QUAD: Quadratic-Bound-based Kernel Density Visualization

Tsz Nam Chan, Reynold Cheng
The University of Hong Kong
{tnchan2,ckcheng}@hku.hk

Man Lung Yiu
Hong Kong Polytechnic University
csmlyiu@comp.polyu.edu.hk

ABSTRACT

Kernel density visualization, or KDV, is used to view and understand data points in various domains, including traffic or crime hotspot detection, ecological modeling, chemical geology, and physical modeling. Existing solutions, which are based on computing kernel density (KDE) functions, are computationally expensive. Our goal is to improve the performance of KDV, in order to support large datasets (e.g., one million points) and high screen resolutions (e.g., 1280×960 pixels). We examine two widely-used variants of KDV, namely approximate kernel density visualization (ϵKDV) and thresholded kernel density visualization (τKDV). For these two operations, we develop fast solution, called QUAD, by deriving quadratic bounds of KDE functions for different types of kernel functions, including Gaussian, triangular etc. We further adopt a progressive visualization framework for KDV, in order to stream partial visualization results to users continuously. Extensive experiment results show that our new KDV techniques can provide at least one-order-of-magnitude speedup over existing methods, without degrading visualization quality. We further show that QUAD can produce the reasonable visualization results in real-time (0.5 sec) by combining the progressive visualization framework in single machine setting without using GPU and parallel computation.

ACM Reference Format:
Tsz Nam Chan, Reynold Cheng and Man Lung Yiu. 2019. QUAD: Quadratic-Bound-based Kernel Density Visualization. In *Proceedings of ACM (SIGMOD).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Data visualization [8, 24, 25] is an important tool for understanding a dataset. In this paper, we study *kernel-density-estimation-based visualization* [43] (termed *kernel density visualization* (KDV) here), which is one of the most commonly used data visualization solutions. KDV is often used in hotspot detection (e.g., in criminology and transportation) [4, 20, 45, 49, 51] and data modeling (e.g., in ecology, chemistry, and physics) [3, 10, 29, 30, 46]. Most of these applications restrict the dimensionality of datasets to be smaller than 3 [4, 20, 29, 30, 45, 46, 49, 51].[1] Figure 1 shows the use of KDV in analyzing motor vehicle thefts in Arlington, Texas in 2007. Here, each black dot is a data point, which denotes the place where a crime has been committed. A color map, which represents the criminal risk in different places, is generated by KDV; for instance, a red region indicates the highest risk of vehicle thefts in that area. As discussed in [4, 20], color maps are often used by social scientists or criminologists for data analysis.

Table 1 summarizes the usage of KDV in different domains. Due to its wide applicability, KDV is often provided in data analytics platforms, including Scikit-learn [2], ArcGIS [3], and QGIS [4].

Table 1: KDV Applications

Type	domain	Usage	Ref.
Hotspot detection	Criminology	Detection of crime regions	[4, 20, 53]
	Transportation	Detection of traffic hotspots	[45, 49, 51]
Data modeling	Ecology	Visualization of polluted regions	[21, 29, 30]
	Chemistry	Visualization of detrial age distributions	[46]
	Physics	Particle searching	[3, 10]

To generate a color map (e.g., Figure 1), KDV determines the color value of each pixel $\mathbf{q}$ on the two-dimensional computer screen by a *kernel density* (KDE) function, denoted by $\mathcal{F}_P(\mathbf{q})$ [43]. Equation 1 shows one example of $\mathcal{F}_P(\mathbf{q})$ with Gaussian kernel, where P and $dist(\mathbf{q}, \mathbf{p_i})$ are the set of two-dimensional data points and Euclidean distance respectively.

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w \cdot \exp(-\gamma dist(\mathbf{q}, \mathbf{p_i})^2) \quad (1)$$

SIGMOD, 2020

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] For high-dimensional dataset, one approach is to first use dimension reduction techniques (e.g., [48]) to reduce the dimension to 1 or 2 and then utilize KDV to generate color map.
[2] https://scikit-learn.org/
[3] http://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/how-kernel-density-works.htm
[4] https://docs.qgis.org/2.18/en/docs/user_manual/plugins/plugins_heatmap.html

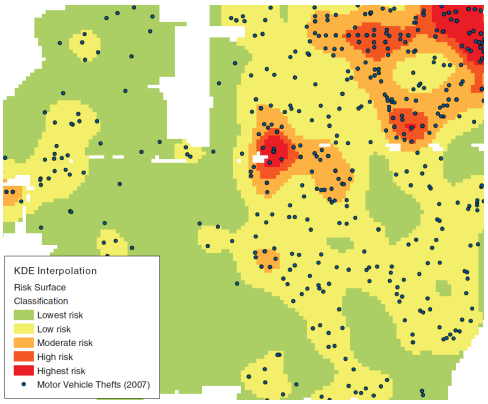


Figure 1: A color map for motor vehicle thefts (black dots) in Arlington, Texas in 2007 (Cropped from [20])

In this paper, we will also consider $\mathcal{F}_P(\mathbf{q})$ with other kernel functions in Section 5. As a remark, all kernel functions, that we consider in this paper, are adopted in famous software, e.g., Scikit-learn [35] and QGIS [40].

A higher $\mathcal{F}_P(\mathbf{q})$ value indicates a higher density of data points in the region around $\mathbf{q}$. The above KDE function is computationally expensive to compute. Given a data set with 1 million 2D points, KDV involves over 2 trillion operations [36] on a 1920×1080 screen. As pointed out in [13, 16, 52, 55, 56], KDV cannot scale well to handle many data points and display of color maps on high-resolution screens. To address this problem, researchers have proposed two variants of KDV, which aim to improve its performance:

• ϵ**KDV**: This is an approximate version of KDV. A relative error parameter, ϵ, is used, such that for each pixel $\mathbf{q}$, the pixel color is within $(1 \pm \epsilon)$ of $\mathcal{F}_P(\mathbf{q})$. Figure 2a shows a color map generated by the original (exact) KDV, while Figure 2b illustrates the corresponding color map for ϵKDV with ϵ equal to 0.01. As we can see, the two color maps do not look different. ϵKDV runs faster than exact KDV [7, 17, 54–56], and is also supported in data analytics software (e.g., Scikit-learn [35]).

• τ**KDV**: In tasks such as hotspot detection [4, 20], a data visualization user only needs to know which spatial region has a high density (i.e., hotspot), and not the other areas. One such hotspot is the red region in Figure 1. A color map with two colors are already sufficient. Figure 2c shows such a color map. To generate this color map, the τKDV can be used, where a threshold, τ, detemines the color of a pixel: a color for $\mathbf{q}$ when $\mathcal{F}_P(\mathbf{q}) \geq \tau$ (to indicate high density), and another color otherwise. This method, recently studied in [7, 13], is shown to be faster than exact KDV.

Although ϵKDV and τKDV perform better than exact KDV, they still require a lot of time. On a $270k$-point crime dataset [1], displaying a color map on a screen with 1280×960 pixels takes over an hour for most methods, including the ϵKDV solution implemented in Scikit-learn. In fact, these existing methods often cannot deliver *real-time* performance, which allows color maps to be generated quickly, thereby saving the precious waiting time of data analysts.

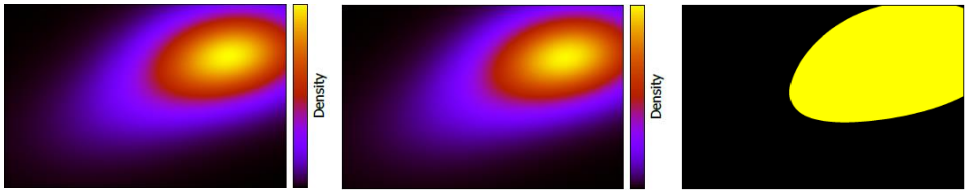
Our contributions. In this paper, we develop a solution, called QUAD, in order to improve the performance of ϵKDV and τKDV. The main idea is to derive lower and upper bounds of the KDE function (i.e., Equation 1) in terms of quadratic functions (cf. Section 4). These *quadratic bounds* are theoretically tighter than the existing ones (in aKDE [17], tKDC [13], and KARL [7]), enabling faster pruning. In addition, many KDV-based applications [11, 15, 20, 27] also utilize other kernel functions, including triangular, cosine kernels etc. Therefore, we extend our techniques to support other kernel functions (cf. Section 5), which cannot be supported by the state-of-the-art solution, KARL [7]. In our experiments on large datasets in a single machine, QUAD is at least one-order-of-magnitude faster than existing solutions. For ϵKDV, QUAD takes 100-1000 sec to generate color map for each large-scale dataset (0.17M to 7M) with 2560×1920 pixels, using small relative error $\epsilon = 0.01$. However, most of the other methods fail to generate the color map within 2 hours under the same setting. For τKDV, QUAD can achieve nearly 10 sec with 1280×960 pixels, using different thresholds.

We further adopt a progressive visualization framework for KDV (cf. Section 6), in order to continuously output partial visualization results (by increasing the resolution). A user can terminate the process anytime, once the partial visualization results are satisfactory, instead of waiting for the precise color map to be generated. Experiment results show that we can achieve real-time (0.5 sec) in single machine without using GPU and parallel computation by combining this framework with our solution QUAD.

The rest of the paper is organized as follows. We first review existing work in Section 2. We then discuss the background in Section 3. Later, we present quadratic bound functions for KDE in Section 4. After that, we extend our quadratic bounds to other kernel functions in Section 5. We then discuss our progressive visualization framework for KDV in Section 6. Lastly, we show our results in Section 7, and conclude in Section 8. All the proofs are shown in Section 9.

2 RELATED WORK

Kernel density visualization (KDV) is widely used in many application domains, such as: ecological modeling [29, 30], crime [4, 20, 53] or traffic hotspot detection [45, 49, 51], chemical geology [46] and physical modeling [10]. For each application, they either need to compute the approximate kernel density values with theoretical guarantee [17] (ϵKDV) or test whether density values are above a given threshold [13] (τKDV) in the spatial region. Due to the high computational complexity, many existing algorithms have been developed for efficient computation of these two variants of KDV, which can be divided into three camps (cf. Table 2).



(a) Exact KDV (b) εKDV, ε=0.01 (c) τKDV

Figure 2: Illustrating ϵKDV and τKDV

Table 2: Existing methods in KDV

Techniques	Ref.	Summary	Software
Function approximation	[16, 41, 50]	Heuristics	-
Dataset sampling	[22, 54–56] [37–39]	Quality-preserving (probabilistic guarantee)	-
Bound functions	[7, 13, 17]	Quality-preserving (deterministic guarantee)	Scikit-learn [35]

In the first camp, researchers propose function approximation methods for KDE function (cf. Equation 1). Raykar et al. [41] and Yang et al. [50] propose using fast Gauss transform to efficiently and approximately compute KDE function. However, this type of methods normally does not provide the theoretical guarantee between the returned value and exact result.

