Interpolants and Error Bounds for Modeling and Predicting Variability in Computer Systems

Thomas C. H. Lux

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

Layne Watson, Co-chair

Yili Hong, Co-chair

Kirk Cameron

Bert Huang

Gang Wang

Yang Cao

May 16, 2019

Blacksburg, Virginia

Keywords: Approximation Theory, Numerical Analysis, High Performance Computing, Computer Security, Nonparametric Statistics, Mathematical Software

Interpolants and Error Bounds for Modeling and Predicting Variability in Computer Systems

Thomas C. H. Lux

(ABSTRACT)

Function approximation is an important problem. This work presents applications of interpolants to modeling random variables. Specifically, this work studies the prediction of distributions of random variables applied to computer system throughput variability. New methods for interpolating data with moderately high dimension are presented and novel theoretical error bounds are constructed for piecewise linear interpolation. Finally, a mathematical software for approximating distributions from data samples via monotone quintic splines is proposed.

Interpolants and Error Bounds for Modeling and Predicting Variability in Computer Systems

Thomas C. H. Lux

(GENERAL AUDIENCE ABSTRACT)

It is common for scientists to collect data on something they are studying. Often scientists want to create a (predictive) model of that phenomenon based on the data, but the choice of *how* to model the data is a difficult one to answer. This work proposes methods for modeling data that operate under very few assumptions that are broadly applicable across science. Finally, a software package is proposed that would allow scientists to better understand the *true* distribution of their data given relatively few observations.

Contents

List of Figures

List of Tables

Chapter 1

The Importance and Applications of Variability

Computational variability presents itself in a variety of forms. Processes that are apparently identical in a cloud computing or high performance computing (HPC) environment may take different amounts of time to complete the same job. This variability can cause unintentional violations of service level agreements in cloud computing applications or indicate suboptimal performance in HPC applications. The sources of variability, however, are distributed throughout the system stack and often difficult to identify. The methodology presented in this work is applicable to modeling the expected variability of useful computer system performance measures without any prior knowledge of system architecture. Some examples of interesting performance measures that could be modeled with the techniques in this work include computational throughput, power consumption, processor idle time, number of context switches, and RAM usage, as well as any other numeric measure of performance.

Predicting performance variability in a computer system is a challenging problem that has primarily been attempted in one of two ways: (1) build a statistical model of the performance data collected by running experiments on the system at select settings, or (2) run artificial experiments using a simplified simulation of the target system to estimate architecture and application bottlenecks. In this work, modeling techniques rest in the first category and represent a notable increase in the ability to model precise characteristics of variability.

Many previous works attempting to model system performance have used simulated environments [33, 62, 63]. Grobelny et al. [33] refer to statistical models as being oversimplified and not capable of capturing the true complexity of the underlying system. Historically, statistical models have been limited to modelling at most one or two system parameters and have therefore not been capable of modeling the complexity of the underlying system. [4, 5, 57, 65]. These limited statistical models have provided satisfactory performance in narrow application settings. However these techniques require additional code annotations, hardware abstractions, or additional assumptions about the behavior of the application in order to generate models. In contrast, the approaches that are presented here require no modifications of applications, no architectural abstractions, nor any structural descriptions of the input data being modeled. The techniques used in this work only require performance data as input.

Most existing work on performance variability has focused on operating system (OS) induced variability [7, 19]. Yet, system I/O variability has been particularly difficult for statistical models to capture [4]. The prior work attempting to model I/O variability has been similarly limited to one or two system parameters [42].

Chapters 3 and 5 present and evaluate applications of multivariate interpolation to the domain of computer system I/O throughput. Interpolants and regressors are used to predict the different measures of variability (e.g., mean, variance, cumulative distribution functions) for the expected I/O throughput on a system with previously unseen configurations. Beyond these I/O case studies, the techniques discussed can tractably model tens of interacting system parameters with tens of thousands of known configurations. A major contribution of this work is a modeling framework that uses multivariate interpolation to capture precise characteristics (via cumulative distribution functions) of arbitrary performance measure on any type of computer system.

1.1 Broader Applications of Approximation

Regression and interpolation are problems of considerable importance that find applications across many fields of science. Pollution and air quality analysis [22], energy consumption management [39], and student performance prediction [16, 43] are a few of many interdisciplinary applications of multivariate regression for predictive analysis. As discussed later, these techniques can also be applied to prediction problems related to forest fire risk assessment [15], Parkinson's patient clinical evaluations [59], local rainfall and weather [64], credit card transactions [50], and high performance computing (HPC) file input/output (I/O) [44].

Regression and interpolation have a considerable theoretical base in one dimension [12]. Splines in particular are well understood as an interpolation technique in one dimension [20], particularly B-splines. Tensor products of B-splines [61] or other basis functions have an unfortunate exponential scaling with increasing dimension. Exponential scaling prohibits tensor products from being reasonably applied beyond three-dimensional data. In order to address this dimensional scaling challenge, C. de Boor and others proposed box splines [21], of which one of the approximation techniques in this work is composed [45].

The theoretical foundation of low dimensional interpolation allows the construction of strong error bounds that are absent from high dimensional problems. Chapter 6 extends some known results regarding the secant method [23] to construct an interpolation error bound for problems of arbitrary dimension. These error bounds are useful, considering the same cannot be said for regression algorithms in general. The maximum complexity of an interpolant is bounded by the amount of data available, while the maximum complexity of a regressor is bounded by both the amount of data and the chosen parametric form. For this reason, generic uniform bounds are largely unobtainable for regression techniques on arbitrary approximation problems, even when the approximation domain is bounded. These generic bounds imply that interpolants may be suitable for a broader class of

approximation problems than (heuristically chosen) regression techniques.

Aside from theoretical motivation for the use of interpolants, there are often computational advantages as well. Interpolants do not have the need for *fitting* data, or minimizing error with respect to model parameters. In applications where the amount of data is large and the relative number of predictions that need to be made for a given collection of data is small, the direct application of an interpolant is much less computationally expensive.

In this work, multivariate interpolation is defined given a closed convex subset Y of a metrizable topological vector space with metric s, some function $f : \mathbb{R}^d \to Y$ and a set of points $X = \left\{x^{(1)}, \ldots, x^{(n)}\right\} \subset \mathbb{R}^d$, along with associated function values $f\left(x^{(i)}\right)$. The goal is to construct an approximation $\hat{f} : \mathbb{R}^d \to Y$ such that $\hat{f}\left(x^{(i)}\right) = f\left(x^{(i)}\right)$ for all $i = 1, \ldots, n$. It is often the case that the form of the true underlying function f is unknown, however it is still desirable to construct an approximation $\hat{f}$ with small approximation error at $y \notin X$. The two metric spaces that will be discussed in this work are the real numbers with metric $s(x,y) = |x-y|$, and the set of cumulative distribution functions (CDFs) with the Kolmogorov-Smirnov (KS) statistic [41] as metric.

Multivariate regression is often used when the underlying function is presumed to be stochastic, or stochastic error is introduced in the evaluation of f. Hence, multivariate regression relaxes the conditions of interpolation by choosing parameters P defining $\hat{f}(x;P)$ to minimize the error vector $\left(\left|\hat{f}\left(x^{(1)};P\right) - f\left(x^{(1)}\right)\right|, \ldots, \left|\hat{f}\left(x^{(n)};P\right) - f\left(x^{(n)}\right)\right|\right)$ in some norm. The difficult question in the case of regression is often what parametric form to adopt for any given application.

The most challenging problem when scaling in dimension is that the number of possible interactions between dimensions grows exponentially. Quantifying all possible interactions becomes intractable, and hence beyond three-dimensional data mostly linear models are used. That is not to say nonlinear models are absent, but nonlinearities are often either preconceived or model pairwise interactions between dimensions at most. Even globally nonlinear approximations such as neural

networks are constructed from compositions of summed low-interaction functions [14].

Provided the theoretical and practical motivations for exploring interpolants, this work aims to study the empirical performance differences between a set of scalable (moderately) interpolation techniques and a set of common regression techniques. These techniques are applied to a collection of moderately high dimensional problems ($5 \leq d \leq 30$) and the empirical results are discussed.

1.2 An Initial Application of Approximation

Performance tuning is often an experimentally complex and time intensive chore necessary for configuring High Performance Computing (HPC) systems. The procedures for this tuning vary largely from system to system and are often subjectively guided by the system engineer(s). Once a desired level of performance is achieved, an HPC system may only be incrementally reconfigured as required by updates or specific jobs. In the case that a system has changing workloads or nonstationary performance objectives that range from maximizing computational throughput to minimizing power consumption and system variability, it becomes clear that a more effective and automated tool is needed for configuring systems. This scenario presents a challenging and important application of multivariate approximation and interpolation techniques.

Predicting the performance of an HPC system is a challenging problem that is primarily attempted in one of two ways: (1) build a statistical model of the performance by running experiments on the system at select settings, or (2) run artificial experiments using a simplified simulation of the target system to estimate architecture and application bottlenecks. In this paper the proposed multivariate modeling techniques rest in the first category, and they represent a notable increase in the ability to model complex interactions between system parameters.

Many previous works attempting to model system performance have used simulated environments

to estimate the performance of a system [33, 62, 63]. Some of these works refer to statistical models as being oversimplified and not capable of capturing the true complexity of the underlying system. This claim is partially correct, noting that a large portion of predictive statistical models rely on simplifying the model to one or two parameters [4, 5, 57, 65]. These limited statistical models have provided satisfactory performance in very narrow application settings. Many of the aforementioned statistical modeling techniques claim to generalize, while simultaneously requiring additional code annotations, hardware abstractions, or additional application level understandings in order to generate models. The approach presented here requires no modifications of the application, no architectural abstractions, nor any structural descriptions of the input data being modeled. The techniques used are purely mathematical and only need performance data as input.

Among the statistical models presented in prior works, [4] specifically mention that it is difficult for the simplified models to capture variability introduced by I/O. System variability in general has become a problem of increasing interest to the HPC and systems communities, however most of the work has focused on operating system (OS) induced variability [7, 19]. The work that has focused on managing I/O variability does not use any sophisticated modeling techniques [42]. Hence, Chapter 3 presents a case study applying advanced mathematical and statistical modeling techniques to the domain of HPC I/O characteristics. The models are used to predict the mean throughput of a system and the variance in throughput of a system. The discussion section that follows outlines how the techniques presented can be applied to any performance metric and any system.

Chapter 2

Algorithms for Constructing Approximations

2.1 Multivariate Regression

Multivariate regressors are capable of accurately modeling a complex dependence of a response (in Y) on multiple variables (represented as a points in $\mathbb{R}^d$). The approximations to some (unknown) underlying function $f : \mathbb{R}^d \to Y$ are chosen to minimize some error measure related to data samples $f\left(x^{(i)}\right)$. For example, least squares regression uses the sum of squared differences between modeled function values and true function values as an error measure. In this section and the next, some techniques are limited to approximating real valued functions ($Y \subset \mathbb{R}$). These techniques can be extended to real vector-valued ranges by repeating the construction for each component of the vector output. Throughout the following, x denotes a d-tuple, x_i the ith component of x, and $x^{(i)}$ the ith d-tuple data point. Different symbols are used to represent the approximation function $\hat{f}$.

2.1.1 Multivariate Adaptive Regression Splines

This approximation was introduced in [27] and subsequently improved to its current version in [28], called fast multivariate adaptive regression splines (Fast MARS). In Fast MARS, a least squares fit model is iteratively built by beginning with a single constant valued function and adding

two new basis functions at each iteration of the form

$$B_{2j-1}(x) = B_l(x)\left(x_i - x_i^{(p)}\right)_+,$$
$$B_{2j}(x) = B_k(x)\left(x_i - x_i^{(p)}\right)_-,$$

where j is the iteration number, $1 \leq p \leq n$, and $B_l(x)$, $B_k(x)$ are basis functions from the previous iteration,

$$w_+ = \begin{cases} w, & w \geq 0 \\ 0, & w < 0 \end{cases},$$

and $w_- = (-w)_+$. After iteratively constructing a model, MARS then iteratively removes basis functions that do not contribute to goodness of fit. In effect, MARS creates a locally component-wise tensor product approximation of the data. The overall computational complexity of Fast MARS is $\mathcal{O}(ndm^3)$ where m is the maximum number of underlying basis functions. This paper uses an implementation of Fast MARS [54] with $m = 200$ throughout all experiments.

2.1.2 Support Vector Regressor

Support vector machines are a common method used in machine learning classification tasks that can be adapted for the purpose of regression [6]. How to build a support vector regressor (SVR) is beyond the scope of this summary, but the resulting functional fit $p : \mathbb{R}^d \to \mathbb{R}$ has the form

$$p(x) = \sum_{i=1}^{n} a_i K\left(x, x^{(i)}\right) + b,$$

where K is the selected kernel function, $a_i \in \mathbb{R}^n$, $b \in \mathbb{R}$ are coefficients to be solved for simultaneously. The computational complexity of the SVR is $\mathcal{O}(n^2dm)$, with m being determined by the minimization convergence criterion. This paper uses the scikit-learn SVR [49] with a polynomial

kernel function.

2.1.3 Multilayer Perceptron Regressor

The neural network is a well studied and widely used method for both regression and classification tasks [36, 55]. When using the rectified linear unit (ReLU) activation function [18] and fitting the model with stochastic gradient descent (SGD) or BFGS minimization techniques [30, 46, 53], the model built by a multilayer perceptron uses layers $l : \mathbb{R}^i \to \mathbb{R}^j$ defined by

$$l(u) = \left(u^t W_l + c_l\right)_+,$$

where $c_l \in \mathbb{R}^j$ and W_l is the $i \times j$ weight matrix for layer l. In this form, the multilayer perceptron (MLP) produces a piecewise linear model of the input data. The computational complexity of training a multilayer perceptron is $\mathcal{O}(ndm)$, where m is determined by the sizes of the layers of the network and the stopping criterion of the minimization used for finding weights. This paper uses an MLP built from Keras and Tensorflow to perform regression [1, 13] with ten hidden layers each having thirty two nodes (a total of approximately ten thousand parameters), ReLU activation, and five thousand epochs of SGD for training.

2.2 Multivariate Interpolation

The following interpolation techniques demonstrate a reasonable variety of approaches to interpolation. All of the presented interpolants produce approximations that are continuous in value, which is often a desirable property in applied approximation problems.

2.2.1 Delaunay

The Delaunay triangulation is a well-studied geometric technique for producing an interpolant [40]. The Delaunay triangulation of a set of data points into simplices is such that there are no data points inside the sphere defined by the vertices of each simplex. For a d-simplex S with vertices $x^{(0)}$, $x^{(1)}$, ..., $x^{(d)}$, and function values $f\left(x^{(i)}\right)$, $i = 0, \ldots, d$, $y \in S$ is a unique convex combination of the vertices,

$$y = \sum_{i=0}^{d} w_i x^{(i)}, \quad \sum_{i=0}^{d} w_i = 1, \quad w_i \geq 0, \quad i = 0, \ldots, d,$$

and the Delaunay interpolant $\hat{f}(y)$ at y is given by

$$\hat{f}(y) = \sum_{i=0}^{d} w_i f\left(x^{(i)}\right).$$

The computational complexity of Delaunay interpolation (for the implementation used [11]) is $\mathcal{O}(knd^3)$. Given pathological data the entire triangulation could be computed with $k = n^{\lceil d/2 \rceil}$, but the average case yields $k = d \log d$ and is capable of scaling reasonably to $d \leq 50$. In the present application, a Delaunay simplex S containing y is found, then the $d+1$ vertices (points in X) of S are used to assign weights to each vertex and produce the predicted function value for point y.

2.2.2 Modified Shepard

The modified Shepard method used here (ShepMod) generates a continuous approximation based on Euclidean distance and resembles a nearest neighbor interpolant [17]. This model is a type of *metric interpolation*, also called a Shepard method [31, 56]. The interpolant has the form

$$p(x) = \frac{\sum_{k=1}^{n} W_k(x) f\left(x^{(k)}\right)}{\sum_{k=1}^{n} W_k(x)},$$

where $W_k(x) = \left(r_k - \left\|x - x^{(k)}\right\|\right)_+^2 / \left\|x - x^{(k)}\right\|_2^2$ and $r_k \in \mathbb{R}$ is the smallest radius such that at least $d+1$ other points $x^{(j)}$, $j \neq k$, are inside the closed Euclidean ball of radius r_k about $x^{(k)}$. The computational complexity of ShepMod is $\mathcal{O}(n^2 d)$. This paper uses a Fortran 95 implementation of ShepMod derived from SHEPPACK [58].

2.2.3 Linear Shepard

The linear Shepard method (LSHEP) is a blending function using local linear interpolants, a special case of the general Shepard algorithm [58]. The interpolant has the form

$$p(x) = \frac{\sum_{k=1}^{n} W_k(x) P_k(x)}{\sum_{k=1}^{n} W_k(x)},$$

where $W_k(x)$ is a locally supported weighting function and $P_k(x)$ is a local linear approximation to the data satisfying $P_k\left(x^{(k)}\right) = f\left(x^{(x)}\right)$. The computational complexity of LSHEP is $\mathcal{O}(n^2 d^3)$. This paper uses the Fortran 95 implementation of LSHEP in SHEPPACK [58].

Chapter 3

Naïve Approximations of Variability

This chapter compares five of the multivariate approximation techniques that operate on inputs in $\mathbb{R}^d$ (d-tuples of real numbers) and produce predicted responses in $\mathbb{R}$. Three of the chosen techniques are regression based and the remaining two are interpolants, providing reasonable coverage of the varied mathematical strategies that can be employed to solve continuous modeling problems.

3.1 I/O Data

In order to evaluate the viability of multivariate models for predicting system performance, this paper presents a case study of a four-dimensional dataset produced by executing the IOzone benchmark from Norcott [47] on a homogeneous cluster of computers. Multiple I/O Zone data sets will be used throughout the work, however this chapter relies on this specific four-dimensional dataset. All experiments were performed on parallel shared-memory nodes common to HPC systems. Each system had a lone guest Linux operating system (Ubuntu 14.04 LTS//XEN 4.0) on a dedicated 2TB HDD on a 2 socket, 4 core (2 hyperthreads per core) Intel Xeon E5-2623 v3 (Haswell) platform with 32 GB DDR4. The system performance data was collected by executing IOzone 40 times for each of a select set of system configurations. A single IOzone execution reports the max I/O throughput seen for the selected test in kilobytes per second. The 40 executions for each system configuration are converted into the mean and variance, both values in $\mathbb{R}$ capable of being modeled individually by the multivariate approximation techniques presented in Chapter 2. The summary

System Parameter	Values
File Size	64, 256, 1024
Record Size	32, 128, 512
Thread Count	1, 2, 4, 8, 16, 32, 64, 128, 256
Frequency	$\{12, 14, 15, 16, 18, 19, 20, 21, 23, 24, 25, 27, 28, 29, 30, 30.01\} \times 10^5$
Response Values	
Throughput Mean	$[2.6 \times 10^5, 5.9 \times 10^8]$
Throughput Variance	$[5.9 \times 10^{10}, 4.7 \times 10^{16}]$

Table 3.1: A description of the system parameters being considered in the IOzone tests. Record size must not be greater than file size and hence there are only six valid combinations of the two. In total there are $6 \times 9 \times 16 = 864$ unique system configurations.

of data used in the experiments for this chapter can be seen in Table 3.1. Distributions of raw throughput values being modeled can be seen in Figure 3.1.

3.1.1 Dimensional Analysis

This analysis utilizes an extension to standard k-fold cross validation that allows for a more thorough investigation of the expected model performance in a variety of real-world situations. Alongside randomized splits, two extra components are considered: the amount of training data provided, and the dimension of the input data. It is important to consider that algorithms that perform well with less training input also require less experimentation. Although, the amount of training data required may change as a function of the dimension of the input and this needs to be studied as well. The framework used here will be referred to as a multidimensional analysis (MDA) of the IOzone data.

Multidimensional Analysis

This procedure combines random selection of training and testing splits with changes in the input dimension and the ratio of training size to testing size. Given an input data matrix with n rows

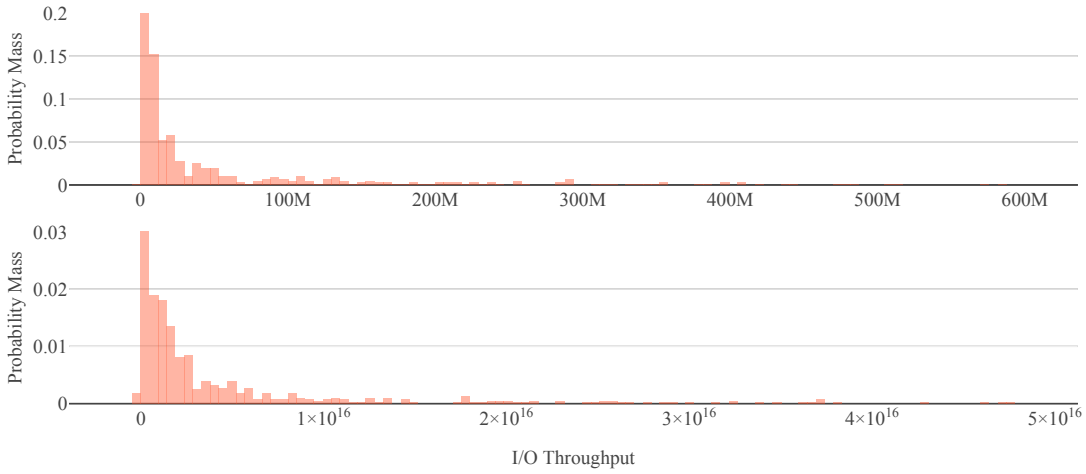

Figure 3.1: Histograms of 100-bin reductions of the PMF of I/O throughput mean (top) and I/O throughput variance (bottom). In the mean plot, the first 1% bin (truncated in plot) has a probability mass of .45. In the variance plot, the second 1% bin has a probability mass of .58. It can be seen that the distributions of throughputs are primarily of lower magnitude with occasional extreme outliers.

