Scalable Distribution-to-Distribution Regression

Andrea Klein

CMU-CS-14-123
July 2014

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Shirley Ho, Department of Physics
Jeff Schneider, Robotics Institute
Karl Crary

Submitted in partial fulfillment of the requirements for the degree of Masters in Computer Science.

Keywords: Machine Learning, Cosmology, Dark Matter, N-body Simulations

For my grandfather.

Abstract

In this thesis I present a scalable approach to distribution-to-distribution regression on large, multi-dimensional datasets. The basic algorithm is demonstrated on 1-dimensional toy data, then modified for efficiency and scalability. Key enhancements include parallel computation of non-parametric estimators and the use of a ball tree to support efficient nearest-neighbor search in high dimension. I then explore the ability of this technique to compute the final states of cosmological N-body simulations. An existing method uses cosmological perturbation theory to rapidly approximate the evolution of simulations; I attempt to learn the unknown function from the approximate to the true distributions, thereby exploiting the speed of perturbative approximation while still approaching the accuracy of a true N-body simulation. I investigate whether it is possible to train the algorithm on $O(1)$ simulations that have been run both exactly and approximately, thereby making it possible to quickly generate many more final simulation states via regression.

Acknowledgments

I thank my advisors in computer science, for letting me get away with all this, and my advisors in machine learning and cosmology, for helping me pull it off.

Contents

Chapter 1

Introduction

In statistics, a traditional regression model relates some dependent variables $\mathbf{Y}$ to a function f of some independent "feature" variables $\mathbf{X}$ and perhaps some unknown parameters β: $\mathbf{Y} \approx f(\mathbf{X}, \beta)$. Here, $\mathbf{X}$ and $\mathbf{Y}$ are typically vectors with real-valued components. However, the domain to which regression methods can be applied has begun to expand. In particular, there has been growing interest in performing regression on or between infinite-dimensional domains, such as the space of probability distributions. Recent efforts in this direction include regression from a probability distribution to a real-valued scalar response [1] and from one probability distribution to another [7].

The work by [7], especially, opens up an interesting new domain of real-world problems to which regression analysis may be applied. Their algorithm makes only weak assumptions about the nature of the input and output distributions, so its potential applications are wide-ranging. Indeed, they demonstrate an implementation on both synthetic and real data. However, the algorithm as presented in [7] is most suitable for applications in which there is an abundant amount of training data and the size of each training sample is small. In some real scenarios, the difficulties involved in making training samples may be the motivation for using regression in the first place. Indeed, the samples may be computationally expensive both to generate and to manipulate. In this case, the algorithm suffers from two basic problems: first, there may not be enough training data in order to make accurate predictions for new input instances; and second, the computations involved in the algorithm may be insufficiently efficient to justify its use.

Consider one promising scientific domain in which this kind of algorithm may be useful, namely the simulation of cosmic structure. Cosmologists use N-body simulations to study the formation of galaxies, measure cosmological parameters, constrain the properties of dark matter, and assist with other key areas of research. Though invaluable, these simulations can be very computationally expensive to make, and it is generally necessary to sacrifice quality for quantity in order to make enough simulations to derive meaningful statistics.

One way of computing a fast but approximate final simulation state is by using cosmological perturbation theory. Without going into detail here (see Chapter 5 for a brief technical overview of the theory), perturbation theory offers a way to approximate the evolution of a cosmological system to greater and greater accuracy by computing successively higher-order terms. When only, say, first- and second-order terms are included, the computation is typically much faster than the equivalent N-body simulation.

In one sense, perturbative methods clearly compete with machine learning in this domain: they offer a fast, controllably approximate alternative to running a full simulation. At the same time, however, they may contribute an alternative source of training data for machine learning algorithms. That is, suppose you need a large number of simulations run. You could run a few of them to completion, both exactly using N-body simulation, and approximately using perturbative theory. The density distributions in the exact final simulation states may be regarded as an unknown function of the distributions in the approximate final states.

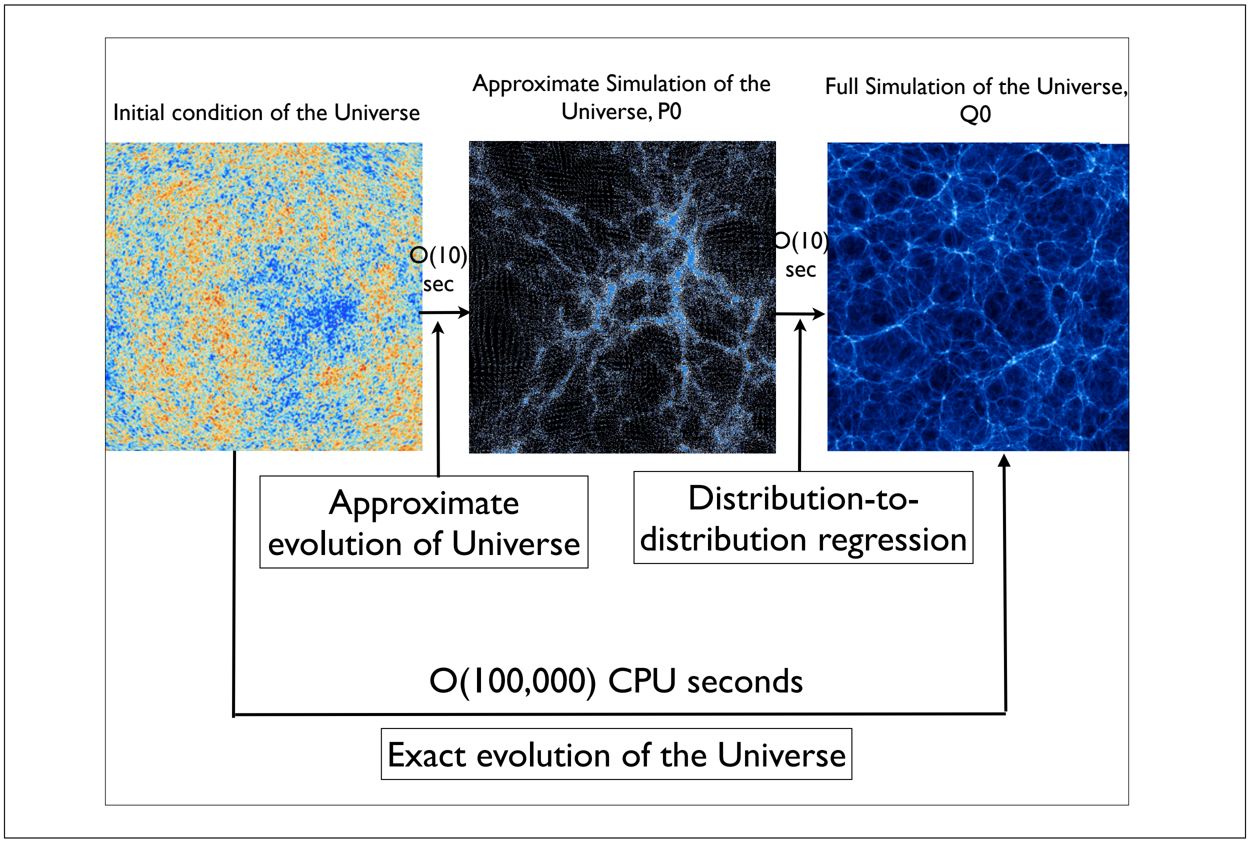

Initial condition of the Universe
Approximate Simulation of the Universe, P0
Full Simulation of the Universe, Q0
O(10) sec
O(10) sec
Approximate evolution of Universe
Distribution-to-distribution regression
O(100,000) CPU seconds
Exact evolution of the Universe

Figure 1.1: Comparison between the intended DTDR application and ordinary N-body simulation. Image courtesy of Professor Shirley Ho.

Now suppose machine learning, specifically distribution-to-distribution regression, could be used to determine that function after training on only a few of the simulations. If this worked well enough, you could run approximate versions of the remaining simulations, then use the approximate final states as input to the regression algorithm. The regressor could then estimate the final states that would have resulted from running the approximated simulations to completion as N-body simulations. That is, once the regressor had been "trained" on enough examples of approximate and exact simulations, it could be used to produce many more final simulation states relatively cheaply from their initial conditions, with the perturbative approximation as an intermediate step. Indeed, this is the approach to simulation regression that I will pursue in this thesis. The hope, at illustrated in Figure 1.1, is that the regression step can be performed about as fast as the perturbative approximation, so that the computation is much faster than comparatively brute-force N-body simulation.

If machine learning could be used to efficiently generate final, high-resolution simulations from initial conditions, a large body of research would be enabled or enhanced as a result. However, only a few high-resolution simulations may reasonably be run to completion for use as training data, and each simulation dataset is quite large, consisting of perhaps hundreds of millions or even billions of particles. It is therefore necessary to work with a scaled version of the algorithm in this domain, if this kind of regression approach is to be useful. I will pursue one such approach to scaling, with some associated algorithmic optimizations, and analyze its effectiveness on simulation data.

In this thesis, I explain the original algorithm from [7], as well as the related mathematics, in Chapter 2. In Chapter 3, I recreate the success of the original algorithm on the 1D toy example in [7]. I introduce and justify certain modifications that allow for a more efficient implementation. I then develop a scaled version of the algorithm, suitable for applications involving much larger, higher-dimensional datasets, in Chapter 4. This necessitates some discussion of implementation details with regards to managing and performing computations efficiently on large training samples. I demonstrate the effectiveness of the scaled version on datasets from N-body simulations of dark matter in Chapter 5. Finally, I conclude the thesis with some discussion of the domains to which both versions of the algorithm may reasonably be applied.

Chapter 2

Original Algorithm

The approach taken in [7] may be understood by analogy to regression between real-valued scalars: given a number of training examples of the form (x, y), where x and y are both scalar values related by a function $f(x) = y$, the goal is to estimate $g(x_0)$ for some arbitrary value x_0 that likely does not appear in the training set. Often, a linear smoother is used as a nonparametric estimator for g. Intuitively, those values of Y corresponding to X instances close to x_0 should yield a better approximation to its true function value y_0. So we could estimate y_0 as $\hat{f}(x_0) = \sum_i w(x_i - x_0)y(x_i)$, where the index ranges over all the training examples, and the weighting function w gives greater weight to training examples more "similar" to x_0.

The proposed algorithm works similarly, with a few key distinctions. In particular, rather than a query scalar x_0, we have a query distribution P_0. In practice, we do not observe P_0 directly; rather, we observe it indirectly through a sample $X_0 = \{X_{01}, \ldots, X_{0n_0}\} \overset{iid}{\sim} P_0$. Using samples taken from the input and output distributions, we can estimate their pdfs; we can then estimate the pdf $f(p_0)$ with a linear smoother of the form:

$$\hat{f}(\hat{p_0}) = \sum_{i=1}^{M} \hat{q}_i W(\hat{P}_i, \hat{P}_0) \tag{2.1}$$

Here, $\hat{P}_0$ is the estimator of P_0 obtained from the sample X_0.

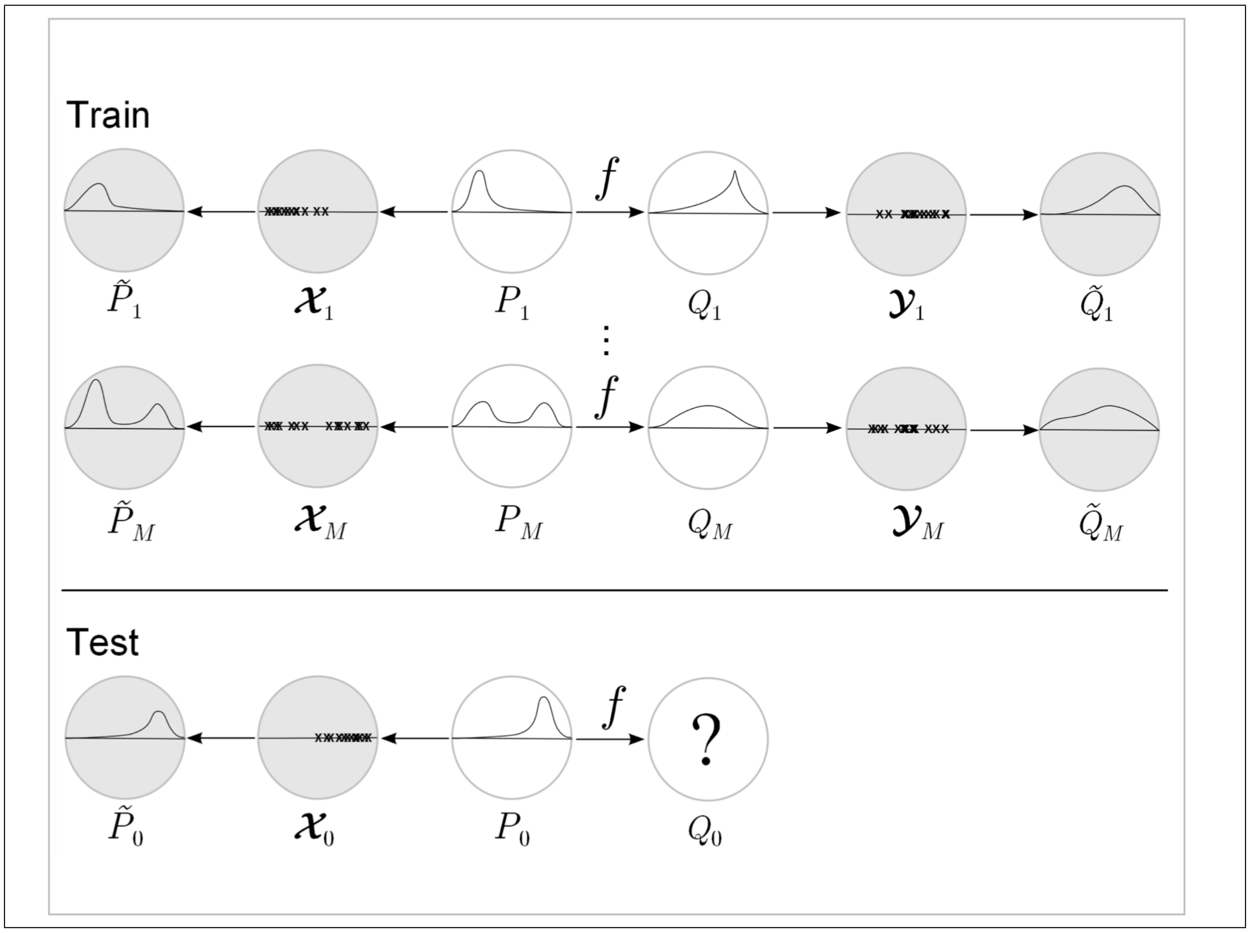


Figure 2.1: Graphical overview of the structure of the data. The training data consists of M input-output pairs, each of which is considered to be a sample from an unknown probability density function. The goal is to learn the function f in order to compute $f(X_0)$ for a test instance X_0.

Recall that we do not directly observe the output densities $q_i = f(p_i)$; rather, in each case, we must work with a finite sample drawn from q_i. This concept is illustrated in Figure 2.1, taken from [7]. Note the structure of the data: each training instance consists of two samples, X_i and Y_i, drawn from unknown distributions $\hat{P}_i$ and $\hat{Q}_i$, respectively.

Since the true distributions that gave rise to the training data are unknown, it is necessary to infer some estimate of q_i from each sample. This can be done in a nonparametric way using an orthogonal series estimator (see e.g. [12], [6]), which allows you to build an arbitrary distribution out of some set of basis functions.

