Efficient Online Non-Parametric Kernel Density Estimation

Christophe G. Lambert*
Scott E. Harrington†
Campbell R. Harvey‡
Arman Glodjo§

Abstract

Non-parametric density estimation has broad applications in computational finance especially in cases where high frequency data are available. However, the technique is often intractable, given the run times necessary to evaluate a density. We present a new and efficient algorithm based on multipole techniques. Given the n kernels that estimate the density, current methods take $O(n)$ time to directly sum the kernels to perform a single density query. In an on-line algorithm where points are continually added to the density, the cumulative $O(n^2)$ running time for n queries makes it very costly, if not impractical, to compute the density for large n. Our new Multipole-accelerated Online Density Estimation (MODE) algorithm is general in that it can be applied to any kernel (in arbitrary dimensions) that admits a Taylor series expansion. The running time for a density query reduces to $O(\log n)$ or even constant time, depending on the kernel chosen, and hence, the cumulative running time is reduced to $O(n \log n)$ or $O(n)$, respectively. Our results show that the MODE algorithm provides dramatic advantages over the direct approach to density evaluation. For example, we show using a modest computing platform that on-line density updates and queries for one million points and two dimensions take 8 days to compute using the direct approach versus 40 seconds with the MODE approach.

1 Overview and Motivation

The motivating application for the development of the new algorithms in this paper is online density estimation in computational finance, where high frequency data are available, and the data set grows over time. Non-parametric density estimation techniques have been used in finance by Aït-Sahalia and Lo [1] to estimate the state-price density for options, by Boudoukh, Richardson, Stanton, and Whitelaw [7] and Harvey [17] to design hedging strategies for mortgage-backed securities, by Harvey [17] to describe dynamic risk functions and by Pagan and Schwert [22] to describe the evolution of equity market volatility.

Statistics is all about studying distributions of data, or the probability density function (pdf) that the data follow (see Trivedi [32] for background on pdf's). With a finite sample set, we are interested in estimating the density of a random variable. If the form of the density function is known to be say, Gaussian, then it is a simple matter of finding the best fit of a Gaussian to the data, thereby parameterizing the density. However, when the density function is not known a priori, non-parametric density estimation must be employed. This approach "lets the data speak for itself." Rather than imposing assumptions, the non-parametric technique allows us to directly approximate the d-dimensional density that describes how these variables interact. A good reference on density estimation is [29]. One technique for non-parametric density estimation that most people

*Transworld Numerics Limited, #4 Nantucket Lane, Smiths Parish, FL05, Bermuda, lambert@ibl.bm
†Transworld Numerics Limited, #4 Nantucket Lane, Smiths Parish, FL05, Bermuda, scotth@ibl.bm
‡Fuqua School of Business, Duke University, Durham, NC 27708, charvey@mail.duke.edu
§Fuqua School of Business, Duke University, Durham, NC 27708, and Transworld Numerics Limited, #4 Nantucket Lane, Smiths Parish, FL05, Bermuda, arman@ibl.bm

are familiar with is the use of the histogram. The data is divided into rectangular bins of a certain width, and then the height of the bins is based on the number of points that fall in the bin range. By visually observing the histogram we can observe a probability distribution of the data. It is non-parametric in that we are not trying to fit the data to a particular curve or distribution.

This paper deals with a more powerful technique of density estimation called *kernel density estimation.* Instead of summing points within bins to create a pictorial estimate of the distribution, a sum of kernel functions (typically Gaussians) centered at the data points creates the estimate. Note that choosing the Gaussian as a kernel function is *not synonymous* with fitting the distribution to a Gaussian model. The kernel is only used as a weighting function for the points, and the method is most useful for estimating unknown distributions. The *bandwidth* of the kernel function, h, determines how much smoothing is done, and corresponds to how quickly the kernel function decreases and hence how widely it extends away from the datapoint. This bandwidth is analogous to the bin width used with a histogram. Smoothing refers to the process of spreading out the data by weighting it with kernels so that a density estimate can be determined where there are no actual datapoints. In general more points allow a narrower bandwidth and a better density estimate. This technique was introduced by Rosenblatt [26] and Parzen [23], and is sometimes called Parzen density estimation. This approach is used for more than just plotting a density estimate. It gives a differentiable function, and conditional expectations can be estimated from the density *without knowing the distribution.* In the limit of an infinite sample size, the density estimate approaches that of the underlying distribution.

Non-parametric kernel density estimation is being used more widely as faster computers and computational procedures become available. Various methods have been developed to make the process of non-parametric kernel density estimation more efficient. Binning techniques, sometimes coupled with the Fast Fourier Transform, are reasonably efficient in low dimensions [16, 28, 29, 34]. However, they are restricted to uniform grids, and typically, the density is computed over every grid point which becomes very large as the dimension is increased, or loss of accuracy results if the grid is made too coarse. Various data-reduction techniques are also used to speed up the computations. The k-th nearest neighbor techniques of [11] and the clustering based branch and bound technique [18] exclude the contribution of distant kernel functions whose contribution is negligible. These methods perform better in higher dimensions, but the error that must be introduced to obtain a computational savings depends on the bandwidth being narrow, or of finite extent such as for the Epanechnikov kernel. For broad bandwidths, nearly every point may contribute significantly to the density, and then little savings is realized as every point's contribution needs to be computed. Many of the existing fast methods do not perform as well with the Gaussian kernel, which is usually the kernel of choice if one can afford to compute it.

While the non-parametric approach is very appealing, it is not without disadvantages. The rate of convergence is slower than parametric models. For density estimation to be reliable, very large data sets are required. Hence, one of the most promising applications of these methods focuses on trading applications with high frequency data. However, the estimation of the density at a single point is an order-n problem. The cumulative time of online density queries for n points thus takes $O(n^2)$ time. This can make real-time updating of the multivariate density prohibitively expensive even for data sets of moderate size.

Our contribution is to propose and implement an algorithm for online updating of the multivariate density that can both compute the density at any point, as well as add a new observation to the density in $O(\log n)$ or even constant time. The Gaussian kernel is used as a case study, and we go on to show how to extend the technique to a wide class of kernels in arbitrary dimensions. In addition, we will discuss algorithm modifications to determine conditional expectations from the density function.

We begin by giving an overview of the framework for multipole-based algorithms, and then online non-parametric density estimation is cast into that framework. We give a detailed presentation of the algorithm, then describe how to extend the algorithm to efficiently implement online kernel regression. We then present the results of implementing these methods, both in terms of speed and accuracy, for uniformly and multivariate normally distributed points. Further improvements and extensions are then discussed. Appendix A shows how to derive the necessary computationally efficient series expansions and translation operators for any kernel function that can be represented by a Taylor series.

2 Multipole-Based Algorithms

2.1 Background

The Fast Multipole Algorithm (FMA) [12] and its variants [2, 3, 5, 6, 8, 9, 13, 20, 21, 27, 30, 35] have gained wide popularity in potential field problems where a set of n points interact according to a given potential function where the task is to compute at arbitrary points the field due to every given point. Applications are found in molecular dynamics, astrophysics, circuit device simulation and other electromagnetic field problems. For the most part, these applications have used the rather simple $\frac{1}{r}$-type Coulombic (or gravitational) potential function. One exception is the extension of the FMA by Elliott [9] to arbitrary $\frac{1}{r^{2\mu}}$ potentials. Additionally, a paper by Smith [30] sketches how to compute multipole expansions for general potentials, but the bulk of the paper is devoted to the Coulomb potential.

