boldsymbol{\beta}||_1 = \sum_i |\beta_i| \tag{13}$$

In this case, because the derivative

$$\frac{d|\beta|}{d\boldsymbol{\beta}} = sgn(\boldsymbol{\beta}) = \begin{cases} 1 & ,\beta > 0 \\ -1 & ,\beta < 0 \end{cases} \tag{14}$$

minimizing the **MSE** and following the same steps as before lead to the following expression for β:

$$\frac{\partial \boldsymbol{MSE}}{\partial \boldsymbol{\beta}} = -2\boldsymbol{X^T} \cdot (\mathbf{y} - \boldsymbol{X\beta}) + \lambda \cdot sgn(\boldsymbol{\beta}) = 0 \Rightarrow \boldsymbol{X^T X \beta} + \lambda \cdot sgn(\boldsymbol{\beta}) = 2\boldsymbol{X^T y} \tag{15}$$

This doesn't lead to an analytical expression for $\boldsymbol{\beta}$ as before. Indeed, to calculate the polynomial coefficients, we have to work on the level of the matrix elements.

The final expression we get for $\boldsymbol{\beta}$ is:

$$\beta_i = \begin{cases} y_i - \frac{\lambda}{2} & , y_i > \frac{\lambda}{2} \\ y_i + \frac{\lambda}{2} & , y_i < -\frac{\lambda}{2} \\ 0 & , y_i \leq |\frac{\lambda}{2}| \end{cases} \tag{16}$$

Again here, λ is a tuneable hyperparameter that can minimize the **MSE**.

It would be interesting at this point to visualize the effect of the hyperparameter λ of Ridge and LASSO Regressions on the $\boldsymbol{\beta}$ coefficients.

Comparing eqs.**(9)** and **(11)**, we get that:

$$\boldsymbol{\beta_{Ridge}} = \frac{\boldsymbol{\beta_{OLS}}}{1+\lambda} = \frac{\mathbf{y}}{1+\lambda} \tag{17}$$

, for $\boldsymbol{X}$ being an orthogonal matrix.

This and eq.**(16)** can be expressed in the following plot:

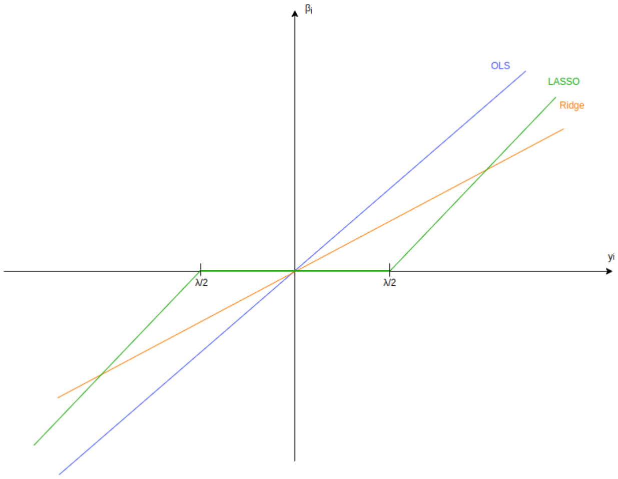


Figure 2: The effect of each regression method on the polynomial coefficients, $\boldsymbol{\beta}$

As we can see, Ridge regression scales down the $\boldsymbol{\beta}$ coefficients, while LASSO regression sets to zero certain values of $\boldsymbol{\beta}$ for a part of the λ domain. This is commonly known as "soft-thresholding" ([Mehta et al., 2019]).

2.2 Assessing a model: the bias-variance trade-off

Following the reasoning of [Mehta et al., 2019], we split the **MSE** into the *Bias*^{2}, *Variance* and *Noise* components as follows:

$$\begin{aligned} \mathbf{MSE} = \frac{1}{n} \cdot (\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X} \cdot \boldsymbol{\beta}) = \mathbb{E}[\sum_i (y_i - \tilde{y}_i)^2] = \\ \mathbb{E}[\sum_i (y_i - f(x_i) + f(x_i) - \tilde{y}_i)^2] = \\ \sum_i \mathbb{E}[(y_i - f(x_i))^2] + \mathbb{E}[(f(x_i) - \tilde{y}_i)^2] + 2\mathbb{E}[(y_i - f(x_i))(f(x_i) - \tilde{y}_i)], \end{aligned} \tag{18}$$

where:

- $[y_i - f(x_i)], [f(x_i) - \tilde{y_i}]$ statistically independent, therefore:

  $$\mathbb{E}[(y_i - f(x_i))(f(x_i) - \tilde{y_i})] = \mathbb{E}[(y_i - f(x_i))] \cdot \mathbb{E}[f(x_i) - \tilde{y_i}]$$

  Now, because we've defined: $y_i = f(x_i) + \epsilon \Rightarrow y_i - f(x_i) = \epsilon$, so:

  $$\mathbb{E}[(y_i - f(x_i))(f(x_i) - \tilde{y_i})] = \mathbb{E}[\epsilon] \cdot \mathbb{E}[f(x_i) - \tilde{y_i}] = 0 \cdot \mathbb{E}[f(x_i) - \tilde{y_i}] = 0 \Rightarrow$$
  $$\Rightarrow 2\mathbb{E}[(y_i - f(x_i))(f(x_i) - \tilde{y_i})] = 0 \tag{19}$$

- $\mathbb{E}[(y_i - f(x_i))^2] = \mathbb{E}[\epsilon^2] = \sigma^2$

  $$\tag{20}$$

- $\mathbb{E}[(f(x_i) - \tilde{y_i})^2] = \mathbb{E}[(f(x_i) - \mathbb{E}[\tilde{y_i}] + \mathbb{E}[\tilde{y_i}] - \tilde{y_i})^2] =$

  $$= \mathbb{E}[(f(x_i) - \mathbb{E}[\tilde{y_i}])^2] + \mathbb{E}[(\mathbb{E}[\tilde{y_i}] - \tilde{y_i})^2] + 2\mathbb{E}[(f(x_i) - \mathbb{E}[\tilde{y_i}])(\mathbb{E}[\tilde{y_i}] - \tilde{y_i})], \tag{21}$$

  where: $[f(x_i) - \mathbb{E}[\tilde{y_i}]], [\mathbb{E}[\tilde{y_i}] - \tilde{y_i}]$ statistically independent, therefore:

  $$\begin{aligned}\mathbb{E}[(f(x_i) - \mathbb{E}[\tilde{y_i}])(\mathbb{E}[\tilde{y_i}] - \tilde{y_i})] &= \mathbb{E}[(f(x_i) - \mathbb{E}[\tilde{y_i}])] \cdot \mathbb{E}[\mathbb{E}[\tilde{y_i}] - \tilde{y_i}] = \\ &= \mathbb{E}[(f(x_i) - \mathbb{E}[\tilde{y_i}])] \cdot \mathbb{E}[\mathbb{E}[\tilde{y_i}] - \mathbb{E}[\tilde{y_i}] = \\ &= \mathbb{E}[(f(x_i) - \mathbb{E}[\tilde{y_i}])] \cdot \mathbb{E}[\tilde{y_i}] - \mathbb{E}[\tilde{y_i}] = \\ &= \mathbb{E}[(f(x_i) - \mathbb{E}[\tilde{y_i}])] \cdot 0 = 0 \end{aligned} \tag{22}$$

Therefore:

$$\mathbf{MSE} = \sum_i \mathbb{E}[(f(x_i) - \mathbb{E}[\tilde{y_i}])^2] + \sum_i \mathbb{E}[(\mathbb{E}[\tilde{y_i}] - \tilde{y_i}])^2 + \sum_i \sigma^2 \tag{23}$$

The first term on the right-hand side of eq.**(23)** is the $Bias^2$, measuring the deviation of the estimate from the true value of the model.

The second term is the *Variance*, that measures the fluctuation of the estimate from the actual value in proportion to the number of data each time available.

The last term is the effect of the noise we've added to our model and it is an *irreducible* error ([Hastie et al., 2009]). So, because both $Bias^2$ and Variance cannot take any negative values, the total MSE cannot drop below the value of the model's noise.

To understand the concept of *bias-variance trade-off*, we need to focus on eq.**(23)**. As we can see, in order to have a low **MSE**, we need to aim for low bias and variance. Yet one of the main challenges in machine learning is finding a balance between the $Bias^2$ and the variance. Keeping the **MSE** constant we see that raising the bias of the model (by choosing it to be of small complexity) leads to a lower variance. Therefore, even if the amount of data is limited, that would cause very little noise.

This is also a good point to mention the concepts of *underfitting* and *overfitting* (see also [Hastie et al., 2009], pg. 221).

When we start from small polynomial degrees while training a model, there is a high chance that our model will not fit the data accurately. For example, in Figs. 5 it is obvious that for polynomial of degree lower or equal to 3 the MSE is significant and the difference between the training and test **MSE** is prominent. This is the case of underfitting. If, however, we raise the polynomial degree (or the complexity of the model in general), the model fits the training data too much, not recognising the test data as efficiently. This is the case of overfitting. Both cases should be kept in mind when evaluating the model's bias-variance trade-off.

2.3 Resampling methods

During resampling, the model of interest if being fitted to a number of random samples, repeatedly drawn from the train dataset. In that way, we can obtain information that would not be available from fitting the model only once using just the original training sample (Hjorth-Jensen).

2.3.1 Bootstrap resampling

Suggested by [Efron, 1979] the Bootstrap resampling technique can be described by the following phseudocode:

Algorithm 1: Bootstrap resampling

After splitting out dataset to test and train data, we pick up a number or random datapoints from each set and evaluate the MSE of each subset. The final MSE is the mean of the MSE of each subset. Fig.3 is a good visualization of this technique.

```
for i=1, M (number of bootstrap resamplings):
    -pick up a random number of the train dataset (D) and create a new dataset (D*), of equal size, replacing
    the missing slots with already selected data values.
    -train the model.
    -calculate the $MSE_{test}(i)$ for the original (untouched) test dataset.
end loop
compute: $MSE = \frac{1}{M}\sum_{i=1}^{M} MSE_{test}(i)$
```

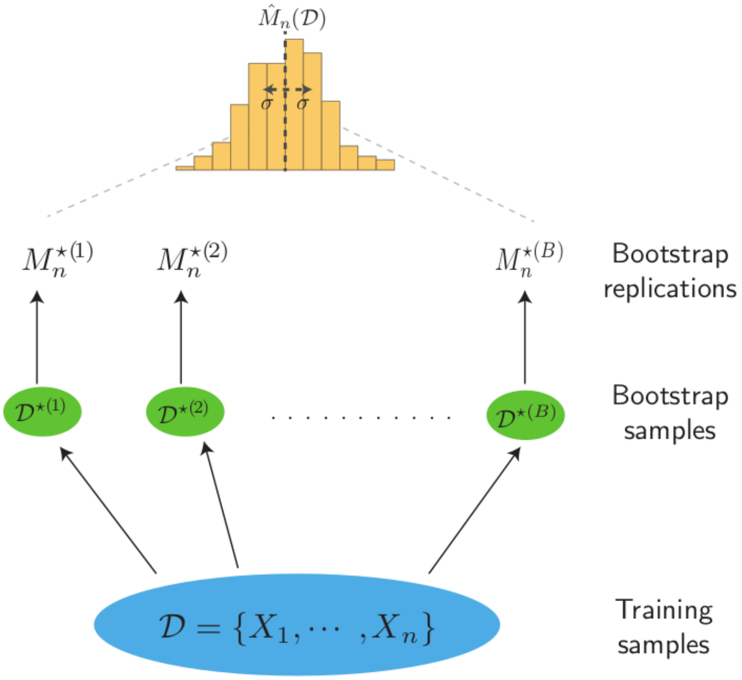


Figure 3: The Bootstrap resampling techniques, as explained by [Mehta et al., 2019]. D is the originally split training dataset. D* are the randomly selected subgroups (bootstrapped datasets). For each one of them, the MSE (M*) is calculated, while the final MSE is the mean of all M*s.

The number of bootstraps is a hyperparameter in itself, ranging from 1 to the total number of datapoints (a computationally expensive option).

Working with different training datasets during the bootstrapping procedure reduces the variance of our model. Yet, we pay this with a raise in the bias of the model ([Mehta et al., 2019], pg.39).

2.3.2 Cross-Validation (CV) resampling

Cross-Validation resampling, first studied in a regression context by Stone ([Stone, 1974], [Stone, 1977]) for the LOOCV case (see below Fig.4) and Geisser ([GEISSER, 1974], [Geisser, 1975]) for the V-fold case ([Celisse, 2014]) can be described through the following pheudocode:

Algorithm 2: Cross Validation resampling

We split the dataset into a number of k, roughly equal-sized, subsets, called *folds*. Unlike in the bootstrap case, the subsets we split the datasets into are now *mutually exclusive*. Each time, one of the subsets is the test dataset and the rest the training datasets. Every subset gets to be a test set in turn. The MSE of the dataset is calculated for every test dataset and the final MSE is the mean of the MSE of each test subset.

```
shuffle the dataset randomly.
split the dataset into k groups.
for i in k:
    -decide which group to use as set for test data
    -take the remaining groups as a training data set
    -train the model
calculate the $MSE_{test}(i)$ the test dataset
end loop
compute: $MSE = \frac{1}{k}\sum_{i=1}^{k} MSE_{test}(i)$
```

Fig.(4) is a good visualization of the splitting of the dataset.

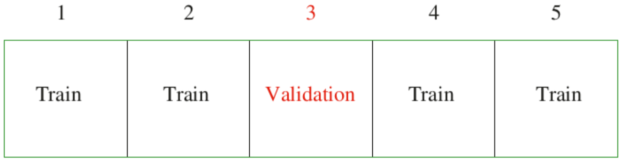


Figure 4: The splitting of the dataset in training and test subsets. Image borrowed from [Hastie et al., 2009]

The number of k-folds is usually chosen to be 5 or 10. The case in which it is equal to the number of samples is called *Leave-One-Out Cross-Validation* (LOOCV). LOOCV tends to have a high variance and is also computationally expensive. Yet it produces almost zero bias. On the other hand, for k=5 or 10, the bias is larger but the variance smaller [Hastie et al., 2009]. The choice of the optimum number of k-folds depends on the model to fit. As we will see later, in the present study, the LOOCV tends to give the best results, without any significant computational cost.