In the second camp, researchers propose efficient algorithm by sampling the original datasets. Zheng et al. [54–56] and Phillips et al. [37–39] pre-sample the datasets into smaller sizes and apply exact KDV for the reduced datasets. They proved that the output result is near the exact KDE value for each pixel in original datasets. However, their work [37, 54–56] only aim to provide the probabilistic error guarantee (e.g., $\epsilon = 0.01$ with probability 0.8). On the other hand, deterministic guarantee (e.g., $\epsilon = 0.01$), used in our problem settings [7, 13, 17], are adopted in existing software, e.g., Scikit-learn [35]. Moreover, their algorithms still need to evaluate the exact KDV for the reduced datasets, which can still be time-consuming. Some other research studies [33, 34] also adopt the sampling methods for different tasks (e.g., visualization and query processing). Park et al. [33] propose advanced method for sampling the datasets in the preprocessing stage, which is possible to further reduce the sample size, and generate the visualization in the online stage. However, the large preprocessing time is not acceptable in our setting, since the visualized datasets are not known in advance for the scientific applications (cf. Table 1) and software (e.g., Scikit-learn, ArcGIS and QGIS). In addition, unlike the sampling methods for KDV (e.g., [54]), since these methods [33, 34] do not discuss how to appropriately update the weight value for each data point in the output sample set for the new kernel aggregation function [5], they cannot support KDV.

In the third camp, which we are interested in, researchers propose different efficient lower and upper bound functions with index structure (e.g., kd-tree) to evaluate KDE functions (e.g., Equation 1). Albeit there are many existing work for developing new bound functions [7, 13, 17], there is one key difference between our proposal QUAD and these methods. Our proposed quadratic bound functions are tighter than all existing bound functions in previous literatures [7, 13, 17] with only a slight overhead ($O(d^2)$ time complexity) for Gaussian kernel, which is affordable as the dimensionality d is normally small ($d < 3$) in KDV, and only in $O(d)$ time complexity for other kernel functions (e.g., triangular and cosine kernels). On the other hand, some other research studies in approximation theory [9, 47] also focus on utilizing the polynomial functions to approximate the more complicated functions, e.g., interpolation and curve-fitting. However, these methods cannot provide the correctness guarantee of lower and upper bounds for kernel aggregation function (cf. Equation 1). In addition, high-order polynomial functions cannot provide fast evaluation in our setting (e.g., $O(d^2)$ time complexity).

Interactive visualization is a well-studied problem [12, 14, 24–26, 36, 48, 57]. Users can control the visualization quality under different resolutions (e.g., 256×256 or 512×512) [36]. However, existing work either utilize modern hardware (e.g., GPU) [26, 36]/distributed algorithms (e.g., MapReduce) [36] to support KDV in real-time or do not focus on KDV [12, 14, 24, 25, 48, 57]. In this work, we adopt a progressive visualization framework for KDV which aims to continuously provide coarse-to-fine partial visualization results to users. Users can stop the process at any time once they are satisfied with the visualization results. By using our solution QUAD and this framework, we can achieve satisfactory visualization quality (nearly no degradation) in real-time (0.5 sec) in single machine setting without using GPU and parallel computation.

There are also many other studies for utilizing parallel/distributed computation, e.g., MapReduce [54], and modern hardware, e.g., GPU [52] and FPGA [16], to further boost the efficiency evaluation of exact KDV. In this work, we focus on single machine setting with CPU and leave the combination of our method QUAD with these optimization opportunities in our future work.

In both database and visualization communities, there are also many recent visualization tools, which are not based on KDV [18, 19, 23, 28, 31, 32, 42, 48]. This type of work mainly focuses on avoiding the overplotting effect (i.e., too

[5] We replace P and w by output sample set and w_i in Equation 1 respectively.

many points in the spatial region) of the visualization of data points in maps or scatter plots. Some other visualization tools are also summarized in the book [44]. However, in some applications, e.g., hotspot detection and data modeling (cf. Table 1), the users mainly utilize KDV for visualizing the density of different regions in which these work cannot be applied in this scenario.

3 PRELIMINARIES

In this section, we first revisit the concepts [7, 13, 17] of bound functions (cf. Section 3.1) and the indexing framework (cf. Section 3.2). Then, we also illustrate the state-of-the-art bound functions [7] (cf. Section 3.3), which are mostly related to our work.

3.1 Bound Functions

In existing literatures [7, 13, 17], they develop the lower bound $LB(\mathbf{q})$ and upper bound $UB(\mathbf{q})$ for $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 1), given the pixel $\mathbf{q}$, which must fulfill the following correctness condition:

$$LB(\mathbf{q}) \leq \mathcal{F}_P(\mathbf{q}) \leq UB(\mathbf{q})$$

For ϵKDV and τKDV, we can avoid evaluating the computationally expensive operation $\mathcal{F}_P(\mathbf{q})$ if $UB(\mathbf{q}) \leq (1+\epsilon)LB(\mathbf{q})$ for ϵKDV and $LB(\mathbf{q}) \geq \tau$ or $UB(\mathbf{q}) \leq \tau$ for τKDV [7]. Therefore, once the bound functions are (1) tighter (near the exact $\mathcal{F}_P(\mathbf{q})$) and (2) fast to evaluate (much faster than $\mathcal{F}_P(\mathbf{q})$), we can achieve significant speedup compared with the exact evaluation of $\mathcal{F}_P(\mathbf{q})$.

3.2 Indexing Framework for Bound Functions

Existing literatures [7, 13, 17] adopt the hierarchical index structures (e.g., kd-tree) to index the point set P (cf. Figure 3).[6] We illustrate the running steps (cf. Table 3) for evaluating the ϵKDV and τKDV in this indexing framework. In the following, we denote the exact value, lower and upper bounds between the pixel $\mathbf{q}$ and node R_i to be $\mathcal{F}_{R_i}(\mathbf{q})$, $LB_{R_i}(\mathbf{q})$ and $UB_{R_i}(\mathbf{q})$ respectively, where: $LB_{R_i}(\mathbf{q}) \leq \mathcal{F}_{R_i}(\mathbf{q}) \leq UB_{R_i}(\mathbf{q})$.

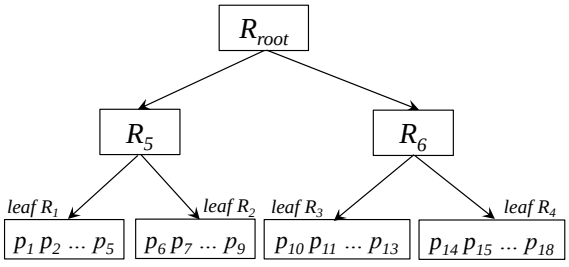


Figure 3: Hierarchical index structure for different bound functions [7, 13, 17]

[6]The data analytics software, Scikit-learn [35], utilizes kd-tree by default for solving ϵKDV.

Initially, the algorithm evaluates the bound functions of root node R_{root} and then pushes back this node into the priority queue. This priority queue manages the evaluation order of different nodes R_i based on the decreasing order of bound difference $UB_{R_i}(\mathbf{q}) - LB_{R_i}(\mathbf{q})$. In each iteration, the algorithm pops out the node with the highest priority, pushes back its child nodes and maintain the incremental bound values $\widehat{lb}$ and $\widehat{ub}$ (e.g., R_{root} is popped, its child nodes R_5 and R_6 are added and the bounds $\widehat{lb}$ and $\widehat{ub}$ are updated in step 2). Once the popped node is the leaf (e.g., R_1 in step 4), the exact value for that node is evaluated.

Table 3: Running steps for each $\mathbf{q}$ in the indexing framework

Step	Priority queue	Maintenance of lower bound $\widehat{lb}$ and upper bound $\widehat{ub}$	Popped node
1	R_{root}	$\widehat{lb} = LB_{R_{root}}(\mathbf{q})$, $\widehat{ub} = UB_{R_{root}}(\mathbf{q})$	
2	R_5, R_6	$\widehat{lb} = LB_{R_5}(\mathbf{q}) + LB_{R_6}(\mathbf{q})$, $\widehat{ub} = UB_{R_5}(\mathbf{q}) + UB_{R_6}(\mathbf{q})$	R_{root}
3	R_1, R_6, R_2	$\widehat{lb} = LB_{R_1}(\mathbf{q}) + LB_{R_6}(\mathbf{q}) + LB_{R_2}(\mathbf{q})$, $\widehat{ub} = UB_{R_1}(\mathbf{q}) + UB_{R_6}(\mathbf{q}) + UB_{R_2}(\mathbf{q})$	R_5
4	R_6, R_2	$\widehat{lb} = \mathcal{F}_{R_1}(\mathbf{q}) + LB_{R_6}(\mathbf{q}) + LB_{R_2}(\mathbf{q})$, $\widehat{ub} = \mathcal{F}_{R_1}(\mathbf{q}) + UB_{R_6}(\mathbf{q}) + UB_{R_2}(\mathbf{q})$	R_1

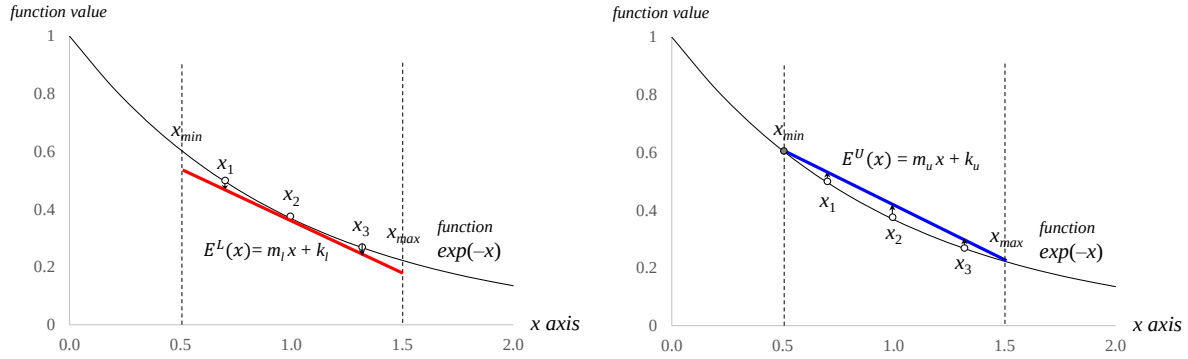
The algorithm terminates for each pixel $\mathbf{q}$ in (1) ϵKDV and (2) τKDV once the incremental bounds satisfy (1) $\widehat{ub} \leq (1+\epsilon)\widehat{lb}$ and (2) $\widehat{lb} \geq \tau$ or $\widehat{ub} \leq \tau$ respectively (cf. Section 3.1). Since we follow the same indexing framework as [7, 13, 17], we omit the detailed algorithm here. Interested readers can refer to the algorithm from the supplementary notes[7] of [7]. Similar technique can be also found in similarity search community [5, 6]. Even though the worst case time complexity of this algorithm for a given pixel $\mathbf{q}$ is $O(nB \log n + nd)$ time (B is the evaluation time of each $LB_{R_i}(\mathbf{q})/UB_{R_i}(\mathbf{q})$), which is even higher than directly evaluating $\mathcal{F}_P(\mathbf{q})$ ($O(nd)$ time), this algorithm is very efficient if we utilize the fast and tight bound functions.

3.3 State-of-the-art Bound Functions

Among most of the existing bound functions, Chan et al. [7] developed the most efficient and tightest bound functions for Equation 1. They utilize the linear function $Lin_{m,k}(x) = mx + k$ to approximate the exponential function $\exp(-x)$. We denote $E^L(x) = m_l x + k_l$ and $E^U(x) = m_u x + k_u$ for the lower and upper bounds of $\exp(-x)$ respectively, i.e., $E^L(x) \leq \exp(-x) \leq E^U(x)$, as shown in Figure 4.