The algorithm of Greengard and Strain [14] for computing the discrete Gauss transform, which they note has applications to non-parametric statistics, is the only precedent to the current work. It solves an N-body problem with the multivariate Gaussian potential, and was implemented in 2D. It is much different from this work. Although it uses series expansions to represent the sum of distant Gaussians, it is not an FMA-like hierarchical algorithm with the full complement of translation operations. Instead it chops space into a uniform set of boxes, with the size of the boxes based on the target error and the bandwidth. Then each box interacts with some constant number of boxes, that grows as dimensionality is increased and as error is decreased. It is designed for the Gaussian kernel, and they note it can be extended to kernels that are the product of a multivariate polynomial function with a Gaussian. It relies on being able to represent the function by Hermite polynomials. Another paper by Strain [31] extends the above algorithm to handle the case where the Gaussians summed may have different bandwidths.

Our new algorithm will draw on techniques from the FMA, but applied novelly to create an online* non-parametric estimation algorithm. A brute force online algorithm contains a double summation, and would cost $O(n^2)$ (quadratic) time, while the FMA can compute this sum in $O(n)$(or linear) time, to any desired precision. Like the FMA, in online density estimation we are given n points that each contribute to the density according to the kernel function chosen. In contrast to the FMA, we wish to compute the contributions of the n kernel functions at *only* a single point. Furthermore, n is not fixed as within the FMA, but grows as more data become available. Thus after each new datapoint is added to the density, one would like to be able to do another density query and somehow avoid recomputing this $O(n)$ sum from scratch.

*An online algorithm is defined as an algorithm operating in a simulated real-time environment, where the input dataset grows with each new point seen (or grows in blocks), and recomputes the estimates by adding input points sequentially.

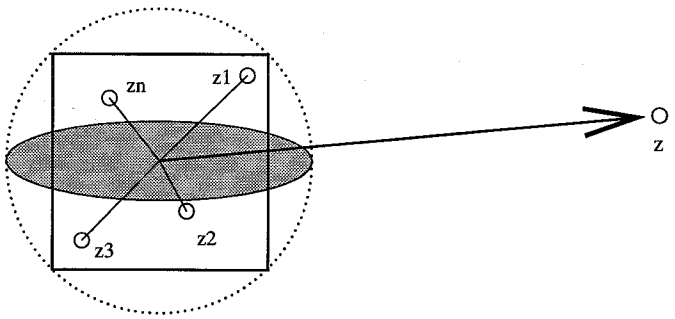

Figure 1: Multipole expansion

A multipole expansion describes the potential at $\vec{z}$ due to a group of particles $\vec{z}_1 \ldots \vec{z}_n$, and converges at some distance from the center of the enclosing hypersphere.

The chief difficulty in implementing multipole-based algorithms is the complexity of deriving translation operators for the series expansions involved in the algorithm, and bounding the error in using the expansion approximations. Elliott [9] makes great strides in simplifying the mathematics of manipulating multipole expansions for the Coulomb and Lennard-Jones potentials, with the key observation that if written as a power series in a certain way, the translations can be described in terms of the initial series forms with minor index changes. This work used spherical coordinates which simplified the computations for electrostatic potentials, but obscured the generalizable approaches of how to perform multipole translations for general functions. We will generally follow the notation of [9] in appendix A, showing how to perform multipole translations for any function that can reasonably be approximated by a Taylor series. We also present a new technique for bounding the error for general kernels.

2.2 Multipole Expansions

Given any potential function, $\Phi(\vec{z}) \rightarrow \Re^1$ with $\vec{z} \in \Re^d$, a *multipole expansion* is synonymous with forming a Taylor expansion for $\Phi(\vec{z} - \vec{z}_0)$ in the variable $\vec{z}_0$ about the origin in an appropriate coordinate system. The term *multipole expansion* comes from potential field theory, where there is a natural physical mapping from terms of the expansion to monopole, dipole, quadrupole (and so on) properties of groups of charges. We thus use (or perhaps abuse) the term loosely to refer to the Taylor series for a potential function that converges at some distance from the center (often outside) of the hypersphere containing the particles within the expansion (see figure 1).

The Taylor series for $\Phi(\vec{z} - \vec{z}_0)$ can be split into a dot product of two vectors, F and G, that separate the variables $\vec{z}$ and $\vec{z}_0$. The terms of the two series have a 1-1 correspondence, and their dot product forms the potential function:

$$\Phi(\vec{z} - \vec{z}_0) = F(\vec{z}_0) \bigodot G(\vec{z}). \tag{1}$$

Computationally, the series for F is truncated to a certain order, p and stored as an array of various powers of the coordinate variables. One can then evaluate the potential at $\vec{z}$ due to the particle at $\vec{z}_0$ by computing the terms for $G(\vec{z})$ and element-wise multiplying corresponding terms with $F(\vec{z}_0)$, and taking the sum of products. The advantage of separating Φ into a product of vectors F and G is that one can form a net multipole expansion due to many points by:

$$\sum_{i=1}^{n} \Phi(\vec{z} - \vec{z}_i) = G(\vec{z}) \bigodot \sum_{i=1}^{n} F(\vec{z}_i), \tag{2}$$

obtaining a single set of coefficients for all of the particles $\vec{z}_i$, and then evaluate the potential at any given point $\vec{z}$ at a cost related only to the number of terms in the expansion, not the number of contributing points, n included in the expansion. The appendix describes how to form these F and G components of the expansion for general functions in arbitrary dimensions. We discuss the error in the series approximations in section 3.2.

3 Density Estimation by Multipole Methods

3.1 Multipole Formulation for Density

The density at a point $\vec{z} \in \Re^d$, due to n points, $\vec{z}_1 \ldots \vec{z}_n$, using the kernel function $\mathcal{K}$, is estimated by the sum:

$$\hat{f}(\vec{z}) = \frac{1}{h^d n} \sum_{i=1}^{n} \mathcal{K}\left(\frac{\vec{z} - \vec{z}_i}{h}\right), \tag{3}$$

where h is the bandwidth parameter which controls the smoothing of the density. The kernel function is a continuous, bounded and symmetric real function which integrates to one [Hardle][15]:

$$\int \mathcal{K}(\vec{z})\, d\vec{z} = 1. \tag{4}$$

A commonly used kernel is the multivariate normal,

$$\mathcal{K}(\vec{z}) = (2\pi)^{-\frac{d}{2}} \exp\left(-\frac{||\vec{z}||^2}{2}\right), \tag{5}$$

which gives the following form for the kernel density, due to n points:

$$\hat{f}(\vec{z}) = \frac{1}{h^d n} (2\pi)^{-\frac{d}{2}} \sum_{i=1}^{n} \exp\left(-\frac{||\vec{z} - \vec{z}_i||^2}{2h^2}\right). \tag{6}$$

We can cast this into a multipole framework by simply forming the multivariate Taylor series for $\mathcal{K}\left(\left(\vec{z} - \vec{z}_0\right)/h\right)$, expanding in $\vec{z}_0$ about 0. One can then form a net multipole expansion due to many kernel functions by

$$\sum_{i=1}^{n} \mathcal{K}\left(\frac{\vec{z} - \vec{z}_i}{h}\right) = G(\vec{z}) \bigodot \sum_{i=1}^{n} F(\vec{z}_i), \tag{7}$$

and evaluate the density at any given point z at a cost related only to the number of terms in the expansion, not the number of kernel functions, n included in the expansion. We drop the normalization factor

$$\frac{1}{h^d n} (2\pi)^{-\frac{d}{2}}, \tag{8}$$

as it can be applied after all multipole expansions have been summed within an online algorithm.