Orthogonal series density estimation is simplest to understand in a single dimension. Suppose we have a random variable X whose range is restricted to $[0, 1]$. Suppose also that the probability density f of X is square integrable. Then, given some orthonormal basis $\{\phi_j\}$, f may be approximated to arbitrary accuracy by a partial sum of the form

$$f_J(x) := \sum_{j=0}^{J} \theta_j \phi_j(x), 0 \leq x \leq 1 \tag{2.2}$$

where

$$\theta_j = \int_0^1 \phi_j(x) f(x) dx. \tag{2.3}$$

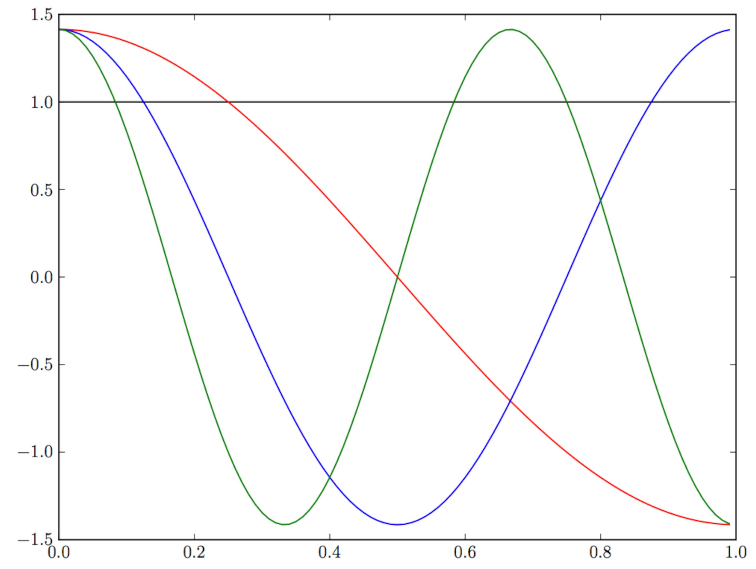


Figure 2.2: Cosine basis functions ϕ_j for j = 0, 1, 2, and 3.

For instance, $\{\phi_j\}$ could be the cosine basis: $\{\phi_0(x) = 1, \phi_j(x) = \sqrt{2}\cos(\pi j x), j = 1, 2, \dots\}$. Indeed, this is the basis I use throughout the project; the first few functions are shown in Figure 2.2. The parameter J is known as the cutoff, and in this case, the coefficient θ_j is called the jth Fourier coefficient of f. Now, we automatically know the value of the 0th coefficient: $\theta_0 = \int_0^1 f(x)dx = P(X \in [0,1]) = 1$, since $[0,1]$ is the range of the variable X. For the remaining coefficients, note that

$$\theta_j = \int_0^1 f(x)\phi_j(x)dx = E\{\phi_j(X)\}. \tag{2.4}$$

That is, since each of the coefficients is simply an expectation value, we may estimate each one via the sample mean estimator:

$$\theta_j = \frac{1}{n}\sum_{l=1}^{\hat{n}} \phi_j(X_l). \tag{2.5}$$

Now suppose we are dealing more generally with a multi-dimensional output distribution q defined on the unit hypercube; i.e. the domain of the output distribution, $\Lambda^l \subseteq \mathbb{R}^l$, and $\Lambda = [0,1]$.

Then, given an orthonormal basis $\{\phi_i\}_{i\in\mathbb{Z}}$, the tensor product $\{\phi_{\alpha_{\alpha\in\mathbb{Z}^l}}\}$ of that basis serves as an orthonormal basis for $L_2(\Lambda^l)$, the space of square-integrable functions on Λ^l, where:

$$\phi_\alpha = \Pi_{i=1}^l \phi_{\alpha_i}(x_i), x \in \Lambda^l \tag{2.6}$$

In other words, $\forall \alpha, \gamma \in \mathbb{Z}^l$, the inner product $\langle \phi_\alpha, \phi_\gamma \rangle = I_{\alpha=\gamma}$, where I is the unit matrix.

Then, the output distribution q may be expressed as

$$q(x) = \sum_{\alpha\in\mathbb{Z}^l} a_\alpha(Q)\phi_\alpha(x) \tag{2.7}$$

where

$$a_\alpha(Q) = \langle \phi_\alpha, q \rangle = \int_{\Lambda^l} \phi_\alpha(z) dQ(z) \in \mathbb{R}. \tag{2.8}$$

Given estimators for each $\tilde{q_i}$, the estimator $\hat{q_0} = \hat{f}(\tilde{p_0})$ may be expressed as:

$$\begin{aligned}
\hat{q_0}(x) = [\hat{f}(\tilde{p_0})](x) &= \sum_{i=1}^{M} \tilde{q_i}(x) W(\tilde{P_i}, \tilde{P_0}) && (2.9)\\
&= \sum_{i=1}^{M} \left(\sum_{\alpha \in A_t} a_\alpha(\hat{Q}_i)\phi_\alpha(x) \right) W(\tilde{P_i}, \tilde{P_0}) && (2.10)\\
&= \sum_{\alpha \in A_t} \left(\sum_{i=1}^{M} a_\alpha(\hat{Q}_i) W(\tilde{P_i}, \tilde{P_0}) \right) \phi_\alpha(x) && (2.11)\\
&= \sum_{\alpha \in A_t} \hat{a_\alpha} \phi_\alpha(x) && (2.12)
\end{aligned}$$

where $\hat{a_\alpha} = \sum_{i=1}^{M} a_\alpha(\hat{Q}_i) W(\tilde{P_i}, \tilde{P_0})$ and the weights $W(\tilde{P_i}, \tilde{P_0})$ are obtained by kernel smoothing:

$$W(\tilde{P_i}, \tilde{P_0}) = \begin{cases} \frac{K(\frac{D(\tilde{P_i}, \tilde{P_0})}{b})}{\sum_{j=1}^{M} K(\frac{D(\tilde{P_j}, \tilde{P_0})}{b})}, & \text{if } \sum_{j=1}^{M} K(\frac{D(\tilde{P_j}, \tilde{P_0})}{b}) > 0. \\ 0, & \text{otherwise}. \end{cases} \tag{2.13}$$

Here, D is a metric, K is a kernel function, and b is a bandwidth parameter equal to the sum of the distances from any test input distribution to all the training test distributions. Note that the bandwidth effectively normalizes $D(\tilde{P_i}, \tilde{P_0})$, controls the degree of smoothing, and affects whether the input to K is between 0 and 1, which is especially significant if K is only defined on that interval. In [7], D is taken to be the L_1 distance, $D(\tilde{P_i}, \tilde{P_0}) = \|\tilde{p_0} - \tilde{p_i}\|_1 = \int |\tilde{p_0}(x) - \tilde{p_i}(x)| dx$ (which is indeed defined only on [0,1]), and the regression weights are computed using the triangle kernel: $(1 - |x|)_+$. The kernel function is strictly decreasing on the positive x-axis, so more similar training examples (i.e. with lower distances to a test input) are weighted higher.

Chapter 3

1D Implementation and Experiments

3.1 Toy Data

I will now demonstrate an implementation of the algorithm just described on synthetic, 1-dimensional data drawn from distributions defined on $[0, 1]$. I used distributions of the same form as used in [7], and I learned the same function, namely reflection about a vertical axis at $x = 0.5$. To create each input/output pair, I first drew $\mu_1, \mu_2 \sim$ Unif[0, 1] and $\sigma_1, \sigma_2 \sim$ Unif[.05, .1]. Then the input $p(x)$ and output $q(x)$ were defined as follows:

$$p(x) = \frac{1}{2}g(x; \mu_1, \sigma_1) + \frac{1}{2}g(x; \mu_2, \sigma_2) \tag{3.1}$$

$$q(x) = \frac{1}{2}g(x; 1 - \mu_1, \sigma_1) + \frac{1}{2}g(x; 1 - \mu_2, \sigma_2) \tag{3.2}$$

Here, $g(x)$ is the truncated normal pdf on [0, 1]:

$$g(x; \mu, \sigma) = \frac{\frac{1}{\sigma}\phi(\frac{x-\mu}{\sigma})}{\Phi(\frac{1-\mu}{\sigma}) - \Phi(\frac{-\mu}{\sigma})} \tag{3.3}$$

where ϕ and Φ refer to the standard normal pdf and cdf, respectively. See Figure 3.1 for an example of an input/output distribution pair.

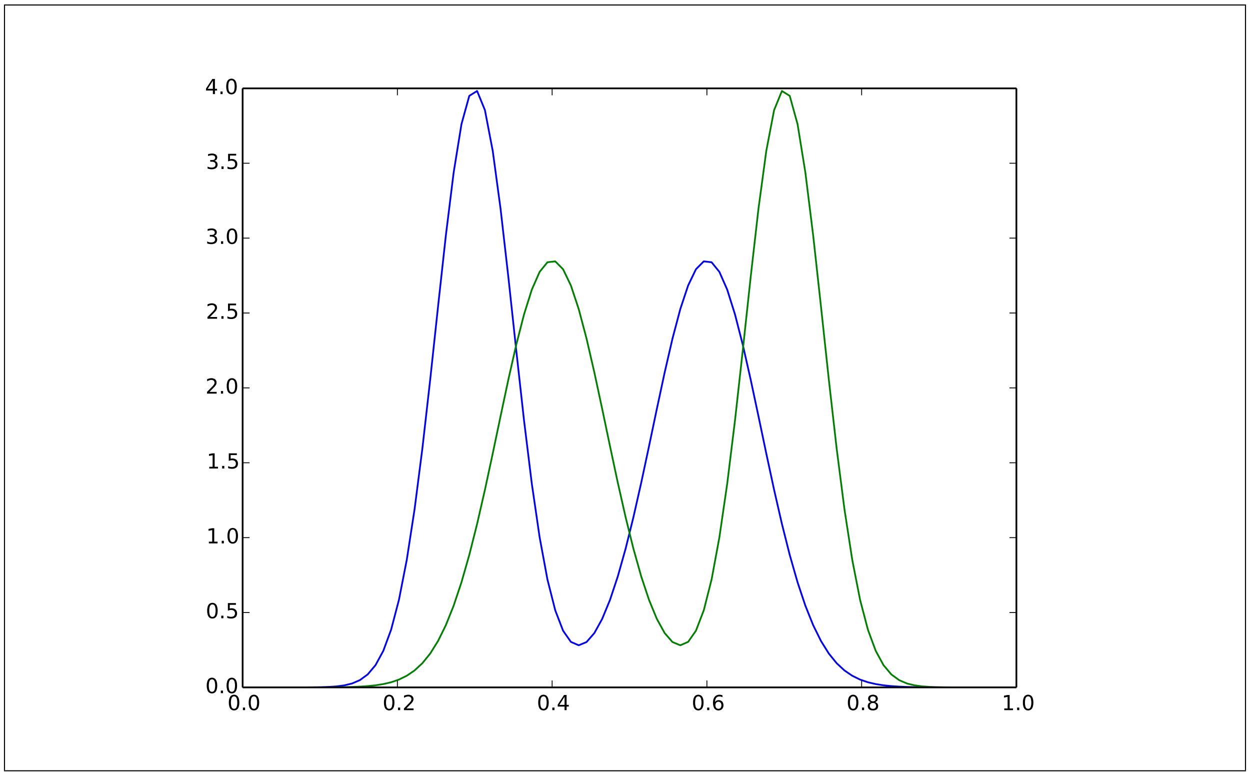
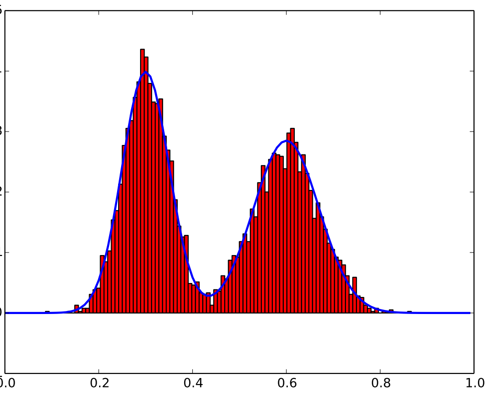


Figure 3.1: An example input (blue) and output (green) distribution pair with $\mu_1 = .3$, $\mu_2 = .6$, $\sigma_1 = .05$, $\sigma_2 = .07$.

In each experiment, I generated M training instances of the form (p_i, q_i). Then for each pair, I drew a sample X_i from p_i and a sample Y_i from q_i, where both samples consisted of η points: $|X_i| = |Y_i| = \eta$. Since both M and η have the same impact on the L_2 error [7], for convenience, I held $M = \eta$. I fit each sample by computing the first T coefficients of a Fourier series approximation (see Figure **??**). I retained 10% of the training instances for cross-validation of the bandwidth, which is chosen from a hard-coded set of representative values. Finally, I generated another $.10M$ testing instances in the same manner, and I computed estimators for the testing output distributions according to the prescription in Chapter 2.



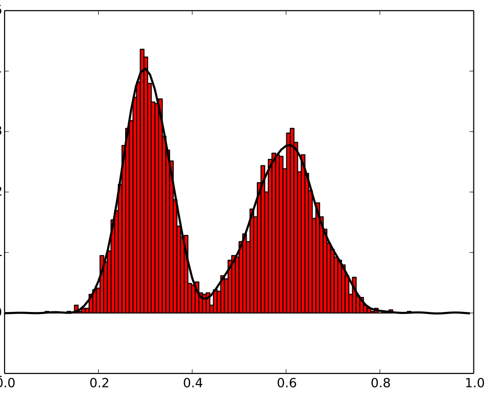


Figure 3.2: Figure (a) shows an example distribution with $\mu_1 = .3$, $\mu_2 = .6$, $\sigma_1 = .05$, $\sigma_2 = .07$ (blue curve). Normalized sample of 1000 points drawn from the distribution (red histogram). Figure (b) shows the same sample from (a), shown with 20-term orthogonal series density estimator of the parent distribution (black curve).

3.2 Modifications to the Original Algorithm

Here I introduce some immediate modifications to the original algorithm, independently of any effort at scaling. These changes are, for the most part, relatively minor, and should only serve to make the algorithm more efficient in practice. They do anticipate some challenges associated with the scaled algorithm, which will be addressed in greater detail in the next section.

Now, recall that the original version of the algorithm defined the "distance" between two distribution in terms of the L_1 norm. However, the L_1 norm is an integral of an absolute value, so in practice, this requires computing a numerical integral for each pair of distributions. This is quite inefficient, especially when done to a desirable accuracy. It is possible to use the L_2 norm instead without corrupting any of the proofs in [7], and indeed, this turns out to be preferable for several reasons. In particular, note that the L_2 distance simplifies to a sum of squared differences of the coefficients c, i.e.

$D(\tilde{P}_i, \tilde{P}_0) = \|\tilde{p_0} - \tilde{p_i}\|_2^2 = \sum_j (c_{0,j} - c_{i,j})^2$. This makes it possible to work directly with the coefficients rather than actually building the nonparametric estimator functions, and in testing it speeds up the computation of pairwise distances by a factor of $\sim 10,000$. I have also used the L_2 distance as an error metric when choosing the bandwidth: the bandwidth that yields the lowest average L_2 distance between the predicted and approximate distributions for the cross-validation instances is selected for regression on the test set. Ultimately, the performance on the test set is also measured in terms of the L_2 distance.

