Stockholms universitet

Comparing Accuracy of Surface Fitting between Artificial Neural Network and Interpolation With Cubic Splines

Lav Radosavljević

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2020:6
Matematisk statistik
Juni 2020

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Matematiska institutionen

Mathematical Statistics
Stockholm University
Bachelor Thesis **2020:6**
http://www.math.su.se

Comparing Accuracy of Surface Fitting between Artificial Neural Network and Interpolation With Cubic Splines

Lav Radosavljević*

June 2020

Abstract

This paper seeks to simulate a real-life problem of fitting a surface from observed data with random errors. We will seek to determine which of two methods, Artificial Intelligence with an Artificial Neural Network or interpolation with cubic splines, will produce the most accurate fitting of the surface we have chosen for the simulation. The measure of accuracy used will be the mean integrated squared error. In order to examine which method is more appropriate to use under different circumstances we will also vary the learning time for the Artificial Neural Network as well as the standard deviation of the random errors in the data. This goal will be achieved by determining which method has the lowest mean integrated squared error for different combinations of error standard deviation and learning time for the Artificial Neural Network.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: val.vic53@yahoo.com. Supervisor: Ola Hössjer, Taras Bodnar.

Contents

1 Introduction

1.1 Problem Inspiration

The idea for this paper came from a concrete engineering problem I had to solve at a mathematical event at the University of Karlstad, Sweden (link to event page [1] and to the problem in question [2]). The task was given to my group by a company called Uddeholms AB that produces steel. They wanted a function that describes the toughness of a steel ingot they produced as a function of the width and height of the ingot. They could not produce one ingot for each possible combination of height and width as this would be extremely expensive. Instead they gave us the measured toughness for 226 ingots with different values of height and width. Because the process of producing an ingot is extremely complicated there was no hope of calculating it with the help of physics. Instead we decided to use two different numeric methods to obtain the function, a artificial neural network (or ANN for short) and cubic spline interpolation. There was disagreement about which method was most appropriate to use, i.e., which method was best at predicting the outcome given a limited amount of computing time. Two important factors that determine this is how accurate the provided data are (i.e. how much error there is in the companies' measurements) and how much time we use to fit the hyperparameters of the ANN, as we cannot use unlimited CPU. I therefore thought it would be appropriate to perform a simulation study that investigates which of the two methods has lower MISE (Mean Integrated Squared Error, see Subsection 2.2.1) for different combinations of number of iterations and error standard deviation. It is important to note that I will not be using any data from the event in Karlstad. I will be simulating data myself in order to see which method is better at interpolating a 3D surface.

1.2 Problem Description

The aim of the thesis is to investigate which of the two methods is better for different combinations of error standard deviation and number of iterations. Of course this will also depend heavily upon the function which is fitted by the two considered methods. We will only be looking at a very simple function throughout the thesis, namely the function

$$f : (0,1)^2 \to \mathbb{R} \ , \ f(x_1, x_2) = 0.3 + \frac{1}{12} e^{x_1 + x_2},$$

as seen in Figure 1. It is a convex function defined on $(0,1)^2$. Moreover, it holds that

$$\inf_{(x_1,x_2)\in(0,1)^2} 0.3 + \frac{1}{12} e^{x_1+x_2} = 0.3 + \frac{1}{12} e^0 \approx 0.383$$

and

$$\sup_{(x_1,x_2)\in(0,1)^2} 0.3 + \frac{1}{12} e^{x_1+x_2} = 0.3 + \frac{1}{12} e^2 \approx 0.916,$$

so the function assumes relatively small values. We can also easily see that all its derivatives are continuous.

We want to solve the following problem for a given number of iterations N used to fit the ANN and a given error standard deviation σ based on a set of $n = 100$ training points

$$(X_{1,1}, X_{1,2}, Y_1), (X_{2,1}, X_{2,2}, Y_2), ..., (X_{100,1}, X_{100,2}, Y_{100}),$$

such that

$$Y_i = 0.3 + \frac{1}{12} \exp\left(X_{i,1} + X_{i,2}\right) + \epsilon_i \ \forall \ i = 1, 2, ..., 100,$$

where

$$\epsilon_i \sim N(0, \sigma^2)$$

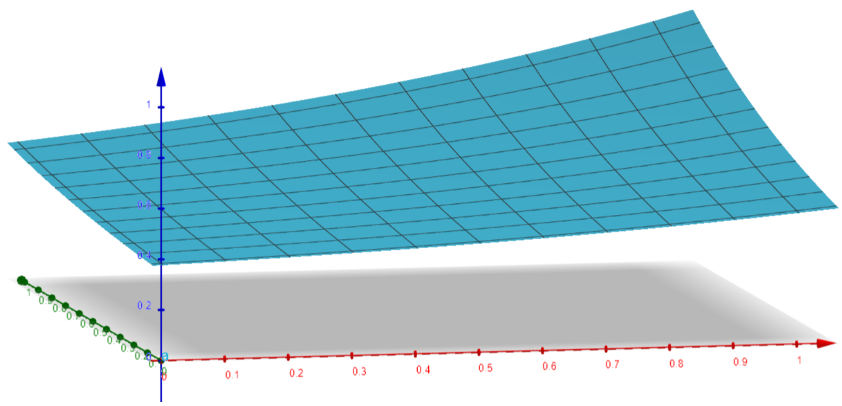

Figure 1: The function $f(x_1, x_2) = 0.3 + \frac{1}{12}e^{x_1+x_2}$ for $x_1, x_2 \in (0, 1)$ used in the simulation study.

are independently and normally distributed error terms. We want to calculate the MISE for each method and see which one is smaller. This will be done for $N = 10^4, 10^5, 10^6$ and $\sigma = 0, 0.01, 0.02, 0.03, 0.04, 0.05$, in order to see which method performs better under which conditions. Our initial suspicion is that

1. The MISE becomes lower for the ANN as N increases

2. The MISE for the ANN becomes relatively smaller compared to that of the cubic interpolation method as σ increases.

We suspect 1. because the ANN will usually perform better if we allow more iterations. We suspect 2. because cubic interpolation is prone to over-fitting, which will make it to perform poorly in particular when we have a large error standard deviation.

1.3 Relevance of the Problem

The comparison of Artificial Neural Networks with older models such as regression or interpolation is very common today in various fields of science [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]. Therefore, the question whether Artificial Neural Networks, or other forms of machine learning, are preferable to older models is a highly relevant in modern science and engineering. This paper seeks to perform this comparison for a special case by simulating a real-world problem.

2 Theory

2.1 General Simulation Description

The training data will be such that each point (X_1, X_2, Y) satisfies

$$Y = 0.3 + \frac{1}{12}e^{X_1+X_2} + \epsilon$$

where $\epsilon \sim N(0, \sigma^2)$ is normally distributed with expected value 0.

2.2 Definitions of Important Terms

Before I go ahead and formulate the problem that I want to solve I will define a few terms which one needs to understand the problem and the contents of the paper.

2.2.1 MISE

MISE is short for "Mean Integrated Squared Error". It is a means by which we quantify how well a curve has been fitted. If we have an unobservable function $f : A \to \mathbb{R}$ and we fit from sample data a function $\hat{f} : A \to \mathbb{R}$, then

$$MISE = \int E\left[(f(x) - \hat{f}(x))^2\right] f_X(x)dx,$$

where $x = (x_1, x_2, \dots, x_d) \in A$ (we will be studying the case $d = 2$) and f_X is the probability density function for the validation points generated to test the fit of the curve. This quantity is an expected value which we usually cannot calculate analytically. Instead we use the empirical equivalent given by

$$\widehat{MISE} = \frac{1}{m}\sum_{k=1}^{m}(f(x_k) - \hat{f}(x_k))^2$$

for some iid validation points $x_1, x_2, ..., x_m$ with probability density function f_X, which are not used to fit the function. And if we use sufficiently large values of m this is a very good approximation [13].

Suppose new observations $Y_k^* = f(x_k) + \epsilon_k^*$ are generated for each one of test data points x_k, $k = 1, \dots, m$, where $\epsilon_k^* \sim N(0, \sigma^2)$ are new error terms, independent of the error terms ϵ_i of the training data set. The the Mean Squared Error of Prediction is

$$\begin{aligned} MSEP &= \tfrac{1}{m}\textstyle\sum_{k=1}^{m} E[(Y_k^* - \hat{f}(x_k))^2] \\ &= \sigma^2 + \widehat{MISE}. \end{aligned}$$

Since MSEP only differs from the estimated MISE by a constant σ^2, we will use the latter as performance criterion, since it is easier to compare between data sets with varying σ.