Once they set $x = \gamma dist(\mathbf{q}, \mathbf{p_i})^2$, they can obtain the linear lower and upper bound functions $\mathcal{FL}_P(\mathbf{q}, Lin_{m,k}) = \sum_{\mathbf{p_i} \in P} w(m\gamma dist(\mathbf{q}, \mathbf{p_i})^2 + k)$ for $\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2)$, given the suitable choices of m and k. They

[7]https://github.com/edisonchan2013928/KARL-Fast-Kernel-Aggregation-Queries



(a) Linear lower bound (b) Linear upper bound

Figure 4: Linear lower and upper bound functions for exponential function $\exp(-x)$ (from [7])

prove that these bounds are tighter than existing bound functions [13, 17]. In addition, they also prove that their linear bounds $\mathcal{FL}_P(\mathbf{q}, Lin_{m,k})$ can be computed in $O(d)$ time [7] (cf. Lemma 1). More details can be found from [7][7].

LEMMA 1. *[7] Given two values m and k, $\mathcal{FL}_P(\mathbf{q}, Lin_{m,k}) = \sum_{\mathbf{p_i}\in P} w(m\gamma dist(\mathbf{q}, \mathbf{p_i})^2 + k)$ can be computed in $O(d)$ time.*

The main idea of achieving these efficient bounds is based on the fast evaluation of the sum of squared distance, which can be achieved in $O(d)$ time [7], given the precomputed $\mathbf{a_P} = \sum_{\mathbf{p_i}\in P} \mathbf{p_i}$, $b_P = \sum_{\mathbf{p_i}\in P} ||\mathbf{p_i}||^2$ and $|P|$.

$$\begin{aligned}\sum_{\mathbf{p_i}\in P} dist(\mathbf{q}, \mathbf{p_i})^2 &= \sum_{\mathbf{p_i}\in P} (||\mathbf{q}||^2 - 2\mathbf{q}\cdot\mathbf{p_i} + ||\mathbf{p_i}||^2) \\ &= |P|\cdot||\mathbf{q}||^2 - 2\mathbf{q}\cdot\mathbf{a_P} + b_P\end{aligned}$$

4 QUADRATIC BOUNDS

As illustrated in Section 3, if we can develop the fast and tighter bound functions for $\mathcal{F}_P(\mathbf{q})$, we can achieve significant speedup for solving both ϵKDV and τKDV. Therefore, we ask a question, can we develop the new bound functions, which are fast and tighter, compared with the state-of-the-art linear bounds [7] (i.e., $\mathcal{FL}_P(\mathbf{q}, Lin_{m,k})$)? To answer this question, we utilize the quadratic function $Q(x) = ax^2 + bx + c$, $(a > 0)$, to approximate the exponential function $\exp(-x)$. Observe from Figure 5, quadratic function can achieve the lower and upper bounds ($Q^L(x)$ and $Q^U(x)$ respectively) of $\exp(-x)$ with the suitable choice of the parameters a, b and c, given $[x_{min}, x_{max}]$ as the bounding interval of $x_i = \gamma dist(\mathbf{q}, \mathbf{p_i})^2$ (white circle), i.e., $x_{min} \le x_i \le x_{max}$, where x_{min} and x_{max} are based on the minimum and maximum distances between $\mathbf{q}$ and the bounding rectangle of $\mathbf{p_i}$ respectively [7] ($O(d)$ time). In this paper, we let $Q^L(x)$ and $Q^U(x)$ be:

$$\begin{aligned}Q^L(x) &= a_\ell x^2 + b_\ell x + c_\ell \\ Q^U(x) &= a_u x^2 + b_u x + c_u\end{aligned}$$

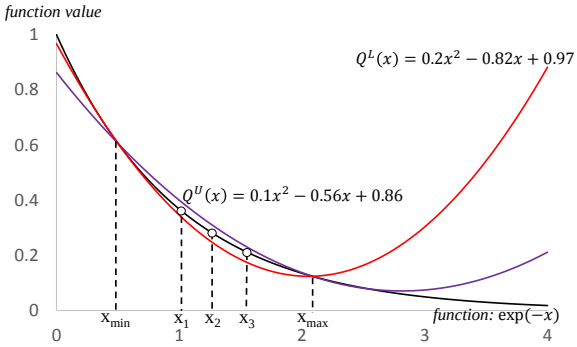


Figure 5: Quadratic lower (red) and upper (purple) bound functions of $\exp(-x)$ in the range $[x_{min}, x_{max}]$

We define the aggregation of the quadratic function as:

$$\mathcal{FQ}_P(\mathbf{q}, Q) = \sum_{\mathbf{p_i}\in P} w\Big(a(\gamma dist(\mathbf{q}, \mathbf{p_i})^2)^2 + b\gamma dist(\mathbf{q}, \mathbf{p})^2 + c\Big) \tag{2}$$

With the above concept, $\mathcal{FQ}_P(\mathbf{q}, Q^L)$ and $\mathcal{FQ}_P(\mathbf{q}, Q^U)$ can serve as the lower and upper bounds respectively for $\mathcal{F}_P(\mathbf{q})$, as stated in Lemma 2. We include the formal proof of Lemma 2 in Section 9.1.

LEMMA 2. *If $Q^L(x)$ and $Q^U(x)$ are the lower and upper bounds for $\exp(-x)$ respectively, we have $\mathcal{FQ}_P(\mathbf{q}, Q^L) \le \mathcal{F}_P(\mathbf{q}) \le \mathcal{FQ}_P(\mathbf{q}, Q^U)$.*

4.1 Fast Evaluation of Quadratic Bounds

In the following lemma, we illustrate how to efficiently compute the bound function $\mathcal{FQ}_P(\mathbf{q}, Q)$ in $O(d^2)$ time in the query stage (cf. Lemma 3). We leave the proof in Section 9.2.

LEMMA 3. *Given the coefficients a, b and c of quadratic function Q, $\mathcal{FQ}_P(\mathbf{q}, Q)$ can be computed in $O(d^2)$ time.*

Although the computation time of $\mathcal{FQ}_P(\mathbf{q}, Q)$ is slightly worse than [7], which is only in $O(d)$ time, this overhead is negligible as the dimensionality of datasets is smaller than 3

in KDV. However, as we will show in the following sections, our bounds are tighter than the existing bound functions [7].

4.2 Tight Quadratic Upper Bound Function

To obtain the (1) correct and (2) tighter upper bound function compared with existing chord-based upper bound functions [7], $Q^U(x)$ remains above $\exp(-x)$ and below the linear function $E^U(x)$ for $x \in [x_{min}, x_{max}]$, as shown in Figure 6.

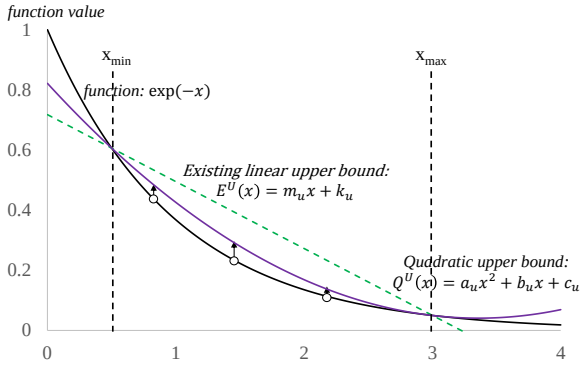


Figure 6: Correct and tighter quadratic bound $Q^U(x)$ for $\exp(-x)$ (purple curve)

Observe that $Q^U(x)$ should pass through the points $(x_{min}, \exp(-x_{min}))$ and $(x_{max}, \exp(-x_{max}))$. By simple algebraic operations, we can represent each b_u and c_u by a_u.

$$b_u = \frac{\exp(-x_{max}) - \exp(-x_{min})}{x_{max} - x_{min}} - a_u(x_{min} + x_{max})$$

$$c_u = \frac{\exp(-x_{min})x_{max} - \exp(-x_{max})x_{min}}{x_{max} - x_{min}} + a_u x_{min} x_{max}$$

Therefore, the shape of the parabola is controlled by the parameter a_u. Observe from Figure 7, once the parameter a_u becomes larger, the curvature of the parabola is larger and as such, it can achieve tighter upper bound function (e.g., 0.05). However, it violates the upper bound condition once a_u is too large (e.g., 0.1 and 0.15).

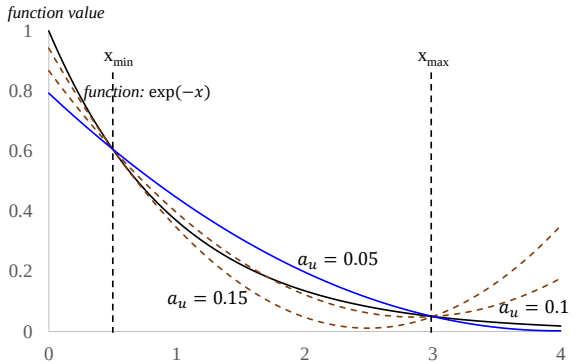


Figure 7: Correct upper bound (solid line)/wrong upper bound (dashed line)

In Theorem 1, we claim that the best a_u can be selected to achieve the correct and tighter upper bound function. We leave the proof in Section 9.3.

THEOREM 1. *The correct and tighter upper bound is achieved ($m_u x + k_u \geq Q^U(x) \geq \exp(-x)$) when $a_u = a_u^*$:*

$$a_u^* = \frac{(x_{max} - x_{min} + 1)\exp(-x_{max}) - \exp(-x_{min})}{(x_{max} - x_{min})^2}$$

Based on Lemma 2 and Theorem 1, we can conclude that our upper bound $\mathcal{FQ}_P(\mathbf{q}, Q^U)$ is correct and tighter than the state-of-the-art, i.e.,

$$\mathcal{F}_P(\mathbf{q}) \leq \mathcal{FQ}_P(\mathbf{q}, Q^U) \leq \mathcal{FL}(\mathbf{q}, Lin_{m_u, k_u})$$

4.3 Tight Quadratic Lower Bound Function

Compared with tangent-based linear lower bound function to $\exp(-x)$, there exist another tighter quadratic lower bound function, i.e., $\exp(-x) \leq Q^L(x) \leq m_l x + k_l$, which passes through the tangent point $(t, \exp(-t))$ and also $(x_{max}, \exp(-x_{max}))$, as shown in Figure 8.

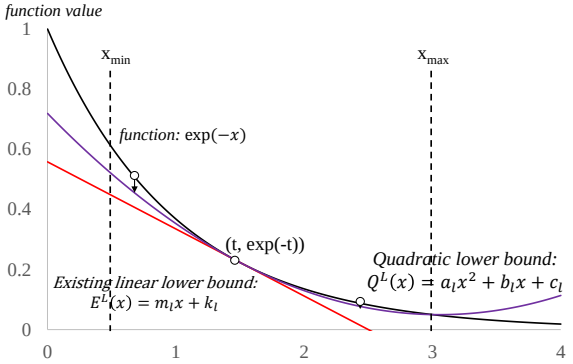


Figure 8: The tighter quadratic lower bound function for $\exp(-x)$

By simple differentiation and algebraic operations, we can obtain:

$$a_l = \frac{\exp(-x_{max}) + (x_{max} - 1 - t)\exp(-t)}{(x_{max} - t)^2}$$

$$b_l = -\exp(-t) - \frac{2t(\exp(-x_{max}) + (x_{max} - 1 - t)\exp(-t))}{(x_{max} - t)^2}$$

$$c_l = (1 + t)\exp(-t) + \frac{t^2(\exp(-x_{max}) + (x_{max} - 1 - t)\exp(-t))}{(x_{max} - t)^2}$$

We omit the correctness and tightness proof of $\exp(-x) \geq Q^L(x) \geq m_l x + k_l$, as it can be easily proved based on the proof of Theorem 1 for left part and the convex property for right part. By Lemma 2 and the above inequality, we have:

$$\mathcal{F}_P(\mathbf{q}) \geq \mathcal{FQ}_P(\mathbf{q}, Q^L) \geq \mathcal{FL}(\mathbf{q}, Lin_{m_l, k_l})$$

Now, we proceed to choose the parameter t ($x_{min} \leq t \leq x_{max}$) for quadratic function. However, unlike [7], finding the

best t is very tedious, which does not have close form solution. We therefore follow [7] and choose t to be t^*, where:

$$t^* = \frac{\gamma}{|P|} \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2 \quad (3)$$

5 OTHER KERNEL FUNCTIONS

In previous sections, we mainly focus on the Gaussian kernel function. However, many existing literatures [11, 15, 20, 27] also use other kernel functions, e.g., triangular kernel, cosine kernel, for hotspot detection or ecological modeling. Therefore, different types of existing software, including QGIS, ArcGIS and Scikit-learn, also support different kernel functions. In this section, we study the problems ϵKDV and τKDV with the following kernel aggregation function.

