FiNANCE FOR ENERGY MARKET RESEARCH CENTRE

Fast and stable multivariate kernel density estimation by fast sum updating

Nicolas Langrené, and Xavier Warin,

Working Paper
RR-FiME-18-04

June 2018

DAUPHINE UNIVERSITÉ PARIS

CREST Centre de Recherche en Économie et Statistique

ÉCOLE POLYTECHNIQUE

EDF

Fast and stable multivariate kernel density estimation by fast sum updating

Nicolas Langrené[*], Xavier Warin[†]

June 6, 2018

Abstract

Kernel density estimation and kernel regression are powerful but computationally expensive techniques: a direct evaluation of kernel density estimates at M evaluation points given N input sample points requires a quadratic $\mathcal{O}(MN)$ operations, which is prohibitive for large scale problems. For this reason, approximate methods such as binning with Fast Fourier Transform or the Fast Gauss Transform have been proposed to speed up kernel density estimation. Among these fast methods, the Fast Sum Updating approach is an attractive alternative, as it is an exact method and its speed is independent of the input sample and the bandwidth. Unfortunately, this method, based on data sorting, has for the most part been limited to the univariate case. In this paper, we revisit the fast sum updating approach and extend it in several ways. Our main contribution is to extend it to the general multivariate case for general input data and rectilinear evaluation grid. Other contributions include its extension to a wider class of kernels, including the triangular, cosine and Silverman kernels, its combination with parsimonious additive multivariate kernels, and its combination with a fast approximate k-nearest-neighbors bandwidth for multivariate datasets. Our numerical tests of multivariate regression and density estimation confirm the speed, accuracy and stability of the method.

Keywords: adaptive bandwidth; fast k-nearest-neighbors; fast kernel density estimation; fast kernel regression; fast kernel summation; balloon bandwidth; multivariate partition; fast convolution

[*]CSIRO Data61, RiskLab Australia, nicolas.langrene@csiro.au

[†]EDF R&D, FiME (Laboratoire de Finance des Marchés de l'Énergie), warin@edf.fr

1 Introduction

Let $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ be a sample of N input points x_i and output points y_i drawn from a joint distribution (X, Y). The kernel density estimator (aka Parzen-Rosenblatt estimator) of the density of X at the evaluation point z is given by:

$$\hat{f}_{\mathrm{KDE}}(z) := \frac{1}{N} \sum_{i=1}^{N} K_h(x_i - z) \tag{1}$$

where $K_h(u) := \frac{1}{h} K\left(\frac{u}{h}\right)$ with kernel K and bandwidth h. The Nadaraya-Watson kernel regression estimator of $\mathbb{E}\left[Y \,|X = z\right]$ is given by:

$$\hat{f}_{\mathrm{NW}}(z) := \frac{\sum_{i=1}^{N} K_h(x_i - z) y_i}{\sum_{i=1}^{N} K_h(x_i - z)} \tag{2}$$

The estimator $\hat{f}_{\mathrm{NW}}(z)$ performs a kernel-weighted local average of the response points y_i that are such that their corresponding inputs x_i are close to the evaluation point z. It can be described as a locally constant regression. More generally, locally linear regressions can be performed:

$$\hat{f}_{\mathrm{L}}(z) := \min_{\alpha(z), \beta(z)} \sum_{i=1}^{N} K_h(x_i - z) \left[y_i - \alpha(z) - \beta(z) x_i\right]^2 \tag{3}$$

In this case, a weighted linear regression is performed for each evaluation point z. Properties and performance of these classical kernel smoothers (1-2-3) can be found in various textbooks, such as Loader (1999), Härdle et al. (2004), Hastie et al. (2009), Scott (2014). The well known computational problem with the implementation of the kernel smoothers (1-2-3) is that their direct evaluation on a set of M evaluation points requires $\mathcal{O}(M \times N)$ operations. In particular, when the evaluation points coincide with the input points, a direct evaluation requires a quadratic $\mathcal{O}(N^2)$ number of operations. To cope with this computational limitation, several approaches have been proposed over the years.

Data binning consists in summarizing the input sample into a set of equally spaced bins, so as to compute the kernel smoothers more quickly on the binned data. This data preprocessing allows for significant speedup, either by Fast Fourier Transform (Wand (1994), Gramacki and Gramacki (2017)) or by direct computation, see Silverman (1982), Scott (1985), Fan and Marron (1994), Turlachand and Wand (1996), Bowman and Azzalini (2003). The ***fast sum updating*** method is based on the sorting of the input data and on a translation

of the kernel from one evaluation point to the next, updating only the input points which do not belong to the intersection of the bandwidths of the two evaluation points, see Gasser and Kneip (1989), Seifert et al. (1994), Fan and Marron (1994), Werthenbach and Herrmann (1998), Chen (2006). The ***Fast Gauss Transform***, also known as Fast Multipole Method, is based on the expansion of the Gaussian kernel to disentangle the input points from the evaluation points and speed up the evaluation of the resulting sums, see Greengard and Strain (1991), Greengard and Sun (1998), Lambert et al. (1999), Yang et al. (2003), Morariu et al. (2009), Raykar et al. (2010), Sampath et al. (2010), Spivak et al. (2010). The ***dual-tree*** method is based on space partitioning trees for both the input dataset and the evaluation points. These tree structures are then used to compute distances between input points and evaluation points more quickly, see Gray and Moore (2001), Gray and Moore (2003), Lang et al. (2005), Lee et al. (2006), Ram et al. (2009), Curtin et al. (2013), Griebel and Wissel (2013), Lee et al. (2014). Among all these methods, the fast sum updating is the only one which is exact (no extra approximation is introduced). Moreover, its speed is independent of the input data, the kernel and the bandwidth. Its main drawback is that the required sorting of the input points has mostly limited this literature to the univariate case. Werthenbach and Herrmann (1998) attempted to extend the method to the bivariate case, under strong limitations, namely rectangular input sample, evaluation grid and kernel support. In this paper, we revisit the fast sum updating approach and extend it to the general multivariate case. This extension requires a rectilinear evaluation grid and kernels with box support, but has no restriction on the input sample and can accommodate adaptive bandwidths. Moreover, it maintains the desirable properties of the fast sum updating approach, making it, so far, the only fast and exact algorithm for multivariate kernel smoothing under general input sample and general bandwidth.

2 Fast sum updating

2.1 Univariate case

In this section, we recall the fast sum updating algorithm in the univariate case. Let $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ be a sample of N input (source) points x_i and output points

y_i, and let $z_1, z_2, \ldots, z_M$ be a set of M evaluation (target) points. We first sort the input points and evaluation points: $x_1 \leq x_2 \leq \ldots \leq x_N$ and $z_1 \leq z_2 \leq \ldots \leq z_M$. In order to compute the kernel density estimator (1), the kernel regression (2) and the locally linear regression (3) for every evaluation point z_j, one needs to compute sums of the type

$$\mathbf{S}_j = \mathbf{S}_j^{p,q} := \frac{1}{N}\sum_{i=1}^{N} K_h(x_i - z_j) x_i^p y_i^q = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{x_i - z_j}{h}\right) x_i^p y_i^q \,, p = 0, 1,\ q = 0, 1 \quad (4)$$

for every $j \in \{1, 2, \ldots, M\}$. The direct, independent evaluation of these sums would require $\mathcal{O}(N \times M)$ operations (a sum of N terms for each $j \in \{1, 2, \ldots, M\}$). The idea of fast sum updating is to use the information from the sum $\mathbf{S}_j$ to compute the next sum $\mathbf{S}_{j+1}$ without going through all the N input points again. We illustrate this idea with the Epanechnikov (parabolic) kernel $K(u) = \frac{3}{4}(1-u^2)\mathbb{1}\{|u| \leq 1\}$. With this choice of kernel:

$$\begin{aligned}
\mathbf{S}_j^{p,q} &= \frac{1}{Nh}\sum_{i=1}^{N} \frac{3}{4}\left(1 - \left(\frac{x_i - z_j}{h}\right)^2\right) x_i^p y_i^q \mathbb{1}\{z_j - h \leq x_i \leq z_j + h\} \\
&= \frac{1}{Nh}\frac{3}{4}\sum_{i=1}^{N}\left(1 - \frac{z_j^2}{h^2} + 2\frac{z_j}{h^2}x_i - \frac{1}{h^2}x_i^2\right) x_i^p y_i^q \mathbb{1}\{z_j - h \leq x_i \leq z_j + h\} \quad (5) \\
&= \frac{3}{4Nh}\left\{\left(1 - \frac{z_j^2}{h^2}\right)\mathcal{S}^{p,q}([z_j - h, z_j + h]) + 2\frac{z_j}{h^2}\mathcal{S}^{p+1,q}([z_j - h, z_j + h]) - \frac{1}{h^2}\mathcal{S}^{p+2,q}([z_j - h, z_j + h])\right\}
\end{aligned}$$

where

$$\mathcal{S}^{p,q}([L, R]) := \sum_{i=1}^{N} x_i^p y_i^q \mathbb{1}\{L \leq x_i \leq R\} \quad (6)$$

These sums $\mathcal{S}^{p,q}([z_j - h, z_j + h])$ can be evaluated quickly from $j = 1$ to $j = M$ as long as the input points x_i and the evaluation points z_j are sorted in increasing order. Indeed,

$$\begin{aligned}
&\mathcal{S}^{p,q}([z_{j+1} - h, z_{j+1} + h]) = \sum_{i=1}^{N} x_i^p y_i^q \mathbb{1}\{z_{j+1} - h \leq x_i \leq z_{j+1} + h\} \\
&= \sum_{i=1}^{N} x_i^p y_i^q \mathbb{1}\{z_j - h \leq x_i \leq z_j + h\} - \sum_{i=1}^{N} x_i^p y_i^q \mathbb{1}\{z_j - h \leq x_i < z_{j+1} - h\} + \sum_{i=1}^{N} x_i^p y_i^q \mathbb{1}\{z_j + h < x_i \leq z_{j+1} + h\} \\
&= \mathcal{S}^{p,q}([z_j - h, z_j + h]) - \mathcal{S}^{p,q}([z_j - h, z_{j+1} - h[) + \mathcal{S}^{p,q}(]z_j + h, z_{j+1} + h]) \quad (7)
\end{aligned}$$

Therefore one can simply update the sum $\mathcal{S}^{p,q}([z_j - h, z_{j+1} + h])$ for the evaluation point z_j to obtain the next sum $\mathcal{S}^{p,q}([z_{j+1} - h, z_{j+1} + h])$ for the next evaluation point z_{j+1} by subtracting the terms $x_i^p y_i^q$ for which x_i lie between $z_j - h$ and $z_{j+1} - h$, and adding

the terms $x_i^p y_i^q$ for which x_i lie between $z_j + h$ and $z_{j+1} + h$. This can be achieved in a fast $\mathcal{O}(M + N)$ operations by going through the input points x_i, stored in increasing order at a cost of $\mathcal{O}(N \log N)$ operations, and through the evaluation points z_j, stored in increasing order at a cost of $\mathcal{O}(M \log M)$ operations. In the case of the Epanechnikov kernel, the expansion of the quadratic term $\left(\frac{x_i - z_j}{h}\right)^2$ separates the sources x_i from the targets z_j (equation (5)), which makes the fast sum updating approach possible. Such a separation occurs with other classical kernels as well, including the rectangular kernel, the triangular kernel, the cosine kernel and the Silverman kernel (which, along with the compatible Laplace kernel, has infinite support), see Appendix A. Not every kernel admits such a separation between sources and targets, the most prominent example being the Gaussian kernel $K(u) = \frac{1}{\sqrt{2\pi}} \exp(-u^2/2)$, for which the cross term $\exp(x_i z_j / h)$ cannot be split between one source term (depending on i only) and one target term (depending on j only). Approximating the cross-term to obtain such a separation is the path followed by the Fast Gauss Transform approach. While we can use any kernel listed in Appendix A, we choose to use for the rest of the paper the Epanechnikov kernel $K(u) = \frac{3}{4}(1 - u^2)\mathbb{1}\{|u| \leq 1\}$ for two reasons: this popular kernel is optimal in the sense that it minimizes the asymptotic mean integrated squared error (Epanechnikov (1969)), and it supports fast sum updating with adaptive bandwidth $h = h_i$ or $h = h_j$ (see Appendix A and subsection 3.2).