2.2.2 Stochastic Gradient Descent and Number of Iterations

In order to fit hyperparameters from data in an Artificial Neural Network, we will use a algorithm called Stochastic Gradient Descent (SGD). Suppose we want to minimise a function $Q : \Omega \to \mathbb{R}$, (where Ω is the parameter space of our fitting method) , such that

$$Q(\omega) = \sum_{i=1}^{n} Q_i(\omega) \ \forall \ \omega = (\omega_1, ..., \omega_d) \in \Omega$$

We can do this by performing the following iteration N times:

$$\omega_j = \omega_{j-1} - \eta \nabla Q_{i_j}$$

where each index i_j for $j = 1, 2, ..., N$ is selected with uniform probability from the numbers $1, 2, ..., n$, where n is the number of training data points, and we have an initial randomly generated guess ω_0. When we refer to "number of iterations" in this paper, we mean the above number N, i.e the number of iterations in the SGD algorithm.
The constant η is called the learning rate and it is the step size in the SGD algorithm. In our case we have chosen the value $\eta = 0.05$.

2.2.3 Error Standard Deviation

The error standard deviation σ is the standard deviation for the "measurement error" in our study. In other words we assume that each response variable is observed with an error and that these errors are independent and normally distributed $N(0, \sigma^2)$.

2.2.4 Activation Function

The activation function is a function used to transform input data inside a neural network so that the network can be used to approximate non-linear functions. In our case we have chosen the following activation function

$$\alpha(x) = \sinh^{-1}(x) = \ln(x + \sqrt{x^2 + 1}).$$

It is important to note that in this paper we will be applying this activation function elementwise to vectors. What this means is that if

$$v = \begin{pmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{pmatrix}$$

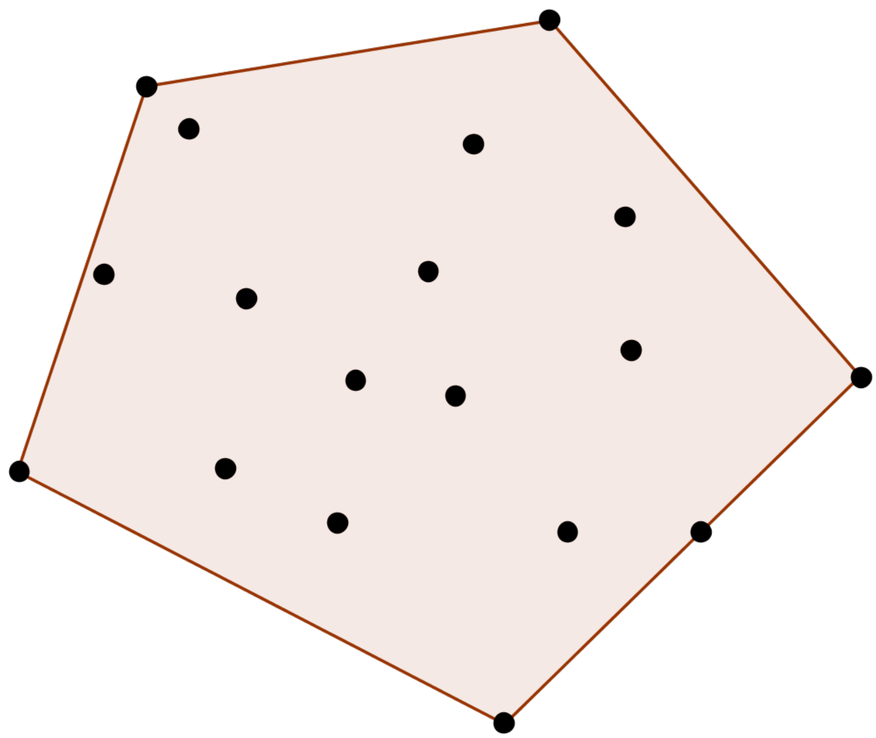

Figure 2: Convex hull of a set of points

then

$$\alpha(v) = \begin{pmatrix} \alpha(y_1) \\ \alpha(y_2) \\ . \\ . \\ . \\ \alpha(y_n) \end{pmatrix}.$$

2.2.5 Convex Hull

The convex hull of a set of points $X_1, X_2, .., X_n \in \mathbb{R}^2$ is the unique minimal convex subset of $\mathbb{R}^2$ containing all points. Another way of defining the convex hull of $X_1, X_2, ..., X_n$ is the union of all polygons with corners in the set $\{X_1, X_2, ..., X_n\}$.

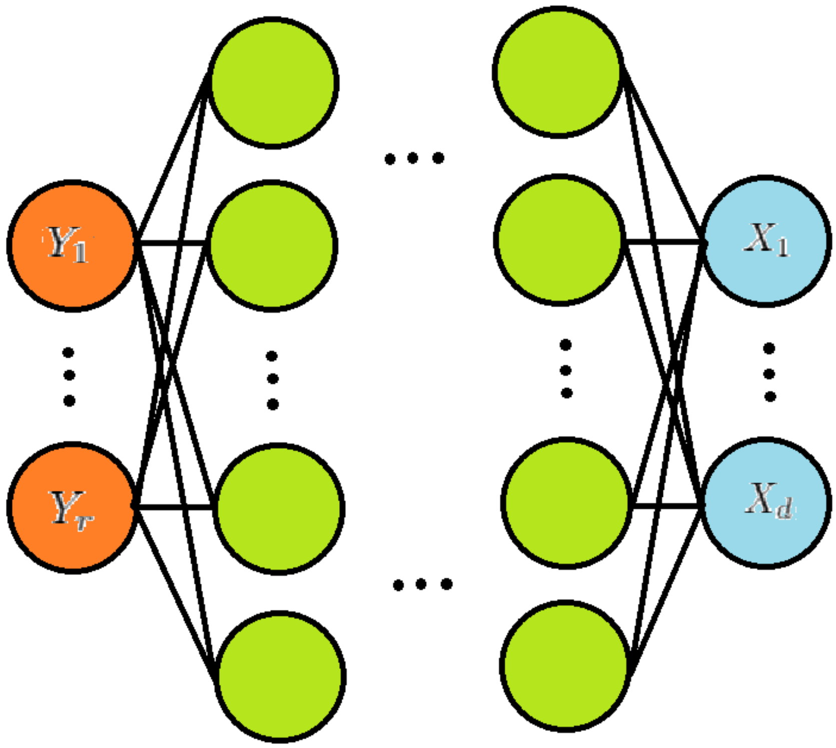

Figure 3: ANN with input layer $X = (X_1, ..., X_d)$, several hidden layers and output layer $Y = (Y_1, ..., Y_r)$. We will be dealing with the case $d = 2$, $r = 1$ and three hidden layers.

2.2.6 Hadamard Product

The Hadamard product is a form of matrix multiplication which is only defined for matrices of the same dimensions. The Hadamard product between two matrices of the same dimensions is another matrix of said dimensions where each element is the product of the two elements in that same position for the two matrices that we are multiplying. We use the symbol $\circ$ to symbolise Hadamard multiplication. In the 3×3 case, Hadamard multiplication looks like this:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \circ \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{23}b_{23} \\ a_{31}b_{31} & a_{32}b_{32} & a_{33}b_{33} \end{pmatrix}.$$

2.3 Architecture of the ANN

Let us describe the architecture of the neural network which we will be using. This is a neural network with three hidden layers, one input layer and one output layer. This type of network will have four weights and four biases. We also have an activation function, which will be $\alpha(x) = \sinh^{-1}(x)$ for reasons that will be discussed later on in the paper. Given this structure we get the following function

$$f(x) = \alpha(W_5\alpha(W_4\alpha(W_3\alpha(W_2x + b_2) + b_3) + b_4) + b_5),$$

where W_2, W_3, W_4, W_5 are weight matrixes, b_2, b_3, b_4, b_5 are bias vectors and $\alpha(\cdot)$ is applied element-wise as described in Subsection 2.2.4. We now want to construct a cost function as a function of the weights and biases in the ANN that will help us estimate the parameters in the weights and biases. We choose the least squares method of optimisation, which means that the cost function is

$$C(W_2, W_3, W_4, W_5, b_2, b_3, b_4, b_5) = \sum_{i=1}^{n} \|y_i - f(x_i)\|^2 =$$

$$= \sum_{i=1}^{n} \|y_i - \alpha(W_5\alpha(W_4\alpha(W_3\alpha(W_2x + b_2) + b_3) + b_4) + b_5)\|^2,$$

given n vectors explanatory variable vectors $x_1, x_2, ..., x_n$, n response vectors $y_1, y_2, ..., y_n$ and $\| \cdot \|$ refers to the Euclidian vector norm. Our least square estimates are

$$(\hat{W}_2, \hat{W}_3, \hat{W}_4, \hat{W}_5, \hat{b}_2, \hat{b}_3, \hat{b}_4, \hat{b}_5) = argmin\, C(W_2, W_3, W_4, W_5, b_2, b_3, b_4, b_5).$$

