Wavelet based density estimation for multidimensional streaming data

Daniel Weinand[a], Gedeon Nyengele[b], Mark Moyou[c], and Dr. Adrian Peter[d]

[a]Department of Computer Science, Pomona College, Claremont, California; daniel.weinand@pomona.edu
[b]Department of Electrical Engineering, Georgia Perimeter College, Decatur, Georgia; gedeonnyengele@hotmail.com
[c]Department of Engineering Systems, Florida Institute of Technology, Melbourne, Florida; mmoyou@fit.edu
[d]Department of Engineering Systems, Florida Institute of Technology, Melbourne, Florida; apeter@fit.edu

ABSTRACT

In the present information economy, the colossal amount of data generated daily has spawned the need for real-time data driven algorithmsrequiring minimal user influence. Density estimation is one path to extracting actionable intelligence from data, and wavelets serve as the building blocks on which this data-driven framework will be built. The goal of this project is to develop a multidimensional computational implementation of this wavelet density estimation (WDE) framework and showcase its utility on modeling several market related variables. The report begins with a background overview of wavelet theory and density estimation, then a description of the computational framework involved in this process. Finally, we describe the application in detail and conclude with an analysis of the estimator on several data sets.

Keywords: Non-parametric Estimators, Streaming Data, Wavelet Density Estimation

1. BACKGROUND

1.1 Layout

Before diving directly into wavelet based density estimators for streaming data, some background knowledge is necessary. First we provide a brief discussion of wavelets, their use as a basis for the L^2 function space, and the wavelet series approximation. This is followed by a brief description of density estimation and non-parametric density estimators in particular (of which wavelet density estimators are one class). Then, we briefly touch on the streaming data paradigm to transition into an analysis of two algorithms using wavelet density estimators on streaming data.

1.2 Wavelets:

A wavelet is a function with an average value of zero and finite energy.[18] One of the simplest examples of such a function is the Haar wavelet developed by Alfred Haar in 1909, which is defined as

$$f(x) = \begin{cases} +1 & 0 \leq x < \frac{1}{2} \\ -1 & \frac{1}{2} \leq x < 1 \\ 0 & otherwise \end{cases}$$

An essential feature of this wavelet is that the inner product of the wavelet and any integer shift of the wavelet is equal to zero. In other words, the Haar wavelet is orthogonal under integer translation, a feature it shares with the other major wavelet families such as the Daubechies.[5] Another important component of these particular types of wavelets is that they may have their support split in half while maintaining orthogonality.[5] This puts them in the class of so called "dyadic" wavelets which can be repeatedly halved in size in order to better approximate a function, allowing them to be used in multi-resolution analysis (described in subsection 1.10).

1.3 Wavelets as a basis for the L^2 function space

The L^2 function space is defined as the set of square integrable functions. In other words, this space is composed of functions whose square when integrated from $-\infty$ to ∞ is finite. Intuitively, one can see that functions in this space will only be defined as significantly different from zero over a relatively small interval. An essential feature of wavelets is that they have finite energy, in fact, all wavelets square integrate to one. Thus wavelets fall into the L^2 function space and are normal (magnitude 1). Furthermore, we have already seen that an essential feature of wavelets is that they remain orthogonal under integer translation and dyadic segmentation. Combining these facts we can see that wavelets act as an orthonormal basis for the L^2 function space. In order to find the coordinates (b) of any square integrable function in terms of the wavelet basis, we simply calculate the inner product of respective basis function with the function. Then we have $f * \varphi_{j,k} = \langle f, \varphi_{j,k} \rangle = b_{j,k}$. The reconstruction of the function $f(x)$ with the wavelet coefficients $b_{j,k}$ is shown in Equation 1.1.

$$f(x) = \sum_j \sum_k b_{j,k} \psi_{j,k}(x), \quad j \in \mathbb{N} \tag{1.1}$$

1.4 The Wavelet Series Expansion:

In many practical applications, there exists a large class of signals with components varying in both time and frequency. When using a Fourier transform on these kinds of signals, this local variation in the signal can be lost.[18] One way to overcome this difficulty is to use the Short-time Fourier Transform, which applies the Fourier transform to successive finite portions of the data.[18] An alternative is to use the wavelet transform which, instead of decomposing a signal into a series of sinusoidal waves, decomposes the waves into a sum of wavelets.[5] The wavelet transform is widely used in signal processing when the signal of interest has locally important information. Since wavelets can have finite supports, and are guaranteed to have finite energies, they are more suited to capturing sharp variations. This is due to the fact that these sharp transitions may be described by the relatively small number of wavelets whose support contains the transition. In contrast, the sine and cosine waves of the Fourier transform support the entire number line and therefore it is difficult to isolate a sharp variation by scaling the waves without losing performance elsewhere in the approximation.

Approximating a function $f(t)$ can be done through the use of scaling basis functions (ϕ) which capture the average value of the function by a combination of scaling and translation of the basis function. If we let j be the resolution of our approximation and k be the integer translation then the scaling basis functions (ϕ) are defined as

$$\phi_{j_0,k}(t) = 2^{j/2}\phi(2^j t - k) \tag{1.2}$$

Wavelet basis functions (ψ) complement scaling basis functions by capturing sharp jumps or abrupt transitions in $f(t)$. The approximation and detail capturing capabilities of wavelets is shown in the expansion of $f(t)$ below in a wavelet basis

$$f(t) = \sum_k \alpha_{j_0,k}\phi_{j_0,k}(t) + \sum_{j_0 \leq j} \sum_k \beta_{j,k}\psi_{j,k}(t) \tag{1.3}$$

The remaining question is how to calculate the α and β coefficients. Since the wavelets we work with form an orthogonal basis for$L^2(\mathbb{R})$ or the set of square integrable functions, the coefficients may be readily obtained via the inner product of the function with the relevant scaling basis or wavelet basis functions.[19]This lets us approximate the function to an arbitrarily fine degree of accuracy by varying j. For practical purposes, exactly describing the original signal using a very high resolution is not necessary. In fact, the coefficients of the highest resolution wavelets are almost always dominated by noise so, when used in signal processing, thresholding the wavelets at a lower resolution improves the signal to noise ratio.

When actually computing the β coefficients it is possible to start with the scaling basis function for the j corresponding to the highest resolution, and then applying a high-pass filter to this function we may obtain both the scaling basis function and the wavelet basis function for the next highest resolution.[5] In this manner it is possible to recursively compute all of the coefficients using a bank of filters.

1.5 (Parametric) Density Estimation:

At its core, density estimation attempts to approximate the probability density function that of a particular random variable. This is a vital component in describing a set of data, as a density estimate is a readily visualized and understood statistical measure which provides a large quantity of information about the data studied.[27] Though important, determining how to approximate the unknown distribution can be difficult. Broadly speaking, we can divide the approaches to tackling this challenge into parametric and nonparametric models.

The parametric modeling paradigm assumes the data comes from some set type of distribution (commonly a Gaussian). From there, finding the best fitting parameters using maximum likelihood estimation is typically straightforward.[10] The parametric approach is effective when there is a certain amount of prior knowledge regarding the distribution of the data being studied, however, when the distribution of the data is truly unknown we would prefer not to force our model into a certain class of distributions.

1.6 Nonparametric Density Estimation:

In contrast to parametric models, non-parametric models limit the assumptions which are made about the data's true distribution function. Typically, they have a parameter which controls the smoothing of the density estimate. Regardless of the model type selected, determining how much to smooth the estimate is an extremely important question.

Histograms are arguably the simplest nonparametric density estimator. The essential idea is to segment one dimensional data into a series of 'bins' of a given width and then assign the data into the bin which it falls. Then the histogram is constructed by creating a series of rectangles with width of the bin size and whose height is proportional to the number of samples which fall into the bin (see Figure 1.1). The size of the bin determines the amount of smoothing to perform. Large bins are insensitive to both noise and fluctuations in the density.

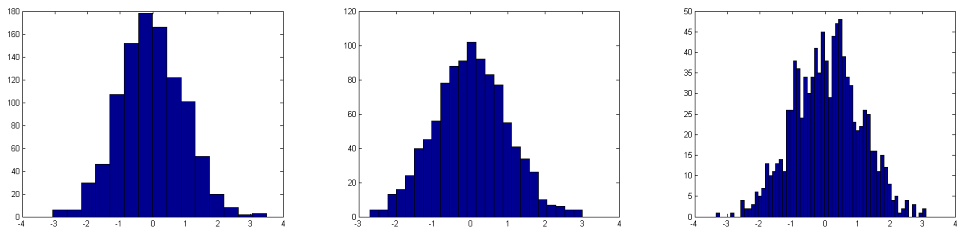


Figure 1.1: Histograms for normal data with: 15, 24 and 60 bins

A potential downside for histograms is their discontinuous nature. While it is possible that the true density function is discontinuous, we generally expect a smoother generating function. Kernel estimators are one non-parametric density estimator which solves this problem by giving us a smooth estimate. The basic idea of kernel estimators can be described by the equation below

$$\hat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K(x - x_i)$$

In the above equation, $K(*)$ is the kernel. A kernel is a function which determines the relative weight to give to a sample. Typical kernels are symmetric, integrate to one, and are non-zero. Examples include the uniform kernel defined as $K(u) = \frac{1}{2}1_{\{|u|\leq 1\}}$ and the Gaussian kernel defined by $K(u) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}u^2}$. Typically a Gaussian kernel is used, although other kernels may sometimes be appropriate. The parameter h controls the neighborhood size to be used in the kernel estimate, and thus determines the smoothing (see Figure 1.2). A large h works like a large bin size, decreasing noise sensitivity and reducing the responsiveness to local changes in the distribution function.

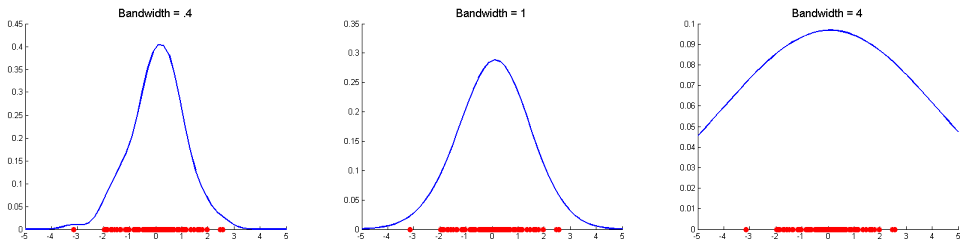


Figure 1.2: Kernel density estimates for normal data with h = .4, 1, and 4

1.7 Orthogonal Series as Density Estimators:

The class of orthogonal series estimators is typically associated with the Fourier series, which approximates data with a series of sinusoidal waves. As touched upon previously, the Fourier transform and other estimators based on global basis functions are unable to capture local aspects of the data distribution. Wavelet estimators, on the other hand, excel at representing local features such as discontinuities.[33]

In order to find the wavelet series expansion for a density function, we must compute the coefficients for its basis functions. In order to compute these coefficients, it is useful to recall that a density function is by definition square integrable since the total probability must be one and the function is never negative. This puts density functions into $L^2(\mathbb{R})$, which implies that they may be expressed as a sum of orthogonal basis functions of the form:

$$f(x) = \sum_j b_j \psi_j(x), \quad j \in J$$

In this case, J consists of the resolution levels of the basis functions. Recall that the exact value of $b_j = \langle f, b_j \rangle$. While $f(x)$ is not known, Cencov[7] describes how to use the expectation of ψ to approximate b_j as

$$\hat{b}_j = \int \psi_j(x) f(x) dx = E[\psi_j(x)] \approx \frac{1}{n} \sum_{i=0}^{n} \psi_j(X_i)$$

Once the coefficients are known, the estimate of the corresponding density can be readily calculated as

$$\hat{f}(x) = \sum_j b_j \psi_j(x) \tag{1.4}$$

These orthogonal series estimates, while possessing many beneficial properties in density estimation such as expressing local trends and allowing for limited support, do not generally produce proper probability density functions. At times this is acceptable as we are chiefly interested in the shape of the distribution rather than the actual probability in an interval, but there are times when negative probabilities in a region may be more problematic. The author's of[16] describe an iterative algorithm which forces a density estimator to give true probability density functions. Convergence is rapid so generally a small number of iterations suffice to give a function which both resembles the estimate from the wavelet transform and acts as a proper density function.