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w \cdot \mathcal{K}(\mathbf{q}, \mathbf{p}) \quad (4)$$

where different $\mathcal{K}(\mathbf{q}, \mathbf{p})$ functions are defined in Table 4.

Table 4: Types of kernel functions

Kernel function	Equation ($\mathcal{K}(\mathbf{q}, \mathbf{p})$)	Used in
Triangular	$\max(1 - \gamma \cdot dist(\mathbf{q}, \mathbf{p}), 0)$	[15, 20]
Cosine	$\begin{cases} \cos(\gamma dist(\mathbf{q}, \mathbf{p})) & \text{if } dist(\mathbf{q}, \mathbf{p_i}) \leq \frac{\pi}{2\gamma} \\ 0 & \text{otherwise} \end{cases}$	[11, 20, 27]
Exponential	$\exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p}))$	[20]

We first illustrate the weakness of existing methods [7, 13, 17] in Section 5.1 and explore how to extend our quadratic bounds for these kernel functions in Section 5.2.

5.1 Weakness of Existing Methods

Recall from Lemma 1 (cf. Section 3.3), the state-of-the-art linear bound functions [7] can be efficiently evaluated (in $O(d)$ time) with Gaussian kernel function due to the efficient evaluation of the term $\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2$. However, observe from Table 4, all these kernel functions only depend on the term $dist(\mathbf{q}, \mathbf{p})$ rather than $dist(\mathbf{q}, \mathbf{p})^2$. Therefore, the linear bound function [7] for Equation 4 with these kernel functions can be derived as (by setting $x_i = \gamma dist(\mathbf{q}, \mathbf{p_i})$):

$$\mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) = \sum_{\mathbf{p_i} \in P} w(m \cdot \gamma dist(\mathbf{q}, \mathbf{p_i}) + k)$$

Since $\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})$ cannot be efficiently evaluated, the state-of-the-art lower and upper bound functions [7] cannot achieve $O(d)$ time for these kernel functions. Therefore, we can only choose other approach [13, 17], which utilizes x_{min} and x_{max}, to efficiently evaluate the bound functions for Equation 4 in $O(d)$ time, where x_{min} and x_{max} are based on the minimum and maximum distances between $\mathbf{q}$ and the minimum bounding rectangle of all $\mathbf{p_i}$ respectively. Using triangular kernel function as an example, the lower and upper bound functions for $\mathcal{F}_P(\mathbf{q})$ are:

$$LB_R(\mathbf{q}) = w|P| \max(1 - x_{max}, 0) \quad (5)$$

$$UB_R(\mathbf{q}) = w|P| \max(1 - x_{min}, 0) \quad (6)$$

However, these bound functions are not tight. Therefore, one natural question is whether we can develop the efficient and tighter quadratic bounds (e.g., $O(d)$ time) for these kernel functions.

5.2 Quadratic Bounds for Other Kernel Functions

In this section, we mainly focus on the triangular kernel function, but our techniques can also be extended to other kernel functions in Table 4 (cf. Section 5.2.3). To avoid the term $\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})$, we utilize the quadratic function $Q(x) = ax^2 + c$ ($a < 0$), which sets the coefficient b to 0, to approximate the function $\max(1 - x, 0)$, as shown in Figure 9.

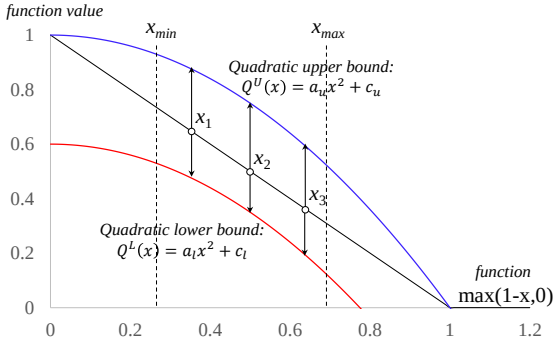


Figure 9: Quadratic lower (red) and upper (blue) bound functions of $\max(1 - x, 0)$ in the range $[x_{min}, x_{max}]$

With $x_i = \gamma dist(\mathbf{q}, \mathbf{p_i})$, we redefine the aggregation of quadratic function as:

$$\mathcal{FQ}_P(\mathbf{q}, Q) = \sum_{\mathbf{p_i} \in P} w(a(\gamma dist(\mathbf{q}, \mathbf{p_i}))^2 + c) \quad (7)$$

Observe that this bound function only depends on $\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2$, therefore, it can be evaluated in $O(d)$ time (cf. Section 3.3), as stated in Lemma 4.

LEMMA 4. *The bound function $\mathcal{FQ}_P(\mathbf{q}, Q)$ (cf. Equation 7) can be computed in $O(d)$ time.*

5.2.1 *Tighter quadratic upper bound function.*

To ensure $Q^U(x)$ to be the correct and tight upper bound of $\max(1 - x, 0)$, the quadratic function should pass through two points $(x_{min}, \max(1 - x_{min}, 0))$ and $(x_{max}, \max(1 - x_{max}, 0))$, as shown in Figure 10.

Therefore, we can obtain the parameters a_u and c_u by some algebraic operations:

$$a_u = \frac{\max(1 - x_{max}, 0) - \max(1 - x_{min}, 0)}{x_{max}^2 - x_{min}^2}$$

$$c_u = \frac{x_{max}^2 \max(1 - x_{min}, 0) - x_{min}^2 \max(1 - x_{max}, 0)}{x_{max}^2 - x_{min}^2}$$

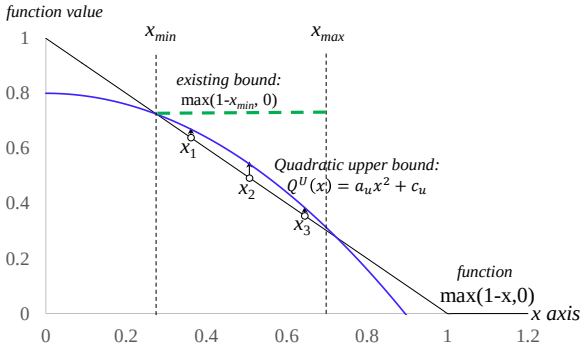


Figure 10: Correct and tight quadratic upper bound function (Green line represents the upper bound $UB_R(\mathbf{q})$ (cf. Equation 6))

Observe from Figure 10, our quadratic upper bound function $\mathcal{FQ}_P(\mathbf{q}, Q^U)$ can be tighter than existing bound function $UB_R(\mathbf{q})$ (cf. Lemma 5).

LEMMA 5. *Given the above a_u and c_u, our quadratic upper bound function is tighter than $UB_R(\mathbf{q})$, i.e., $\mathcal{FQ}_P(\mathbf{q}, Q^U) \leq UB_R(\mathbf{q})$.*

5.2.2 Tighter quadratic lower bound function. We proceed to develop the quadratic lower bound function $Q^L(x) = a_l x^2 + c_l$ for $\max(1-x, 0)$. To simplify the discussion, we first let all $x_i = \gamma dist(\mathbf{q}, \mathbf{p_i}) \leq 1$, i.e., all points are in the linear line $1-x$ region, but not the zero region (cf. Figure 9). To achieve the better lower bound in this case, we can shift the quadratic curve until it touches the function $1-x$, as shown in Figure 11.

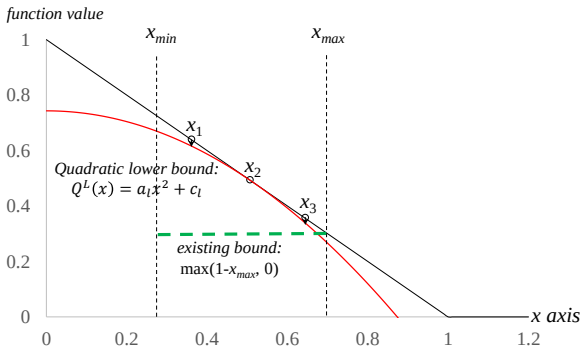


Figure 11: Correct and tight quadratic lower bound function (Green line represents the lower bound $LB_R(\mathbf{q})$ (cf. Equation 5))

Therefore, there is only one root in the following equation.

$$a_l x^2 + c_l = 1 - x \implies a_l x^2 + x + c_l - 1 = 0$$

which also implies:

$$c_l = 1 + \frac{1}{4a_l} \tag{8}$$

Based on the concave property of $Q^L(x)$, we immediately have $1 - x \geq Q^L(x)$, which can further achieve the correct lower bound property $\mathcal{F}_P(\mathbf{q}) \geq \mathcal{FQ}_P(\mathbf{q}, Q^L)$.

Observe from Equation 8, a_l can still be varied which can affect the lower bound value $\mathcal{FQ}_P(\mathbf{q}, Q^L)$. In Theorem 2, we claim that we can obtain the tightest lower bound by setting $a_l = a_l^*$ (with $O(d)$ time). We leave the proof in Section 9.4.

THEOREM 2. *Let $\mathcal{FQ}_P(\mathbf{q}, Q^L)$ be the function of a_l with $c_l = 1 + \frac{1}{4a_l}$ (cf. Equation 8). We can obtain the tightest lower bound when $a_l = a_l^*$, where:*

$$a_l^* = -\sqrt{\frac{|P|}{4\gamma^2 \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2}} \tag{9}$$

Apparently, our new bound function is tighter compared with existing $LB_R(\mathbf{q})$ (cf. green line in Figure 11). In Lemma 6, we claim it is always true and leave the proof in Section 9.5.