2.4 Least Squares Estimation of Hyperparameters

In order to find our estimate we need to minimise the cost function. The total number of parameters in the cost function is $p = |W_2| + |b_2| + |W_3| + |b_3| + |W_4| + |b_4| + |W_5| + |b_5| = 2 \cdot 2 + 2 + 2 \cdot 3 + 3 + 3 \cdot 2 + 2 + 2 \cdot 1 + 1 = 26$. We will use the method called Stochastic Gradient Descent described in Subsection 2.2.2. Specifically we use the iterative version of the method. The method finds the minimum of the following sum:

$$Q(\omega) = \sum_{i=1}^{n} Q_i(\omega).$$

Let us now think back to our problem. We wish to minimize the function

$$C(W_2, W_3, W_4, b_2, b_3, b_4) = \sum_{i=1}^{n} (y_i - f(x_i))^2 =$$

$$= \sum_{i=1}^{n} (y_i - \alpha(W_5\alpha(W_4\alpha(W_3\alpha(W_2x + b_2) + b_3) + b_4) + b_5))^2.$$

We now observe that if we let

$$\omega = (W_2, W_3, W_4, W_5, b_2, b_3, b_4, b_5)$$

and

$$Q_i(\omega) = (y_i - \alpha(W_5\alpha(W_4\alpha(W_3\alpha(W_2x + b_2) + b_3) + b_4) + b_5))^2,$$

our fitting of hyperparameters can be solved using stochastic gradient descent. The only thing we need to do is to find a way of calculating the vector $\nabla Q_i(\omega)$. In order to do this we will be using the chain rule, so what we observe about our activation function $\alpha(x) = \sinh^{-1}(x)$ is that

$$\alpha'(x) = \frac{1}{\sqrt{1 + x^2}} = g(\alpha(x)),$$

where

$$g(y) = \frac{1}{\sqrt{1 + \sinh(y)^2}}.$$

We also introduce the following notation to shorten the derived expressions:

$$a_2 = \alpha(W_2x_i + b_2)$$

$$a_3 = \alpha(W_3\alpha(W_2x_i + b_2) + b_3)$$
$$a_4 = \alpha(W_4\alpha(W_3\alpha(W_2x_i + b_2) + b_3) + b_4)$$
$$a_5 = \alpha(W_5\alpha(W_4\alpha(W_3\alpha(W_2x_i + b_2) + b_3) + b_4) + b_5)$$
$$\delta_5 = \frac{\partial Q_i}{\partial b_5}$$
$$\delta_4 = \frac{\partial Q_i}{\partial b_4}$$
$$\delta_3 = \frac{\partial Q_i}{\partial b_3}$$
$$\delta_2 = \frac{\partial Q_i}{\partial b_2}$$

Using the chain rule, we get that

$$\delta_5 = \frac{\partial Q_i}{\partial b_5} = -2(y_i - a_5) \circ \alpha'(W_5a_4 + b_5) = -2(y_i - a_5) \circ g(a_5),$$

where $g(\cdot)$ is applied element-wise to vectors just as $\alpha(\cdot)$ and

$$\delta_4 = \frac{\partial Q_i}{\partial b_4} = W_5^T[-2(y_i - a_5) \circ \alpha'(W_5a_4 + b_5)] \circ \alpha'(W_4a_3 + b_4) =$$

$$= W_5^T[-2(y_i - a_5) \circ g(a_5)] \circ g(a_4) = W_5^T\delta_5 \circ g(a_4).$$

Using the same logic of chain differentiation, we get that

$$\delta_3 = W_4^T \delta_4 \circ g(a_3)$$

and

$$\delta_2 = W_3^T \delta_3 \circ g(a_2).$$

Now that we have calculated the gradient for all biases, it is time to calculate it for the weights, that is to say

$$\frac{\partial Q_i}{\partial W_j}$$

for all $j = 2, 3, 4, 5$. We observe that

$$\frac{\partial Q_i}{\partial W_5} = [-2(y_i - a_5) \circ \alpha'(W_5 a_4 + b_5)] a_4^T = [-2(y_i - a_5) \circ g(a_5)] a_4^T = \delta_5 a_4^T$$

and

$$\frac{\partial Q_i}{\partial W_4} = [W_5^T [-2(y_i - a_5) \circ \alpha'(W_5 a_4 + b_5)] \circ \alpha'(W_4 a_3 + b_4)] a_3^T =$$

$$= [W_5^T [-2(y_i - a_5) \circ g(a_5)] \circ g(a_4)] a_3^T = \delta_4 a_3^T.$$

Using the same logic of chain differentiation, we also get that

$$\frac{\partial Q_i}{\partial W_3} = \delta_3 a_2^T$$

and

$$\frac{\partial Q_i}{\partial W_2} = \delta_2 x_i^T.$$

So we have found a way to calculate a_2, a_3, a_4, a_5 and δ_2, δ_3, δ_4, δ_5. Hence, using the above equations, we can perform the iteration

$$\omega := \omega - \eta \nabla Q_{i_j}(\omega)$$

by performing the following iterations:

$$b_5 := b_5 - \eta \delta_5$$

$$b_4 := b_4 - \eta \delta_4$$

$$b_3 := b_3 - \eta \delta_3$$

$$b_2 := b_2 - \eta \delta_2$$

and

$$W_5 := W_5 - \eta \delta_5 a_4^T$$

$$W_4 := W_4 - \eta \delta_4 a_3^T$$

$$W_3 := W_3 - \eta \delta_3 a_2^T$$

$$W_2 := W_2 - \eta \delta_2 x_i^T.$$

And this process is repeated for 10^4, 10^5 or 10^6 iterations.

2.5 Cubic Interpolation

Now we want to describe the process by which we interpolate our surface given 100 points in $\mathbb{R}^3$. The constraints are that the surface has to go through all points

$$(X_{1,1}, X_{1,2}, Y_1), (X_{2,1}, X_{2,2}, Y_2), ...(X_{100,1}, X_{100,2}, Y_{100}),$$

and it has to be a C^2-surface. We will be using the built-in MATLAB function "griddata" and its built-in option "cubic" for this purpose. This function performs the cubic interpolation in two steps, Delauney triangulation and estimation of parameters. As I could not find a clear answer on what algorithm is used for estimating the parameters in the case of 100 points, I will just include the code for griddata in the appendix and explain how it can be done for one example with 4 points.

2.5.1 Delaunay Triangulation

Delaunay triangulation is a way in which we divide the convex hull of the points

$$(X_{1,1}, X_{1,2}), ..., (X_{100,1}, X_{100,2}) \in \mathbb{R}^2$$

in triangles such that none of the points $(X_{i,1}, X_{i,2})$ is strictly inside the circumcircle of any of the triangles. I do not now exactly which algorithm is used to perform this division in griddata, but I found a good paper [14] where two appropriate algorithms are described. It is guaranteed that there exists a unique way of performing Delaunay triangulation as long as there are no three points on a line or four points on the same circle. Because we generate our points on $(0,1)^2$ according to a uniform distribution, the probability that three of them are on a line or four of them are on a circle is 0.