2.4 Results

All the three regression methods mentioned above were tested against the dataset described in Sec.1.2. After trying the plain version of all, I applied bootstrap and cross validation resampling, in search for a definite value of the polynomial degree. The results for OLS and Ridge regression as well as their resampling variations are all listed and analysed in the sections that follow. Yet, because the results of LASSO regression did not prove very helpful in deciding upon the polynomial degree, they are not mentioned here. The interested reader can easily reproduce them, using the respective code from my GitHub page.

2.4.1 Plain OLS Regression

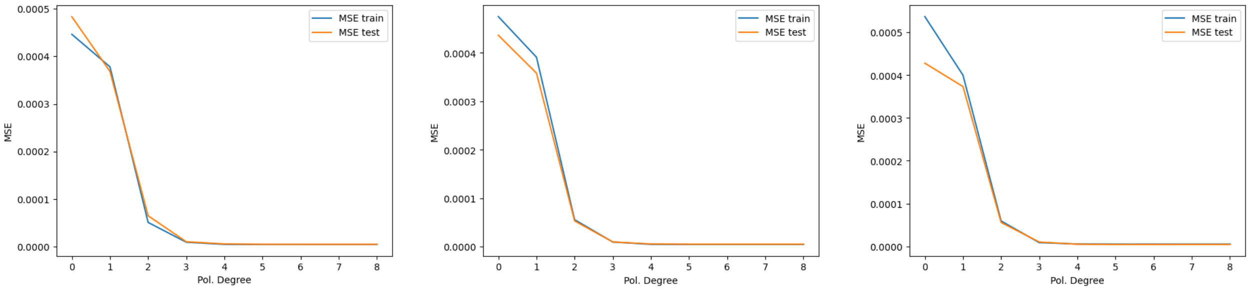
I performed an OLS fitting to the noisy version of eq.**(4)**, varying the number of datapoints (200, 500 and 1000) and the percentage of the training and test data (test data being 25%, 50% and 75% of the total dataset).

The calculated MSE for each combination is given in Figs. 5-7, while the respective correlation figures can be found in Appendix B.

Raising the number of datapoints gives, as expected, a smaller MSE and a better match between the training and test MSE values. Yet, since the aim of this analysis is to settle on a polynomial degree, acquiring the smallest value of the MSE is of secondary importance. Already from the first set of runs (for n=200) we can see that the MSE flattens to a near-zero value for polynomial degree greater or equal to 4, a result that we do get from the other two sets of runs (n=500 and n=1000) but in a longer set of iterations. Therefore, it is safe to settle on **n=200** datapoints for the rest of the study. This is a fairly small number but data availability is not always ample, so it is preferable to develop a model, capable of fair predictions on a small dataset.

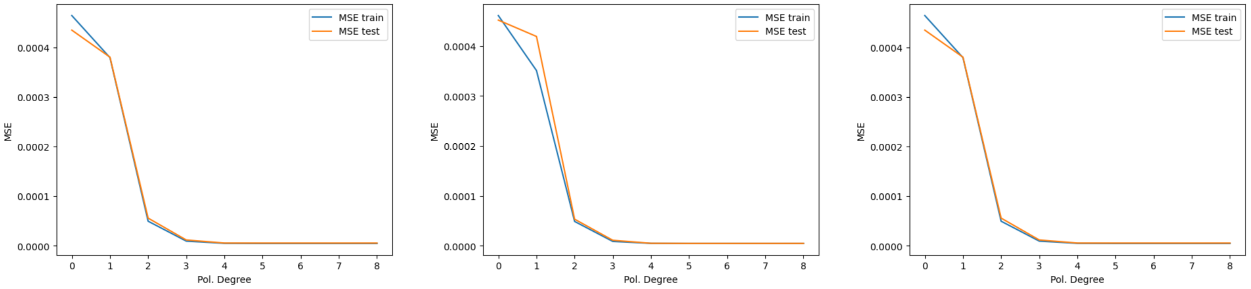
In this context, one can also observe that there is obvious underfitting happening for polynomial degrees lower than or equal to 3 but no overfitting happening afterwards. As we mentioned in Sec. (2.2) a small polynomial degree leads to a model of high bias. Therefore, although the variance is in this case small, the contribution of the bias to the MSE is significant, leading to large values of the latter. However, we also discussed that the variance of the model depends on the number of datapoints available. If the dataset is large enough, that leads to a low variance, regardless of the model's complexity. This is what is happening here; because the number of datapoints is more than enough, even from the first set of runs with n=200, the variance is always small and therefore for polynomial degrees larger than 3, where the bias gets small as well, the total MSE, being the sum of the bias, the variance and the default model noise (see eq. **(23)**) gets a very small value as well. Had the dataset been smaller, the variance would have been of a larger default value, leading to overfitting way earlier in the tested range of polynomial degrees.

The other parameter I varied was the percentage of the test data with respect to the whole dataset. It gets more obvious at the last two runs (for n=500 and n=1000) that the case where the test dataset is 50% of the entire dataset is the one with the largest difference between the test and training MSE. The results for 25% and 75% are in general very similar, inspite 75% being a rather high percentage for a test dataset. Finally, the **test dataset was chosen to be 25%** of the whole dataset, as this is a usual value.



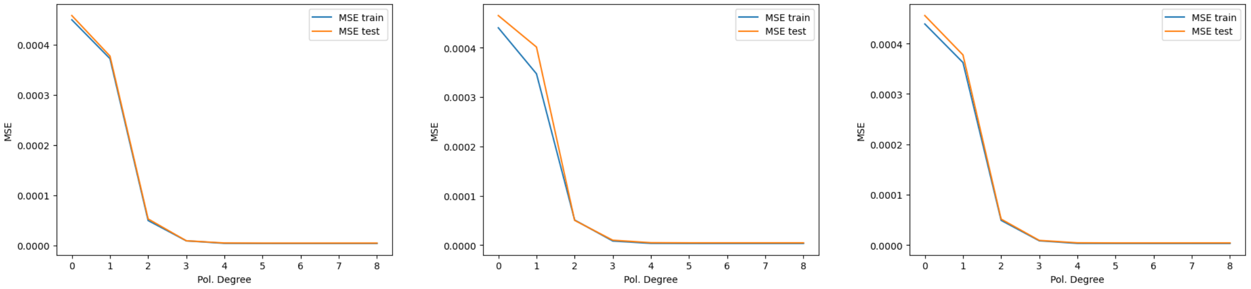
(a) n=200, test size = 25%. **(b) n=200, test size = 50%.** **(c) n=200, test size = 75%.**

Figure 5: MSE for the plain OLS case with n=200 datapoints.



(a) n=500, test size = 25%. **(b) n=500, test size = 50%.** **(c) n=500, test size = 75%.**

Figure 6: MSE for the plain OLS case with n=500 datapoints.



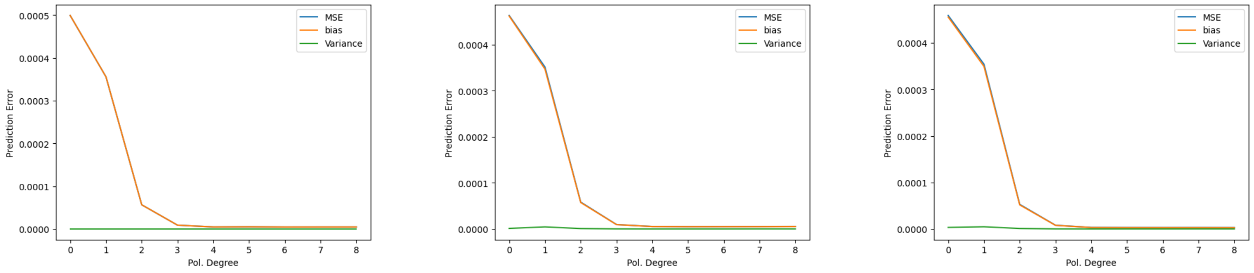
(a) n=1000, test size = 25%. **(b) n=1000, test size = 50%.** **(c) n=1000, test size = 75%.**

Figure 7: MSE for the plain OLS case with n=1000 datapoints.

2.4.2 Bootstrap resampling on OLS

Keeping the number of datapoints equal to 200 and the test set percentage equal to 25% from above, I only varied the number of bootstrap resamplings in this case (1, 20 and 100).

Studying now the bias-variance trade-off of the model, it is interesting to see in Figs.8 that raising the number of bootstraps leads to a slight increase of the variance. This could be attributed to the fact that the variance is calculated as the mean of all the variances of for each bootstrap sample. Although each one of them has the same number of datapoints as the original training set, it is qualitatively different, due to the randomness of the resampling. Since variance is affected by the datapoints themselves, a qualitative change in the dataset it is calculated for (for example if some data get repeated many times in one sample) could affect its final value.



(a) number of bootstraps = 1. **(b) number of bootstraps = 20.** **(c) number of bootstraps = 100.**

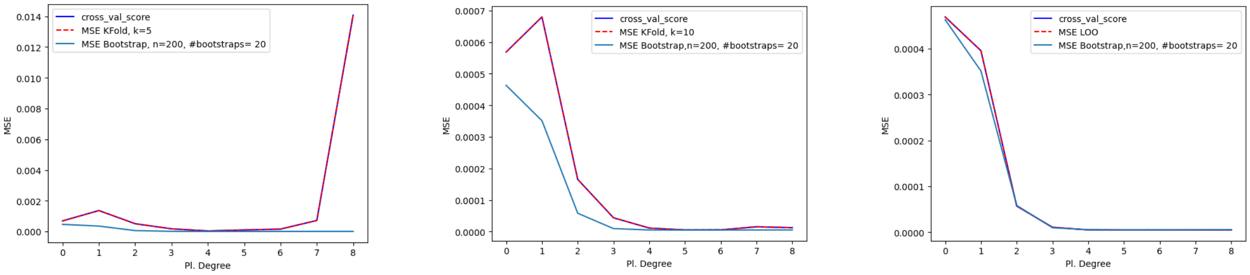
Figure 8: Bias-variance trade-off for bootstrap resampling on OLS Regression with n=200 datapoints.

2.4.3 Cross-validation resampling on OLS

Keeping the same values for the number of datapoints and test size, I applied CV resampling on the OLS regression, varying the number of k-folds (k=5, 10 or 200, corresponding to the LOO-CV case, since the number of datapoints is also 200) and compared the results with the case where the number of bootstraps is 20 (see Fig.8b).

It is interesting here that the result get closer to the bootstrap case as the number of k-folds raises, with the best ones belonging to the LOO-CV case, where the number of k-folds is 200. And yet, the results do not match completely before the polynomial degree gets equal to 4. It is important at this point to remember that both Bootstrap and CV are model *assessing* techniques: they were used here to settle on the polynomial degree, the MSE values themselves not being as important yet. Therefore, it is easy now to see that both techniques agree that the degree of the polynomial must be over 4, a result we already had from the plain OLS Regression.

Unfortunately, we cannot go much further with OLS. The MSE seems to be very similar for polynomial degrees larger than 4 and, thus, at this point, we can make no clear assumptions. For this reason, we need to seek for another technique that highlights the impact of the polynomial degree on the MSE more. As we will see in the next section, Ridge regression can provide a more definite answer.



(a) number of kfolds= 5. **(b) number of kfolds= 10.** **(c) LOO cross-validation.**

Figure 9: MSE for CV resampling on OLS with n=200 datapoints.

2.4.4 Plain Ridge Regression

In this case, the polynomial degree was varied in the same range as before, that is from 0 to 9, and the hyperparameter λ in the range $[10^{-4}, 10^{10}]$.

The calculated MSE for all combinations can be seen in Fig.10. This figure is particularly important because it enables us to decide on the polynomial degree just by one look! Indeed, it is easy to see that the band of polynomial degrees equal 5 and 6 give the minimum MSE. Because there is random noise included in the modelled heat flux function, the calculated MSE and thus the value of the polynomial degree that gives the best results varies for each run. Yet it always lies in this area. If the code is run many times, one can conclude that choosing the **polynomial degree to be equal to 6** is a safe decision for this modelling.

It is interesting to notice here that overfitting is not out of the question for this case. Indeed, in Figs.11 there is always some overfitting happening for $\lambda > 10^3$ but it gets more prominent at the largest tested value of the polynomial degree, 9. An effort for interpreting this is given in the following section, where the bias-variance trade-off of the model is tested through bootstrap resampling. The case where the polynomial degree is equal to 6 is the one that provides the lowest amount of overfitting (but also underfitting) of the 3 cases plotted in Fig.11.

2.4.5 Bootstrap on Ridge

Having settled on the polynomial degree, we can test the bias-variance trade-off of the model by applying bootstrap resampling on the previous Ridge regression algorithm. As can be seen from Figs.12, the variance still remains low, the number of datasets being enough for this case too. It is also interesting to see how radically the bias increases by increasing the values of λ.

Compared to Fig.11b, these curves have the same sigmoid structure but reach to a higher MSE than in the plain Ridge regression case. This did not happen when we applied bootstrap resampling on OLS.

Again the little variance that exists seems to be raised slightly when raising the number of bootstraps. Yet increasing the number of bootstraps seems to give a less noisy curve.

