Fast multivariate empirical cumulative distribution function with connection to kernel density estimation

Nicolas Langrené[*], Xavier Warin[†]

May 6, 2020

Abstract

This paper revisits the problem of computing empirical cumulative distribution functions (ECDF) efficiently on large, multivariate datasets. Computing an ECDF at one evaluation point requires $\mathcal{O}(N)$ operations on a dataset composed of N data points. Therefore, a direct evaluation of ECDFs at N evaluation points requires a quadratic $\mathcal{O}(N^2)$ operations, which is prohibitive for large-scale problems. Two fast and exact methods are proposed and compared. The first one is based on fast summation in lexicographical order, with a $\mathcal{O}(N\log N)$ complexity and requires the evaluation points to lie on a regular grid. The second one is based on the divide-and-conquer principle, with a $\mathcal{O}(N\log(N)^{(d-1)\vee 1})$ complexity and requires the evaluation points to coincide with the input points. The two fast algorithms are described and detailed in the general d-dimensional case, and numerical experiments validate their speed and accuracy. Secondly, the paper establishes a direct connection between cumulative distribution functions and kernel density estimation (KDE) for a large class of kernels. This connection paves the way for fast exact algorithms for multivariate kernel density estimation and kernel regression. Numerical tests with the Laplacian kernel validate the speed and accuracy of the proposed algorithms. A broad range of large-scale multivariate density estimation, cumulative distribution estimation, survival function estimation and regression problems can benefit from the proposed numerical methods.

Keywords: fast CDF; fast KDE; empirical distribution function; survival function; Laplacian kernel; Sargan density; nonparametric copula estimation; fast convolution

MSC codes: 65C60; 62G30; 62G07; **ACM codes**: G.3; F.2.1; G.1.0

1 Introduction

Let $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$ be a sample of N input (source) points $x_i = (x_{1,i}, x_{2,i}, \dots, x_{d,i}) \in \mathbb{R}^d$ and output points $y_i \in \mathbb{R}$. Consider an evaluation (target) point $z = (z_1, z_2, \dots, z_d) \in \mathbb{R}^d$. We define a generalized multivariate empirical cumulative distribution function (ECDF) as follows:

$$F_N(z) = F_N(z; x, y) \triangleq \frac{1}{N} \sum_{i=1}^{N} y_i \mathbb{1}\{x_{1,i} \leq z_1, \dots, x_{d,i} \leq z_d\}\,. \tag{1}$$

In a similar manner, we define a generalized multivariate empirical survival function (ESF) (a.k.a. complementary cumulative distribution function) as follows:

$$\bar{F}_N(z) = \bar{F}_N(z; x, y) \triangleq \frac{1}{N} \sum_{i=1}^{N} y_i \mathbb{1}\{x_{1,i} > z_1, \dots, x_{d,i} > z_d\}\,. \tag{2}$$

The particular case $y \equiv 1$ corresponds to the classical joint empirical distribution function $F_N(z) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\{x_{1,i} \leq z_1, \dots, x_{d,i} \leq z_d\}$.

[*]CSIRO Data61, RiskLab Australia, nicolas.langrene@csiro.au
[†]EDF Lab, FiME (Laboratoire de Finance des Marchés de l'Énergie), warin@edf.fr

More generally, define the following multivariate ECDF:

$$F_N(z, \delta) = F_N(z, \delta; x, y) \triangleq \frac{1}{N} \sum_{i=1}^{N} y_i \mathbb{1}\{x_{1,i} \leq_{\delta_1} z_1, \ldots, x_{d,i} \leq_{\delta_d} z_d\} \tag{3}$$

where $\delta = \{\delta_1, \delta_2, \ldots, \delta_d\} \in \{-1, 1\}^d$, and where the generalized inequality operator $\leq_c$ corresponds to $\leq$ (lower or equal) if $c \geq 0$, and to $<$ (strictly lower) if $c < 0$. In particular $F_N(z) = F_N(z, 1; x, y)$ and $\bar{F}_N(z) = F_N(-z, -1; -x, y)$ respectively.

Cumulative distribution functions and their empirical counterparts are a cornerstone of statistical theory. In particular, classical statistical tests of equality of probability distributions such as the Kolmogorov-Smirnov, Cramér-von Mises and Anderson-Darling tests are based upon empirical distribution functions (Green & Hegazy, 1976).

The multivariate versions of these tests are methodologically and computationally more involved (Justel et al. 1997, Chiu & Liu 2009) due to the greater complexity of multivariate ECDFs (1) compared to their univariate counterpart.

A copula is a particular case of multivariate cumulative distribution function with uniform marginals (Durante & Sempi, 2010). Empirical copulas appear in the computation of multivariate measures of association (generalizing the bivariate Spearman rho, Schmid & Schmidt 2007, Schmid et al. 2010).

The focus of this article is on the numerical computation of generalized multivariate empirical cumulative distribution functions as defined in equation (3). As the computation of the ECDF (3) at one evaluation point z requires $\mathcal{O}(N)$ operations, a direct implementation of equation (3) 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 $x_1, x_2, \ldots, x_N$, a direct evaluation requires a quadratic $\mathcal{O}(N^2)$ operations.

The main contribution of this article is to propose an exact $\mathcal{O}(N \log N)$ algorithm to perform this task, based on independent data sorting in each dimension, combined with a fast lexicographical-sweep summation algorithm (subsection 2.1). Should the input data be already sorted, the computational complexity is reduced to an optimal $\mathcal{O}(N)$. This new algorithm is compared with the state-of-the-art for fast multivariate ECDF computation, namely the fast divide-and-conquer recursion of Bentley (1980) with $\mathcal{O}(N \log(N)^{(d-1)\vee 1})$ computational complexity (subsection 2.2).

The second main contribution of this article is to establish that a large class of kernel density estimators can be decomposed into a sum of ECDFs (subsection 3.1), which yields an exact $\mathcal{O}(N \log N)$ kernel density estimation approach in the lines of Langrené & Warin (2019), as well as a novel $\mathcal{O}(N \log(N)^{(d-1)\vee 1})$ kernel density estimation algorithm based on the divide-and-conquer approach of Bentley (1980). The table below summarizes the contributions of this paper.

Contributions	multivariate CDF	multivariate KDE
fast summation	this paper	Langrené & Warin (2019)
divide-and-conquer	Bentley (1980)	this paper

The class of compatible kernels contain popular kernels such as the uniform, Epanechnikov and Laplacian kernels (subsection 3.2). It also contains the lesser-known class of Sargan kernels, which can be used to uniformly approximate any incompatible kernel such as the Gaussian kernel to arbitrary precision (subsection 3.3).

The numerical tests reported in Section 4 illustrate the speed and accuracy of the proposed numerical methods. In practice, the fast summation algorithm requires the evaluation points to lie on a rectilinear grid, while the divide-and-conquer algorithm requires the evaluation points to be the same as the input points. These constraints mean that depending on the chosen algorithm and the set of evaluation points, an additional interpolation of the results might be necessary, the impact of which on accuracy can be deemed acceptable (Figures 4 and 6 in Section 4).

The contributions of this article can benefit any numerical procedure requiring a nonparametric estimation of univariate or multivariate cumulative density functions, survival functions or probability density functions. In particular, statistical tests of equality of probability distributions (Green & Hegazy, 1976), nonparametric empirical copula estimation (Choroś et al., 2010), kernel density estimation and kernel regression all benefit from the proposed fast computation of ECDFs.

2 Fast computation of multivariate cumulative distribution

This section presents two fast algorithms to compute the generalized empirical distributions (1-2-3).

The first one is based on the fast sum updating idea (Chen 2006, Langrené & Warin 2019). It requires a rectilinear evaluation grid, and its computational complexity is $O(N \log N)$, or $O(N)$ in the case of a uniform grid. It is described in subsection 2.1. To the authors' knowledge, it is the first time this computational technique is used to compute multivariate ECDFs.

The second one is based on the divide-and-conquer principle (Bentley 1980, Bouchard & Warin 2012, Lee & Joe 2018). It requires the evaluation points to be equal to the input points, and its computational complexity is $O(N \log(N)^{(d-1)\vee 1})$ where d is the dimension of the multivariate input data. It is described in subsection 2.2.

Another fast ECDF algorithm proposed in the literature can be found in Perisic & Posse (2005); however, this algorithm has been specifically designed for the bivariate case and cannot be extended to higher dimensional ECDFs.

2.1 Fast sum updating in lexicographical order

Let $z_j = (z_{1,j}, z_{2,j}, \dots, z_{d,j}) \in \mathbb{R}^d$, $j \in \{1, 2, \dots, M\}$, be a set of M evaluation (target) points.

We require this evaluation grid to be rectilinear, i.e., the M evaluation points $z_1, z_2, \dots, z_M$ lie on a regular grid with possibly non-uniform mesh, of dimension $M_1 \times M_2 \times \dots \times M_d = M$:

$$\mathbf{z} = \left\{ (z_{1,j_1}, z_{2,j_2}, \dots, z_{d,j_d}) \in \mathbb{R}^d,\ j_k \in \{1, 2, \dots, M_k\},\ k \in \{1, 2, \dots, d\} \right\}$$

For convenience, we extend the definition of the grid with the notational conventions $z_{k,0} \triangleq -\infty$ and $z_{k,M_k+1} \triangleq \infty$.

In each dimension $k \in \{1, 2, \dots, d\}$, the vector $(z_{k,1}, z_{k,2}, \dots, z_{k,M_k}) \in \mathbb{R}^{M_k}$ is assumed to be sorted in increasing order:

$$z_{k,1} < z_{k,2} < \dots < z_{k,M_k},\ k \in \{1, 2, \dots, d\}$$

We partition the input data $\mathbf{x}$ along this evaluation grid $\mathbf{z}$. For each evaluation grid index $(j_1, j_2, \dots, j_d) \in \{1, 2, \dots, M_1 + 1\} \times \dots \times \{1, 2, \dots, M_d + 1\}$ we define the following local sum

$$s_{j_1,j_2,\dots,j_d} := \frac{1}{N} \sum_{i=1}^{N} y_i \mathbb{1}\{z_{1,j_1-1} < x_{1,i} \le z_{1,j_1}, \dots, z_{d,j_d-1} < x_{d,i} \le z_{d,j_d}\} \tag{4}$$

Together, the sums (4) form a generalized multivariate histogram (classical histogram in the case $y \equiv 1$). For completeness, the computation of the local sums (4) is detailed in Appendix A.

In particular, using equation (1), the following key equality holds:

$$F_N(z) = \sum_{l_1=1}^{j_1} \sum_{l_2=1}^{j_2} \cdots \sum_{l_d=1}^{j_d} s_{l_1,l_2,\dots,l_d} \tag{5}$$

for any evaluation point $z = (z_{1,j_1}, z_{2,j_2}, \dots, z_{d,j_d}) \in \mathbf{z}$. We propose a simple fast summation algorithm, Algorithm 1, to compute the ECDFs $F_N(z)$ for every $z \in \mathbf{z}$ in lexicographical order based on the local sum decomposition (5). One can easily verify that the number of operations is proportional to $M_1 \times M_2 \times \dots \times M_d = M$. As Appendix A shows that the computation of the local sums (4) costs $\mathcal{O}(N \log N)$ operations (or only $\mathcal{O}(N)$ if the grid is uniform or the data already sorted), the overall computational complexity of Algorithm 1 is $\mathcal{O}(M + N \log N)$, or $\mathcal{O}(N \log N)$ when $M \approx N$ (respectively $\mathcal{O}(M + N)$ and $\mathcal{O}(N)$ when the grid is uniform or the data already sorted).

Remark 2.1. One can alternatively define the local sums (4) without the $1/N$ scaling factor, and apply the division by N to the output of Algorithm 1 (equation (5)). This modification ensures Algorithm 1 does not generate any float rounding error in the case when the y_i take integer values, which includes the classical CDF case $y \equiv 1$.