Using the 2D Gaussian kernel as an example, the sum at $\vec{z} = (x, y)$, due to n kernel functions can be expressed as a bivariate Taylor series to fourth order:

$$\sum_{i=1}^{n} \mathcal{K}\left(\frac{\vec{z} - \vec{z}_i}{h}\right) = \sum_{i=1}^{n} \exp\left(-\frac{(x - x_i)^2 + (y - y_i)^2}{2h^2}\right) = G(\vec{z}) \bigodot \sum_{i=1}^{n} F(\vec{z}_i)$$

$$\approx \exp\left(-\frac{x^2+y^2}{2h^2}\right)\left\{\begin{array}{ccccc} & & 1 & & \\ & \frac{x}{h^2} & & \frac{y}{h^2} & \\ \frac{x^2}{h^4}-\frac{1}{h^2} & & \frac{xy}{h^4} & & \frac{y^2}{h^4}-\frac{1}{h^2} \\ \frac{x^3}{h^6}-\frac{3x}{h^4} & & \frac{x^2y}{h^6}-\frac{y}{h^4} & & \frac{xy^2}{h^6}-\frac{x}{h^4} \qquad \frac{y^3}{h^6}-\frac{3y}{h^4} \end{array}\right\} \odot$$

$$\sum_{i=1}^{n}\left\{\begin{array}{ccccc} & & 1 & & \\ & x_i & & y_i & \\ \frac{x_i^2}{2} & & x_iy_i & & \frac{y_i^2}{2} \\ \frac{x_i^3}{6} & & \frac{x_i^2y_i}{2} & & \frac{x_iy_i^2}{2} \qquad \frac{x_i^3}{6} \end{array}\right\}. \tag{9}$$

Although represented in a two-dimensional tabular form, one can think of the F and G components of the multipole expansions as one-dimensional vectors. The summation is just a component-wise vector sum, and the dot product follows the usual definition of a vector dot product. In higher dimensions it becomes computationally convenient and very efficient to flatten the multidimensional triangular tables of coefficients into one-dimensional vectors.

Note that the F component of the expansion is just powers of the point locations relative to the origin of the expansion. This means that if one desires to change the bandwidth, or even use an entirely different kernel function, that density can be computed from the same data structure.

3.2 Error Bounds

Much has been written on error bounds of multipole-based algorithms [4, 9, 12, 24, 25]. It is quite a complex topic, and one that has often been shaped more by practical efficiency considerations than by mathematical rigor. For a general kernel, the error of evaluating a multipole expansion to order p is given by summing the tail of the series. For example, the multipole expansion for the 1D Gaussian is given by:

$$\exp\left(-\frac{(x-x_0)^2}{2h^2}\right) = \exp\left(-\frac{x^2}{2h^2}\right)\sum_{n=0}^{\infty}\frac{\left(\frac{x_0}{h\sqrt{2}}\right)^n H_n\left(\frac{x\sqrt{2}}{2h}\right)}{n!}, \tag{10}$$

with $H_n(x)$ the Hermite polynomials. The error of truncating the expansion to order p is given by:

$$\left|\exp\left(-\frac{(x-x_0)^2}{2h^2}\right) - \exp\left(-\frac{x^2}{2h^2}\right)\sum_{n=0}^{p-1}\frac{\left(\frac{x_0}{h\sqrt{2}}\right)^n H_n\left(\frac{x\sqrt{2}}{2h}\right)}{n!}\right|. \tag{11}$$

Bounds on this error can be derived analytically (see [14]). However, we find in practice that setting the parameters to achieve these bounds often gives 4 to 5 significant figures more accuracy than desired. Of course, it is comforting that one can compute kernels with expansions to well within a specified error tolerance. However that extra accuracy takes up much additional CPU time that is wasted if a more modest accuracy is required for a given application.

We decided a new approach was needed to obtain the tightest possible bounds. Given fixed p, x_0 and tolerance ϵ, we numerically compute the smallest x such that Eq. 11 is less than ϵ and such that there is no greater x such that Eq. 11 is greater than ϵ. Eq. 11 is an oscillating polynomial decaying as fast as the tails of the Gaussian decrease. There are maxima and minima at all scales and one has to be very cautious in writing convergent minimization schemes. Such schemes do not have to be particularly fast as the bounds need be computed only once at the beginning of our online algorithm for a fixed bandwidth, h.

It should also be noted that we use absolute error bounds, which departs from the approach

used with the Coulomb potential. With the Coulomb potential, the relative error continues to die off as the multipole expansion is evaluated further and further away. However, with the Gaussian kernel, when you form a relative error, the error goes down as you get farther away, then at a certain point goes up again, as you try to approximate the tails of the Gaussian which decrease faster than polynomially. Thus there is a lower and upper distance range within which the relative error falls within tolerance, as opposed to the Coulomb potential which has a one-sided range beyond which the relative error is within tolerance.

4 The MODE Algorithm

4.1 Algorithm

We now present a description of the Multipole-accelerated Online Density Estimation (MODE) algorithm. The algorithm is quite general and allows implementation of a wide variety of kernels. Kernels not differentiable over the whole computational region, such as the Epanechnikov kernel, cannot be implemented within the MODE algorithm. On input to the algorithm, one specifies the number of dimensions, d, a target error, ϵ, the bandwidth, h, and the order of the expansions, p.

4.1.1 Data structure

The datapoints are enclosed in a regular d-dimensional box. As points are added to the box and its sub-boxes, a 2^d-ary tree is formed, describing the adaptive spatial decomposition, with points stored at leaf nodes. Each box has 2^d children, each of side length $1/2$ that of the parent. We scale all points to fit in a unit box of side length 1 by dividing every datapoint by twice the length of the longest dimension. The bandwidth should also be scaled accordingly by dividing it by the same factor.

4.1.2 Computing error bounds

Currently we use the 1D error equation for the given kernel (analogous to Eq. 11) for all dimensions. A multipole expansion for a given box is a sum of multipole expansions for all points in the box. The maximum distance a point can be from the center of a d-dimensional box of side length 1 is the radius of a d-dimensional hypersphere that encloses the box:

$$r = \frac{\sqrt{d}}{2}. \tag{12}$$

Using the method mentioned in section 3.2 and given some upper bound on the depth of the tree, ν, and given p, h, and ϵ, we compute for each size of box, $2^0, \ldots, 2^{-\nu}$, the closest point x that it is *safe* to evaluate a multipole expansion at, with x_0 in Eq. 11 given by $r/2^i$. By *safe*, we mean that the error in truncating the multipole expansion to order p is less than ϵ. With certain kernels such as the Gaussian, the multipole expansion is absolutely convergent. Hence at a certain depth, x can become smaller than x_0 or even go to zero. There is no need to make the tree go any deeper than the point where the expansion is convergent for boxes of size 2^{-i} for all x within the unit box. We can set ν to this depth.

4.1.3 Adding a point

Whenever a point is added, one traverses down the tree to find the leaf node that the point belongs to. If that leaf node has less than some constant number of points (say 40), or the depth is ν,

add the new point to the leaf node, and add the contribution of that kernel to the multipole expansion for the points in that box. Otherwise, sub-divide that space into 2^d sub-spaces, dividing the points among those new leaves according to their position in space. Then add the datapoint to the appropriate leaf, and compute multipole expansions of order p for each leaf.

4.1.4 Deleting a point

It should also be noted that it is quite easy to delete a point, which is a common operation in online algorithms. To do this, one effectively adds a counteracting point of opposite magnitude at the same location as the point being deleted. This is done by adding a multipole expansion of opposite sign to each multipole expansion along the path to where the point is stored in the tree. Then delete the point stored in the leaf node, reducing the number of points in the data structure by one. If all of the points in all of the leaves of a parent are deleted, the tree can be compressed by deleting those leaves, and converting the parent node into a leaf.