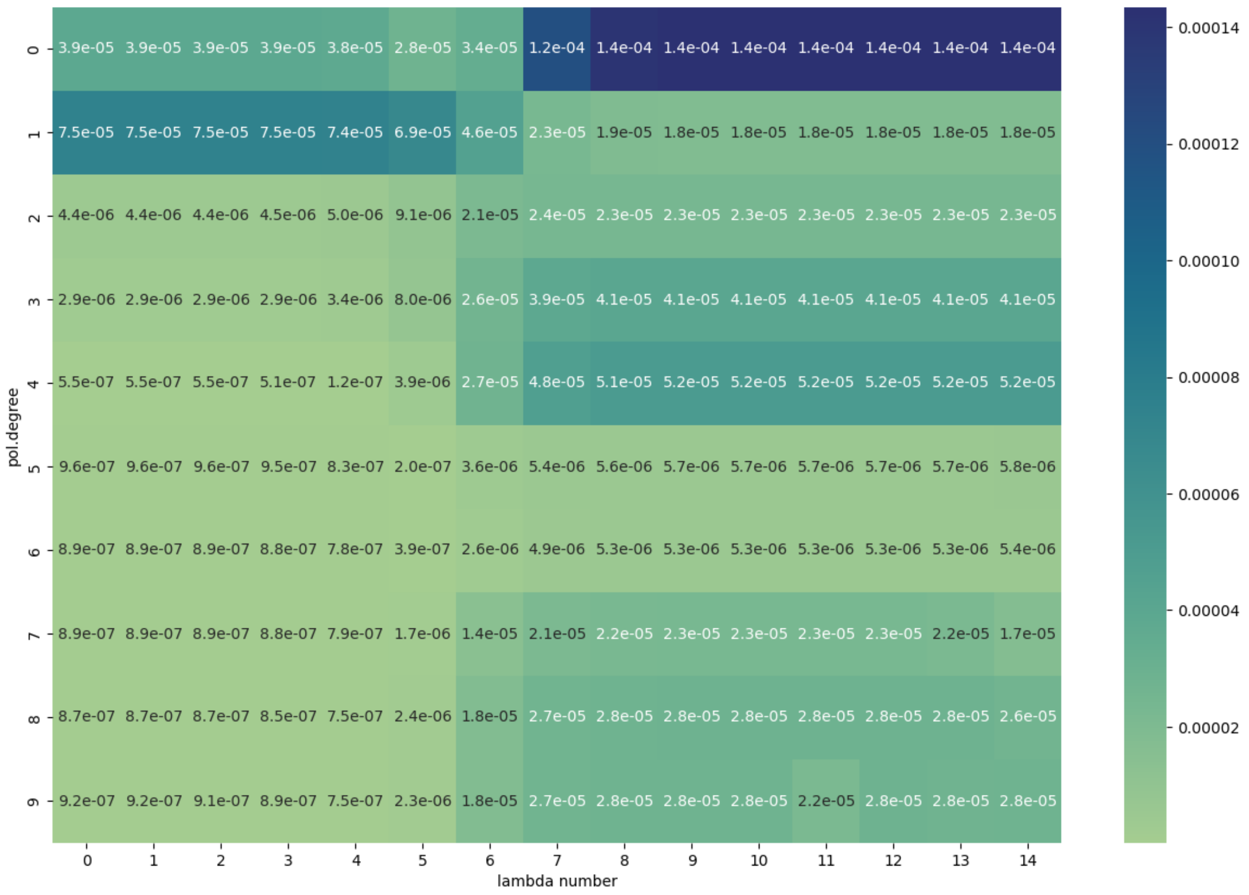


Figure 10: MSE varying with the polynomial degree and the hyperparameter λ. Attention must be paid during reading this plot: each value of the x-axis corresponds to the *numbering* of an integer power of λ. For example, 0 is for $\lambda = 10^{-4}$ (the beginning of the spectrum), 1 is for $\lambda = 10^{-4}$ e.t.c. The numbering of the y-axis represents the polynomial degree as it is. This plot was created using Scikit'learn's package, *seaborn*.

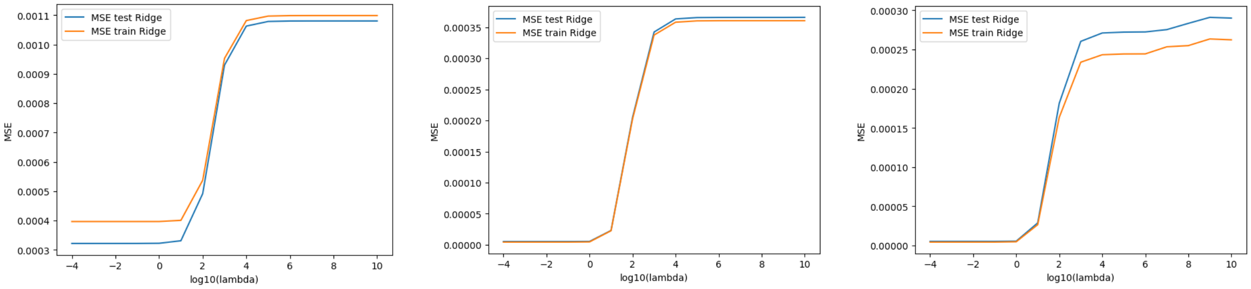
2.4.6 Cross-validation on Ridge

Keeping the polynomial degree and the range of λ stable, I applied CV to the Ridge regression algorithm, in order to compare the results with the bootstrap resampling case above. Varying the number of k-folds (5, 10 or 200 for the LOO C-V case) leads to the results plotted in Figs.13. Again, the best agreement with the bootstrap case (number of bootstraps=20) comes from the LOO C-V. Although the algorithm's MSE and the one we get from the scikit-learn CV algorithm match perfectly, the behaviour of the CV curves is altogether completely different than the bootstrap ones, save from he LOO CV case.

Yet, again because both Bootstrap and CV resampling are model-assessing techniques, if we had to settle on a value of λ, we can see that the area $[10^{-4}, 10^{-2}]$ is quite safe, whereas for greater values the MSE blows up abruptly.

3 Methods for optimizing the polynomial coefficients

Having settled on the polynomial degree, we are now looking for the coefficients that will give the minimum MSE and will, therefore, approach the model best. For this I employed the methods of Gradient Descent (GD) and Stochastic Gradient Descent (SGD) both with and without momentum. What this means is explained in the theory section that follows, along with the methods I used for tuning the learning rate, γ. Then my results (see Sec.3.3.1) and my final decision on the method that fits this case best can be better understood. This entire section reproduces the reasoning of [Mehta et al., 2019], pg.14-18.

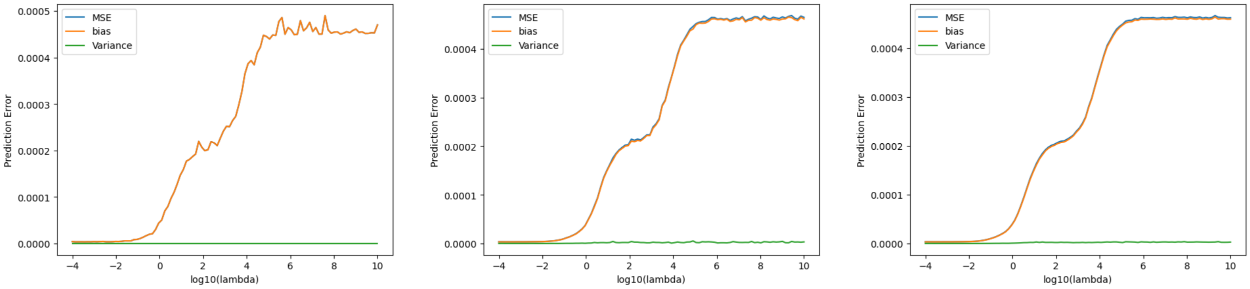


(a) Pol. degree=1.

(b) Pol. degree=6.

(c) Pol. degree=9.

Figure 11: MSE for plain Ridge regression.

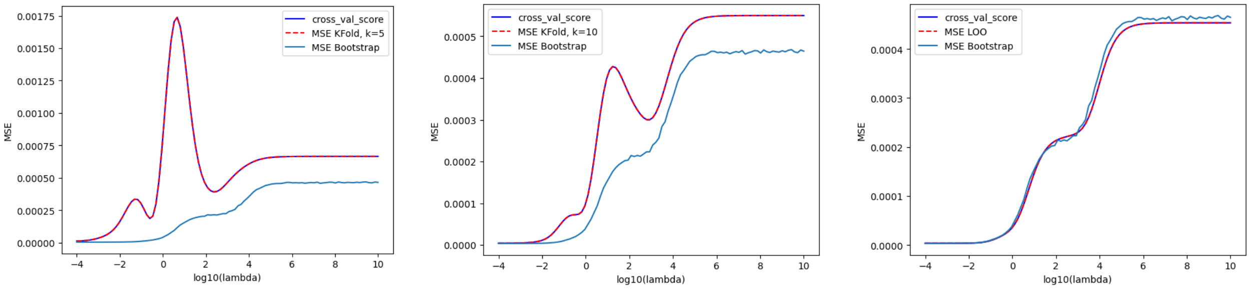


(a) number of bootstraps=1.

(b) number of bootstraps=20.

(c) number of bootstraps=100.

Figure 12: Bias-variance trade-off for bootstrap resampling on Ridge Regression with pol.degree=6.



(a) number of kfolds=5.

(b) number of kfolds=10.

(c) LOO C-V.

Figure 13: The MSE calculated with CV on Ridge regression, compared with the boostrap resampling on Ridge regression for number of bootstraps = 20. CV was both calculated (red dashed line) and directly implemented from scikit-learn (solid purple line).

3.1 Optimization techniques

3.1.1 The Gradient Descent (GD)

Gradient Descent is inspired by the Newton-Raphson method, according to which we Taylor-expand the cost function (here, the MSE) around the optimum value of $\boldsymbol{\beta}$, $\hat{\boldsymbol{\beta}}$ in the following way:

$$\boldsymbol{MSE}(\hat{\beta}) = \boldsymbol{MSE}(\boldsymbol{\beta_k}) + \boldsymbol{g^T}(\boldsymbol{\beta_k}) \cdot (\boldsymbol{\beta} - \boldsymbol{\beta_k}) + \frac{\mathbf{1}}{\mathbf{2}} \cdot (\boldsymbol{\beta} - \boldsymbol{\beta_k})^{\boldsymbol{T}} \cdot \boldsymbol{H}(\boldsymbol{\beta_k}) \cdot (\boldsymbol{\beta} - \boldsymbol{\beta_k}) + \boldsymbol{O}((\boldsymbol{\beta} - \boldsymbol{\beta_k})^{\mathbf{3}}), \quad \mathbf{(24)}$$

where:

- $\boldsymbol{g^T}(\boldsymbol{\beta_k}) = (\frac{\boldsymbol{\partial MSE}}{\boldsymbol{\partial \beta_k}})^{\boldsymbol{T}} = \frac{n}{2} \cdot \boldsymbol{X^T} \cdot (\boldsymbol{X} \cdot \boldsymbol{\beta_k} - \mathbf{y})$, following the differentiation process in Sec.2.1.1, is the gradient of the MSE with respect to the β calculated from the previous iteration, β_k.
- $\boldsymbol{H}(\boldsymbol{\beta_k})$ is the Hessian matrix of the second derivatives of MSE with respect to $\boldsymbol{\beta_k}$: $\boldsymbol{H}(\boldsymbol{\beta_k}) = \boldsymbol{J}_{\frac{\partial \mathbf{MSE}}{\partial \boldsymbol{\beta_k}}}$.
- $\boldsymbol{O}((\boldsymbol{\beta} - \boldsymbol{\beta_k})^{\mathbf{3}})$ are higher-order terms that get truncated.

Setting $\boldsymbol{b} = \boldsymbol{\beta} - \boldsymbol{\beta_k}$ we differentiate eq.(24) with respect to $\mathbf{b}$, remembering that $\frac{\partial \mathbf{MSE}}{\partial \mathbf{b}} = 0$. Therefore, we get:

$$\mathbf{b} = -\boldsymbol{H^{-1}}(\boldsymbol{\beta_k}) \cdot \boldsymbol{g^T}(\boldsymbol{\beta_k}) \Rightarrow$$

$$\hat{\boldsymbol{\beta}} = \boldsymbol{\beta_k} \text{ - } \boldsymbol{H^{-1}}(\boldsymbol{\beta_k}) \cdot \boldsymbol{g^T}(\boldsymbol{\beta_k}) \quad \mathbf{(25)}$$

Yet, as we discussed in Sec. 2.1.1, finding the $\boldsymbol{H^{-1}}$ is not always feasible, due to the singularities that may lie in $\boldsymbol{H}$. A way to circumvent this is to replace $\boldsymbol{H^{-1}}$ by a hyperparameter, γ_k, called the *learning rate*, that can either be a constant or change in every iteration. A brief study of the learning rate is presented in Sec. 3.2.

We can now set a pheudocode for the Gradient Descent algorithm as follows:

Algorithm 3: Gradient Descent (GD). The given to β is used for the calculation of the gradient and the parameters β into a loop of iterations.

```
initialize $\boldsymbol{\beta}$
for k in range(iterations):
    $\boldsymbol{g_k^T} = \frac{n}{2} \cdot \boldsymbol{X^T} \cdot (\boldsymbol{X} \cdot \boldsymbol{\beta_k} - \mathbf{y})$
    $\boldsymbol{\beta_{k+1}} = \boldsymbol{\beta_k} - \gamma_k \cdot \boldsymbol{g_k^T}$
```

However, it is more relevant to mention here some of the restrictions that exist in choosing it. One can actually show that the optimum value of γ_k needs to be lower than $\frac{2}{\lambda_{max}}$, where λ_{max} is the largest eigenvalue of the Hessian matrix. A γ_k much lower than this value is computationally expensive, since then a large number of iterations is needed to reach the minimum (Fig.3.2 **A**). Yet, if the value of γ_k is higher that the optimum value, the solution can either oscillate around the minimum (Fig.3.2 **C**) or even diverge (Fig.3.2 **D**). In practice, γ_k is usually set to change with each iteration, according to a learning schedule. More about the γ_k schedulers in Sec. 3.2.1.