(points) and d columns (components), MDA proceeds as follows:

1. For all $k = 1, \ldots, d$ and for all nonempty subsets $F \subset \{1, 2, \ldots, d\}$, reduce the input data to points $(z, f_F(z))$ with $z \in \mathbb{R}^k$ and $f_F(z) = E\left[\left\{f\left(x^{(i)}\right) \mid \left(x_F^{(i)} = z\right)\right\}\right]$, where $E[\cdot]$ denotes the mean and $x_F^{(i)}$ is the subvector of $x^{(i)}$ indexed by F.

2. For all r in $\{5, 10, \ldots, 95\}$, generate N random splits $(train, test)$ of the reduced data with r percentage for training and $100 - r$ percentage for testing.

3. When generating each of N random $(train, test)$ splits, ensure that all points from $test$ are in the convex hull of points in $train$ (to prevent extrapolation); also ensure that the points in $train$ are well spaced.

In order to ensure that the testing points are in the convex hull of the training points, the convex hull vertices of each set of (reduced dimension) points are forcibly placed in the training set. In

order to ensure that training points are well spaced, a statistical method for picking points from Amos et al. [3] is used:

1. Generate a sequence of all pairs of points $\left(z^{(i_1)}, z^{(j_1)}\right), \left(z^{(i_2)}, z^{(j_2)}\right), \ldots$ sorted by ascending pairwise Euclidean distance between points, so that $\left|\left|z^{(i_k)} - z^{(j_k)}\right|\right|_2 \leq \left|\left|z^{(i_{k+1})} - z^{(j_{k+1})}\right|\right|_2$.

2. Sequentially remove points from candidacy until only $|train|$ remain by randomly selecting one point from the pair $\left(z^{(i_m)}, z^{(j_m)}\right)$ for $m = 1, \ldots$ if both $z^{(i_m)}$ and $z^{(j_m)}$ are still candidates for removal.

Given the large number of constraints, level of reduction, and use of randomness in the MDA procedure, occasionally N unique training/testing splits may not be created or may not exist. In these cases, if there are fewer than N possible splits, then deterministically generated splits are used. Otherwise after $3N$ attempts, only the unique splits are kept for analysis. The MDA procedure has been implemented in Python#3 while most regression and interpolation methods are Fortran wrapped with Python. All randomness has been seeded for repeatability.

For any index subset F (of size k) and selected value of r, MDA will generate up to N multivariate models $f_F(z)$ and predictions $\hat{f}_F\left(z^{(i)}\right)$ for a point $z^{(i)} \in \mathbb{R}^k$. There may be fewer than N predictions made for any given point. Extreme points of the convex hull for the selected index subset will always be used for training, never for testing. Points that do not have any close neighbors will often be used for training in order to ensure well-spacing. Finally, as mentioned before, some index subsets do not readily generate N unique training and testing splits. The summary results presented in this work use the median of the (N or fewer) values $\hat{f}_F(z)$ at each point as the model estimate for error analysis.

3.2 Naïve Variability Modeling Results

A naïve multivariate prediction technique such as nearest neighbor could experience relative errors in the range $[0, \left(\max_x f(x) - \min_x f(x)\right) / \min_x f(x)]$ when modeling a nonnegative function $f(x)$ from data. The IOzone mean data response values span three orders of magnitude (as can be seen in Table 3.1) while variance data response values span six orders of magnitude. It is expected therefore, that all studied multivariate models perform better than a naïve approach, achieving relative errors strictly less than 10^3 for throughput mean and 10^6 for throughput variance. Ideally, models will yield relative errors significantly smaller than 1. The time required to compute thousands of models involved in processing the IOzone data through MDA was approximately five hours on a CentOS workstation with an Intel i7-3770 CPU at 3.40GHz. In four dimensions for example, each of the models could be constructed and evaluated over hundreds of points in less than a few seconds.

3.2.1 I/O Throughput Mean

Almost all multivariate models analyzed make predictions with a relative error less than 1 for most system configurations when predicting the mean I/O throughput of a system given varying amounts of training data. The overall best of the multivariate models, Delaunay, consistently makes predictions with relative error less than .05 (5% error). In Figure 3.3 it can also be seen that the Delaunay model consistently makes good predictions even with as low as 5% training data (43 of the 864 system configurations) regardless of the dimension of the data.

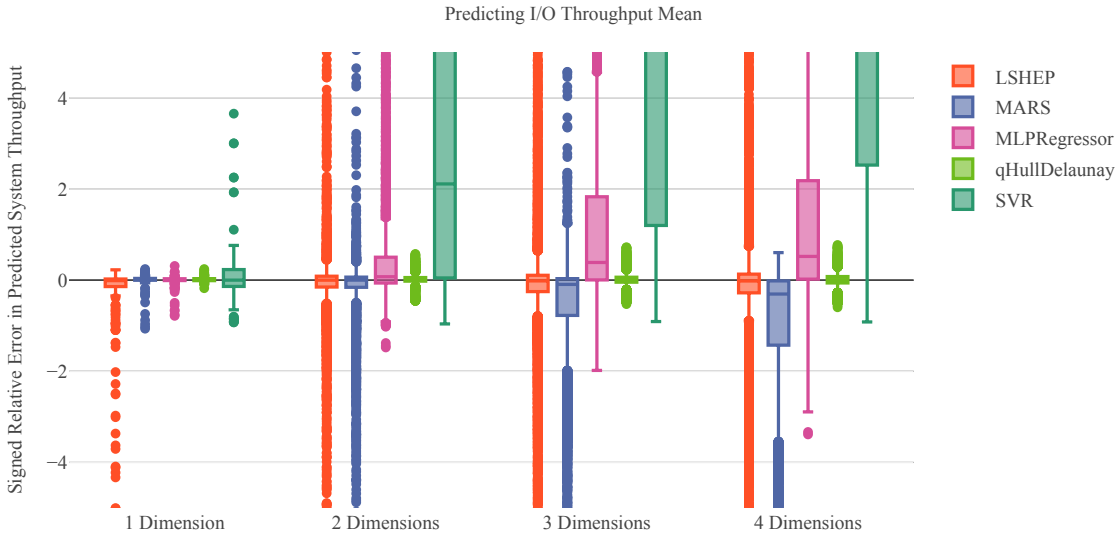

Figure 3.2: These box plots show the prediction error of mean with increasing dimension. The top box whisker for SVR is 40, 80, 90 for dimensions 2, 3, and 4, respectively. Notice that each model consistently experiences greater magnitude error with increasing dimension. Results for all training percentages are aggregated.

3.2.2 I/O Throughput Variance

The prediction results for variance resemble those for predicting mean. Delaunay remains the best overall predictor (aggregated across training percentages and dimensions) with median relative error of .47 and LSHEP closely competes with Delaunay having a median signed relative error of -.92. Outliers in prediction error are much larger for all models. Delaunay produces relative errors as large as 78 and other models achieve relative errors around 10^3. The relative errors for many models scaled proportional to the increased orders of magnitude spanned by the variance response compared with mean response. As can be seen in Figure 3.4, all models are more sensitive to the amount of training data provided than their counterparts for predicting mean.

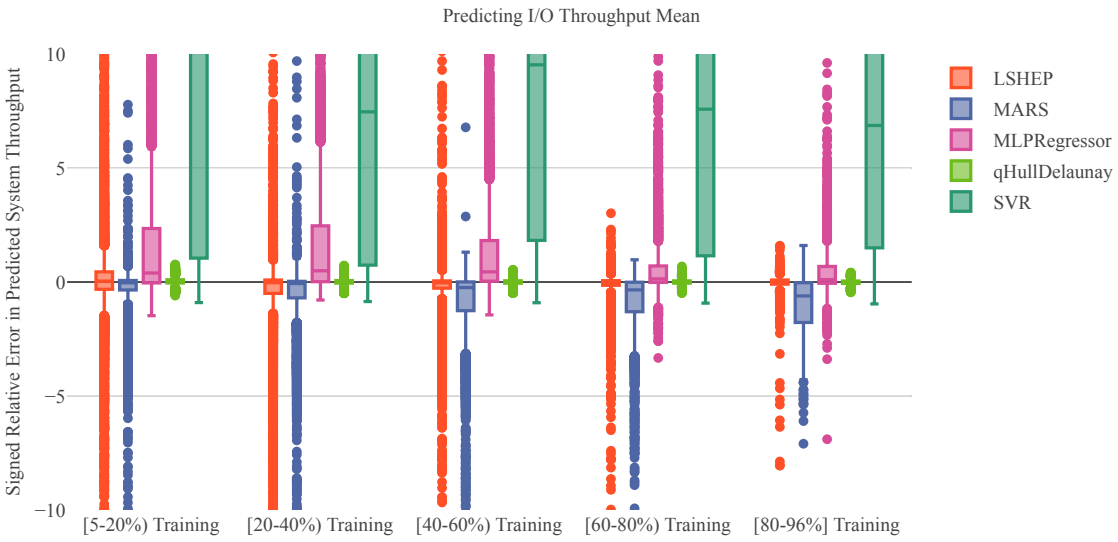

Figure 3.3: These box plots show the prediction error of mean with increasing amounts of training data provided to the models. Notice that MARS is the only model whose primary spread of performance increases with more training data. Recall that the response values being predicted span three orders of magnitude and hence relative errors should certainly remain within that range. For SVR the top box whisker goes from around 100 to 50 from left to right and is truncated in order to maintain focus on better models. Results for all dimensions are aggregated. Max training percentage is 96% due to rounding training set size.

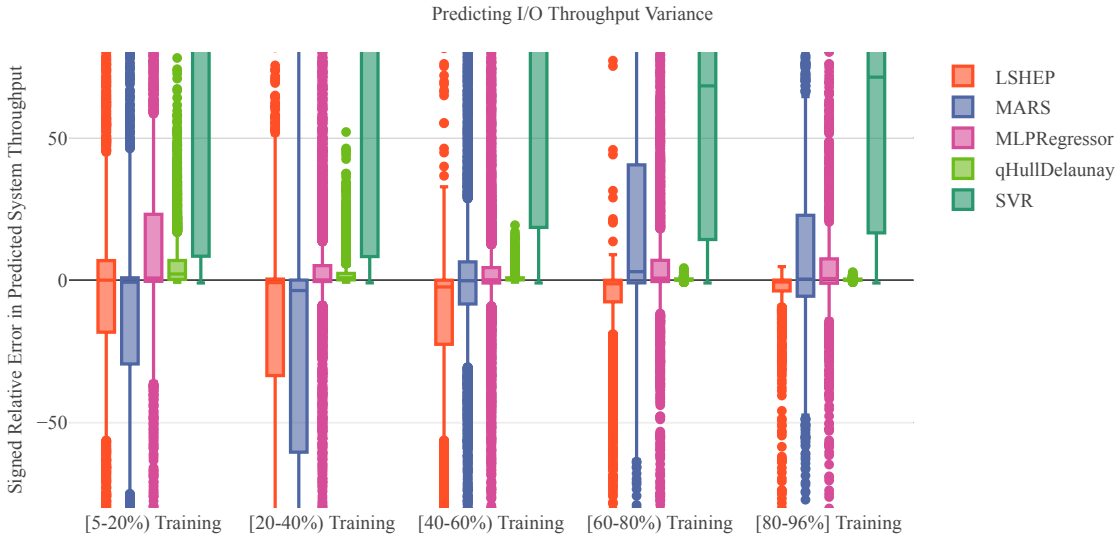

Figure 3.4: These box plots show the prediction error of variance with increasing amounts of training data provided to the models. The response values being predicted span six orders of magnitude. For SVR the top box whisker goes from around 6000 to 400 (decreasing by factors of 2) from left to right and is truncated in order to maintain focus on better models. Results for all dimensions are aggregated. Max training percentage is 96% due to rounding training set size.

3.2.3 Increasing Dimension and Decreasing Training Data

As can be seen in Figure 3.2, all of the models suffer increasing error rates in higher dimension. This is expected, because the number of possible interactions to model grows exponentially. However, LSHEP and Delaunay maintain the slowest increase in relative error. The increase in error seen for Delaunay suggests that it is capable of making predictions with a range of relative errors that grows approximately linearly with increasing dimension input. This trend suggests that Delaunay would remain a viable technique for accurately modeling systems with 10's of parameters given only small amounts of training data. All models, with the exception of MARS, produce smaller errors given more training data. Increasing the amount of training data most notably reduces the number of prediction error outliers.

3.3 Discussion of Naïve Approximations

The results presented above demonstrate that a straightforward application of multivariate modeling techniques can be used to effectively predict HPC system performance. Some modeling effort on the part of a systems engineer combined with a significant amount of experimentation (days of CPU time for the IOzone data used here) can yield a model capable of accurately tuning an HPC system to the desired performance specification, although qualitatively correct predictions can be achieved with much less (10%, say) effort.

3.3.1 Modeling the System

The modeling techniques generated estimates of drastically different quality when predicting I/O throughput mean and variance. A few observations: SVR has the largest range of errors for all selections of dimension and amounts of training data; MARS and LSHEP produce similar magnitude errors while the former consistently underestimates and the latter consistently overestimates; Delaunay has considerably fewer outliers than all other methods. SVR likely produces the poorest quality predictions because the underlying parametric representation is global and oversimplified (a single polynomial), making it unable to capture the complex local behaviors of system I/O. It is still unclear, however, what causes the behaviors of LSHEP, MARS, and Delaunay. An exploration of this topic is left to future work.

The Delaunay method appears to be the best predictor in the present IOzone case study. Particularly a piecewise linear interpolant like Delaunay appears well-suited for prediction when relatively small amounts of data are available to mode la function. It is important to note that the Delaunay computational complexity in the dimension of the input is worse than other techniques.

Finally, the ability of the models to predict variance was significantly worse than for the I/O mean.

The larger scale in variance responses alone do not account for the increase in relative errors witnessed. This suggests that system variability has a greater underlying functional complexity than the system mean and that latent factors are reducing prediction performance.

3.3.2 Extending the Analysis

System I/O throughput mean and variance are simple and useful system characteristics to model. The process presented in this chapter is equally applicable to predicting other useful performance characteristics of HPC systems such as: computational throughput, power consumption, processor idle time, context switches, RAM usage, or any other ordinal performance metric. For each of these there is the potential to model system variability as well. This chapter uses variance as a measure of variability, but the techniques are applied to more precise measures of variability (the entire distribution itself) in Chapter 5.

3.4 Takeaway From Naïve Approximation

Multivariate models of HPC system performance can effectively predict I/O throughput mean and variance. These multivariate techniques significantly expand the scope and portability of statistical models for predicting computer system performance over previous work. In the IOzone case study presented, the Delaunay method produces the best overall results making predictions for 821 system configurations with less than 5% error when trained on only 43 configurations. Analysis also suggests that the error in the Delaunay method will remain acceptable as the number of system parameters being modeled increases. These multivariate techniques should be applied to HPC systems with more than four tunable parameters in order to identify optimal system configurations that may not be discoverable via previous methods nor by manual performance tuning, which will be explored in later chapters.

Chapter 4

Box Splines: Uses, Constructions, and Applications

Chapter 3 demonstrated that the modeling and approximation techniques described in Chapter 2 can be used to model and predict numeric approximations of system variability. The case study was only for four-dimensional data, and the key weakness of the best predictor (Delaunay) is scaling with dimension. This Chapter discusses a modeling approach that may achieve better scaling with dimension, proposes a quasi-mesh construction, and analyzes the performance of this proposed technique.

4.1 Box Splines

A box spline in $\mathbb{R}^d$ is defined by its *direction vector set* A, composed of s d-vectors where $s \geq d$. Further, A will be written as a $d \times s$ matrix. The first m column vectors of A are denoted by A_m, $m \leq s$. A_d is required to be nonsingular. Consider the unit cube in s dimensions $Q_s = [0,1)^s$. $A_s(Q_s)$ is now the image (in d dimensions) of Q_s under the linear map A. This image is the region of support for the box spline defined by A_s in d dimensions. The box spline function in d

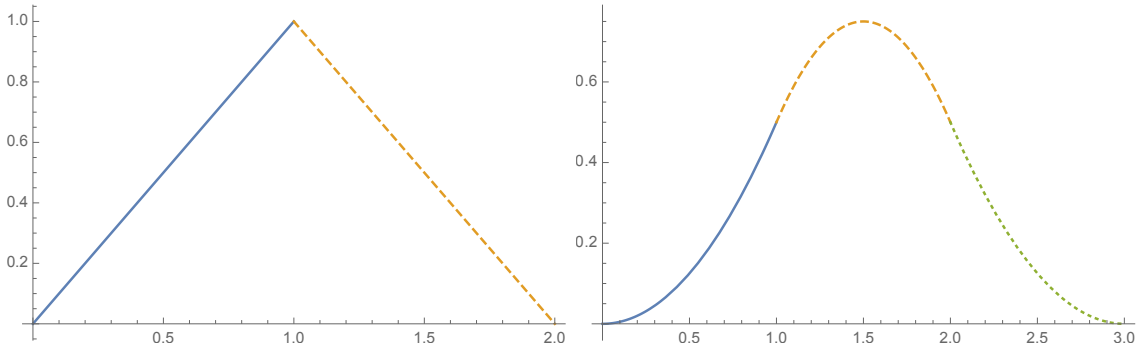

Figure 4.1: 1D linear (order 2) and quadratic (order 3) box splines with direction vector sets $\begin{pmatrix} 1 & 1 \end{pmatrix}$ and $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ respectively. Notice that these direction vector sets form the B-Spline analogues, order 2 composed of two linear components and order 3 composed of 3 quadratic components (colored and styled in plot).

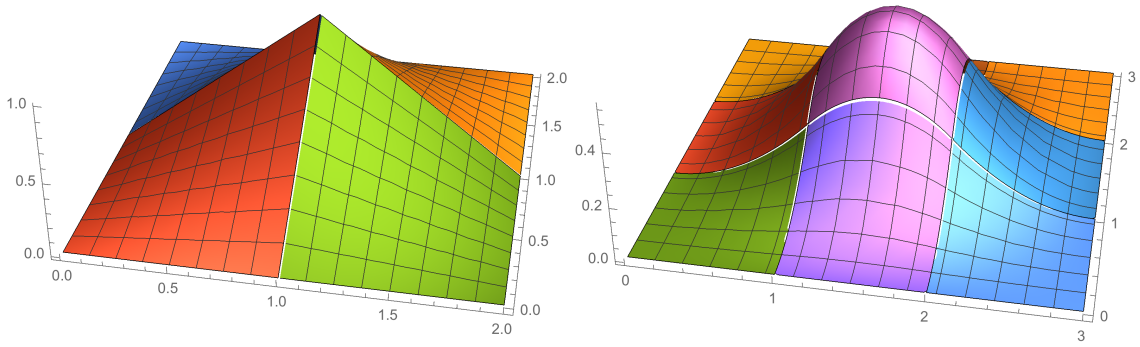

Figure 4.2: 2D linear (order 2) and quadratic (order 3) box splines with direction vector sets $\begin{pmatrix} I & I \end{pmatrix}$ and $\begin{pmatrix} I & I & I \end{pmatrix}$ respectively, where I is the identity matrix in two dimensions. Notice that these direction vector sets also produce boxes with order^2 subregions (colored in plot).

dimensions for A_d is defined as

$$B(x \mid A_d) = \begin{cases} (\det(A_d))^{-1}, & x \in A_d(Q_d), \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

For A_s when $s > d$ the box spline is computed as

$$B(x \mid A_s) = \int_0^1 B(x - tv_s \mid A_{s-1})dt, \tag{4.2}$$

where v_s is the sth direction vector of A.

The application of box splines presented in this chapter always utilizes the d-dimensional identity matrix as A_d. This simplifies the computation in Equation 4.1 to be the characteristic function for the unit cube. Composing A strictly out of k repetitions of the identity matrix forms the kth order B-spline with knots located at 0, 1, …, $k-1$, k along each axis (see Figure 4.1). Furthermore, while the number of subregions for the kth order d-dimensional box spline grows as k^d (see Figure 4.2), the symmetry provided by direction vector sets composed of repeated identity matrices allows the computation of box splines to be simplified. The value of a box spline at any location is then the product of all axis-aligned 1-dimensional kth order box splines.