Algorithm 1: Fast joint empirical cumulative distribution function

Input: precomputed sums $s_{l_1,l_2,\dots,l_d}$

$\mathcal{S}_{1,l_2,l_3,\dots,l_d} = 0$

for ($j_1 = 1, \dots, M_1 + 1$) **do**

  $\mathcal{S}_{1,l_2,l_3,\dots,l_d}$ += $s_{j_1,l_2,l_3,\dots,l_d}$, $\forall l_k \in \{1,2,\dots,M_k+1\}$, $k \in \{2,3,\dots,d\}$

  ▷ Here $\mathcal{S}_{1,l_2,l_3,\dots,l_d} = \sum_{l_1=1}^{j_1} s_{l_1,l_2,\dots,l_d}$, $\forall l_k \in \{1,2,\dots,M_k+1\}$, $k \in \{2,3,\dots,d\}$

  $\mathcal{S}_{2,l_3,\dots,l_d} = 0$

  for ($j_2 = 1, \dots, M_2 + 1$) **do**

    $\mathcal{S}_{2,l_3,\dots,l_d}$ += $\mathcal{S}_{1,j_2,l_3,\dots,l_d}$, $\forall l_k \in \{1,2,\dots,M_k+1\}$, $k \in \{3,\dots,d\}$

    ▷ Here $\mathcal{S}_{2,l_3,\dots,l_d} = \sum_{l_1=1}^{j_1} \sum_{l_2=1}^{j_2} s_{l_1,l_2,\dots,l_d}$, $\forall l_k \in \{1,\dots,M_k+1\}$, $k \in \{3,\dots d\}$

    ⋮

    $\mathcal{S}_d = 0$

    for ($j_d = 1, \dots, M_d + 1$) **do**

      $\mathcal{S}_d$ += $\mathcal{S}_{d-1,j_d}$

      ▷ Here $\mathcal{S}_d = \sum_{l_1=1}^{j_1} \sum_{l_2=1}^{j_2} \cdots \sum_{l_d=1}^{j_d} s_{l_1,l_2,\dots,l_d}$

      ▷ $= F_N(z_{1,j_1}, z_{2,j_2}, \dots, z_{d,j_d}) = F_N(z)$ from equation (5)

    end

  end

end

Output: $F_N(z)$ for all $z \in \mathbf{z}$

2.2 Fast divide-and-conquer recursion

Consider the case when the evaluation points z_j are equal to the input points x_i. The calculation of the ECDFs $\{F_N(x_i)\}_{i=1,N}$ (equation (1)) corresponds to a domination problem in dimension d. An algorithm based on a recursive divide-and-conquer sequence has first been proposed in Bentley (1980) for this problem. An adaptation was proposed in Bouchard & Warin (2012) to solve this problem for the case of the calculation of conditional expectation using Malliavin weights. The computational complexity was shown to be $O(c(d)N\log(N)^{(d-1)\vee 1})$. This algorithm has been rediscovered recently in Lee & Joe (2018). They give an extensive study based on the quicksort algorithm providing an optimized version of the algorithm of Bentley (1980) and Bouchard & Warin (2012). Then they extend the approach to the mergesort algorithm.

In all the aforementioned papers, although the different authors insist that the algorithm can be generalized in any dimension, the algorithm descriptions are restricted to dimension 3 for the sake of clarity and simplicity. In the sequel we choose to provide the general d-dimensional version of this important algorithm, and refer to the aforementioned papers for the general conceptual ideas about the divide-and-conquer approach to this problem.

The pseudo-code is organized as follows: Algorithm 2 is the main function call, which triggers the divide-and-conquer recursive algorithm 3 w.r.t. dimension, starting from the last dimension. At each recursive iteration, the merge algorithm 4 is used in dimensions below the current dimension. The special 2D case is dealt with the call of the 1D merge algorithm 5. Further details regarding how the algorithm works:

- The n-dimensional merge algorithm 4 is defined using two sets of points κ_1 and κ_2 such that each point of κ_2 dominates the points of κ_1 in the dimension above the current one I_{dim}. A divide-and-conquer algorithm is used in the current dimension, splitting κ_1 (respectively κ_2) into two sets $\kappa_{1,1}$ and $\kappa_{1,2}$ (respectively $\kappa_{2,1}$ and $\kappa_{2,2}$) where each point in $\kappa_{1,2} \cup \kappa_{2,2}$ dominates all points in $\kappa_{1,1}$ and $\kappa_{2,1}$ in the current dimension.

- The n-dimensional merge is called recursively in the current dimension organizing a divide-and-conquer algorithm for the couple of sets where no clear dominance is available $((\kappa_{1,1}, \kappa_{2,1}), (\kappa_{12}, \kappa_{2,2}))$.

- For the couple of sets where dominance is clear in the current dimension $(\kappa_{1,1}, \kappa_{2,2})$, the n-dimensional merge algorithm is called in the dimension below. In the case when $I_{dim} = 2$, a

direct call to the one-dimensional merge algorithm 5 is performed.

Note that in the algorithm given below, we compute the F_N version excluding the current point. Adding the self contribution for all F_N is linear in time. In addition, some tests to check that sets are not empty are omitted for conciseness.

Algorithm 2: Calculate ECDF $F(x_j) = \sum_{i=1}^{N} y_i \mathbb{1}\{x_{1,i} < x_{1,j}, \ldots, x_{d,i} < x_{d,j}\}, \quad j = 1, N$

Input: $x = (x_i, \ldots, x_N)$, $y = (y_i, \ldots, y_N)$, for all $i = 1, \ldots, N$
Calculate the permutation ϕ^j , $j = 1, \ldots, d$ such that $x_{j,\phi^j(1)} \leq x_{j,\phi^j(2)} \leq \cdots \leq x_{j,\phi^j(N)}$
$F(x_i) = 0$ for $i = 1, \ldots, N$
RecurSplittingECDF(x, y, ϕ, F, N)
Output: $F(x_i)$ for all $i \in [1, N]$

Algorithm 3: Recursive splitting function **RecurSplittingECDF**

Input: x, y, F, $\phi^j(i)$ for $i = 1, M$, $j = 1, d$
▷ Split sorted data in two sets according to last dimension
$\kappa_1 = \{\phi^d(i), i = 1, \frac{M}{2}\}$, ϕ_1 with values in κ_1 s.t. $x_{j,\phi_1^j(1)} \leq x_{j,\phi_1^j(2)} \leq \cdots \leq x_{j,\phi_1^j(\frac{M}{2})}$, $j = 1, d$
$\kappa_2 = \{\phi^d(i), i = \frac{M}{2} + 1, M\}$, ϕ_2 in κ_2 s.t. $x_{j,\phi_2^j(1)} \leq x_{j,\phi_2^j(2)} \leq \cdots \leq x_{j,\phi_2^j(\frac{M}{2})}, j = 1, d$
RecurSplittingECDF$(x, y, \phi_1, F, M/2)$
RecurSplittingECDF$(x, y, \phi_2, F, M/2)$
if ($d > 2$) **then**
  ▷ Recursive merge for dimension above 2
  MergeNDECDF$(x, \phi_1, \phi_2, d-1, y, F, M/2, M/2)$
else
  ▷ Merge 1D
  Merge1D$(x, \phi_1^1, \phi_2^1, y, F)$,
end
Output: F updated

Algorithm 4: Recursive merge nD **MergeNDECDF** in given dimension I_{dim}

Input: x, y, F, $\phi_1^j(i)$, for all $i = 1, M_1$, $\phi_2^j(i)$, for all $i = 1, M_2$ with values in $[1, N]$ for $j = 1, I_{dim}$

$\kappa_1 = \{\phi_1^{I_{dim}}(i), i = 1, M_1\}$, $\kappa_2 = \{\phi_2^{I_{dim}}(i), i = 1, M_2\}$

▷ Merge the two sets involved and find median coordinate in dimension I_{dmin}: linear cost with the number of particles

$\kappa = \kappa_1 \cup \kappa_2$, x_{med} s.t; $\#\{x_j, j \in \kappa, x_{I_{dim},j} \leq x_{med}\} = \#\{x_j, j \in \kappa, x_{I_{dim},j} > x_{med}\}$

$\kappa_{l,1} = \{i \in \kappa_l, x_{I_{dim},i} \leq x_{med}\}$, $M_{l,1} = \#\kappa_{l,1}$, for $l = 1, 2$,

$\kappa_{l,2} = \{i \in \kappa_l, x_{I_{dim},i} > x_{med}\}$, $M_{l,2} = \#\kappa_{l,2}$, for $l = 1, 2$

▷ Sort each set for all dimension below or equal to I_{dim} : linear in time using $\phi_1^j(i)$, $\phi_2^j(i)$

Create $\phi_{l,m}^j(i)$, $i = 1, \dots M_{l,m}$ s.t. $\phi_{l,m}^j(i) \in \kappa_{l,m}$, and

$$x_{j,\phi_{l,m}^j(1)} \leq x_{j,\phi_{l,m}^j(2)} \leq \cdots \leq x_{j,\phi_{l,m}^j(M_{l,m})}, \text{ for } j \leq I_{dim}, \quad l = 1, 2, \quad m = 1, 2$$

. ▷ Merge for set $\kappa_{1,l}$ and $\kappa_{2,l}$ for $l = 1, 2$ in same dimension I_{dim}

MergeNDECDF$(x, \phi_{1,l}, \phi_{2,l}, I_{dim}, y, F, M_{1,l}, M_{2,l})$, for $l = 1, 2$

▷ Clear dominance relation between set below in current dimension: merge in dimension below

if ($I_{dim} == 2$) **then**

  ▷ Merge the set of 3D problem directly without recursion

  Merge1D$(x, \phi_{1,1}^1, \phi_{2,2}^1, y, F)$

else

  ▷ Merge in dimension below

  mergedNDECDF$(x, \phi_{1,1}, \phi_{2,2}, I_{dim} - 1, y, F, M_{1,l}, M_{2,l})$,

end

Output: F updated

Algorithm 5: Final merge function in dimension one : **Merge1D** , between $\{x_{\phi_1(i)}, i = 1, M_1\}$ and $\{x_{\phi_2(i)}, i = 1, M_2\}$

Input: x, y, F, ϕ_k s.t. $\phi_k(i) \leq \phi_k(i+1)$, for all $i = 1, M_k - 1$, $k = 1, 2$

$S = 0$, $j = 1$

for ($i = 1, M_2$) **do**

  while ($(x_{\phi_2(i),1} \geq x_{\phi_1(j),1})$ *and* $j \leq M_1$) **do**

    $S+ = y_{\phi_1(j)}$, $j = j + 1$

  end

  $F(\phi_2(i))+ = S$

  if ($j == M_1 + 1$) **then**

    for ($k = i + 1, M_2$) **do**

      $F(\phi_2(k))+ = S$

    end

    $i = M_2 + 1$

  end

end

Output: F updated

3 Fast kernel density estimation

This section establishes an explicit connection between the computation of empirical cumulative distribution functions and the problem of empirical density estimation, more specifically with kernel density estimation (KDE). The main consequence of this connection is that the fast empirical CDF algorithms introduced in Section 2 also provide a fast way to compute multivariate kernel density estimators.

3.1 CDF decomposition of KDE

Using the notations from Section 1, the (univariate) weighted kernel density estimator (aka Parzen-Rosenblatt estimator) at the evaluation point z is given by:

$$\hat{f}_{\mathrm{KDE}}(z) := \frac{1}{N}\sum_{i=1}^{N} w_i K_h(x_i - z) \tag{6}$$

where $K_h(u) := \frac{1}{h}K\left(\frac{u}{h}\right)$ with kernel K and bandwidth h. The classical KDE estimator corresponds to the weights $w_i \equiv 1$. Allowing general weights brings more flexibility, and does not affect the analysis of this section. For example w_i can contain the value of a response variable, as in local kernel regression estimation (Nadaraya 1964, Watson 1964). Another possible use of w_i concerns repeated values: should the input sample $(x_1, x_2, \ldots, x_N)$ contain repeated values, one can w.l.o.g. compute the kernel sum (6) on the unique values of the input sample, weighted by the time each value appears in the original sample (Titterington, 1980). Finally, this setting also encompasses kernel quantile estimators (Parzen 1979, Sheather & Marron 1990, Franke et al. 2009) and some kernel distribution function estimators (Azzalini 1981, Kim et al. 2005).

In the following, we focus on the Laplacian kernel, defined by

$$K(u) = \frac{1}{2}e^{-|u|} \tag{7}$$

Subsections 3.2 and 3.3 will discuss other possible kernel choices in detail. Following (6), the Laplacian kernel density estimator is defined by:

$$\hat{f}_{\mathrm{KDE}}(z) = \frac{1}{N}\sum_{i=1}^{N} \frac{w_i}{2h} e^{-\frac{|x_i - z|}{h}} \tag{8}$$

This kernel density estimator can be decomposed as follows

$$\begin{aligned}
\hat{f}_{\mathrm{KDE}}(z) =& \frac{1}{N}\sum_{i=1}^{N} \frac{w_i}{2h} e^{-\frac{|x_i - z|}{h}} \\
=& \frac{1}{2hN}\left(\sum_{i=1}^{N} w_i e^{\frac{x_i - z}{h}} \mathbb{1}\{x_i \le z\} + \sum_{i=1}^{N} w_i e^{\frac{z - x_i}{h}} \mathbb{1}\{x_i > z\}\right) \\
=& \frac{1}{2hN}\left(e^{-\frac{z}{h}}\sum_{i=1}^{N} w_i e^{\frac{x_i}{h}} \mathbb{1}\{x_i \le z\} + e^{\frac{z}{h}}\sum_{i=1}^{N} w_i e^{-\frac{x_i}{h}} \mathbb{1}\{x_i > z\}\right) \\
=& \frac{1}{2h}\left(e^{-\frac{z}{h}} F_N(z; x, we^{\frac{x}{h}}) + e^{\frac{z}{h}} \bar{F}_N(z; x, we^{-\frac{x}{h}})\right)
\end{aligned} \tag{9}$$

where the empirical CDF F_N and the empirical complementary CDF $\bar{F}_N$ are defined by equations (1) and (2) respectively.