Similarly, the use of the triangle kernel was not essential to the original algorithm; any kernel would do. Among the disadvantages of the triangle kernel is that it requires checking whether each input lies in the range [0,1]; if not, the kernel output must be set to 0. If you try to force the inputs into that range via the bandwidth, you risk oversmoothing your estimator, which may have achieved a lower cross-validated error for an even smaller bandwidth. Consequently, I have worked instead with the Gaussian (RBF) Kernel, $K(x) = e^{-x^2/2}$, which is defined for all non-negative x.

Another inhibiting factor in the speed of the algorithm, even on small datasets, is the need to consider every training instance when regressing on a single test instance. Certainly, for most datasets, the vast majority of the training instances will be quite dissimilar to a given test instance, so their impact on the quality of the regression will be minimal. For this reason, it makes sense to consider only the K nearest neighbors of a test instance, i.e. the K most similar training instances, during the regression phase. Unlike the previous modifications, this change has a potential to impact the accuracy of the algorithm in favor of speeding it up. Consequently, the performance implications will be examined carefully in the next section.

3.3 Free Parameters and Performance

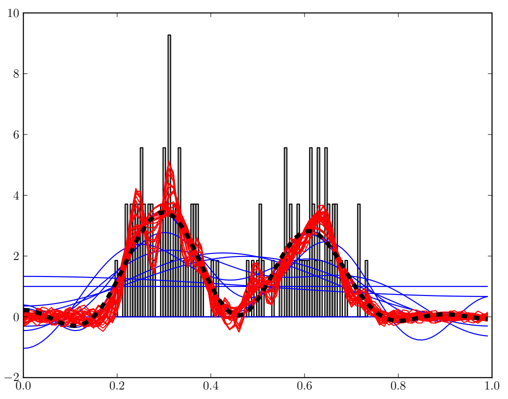
Since training data consists of samples from distributions rather than distributions themselves, there is necessarily some approximation inherent to their representation. Increasing the number of terms T in the nonparametric estimator improves the accuracy of that approximation at the cost of slowing the computation somewhat. However, it's clear that the number of samples available from each distribution also limits the accuracy to which it can be approximated, so past a certain point, adding more terms to the nonparametric estimator should yield no further benefit.

This effect can be seen in Figure 3.3. In particular, when the number of samples is small (e.g. 100), a high value of T causes visible overfitting. In general, a reasonable value of T may be ascertained empirically. But formally, the optimal T value minimizes the unbiased risk, given by:

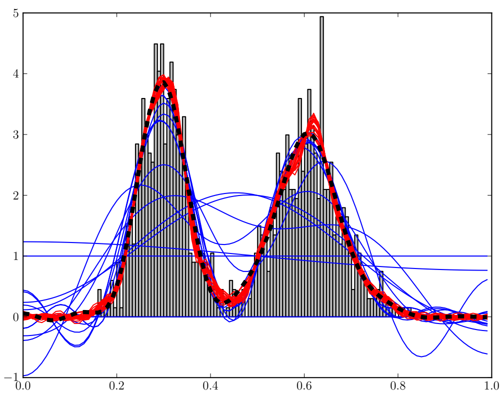
$$\hat{J}(T) = \frac{1}{n-1} \sum_{j=1}^{T} \left[\frac{2}{n} \sum_{i=1}^{n} \phi_j^2(X_i) - (n+1)\hat{c}_j^2 \right] \tag{3.4}$$

It's worth noting that increasing T past a certain point may simultaneously waste computational effort and cause overfitting, especially for small sample size. This could be avoided by determining T via cross-validation, alongside the bandwidth, although this would require testing a grid of (T, b) values and would therefore be inefficient relative to the expected gains in performance. So instead, as a rule of thumb, I let $T = 5\log(x)$, where x is the number of samples taken from the distribution. This rule was used to determine the cutoff values of T in Figure 3.3.

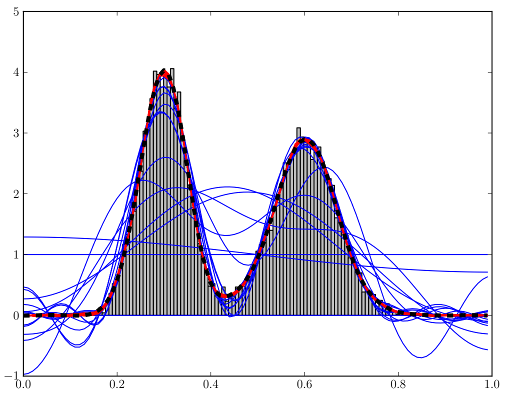
While the exact value of $\hat{J}(T)$ will naturally vary with the particular sample involved, this rule tends to hold up well when $\hat{J}(T)$ is explicitly calculated for a toy sample. Consider the curve in Figure 3.4, for example, which shows that $\hat{J}$ is minimized at or above the recommended value of $T(N)$. Conveniently, this sort of plot can still be made when the samples, coefficients, etc. have more than one dimension, i.e.



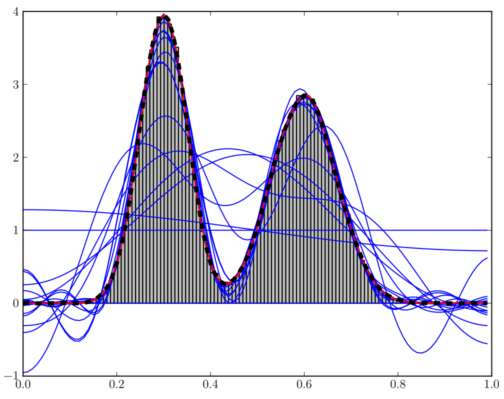
(a) 100 Samples; T_{cut}=10



(b) 1,000 Samples; T_{cut}=15



(c) 10,000 Samples; T_{cut}=20



(d) 100,000 Samples; T_{cut}=25

Figure 3.3: Samples from distribution with $\mu_1 = .3$, $\mu_2 = .6$, $\sigma_1 = .05$, $\sigma_2 = .07$. Curves shown use T values from 0 through 40. In each image, the curve from a cutoff value T_{cut} is shown in dashed black; curves with smaller values of T are in blue, and those with larger values are in red.

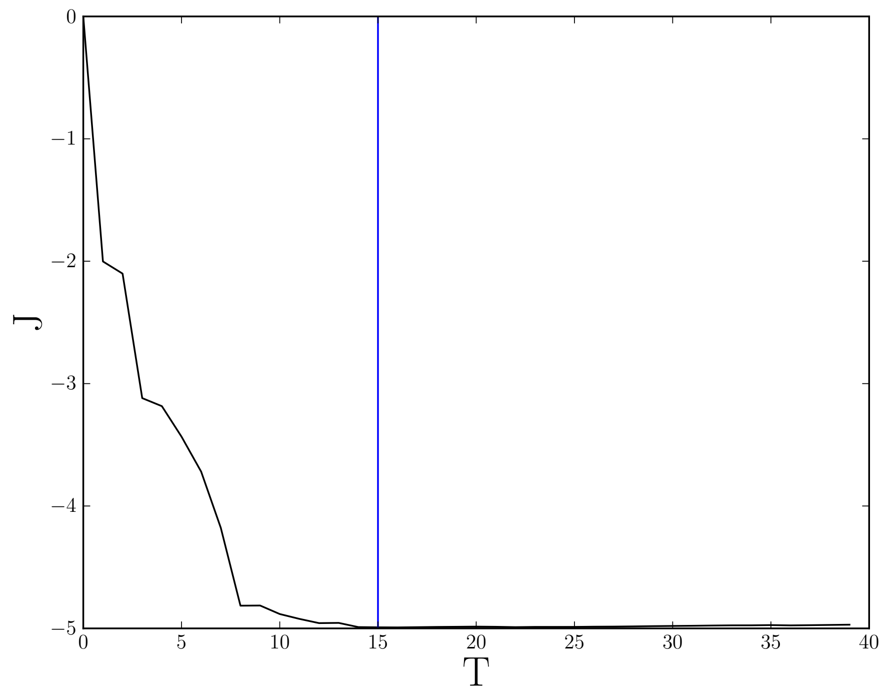


Figure 3.4: $\hat{J}(T)$ for a set of 1,000 samples. The rule-of-thumb cutoff value of T, i.e. $5\log(1000) = 15$, is indicated with the vertical blue line.

when plots in the style of Figure 3.3 are no longer useful. That is, $\hat{J}$ can still be used to determine a reasonable number of coefficients in the higher-dimensional case, a fact that will be exploited in Section 5.

Once the bandwidth and T are chosen, it remains to set the value of K, i.e. the number of nearest neighbors to use during regression. There's little theoretical motive for choosing one value of K over another; in general, an appropriate balance between efficiency and accuracy must be found empirically. Indeed, since there may be relatively few training instances quite similar to a given test instance, good testing performance may be achievable even with a surprisingly small value of K.

In fact, in some cases, there may be a training instance so similar to a test instance that even with K = 1, the regressed output is already quite close to the (approximate) true output distribution. In part, this reflects the quality and abundance of the training data; if, for example, there were fewer training instances, it would be reasonable to expect some unusual features in the regressed distribution for very small K. In any case, for this example, bringing K to 10 smoothes out the tails of the distribution a bit; however, the improvements gained by bringing K from 10 to 10,000 appear to be minimal.

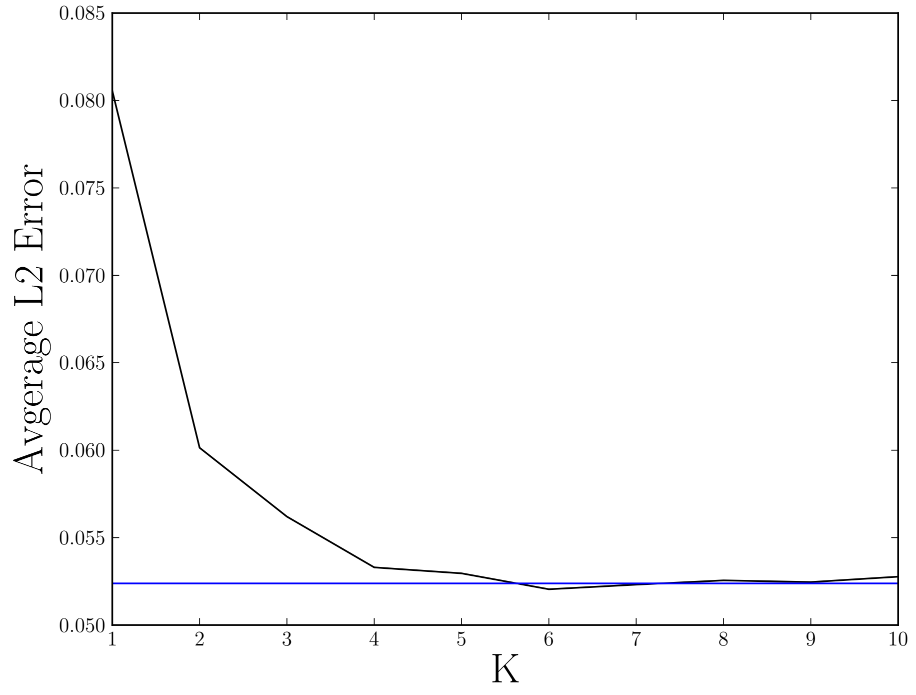


Figure 3.5: Average test error over 100 test instances, using training data with $M = 1000$, $\eta = 1000$. For comparison, the average test error using K = 1000 (all the training data) is indicated in blue.

This intuition can be confirmed by studying how the average L_2 test error varies with K. The results of experiments on toy data with $M = 1000$, $\eta = 1000$ are shown in Figure 3.5. It's clear that even for small K, around 5 or 6, the test error is competitive with that achieved by using all of the training data. However, finding these neighbors by brute-force, e.g. by sorting all the training instances by their distance to a given test instance and selecting the first K, is computationally expensive. Suppose the number of test instances we wish to regress on is $\phi \times M$, for some constant fraction ϕ of the size of the training data. Then we must sort all M training instances for each test instance, for a total runtime of $O(M^2 \log(M))$. For large datasets, this is completely infeasible, so efficient KNN will be addressed in the next section as part of the scaled algorithm.

Chapter 4

Scaled Algorithm

In this section I begin to make use of 6-dimensional data, in certain places, either to compare with 1-dimensional data or to demonstrate how some component of the algorithm scales with the number of dimensions involved. While the 6-dimensional data is taken from N-body simulations, its content is not important at this stage, and it is included only for illustrative purposes. The precise nature of the scientific data will be elaborated upon more fully in Chapter 6.

4.1 Coefficient Computation in Many Dimensions

Recall that for square-integrable densities defined in D dimensions, each basis function is indexed by a vector α of length D. In particular, in 6 dimensions,

$$\phi_\alpha = \Pi_{i=1}^{6} \phi_{\alpha_i}(x_i) \tag{4.1}$$

where the functions ϕ are 1-dimensional basis functions as described previously. For concreteness, here is an example: fitting a sample to 3rd degree would require basis functions indexed from $\alpha = [1,1,1,1,1,1]$ to $\alpha = [3,3,3,3,3,3]$. This estimator would include intermediate terms indexed by e.g. $\alpha = [3,2,1,1,3,3]$, where in this case $\phi_\alpha = \phi_{[3,2,1,1,3,3]}(\vec{x}) = \phi_3(x_1)\phi_2(x_2)\phi_1(x_3)\phi_1(x_4)\phi_3(x_5)\phi_3(x_6)$. Note that the coefficient corresponding to a given ϕ_α is, as before, given by the average value of ϕ_α on all the samples being fit.

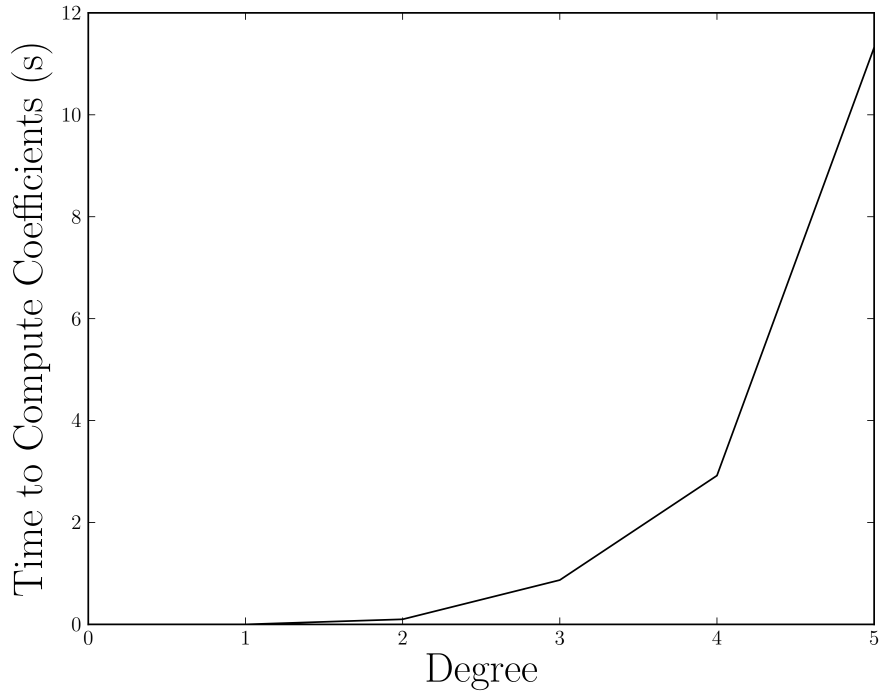


Figure 4.1: Timing curve for Fourier estimator computation on a sample of 100 6-dimensional particles.