The box splines as presented are viable basis functions. Each box spline can be shifted and scaled without modifying the underlying computation (similar to wavelets), yet the underlying computation is simple and scales linearly with dimension. For a more thorough introduction and exploration of box splines in their more general form, readers are referred to [21].

Throughout this section, the notation will be reused from Chapter 2. $X \subset \mathbb{R}^d$ is a finite set of points with known response values $f(x)$ for all $x \in X$. Also let $L, U \in \mathbb{R}^d$ define a bounding box for X such that $L < x < U$ for all $x \in X$.

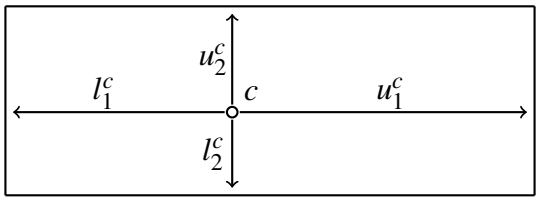

Figure 4.3: An example box in two dimensions with anchor c, upper widths u_1^c, u_2^c, and lower widths l_1^c, l_2^c. Notice that c is not required to be equidistant from opposing sides of the box, that is $u_i^c \neq l_i^c$ is allowed.

Define a box $b^c = (c, l^c, u^c)$ in d dimensions with anchor $c \in \mathbb{R}^d$, lower width vector $l^c \in \mathbb{R}^d_+$, and upper width vector $u^c \in \mathbb{R}^d_+$ (where u_i^c refers to the ith component of u^c). A visual example of a box in two dimensions can be seen in Figure 4.3. Now, define a componentwise rescaling function $g : \mathbb{R}^d \to \mathbb{R}^d$ at point $x \in \mathbb{R}^d$ to be

$$\left(g^c(x)\right)_r = \frac{k}{2}\left(1 - \frac{(x_r - c_r)_-}{l_r^c} + \frac{(x_r - c_r)_+}{u_r^c}\right), \tag{4.3}$$

where $y_+ = \max\{y, 0\}$, $y_- = (-y)_+$, k is the order of the box spline as described in Section 4.1. Finally, each box spline in a box mesh can be evaluated as $B^c(x) = B(g^c(x) \mid A)$ presuming the order of approximation implies A. Both box meshes described in the following sections use box spline basis functions of this form.

A notable property of boxes defined with the linear rescaling function g^c, is that C^0 and C^1 continuity of the underlying box spline are maintained. C^0 continuity is maintained through scaling. C^1 continuity is maintained for all box splines with C^1 continuity (order ≥ 3) because the scaling discontinuity is located at c, where all box splines (of the presented form) have first derivative zero. All continuity beyond the first derivative is lost through the rescaling function g^c.

4.2 Max Box Mesh

The first of the three meshes produces a set of boxes around chosen control points and each box has maximal distance between the control point and the nearest side of the box. This centrality property is one mechanism for creating the largest reasonable regions of support for the underlying basis functions. The individual boxes are constructed via the following procedure given a set of control points $C \subseteq X$, $c^{(i)} \in C$,

1. Initialize a box $b^{c^{(1)}} = \left(c^{(1)}, (c^{(1)} - L), (U - c^{(1)})\right)$.

2. Identify $c^{(i)}$ over $\left\{ j \;\middle|\; j \neq 1, B^{c^{(1)}}\left(c^{(j)}\right) \neq 0 \right\}$ that minimizes $\left\| c^{(j)} - c^{(1)} \right\|_\infty$.

3. Change the the box $b^{c^{(1)}}$ along the first dimension r such that $\left|\left| c^{(1)} - c^{(i)} \right|\right|_\infty = \left| c^{(1)} - c^{(i)} \right|_r$, to exclude $c^{(i)}$ from the support of $B^{c^{(1)}}$.

4. Repeat steps 2 and 3 until no point in C is in the support of $B^{c^{(1)}}$ (at most $2d$ times, once for each boundary of a box).

The same process is used to construct boxes around all control points in C. In order to improve the generality of the approximation, a set of control points is initially chosen to be well-spaced using a statistical method from Amos et al. [3]:

1. Generate a sequence of all pairs of points sorted by ascending pairwise Euclidean distance between points $\left(x^{(i_1)}, x^{(j_1)}\right)$, $\left(x^{(i_2)}, x^{(j_2)}\right)$, ... , so that $\left\| x^{(i_k)} - x^{(j_k)} \right\|_2 \leq \left\| x^{(i_{k+1})} - x^{(j_{k+1})} \right\|_2$.

2. Sequentially remove points from candidacy until only $|C|$ remain by randomly selecting a single point from each pair $\left(x^{(i_m)}, x^{(j_m)}\right)$ for $m = 1, \ldots$ if both $x^{(i_m)}$ and $x^{(j_m)}$ are still candidates for removal.

Once the boxes for a max box mesh have been constructed, the parameters can be identified via a least squares fit. The max box mesh (denoted *MBM*) is used to generate a $|X| \times |C|$ matrix M of box spline basis function evaluations at all points in X. The solution to the least squares problem $\min_P \left\| M\ P - f(X) \right\|_2$ is the parameterization of *MBM*. When $C = X$, M is the $|X| \times |X|$ identity matrix, making the max box mesh approximation $\hat{f}$ an interpolant.

While setting the number of boxes equal to the number of points causes the max box mesh to be an interpolant, the generality of the max box mesh approximation can often be improved by bootstrapping the selection of control points. Given a user-selected batch size $s \leq |X|$, start with s well-spaced control points. Next, measure the approximation error at $x \notin C$ and if the error is too large (determined by user), pick s points at which the magnitude of approximation error is largest, add those points to C, and recompute the max box mesh. The user is left to decide the batch size s and the error tolerance based on validation performance and computability. This work uses a batch size of one.

Definition 4.1. The hyperplane $x_r = c_r + u_r^c$ is the upper boundary of box b^c along dimension r, and similarly $x_r = c_r - l_r^c$ is the lower boundary of box b^c. When the anchor point y, for some box b^y, lies in the hyperplane (and facet of box b^c) defining either boundary along dimension r of b^c it is said that b^y *bounds* b^c in dimension r and is denoted $\left(b^c \,\middle|_r\, b^y\right)$.

Throughout all experiments and all repeated trials conducted for this study, all tested interpolation points were covered by at least one box in the *MBM*. However, it is possible for the *MBM* to not form a covering of $[L,U]$ when there are cyclic boundaries. Consider the following example in

three dimensions:

$$
\begin{aligned}
C &= \big\{(0,0,0),(1,0,2/3),(1,1,4/3)\big\},\\
b^{c^{(1)}} &= \big((0,0,0),\ (*,*,*),\ (1,*,4/3)\big),\\
b^{c^{(2)}} &= \big((1,0,2/3),\ (1,*,*),\ (*,1,*)\big),\\
b^{c^{(3)}} &= \big((1,1,4/3),\ (*,1,4/3),\ (*,*,*)\big).
\end{aligned}
$$

Asterisks are used to represent boxes that are not bounded by other boxes along some dimensions. The point $(2,2,-3)$ is not in any of the max boxes defined above. In this case, there is a cycle in box boundaries that looks like $\big(b^{c^{(1)}} \,\big|_1\, b^{c^{(2)}} \,\big|_2\, b^{c^{(3)}} \,\big|_3\, b^{c^{(1)}}\big)$. This example demonstrates that it is geometrically possible for the max box mesh to fail to cover a space, however experiments demonstrate that it is empirically unlikely.

The max box mesh remains a viable strategy for computed approximations. Given a maximum of c control points in d dimensions with n points, the computational complexities are: $\mathcal{O}(c^2 d)$ for computing boxes, $\mathcal{O}(cd^2 + d^3)$ for a least squares fit, and $\mathcal{O}(n/s)$ for bootstrapping (which is multiplicative over the fitting complexities). Evaluating the max box mesh requires $\mathcal{O}(cd)$ computations.

4.3 Iterative Box Mesh

The iterative box mesh (*IBM*) comprises box-shaped regions that each contain exactly one control point in their interior just as in the *MBM*. However, the mesh is a covering for $[L,U]$ by construction and places boxes in a way that reduces apparent error. The boxes are constructed via the following procedure given a finite set of points $X \subset \mathbb{R}^d$, where $C \subseteq X$ is the (initially empty) set of control points.

1. Add the box that covers $[L,U]$ anchored at the most central point $x^{(k)} \in X$, add $x^{(k)}$ to C, and least squares fit the *IBM* model to all $x \in X$.

2. Add a new box $[L,U]$ anchored at $x^{(i)} \notin C$ such that $\left|IBM(x^{(i)}) - f(x^{(i)})\right| = \max_{x \in X \setminus C} \left|IBM(x) - f(x)\right|$, reshaping all boxes $b^{x^{(j)}}$ that contain $x^{(i)}$ by bounding the first dimension r such that $\left|x_r^{(j)} - x_r^{(i)}\right| = \left\|x^{(j)} - x^{(i)}\right\|_\infty$ (also reshaping the box $b^{x^{(i)}}$ symmetrically), add $x^{(i)}$ to C, and then least squares fit the *IBM* model to all $x \in X$.

3. Repeat Step 2 until model approximation error is below tolerance t.

Just as for the *MBM*, the parameters can be identified via a least squares fit. The iterative box mesh is used to generate a $|X| \times |C|$ matrix M of box spline function evaluations at all points in X. Now the box spline coefficients are the solution to the least squares problem $\min_P \left|\left|M\,P - f(X)\right|\right|_2$. Also as for the *MBM*, $C = X$ causes M to equal the $|X| \times |X|$ identity, making the iterative box mesh approximation $\hat{f}$ an interpolant.

As opposed to the max box mesh, the bootstrapping procedure is built into the iterative box mesh. The user is left to decide the most appropriate error tolerance, however a decision mechanism and analysis is presented in Section 4.5.4. As mentioned earlier, the iterative box mesh is a covering for $[L,U]$ by construction and this can be proved by an inductive argument.

An *IBM* least squares fit $\hat{f}(z) = \sum_j P_j B^{c^{(i_j)}}(z)$ can generate approximations at new points $z \in Z \subset \mathbb{R}^d$ by evaluating $\hat{f}(z)$. The computational complexity for generating the mesh is $\mathcal{O}(c^2 nd)$ where c is the number of control points determined by the minimum error threshold and $n = |X|$. The computational complexity of evaluating the mesh at a single point is $\mathcal{O}(cd)$.

4.4 Voronoi Mesh

The final of the three meshes utilizes 2-norm distances to define boundaries rather than max norm distances. It also does not rely on box splines as the underlying basis function. A well-studied technique for classification and approximation is the nearest neighbor algorithm [17]. Nearest neighbor inherently utilizes the convex region $v^{x^{(i)}}$ (Voronoi cell [24]) consisting of all points closer to $x^{(i)}$ than any other point $x^{(j)}$. The Voronoi mesh smooths the nearest neighbor approximation by utilizing the Voronoi cells to define support via a generic basis function $V : \mathbb{R}^d \to \mathbb{R}_+$ given by

$$V^{x^{(i)}}(y) = \left(1 - \frac{\left\|y - x^{(i)}\right\|_2}{2\, d(y \mid x^{(i)})}\right)_+,$$

where $x^{(i)}$ is the center of the Voronoi cell, $y \in \mathbb{R}^d$ is an interpolation point, and $d(y \mid x^{(i)})$ is the distance between $x^{(i)}$ and the boundary of the Voronoi cell $v^{x^{(i)}}$ in the direction $y - x^{(i)}$. $V^{x^{(i)}}\left(x^{(j)}\right) = \delta_{ij}$ and $V^{x^{(i)}}$ has local support. While $V^{x^{(i)}}(x^{(i)}) = 1$, the 2 in the denominator causes all basis functions to go linearly to 0 at the boundary of the twice-expanded Voronoi cell. Note that this basis function is C^0 because the boundaries of the Voronoi cell are C^0. In the case that there is no boundary along the vector w, the basis function value is always 1.

While the cost of computing the exact Voronoi cells for any given set of points grows exponentially [25], the calculation of d is linear with respect to the number of control points and dimensions. Given any center $x^{(i)} \in \mathbb{R}^d$, set of control points $C \subseteq X$, and interpolation point $y \in \mathbb{R}^d$, $d\left(y \mid x^{(i)}\right)$ is the solution to

$$\max_{c \in C \setminus \{x^{(i)}\}} \frac{\left\|y - x^{(i)}\right\|_2}{2} \frac{y \cdot \left(c - x^{(i)}\right) - x^{(i)} \cdot \left(c - x^{(i)}\right)}{c \cdot \left(c - x^{(i)}\right) - x^{(i)} \cdot \left(c - x^{(i)}\right)}. \tag{4.4}$$

The parameters of the VM can now be computed exactly as for the MBM and IBM. The Voronoi mesh is used to generate a $|X| \times |C|$ matrix M of basis function evaluations at all points in X. Now the VM coefficients are the solution to the least squares problem $\min_P \left\|M\ P - f(X)\right\|_2$. When

$X = C$, M is the identity making the mesh an interpolant. Bootstrapping can be performed with an identical procedure to that for the *IBM*.

1. Pick the most central point $x^{(k)} \in X$ to be the first control point in C and fit the VM model to all $x \in X$.

2. Identify a control point $x^{(i)} \notin C$ such that $\left|VM(x^{(i)}) - f(x^{(i)})\right| = \max_{x \in X \setminus C} \left|VM(x) - f(x)\right|$, add $x^{(i)}$ to C, and then fit the VM model to all $x \in X$.

3. Repeat Step 2 until approximation error is below tolerance t.

Any VM is naïvely a covering for $[L,U]$, since any possible interpolation point will have a nearest neighbor control point. The computational complexity of evaluating a parameterized Voronoi mesh with c control points is $\mathcal{O}(c^2 d)$. Bootstrapping the generation of a Voronoi mesh requires $\mathcal{O}(c^2 nd)$ computations for a maximum number of basis functions c determined by the error threshold.

4.5 Data and Analysis

Some data sets will be used to evaluate the interpolation and regression meshes proposed above. This chapter utilizes three data sets of varying dimension and application. In the following subsections the sources and targets of each data set are described, as well as challenges and limitations related to interpolating and approximating these data sets. The distributions of response values being modeled can be seen in Figure 4.4. The preprocessing and approximation processes are described in Section 4.5.4.

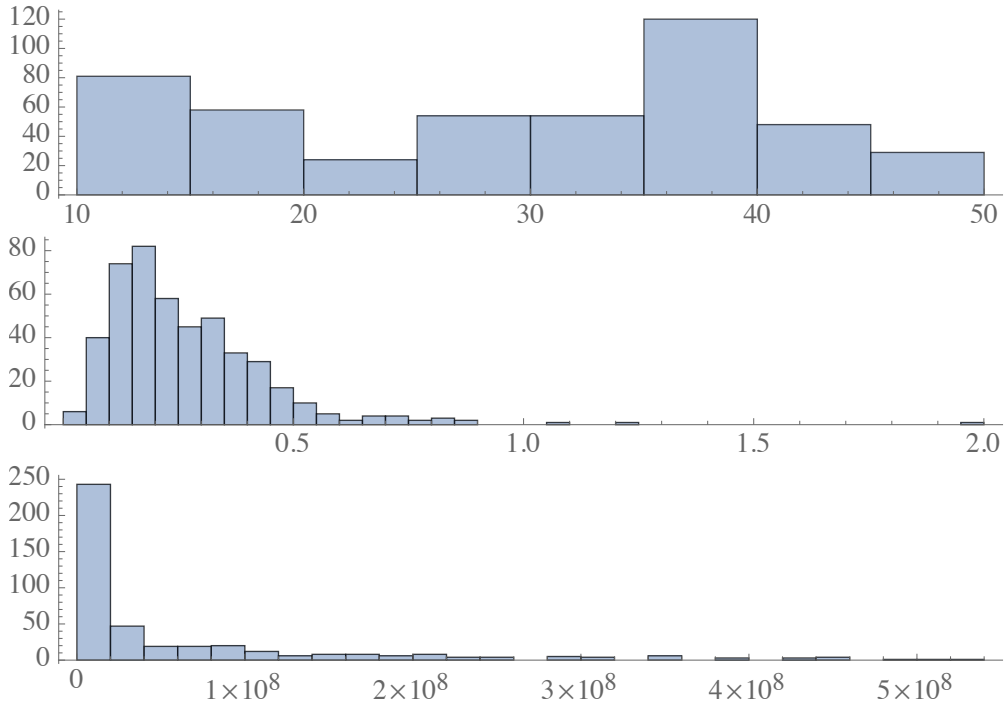

Figure 4.4: Histograms of Parkinsons (total UPDRS), forest fire (area), and HPC I/O (mean throughput) response values respectively. Notice that both the forest fire and HPC I/O data sets are heavily skewed.

4.5.1 High Performance Computing I/O ($n = 532, d = 4$)

The first of three data sets is a four-dimensional data set produced by executing the IOzone benchmark from [47] on a homogeneous cluster of computers. The system performance data was collected by executing IOzone 40 times for each of a select set of system configurations. A single IOzone execution reports the max I/O file-read throughput seen. The 40 executions for each system configuration are converted to their mean, which is capable of being modeled by each of the multivariate approximation techniques presented earlier in this chapter. The four dimensions being modeled to predict throughput mean are file size, record size, thread count, and CPU frequency.

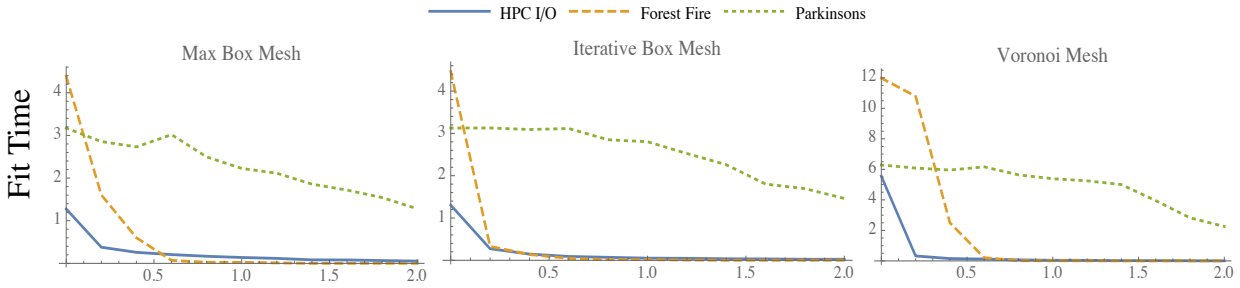

Figure 4.5: Time required to generate model fits for each technique with varying relative error tolerance during bootstrapping.

4.5.2 Forest Fire ($n = 517, d = 12$)

The forest fire data set [15] describes the area of Montesinho park burned on specific days and months of the year in terms of the environmental conditions. The twelve dimensions being used to model burn area are the x and y spatial coordinates of burns in the park, month and day of year, the FFMC, DMC, DC, and ISI indices (see source for details), the temperature in Celsius, relative humidity, wind speed, and outdoor rain. The original analysis of this data set demonstrated it to be difficult to model, likely due to the skew in response values.

4.5.3 Parkinson's Telemonitoring ($n = 468, d = 16$)

The final data set for evaluation [59] is derived from a speech monitoring study with the intent to automatically estimate Parkinson's disease symptom development in Parkinson's patients. The function to be predicted is a time-consuming clinical evaluation measure referred to as the UPDRS score. The total UPDRS score given by a clinical evaluation is estimated through 16 real numbers generated from biomedical voice measures of in-home sound recordings.

Data Set	Technique	Tolerance	Average Error
HPC I/O	MBM	1.2	0.597
Forest Fire	MBM	1.8	3.517
Parkinson's	MBM	0.6	0.114
HPC I/O	IBM	0.4	0.419
Forest Fire	IBM	1.8	3.615
Parkinson's	IBM	1.8	0.121
HPC I/O	VM	0.2	0.382
Forest Fire	VM	1.0	4.783
Parkinson's	VM	2.0	1.824

Table 4.1: The optimal error tolerance bootstrapping parameters for each technique and each data set as well as the average absolute relative errors achieved by that tolerance. Notice that large relative error tolerances occasionally yield even lower evaluation errors, demonstrating the benefits of approximation over interpolation for noisy data sets.

4.5.4 Performance Analysis

The performance of the approximation techniques varies considerably across the three evaluation data sets. Relative errors for the most naïve approximators such as nearest neighbor can range from zero to $\left(\max_x f(x) - \min_x f(x)\right) / \min_x f(x)$ when modeling a positive function $f(x)$ from data. Each of the approximation techniques presented remain within these bounds and all errors are presented in signed relative form $(\hat{f}(x) - f(x))/f(x)$. Before the models are constructed all data values (components $x_r^{(i)}$ of $x^{(i)} \in X$) are shifted and scaled to be in the unit cube $[0,1]^d$, while the response values are taken in their original form. All models are evaluated with 10 random $80/20$ splits of the data.