Crucially, such a CDF decomposition of KDE also holds in the multivariate setting. The multivariate Laplacian kernel is defined by

$$K_d(u) = \frac{1}{2^d}e^{-|u|} = \frac{1}{2^d}e^{-\sum_{k=1}^{d}|u_k|} \tag{10}$$

and the weighted multivariate Laplacian kernel density estimator is given by

$$\hat{f}_{\mathrm{KDE}}(z) = \frac{1}{2^d N \Pi_{k=1}^{d} h_k}\sum_{i=1}^{N} w_i e^{-\left|\frac{x_i - z}{h}\right|} = \frac{1}{2^d N \Pi_{k=1}^{d} h_k}\sum_{i=1}^{N} w_i e^{-\sum_{k=1}^{d}\frac{|x_{k,i} - z_k|}{h_k}} \tag{11}$$

where $h = (h_1, h_2, \ldots, h_d) \in \mathbb{R}^d$ is a multivariate bandwidth. The general matrix bandwidth case is discussed in Appendix B.

Using the same approach as equation (9), the sum (11) can be decomposed as follows:

$$\begin{aligned}
\hat{f}_{\mathrm{KDE}}(z) =& \frac{1}{2^d N \Pi_{k=1}^d h_k} \sum_{i=1}^N w_i \prod_{k=1}^d \left(e^{-\frac{z_k}{h_k}} e^{\frac{x_{k,i}}{h_k}} \mathbb{1}\{x_{k,i} \le z_k\} + e^{\frac{z_k}{h_k}} e^{-\frac{x_{k,i}}{h_k}} \mathbb{1}\{-x_{k,i} < -z_k\} \right) \\
=& \frac{1}{2^d N \Pi_{k=1}^d h_k} \sum_{i=1}^N w_i \sum_{\delta \in \{-1,1\}^d} \prod_{k=1}^d e^{-\frac{\delta_k z_k}{h_k}} e^{\frac{\delta_k x_{k,i}}{h_k}} \mathbb{1}\{\delta_k x_{k,i} \le_{\delta_k} \delta_k z_k\} \\
=& \frac{1}{2^d \Pi_{k=1}^d h_k} \sum_{\delta \in \{-1,1\}^d} e^{-\sum_{k=1}^d \frac{\delta_k z_k}{h_k}} \frac{1}{N} \sum_{i=1}^N w_i e^{\sum_{k=1}^d \frac{\delta_k x_{k,i}}{h_k}} \mathbb{1}\{\delta_1 x_{1,i} \le_{\delta_1} \delta_1 z_1, \ldots, \delta_d x_{d,i} \le_{\delta_d} \delta_d z_d\} \\
=& \frac{1}{2^d \Pi_{k=1}^d h_k} \sum_{\delta \in \{-1,1\}^d} e^{-\sum_{k=1}^d \frac{\delta_k z_k}{h_k}} F_N(\delta z, \delta; \delta x, y) \qquad (12)
\end{aligned}$$

with $y_i = y_i(\delta) := w_i e^{\sum_{k=1}^d \frac{\delta_k x_{k,i}}{h_k}}$, where we used the definition of the generalized empirical CDF $F_N(z, \delta; x, y)$ (equation (3)) and its generalized inequality operator $\le_c$.
Equation (12) shows that the computation of the multivariate Laplacian kernel density estimator (28) can be decomposed into the computation of 2^d generalized empirical CDF (3), which can be computed efficiently using the algorithms described in Section 2.

3.2 Compatible kernels

In the previous subsection, we used the Laplacian kernel (7)-(10) to illustrate the concept of CDF decomposition of KDE. Such a decomposition is not restricted to the Laplacian kernel; actually, a large class of kernels (though not all kernels) is compatible with such a decomposition. Let us start with the simplest one, namely the uniform kernel

$$K(u) = \frac{1}{2} \mathbb{1}_{\{|u| \le 1\}}. \qquad (13)$$

The weighted uniform kernel density estimator is given by

$$\hat{f}_{\mathrm{KDE}}(z) = \frac{1}{N} \sum_{i=1}^N \frac{w_i}{2h} \mathbb{1}_{\{|\frac{x_i - z}{h}| \le 1\}} \qquad (14)$$

and can be decomposed as follows:

$$\begin{aligned}
\hat{f}_{\mathrm{KDE}}(z) =& \frac{1}{2hN} \left(\sum_{i=1}^N w_i \mathbb{1}_{\{x_i \le z+h\}} - \sum_{i=1}^N w_i \mathbb{1}_{\{x_i < z-h\}} \right) \\
=& \frac{F_N(z+h, 1; x, w) - F_N(z-h, -1; x, w)}{2h}. \qquad (15)
\end{aligned}$$

The multivariate uniform kernel density estimator is given by

$$K_d(u) = \frac{1}{2^d} \mathbb{1}_{\{\|u\|_\infty \le 1\}} \qquad (16)$$

and its corresponding weighted multivariate kernel density estimator

$$\hat{f}_{\mathrm{KDE}}(z) = \frac{1}{2^d N \Pi_{k=1}^d h_k} \sum_{i=1}^N w_i \mathbb{1}_{\{\|\frac{x_i - z}{h}\|_\infty \le 1\}} = \frac{1}{2^d N \Pi_{k=1}^d h_k} \sum_{i=1}^N w_i \prod_{k=1}^d \mathbb{1}_{\{|\frac{x_{k,i} - z_k}{h_k}| \le 1\}} \qquad (17)$$

can be decomposed as follows:

$$\begin{aligned}
\hat{f}_{\mathrm{KDE}}(z) =& \frac{1}{2^d N \Pi_{k=1}^d h_k} \sum_{i=1}^N w_i \prod_{k=1}^d \left(\mathbb{1}_{\{x_{k,i} \le z_k + h_k\}} - \mathbb{1}_{\{x_{k,i} < z_k - h_k\}} \right) \\
=& \frac{1}{2^d N \Pi_{k=1}^d h_k} \sum_{i=1}^N w_i \sum_{\delta \in \{-1,1\}^d} \prod_{k=1}^d \delta_k \mathbb{1}\{x_{k,i} \le_{\delta_k} z_k + \delta_k h\} \\
=& \frac{1}{2^d \Pi_{k=1}^d h_k} \sum_{\delta \in \{-1,1\}^d} \left(\prod_{k=1}^d \delta_k \right) F_N(z + \delta h, \delta; x, w). \qquad (18)
\end{aligned}$$

The uniform kernel is the simplest example of the large compatible class of kernels called symmetric beta kernels (Marron & Nolan 1988, Duong 2015), defined in the univariate case by:

$$K(u) = \frac{(1-u^2)^\alpha}{2^{2\alpha+1}B(\alpha+1,\alpha+1)}\mathbb{1}\{|u| \leq 1\} \tag{19}$$

where we used the Beta function $B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$. This class of kernels includes the uniform ($\alpha = 0$), Epanechnikov ($\alpha = 1$), biweight ($\alpha = 2$) and triweight ($\alpha = 3$) as particular cases. The fast sum updating decompositions in Gasser & Kneip (1989) and Seifert et al. (1994) (univariate case) and Langrené & Warin (2019) (multivariate case) can be recognised as CDF decompositions and show that the class (19) in particular is compatible with CDF decomposition. While equivalent to fast sum updating decomposition, one can argue that kernel sum decomposition in terms of CDFs makes the approach clearer and easier to understand, especially in the multivariate setting (see equations (12) and (18)).

In view of this discussion, we can infer from Langrené & Warin (2019) that other kernels such as the tricube kernel $K(u) = \frac{70}{81}(1-|u|^3)^3\mathbb{1}\{|u| \leq 1\}$ and the cosine kernel $K(u) = \frac{\pi}{4}\cos\left(\frac{\pi}{2}u\right)\mathbb{1}\{|u| \leq 1\}$ admit a CDF decomposition of KDE. Kernels based around the Laplacian kernel, such as the Silverman kernel $K(u) = \frac{1}{2}\exp\left(-\frac{|u|}{\sqrt{2}}\right)\sin\left(\frac{|u|}{\sqrt{2}}+\frac{\pi}{4}\right)$ are also compatible, and one can build upon compatible kernels to create new ones, as shown in subsections 3.3 and 3.4.

3.3 New compatible infinite-support kernels

Unfortunately, some kernels are simply incompatible with CDF decomposition. They are such that the term $K(\frac{x-z}{h})$ cannot be decomposed into terms depending on x only and terms depending on z only. Most incompatible kernels have unbounded support, such as the logistic kernel $K(u) = \frac{1}{e^u+2+e^{-u}}$, the Cauchy kernel $K(u) = \frac{1}{\pi(1+u^2)}$, the Fejér-de la Vallée Poussin kernel $K(u) = \frac{1}{\pi}\frac{\sin^2(u)}{u^2}$, and most importantly the popular Gaussian kernel $K(u) = \frac{1}{\sqrt{2\pi}}e^{-\frac{u^2}{2}}$.

As the finite-support Epanechnikov kernel $K(u) = \frac{3}{4}\left(1-u^2\right)\mathbb{1}\{|u| \leq 1\}$ is known to be optimal in terms of asymptotic mean integrated squared error (AMISE, Epanechnikov 1969), one can wonder whether such limitation is actually problematic in practice. However, infinite-support kernels are not devoid of merit for multiple reasons. For example, more robust non-asymptotic Fourier-based kernel selection criteria rule out the Epanechnikov kernel (Cline 1988, Tsybakov 2009) and recommend infinite-support kernels of Fejér type, in particular the Fejér-de la Vallée Poussin kernel (Stepanova 2013, Kosta & Stepanova 2015). Moreover, infinite-support kernel have been recommended for consistent likelihood cross-validation (Brewer 2000, Zhang et al. 2006, Hofmeyr 2020), and for tail probability estimation (Lall & Moon, 1993). Finally, kernels with unbounded support produce smooth prediction functions, which is a desirable feature for density visualization (Berthold et al., 2010).

As pointed out in Hofmeyr (2020), all known infinite-support kernels compatible with fast recursions are based around the Laplacian kernel (7), which is why subsection 3.1 focused on this important kernel. In the multivariate case, infinite-support kernels are more straightforward to decompose into CDFs than finite-support kernels. Indeed the decomposition of multivariate Beta kernels in Langrené & Warin 2019 requires the support of the kernel to be a hyperrectangle, which holds for product kernels but not for radially symmetric kernels. By contrast, equation (12) shows that obviously no such limitation exists for the Laplacian kernel.

In this subsection, we introduce an important class of kernels, known as *Sargan kernels* (Goldfeld & Quandt 1981; or double Gamma kernel sums, Nguyen & Chen 2009) which is compatible with fast recursion and can be used to approximate all the incompatible kernels mentioned so far. It is defined by

$$K(u) = \frac{1+\sum_{k=1}^p \gamma_k c^k |u|^k}{1+\sum_{k=1}^p \gamma_k k!}\frac{c}{2}e^{-c|u|} \tag{20}$$

with $c > 0$ and $\gamma_k \geq 0$, $k = 1, 2, \ldots, p$. It is obtained by multiplying the Laplacian kernel by a polynomial term in $|u|$. Such a distribution occurs when averaging $p+1$ i.i.d. Laplace distributions (Craig 1932, Weida 1935, Kotz et al. 2001).

As pointed out in Kafaei & Schmidt (1985), the theoretical foundation for considering kernels of the type (20) is the generalization of the Stone-Weierstrass theorem in Stone (1962, Section 11) which states that any continous function can be uniformly approximated by functions of the form (20) (without the scaling constant). In particular, any continuous density/kernel function can be uniformly approximated by (20) to arbitrary precision for sufficiently large p. This includes all the kernels incompatible with fast recursion such as the Gaussian kernel.

The case $p = 0$ corresponds to the Laplacian kernel (7). One shortcoming of the Laplacian kernel is its non-differentiability at zero, which is undesirable for density plotting for example. By contrast, the class of kernels (20) contains differentiable kernels as particular cases, for example

$$K^{(1)}(u) = \frac{c}{4}\left(1 + c\,|u|\right) e^{-c|u|} \tag{21}$$

$$K^{(2)}(u) = \frac{c}{6}\left(1 + c\,|u| + \frac{c^2}{2}\,|u|^2\right) e^{-c|u|} \tag{22}$$

see Goldfeld & Quandt (1981). One can see that the polynomial component brings additional regularity to the kernel: while the Laplacian kernel is continuous but not $\mathcal{C}^1$ (as it is not continuously differentiable at $u = 0$), the kernel $K^{(1)}$ (21) is $\mathcal{C}^2$ and the kernel $K^{(2)}$ (22) is $\mathcal{C}^3$. The general case $K^{(p)}$, $p \geq 1$, is defined in Appendix C, and is of class $\mathcal{C}^{p+1}$.

The free parameter $c > 0$ can be chosen in different ways: one can fix it to 1 for simplicity, to the value defining the canonical shape of the kernel ($c = \left(\int_{\mathbb{R}} u^2 K(u)du\right)^{2/5} / \left(\int_{\mathbb{R}} K^2(u)du\right)^{1/5}$, Marron & Nolan 1988), or in such a way as to ease the visual comparison of the kernel shape to some other kernels. More specifically, setting c such that the value $K^{(p)}(0) = \frac{c}{2(p+1)}$ matches the one of another kernel makes visual comparison easier, for example setting $c = (p+1)$ for comparison to the Laplacian kernel (Figure 1) or $c = \frac{2(p+1)}{\sqrt{2\pi}}$ for comparison to the Gaussian kernel (Figure 2).