2.2 Numerical stability

In Seifert et al. (1994), the direct fast sum updating approach (7) was discarded for numerical stability reasons. With floating-point arithmetic, the difference $(x + y) - x$ is in general equal to $y \pm \varepsilon$, where ε corresponds to the floating point rounding error. In addition, the greater the scale difference between two floating numbers x and y, the greater the rounding error when computing $x + y$. Consequently, adding and subtracting N numbers in sequence has a worst-case rounding error that grows proportional to N. In this paper, we argue that the advances in floating-point accuracy and stable floating-point summation in the past decades have made the direct fast sum updating approach viable and immune to numerical error. In addition to simple precautions such as normalization of input data and use of accurate floating-point formats such as quadruple-precision floating-point, a long list of

stable summation algorithms have been proposed in the past fifty years, see among others Møller (1965), Kahan (1965), Linnainmaa (1974), Priest (1991) Higham (1993), Demmel and Hida (2003), McNamee (2004) and Boldo et al. (2017). The usual idea is to keep track of the current amount of floating-point rounding error, and to propagate it when adding new terms in the sum. Recently, a number of exact summation algorithms have been proposed, see Rump et al. (2008), Pan et al. (2009), Zhu and Hayes (2010) and Neal (2015). These algorithms are exact in the sense that the final result is the closest floating-point number, within the precision of the chosen floating-point format, to the exact mathematical sum of the inputs. Importantly, the computational complexity of exact summation remains linear in the number N of data points to sum. For example, for the recent Neal (2015), exact summation is less than a factor two slower than naive summation. To sum up, with little modification, direct fast sum updating such as (7) can be made completely immune to numerical instability. As a simple illustration, we use the stable Møller-Kahan summation algorithm (Møller (1965)) for our numerical experiments (Section 4). For simplicity and clarity, the fast sum updating algorithms presented in this paper omit the stabilisation components. All of them can be implemented with perfect numerical stability using the stable summation algorithms mentioned in this subsection.

2.3 Multivariate case

We now turn to the multivariate case. Let d be the dimension of the inputs. We consider again a sample $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ of N input points x_i and output points y_i, where the input points are now multivariate: $x_i = (x_{1,i}, x_{2,i}, \ldots, x_{d,i})$, $i \in \{1, 2, \ldots, N\}$.

2.3.1 Multivariate kernel smoothers The kernel smoothers (1), (2) and (3) can be extended to the multivariate case. A general form for a multivariate kernel is $K_{d,H}(u) = |H|^{-1/2} K_d(H^{-1/2}u)$, where $u = (u_1, u_2, \ldots, u_d) \in \mathbb{R}^d$ and where H is a symmetric positive definite $d \times d$ bandwidth matrix (see Wand and Jones (1995) for example). The eigenvalue decomposition of H yields $H = R\Delta^2 R^\top$ where R is a rotation matrix and $\Delta = \mathrm{diag}(h)$ is a diagonal matrix with strictly positive diagonal elements $h = (h_1, h_2, \ldots, h_d) \in \mathbb{R}^d$. Therefore, without loss of generality, one can focus on the diagonal bandwidth case $K_{d,h}(u) =$

$\frac{1}{\Pi_{k=1}^d h_k} K_d(\frac{u_1}{h_1}, \frac{u_2}{h_2}, \ldots, \frac{u_d}{h_d})$ after a rotation of the input points x_i and the evaluation points z_j using R. Subsection 3.1 will discuss the choice of data rotation and subsection 2.3.3 will discuss the possible choices of multivariate kernels K_d compatible with fast sum updating. One can show (cf. Appendix B) that the computation of the multivariate version of the kernels smoothers (1), (2) and (3) boils down to the following sums:

$$\begin{aligned} \mathbf{S}_j &= \mathbf{S}_{k_1,k_2,j}^{p_1,p_2,q} := \frac{1}{N} \sum_{i=1}^{N} K_{d,h}(x_i - z_j) x_{k_1,i}^{p_1} x_{k_2,i}^{p_2} y_i^q \\ &= \frac{1}{N \Pi_{k=1}^d h_k} \sum_{i=1}^{N} K_d \left(\frac{x_{1,i} - z_{1,j}}{h_1}, \frac{x_{2,i} - z_{2,j}}{h_2}, \ldots, \frac{x_{d,i} - z_{d,j}}{h_d} \right) x_{k_1,i}^{p_1} x_{k_2,i}^{p_2} y_i^q \end{aligned} \tag{8}$$

for each evaluation point $z_j = (z_{1,j}, z_{2,j}, \ldots, z_{d,j}) \in \mathbb{R}^d$, $j \in \{1, 2, \ldots, M\}$, for powers $p_1, p_2, q = 0, 1$ and for dimension indices $k_1, k_2 = 1, 2, \ldots, d$. Before expanding the sum (8), we first introduce the two conditions required for fast multivariate sum updating (subsection 2.3.2) and then discuss the choice of multivariate kernel (subsection 2.3.3).

2.3.2 Conditions In order to extend the fast sum updating algorithm to the multivariate case, we require the following two conditions:

Condition 1. [Evaluation grid] We require the evaluation grid to be rectilinear, i.e., the M evaluation points $z_1, z_2, \ldots, z_M$ lie on a grid with possibly non-uniform mesh, of dimension $M_1 \times M_2 \times \ldots \times M_d = M$: $\big\{(z_{1,j_1}, z_{2,j_2}, \ldots, z_{d,j_d}) \in \mathbb{R}^d,\ j_k \in \{1, 2, \ldots, M_k\},\ k \in \{1, 2, \ldots, d\}\big\}$. Figure 2 page 16 provides two examples of rectilinear evaluation grids in the bivariate case.

Condition 2. [Kernel support] We allow the bandwidths to vary with the evaluation points (balloon estimators, see subsection 3.2) but require them to follow the shape of the evaluation grid. In other words, each evaluation point $z_j = (z_{1,j_1}, z_{2,j_2}, \ldots, z_{d,j_d})$ is associated with its own bandwidth $h_j = (h_{1,j_1}, h_{2,j_2}, \ldots, h_{d,j_d})$. For kernels with finite support, this means that the kernel support must be a hyperrectangle, i.e. the box $\prod_{k=1}^d [z_{k,j_k} - h_{k,j_k}, z_{k,j_k} + h_{k,j_k}] := \big\{(u_1, \ldots, u_d) \in \mathbb{R}^d \,|\, u_k \in [z_{k,j_k} - h_{k,j_k}, z_{k,j_k} + h_{k,j_k}], k = 1, \ldots, d\big\}$.

The reason for these two conditions will become clear after the description of the multivariate sweeping algorithm for multivariate sum updating. In the rest of this section, we assume these two conditions are satisfied.

2.3.3 Multivariate kernel To extend the definitions of the smoothing kernels (1), (2) and (3) to the multivariate case, one needs kernel functions defined in a multivariate setting. There exists different ways to extend a univariate kernel to the multivariate case, see Härdle and Müller (2000) for example. As an illustration, Figure 1 displays three different ways to extend the Epanechnikov kernel $K_1(u) = \frac{3}{4}(1-u^2)$ to the multivariate (bivariate) case.

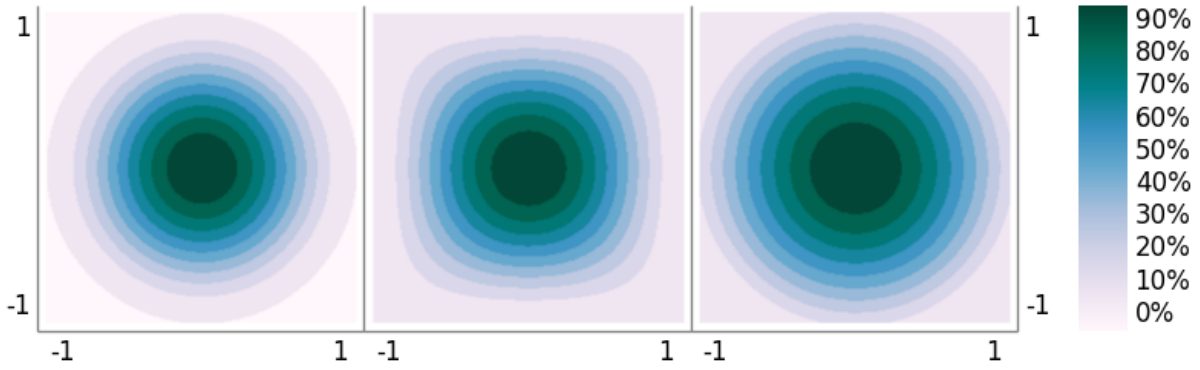

Figure 1: Bivariate parabolic kernels

The left-side kernel in Figure 1 corresponds to the *spherical* or *radially symmetric* kernel:

$$K_d^S(u_1,\ldots,u_d) = \frac{\Gamma\left(2+\frac{d}{2}\right)}{\pi^{\frac{d}{2}}}\left(1-\|u\|^2\right)\mathbb{1}\{\|u\|\leq 1\} \tag{9}$$

for which the norm of the vector u is used as an input to the univariate kernel (with a proper normalization constant, see Fukunaga and Hostetler (1975)). This multivariate kernel is the most efficient in terms of asymptotic mean integrated squared error (see Wand and Jones (1995) for example). Unfortunately, this kernel is not compatible with fast sum updating, as its support is a hypersphere, while Condition 2 requires a hyperrectangle support. The middle kernel in Figure 1 corresponds to the *multiplicative* or *product* kernel:

$$K_d^P(u_1,\ldots,u_d) = \prod_{k=1}^{d} K_1(u_k) = \left(\frac{3}{4}\right)^d \prod_{k=1}^{d}\left\{\left(1-u_k^2\right)\mathbb{1}\{|u_k|\leq 1\}\right\}, \tag{10}$$

obtained by multiplying univariate kernels. Its support is a hyperrectangle. Finally, the right-side kernel in Figure 1 corresponds to the *additive* or *arithmetic average* kernel:

$$K_d^A(u_1,\ldots,u_d) = \frac{1}{d2^{d-1}}\sum_{k=1}^{d} K_1(u_k)\prod_{\substack{k_0=1\\k_0\neq k}}^{d}\mathbb{1}\{|u_{k_0}|<1\} = \frac{3}{d2^{d+1}}\sum_{k=1}^{d}\left(1-u_k^2\right)\prod_{k_0=1}^{d}\mathbb{1}\{|u_{k_0}|<1\} \tag{11}$$

which is obtained by averaging univariate kernels, and is another general way of producing multivariate kernels. The support of this kernel is also a hyperrectangle. As Condition 2 rules out the spherical kernel (9), we have to make a choice between the product kernel (10) and the average kernel (11). When it comes to choosing a kernel, the following quote from Silverman (1982) summarizes the general consensus in the literature: *"Both theory and practice suggest that the choice of kernel is not crucial to the statistical performance of the method and therefore it is quite reasonable to choose a kernel for computational efficiency"*. In our context, this observation means that the average kernel (11) is to be preferred over the product kernel (10) for its greater computational efficiency. Indeed, while additive kernels are not as efficient[1] as product kernels (see Table 1) they contains much fewer sums to track down for the fast sum updating algorithm (after expanding the squared terms $(x_{k,i} - z_{k,j})^2/h_k^2$, the sum (8) is composed of 3^d different sums over $i = 1, \ldots, N$ for the product kernel (10), compared to only $2d + 1$ sums for the additive kernel (11)). In the end, to achieve the same accuracy, the additive kernel (11) is vastly faster than the product kernel (10) when using the fast sum updating approach (around 80% faster for bivariate problems, more than 18 times faster for five-dimensional problems, see Table 1). For this reason, we henceforth use the additive multivariate kernel (11) in the rest of the paper.