A major difficulty in creating a good approximation of the true density is selecting the resolution level j. Higher resolutions give a density distribution which is more responsive to local fluctuations but which is also more susceptible to noise. The ideal resolution balances these two constraints by smoothing out noise without losing features of the density distribution. A generalized cross validation method for wavelet resolution thresholding applicable to the case of stationary density distributions has been developed which operates in O(n) time for each resolution to compare.[21]

1.8 Thresholding of Wavelet Coefficients

When expanding a function in a wavelet basis, it can be seen that only a finite number of coefficients need to be noticeably non-zero in order to achieve a good approximation. As a result of this, we expect a few relatively large coefficients while the majority are set to zero when analyzing the true function. However, noise in the sample will almost always slightly increase the absolute value of the coefficients, which should be zero. The end result is that the approximation of a function is often improved by thresholding the values of the coefficients, that is to say, either zeroing or shrinking the value of any very small coefficients. There are three chief methods of thresholding: local, global, and block.

In local thresholding, individual coefficients are thresholded. The thresholding function can be hard, where the coefficients are set to zero if they are under a predetermined value t, or soft, where the coefficients all have their absolute value shrunk by a fixed amount t. In either case, the difficulty lies with picking a good threshold value t. While the optimal t is unknown unless the true function is known, it has been shown that t should be proportional to $\sqrt{\frac{log(n)}{n}}$.[20] Empirically, a common choice is to use a value of t which preserves 90% of the function's energy or a value equal to 60% of the largest coefficient. For a data driven approach, see[21] for a generalized cross validation method for wavelet resolution thresholding in the case of stationary density distributions. Other thresholding techniques such as SureShrink[12] and BayesShrink[8] have also been proposed for the stationary case. Rigorous theoretically grounded methods for optimizing the resolution given a non-stationary distribution have not been proposed to the extent of our knowledge.

In global thresholding, entire j-levels of coefficients are thresholded together. The basic idea of hard and soft thresholding is kept and likewise a threshold value needs to be specified. The computation of the optimal threshold is highly demanding and has not been generalized to deal with non-stationary generating functions. Owing to these considerations, we will not go into more depth on the method but point the reader to[13] for further information. However, it should be noted as a point in the technique's favor that it is not necessary to specify any parameters when using global thresholding, making the approach entirely data driven.

1.9 Data Streams:

Ordinarily, data is considered to be gathered at one point in time and then analyzed at another. In this fixed static data paradigm, we have complete access to the entirety of the data set to analyze at any point in time. Furthermore, the amount of data in the set is ordinarily sufficiently small to analyze together without major computational difficulties occurring. In contrast, when analyzing a data stream it is frequently desirable to take new data points into the model as they arrive in real time using online processing.[1] The total data set can be unwieldy in size, so typically only a subset is analyzed at a given point in time. Data streams thus present difficulties both in terms of memory management owing to their potentially unlimited size and the necessity for algorithmic efficiency since a new estimate must be ready before the next set of data comes in. Wegman and Caudle[34] describe a recursive method of updating the wavelet coefficients for a density estimate with computation time scaling linearly with the number of samples read from the stream and $O(1)$ memory requirements. The constant memory requirement comes from the algorithm's single use of a given data sample. As a result, the only values which need to be stored long-term are the coefficients for the wavelet functions, which is a constant amount of memory.

In addition to the above complexities, data streams bring up the question of whether we should view the distribution function as stationary[27],[33] or non-stationary[34].[17] Put another way, data streams let us consider whether or not the distribution of data might change over time. In order to capture the temporally local nature of the non-stationary distribution, it is necessary to somehow discount older samples versus new samples.[34] Wegman and Caudle (2006) addressed this problem via a parameter θ which controls the speed with which the wavelet coefficients decay. Roughly speaking, this is an exponential discounting scheme where a larger θ produces a more precise density function and a smaller θ creates a model more sensitive to local changes in the density function. In contrast to this smooth discounting scheme, García-Treviño and Barria[17] describe a discontinuous scheme using a sliding window of ω samples to calculate the coefficients for both scaling and wavelet functions. Similarly to Wegman and Caudle's θ, ω gives precise answers when large and is sensitive to local changes when small. The advantages claimed for the windowing technique are chiefly computational speed and the capability to give equal weight to all data points in a region of time. The coefficient update equations for the two methods moving from time step n to step $n+1$ are given in 1.5 and 1.6.

$$\hat{b}_{j,n+1} = \theta \hat{b}_{j,n} + (1-\theta)\varphi_j(X_{n+1}) \quad j,n \in \mathbb{N} \tag{1.5}$$

$$\hat{b}_{j,n+1} = \hat{b}_{j,n} + \frac{\varphi_j(X_{n+1})}{\omega} - \frac{\varphi_j(X_{n-\omega+1})}{\omega} \quad j,n \in \mathbb{N} \tag{1.6}$$

1.10 Multiresolution Analysis

Wavelets can be introduced in two major ways: one is through the continous wavelet transform and another is through multiresolution analysis.[22] The concept of muliresolution analysis was first introduced by Stephane Mallat and Yves Meyer in the late 80's and was first applied to image processing.[25] In image processing, images can be analyzed at different resolutions with each higher resolution providing finer details on the image being analyzed. It is not possible to analyze an image completely at only one predefined resolution; therefore, one can reorganize the image information into sets of image details appearing at different resolutions.[25] With this setup, an image analyzed at a resolution $j+1$, $j \in \mathbb{Z}$,contains more details than the same image analyzed at resolution j. The details of the image at resolution $j+1$ are therefore defined as the difference of information between the approximation for the image at resolution $j+1$and its approximation at the lower resolution j.[25]Usually, in the analysis of an image, one would start the analysis at a coarse resolution j_0, which is a resolution level that provides the overall structure of the image, and then incrementally change the resolution to obtain finer details.

Given this setup, Mallat introduced orthogonal wavelet bases in the context of multiresolution analysis, as a direct sum decomposition of the $L^2(\mathbb{R})$ into a sequence of closed subspaces $\{V_j, j \in \mathbb{Z}\}$ such that:

$$(i) \quad \forall j \in \mathbb{Z}, V_j \subset V_{j+1}$$

In other terms, $\cdots \subset V_{-1} \subset V_0 \subset V_1 \subset \cdots$.

The subspaces V_j are embedded within each other such that if $f(x) \in V_j$, then $f(2x) \in V_{j+1}$. In the example of image processing, if one starts the analysis in V_0, moving to V_1will yield more details of the analyzed image or finer features of the image. On the other hand, moving to V_{-1}will yield broader features of the image being analyzed. Similarly, in a time signal, the high frequency content of the signal can be obtained at higher resolutions whereas low frequency content can be obtained at lower resolutions.

$$(ii)] \quad \bigcup_{j=-\infty}^{\infty} V_j = is\ dense\ in\ L^2(\mathbb{R})$$

This means that any function in $L^2(\mathbb{R})$ can be completely represented using the bases our the V_m subpaces.

$$(iii) \quad \bigcap_{j \in \mathbb{Z}} V_j = \{0\}$$

This property suggests that given a resolution level n and a subspace V_n, the signal content at this resolution n does not intersect with the content at the next resolution, or simply, the signal content at resolution $n+1$ is hidden to the content at resolution n.

$$(iv)$$

There exists a function $\phi \in V_0$ with $\phi_{0,n} = \phi(x-n)$ such that the family $\{\phi(x-k),\ k \in \mathbb{Z}\}$is an orthogonal basis for V_0 and $||f||^2 = \int_{-\infty}^{\infty} |f(x)|^2 dx = \sum |\langle f(x),\ \phi_{0,n}(x)|^2$.

If

$$\phi_{j,k}(x) = 2^{j/2}\phi(2^j x - k),$$

then the familty $\{\phi_{j,k}(x) = 2^{j/2}\phi(2^j x - k\}$ is an orthogonal basis for V_j and ϕ is called *father wavelet*. The decomposition of the subspace V_{J+1} is given as

$$(v) \quad V_{j+1} = V_j \oplus W_j,\ j \in \mathbb{Z}$$

where W_j denotes the closure of the linear span of $\{\varphi_{j,k} :\ k \in \mathbb{Z}\}$ with $\varphi_{j,k}$ representing any wavelet function.

If we let A_j be the operator that approximates a function f at a resolution level j, then $A_j$is the orthogonal projection operator onto the space V_j such that

$$A_j f = \sum_j C_{j,k}\phi_{j,k}$$

where the coeffcients $C_{j,k}$ are the coefficients of the dilated and translated wavelet bases at resolution j. The coefficients $C_{j,k}$ are obtained by projecting the function f onto the wavelet bases $\phi_{j,k}$. Therefore, $C_{j,k} = \langle f,\ \phi_{j,k}\rangle$.

At each resolution j we can find the approximation $\hat{f}_j$ of f as given by $\hat{f}_j = A_j f$. At a higher resolution $j+1$, the approximation of f is obtained by adding to $\hat{f}_j$ some additional details about f. These details can be modeled at resolution j by W_j, the orthogonal complement of V_j in V_{j+1}.[33] Therefore,

$$V_j = \bigoplus_{-\infty}^{j-1} W_j \tag{1.7}$$

Then

$$L^2(\mathbb{R}) = \bigoplus_{j\in\mathbb{Z}} W_j = V_{j_0} \oplus \ \bigoplus W_j$$

The projection of f onto the V_0 space is given by

$$\sum_k C_{j_0,k}\phi_{j_0,k}(x)$$

which provides the approximation of the function f at the coarse resolution j_0 with $\phi_{j_0,k}(x) = 2^{j_0/2}\phi(2^{j_0}x - k)$.

Given the functions $\psi_{j,k} = 2^{j/2}\psi(2^j x - k)$ such that the family $\{\psi_{j,k}(x),\ j,k \in \mathbb{Z}\}$ is an orthogonal basis for W_j then the function

$$f(x) = \sum_k C_{j_o,k}\phi_{j_0,k}(x) \quad + \quad \sum_{j\ge j_0}\sum_k D_{j,k}\psi_{j,k}(x)$$

1.11 Comparison of Density Estimators for Streaming Data

Early work on density estimation over data streams has concentrated on kernel methods.[3,6,24,35] The main advantages of kernel estimators is that the only parameter which needs to be specified, the bandwidth, is readily understandable and that the estimator is already extremely well-understood in the stationary case. A major problem with this class of estimators is that they fail to properly capture the features of heavy-tailed distributions common to stock and risk analysis, although some progress has been made in this area when dealing with batch data sets.[4] However, even in the stationary case kernel estimators struggle with sharp discontinuities in the density function. Another, much more rarely used, technique for density estimation is the dynamic histogram[32] which suffers from discontinuities at bin changes .