In order to fit a sample to an estimator of degree T, it is now necessary to allow each element of α to range between 0 and T. Hence, an estimator of degree T involves T^6 terms, i.e. the number of terms (and hence coefficients that must be computed) scales exponentially with the degree of the estimator. This is immediately problematic for data of even so few as 6 dimensions. Consider, for example, the timing curve in Figure 4.1. Here, coefficients are being computed for a mere 100 samples from some 6-dimensional distribution. Note that for a degree-5 estimator, the number of terms is already $5^6 = 15,625$, requiring $O(10)$ seconds to compute. It's evident not only that this computation could benefit from some degree of parallelization, but indeed, that it demands it, as the next subsection will address.

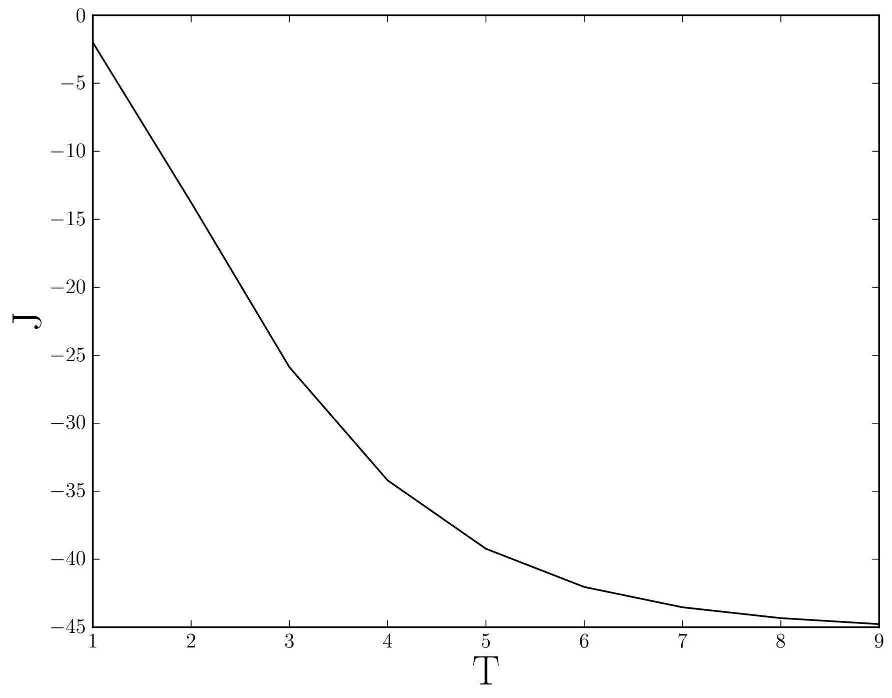


Figure 4.2: $\hat{J}(T)$ for a set of ∼25,000 3-dimensional samples.

It is reasonable to question how many of these terms must actually be computed; after all, on 1-dimensional data, empirical measurement of $\hat{J}$ indicated that only a few coefficients were actually needed. Intuitively, however, the degree of the estimator is a better indicator of its quality than the absolute number of terms. Indeed, similar experiments on multi-dimensional data, e.g. 4.2, indicate a need for degrees in the range $O(1)$ to $O(10)$, similar to the absolute number of coefficients needed for 1-dimensional data.

4.2 Parallelization

Although a nearest-neighbors tree must, if desired, be constructed sequentially, the computation of the nonparametric coefficients is inherently highly parallel. Indeed, dividing this task between workers in a pool of processes can significantly reduce the amount of time needed for the training phase. Here I will describe some experiments performed on 1-dimensional data that illustrate the advantage of parallelization relative to a naive, sequential computation.

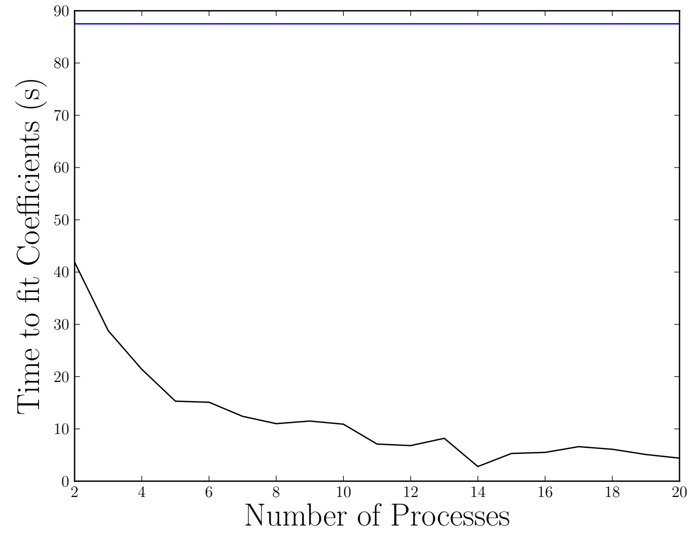


Figure 4.3: Timing curve for parallel coefficient fitting, with different numbers of processes, on a set of 1000 1-dimensional training samples. For comparison, the sequential time is indicated with a horizontal line.

For moderately-sized 1-dimensional datasets, the advantage is principally due to the map structure of the parallel computation, rather than the number of workers. This can be seen in 4.3, which refers to a daset with 1000 training instances. In general, using more processes speeds up the computation, as expected. Though past a certain point, using more processes on a dataset of this size simply introduces more overhead, which roughly balances any increase in efficiency gained from further subdividing the problem. Indeed, the gains past ~ 20 processes are essentially negligible.

The advantage of parallelization on 1-dimensional datasets with a broader range of sizes is clearly reflected in 4.4. Indeed, the speedup factor, i.e. the ratio of the sequential to the parallel runtime, is generally on the order of 10-15 across all measured dataset sizes.

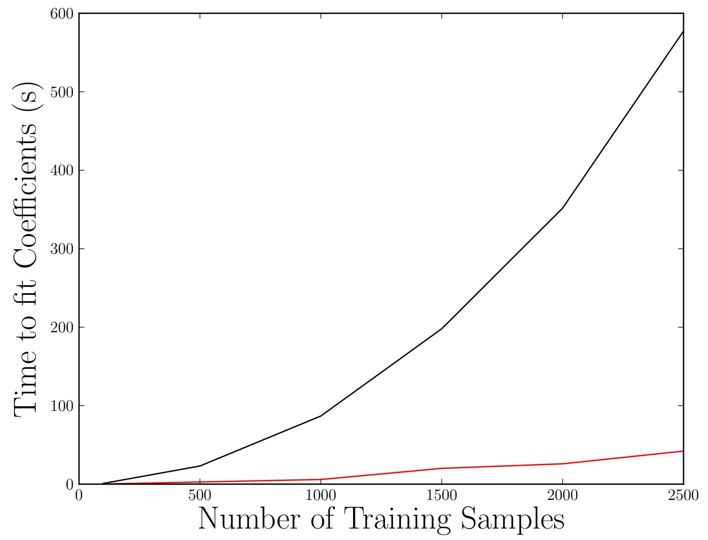


Figure 4.4: Timing curves for parallel coefficient fitting, using a fixed number of processes (20), on 1-dimensional datasets of different sizes. The sequential curve is shown in black; the parallel curve is shown in red.

Note that in-line timing of parallel code tended to give misleading results. Consequently, all these timings were taken with the built in unix **time** command, called on a custom script, to determine the real time taken to perform the fits. The cross-validation step was omitted, and the overhead of acquiring the data was separately timed and subtracted out.

4.3 KNN in Many Dimensions

4.3.1 Overview

As previously mentioned, one way to potentially reduce the regression time is to identify some constant number K of nearest neighbors for a given test instance and include only those in the regression. The trouble with this approach is that KNN becomes inefficient when the search is over high-dimensional spaces. In this case, the dimensionality of the search space is the number of coefficients being used to estimate the densities nonparametrically. The total number of coefficients scales with D^T, where D is the dimensionality of the data and T is the degree of the estimators.

Note that KNN is, in practice, typically considered inadvisable for search space dimensionality larger than ~ 20. However, we will have to consider somewhat higher-dimensional search spaces to keep the exercise nontrivial for $D = 6$, since the only degree that yields fewer than 20 coefficients is $T = 1$, which gives a degenerate estimator (all coefficients are 1). Fortunately, as we will see, in the cosmological application of interest here, similarity along 3 of the axes is far more significant than similarity along the other 3. Therefore, it is not necessarily a waste of time to experiment with KNN in this case.

The aim of this exercise will be to determine, for a given data dimensionality D, the range of parameters for which KNN offers a net gain in efficiency. That is, given some number M of training instances and some constant fraction, e.g. $.10M$, of test instances, when is the total time needed to train on the training instances and regress on the test instances reduced by introducing the extra step of building a KNN-supporting data structure? The boundary is, in general, a function of the dimensionality of the data itself, as well as the accuracy to which it needs to be estimated in order to yield good performance during the test phase.

There are several space-partitioning data structures that support KNN search in many dimensions. These include KD-trees, cover trees, and ball trees. Building a KD-tree using sorted data takes $O(kn \log n)$ time, where k is the dimensionality of the data, and retrieval of a constant number of nearest neighbors can then be done in $O(\log n)$ time. In principle, a cover tree can be constructed asymptotically more quickly, in time $O(c^6 \log n)$, where c is the expansion constant of the dataset. The trouble is that c depends on the dimensionality of the data, so this advantage disappears for high-dimensional data. Likewise retrieval of a nearest neighbor, which takes $O(\eta \log n)$ time, depends on the dimensionality of the data through a constant η; therefore, cover trees are, in general, less suitable than KD-trees for high-dimensional applications.

In contrast, a ball tree is a more viable alternative to a KD-tree, especially when the data is high-dimensional. Its construction can be somewhat slower than that of a KD-tree, but retrieval times are more robust to dimensionality. That is, as the dimension D of the data grows, the average retrieval time of a nearest neighbor from a ball tree continues to scale with $O(D \log n)$. For a KD-tree, this only holds for small D; for large enough D, the retrieval time approaches $O(Dn)$ and may even become slower than brute-force. For these reasons, I've opted to work with ball trees while exploring KNN approaches to regression.

4.3.2 Ball Tree Experiments

The basic structure of these experiments will be as follows: first, I will consider some dimension from 1 to 6. I will pick a fixed number of training samples M large enough to constitute a meaningful problem. Then, I will consider varying values of the estimator degree T, keeping in mind considerations of $\hat{J}$ described previously, up through some value that causes the number of coefficients to exceed 20, i.e. such that the efficiency of ball tree operations is compromised. For each value of T, I will train on M samples and regress on a test set of size $.10M$, either by building a ball tree and computing KNN for a small, fixed value of K, or by regressing using all of the training data. I will measure the time required to build the tree and regress with KNN and compare it to the time needed to regress using all the data. Thus in each case I will determine a range of parameters for which it makes sense to use KNN rather than complete regression.

Let's begin in 1 dimension, with 900 training samples and 100 test samples; it's clear in light of previous experiments that the problem is well-defined at this scale. In this case, the number of coefficients exactly equals T. Although I've shown that you don't really need more than 15 coefficients for this value of M, I'll consider T up to 20 in order to better represent the performance of the tree. Furthermore, as previously demonstrated, the KNN error is essentially negligible for K $\geq$ 5, so any improvements in efficiency do not sacrifice accuracy.

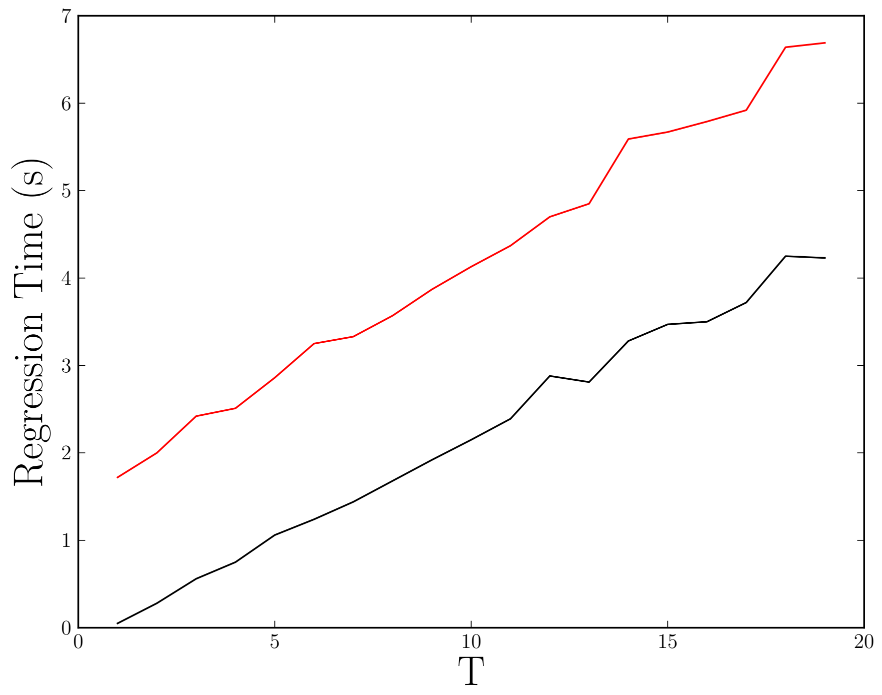


Figure 4.5: Timing curves for regression upon 100 1-dimensional training instances with 100 samples each. Full regression shown in red; KNN regression with 5 nearest neighbors shown in black.

The 1-dimensional results are shown in Figure 4.5. For simplicity, all the calculations have been done sequentially. Here it's evident that KNN offers a significant advantage during regression, as the runtime is decreased by a factor of approximately 2 across all values of T. The advantage ratio decreases slightly with T, but the absolute difference in runtime actually increases somewhat.

Now consider the same test in 2 dimensions, again using 900 training samples and

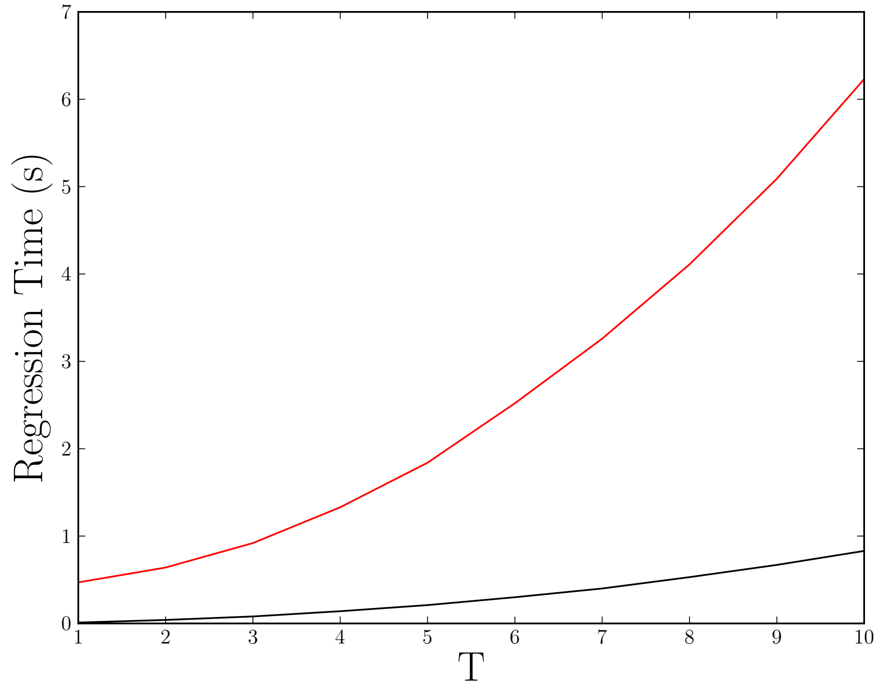


Figure 4.6: Timing curves for regression upon 100 2-dimensional training instances with 100 samples each. Full regression shown in red; KNN regression with 10 nearest neighbors shown in black.