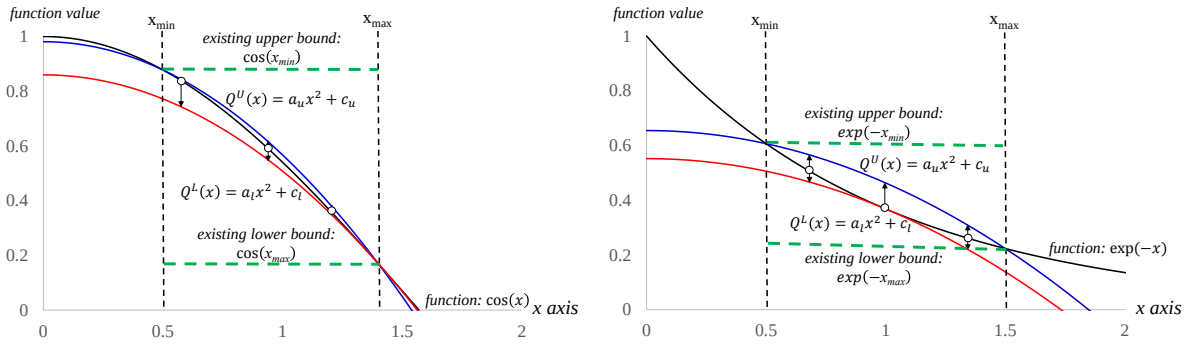
LEMMA 6. *If $x_i = \gamma dist(\mathbf{q}, \mathbf{p_i}) \leq 1$ for all $\mathbf{p_i} \in P$ and $Q^L(x) = a_l^* x + c_l^*$ (where $c_l^* = 1 + \frac{1}{4a_l^*}$), our bound function $\mathcal{FQ}_P(\mathbf{q}, Q^L)$ is tighter than $LB_R(\mathbf{q})$, i.e., $\mathcal{FQ}_P(\mathbf{q}, Q^L) \geq LB_R(\mathbf{q})$.*

In above discussion, we have the assumption that $x_i = \gamma dist(\mathbf{q}, \mathbf{p}_i) \leq 1$. In fact, our bound function can be even negative when $x_i > 1$. However, since $\mathcal{F}_P(\mathbf{q})$ must be non-negative, we can directly set the bound value to be 0, the same as $LB_R(\mathbf{q})$, when $x_i > 1$. This explains why we can always get the tighter lower bound compared with $LB_R(\mathbf{q})$.

5.2.3 Cosine and exponential kernels. We can utilize the similar techniques to develop the lower and upper bounds for the kernel aggregation function $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 4) with cosine and exponential kernel functions (cf. Table 4). By finding the suitable coefficients of $Q^L(x) = a_l x^2 + c_l$ and $Q^U(x) = a_u x^2 + c_u$ (cf. Figure 12), we can obtain the tighter lower and upper bounds $\mathcal{FQ}_P(\mathbf{q}, Q)$ (cf. Equation 7), in $O(d)$ time, for $\mathcal{F}_P(\mathbf{q})$.

6 PROGRESSIVE VISUALIZATION FRAMEWORK FOR KDV

Even though KDV has been extensively studied in the literature [7, 13, 17, 54–56] and adopted in different types of software, e.g. Scikit-learn, QGIS and ArcGIS, they only focus on developing the fast algorithms for evaluating the kernel aggregation functions and all pixels are evaluated in order to generate the color map. However, it can be time consuming to evalaute all pixels, especially for high-resolution screen (e.g., 2560×1920). Instead of waiting for a long time to obtain ϵKDV or τKDV, users (e.g., scientists) want to continuously visualize some partial/coarse results [36] for exploring different pairs of attributes and they can terminate the process at any time t once the visualization results are satisfactory or not useful. Even though existing work in KDV [26, 36] also support this type of interactive visualization, they only focus on using the GPU [26, 36] and distributed algorithm



(a) Quadratic bounds for $\cos(x)$ (b) Quadratic bounds for $\exp(-x)$

Figure 12: Quadratic bounds for cosine and exponential kernel functions

[36] to achieve real-time performance. In this section, we show that, by considering the proper pixel evaluation order, the progressive visualization framework can achieve high visualization quality in single machine setting without using GPU and parallel computation, even though the time t is very small.

To simplify the discussion, we assume the resolution for the visualized region is $2^r \times 2^r$, where r is the positive integer. However, our method can also handle all other resolutions. Instead of using row/column-major order to evaluate density value of each pixel in the visualized region, we adopt the quad-tree like order [12] to perform the evaluation since two close spatial coordinates normally have similar KDE value (cf. Figure 1). Initially, our algorithm evaluates the approximate KDE value (e.g., $\epsilon = 0.01$) in the central pixel (cf. (1) in Figure 13) as an approximation in the whole region. Then, it iteratively evaluates more density values ((2), (3), (4), (5)... in Figure 13) when more time is provided. Our algorithm stops once the user terminates the process or all density values (pixels) are evaluated.

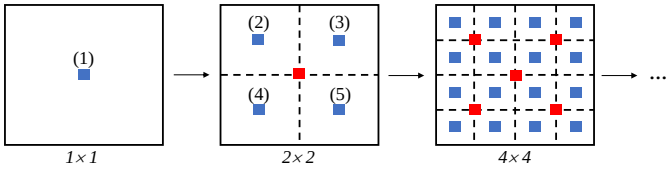


Figure 13: A progressive approach to evaluate the density value of each pixel (blue) in the visualized region (with order (1), (2),...), each density value of blue pixels represents the density value in the corresponding sub-region, except for red pixels, in which the density values have been evaluated.

7 EXPERIMENTAL EVALUATION

We first introduce the experimental setting in Section 7.1. Later, we demonstrate the efficiency performance in different methods for ϵKDV and τKDV in Section 7.2. After that, we compare the tightness of the state-of-the-art bound functions KARL and our proposal QUAD in Section 7.3. Next, we provide the quality comparison with QUAD and other methods in Section 7.4. Then, we demonstrate the quality performance of progressive visualization framework with different methods in Section 7.5. After that, we test the efficiency performance for other kernel functions (e.g., triangular and cosine kernel functions) in Section 7.6. Lastly, we further test whether our solution QUAD can still be efficient, compared with other methods, for general kernel density estimation, with higher dimensions in Section 7.7.

7.1 Experimental Setting

We use four large-scale real datasets (up to 7M) for conducting the experiments, as shown in Table 5. In the following experiments, we choose two attributes for each dataset for visualization. We adopt the Scott's rule [7, 13] to obtain the parameter γ and the weighting parameter w. By default, we set the resolution to be 1280×960. In addition, we focus on Gaussian kernel function in Sections 7.2-7.5, 7.7 and other kernel functions in Section 7.6.

Table 5: Datasets

Name	n	Selected attributes (2d)
El nino [2]	178080	sea surface temperature (depth=0/500)
crime [1]	270688	latitude/longitude
home [2]	919438	temperature/humidity
hep [2]	7000000	$1^{st}/2^{nd}$ dimensions

In our experimental study, we compare different state-of-the-art methods with our solution, as shown in Table 6. EXACT is the sequential scan method, which does not adopt any efficient algorithm. Scikit-learn (abbrev. Scikit) [35] is the machine learning software which can also support ϵKDV. Z-order [54, 55] is the state-of-the-art dataset sampling method which provides probabilistic error guarantee for ϵKDV. tKDC [13] and aKDE [17] are indexing-based methods for τKDV and ϵKDV respectively. In the offline stage, they pre-build one index, e.g., kd-tree, on the dataset. Each index node stores

the information, e.g., bounding rectangles, for the bound functions. This approach facilitates the bound evaluations in the online stage. Both KARL [7] and this paper also follow this approach and utilize the indexing structure to provide speedup for bound evaluations. The main difference between our work QUAD and existing work aKDE, tKDC and KARL is the newly developed tighter bound functions. We implemented all methods in C++ (except for Scikit, which is originally implemented in Python) and conducted experiments on an Intel i7 3.4GHz PC using Ubuntu. In this paper, we use the response time (sec) to measure the efficiency of all methods and only report the response time which is smaller than 7200 sec (i.e., 2 hours).

Table 6: Existing methods for two variants of KDV

Type	EXACT	Scikit [35]	Z-Order [54, 55]	aKDE [17]	tKDC [13]	KARL [7]	QUAD (ours)
ϵKDV	✓	✓	✓	✓	×	✓	✓
τKDV	✓	×	×	×	✓	✓	✓

7.2 Efficiency for ϵKDV and τKDV

In this section, we investigate the following four research questions of efficiency issues for ϵKDV and τKDV.

(1) How does the relative error ϵ affect the efficiency performance of all methods in ϵKDV?
(2) How does the threshold τ affect the efficiency performance of all methods in τKDV?
(3) How scalable can QUAD achieve in different resolutions compared with all other existing methods?
(4) How scalable can QUAD achieve in different dataset sizes compared with all other existing methods?

As a remark, both EXACT and Scikit always run out of time (> 7200 sec). Therefore, these two curves are not shown in most of the following experimental figures.

Varying ϵ for ϵKDV:

We vary the relative error ϵ for ϵKDV from 0.01 to 0.05. Figure 14 shows the response time of all methods. Even though Z-Order method downsamples the original dataset to the small scale dataset, they still need to evaluate the exact KDE (EXACT) in this reduced dataset for each pixel. Therefore, the evaluation time is still long compared with our method QUAD. On the other hand, due to the superior tightness for our bounds compared with the state-of-the-art bound functions (cf. Sections 4.2 and 4.3), QUAD can provide another one order of magnitude speedup compared with KARL. Even though we choose the relative error ϵ to be 0.01, which is very small, QUAD can achieve 100-400 sec for each large-scale dataset in a single machine.

Varying τ for τKDV:

In order to test the efficiency performance for τKDV, we select seven thresholds ($\mu-0.3\sigma$, $\mu-0.2\sigma$, $\mu-0.1\sigma$, μ, $\mu+0.1\sigma$, $\mu+0.2\sigma$, $\mu+0.3\sigma$) for each dataset, where:

$$\mu = \frac{\sum_{\mathbf{q}\in Q} \mathcal{F}_P(\mathbf{q})}{|Q|} \text{ and } \sigma = \sqrt{\frac{\sum_{\mathbf{q}\in Q}(\mathcal{F}_P(\mathbf{q}) - \mu)^2}{|Q|}}$$

Figure 15 shows the time performance for all seven thresholds. We can observe that our method QUAD can provide at least one-order of magnitude speedup compared with existing methods tKDC and KARL regardless of the chosen threshold. Our method can attain at most 10 sec for τKDV.

Varying the resolution:

We investigate the scalability issue for different resolutions. Four resolutions, 320×240, 640×480, 1280×960 and 2560×1920, are chosen in this experiment. With higher resolution, the larger the response time. We conduct this experiment in ϵKDV with fixed relative error $\epsilon = 0.01$, as shown in Figure 16. No matter which resolution we choose, our method QUAD can provide significant speedup compared with all existing methods.

Varying the dataset size:

We proceed to test how the dataset size affects the efficiency performance of both ϵKDV and τKDV. We choose the largest dataset hep (in Table 5) and vary the size of the datasets via sampling, the sample sizes are 1M, 3M, 5M and 7M. We fix ϵ and τ to be 0.01 and μ respectively in this experiment. Figure 17 shows the response time. Our method QUAD outperforms the existing methods by one order of magnitude speedup in ϵKDV and τKDV in different dataset sizes.

7.3 Tightness Analysis for Bound Functions

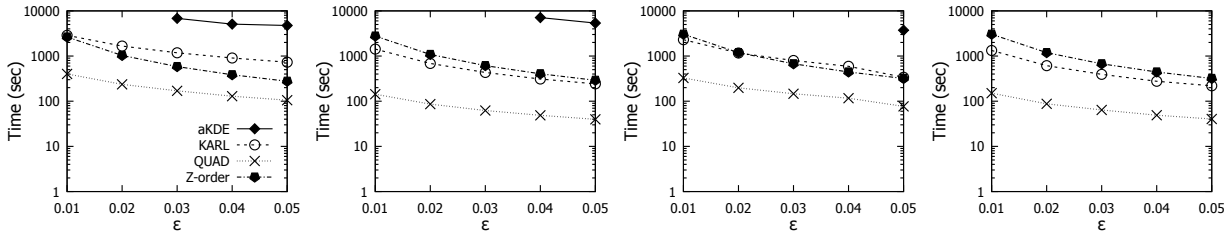
Recall from Section 4, we have already shown that our developed bound functions LB_{QUAD} and UB_{QUAD} are tighter than the state-of-the-art bound functions LB_{KARL} and UB_{KARL}. In this section, we provide the case study to see how tight our bound functions can achieve using the existing indexing framework with kd-tree (cf. Section 3.2) in ϵKDV, with $\epsilon = 0.01$. We sample one pixel with the highest KDE value in home dataset. We test the changes of the bound value by varying different iterations. Observe from Figure 18, QUAD can stop significantly earlier than previous method KARL which also justifies why QUAD can perform significantly faster than KARL in previous experiments.