dimension	2D	3D	4D	5D
K^P efficiency	98.2%	95.3%	91.6%	87.4%
K^A efficiency	96.5%	88.9%	80.4%	71.8%
K^P number of sums	9	27	81	243
K^A number of sums	5	7	9	11
speedup factor of K^A over K^P	1.8	3.6	7.9	18.2

Table 1: product kernel K^P vs. average kernel K^A

[1]The efficiency $\text{eff}(K)$ of a kernel K is defined as the ratio $R(K^S)\mu_2^{d/2}(K^S)/(R(K)\mu_2^{d/2}(K))$ where $R(K) := \int \cdots \int K^2(u_1, \ldots, u_d)du_1 \ldots du_d$, $\mu_2(K) := \int \cdots \int u_1^2 K(u_1, \ldots, u_d)du_1 \ldots du_d$ and K^S is the spherical kernel (9), see Wand and Jones (1995). The speedup of K^A over K^P to achieve the same accuracy (Table 1), is defined as $3^d\text{eff}(K^P)/((2d+1)\text{eff}(K^A)) = (18/5)^d(3d/(5d-2))^{d/2}5d/((2d+1)(5d+1))$.

2.3.4 Kernel expansion Using the multivariate kernel (11), one can expand the sum (8) as follows:

$$\begin{aligned}\mathbf{S}_j &= \frac{3}{d2^{d+1}N\prod_{k=1}^d h_k}\sum_{i=1}^N\sum_{k=1}^d\left(1-\frac{(x_{k,i}-z_{k,j})^2}{h_k^2}\right)x_{k_1,i}^{p_1}x_{k_2,i}^{p_2}y_i^q\prod_{k_0=1}^d\mathbb{1}\{|x_{k_0,i}-z_{k_0,j}|\leq 1\}\\ &= \frac{3}{d2^{d+1}N\prod_{k=1}^d h_k}\sum_{k=1}^d\left\{\left(1-\frac{z_{k,j}^2}{h_k^2}\right)\mathcal{S}_{[k,k_1,k_2]}^{[0,p_1,p_2],q}([z_j-h_j,z_j+h_j])+\right.\\ &\left.2\frac{z_{k,j}}{h_k^2}\mathcal{S}_{[k,k_1,k_2]}^{[1,p_1,p_2],q}([z_j-h_j,z_j+h_j])-\frac{1}{h_k^2}\mathcal{S}_{[k,k_1,k_2]}^{[2,p_1,p_2],q}([z_j-h_j,z_j+h_j])\right\}\end{aligned}\tag{12}$$

where for any hyperrectangle $[\mathbf{L},\mathbf{R}]:=[L_1,R_1]\times[L_2,R_2]\times\ldots\times[L_d,R_d]\subseteq\mathbb{R}^d$:

$$\mathcal{S}^{\text{idx}}([\mathbf{L},\mathbf{R}]):=\mathcal{S}_{\mathbf{k}}^{\mathbf{p},q}([\mathbf{L},\mathbf{R}]):=\sum_{i=1}^N\left(\prod_{l=1}^3(x_{k_l,i})^{p_l}\right)y_i^q\prod_{k_0=1}^d\mathbb{1}\{L_{k_0}\leq x_{k_0,i}\leq R_{k_0}\}\tag{13}$$

for powers $\mathbf{p}:=(p_1,p_2,p_3)\in\mathbb{N}^3$, $q\in\mathbb{N}$, indices $\mathbf{k}:=(k_1,k_2,k_3)\in\{1,2,\ldots,d\}^3$, and where $[z_j-h_j,z_j+h_j]:=[z_{1,j}-h_{1,j},z_{1,j}+h_{1,j}]\times\ldots\times[z_{d,j}-h_{d,j},z_{d,j}+h_{d,j}]$. To simplify notations, we make use of the multi-index $\text{idx}:=(\mathbf{p},q,\mathbf{k})$.

To sum up what has been obtained so far, computing multivariate kernel smoothers (kernel density estimation, kernel regression, locally linear regression) boils down to computing sums of the type (13) on hyperrectangles of the type $[z_j-h_j,z_j+h_j]$ for every evaluation point $j\in\{1,2,\ldots,M\}$. In the univariate case, these sums could be computed efficiently by sorting the input points x_i, $i\in\{1,2,\ldots,N\}$ and updating the sums from one evaluation point to the next (equation (7)). Our goal is now to set up a similar efficient fast sum updating algorithm for the multivariate sums (13). To do so, we first partition the input data into a multivariate rectilinear grid (subsection 2.3.5), by taking advantage of the fact that the evaluation grid is rectilinear (Condition 1) and that the support of the kernels has a hyperrectangle shape (Condition 2). Then, we set up a fast sweeping algorithm using the sums on each hyperrectangle of the partition as the unit blocks to be added and removed (subsection 2.3.6), unlike the univariate case where the input points themselves were being added and removed iteratively. Finally, the computational speed of this new algorithm is discussed in subsection 2.4.

2.3.5 Data partition The first stage of the multivariate fast sum updating algorithm is to partition the sample of input points into boxes. To do so, define the sorted lists

$\tilde{\mathcal{G}}_k = \{\tilde{g}_{k,1}, \tilde{g}_{k,2}, \ldots, \tilde{g}_{k,2M_k}\} := \text{sort}\Big(\{z_{k,j_k} - h_{k,j_k}\}_{j_k \in \{1,2,\ldots,M_k\}} \bigcup \{z_{k,j_k} + h_{k,j_k}\}_{j_k \in \{1,2,\ldots,M_k\}}\Big)$ in each dimension $k \in \{1, 2, \ldots, d\}$, and define the partition intervals $\tilde{I}_{k,l} := [\tilde{g}_{k,l}, \tilde{g}_{k,l+1}]$ for $l \in \{1, 2, \ldots, 2M_k - 1\}$. By definition of $\tilde{\mathcal{G}}_k$, all the bandwidths edges $z_{k,j_k} - h_{k,j_k}$ and $z_{k,j_k} + h_{k,j_k}$, $j_k \in \{1, 2, \ldots, M_k\}$, belong to $\tilde{\mathcal{G}}_k$. Therefore, there exists some indices $\tilde{L}_{k,j_k}$ and $\tilde{R}_{k,j_k}$ such that $[z_{k,j_k} - h_{k,j_k}, z_{k,j_k} + h_{k,j_k}] = [\tilde{g}_{k,\tilde{L}_{k,j_k}}, \tilde{g}_{k,\tilde{R}_{k,j_k}+1}] = \bigcup_{l_k \in \{\tilde{L}_{k,j_k}, \ldots, \tilde{R}_{k,j_k}\}} \tilde{I}_{k,l_k}$. From there, for any evaluation point $z_j = (z_{1,j_1}, z_{2,j_2}, \ldots, z_{d,j_d}) \in \mathbb{R}^d$, the box $[z_j - h_j, z_j + h_j] \subset \mathbb{R}^d$ can be decomposed into a union of smaller boxes:

$$\begin{aligned}
[z_j - h_j, z_j + h_j] &= [z_{1,j_1} - h_{1,j_1}, z_{1,j_1} + h_{1,j_1}] \times \ldots \times [z_{d,j_d} - h_{d,j_d}, z_{d,j_d} + h_{d,j_d}] \\
&= \Big[\tilde{g}_{1,\tilde{L}_{1,j_1}}, \tilde{g}_{1,\tilde{R}_{1,j_1}+1}\Big] \times \ldots \times \Big[\tilde{g}_{d,\tilde{L}_{d,j_d}}, \tilde{g}_{d,\tilde{R}_{d,j_d}+1}\Big] \\
&= \bigcup_{(l_1,\ldots,l_d) \in \{\tilde{L}_{1,j_1},\ldots,\tilde{R}_{1,j_1}\} \times \ldots \times \{\tilde{L}_{d,j_d},\ldots,\tilde{R}_{d,j_d}\}} \tilde{I}_{1,l_1} \times \ldots \times \tilde{I}_{d,l_d} \qquad (14)
\end{aligned}$$

In other words, the set of boxes $\tilde{I}_{1,l_1} \times \tilde{I}_{2,l_2} \times \ldots \times \tilde{I}_{d,l_d}$ s.t. $l_k \in \{\tilde{L}_{k,j_k}, \tilde{L}_{k,j_k} + 1, \ldots, \tilde{R}_{k,j_k}\}$ in each dimension $k \in \{1, 2, \ldots, d\}$) forms a partition of the box $[z_j - h_j, z_j + h_j]$. Consequently, the sum (13) evaluated on the box $[z_j - h_j, z_j + h_j]$ can be decomposed as follows:

$$\mathcal{S}^{\text{idx}}([z_j - h_j, z_j + h_j]) = \sum_{(l_1,\ldots,l_d) \in \{\tilde{L}_{1,j_1},\ldots,\tilde{R}_{1,j_1}\} \times \ldots \times \{\tilde{L}_{d,j_d},\ldots,\tilde{R}_{d,j_d}\}} \mathcal{S}^{\text{idx}}\Big(\tilde{I}_{1,l_1} \times \ldots \times \tilde{I}_{d,l_d}\Big) \qquad (15)$$

where we assume without loss of generality that the bandwidth grid $h_j = (h_{1,j_1}, h_{2,j_2}, \ldots, h_{d,j_d})$, $j_k \in \{1, 2, \ldots, M_k\}$, $k \in \{1, 2, \ldots d\}$ is such that the list $\tilde{\mathcal{G}}_k$ does not contain any input $x_{k,i}$, $i \in \{1, 2, \ldots, N\}$ (as such boundary points would be counted twice in the right-hand side of (15)). This simple condition is easy to satisfy, as shown by the adaptive bandwidth example provided in subsection 3.2. The sum decomposition (15) is the cornerstone of the fast multivariate sum updating algorithm, but before going further, one can simplify the partitions $\tilde{\mathcal{G}}_k$ while maintaining a sum decomposition of the type (15). Indeed, in general some intervals $\tilde{I}_{k,l}$ might be empty (i.e. they might not contain any input point $x_{k,i}$). To avoid keeping track of sums $\mathcal{S}^{\text{idx}}$ on boxes known to be empty, one can trim the partitions $\tilde{\mathcal{G}}_k$ by replacing each succession of empty intervals by one new partition threshold. For example, if $\tilde{I}_{k,l} = [\tilde{g}_{k,l}, \tilde{g}_{k,l+1}]$ is empty, one can remove the two points $\tilde{g}_{k,l}$ and $\tilde{g}_{k,l+1}$ and replace them by, for example, $(\tilde{g}_{k,l} + \tilde{g}_{k,l+1})/2$. Denote by $\mathcal{G}_k = \{g_{k,1}, g_{k,2}, \ldots, g_{k,m_k}\}$ the sorted simplified list, where $2 \leq m_k \leq 2M_k$, $k \in \{1, 2, \ldots, d\}$, and $m := \prod_{k=1}^{d} m_k \leq 2^d M$.