100 test samples. Here we suppose that the number of nearest neighbors might need to be moderately larger, so let K be 10. We also consider a smaller range of T values, up to and including 10; note that the actual number of coefficients is as high as $2^10 = 100$.The resulting timing curve is shown in Figure 4.6, where the advantage of the KNN approach is clear. Indeed, the advantage is retained, though somewhat less, even for unreasonably large values of K (not shown).

In general, these experiments validate the use of a KNN approach, even for multi-dimensional datasets. However, it's worth keeping in mind that for an arbitrary application, the relative efficiency of KNN vs. full regression may still depend on the parameter ranges involved.

4.4 Preprocessing and Binning High-Dimensional Data

In order to apply the methodologies described to real data, a certain amount of preprocessing is required. The simplest requirement is that the data lie in the unit hypercube. In general, this can be accomplished simply by scanning once through the entire dataset, computing the ranges of the various components, and normalizing appropriately. In the specific case of 6-dimensional N-body simulation data, the ranges of the position variables will prove to be more meaningful than the velocity ranges, since, unlike the velocities, they are set by the box size of the simulation itself. Indeed, when using only the positions to cluster particles into training samples, it is only necessary to normalize the positions.

Similarly, binning the particles is more physically meaningful when the bins are determined by spatial rather than velocity information. For this reason, binning will be performed only along the positional axes. For a given bin size b (scaled to lie between 0 and 1), the binning defines $(\frac{X}{b})(\frac{Y}{b})(\frac{Z}{b})$ cubes, where X, Y, and Z are the ranges of the three spatial dimensions in the simulation box. Each cube claims some number of particles and constitutes a single training instance. Given any such cube in an approximate simulation, i.e. a nonparametric estimation of its density, the goal is to compute the density (in all 6 dimensions) of the corresponding cube in the exact simulation.

Once the normalization has been applied to the spatial dimensions, the binning logic is actually rather simple, and it requires only a single pass through the data. The ranges that define the boundaries of each spatial cube are well-defined, i.e. each bin may be identified by a 3-dimensional index. So while scanning through the data, each particle may be assigned in memory to its bin. Once this process is complete, this bins may be processed immediately or written to disc in any order, since there is no ordering imposed on training data. Note that for n_b bins (each of size b), this procedure takes time $O(n_b DN)$ on N particles, where D is the number of dimensions being considered in the binning.

Although n_b and D may be regarded as constants, such that the total preprocessing time is linear in the number of particles, it could nontheless become inefficient for high D or, especially, if n_b is made to be too large. Reducing the size of the bins does tend to reduce their resolution, however, since it reduces η, the number of samples per training instance. This offers an additional motivation to keep the number of bins from becoming too high. On the other hand, if the number of bins is too low, the number of training instances M becomes low, which either reduces the accuracy of the regression or demands that more computationally intensive simulations be run to completion in order to provide additional training data.

It seems that a reasonable strategy, in general, is to choose a bin size that maintains a balance between M and η. This may be estimated by supposing that the particles are distributed in a roughly even fashion across the simulation. For example, let there be N particles to be binned across 3 dimensions, as is the case with the N-body data. A bin size n_b defines n_b^3 bins (M), each of which is expected to hold about $\frac{N}{{n_b}^3}$ particles (η). So a good guess for n_b is that which sets $M = \eta$, i.e. ${n_b}^6 = N$, or $n_b = \sqrt[6]{N}$. More generally, of course, this reasoning suggests using $n_b = \sqrt[2D]{N}$ to bin N particles across D dimensions. Indeed, this is the default binning rule in my code, although the bin may also be chosen by hand in case there is some motivation to do so.

In practice, the plausibility of this binning scheme for a given dataset may be judged by looking at the resulting distribution of bin densities. If the distribution is roughly uniform, i.e. if most of the bins have about the average number of samples, then the binning reasoning applies. If the distribution is highly nonuniform, however, it may be necessary to use smaller bins, or possibly an adaptive scheme that uses smaller bins in denser regions of the data.

Chapter 5

Scaled Application: Dark Matter N-body Simulations

5.1 A Brief Introduction to Cosmology

The ΛCDM model is the simplest parametrization of the Big Bang cosmological model that is broadly consistent with observation. The name of the theory indicates its two principal components: Λ refers to a cosmological constant, i.e. the energy density or **Dark Energy** of the vacuum, and CDM stands for **Cold Dark Matter** (to be explained in further detail shortly). The model also posits an inflationary epoch, a period of extremely rapid universal expansion that took place shortly after the Big Bang. ΛCDM has successfully predicted a wide variety of observations related to the cosmological background radiation, large scale structure, gravitational lensing, and other critical areas of cosmology [8]. As a result, ΛCDM has reached the status of a paradigm, and it is often referred to as the standard model of cosmology.

Dark matter is a hypothetical form of matter proposed to account for a wide variety of cosmological observations that seem to imply the presence of "missing mass." It is called dark because it interacts weakly, if at all, through electromagnetism. Since it does not emit or absorb light like ordinary matter, its distribution must be inferred from its gravitational effects on that matter. There exists a large body of indirect evidence for dark matter drawn from studies of galactic rotation, galactic mergers, gravitational lensing, the cosmic microwave background radiation, baryon acoustic oscillations, the lyman-alpha forest, and other cosmological phenomena (see [3] for an overview). However, to date there has been no accepted direct detection of a dark matter particle.

There are many competing theories for the composition and properties of dark matter. It is generally accepted that most cosmological observations can be most simply explained within ΛCDM by assuming that dark matter is "cold." This means that the dark matter particles move slowly (subrelativistically), or equivalently that their free-streaming length is small relative to the size of a protogalaxy. Generally speaking, it is otherwise difficult to explain key observational evidence, such as the formation of large-scale structure (see e.g. [4]).

The dark matter itself may consist of one or several components with similar properties. Popular candidates include Weakly Interacting Massive Particles (WIMPs) and axions, both of which are hypothetical but whose existence is independently motivated by theory (see again [3], as well as [9] for a additional, less formal discussion of dark matter candidates). In any case, for purposes of modeling and simulating structure formation, the dark matter "particle" is generally treated as a single entity with uniform properties. Note also that the actual particles in a dark matter simulation are typically much more massive than the actual theoretical mass of e.g. a WIMP. The intention of simulation studies is to gain insight into the process of structure formation at some imperfect level of resolution.

5.2 Cosmological N-body Simulations

The N-body problem is, essentially, an unsolved problem in computational physics. The question is: given N “bodies” (or particles) acting under the mutual influence of gravity, what are their trajectories as a function of time? While no exact solution is known, approximate solutions are valuable in a variety of domains of physics, particularly cosmology, since the evolution of many cosmological systems is based in gravitational interaction. In particular, dark matter interacts either exclusively or predominantly under the influence of gravity, and its interactions define the structure of galaxies and other large scale structure.

Computationally, an N-body simulation approximates the evolution of a system of particles under gravity in the following manner: at each timestep, the collective influence on each particle of every other particle is computed; then, their positions and velocities are updated accordingly. This kind of simulation makes it possible to study the formation of non-linear structure under gravity, and, as such, it has wide-ranging applications. Cosmological systems modeled well by N-body simulations include star clusters, galaxy filaments, planetary systems, and environments dominated by dark matter. In particular, the evolution of dark matter-dominated systems has been studied via simulations on universal, halo (galactic), and sub-halo scales.

There exist a variety of techniques to assist in the efficiency of an N-body computation. Most of these concentrate on the task of quickly locating the particles spatially near to a given particle so that only non-negligible forces are calculated. This tends to greatly reduce the number of pairwise interactions that must be computed. Tree-based methods, e.g. those used by a BarnesHut simulation, may be used to distinguish near from far neighbors, thereby limiting the number of explicitly computed pairwise interactions. Alternatively, particle mesh methods may be used to partition a simulation into density bins. The latter option reduces the problem from a computation on particles to a computation on relatively fewer grid points, and it enables the use of highly efficient Fourier transform techniques.

Note that there is also a class of methods to deal with cases in which (Newtonian) gravity is the principal, but not the exclusive, interaction in a physical system. For example, there are ways to compensate for relativistic effects that can arise in comoving coordinate systems. It is also sometimes necessary to introduce nongravitational corrections into N-body simulations. For instance, there is ongoing work to account for e.g. baryonic effects in dark matter-dominated systems. In particular, the effect of baryons may have a nontrivial effect in the formation of galactic structure, especially galactic cores. Such considerations are beyond the scope of this project and are mentioned only for completeness; in this work, the simulation data will be taken from pure N-body simulations of cold dark matter.

5.3 Cosmological Perturbation Theory

An N-body simulation as described above models matter as a collection of discrete particles. Structure is seen to arise as the particles move under the influence of mutual Newtonian gravitational interactions. Cosmological perturbation theory is an alternative model of the same process that offers a different way to think about - and compute - structure formation.

The basic idea is to treat the universe as a Minkowski space-time filled with incompressible hydrodynamic matter. That is, the actual matter content of the universe is regarded as an incompressible fluid having the equation of state $p = 0$. In this model, small inhomogeneities can arise due to thermal fluctuations in the matter. A small, local overdensity will start to attract nearby matter according to Newtonian gravitation; over time, this process eventually gives rise to structure formation.

More formally, the state of the matter is given by its energy density $\rho(\mathbf{x}, t)$. When an inhomogeneity $\delta\rho$ arises at some point in space, it attracts surrounding matter in proportion to $\delta\rho$, in accordance with Newton's second law. That is, in the absence of background expansion,

$$\ddot{\delta\rho} \sim G\delta\rho \tag{5.1}$$

where G is the universal gravitational constant.

Consider first the (unrealistic) case in which perturbative fluctuations evolve in a non-expanding space-time. Matter can be regarded to be a perfect fluid defined by its energy density ρ, pressure p, fluid velocity $\mathbf{v}$, and entropy density S. Gravity is provided by a Newtonian gravitational well ϕ. The system is governed by the following hydrodynamical equations:

$$\dot{\rho} + \nabla_p \cdot (\rho\mathbf{v}) = 0 \tag{5.2}$$

$$\dot{\mathbf{v}} + (\mathbf{v} \cdot \nabla_p)\mathbf{v} + \frac{1}{\rho}\nabla_p p + \nabla_p \phi = 0 \tag{5.3}$$

$$\nabla_p^2 \phi = 4\pi G\rho \tag{5.4}$$

$$\dot{S} + (\mathbf{v} \cdot \nabla_p)S = 0 \tag{5.5}$$

$$p = p(\rho, S) \tag{5.6}$$

These are the continuity equation, the Euler force equation, the Poisson equation of Newtonian gravity, entropy conservation, and the equation of state of matter, respectively. Note that comoving coordinates are implied unless otherwise indicated with a subscript p, which refers to physical coordinates. All derivatives are taken with respect to time.

The background values of these variables are the background energy density ρ_0, the background pressure p_0, a vanishing (0) background velocity, a constant gravitational potential ϕ_0, and a constant entropy density S_0. The response of the system to small perturbations may be ascertained by adding a separate perturbation to the background value of each variable, plugging them into the hydrodynamic equations, and neglecting higher-order terms. That is, let:

$$\rho = \rho_0 + \delta\rho \tag{5.7}$$

$$\mathbf{v} = \delta\mathbf{v} \tag{5.8}$$

$$p = p_0 + \delta p \tag{5.9}$$

$$\phi = \phi_0 + \delta\phi \tag{5.10}$$

$$S = S_0 + \delta S \tag{5.11}$$

Putting these variables into the hydronamic equations, linearizing, and combining the resulting first-order equations yields differential equations for the energy density fluctuation $\delta\rho$ and the entropy perturbation δS:

$$\ddot{\delta\rho} - c_s^2\nabla_p^2\delta\rho - 4\pi G\rho_0\delta\rho = \sigma\nabla_p^2\delta S \tag{5.12}$$

$$\dot{\delta S} = 0 \tag{5.13}$$

Here, c_s refers to the speed of sound in the fluid: $c_s^2 = (\frac{\delta p}{\delta\rho})_{|s}$. Together with σ, it describes the equation of state:

$$\delta p = c_s^2\delta\rho + \sigma\delta S \tag{5.14}$$

The analysis continues by noting that since the equations are linear, it is possible to work in Fourier space; most notably, each Fourier component $\delta\rho_k(t)$ of the fluctuation field $\delta\rho(\mathbf{x}, t) = \int e^{i\mathbf{k}\cdot\mathbf{x}}\delta\rho_k(t)$ evolves independently. This treatment can be expanded to account for the case in which space-time is expanding, i.e. when the fluid has a background velocity

$$\mathbf{v}_0 = H(t)x = \frac{\dot{a}}{a}x \tag{5.15}$$

corresponding to the Hubble flow, where the variable a refers to the fact that physical distances are scaled as $\mathbf{r} = a(t)\mathbf{x}$. The calculation may also be done more generally for a relativistic system. I refer the interested reader to pedagical overviews available on the web, especially [10] and [11], from which this explanation is partially derived. Note that simulations typically use linear or second-order Newtonian perturbative theory with expansion.

5.4 Dark Matter Datasets

5.4.1 Properties and Parameters

The key attributes of an N-body simulation are the physical length L of the simulation in each dimension and the number of particles N it resolves. A large-scale cosmological N-body simulation also necessarily assumes values for certain cosmological parameters. For example, a simulation modeled according ΛCDM must assume values of the matter density (Ω_m), dark energy density (Ω_Λ), baryon density (Ω_b), Hubble constant (H_0), fluctuation amplitude at $8h^1$ Mpc (σ_8), and scalar spectral index (n_s).

The particular N-body simulation used in this work was generated with the parameters given in 5.4.1. The meaning and significance of the various parameters are elaborated on briefly below:

1. Ω_m refers to the total matter density, i.e. both dark matter and baryonic (ordinary, non-dark) matter. Baryonic matter comprises only about 17% of the total matter content of the universe ($\Omega_b = .17\Omega_m$); the rest is dark matter. The total matter density is in turn dominated by the dark energy density Ω_Λ, which contributes almost 3/4ths

L	1000 Mpc/h
N	2^{30}
Ω_m	0.27
Ω_Λ	0.73
Ω_b	0.045
H_0	70 km/s/Mpc
σ_8	0.80
n_s	0.96
w	-1

Table 5.1: Key simulation parameters.

of the total mass-energy density of the universe [2]. That total mass-energy is given by $\Omega_{tot} = \Omega_m + \Omega_{rel} + \Omega_\Lambda$, where Ω_{rel} is a much smaller component consisting of the effective mass density of relativistic particles (i.e. photons and neutrinos). Interestingly, the measured value of Ω_{tot} is very close to the critical density 1 at which the Universe would expand forever [2].

2. Hubble's constant H_0 describes the expansion rate of the universe. It appears in Hubble's law, $v = H_0 D$: H_0 is the constant of proportionality between a galaxy's recession velocity v and its proper distance D.

3. σ_8 gives the amplitude of mass fluctuations, i.e. the anisotropy of the distribution of mass, on a useful scale.