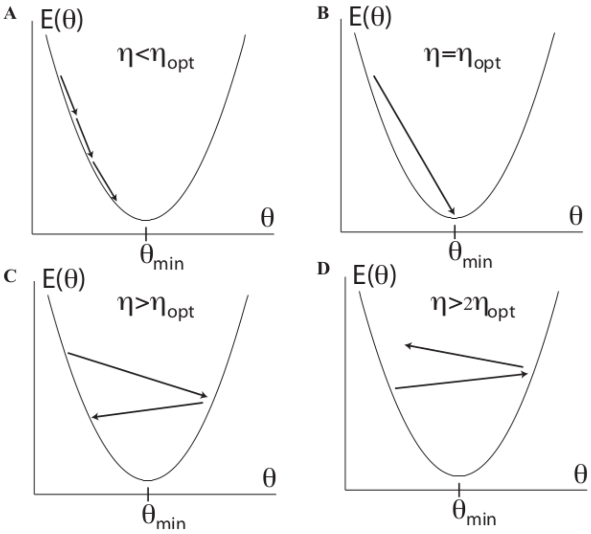


Figure 14: The effects of different values of the learning rate (η on the graph) on solution convergence, borrowed from [Mehta et al., 2019].

3.1.2 The Stochastic Gradient Descent (SGD)

It is clear that the GD method is sensitive to both the initial value of β_k (if the initial guess is not good, we might get trapped into a *local* minimum) and the values of the learning rate. A way to overpass these dependencies it to introduce randomness into the algorithm; this can be achieved by using Stochastic Gradient Descent to calculate $\boldsymbol{\beta}$.

In this method, the dataset is divided into smaller subsets, called *minibatches*. The gradient is calculated exactly as in GD within each minibatch. Each gradient approximation calculated for each minibatch is used to update the parameter β of the next minibatch. A full iteration over all of the minibatches is called an *epoch*. A pheudocode for the SGD algorithm can be written as:

Algorithm 4: Stochastic Gradient Descent (SGD). The dataset of size n, is divided into M, almost equally-sized, minibatches of size $m = \frac{n}{M}$. The gradient and β are calculated for each minibatch and are used as initial values for the first minibatch of the next epoch.

```
initialize $\boldsymbol{\beta_{k,i}}$
for k in range(epochs):
    for i in range (m):
        $\boldsymbol{g_i^T} = \frac{n}{2} \cdot \boldsymbol{X^T} \cdot (\boldsymbol{X} \cdot \boldsymbol{\beta_{k,i}} - \mathbf{y})$
        $\boldsymbol{\beta_{k,i+1}} = \boldsymbol{\beta_{k,i}} - \gamma_k \cdot \boldsymbol{g_i^T}$
    end loop
    $\boldsymbol{\beta_{k+1,i}} = \boldsymbol{\beta_{k,m}}$
end loop
$\boldsymbol{\beta} = \boldsymbol{\beta_{last\_epoch,m}}$
```

The size of the mini-batches is a hyperparameter but it is not very common to cross-validate or bootstrap it. It is usually based on memory constraints (if any), or set to some value, e.g. 32, 64 or 128. We use powers of 2 in practice because many vectorized operation implementations work faster when their inputs are sized in powers of 2 (Hjorth-Jensen).

By distributing the datapoints into the minibatches randomly, we introduce stochasticity into the Gradient Descent algorithm. Thus, the chance of getting trapped in isolated local minima while fitting the algorithm, decreases. Also, calculations are significantly been sped-up since the gradients are being calculated, using smaller datasets (the minibatches) during each epoch.

3.1.3 Adding momentum

Adding "momentum" (or inertia) to the GD or SGD serves as a memory of the direction we are moving in the parameter space. This can be implemented in the gradient calculation as follows:

Algorithm 6: Adding momentum. A momentum factor, $\delta | 0 \leq \delta \leq 1$ is being multiplied to the difference (change) $\Delta \boldsymbol{\beta_{k-1}} = \boldsymbol{\beta_k} - \boldsymbol{\beta_{k-1}}$. This product is added to the previous gradient term, in order to calculate the new difference $\Delta \boldsymbol{\beta_k} = \boldsymbol{\beta_{k+1}} - \boldsymbol{\beta_k}$.

```
for k in range(iterations):
    $\Delta \boldsymbol{\beta_k} = \delta \cdot \Delta \boldsymbol{\beta_{k-1}} + \gamma_k \cdot \boldsymbol{g_k^T}$
    $\boldsymbol{\beta_{k+1}} = \boldsymbol{\beta_k} - \Delta \boldsymbol{\beta_k}$
```

Obviously, for $\delta = 0$ we are back into the (Stochastic) Gradient case.

Adding momentum to the Gradient Descent method helps the algorithm gain speed in directions with persistent but small gradients, even in the presence of stochasticity, while suppressing oscillations in high-curvature directions. This comes in handy when the surface of the MSE is shallow and flat in some directions and narrow and steep in others ([Mehta et al., 2019]).

3.2 Tuning the learning rate, γ

As we discussed above, coming up with a learning rate is a way to circumvent the computation of the Hessian matrix, $\boldsymbol{H}$. There are many recipes for tuning the learning rate for (Stochastic) Gradient Descent. Keeping it constant and running the entire algorithm for different values of γ is a very common brute-force way of picking up a good value for it. But there also exist some more complicated recipes, from simply setting it to adapt linearly or exponentially, according to the number of iterations to tracking not only the gradient, but also the second moment of the gradient. Some of the most popular techniques are outlined in the sections that follow.

3.2.1 The Linear and exponential scedulers

One of the simplest methods for adapting the learning rate with the number of iterations, is choosing a linear dependency as follows:

$$\gamma_k = (1-\alpha)\gamma_0 + \alpha\gamma_t, \tag{26}$$

where:

- $\alpha = \frac{k}{t}$, k being the number of iterations and t a constant-to-tune.
- γ_t is a constant, proportional to $\frac{\gamma_0}{100}$

The learning rate could also change inversely proportionally to the number of iterations as:

$$\gamma_k = \frac{\gamma_0}{1 + k \cdot \gamma_t} \tag{27}$$

Another popular solution is the learning rate decaying exponentially with the number of iterations as can be seen below:

$$\gamma_k = \gamma_0 \cdot e^{-k\gamma_t} \tag{28}$$

The best scheduler for the learning rate can be decided ad hoc. There is no way to determine a priori whether one technique is going to perform better that another, unless it gets applied to the algorithm. For example, as the reader will see in the Sec.3.3.3, I used an exponentially varying learning rate in plain SGD, as the linear and inversely proportional one were performing very poorly no matter the efforts for tuning their hyperparameters t and γ_0.

The next set of tuning techniques is, however, very interesting. By tracking the second moment of the gradient, these *adaptive* methods create a faster momentum towards convergence. It is exciting to see in Secs. 3.3.5-3.3.7 later on the way all three of these methods affect the SGD algorithm.

3.2.2 The Adaptive Gradient Algorithm (AdaGrad) [Duchi et al., 2011]

Algorithm 7: AdaGrad. The update of β is calculated in terms of an adaptive learning rate with values that change in each iteration(/epoch for SGD).

Requirements:

- an initial learning rate, γ_0.
- an initial guess for the coefficients, β_0.
- a small constant, $\epsilon \propto 10^{-8}$, in order to avoid divisions by zero.

```
for k in range (iterations)
    (SGD sample minibatches + epochs)
    compute gradient, g
    compute the accumulated square gradient:
    $r_{i+1} = r_i + g \odot g$, where $\odot$ (is the Hadamard product)
    compute $\frac{\gamma_0}{\delta + \sqrt{r_{i+1}}} \cdot g$
    update:
    $\beta_{i+1} = \beta_i - \frac{\gamma_0}{\delta + \sqrt{r_{i+1}}} \cdot g$
```

3.2.3 The Root Mean Square propagation (RMS-prop)[Tieleman et al., 2012]

Algorithm 8: RMS-prop. The learning rate is reduced in directions where the norm of the gradient is consistently large. This greatly speeds up the convergence by allowing us to use a larger learning rate for flat directions (Hjorth-Jensen).

Requirements:

- a global learning rate, $\gamma \propto 10^{-3}$.
- a decay rate, $\rho \propto 0.9$, controlling controls the averaging time of the second moment.
- an initial guess for the coefficients, β_0.
- a small $\epsilon \propto 10^{-8}$ constant for numerical stability.
- an initial value for the accumulate sq. gradient: r=0.

```
for k in range (iterations)
    (SGD sample minibatches + epochs)
    compute gradient, g
    compute the accumulated square gradient:
    $r_{i+1} = \rho \cdot r_i + (1 - \rho) \cdot g \odot g$
    compute the parameter:
    $\Delta\beta = -\frac{\gamma}{\delta+\sqrt{r_{i+1}}} \odot g$
    update:
    $\beta_{i+1} = \beta_i + \Delta\beta$
```

3.2.4 The ADAptive Moment estimation (ADAM) [Kingma and Ba, 2017]

Algorithm 9: ADAM. A running average of both the first and second moment of the gradient is kept and used to adaptively change the learning rate for different parameters. An additional bias correction is also performed to account for the fact that the first two moments of the gradient are estimated using a running average (Hjorth-Jensen).

Requirements:

- an initial learning rate, $\gamma_0 \propto 10^{-3}$.
- 2 decay rates, $\rho_1 \propto 0.9, \rho_2 \propto 0.999$.
- an initial guess for the coefficients, β_0.
- a small $\epsilon \propto 10^{-8}$ constant for numerical stability.
- an initial value for the accumulate sq. gradients: s, r=0.
- an initial value for the timestep counter: t=0.

```
for k in range (iterations)
    (SGD sample minibatches + epochs)
    compute gradient, g
    $t \leftarrow t + 1$ update timestep
    update the first momentum:
    $s_{i+1} = \rho_1 \cdot s_i + (1 - \rho_1) \cdot g$
    update the second momentum:
    $r_{i+1} = \rho_2 \cdot r_i + (1 - \rho_2) \cdot g \odot g$
    correct bias in the 1st momentum:
    $s_{i+1} = \frac{s_i}{1-\rho_1^t}$
    correct bias in the 2nd momentum:
    $r_{i+1} = \frac{r_i}{1-\rho_2^t}$
    compute the update:
    $\Delta\beta = -\frac{\gamma_0 \cdot s}{\delta+\sqrt{r_{i+1}}}$
```

update:

$\beta_{i+1} = \beta_i + \Delta\beta$

3.3 Results

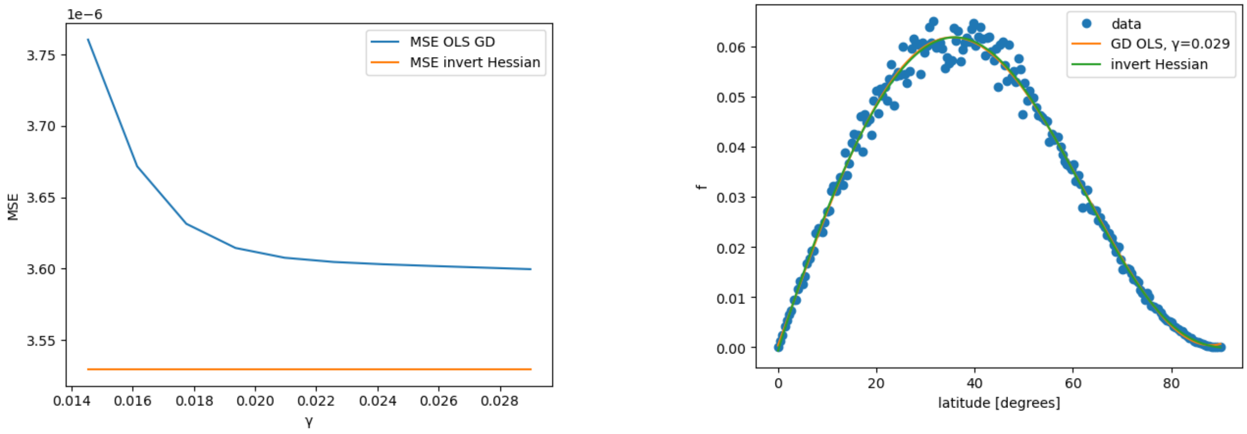
It is now time to try applying the techniques mentioned above to the dataset in question. Adding momentum to both GD and SGD, coupling these two techniques with various learning rate scedulers and applying everything to both OLS and Ridge regression, resulted in 24 methods overall. The fitting behaviour of each one of them can be studied in the sections that follow.

3.3.1 Plain Gradient Descent for OLS and Ridge

Starting simple, I added a GD algorithm to the OLS and Ridge regression with and without momentum. The learning rate was kept constant but was varied in the range $[\frac{1}{\lambda_{max}}, 0.029]$, where λ_{max} is the maximum eigenvalue of the Hessian matrix. The hyperparameter λ of Ridge regression (not to be confused with the eigenvalues of the Hessian matrix, also denoted by λ) was varied in the range $[10^{-4}, 1]$. For larger values of λ, calculations overflowed, while for lower ones the case was thought to be too similar to OLS.

OLS regression: comparison with the theoretical solution

The model I am trying to fit to the dataset, being a 6th degree polynomial, is fairly simple. Therefore it would be interesting to calculate, at least once, the predicted values of $\mathbf{y}$, using the theoretical solution, that is the inversion the Hessian matrix.



(a) MSE calculated for both GD OLS and analytically.

(b) The resulting model for both GD OLS and Hessian matrix inversion.

Figure 15: The application of GD to OLS.

Because the hyperparameter, γ is used, as we discussed previously, to avoid the computation of the inverse of $\boldsymbol{H}$, the analytical solution does not include this hyperparameter at all. Therefore, the MSE is independent of γ, hence its shooting directly to its minimum value in Fig.15a. Calculating the MSE using plain GD, though, involves γ which, as described above, was varied within a grid of values but kept constant during each loop of calculations. From Fig.15a the MSE decreases with an increasing γ. However, after a certain value of γ (approx. 0.022) the MSE stabilizes. For the optimal $\mathbf{y}$ of the model, its last value ($\gamma = 0.029$) was used. For values greater than that the calculations overflow.

Judging from Fig. (15b), the simple choice of using plain GD in OLS for a fixed learning rate approaches the data rather well, having a MSE of the order of 10^{-6}.

Ridge regression: comparison with the OLS regression

Although computationally more efficient than the OLS case ($2.575 \cdot 10^{-5}$ s vs $3.0756 \cdot 10^{-5}$ s) and inspite the fact that less iterations are required (10^5 for Ridge vs 10^6 for OLS), the extra loop for the Ridge parameter, λ, adds a lot to the overall computational cost, since an investigation of that value has to be made always.