In nonparametric estimation of streaming data, one can use batch based or online processing of the data. The batch approach chunks the stream into blocks and waits until receiving an entire block before updating the estimate.[2] The online approach allows for a new estimate of the density to be obtained when each new sample comes in, or whenever desired. Online wavelet based estimators have been theorized and implemented by[17,34] but are limited to single dimensional data. Furthermore, they do not propose sound methods for selecting how rapidly to age the data. To the extent of our knowledge, we describe the first implementation of a multi-dimensional wavelet based density estimator for streaming data and a novel heuristic for selecting its aging parameter. Since the vast majority of data in the world is multi-dimensional, this presents a step forward in modeling complex, time dependent distributions.

2. MATLAB PROTOTYPES

In coding the 1-dimensional streaming data density estimator we used Mark Moyou and Eddy Ihou's architecture, illustrated in 2.1. The 2-dimensional prototype kept this architecture and file structure, only incorporating some minor optimizations to making working in higher dimensions more manageable.

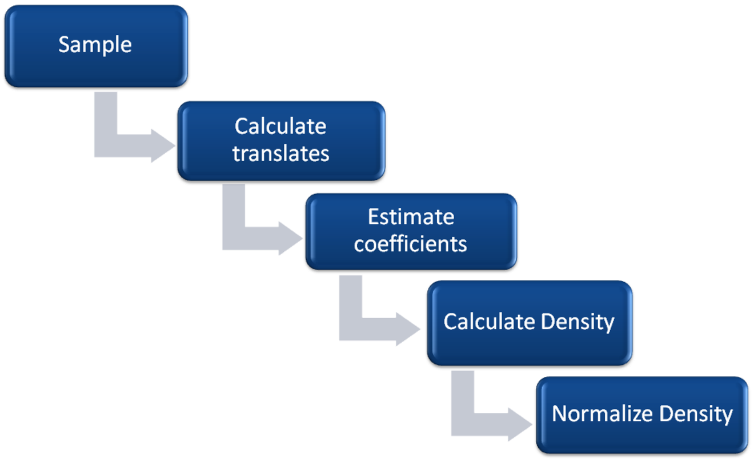


Figure 2.1: Density Estimation Process

2.1 Obtain Samples

We assume that data has been collected previously and processed into a set with uniform ranges across dimensions. It is likewise assumed that the maximum and minimum range for the data (or at least the desired support) is known.

2.2 Calculate Valid translates

The dyadic Discrete Wavelet Transform (DWT) assumes wavelet functions (father and mother) of the form:

$$\varphi_{j,k}(x) = 2^{j/2}\varphi(2^j x - k)$$

where $j, k \in \mathbb{Z}$ and φ refers to both the father and the mother wavelet functions.

Also, the majority of the wavelet families used in this implementation of the density estimator have a defined support of $[0, 2N-1]$ where N is the order of the wavelet. For example, "$db6$" (Daubechies wavelet family) is of order 6 and therefore its support is $[0, 11]$. For the wavelet function $\varphi_{j,k}(x)$ to be evaluated at a given point, its argument $2^j x - k$ has to lie within its support. That is:

$$0 \leq 2^j x - k \leq 2N - 1$$

Solving for k, we obtain:

$$2^j x - 2N + 1 \leq k \leq 2^j x$$

Therefore, if the support of the density function is given as $[a, b]$ and the order of the wavelet N is known, the valid translates k will meet the condition

$$\lceil 2^j a \rceil - 2N + 1 \leq k \leq \lfloor 2^j b \rfloor \tag{2.1}$$

2.3 Estimate Coefficients

The bulk of the coding complexity falls into this section. All the coefficients are initialized at zero and then begin to build up from there as described in equation (1.5) and (1.6). We support two methods for updating the coefficients: the sliding window method[17] as well as the decaying method.[34]

The sliding window method considers only a finite section of the total data set at a time in order to estimate the density function. When a data point leaves the data set, it subtracts from the coefficients the exact amount which it originally added to them. Given the coefficient at the previous time step $\hat{b}_{j,n}$, the window size w, and both the entering (X_{n+1}) and leaving (X_{n-w+1}) data points, the coefficient at the next step is readily calculated by:

$$\hat{b}_{j,n+1} = \hat{b}_{j,n} + \frac{\varphi(X_{n+1})}{w} - \frac{\varphi(X_{n-w+1})}{w}$$

The decaying method of Caudle and Wegman[34] likewise lends itself well to a simple recursive updating procedure. Specifying a decay rate 0 ¡ θ ¡ 1, the coefficient at the next time step is given by:

$$\hat{b}_{j,n+1} = \theta\hat{b}_{j,n} + (1-\theta)\varphi(X_{n+1})$$

In both of the recursive updating schemes, it is not necessary to actually calculate $\varphi_{j,k}(X_{n+1})$ for many of the basis functions. This is since whenever X_{n+1} falls outside of the support of the function, $\varphi_{j,k}(X_{n+1})$ is guaranteed to be equal to zero. Thus, by simply checking whether or not a point is supported by a basis function before evaluating it we save significant amounts of computation time.[17] It is also worth noting that for the majority of the wavelets used in estimation, no closed form approximation exists. Due to this, a table of values defining the wavelet over its support is pre-emptively calculated for each type of wavelet. Then interpolation was used among these values to calculate the output of the scaling basis and wavelet basis functions at a particular value. Profiling of the algorithm's performance puts this interpolation (which must be performed here as well as in density calculation) as the bottleneck where most of computation time is spent. Depending on the granularity of the wavelet table, linear interpolation may give satisfactory results while for more widly spaced tables cubic interpolation may be necessary to achieve a good approximation of the interpolated value.

2.4 Calculate Density

The computational bottleneck lies with this section. The actual calculation of the density is fairly simple. For each point x in the support, the density is given by:

$$f(x) = \sum_k c_{j_0,k}\phi_{j_0,k}(x) + \sum_{j_0 \leq j}\sum_k d_{j,k}\psi_{j,k}(x)$$

As it is plainly infeasible to calculate the density at every point in the support, a discretization for how frequently to calculate the density is selected. Empirical tests give that the computational load for updating the coefficients given one new data point is roughly equal to that for calculating the density of six points. A finely discretized density function frequently evaluated can drastically slow down the algorithm. Likewise, having a large discretization and reducing the frequency at which the entire density is calculated both significantly speed up the algorithm.

2.5 Normalize Density

While the above calculations will give the proper shape for the density, they do not yet create a true probability density function. Regions sparse in data may have negative densities and the integral over the region almost certainly does not sum to unity. In order to correct these types of errors, we use the iterative method for improving density methods described in.[16] The essence of the method is to first set all negative values in the density to zero, then force the function to integrate to one, and continue to repeat until the difference between the integral and unity is less than a prespecified ε (see Figure 1). To be more precise, the algorithm is defined as follows where f_k is the kth approximation for the true density function and $h(x)$ is the weight function at x. A uniform weight function was used in our implementation.

Algorithm 1 Normalization of Density Estimate

```
$f_0(\cdot, x^n) = \hat{f}(\cdot, x^n), \quad k = 0$
$f_1(\cdot, x^n) = max(0, f_k(\cdot, x^n)), \quad k = 1$
$C_1 = \int_S f_{k+1}(t)dt$

while $|C_k - 1| > \epsilon$
    $f_{k+1}(\cdot, x^n) = f_k(\cdot, x^n) - (C_k - 1)/[h(\cdot) \int_S 1/h(t)dt$
    $f_{k+2}(\cdot, x^n) = max(0, f_{k+1}(\cdot, x^n))$
    $C_{k+2} = \int_S f_{k+2}(t)dt$
    $k = k + 2$
```

2.6 Extension to Higher Dimensional Data

In extending the concept of wavelet analysis to higher dimensions, it is useful to recall$\mathbf{V}_j$ which is defined as the space of functions which can be expressed by wavelets at the jth resolution level. Wavelet analysis can then be extended to m-dimensions via defining the m-dimensional vector space $\mathbf{V}_j^m$ as the tensor power of the one dimensional $\mathbf{V}_j$. The scaling basis functions in m dimensional space are then given as:

$$\phi_{j,k_1,k_2...k_m}(x_1, x_2, ..., x_m) = \phi_{j,k_1}(x_1)\phi_{j,k_2}(x_2)...\phi_{j,k_m}(x_m) \tag{2.2}$$

From this we can see that it is necessary to have a coefficient $b_{j,k_1,k_2...k_m}$ defined across each translate for each dimension of our data. Thus, in the 1-dimensional case we needed a vector for the coefficients of the scaling basis functions, in the 2-dimensional case we need a matrix for those coefficients, and so on in higher dimensions. Also important to note, while in the 1-dimensional case we have a ψ and a ϕ function, with 2-dimensional wavelets we have the tensor product combinations of the 1D functions $\phi_1\phi_2$, $\phi_1\psi_2$, $\psi_1\phi_2$, and $\psi_1\psi_2$.

In the 2-dimensional Matlab prototype, the algorithms discussed work in the same manner as previously explained for the 1-dimensional case, with the minor change that we add another loop across the translates in the second dimension. Furthermore, an early implementation of the prototype spent virtually all of its time calculating the densities once the coefficients were known. In particular, vast amounts of time were spent interpolating amongst the φ tables (φ represents both scaling and wavelet basis function tables). Since the meshgrid across which we define our density approximation does not change, we can initially calculate all of these interpolations which we need given the support range and discretization (merely looking up these calculations in the future). This resulted in speedup of several orders of magnitude, and plotting the density function for 2-dimensional data now occurs close to interactive speeds. An example of a 2D density plot is shown in Figure 2.2

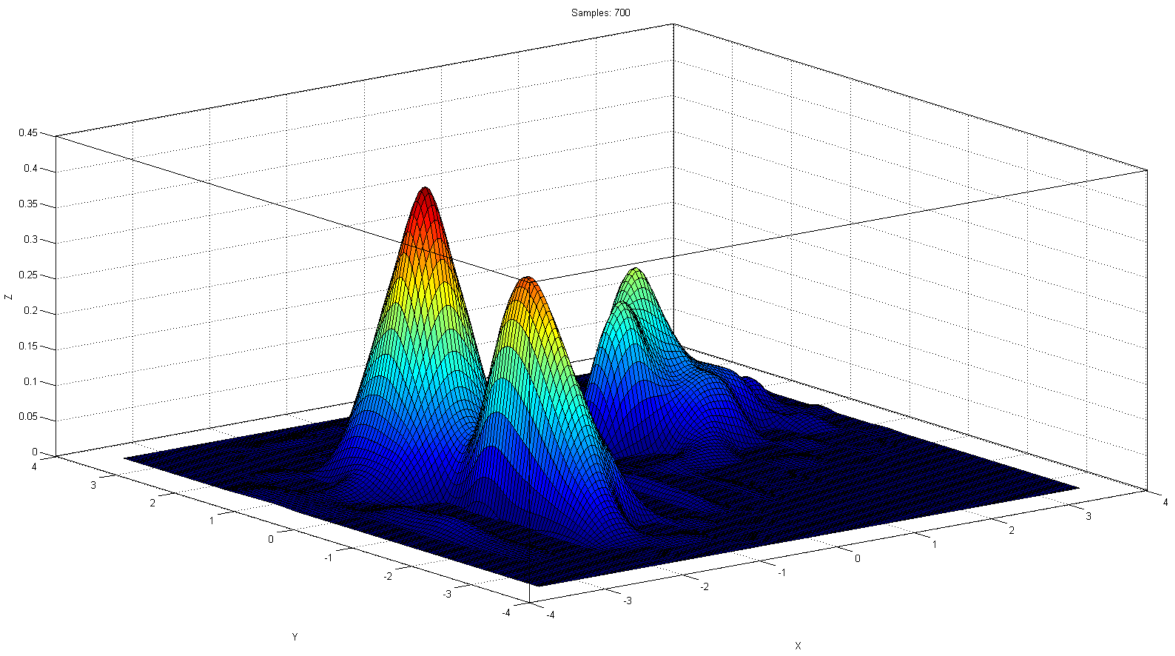


Figure 2.2: Matlab 2-D Prototype performance at approximating a tri-modal Gaussian mixture model

The extension into 3-dimensional data made visualization extremely difficult. The choice was made to use isosurfaces which display a 3-dimensional region corresponding to a given density. However, the isosurfaces for different densities may vary significantly and it is difficult to predict the nature of this variability. Furthermore, this density is allowed to change throughout time. In the end, we decided to allow the user to specify the value of the isocontour to display at each update period. An example of the display is shown in Figure 2.3. All isosurfaces of a 3-dimensional Gaussian should be spheres, and the estimator succeeds on this front.