Define the new partition intervals $I_{k,l} := [g_{k,l}, g_{k,l+1}]$, $l \in \{1, 2, \ldots, m_k - 1\}$. Because the trimming from $\tilde{\mathcal{G}}_k$ to $\mathcal{G}_k$ only affects the empty intervals, the following still holds:

Lemma 2.1. *For any evaluation point $z_j = (z_{1,j_1}, z_{2,j_2}, \ldots, z_{d,j_d}) \in \mathbb{R}^d$, $j_k \in \{1, 2, \ldots, M_k\}$, $k \in \{1, 2, \ldots d\}$, there exists indices $(L_{1,j_1}, L_{2,j_2}, \ldots, L_{d,j_d})$ and $(R_{1,j_1}, R_{2,j_2}, \ldots, R_{d,j_d})$ where L_{k,j_k} and $R_{k,j_k} \in \{1, 2, \ldots, m_k - 1\}$ with $L_{k,j_k} \leq R_{k,j_k}$ such that*

$$\mathcal{S}^{\text{idx}}([z_j - h_j, z_j + h_j]) = \sum_{(l_1,\ldots,l_d) \in \{L_{1,j_1},\ldots,R_{1,j_1}\} \times \ldots \times \{L_{d,j_d},\ldots,R_{d,j_d}\}} \mathcal{S}^{\text{idx}}(I_{1,l_1} \times \ldots \times I_{d,l_d}) \tag{16}$$

For later use, we introduce the compact notation $\mathcal{S}^{\text{idx}}_{l_1,l_2,\ldots,l_d} := \mathcal{S}^{\text{idx}}(I_{1,l_1} \times \ldots \times I_{d,l_d})$. Recalling equation (13), the sum $\mathcal{S}^{\text{idx}}_{l_1,l_2,\ldots,l_d}$ corresponds to the sum of the polynomials $(\prod_{l=1}^{3}(x_{k_l,i})^{p_l})y_i^q$ over all the data points within the box $I_{1,l_1} \times \ldots \times I_{d,l_d}$.

2.3.6 Fast multivariate sweeping algorithm So far, we have shown that computing multivariate kernel smoothers is based on the computation of the kernel sums (8), which can be decomposed into sums of the type (13), which themselves can be decomposed into the smaller sums (16) by decomposing every kernel support of every evaluation point onto the box partition described in the previous subsection 2.3.5. The final task is to define an efficient algorithm to traverse all the hyperrectangle unions $\bigcup_{(l_1,\ldots,l_d) \in \{L_{1,j_1},\ldots,R_{1,j_1}\} \times \ldots \times \{L_{d,j_d},\ldots,R_{d,j_d}\}} I_{1,l_1} \times \ldots \times I_{d,l_d}$, so as to compute the right-hand side sums in equations (16) (Lemma 2.1) in an efficient fast sum updating way similar to the univariate (7). We precompute all the sums $\mathcal{S}^{\text{idx}}_{l_1,l_2,\ldots,l_d}$ with $\text{idx} = (\mathbf{p}, q, \mathbf{k}) \in \{0, 1, 2\} \times \{0, 1\}^3 \times \{1, 2, \ldots, d\}^3$, and use them as input material for fast multivariate sum updating.

We start with the bivariate case. We first provide an algorithm to compute the sums $\mathcal{T}^{\text{idx}}_{1,l_2} := \sum_{l_1=L_{1,j_1}}^{R_{1,j_1}} \mathcal{S}^{\text{idx}}_{l_1,l_2}$, for every $l_2 \in \{1, 2, \ldots, m_2 - 1\}$ and every index interval $[L_{1,j_1}, R_{1,j_1}]$, $j_1 \in \{1, 2, \ldots, M_1\}$. Starting with $j_1 = 1$, we first compute $\mathcal{T}^{\text{idx}}_{1,l_2} = \sum_{l_1=L_{1,1}}^{R_{1,1}} \mathcal{S}^{\text{idx}}_{l_1,l_2}$ for every $l_2 \in \{1, 2, \ldots, m_2 - 1\}$. Then we iteratively increment j_1 from $j_1 = 1$ to $j_1 = M_1$. After each incrementation of j_1, we update $\mathcal{T}^{\text{idx}}_{1,l_2}$ by fast sum updating

$$\sum_{l_1=L_{1,j_1}}^{R_{1,j_1}} \mathcal{S}^{\text{idx}}_{l_1,l_2} = \sum_{l_1=L_{1,j_1-1}}^{R_{1,j_1-1}} \mathcal{S}^{\text{idx}}_{l_1,l_2} + \sum_{l_1=R_{1,j_1-1}+1}^{R_{1,j_1}} \mathcal{S}^{\text{idx}}_{l_1,l_2} - \sum_{l_1=L_{1,j_1-1}}^{L_{1,j_1-1}} \mathcal{S}^{\text{idx}}_{l_1,l_2} \tag{17}$$

Algorithm 1: Fast multivariate kernel smoothing

Input: precomputed sums $\mathcal{S}^{\text{idx}}_{l_1,l_2,\ldots,l_d}$

$\text{iL}_1 = 1$, $\text{iR}_1 = 1$, $\mathcal{T}^{\text{idx}}_{1,l_2,l_3,\ldots,l_d} = 0$

for $j_1 = 1, \ldots, M_1$ **do**

  while *(* $\text{iR}_1 < m_1$ *) and (* $\text{iR}_1 \leq R_{1,j_1}$ *)* **do**

    $\mathcal{T}^{\text{idx}}_{1,l_2,l_3,\ldots,l_d} = \mathcal{T}^{\text{idx}}_{1,l_2,l_3,\ldots,l_d} + \mathcal{S}^{\text{idx}}_{\text{iR}_1,l_2,l_3,\ldots,l_d}, \forall l_k \in \{1,2,\ldots,m_k-1\}, k \in \{2,3,\ldots,d\}$

    $\text{iR}_1 = \text{iR}_1 + 1$

  end

  while *(* $\text{iL}_1 < m_1$ *) and (* $\text{iL}_1 < L_{1,j_1}$ *)* **do**

    $\mathcal{T}^{\text{idx}}_{1,l_2,l_3,\ldots,l_d} = \mathcal{T}^{\text{idx}}_{1,l_2,l_3,\ldots,l_d} - \mathcal{S}^{\text{idx}}_{\text{iL}_1,l_2,l_3,\ldots,l_d}, \forall l_k \in \{1,2,\ldots,m_k-1\}, k \in \{2,3,\ldots,d\}$

    $\text{iL}_1 = \text{iL}_1 + 1$

  end // Here $\mathcal{T}^{\text{idx}}_{1,l_2,l_3,\ldots,l_d} = \sum_{l_1=L_{1,j_1}}^{R_{1,j_1}} \mathcal{S}^{\text{idx}}_{l_1,l_2,\ldots,l_d}, \forall l_k \in \{1,2,\ldots,m_k-1\}, k \in \{2,3,\ldots,d\}$

  $\text{iL}_2 = 1$, $\text{iR}_2 = 1$, $\mathcal{T}^{\text{idx}}_{2,l_3,\ldots,l_d} = 0$

  for $j_2 = 1, \ldots, M_2$ **do**

    while *(* $\text{iR}_2 < m_2$ *) and (* $\text{iR}_2 \leq R_{2,j_2}$ *)* **do**

      $\mathcal{T}^{\text{idx}}_{2,l_3,\ldots,l_d} = \mathcal{T}^{\text{idx}}_{2,l_3,\ldots,l_d} + \mathcal{T}^{\text{idx}}_{1,\text{iR}_2,l_3,\ldots,l_d}, \ \forall l_k \in \{1,2,\ldots,m_k-1\}, \ k \in \{3,\ldots,d\}$

      $\text{iR}_2 = \text{iR}_2 + 1$

    end

    while *(* $\text{iL}_2 < m_2$ *) and (* $\text{iL}_2 < L_{2,j_2}$ *)* **do**

      $\mathcal{T}^{\text{idx}}_{2,l_3,\ldots,l_d} = \mathcal{T}^{\text{idx}}_{2,l_3,\ldots,l_d} - \mathcal{T}^{\text{idx}}_{1,\text{iL}_2,l_3,\ldots,l_d}, \ \forall l_k \in \{1,2,\ldots,m_k-1\}, \ k \in \{3,\ldots,d\}$

      $\text{iL}_2 = \text{iL}_2 + 1$

    end // $\mathcal{T}^{\text{idx}}_{2,l_3,\ldots,l_d} = \sum_{l_1=L_{1,j_1}}^{R_{1,j_1}} \sum_{l_2=L_{2,j_2}}^{R_{2,j_2}} \mathcal{S}^{\text{idx}}_{l_1,l_2,\ldots,l_d}, \forall l_k \in \{1,\ldots,m_k-1\}, k \in \{3,\ldots d\}$

    $\vdots$

    $\text{iL}_\text{d} = 1$, $\text{iR}_\text{d} = 1$, $\mathcal{T}_d = 0$

    for $j_d = 1, \ldots, M_d$ **do**

      while *(* $\text{iR}_\text{d} < m_d$ *) and (* $\text{iR}_\text{d} \leq R_{d,j_d}$ *)* **do**

        $\mathcal{T}^{\text{idx}}_d = \mathcal{T}^{\text{idx}}_d + \mathcal{T}^{\text{idx}}_{d-1,\text{iR}_\text{d}}$

        $\text{iR}_\text{d} = \text{iR}_\text{d} + 1$

      end

      while *(* $\text{iL}_\text{d} < m_d$ *) and (* $\text{iL}_\text{d} < L_{d,j_d}$ *)* **do**

        $\mathcal{T}^{\text{idx}}_d = \mathcal{T}^{\text{idx}}_d - \mathcal{T}^{\text{idx}}_{d-1,\text{iL}_\text{d}}$

        $\text{iL}_\text{d} = \text{iL}_\text{d} + 1$

      end // Here $\mathcal{T}^{\text{idx}}_d = \mathcal{S}^{\mathbf{p},q}_{\mathbf{k}}([z_j - h_j, z_j + h_j])$ from equation (16)

      Compute $\mathbf{S}_j$ using $\mathcal{T}^{\text{idx}}_d$ and equation (12)

    end

  end

end

Output: Multivariate kernel smoothers

The second stage is to perform a fast sum updating in the second dimension, with the sums $\mathcal{T}^{\text{idx}}_{1,l_2} = \sum_{l_1=L_{1,j_1}}^{R_{1,j_1}} \mathcal{S}^{\text{idx}}_{l_1,l_2}$ as input material. Our goal is to compute the sums $\mathcal{T}^{\text{idx}}_{2} := \sum_{l_2=L_{2,j_2}}^{R_{2,j_2}} \mathcal{T}^{\text{idx}}_{1,l_2}$ for every indices interval $[L_{2,j_2}, R_{2,j_2}]$, $j_2 \in \{1, 2, \ldots, M_2\}$. In a similar manner, we start from $j_2 = 1$ with the initial sum $\mathcal{T}^{\text{idx}}_{2} = \sum_{l_2=L_{2,1}}^{R_{2,1}} \mathcal{T}^{\text{idx}}_{1,l_2}$. We then increment j_2 from $j_2 = 1$ to $j_2 = M_2$ iteratively. After each incrementation of j_2, we update $\mathcal{T}^{\text{idx}}_{2}$ by fast sum updating:

$$\sum_{l_2=L_{2,j_2}}^{R_{2,j_2}} \mathcal{T}^{\text{idx}}_{1,l_2} = \sum_{l_2=L_{2,j_2-1}}^{R_{2,j_2-1}} \mathcal{T}^{\text{idx}}_{1,l_2} + \sum_{l_2=R_{2,j_2-1}+1}^{R_{2,j_2}} \mathcal{T}^{\text{idx}}_{1,l_2} - \sum_{l_2=L_{2,j_2-1}}^{L_{2,j_2}-1} \mathcal{T}^{\text{idx}}_{1,l_2} \tag{18}$$

Using Lemma 2.1, the resulting sum $\sum_{l_2=L_{2,j_2}}^{R_{2,j_2}} \mathcal{T}^{\text{idx}}_{1,l_2} = \sum_{l_1=L_{1,j_1}}^{R_{1,j_1}} \sum_{l_2=L_{2,j_2}}^{R_{2,j_2}} \mathcal{S}^{\text{idx}}_{l_1,l_2}$ is equal to $\mathcal{S}^{\text{idx}}([z_j - h_j, z_j + h_j])$, which can be used to compute the kernel sums $\mathbf{S}_j$ using equation (12), from which the bivariate kernel smoothers can be computed.