4.1.5 Updating the density

There are two ways to do this. If density queries are very frequent, we update the density when adding points by simply adding the expansion for the point at each successive level to the non-leaf nodes as we descend the tree. If density queries are very infrequent, we add points as described in section 4.1.3, that is, only updating the density at the leaves. Then whenever a density query is done, some subset of the boxes at various levels will need to have their densities updated. We do this by recursively descending from those boxes to all of their leaves, and then from the leaf node, pass the multipole expansion for that node upward via a M2M translation, translating the expansion for it and its $2^d - 1$ siblings to the center of their parent box to get a net expansion for every particle that is a descendant of the parent. Continue this process recursively upwards. This is like the *upward pass* of the FMA described in [12], only it is performed on a subset of the tree.

4.1.6 Density query

To obtain the density at a point $\vec{z}$, one recursively traverses the tree downwards, evaluating the multipole expansion for a given box at $\vec{z}$ if the distance between the box center and $\vec{z}$ is sufficient, given the error bounds computed at the beginning of the algorithm. If the box is not sufficiently separated, recursively traverse its children. If a leaf node is reached and is not sufficiently separated from $\vec{z}$, compute the effect of those points on $\vec{z}$ directly.

4.2 Number of Expansion Terms Versus Increased Cell Separation

There is a tradeoff between increasing the separation criteria versus increasing the number of terms in the expansions. With more expansion terms, one can evaluate a multipole expansion at the same distance and obtain a smaller error. However, the size of those expansions grows as p^d, and if one performs M2M translation operations, those operations scale as $O(p^{2d})$. One can reduce the error by using a greater cell separation, but then more interactions have to be directly computed rather than accumulated with multipole expansions. Typically a balance is experimentally determined, as the cost of direct interactions depends on the speed of computation of the kernel function on the target architecture (a costly exponential for the Gaussian), while the speed of multipole computations is primarily the cost of floating point operations on polynomials $(+, -, *, /)$.

4.3 Upper Distance Bounds

With the case of the Gaussian kernel function, and other functions that die off faster than polynomially, it is possible to save some additional work. Given a user-specified tolerance, if an expansion is so far away from the evaluation point that the tails of the Gaussian are below the tolerance, then the expansion need not be evaluated at all. This is akin to the *cutoff* methods used with the Coulomb potential.

4.4 Computational Complexity

4.4.1 Adding a point

To add a point, we must traverse the tree to find its leaf node, and possibly add a multipole expansion to each node in the tree along the path to the leaf. The expected tree depth in d dimensions is $O(\log_{2^d} n)$ for most reasonable distributions, but see [12] for a discussion of the adaptive decomposition. The size of a multipole expansion of order p in d dimensions is p^d. Hence adding a point costs an expected $O(p^d \log_{2^d} n)$ time, or $O(p^d + \log_{2^d} n)$ if density is only updated at the leaves.

With the Gaussian there is always a depth at which the box is sufficiently small relative to h that one can evaluate the expansion at any distance within or without the cell. This implies a constant depth tree, giving an update time of $O(p^d)$.

4.4.2 A density query

The running time for a density query is proportional to the number of multipole expansions that have to be evaluated. The cost of evaluating an expansion is the cost of computing the G part of the expansion and taking a dot product with the F part. Both F and G are $O(p^d)$ in size, and the individual terms in G may take up to $O(p)$ time to compute. Thus evaluating an expansion costs $O(p^d)$ time. The number of expansions that have to be evaluated is highly dependent on the kernel function, p, h, and ϵ. With kernels that die off very slowly and have absolutely convergent multipole expansions, or with kernels that die off faster than polynomially, the number of expansions to be evaluated can be constant. In between these cases (as say with the Coulomb potential function [4]), the number of expansions to be evaluated is generally logarithmic in n.

If the M2M translation scheme is used for infrequent density queries, and there have been k updates since the last density query, then as many as $\min(2^d k, L_t)$ leaves will need to have their expansions shifted, where L_t is the number of leaves in the tree. The 2^d factor is the number of siblings of each leaf, and if any leaf is modified, all of its 2^d brethren must be combined to update the parent. The query time at worst is $O(\min(2^d k, L_t))$ times the cost of the M2M translation. Each M2M translation is the cost of convolving two polynomials of size p^d, which takes $O(p^{2d})$ time. This can be reduced to $O(p^d \log p)$ with FFT techniques (see [9, 10]), but in general the FFT techniques only give a computational savings with large p in low dimensions [9]. If only a small region of the space gets modified many times, or L_t is small, then the query time can be small with this method.

5 Online Kernel Regression

Like online kernel *density estimation*, online kernel *regression* has two operations: add a d dimensional sample point ($d-1$ independent regressors and one dependent variable), and query for the expected value of the dependent variable at the $d-1$ dimensional vector of independent regressors.

Ideally, we would like to formulate an algorithm for kernel regression in terms of the multipole expansions for the density. It turns out this is possible. We will refer to the $d-1$ independent regressors as $\vec{x}$ and the dependent variable as y, and let $\vec{z} = (y, \vec{x})$ be the vector of all d random variables. The regression equation is [33]:

$$E(y|\vec{x}) = \frac{\int y f(\vec{z}) dy}{\int f(\vec{z}) dy}. \tag{13}$$

If we make the substitution

$$\hat{f}(\vec{z}) = \frac{1}{h^d n} \sum_{i=1}^{n} \mathcal{K}\left(\frac{\vec{z} - \vec{z}_i}{h}\right), \tag{14}$$

we obtain the non-parametric density form for the conditional expectation:

$$E(y|\vec{x}) = \frac{\int y \sum_{i=1}^{n} \mathcal{K}\left(\frac{\vec{z}-\vec{z}_i}{h}\right) dy}{\int \sum_{i=1}^{n} \mathcal{K}\left(\frac{\vec{z}-\vec{z}_i}{h}\right) dy}. \tag{15}$$

It can be shown [33] that this equation is equivalent to

$$\frac{\sum_{i=1}^{n} y_i \mathcal{K}_1\left(\frac{\vec{x}-\vec{x}_i}{h}\right)}{\sum_{i=1}^{n} \mathcal{K}_1\left(\frac{\vec{x}-\vec{x}_i}{h}\right)}, \tag{16}$$

where $\mathcal{K}_1 : \Re^{d-1} \rightarrow \Re$ is the reduced-dimension result of integrating the kernel function over the dependent dimension; that is,

$$\mathcal{K}_1(\vec{x}) = \int \mathcal{K}(\vec{z}) dy. \tag{17}$$

Conveniently, any normalization constants can be dropped as they cancel out in the division. Even more conveniently, the integral of a Gaussian kernel is itself a reduced dimension Gaussian. Hence, the conditional expectation is given by the fraction of two reduced dimensional Gaussian kernel sums. One sum is weighted by the dependent variable, and the other has a unit weighting,

$$E(y|\vec{x}) = \frac{G_1(\vec{x}) \odot \sum_{i=1}^{n} y_i F_1(\vec{x}_i)}{G_1(\vec{x}) \odot \sum_{i=1}^{n} F_1(\vec{x}_i)}. \tag{18}$$

Computationally, two multipole expansions must be stored, but the work in forming the expansions, and evaluating them can largely be combined, as they differ only by a constant weighting factor.

6 Computational Results

6.1 Online Density Estimation

We tested the computational efficiency of the MODE algorithm implemented with a Gaussian kernel in one through five dimensions. All of the Taylor expansions were done with the Maple mathematical software system. A symbolic package such as Maple is essential for eliminating bookkeeping errors, both when developing the methods and when generating optimized code.