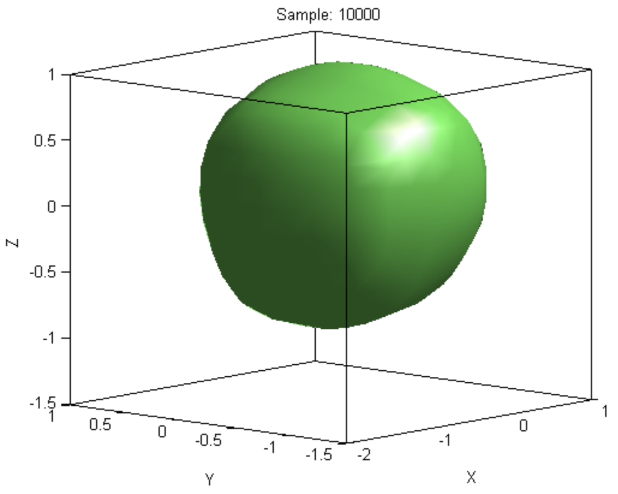


Figure 2.3: Isosurface of a 3-dimensional uncorrelated Gaussian

3. 1-DIMENSIONAL JAVA APPLET

3.1 Architecture

With the 1D Matlab implementation explained, the next phase of the project is to implement 1D wavelet framework in a Java applet. In order to accomplish this we provide an overarching outline of the necessary tasks which the code needs to perform, shown in Fig.3.1.

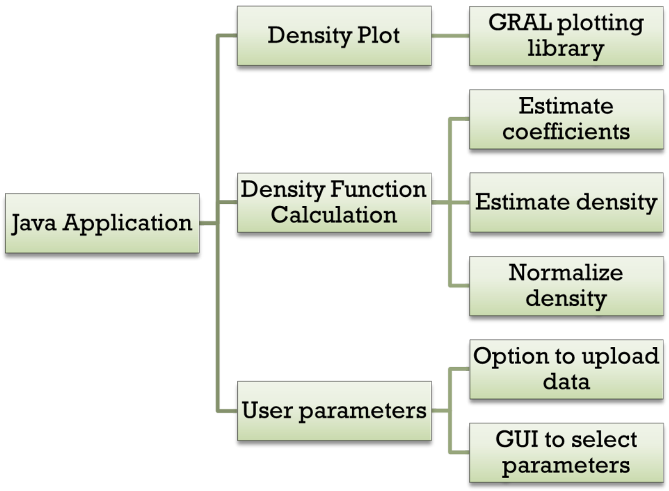


Figure 3.1: Java applet architecture

These different functionalities were divided up among classes as was thought logical.All code is freely available on Github via this link.

3.1.1 Settings

This class acts as a wrapper for all of the user specified values. When the user selects a wavelet type, resolution, etc using the GUI, this automatically updates the appropriate values in this class. All fields are public and so may be accessed by any other class. The fields include

- startLevel and stopLevel: the starting and stopping resolution levels for the estimator
- waveletType: a string referring to the wavelet family and order (if applicable)
- agingFlag: the type of aging to be used on the data (options are windowing, exponential decay, and none)
- discretization: how widely to space the points over which to calculate the density
- updateFrequency: how many samples are read in between updating the plot
- windowSize: when using the windowing aging process, determines the size of the window
- agingTheta: when using the exponential decay aging process, determines the speed of the aging
- densityRange: the minimum and maximum values over which the density is defined
- waveletFlag: whether or not to use wavelets in addition to the scaling basis functions

3.1.2 EstimatorGUI

This class forms the actual display for the density estimator. Normally, it has a simple toolbar (with 'Start', 'Pause', 'Reset', and 'Settings') at the top and most of the space is devoted to plot for the probability density function estimate. Settings may only be edited once the estimating process has been stopped. Selecting settings will open up the SettingsUI.

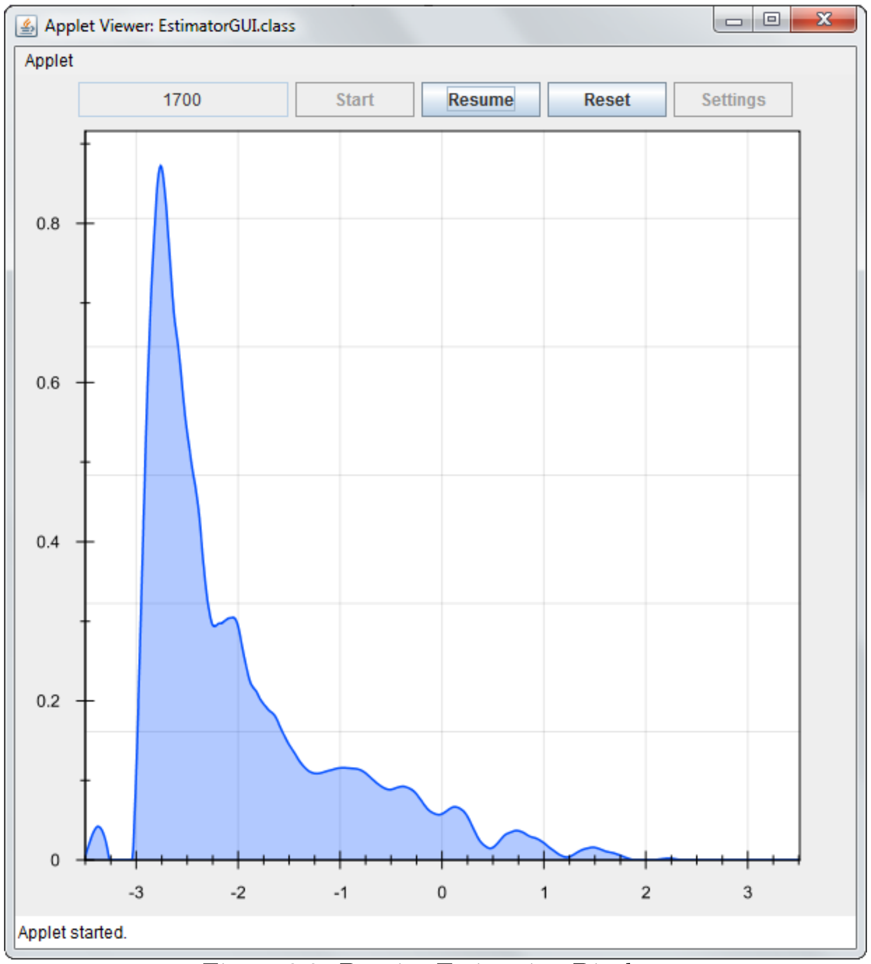


Figure 3.2: Density Estimation Display

The actual display of the plot uses the GRAL (GRAphing Library) Java package, from http://trac.erichseifert.de/gral/ The plot supports zooming and scrolling while the estimation process is paused. One advantage of the GRAL package is that their plots are implemented with data listeners. In essence the data listeners work so that, when the table corresponding to the density at each point in the domain is updated, its plot is automatically updated.

Pausing and unpausing can be done freely during the density estimation process. During density estimation, the current index of the sample being evaluated is displayed above the plot so that the current progress is known as shown in Fig 3.2.

3.1.3 Transform

This class wraps the coefficients and translates. It has no further functionality.

3.1.4 Wavelet

This class performs the functionality of a wavelet. We operate under the assumption that the data point has already been scaled and translated as appropriate for the corresponding wavelet's resolution and translation.

Initializing a wavelet loads the proper table of ϕ and ψ values from csv files. Once initialized, the inSupport function determines whether or not an incoming data point is supported by the wavelet. Furthermore, the class supports methods to return the value of ϕ and ψ at a given point in its support. Interpolation between values in the support is done linearly.

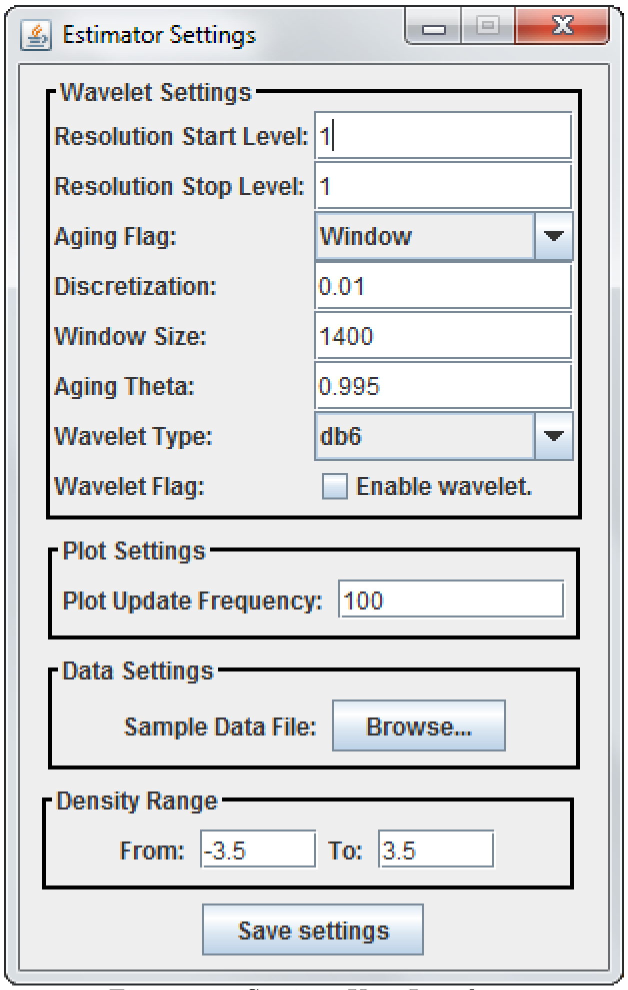
3.1.5 DensityHelper

This class performs the bulk of the actual density estimation algorithm. It is responsible for initializing and updating the coefficients appropriately, and creating a normalized density estimate when one is requested. The current implementation is loop based and we are exploring whether vectorization would offer significant speedups.

First, the valid translates given the resolution, density domain, and wavelet type used are calculated (initializeTranslates method). Next the coefficients for both the scaling basis functions and wavelet basis functions must be initialized to zero (initializeCoefficients).