This ends the description of the fast sum updating algorithm in the bivariate case. The reason for enforcing Condition 1 and Condition 2 is now clear: they pave the way for the rectilinear partition described in subsection 2.3.5, from which the iterative fast sum updating, one dimension at a time, can cover all the multivariate bandwidths of all the evaluation points on the evaluation grid. Finally, the general multivariate case is a straightforward extension of the bivariate case, and is summarized in Algorithm 1.

2.4 Complexity

One can verify that the number of operations in the multivariate fast sum updating algorithm 1 is proportional to the number of evaluation points $M = M_1 \times M_2 \times \ldots \times M_d$. Indeed, recall from subsection 2.3.5 that in each dimension $k \in \{1, 2, \ldots, d\}$, $m_k - 1$ is the number of intervals in the k-th dimension of the data partition, with $2 \leq m_k \leq 2M_k$. The first two *while* loops over iR_1 and iL_1 in Algorithm 1 generate $2(m_1 - 1)$ updates of the sums $\mathcal{T}^{\text{idx}}_{1,l_2,l_3,\ldots,l_d}$ of size $(m_2 - 1) \times \ldots \times (m_d - 1)$, for a total of $\mathcal{O}(M)$ operations. Then, the two subsequent *while* loops over iR_2 and iL_2 generate $M_1 \times 2(m_2 - 1)$ updates of the sums $\mathcal{T}^{\text{idx}}_{2,l_3,\ldots,l_d}$ of size $(m_3 - 1) \times \ldots \times (m_d - 1)$, for a total of $\mathcal{O}(M)$ operations. The final *while* loops over iR_d and iL_d generate $M_1 \times \ldots \times M_{d-1} \times 2(m_d - 1) = \mathcal{O}(M)$ updates of the sum $\mathcal{T}^{\text{idx}}_{d}$ of size 1. The computational complexity of Algorithm 1 is therefore $\mathcal{O}(M)$. In addition to this cost, Algorithm 1 requires the construction of the partition $\mathcal{G}_k$ and of the

threshold indices $L_{k,j_k} \in \{1, 2, \ldots, m_k - 1\}$ and $R_{k,j_k} \in \{1, 2, \ldots, m_k - 1\}$ (recall Lemma 2.1), which costs $\mathcal{O}(M)$ operations or $\mathcal{O}(M \log M)$ if the evaluation points are not sorted. The precomputation of the sums $\mathcal{S}^{\text{idx}}_{l_1,l_2,\ldots,l_d}$ costs $\mathcal{O}(N)$ operations once the input sample $(x_{1,i}, x_{2,i}, \ldots, x_{d,i})$, $i \in \{1, 2, \ldots, N\}$ has been sorted in each dimension independently, at a cost of $\mathcal{O}(N \log N)$ operations. The total computational complexity of the multivariate fast sum updating algorithm described in this section is therefore $\mathcal{O}(M \log M + N \log N)$, which is a considerable improvement over the $\mathcal{O}(M \times N)$ complexity of the naive approach. The memory consumption of Algorithm 1 stems from the simultaneous storage of the sums $\mathcal{S}^{\text{idx}}_{l_1,l_2,\ldots,l_d}$, $\mathcal{T}^{\text{idx}}_{1,l_2,\ldots,l_d}, \ldots$, $\mathcal{T}^{\text{idx}}_{d}$ for every $l_k \in \{1, 2, \ldots, m_k - 1\}$, $k \in \{2, 3, \ldots, d\}$ and $\text{idx} = (\mathbf{p}, q, \mathbf{k}) \in \{0, 1, 2\} \times \{0, 1\}^3 \times \{1, 2, \ldots, d\}^3$, resulting in a memory complexity of $\mathcal{O}(M)$.

3 Evaluation grid and adaptive bandwidth

This section suggests some suitable choices of evaluation grid and adaptive bandwidth compatible with the two conditions 1 and 2, so as to ensure a wide applicability of the fast kernel smoothers described in this paper.

3.1 Shape of evaluation grid

As explained in 2.3.1, kernel smoothing estimates can generally be improved by prerotating the input dataset into a better basis. Rotating the dataset before performing kernel density estimations has been advocated in Wand (1994) and Scott and Sain (2005) for example. Condition 1 adds another motivation for rotating the dataset. Indeed, when the natural evaluation sample is not a grid, for example when the evaluation points z_j, $j \in \{1, 2, \ldots, M\}$ are equal to the input points x_i, $i \in \{1, 2, \ldots, N\}$, one needs to build a suitable intermediate evaluation grid to properly cover the input sample. The left-side of Figure 2 illustrates on a bivariate example with $N = 100$ input points the potential problem of rectilinear evaluation grids when the dimensions of the input dataset are dependent: some evaluation points can be left away from the dataset. As some evaluation points are located in empty areas, the effective number of evaluation points is decreased. A rotation of the input dataset can mitigate or eliminate this problem, as shown on the right-side of Figure 2.

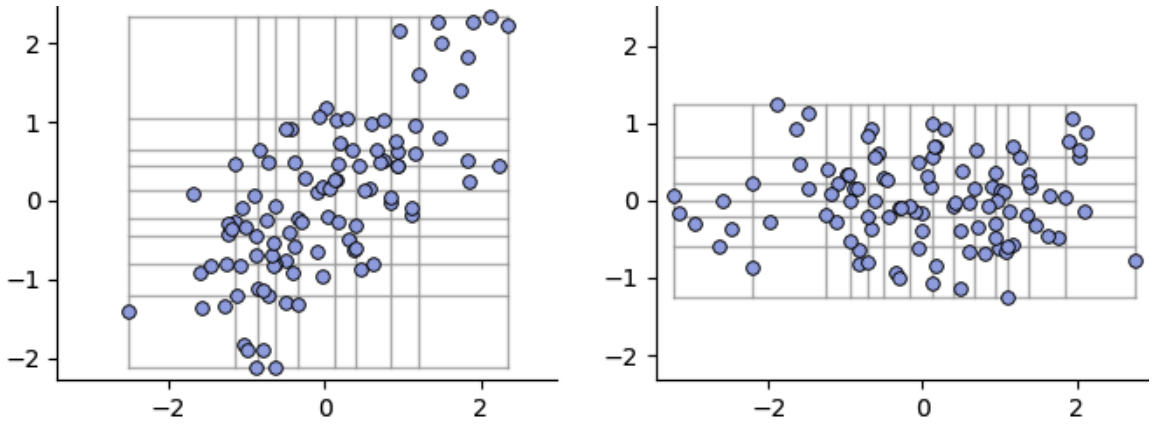

Figure 2: Evaluation grid: rotation

To construct the rotation, several techniques can be used. One possible choice is to rotate the dataset onto its principal components (as shown on Figure 2). To define the evaluation grid $\{(z_{1,j_1}, z_{2,j_2}, \ldots, z_{d,j_d}),\ j_k \in \{1, 2, \ldots, M_k\},\ k \in \{1, 2, \ldots, d\}\}$, one can first set each M_k to $M^{\frac{1}{d}}$ and define $z_{k,j_k} = x_{k,\text{round}\left(1+(N-1)\times\frac{j_k-1}{M_k-1}\right)}$, where round$(u)$ is the closest integer to $u \in \mathbb{R}$ and the input set x_i, $i \in \{1, 2, \ldots, N\}$ is sorted in increasing order. Such a grid is illustrated on the left-side of Figure 2 (original dataset without rotation). One alternative choice for M_k is to set it proportional to the k-th singular value associated with the k-th principal component. In addition to improving the coverage of the input dataset by the evaluation grid (more evaluation points along the more variable input dimensions), this choice of M_k can reduce the dimension of the problem whenever some M_k are set to one due to a small singular value. This choice of M_k is illustrated on the right-side of Figure 2 (after the rotation of the input dataset), and is the one we use in the numerical section 4. Once the kernel density estimates have been obtained on the intermediate evaluation grid using the fast algorithm described in Section 2, one can interpolate the estimates from the grid to the evaluation points of interest by simple multilinear interpolation. Alternatively, one can interpolate by Inverse Distance Weighting (Shepherd (1968)). When the weights are chosen as kernels from Appendix A, this interpolation bears some similarity with kernel density estimation, and can benefit from the fast sum updating algorithm described in Section 2.

3.2 Fast adaptive bandwidth

The kernel smoothers (1), (2) and (3) can be defined with a fixed bandwidth h, or with an adaptive bandwidth which varies with either the input points or the evaluation points. When the input design is random as on Figure 2, some areas might be sparse while others will be dense. In such cases, the benefit of adaptive bandwidth is that one can maintain a uniform quality of density estimates by using a larger bandwidth in sparse areas and a smaller bandwidth in dense areas. There exists two main ways to define adaptive bandwidths: balloon bandwidths $h = h_j$ which vary with the evaluation point $j \in \{1, 2, ..., M\}$, and sample point bandwidths $h = h_i$ which vary with the input point $i \in \{1, 2, ..., N\}$, see Terrell and Scott (1992) or Scott and Sain (2005). While many univariate kernels from Appendix A are compatible with both balloon bandwidths and sample point bandwidths, the data partition from subsection 2.3.5, which is required in the multivariate case, has been tailored for the balloon formulation (Condition 2) adopted in this paper. For the construction of the adaptive bandwidth, we adopt the K-nearest neighbor bandwidth suggested in Loftsgaarden and Quesenberry (1965), as it was shown in Terrell and Scott (1992) to perform well in multivariate settings. In addition, such a choice of bandwidth ensures that the bandwidth boundaries $\{z_{k,j_k} - h_{k,j_k}\}_{j_k=1,...,M_k}$ and $\{z_{k,j_k} + h_{k,j_k}\}_{j_k=1,...,M_k}$, $k = \{1, 2, \ldots, d\}$ remain in increasing order, which was implicitly assumed in Algorithm 1 for simplicity (one can easily adjust the loops in Algorithm 1 to decrement instead of increment the grid indices $\mathrm{iL_k}$ and $\mathrm{iR_k}$ whenever the bandwidth boundaries are not in increasing order).