2.5.2 Estimation of parameters

When we have finished the Delaunay triangulation we want to assign to each triangle T_i a cubic polynomial

$$p_i(x_1, x_2) = a_0^{(i)} + a_1^{(i)} x_1 + a_2^{(i)} x_2 + a_3^{(i)} x_1^2 + a_4^{(i)} x_1 x_2 + a_5^{(i)} x_2^2 + a_6^{(i)} x_1^3 + a_7^{(i)} x_1^2 x_2 + a_8^{(i)} x_1 x_2^2 + a_9^{(i)} x_2^3.$$

Now we have to fit all parameters $a_k^{(i)}$ so that the obtained surface goes through all $n = 100$ training points and is a C^2-surface. As we can see there are 10 unknown parameters for each triangle. I will show how this is done in a case where we have 4 points

$$(X_{1,1}, X_{1,2}, Y_1), (X_{2,1}, X_{2,2}, Y_2), (X_{3,1}, X_{3,2}, Y_3), (X_{4,1}, X_{4,2}, Y_4)$$

and 2 triangles

$$T_1, T_2.$$

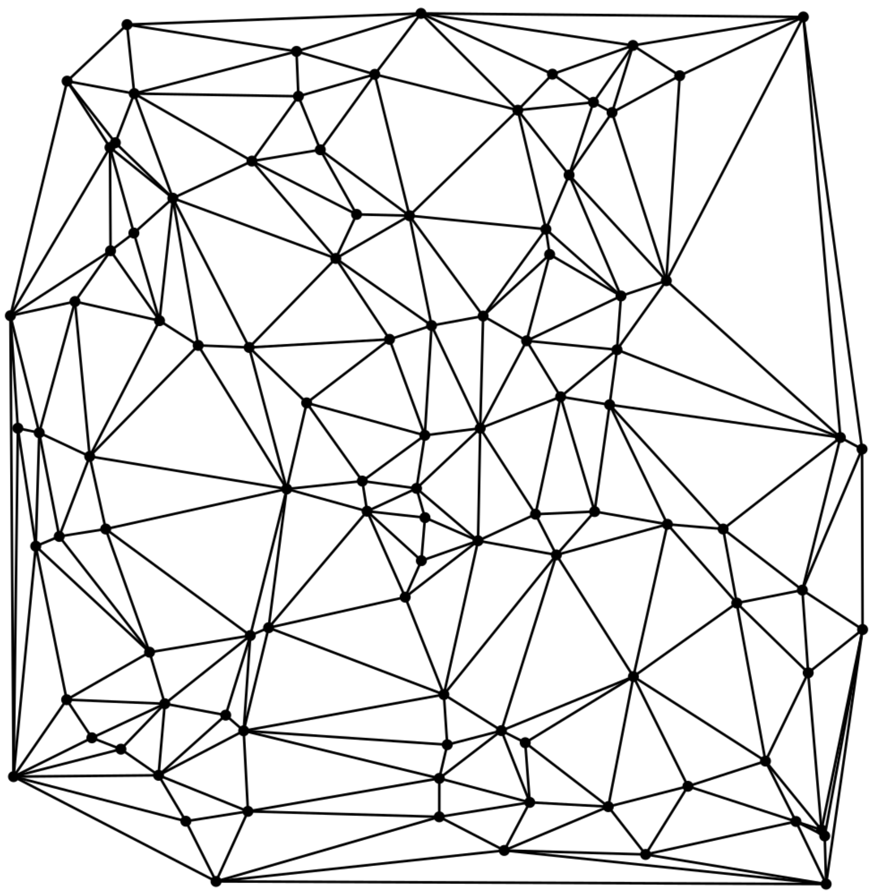

Figure 4: Example of a Delaunay triangulation with 100 points

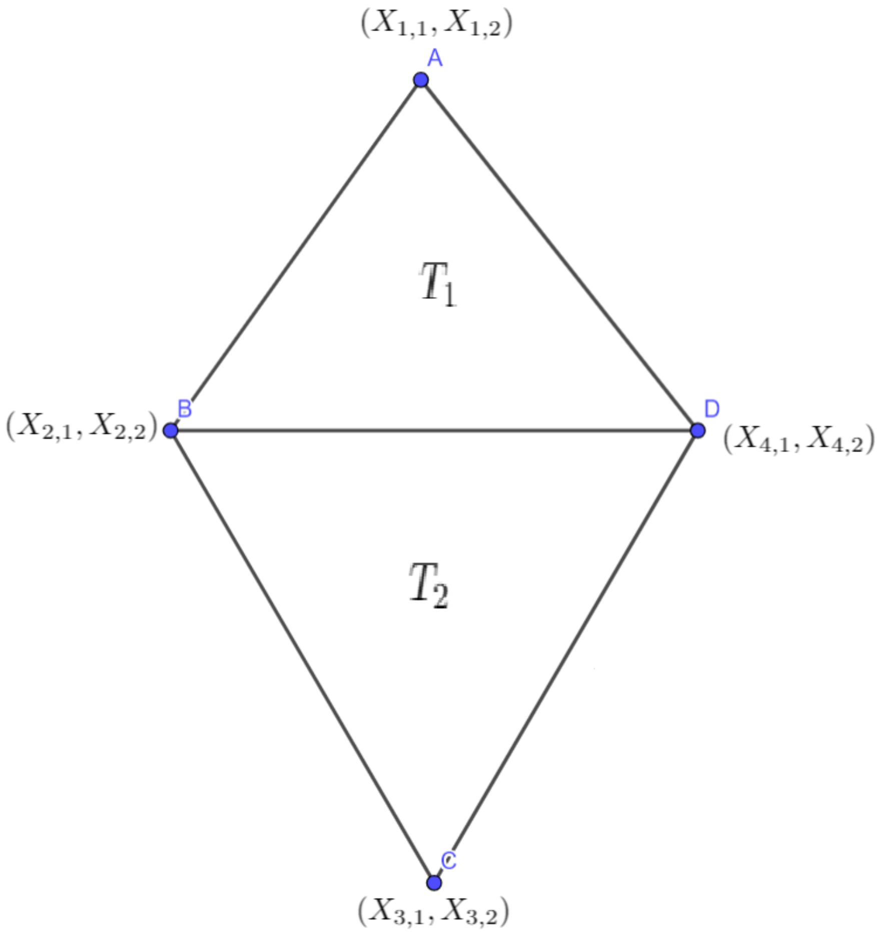

Figure 5: A Delaunay triangulation with 4 points and 2 triangles

If we look at Figure 4 we see that we want to fit 2 polynomials p_1, p_2, one for each triangle, in such a way that the obtained surface goes through all 4 points and is a C^2-surface. The points A and C in figure 4 give us that

$$p_1(X_{1,1}, X_{1,2}) = Y_1$$

$$p_2(X_{3,1}, X_{3,2}) = Y_3.$$

The points B and C give us, under the constraint that the interpolated surface has to be a C^2-surface, that

$$\begin{aligned}
p_1(X_{2,1}, X_{2,2}) = p_2(X_{2,1}, X_{2,2}) &= Y_2 \\
\nabla p_1(X_{2,1}, X_{2,2}) &= \nabla p_2(X_{2,1}, X_{2,2}) \\
Hp_1(X_{2,1}, X_{2,2}) &= Hp_2(X_{2,1}, X_{2,2})
\end{aligned}$$

and

$$\begin{aligned}
p_1(X_{4,1}, X_{4,2}) = p_2(X_{4,1}, X_{4,2}) &= Y_4 \\
\nabla p_1(X_{4,1}, X_{4,2}) &= \nabla p_2(X_{4,1}, X_{4,2}) \\
Hp_1(X_{4,1}, X_{4,2}) &= Hp_2(X_{4,1}, X_{4,2}).
\end{aligned}$$

We can see that the above equations are linear equations with unknown parameters $a_k^{(i)}$. The points A and C give us one linear equation each, and the points B and D give us $2+2+3 = 7$ linear equations each. So in total we have $2 \cdot 1 + 2 \cdot 7 = 16$ linear equations. But because we have 2 triangles and therefore two polynomials with 10 coefficients $a_k^{(i)}$ each, this means that we have 16 linear equations and 20 unknowns. And because the data is generated randomly we can be sure to find at least one solution with probability 1. I do not know which exact method is used to choose a solution. One possibility is to chose the solution that minimizes the sum