During the actual online density estimation process, (updateCoefficients). In order to save computational time, only those coefficients whose support actually contains the incoming data point are updated in accordance with the recursive updating procedure outlined by equations (1.5) and (1.6). Once the coefficients are computed, the unnormalized density is calculated in getDensity followed by normalization. Normalization is performed iteratively according to the algorithm proposed by Leslaw Gajek[16] in the function normalizeDensity.

3.1.6 SettingsUI


Estimator Settings

Wavelet Settings

Resolution Start Level: 1

Resolution Stop Level: 1

Aging Flag: Window

Discretization: 0.01

Window Size: 1400

Aging Theta: 0.995

Wavelet Type: db6

Wavelet Flag: Enable wavelet.

Plot Settings

Plot Update Frequency: 100

Data Settings

Sample Data File: Browse...

Density Range

From: -3.5 To: 3.5

Save settings

Figure 3.3: Settings User Interface

This class allows the end user to easily select the parameters controlling the density estimation process. These are the settings referenced in subsection 3.1.1. It also allows the user to select a data file of their own to upload and use.

3.1.7 DensityRunner

The density estimator is built using JAVA Swing components. It turns out that all Swing applications have a single thread, called the *event dispatch thread*, that manages all interactions with the application's GUI components. All tasks that require interaction with an application's GUI are placed in an *event queue* and are executed sequentially by the event dispatch thread. Performing intensive calculations while simultaneously updating GUI components can tie up the event dispatch thread in the calculations and consequently make the application to become unresponsive. The solution to this issue is to perform all intensive computations in a separate thread, usually referred to as the *worker thread*, and let the event dispatch thread update the GUI components based on the results obtained from this worker thread.

The *DensityRunner* extends the JAVA SwingWorker class and is a separate thread that performs all the calculations related to density estimation. Once enough results are available for the plot update, the DensityRunner delegates the plot updating task to the event dispatch thread. Splitting the tasks between the DensityRunner thread and the event dispatch thread in this fashion allows the application to stay responsive at all times and to rapidly respond to user interactions such as stopping, resuming, and resetting the density estimator.

In a nutshell, the DensityRunner is responsible for reading sample data from the user-provided sample data file, calculating and updating the coefficients of the wavelet functions, updating the density function, and making the event dispatch thread update the density plot at the user-defined frequency.

3.2 Optimization

The computational bottleneck in the Matlab code was the actual density estimation once the coefficients were known. With this in mind, we are setting the default update frequency in the applet to read more samples between updates to the plot. Even without this modification, the unoptimized Java code already ran significantly faster than the Matlab implementation. An early time trial gave a rate of 1 million samples every four and a half minutes. Settings were as follows: data drawn from a normal distribution (mean 0 and standard deviation 1), resolution level of 2, density range -3.5 to 3.5, discretization of .01, wavelet type Daubechies 6, update frequency 2000 samples, using a window size 1400 aging, and run on an Intel Core 2.8GHz processor.

An early concern was memory management when working with large files. In order to address this concern, a BufferedReader was used along with a single array storing the window of old samples. This should keep the marginal memory requirements of the algorithm linear with respect to the window size selected and independent of the size of the data file being used. Memory will most likely not be a concern, however, as a data file with over 10 million samples can easily fit into working memory.

3.3 Recommendations for use

3.3.1 Data sets

The current default settings work well for normalized data (mean 0 and standard deviation of 1) but are not necessarily limited to such data sets. However, if non-normalized data is to be used, we strongly recommend that the density domain setting be edited in order to properly accommodate the data being used.

3.3.2 Selecting a resolution level

The resolution to select is arguably the most important parameter. Higher resolutions give a density distribution which is more responsive to local fluctuations but which is also more susceptible to noise. While resolution levels are most typically thought of as starting at integer values, the only real requirement is an integer distance between the starting and stopping resolutions. In light of this, we allow the user to select real valued resolutions. For a proof of the orthogonality of arbitrary starting resolution see Appendix A.

3.3.3 Deciding how to age the data

Many data sets are stationary with respect to their distribution function. It is typically a good idea to consider whether or not it makes sense to look at the data as non-stationary before trying to optimize aging parameters. Assuming the data is non-stationary, an important consideration is how to implement aging. If exponential aging like that proposed by Caudle and Wegman[34] is used, the optimal value of θ needs to be found. As discussed by García-Treviño and Barria,[17] selecting the optimal θ can be an exceedingly difficult process and rather opaque to the end user. In general, increasing θ reduces the model's sensitivity to noise and local changes in the underlying density function while decreasing the parameter has the reverse effect. While a value of θ near .999 has seen some success empirically, it is difficult to understand how much to modify the parameter in order fine tune the estimate's sensitivity to both noise and local changes in the nature of the distribution function. In contrast, using the sliding window approach generally feels much more intuitive. Since the technique simply uses a sliding average of ω samples, the user may pick how many samples to average at a time and the tradeoff between noise insensitivity versus rapid adjustment to changes in the distribution function is fairly straightforward. Based on this, we recommend the sliding window approach for simplicity.

3.3.4 Other considerations

Enabling wavelets at a stopping resolution level of J is exactly equivalent to disabling wavelets at a starting resolution of $(J+1)$ in terms of the density distribution function. The wavelet type and order to be selected are relatively unimportant in actually finding the density estimate, although a general rule is that higher orders of a given wavelet family will give smoother estimates. If wavelets are to be used, make sure to select a wavelet type which is dyadic in nature (the Daubechies, Symlet or Coiflet wavelets are all dyadic).

There are several factors which influence the speed of the density estimation process. The discretization level determines how well the plot shown demonstrates the actual calculated density distribution. Reducing the level will increase the speed of the algorithm but may result in a plot which appears to be somewhat piece-wise linear and jagged. The update frequency influences the algorithm's speed both by determining how often the plot is visually updating (repainting the plot can easily begin to dominate computation for very frequent updates) and how often the density must be updated. Broadly speaking, if speed is a concern one should update as rarely as feasible. Finally, higher order wavelets have larger tables defined and thus computing ϕ and ψ will take longer to interpolate. For this reason mid-range Daubechies wavelets are recommended as generally a good balance between smoothness and speed.

4. 2-DIMENSIONAL JAVA APPLET

The Java 2-D implementation of the density estimator extends the methodology used in the 1-D Java implementation. The slight change in the overall structure of the applet is the choice of a different plotting library called Java Surface Plotting library in lieu of the GRAL plotting library as the latter than does not provide surface plotting capabilities. The Java 2-D architecture is shown in figure 4.1 below.

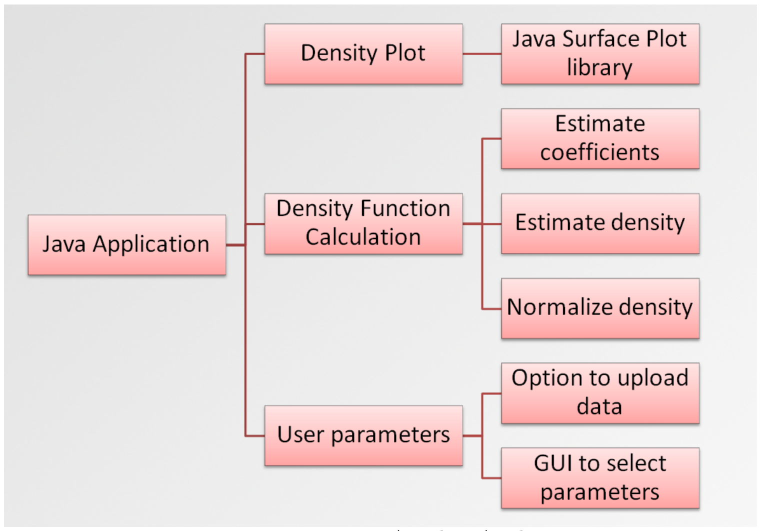


Figure 4.1: Java 2-D Applet Architecture

The essential theory behind this 2-D Java implementation of the density estimator is the extension of wavelet-based density estimator to higher dimensions. This extension to higher dimension is discussed in section 2.6.

All the Java classes used the 2-D Java implementation are structurally similar to those used the 1-D Java implementation with the obvious changes made to satisfy the conditions posed by the higher dimension extension of the wavelet analysis.

5. APPLICATIONS

In this section we see how the estimator performs at describing the potentially non-stationary relationship between several variables of interest in stock market analysis. Other applications of interest include anomaly detection in internet traffic data[29] and change point detection for complex models[23, 28] .

5.1 Stock trading: the variability/volume relationship

Stock traders are constantly seeking more efficient ways to mitigate risks and increase profits on their investments; low risk investments may bring in a small trickle of steady income but have little chance of making large gains. On the other hand, high risk investments are often paired with the potential for huge profits. A properly diversified investment portfolio has some combination of the two types of investments according to the manager's personal concern for massive loss versus above average gain. There are a variety of factors which influence the risk of a given investment. For example, startup companies tend to have a higher chance for failure but also increase their stock prices much more than an established company if they succeed. Other risk factors are time dependent; making trades during a market crash carries much more uncertainty than trading during more normal periods.

One well established risk relationship is that the volume of a stock being traded tends to correlate with that stock's absolute change in price.[31] To put it another way, if a stock is being exchanged at unusually large volumes it is very likely that its price has gone up or down by a significant amount. The result of this is that trading on particularly high volume days entails a great deal of risk.

Two competing theories exist to account for the volume versus variability correlation. In the first, variance of daily price change is viewed as the sum of a random number of intra-day price changes. The variance of daily price change is then correlated with the number of transactions occurring that day.[9] The second model assumes a positive relationship between how much traders disagree when revising their reservation prices and the absolute change in the stock's market price. The volume versus variability relationship springs from the correlation between trading volume and trader disagreement in revising reservation prices.[15]

Regardless of the underlying process for the phenomenon, traders need to keep the volume versus variability relation in mind when examining the risk of trading. Knowledge of how this relationship varies over time could give traders a better sense of the risk they accept when trading on high volume days.

5.1.1 Data and preprocessing

Original data comes from the historical stock data set on Quant Quote (https://quantquote.com/historical-stock-data) on June 9th, 2014. The raw data contained the following variables: the Date, Ticker (which comes from the file name), Open, High, Low, Close, Volume for the day. We use the 50 stocks with most complete information in the Quant Quote set (completeness was decided by the size of the data file for the company). Since this selects for companies which lasted the roughly 14 year period (late 1998 until early 2013), survivorship bias is a potential source of error in our subsequent analyses. To put it another way, it is possible that the following analyses only hold for companies which will not dissolve in the immediate future. Thus, if these techniques were used with modern companies (which may dissolve soon) then different results may be generated.