Additionally, the application of GD to Ridge regression is slightly less reliable. The slight curvature at 90° is not ideal, the North Pole, being the main area of interest for studying Arctic Amplification, caused by meridional heat transport. However, this does not appear in every run (due to the randomness of the noise in the dataset) and the rest of the data are being approached rather good.

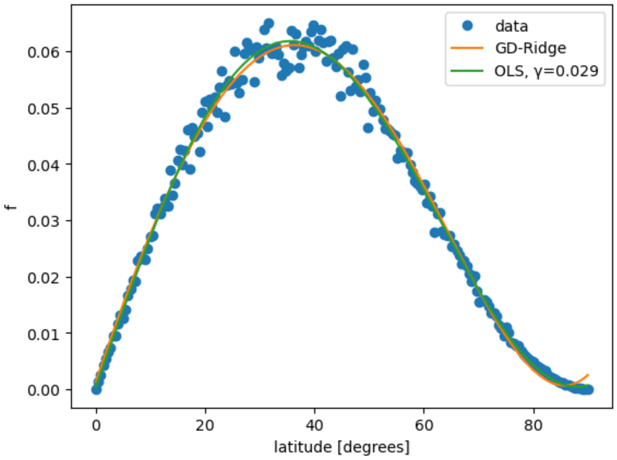


Figure 16: The final model from Ridge regression, fit to the dataset, compared with the OLS solution.

Studying the MSE of this method is rather interesting. Because it depends now on both the Ridge parameter, λ, and the hyperparameter, γ, the results need to be presented in a table form. Scikit-learn's package *seaborn* is rather useful for cases like this. The color code used in this package enables us to see the dependency of MSE on both parameters very easily. Thus, it is pretty obvious that MSE increases with increasing values of λ but decreases as γ increases, until it reaches a plateau, similar to the one in the OLS case (Fig.15a), where it remains stable for increasing γ. In order to avoid extensive computation time, a "safety band" of λ and γ can be picked up from this table, in order to narrow down the investigation area.

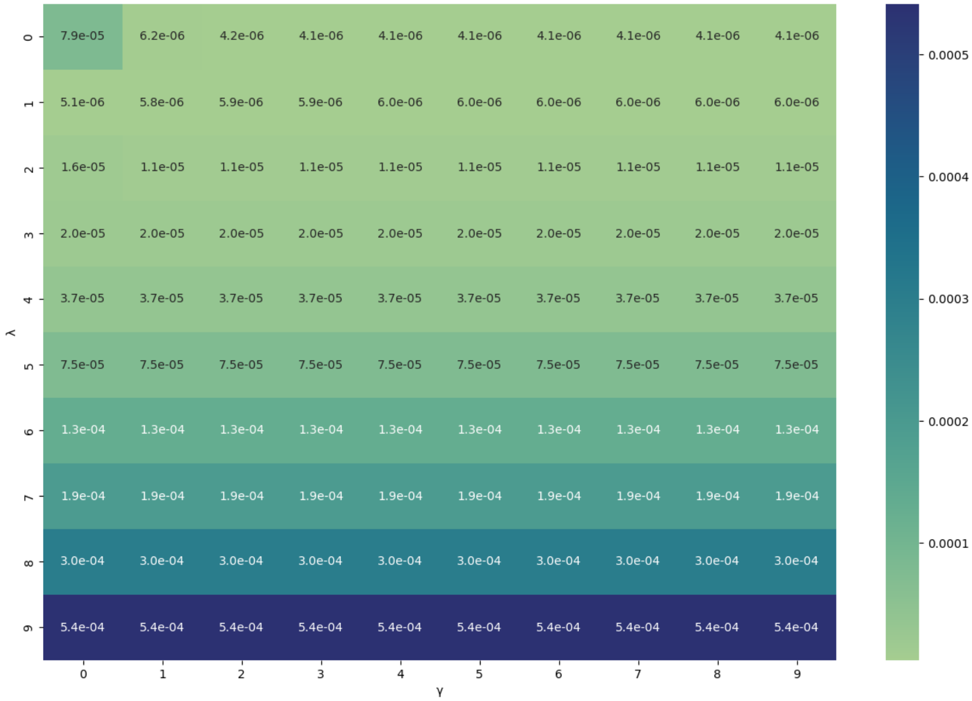


Figure 17: The MSE for GD in Ridge regression, calculated for every combination of γ and λ. Again, a lot of care should be taken when studying this figure: the numbers on *both* axes denote the *numbering* of the parameters, 0 standing for the first parameter and so on. So, for example, the 0th value of γ is $\frac{1}{\lambda_{max}}$ and the 9nth one is 0.029.

3.3.2 Gradient Descent with momentum (GD-m) for OLS and Ridge.

The spectrum for the hyperparameter γ is now $[\frac{1}{\lambda_{max}}, 0.03]$, while for the Ridge parameter, λ, it is $[10^{-5}, 1]$.

Oddly enough, adding momentum to the Gradient descent proved computationally more expensive for both OLS ($6.438 \cdot 10^{-5}$ s vs $3.076 \cdot 10^{-5}$ s) and Ridge Regression ($2.838 \cdot 10^{-5}$ s vs $2.575 \cdot 10^{-5}$ s). Yet, as can be seen from both Fig. 18 and Fig. 19 the MSE of the model is lower than in the plain GD case, although it varied with γ and λ exactly as before.

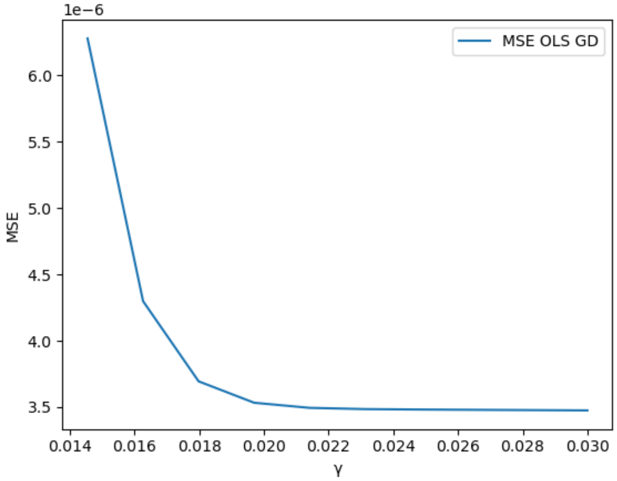


Figure 18: The MSE of GD-m, for OLS regression against the γ hyperparameter.

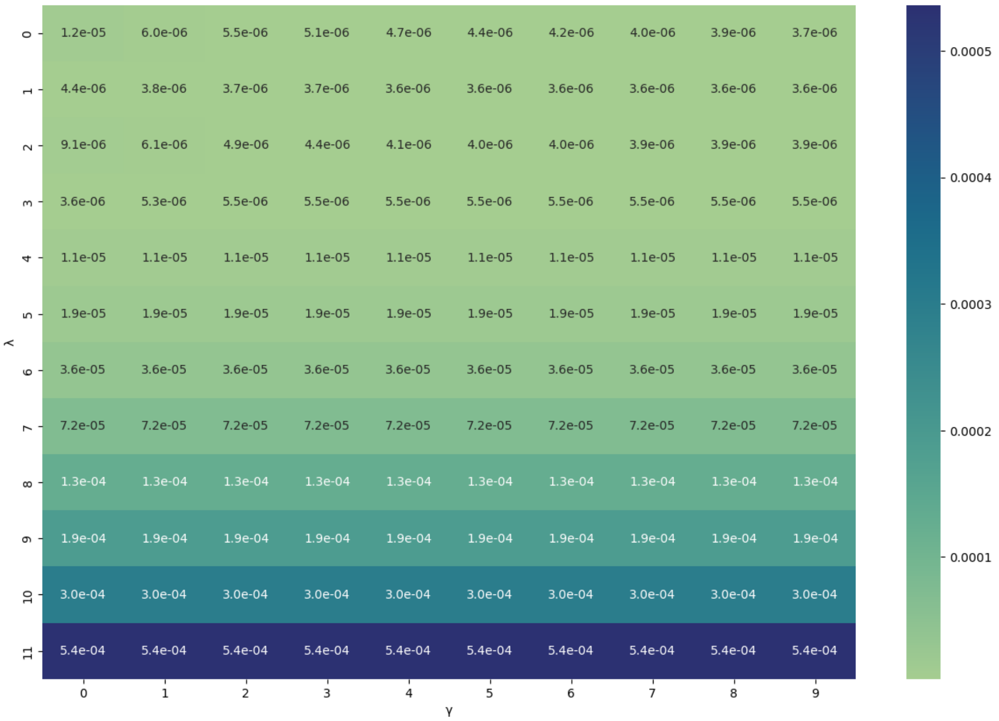


Figure 19: The MSE for GD-m in Ridge regression, calculated for every combination of γ and λ. Again, a lot of care should be taken when studying this figure: the numbers on *both* axes denote the *numbering* of the parameters, 0 standing for the first parameter and so on. So, for example, the 0th value of γ is $\frac{1}{\lambda_{max}}$ and the 9nth one is 0.03.

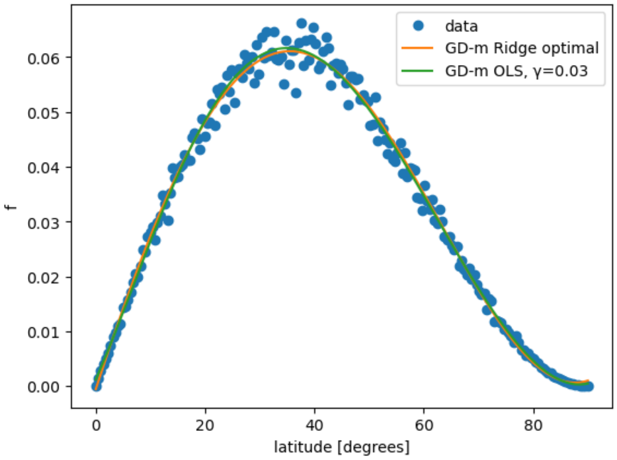


Figure 20: The model for GD-m on OLS and Ridge regression, fit to the dataset.

3.3.3 Plain Stochastic Gradient Descent (SGD) for OLS and Ridge

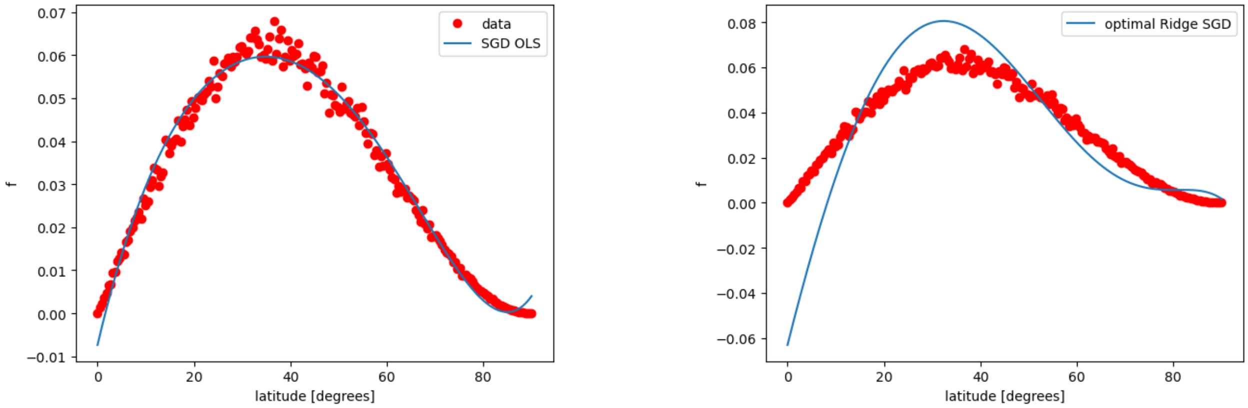
This time, an exponential scheduler was used for tuning the hyperparameter, γ, following the relationship **(28)** with k= number of epochs and $\gamma_0 = 0.005$.

The number of minibatches varied in the powers of 2, in order to accelerate the calculations.

Although a large number of epochs (10^6 for OLS and 10^5 for Ridge regression) is needed during these runs, raising it even further has almost no contribution on the quality of the model. A good result (MSE= $7.156 \cdot 10^{-6}$) can be attained from the OLS case, but SGD doesn't perform very well in Ridge regression, raising the order of magnitude of the MSE up to 10^{-3}.

The best results (the ones with the minimum MSE) are presented in Fig.21. As we can see, SGD can potentially work well on OLS regression but this is definitely not the case for Ridge regression.

Studying how the curve changes with the number of minibatches used (see App. C for the OLS case), one can see that it is a definitive factor for the results. In this case, we get the best results for 1 minibatch which is essentially the GD case.



(a) The model calculated with SGD for OLS regression, using M=1 minibatches.

(b) The model calculated with SGD for Ridge regression, using M=1 minibatches.

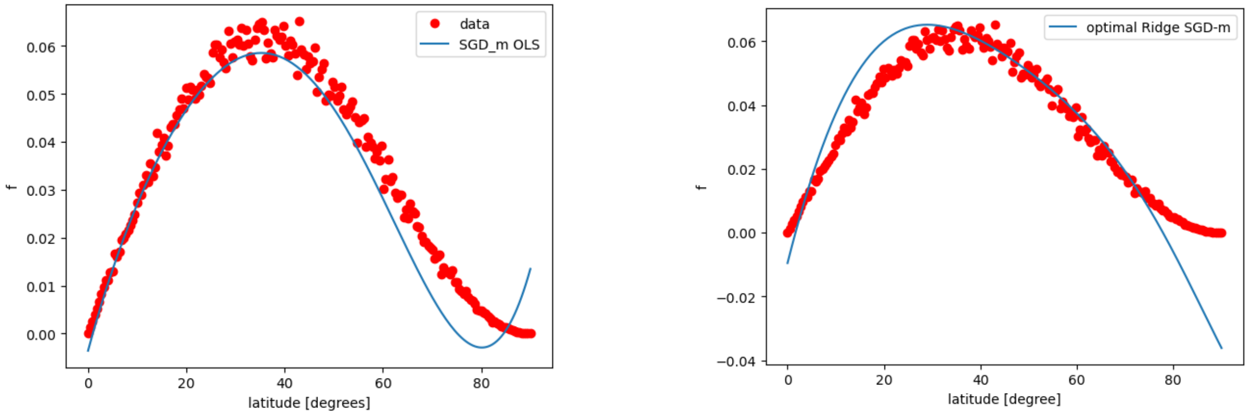
Figure 21: The model calculated with SGD for OLS, using M=1 minibatches.