$$\sum_{k=0}^{9} \sum_{i=1}^{2} [a_k^{(i)}]^2.$$

3 Description of Simulation Study

As we know the goal is to calculate the $\widehat{MISE}_{ANN}$ and $\widehat{MISE}_{INTER}$ for $\sigma = 0, 0.01, 0.02, 0.03, 0.04, 0.05$ and where the number of iterations in the ANN is 10^4, 10^5 and 10^6 respectively. In order to calculate $\widehat{MSPE}_{ANN}$ and $\widehat{MISE}_{INTER}$ for each one of the 18 combinations of error standard deviation and number of iterations for the ANN, we will simulate 18 data sets, with 500 data sets each, that will be used to calculate the desired quantities for all combinations. The simulation for a given number of iterations and error standard deviation starts by drawing randomly and independently 500 data sets with $n + m = 200$ points, all of which are in $(0, 1)^2$. Let $(X_{i,1}^{(\sigma,N,K)}, X_{i,2}^{(\sigma,N,K)})$ be ith point in data set number K. The first $n = 100$ points in each data set will be used for estimating the hyperparameters in the ANN and the coefficients in the cubic interpolation. Therefore we simulate them independently according to a uniform distribution on the set $(0,1)^2$. After that we simulate another $m = 100$ points independently according to a uniform distribution on the convex hull of the previous $n = 100$ points. This is done because the last $m = 100$ points will be used for testing the two methods and calculate their respective $\widehat{MISE}$ and using 2D-interpolation for predicting values is only useful inside the convex hull of the training data. Now let $\widehat{MISE}_{INTER}^{(\sigma,N)}$ and $\widehat{MISE}_{ANN}^{(\sigma,N)}$ be the calculated $\widehat{MISE}$ for the two methods with error standard deviation σ and number of iterations N. Let further $\widehat{MISE}_{INTER}^{(\sigma,N,K)}$ and $\widehat{MISE}_{ANN}^{(\sigma,N,K)}$ be the calculated $\widehat{MISE}_{INTER}^{(\sigma,N)}$ and $\widehat{MISE}_{ANN}^{(\sigma,N)}$ for data set number K. We define

$$\widehat{MISE}_{INTER}^{(\sigma,N)} = \frac{1}{500}\sum_{K=1}^{500} \widehat{MISE}_{INTER}^{(\sigma,N,K)}$$

and

$$\widehat{MISE}_{ANN}^{(\sigma,N)} = \frac{1}{500}\sum_{K=1}^{500} \widehat{MISE}_{ANN}^{(\sigma,N,K)}.$$

Now let $\hat{f}_{INTER}^{(\sigma,N,K)} : \Omega^{(\sigma,N,K)} \to \mathbb{R}$ and $\hat{f}_{ANN}^{(\sigma,N,K)} : \Omega^{(\sigma,N,K)} \to \mathbb{R}$, where $\Omega^{(\sigma,N,K)}$ is the convex hull of the $n = 100$ points

$$(X_{1,1}^{(\sigma,N,K)}, X_{1,2}^{(\sigma,N,K)}), (X_{2,1}^{(\sigma,N,K)}, X_{2,2}^{(\sigma,N,K)}), ..., (X_{100,1}^{(\sigma,N,K)}, X_{100,2}^{(\sigma,N,K)}),$$

be the surface that we obtain from data set number K with error standard deviation σ and number of iterations N for interpolation and ANN respectively. Then by definition it follows that

$$\widehat{MISE}_{INTER}^{(\sigma,N,K)} = \sum_{k=101}^{200} \left(f\left(X_{k,1}^{(\sigma,N,K)}, X_{k,2}^{(\sigma,N,K)}\right) - \hat{f}_{INTER}^{(\sigma,N,K)}\left(X_{k,1}^{(\sigma,N,K)}, X_{k,2}^{(\sigma,N,K)}\right)\right)^2$$

and

$$\widehat{MISE}_{ANN}^{(\sigma,N,K)} = \sum_{k=101}^{200} \left(f\left(X_{k,1}^{(\sigma,N,K)}, X_{k,2}^{(\sigma,N,K)}\right) - \hat{f}_{ANN}^{(\sigma,N,K)}\left(X_{k,1}^{(\sigma,N,K)}, X_{k,2}^{(\sigma,N,K)}\right)\right)^2.$$

4 Results

Let us now look at the estimated MISE for each method and for all combinations

$$N = 10^4, 10^5, 10^6$$

and

$$\sigma = 0, 0.01, 0.02, 0.03, 0.04, 0.05.$$

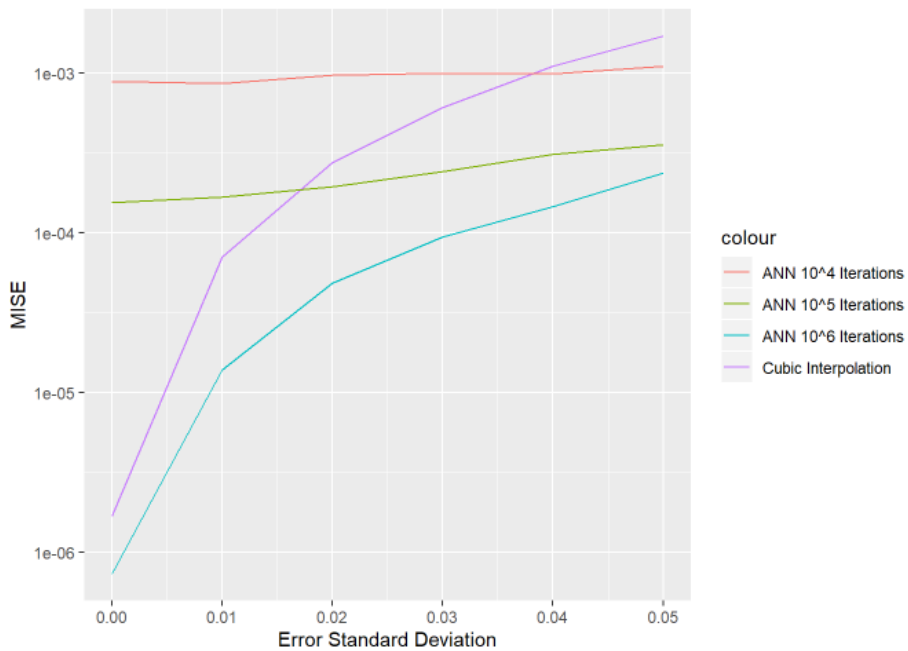

Figure 6: Plot of the $\widehat{MISE}$ for Cubic Interpolation and ANN, with different number of iterations, as a function of the error standard deviation σ.

Table 1: $\widehat{MISE}$ for the Artificial Neural Network

	$N = 10^4$	$N = 10^5$	$N = 10^6$
$\sigma = 0.00$	$8.9 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$	$7.4 \cdot 10^{-7}$
$\sigma = 0.01$	$8.6 \cdot 10^{-4}$	$1.7 \cdot 10^{-4}$	$1.4 \cdot 10^{-5}$
$\sigma = 0.02$	$9.7 \cdot 10^{-4}$	$1.9 \cdot 10^{-4}$	$4.9 \cdot 10^{-5}$
$\sigma = 0.03$	$1.0 \cdot 10^{-3}$	$2.4 \cdot 10^{-4}$	$9.4 \cdot 10^{-5}$
$\sigma = 0.04$	$9.9 \cdot 10^{-4}$	$3.1 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$
$\sigma = 0.05$	$1.1 \cdot 10^{-3}$	$3.5 \cdot 10^{-4}$	$2.4 \cdot 10^{-4}$

Table 2: $\widehat{MISE}$ for the Cubic Interpolation Method

$\sigma = 0.00$	$1.7 \cdot 10^{-6}$
$\sigma = 0.01$	$7.1 \cdot 10^{-5}$
$\sigma = 0.02$	$2.7 \cdot 10^{-4}$
$\sigma = 0.03$	$6.1 \cdot 10^{-4}$
$\sigma = 0.04$	$1.1 \cdot 10^{-3}$
$\sigma = 0.05$	$1.7 \cdot 10^{-3}$

Table 3: Which Method is Better for Different Cases?

	$N = 10^4$	$N = 10^5$	$N = 10^6$
$\sigma = 0.00$	$CUBIC$	$CUBIC$	ANN
$\sigma = 0.01$	$CUBIC$	$CUBIC$	ANN
$\sigma = 0.02$	$CUBIC$	ANN	ANN
$\sigma = 0.03$	$CUBIC$	ANN	ANN
$\sigma = 0.04$	ANN	ANN	ANN
$\sigma = 0.05$	ANN	ANN	ANN

In Figure 6 we can see the plot of the $\widehat{MISE}$ as a function of σ for the three different cases $N = 10^4$, $N = 10^5$ and $N = 10^6$, each with a separate colour. We also have the plot of the $\widehat{MISE}$ for the cubic interpolation method as a function of σ in the same graph. Table 1 and Table 2 are the raw data for the $\widehat{MISE}$ that is used in the graph for the ANN and the cubic interpolation method respectively. Table 3 shows which method is more appropriate to use for different combinations of N and σ, i.e., it answers whether

$$\widehat{MISE}_{ANN}^{(N,\sigma)} < \widehat{MISE}_{INTER}^{(N,\sigma)} \text{ or } \widehat{MISE}_{ANN}^{(N,\sigma)} > \widehat{MISE}_{INTER}^{(N,\sigma)}.$$

A detailed discussion of the obtained results is provided in Section 5.1.