Tables 1 through 5 report the total elapsed wall-clock time for running the algorithm on 50000 points in $d = 1 \ldots 5$ dimensions with bandwidths, h, ranging from from 1 to $\frac{1}{10000}$, and the target error, ϵ, ranging from 1e-1 to 1e-8. We started with one datapoint, then queried the density at the new datapoint, and then added that new point and repeated the process 50000 times. To test the

variability of runtime with distribution, we ran the algorithm on three different distributions: 50000 uniformly distributed points, 50000 multivariate normal points with a heteroskedastic covariance matrix and 50000 normally distributed points with an ill-conditioned heteroskedastic covariance matrix, having a condition number of about 10^6. All data were scaled into a unit hypercube of side length 1 by dividing every datapoint by twice the length of the longest dimension. We did not scale the bandwidth. The runs were done for values of p from 2 through 6 for each level of ϵ, and the shortest runtime was reported. For a given target error, ϵ, the worst case error of any given density query is at or below ϵ. Typically, the average error over all density queries is one to three orders of magnitude smaller than the target. In one through five dimensions the direct method takes between 1600 and 2000 seconds, while the MODE times are much better, though they vary with the error, and the bandwidth. The initial version of the error-bounding algorithm also has come troubles finding tight enough bounds in those cases which leads to the longer running times. Interestingly, it is the bandwidths, h of between $\frac{1}{2}$ and $\frac{1}{100}$ that give the slower results. The very broad bandwidths are easily approximated by a low-degree polynomial and run very fast. The very sharp bandwidths fall off so quickly that the interactions do not extend very far, and hence multipole expansions can be evaluated a very short distances, giving fast runtimes. Further optimizations are underway. All computations were done on a dedicated Power Challenge machine with a 75Mhz MIPS R8000 processor.

To illustrate the advantage over the direct method, we compared the running time of a brute force density computation versus the MODE method with $h = 1/2$ and $\epsilon = 0.01$, for $n = 1 \ldots 1,000,000$ for each of the five dimensions. Figure 2 shows these results. The running times are somewhat better than the corresponding ones in the tables because we have tuned the target ϵ and p to give a worst case error of just slightly below 0.01. In contrast, the worst case errors for the runs in the tables are sometimes 2 to 3 orders of magnitude smaller than the target ϵ. Hence instead of ranging from 1e-1 to 1e-8 most of the run times in the tables are really for worst case accuracies ranging from 1e-3 to 1e-11, and average case accuracies from 1e-4 to 1e-12. Further work on fine-tuned error control is still required.

6.2 Online Kernel Regression

We implemented online kernel regression in $d = 2 \ldots 5$, where the fraction of two $d-1$ dimensional densities must be computed. Through clever combination of computations, the running times in all cases were less than 120 percent of the cost of performing one $d-1$ dimensional density query. That is, computing d dimensional conditional expectation runs in about the same time as $d-1$ density estimation. The errors were well-behaved, though one has to be cautious about performing an expectation query in a region of near zero density, lest division by zero occur.

7 Further Work

7.1 $O(n)$ Offline Non-parametric Density Estimation

Although it is desirable to have an up-to-date estimation of the non-parametric density, it may be the case that a periodic recomputation of the density is sufficient. Also, one may be simply interested in plotting the non-parametric density for a fixed set of points. One approach to these problems is to solve the offline problem of computing the density over a large grid of $g = O(n)$ points. This problem can be solved in $O(n)$ time by straightforward application of the FMA with the appropriate kernel function. The complexity of implementing the FMA has traditionally been difficult for the rather simple potentials used in physics problems, let alone the more complex

1D Density n=50000		$h=1$	$h=\frac{1}{2}$	$h=\frac{1}{10}$	$h=\frac{1}{100}$	$h=\frac{1}{1000}$	$h=\frac{1}{10000}$
ϵ =1e-1	Uniform	0.420	0.505	1.60	3.84	9.14	10.2
	Normal	0.462	0.539	1.90	3.96	5.92	8.33
	Ill Cond.	0.461	0.545	1.72	3.63	5.55	7.55
ϵ =1e-2	Uniform	0.463	0.601	3.18	6.33	9.46	10.4
	Normal	0.509	0.641	2.83	5.00	7.25	9.82
	Ill Cond.	0.502	0.648	2.46	4.69	6.84	9.25
ϵ =1e-3	Uniform	0.569	0.927	3.54	7.82	10.1	11.7
	Normal	0.612	1.06	3.33	5.85	8.77	11.4
	Ill Cond.	0.613	0.968	3.01	5.36	8.30	10.8
ϵ =1e-4	Uniform	0.643	1.05	6.57	9.27	14.6	11.9
	Normal	0.649	1.09	3.81	7.45	9.63	11.9
	Ill Cond.	0.652	1.00	3.23	7.08	9.17	11.4
ϵ =1e-5	Uniform	0.931	1.84	6.46	10.2	13.3	13.4
	Normal	1.08	1.49	5.33	8.29	10.3	13.7
	Ill Cond.	0.969	1.46	4.89	7.92	9.88	12.4
ϵ =1e-6	Uniform	0.966	2.59	5.80	14.2	22.0	12.6
	Normal	1.11	1.81	5.67	10.4	13.0	14.2
	Ill Cond.	1.01	1.59	5.16	9.87	12.5	12.9
ϵ =1e-7	Uniform	1.60	3.37	11.8	17.3	22.6	12.9
	Normal	1.11	1.88	7.72	12.2	14.5	14.8
	Ill Cond.	1.07	1.65	7.48	11.6	13.7	13.5
ϵ =1e-8	Uniform	1.83	3.53	15.8	21.2	32.0	13.3
	Normal	1.79	3.12	9.89	13.9	19.4	15.6
	Ill Cond.	1.59	2.76	9.47	13.0	18.5	14.0

Table 1: Running times for 1D MODE density computations (sec.)

Runtimes are presented for three different data distributions: 50000 uniformly distributed points, 50000 multivariate normal points with a heteroskedastic covariance matrix and 50000 normally distributed points with an ill-conditioned heteroskedastic covariance matrix.

2D Density n=50000		$h=1$	$h=\frac{1}{2}$	$h=\frac{1}{10}$	$h=\frac{1}{100}$	$h=\frac{1}{1000}$	$h=\frac{1}{10000}$
ϵ =1e-1	Uniform	0.651	1.58	11.9	19.8	9.49	8.30
	Normal	0.697	1.60	10.1	18.8	13.8	10.8
	Ill Cond.	0.683	1.43	9.10	17.7	12.7	9.80
ϵ =1e-2	Uniform	1.05	1.93	13.3	26.4	9.69	8.50
	Normal	1.08	2.08	14.5	28.9	15.5	10.9
	Ill Cond.	1.08	1.91	13.3	27.4	14.0	9.88
ϵ =1e-3	Uniform	1.56	2.53	17.6	33.5	10.2	8.32
	Normal	1.57	2.78	18.0	44.1	16.7	11.1
	Ill Cond.	1.57	2.53	17.1	41.3	15.1	9.98
ϵ =1e-4	Uniform	2.29	5.92	34.7	39.4	10.7	8.36
	Normal	2.51	6.10	34.1	60.7	17.6	11.2
	Ill Cond.	2.27	5.37	32.3	58.0	16.0	10.0
ϵ =1e-5	Uniform	2.46	6.13	48.7	44.8	10.5	8.49
	Normal	2.73	6.09	42.9	93.9	18.8	11.1
	Ill Cond.	2.53	5.44	41.2	90.2	16.9	10.1
ϵ =1e-6	Uniform	3.05	7.15	102.	50.4	10.7	8.42
	Normal	3.05	7.09	89.0	158.	19.8	11.1
	Ill Cond.	3.04	6.38	85.8	152.	17.8	10.1
ϵ =1e-7	Uniform	6.12	23.5	156.	55.8	11.2	8.51
	Normal	6.06	16.9	120.	212.	20.6	11.3
	Ill Cond.	5.45	16.1	115.	200.	18.6	10.1
ϵ =1e-8	Uniform	7.14	24.1	262.	60.7	11.5	8.66
	Normal	6.97	16.8	178.	302.	21.4	11.1
	Ill Cond.	6.37	16.1	173.	285.	19.3	10.2

Table 2: Running times for 2D MODE density computations (sec.)