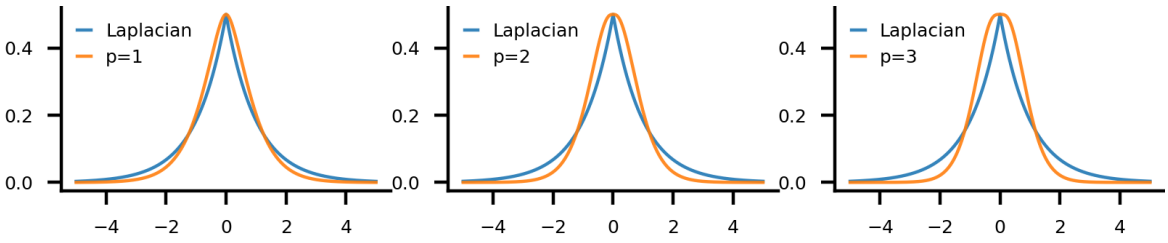

Figure 1: Comparison to Laplacian kernel

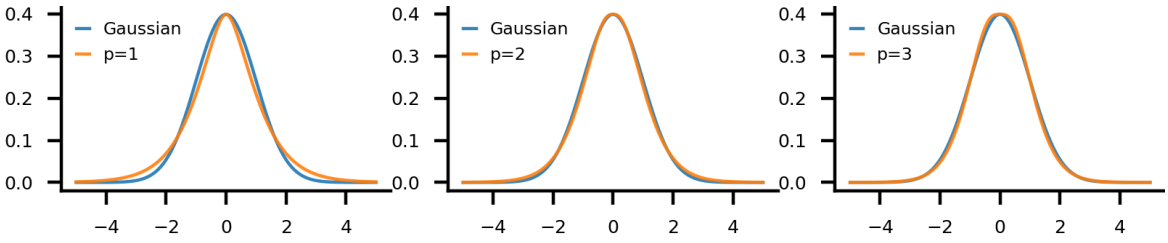

Figure 2: Comparison to Gaussian kernel

In Goldfeld & Quandt (1981), the motivation to investigate the class of distributions (20) was to approximate the Gaussian distribution by a more tractable distribution with explicit integrals (see also Missiakoulis 1983, Kafaei & Schmidt 1985, Tse 1987 and Hadri 1996 for specific kernel suggestions within the class (20)). Figure 2 suggests that computationally-attractive low-order Sargan kernels such as (22) or even (21) might suffice to approximate the shape of a Gaussian kernel. In the context of kernel density estimation, the fact that Sargan kernels are compatible with fast recursions and CDF decompositions make them even more attractive than Gaussian kernels.

3.4 New compatible higher-order kernels

Finally, another interesting set of kernels is the class of higher-order kernels.

Definition 3.1. (see for example Silverman 1986). A kernel K is said to be of order p if and only if

$$\int u^j K(u)du = \begin{cases} 1 & \text{if } j = 0 \\ 0 & \text{if } 1 \leq j \leq p-1 \\ c_k \neq 0, |c_k| < \infty & \text{if } j = p \end{cases}$$

The order p of a kernel is even when K is chosen symmetric. The kernel order has a direct connection to the best AMISE, namely $\mathcal{O}\left(N^{-\frac{p}{2p+1}}\right)$, of the KDE estimator (Gasser et al. 1985, Silverman 1986). This suggests that high-order kernels should asymptotically perform better (though see Silverman (1986) and Marron & Wand (1992) on the usefulness of such kernels on moderate sample sizes).

It is known that any kernel defined as a symmetric probability density function with finite variance is necessarily of order 2 (Schucany 1989, Jones & Foster 1993). One consequence is that kernels of order $p > 2$ necessarily take negative values in places.

Examples of fourth-order kernels include $K(u) = \frac{9}{8}\left(1 - \frac{5}{3}u^2\right)\mathbb{1}\{u \leq |1|\}$ (Bartlett, 1963), and $K(u) = \frac{15}{32}\left(3 - 10u^2 + 7u^4\right)\mathbb{1}\{|u| \leq 1\}$ (Gasser et al., 1985) which, being polynomial kernels, are compatible with fast recursion (see subsection 3.2). More generally, there exists various ways to turn a second-order kernel into a fourth order kernel (Schucany & Sommers 1977, Jones & Foster 1993, Devroye 1997). For example, K being a second-order kernel, the kernels $\frac{4}{3}K(u) - \frac{1}{6}K\left(\frac{u}{2}\right)$, $\frac{3}{2}K(u) + \frac{1}{2}uK'(u)$, and $\frac{(s_4 - s_2u^2)K(u)}{s_4 - s_2^2}$, $s_p \triangleq \int_{\mathbb{R}} u^p K(u)du$ are known to be fourth-order kernels, among many other examples. In the case of the (second-order) Laplacian kernel $K(u) = \frac{1}{2}e^{-|u|}$ (equation (7)), we obtain the following infinite-support fourth-order kernels:

$$\frac{1}{3}\left(2e^{-|u|} - \frac{1}{4}e^{-\left|\frac{u}{2}\right|}\right) \tag{23}$$

$$\frac{1}{4}\left(3 - |u|\right)e^{-|u|} \tag{24}$$

$$\frac{1}{5}\left(3 - \frac{1}{4}u^2\right)e^{-|u|} \tag{25}$$

which are all compatible with fast recursion (see the decompositions of the similar kernels from subsection 3.3 and Appendix C). Beyond these simple examples, the Laplacian kernel is also the root of the high-order class of Laguerre kernels (Berlinet, 1993).

As pointed out previously, the fourth-order kernels (23)-(24)-(25) necessarily take negative values, which can be deemed undesirable in a variety of application contexts. Higher-order kernels can be fixed to become non-negative (Glad et al. 2003, Oudjane & Musso 2005) without loss of statistical performance, however the fast recursion compatibility would be lost in the truncation process.

As a final remark, while there exists "superkernels" of infinite-order (Devroye 1992, Politis & Romano 1999, Hansen 2005, Chacón et al. 2007), to our knowledge none of them is compatible with fast recursion.

4 Numerical results

Finally, this section reports numerical speed and accuracy results for multivariate CDF computation (subsection 4.1) and multivariate KDE computation (subsection 4.2). Three approaches will be compared:

- the naive approach (direct computation of the sums (1) and (11) independently for each evaluation point),
- the fast summation approach (subsection 2.1), and
- the fast divide-and-conquer approach (subsection 2.2)

While the first approach is much slower than the other two, its results will serve as a benchmark for checking the accuracy of the other two methods.

Unless otherwise stated, we set the number of evaluation points M to be equal to the number of input points N:

- For the fast summation algorithm , we create an evaluation grid of shape $M_1 \times M_2 \times \ldots \times M_d$ with $M_1 = M_2 = \ldots = M_d \triangleq N^{1/d}$, which ensures that $M = N$.
- For the fast divide-and-conquer algorithm , the evaluation points are equal to the input points, which also ensures that $M = N$.
- For the naive algorithm, we set the evaluation sample to the evaluation grid when comparing to the fast summation algorithm, and to the input points when comparing to the divide-and-conquer algorithm.

The choices of input sample and bandwidth do not affect the speed or accuracy of the two proposed algorithms. For this reason, and for the sake of simplicity, we arbitrarily choose to draw the N input points from a d-dimensional Gaussian random variable $X \sim \mathbb{N}(0, \mathbf{1_d})$ and to fix the bandwidth to $h = 0.1$ in each dimension.
We perform the tests on an Intel® CPU i7-6820HQ @ 2.70GHz[1]. The code was written in C++ and is available in the StOpt[2] library (Gevret et al., 2020). Beyond CDF and KDE, StOpt implements fast kernel regression as well, as the weights ω_i in equation (6) can be chosen in such a way as to cover all the terms needed to perform a Nadaraya-Watson kernel regression or a locally linear kernel regression (see for example Appendix B in Langrené & Warin 2019).

4.1 Cumulative distribution function

Table 1 reports CDF calculation time (in seconds) on a bivariate example ($d = 2$) with the naive, fast summation and divide-and-conquer approaches. We observe that, as expected, the fast summation and divide-and-conquer methods offer a massive speedup compared to naive summation (around 1 second for the fast algorithms vs. more than two hours for the direct computation for 1,28 million points for example), and both fast computation times are of the same order (as expected since $\mathcal{O}(N\log(N)^{d-1}) = \mathcal{O}(N\log N)$ when $d = 2$).

Nb particles	20,000	40,000	80,000	160,000	320,000	640,000	1,280,000
Fast summation time	0.01	0.01	0.02	0.04	0.07	0.15	0.32
Divide-and-conquer time	0.01	0.02	0.05	0.1	0.29	0.66	1.5
Naive time	1.81	6.98	28	112	451	1939	7586

Table 1: 2D CDF calculation time (in seconds)

As the dimension increases, the computation time gap between divide-and-conquer and fast summation grows as expected, as shown on Table 2.

Nb particles	20,000	40,000	80,000	160,000	320,000	640,000	1,280,000
Fast summation time	0.01	0.02	0.05	0.09	0.22	0.47	0.96
Divide-and-conquer time	1.2	3.1	7.8	19.7	50.1	125.1	312.3

Table 2: 6D CDF calculation time (in seconds)

Figure 3 reports time calculation as a function of $N\log N$ for the fast summation approach and as a function of $c_d N \log(N)^{d-1}$ for the divide-and-conquer approach, with the scaling constants $c_3 = 3000$, $c_4 = 200$, $c_5 = 15$, $c_6 = 1$ chosen to make the visual comparison easier. The resulting straight lines confirm the theoretical complexity.

[1] https://ark.intel.com/content/www/fr/fr/ark/products/88970/intel-core-i7-6820hq-processor-8m-cache-up-to-3-60-ghz.html
[2] https://gitlab.com/stochastic-control/StOpt

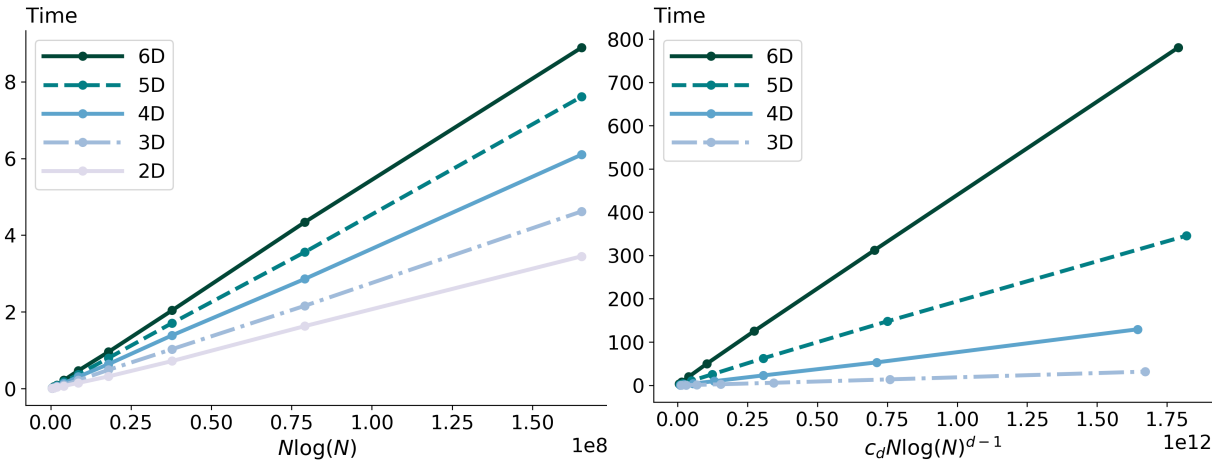

Figure 3: Runtime for CDF estimation (left: fast summation; right: divide-and-conquer)

The CDF values calculated by the naive approach and the two fast methods are exactly the same with no rounding error whatsoever since the $y \equiv 1$ case is a counting problem (integer count values with final division by N; see Remark 2.1 on the fast summation case).
Suppose now that we specifically want to estimate the CDF values at the input points. The divide-and-conquer approach does this by design, while the fast summation approach requires an interpolation from the grid points to the input points. Figure 4 reports, for different numbers M of evaluation points, the maximum interpolation error over the N sample points between the CDF values computed by fast summation and linearly interpolated to the input points, and the divide-and-conquer CDF values (taken as reference).