5 Discussion

5.1 Discussion of Results

What we see is that both our suspicions (see end of Subsection 2.2) have been fulfilled. Most likely this is due to the fact that the cubic spline method requires us to estimate a very large number of parameters which will lead to over-fitting and thus larger MISE when the error standard deviation increases. As we can see in Figure 3, a great number of triangles is formed with Delaunay triangulation for 100 points, and each of these will require us to estimate 10 parameters each in order to satisfy the constraints of the algorithm for interpolation with cubic splines. One of the constraints is that the obtained surface has to go through all of the data points which will lead to a much larger MISE when σ grows.

Furthermore we see that the ANN with $N = 10^6$ performs better than the cubic interpolation method for all values of σ. The problem is that it takes several hours to perform 10^6 iterations for the SGD algorithm. Therefore we can conclude that the most desirable model would be one which does not have the problem of over-fitting, while at the same time we do not need several hours to fit the parameters of the model.

One other interesting observation is that if we look at the case $N = 10^6$ in figure 6, we can see that it there is a tendency towards over-fitting i.e. the $MISE$ rises heavily in response to a rise in error standard deviation. Indeed, one might suspect that the cases $N = 10^7$ or $N = 10^8$ might perform worse for large values of σ than the case $N = 10^6$. This suspicion is confirmed by the literature [15], which says that it is not desirable to have too long a training time because it can lead to over-fitting and therefore worse predictions.

5.2 Why do we use SGD instead of Gradient Descent?

The Gradient descent algorithm provides an alternative approach to the Stochastic Gradient Descent Algorithm to fit the model by the ANN. The Gradient Descent algorithm could possibly yield the same or better result as the SGD algorithm, but with lower computational time. We have that

$$Q(\omega) = \sum_{i=1}^{100} Q_i(\omega) \Longrightarrow \frac{\partial Q}{\partial \omega} = \sum_{i=1}^{100} \frac{\partial Q_i}{\partial \omega}.$$

We already have the method for calculating $\frac{\partial Q_i}{\partial \omega}$ for all i described in Subsection 3.4. Therefore we can use the gradient descent algorithm by performing the recursion

$$\omega_j = \omega_{j-1} - \eta \nabla Q(\omega_{j-1}).$$

Using this summation, I implemented the Gradient Descent algorithm to solve the same problem, but the result was unsuccessful, as the calculated $\widehat{MISE}$ was very high. When I looked in the literature, I found that the Gradient Descent algorithm is so far unsuccessful for problems where the number of training data points n is large [16]. Because $n = 100$ in our case, that might explain why we it is preferable to use the Stochastic Gradient Descent algorithm instead of the Gradient Descent Algorithm.

5.3 Smoothing Splines

As was mentioned in Subsection 5.1, we want a method that avoids over-fitting while at the same time does not require several hours of time to fit the param-

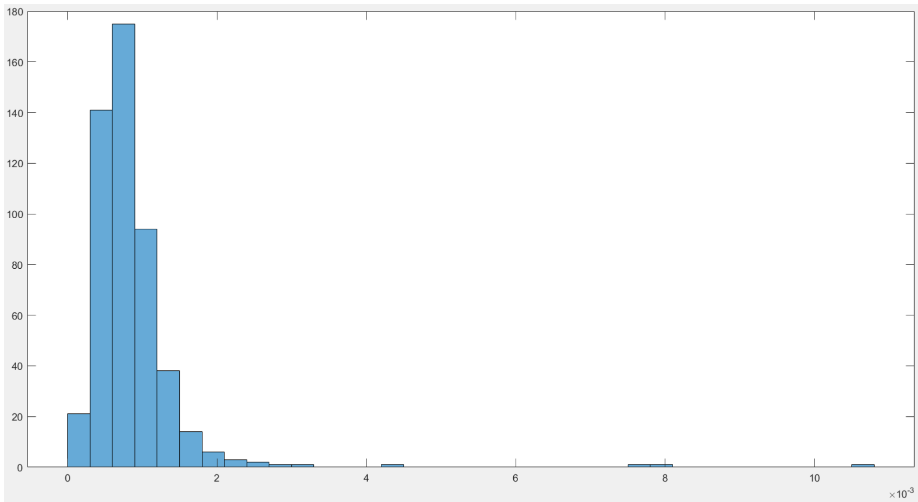

Figure 7: ANN, Histogram of Estimated MISE for $N = 10^4$ and $\sigma = 0.02$

eters of the model. A way of obtaining this is by using smoothing splines. The method finds the fitted the function $\hat{f} : (0,1)^2 \to \mathbb{R}$ from data by minimizing the value of

$$\sum_{i=1}^{100}(Y_i - \hat{f}(X_{i,1}, X_{i,2}))^2 + \lambda \int_0^1 \int_0^1 [H(\hat{f}(x,y))]dxdy.$$

The coefficient λ is a coefficient for determining how much we will penalise lack of smoothness. Here we can see that if $\lambda = 0$ we will obtain the minimum value from our cubic interpolation method because in that case

$$Y_i = \hat{f}(X_{i,1}, X_{i,2}) \ \forall \ i = 1, 2, ..., 100.$$

If $\lambda \to \infty$ the problem is reduced to multiple linear regression. So some intermediate value of λ must be chosen.
This is a topic for further study as it may be a method that both avoids overfitting and is computationally cheap. I have also found good literature on the topic. [17]

5.4 Histogram of MISEs

I have looked at the histograms of the MISEs for each pair of N and σ i.e. for the 500 data sets that we generate for each such pair. In general it can be said that the ANN method has a fat right tail whereas the cubic interpolation method has a more bell-shaped histogram. A histogram of the MISEs for the case $N = 10^4$ and $\sigma = 0.02$ are shown for the ANN and for the cubic interpolation method in Figure 7 and Figure 8 respectively.

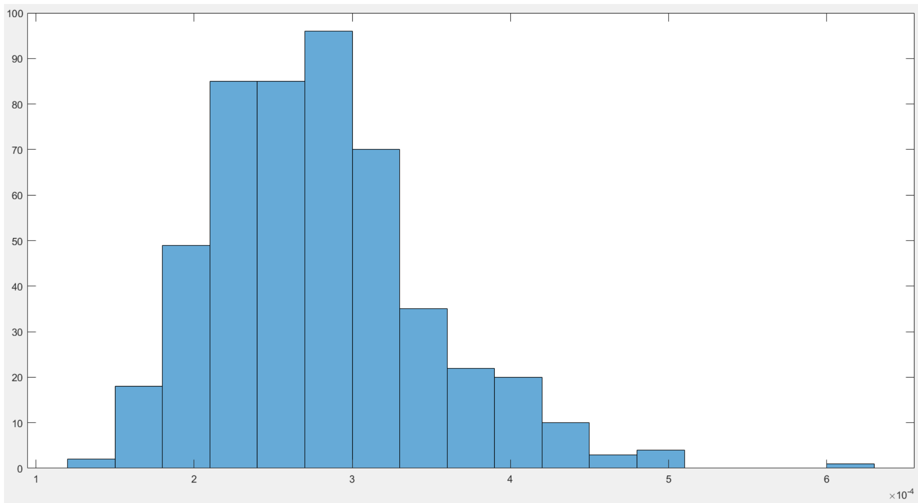

Figure 8: Cubic Interpolation, Histogram of Estimated MISE for $N = 10^4$ and $\sigma = 0.02$

References

[1] URL: `https://www.kau.se/MIMM-day`.

[2] URL: `https://www.kau.se/files/2019-11/Interpolation%5C%20Problem.pdf`.

[3] Barenya Bikash Hazarika, Deepak Gupta, Mohanadhas Berlin, et al. "A Comparative Analysis of Artificial Neural Network and Support Vector Regression for River Suspended Sediment Load Prediction". In: *First International Conference on Sustainable Technologies for Computational Intelligence*. Springer. 2020, pp. 339–349.

[4] Milica Arsić et al. "Prediction of Ozone Concentration in Ambient Air Using Multilinear Regression and the Artificial Neural Networks Methods". In: *Ozone: Science & Engineering* 42.1 (2020), pp. 79–88.