7.4 Quality of Different KDV Methods

We proceed to compare the visualization quality between different methods in ϵKDV. Since all methods aKDE, Z-order, KARL and QUAD provide the error guarantee between the returned density value and the exact result, all these methods do not degrade the quality of visualization for very small relative error, e.g., $\epsilon = 0.01$, as shown in Figure 19.

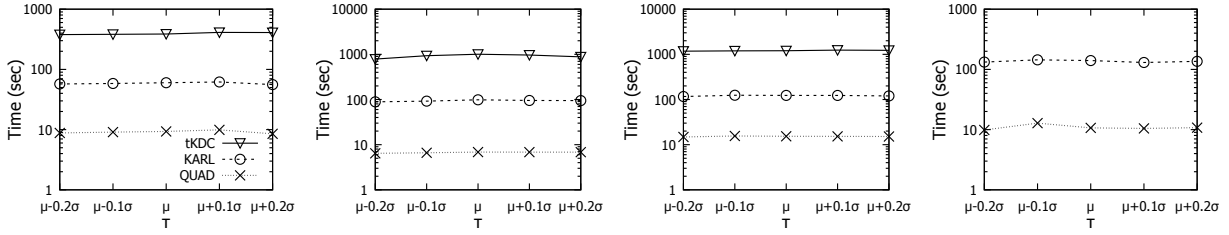
7.5 Progressive Visualization Framework

In this section, we test the progressive visualization framework (cf. Section 6) with different existing methods, including EXACT, aKDE, KARL, Z-Order and our method QUAD. First, we test the quality with five different timestamps (0.01 sec, 0.05 sec, 0.25 sec, 1.25 sec and 6.25 sec) in all datasets,



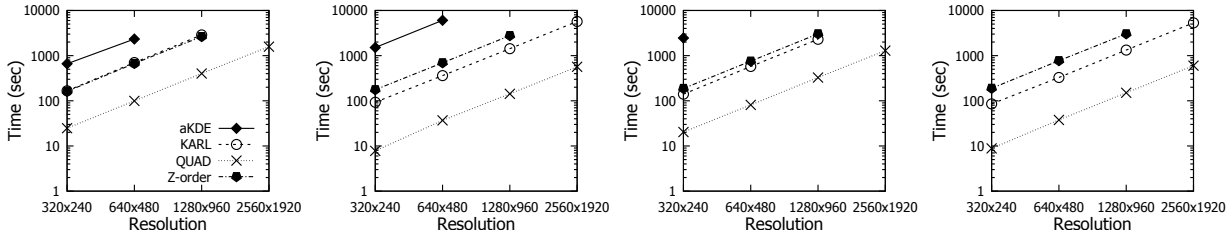
(a) El nino (b) crime (c) home (d) hep

Figure 14: Response time for ϵKDV with resolution 1280×960, varying the relative error ϵ



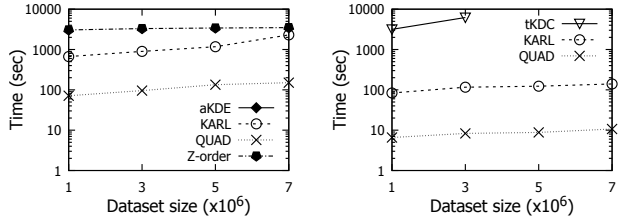
(a) El nino (b) crime (c) home (d) hep

Figure 15: Response time for τKDV with resolution 1280×960, varying the threshold τ



(a) El nino (b) crime (c) home (d) hep

Figure 16: Response time for ϵKDV with fixed relative error $\epsilon = 0.01$, varying the resolution



(a) ϵKDV (b) τKDV

Figure 17: Response time for ϵKDV with $\epsilon = 0.01$ and τKDV with $\tau = \mu$ in hep dataset, varying the dataset size

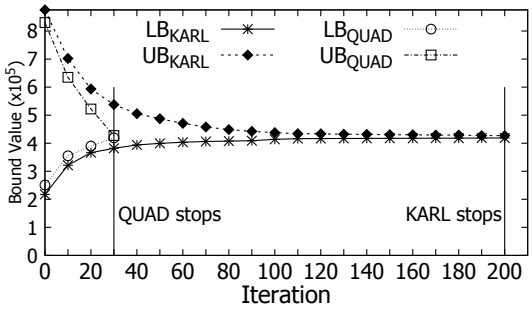


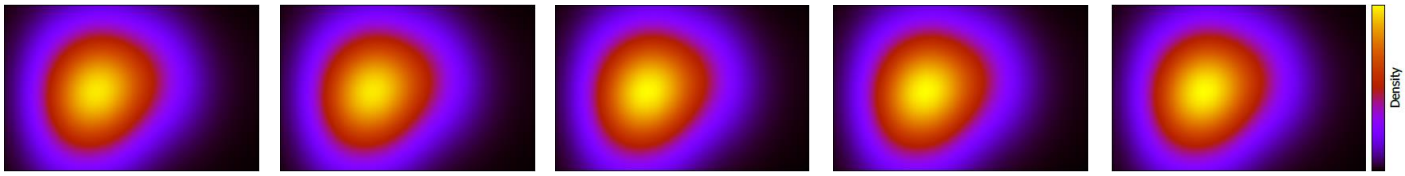
Figure 18: Bound values of KARL and QUAD v.s. the number of iterations in ϵKDV, $\epsilon = 0.01$, in home dataset

as shown in Figure 20. We use the average relative error $\frac{1}{|Q|}\sum_{\mathbf{q}\in Q}\frac{|R(\mathbf{q})-\mathcal{F}_P(\mathbf{q})|}{\mathcal{F}_P(\mathbf{q})}$ for the quality measure, where $R(\mathbf{q})$ is the returned result of pixel $\mathbf{q}$. For each approximation method, we select the relative error parameter $\epsilon = 0.01$. In this experiment, we fix the resolution to be 1280×960. Since QUAD is faster than all other methods, it can evaluate more pixels under the same time limit t. Therefore, it explains why the average relative error is smaller than other methods with the same t.

Figure 21 shows five visualization figures, which correspond to five timestamps (0.02 sec, 0.05 sec, 0.2 sec, 0.5 sec and 2 sec), in home dataset with our best method QUAD. We can notice that once the time t is set to be 0.5 sec, QUAD can already be able to produce the reasonable visualization result.

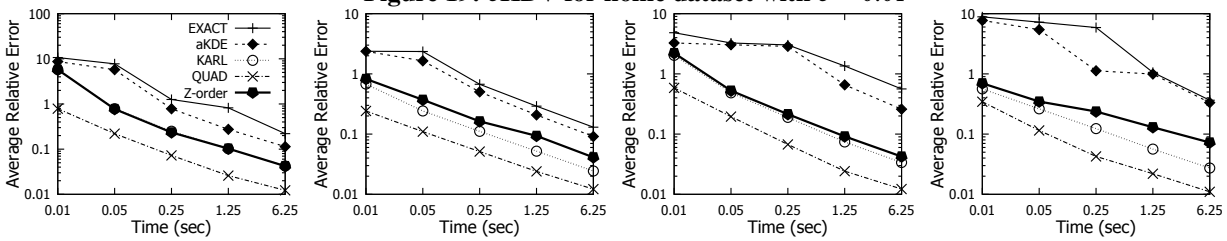
7.6 Efficiency for Other Kernel Functions

In this section, we conduct the efficiency experiments for other kernel functions, as stated in Table 4. Recall from Section 5.1, KARL [7] cannot provide the efficient linear bounds for these kernel functions. Therefore, we omit the comparison of this method in this section. Due to the space limitation, we only report the results for triangular and cosine kernel functions in this section.



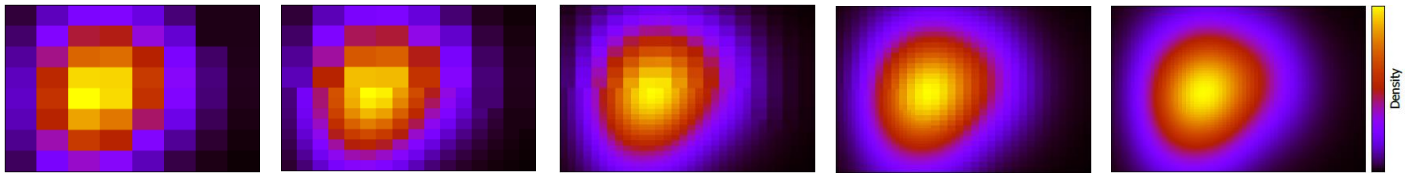
Exact aKDE KARL QUAD Z-Order

Figure 19: ϵKDV for home dataset with $\epsilon = 0.01$



(a) El nino (b) crime (c) home (d) hep

Figure 20: Average relative error for different methods under progressive visualization framework, varying five timestamps t



t = 0.02 sec t = 0.05 sec t = 0.2 sec t = 0.5 sec t = 2 sec

Figure 21: QUAD-based progressive visualization in home dataset, varying five timestamps t

In the first experiment, we vary different ϵ (from 0.01 to 0.05) and test the efficiency performance of different methods, including EXACT, aKDE, Z-Order and QUAD, in different kernel functions. Figure 22 reports the results for ϵKDV in crime and hep datasets with triangular and cosine kernels. Observe that the response time of our method QUAD can still outperform aKDE by at least one-order-of-magnitude faster since our quadratic lower and upper bounds are theoretically tighter than these bound functions used in aKDE but the time complexity still remains the same, i.e., $O(d)$ time. On the other hand, QUAD is also faster than the state-of-the-art data sampling method Z-Order, especially for small ϵ (e.g., 0.01). Even though Z-Order can significantly reduce the dataset size, it still needs to run the method EXACT for the reduced dataset, which can still be slow once the relative error ϵ is small.

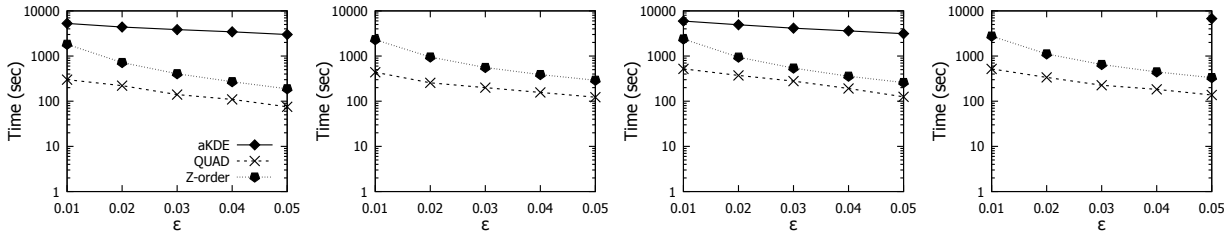
In the second experiment, we test the response time for τKDV in different methods, including tKDC and QUAD. We vary different thresholds τ (from $\mu - 0.2\sigma$ to $\mu + 0.2\sigma$) in this experiment. Observe from Figure 23, our method QUAD can outperform the state-of-the-art method tKDC by at least one-order-of-magnitude.