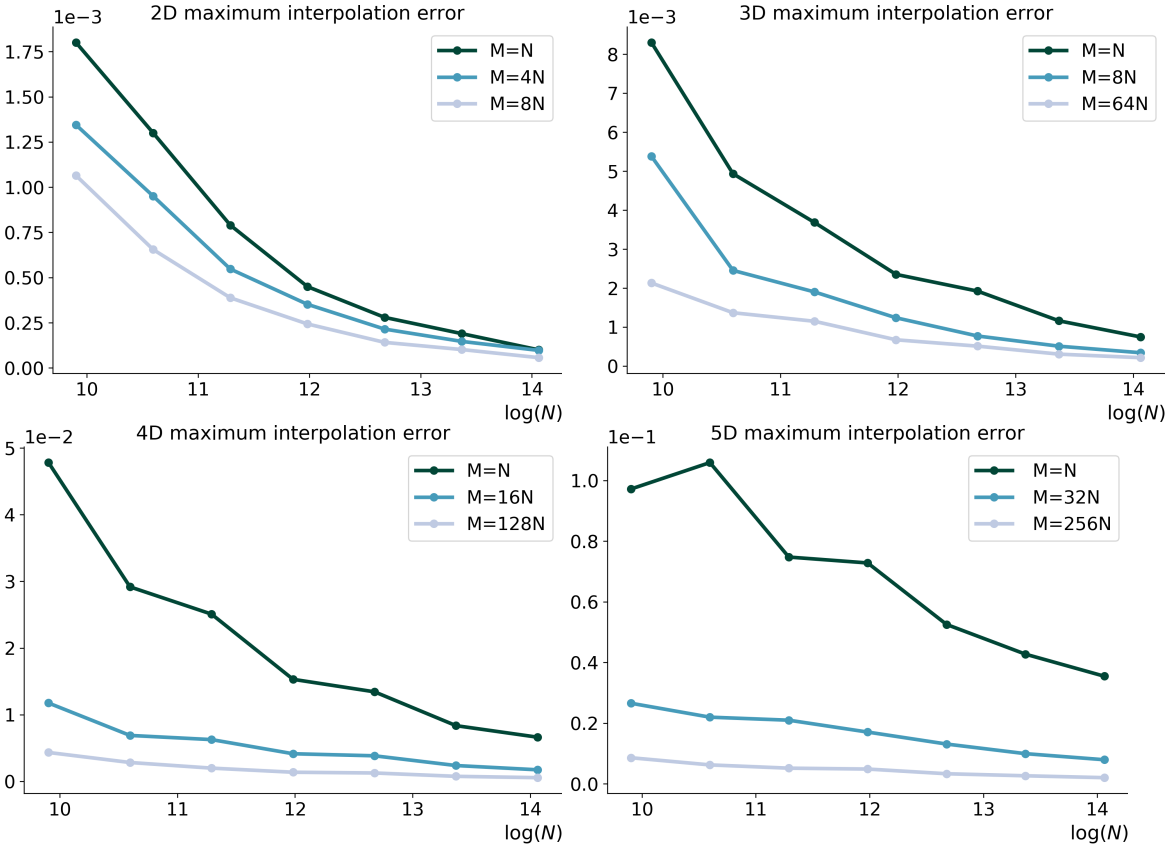

Figure 4: Maximal error on CDF for evaluation at sample points using fast summation and interpolation taking as reference the divide and conquer calculation.

When $M = N$, one can see that the worst-case interpolation error ranges from around 1 E-4 for $d = 2$ and $N = 1,280,000$ to around 1 E-1 for $d = 5$ and $N = 20,000$. This worst-case interpolation error is lower for small d and large N, and can be reduced by using a finer evaluation grid, i.e. taking M larger than N, as shown by the three curves on Figure 4. Beyond linear interpolation, one could also resort to higher-order interpolation to reduce this error. Nevertheless, these results show that computing CDF values at input points by fast summation + interpolation is a viable method, with better results in the small d high N case.

4.2 Kernel density estimation

We now perform the same numerical tests for kernel density estimation, more specifically Laplacian kernel density estimation (equation (12)).
Table 3 reports KDE calculation time (in seconds) on a bivariate example with the naive, fast summation and divide-and-conquer approaches. Once again, the fast summation and divide-and-conquer methods offer a massive speedup compared to naive summation (respectively 0.34s and 2.29s vs. almost eight hours for the direct computation of (11) for 0,64 million points for example), and both fast computation times are of the same order, up to a constant factor (around 6.0).

Nb particles	20,000	40,000	80,000	160,000	320,000	640,000
Fast summation time	0.01	0.01	0.04	0.08	0.14	0.34
Divide-and-conquer time	0.05	0.08	0.19	0.43	0.99	2.29
Naive time	28	115	439	1742	7198	28132

Table 3: 2D KDE calculation time (in seconds)

As in the CDF case, the computation time gap between the two fast methods grows with the dimension, as shown on Table 4.

Nb particles	20,000	40,000	80,000	160,000	320,000	640,000	1,280,000
Fast summation time	0.18	0.26	0.65	1.51	3.59	7.97	16.11
Divide-and-conquer time	15	41	111	294	777	2040	5344

Table 4: 6D KDE calculation time (in seconds)

Figure 5 reports time calculation as a function of $N\log N$ for the fast summation approach and as a function of $c_d N \log(N)^{d-1}$ for the divide-and-conquer approach (with scaling constants $c_3 = 4000$, $c_4 = 250$, $c_5 = 20$, $c_6 = 1$). Once again, the resulting straight lines confirm the theoretical complexity.

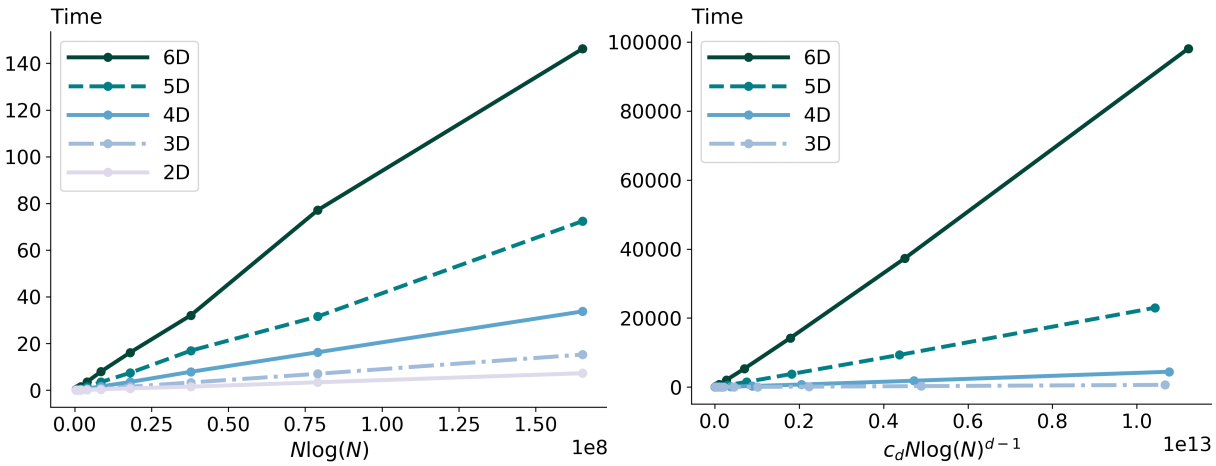

Figure 5: Runtime for KDE estimation (left: fast summation; right: divide-and-conquer)

As for accuracy, the maximum difference between the KDE values of the naive approach and those of both fast methods, caused by float rounding errors, remains below 1 E-14 independently of the dimension of the problem.

Finally, we also test the accuracy of the fast summation approach when the evaluation points are required to coincide with the input points, which requires an interpolation from the grid points. Figure 6 reports the maximum interpolation error over the N sample points between the linearly interpolated CDF values computed by fast summation and the divide-and-conquer CDF values.

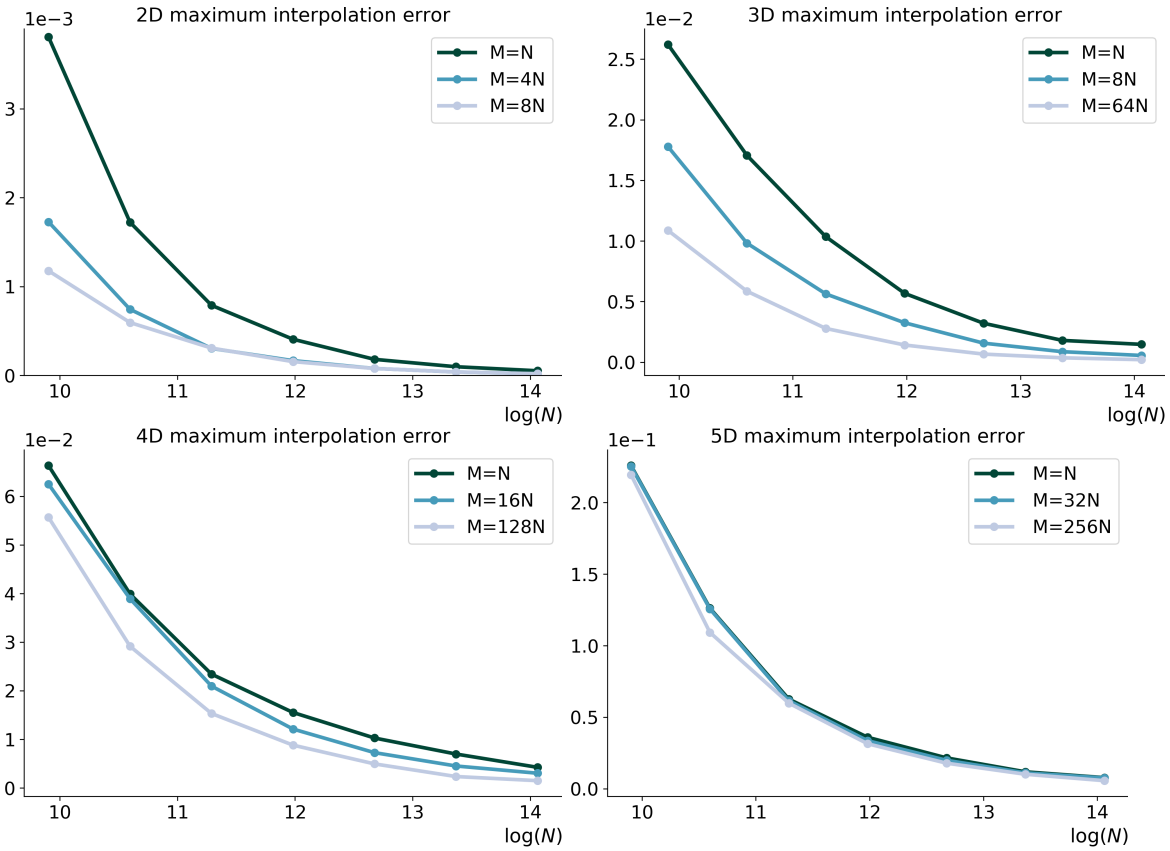

Figure 6: Maximal error on KDE evaluation at sample points using fast summation and interpolation taking as reference the divide and conquer calculation.

As in the CDF case, the worst-case interpolation error ranges between around 1 E-4 and 2 E-1, is smaller for small d, large N or large M. However, the accuracy improvements obtained by increasing M get smaller in higher dimension. Nevertheless, the fast summation + interpolation approach can still be considered a viable option for KDE estimation at the input data points, provided d is small or N is large.

5 Conclusion

A new algorithm based on fast summation in lexicographical order has been developed to efficiently calculate multivariate empirical cumulative distribution functions (ECDFs) with $O(N\log N)$ computational cost for N arbitrary data points and N evaluation points on a rectilinear grid. Numerical tests and comparisons to a state-of-the-art $\mathcal{O}(N\log(N)^{(d-1)\vee 1})$ divide-and-conquer algorithm confirm the speed of this exact algorithm.
Besides, we establish a multivariate decomposition formula of kernel density estimators (KDEs) into a weighted sum of generalized ECDFs for a large class of kernels. This connection leads to new fast KDE algorithms: one based on fast summation with $O(N\log N)$ complexity, and one based on divide-and-conquer recursion with $\mathcal{O}(N\log(N)^{(d-1)\vee 1})$ complexity.

The class of compatible kernels includes classical kernels such as the uniform, Epanechnikov and Laplacian kernels. We show that it also includes the lesser-known class of Sargan kernels, which can be used to approximation incompatible kernels such as the Gaussian kernel.
Following our computational breakthrough, several possible extensions and potential future work come to mind:

- The investigation of computational methods for the related kernel *distribution* estimation problem (Yamato 1973, Liu & Yang 2008) based on the algorithmic approaches developed in this paper.
- Further investigation of the promising class of multivariate Sargan kernels, in particular their ability to approximate multivariate kernels, and their ability to speed up statistical techniques based on multivariate Gaussian variables using the fast algorithms from this paper.
- The application of fast kernel regression for image processing, as uniform pixel grids are an ideal ground for Algorithm 1 for which its computational complexity is an optimal $\mathcal{O}(N)$.
- The comparison, more generally, of our algorithms to fast convolution methods such as the Fast Fourier Transform (FFT) for compatible convolution kernels.