Each of the approximation techniques presented incorporates bootstrapping based on an allowable error tolerance t. An analysis of the effects of bootstrapping error tolerances on validation accuracy can be seen in Figure 4.6. The approximation meshes perform best on the forest fire and Parkinson's data sets when the error tolerance used for fitting is large (smoothing rather than interpolating), while near-interpolation generally produces the most accurate models for HPC I/O. Another performance result of note is that the *MBM* and *IBM* have very similar basis functions

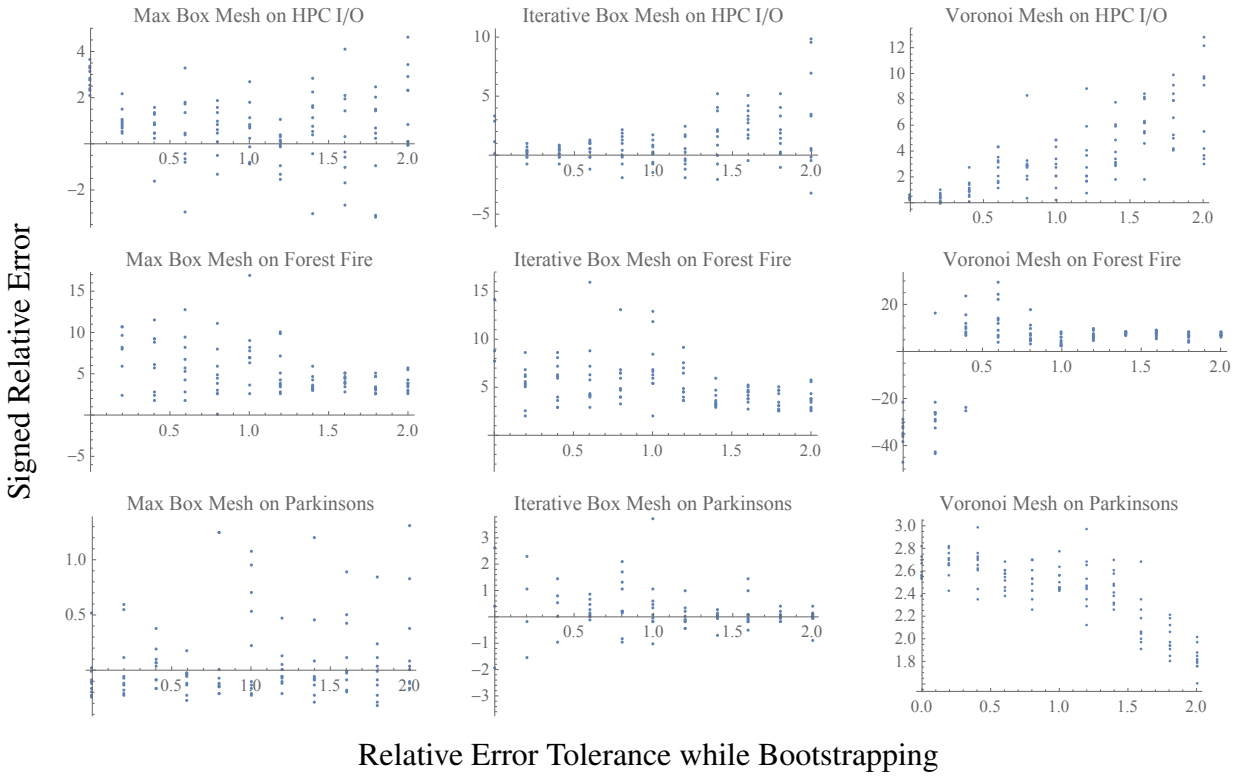

Figure 4.6: The performance of all three techniques with varied relative error tolerance for the bootstrapping parameter. The columns are for Max Box Mesh, Iterative Box Mesh, and Voronoi Mesh, respectively. The rows are for HPC I/O, Forest Fire, and Parkinson's respectively. Notice the techniques' behavior on the Parkinson's and Forest Fire data sets, performance increases with larger error tolerance.

with largely different outputs.

The selection of bootstrapping error tolerance also effects the computation time required to fit each of the models to data. Figure 4.5 presents the time required to construct approximations for each model and each data set with varying t. The rapid reduction in computation time required for the forest fire and HPC I/O data sets suggests that large reductions in error can be achieved with relatively few basis functions. The Parkinson's data set however presents a more noisy response, with increasing number of basis functions reducing error much less quickly.

The distributions of errors experienced by each approximation technique when the optimal boot-

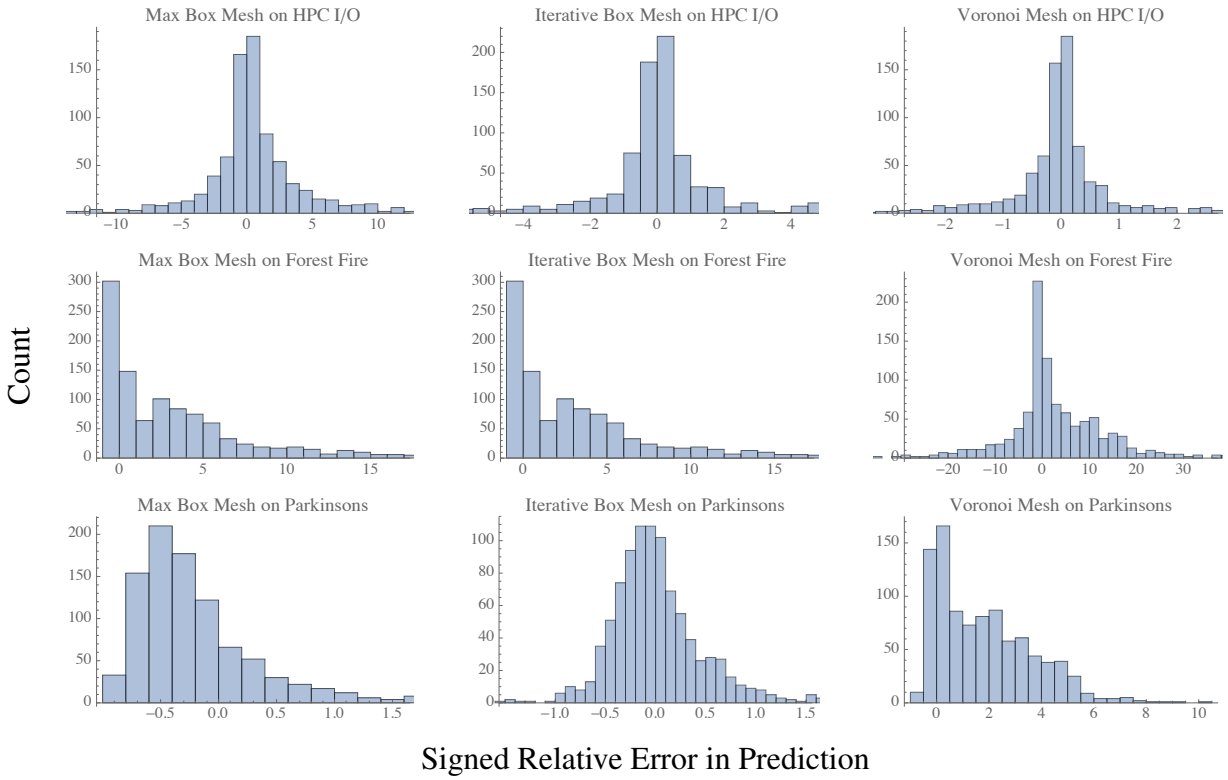

Figure 4.7: A sample of relative errors for all three techniques with optimal selections of error tolerance. The columns are for Max Box Mesh, Iterative Box Mesh, and Voronoi Mesh, respectively. The rows are for HPC I/O, Forest Fire, and Parkinson's respectively.

strapping relative error tolerance is selected can be seen in Figure 4.7. HPC I/O exhibits the most normal approximation errors, which suggests that the models are converging on the random noise of the response for the data set. The worst relative approximation errors are produced by the Voronoi mesh on the forest fire data set. The small magnitude true response values contribute to the larger relative errors. Regardless, the VM errors are unacceptably large.

4.6 Discussion of Mesh Approximations

The bootstrapping procedure presented for each quasi-mesh approximation is very computationally expensive and does not provide much improvement over the interpolatory approach. Analysis suggests that the appropriate relative error tolerance needs to be discovered empirically for each application of a modeling technique. Further analytic studies could arrive at methods for determining optimal error tolerances at runtime, however increases in runtime complexity may not be afforded in many applications.

The box-shaped basis functions and the construction algorithms used for the MBM and IBM could become a source of error when d (the dimension of the data X) is comparable to n (the number of known points). The blending regions in which multiple basis functions overlap are always axis aligned and in applications such as image analysis, any single dimension may be unsuitable for approximating the true underlying function. The Voronoi mesh attempts to address this problem by utilizing boundaries between points in multiple dimensions simultaneously. However, it is empirically unclear whether the true benefits of the VM are seen in applications where $d \ll n$.

Each of the case studies presented have fewer than 1000 points. The complexity of the presented approximation techniques are suitable for large dimension, but the increased complexity associated with brute-force bootstrapping prohibits their use on larger data sets. The Voronoi mesh in particular has a large complexity with respect to n which could be significantly improved by dropping

bootstrapping, as is done in Chapter 5. While each technique requires less than ten seconds on average to produce a fit in the presented case studies, the fit time required quickly grows into minutes around 1000 points. These results demonstrate the limits of expensive bootstrapping. However, the viability of each mesh encourages the further applications explored in later chapters.

4.7 Implications of Quasi-Mesh Results

The Max Box Mesh, Iterative Box Mesh, and Voronoi Mesh each provide novel strategies for effectively approximating multivariate phenomonon. The underlying constructions are theoretically straightforward. The computational complexities of each make them particularly suitable for applications in many dimensions, while the bootstrapping error tolerance parameter allows a balance between smoothing and interpolation to be explored empirically with each application. However, the expense of bootstrapping prohibits its use on larger data sets.

Chapter 5

Stronger Approximations of Variability

Performance and its variability can be summarized by a variety of statistics. Mean, range, standard deviation, variance, and interquartile range are a few summary statistics that describe performance and variability. However, the most precise characterization of any performance measure is the cumulative distribution function (CDF), or its derivative the probability density function (PDF). Previous techniques for predicting system performance have strictly modeled real-valued summary statistics because there exists a large base of mathematical techniques capable of approximating functions of the form $f : \mathbb{R}^d \to \mathbb{R}$. However, there is little systems work approximating functions $f : \mathbb{R}^d \to \{g \mid g : \mathbb{R} \to \mathbb{R}\}$.

5.1 Measuring Error

When the range of an approximation is the real numbers, error is reported with summary statistics including: min absolute error, max absolute error, and absolute error quartiles. When the range of an approximation is the space of cumulative distribution functions, the Kolmogorov-Smirnov statistic (max-norm difference between the functions) is used.

A hurdle when modeling function-valued outputs such as cumulative distribution functions (CDFs) or probability density functions (PDFs) is that certain properties must be maintained. It is necessary that a PDF $f : \mathbb{R} \to \mathbb{R}$ have the properties $f(x) \geq 0$ and $\int_{-\infty}^{\infty} f(x)dx = 1$. However, for a CDF $F : \mathbb{R} \to \mathbb{R}$ the properties are $F(x) \in [0,1]$ and $F(x)$ is absolutely continuous and nondecreasing.

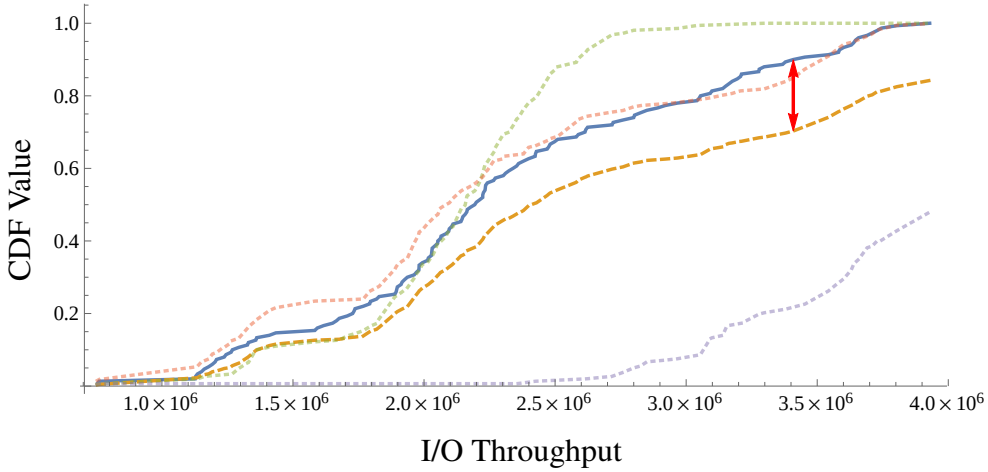

Figure 5.1: In this HPC I/O example, the general methodology for predicting a CDF and evaluating error can be seen. The Delaunay method chose three source distributions (dotted lines) and assigned weights $\{.3, .4, .3\}$ (top to bottom at arrow). The weighted sum of the three known CDFs produces the predicted CDF (dashed line). The KS Statistic (arrow) computed between the true CDF (solid line) and predicted CDF (dashed line) is 0.2 for this example. The KS test null hypothesis is rejected at p-value 0.01, however it is not rejected at p-value 0.001.

This work utilizes the fact that a convex combination of CDFs (or PDFs) results in a valid CDF (or PDF). Given $G(x) = \sum_i w_i F_i(x)$, $\sum_i w_i = 1$, $w_i \geq 0$, and each F_i is a valid CDF, G must also be a valid CDF. A demonstration of how this is applied can be seen in Figure 5.1.

The performance of approximation techniques that predict probability functions can be analyzed through a variety of summary statistics. This work uses the max absolute difference, also known as the Kolmogorov-Smirnov (KS) statistic [41] for its compatibility with the KS test.

The two-sample KS test is a useful nonparametric test for comparing two CDFs while only assuming stationarity, finite mean, and finite variance. The null hypothesis (that two CDFs come from the same underlying distribution) is rejected at level $p \in [0,1]$ when

$$KS > \sqrt{-\frac{1}{2}\ln\left(\frac{p}{2}\right)}\sqrt{\frac{1}{n_1}+\frac{1}{n_2}},$$

with distribution sample sizes $n_1, n_2 \in \mathcal{N}$. For all applications of the KS test presented in this

System Parameter	Values
File Size (KB)	4, 16, 64, 256, 1024, 4096, 8192, 16384
Record Size (KB)	4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384
Thread Count	1, 8, 16, 24, 32, 40, 48, 56, 64
Frequency (GHz)	1.2, 1.6, 2, 2.3, 2.8, 3.2, 3.5
Test Type	Readers, Rereaders, Random Readers, Initial Writers, Rewriters, Random Writers

Table 5.1: A description of system parameters considered for IOzone. Record size must be $\leq$ file size during execution.

work $n_1 = n_2$. An example of the round-trip prediction methodology from known and predicted distributions to the calculation of error can be seen in Figure 5.1.

5.1.1 Feature Weighting

It is well-known that an important procedure in any application of predictive methodologies is identifying those features of the data that are most relevant to making accurate predictions [34]. Selection strategies such as the floating searches studied in [51] or others compared in [26] can be too expensive for large approximation problems. Rather, this work poses feature selection as a continuous optimization problem. Let X be an $n \times d$ matrix of n known system configurations with d parameters each normalized to be in $[0,1]$. Define an error function that computes the error of a predictive model trained on X diag w, $w \in \mathbb{R}^d$, by performing ten random splits with 80% of the rows of X diag w for training and 20% for testing. A minimum of this error function could be considered an optimal weighting of the features of X. Minimization is performed using a zero order method in the absence of a readily computable gradient.

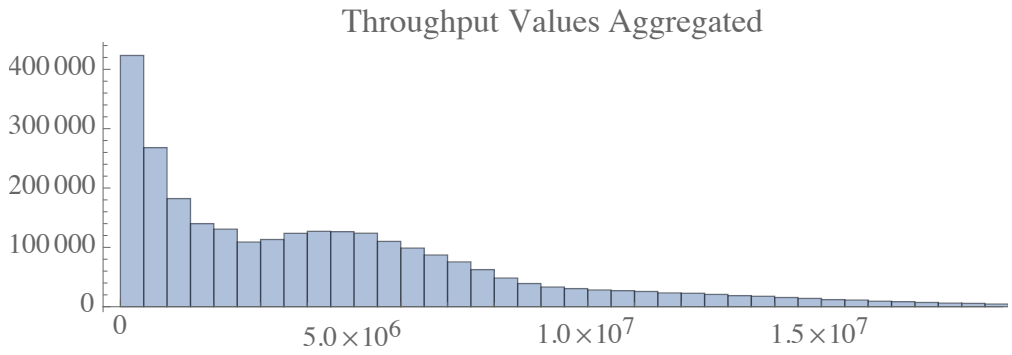

Figure 5.2: Histogram of the raw throughput values recorded during all IOzone tests across all system configurations. The distribution is skewed right, with few tests having significantly higher throughput than most others.

5.2 Variability Data

This chapter utilizes a variability modeling case study with a five-dimensional dataset produced by executing the IOzone benchmark [47] on a homogeneous cluster of computers. Each node contains two Intel Xeon E5-2637 CPUs offering a total of 16 CPU cores with 16GB of DRAM. While the CPU frequency varies depending on the test configuration, the I/O from IOzone is performed by an ext4 filesystem sitting above an Intel SSDSC2BA20 SSD drive. At the time of data collection, Linux kernel Version 4.13.0 was used. The system performance data was collected over two weeks by executing IOzone 150 times for each of a select set of approximately 18K system configurations, for a total of approximately 2.7M executions of IOzone. A single IOzone execution reports the max I/O throughput in kilobytes per second seen for the selected test type. The summary of the data components in $x^{(i)}$ for the experiments for this chapter can be seen in Table 5.1. Distributions of raw throughput values being modeled can be seen in Figure 5.2.

Some mild preprocessing was necessary to prepare the data for modeling and analysis. All features were shifted by their minimum value and scaled by their range, mapping each feature independently into $[0,1]$. This normalization ensures each feature is treated equally by the interpolation techniques and should be performed on all data before building models and making predictions

regardless of application. All 150 repeated trials for a system configuration were grouped with that configuration. The only nonordinal feature in this data is the test type. All test types were treated as different applications and were separated for modeling and analysis, i.e., predictions for the "readers" test type were made using only known configurations for the "readers" test type.

5.3 Distribution Prediction Results

All three interpolation techniques are used to predict the distribution of I/O throughput values at previously unseen system configurations. In order to improve robustness of the error analysis, ten random selections of 80% of the IOzone data are used to train each model and the remaining 20% provide approximation error for each model. The recorded errors are grouped by unique system configuration and then averaged within each group. The samples are identical for each interpolation technique, ensuring consistency in the training and testing sets.