3D Density n=50000		$h=1$	$h=\frac{1}{2}$	$h=\frac{1}{10}$	$h=\frac{1}{100}$	$h=\frac{1}{1000}$	$h=\frac{1}{10000}$
ϵ =1e-1	Uniform	1.22	3.43	38.6	23.0	16.4	16.0
	Normal	1.26	3.74	42.3	65.7	26.9	23.3
	Ill Cond.	1.27	3.41	35.2	60.0	23.6	20.9
ϵ =1e-2	Uniform	2.75	5.33	90.7	26.5	15.9	15.0
	Normal	2.78	5.82	74.2	97.0	27.4	23.3
	Ill Cond.	2.73	5.35	63.0	89.5	24.6	20.9
ϵ =1e-3	Uniform	4.65	9.45	256.	29.6	16.2	15.2
	Normal	5.04	9.74	191.	129.	28.4	23.5
	Ill Cond.	4.67	9.36	155.	118.	25.3	21.0
ϵ =1e-4	Uniform	6.66	28.6	540.	32.5	16.6	15.0
	Normal	7.05	22.1	255.	163.	30.3	24.1
	Ill Cond.	6.60	18.2	212.	147.	26.1	21.1

Table 3: Running times for 3D MODE density computations (sec.)

4D Density n=50000		$h=1$	$h=\frac{1}{2}$	$h=\frac{1}{10}$	$h=\frac{1}{100}$	$h=\frac{1}{1000}$	$h=\frac{1}{10000}$
ϵ =1e-1	Uniform	3.58	10.3	215.	44.1	29.3	32.3
	Normal	5.55	15.8	263.	174.	60.8	57.4
	Ill Cond.	3.60	9.92	134.	110.	57.6	53.4
ϵ =1e-2	Uniform	8.54	102.	478.	48.1	29.7	32.2
	Normal	9.08	49.4	644.	150.	62.7	57.6
	Ill Cond.	8.32	41.5	427.	144.	59.0	53.8
ϵ =1e-3	Uniform	14.3	147.	705.	52.3	29.9	33.2
	Normal	14.3	64.7	1119.	184.	64.3	58.0
	Ill Cond.	14.0	55.4	755.	177.	60.4	53.7
ϵ =1e-4	Uniform	121.	1719.	934.	54.6	30.1	32.8
	Normal	53.6	217.	3224.	214.	66.0	57.4
	Ill Cond.	43.4	176.	2498.	206.	61.3	54.0

Table 4: Running times for 4D MODE density computations (sec.)

5D Density n=50000		$h=1$	$h=\frac{1}{2}$	$h=\frac{1}{10}$	$h=\frac{1}{100}$	$h=\frac{1}{1000}$	$h=\frac{1}{10000}$
ϵ =1e-1	Uniform	6.88	30.1	547.	101.	79.8	78.7
	Normal	6.78	31.6	861.	260.	156.	150.
	Ill Cond.	6.83	28.1	593.	206.	126.	119.
ϵ =1e-2	Uniform	29.6	414.	985.	111.	80.6	78.7
	Normal	20.4	122.	3990	320.	160.	150.
	Ill Cond.	18.3	103.	2437	251.	128.	120.
ϵ =1e-3	Uniform	37.6	831.	1291	118.	81.0	77.6
	Normal	37.8	205.	5679	373.	163.	151.
	Ill Cond.	35.0	177.	3852	292.	130.	120.
ϵ =1e-4	Uniform	788.	4232	1565	123.	81.4	78.7
	Normal	186.	927.	5523	425.	165.	508.
	Ill Cond.	161.	676.	4826	328.	131.	119.

Table 5: Running times for 5D MODE density computations (sec.)

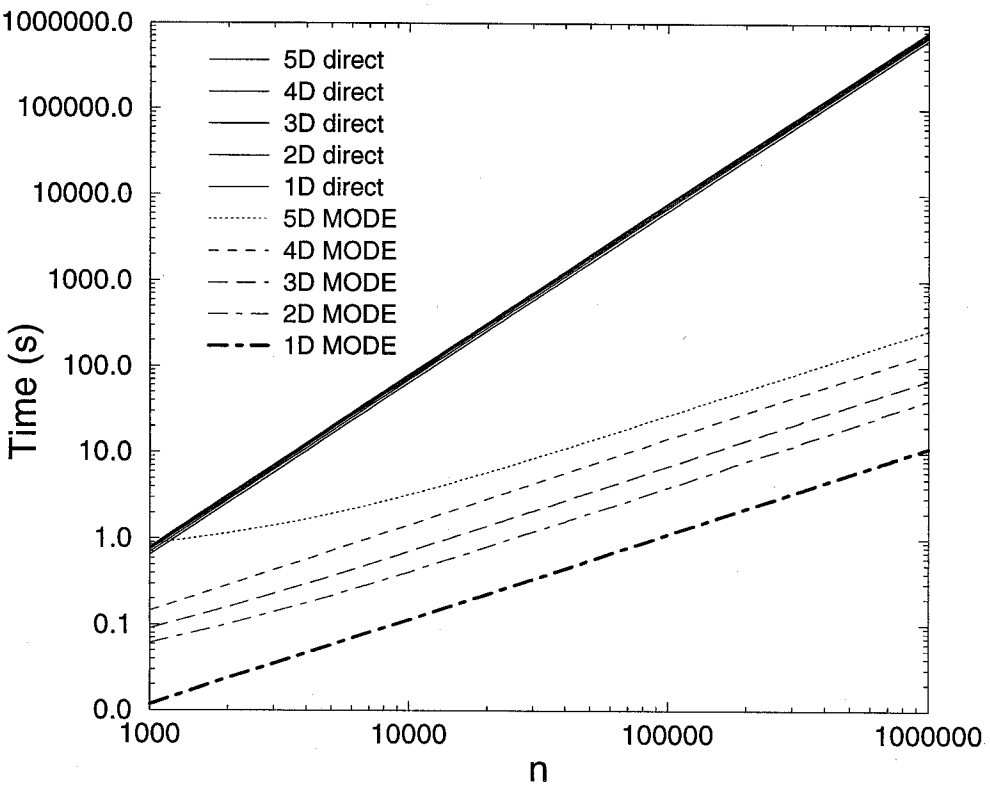

Figure 2: MODE versus direct running times

We compare the "brute force" direct online density computation to the MODE algorithm, for $n = 1000 \ldots 1000000$ in one through five dimensions. We selected a bandwidth of 0.5 and an error tolerance of 0.01, and used uniformly distributed data for convenience. For the brute force approach, we extrapolated from 50000 points to 1000000 as the running times are impossibly long with its $O(n^2)$ running time. The running times for the direct approach fall in a narrow band at the top of the graph. For a million points, the direct approach would take from 7 to 10 days to compute, depending on the number of dimensions. In one through 5 dimensions, the MODE approach respectively takes 11, 40, 70, 145, and 264 seconds to do online density updates and queries for the million points. The plot is on a log-log scale.

kernel functions contemplated in density estimation context. However, in appendix A, we outline how to derive translations operations for the series expansions involved in the FMA. With symbolic mathematics packages such as Maple, one can readily derive and even generate optimized code for the appropriate routines. Lambert's thesis [19] shows how to do this. Of course the online algorithm can be used without modification to solve the offline problem: simply perform n density updates followed by g density queries at the gridpoint locations at a cost of $O(n \log n + g \log n)$.