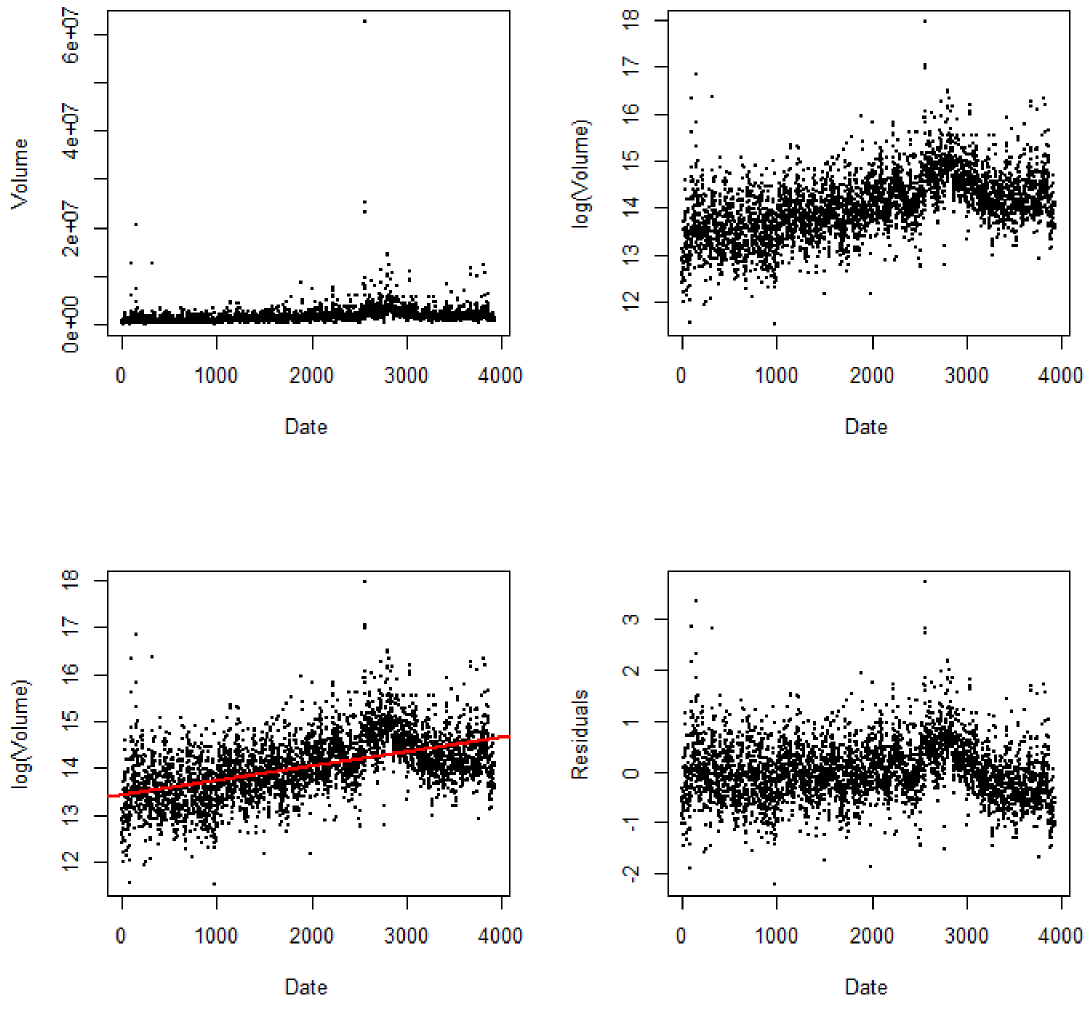


Figure 5.1: Volume Normalization Process ('hum' ticker)

Since the raw price for the various companies was frequently very different, it was necessary to normalize the data by examining the percent price change instead of the original price. Likewise, different stocks are traded at radically varying volumes and these volumes are largely time dependent. In order to account for this, we use the standard technique of taking the logarithm of the individual company data. After this, we fit a linear model to the log trading volume versus time and examine the residuals.[30] This process can be seen in Figure 5.1.

In order to keep the two variables on the same scale for the eventual density estimator, both were recentered about their mean and then divided by their standard deviation. We are, therefore, simply working with the variables in terms of their standard deviations.

5.1.2 A new heuristic for the selection of the resolution level and θ value

The true density distribution over the two variables (trading volume residuals and percent price change) is not known a priori. We do however, believe that the distribution exhibits a certain degree of smoothness. This smoothness can be controlled via the resolution level (j_0) of the estimator. Further, we believe that the distribution does not change instantaneously so that the current density should be indicative of points in the

Evaluation results			
	Resolution level		
	0	1	2
θ			
.995	-520,000	-533,000	-617,000
.999	-511,000	-504,000	-524,000
.9995	-512,000	-502,000	-513,000
.9999	-523,000	-513,000	-514,000

Table 1: Sum log probability results for selected θ and j_0 values.

immediate future. The speed of the estimator's response to changes in the underlying distribution can be controlled by the aging parameter θ. We opted for the exponential decay model rather than the windowing method as it seems reasonable that stock data is most closely related to data immediately previous to it with a slow decay of this relationship over time. The windowing method would assume an equal relation between data points for a fixed period of time after which there is utterly no relation, a rather odd idea for real world data.

We wish to select the 'best' pair (θ, j_0) for our data. We define the 'best' estimator as the one for which the sum of the log probabilities of the incoming points is maximized. Describing the heuristic in more depth, each time we take a point in from the data set and calculate its probability given the current density model, take the log of that value, and then add it to the model's total score. This method can be applied to the entire dataset (as we did) or one can use a subset of the data. The major advantage of using a subset of the data would be speed as the heuristic seems fairly robust to overfitting. Choosing the model with the least negative total score selects for effectiveness at predicting future data. It is possible to run the model with several combinations of the parameters in parallel in order to select the best combination when analyzing real-time streaming data. From Table 1 it can be seen that the optimal parameters for this data set were a j_0 of 1 and a θ of .9995.

5.1.3 Analysis of volume/variability mutual information

Briefly, mutual information is a measure of how much information about one variable is contained in the other.[26] In the continuous domain, the mutual information of two variables may be computed as

$$MI(X;Y) = \int_Y \int_X p(x,y) log(\frac{p(x,y)}{p(x)p(y)}) dx\, dy \tag{5.1}$$

The model parameters selected as optimal by the validation scheme described above was run on the data set while computing the mutual information of the two variables every 100 samples. In other words, we found how much information the stock trading volume gives about the probable price change at a given point in time. Figure 5.2 shows the evolution of this metric over time. At first glance, it seems difficult to extract much useful knowledge from the raw graph. However, the sharpest peak corresponds to the August 2011 United States debt ceiling crisis while the most noticeable trough occurs during the period directly following the September 2008 stock market crash. Other features of the chart may very well correspond to significant periods for the market, but we limit our analysis to highlighting the two most noticeable periods.

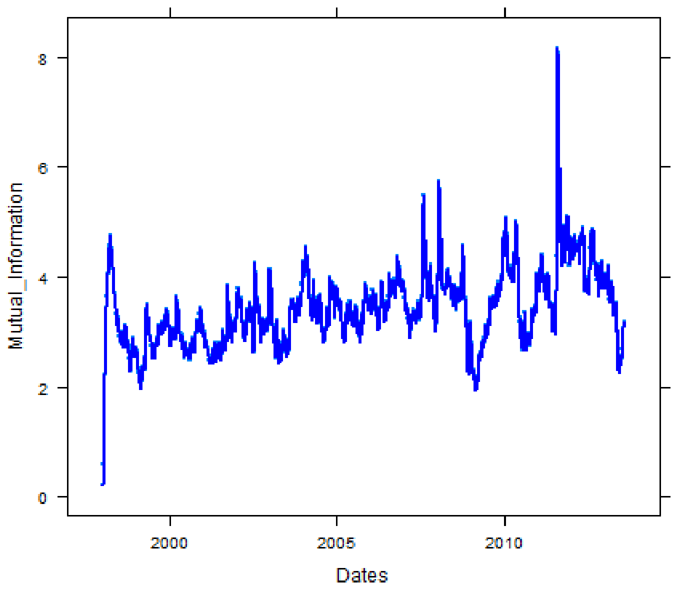


Figure 5.2: Mutual information between stock trading volume and variability over time

Using the same model parameters on randomly distributed, uncorrelated Gaussian data the mutual information index stayed near to 1 once training had occurred as shown by Figure 5.3. The contrast between the two plots is stark, suggesting that a large amount of information about stock price changes can be determined from that stock's trading volume for that day. This agrees with literature reporting a noticeable relationship between stock trading volume and price variability. It is worth noting that the initial spike seen in both plots corresponds to the 'learning' period of the density estimation model and can be safely ignored.

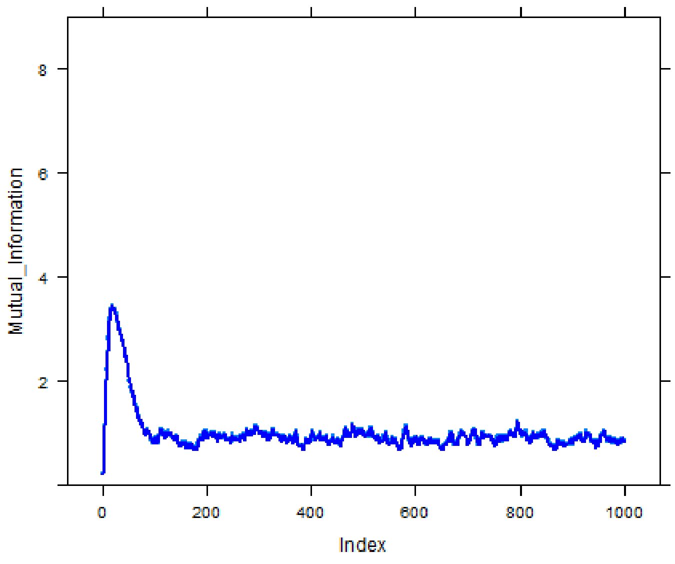


Figure 5.3: Mutual information between Gaussian, uncorrelated variables

5.2 Option trading: the volume and put/call ratio relationship

Stock options are a form of derivative financial instrument which serve to move risk between investors.[11] They are a contract which gives the buyer the option to purchase (in the case of a put) or sell (in the case of a call) a stock at a fixed price at some point in the future. It is worth noting that the decision of whether or not to exercise the option remains in the buyer's hands, so obviously it will only be used if it is profitable to do so. If an option would be profitable to exercise, it is said to be "in the money." Options are extremely high risk, if a put is made and the stock price goes down then the put will be worthless. This is in contrast to a true stock, which will retain most of its worth even during a significant crisis. On the other hand, options present massive opportunities for profit particularly in cases of insider trading.[14]

It is generally expected that the ratio between puts and calls stay relatively close to 1. Large positive (or negative) deviations from this value suggest that the stock's price is expected to increase (decrease) strongly in the future, and if this expectation is widespread it will affect both the price of puts (calls) as well as the price of underlying stock which will work to bring the ratio back into balance. Worth noting, however, is that the ratio will generally be slightly lower than 1 (closer to .93) due to the use of calls acting as short positions to hedge the mostly long positions widely held in the market. For example, if a mutual fund holds a large amount of Apple stock they could reduce their risk exposure by purchasing calls on Apple stock. If Apple's stock price goes up, the fund sees profit through increase in the stock price but the call becomes worthless. On the other hand, if the price of Apple's stock falls then the fund will lose money from the actual stocks they hold but receive profit from selling the call. This shows how puts can reduce overall risk exposure at the cost of reducing expected profit.

5.2.1 Analysis of volume and put/call ratio mutual information

The selection of the resolution level j_0 and the aging parameter θ occurred using the same type of grid search previously described. The optimal levels were selected as $j_0 = 0$ and $\theta = .992$. The option data has far fewer points per period of time when compared to the stock price data. As such, we could predict that if the two distributions change a roughly equal amount over the same period of time then the option data should require a lower θ to take this into account. Further, other researchers[17,34] have noted that the more rapid the aging the lower the resolution must be to compensate. This is due to the fact that a higher resolution level demands a larger data in order to not be dominated by noise and more rapid aging reduces the effective size of the data set the estimator uses at a point in time. The optimal levels chosen for the trading fulfill these expectations.