[5] Panagiotis Barmpalexis et al. "Application of Multiple Linear Regression and Artificial Neural Networks for the Prediction of the Packing and Capsule Filling Performance of Coated and Plain Pellets Differing in Density and Size". In: *Pharmaceutics* 12.3 (2020), p. 244.

[6] Mumtaz Ali and Ravinesh C Deo. "Modeling wheat yield with data-intelligent algorithms: artificial neural network versus genetic programming and minimax probability machine regression". In: *Handbook of Probabilistic Models*. Elsevier, 2020, pp. 37–87.

[7] Aditya Rana et al. "Predicting Blast-Induced Ground Vibrations in Some Indian Tunnels: a Comparison of Decision Tree, Artificial Neural Network and Multivariate Regression Methods". In: *Mining, Metallurgy & Exploration* (2020), pp. 1–15.

[8] Bulent Ekiz et al. "Comparison of the decision tree, artificial neural network and multiple regression methods for prediction of carcass tissues composition of goat kids". In: *Meat science* 161 (2020), p. 108011.

[9] Ali Komeilibirjandi et al. "Thermal conductivity prediction of nanofluids containing CuO nanoparticles by using correlation and artificial neural network". In: *Journal of Thermal Analysis and Calorimetry* 139.4 (2020), pp. 2679–2689.

[10] Jakub Horák, Petr Šuleř, and Jaromır Vrbka. "Comparison of neural networks and regression time series when predicting the export development from the USA to PRC". In: *International Scientific Conference „Contemporary Issues in Business, Management and Economics Engineering"*. 2019.

[11] Mingjun Li and Junxing Wang. "An empirical comparison of multiple linear regression and artificial neural network for concrete dam deformation modelling". In: *Mathematical Problems in Engineering* 2019 (2019).

[12] Wen-Jing Niu et al. "Comparison of multiple linear regression, artificial neural network, extreme learning machine, and support vector machine in deriving operation rule of hydropower reservoir". In: *Water* 11.1 (2019), p. 88.

[13] Wolfgang Hardle. "Approximations to the mean integrated squared error with applications to optimal bandwidth selection for nonparametric regression function estimators". In: *Journal of multivariate analysis* 18.1 (1986), pp. 150–168.

[14] Der-Tsai Lee and Bruce J Schachter. "Two algorithms for constructing a Delaunay triangulation". In: *International Journal of Computer & Information Sciences* 9.3 (1980), pp. 219–242.

[15] Gilbert Strang. "Linear Algebra and Learning from Data". In: Wesley-Cambridge Press, 2019. Chap. VI.5.

[16] Gilbert Strang. "Linear Algebra and Learning from Data". In: Wesley-Cambridge Press, 2019. Chap. VI.5.

[17] B.W Silverman P.J Green. *Nonparametric Regression and Generalized Linear Models: A roughness penalty approach.* Chapman and Hal, 1994.

Appendix

The code used for ANN with three hidden layers

I borrowed segments of this code from my solution to a homework assignment in the course MM7024 with Professor Zhaojun Bai.

```
for bla = 0:5
mises1 = []; %vector where the 500 MISEs for the cubic method are stored
mises2 = []; %vector where the 500 MISEs for the ANN are stored
N = 10000; %number of iterations
sigma = bla*0.01; %error standard deviation
for temp = 1:500
x1 = rand(1,100);
x2 = rand(1,100);
y = 0.3+exp(x1+x2)/12+normrnd(0,sigma,1,100);
[xq,yq] = meshgrid((0:1000)*0.001, (0:1000)*0.001);
vq = griddata(x1,x2,y,xq,yq,'cubic'); %surface is interpolated
vq = vq';

W2 = 0.5*randn(2,2);
W3 = 0.5*randn(3,2);
W4 = 0.5*randn(2,3);
W5 = 0.5*randn(1,2);
b2 = 0.5*randn(2,1);
b3 = 0.5*randn(3,1);
b4 = 0.5*randn(2,1);
b5 = 0.5*randn(1,1);

eta = 0.05; % learning rate
for counter = 1:N %
 k = randi(100); %point is chosen randomly

 x = [x1(k); x2(k)];

 a2 = act(x,W2,b2);
 a3 = act(a2,W3,b3);
 a4 = act(a3,W4,b4);
 a5 = act(a4,W5,b5);

 d5 = der(a5).*(a5-y(k));
 d4 = der(a4).*(W5'*d5);
 d3 = der(a3).*(W4'*d4);
 d2 = der(a2).*(W3'*d3);
 % recursion
 W2 = W2 - eta*d2*x';
 W3 = W3 - eta*d3*a2';
 W4 = W4 - eta*d4*a3';
 W5 = W5 - eta*d5*a4';
 b2 = b2 - eta*d2;
```

```
 b3 = b3 - eta*d3;
 b4 = b4 - eta*d4;
 b5 = b5 - eta*d5;
end
MISE1 = 0;
MISE2 = 0;
count = 0;
while count < 101
    i = randi(1000);
    j = randi(1000);
    if ~isnan(vq(i,j)) %checking if generated point is
        inside the convex hull
        MISE1 = MISE1 + (vq(i,j)-(0.3+exp((i+j)*0.001)
            /12))^2;
        MISE2 = MISE2 + (f(W2,W3,W4,W5,b2,b3,b4,b5,i
            *0.001,j*0.001)-(0.3+exp((i+j)*0.001)/12))^2;
        count = count+1;
    end
end
MISE1=MISE1/100;
MISE2=MISE2/100;
mises1 = [mises1 MISE1];
mises2 = [mises2 MISE2];

end
bla
medel1 = mean(mises1)
medel2 = mean(mises2)
end

function val = f(W2,W3,W4,W5,b2,b3,b4,b5 ,x,y)

 z =[x ; y];
 a2 = act(z,W2,b2);
 a3 = act(a2,W3,b3);
 a4 = act(a3,W4,b4);
 val = act(a4,W5,b5);
end

 function y = act(x,W,b)

 y = asinh(W*x+b);

 end

 function g = der(y)
```

```
        g = 1./sqrt(1+sinh(y).^2);
end
```