3.3.4 Stochastic Gradient Descent with momentum (SGD-m) for OLS and Ridge

Varying the hyperparameters γ and λ exactly as in the case of plain SGD and also trying different values for the minibatches but always in powers of two, I added momentum to the SGD algorithm of the OLS and

Ridge regression. Both efforts proved unsuccessful as the MSE is inevitably high, reaching up to $1.126 \cdot 10^{-5}$ in OLS case (1 order of magnitude higher than before) and again up to 10^{-3} in Ridge regression.

The number of minibatches is here also an important factor for the quality of the model. This time, however, we get the best results for M=16 and these are plotted in Figs. 22.



(a) The model calculated with SGD-m for OLS regression, using M=16 minibatches.

(b) The model calculated with SGD for Ridge regression, using M=1 minibatches.

Figure 22: The model calculated with SGD-m for OLS, using M=16 minibatches.

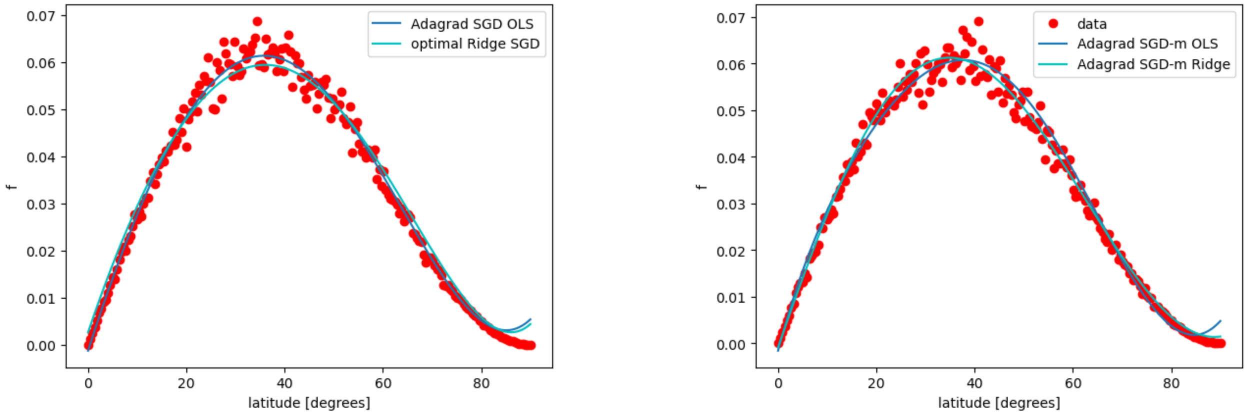
Overall, applying SGD to OLS and Ridge regression either with or without momentum led to very unsuccessful results. But there are still have a few more options to try out! In the following sections the 3 adaptive methods for tuning the learning rate that we discussed earlier (Sec.3.2.2) are being applied to SGD with and without momentum, in an effort to upgrade the model's accuracy.

The AdaGrad method, in particular, was also applied to plain GD and momentum GD, for curiosity's sake. However, because GD performed rather well from the beginning and hence has no need for improvement, this investigation is laid out in Appendix D.

3.3.5 Adagrad Stochastic Gradient Descent with and without momentum for OLS and Ridge

Can the AdaGrad method improve the results for SGD on OLS and Ridge Regression?

Varying the initial value of γ in the range $[10^{-3}, 0.1]$ and λ in the range $[10^{-5}, 0.1]$, we get the best results for M=1 minibatches, falling again into the category of GD. Yet this time, adjusting the learning rate after every iteration according to the AdaGrad algorithm yields much better results than before (Figs.23).



(a) The model for SGD in OLS and Ridge regression, using the AdaGrad method to tune the learning rate, γ.

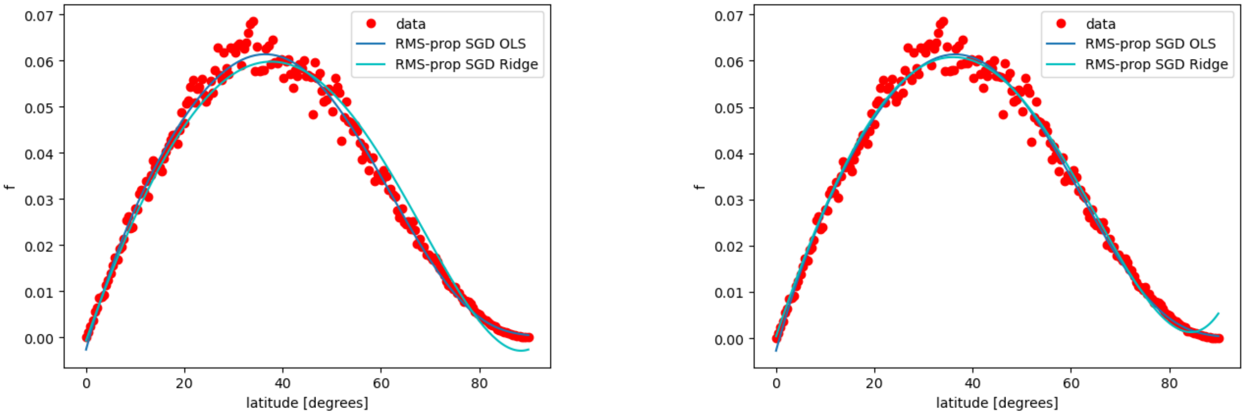
(b) The model for SGD-m in OLS and Ridge regression, using the AdaGrad method to tune the learning rate, γ.

Figure 23: Adagrad for tuning the learning rate, γ in SGD, SGD-m.

In both SGD and SGD-m cases, there is a slight overestimation at the North Pole and an underestimation of the data around the curve's maximum (35°- 40°), so AdaGrad might not be the best option for fitting the present dataset. The effect of the method on the quality of the curve is, however, very impressive and it would be very interesting to see if it can be better improved using the rest of the adaptive techniques for tuning the learning rate.

3.3.6 RMS-prop on SGD with and without momentum

Varying the hyperparameters γ and λ exactly as in the AdaGrad case, we get good enough results for both M=1 and M=2 minibatches (Figs. 24) for the plain SGD case. Although there is still some underestimation around 35°- 40°, the OLS case performs very well at the North Pole. Ridge regression, however can either underestimate (Fig. 24a) or overestimate (Fig. 24a) the meridional heat transport at this region.



(a) M=1 minibatch.

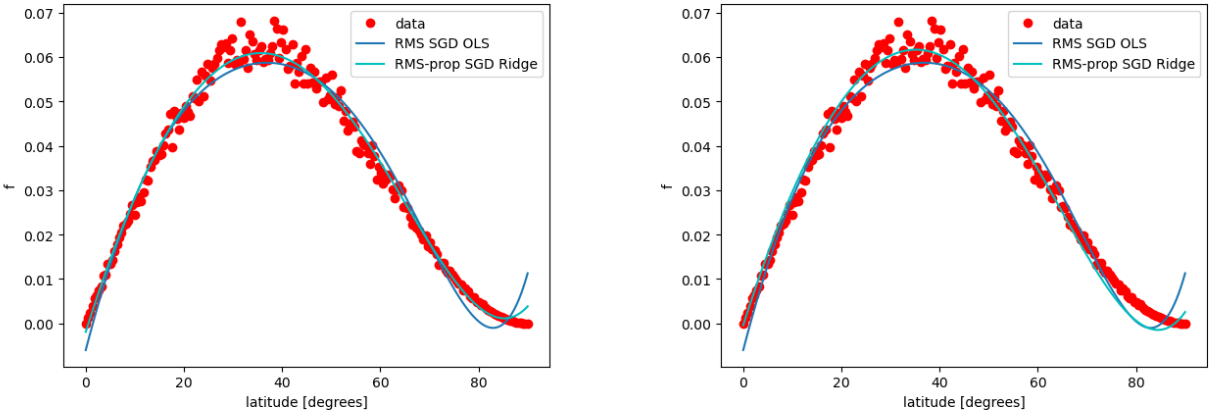
(b) M=2 minibatches.

Figure 24: The model for RMS-prop for SGD on OLS and Ridge regression. Note that the Ridge regression resluts are in both plots compared to the OLS solution for M=1 minibatch.

Adding momentum to the SGD algorithm enables us to work with a larger number of minibatches. From Figs.25 one can see that we get the best results for Ridge regression for M=4 minibatches, at least close to the North Pole.

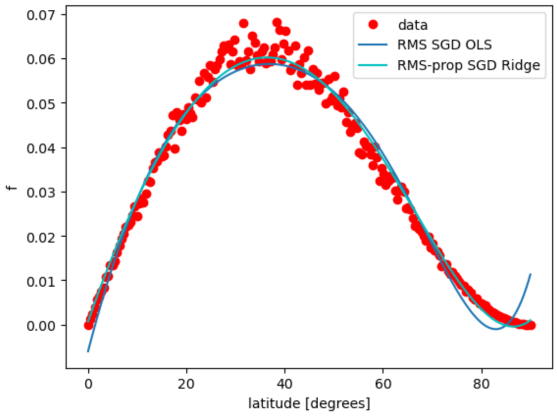
In all three cases the Ridge regression results are compared to the OLS ones for M=1. Although this was the case with the minimum MSE, we can clearly see that it underestimated the maximum values of f and does not give a good prediction for its values close to the North Pole.

This time it is Ridge regression that performs better but only in comparison to OLS, since the underestimation at the curve's maximum is still apparent.



(a) M=1 minibatch.

(b) M=2 minibatches.



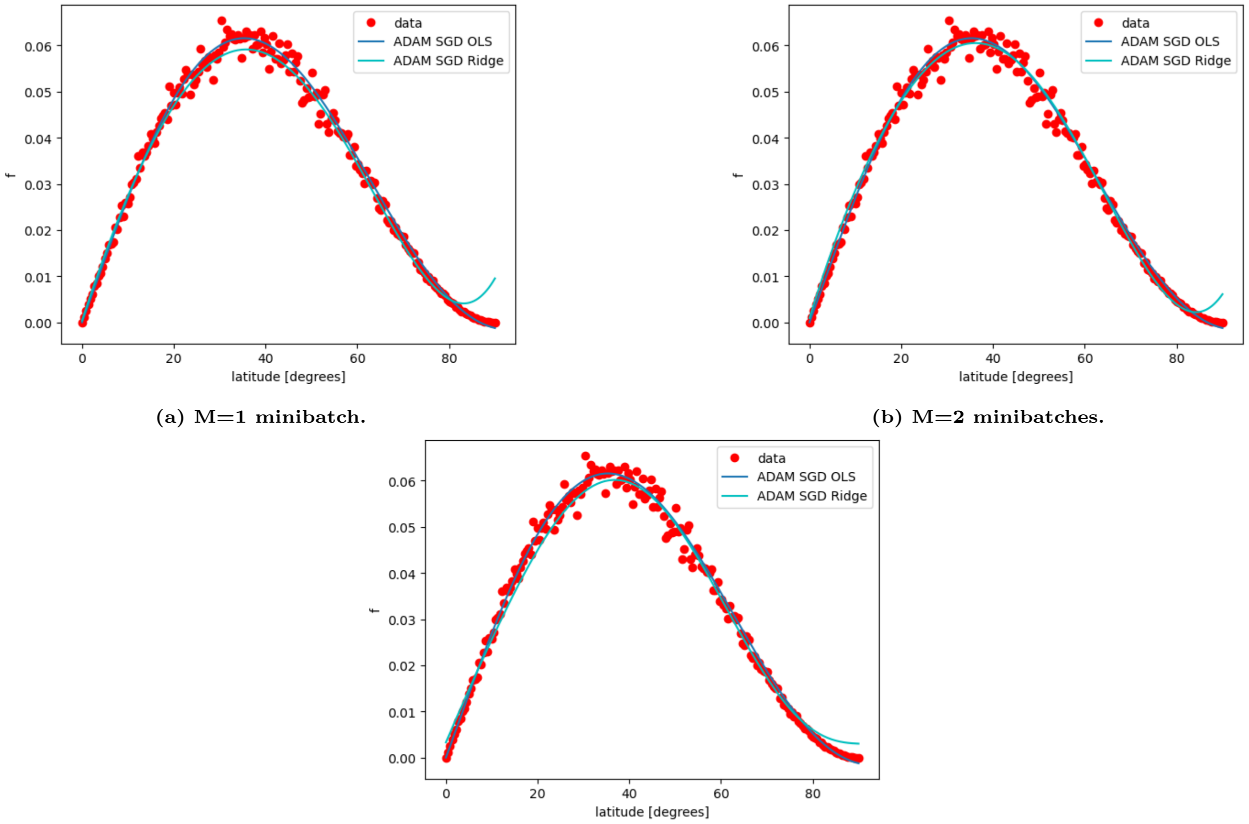
(c) M=4 minibatches.

Figure 25: The model for RMS-prop for SGD-m on OLS and Ridge regression. Only the case of M=1 minibatch is being plotted for the OLS regression in all 3 plots.

3.3.7 ADAM on SGD with and without momentum

The ADAM method is now applied to plain SGD for OLS and Ridge regression, varying the hyperparameters in the same ranges as before.

In Figs.26, the OLS case for 2 minibatches is plotted against the Ridge regression case for M=1, 2 and 4 minibatches. As can be seen, the OLS case fits the dataset better that the Ridge regression, the latter still underestimating the heat flux values in the maximum of the curve and overestimating its values at the North Pole. Yet, for M=4 minibatches this overestimation if far less prominent than it is for M=1, 2.