The aggregation of errors across all IOzone tests given 80% of the data as training can be seen in Figure 5.3. Agglomerate errors for each technique resemble a Gamma distribution. The percentages of significant prediction errors with varying p-values are on display in Table 5.2. The primary p-value used for analyses in this work is 0.001, chosen because close to 2K predictions are made for each test type. Also, applications executed in cloud and HPC systems that could benefit from statistical modeling will be executed at least thousands of times. In line with this knowledge, it is important to ensure that only a small fraction of interpretable results could occur solely under the influence of random chance. When considering the $p = 0.001$ results for each technique, a little under half of the predicted CDFs are significantly different from the measured (and presumed) correct CDFs. A rejection rate of 45% would seem a poor result, however in this situation the complexity of the problem warrants a slightly different interpretation. These predictions are a very *precise* characterization of performance variability, in fact the cumulative distribution function of

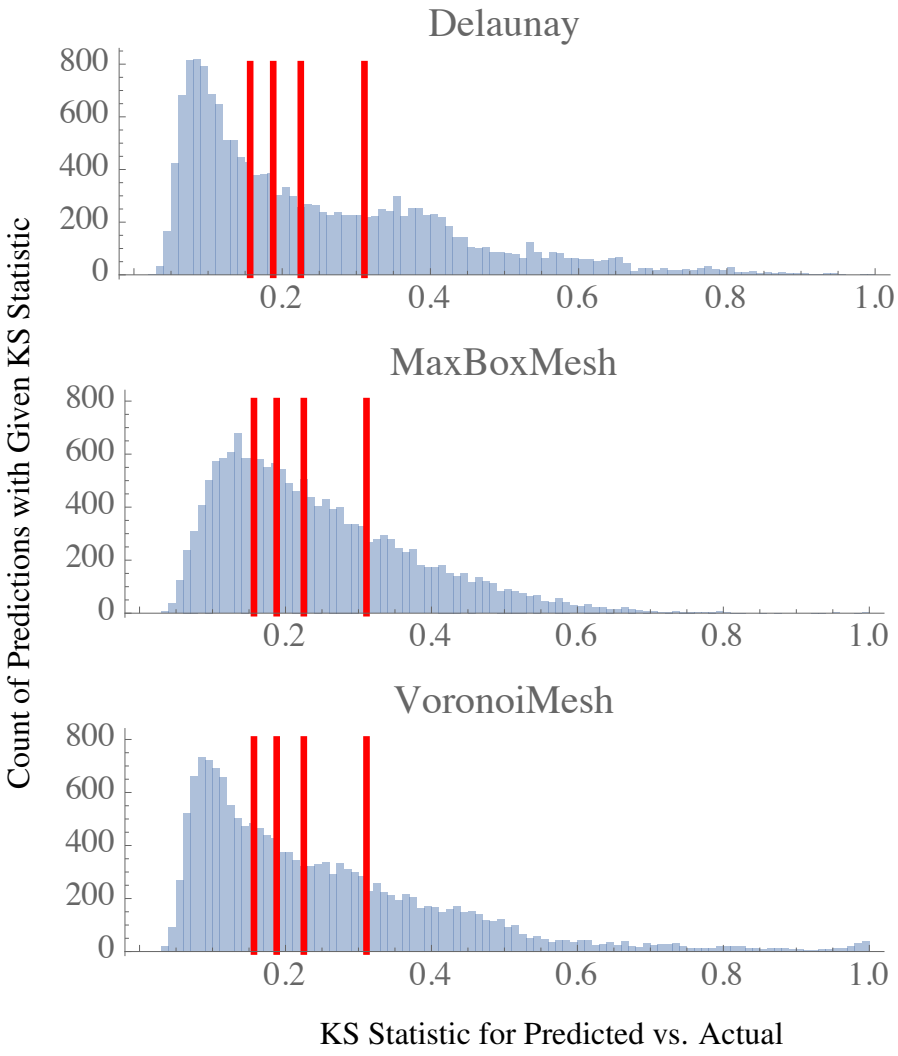

Figure 5.3: Histograms of the prediction error for each modeling algorithm from ten random splits when trained with 80% of the data aggregated over all different test types. The distributions show the KS statistics for the predicted throughput distribution versus the actual throughput distribution. The four vertical red lines represent commonly used p-values {0.05, 0.01, 0.001, 1.0e-6} respectively. All predictions to the right of a red line represent CDF predictions that are significantly different (by respective p-value) from the actual distribution according to the KS test.

Algorithm	*P*-Value	% N.H. Rejections
Delaunay	.05	58.4
Max Box Mesh		69.3
Voronoi Mesh		61.9
Delaunay	.01	51.1
Max Box Mesh		58.4
Voronoi Mesh		53.4
Delaunay	.001	44.1
Max Box Mesh		46.9
Voronoi Mesh		45.1
Delaunay	1.0e-6	31.4
Max Box Mesh		26.6
Voronoi Mesh		28.7

Table 5.2: Percent of null hypothesis rejections rate by the KS-test when provided different selections of p-values. These accompany the percent of null hypothesis rejection results from Figure 5.3.

a random variable is the strongest possible characterization of variability that can be predicted. Globally, only a little under half of the predictions fail to capture *all* of the characteristics of performance variability at new system configurations. It is also demonstrated later in this Section that this result can likely be improved.

While interpreting null hypothesis rejection rates for these interpolation techniques, it is important to consider how the rejection rate reduces with increasing amounts of training data. Figure 5.4 displays the change in $p = 0.001$ null hypothesis rejection rate with increasing density of training data up to the maximum density allowed by this set. Delaunay interpolation provides the best results with the least training data by about 5%, but these low density rejection rates are unacceptably high (90%). Figure 5.4 clearly shows that this data set and/or the system variables used in the models of performance variability is inadequate to capture the full variability map from system parameters to performance CDF. Which or both obtains is not clear. A few well chosen data points can significantly improve the interpolants, and thus a careful study of the rejection instances is warranted, besides enlarging the set of system variables being modeled.

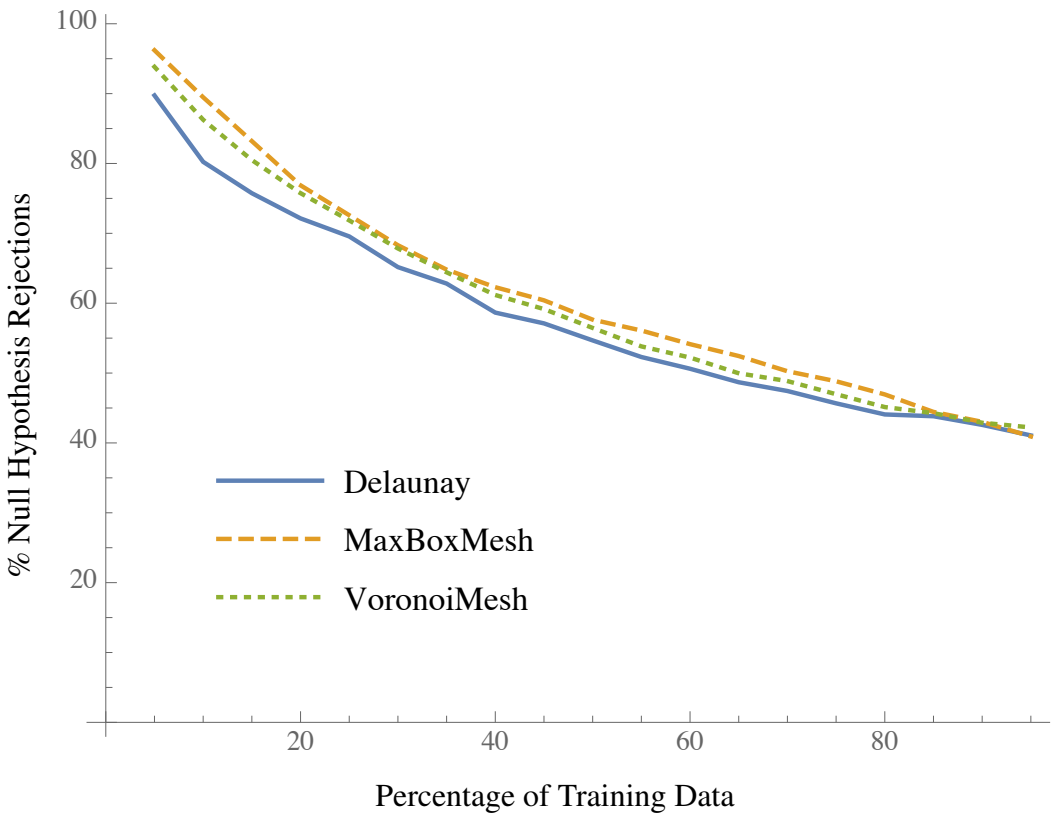

Figure 5.4: The performance of each algorithm on the KS test ($p = 0.001$) with increasing amounts of training data averaged over all IOzone test types and ten random splits of the data. The training percentages range from 5% to 95% in increments of 5%. Delaunay is the best performer until 95% of data is used for training, at which Max Box mesh becomes the best performer by a fraction of a percent.

It may be misleading to consider the global performance of each prediction technique across all test types, as some test types are more difficult than others to predict and have more apparent latent variables. In Figure 5.5, the relative difficulty of each IOzone test type can be compared. The I/O test types analyzing reads are typically approximated with lower error than those test types analyzing writes. Regardless of test type, in the aggregate results the KS statistics hover consistently around 0.15, demonstrating an impressively low KS statistic for predictions. In order to address the opacity of aggregate analysis, another case study and an application of the methodology from Section 5.1.1 is presented in Table 5.3.

The results presented in Table 5.3 are achieved by permitting each approximation technique 300 iterations of simulated annealing. In each iteration, the impact of potential weights on the average KS statistic were considered. All weights were kept in the range [0,2], and were applied to the normalized features for frequency, file size, record size, and number of threads. All three approximation techniques had similar optimal weights achieved by simulated annealing of approximately $(.001, 2, 1.7, 1.5)$ for frequency, file size, record size, and number of threads, respectively. Recall that each interpolation technique uses small distances to denote large influences on predictions, meaning that frequency was the most important feature when predicting variability for the "readers" test type, followed not-so-closely by number of threads, then record size.

The "readers" test type results demonstrate that the underlying prediction techniques work and are capable of seeing rejection rates below 5% when tuned for a given application. It is important to emphasize that the roughly 95% of predictions for which the null hypothesis was not rejected are predicting the *precise* distribution of I/O throughput that will be witnessed at a previously unseen system configuration. To the authors' knowledge, there is no existing methodology that is generically applicable to any system performance measure, agnostic of system architecture, and capable of making such powerful predictions.

5.4 Discussion of Distribution Prediction

The results of the IOzone case study indicate that predicting the CDF of I/O throughput at previously unseen system configurations is a challenging problem. The KS statistic captures the worst part of any prediction and hence provides a conservatively large estimate of approximation error. The average absolute errors in the predicted CDFs are always lower than the KS statistics. However, the KS statistic was chosen because of the important statistical theory surrounding it as an error measure. Considering this circumstance, a nonnegligible volume of predictions provide im-

Algorithm	*P*-Value	Unweighted % N.H. Rejection	Weighted % N.H. Rejection
Delaunay	.05	24.9	30.2
Max Box Mesh		21.3	21.2
Voronoi Mesh		18.7	11.3
Delaunay	.01	21.6	27.4
Max Box Mesh		16.4	16.4
Voronoi Mesh		14.9	7.0
Delaunay	.001	19.7	25.4
Max Box Mesh		13.1	13.1
Voronoi Mesh		12.3	4.6
Delaunay	1.0e-6	17.9	23.4
Max Box Mesh		11.3	11.3
Voronoi Mesh		8.5	2.3

Table 5.3: The null hypothesis rejection rates for various p-values with the KS-test. These results are strictly for the "readers" IOzone test type and show unweighted results as well as the results with weights tuned for minimum error (KS statistic) by 300 iterations of simulated annealing. Notice that the weights identified for the Delaunay model cause data dependent tuning, reducing performance. MaxBoxMesh performance is improved by a negligible amount. VoronoiMesh performance is notably improved.

pressively low levels of error. Powerful predictive tools such as those presented in this work allow for more in-depth analysis of system performance variability. For example, system configurations that are most difficult to predict in these tests are likely "outlier" configurations that do not resemble those configurations that share many similar parameters. Analysis of these configurations may provide valuable insight into effective application specific operation of computer systems.

As mentioned at the beginning of this chapter, no prior work has attempted to model an arbitrary performance measure for a system to such a high degree of precision. All previous statistical modeling attempts capture a few (< 3) ordinal performance measures. Generating models that have such high degrees of accuracy allows system engineers to identify previously unused configurations that present desired characteristics. Service level agreements (SLAs) in cloud computing environments are cause for capital competition that is affected heavily by system performance [48]. Users prefer SLAs that allow the most computing power per monetary unit, incentivizing

service providers to guarantee the greatest possible performance. Overscheduling and irregular usage patterns force cloud service providers to occasionally overload machines, in which case precise models of system performance can be used to statistically minimize the probability of SLA violation. Similar targeted performance tuning techniques can be applied to HPC system configuration to maximize application throughput or minimize system power consumption.

A final application domain affected by this methodology is computer security. Collocated users on cloud systems have received attention recently [2]. If a malicious collocated user is capable of achieving specific insight into the configuration of the system, or the activity of other collocated users by executing performance evaluation programs (i.e., IOzone), a new attack vector may present itself. Malicious users could be capable of identifying common performance distributions of vulnerable system configurations and vulnerable active user jobs. This knowledge may allow targeted exploits to be executed. Light inspection of raw IOzone I/O throughputs provides substantial evidence that distinct performance distributions coincide closely with specific system configuration parameters. Conversely, a service provider may defend against such attacks by deliberately obfuscating the performance of the machine. Models such as those presented in this chapter could identify optimal staggering and time-delay whose introduction into the system would prevent malicious users from identifying system configurations and active jobs.

Results presented in Table 5.3 are particularly interesting, demonstrating that Delaunay appears most vulnerable to data dependent tuning, Max Box mesh is largely insensitive to such tuning, and Voronoi mesh benefits (for this data set) from the tuning.

There are many avenues for extending this modeling methodology. One extension is to add categorical variables to the models. Presently the rejection rate of distribution predictions can only be reduced with large volumes of performance data, however the judicious choice (via experimental design, e.g.) of new data points may be able to effectively reduce the amount of training data required. Finally, more case studies need to be done to test the robustness of the present modeling

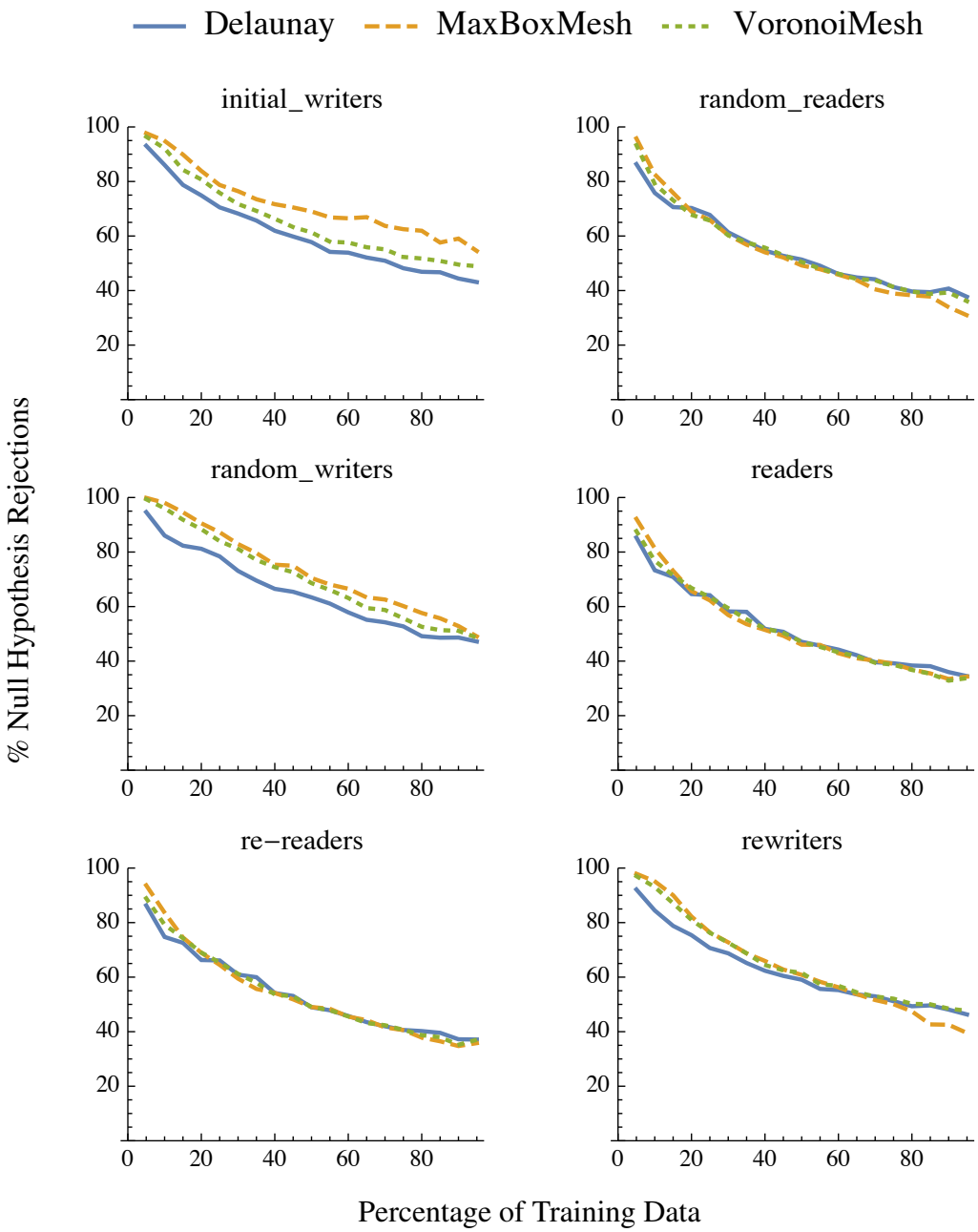

Figure 5.5: The percentage of null hypothesis rejections for predictions made by each algorithm on the KS test ($p = 0.001$) over different IOzone test types with increasing amounts of training data. Each percentage of null hypothesis rejections is an average over ten random splits of the data. The training percentages range from 5% to 95% in increments of 5%. The read test types tend to allow lower rejection rates than the write test types.

techniques to changes in domain and performance measure.

5.5 The Power of Distribution Prediction

The methodology presented is capable of providing new insights, extending existing analyses, and improving the management of computational performance variability. Delaunay, Max Box mesh, and Voronoi mesh interpolation are viable techniques for constructing approximations of performance cumulative distribution functions. A case study on I/O throughput demonstrated that the models are capable of effectively predicting CDFs for most unseen system configurations for any of the available I/O test types. The present methodology represents a notable increase in the ability to statistically model arbitrary system performance measures involving the interaction of many ordinal system parameters.

Chapter 6

An Error Bound on Piecewise Linear Interpolation

This chapter presents the theoretical results bounding the error of (piecewise) linear interpolation. The error analysis relies on linear interpolation for three reasons: (1) second order results can be obtained utilizing a Lipschitz constant on the gradient of a function, rather than standard Lipschitz bounds; (2) the results directly apply to Delaunay interpolation; and (3) multiple other interpolants in this work compute predictions as convex combinations of observed function values, which may allow for straight forward extensions of this error bound.

Lemma 6.1. *Let $S \subset \mathbb{R}^d$ be open and convex, $f : \mathbb{R}^d \to \mathbb{R}$, and $\nabla f \in Lip_{(\gamma,\|\cdot\|_2)}(S)$, the set of γ-Lipschitz continuous functions in the 2-norm. Then for all $x, y \in S$*

$$\left|f(y) - f(x) - \langle \nabla f(x), y - x \rangle\right| \leq \frac{\gamma \|y - x\|_2^2}{2}.$$

Proof. Consider the function $g(t) = f\big((1-t)x + ty\big)$, $0 \leq t \leq 1$, whose derivative $g'(t) = \big\langle \nabla f\big((1-$

$t)x+ty\big),y-x\big\rangle$ is the directional derivative of f in the direction $(y-x)$.

$$\begin{aligned}
\big|f(y)-f(x)-\langle\nabla f(x),y-x\rangle\big| &= \big|g(1)-g(0)-g'(0)\big| \\
&= \left|\int_0^1 g'(t)-g'(0)\,dt\right| \\
&\le \int_0^1 \big|g'(t)-g'(0)\big|\,dt \\
&= \int_0^1 \left|\big\langle\nabla f\big((1-t)x+ty\big)-\nabla f(x),y-x\big\rangle\right|\,dt \\
&\le \int_0^1 \big\|\nabla f\big((1-t)x+ty\big)-\nabla f(x)\big\|_2\,\|y-x\|_2\,dt \\
&\le \int_0^1 \big(\gamma\,\|y-x\|_2\big)\,\big(\|y-x\|_2\big)t\,dt \\
&= \frac{\gamma\|y-x\|_2^2}{2}.
\end{aligned}$$

□

Lemma 6.2. *Let $x,y,v_i\in\mathbb{R}^d$, $c_i\in\mathbb{R}$, and $|\langle y-x,v_i\rangle|\le c_i$ for $i=1,\ \ldots,\ d$. If $M=(v_1,\ \ldots,\ v_d)$ is nonsingular, then*

$$\|y-x\|_2^2\le\frac{1}{\sigma_d^2}\sum_{i=1}^d c_i^2,$$

where σ_d is the smallest singular value of M.