MATLAB code of cubic interpolation for built-in function griddata

```

function [xq,yq,vq] = griddata(varargin)
%GRIDDATA Interpolates scattered data - generally to produce gridded data
%   Vq = griddata(X,Y,V,Xq,Yq) fits a surface of the form V = F(X,Y) to the
%   scattered data in (X, Y, V). The coordinates of the data points are
%   defined by the vectors (X,Y) and V defines the corresponding values.
%   griddata interpolates the surface F at the query points (Xq,Yq) and
%   returns the values in Vq. The query points (Xq, Yq) generally represent
%   a grid obtained from NDGRID or MESHGRID, hence the name GRIDDATA.
%
%   Vq = griddata(X,Y,Z,V,Xq,Yq,Zq) fits a hyper-surface of the form
%   V = F(X,Y,Z) to the scattered data in (X, Y, Z, V). The coordinates of
%   the data points are defined by the vectors (X,Y,Z) and V defines the
%   corresponding values. griddata interpolates the surface F at the query
%   points (Xq,Yq,Zq) and returns the values in Vq.
%
%   Vq = griddata(X,Y,V, xq, yq) where xq is a row vector and yq is a
%   column vector, expands (xq, yq) via [Xq, Yq] = meshgrid(xq,yq).
%   [Xq, Yq, Vq] = griddata(X,Y,V, xq, yq) returns the grid coordinates
%   arrays in addition.
%   Note: The syntax for implicit meshgrid expansion of (xq, yq) will be
%   removed in a future release.
%
%   GRIDDATA(..., METHOD) where METHOD is one of
```

```
%       'nearest'   - Nearest neighbor interpolation
%       'linear'    - Linear interpolation (default)
%       'natural'   - Natural neighbor interpolation
%       'cubic'     - Cubic interpolation (2D only)
%       'v4'        - MATLAB 4 griddata method (2D only)
%   defines the interpolation method. The 'nearest' and 'linear' methods
%   have discontinuities in the zero-th and first derivatives respectively,
%   while the 'cubic' and 'v4' methods produce smooth surfaces.  All the
%   methods except 'v4' are based on a Delaunay triangulation of the data.
%
%   Example 1:
%      % Interpolate a 2D scattered data set over a uniform grid
%      xy = -2.5 + 5*gallery('uniformdata',[200 2],0);
%      x = xy(:,1); y = xy(:,2);
%      v = x.*exp(-x.^2-y.^2);
%      [xq,yq] = meshgrid(-2:.2:2, -2:.2:2);
%      vq = griddata(x,y,v,xq,yq);
%      mesh(xq,yq,vq), hold on, plot3(x,y,v,'o'), hold off
%
%   Example 2:
%      % Interpolate a 3D data set over a grid in the x-y (z=0) plane
%      xyz = -1 + 2*gallery('uniformdata',[5000 3],0);
%      x = xyz(:,1); y = xyz(:,2); z = xyz(:,3);
%      v = x.^2 + y.^2 + z.^2;
%      d = -0.8:0.05:0.8;
%      [xq,yq,zq] = meshgrid(d,d,0);
%      vq = griddata(x,y,z,v,xq,yq,zq);
%      surf(xq,yq,vq);
%
%   See also scatteredInterpolant, GRIDDATAN, MESHGRID, NDGRID, DELAUNAY,
%   INTERPN.

%   Copyright 1984-2015 The MathWorks, Inc.

narginchk(5,9);

numarg = nargin;
method = 'linear';
```

```
if iscell(varargin{numarg})
    error(message('MATLAB:griddata:DeprecatedOptions'));
elseif ischar(varargin{numarg}) || (isstring(varargin{numarg}) && isscalar(varargin{numarg}))
    method = varargin{numarg};
    method = lower(method);
    numarg = numarg-1;
end

if ~any(strcmp(method, {'nearest', 'linear', 'natural', 'cubic', 'v4'}))
    error(message('MATLAB:griddata:UnknownMethod'));
end

if  numarg == 5
    numdims = 2;
elseif numarg == 7
    numdims = 3;
else
    error(message('MATLAB:griddata:InvalidNumInputArgs'));
end

for i=1:(2*numdims)+1
    if (i ~= (numdims+1) && ~isreal(varargin{i}) )
        error(message('MATLAB:griddata:InvalidCoordsComplex'));
    elseif ~isnumeric(varargin{i})
        error(message('MATLAB:griddata:InvalidInputArgs'));
    end
end


for i=1:numarg
    if ndims(varargin{i}) > numdims
        error(message('MATLAB:griddata:HigherDimArray'));
    elseif ( issparse(varargin{i}) )
        error(message('MATLAB:griddata:InvalidDataSparse'));
    end
end

if  numarg == 5
    % potentially 2D validate the data
    % The xyzchk generates a meshgrid - support for this
```

```
        will be removed
    % in a future release.
    x = varargin{1};
    y = varargin{2};
    v = varargin{3};
    xq = varargin{4};
    yq = varargin{5};
    [msg,x,y,~,xq,yq] = xyzchk(x,y,v,xq,yq);
    if ~isempty(msg), error(message(msg.identifier)); end
    inputargs = {x,y,v,xq,yq};
elseif numarg == 7
    % Potentially 3D, check support for the method
    inputargs = varargin;
    if strcmp(method, 'cubic')
        error(message('MATLAB:griddata:CubicMethod3D'));
    elseif strcmp(method, 'v4')
        error(message('MATLAB:griddata:V4Method3D'));
    end
end

switch method
    case 'nearest'
        vq = useScatteredInterp(inputargs, numarg, method, 'nearest');
    case {'linear', 'natural'}
        vq = useScatteredInterp(inputargs, numarg, method, 'none');
    case 'cubic'
        vq = cubic(x,y,v,xq,yq);
    case 'v4'
        vq = gdatav4(x,y,v,xq,yq);
end

if nargout<=1, xq = vq; end

% ---------------------------------------------------------------------

function [x, y, v] = mergepoints2D(x,y,v)

% Sort x and y so duplicate points can be averaged

%Need x,y and z to be column vectors
sz = numel(x);
x = reshape(x,sz,1);
```

```
y = reshape(y,sz,1);
v = reshape(v,sz,1);
myepsx = eps(0.5 * (max(x) - min(x)))^(1/3);
myepsy = eps(0.5 * (max(y) - min(y)))^(1/3);


% look for x, y points that are indentical (within a tolerance)
% average out the values for these points
if isreal(v)
    xyv = builtin('_mergesimpts', [y, x, v], [myepsy, myepsx, Inf], 'average');
    x = xyv(:,2);
    y = xyv(:,1);
    v = xyv(:,3);
else
    % if z is imaginary split out the real and imaginary parts
    xyv = builtin('_mergesimpts', [y, x, real(v), imag(v)], ...
        [myepsy, myepsx, Inf, Inf], 'average');
    x = xyv(:,2);
    y = xyv(:,1);
    % re-combine the real and imaginary parts
    v = xyv(:,3) + 1i*xyv(:,4);
end
% give a warning if some of the points were duplicates (and averaged out)
if sz>numel(x)
    warning(message('MATLAB:griddata:DuplicateDataPoints'));
end

% ----------------------------------------------------------------------

function vq = useScatteredInterp(inargs, numarg, method, emeth)

% Reference (nearest, linear):
%    David F. Watson, "Contouring: A guide to the analysis and display
%       of spacial data", Pergamon, 1994.
%
% Reference (natural):
```

```
%    Sibson, R. (1981). "A brief description of natural neighbor
%        interpolation (Chapter 2)". In V. Barnett. Interpreting
%        Multivariate Data. Chichester: John Wiley. pp. 21--36.

if numarg == 5
    F = scatteredInterpolant(inargs{1}(:),inargs{2}(:), inargs{3}(:), ...
         method, emeth);
    vq = F(inargs{4},inargs{5});
elseif numarg == 7
    F = scatteredInterpolant(inargs{1}(:),inargs{2}(:), inargs{3}(:), ...
         inargs{4}(:), method, emeth);
    vq = F(inargs{5},inargs{6},inargs{7});
end

% ------------------------------------------------------------------------

function vq = cubic(x,y,v,xq,yq)
%TRIANGLE Triangle-based cubic interpolation

%   Reference: T. Y. Yang, "Finite Element Structural Analysis",
%   Prentice Hall, 1986.  pp. 446-449.
%
%   Reference: David F. Watson, "Contouring: A guide
%   to the analysis and display of spacial data", Pergamon, 1994.

% Triangulate the data

[x, y, v] = mergepoints2D(x,y,v);

dt = delaunayTriangulation(x,y);
scopedWarnOff = warning('off', 'MATLAB:triangulation:EmptyTri2DWarnId');
restoreWarnOff = onCleanup(@()warning(scopedWarnOff));
dtt = dt.ConnectivityList;
if isempty(dtt)
    warning(message('MATLAB:griddata:EmptyTriangulation'));
```

```
    vq = [];
    return
end

tri = dt.ConnectivityList;
% Find the enclosing triangle (t)
siz = size(xq);
t = dt.pointLocation(xq(:),yq(:));
t = reshape(t,siz);

if(isreal(v))
    vq = cubicmx(x,y,v,xq,yq,tri,t);
else
    vre = real(v);
    vim = imag(v);
    vqre = cubicmx(x,y,vre,xq,yq,tri,t);
    vqim = cubicmx(x,y,vim,xq,yq,tri,t);
    vq = complex(vqre,vqim);
end

% ------------------------------------------------------------------------

function vq = gdatav4(x,y,v,xq,yq)
%GDATAV4 MATLAB 4 GRIDDATA interpolation

%   Reference:  David T. Sandwell, Biharmonic spline
%   interpolation of GEOS-3 and SEASAT altimeter
%   data, Geophysical Research Letters, 2, 139-142,
%   1987.  Describes interpolation using value or
%   gradient of value in any dimension.

[x, y, v] = mergepoints2D(x,y,v);

xy = x(:) + 1i*y(:);

% Determine distances between points
d = abs(xy - xy.');

% Determine weights for interpolation
g = (d.^2) .* (log(d)-1);   % Green's function.
% Fixup value of Green's function along diagonal
g(1:size(d,1)+1:end) = 0;
weights = g \ v(:);

```

```
[m,n] = size(xq);
vq = zeros(size(xq));
xy = xy.';

% Evaluate at requested points (xq,yq).  Loop to save
    memory.
for i=1:m
    for j=1:n
        d = abs(xq(i,j) + 1i*yq(i,j) - xy);
        g = (d.^2) .* (log(d)-1);   % Green's function.
        % Value of Green's function at zero
        g(d==0) = 0;
        vq(i,j) = g * weights;
    end
end
```