References

Azzalini, A. (1981). A note on the estimation of a distribution function and quantiles by a kernel method. *Biometrika*, *68*(1), 326–328.
(cited on page 7)

Bartlett, M. (1963). Statistical estimation of density functions. *Sankhyā: The Indian Journal of Statistics, Series A*, *25*(3), 245–254.
(cited on page 11)

Bentley, J. L. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, *23*(4), 214–229.
(cited on pages 2, 3, and 4)

Berlinet, A. (1993). Hierarchies of higher order kernels. *Probability Theory and Related Fields*, *94*(4), 489–504.
(cited on page 11)

Berthold, M., Borgelt, C., Höppner, F., & Klawonn, F. (2010). *Guide to intelligent data analysis: how to intelligently make sense of real data.* Springer.
(cited on page 9)

Bouchard, B., & Warin, X. (2012). Monte Carlo valuation of American options: facts and new algorithms to improve existing methods. In *Numerical methods in finance* (pp. 215–255). Springer.
(cited on pages 3, 4, and 21)

Brewer, M. (2000). A Bayesian model for local smoothing in kernel density estimation. *Statistics and Computing*, *10*(4), 299–309.
(cited on page 9)

Chacón, J., Montanero, J., & Nogales, G. (2007). A note on kernel density estimation at a parametric rate. *Nonparametric Statistics*, *19*(1), 13–21.
(cited on page 11)

Chen, A. (2006). Fast kernel density independent component analysis. In *Independent component analysis and blind signal separation* (Vol. 3889, pp. 24–31). Springer.
(cited on page 3)

Chiu, S., & Liu, K. (2009). Generalized Cramér-von Mises goodness-of-fit tests for multivariate distributions. *Computational Statistics and Data Analysis*, *53*(11), 3817–3834.
(cited on page 2)

Choroś, B., Ibragimov, R., & Permiakova, E. (2010). Copula estimation. In P. Jaworski, F. Durante, W. Härdle, & T. Rychlik (Eds.), *Copula theory and its applications* (Vol. 198, pp. 77–91). Springer.
(cited on page 2)

Cline, D. (1988). Admissible kernel estimators of a multivariate density. *Annals of Statistics*, *16*(4), 1421–1427.
(cited on page 9)

Craig, A. (1932). On the distributions of certain statistics. *American Journal of Mathematics*, *54*(2), 353–366.
(cited on page 9)

Devroye, L. (1992). A note on the usefulness of superkernels in density estimation. *Annals of Statistics*, *20*(4), 2037–2056.
(cited on page 11)

Devroye, L. (1997). Universal smoothing factor selection in density estimation: theory and practice. *Test*, *6*(2), 223–320.
(cited on page 11)

Duong, T. (2015). Spherically symmetric multivariate beta family. *Statistics and Probability Letters*, *104*, 141–145.
(cited on page 9)

Durante, F., & Sempi, C. (2010). Copula theory: an introduction. In P. Jaworski, F. Durante, W. Härdle, & T. Rychlik (Eds.), *Copula theory and its applications* (Vol. 198, pp. 3–31). Springer.
(cited on page 2)

Epanechnikov, V. (1969). Non-parametric estimation of a multivariate probability density. *Theory of Probability and its Applications*, *14*(1), 153–158.
(cited on page 9)

Franke, J., Kreiss, J.-P., & Mammen, E. (2009). Nonparametric modeling in financial time series. In T. Andersen, R. Davis, J.-P. Kreiß, & T. Mikosch (Eds.), *Handbook of financial time series* (pp. 927–952). Springer.
(cited on page 7)

Gasser, T., & Kneip, A. (1989). Discussion: linear smoothers and additive models. *The Annals of Statistics*, *17*(2), 532–535.
(cited on page 9)

Gasser, T., Müller, H.-G., & Mammitzsch, V. (1985). Kernels for nonparametric curve estimation. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *47*(2), 238–252.
(cited on page 11)

Gevret, H., Langrené, N., Lelong, J., Lobato, R., Ouillon, T., Warin, X., & Maheshwari, A. (2020). *STochastic OPTimization library in C++* (Tech. Rep.). EDF Lab.
(cited on page 12)

Glad, I., Hjort, N., & Ushakov, N. (2003). Correction of density estimators that are not densities. *Scandinavian Journal of Statistics*, *30*(2), 415–427.
(cited on page 11)

Goldfeld, S., & Quandt, R. (1981). Econometric modelling with non-normal disturbances. *Journal of Econometrics*, *17*(2), 141–155.
(cited on pages 9 and 10)

Green, J., & Hegazy, Y. (1976). Powerful modified-EDF goodness-of-fit tests. *Journal of the American Statistical Association*, *71*(353), 204–209.
(cited on page 2)

Hadri, K. (1996). A note on Sargan densities. *Journal of Econometrics*, *71*(1–2), 285–290.
(cited on page 10)

Hansen, B. (2005). Exact mean integrated squared error of higher order kernel estimators. *Econometric Theory*, *21*(6), 1031–1057.
(cited on page 11)

Hofmeyr, D. (2020). Fast exact evaluation of univariate kernel sums. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. (to appear)
(cited on pages 9 and 23)

Jones, M., & Foster, P. (1993). Generalized jackknifing and higher order kernels. *Journal of Nonparametric Statistics*, *3*(1), 81–94.
(cited on page 11)

Justel, A., Peña, D., & Zamar, R. (1997). A multivariate Kolmogorov-Smirnov test of goodness of fit. *Statistics and Probability Letters*, *35*(3), 251–259.
(cited on page 2)

Kafaei, M.-A., & Schmidt, P. (1985). On the adequacy of the "Sargan distribution" as an approximation to the normal. *Communications in Statistics - Theory and Methods*, *14*(3), 509–526.
(cited on page 10)

Kim, C., Bae, W., Choi, H., & Park, B. (2005). Non-parametric hazard function estimation using the Kaplan-Meier estimator. *Nonparametric Statistics*, *17*(8), 937–948.
(cited on page 7)

Kosta, O., & Stepanova, N. (2015). Efficient density estimation and value at risk using Fejér-type kernel functions. *Journal of Mathematical Finance*, *5*(5), 480–504.
(cited on page 9)

Kotz, S., Kozubowski, T., & Podgórski, K. (2001). *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*. Springer.
(cited on page 9)

Lall, U., & Moon, Y.-I. (1993). Kernel flood frequency estimators: bandwidth selection and kernel choice. *Water Resources Research*, *29*(4), 1003–1015.
(cited on page 9)

Langrené, N., & Warin, X. (2019). Fast and stable multivariate kernel density estimation by fast sum updating. *Journal of Computational and Graphical Statistics*, *28*(3), 596–608.
(cited on pages 2, 3, 9, 12, and 22)

Lee, D., & Joe, H. (2018). Efficient computation of multivariate empirical distribution functions at the observed values. *Computational Statistics*, *33*(3), 1413–1428.
(cited on pages 3 and 4)

Liu, R., & Yang, L. (2008). Kernel estimation of multivariate cumulative distribution function. *Journal of Nonparametric Statistics*, *20*(8), 661–677.
(cited on page 16)

Marron, J., & Nolan, D. (1988). Canonical kernels for density estimation. *Statistics and Probability Letters*, *7*(3), 195–199.
(cited on pages 9 and 10)

Marron, J., & Wand, M. (1992). Exact mean integrated squared error. *Annals of Statistics*, *20*(2), 712–736.
(cited on page 11)

Missiakoulis, S. (1983). Sargan densities which one? *Journal of Econometrics*, *23*(2), 223–233.
(cited on page 10)

Nadaraya, E. (1964). Theory of probability and its applications. *On estimating regression*, *9*(1), 141–142.
(cited on page 7)

Nguyen, T., & Chen, J. (2009). A connection between the double gamma model and Laplace sample mean. *Statistics and Probability Letters*, *79*(10), 1305–1310.
(cited on page 9)

Oudjane, N., & Musso, C. (2005). l^2-density estimation with negative kernels. In *Proceedings of the 4th international symposium on image and signal processing and analysis (ispa 2005)* (pp. 34–39). IEEE.
(cited on page 11)

Parzen, E. (1979). Nonparametric statistical data modeling. *Journal of the American Statistical Association*, *74*(365), 105–121.
(cited on page 7)

Perisic, I., & Posse, C. (2005). Projection pursuit indices based on the empirical distribution function. *Journal of Computational and Graphical Statistics*, *14*(3), 700–715.
(cited on page 3)

Politis, D., & Romano, J. (1999). Multivariate density estimation with general flat-top kernels of infinite order. *Journal of Multivariate Analysis*, *68*(1), 1–25.
(cited on page 11)

Schmid, F., & Schmidt, R. (2007). Multivariate extensions of Spearman's rho and related statistics. *Statistics and Probability Letters*, *77*(4), 407–416.
(cited on page 2)

Schmid, F., Schmidt, R., Blumentritt, T., Gaißer, S., & Ruppert, M. (2010). Copula-based measures of multivariate association. In P. Jaworski, F. Durante, W. Härdle, & T. Rychlik (Eds.), *Copula theory and its applications* (Vol. 198, pp. 209–236). Springer.
(cited on page 2)

Schucany, W. (1989). On nonparametric regression with higher-order kernels. *Journal of Statistical Planning and Inference*, *23*(2), 145–151.
(cited on page 11)

Schucany, W., & Sommers, J. (1977). Improvement of kernel type density estimators. *Journal of the American Statistical Association*, *72*(358), 420–423.
(cited on page 11)

Seifert, B., Brockmann, M., Engel, J., & Gasser, T. (1994). Fast algorithms for nonparametric curve estimation. *Journal of Computational and Graphical Statistics*, *3*(2), 192–213.
(cited on page 9)

Sheather, S., & Marron, J. (1990). Kernel quantile estimators. *Journal of the American Statistical Association*, *85*(410), 410–416.
(cited on page 7)

Silverman, B. (1986). *Density estimation for statistics and data analysis* (Vol. 26). Chapman & Hall.
(cited on page 11)

Stepanova, N. (2013). On estimation of analytic density functions in L_p. *Mathematical Methods of Statistics*, *22*(2), 114–136.
(cited on page 9)

Stone, M. (1962). A generalized Weierstrass approximation theorem. In R. Buck (Ed.), *Studies in modern analysis* (Vol. 1, pp. 30–87). Prentice Hall.
(cited on page 10)

Titterington, D. (1980). A comparative study of kernel-based density estimates for categorical data. *Technometrics*, *22*(2), 259–268.
(cited on page 7)

Tse, Y. (1987). A note on Sargan densities. *Journal of Econometrics*, *34*(3), 349–354.
(cited on page 10)

Tsybakov, A. (2009). *Introduction to nonparametric estimation*. Springer.
(cited on page 9)

Wand, M., & Jones, M. (1995). *Kernel smoothing*. Chapman & Hall.
(cited on page 22)

Watson, G. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, *26*(4), 359–372.
(cited on page 7)

Weida, F. (1935). On certain distribution functions when the law of the universe is Poisson's first law of error. *Annals of Mathematical Statistics*, *6*(2), 102–110.
(cited on page 9)

Yamato, H. (1973). Uniform convergence of an estimator of a distribution function. *Bulletin of Mathematical Statistics*, *15*(3–4), 69–78.
(cited on page 16)

Zhang, X., King, M., & Hyndman, R. (2006). A Bayesian approach to bandwidth selection for multivariate kernel density estimation. *Computational Statistics and Data Analysis*, *50*(11), 3009–3031.
(cited on page 9)

A Computation of local sums

This appendix details how to efficiently compute the local sums $s_{j_1,j_2,\dots,j_d}$ defined in equation (4) (subsection 2.1). Subsection A.1 details the general case, based on independent input data sorting in each dimension, for a $\mathcal{O}(N \log N)$ computational cost. Subsection A.2 details the uniform grid case: in this special case, the computational cost can be brought down to $\mathcal{O}(N)$ by using constant mesh divisions as a substitute to sorting.

A.1 General case

Algorithm 6: Fast computation of local sums by independent sorting in each dimension

Input: input sample $x_i = (x_{1,i}, \dots, x_{d,i})$, $i = 1, 2, \dots, N$

Input: evaluation grid $(z_{1,j_1}, z_{2,j_2}, \dots, z_{d,j_d})$, $j_k \in \{1, 2, \dots, M_k\}$, $k \in \{1, 2, \dots, d\}$

Define index matrix $\text{INDEX}[k, i]$ ▷ local sum index $\in \{1, 2, M_k + 1\}$
where $k = 1, 2, \dots, d$ and $i = 1, 2, \dots, N$

for ($k = 1, 2, \dots, d$) **do**

  Sort the set $\{x_{k,1}, \dots x_{k,N}\}$ in increasing order, using for example quicksort or mergesort ($\mathcal{O}(N \log N)$): define the permutation $\phi_k : \{1, 2, \dots, N\} \mapsto \{1, 2, \dots, N\}$ such that

$$x_{k,\phi_k(1)} < x_{k,\phi_k(2)} < \dots < x_{k,\phi_k(N)} \tag{26}$$

  $x_{\text{idx}} = 1$ ▷ input index $\in \{1, 2, \dots, N\}$

  $z_{\text{idx}} = 1$ ▷ evaluation grid index $\in \{1, 2, \dots, M_k\}$

  while ($x_{\text{idx}} \leq N$) **do**

    if ($x_{k,\phi_k(x_{\text{idx}})} \leq z_{k,z_{\text{idx}}}$) **then**

      $\text{INDEX}[k, \phi_k(x_{\text{idx}})] = z_{\text{idx}}$

      $x_{\text{idx}} \mathrel{+}= 1$

    else

      $z_{\text{idx}} \mathrel{+}= 1$

    end

  end

end

$s_{j_1,j_2,\dots,j_d} = 0$, $\forall (j_1,j_2,\dots,j_d) \in \{1,2,\dots,M_1+1\} \times \dots \times \{1,2,\dots,M_d+1\}$

for ($i = 1, 2, \dots, N$) **do**

  $s_{\text{INDEX}[1,i], \text{INDEX}[2,i], \dots, \text{INDEX}[d,i]} \mathrel{+}= y_i / N$

end

Output: $s_{j_1,j_2,\dots,j_d} = \frac{1}{N} \sum_{i=1}^{N} y_i \mathbb{1}\{z_{1,j_1-1} < x_{1,i} \leq z_{1,j_1}, \dots, z_{d,j_d-1} < x_{d,i} \leq z_{d,j_d}\}$
for every local sum index $(j_1,j_2,\dots,j_d) \in \{1,2,\dots,M_1+1\} \times \dots \times \{1,2,\dots,M_d+1\}$

Algorithm 6 has a $\mathcal{O}(N \log N)$ computational complexity, owing to the data sorting in each dimension. Its memory complexity is $\mathcal{O}(N + M)$.