7.7 Efficiency for Kernel Density Estimation

In this section, we proceed to test whether our method QUAD can be efficiently applied in kernel density estimation with higher dimensions in ϵKDV. We choose the home and hep datasets [2] (cf. Table 5), and follow the existing work [7, 13] to vary the dimensionality of these datasets via PCA dimensionality reduction method and use throughput (queries/sec) to measure the efficiency. Observe from Figure 24, once the dimensionality of the datasets increases, the response throughput of bound-based methods aKDE, KARL and QUAD decreases. However, our method QUAD can still outperform other methods for both large-scale datasets (million-scale) with 10 dimensions in single machine setting. As a remark, we omit Z-Order [54] in this experiment, as this method only focuses on one to two-dimensional setting. Even though QUAD may not scale well with respect to the dimensionality, KDE is normally applied for low-dimensional setting (e.g., $d \leq 6$ [16]) in real applications [13] due to the curse of dimensionality issue [16, 43].

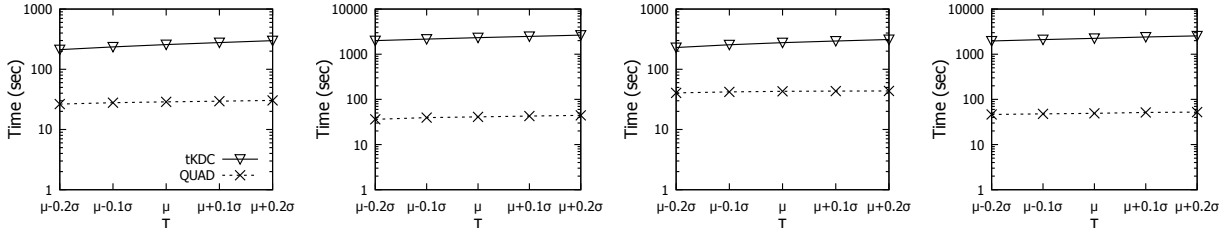
8 CONCLUSION

In this paper, we study the kernel density visualization (KDV) problem with two widely-used variants of KDV, namely approximate kernel density visualization (ϵKDV) and thresholded kernel density visualization (τKDV). The contribution of our work is to develop QUAD which consists of the tightest lower and upper bound functions compared with the existing bound functions [7, 13, 17] with evaluation time complexity $O(d^2)$ for Gaussian kernel, which is negligible for KDV applications, and $O(d)$ for other kernels (e.g., triangular and cosine



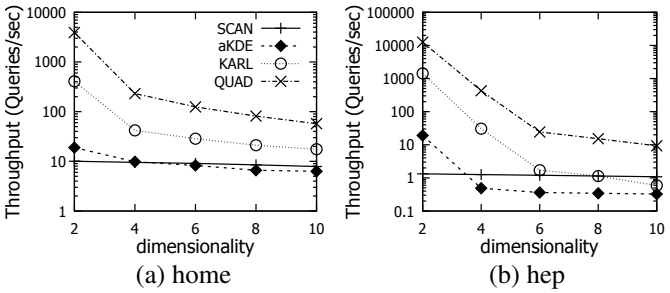
(a) crime (triangular) (b) hep (triangular) (c) crime (cosine) (d) hep (cosine)

Figure 22: Response time for different methods in crime and hep datasets, varying the relative error ϵ and using triangular and cosine kernel functions



(a) crime (triangular) (b) hep (triangular) (c) crime (cosine) (d) hep (cosine)

Figure 23: Response time for different methods in crime and hep datasets, varying the threshold τ and using triangular and cosine kernel functions



(a) home (b) hep

Figure 24: Response throughput (queries/sec) for different methods in home and hep datasets (with Gaussian kernel and $\epsilon = 0.01$), varying the dimensionality

kernels). Our method QUAD can provide at least one-order-of-magnitude speedup compared with different state-of-the-art methods under small relative error ϵ and different thresholds τ. The combination of QUAD and progressive visualization framework can further provide reasonable visualization results with real-time performance (0.5 sec) in single machine setting without using GPU and parallel computation.

In the future, we will further apply QUAD to other kernel-based machine learning models, e.g., kernel regression, kernel SVM and kernel clustering. Moreover, we will explore the oppontunity to utilize the parallel/distributed computation [54] and modern hardware [16, 52] to further speed up our solution.

9 PROOFS

9.1 Proof of Lemma 2

PROOF. We only prove the upper bound $\mathcal{F}_P(\mathbf{q}) \leq \mathcal{FQ}_P(\mathbf{q}, Q^U)$ but it can be extended to lower bound in a straightforward way. We first substitute $x = \gamma dist(\mathbf{q}, \mathbf{p_i})^2$ in the following inequality, $\exp(-x) \leq Q^U(x) = a_u x^2 + b_u x + c_u$. Then, we have:

$$\exp(-\gamma dist(\mathbf{q}, \mathbf{p_i})^2) \leq a_u(\gamma dist(\mathbf{q}, \mathbf{p_i})^2)^2 + b_u \gamma dist(\mathbf{q}, \mathbf{p_i})^2 + c_u$$

By taking the summation in both sides with respect to each $\mathbf{p_i} \in P$ and then multiplying both sides with the constant w, we can prove $\mathcal{F}_P(\mathbf{q}) \leq \mathcal{FQ}_P(\mathbf{q}, Q^U)$. □

9.2 Proof of Lemma 3

PROOF. From Equation 2, we have:

$$\mathcal{FQ}_P(q, Q) = wa\gamma^2 \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^4 + wb\gamma \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2 + cw|P|$$

Recall from Section 3.3, $\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2$ can be computed in $O(d)$ time. We proceed to show $\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^4$ can be computed in $O(d^2)$ time.

$$\begin{aligned}&\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^4 \\ &= |P|||\mathbf{q}||^4 - 4||\mathbf{q}||^2 \mathbf{q}^T \mathbf{a_P} - 4\mathbf{q}^T \mathbf{v_P} + 2||\mathbf{q}||^2 b_P + h_P + 4 \sum_{\mathbf{p_i} \in P} (\mathbf{q}^T \mathbf{p_i})^2\end{aligned}$$

where $\mathbf{a_P} = \sum_{\mathbf{p_i} \in P} \mathbf{p_i}$, $\mathbf{v_P} = \sum_{\mathbf{p_i} \in P} ||\mathbf{p_i}||^2 \mathbf{p_i}$, $b_P = \sum_{\mathbf{p_i} \in P} ||\mathbf{p_i}||^2$ and $h_P = \sum_{\mathbf{p_i} \in P} ||\mathbf{p_i}||^4$. All of these terms only depend on P and can be computed once and stored when we build the indexing structure (cf. Figure 3). As such, the first five terms can be computed in $O(d)$ time in the query stage. We now show that the last term can be computed in $O(d^2)$ time.

$$\sum_{\mathbf{p_i} \in P} (\mathbf{q}^T \mathbf{p_i})^2 = \sum_{\mathbf{p_i} \in P} (\mathbf{q}^T \mathbf{p_i})(\mathbf{p_i}^T \mathbf{q}) = \mathbf{q}^T \Big(\sum_{\mathbf{p_i} \in P} \mathbf{p_i} \mathbf{p_i}^T \Big) \mathbf{q} = \mathbf{q}^T \mathbf{C} \mathbf{q}$$

where C is the matrix which depends on P and can be computed when we build the indexing structure.

In the query stage, the computation time of $\mathbf{q}^T\mathbf{C}\mathbf{q}$ is in $O(d^2)$ time. Hence, the time complexity for evaluating $\mathcal{FQ}_P(q, Q)$ is $O(d^2)$. □

9.3 Proof of Theorem 1

PROOF. We first investigate the slope (1^{st} derivative) curves of both $\exp(-x)$ and $Q^U(x) = a_u x^2 + b_u x + c_u$ which are $-\exp(-x)$ and $2a_u x + b_u$ respectively, as shown in Figure 25.

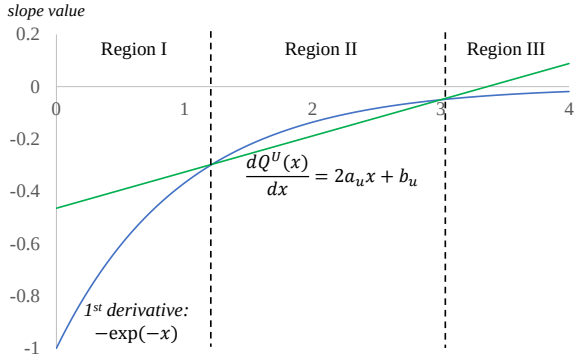


Figure 25: The slope curves of both $Q^U(x)$ and $\exp(-x)$

In general, $2a_u x + b_u$ may not intersect with $-\exp(-x)$ by two points. However, it is impossible here. Once the slope of $Q^U(x)$ is always larger than the slope of $\exp(-x)$, $Q^U(x)$ and $\exp(-x)$ can only intersect with at most one point. However, $Q^U(x)$ must intersect with $\exp(-x)$ by $(x_{min}, \exp(-x_{min}))$ and $(x_{max}, \exp(-x_{max}))$, which leads to contradiction. As such, $2a_u x + b_u$ must intersect with $-\exp(-x)$ by two points.

Observe from Figure 25, the slope of one curve is always larger than another one in each region. Therefore, once they have the intersection point in this region, they must not have another intersection point again in this region as one curve always move "faster" than another one.

LEMMA 7. *For each region in Figure 25, there is at most one intersection point in* $\exp(-x)$ *and* $Q^U(x)$.

Once we have Lemma 7, we can use it to prove this theorem (correctness and tightness).

Correct upper bound for our chosen a_u^*: Observe from Figure 7, we have two conditions for the correct upper bound function.

- The slope of quadratic function $\frac{dQ^U(x)}{dx}$ at the point $(x_{max}, \exp(-x_{max}))$ must be more negative than the slope of $\exp(-x)$.
- There is no other intersection point, except $(x_{min}, \exp(-x_{min}))$ and $(x_{max}, \exp(-x_{max}))$, for the functions $\exp(-x)$ and $Q^U(x)$ in the interval $[x_{min}, x_{max}]$.

Based on the first condition, x_{max} must be in region II in Figure 25. By Lemma 7, x_{min} can only be in region I and there is no other intersection point between x_{min} and x_{max}, which fulfills the second condition. Therefore, we conclude:

LEMMA 8. *If* $Q^U(x)$ *is the proper upper bound of* $\exp(-x)$, x_{max} *must be in region II.*

Since x_{max} must be in region II, we have:

$$\begin{aligned} \frac{dQ^U(x)}{dx}\Big|_{x=x_{max}} &\leq -\exp(-x_{max}) \\ 2a_u x_{max} + b_u &\leq -\exp(-x_{max}) \end{aligned}$$

By substituting b_u in terms of the function of a_u (cf. Section 4.2), we have $a_u \leq a_u^*$. Therefore, our selected a_u^* is within this region and hence it achieves correct upper bound function.

The tighter upper bound for our chosen $a_u = a_u^*$: To prove this part, we substitute b_u and c_u with respect to a_u into $Q^U(x) = a_u x^2 + b_u x + c_u$. Then, we have:

$$Q^U(x) = a_u(x - x_{min})(x - x_{max}) + m_u x + k_u$$

where m_u and k_u are the slope and intercept, respectively, of the linear line (chord) which passes through $(x_{min}, \exp(-x_{min}))$ and $(x_{max}, \exp(-x_{max}))$.