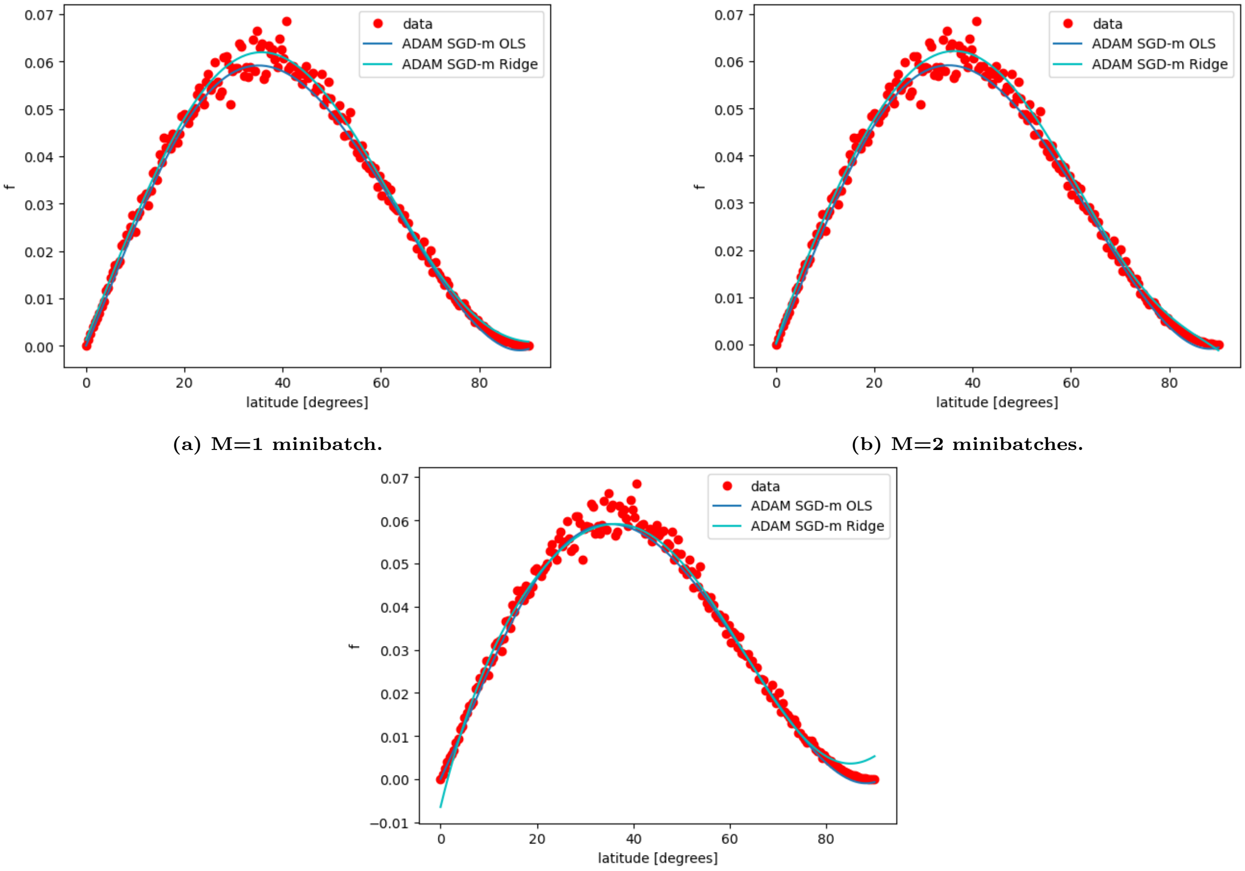
(a) M=1 minibatch.

(b) M=2 minibatches.

(c) M=4 minibatches.

Figure 26: The model for ADAM for SGD on OLS and Ridge regression.

Applying the same tuning method to momentum SGD favours Ridge regression just as RMS-prop on SGD-m did. Studying Figs.27 one can see that the Ridge regression model fits the dataset very well for both M=1 and M=2 minibatches. Unlike the OLS case, it does not underestimate the heat flux values in the maximum of the curve and it fits the North Pole area rather well. The only case where it doesn't perform as well is for M=4 minibatches, where both regression methods underestimate the f-values around 35°- 40° but the OLS regression approaches the North Pole area much better than the Ridge regression model.



(a) M=1 minibatch.

(b) M=2 minibatches.

(c) M=4 minibatches.

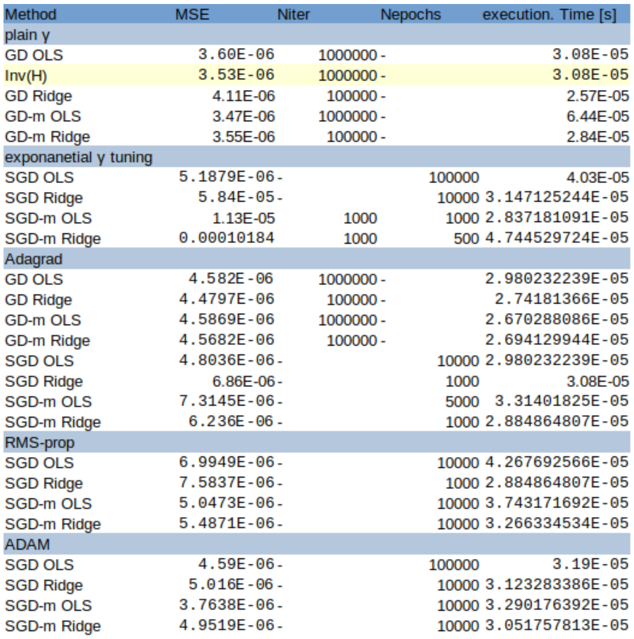
Figure 27: The model for ADAM for SGD-m on OLS and Ridge regression.

4 Discussion - conclusions

A polynomial fit was performed on a dataset, extracted from Stone's equation **(4)** for calculating the meridional heat transport. In order to define the polynomial degree, both Ordinary Least Squares (OLS) and Ridge regression were applied to the dataset, along with bootstrap and Cross-Validation (CV) resampling. Although the OLS regression could provide a minimum value for the polynomial degree, Ridge regression made it clear that for **pol. degree = 6** we get the lowest value of the Mean Squared Error (MSE).

In order to optimize the polynomial coefficients, 24 optimization techniques were performed on the dataset. Both Gradient Descent (GD) and Stochastic Gradient Descent (SGD) were tried out for OLS and Ridge Regression. The case of adding momentum to these algorithms was also investigated, along with 5 ways of tuning the learning rate. Table 1 provides an outline of all of the combinations of these techniques that were applied, along with the value of the MSE and the execution time they need.

Because inverting the Hessian matrix proved to be a computationally light calculation for this application, the theoretical OLS solution would be the first method to try is a future data analysis. In this case, the polynomial that approaches the dataset has the following expression (see the running on my GitHub page):



Method	MSE	Niter	Nepochs	execution. Time [s]
plain γ				
GD OLS	3.60E-06	1000000	-	3.08E-05
Inv(H)	3.53E-06	1000000	-	3.08E-05
GD Ridge	4.11E-06	100000	-	2.57E-05
GD-m OLS	3.47E-06	1000000	-	6.44E-05
GD-m Ridge	3.55E-06	100000	-	2.84E-05
exponanetial γ tuning				
SGD OLS	5.1879E-06	-	100000	4.03E-05
SGD Ridge	5.84E-05	-	10000	3.147125244E-05
SGD-m OLS	1.13E-05	1000	1000	2.837181091E-05
SGD-m Ridge	0.00010184	1000	500	4.744529724E-05
Adagrad				
GD OLS	4.582E-06	1000000	-	2.980232239E-05
GD Ridge	4.4797E-06	100000	-	2.74181366E-05
GD-m OLS	4.5869E-06	1000000	-	2.670288086E-05
GD-m Ridge	4.5682E-06	100000	-	2.694129944E-05
SGD OLS	4.8036E-06	-	10000	2.980232239E-05
SGD Ridge	6.86E-06	-	1000	3.08E-05
SGD-m OLS	7.3145E-06	-	5000	3.31401825E-05
SGD-m Ridge	6.236E-06	-	1000	2.884864807E-05
RMS-prop				
SGD OLS	6.9949E-06	-	10000	4.267692566E-05
SGD Ridge	7.5837E-06	-	1000	2.884864807E-05
SGD-m OLS	5.0473E-06	-	10000	3.743171692E-05
SGD-m Ridge	5.4871E-06	-	10000	3.266334534E-05
ADAM				
SGD OLS	4.59E-06	-	100000	3.19E-05
SGD Ridge	5.016E-06	-	10000	3.123283386E-05
SGD-m OLS	3.7638E-06	-	10000	3.290176392E-05
SGD-m Ridge	4.9519E-06	-	10000	3.051757813E-05

Table 1: The 24 optimization methods applied to the dataset, along with the calculated MSE and the execution time. Note that both of these values change after each run, due to the added noise. Yet the order of magnitude remains the same.

$$y = \mathbf{2.28959173 \cdot 10^{-5}} + \mathbf{1.59190196 \cdot 10^{-1}} \cdot x + \mathbf{1.00758533 \cdot 10^{-1}} \cdot x^2 \, \mathbf{-2.21102042 \cdot 10^{-1}} \cdot x^3 +$$

$$+\mathbf{4.46013345 \cdot 10^{-2}} \cdot x^4 + \mathbf{7.13140276 \cdot 10^{-2}} \cdot x^5 \, \mathbf{-2.47365356 \cdot 10^{-2}} \cdot x^6, \qquad \mathbf{(29)}$$

where y is the heat flux, f and $x = sin(\phi)$ is the latitude.

The next best option would be, again, the simplest one, that is **GD for OLS** with a constant hyperparameter, γ. The latter was the only hyperparameter to tune and after settling on its optimal value, there is no need for an extra loop of investigation, so the code runs relatively fast (the reader can try in the same part of my GitHub page). Being so close in concept to the theoretical solution (the Hessian matrix basically gets replaced by a constant value, γ) it yields an extremely good fit (MSE of the order of 10^{-6}), inspite its simplicity. The resulting polynomial follows the expression:

$$y = \mathbf{-0.00046354} + \mathbf{0.16718365} \cdot x \, \mathbf{-0.02067232} \cdot x^2 \, \mathbf{-0.1772271} \cdot x^3 +$$

$$+\mathbf{0.02146752} \cdot x^4 + \mathbf{0.07208737} \cdot x^5 \, \mathbf{-0.02290923} \cdot x^6, \qquad \mathbf{(30)}$$

with y being again the heat flux, f and $x = sin(\phi)$ the latitude.

5 Future work

Transforming Stone's eq.**(4)** into a dataset-to-fit was not a necessary step to study the meridional heat transport. His equation, describing it, has been there from the beginning! Modelling this function with a polynomial fit was just an excuse to study some of the most popular optimization methods out there.

Bringing my current research interests into this project took approximately 1 month, so when I was finally ready to proceed I had to move fast. Because of all the interesting questions that arise from studying real datasets, bringing them into this project was a time-wise investment I could not afford at the time. Thus I made up the dataset, described in Sec.1.2.

Yet now, knowing which tools are the best for studying this quantity, I can move to applying them to actual datasets, like the ERA5. Perfoming a new polynomial fit, using Ridge regression to assess its degree and

GD on OLS regression to optimize its coefficients will be an interesting way to test the robustness of this research. And, this time, data from the Southern Hemisphere could also be used. Plus, Stone's equation, in fact describes, the *annually averaged* meridional heat transport. Could we use these datasets to predict a *seasonally varying* climate? Will the final model compare well with other climate simulations like in NorESM?

Studying machine learning techniques has been a good start in understanding how to address those questions and how to deal with datasets in order to move towards answering them.

Appendices

A Singular Value decomposition for finding the polynomial coefficients, $\boldsymbol{\beta}$ in Ordinary Least Squares Regression

We proved in Sec.2.1.1 that the polynomial coefficients in OLS Regrassion are given by the expression:

$$\boldsymbol{\beta} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y} \tag{31}$$

, where $\mathbf{X}$ is the design matrix and $\mathbf{y}$ the initial datapoints.

Given that every matrix can be singular value- decomposed [Sudipto Banerjee, 2014], we can give $\mathbf{X}$ the following expression:

$$\mathbf{X} = \boldsymbol{U} \cdot \boldsymbol{\Sigma^T}, \tag{32}$$

where:

- $\mathbf{U}$ is an orthogonal matrix and therefore it holds that: $\boldsymbol{U^T \cdot U = I}$
- $\mathbf{V}$ is also an orthogonal matrix and so: $\boldsymbol{V^T \cdot V = I}$.

  We will also use the definition of an orthogonal matrix, that is: $\boldsymbol{V^T = V^{-1}}$
- $\boldsymbol{\Sigma}$ is a diagonal matrix, containing the *singular values* of $\mathbf{X}$.

Substituting **(32)** into **(31)**, we get:

$$\begin{aligned}
&\boldsymbol{\beta = (V \cdot \Sigma^T \cdot \Sigma \cdot (V^T)^{-1} \cdot V \cdot \Sigma^T \cdot U^T} = \\
&\quad = \boldsymbol{(V^T)^{-1} \cdot \Sigma^{-1} \cdot (\Sigma^T)^{-1} \cdot V^{-1} \cdot V \cdot \Sigma^T \cdot U^T} = \\
&\quad = \boldsymbol{(V^T)^{-1} \cdot \Sigma^{-1} \cdot U^T} \Rightarrow \\
&\quad \Rightarrow \boldsymbol{\beta = V \cdot \Sigma^{-1} \cdot U^T}
\end{aligned} \tag{33}$$

, given the orthogonal matrix properties stated above.

Now, in the above expression, $\mathbf{V}$ is known $\boldsymbol{\Sigma^{-1}}$ is easily computed, since $\boldsymbol{\Sigma}$ is diagonal and $\boldsymbol{U^T}$ is also easy to calculate.