4. A power spectrum $P(k)$ describes the power of primodial mass variations as a function of spatial scale. According to many inflationary models, the scalar component

follows a power law $P_s(k) \propto k^{n_s - 1}$, where k is the wavenumber of the fluctuations, and n_s is referred to as the scalar spectral index.

5. w, which refers to the dark energy equation of state, is defined as the ratio of pressure p that dark energy puts on the universe to the energy per unit volume ρ: $w = p/\rho$. Setting $w = -1$ corresponds to assuming a model with a true, i.e. non-time-varying, cosmological constant.

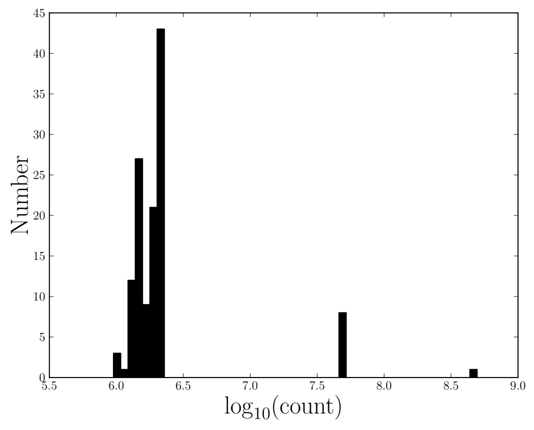
Note also the scale of the simulation: the side length L is 1000 Mpc/h, where a typical intergalactic distance is on the order of 50 Mpc/h; that is, this is a large-scale simulation resolving many galaxies.

5.4.2 Binning Considerations

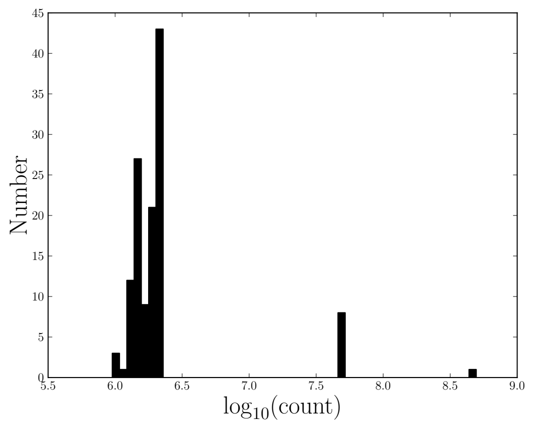
By the binning rule provided in the previous chapter, this dataset offers $\sqrt[6]{(2^{30})} = 32,768$ training instances, with an average of 32,768 particles per bin. These numbers suggest a generous amount of well-defined training data; however, various complications may arise in practice. For example, if the particles are highly clustered, there may be many empty bins and relatively fewer bins containing many particles. In that case it might be difficult to learn the desired function, since the number of informative training samples would be lower than expected.

Another potential problem is a misalignment between structure in the approximate vs. the true simulation. If the spatial offset between corresponding structures is too great, then the same structure (for example, a galaxy or filament) might fall into different bins. If corresponding structures are not grouped together in the training data, the algorithm cannot reasonably be expected to learn the desired mapping.

In order to guard against potential problems of this nature, I think it is informative



(a) N-body dataset



(b) Perturbative dataset

Figure 5.1: Distribution of particle counts per bin when the simulations are partitioned spatially into 125 equally-sized cubes.

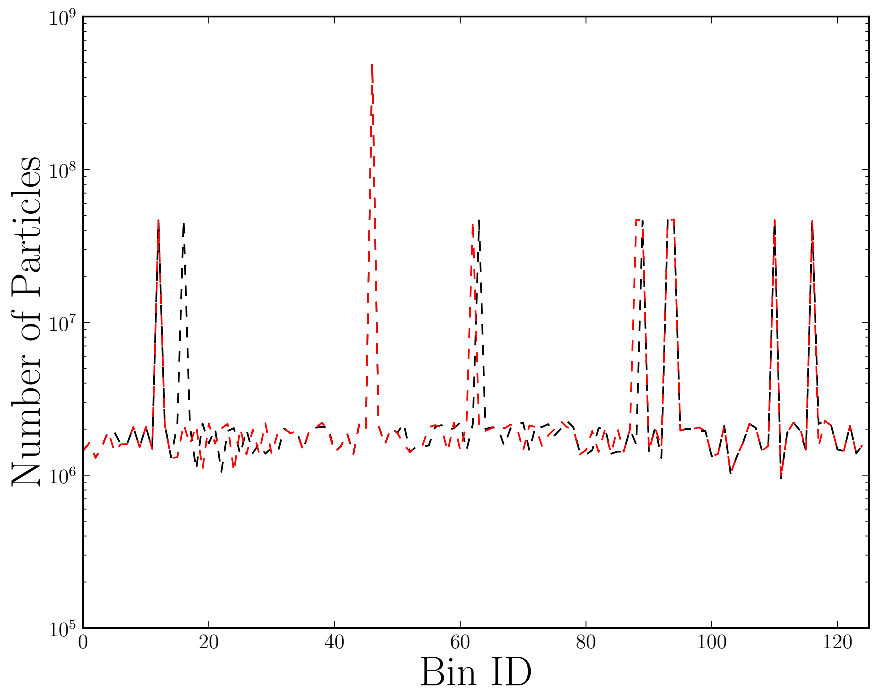


Figure 5.2: Particle counts for corresponding bins in the exact (black) and approximate (red) datasets.

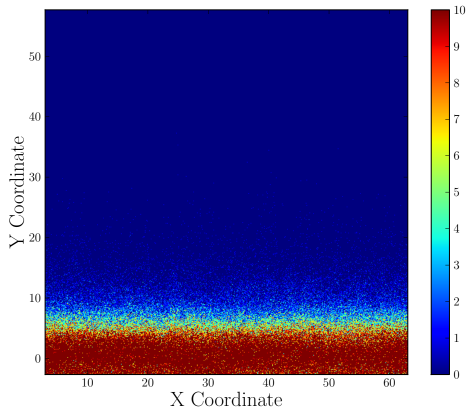
to bin all the available training data, at least on a coarse scale. For this application I have defined 5 divisions along each spatial dimension, for a total of 125 3D bins, and counted the number of particles that fall into each bin for both the approximate and the true simulation. The results, shown in Fig. 5.4.2 and 5.2, show that at least on this large scale, the density profiles of the simulations are quite similar. Furthermore, the majority of the bins have a similar number of particles, which indicates that the particle distribution is roughly even throughout the simulation. These features of the data are encouraging, although it remains to examine the simulation more closely on a smaller scale, as I will do in the next section.

5.4.3 A Closer Look

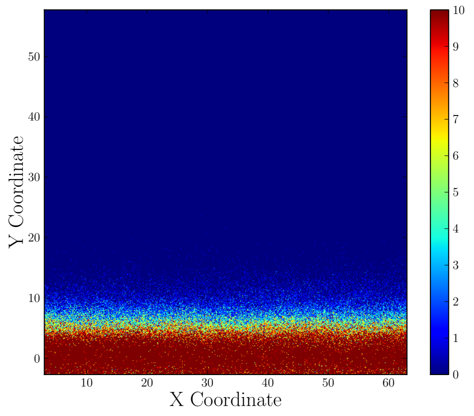
It is also informative to isolate a smaller, contiguous region of the simulation and examine it more closely in both the approximate and exact datasets. For this exercise I divided each spatial axis into 17 pieces, for a total of 4,913 3D bins, and selected a bin with relatively abundant structure. In the exact simulation, this bin contains 9,024,511 particles; the corresponding bin in the approximate simulation contains 9,167,451 particles. The side lengths of the cube are large enough to accommodate a large amount of structure; however, the number of particles is a factor of ~ 100 smaller than the entire simulation, which makes more detailed calculations possible within a reasonable amount of time.

One way to visualize this data is to examine "slices" defined by further binning the data along some axis. In Fig. 5.3, I show the top and bottom slices given by subdividing the data into 100 bins along the z-axis. The basic resemblance between the approximate and the exact data is clear, although some differences are also evident. In particular, the N-body data tends to show small substructure not evident in the approximate data.

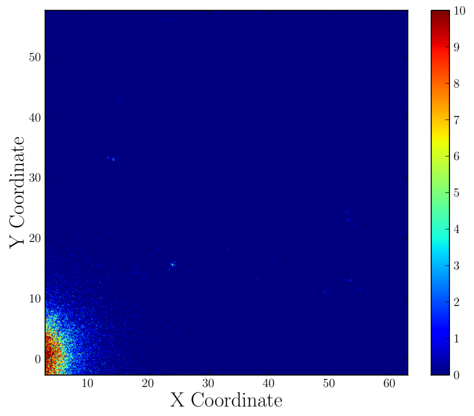
This effect is perhaps more evident in a scatter plot, as in Fig. 5.4. It's also easier



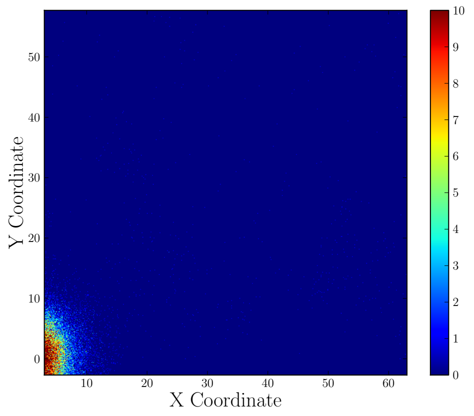
(a) N-body dataset: bottom slice



(b) Perturbative dataset: bottom slice

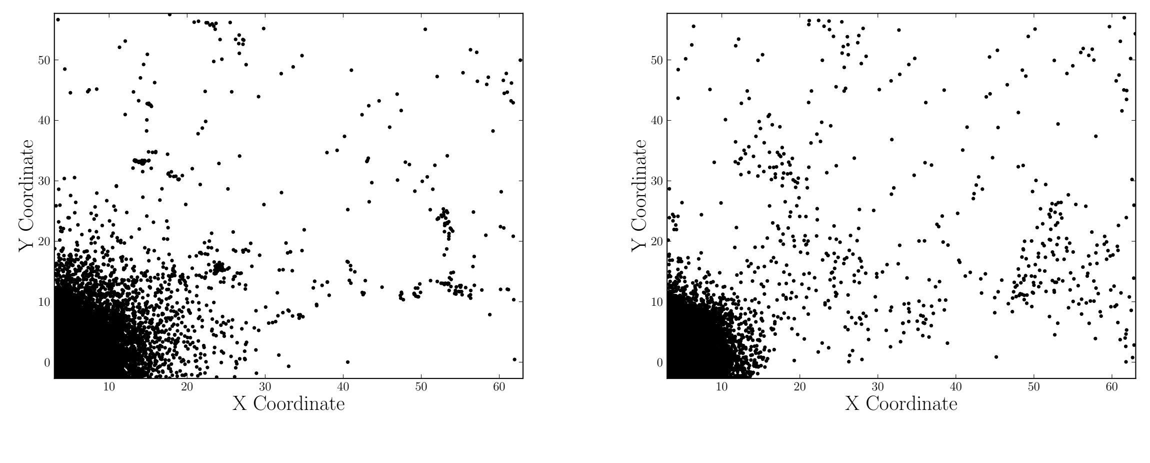


(c) N-body dataset: top slice



(d) Perturbative dataset: top slice

Figure 5.3: Corresponding slices from example bin.

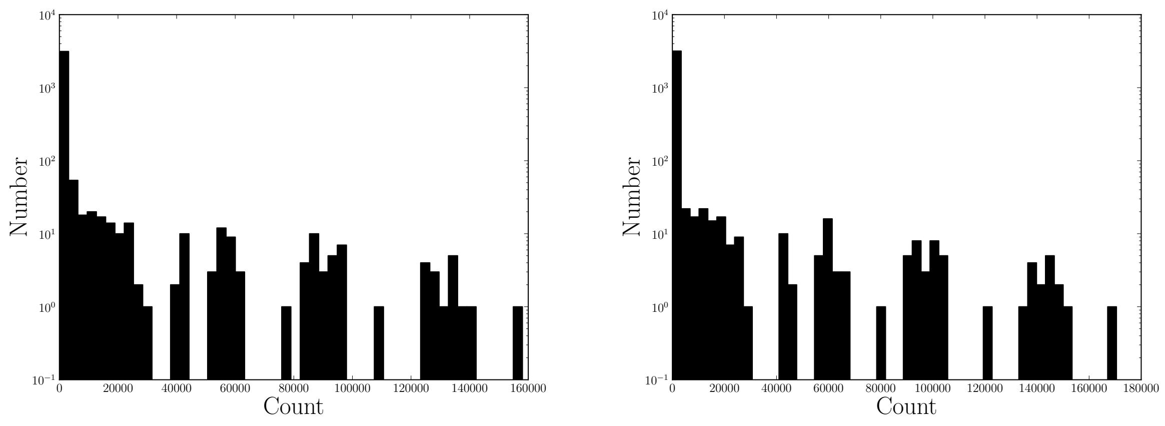


(a) N-body dataset: top slice

(b) Perturbative dataset: top slice

Figure 5.4: Alternate, scatter-plot view of top slice of example bin.

to see that even the low-density regions typically contain some particles. That is, while much of the volume of the data has a relatively low density, it would have to be binned quite finely in order for most of the bins to be entirely empty.



(a) N-body dataset

(b) Perturbative dataset

Figure 5.5: Distribution of particle counts per bin when the example bin is partitioned into 3,375 smaller bins.

It also seems that at this scale, the particles are fairly clustered; that is, there are large regions of low particle density. Indeed, consider binning this data "optimally," i.e. so that the number of bins roughly matches the average number of particles per bin. With 15 divisions per axis, there are 3,375 bins, with an average of about 2674 (exact) and 3341 (approx) particles each. Assigning particles according to this binning scheme gives the density distributions in Fig. 5.5. Note that these are similar, but not identical. Although there are plenty of high-density bins to work with, more bins fall into the lowest-density category than any other. Still, only about 2.7 and 1.4 percent of the bins are truly empty in the exact and approximate datasets, respectively.

5.5 Targeting the DTDR Strategy

There was a previous effort by a machine learning graduate student at Carnegie Mellon to use distribution-to-distribution regression on similar simulation data. There, the approach was somewhat different: the idea was to map the simulations from one point in time to another, and there was no use of approximate simulations. Experiments using one billion-year time intervals were never successful; in fact, the algorithm seemed unable even to learn the identity distribution.

There are various reasons that this strategy might not work well for real simulation data. First of all, the binning is necessarily unphysical, and each bin is considered independently of the others. After a billion years, many particles would tend to cross bin boundaries, yet the algorithm has no way to capture this behavior. Consider also that a regressor trained in this way on all 6 dimensions would tend to map bins with fast-moving particles to emptier bins, since the particles would leave the frame. This kind of effect could lead to widespread violation of mass conservation in the regressor's predictions, even if the algorithm learned the function well.

Independently of these considerations, the problems with the identity distribution point to additional issues. For example, learning this kind of function in 6 dimensions might require prohibitive amounts of training data, or a large number of parametric coefficients may be necessary in order to represent the bin densities in sufficient detail. The bins might have been too small or too large for the algorithm to work well. Furthermore, the billion-year time intervals may have been too large to expect a consistently learnable function from one time slice to the next.