Proof. Using the facts that M and M^t have the same singular values, and $\|M^tw\|_2\ge\sigma_d\|w\|_2$, gives

$$\begin{aligned}
\|y-x\|_2^2 &\le \frac{\|M^t(y-x)\|_2^2}{\sigma_d^2} \\
&= \frac{1}{\sigma_d^2}\sum_{i=1}^d\langle y-x,v_i\rangle^2 \\
&\le \frac{1}{\sigma_d^2}\sum_{i=1}^d c_i^2.
\end{aligned}$$

□

Lemma 6.3. *Given f, γ, S as in Lemma 6.1, let $X = \{x_0, x_1, \ldots, x_d\} \subset S$ be the vertices of a d-simplex, and let $\hat{f}(x) = \langle c, x - x_0 \rangle + f(x_0)$, $c \in \mathbb{R}^d$ be the linear function interpolating f on X. Let σ_d be the smallest singular value of the matrix $M = (x_1 - x_0, \ldots, x_d - x_0)$, and $k = \max_{1 \le j \le d} \|x_j - x_0\|_2$. Then*

$$\left\|\nabla f(x_0) - c\right\|_2 \le \sqrt{d}\frac{\gamma k^2}{\sigma_d}.$$

Proof. Consider $f(x) - \hat{f}(x)$ along the line segment $z(t) = (1-t)x_0 + tx_j$, $0 \le t \le 1$. By Rolle's Theorem, for some $0 < \hat{t} < 1$, $\left\langle \nabla f\big(z(\hat{t})\big) - c, x_j - x_0 \right\rangle = 0$. Now

$$\begin{aligned}
\left|\left\langle \nabla f(x_0) - c, x_j - x_0 \right\rangle\right| &= \left|\left\langle \nabla f(x_0) - \nabla f\big(z(\hat{t})\big) + \nabla f\big(z(\hat{t})\big) - c, x_j - x_0 \right\rangle\right| \\
&= \left|\left\langle \nabla f(x_0) - \nabla f\big(z(\hat{t})\big), x_j - x_0 \right\rangle\right| \\
&\le \left\|\nabla f(x_0) - \nabla f\big(z(\hat{t})\big)\right\|_2 \|x_j - x_0\|_2 \\
&\le \gamma \|x_0 - z(\hat{t})\|_2 \, \|x_j - x_0\|_2 \\
&\le \gamma \|x_j - x_0\|_2^2 \le \gamma k^2,
\end{aligned}$$

for all $1 \le j \le d$. Using *Lemma 6.2*,

$$\left\|\nabla f(x_i) - c\right\|_2^2 \le \frac{d}{\sigma_d^2}\left(\gamma k^2\right)^2 \implies \left\|\nabla f(x_i) - c\right\|_2 \le \sqrt{d}\frac{\gamma k^2}{\sigma_d}.$$

□

Theorem 6.4. *Under the assumptions of Lemma 6.1 and Lemma 6.3, for $z \in S$,*

$$\left|f(z) - \hat{f}(z)\right| \le \frac{\gamma \|z - x_0\|_2^2}{2} + \sqrt{d}\frac{\gamma k^2}{\sigma_d}\|z - x_0\|_2.$$

Proof. Let $v = \nabla f(x_0) - c$, where $\|v\|_2 \leq \sqrt{d}\gamma k^2/\sigma_d$ by *Lemma 6.3*. Now

$$\begin{aligned}
\left|f(z) - \hat{f}(z)\right| &= \left|f(z) - f(x_0) - \langle c, z - x_0\rangle\right| \\
&= \left|f(z) - f(x_0) - \langle \nabla f(x_0) - v, z - x_0\rangle\right| \\
&= \left|f(z) - f(x_0) - \langle \nabla f(x_0), z - x_0\rangle + \langle v, z - x_0\rangle\right| \\
&\leq \left|f(z) - f(x_0) - \langle \nabla f(x_0), z - x_0\rangle\right| + \left|\langle v, z - x_0\rangle\right| \\
&\leq \left|f(z) - f(x_0) - \langle \nabla f(x_0), z - x_0\rangle\right| + \|v\|_2\|z - x_0\|_2 \\
&\leq \left|f(z) - f(x_0) - \langle \nabla f(x_0), z - x_0\rangle\right| + \tfrac{\gamma k^2\sqrt{d}}{\sigma_d}\|z - x_0\|_2 \\
&\leq \frac{\gamma\|z - x_0\|_2^2}{2} + \sqrt{d}\frac{\gamma k^2}{\sigma_d}\|z - x_0\|_2,
\end{aligned}$$

where the last inequality follows from *Lemma 6.1*. □

In summary, the approximation error of a linear (simplicial) interpolant tends quadratically towards zero when approaching observed data only when the diameter of the simplex is also reduced at a proportional rate. Only linear convergence to the true function can be achieved in practice, without the incorporation of additional observations. Notice that the approximation error is largely determined by the spacing of observed data. Predictions made by simplices whose vertices are not well-spaced (i.e., have large diameter, or are nearly contained in a hyperplane) have higher error. In light of this error bound, an empirical evaluation of presented algorithms follows.

6.1 Data and Empirical Analysis

The theoretical results constructed at the start of this chapter for (piecewise) linear interpolation are promising and apply directly to Delaunay interpolation, however they are difficult to interpret in context with approximation algorithms that do not have similar known uniform error bounds.

For that reason, this chapter utilizes five different data sets of varying dimension and application to construct approximations and compare the accuracy of different techniques.

In the following five subsections the sources and targets of each test data set are described, as well as challenges and limitations related to approximating these data. The distribution of response values being modeled is presented followed by the distribution of approximation errors for each algorithm. The plots for all five data sets have the same format.

All five data sets are rescaled such that the domain of approximation is the unit hypercube. The range of the first four data sets is the real numbers, while the range of the fifth data set is the space of cumulative distribution functions. All approximation techniques are applied to the first four data sets, while only those interpolants whose approximations are convex combinations of observed data are applied to the final data set.

All approximations are constructed using k-fold cross validation as described in [38] with $k = 10$. This approach randomly partitions data into k (nearly) equal sized sets. Each algorithm is then evaluated by constructing an approximation over each unique union of $k-1$ elements of the partition, making predictions for points in the remaining element. As a result, each observed data point is used in the construction of $k-1$ different approximations and is approximated exactly once. The k-fold cross validation method is data-efficient and provides an unbiased estimate of the expected prediction error [38], however it should be noted that neither this method nor others can provide a universally unbiased estimator for the variance of prediction error [8].

In addition to the figures displaying approximation results for each data set, tables of accompanying numerical results are located in the Appendix. All of the test data sets capture underlying functions that are almost certainly stochastic. As described in Section **??**, regression techniques appear most appropriate for these problems. However, typically data grows exponentially more sparse with increasing dimension. Given sparse data, regressors tend towards interpolation.

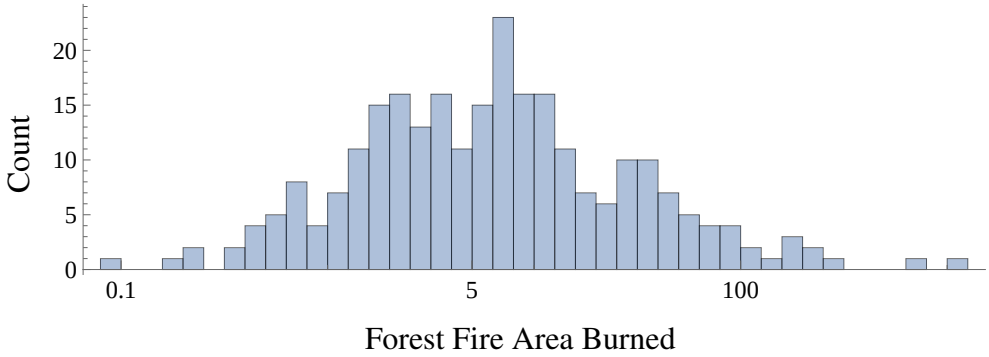

Figure 6.1: Histogram of forest fire area burned under recorded weather conditions. The data is presented on a ln scale because most values are small with exponentially fewer fires on record that burn large areas.

6.1.1 Forest Fire ($n = 504$, $d = 12$)

The forest fire data set [15] describes the area of Montesinho park burned over months of the year along with environmental conditions. The twelve dimensions being used to model burn area are the x and y spatial coordinates of burns in the park, month of year (mapped to x, y coordinates on a unit circle), the FFMC, DMC, DC, and ISI indices (see source for details), the temperature, relative humidity, wind speed, and outdoor rain. The original analysis of this data set demonstrated it to be difficult to model, likely due to the skew in response values.

As suggested by Figure 6.2, the SVR has the lowest absolute prediction errors for 80% of the data, with MLP and Delaunay being the nearest overall competitors. The effectiveness of SVR on this data suggests the underlying function can be defined by relatively few parameters, as well as the importance of capturing the low-burn-area data points.

6.1.2 Parkinson's Telemonitoring ($n = 5875$, $d = 19$)

The second data set for evaluation [59] is derived from a speech monitoring study with the intent to automatically estimate Parkinson's disease symptom development in Parkinson's patients. The

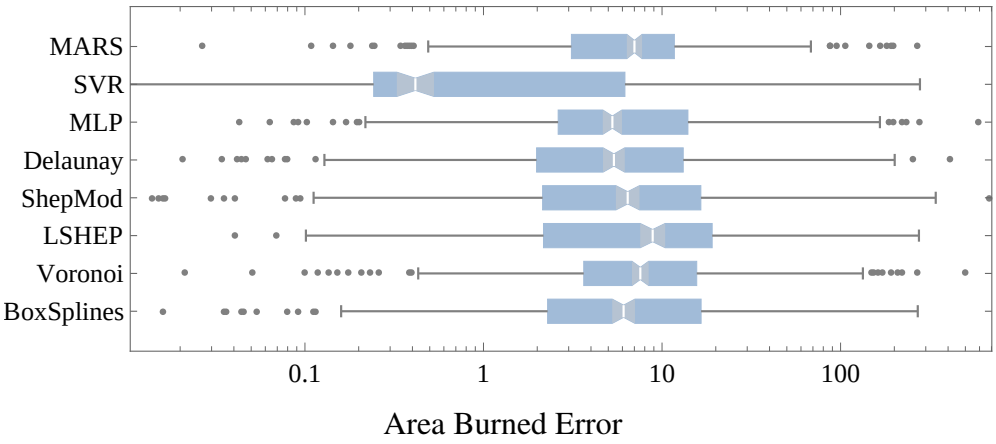

Figure 6.2: All models are applied to approximate the amount of area that would be burned given environment conditions. 10-fold cross validation as described in the beginning of Section 6.1 is used to evaluated each algorithm. This results in exactly one prediction from each algorithm for each data point. These boxes depict the median (middle bar), median 95% confidence interval (cones), quartiles (box edges), fences at $3/2$ interquartile range (whiskers), and outliers (dots) of absolute prediction error for each model. Similar to Figure 6.1, the errors are presented on a ln scale. The numerical data corresponding to this figure is provided in Table A.1 in the Appendix.

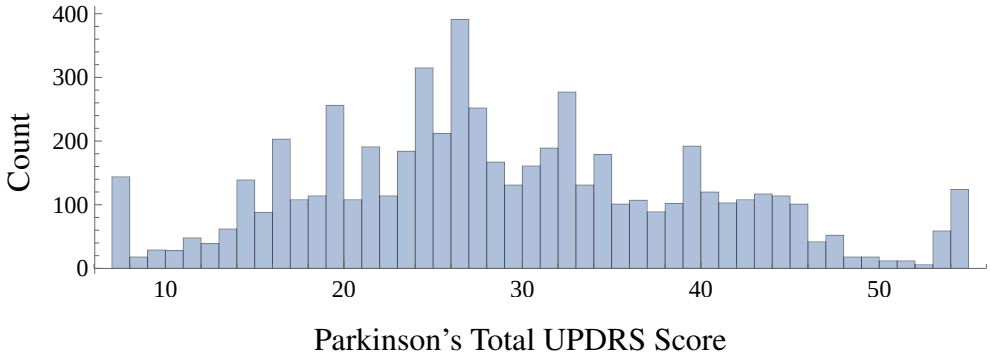

Figure 6.3: Histogram of the Parkinson's patient total UPDRS clinical scores that will be approximated by each algorithm.

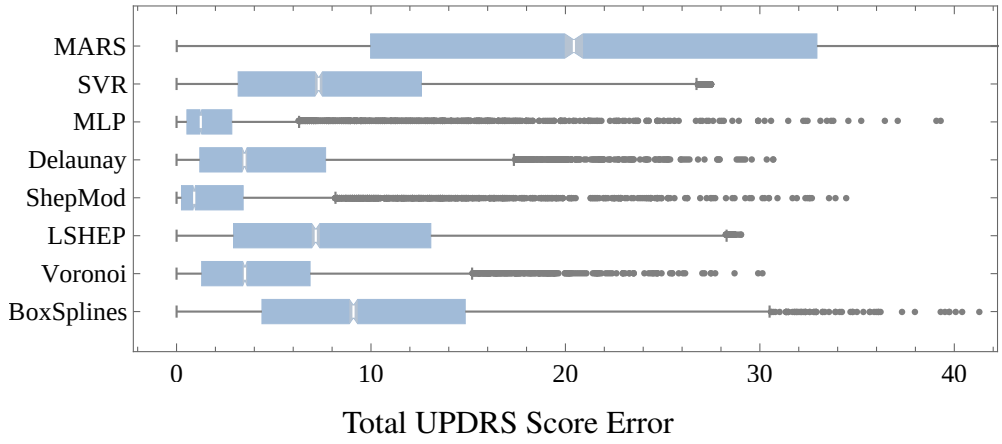

Figure 6.4: All models are applied to approximate the total UPDRS score given audio features from patients' home life, using 10-fold cross validation. These boxes depict the median (middle bar), median 95% confidence interval (cones), quartiles (box edges), fences at $3/2$ interquartile range (whiskers), and outliers (dots) of absolute prediction error for each model. The numerical data corresponding to this figure is provided in Table A.3 in the Appendix.

function to be approximated is a time-consuming clinical evaluation measure referred to as the UPDRS score. The total UPDRS score given by a clinical evaluation is estimated through 19 real numbers generated from biomedical voice measures of in-home sound recordings.

Figure 6.4 shows the ShepMod algorithm has the lowest minimum, first quartile, and median of absolute errors for this problem, while providing the best prediction 66% of the time. The MLP has the lowest third quartile and provides the best prediction for 32% of approximations. The dominance of ShepMod may be due in part to regular-interval total UPDRS scores provided by clinicians, favoring a nearest-neighbor strategy of prediction.

6.1.3 Australian Daily Rainfall Volume ($n = 2609$, $d = 23$)

The third data set for the total daily rainfall in Sydney, Australia [64] provides a slightly higher dimensional challenge for the interpolants and regressors. This data is composed of many meteorological readings from the region in a day including: min and max temperatures, sunshine, wind

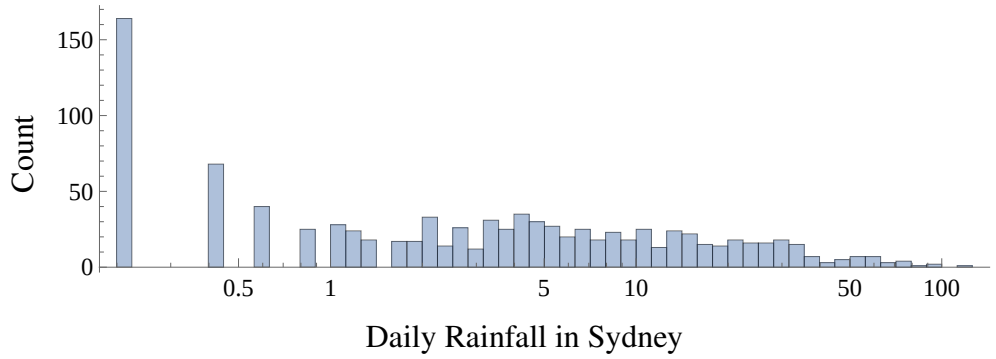

Figure 6.5: Histogram of daily rainfall in Sydney, Australia, presented on a ln scale because the frequency of larger amounts of rainfall is significantly less. There is a peak in occurrence of the value 0, which has a notable effect on the resulting model performance.

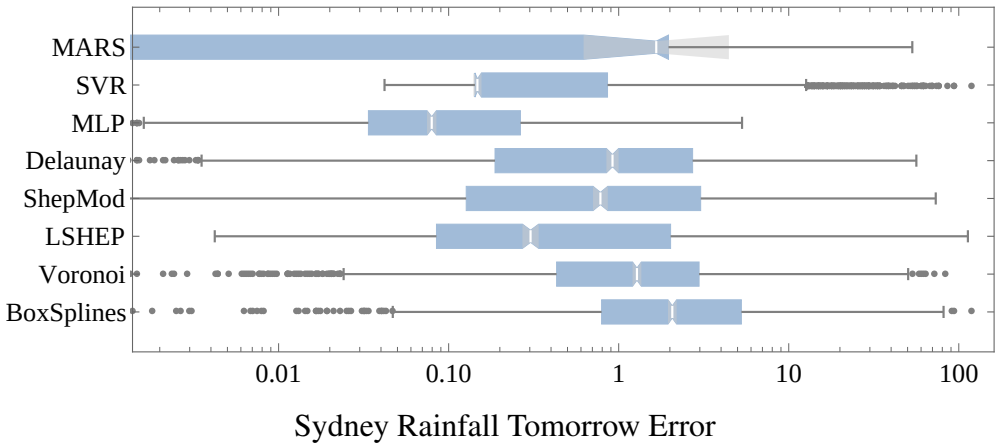

Figure 6.6: All models are applied to approximate the amount of rainfall expected on the next calendar day given various sources of local meteorological data, using 10-fold cross validation. These boxes depict the median (middle bar), median 95% confidence interval (cones), quartiles (box edges), fences at $3/2$ interquartile range (whiskers), and outliers (dots) of absolute prediction error for each model. The errors are presented on a ln scale, mimicking the presentation in Figure 6.5. The numerical data corresponding to this figure is provided in Table A.5 in the Appendix.

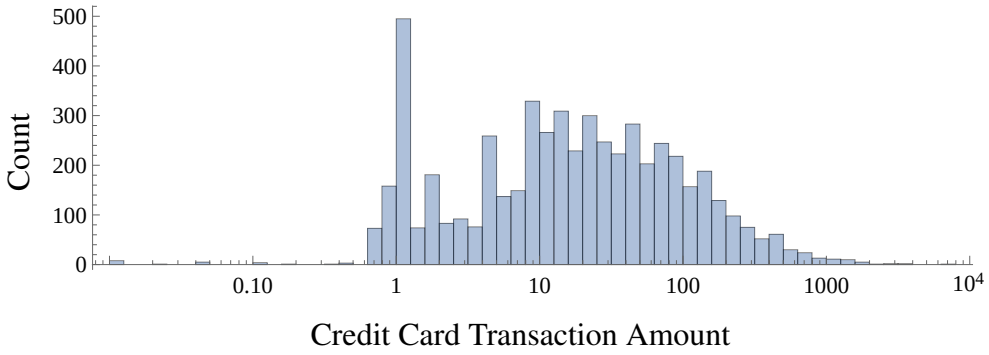

Figure 6.7: Histogram of credit card transaction amounts, presented on a ln scale. The data contains a notable frequency peak around $1 transactions. Fewer large purchases are made, but some large purchases are in excess of five orders of magnitude greater than the smallest purchases.

speed directions (converted to coordinates on a circle), wind speeds, and humidities throughout the day, day of the year (converted to coordinates on a circle), and the model must predict the amount of rainfall tomorrow.

While Figure 6.5 makes MARS look far better than other techniques, it only provides the best prediction for 11% of points. The MLP has the lowest absolute error for 56% of points and LSHEP is best for 28%. MARS likely achieves such a low first quartile due to the high occurrence of the value zero in the data.

6.1.4 Credit Card Transaction Amount ($n = 5562$, $d = 28$)

The fourth test data set, and the final with a real-valued range, is a collection of credit card transactions [50]. The provided data carries no direct real-world meaning, being the output of a principle component analysis on the original hidden source data. This obfuscation is done to protect the information of the credit card users. This data has the largest dimension of all considered, at 28. A model for this data predicts the transaction amount given the vector of principle component information.

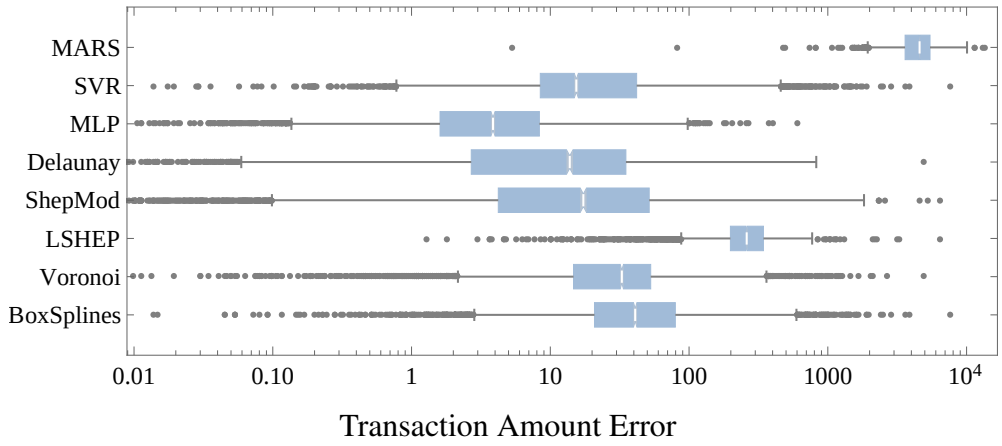

Figure 6.8: All models are applied to approximate the expected transaction amount given transformed (and obfuscated) vendor and customer-descriptive features, using 10-fold cross validation. These boxes depict the median (middle bar), median 95% confidence interval (cones), quartiles (box edges), fences at $3/2$ interquartile range (whiskers), and outliers (dots) of absolute prediction error for each model. The absolute errors in transaction amount predictions are presented on a ln scale, just as in Figure 6.7. The numerical data corresponding to this figure is provided in Table A.7 in the Appendix.