Therefore, the coefficients, $\boldsymbol{\beta}$ of the polynomial can be calculated, avoiding any problems raised by potential singularities in the design matrix, $\mathbf{X}$.

B The score function, R^2 for the plain OLS case

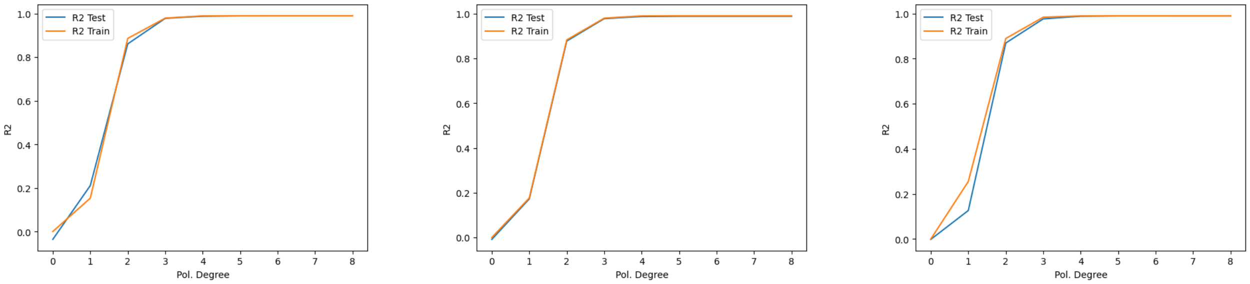
The correlation (score) function is defined as follows ([Devore, 2011], Ch.12):

$$R^2(\mathbf{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y_i})^2}{\sum_{i=0}^{n-1}(y_i - \bar{y_i})^2}, \tag{34}$$

where $\bar{y_i} = \frac{1}{n}\sum_{i=0}^{n-1} y_i$ is the mean value of $\mathbf{y}$.

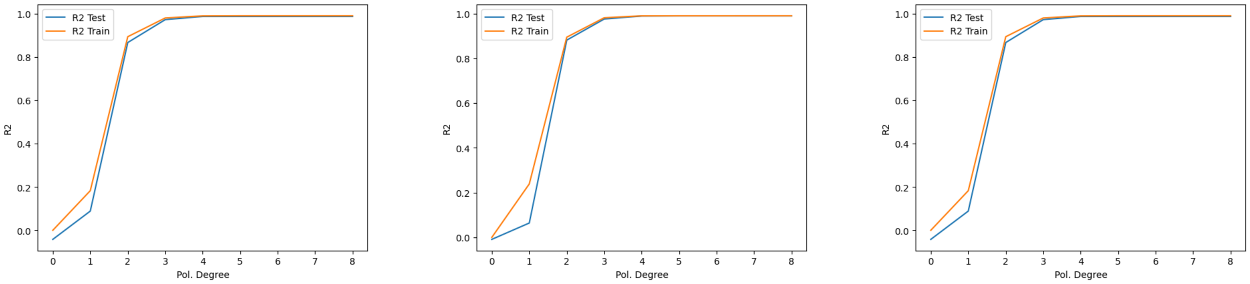
Moving in exactly the same way as in Sec.2.4.1, I varied the number of datapoints (n=200, 500 or 1000) and the size of the test dataset (25, 50 or 75% of the original dataset).

The results can be seen in Figs.28- 30. A high correlation (close to 1) is always achieved for polynomial degrees larger than 3, meaning that from that polynomial degree and on, the model can predict the data almost exactly. This conclusion matches with what we saw in Sec.2.4.1, that the minimum difference between train and test MSE can be found for polynomial degrees larger than 3.



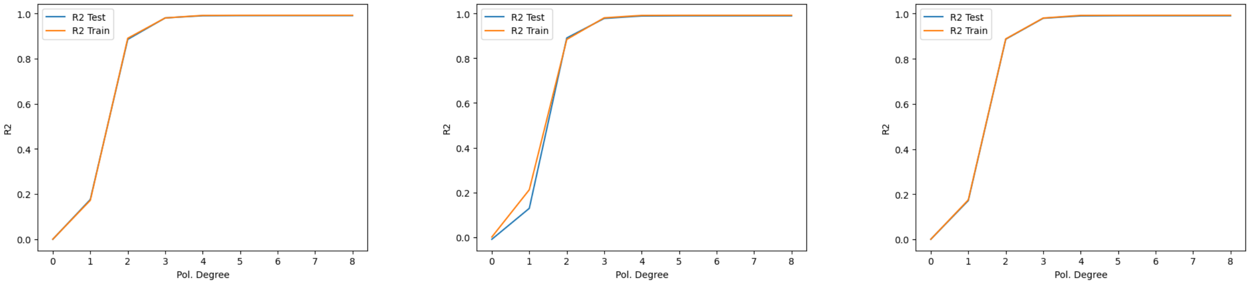
(a) **n=200, test size = 25%.** (b) **n=200, test size = 50%.** (c) **n=200, test size = 75%.**

Figure 28: R^2 for for the plain OLS case with n=200 datapoints.



(a) **n=500, test size = 25%.** (b) **n=500, test size = 50%.** (c) **n=500, test size = 75%.**

Figure 29: R^2 for the plain OLS case with n=500 datapoints.

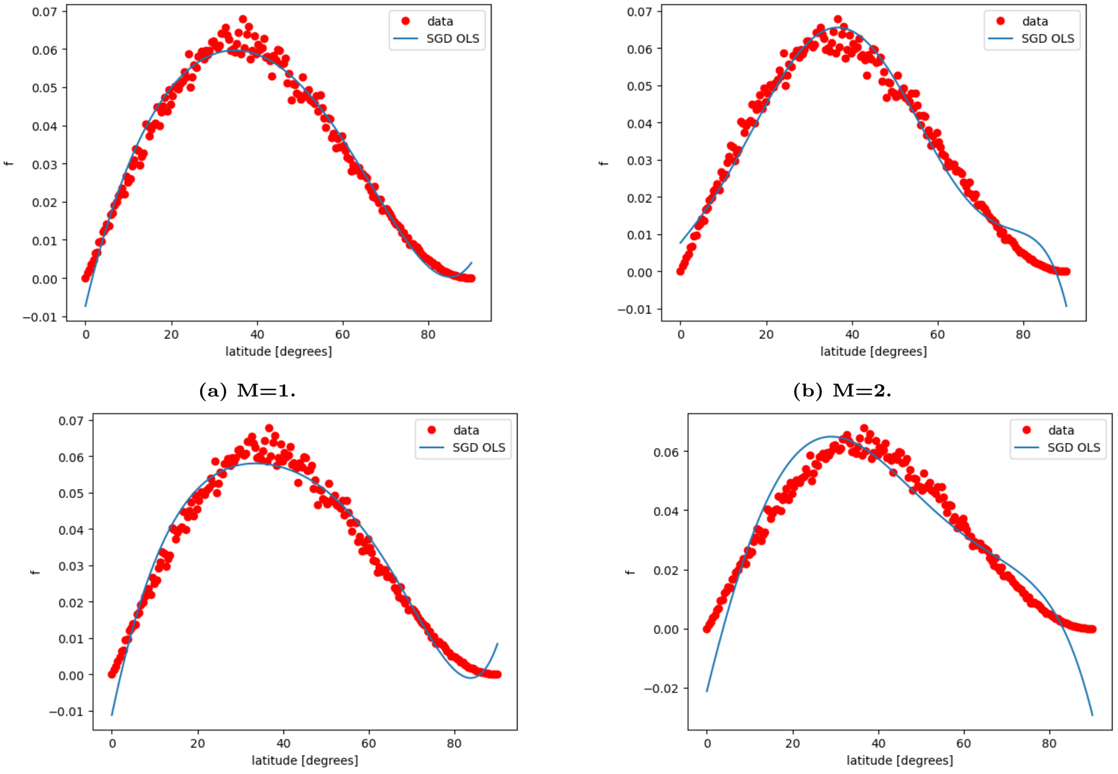


(a) **n=1000, test size = 25%.** (b) **n=1000, test size = 50%.** (c) **n=1000, test size = 75%.**

Figure 30: R^2 for the plain OLS case with n=1000 datapoints.

C Study of the model's dependency on the number of minibatches in SGD.

The number of minibatches is varied in the case of plain SGD on OLS regression (Sec.3.3.3). As can be seen from Figs.31, the number of minibatches has to be very low (we get the best fit for M=1 minibatch) in order for the model to perform well. Raising the number of minibatches leads to progressively worse fittings, even for $M > 32$ (not shown here). So we see that in order for plain SGD to perform well on this dataset, it has to be as close to GD as possible (the case where M=1 considers the entire test dataset as a minibatch, so the algorithm goes through the entire test dataset as before). A high dependency of the result on the number of minibatces was observed through every coded SGD algorithm.



(a) M=1.

(b) M=2.

(c) M=4.

(d) M=32.

Figure 31: The effect of the number of minibatches, M, on the quality of the model.

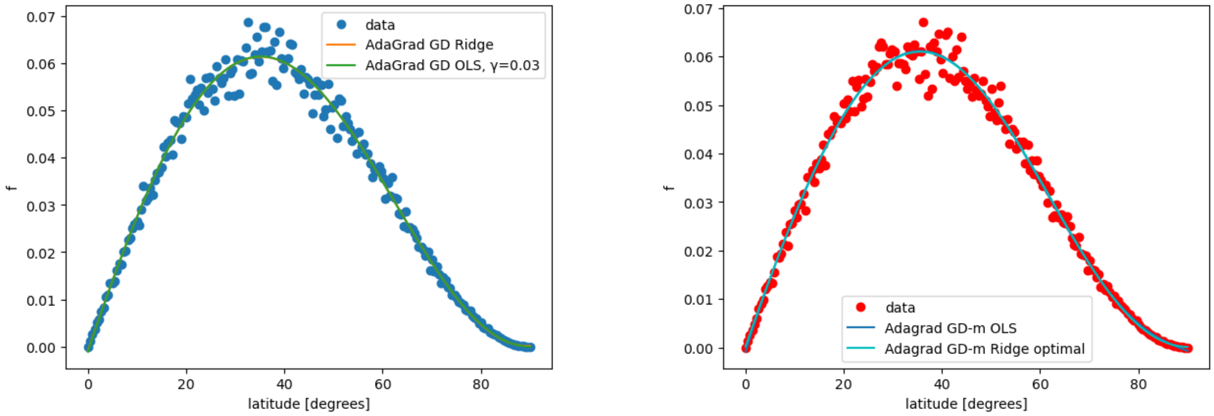
D Adagrad Gradient Descent with and without momentum for OLS and Ridge

Gradient Descent already performed very well using a constant learning rate. However, it is still interesting to see how it behaves with an adaptive learning method instead. The AdaGrad method is implemented on both GD and GD-m for OLS and Ridge regression. Although the learning rate now changes after every iteration, its initial value, γ_0, was varied again within the range $[\frac{1}{\lambda_{max}}, 0.03]$ to decide upon a good initialization.

Upon running the code (for plain GD and GD-m), one can see that for both cases (with and without momentum), the results are very similar to the ones where the learning rate was constant in each iteration. The MSE behaves exactly the same as in the previous GD cases, being independent of γ after a certain value for OLS; for Ridge regression it still gets raised for larger values of λ and drops for larger values of γ.

There is an excellent agreement between the OLS and Ridge regression curves, both being almost indistinguishable from each other (Fig.32a and Fig.32b).

It is important to note at this point, that although this method performs very well, it demands no less number of iterations (10^6 for OLS and 10^5 for Ridge) than the case where the learning rate is kept constant. Lowering the number of iterations affects the results dramatically, unlike the number of epochs in SGD. However, since the MSE stabilizes after a vale of γ_0, we can run the same algorithm, dropping the investigation over the γ_0 initialization and setting it equal to, say 0.03 (the last value of its spectrum) from the start. The number of iterations will not change but at least there would be one less loop to go through and therefore the overall computational time would decrease.



(a) The model for GD in OLS and Ridge regression, using the AdaGrad method to tune the learning rate, γ.

(b) The model for GD-m in OLS and Ridge regression, using the AdaGrad method to tune the learning rate, γ.

Figure 32: Adagrad for tuning the learning rate, γ in GD, GD-m.

References

[Celisse, 2014] Celisse, A. (2014). Optimal cross-validation in density estimation with the L^2-loss. *The Annals of Statistics*, 42(5):1879 – 1910.

[Devore, 2011] Devore, J. L. (2011). *Probability and Statistics for Engineering and the Sciences, 8th Edition.* Cengage Learning, Boston.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159.

[Efron, 1979] Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1 – 26.

[Ellis and Haar, 1976] Ellis, J. S. and Haar, T. H. V. (1976). Zonal average earth radiation budget measurements from satellites for climate studies.

[GEISSER, 1974] GEISSER, S. (1974). A predictive approach to the random effect model. *Biometrika*, 61(1):101–107.

[Geisser, 1975] Geisser, S. (1975). The predictive sample reuse method with applications. *Journal of the American Statistical Association*, 70(350):320–328.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics).*

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[Mehta et al., 2019] Mehta, P., Bukov, M., Wang, C.-H., Day, A. G., Richardson, C., Fisher, C. K., and Schwab, D. J. (2019). A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124.

[Oort and Haar, 1976] Oort, A. H. and Haar, T. H. V. (1976). On the observed annual cycle in the ocean-atmosphere heat balance over the northern hemisphere. *Journal of Physical Oceanography*, 6:781–800.

[Stone, 1974] Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147.

[Stone, 1977] Stone, M. (1977). An asymptotic equivalence of choice of model by cross-validation and akaike's criterion. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):44–47.

[Stone, 1978] Stone, P. H. (1978). Constraints on dynamical transports of energy on a spherical planet. *Dynamics of Atmospheres and Oceans*, 2(2):123–139.

[Sudipto Banerjee, 2014] Sudipto Banerjee, A. R. (2014). *Linear Algebra and Matrix Analysis for Statistics.* Chapman and Hall/CRC, New York.

[Tieleman et al., 2012] Tieleman, T., Hinton, G., et al. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.