In any case, I expect that even if the identity mapping had been successfully learned, mapping across time slices would ultimately have been difficult for the reasons mentioned above. Consequently, I have tried a different approach, namely mapping from approximate to true distributions. Although this mapping is not a priori guaranteed to be well-defined, it minimizes the problems with movement accross bin boundaries, since the training inputs and outputs are from the same point in time. These considerations point to a more general need to target DTDR to the application at hand, especially when determining how to bin the data and what function to try to learn.

5.6 The Identity Distribution

One key test of the viability of this algorithm is its ability to learn the identity distribution in multiple dimensions. That is, given multi-dimensional data, is it possible to learn the mapping from that data to itself? Clearly this task must be possible if more complex functions are to be learned. Indeed, the function that maps approximate to exact N-body simulations, if it exists, should not be too different from the identity function.

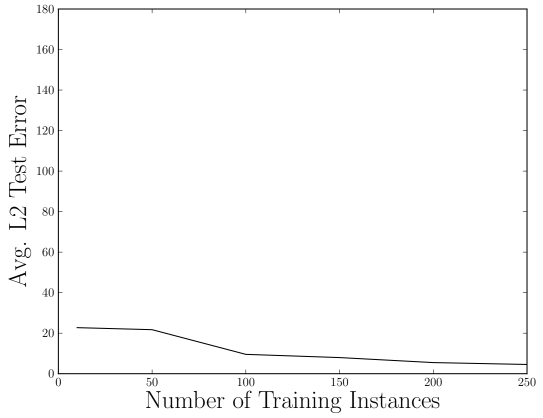
It's evident that the algorithm can learn the identity function in 1 dimension, since it's able to learn the reflection function. On the other hand, $\hat{J}$ tests indicate that a large number of coefficients are already needed to model 3-dimensional data, let alone higher-dimensional data, to nontrivial accuracy. Also, as I will argue in the next section, the 6-dimensional simulation data nonetheless lends itself naturally to a 3-dimensional treatment. Consequently, I concentrate my efforts on 3-dimensional data.

I begin by considering a cube-shaped subset from an exact (N-body) simulation. I partition it into cubes of length ~ 1 Mpc/h along each side, and I retain those bins that contain at least 5 particles. This yields a set of ~ 300 training instances, averaging ~ 70 particles each. Here, the test sets are 10% as long as the train sets. As shown in Figure 5.6, at this scale, the average test error decreases with the number of training samples, as expected. Note that for this data size, increasing the value of T past 3 causes overfitting and decreased performance, i.e. larger and more erratic errors, though for $T < 3$ (not shown), the distributions are not well modeled.

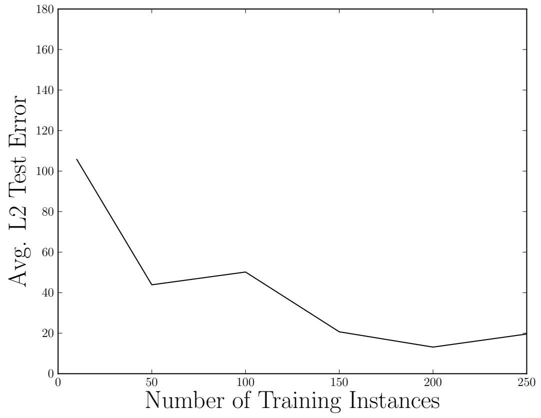
However, for the experiments in 5.7, the drop in the error is so sudden that it cannot obviously be attributed to increasing the size of the training set. And in the largest experiments in 5.8, the correlation between training set size and test error is even less clear. This may be for any number of reasons: for example, the number of training samples and the number of particles per sample may simply be insufficient, especially for the larger experiments. Since the computation already takes several hours at the high end of this scale, this would suggest that the method is too slow to be useful for this data. Alternatively, T may be too small for the larger experiments, although again, increasing T would increase the computation time. Another possibility is that this data set does not present sufficient variety in order for the identity function to be learned. This hypothesis could be tested by generating artificial data in many dimensions in order to test the minimum quality and quantity of data needed to learn the identity function.

5.7 Regression

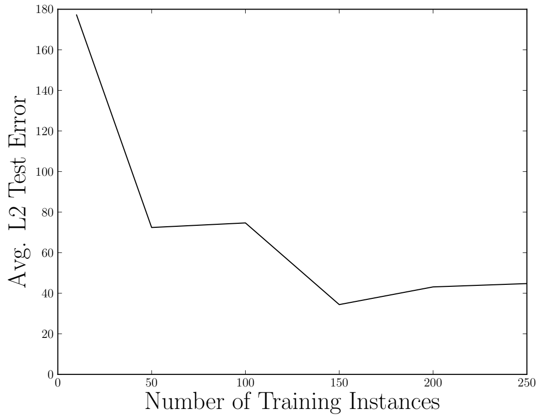
Recall the key features of the simulation datasets suggesting that regression might be possible between them: first, the distribution of particles is fairly even across 3D position bins for both the N-body (Figure 5.4.3) and perturbative (Figure 5.4.2) datasets. This partially justifies the use of a symmetric i.e. non-adaptive binning scheme.



(a) $T = 3$

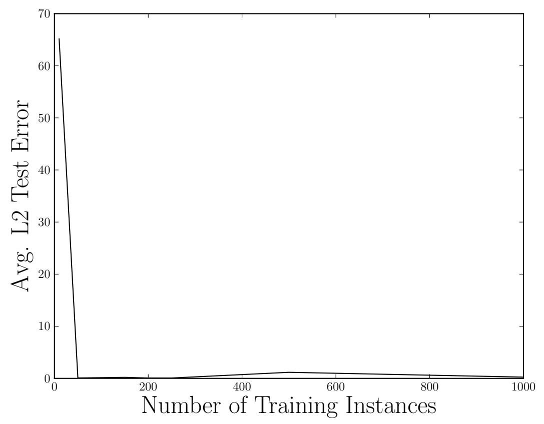


(b) $T = 4$

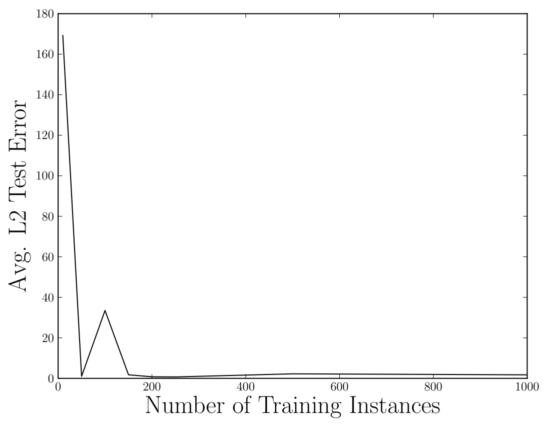


(c) $T = 5$

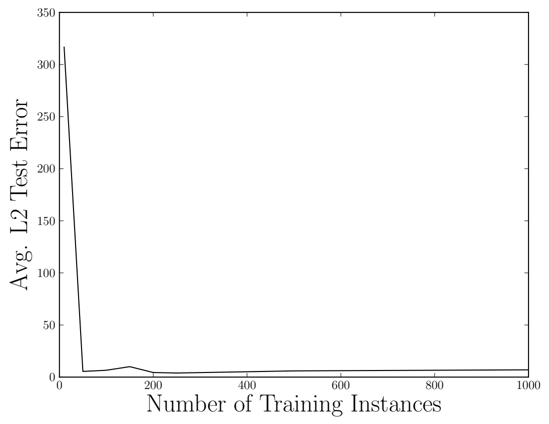
Figure 5.6: Average test error as a function of training data size for 3-dimensional positional data from an exact (N-body) dark matter simulation. Average particles per training instance: ~ 70; Minimum particles per training instance: 5



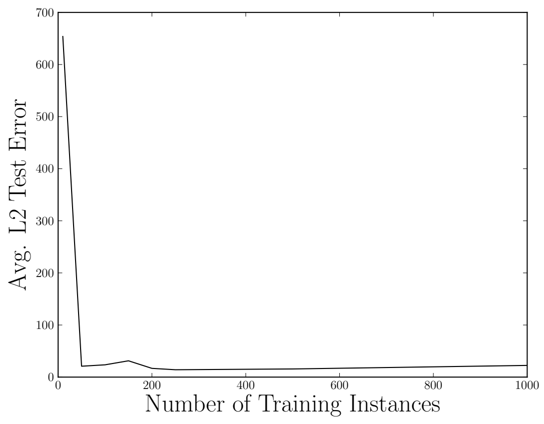
(a) $T = 3$



(b) $T = 4$

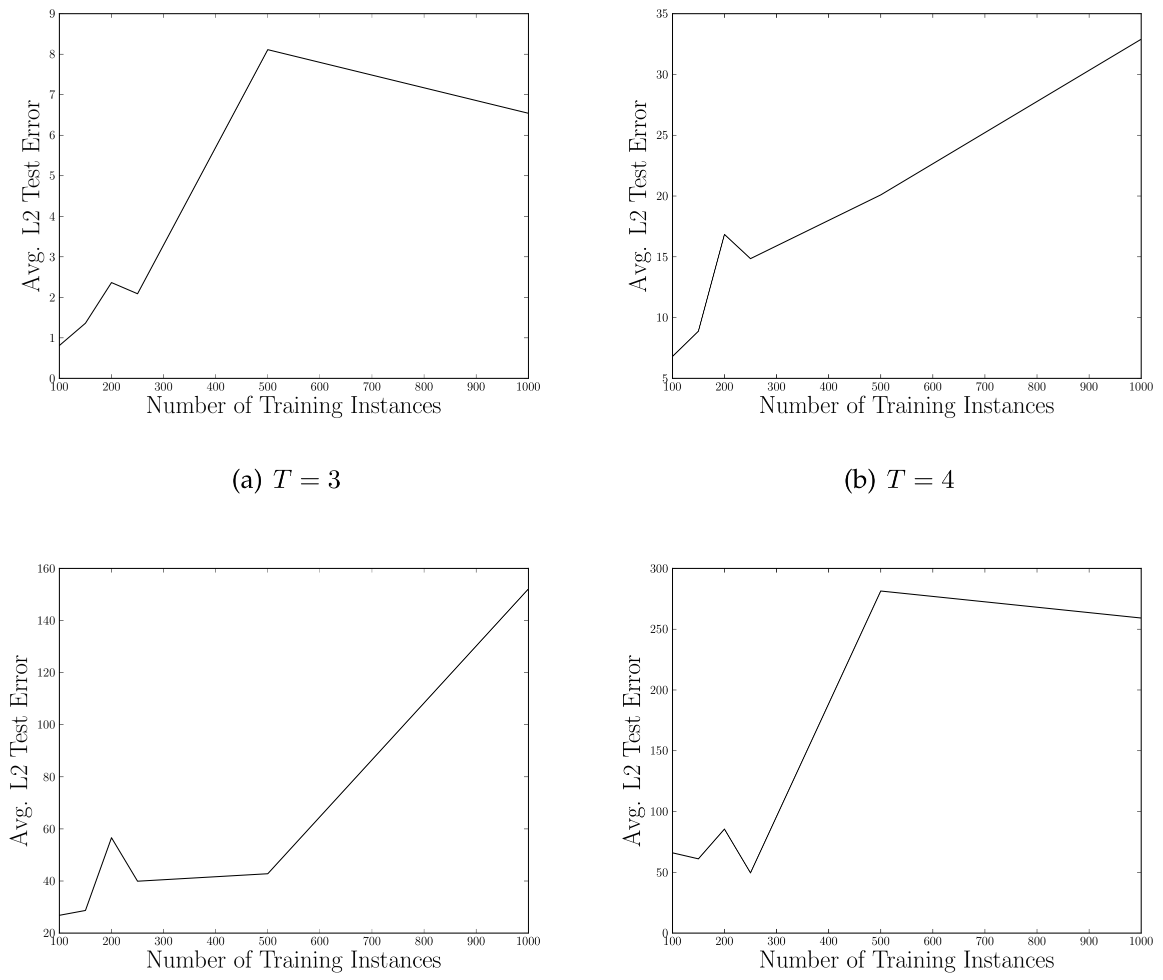


(c) $T = 5$



(d) $T = 6$

Figure 5.7: Average test error as a function of training data size for 3-dimensional positional data from an exact (N-body) dark matter simulation. Average particles per training instance: ~ 1500; Minimum particles per training instance: 50



(a) $T = 3$

(b) $T = 4$

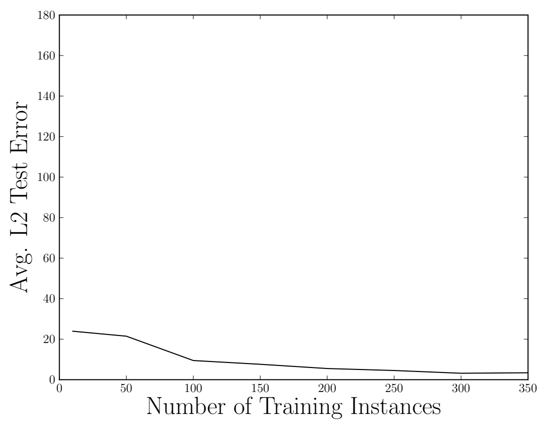
(c) $T = 5$

(d) $T = 6$

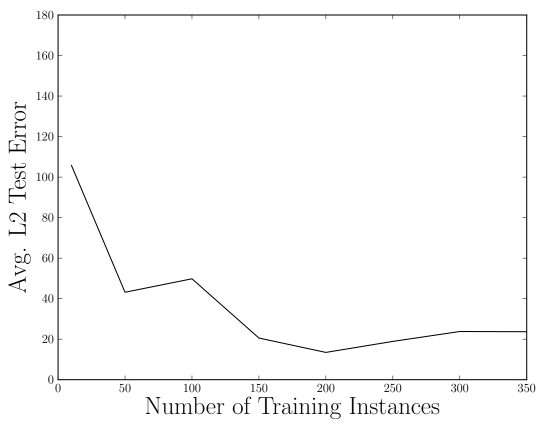
Figure 5.8: Average test error as a function of training data size for 3-dimensional positional data from an exact (N-body) dark matter simulation. Average particles per training instance: ~ 5000; Minimum particles per training instance: 50

Furthermore, corresponding bins in the same datasets tend to have similar particle counts, as shown in Figure 5.2. This suggests that there is minimal shifting of particles across bin boundaries from the input to the output data. In other words, the boundary effects that might result from trying to map simulations from one point in time to another should not arise here, because there is little interaction between adjacent bins, which further justifies treating them independently of one another.

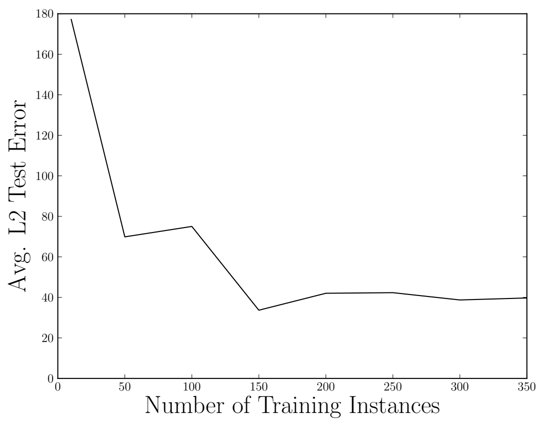
However, in the absence of conclusive results in the identity experiments, it seems unlikely that regression should perform well on this data. Indeed, regressing from the approximate to the N-body data, using the same bins as in the identity experiments, yields similar though non-identical results (see Figures 5.9, 5.10, 5.11). Note that here, the error is being computed only for the positional dimensions, i.e. the dimensions involved in the binning. However, it would be interesting to learn whether position-defined bins could also predict velocity information.