Using the selected values, we calculated the mutual information (5.1) of the volume and put/call ratio over time. The results can be seen in Figure B.2. Several interesting trends come to mind during the initial observation of the plot. First would be the appearance a repeated up down cycle where each cycle lasts approximately 18 months. We are currently unaware of what financial activity might be causing this cycle but the topic deserves further investigation. Also worth noting is the sharp downward trend in the mutual information towards the present era. This seems to suggest that the relationship which was once held between the two variables is becoming weaker, which could be due to improved market efficiency. The increase in market efficiency is implied since there is information about the relative desire of puts or calls from the option trading volume, then any sellers incorporating this information into their option pricing would display increased profits. To be more concrete, if a seller knows that the current trading volume implies a high put/call ratio then that seller can increase their profit by increasing the price of puts and decreasing the price of calls. This increase in profit should lead to the majority of firms taking these trends into account, which would increase the price of the unusually desirable commodity and indirectly reduce the purchases of that commodity back to the expected ratio. A lower mutual information index could then imply increased market efficiency.

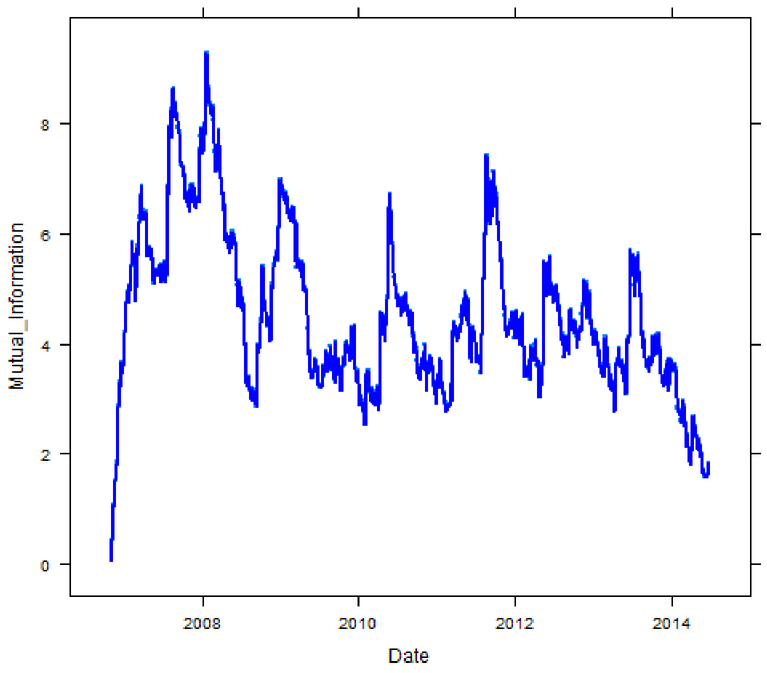


Figure 5.4: Mutual information between option trading volume and put/call ratio

6. CONCLUSIONS

The report covers the steps we took to address the problem of accurate and efficient density estimation for multi-dimensional, streaming data. Since streaming data has becoming increasingly common in applications ranging from stock trading to air pollution monitoring, our solution is highly desirable. The wavelet based approach we take allows for the efficient, online analysis of data from highly complex density distributions. The advantage of using wavelets are that we are able to model the heavy tails and sharp transitions in the density function. The algorithm used differs from batch based approaches in that it allows for the generating distribution to change over time, a major advantage in many applications.

This project follows from a 1-dimensional Matlab prototype of the wavelet based density estimator. From there, we wrapped the algorithm in a Java application in order to take advantage of improved speed due to compilation and make the tool more accessible to data analysts. The next step was a 2-dimensional prototype in Matlab, followed by the extension of the Java application into the second dimension. We also implemented a 3-dimensional prototype in Matlab, but due to the lack of isosurface visualization support in Java (and most other major languages) we did not similarly extend the Java application.

Several data sets were initially considered in testing the application including pollution levels, cyber security attacks, and stock market data. We ultimately chose to use stock market data as the distribution has interesting nonparametric and nonstationary features. This analysis showed the estimator's responsiveness to changes over time, and displayed sharp deviations around major market changes which might be useful in predicting these changes in the future. The research team also wrapped the algorithm in a user friendly Java application. This should vastly simplify density estimation for time dependent data. To date, a rigorous mathematical solution addressing the optimal speed for data aging and accuracy related to resolution level choice has not yet been proposed for non-stationary distributions,but would be an important future research topic. Such a solution would vastly simplify the wavelet density estimation process, which currently relies on user input for those parameters with only heuristics as a guide. Keeping this in mind, we also incorporated a new heuristic for the selection of these important parameters in order to facilitate a more data driven analysis.

APPENDIX A. ORTHOGONALITY OF RESOLUTIONS UNDER ARBITRARY SCALING

We wish to prove that if two basis functions subject to the dyadic wavelet transform with resolution levels $j_1, j_2 \in \mathbb{Z}$ are orthogonal across integer translates, then those functions with new resolution levels $j_1 + m, j_2 + m$ (where $m \in \mathbb{R}$) are as well. Let φ be considered interchangeable for either the father and mother wavelet function. From the orthogonality of the functions, we have

$$\begin{aligned}
0 &= \langle \varphi_{j_1,k_1}, \varphi_{j_2,k_2} \rangle, \forall k \in \mathbb{Z} \\
&= \int_{\mathbb{R}} \varphi_{j_1,k_1}(x)\varphi_{j_2,k_2}(x)dx \\
&= \int_{\mathbb{R}} 2^{j_1/2}\varphi(2^{j_1}x - k_1)2^{j_2/2}\varphi(2^{j_2}x - k_2)dx \\
&= 2^{\frac{j_1+j_2}{2}} \int_{\mathbb{R}} \varphi(2^{j_1}x - k_1)\varphi(2^{j_2}x - k_2)dx
\end{aligned}$$

Now we have a change of variable where $t = 2^{-m}x$ so that

$$\begin{aligned}
&= 2^{m+\frac{j_1+j_2}{2}} \int_{\mathbb{R}} \varphi(2^{j_1+m}t - k_1)\varphi(2^{j_2+m}t - k_2)dt \\
&= 2^{m+\frac{j_1+j_2}{2}} 2^{-m-\frac{j_1+j_2}{2}} \langle \varphi_{j_1+m,k_1} \varphi_{j_2+m,k_2} \rangle \\
&= \langle \varphi_{j_1+m,k_1} \varphi_{j_2+m,k_2} \rangle
\end{aligned}$$

From this we see that the orthogonality of the functions is maintained so long as we only use integer translations and scaling factors. This lets us select any arbitrary real value as our starting resolution j_0 so long as the stopping level J is an integer distance away.

APPENDIX B. ORIGINAL DATA

B.1 Stock price versus volume

The original data is available at Quant Quote and was accessed on June 9th, 2014. The raw data contained the following variables: the Date, Ticker (which comes from the file name), Open, High, Low, Close, Volume for the day. We use the 50 stocks with most complete information in the Quant Quote set (completeness was decided by the size of the data file for the company). In order to process the raw data, move the Quant Quote folder (unzipped) into your working directory in R. Next, use the script normalizeManyStocks.R in order to normalize all stock prices to have a mean of 0 and standard deviation of 1. The time invariant relationship between trading volume and price change is shown in Figure B.1. The final variables are explained below.

- Date: the date to which the observation corresponds
- HighPercentChange: the percent change of the highest price that day versus its opening price
- LowPercentChange: the percent change of the low price that day versus its opening price
- ClosePercentChange: the percent change of the close price that day versus its opening price
- Variability: HighPercentChange – LowPercentChange. The percent fluctuation range for the stock that day.
- NormVolume: the residual of the logarithm of the volume of stock traded versus the date

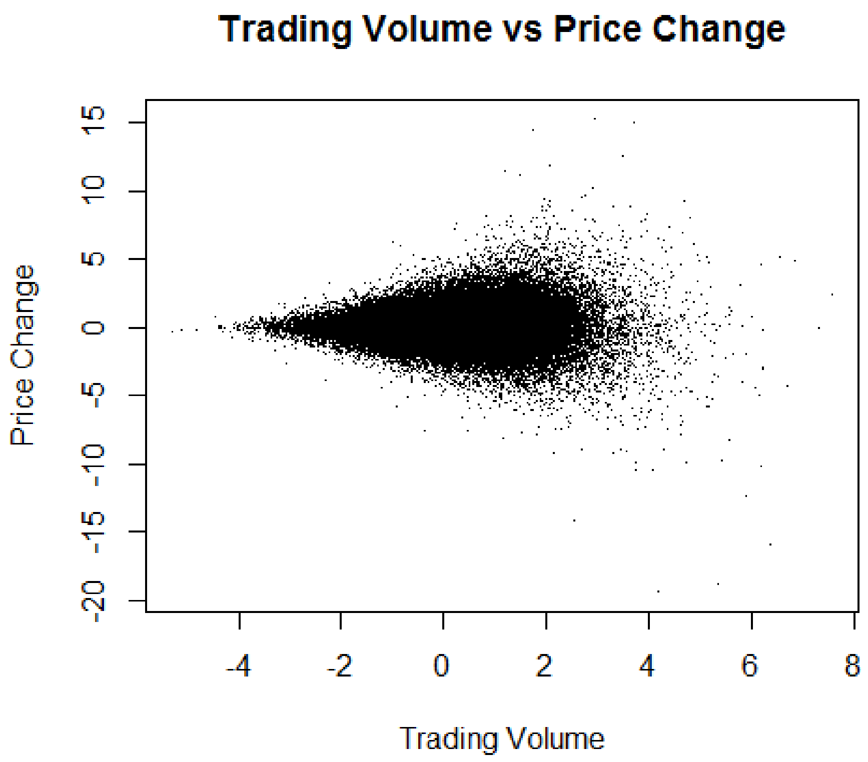


Figure B.1: Overall Trading Volume against Price Change

B.2 Options trading volume versus put/call ratio

The data originally comes from the CBOE Total Exchange Volume and Put/Call Ratios accessed on June 30th. The raw data contains the following variables: Date, Put volume, Call volume, total volume, and put/call ratio. Cleaning the data began with removing the first two rows from the data file. After this, the script normalizeOptions was run from R. The script normalizes all variables to have a mean of zero and standard deviation of 1. The time invariant relationship between the option trading volume and the put/call ratio is shown in Figure B.2. The final variables used are explained below (see subsection 5.2 for details about puts and calls).

- Date: the date to which the observation corresponds
- Puts: the total number of puts sold on the market that day
- Calls: the total number of calls sold on the market that day
- LogP.C.Ratio: the natural logarithm of the number of puts over the number of calls

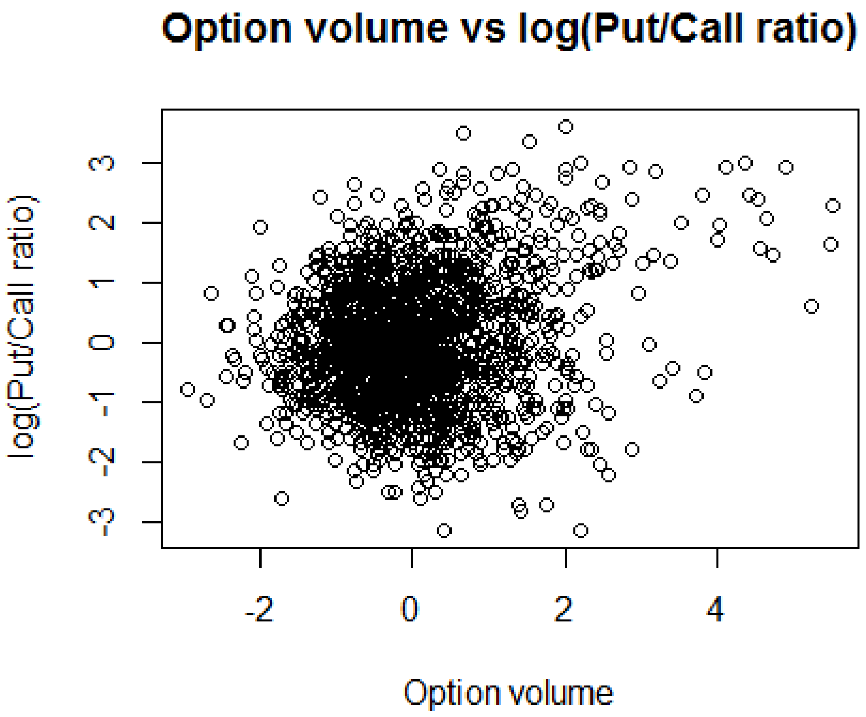


Figure B.2: Overall Option Volume against Put/Call Ratio

APPENDIX C. DATA PROCESSING SCRIPTS

All of the following scripts were writtin in the R programming language. They assume that the folder containing the data set has been downloaded and is in the working directory.