We now describe how to build these bandwidths in a fast $\mathcal{O}(M + N)$ from sorted datasets in the univariate case ($\mathcal{O}(M \log M + N \log N)$ if the datasets need to be sorted beforehand), and then discuss the extension to the multivariate case. Let $x_1 \leq x_2 \leq \ldots \leq x_N$ be a sorted set of N sample points, and $z_1 \leq z_2 \leq \ldots \leq z_M$ be a sorted set of M evaluation points. We want to build M adaptive bandwidths h_j centered around the points z_j, $j = 1, \ldots, M$, such that each bandwidth $[z_j - h_j, z_j + h_i]$ contains exactly K sample points. Define $i_L \in [1, \ldots, N - K + 1]$ and $i_R = i_L + K - 1$. The subset $x_{i_L}, x_{i_L+1}, \ldots, x_{i_R}$ contains exactly K points. The idea of the algorithm is to enumerate all such possible index ranges $[i_L, i_R]$ from left ($i_L = 1$, $i_R = K$) to right ($i_L = N - K + 1$, $i_R = N$), and to match each evaluation point z_j, $j \in \{1, 2, \ldots, N\}$ with its corresponding K-nearest-

neighbors subsample $x_{i_L}, x_{i_L+1}, \ldots, x_{i_R}$. Matching each index j to its corresponding $[i_L, i_R]$ range is simple. When $i_L = 1$, all the points z_j such that $z_j \leq (x_{i_L} + x_{i_R+1})/2$ are such that the subsample $x_{i_L}, x_{i_L+1}, \ldots, x_{i_R}$ corresponds to their K nearest neighbors. Indeed, any point greater than $(x_{i_L} + x_{i_R+1})/2$ is closer to x_{i_R+1} than to x_{i_L}, and therefore its K nearest neighbors are not $x_{i_L}, x_{i_L+1}, \ldots, x_{i_R}$. Once all such z_j are matched to the current $[i_L, i_R]$ range, i_L and i_R are incremented until $(x_{i_L} + x_{i_R+1})/2$ is greater than the next evaluation point z_j to assign. The same procedure is then repeated until all the points are assigned to their K nearest neighbors. Finally, once each point z_j is assigned to its K nearest neighbors $x_{i_L}, x_{i_L+1}, \ldots, x_{i_R}$, one still needs to choose the bandwidth h_j such that $[z_j - h_j, z_j + h_j]$ contains these K nearest neighbors. Such a bandwidth h_j exists but is not unique. We choose to set h_j to the average between the smallest possible h_j (equal to $\max\{z_j - x_{i_L}, x_{i_R} - z_j\}$) and the largest possible h_j (equal to $\min\{z_j - x_{i_L-1}, x_{iR+1} - z_j\}$ when $i_L - 1 \geq 1$ and $i_R + 1 \leq N$). The computational complexity of the described algorithm is a fast $\mathcal{O}(M + N)$ if the set of input points x_i $i \in \{1, 2, \ldots, N\}$ and the set of evaluation points z_j, $j \in \{1, 2, \ldots, M\}$ are already sorted, and $\mathcal{O}(M \log(M) + N \log(N))$ otherwise.

In the multivariate case, given Condition 2, what can be done is to compute approximate multivariate K-nearest-neighbors bandwidths by performing the described algorithm dimension per dimension. Let $p = K/N = p_1 \times p_2 \times \ldots \times p_d$ be the proportion of input points to include within each multivariate bandwidth. In practice, we set p_k to be inversely proportional to the k-th singular value associated with the k-th axis (projected onto $[0, 1]$ if it ends up outside this probability range) and run the fast K-NN algorithm with $K_k = p_k \times N$ in each dimension $k \in \{1, 2, \ldots, d\}$ independently, which should ensure each multivariate bandwidth contains approximately K input points, provided the rotation onto the principal components has been performed beforehand (subsection 3.1).

4 Numerical tests

In this section, we test the fast kernel summation algorithm introduced in Section 2 and compare it to naive summation in terms of speed and accuracy. We consider a sample of N input points, choose the number of evaluation points M approximately equal to N, and build the evaluation grid and the bandwidths as described in Section 3. From sub-

section 2.4, we expect a runtime proportional to $N\log(N)$. We are going to verify this result numerically. Then, we are going to compare the estimates obtained by fast kernel summation to those obtain by naive summation. As discussed in subsection 2.2, we expect small differences coming from the rounding of floats, which can be reduced or removed altogether by the use of stable summation algorithm. As a simple illustration, we measure the accuracy improvement provided by the simple Møller-Kahan algorithm (Møller (1965), Linnainmaa (1974), Ozawa (1983)). Beyond this simple stability improvement, one can instead use exact summation algorithms to remove any float rounding errors while maintaining the $\mathcal{O}(N\log(N))$ complexity (cf. subsection 2.2). The input sample can be chosen arbitrarily as it does not affect the speed or accuracy of the two algorithms. We therefore simply choose to simulate N points from a d-dimensional Gaussian random variable $X \sim \mathbb{N}(0, 0.6\mathbf{1_d})$. In addition to the input sample $x_1, x_2, \ldots, x_N$, we need an output sample $y_1, y_2, \ldots, y_N$, in order to test the locally linear regression. Similarly to the input sample, the output sample can be chosen arbitrarily. We choose to define the output as $Y = f(X) + W$, where the univariate Gaussian noise $W \sim \mathbb{N}(0, 0.7)$ is independent of X, and where $f(x) = \sum_{i=1}^{d} x_i + \exp(-16(\sum_{i=1}^{d} x_i)^2)$. Tables 2 and 3 below report the following values: **Fast kernel time** stands for the computational times in seconds taken by the fast kernel summation algorithm; **Naive time** stands for the computational times in seconds of the naive version; **Accur Worst** stands for the maximum relative error of the fast sum algorithm on the whole grid. For each evaluation point, this relative error is computed as $|E_{\text{fast}} - E_{\text{naive}}|/|E_{\text{naive}}|$ where E_{fast} and E_{naive} are the estimates obtained by the fast sum updating algorithm and the naive summation algorithm, respectively; **Accur Worst Stab** stands for the maximum relative error of the fast sum algorithm with Møller-Kahan stabilization on the whole grid; **Accur Aver** stands for the average relative error on the grid. **Accur Aver Stab** stands for the average relative error of the fast sum algorithm with stabilization on the whole grid. We perform the tests on an Intel® Xeon® CPU E5-2680 v4 @ 2.40GHz (Broadwell)[2]. The code was written in C++ and is available in the StOpt[3] library (Gevret et al. (2016)). Subsection 4.1 focuses on kernel density estimation, while subsection 4.2 considers locally linear regression.

[2] ark.intel.com/products/91754/Intel-Xeon-Processor-E5-2680-v4-35M-Cache-2_40-GHz

[3] https://gitlab.com/stochastic-control/StOpt

4.1 Fast kernel density estimation

This subsection focuses on kernel density estimation (equation (20)). We implement and compare the fast kernel summation and the naive summation algorithms for different sample sizes N. We use the proportion $p = 15\%$ of neighboring sample points to include in each evaluation bandwidth (cf. subsection 3.2).

Table 2 reports our speed and accuracy results in the bivariate case. The fast summation algorithm is vastly faster than naive summation (less than one second versus more than seven hours for $1,28$ million points for example). Moreover, we observe a very good accuracy, even without using any summation stabilization algorithm (subsection 2.2).

Nb particles	20,000	40,000	80,000	160,000	320,000	640,000	1,280,000
Fast kernel time	0.02	0.02	0.04	0.09	0.20	0.43	0.89
Naive time	6.50	26	100	420	1,700	6,700	27,000
Accur Worst	3.2 E-12	1.9 E-12	3.0 E-10	4.5 E-10	4.0 E-11	7.2 E-08	3.5 E-09
Accur Worst Stab	4.4 E-13	1.6 E-13	3.3 E-12	1.7 E-11	4.1 E-13	1.1 E-10	3.0 E-11
Accur Aver	8.3 E-15	3.2 E-15	2.3 E-14	8.0 E-15	1.8 E-14	1.8 E-13	5.3 E-14
Accur Aver Stab	3.7 E-16	3.0 E-16	4.5 E-16	4.9 E-16	3.0 E-16	6.4 E-16	4.3 E-16

Table 2: 2D KDE: speed and accuracy

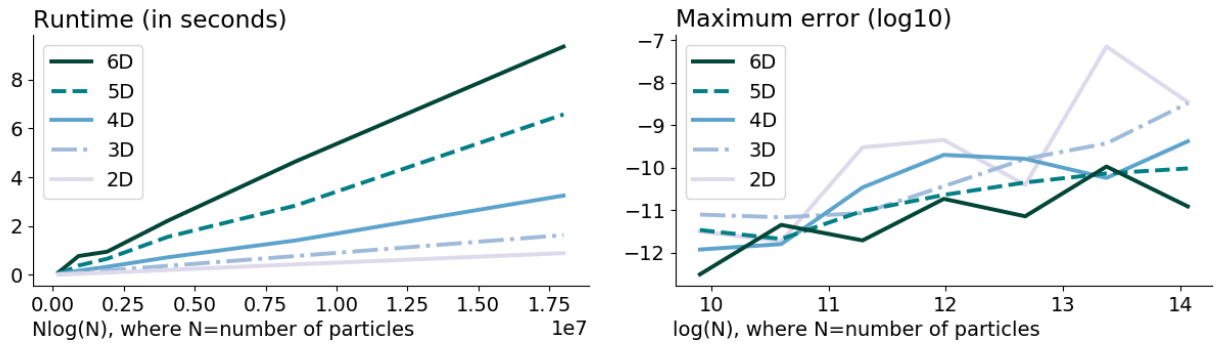

Figure 3: Fast KDE (left: runtime; right: log10 of maximum relative error)

Figure 3 reports multidimensional results up to dimension 6. The left-hand side figure demonstrates that the computational runtime is clearly in $N \log N$, while the right-hand

side figure shows that the accuracy is very good, even without summation stabilization.

4.2 Fast locally linear regression

For comprehensiveness, we now verify that our numerical observations from subsection 4.1 still hold for the harder locally linear regression problem (equation (22)). Table 3 reports our speed and accuracy results in the bivariate locally linear regression case. The results are qualitatively very similar to the kernel density estimation case. Figure 4 reports multivariate locally linear regression results up to dimension 6, demonstrating once again the $N \log N$ computational complexity and the very good accuracy. Note however that compared to the kernel density estimation case, the runtime grows much more quickly with the dimension of the problem. This is due to the higher number of terms to track to perform the locally linear regressions (23) compared to one single kernel density estimation.