(a) $T = 3$

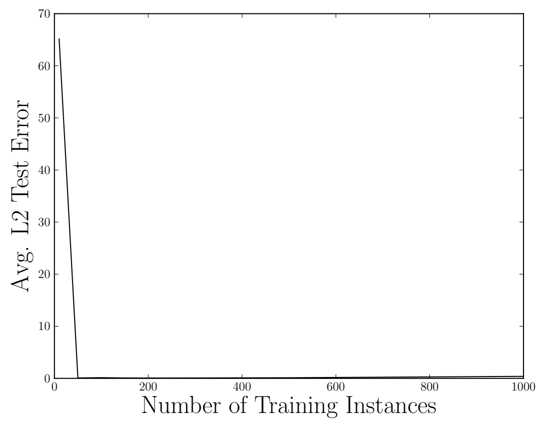


(b) $T = 4$

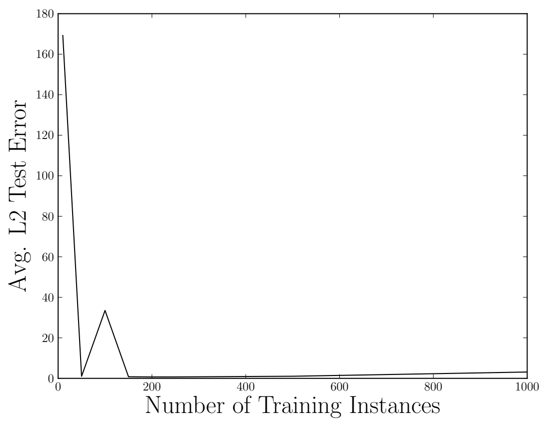


(c) $T = 5$

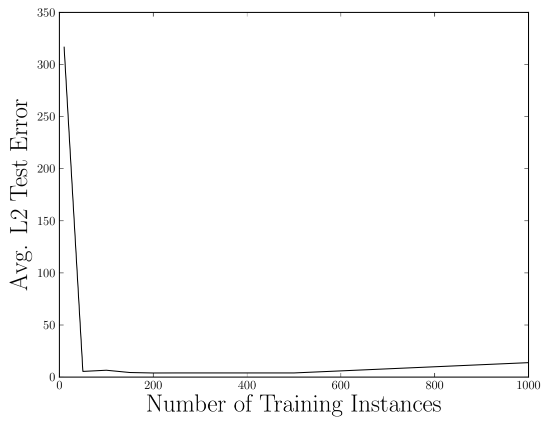
Figure 5.9: Average test error as a function of training data size when mapping from perturbative to N-body data. Average particles per training instance: ~ 70; Minimum particles per training instance: 5



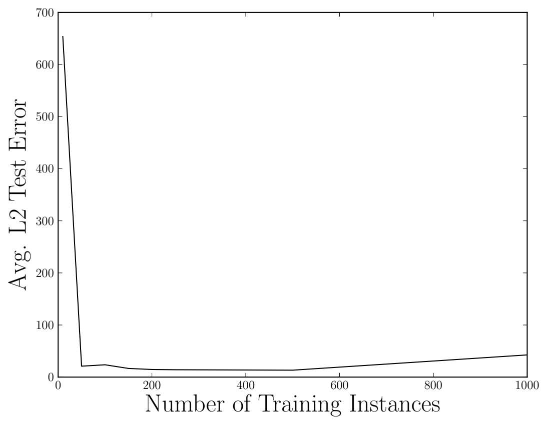
(a) $T = 3$



(b) $T = 4$

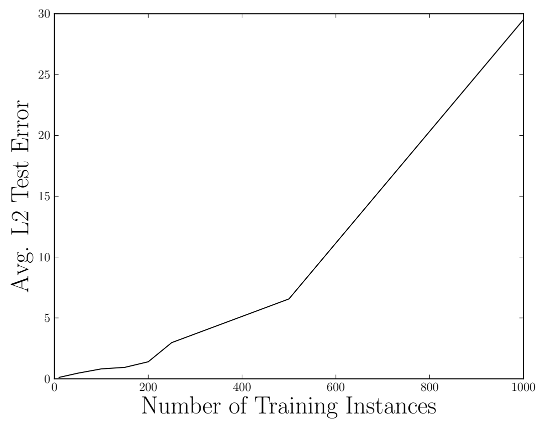


(c) $T = 5$

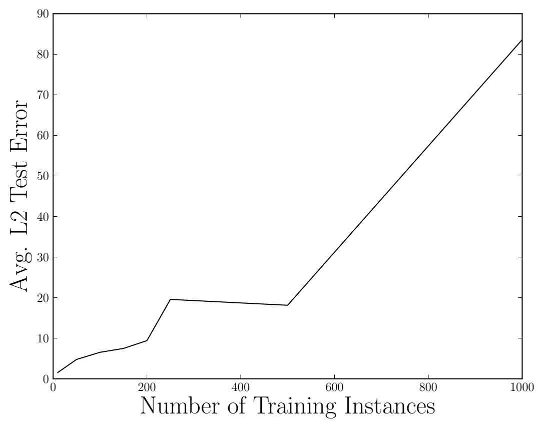


(d) $T = 6$

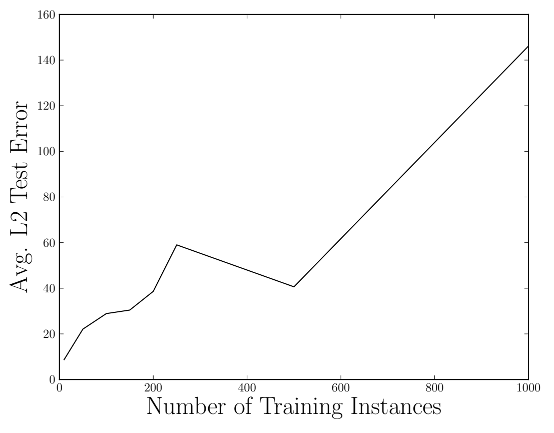
Figure 5.10: Average test error as a function of training data size when mapping from perturbative to N-body data. Average particles per training instance: ~ 1500; Minimum particles per training instance: 50



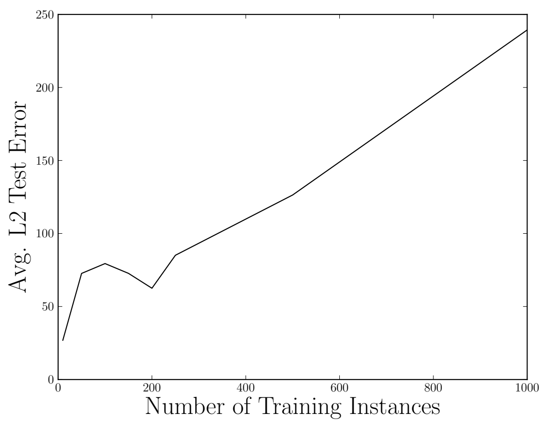
(a) $T = 3$



(b) $T = 4$



(c) $T = 5$



(d) $T = 6$

Figure 5.11: Average test error as a function of training data size when mapping from perturbative to N-body data. Average particles per training instance: ~ 5000; Minimum particles per training instance: 50

Chapter 6

Conclusions

Distribution-to-distribution regression has been proven effective on low-dimensional data, and scaling efforts have brought its use on higher-dimensional data solidly into the realm of possibility. However, the initial application results are fundamentally inconclusive and call for additional large-scale experiments. Further work with multi-dimensional artificial data should be helpful in establishing baseline performance and training requirements for the scaled algorithm. There are also a number of modifications that might be helpful for simulation-style datasets, such as adaptive binning schemes that might better capture large-scale structure, or an alternative metric of the algorithm's performance; for instance, a Hessian matrix might represent the similarity of two simulation datasets more meaningfully than the L_2 norm. In any case, it remains to be seen whether this kind of method can be used to make the generation of large simulation datasets practical.

Appendix A

Python Implementation of DTDR

A.1 Overview

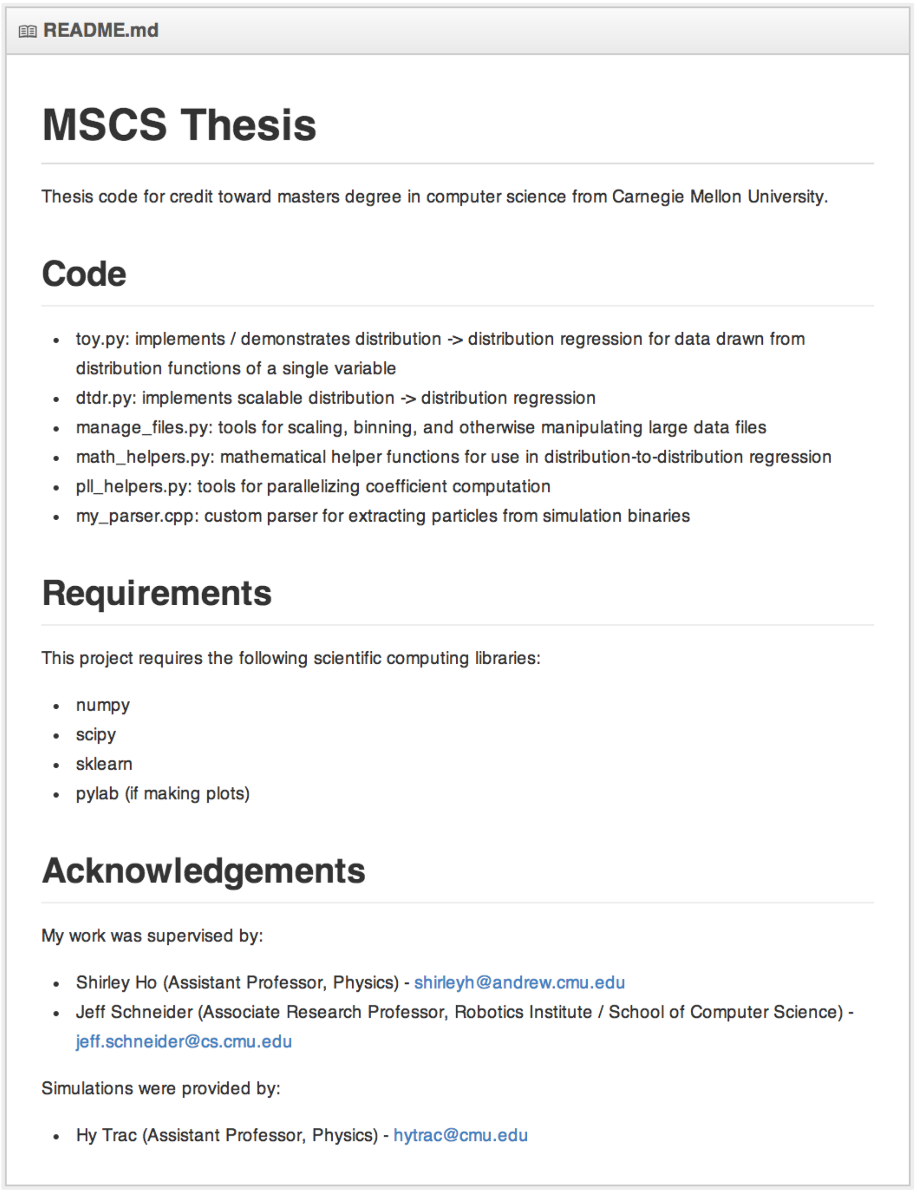
The complete source code for this project is available online at `https://github.com/alklein/MSCS_Thesis`. Note that the full simulation files are large and are therefore not included, so to run the exact experiments described in this thesis, it is necessary to request those files. The principal DTDR tools are not specific to those files, however, and may also be applied to new data.

The key components of the code are described in its README (Figure A.1). For further details on the individual libraries and scripts, please refer to the headers and comments in the files themselves.

A.2 Availability on AutonLab

The complete project, including simulation files, is also available on CMU's Auton cluster. Members of the CMU community with Auton access can find the project in my home directory on lov4. The steps for access, for me, are as follows:

1. `ssh andreakl@lop1.autonlab.org`


README.md

MSCS Thesis

Thesis code for credit toward masters degree in computer science from Carnegie Mellon University.

Code

- toy.py: implements / demonstrates distribution -> distribution regression for data drawn from distribution functions of a single variable
- dtdr.py: implements scalable distribution -> distribution regression
- manage_files.py: tools for scaling, binning, and otherwise manipulating large data files
- math_helpers.py: mathematical helper functions for use in distribution-to-distribution regression
- pll_helpers.py: tools for parallelizing coefficient computation
- my_parser.cpp: custom parser for extracting particles from simulation binaries

Requirements

This project requires the following scientific computing libraries:

- numpy
- scipy
- sklearn
- pylab (if making plots)

Acknowledgements

My work was supervised by:

- Shirley Ho (Assistant Professor, Physics) - shirleyh@andrew.cmu.edu
- Jeff Schneider (Associate Research Professor, Robotics Institute / School of Computer Science) - jeff.schneider@cs.cmu.edu

Simulations were provided by:

- Hy Trac (Assistant Professor, Physics) - hytrac@cmu.edu

Figure A.1: README.md

2. `ssh lov4`

3. `cd /home/scratch/andreakl`

Note that as of Dec. 2013, Auton does not support graphics, so all imports to pylab should be commented out, and no attempt should be made to generate plots.

In order to make use of the scientific computing libraries, it is also necessary to use the python version specified at the head of the scripts. For this reason, permissions must be set to execute each script. E.g., for a script called script.py, you must first type:

```
chmod +x script.py
```

Then you can execute the script directly:

```
./script.py
```

Bibliography

[1] Dougal J. Sutherland Jeff Schneider Barnabas Poczos, Liang Xiong. Nonparametric kernel estimators for image classification. In *CVPR 2012*, 2012. 1

[2] C.L. Bennett, D. Larson, J.L. Weiland, N. Jarosik, G. Hinshaw, et al. Nine-Year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Final Maps and Results. 2012. 5.4.1

[3] G. Bertone, D. Hooper, and J. Silk. Particle Dark Matter: Evidence, Candidates and Constraints. *Phys.Rept.405:279-390,2005*, August 2004. 5.1

[4] G. R. Blumenthal, S. M. Faber, J. R. Primack, and M. J. Rees. Formation of galaxies and large-scale structure with cold dark matter. , 311:517–525, October 1984. doi: 10.1038/311517a0. 5.1

[5] M. Crocce and R. Scoccimarro. Renormalized cosmological perturbation theory. , 73(6):063519, March 2006. doi: 10.1103/PhysRevD.73.063519.

[6] Sam Efromovich. Orthogonal series density estimation. *WIREs Computational Statistics*, 2:467–476, 2010. 2

[7] Jeff Schneider Junier Oliva, Barnabas Poczos. 2013. 1, 1, 2, 2, 2, 3.1, 3.1, 3.2

[8] E. Komatsu et al. Seven-Year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Cosmological Interpretation. *Astrophysical Journal Supplement Series*, 2010. 5.1

[9] A. H. G. Peter. Dark Matter: A Brief Review. *ArXiv e-prints*, January 2012. 5.1

[10] Patrick Peter. Cosmological perturbation theory. 2013. 5.3

[11] Dmitry Podolsky. Newtonian perturbation theory. 5.3

[12] Alexandre B. Tsybakov. Introduction to nonparametric estimation. *International Statistical Review*, 2008. 2