Remark A.1. An alternative algorithm to compute the same local sums has been proposed in Bouchard & Warin (2012). It is based on partial sorts in each dimension and its computational complexity is $\mathcal{O}((\sum_{k=1}^{d} M_k + 1)N)$. This complexity is better than $\mathcal{O}(N \log N)$ when $M \ll \log(N)^d$. However, in the case when $M \approx N$ (and $M_1 = M_2 = \dots = M_d$), its equivalent $\mathcal{O}(N^{1+\frac{1}{d}})$ complexity does not improve over Algorithm 6.

A.2 Uniform grid case

Algorithm 7: Fast computation of local sums by mesh division on uniform grid

Input: input sample $x_i = (x_{1,i}, \dots, x_{d,i})$, $i = 1, 2, \dots, N$
Input: evaluation grid $(z_{1,j_1}, z_{2,j_2}, \dots, z_{d,j_d})$, $j_k \in \{1, 2, \dots, M_k\}$, $k \in \{1, 2, \dots, d\}$
Define index matrix INDEX$[k, i]$ ▷ local sum index $\in \{1, 2, M_k + 1\}$
where $k = 1, 2, \dots, d$ and $i = 1, 2, \dots, N$
for ($k = 1, 2, \dots, d$) **do**
  $\Delta z_k \triangleq z_{k,2} - z_{k,1}$ ▷ constant mesh $= z_{k,3} - z_{k,2} = z_{k,4} - z_{k,3} = \dots$
  for ($i = 1, 2, \dots, N$) **do**
    ▷ mesh division rounded to upper integer
    INDEX$[k, i] = \max(1, \min(M_k + 1, 1 + \lceil (x_{k,i} - z_{k,1})/\Delta z_k \rceil))$
  end
end
$s_{j_1, j_2, \dots, j_d} = 0$, $\forall (j_1, j_2, \dots, j_d) \in \{1, 2, \dots, M_1+1\} \times \dots \times \{1, 2, \dots, M_d+1\}$
for ($i = 1, 2, \dots, N$) **do**
  $s_{\text{INDEX}[1,i], \text{INDEX}[2,i], \dots, \text{INDEX}[d,i]} \mathrel{+}= y_i / N$
end
Output: $s_{j_1, j_2, \dots, j_d} = \frac{1}{N} \sum_{i=1}^{N} y_i \mathbb{1}\{z_{1,j_1-1} < x_{1,i} \leq z_{1,j_1}, \dots, z_{d,j_d-1} < x_{d,i} \leq z_{d,j_d}\}$
for every local sum index $(j_1, j_2, \dots, j_d) \in \{1, 2, \dots, M_1+1\} \times \dots \times \{1, 2, \dots, M_d+1\}$

Algorithm 7 has a $\mathcal{O}(N)$ computational complexity, and $\mathcal{O}(N + M)$ memory complexity.

B General matrix bandwidth

The general multivariate weighted Parzen-Rosenblatt kernel density estimator is defined by:

$$\hat{f}_{\text{KDE}}(z) = \frac{1}{|H|^{1/2} N} \sum_{i=1}^{N} w_i K_d\Big(H^{-1/2}(x_i - z)\Big) \tag{27}$$

where H is a symmetric positive definite $d \times d$ bandwidth matrix, see for example Wand & Jones (1995). As pointed out in Langrené & Warin (2019), one can without loss of generality focus on the diagonal bandwidth case $H = \text{diag}(h)$, where $h = (h_1, h_2, \dots, h_d) \in \mathbb{R}^d$. Indeed, the eigenvalue decomposition of the symmetric positive definite matrix H is given by $H = R\Delta^2 R^\top$ where R is a rotation matrix and $\Delta = \text{diag}(h)$ is a diagonal matrix with strictly positive diagonal elements. Consequently, $H^{-1/2}(x_i - z) = \text{diag}(\frac{1}{h})(R^T x_i - R^T z)$. By rotating both the input points x_i and the evaluation point z using the rotation matrix R^T, the multivariate kernel density estimator (27) becomes

$$\hat{f}_{\text{KDE}}(z) = \frac{1}{N \Pi_{k=1}^{d} h_k} \sum_{i=1}^{N} w_i K_d\left(\frac{x_i - z}{h}\right) \tag{28}$$

where x_i and z denote respectively the input points and evaluation point in the new coordinates. In the Laplacian kernel case, equation (28) turns into the multivariate KDE equation (11) used in Section 3.

C Differentiable Sargan kernels

The generalization of kernel $K^{(1)}$ (21) and kernel $K^{(2)}$ (22) to a polynomial term of order p is given by

$$K^{(p)}(u) = \frac{c}{2(p+1)} \left(\sum_{l=0}^{p} \frac{c^l}{l!} |u|^l \right) e^{-c|u|} \tag{29}$$

This subclass of differentiable Sargan kernels was also identified in Hofmeyr (2020) (with $c = 1$). The 0th polynomial order Sargan kernel corresponds to the classical Laplacian kernel (7), the 1st polynomial order Sargan kernel corresponds to equation (21), and the 2nd polynomial order Sargan kernel corresponds to equation (22). All these kernels are compatible with CDF decomposition.

In terms of regularity, $K^{(p)}$ is a $\mathcal{C}^{p+1}$ kernel. It is such that $\frac{d^l K}{du^l}(0) = 0$, $l = 1, 2, \ldots, p$ and, with the choice $c = p + 1$ (such that $K^{(p)}(0) = 1/2$), approximates the uniform kernel for large p, as shown on Figure 7 below. In other words, equation (29) can be interpreted as an interpolation between the Laplacian kernel ($p = 0$) and the uniform kernel ($p \to \infty$).

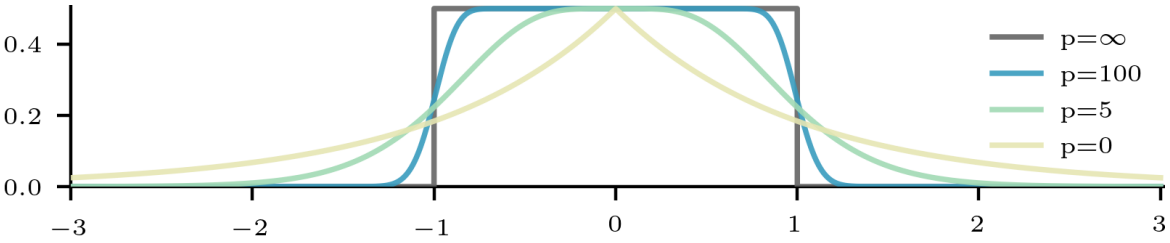

Figure 7: Sargan kernels of polynomial order p

D Multivariate Sargan kernel

There exists different ways to extend the differentiable Sargan kernel (29) to the multivariate setting. Taking the Sargan kernel (29) with $p = 1$ and $c = 1$ as example, one approach, known as product kernel, is to multiply univariate kernels:

$$K_d(u) = \frac{1}{4^d} \prod_{k=1}^{d} (1 + |u_k|)\, e^{-|u_k|} \tag{30}$$

Another approach is to replace the absolute value $|u|$ by the L1 norm $|u| = \sum_{k=1}^{d} |u_k|$, along with a correction of the normalization constant:

$$K_d(u) = \frac{1}{2^d(1+d)} \left(1 + \sum_{k=1}^{d} |u_k|\right) e^{-\sum_{k=1}^{d} |u_k|} \tag{31}$$

The product approach (30) preserves the continuous differentiability of the kernel, which is not the case for the additive approach (31). Nevertheless, the CDF decomposition of the additive kernel (31) contains significantly fewer terms than the one of the product kernel (30). Indeed, the KDE decomposition of (31) is given by

$$
\begin{aligned}
&\frac{1}{N\prod_{k=1}^d h_k}\sum_{i=1}^N w_i K\left(\frac{x_i - z}{h}\right)\\
=&\frac{1}{2^d(1+d)}\frac{1}{N\prod_{k=1}^d h_k}\sum_{i=1}^N w_i\left(1+\sum_{l=1}^d\left|\frac{x_{l,i}-z_l}{h_l}\right|\right)e^{-\sum_{k=1}^d\left|\frac{x_{k,i}-z_k}{h_k}\right|}\\
=&\frac{1}{2^d(1+d)}\frac{1}{N\prod_{k=1}^d h_k}\sum_{i=1}^N w_i\left(1+\sum_{l=1}^d\left|\frac{x_{l,i}-z_l}{h_l}\right|\right)\\
&\times\prod_{k=1}^d\left(e^{-\frac{z_k}{h_k}}e^{\frac{x_{k,i}}{h_k}}\mathbb{1}\{x_{k,i}\le z_k\}+e^{\frac{z_k}{h_k}}e^{-\frac{x_{k,i}}{h_k}}\mathbb{1}\{-x_{k,i}<-z_k\}\right)\Big)\\
=&\frac{1}{2^d(1+d)}\frac{1}{N\prod_{k=1}^d h_k}\sum_{i=1}^N\sum_{\delta\in\{-1,1\}^d}e^{-\sum_{k=1}^d\frac{\delta_k z_k}{h_k}}\left(1+\sum_{l=1}^d\left|\frac{x_{l,i}-z_l}{h_l}\right|\right)\\
&\times w_i e^{\sum_{k=1}^d\frac{\delta_k x_{k,i}}{h_k}}\mathbb{1}\{\delta_1 x_{1,i}\le_{\delta_1}\delta_1 z_1,\dots,\delta_d x_{d,i}\le_{\delta_d}\delta_d z_d\}\Big)\\
=&\frac{1}{2^d(1+d)}\frac{1}{N\prod_{k=1}^d h_k}\sum_{i=1}^N\sum_{\delta\in\{-1,1\}^d}e^{-\sum_{k=1}^d\frac{\delta_k z_k}{h_k}}\left(1-\sum_{l=1}^d\delta_l\frac{x_{l,i}-z_l}{h_l}\right)\\
&\times w_i e^{\sum_{k=1}^d\frac{\delta_k x_{k,i}}{h_k}}\mathbb{1}\{\delta_1 x_{1,i}\le_{\delta_1}\delta_1 z_1,\dots,\delta_d x_{d,i}\le_{\delta_d}\delta_d z_d\}\Big)\\
=&\frac{1}{2^d(1+d)}\frac{1}{\prod_{k=1}^d h_k}\sum_{\delta\in\{-1,1\}^d}e^{-\sum_{k=1}^d\frac{\delta_k z_k}{h_k}}\left(\left(1+\sum_{l=1}^d\frac{\delta_l z_l}{h_l}\right)F_N(\delta z,\delta;\delta x,y^{(0)})-\sum_{l=1}^d\frac{\delta_l}{h_l}F_N(\delta z,\delta;\delta x,y^{(l)})\right)
\end{aligned}
\tag{32}
$$

with $y_i^{(0)} = y_i^{(0)}(\delta) \triangleq w_i e^{\sum_{k=1}^d\frac{\delta_k x_{k,i}}{h_k}}$ and $y_i^{(l)} = y_i^{(l)}(\delta) \triangleq w_i x_{l,i} e^{\sum_{k=1}^d\frac{\delta_k x_{k,i}}{h_k}}$.

This decomposition contains $2^d(d+1)$ CDFs to compute. By contrast, similar computations show that the KDE decomposition of the product kernel (30) contains a total of 4^d CDFs to compute. In other words, the additive kernel (31) is much more attractive than the product kernel (30) from a computational point of view, even when accounting for its lower efficiency. These two kernels are however not as computationally attractive as the Laplacian and uniform kernels, whose CDF decompositions contain 2^d terms ((12) and (18)).

E Divide-and-conquer for Laplacian kernel density estimation

This Appendix explains how to adapt the divide-and-conquer algorithm described in Section 2.2 to compute the 2^d CDF vectors $\{F_N(x_i,\delta)\}_{i=1,N}$ required to compute equation (12). A possible approach would consist in adapting the algorithm 2 used to calculate (3) with $\delta = (1,..,1)$ by applying a modified version 2^d times to calculate the different terms.

We propose a single algorithm, implemented in the StOpt library, which makes it possible to compute the F_N for all the δ in one recursion, avoiding to sort the particles 2^d times.