Nb particles	20,000	40,000	80,000	160,000	320,000	640,000	1,280,000
Fast kernel time	0.03	0.06	0.12	0.26	0.52	1.10	2.22
Naive time	9.80	40	160	630	2,500	10,000	40,000
Accur Worst	8.6 E-11	2.3 E-11	5.6 E-10	4.9 E-10	2.6 E-09	2.7 E-09	4.9 E-09
Accur Aver	5.3 E-14	1.1 E-14	5.1 E-14	2.4 E-14	7.1 E-14	9.2 E-14	1.3 E-13

Table 3: 2D regression: speed and accuracy

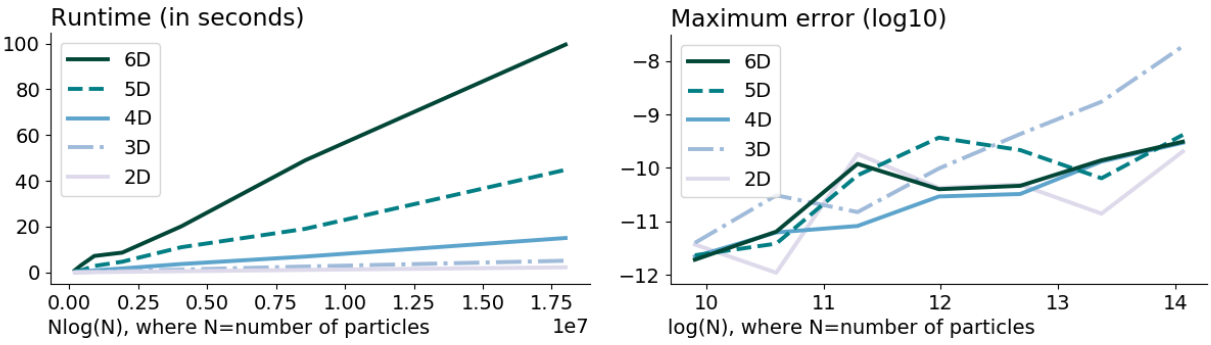

Figure 4: Fast local regression (left: runtime; right: log10 of maximum relative error)

5 Conclusion

Fast and exact kernel density estimation can be achieved by the *fast sum updating* algorithm (Gasser and Kneip (1989), Seifert et al. (1994)). With N input points drawn from the density to estimate, and M evaluation points where this density needs to be estimated, the fast sum updating algorithm requires $\mathcal{O}(M \log M + N \log N)$ operations, which is a vast improvement over the $\mathcal{O}(MN)$ operations required by direct kernel summation. This paper revisits the fast sum updating algorithm and extends it in several ways. The main contribution is the extension, for the first time, of the fast sum updating algorithm to the general multivariate case, opening the door to a vast class of practical density estimation and regression problems. The original concern in Seifert et al. (1994) with floating-point summation instability due to float-rounding errors can be completely addressed by the use of exact floating-point summation algorithms. Our numerical tests show that the cumulative float-rounding error is already negligible when using double-precision floats (in line with Fan and Marron (1994)), and that very simple compensated summation algorithms such as the Møller-Kahan algorithm can already bring significant accuracy improvements. In addition, we show that fast sum updating is compatible with a larger list of kernels, including the triangular kernel, the Silverman kernel, the cosine kernel, and the newly introduced hyperbolic cosine kernel, than what was usually assumed in the literature. We introduce the multivariate additive kernel, which greatly improves the speed of fast sum updating in high dimension compared to product kernels. Importantly, we describe how fast sum updating is compatible with balloon adaptive bandwidths, and propose a fast approximate k-nearest-neighbor algorithm for the adaptive bandwidth. The proposed multivariate extension does not impose any restriction on the input or output samples, but does require the evaluation points to lie on a possibly non-uniform grid. We describe how to prerotate the input data and construct a suitable grid to ease the interpolation of density estimates to any evaluation sample by multilinear interpolation or fast inverse distance weighting. Our multivariate kernel density and locally linear regression tests confirm numerically the vastly improved computational speed compared to naive kernel summation, as well as the accuracy and stability of the method. A natural area for future research would be to examine density estimation or regression applications for which computational speed is a major

issue. It would in particular be worth investigating how this algorithm compares, in terms of speed and accuracy, to alternative fast but approximate density estimation algorithms such as the Fast Fourier Transform with binning or the Fast Gauss Transform.

Acknowledgements The authors are grateful to the anonymous referees for their valuable comments. Xavier Warin acknowledges the financial support of ANR project CAESARS (ANR-15-CE05-0024).

A Kernels compatible with fast sum updating

This Appendix details how to implement the fast sum updating algorithm for several types of kernels. Three classes of kernels admit the type of separation between sources and targets required for the fast sum updating algorithm: polynomial kernels (A.1), absolute kernels (A.2) and cosine kernels (A.3). In addition, fast sum updating is still applicable to kernels which combine features from these three classes (A.4). In the literature, Seifert et al. (1994) covered the case of polynomial kernels, while Chen (2006) covered the Laplace kernel. The present paper extends the applicability of fast sum updating to the triangular kernel, cosine kernel, hyperbolic cosine kernel, and combinations such as the tricube and Silverman kernels. Specifically, we detail how to decompose the sums $\frac{1}{N}\sum_{i=1}^{N} K_h(x_i - z_j)x_i^p y_i^q = \frac{1}{Nh_j}\sum_{i=1}^{N} K\left(\frac{x_i - z_j}{h}\right) x_i^p y_i^q \ , j \in \{1, 2, \ldots, M\}$ into fast adaptable sums of the type $\mathcal{S}^{p,q}(f, [L, R]) := \sum_{i=1}^{N} f(x_i)x_i^p y_i^q \mathbb{1}\{L \leq x_i \leq R\}$. This last sum is a generalization of the sum (6) used in Section 2 for the Epanechnikov kernel. The additional $f(x_i)$ term in the sum is necessary for such kernels as the cosine or Laplace ones. Whenever possible, we will use adaptive kernels $h = h_j$ (balloon estimator, cf. subsection 3.2). Some kernels, such as polynomial kernels, can combine adaptive bandwidths with fast sum updating, but some other kernels cannot, as explained in the subsections below.

A.1 Polynomial kernels The class of polynomial kernels, in particular the class of symmetric beta kernels $K(u) = \frac{(1-u^2)^\alpha}{2^{2\alpha+1}\frac{\Gamma(\alpha+1)\Gamma(\alpha+1)}{\Gamma(2\alpha+2)}} \mathbb{1}\{|u| \leq 1\}$ includes several classical kernels: the uniform/rectangular kernel ($\alpha = 0$), the Epanechnikov/parabolic kernel ($\alpha = 1$), the quartic/biweight kernel ($\alpha = 2$) and the triweight kernel ($\alpha = 3$). Section 2 described how

to decompose the Epanechnikov kernel $K(u) = \frac{3}{4}(1-u^2)\mathbb{1}\{|u| \leq 1\}$. The other kernels within this class can be decomposed in a similar manner by expanding the power terms.

A.2 Absolute kernels The class of absolute kernels contains kernels based on the absolute value $|u|$, such as the triangular kernel and the Laplace kernel. For the triangular kernel, $K(u) = (1-|u|)\mathbb{1}\{|u| \leq 1\}$ and $\sum_{i=1}^N K\Big(\frac{x_i - z_j}{h_j}\Big) x_i^p y_i^q \mathbb{1}\left\{\left|\frac{x_i - z_j}{h_j}\right| \leq 1\right\} =$

$$\sum_{i=1}^N \left(1 - \frac{x_i - z_j}{h_j}\right) x_i^p y_i^q \mathbb{1}\left\{z_j \leq x_i \leq z_j + h_j\right\} + \sum_{i=1}^N \left(1 - \frac{z_j - x_i}{h_j}\right) x_i^p y_i^q \mathbb{1}\left\{z_j - h_j \leq x_i < z_j\right\}$$
$$= \left(1 + \frac{z_j}{h_j}\right) \mathcal{S}^{p,q}\left(1, [z_j, z_j + h_j]\right) - \frac{1}{h_j}\mathcal{S}^{p+1,q}\left(1, [z_j, z_j + h_j]\right)$$
$$+ \left(1 - \frac{z_j}{h_j}\right) \mathcal{S}^{p,q}\left(1, [z_j - h_j, z_j[\right) + \frac{1}{h_j}\mathcal{S}^{p+1,q}\left(1, [z_j - h_j, z_j[\right)$$

For the Laplace kernel, $K(u) = \frac{1}{2}\exp(-|u|)$ and $\sum_{i=1}^N K\big(\frac{x_i - z_j}{h}\big) x_i^p y_i^q =$

$$\frac{1}{2}\sum_{i=1}^N \exp\left(-\frac{x_i - z_j}{h}\right) x_i^p y_i^q \mathbb{1}\left\{z_j \leq x_i\right\} + \frac{1}{2}\sum_{i=1}^N \exp\left(-\frac{z_j - x_i}{h}\right) x_i^p y_i^q \mathbb{1}\left\{x_i < z_j\right\}$$
$$= \frac{1}{2}\exp\left(\frac{z_j}{h}\right) \mathcal{S}^{p,q}\left(\exp(-./h), [z_j, \infty[\right) + \frac{1}{2}\exp\left(-\frac{z_j}{h}\right) \mathcal{S}^{p,q}\left(\exp(./h),]-\infty, z_j[\right)$$

where $\exp(\pm./h)$ denotes the function $u \mapsto \exp(\pm u/h)$. Remark that we used a constant bandwidth h, as neither a balloon bandwidth $h = h_j$ nor a sample point bandwidth $h = h_i$ can separate the term $\exp\big(\frac{x_i - z_j}{h}\big)$ into a product of a term depending on i only and a term depending on j only. Note that an intermediate adaptive bandwidth approach of the type $\frac{x_i}{h_i} - \frac{z_j}{h_j}$ would maintain the ability to separate sources and targets for this kernel.

A.3 Cosine kernels For the cosine kernel, $K(u) = \frac{\pi}{4}\cos\big(\frac{\pi}{2}u\big)\,\mathbb{1}\{|u| \leq 1\}$ and $\sum_{i=1}^N K\big(\frac{x_i - z_j}{h}\big) x_i^p y_i^q \mathbb{1}\big\{\big|\frac{x_i - z_j}{h}\big| \leq 1\big\}$ is equal to

$$\frac{\pi}{4}\sum_{i=1}^N \left\{\cos\left(\frac{\pi}{2}\frac{x_i}{h}\right)\cos\left(\frac{\pi}{2}\frac{z_j}{h}\right) + \sin\left(\frac{\pi}{2}\frac{x_i}{h}\right)\sin\left(\frac{\pi}{2}\frac{z_j}{h}\right)\right\} x_i^p y_i^q \mathbb{1}\left\{z_j - h \leq x_i \leq z_j + h\right\}$$
$$= \frac{\pi}{4}\cos\left(\frac{\pi}{2}\frac{z_j}{h}\right) \mathcal{S}^{p,q}\left(\cos\left(\frac{\pi}{2}\frac{\cdot}{h}\right), [z_j - h, z_j + h]\right) + \frac{\pi}{4}\sin\left(\frac{\pi}{2}\frac{z_j}{h}\right) \mathcal{S}^{p,q}\left(\sin\left(\frac{\pi}{2}\frac{\cdot}{h}\right), [z_j - h, z_j + h]\right)$$

where we used that $\cos(\alpha - \beta) = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta)$. In a similar manner, one can define a new kernel based on the hyperbolic cosine function

$$K(u) = \frac{1}{4 - 2\frac{\sinh(\log(2+\sqrt{3}))}{\log(2+\sqrt{3})}} \left\{2 - \cosh(\log(2+\sqrt{3})u)\right\} \mathbb{1}\{|u| \leq 1\} \tag{19}$$

and use $\cosh(\alpha-\beta)=\cosh(\alpha)\cosh(\beta)-\sinh(\alpha)\sinh(\beta)$ to obtain a similar decomposition.

A.4 Combinations Finally, one can combine the polynomial, absolute, and cosine approaches together to generate additional kernels compatible with fast sum updating. This combination approach contains the tricube kernel $K(u)=\frac{70}{81}(1-|u|^3)^3\mathbb{1}\{|u|\leq 1\}$ (polynomial + absolute value) and the Silverman kernel $K(u)=\frac{1}{2}\exp\left(-\frac{|u|}{\sqrt{2}}\right)\sin\left(\frac{|u|}{\sqrt{2}}+\frac{\pi}{4}\right)$ (absolute value + cosine). Obtaining the updating equations for these combined kernels is a straightforward application of the decomposition tools used in the previous paragraphs.