As can be seen in Figure 6.8, the MLP outperforms all other algorithms at the first, second, third, and fourth quartiles. The MLP produces the lowest absolute error prediction for 80% of transactions, Delaunay bests the remaining 20%. It is likely that with less data, Delaunay would be the best performer.

6.1.5 High Performance Computing I/O ($n = 3016$, $d = 4$)

The final of five data sets is derived from [9], which provides four-dimensional distribution data by executing the IOzone benchmark [47] on a computer system and varying the system's file size, record size, thread count, and CPU frequency. At each configuration, IOzone samples the I/O file-read throughput (in bytes per second) 150 times. Empirical distribution function points are computed from each set of 150 executions, which are interpolated with a piecewise cubic Hermite interpolating polynomial [29] to approximate the CDF. All interpolation algorithms with

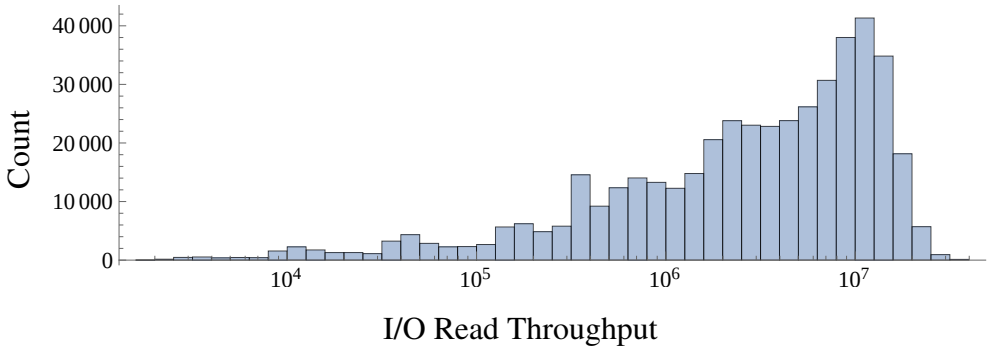

Figure 6.9: Histogram of the raw throughput values recorded during all IOzone tests across all system configurations. The distribution is skewed right, with few tests having significantly higher throughput than most others. The data is presented on a ln scale.

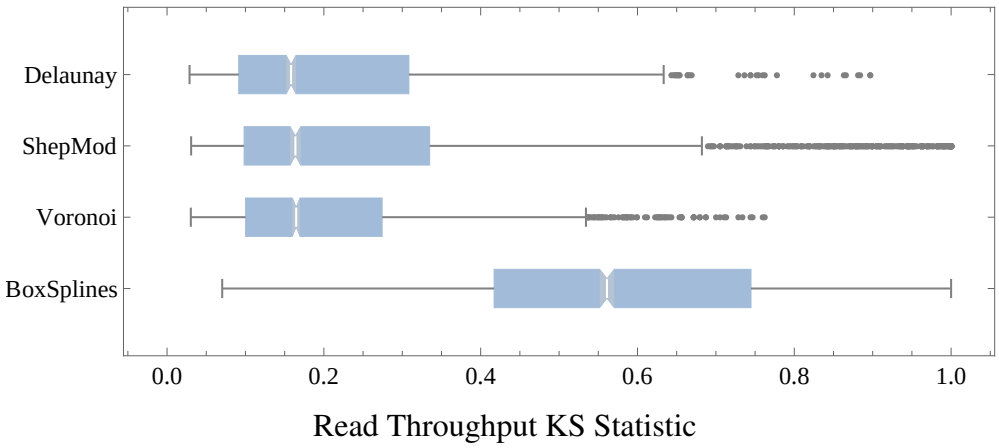

Figure 6.10: The models directly capable of predicting distributions are applied to predicting the expected CDF of I/O throughput at a previously unseen system configuration, using 10-fold cross validation. The KS statistic between the observed distribution at each system configuration and the predicted distribution is recorded and presented above. Note that the above figure is *not* log-scaled like Figure 6.9. The numerical data corresponding to this figure is provided in Table A.9 in the Appendix.

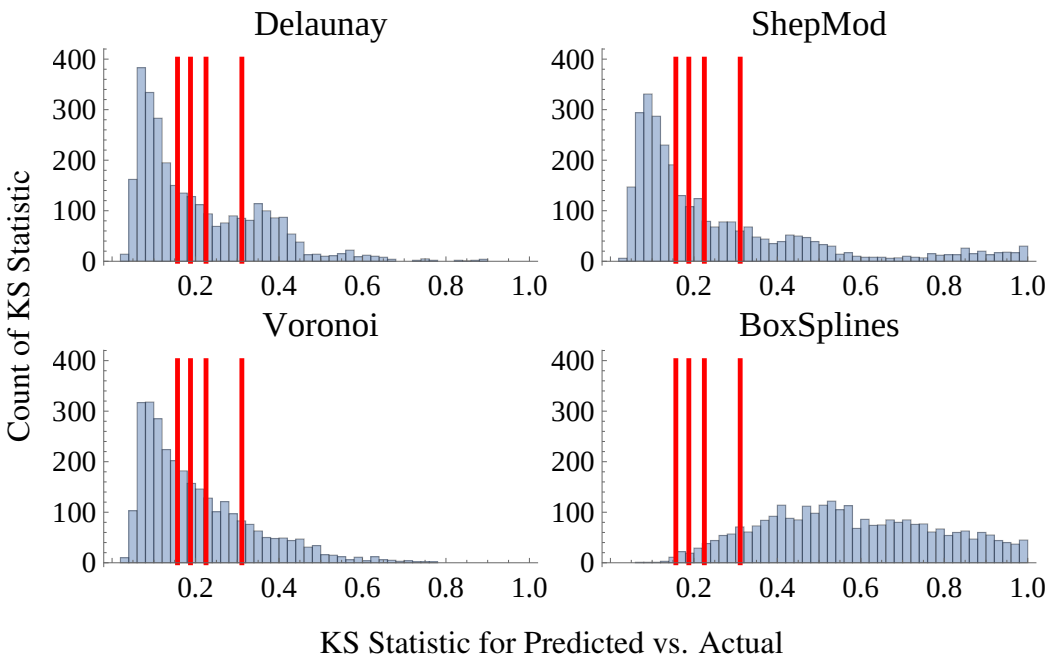

Figure 6.11: Histograms of the prediction error for each interpolant that produces predictions as convex combinations of observed data, using 10-fold cross validation. The histograms show the KS statistics for the predicted throughput distribution versus the actual throughput distribution. The four vertical lines represent cutoff KS statistics given 150 samples for commonly used p-values 0.05, 0.01, 0.001, 10^{-6}, respectively. All predictions to the right of a vertical line represent CDF predictions that are significantly different (by respective p-value) from the actual distribution according to the KS test. The numerical counterpart to this figure is presented in Table 6.1.

the exception of LSHEP are used to approximate these CDFs from system configurations.

Delaunay achieves the lowest KS statistic for 62% of approximations, while Voronoi is best for the remaining 38%. Figure 6.10 shows that while Delaunay may have more best predictions, the behavior of Voronoi may be preferable.

Figure 6.11 expands on the KS statistic results presented in Figure 6.10. Agglomerate errors for each algorithm resemble a Gamma distribution. The percentages of significant prediction errors with varying p-values are on display in Table 6.1. When considering the $p = 0.001$ results for each technique, a little over one third of the predicted CDFs are significantly different from the measured (and presumed) correct CDFs. However, it should be noted that with 150 samples, the error of an

	$p = .05$	$p = .01$	$p = .001$	$p = 10^{-6}$
Delaunay	**50.3**%	*43.5%*	*36.2%*	*24.7%*
ShepMod	*51.4%*	44.8%	38.1%	27.7%
Voronoi	52.6%	**43.4**%	**34.4**%	**19.1**%
BoxSpline	99.5%	98.5%	96.6%	89.5%

Table 6.1: Numerical counterpart of the histogram data presented in Figure 6.11. The columns display the percent of null hypothesis rejections by the KS-test when provided different selections of p-values for each algorithm. The algorithm with the lowest rejection rate at each p is boldface, while the second lowest is italicized.

empirical distribution function (EDF) can reasonably be upwards of .1, which serves as a rough estimate for the lower limit of achievable error. Globally, only a third of Voronoi predictions fail to capture *all* of the characteristics of the CDFs at new system configurations.

6.2 Discussion of Empirical Results

Table 6.2 summarizes results across the four test data sets with real-valued ranges. The interpolants discussed in this chapter produce the *best* approximations roughly one third of the time, and produce competitive approximations for almost all data sets. These test problems are almost certainly *stochastic* in nature, but the high dimension leads to greater data sparsity and model construction cost, making interpolation more competitive.

The major advantages to interpolation lie in the near absence of *fit* time. Delaunay, LSHEP, and ShepMod all require pairwise distance calculations, for numerical robustness (Delaunay) and to determine the radii of influence for data points (LSHEP and ShepMod). At least hundreds, and sometimes hundreds of thousands of predictions can be made by the interpolants before the most widely used regressor (MLP) finishes fitting these relatively small data sets. However, the computational complexities of all interpolants presented are greater than linear in either dimension

Algorithm	Avg. % Best	Avg. Fit or Prep. Time (s)	Avg. App. Time (s)
MARS	4.6	20.0	0.001
SVR	*21.1*	**0.5**	**0.0001**
MLP	**42.1**	200.0	0.001
Delaunay	6.0	1.0	1.0
ShepMod	18.4	*0.7*	**0.0001**
LSHEP	8.4	2.0	**0.0001**
Voronoi	0.5	1.0	0.04
BoxSpline	0.2	0.8	*0.0005*

Table 6.2: This average of Appendix Tables A.2, A.4, A.6, and A.8 provides a gross summary of overall results. The columns display (weighted equally by data set, *not* points) the average frequency with which any algorithm provided the lowest absolute error approximation, the average time to fit/prepare, and the average time required to approximate one point. The times have been rounded to one significant digit, as reasonably large fluctuations may be observed due to implementation hardware. Interpolants provide the lowest error approximation for nearly one third of all data, while regressors occupy the other two thirds. This result is obtained without any customized tuning or preprocessing to maximize the performance of any given algorithm. In practice, tuning and preprocessing may have large effects on approximation performance.

or number of points, whereas the regressors' nonlinear complexity in dimension generally comes from the model fitting optimization.

The theoretical results presented in this chapter directly apply to Delaunay interpolation, however the performance of Delaunay does not appear significantly better than other algorithms on these data sets. This observation may be due to the stochastic nature of the data, but it also speaks to the power of the approximations generated by the different interpolation methods. The strong performance of other interpolants suggests that theoretical results similar to those presented in this work can be achieved for the other interpolants under reasonable assumptions.

Finally, most of the interpolants presented in this work benefit from the ability to model any function over real d-tuples with a range that is closed under convex combinations. The results of the distribution prediction case study indicate that interpolants can effectively predict CDFs. The error analysis for that work relies on the KS statistic, which captures the worst part of any prediction

and hence provides a conservatively large estimate of approximation error. The average absolute errors in the predicted CDFs are always lower than the KS statistics. However, the KS statistic was chosen as a metric because of the important surrounding statistical theory. A nonnegligible volume of predictions provide impressively low levels of average absolute error in that final case study.

6.3 Takeaway from Empirical Tests

The major contributions of this work are: 1) new uniform theoretical error bounds for piecewise linear interpolation in arbitrary dimension; 2) an empirical evaluation across real-world problems that demonstrates interpolants produce competitively accurate models of multivariate phenomenon when compared with common regressors for sparse, moderately high dimensional problems (Section 6.1); and 3) a demonstration that some interpolants generalize to interpolation in function spaces, preserving monotonicity (with CDFs, e.g.), neither of which common regressors can do.

The various interpolants discussed in this work have been demonstrated to effectively approximate multivariate phenomena up to 30 dimensions. The underlying constructions are theoretically straightforward, interpretable, and yield reasonably accurate predictions. Most of the interpolants' computational complexities make them particularly suitable for applications in even higher dimension. The major benefits of interpolation are seen when only a small number of approximations (≤ 1000) are made from data and when there are relatively few data points for the dimension (for empirical results presented, $\log_d n \leq 5$). These findings encourage broader application of interpolants to multivariate approximation in science

Chapter 7

Improving Variability Estimates with Monotone C^2 Splines

Many domains of science rely on smooth approximations to real-valued functions over a closed interval. These smooth approximations are regularly used in automotive and aerospace engineering, architecture, mathematics and especially statistics [37]. While polynomial interpolants or regressors apply broadly, piecewise polynomial functions (splines) are often a good choice because they can approximate globally complex functions while minimizing local complexity of the approximation. It is often the case that the true underlying function or phenomenon being modeled has known properties e.g., convexity, positivity, various levels of continuity, or monotonicity. Given a reasonably large amount of data, it can be impossible to maintain these properties with a single polynomial function.

In general, the maintenance of function properties through interpolation / regression is usually referred to as *shape preserving* [29, 32]. The specific shapes this work will focus on are monotonicity and multiple levels of continuity for a function. These properties are chiefly important to the approximation of cumulative distribution functions and subsequently the effective generation of random numbers from a specified distribution.

In statistics especially, the construction of a monotone interpolating spline that is continuous in second derivative is meaningfully useful [52]. A function with these properties could approximate random variables to a high level of accuracy with relatively few intervals (with even more accu-

racy given greater continuity). A continuously twice differentiable approximation to a cumulative distribution function would also produce a corresponding probability density function that is continuously differentiable, which is a property many commonly occurring parametric distributions maintain.

7.1 Related Work

The current state-of-the-art monotone interpolating spline with a mathematical software implementation is piecewise cubic, continuously differentiable, and was first proposed in Fritsch and Carlson [29] then expanded upon in Carlson and Fritsch [10]. Let $\pi : a = x_1 < x_2 < \cdots < x_n = b$ be a partition of the interval $[a,b]$. Let $\{f(x_i) : i = 1,2,\ldots,n\}$ be a given set of data values at the partition points for a monotone function f, meaning $f(x_i) \leq f(x_{i+1})$ for $i = 1,\ldots,n-1$ or $f(x_i) \geq f(x_{i+1})$ for $i = 1,\ldots,n-1$. Let $\hat{f}$ be a piecewise cubic spline defined in each sub-interval $I_i = [x_i, x_{i+1}]$ by

$$
\begin{aligned}
h_i &= x_{i+1} - x_i \\
u(t) &= 3t^2 - 2t^3 \\
p(t) &= t^3 - t^2 \\
\hat{f}(x) &= f(x_i)\, u\big((x_{i+1}-x)/h_i\big) + f(x_{i+1})\, u\big((x-x_i)/h_i\big) \\
&\quad - \hat{f}\,'(x_i)\, p\big((x_{i+1}-x)/h_i\big) + \hat{f}\,'(x_{i+1})\, p\big((x-x_i)/h_i\big).
\end{aligned}
$$

Notice that it is up to the user to choose values for $\hat{f}\,'$and a viable monotonic cubic spline can be produced by choosing $\hat{f}\,'(x_i) = 0$, $i = 1,\ldots,n$. However, such a spline has too many *wiggles* for

most applications. Fritsch and Carlson proceed to show that simple conditions on the derivative values can guarantee monotonicity, and that these conditions can be enforced in a way that ensures modifications on one interval will not break the monotonicity of cubic polynomials over any neighboring intervals. Consider the terms $\alpha = \frac{\hat{f}\,'(x_i)(x_{i+1}-x_i)}{f(x_{i+1})-f(x_i)}$ and $\beta = \frac{\hat{f}\,'(x_{i+1})(x_{i+1}-x_i)}{f(x_{i+1})-f(x_i)}$, now monotonicity of a cubic polynomial over a sub-interval can be maintained by ensuring that α and β reside in any of the following regions.

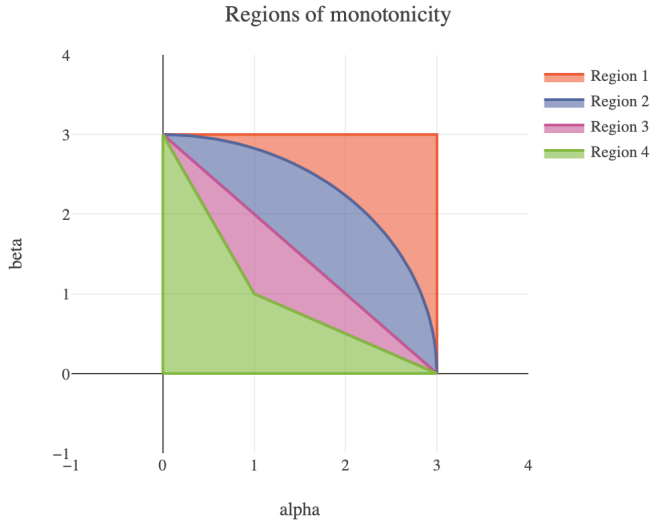

Figure 7.1: These are the feasible regions of monotonicity for cubic splines.

The actual region of monotonicity for a cubic polynomial is larger, but projection of (α, β) into one of these regions ensures that monotonicity will be achieved and not violated for neighboring regions. The user must decide which region is most suitable to project (α, β) onto, Fritsch and Carlson recommend using region 2.

This theoretical work has been extended in [35, 60] to monotone quintic polynomials, but no mathematical software has been produced accordingly. That is where this proposed work comes in.

7.2 Achieved Progress

To expand my own understanding of the current state-of-the-art method for constructing a monotone spline interpolant, I fully implemented the algorithm described in [29] in Python. This code can be used to generate monotone cubic spline fits to data and allows for the user specification of a method((the means by which monotonicity is enforced)) and derivatives at any of the knots or endpoints. Clearly this code is not optimized for numerical robustness, nor speed, but it serves as a portable demonstration and educational tool.

Here we can see the recommended method of projection for moving cubic polynomials into the region of monotonicity.

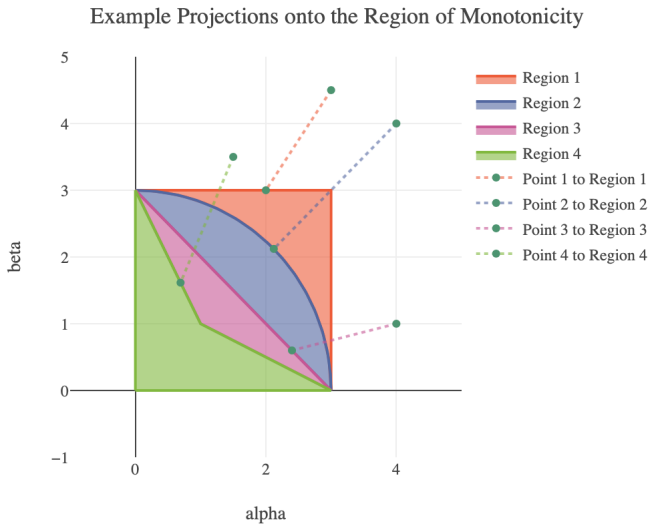

Figure 7.2: A demonstration of projection onto the feasible region for cubic splines.

The resulting interpolating spline is C^1 and has a *smooth* appearance, here is a demonstration when using Region 2.

And the derivative of the above spline is continuous.

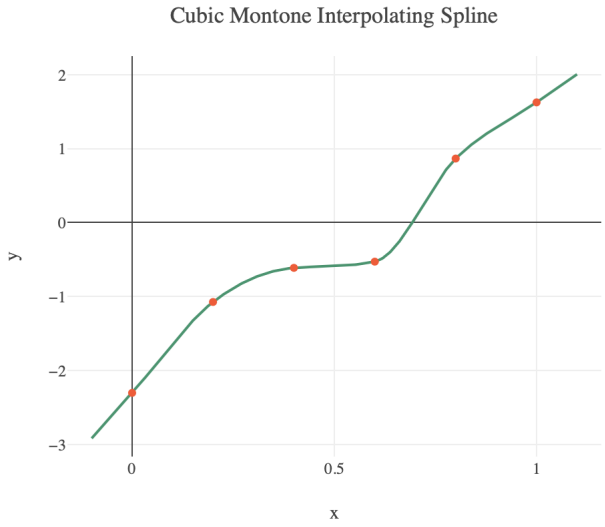

Figure 7.3: A demonstration fit with a cubic spline.

7.3 Research Goal

Produce numerically robust software that computes monotone quintic interpolating splines given data.

7.4 Timeline

The projected research timeline is shown in Table 7.1. It is expected that a significant number of *unpredictable changes* will occur as the code develops and the research progresses. I will update all committee members when appropriate.

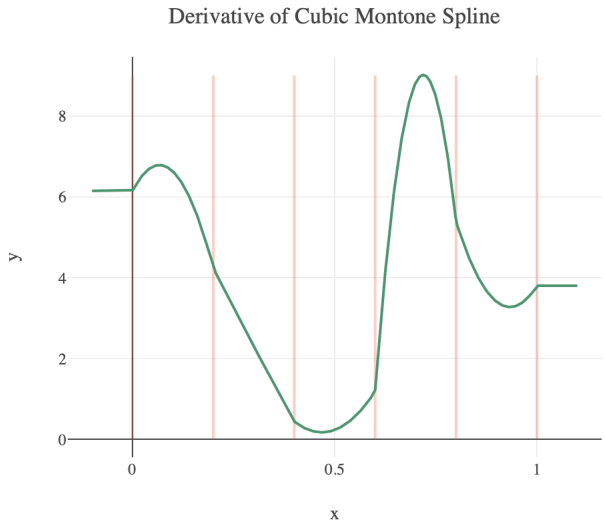

Figure 7.4: The derivative of Figure 7.2.