7.2 Bandwidth Issues

As mentioned in section 3.1, the F components of the multipole expansions are independent of the kernel function and the bandwidth. It is the F parts of the expansions that are stored within the tree, while the G parts, which include the bandwidth, are computed only at query time. Thus it is possible to vary the bandwidth as n increases, or even to apply different kernel functions without recomputing the expansions stored in the tree. The only issue to be dealt with is computing a new set of error bounds for the changed kernel. As long as the bandwidth is not changed often, one can recompute the bounds. If the bandwidth is only gradually decreased, then the same error bounds can be used for a long time – perhaps over the whole life of the computation. In typical applications, the bandwidth is reduced by some constant over n^{d+4} (see [29]), which decreases extremely slowly.

It should also be mentioned that partial derivatives and even multidimensional integrals of the density can also be computed in an online or offline fashion by straightforward differentiation or integration of the G component of the expansion.

The techniques presented very efficiently solve the problem where the bandwidth of the kernel function of all the points is the same value. Often, however, a better density estimate can be obtained by varying the bandwidth in different regions, usually in some manner that is dependent on the density of the data. Where there are fewer points, the bandwidth is made wider, and where there are many points, the bandwidth is made narrower. There are two ways to go about solving this problem. One is to use the MODE algorithm several times, once per bandwidth used. However, this would not allow a continuous varying of bandwidth, but would require discrete levels. To allow continuous varying of bandwidth, the bandwidth parameter, h must be included as a variable of the expansion. This would add an extra dimension to the problem in addition to the spatial dimensions that the points lie in. The error bounds and separation criteria would then have to incorporate the h parameter as part of computing interactions. A very fast algorithm to perform this most general form of kernel density estimation would be of enormous practical value.

References

[1] Y. Aït-Sahalia and A. Lo. Nonparametric estimation of state-price densities implicit in financial asset prices. Working paper, University of Chicago, 1995.

[2] A. W. Appel. An efficient program for many-body simulation. *SIAM Journal of Scientific and Statistical Computing*, 6(1):85–103, 1985.

[3] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324(4):446–449, December 1986.

[4] J. E. Barnes and P. Hut. Error analysis of a tree code. *The Astrophysical Journal Supplement Series*, 70:389–417, 1989.

[5] J. A. Board, Jr., J. W. Causey, J. F. Leathrum, Jr., A. Windemuth, and K. Schulten. Accelerated molecular dynamics simulation with the parallel fast multipole algorithm. *Chemical Physics Letters*, 198:89–94, 1992.

[6] J. A. Board, Jr., Z. S. Hakura, W. D. Elliott, and W. T. Rankin. Scalable variants of multipole-accelerated algorithms for molecular dynamics applications. In *Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 295–300, Philadelphia, PA, 1995.

[7] J. Boudoukh, M. Richardson, R. Stanton, and R. F. Whitelaw. A new strategy for dynamically hedging mortgage-backed securities. Working paper, New York University, 1995.

[8] H.-Q. Ding, N. Karasawa, and W. A. Goddard III. Atomic level simulations on a million particles: The cell multipole method for Coulomb and London nonbond interactions. *Journal of Chemical Physics*, 97(6):4309–4314, 1992.

[9] W. D. Elliott. *Multipole Algorithms for Molecular Dynamics Simulation on High Performance Computers.* PhD thesis, Duke University, Dept. Electrical Engineering, 1995.

[10] W. D. Elliott and J. A. Board, Jr. Fast Fourier transform accelerated fast multipole algorithm. *SIAM J. Sci. Comput.*, 17(2):398–415, 1996.

[11] K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing K-nearest neighbors. *IEEE Trans. Comput.*, C-24:750–753, 1975.

[12] L. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems.* MIT Press, Cambridge, MA, 1988.

[13] L. Greengard and V. Rokhlin. The rapid evaluation of potential fields in three dimensions. *Lecture Notes in Mathematics*, 1360:121–141, 1988.

[14] L. Greengard and J. Strain. The fast Gauss transform. *SIAM J. Sci. Stat. Comput.*, 12(1):79–94, 1991.

[15] W. Härdle. *Applied Nonparametric Regression.* Cambridge University Press, 1989.

[16] W. K. Härdle and D. W. Scott. Smoothing by weighted averaging of shifted points. *Computational Statistics*, 7:97–128, 1992.

[17] C. R. Harvey. The specification of conditional expectations. Working paper, Duke University, 1991.

[18] B. Jeon and D. A. Landgrebe. Fast Parzen density estimation using clustering-based branch and bound. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):950–954, 1994.

[19] C. G. Lambert. *Multipole-Based Algorithms in Molecular Biophysics and Non-Parametric Statistics.* PhD thesis, Duke University, Department of Computer Science, 1997.

[20] C. G. Lambert, T. A. Darden, and J. A. Board, Jr. A multipole-based algorithm for efficient calculation of forces and potentials in macroscopic periodic assemblies of particles. *J. Comp. Phys.*, 126:274–285, 1996.

[21] K. Nabors, F. T. Korsmeyer, F. T. Leighton, and J. White. Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations of potential theory. *SIAM J. Sci. Comput.*, 15(3):713–735, 1994.

[22] A. R. Pagan and G. W. Schwert. Alternative models for conditional stock volatility. *Journal of Econometrics*, 45:267–290, 1990.

[23] E. Parzen. On the estimation of a probability density and mode. *Ann. Math. Statist.*, 33:1065–1076, 1962.

[24] H. G. Petersen, E. R. Smith, and D. Soelvason. Error estimates for the fast multipole method. II. the three-dimensional case. *Proc. R. Soc. Lond. A*, 448:401–418, 1995.

[25] H. G. Petersen, D. Soelvason, J. W. Perram, and E. R. Smith. Error estimates for the fast multipole method. I. the two-dimensional case. *Proc. R. Soc. Lond. A*, 448:389–400, 1995.

[26] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Ann. Math. Statist.*, 27, 1956.

[27] J. Shimada, H. Kaneko, and T. Takada. Performance of fast multipole methods for calculating electrostatic interactions in biomacromolecular simulations. *Journal of Computational Chemistry*, 15(1):28–43, 1994.

[28] B. W. Silverman. Kernel density estimation using the fast Fourier transform. *Appl. Statist.*, 31:93–97, 1982. Statistical Algorithm AS176.

[29] B. W. Silverman. *Density estimation for statistics and data analysis.* Chapman & Hall, New York, 1986.

[30] E. R. Smith. Fast moment methods for general potentials in molecular dynamics. *Molecular Physics*, 86(4):769–796, 1995.

[31] J. Strain. The fast Gauss transform with variable scales. *SIAM J. Sci. Statist. Comput.*, 12(5):1131–1139, 1991.

[32] K. S. Trivedi. *Probability and statistics with reliability, queuing, and computer science applications.* Prentice-Hall, 1982.

[33] A. Ullah. Non-parametric estimation of econometric functionals. *Canadian Journal of Economics*, 21(3):625–658, 1988.