B Multivariate kernel smoothers

In a multivariate setting, the kernel density estimator (1) becomes

$$\hat{f}_{\text{KDE}}(z):=\frac{1}{N}\sum_{i=1}^{N}K_{d,H}(x_i-z) \tag{20}$$

where $x_i=(x_{1,i},x_{2,i},\ldots,x_{d,i})$, $i\in\{1,2,\ldots,N\}$ are the input points, $z=(z_1,z_2,\ldots,z_d)$ is the evaluation point, and $K_{d,H}(u)=|H|^{-1/2}K_d(H^{-1/2}u)$ is a multivariate kernel with symmetric positive definite matrix bandwidth $H\in\mathbb{R}^{d\times d}$. Subsection 2.3.3 discusses the choice of kernel, and Condition 2 and subsection 3.2 discuss the possibility of adaptive bandwidth. The multivariate Nadaraya-Watson kernel regression estimator is:

$$\hat{f}_{\text{NW}}(z):=\frac{\sum_{i=1}^{N}K_{d,H}(x_i-z)y_i}{\sum_{i=1}^{N}K_{d,H}(x_i-z)} \tag{21}$$

where y_i are the output points. Finally, the multivariate locally linear regression is:

$$\hat{f}_{\text{L}}(z):=\min_{\alpha(z),\beta_1(z),\ldots,\beta_d(z)}\sum_{i=1}^{N}K_{d,H}(x_i-z)\left[y_i-\alpha(z)-\sum_{k=1}^{d}\beta_k(z)x_{k,i}\right]^2 \tag{22}$$

By solving the minimization problem (22), the multivariate locally linear regression estimate $\hat{f}_{\text{L}}(z)$ is explicitly given by:

$$\begin{bmatrix}1\\z_1\\z_2\\\vdots\\z_d\end{bmatrix}^T\begin{bmatrix}\sum_{i=1}^{N}K_{d,H}(z,x_i) & \sum_{i=1}^{N}x_{1,i}K_{d,H}(z,x_i) & \cdots & \sum_{i=1}^{N}x_{d,i}K_{d,H}(z,x_i)\\ \sum_{i=1}^{N}x_{1,i}K_{d,H}(z,x_i) & \sum_{i=1}^{N}x_{1,i}x_{1,i}K_{d,H}(z,x_i) & \cdots & \sum_{i=1}^{N}x_{1,i}x_{d,i}K_{d,H}(z,x_i)\\ \vdots & \vdots & \ddots & \vdots\\ \sum_{i=1}^{N}x_{d,i}K_{d,H}(z,x_i) & \sum_{i=1}^{N}x_{d,i}x_{1,i}K_{d,H}(z,x_i) & \cdots & \sum_{i=1}^{N}x_{d,i}x_{d,i}K_{d,H}(z,x_i)\end{bmatrix}^{-1}\begin{bmatrix}\sum_{i=1}^{N}y_iK_{d,H}(z,x_i)\\ \sum_{i=1}^{N}y_ix_{1,i}K_{d,H}(z,x_i)\\ \vdots\\ \sum_{i=1}^{N}y_ix_{d,i}K_{d,H}(z,x_i)\end{bmatrix} \tag{23}$$

To sum up, computing $\hat{f}_{\mathrm{KDE}}(z)$ requires one sum, computing $\hat{f}_{\mathrm{NW}}(z)$ requires two sums, and finally one can check that computing $\hat{f}_{\mathrm{L}}(z)$ requires a total of $(d+1)(d+4)/2$ sums. Remark that more general multivariate regressions, such as locally quadratic regression and locally polynomial Ridge regression, can also be implemented by fast sum updating.

References

Boldo, S., S. Graillat, and J.-M. Muller (2017). On the robustness of the 2sum and fast2sum algorithms. *ACM Transactions on Mathematical Software 44*(1), 4:1–4:14.

Bowman, A. and A. Azzalini (2003). Computational aspects of nonparametric smoothing with illustrations from the sm library. *Computational Statistics and Data Analysis 42*(4), 545–560.

Chen, A. (2006). Fast kernel density independent component analysis. In *Independent Component Analysis and Blind Signal Separation*, Volume 3889 of *Lecture Notes in Computer Science*, pp. 24–31. Springer.

Curtin, R., W. March, R. P., D. Anderson, A. Gray, and C. Isbell Jr. (2013). Tree-independent dual-tree algorithms. In *Proceedings of the 30th International Conference on Machine Learning*, Volume 28, pp. 1435–1443.

Demmel, J. and Y. Hida (2003). Accurate and efficient floating point summation. *SIAM Journal on Scientific Computing 25*(4), 1214–1248.

Epanechnikov, V. (1969). Non-parametric estimation of a multivariate probability density. *Theory of Probability and its Applications 14*(1), 153–158.

Fan, J. and J. Marron (1994). Fast implementation of nonparametric curve estimators. *Journal of Computational and Graphical Statistics 3*(1), 35–56.

Fukunaga, K. and L. Hostetler (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory 21*(1), 32–40.

Gasser, T. and A. Kneip (1989). Discussion: linear smoothers and additive models. *The Annals of Statistics 17*(2), 532–535.

Gevret, H., N. Langrené, J. Lelong, X. Warin, and A. Maheshwari (2016). STochastic OPTimization library in C++. Technical report, EDF Lab.

Gramacki, A. and J. Gramacki (2017). FFT-based fast computation of multivariate kernel density estimators with unconstrained bandwidth matrices. *Journal of Computational and Graphical Statistics 26*(2), 459–462.

Gray, A. and A. Moore (2001). 'n-body' problems in statistical learning. In *Advances in Neural Information Processing Systems 13*, pp. 521–527.

Gray, A. and A. Moore (2003). Nonparametric density estimation: Toward computational tractability. In *Proceedings of the SIAM International Conference on Data Mining*, pp. 203–211.

Greengard, L. and J. Strain (1991). The Fast Gauss Transform. *SIAM Journal on Scientific and Statistical Computing 12*(1), 79–94.

Greengard, L. and X. Sun (1998). A new version of the Fast Gauss Transform. *Documenta Mathematica Extra Volume ICM III*, 575–584.

Griebel, M. and D. Wissel (2013). Fast approximation of the discrete Gauss transform in higher dimensions. *Journal of Scientific Computing 55*(1), 149–172.

Härdle, W. and M. Müller (2000). Multivariate and semiparametric kernel regression. In M. Schimek (Ed.), *Smoothing and regression*, pp. 357–391. Wiley.

Härdle, W., M. Müller, S. Sperlich, and A. Werwatz (2004). *Nonparametric and Semiparametric Models*. Springer Series in Statistics. Springer.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer Series in Statistics. Springer.

Higham, N. (1993). The accuracy of floating point summation. *SIAM Journal on Scientific Computing 14*(4), 783–799.

Kahan, W. (1965). Pracniques: further remarks on reducing truncation errors. *Communications of the ACM 8*(1), 40.

Lambert, C., S. Harrington, C. Harvey, and A. Glodjo (1999). Efficient on-line nonparametric kernel density estimation. *Algorithmica 25*(1), 37–57.

Lang, D., M. Klaas, and N. de Freitas (2005). Empirical testing of fast kernel density estimation algorithms. Technical report, University of British Columbia.

Lee, D., A. Moore, and A. Gray (2006). Dual-tree Fast Gauss Transforms. In *Advances in Neural Information Processing Systems 18*, pp. 747–754.

Lee, D., P. Sao, R. Vuduc, and A. Gray (2014). A distributed kernel summation framework for general-dimension machine learning. *Statistical Analysis and Data Mining 7*(1), 1–13.

Linnainmaa, S. (1974). Analysis of some known methods of improving the accuracy of floating-point sums. *BIT Numerical Mathematics 14*(2), 167–202.

Loader, C. (1999). *Local Regression and Likelihood*. Statistics and Computing. Springer.

Loftsgaarden, D. and C. Quesenberry (1965). A nonparametric estimate of a multivariate density function. *The Annals of Mathematical Statistics 36*(3), 1049–1051.

McNamee, J. (2004). A comparison of methods for accurate summation. *ACM SIGSAM Bulletin 38*(1), 1–7.

Møller, O. (1965). Quasi double-precision in floating-point addition. *BIT Numerical Mathematics 5*(1), 37–50.

Morariu, V., B. Srinivasan, V. Raykar, R. Duraiswami, and L. Davis (2009). Automatic online tuning for fast Gaussian summation. In *Advances in Neural Information Processing Systems 21*, pp. 1113–1120.

Neal, R. (2015). Fast exact summation using small and large superaccumulators. Technical report, University of Toronto.

Ozawa, K. (1983). Analysis and improvement of Kahan's summation algorithm. *Journal of Information Processing 6*(4), 226–230.

Pan, V., B. Murphy, G. Qian, and R. Rosholt (2009). A new error-free floating-point summation algorithm. *Computers and Mathematics with Applications 57*(4), 560–564.

Priest, D. (1991). Algorithms for arbitrary precision floating point arithmetic. In *Proceedings 10th IEEE Symposium on Computer Arithmetic*, pp. 132–143.

Ram, P., D. Lee, W. March, and A. Gray (2009). Linear-time algorithms for pairwise statistical problems. In *Advances in Neural Information Processing Systems 22*, pp. 1527–1535.

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

Rump, S., T. Ogita, and S. Oishi (2008). Accurate floating-point summation Part I: Faithful rounding. *SIAM Journal on Scientific Computing 25*(1), 189–224.

Sampath, R., H. Sundar, and S. Veerapaneni (2010). Parallel Fast Gauss Transform. In *Proceedings of the IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*.

Scott, D. (1985). Averaged shifted histograms: effective nonparametric density estimators in several dimensions. *Annals of Statistics 13*(3), 1024–1040.

Scott, D. (2014). *Multivariate density estimation: theory, practice and visualization* (2nd ed.). Wiley Series in Probability and Statistics. Wiley.

Scott, D. and S. Sain (2005). Multivariate density estimation. In *Data Mining and Data Visualization*, Volume 24 of *Handbook of Statistics*, Chapter 9, pp. 229–261. Elsevier.

Seifert, B., M. Brockmann, J. Engel, and T. Gasser (1994). Fast algorithms for nonparametric curve estimation. *Journal of Computational and Graphical Statistics 3*(2), 192–213.

Shepherd, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 ACM National Conference*, pp. 517–524.

Silverman, B. (1982). Algorithm AS 176: Kernel density estimation using the Fast Fourier Transform. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 31*(1), 93–99.

Spivak, M., S. Veerapaneni, and L. Greengard (2010). The Fast Generalized Gauss Transform. *SIAM Journal on Scientific Computing 32*(5), 3092–3107.

Terrell, G. and D. Scott (1992). Variable kernel density estimation. *The Annals of Statistics 20*(3), 1236–1265.

Turlachand, B. and M. Wand (1996). Fast computation of auxiliary quantities in local polynomial regression. *Journal of Computational and Graphical Statistics 5*(4), 337–350.

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

Wand, M. and M. Jones (1995). *Kernel Smoothing*. Chapman & Hall.

Werthenbach, C. and E. Herrmann (1998). A fast and stable updating algorithm for bivariate nonparametric curve estimation. *Journal of Computational and Graphical Statistics 7*(1), 61–76.

Yang, C., R. Duraiswami, N. Gumerov, and L. Davis (2003). Improved Fast Gauss Transform and efficient kernel density estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 464–471.

Zhu, Y.-K. and W. Hayes (2010). Algorithm 908: Online exact summation of floating-point streams. *ACM Transactions on Mathematical Software 37*(3).

Finance for Energy Market Research Centre

Institut de Finance de Dauphine, Université Paris-Dauphine

1 place du Maréchal de Lattre de Tassigny

75775 PARIS Cedex 16

www.fime-lab.org