7.5 Final Remarks

This project has the potential to be very widely utilized. As described earlier, many statistical applications could greatly benefit from having a C^2 approximation to CDFs, because the resulting PDF will be both aesthetically pleasing and more accurate for continuous distributions.

Date	**Milestone**
June 2019	Implementation of monotone quintic polynomial *interpolant*.
August 2019	Implementation of monotone quintic polynomial *spline*.
October 2019	First draft of TOMS paper on algorithm.
December 2019	Research Defense of work.
March 2020	Submission of TOMS paper and code.
April 2020	Final Defense of Ph.D.

Table 7.1: This table depicts my expected timeline going forward. For details on when I achieved previous milestones, please contact me and I can provide the list. I refrained from including it here for brevity.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Mazhar Ali, Samee U Khan, and Athanasios V Vasilakos. Security in cloud computing: Opportunities and challenges. *Information Sciences*, 305:357–383, 2015.

[3] Brandon D. Amos, David R. Easterling, Layne T. Watson, William I. Thacker, Brent S. Castle, and Michael W. Trosset. Algorithm xxx: Qnstopquasi-newton algorithm for stochastic optimization. *Technical Report 14-02, Dept. of Computer Science, VPI&SU, Blacksburg, VA*, 2014.

[4] David H Bailey and Allan Snavely. Performance modeling: Understanding the past and predicting the future. In *European Conference on Parallel Processing*, pages 185–195. Springer, 2005.

[5] Kevin J. Barker, Kei Davis, Adolfy Hoisie, Darren J. Kerbyson, Michael Lang, Scott Pakin, and José Carlos Sancho. Using performance modeling to design large-scale systems. *Computer*, 42(11), 2009.

[6] Debasish Basak, Srimanta Pal, and Dipak Chandra Patranabis. Support vector regression. *Neural Information Processing-Letters and Reviews*, 11(10):203–224, 2007.

[7] Pete Beckman, Kamil Iskra, Kazutomo Yoshii, Susan Coghlan, and Aroon Nataraj. Benchmarking the effects of operating system interference on extreme-scale parallel machines. *Cluster Computing*, 11(1):3–16, 2008.

[8] Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, 5(Sep):1089–1105, 2004.

[9] K. W. Cameron, A. Anwar, Y. Cheng, L. Xu, B. Li, U. Ananth, J. Bernard, C. Jearls, T. C. H. Lux, Y. Hong, L. T. Watson, and A. R. Butt. Moana: Modeling and analyzing i/o variability in parallel system experimental design. *IEEE Transactions on Parallel and Distributed Systems*, 2019. ISSN 1045-9219. doi: 10.1109/TPDS.2019.2892129.

[10] R. Carlson and F. Fritsch. Monotone piecewise bicubic interpolation. *SIAM Journal on Numerical Analysis*, 22(2):386–400, 1985. doi: 10.1137/0722023. URL https://doi.org/10.1137/0722023.

[11] Tyler H. Chang, Layne T. Watson, Thomas C. H. Lux, Bo Li, Li Xu, Ali R. Butt, Kirk W. Cameron, and Yili Hong. A polynomial time algorithm for multivariate interpolation in arbitrary dimension via the delaunay triangulation. In *Proceedings of the ACMSE 2018 Conference*, ACMSE '18, pages 12:1–12:8, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5696-1. doi: 10.1145/3190645.3190680. URL http://doi.acm.org/10.1145/3190645.3190680.

[12] Elliott Ward Cheney and William Allan Light. *A Course in Approximation Theory*, volume 101. American Mathematical Soc., 2009.

[13] François Chollet et al. Keras. https://keras.io, 2015.

[14] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[15] Paulo Cortez and Aníbal de Jesus Raimundo Morais. A data mining approach to predict forest fires using meteorological data. *13th Portuguese Conference on Artificial Intelligence*, 2007.

[16] Paulo Cortez and Alice Maria Gonçalves Silva. Using data mining to predict secondary school student performance. *Proceedings of 5th Annual Future Business Technology Conference, Porto*, 2008.

[17] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[18] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013*, pages 8609–8613. IEEE, 2013.

[19] Pradipta De, Ravi Kothari, and Vijay Mann. Identifying sources of operating system jitter through fine-grained kernel instrumentation. In *IEEE International Conference on Cluster Computing*, pages 331–340. IEEE, 2007.

[20] Carl De Boor, Carl De Boor, Etats-Unis Mathématicien, Carl De Boor, and Carl De Boor. *A practical guide to splines*, volume 27. Springer-Verlag New York, 1978.

[21] Carl De Boor, Klaus Höllig, and Sherman Riemenschneider. *Box splines*, volume 98. Springer Science & Business Media, 2013.

[22] S De Vito, E Massera, M Piga, L Martinotto, and G Di Francia. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sensors and Actuators B: Chemical*, 129(2):750–757, 2008.

[23] John E Dennis Jr and Robert B Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, volume 16. Siam, 1996.

[24] G Lejeune Dirichlet. Über die reduction der positiven quadratischen formen mit drei unbestimmten ganzen zahlen. *Journal für die Reine und Angewandte Mathematik*, 40:209–227, 1850.

[25] Mathieu Dutour Sikirić, Achill Schürmann, and Frank Vallentin. Complexity and algorithms for computing voronoi cells of lattices. *Mathematics of Computation*, 78(267):1713–1731, 2009.

[26] F Ferri, P Pudil, M Hatef, and J Kittler. Comparative study of techniques for large-scale feature selection. *Pattern Recognition in Practice IV*, 1994:403–413, 1994.

[27] Jerome H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, pages 1–67, 1991.

[28] Jerome H. Friedman and the Computational Statistics Laboritory of Stanford University. Fast mars. 1993.

[29] F. Fritsch and R. Carlson. Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246, 1980. doi: 10.1137/0717021. URL https://doi.org/10.1137/0717021.

[30] Gabriel Goh. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006. URL http://distill.pub/2017/momentum.

[31] William J Gordon and James A Wixom. Shepards method of metric interpolation to bivariate and multivariate interpolation. *Mathematics of Computation*, 32(141):253–264, 1978.

[32] John A Gregory. Shape preserving spline interpolation. *Brunel University*, 1985. URL https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19850020252.pdf.

[33] Eric Grobelny, David Bueno, Ian Troxel, Alan D. George, and Jeffrey S. Vetter. Fase: A framework for scalable performance prediction of hpc systems and applications. *Simulation*, 83(10):721–745, 2007.

[34] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182, 2003.

[35] Walter Hess and Jochen W Schmidt. Positive quartic, monotone quintic c2-spline interpolation in one and two dimensions. *Journal of Computational and Applied Mathematics*, 55(1): 51–67, 1994. doi: 10.1016/0377-0427(94)90184-8.

[36] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[37] Gary D Knott. *Interpolating cubic splines*, volume 18. Springer Science & Business Media, 2012. doi: 10.1007/978-1-4612-1320-8.

[38] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[39] Dimitris Lazos, Alistair B Sproul, and Merlinde Kay. Optimisation of energy management in commercial buildings with weather forecasting inputs: A review. *Renewable and Sustainable Energy Reviews*, 39:587–603, 2014.

[40] Der-Tsai Lee and Bruce J Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980.

[41] Hubert W Lilliefors. On the kolmogorov-smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62(318):399–402, 1967.

[42] Jay Lofstead, Fang Zheng, Qing Liu, Scott Klasky, Ron Oldfield, Todd Kordenbrock, Karsten Schwan, and Matthew Wolf. Managing variability in the io performance of petascale storage

systems. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2010*, pages 1–12. IEEE, 2010.

[43] Thomas C H Lux, Randall Pittman, Maya Shende, and Anil Shende. Applications of supervised learning techniques on undergraduate admissions data. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 412–417. ACM, 2016.

[44] Thomas C H Lux, Layne T Watson, Tyler H Chang, Jon Bernard, Bo Li, Xiaodong Yu, Li Xu, Godmar Back, Ali R Butt, Kirk W Cameron, et al. Nonparametric distribution models for predicting and managing computational performance variability. In *SoutheastCon 2018*, pages 1–7. IEEE, 2018.

[45] Thomas C H Lux, Layne T Watson, Tyler H Chang, Jon Bernard, Bo Li, Xiaodong Yu, Li Xu, Godmar Back, Ali R Butt, Kirk W Cameron, et al. Novel meshes for multivariate interpolation and approximation. In *Proceedings of the ACMSE 2018 Conference*, page 13. ACM, 2018.

[46] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[47] W. D. Norcott. Iozone filesystem benchmark. 2017. URL http://www.iozone.org. http://www.iozone.org [Online; accessed 2017-11-12].

[48] Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. *Wright State University CORE Scholar Repository*, 2009.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[50] A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. Calibrating probability with undersampling for unbalanced classification. In *IEEE Symposium Series on Computational Intelligence*, pages 159–166. IEEE, Dec 2015. doi: 10.1109/SSCI.2015.33. URL https://www.kaggle.com/mlg-ulb/creditcardfraud. [Online; accessed 2019-01-25].

[51] Pavel Pudil, Jana Novovičová, and Josef Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119–1125, 1994.

[52] James O Ramsay et al. Monotone regression splines in action. *Statistical science*, 3(4): 425–441, 1988.

[53] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[54] J. Rudy and M. Cherti. Py-earth: A python implementation of multivariate adaptive regression splines. 2017. URL https://github.com/scikit-learn-contrib/py-earth. https://github.com/scikit-learn-contrib/py-earth [Online; accessed 2017-07-09].

[55] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[56] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524. ACM, 1968.

[57] Allan Snavely, Laura Carrington, Nicole Wolter, Jesus Labarta, Rosa Badia, and Avi Purkayastha. A framework for performance modeling and prediction. In *Supercomputing, ACM/IEEE Conference*, pages 21–21. IEEE, 2002.

[58] William I. Thacker, Jingwei Zhang, Layne T. Watson, Jeffrey B. Birch, Manjula A. Iyer, and Michael W. Berry. Algorithm 905: Sheppack: Modified shepard algorithm for interpolation

of scattered multivariate data. *ACM Transactions on Mathematical Software (TOMS)*, 37(3): 34, 2010.

[59] Athanasios Tsanas, Max A Little, Patrick E McSharry, and Lorraine O Ramig. Accurate telemonitoring of parkinson's disease progression by noninvasive speech tests. *IEEE Transactions on Biomedical Engineering*, 57(4):884–893, 2010.

[60] G. Ulrich and L. Watson. Positivity conditions for quartic polynomials. *SIAM Journal on Scientific Computing*, 15(3):528–544, 1994. doi: 10.1137/0915035. URL https://doi.org/10.1137/0915035.

[61] G unther Greiner and Kai Hormann. Interpolating and approximating scattered 3d-data with hierarchical tensor product b-splines. In *Proceedings of Chamonix*, page 1, 1996.

[62] Guanying Wang, Ali R. Butt, Prashant Pandey, and Karan Gupta. A simulation approach to evaluating design decisions in mapreduce setups. In *IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS'09)*, pages 1–11. IEEE, 2009.

[63] Guanying Wang, Aleksandr Khasymski, K. R. Krish, and Ali R. Butt. Towards improving mapreduce task scheduling using online simulation based predictions. In *International Conference on Parallel and Distributed Systems (ICPADS), 2013*, pages 299–306. IEEE, 2013.

[64] Graham J. Williams. Weather dataset rattle package. In *Rattle: A Data Mining GUI for R*, volume 1, pages 45–55. The R Journal, 2009. URL https://www.kaggle.com/jsphyg/weather-dataset-rattle-package. [Online; accessed 2019-01-25].

[65] Kejiang Ye, Xiaohong Jiang, Siding Chen, Dawei Huang, and Bei Wang. Analyzing and modeling the performance in xen-based virtual cluster environment. In *12th IEEE Interna-*

tional Conference on High Performance Computing and Communications (HPCC), pages 273–280. IEEE, 2010.

Appendices

Appendix A

Error Bound Appendices

The tables that follow show the precise experimental results for all data sets presented in Section 5.2. The tests were all run serially on an otherwise idle machine with a CentOS 6.10 operating system and an Intel i7-3770 CPU operating at 3.4 GHz. The detailed performance results in the tables that follow are very much dependent on the problem and the algorithm implementation (e.g., some codes are TOMS software, some industry distributions, and others are from conference paper venues). Different typeface is used to show best performers, however not much significance should be attached to ranking algorithms based on small time (millisecond) differences. The results serve as a demonstration of conceptual validity.

Algorithm	Min	25^{th}	50^{th}	75^{th}	Max
MARS	0.00984	3.11	7.01	*11.7*	1090.0
SVR	*0.00402*	**0.243**	**0.416**	**6.19**	1090.0
MLP	0.0426	2.63	*5.27*	14.0	1090.0
Delaunay	**0.00**	1.98	5.37	13.1	*1080.0*
ShepMod	**0.00**	*1.93*	6.27	16.0	1090.0
LSHEP	0.0400	2.17	8.87	19.1	**1070.0**
Voronoi	0.00982	3.65	7.56	15.6	1090.0
BoxSpline	**0.00**	2.27	5.91	16.4	1090.0

Table A.1: This numerical data accompanies the visual provided in Figure 6.2. The columns of absolute error percentiles correspond to the minimum, first quartile, median, third quartile, and maximum absolute errors respectively. The minimum of each column is boldface, while the second lowest value is italicized. All values are rounded to three significant digits.

Algorithm	% Best
MARS	*7.7*
SVR	**80.2**
MLP	0.0
Delaunay	3.5
ShepMod	3.7
LSHEP	5.1
Voronoi	0.0
BoxSpline	0.4

Fit/Prep. Time (s)	App. Time (s)	Total App. Time (s)
29.1	0.00137	0.0686
0.00584	*0.0000620*	*0.00310*
32.8	0.000871	0.0436
0.0151	0.0234	1.18
0.00634	0.0000644	0.00322
0.0275	**0.0000463**	**0.00231**
0.0152	0.000396	0.0198
0.00724	0.0000978	0.00489

Table A.2: The left above shows how often each algorithm had the lowest absolute error approximating forest fire data in Table A.1. On the right columns are median fit time of 454 points, median time for one approximation, and median time approximating 50 points.

Algorithm	Min	25^{th}	50^{th}	75^{th}	Max
MARS	0.00948	9.98	20.4	32.9	863.0
SVR	0.00138	3.17	7.31	12.6	**27.5**
MLP	0.0000239	*0.533*	*1.25*	**2.84**	39.3
Delaunay	3.72×10^{-12}	1.20	3.50	7.67	30.7
ShepMod	**0.0**	**0.255**	**0.908**	*3.43*	34.5
LSHEP	0.00254	2.93	7.16	13.1	*29.0*
Voronoi	**0.0**	1.29	3.52	6.87	30.1
BoxSpline	0.00287	4.39	9.10	14.8	41.3

Table A.3: This numerical data accompanies the visual provided in Figure 6.4. The columns of absolute error percentiles correspond to the minimum, first quartile, median, third quartile, and maximum absolute errors respectively. The minimum of each column is boldface, while the second lowest value is italicized. All values are rounded to three significant digits.

Algorithm	% Best
MARS	0.0
SVR	0.1
MLP	*32.0*
Delaunay	0.0
ShepMod	**66.4**
LSHEP	0.0
Voronoi	1.6
BoxSpline	0.0

Fit/Prep. Time (s)	App. Time (s)	Total App. Time (s)
37.9	0.00253	1.48
0.859	*0.000181*	*0.106*
348.0	0.00111	0.653
2.47	1.22	720.0
1.13	0.000182	0.107
2.39	**0.000144**	**0.0845**
2.77	0.0274	16.1
1.26	0.000643	0.377

Table A.4: The left above shows how often each algorithm had the lowest absolute error approximating Parkinson's data in Table A.3. On the right columns are median fit time of 5288 points, median time for one approximation, and median time approximating 587 points.

Algorithm	Min	25^{th}	50^{th}	75^{th}	Max
MARS	6.45×10^{-15}	$\mathbf{2.70 \times 10^{-14}}$	1.66	1.96	*53.3*
SVR	0.0420	0.143	0.148	*0.860*	119.0
MLP	0.0000689	0.0337	**0.0795**	**0.264**	**5.31**
Delaunay	**0.0**	0.187	0.919	2.72	56.3
ShepMod	**0.0**	0.0957	0.685	2.90	73.2
LSHEP	**0.0**	*0.0153*	*0.106*	1.17	113.0
Voronoi	**0.0**	0.430	1.28	2.94	83.8
BoxSpline	**0.0**	0.789	2.06	5.25	119.0

Table A.5: This numerical data accompanies the visual provided in Figure 6.6. The columns of absolute error percentiles correspond to the minimum, first quartile, median, third quartile, and maximum absolute errors respectively. The minimum value of each column is boldface, while the second lowest is italicized. All values are rounded to three significant digits.

Algorithm	% Best
MARS	10.7
SVR	4.1
MLP	**56.8**
Delaunay	0.1
ShepMod	3.5
LSHEP	*28.4*
Voronoi	0.2
BoxSpline	0.3

Fit/Prep. Time (s)	App. Time (s)	Total App. Time (s)
0.151	0.000117	0.0304
0.133	**0.0000947**	**0.0246**
169.0	0.00137	0.356
0.664	0.886	230.0
0.265	0.000128	0.0333
0.874	*0.0000975*	*0.0254*
0.675	0.0270	7.01
0.330	0.000406	0.106

Table A.6: Left table shows how often each algorithm had the lowest absolute error approximating Sydney rainfall data in Table A.5. On the right columns are median fit time of 2349 points, median time for one approximation, and median time approximating 260 points.

Algorithm	Min	25^{th}	50^{th}	75^{th}	Max
MARS	5.36	3610.0	4580.0	5450.0	13400.0
SVR	0.0138	8.47	15.4	41.9	7700.0
MLP	0.00151	**1.60**	**3.86**	**8.34**	**604.0**
Delaunay	**0.0**	*2.69*	*13.8*	*35.0*	*4840.0*
ShepMod	**0.0**	4.21	17.3	51.6	6510.0
LSHEP	1.27	199.0	260.0	343.0	6530.0
Voronoi	2.89×10^{-10}	14.7	32.8	52.9	4860.0
BoxSpline	0.00740	20.8	40.9	79.6	7640.0

Table A.7: This numerical data accompanies the visual provided in Figure 6.8. The columns of absolute error percentiles correspond to the minimum, first quartile, median, third quartile, and maximum absolute errors respectively. The minimum value of each column is boldface, while the second lowest is italicized. All values are rounded to three significant digits.

Algorithm	% Best
MARS	0.0
SVR	0.0
MLP	**79.5**
Delaunay	*20.5*
ShepMod	0.1
LSHEP	0.0
Voronoi	0.0
BoxSpline	0.0

Fit/Prep. Time (s)	App. Time (s)	Total App. Time (s)
22.0	0.00148	0.820
1.01	*0.000210*	*0.117*
290.0	0.000714	0.397
3.12	3.71	2070.0
1.45	0.000220	0.122
3.47	**0.000176**	**0.0981**
3.32	0.0950	52.8
1.66	0.000956	0.532

Table A.8: The left above shows how often each algorithm had the lowest absolute error approximating credit card transaction data in Table A.7. On the right columns are median fit time of 5006 points, median time for one approximation, and median time approximating 556 points.

Algorithm	Min	25^{th}	50^{th}	75^{th}	Max
Delaunay	**0.0287**	**0.0914**	**0.158**	*0.308*	*0.897*
ShepMod	0.0307	*0.0983*	*0.164*	0.335	1.00
Voronoi	*0.0303*	0.100	0.165	**0.274**	**0.762**
BoxSpline	0.0703	0.417	0.561	0.745	1.00

Table A.9: This numerical data accompanies the visual provided in Figure 6.10. The columns of absolute error percentiles correspond to the minimum, first quartile, median, third quartile, and maximum KS statistics respectively between truth and guess for models predicting the distribution of I/O throughput that will be observed at previously unseen system configurations. The minimum value of each column is boldface, while the second lowest is italicized. All values are rounded to three significant digits.

Algorithm	% Best
Delaunay	**62.0**
ShepMod	0.0
Voronoi	*38.0*
BoxSpline	0.0

Fit/Prep. Time (s)	App. Time (s)	Total App. Time (s)
0.344	0.00984	2.71
0.0884	**0.000145**	**0.0436**
0.360	0.00253	0.762
0.0972	*0.000210*	*0.0633*

Table A.10: The left above shows how often each algorithm had the lowest KS statistic on the I/O throughput distribution data in Table A.9. On the right columns are median fit time of 2715 points, median time for one approximation, and median time approximating 301 points.