[34] M. P. Wand. Fast computation of multivariate kernel estimators. *Journal of Computational and Graphical Statistics*, 3(4):433–445, 1994.

[35] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Supercomputing '93*. IEEE Computer Society, 1993.

A Multipole Mathematics for General Potential Functions

Traditionally, applications of multipole-based algorithms have been restricted to the rather simple Coulomb potential function (i.e. kernel), because of the enormous complexity in deriving translation operators for the series expansions involved in the algorithm. To our knowledge, no multipole-based algorithm has been implemented in more than three dimensions prior to this work because of this complexity. We show a simple method for deriving these operators that will allows MODE, and indeed, any multipole-based algorithm to be applied to a broad class of kernel/potential functions in arbitrary dimensions with very little development time. See [19] for an exposition on using Maple to facilitate development of N-body algorithms.

B Multipole Expansions

Many multipole-based algorithms rely on the shifting of multipole expansions to derive their efficiency. The Fast Multipole Algorithm [12] contains a superset of these shifting operations, and we show how to compute each of them in turn. We will work in Cartesian coordinates. Let $\vec{z} \in \Re^d = [z_{(1)}, z_{(2)}, \ldots, z_{(d)}]^T$, write

$$F_{i_1,i_2,\ldots,i_d}(\vec{z}) = \prod_{k=1}^{d} \frac{z_{(k)}^{i_k - i_{k+1}}}{(i_k - i_{k+1})!} \tag{19}$$

and

$$G_{i_1,i_2,\ldots,i_d}(\vec{z}) = C_{i_1-i_2,i_2-i_3,\ldots,i_{d-1}-i_d,i_d}(\vec{z}) \prod_{k=1}^{d} (i_k - i_{k+1})! \tag{20}$$

where $i_k >= i_{k+1}$ for all k, $i_{d+1} = 0$ and $C_{i_1,\ldots,i_d}(\vec{z})$ denotes the coefficient of the term of the multivariate Taylor series expansion for $\Phi(\vec{z} - \vec{z}_0)$ about $\vec{z}_0 = 0$ corresponding to $\prod_{k=1}^{d} z_{0(k)}^{i_k}$. That is,

$$C_{i_1,\ldots,i_d}(\vec{z}) = \left(\prod_{k=1}^{d} \frac{1}{i_k!} \frac{\partial^{i_k}}{\partial z_{0(k)}^{i_k}} \right) \Phi(\vec{z} - \vec{z}_0) \Bigg|_{\vec{z}_0=0} \tag{21}$$

The symbolic package Maple has a library function called "coeftayl" which gives $C_{i_1,\ldots,i_d}(\vec{z})$ for arbitrary functions. The following then holds:

$$\Phi(\vec{z} - \vec{z}_0) = \sum_{i_1=0}^{\infty} \sum_{i_2=0}^{i_1} \cdots \sum_{i_d=0}^{i_{d-1}} F_{i_1,i_2,\ldots,i_d}(\vec{z}_0)\, G_{i_1,i_2,\ldots,i_d}(\vec{z}), \tag{22}$$

for $|\vec{z}| > |\vec{z}_0|$. To illustrate, let $\Phi(\vec{z}) = e^{z_{(1)}+z_{(2)}+z_{(3)}}$, then

$$F_{i_1,i_2,i_3}(\vec{z}) = \frac{z_{(1)}^{i_1-i_2}\, z_{(2)}^{i_2-i_3}\, z_{(3)}^{i_3}}{(i_1 - i_2)!\,(i_2 - i_3)!\,i_3!}, \tag{23}$$

$$G_{i_1,i_2,i_3}(\vec{z}) = C_{i_1-i_2,i_2-i_3,i_3}(\vec{z})\,(i_1 - i_2)!\,(i_2 - i_3)!\,i_3!, \tag{24}$$

and

$$C_{i_1,i_2,i_3}(\vec{z}) = \frac{1}{i_1!\,i_2!\,i_3!}\,\frac{\partial^{i_1}}{\partial z_{0(1)}^{i_1}}\frac{\partial^{i_2}}{\partial z_{0(2)}^{i_2}}\frac{\partial^{i_3}}{\partial z_{0(3)}^{i_3}}\Phi(\vec{z}-\vec{z}_0)\bigg|_{\vec{z}_0=0} = \frac{(-1)^{i_1+i_2+i_3}\,e^{z_{(1)}+z_{(2)}+z_{(3)}}}{i_1!\,i_2!\,i_3!}. \tag{25}$$

With the F and G formulation for the multipole expansions, the multipole translation and conversion operations for general functions become easy. One forms a multipole coefficients array, M for n points with weights q_j as:

$$M_{i_1,\ldots,i_d} = \sum_{j=1}^{n} q_j F_{i_1,\ldots,i_d}(\vec{z}_j). \tag{26}$$

This multipole expansion can be stored as a d-dimensional triangular matrix of real numbers, or for greater efficiency may be flattened to a 1D vector. We then have the following general theorems for translating the multiple expansions.

C Multipole to Multipole Translation (M2M)

Let M be the multipole expansion of radius r centered at the origin. Then the center of the expansion can be translated to a new center, $\vec{z}$, and have the resulting multipole expansion M' converge outside a radius of $r' = r + |\vec{z}|$. The elements of M' are given by:

$$M'_{i'_1,\ldots,i'_d} = \sum_{i_1=0}^{i'_1} \sum_{i_2=\max(0,i'_2-i'_1+i_1)}^{\min(i_1,i'_2)} \cdots \sum_{i_d=\max(0,i'_d-i'_{d-1}+i_{d-1})}^{\min(i_{d-1},i'_d)} M_{i_1,\ldots,i_d} F_{i'_1-i_1,\ldots,i'_d-i_d}(-\vec{z}). \tag{27}$$

D Multipole to Local Translation (M2L)

Suppose that n charges of strengths $q_1, q_2, \ldots, q_n$ are located inside a hypersphere S_a of radius a and center at $\vec{z}$, with $|\vec{z}| > a+b$ and $a \geq b$. Then the corresponding multipole expansion converges inside a hypersphere S_b of radius b centered about the origin. At any point inside S_b, the potential due to the charges in S_a can be described by a Taylor series, or local expansion L, translated to the origin whose elements are given by

$$L_{i'_1,\ldots,i'_d} = \sum_{i_1=0}^{p-i'_1} \sum_{i_2=0}^{i_1} \sum_{i_3=0}^{i_2} \cdots \sum_{i_d=0}^{i_{d-1}} M_{i_1,\ldots,i_d} G_{i'_1+i_1,\ldots,i'_d+i_d}(\vec{z}). \tag{28}$$

One can evaluate the potential at a point $\vec{z}$ inside the hypersphere S_b by:

$$\Phi(z) = \sum_{i_1=0}^{p-1} \sum_{i_2=0}^{i_1} \cdots \sum_{i_d=0}^{i_{d-1}} L_{i_1,\ldots,i_d} F_{i_1,i_2,\ldots,i_d}(\vec{z}). \tag{29}$$

E Local to Local Translation (L2L)

The local to local translation is used to pass the effects of distant cells down to cell children by shifting the local expansion from the center of the parent to the center of the child which is properly

contained in the parent cell. The equation for this new shifted local expansion, L' is:

$$L'_{i'_1,\ldots,i'_d} = \sum_{i_1=0}^{p-i'_1} \sum_{i_2=0}^{i_1} \sum_{i_3=0}^{i_2} \cdots \sum_{i_d=0}^{i_{d-1}} L_{i_1,\ldots,i_d} F_{i'_1-i_1,\ldots,i'_d-i_d}(\vec{z}). \tag{30}$$