We give the algorithm obtained in general dimension to calculate for all $\delta \in \{-1,1\}^d$, and given m, l with values in $1,\dots,d$, $(p,q) \in \mathbb{N}^2$ a general term for $j = 1, N$

$$
\sum_{i=1}^N x_{l,i}^p x_{m,i}^q e^{\sum_{k=1}^d\frac{\delta_k x_{k,i}}{h_k}}\mathbb{1}\{\delta_1 x_{1,i} < \delta_1 x_{1,j},\dots,\delta_d x_{d,i} < \delta_d x_{d,j}\}.
$$

Once again observe that the inequalities are strict in the expression above. As before the tests for non-empty sets are dropped out.

- Algorithm 8 is the main calling similar to 2. The special 2D case is dealt with the call of the two different one-dimensional merge algorithm 11 and 12 instead of a single one-dimensional algorithm.

- The n-dimensional merge algorithm 10 is similar to Algorithm 4. Besides, a set Δ of $\delta \in \{-1,1\}^d$ is given as input too such that either $\delta_k x_k \leq \delta_k y_k$ for $k > I_{dim}$ for all $x \in \kappa_1$ and $y \in \kappa_2$ or $\delta_k x_k > \delta_k y_k$ for $k > I_{dim}$ for all $x \in \kappa_1$ and $y \in \kappa_2$.
  For the couple of sets $(\kappa_{1,1}, \kappa_{2,2})$,$(\kappa_{2,1}, \kappa_{1,2})$ where dominance is clear in the current dimension, the n-dimensional merge algorithm is called in the dimension below and some subset of Δ. In the case when $I_{dim} = 2$, a direct call to the one-dimensional merge algorithms 11 and 12 is performed.

- Two one-dimensional merge in dimension 1 are used. The first version **Merge1D1** is used for the δ such that $\delta_1 = 1$ and is the same as the **Merge1D** algorithm except that it works for a set of δ given as input. The second one is for the δ such that $\delta_1 = -1$.

Algorithm 8: Calculate for $1 \leq l \leq m \leq d$, p, q given

$$F(x_j, \delta) = \sum_{i=1}^{N} x_{l,i}^p x_{m,i}^q e^{\sum_{k=1}^{d} \frac{\delta_k x_{k,i}}{h_k}} \mathbb{1}\{\delta_1 x_{1,i} < \delta_1 x_{1,j}, \ldots, \delta_d x_{d,i} < \delta_d x_{d,j}\}, \quad j = 1, N, \delta \in \{-1,1\}^d$$

Input: $x = (x_i, \ldots, x_N)$, $\psi(x_i, \delta) = x_{l,i}^p x_{m,i}^q e^{-\sum_{k=1}^{d} \frac{\delta_k x_{k,i}}{h_k}}$, for all $i = 1, \ldots, N$, $\delta \in \{-1,1\}^d$
Calculate ϕ^j , $j = 1, \ldots, d$ such that $x_{j,\phi^j(1)} \leq x_{j,\phi^j(2)} \leq \cdots \leq x_{j,\phi^j(N)}$
$F(x_i, \delta) = 0$ for $i = 1, \ldots, N$, for all $\delta \in \{-1,1\}^d$
RecurSplitting(x, ψ, ϕ, F, N)
Output: $F(x_i, \delta)$ for all $i \in [1, N]$ and all $\delta \in \{-1,1\}^d$

Algorithm 9: Recursive splitting function **RecurSplitting**

Input: x, ψ, F, $\phi^j(i)$ for $i = 1, M$, $j = 1, d$
$\kappa_1 = \{\phi^d(i), i = 1, \frac{M}{2}\}$, ϕ_1 with values in κ_1 s.t. $x_{j,\phi_1^j(1)} \leq x_{j,\phi_1^j(2)} \leq \cdots \leq x_{j,\phi_1^j(\frac{M}{2})}$, $j = 1, d$
$\kappa_2 = \{\phi^d(i), i = \frac{M}{2} + 1, M\}$, ϕ_2 in κ_2 s.t. $x_{j,\phi_2^j(1)} \leq x_{j,\phi_2^j(2)} \leq \cdots \leq x_{j,\phi_2^j(\frac{M}{2})}$,$j = 1, d$
RecurSplitting$(x, \psi, \phi_1, F, M/2)$
RecurSplitting$(x, \psi, \phi_2, F, M/2)$
if ($d > 2$) **then**
  $\Delta = \{\delta \in \{-1,1\}^d\}$
  MergeND$(x, \phi_1, \phi_2, d-1, \psi, F, M/2, M/2, \Delta)$
else
  ▷ Merge for all δ
  Merge1D1$(x, \phi_1^1, \phi_2^1, \psi, F, \hat{\Delta}, M/2, M/2)$, with $\hat{\Delta} = \{(1,1)\}$
  Merge1D2$(x, \phi_1^1, \phi_2^1, \psi, F, \hat{\Delta}, M/2, M/2)$ with $\hat{\Delta} = \{(-1,1)\}$
  Merge1D1$(x, \phi_2^1, \phi_1^1, \psi, F, \hat{\Delta}, M/2, M/2)$ with $\hat{\Delta} = \{(1,-1)\}$
  Merge1D2$(x, \phi_2^1, \phi_1^1, \psi, F, \hat{\Delta}, M/2, M/2)$ with $\hat{\Delta} = \{(-1,-1)\}$
end
Output: F updated

Algorithm 10: Recursive merge nD **MergeND** in given dimension I_{dim}

Input: x, ψ, F, $\Delta \subset \{\delta \in \{-1,1\}^d\}$, $\phi_1^j(i)$, for all $i = 1, M_1$, $\phi_2^j(i)$, for all $i = 1, M_2$ with values in $[1, N]$ for $j = 1, I_{dim}$

▷ $\delta \in \Delta$ s.t. $\delta_k x_k \leq \delta_k y_k$ for $k > I_{dim}$ for all $x \in \kappa_1$ and $y \in \kappa_2$ or $\delta_k x_k > \delta_k y_k$ for $k > I_{dim}$ for all $x \in \kappa_1$ and $y \in \kappa_2$

$\kappa_1 = \{\phi_1^{I_{dim}}(i), i = 1, M_1\}$, $\kappa_2 = \{\phi_2^{I_{dim}}(i), i = 1, M_2\}$

$\kappa = \kappa_1 \cup \kappa_2$, x_{med} s.t; $\#\{x_j, j \in \kappa, x_{I_{dim},j} \leq x_{med}\} = \#\{x_j, j \in \kappa, x_{I_{dim},j} > x_{med}\}$

$\kappa_{l,1} = \{i \in \kappa_l, x_{I_{dim},i} \leq x_{med}\}$, $M_{l,1} = \#\kappa_{l,1}$, for $l = 1, 2$,

$\kappa_{l,2} = \{i \in \kappa_l, x_{I_{dim},i} > x_{med}\}$, $M_{l,2} = \#\kappa_{l,2}$, for $l = 1, 2$

Create $\phi_{l,m}^j(i)$, $i = 1, \dots M_{l,m}$ s.t. $\phi_{l,m}^j(i) \in \kappa_{l,m}$, and

$$x_{j,\phi_{l,m}^j(1)} \leq x_{j,\phi_{l,m}^j(2)} \leq \cdots \leq x_{j,\phi_{l,m}^j(M_{l,m})}, \text{ for } j \leq I_{dim}, \quad l = 1, 2, \quad m = 1, 2.$$

MergeND$(x, \phi_{1,l}, \phi_{2,l}, I_{dim}, \psi, F, M_{1,l}, M_{2,l}, \Delta)$, for $l = 1, 2$

if ($I_{dim} == 2$) **then**

▷ Merge the set of 3D problem directly without recursion

Merge1D1$(x, \phi_{1,1}^1, \phi_{2,2}^1, \psi, F, \hat{\Delta}, M_{1,1}, M_{2,2})$ so that $\hat{\Delta} = \{(1, 1, 1, ..) \in \Delta\}$

Merge1D2$(x, \phi_{1,1}^1, \phi_{2,2}^1, \psi, F, \hat{\Delta}, M_{1,1}, M_{2,2})$ so that $\hat{\Delta} = \{(-1, 1, 1, \dots) \in \Delta\}$

Merge1D1$(x, \phi_{2,2}^1, \phi_{1,1}^1, \psi, F, \hat{\Delta}, M_{2,2}, M_{1,1})$ so that $\hat{\Delta} = \{(1, -1, -1, \dots) \in \Delta\}$

Merge1D2$(x, \phi_{2,2}^1, \phi_{1,1}^1, \psi, F, \hat{\Delta}, M_{2,2}, M_{1,1})$ so that $\hat{\Delta} = \{(-1, -1, -1, \dots) \in \Delta\}$

Merge1D1$(x, \phi_{1,2}^1, \phi_{2,1}^1, \psi, F, \hat{\Delta}, M_{1,2}, M_{2,1})$ so that $\hat{\Delta} = \{(1, -1, 1, \dots) \in \Delta\}$

Merge1D2$(x, \phi_{1,2}^1, \phi_{2,1}^1, \psi, F, \hat{\Delta}, M_{1,2}, M_{2,1})$ so that $\hat{\Delta} = \{(-1, -1, 1, \dots) \in \Delta\}$

Merge1D1$(x, \phi_{2,1}^1, \phi_{1,2}^1, \psi, F, \hat{\Delta}, M_{2,1}, M_{1,2})$ so that $\hat{\Delta} = \{(1, 1, -1, \dots) \in \Delta\}$

Merge1D2$(x, \phi_{2,1}^1, \phi_{1,2}^1, \psi, F, \hat{\Delta}, M_{2,1}, M_{1,2})$ so that $\hat{\Delta} = \{-1, 1, -1, \dots\} \in \Delta\}$

else

▷ Merge in dimension below

mergedND$(x, \phi_{1,1}, \phi_{2,2}, I_{dim} - 1, \psi, F, M_{1,l}, M_{2,l}, \hat{\Delta})$, $\hat{\Delta} = \{\delta \in \Delta \text{ with } , \delta_{I_{dim}} \delta_{I_{dim}+1} > 0\}$

mergedND$(x, \phi_{2,1}, \phi_{1,2}, I_{dim} - 1, \psi, F, M_{1,l}, M_{2,l}, \hat{\Delta})$, $\hat{\Delta} = \{\delta \in \Delta \text{ with } , \delta_{I_{dim}} \delta_{I_{dim}+1} < 0\}$

end

Output: F updated

Algorithm 11: Final merge function in dimension one : **Merge1D1** for two sets of points $\kappa_1 = \{x_{\phi_1(i)}, i = 1, M_1\}$, $\kappa_2 = \{x_{\phi_2(i)}, i = 1, M_2\}$ such that for $x \in \kappa_1$, $y \in \kappa_2$, $\delta_k x_k \leq \delta_k y_k$ for $k \in [2, d]$, for all $\delta \in \Delta$. All elements δ of Δ are such that $\delta_1 = 1$.

Input: x, ψ, F, ϕ_k s.t. $\phi_k(i) \leq \phi_k(i+1)$, for all $i = 1, M_k - 1$, $k = 1, 2$, $\Delta \subset \{\delta \in \{-1, 1\}^d\}$
$S(\delta) = 0$ for all $\delta \in \Delta$, $j = 0$
for ($i = 1, M_2$) **do**
  while ($(x_{\phi_2(i),1} \geq x_{\phi_1(j),1})$ *and* $j \leq M_1$) **do**
    $S(\delta) += \psi(\phi_1(j), \delta)$ for all $\delta \in \Delta$, $j = j + 1$
  end
  $F(\phi_2(i), \delta) += S(\delta)$ for all $\delta \in \Delta$
  if ($j == M_1 + 1$) **then**
    for ($k = i + 1, M_2$) **do**
      $F(\phi_2(k), \delta) += S(\delta)$ for all $\delta \in \Delta$
    end
    $i = M_2 + 1$
  end
end
Output: F updated

Algorithm 12: Final merge function in dimension one : **Merge1D2** for two sets of points $\kappa_1 = \{x_{\phi_1(i)}, i = 1, M_1\}$, $\kappa_2 = \{x_{\phi_2(i)}, i = 1, M_2\}$ such that for $x \in \kappa_1$, $y \in \kappa_2$, $\delta_k x_k \leq \delta_k y_k$ for $k \in [2, d]$, for all $\delta \in \Delta$. All elements δ of Δ are such that $\delta_1 = -1$.

Input: x, ψ, F, ϕ_k s.t. $\phi_k(i) \leq \phi_k(i+1)$, for all $i = 1, M_k - 1$, $k = 1, 2$, $\Delta \subset \{\delta \in \{-1, 1\}^d\}$
$S(\delta) = 0$ for all $\delta \in \Delta$, $j = M_1$
for ($i = M_2, 1$) **do**
  while ($(x_{\phi_2(i),1} < x_{\phi_1(j),1})$ *and* $j \geq 1$) **do**
    $S(\delta) += \psi(\phi_1(j), \delta)$ for all $\delta \in \Delta$, $j = j - 1$
  end
  $F(\phi_2(i), \delta) += S(\delta)$ for all $\delta \in \Delta$
  if ($j == 0$) **then**
    for ($k = 1, i - 1$) **do**
      $F(\phi_2(k), \delta) += S(\delta)$ for all $\delta \in \Delta$
    end
    $i = 0$
  end
end
Output: F updated