Note that only the first term in $Q^U(x)$ depends on a_u and the term $(x - x_{min})(x - x_{max}) < 0$ for every x in the range $[x_{min}, x_{max}]$. Since $a_u > 0$, $Q^U(x)$ is smaller once a_u is larger, and thus the upper bound is tighter. However, as stated in the correctness proof, the largest possible a_u should be a_u^*. Hence, we can achieve the tighter bound once $a_u = a_u^*$, since the linear function $m_u x + k_u$ is in fact the special case of $Q^U(x)$ with $a_u = 0 \leq a_u^*$. □

9.4 Proof of Theorem 2

PROOF. Let $H(a_l) = \mathcal{FQ}_P(\mathbf{q}, Q^L)$, we have:

$$\begin{aligned} H(a_l) &= \sum_{\mathbf{p_i} \in P} w\left(a_l(\gamma dist(\mathbf{q}, \mathbf{p_i}))^2 + \left(1 + \frac{1}{4a_l}\right)\right) \\ \frac{dH(a_l)}{da_l} &= w\gamma^2 \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2 - \frac{w|P|}{4a_l^2} \end{aligned}$$

By setting $\frac{dH(a_l)}{da_l} = 0$, we can obtain:

$$a_l = a_l^* = -\sqrt{\frac{|P|}{4\gamma^2 \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2}}$$

Based on the basic differentiation theory, we conclude that $a_l = a_l^*$ can achieve the maximum for $\mathcal{FQ}_P(\mathbf{q}, Q^L)$. □

9.5 Proof of Lemma 6

PROOF. By substituting a_l^* (cf. Equation 9) and c_l^* (cf. Equation 8) in $\mathcal{FQ}_P(\mathbf{q}, Q^L)$ (cf. Equation 7), we can obtain:

$$\begin{aligned} \mathcal{FQ}_P(\mathbf{q}, Q^L) &= w|P| - w\sqrt{|P| \sum_{\mathbf{p_i} \in P} (\gamma dist(\mathbf{q}, \mathbf{p_i}))^2} \\ &\geq w|P|(1 - x_{max}) = LB_R(\mathbf{q}) \end{aligned}$$

The last equality is based on the assumption $x_i = \gamma dist(\mathbf{q}, \mathbf{p_i}) \leq 1$. □

REFERENCES

[1] Atlanta police department open data. http://opendata.atlantapd.org/.

[2] UCI machine learning repository. http://archive.ics.uci.edu/ml/index.php.

[3] Comparison of density estimation methods for astronomical datasets. *Astronomy and Astrophysics*, 531, 7 2011.

[4] S. Chainey, L. Tompson, and S. Uhlig. The utility of hotspot mapping for predicting spatial patterns of crime. *Security Journal*, 21(1):4–28, Feb 2008.

[5] T. N. Chan, M. L. Yiu, and K. A. Hua. A progressive approach for similarity search on matrix. In *SSTD*, pages 373–390. Springer, 2015.

[6] T. N. Chan, M. L. Yiu, and K. A. Hua. Efficient sub-window nearest neighbor search on matrix. *IEEE Trans. Knowl. Data Eng.*, 29(4):784–797, 2017.

[7] T. N. Chan, M. L. Yiu, and L. H. U. KARL: fast kernel aggregation queries. In *ICDE*, pages 542–553, 2019.

[8] W. Chen, F. Guo, and F. Wang. A survey of traffic data visualization. *IEEE Trans. Intelligent Transportation Systems*, 16(6):2970–2984, 2015.

[9] E. Cheney and W. Light. *A Course in Approximation Theory*. Mathematics Series. Brooks/Cole Publishing Company, 2000.

[10] K. Cranmer. Kernel estimation in high-energy physics. 136:198–207, 2001.

[11] M. D. Felice, M. Petitta, and P. M. Ruti. Short-term predictability of photovoltaic production over italy. *Renewable Energy*, 80:197 – 204, 2015.

[12] S. Frey, F. Sadlo, K. Ma, and T. Ertl. Interactive progressive visualization with space-time error control. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2397–2406, 2014.

[13] E. Gan and P. Bailis. Scalable kernel density classification via threshold-based pruning. In *ACM SIGMOD*, pages 945–959, 2017.

[14] E. R. Gansner, Y. Hu, S. C. North, and C. E. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *PacificVis*, pages 187–194, 2011.

[15] W. Gong, D. Yang, H. V. Gupta, and G. Nearing. Estimating information entropy for hydrological data: One-dimensional case. *Water Resources Research*, 50(6):5003–5018, 2014.

[16] A. Gramacki. *Nonparametric Kernel Density Estimation and Its Computational Aspects*. Studies in Big Data. Springer International Publishing, 2017.

[17] A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *SDM*, pages 203–211, 2003.

[18] T. Guo, K. Feng, G. Cong, and Z. Bao. Efficient selection of geospatial data on maps for interactive and visualized exploration. In *SIGMOD*, pages 567–582, 2018.

[19] T. Guo, M. Li, P. Li, Z. Bao, and G. Cong. Poisam: a system for efficient selection of large-scale geospatial data on maps. In *SIGMOD*, pages 1677–1680, 2018.

[20] T. Hart and P. Zandbergen. Kernel density estimation and hotspot mapping: examining the influence of interpolation method, grid cell size, and bandwidth on crime forecasting. *Policing: An International Journal of Police Strategies and Management*, 37:305–323, 2014.

[21] Q. Jin, X. Ma, G. Wang, X. Yang, and F. Guo. Dynamics of major air pollutants from crop residue burning in mainland china, 2000–2014. *Journal of Environmental Sciences*, 70:190 – 205, 2018.

[22] S. C. Joshi, R. V. Kommaraju, J. M. Phillips, and S. Venkatasubramanian. Comparing distributions and shapes using the kernel distance. In *SOCG*, pages 47–56, 2011.

[23] P. K. Kefaloukos, M. A. V. Salles, and M. Zachariasen. Declarative cartography: In-database map generalization of geospatial datasets. In *ICDE*, pages 1024–1035, 2014.

[24] J. Kehrer and H. Hauser. Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE Trans. Vis. Comput. Graph.*, 19(3):495–513, 2013.

[25] D. A. Keim. Visual exploration of large data sets. *Commun. ACM*, 44(8):38–44, 2001.

[26] O. D. Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *PacificVis*, pages 171–178, 2011.

[27] H. Lee and K. Kang. Interpolation of missing precipitation data using kernel estimations for hydrologic modeling. *Advances in Meteorology*, pages 1–12, 2015.

[28] M. Li, Z. Bao, F. M. Choudhury, and T. Sellis. Supporting large-scale geographical visualization in a multi-granularity way. In *WSDM*, pages 767–770, 2018.

[29] Y.-P. Lin, H.-J. Chu, C.-F. Wu, T.-K. Chang, and C.-Y. Chen. Hotspot analysis of spatial environmental pollutants using kernel density estimation and geostatistical techniques. *International Journal of Environmental Research and Public Health*, 8(1):75–88, 2011.

[30] Y. Ma, M. Richards, M. Ghanem, Y. Guo, and J. Hassard. Air pollution monitoring and mining based on sensor grid in london. *Sensors*, 8(6):3601–3623, 2008.

[31] A. Mayorga and M. Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1526–1538, Sept 2013.

[32] L. Micallef, G. Palmas, A. Oulasvirta, and T. Weinkauf. Towards perceptual optimization of the visual design of scatterplots. *IEEE Trans. Vis. Comput. Graph.*, 23(6):1588–1599, 2017.

[33] Y. Park, M. J. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. In *ICDE*, pages 755–766, 2016.

[34] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: Universalizing approximate query processing. In *SIGMOD*, pages 1461–1476, 2018.

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

[36] A. Perrot, R. Bourqui, N. Hanusse, F. Lalanne, and D. Auber. Large interactive visualization of density functions on big data infrastructure. In *LDAV*, pages 99–106, 2015.

[37] J. M. Phillips. ϵ-samples for kernels. In *SODA*, pages 1622–1632, 2013.

[38] J. M. Phillips and W. M. Tai. Improved coresets for kernel density estimates. In *SODA*, pages 2718–2727, 2018.

[39] J. M. Phillips and W. M. Tai. Near-optimal coresets of kernel density estimates. In *SOCG*, pages 66:1–66:13, 2018.

[40] QGIS Development Team. *QGIS Geographic Information System*. Open Source Geospatial Foundation, 2009.

[41] V. C. Raykar, R. Duraiswami, and L. H. Zhao. Fast computation of kernel estimators. *Journal of Computational and Graphical Statistics*, 19(1):205–220, 2010.

[42] A. D. Sarma, H. Lee, H. Gonzalez, J. Madhavan, and A. Y. Halevy. Efficient spatial sampling of large geographical tables. In *SIGMOD*, pages 193–204, 2012.

[43] D. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. A Wiley-interscience publication. Wiley, 1992.

[44] A. C. Telea. *Data Visualization: Principles and Practice, Second Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2nd edition, 2014.

[45] L. Thakali, T. J. Kwon, and L. Fu. Identification of crash hotspots using kernel density estimation and kriging methods: a comparison. *Journal of Modern Transportation*, 23(2):93–106, Jun 2015.

[46] P. Vermeesch. On the visualisation of detrital age distributions. *Chemical Geology*, 312-313(Complete):190–194, 2012.

[47] I. A. S. Vladislav Kirillovich Dziadyk. *Theory of Uniform Approximation of Functions by Polynomials*. Walter De Gruyter, 2008.

[48] M. Williams and T. Munzner. Steerable, progressive multidimensional scaling. In *InfoVis*, pages 57–64, 2004.

[49] K. Xie, K. Ozbay, A. Kurkcu, and H. Yang. Analysis of traffic crashes involving pedestrians using big data: Investigation of contributing factors and identification of hotspots. *Risk Analysis*, 37(8):1459–1476, 2017.

[50] C. Yang, R. Duraiswami, and L. S. Davis. Efficient kernel machines using the improved fast gauss transform. In *NIPS*, pages 1561–1568, 2004.

[51] H. Yu, P. Liu, J. Chen, and H. Wang. Comparative analysis of the spatial analysis methods for hotspot identification. *Accident Analysis and Prevention*, 66:80 – 88, 2014.

[52] G. Zhang, A. Zhu, and Q. Huang. A gpu-accelerated adaptive kernel density estimation approach for efficient point pattern analysis on spatial big data. *International Journal of Geographical Information Science*, 31(10):2068–2097, 2017.

[53] X. Zhao and J. Tang. Crime in urban areas: A data mining perspective. *SIGKDD Explorations*, 20(1):1–12, 2018.

[54] Y. Zheng, J. Jestes, J. M. Phillips, and F. Li. Quality and efficiency for kernel density estimates in large data. In *SIGMOD*, pages 433–444, 2013.

[55] Y. Zheng, Y. Ou, A. Lex, and J. M. Phillips. Visualization of big spatial data using coresets for kernel density estimates. In *IEEE Symposium on Visualization in Data Science (VDS '17), to appear*. IEEE, 2017.

[56] Y. Zheng and J. M. Phillips. L∞ error and bandwidth selection for kernel density estimates of large data. In *SIGKDD*, pages 1533–1542, 2015.

[57] M. Zinsmaier, U. Brandes, O. Deussen, and H. Strobelt. Interactive level-of-detail rendering of large graphs. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2486–2495, 2012.