C.1 normalizeStocks.R

```
library(data.table)
#Load in the raw stock data file
NormData <- fread(paste(
                  "quantquote_daily_sp500_83986\\daily\\table_",
                  tickerName, ".csv", sep = ''))
setnames(x = NormData, old = c("V1", "V2", "V3", "V4", "V5", "V6", "V7"),
          new = c("Date", "Ticker", "Open", "High", "Low", "Close", "Volume"))

# Normalize the high, low, and close based on the open price
# Subtract one to center the data about 0
NormData$High <- NormData$High/NormData$Open - 1
NormData$Low <- NormData$Low/NormData$Open - 1
NormData$Close <- NormData$Close/NormData$Open - 1

# Remove the open price as it would always be 1 when normalized
NormData$Open <- NULL

# Remove the ticker as it is always 0 in this data set
NormData$Ticker <- NULL
```

```
# Format dates as the data type date
NormData$Date <- as.character(NormData$Date)
NormData$Date <- as.Date(NormData$Date, "%Y%m%d")

# Add the variability of the stock, the high minus the low
NormData$Variability <- (NormData$High - NormData$Low)

# Take log of volume since it increases exponentially over time
NormData$Volume <- log(NormData$Volume)

# Fit an exponential model of volume versus time
volVtime <- lm(NormData$Volume ~ NormData$Date)
NormData$Volume <- resid(volVtime)
remove(volVtime)

# Normalize volume and close price about their mean & st. dev
NormData$Volume <-(NormData$Volume - mean(NormData$Volume))/ sd(NormData$Volume)
NormData$Close <-(NormData$Close - mean(NormData$Close))/ sd(NormData$Close)

setnames(x = NormData, old = c("Date", "High", "Low", "Close", "Volume",
        "Variability"), new = c("Date", "HighPercentChange",
        "LowPercentChange", "ClosePercentChange", "NormVolume", "Variability"))
```

C.2 normManyStocks.R

```
# Normalize 50 stocks' information and combine them
# into a data table sorted by date
companyTickers = c('hum', 'csc', 'xrx', 'unh',
                    'low', 'cat', 'tyc', 'nke', 'schw',
                   'symc', 'adi', 'wmt', 'pg', 'txn',
                   'spls', 'ppg', 'wfm', 'xlnx', 'sbux',
                   'clx', 'bcr', 'wfc', 'bmy', 'avy',
                   'stj', 'slb', 'ms', 'abt', 'aa',
                   'see', 'hal', 'pki', 'hrl', 'sial',
                   'cvs', 'f', 'gww', 'aet', 'hrs',
                   'pnw', 'pbi', 'luv', 'dte', 'rok',
                   'gps', 'shw', 'mwv', 'aee', 'bby')
tickerName = 'altr'
source('NormalizeStocks.R')
allStockData = NormData

for (tickerName in companyTickers){
  source('NormalizeStocks.R')
  allStockData <- rbind(allStockData, NormData)
}

# Reorder the data based on its chronological order
allStockData <- allStockData[order(Date)]

# Remove unnecessary variables and write to file
allStockData$Date <- NULL
allStockData$HighPercentChange <- NULL
```

```
allStockData$LowPercentChange <- NULL
allStockData$Variability <- NULL
write.csv(allStockData, file = "closeVolume.csv", row.names = F)

# Plot the time invariant relationship
plot(allStockData$NormVolume, allStockData$Close,
        main = "Trading Volume vs Price Change", xlab = "Trading Volume",
        ylab = "Price Change", pch = '.')
```

C.3 normalizeOptions.R

```
# Script to read in and normalize the put/call volumes and ratios
totalpc <- read.csv("~/totalpc.csv", stringsAsFactors=FALSE)
totalpc$TOTAL <-(totalpc$TOTAL - mean(totalpc$TOTAL))/ sd(totalpc$TOTAL)

# It is necessary to take the log of the ratio so that a ratio of t and 1/t
# are seen as equally far from 0
totalpc$LogP.C.Ratio <- log(totalpc$P.C.Ratio)
totalpc$LogP.C.Ratio <-(totalpc$LogP.C.Ratio -
        mean(totalpc$LogP.C.Ratio))/ sd(totalpc$LogP.C.Ratio)

# Remove unused variables
totalpc$PUTS <- NULL
totalpc$CALLS <- NULL
totalpc$DATE <- NULL

# Write to data file
write.csv(totalpc, file = "optionsVolume.csv", row.names = F)

# Plot the time invariant relationship
plot(totalpc$TOTAL, totalpc$P.C.Ratio,
        main = "Option volume vs log(Put/Call ratio)",
        xlab = "Option volume", ylab = "log(Put/Call ratio)")
```

REFERENCES

[1] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.

[2] Björn Blohsfeld, Christoph Heinz, and Bernhard Seeger. Maintaining nonparametric estimators over data streams. In Gottfried Vossen, Frank Leymann, Peter C. Lockemann, and Wolffried Stucky, editors, *BTW*, volume 65 of *LNI*, pages 385–404. GI, 2005.

[3] ArnoldP. Boedihardjo, Chang-Tien Lu, and Feng Chen. Fast adaptive kernel density estimator for data streams. *Knowledge and Information Systems*, pages 1–33, 2013.

[4] Tine Buch-Larsen, Jens P. Nielsen, Montserrat Guillén, and Catalina Bolancé. Kernel density estimation for heavy-tailed distributions using the Champernowne transformation. *Statistics*, 39(6):503–516, 2005.

[5] C. Sidney Burrus, Ramesh A. Gopinath, and Haitao Guo. *Introduction to Wavelets and Wavelet Transforms: A Primer.* Prentice Hall, 1 edition, 1997.

[6] Yuan Cao, Haibo He, and Hong Man. Somke: Kernel density estimation over data streams by sequences of self-organizing maps. *IEEE Trans. Neural Netw. Learning Syst.*, 23(8):1254–1268, 2012.

[7] N Cencov. Evaluation of an unknown distribution density from observations. *Soviet Math*, 4:1559–1562, 1962.

[8] S. Grace Chang, Bin Yu, and Martin Vetterli. Adaptive wavelet thresholding for image denoising and compression. *IEEE Transactions on Image Processing*, 9:1532–1546, 2000.

[9] Peter King Clark. A subordinated stochastic process model with finite variance for speculative prices. *Econometrica*, pages 135–155, 1973.

[10] Harald Cramér. A contribution to the theory of statistical estimation. *Scandinavian Actuarial Journal*, 1946(1):85–94, 1946.

[11] Valdone Daskuviene. *Financial Markets*. Bytautas Magnus University, 2010.

[12] David L. Donoho and Iain M. Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, pages 1200–1224, 1995.

[13] David L. Donoho, Iain M. Johnstone, Gérard Kerkyacharian, and Dominique Picard. Density estimation by wavelet thresholding. *The Annals of Statistics*, 24:508–539, 1996.

[14] Steve Donoho. Early detection of insider trading in option markets. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 420–429, New York, NY, USA, 2004. ACM.

[15] Thomas W Epps and Mary Lee Epps. The Stochastic Dependence of Security Price Changes and Transaction Volumes: Implications for the Mixture-of-Distributions Hypothesis. *Econometrica*, 44:305–21, 1976.

[16] Leslaw Gajek. On improving density estimators which are not bona fide functions. *The Annals of Statistics*, 14(4):1612–1618, 1986.

[17] Edgar S. García-Treviño and Javier A. Barria. Online wavelet-based density estimation for non-stationary streaming data. *Computational Statistics & Data Analysis*, 56(2):327–344, 2012.

[18] Christian Gargour, Marcel Gabrea, Venkatanarayana Ramachandran, and Jean-Marc Lina. A short introduction to wavelets and their applications. *Cir. and Sys. Mag.*, 09(2):57–68, 2009.

[19] C. Heil and D. Walnut. Continuous and discrete wavelet transforms. *SIAM Review*, 31(4):628–666, 1989.

[20] Wolfgang Härdle, Gèrard Kerkyacharian, and Dominique Picard. *Wavelets, approximation, and statistical applications*. Lecture notes in statistics. Springer, New York, 1998.

[21] Maarten Jansen, Maurits Malfait, and Adhemar Bultheel. Generalized cross validation for wavelet thresholding. *Signal Processing*, 56:33–44, 1995.

[22] Björn Jawerth and Wim Sweldens. An overview of wavelet based multiresolution analyses. *SIAM Rev*, 36:377–412, 1993.

[23] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 180–191. VLDB Endowment, 2004.

[24] Ove Daae Lampe and Helwig Hauser. Interactive visualization of streaming data with kernel density estimation. In Giuseppe Di Battista, Jean-Daniel Fekete, and Huamin Qu, editors, *PacificVis*, pages 171–178. IEEE, 2011.

[25] G. Mallat. A theory for multiresolution signal decomposition : the wavelet representation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 1989.

[26] Liam Paninski. Estimation of entropy and mutual information. *Neural Comput.*, 15:1191–1253, 2003.

[27] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London, 1986.

[28] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Statistical change detection for multidimensional data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 667–676, New York, NY, USA, 2007. ACM.

[29] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the 32nd international conference on Very large data bases*, pages 187–198. VLDB Endowment, 2006.

[30] Teruko Takada. Bivariate nonparametric density estimation of stock prices and volume. *Asia-Pacific Financial Markets*, 8:215–236, 2001.

[31] George Tauchen and Mark Pitts. The price variability-volume relationship on speculative markets. *Econometrica*, 51:485–505, 1983.

[32] Nitin Thaper, Sudipto Guha, Piotr Indyk, and Nick Koudas. Dynamic multidimensional histograms. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 428–439, New York, NY, USA, 2002. ACM.

[33] M. Vannucci. Nonparametric density estimation using wavelets. Technical Report 95-26. Institute of Statistics and Decision Sciences, Duke University, 1995.

[34] EdwardJ. Wegman and KyleA. Caudle. Density estimation from streaming data using wavelets. In Alfredo Rizzi and Maurizio Vichi, editors, *Compstat 2006 - Proceedings in Computational Statistics*, pages 231–242. Physica-Verlag HD, 2006.

[35] Aoying Zhou, Zhiyuan Cai, Li Wei, and Weining Qian. M-kernel merging: Towards density estimation over data streams. In *In Proc. of DASFAA*, pages 285–292, 2003.