MEPDF: Multivariate empirical density functions

by

Martin Wiegand and Saralees Nadarajah [1]

School of Mathematics, University of Manchester, Manchester M13 9PL, UK

Abstract: We introduce a new R package and its functions written by the authors. This package computes an empirical density function for arbitrary dimensions with adjustable grid sizes.

Keywords: Empirical density function, Grid size, Multivariate data

1 Introduction

The empirical probability density function (EPDF) or more commonly referred to as histogram, is one of the simplest tools available to estimate the density of any given data set, yet remains one of the most reliable instruments for statisticians. In virtually any field of statistics the EPDF is used to verify a new representation of a density via random sampling, or to get a first attempt at the true distribution of the data at hand.

A number of different approximations for the empirical density function have existed for years (Bentley,1980), along with efficient implementations. For example, Duong (2017) provided a kernel density estimator for higher dimensional data with the ks package.

Despite the widespread use of approximations, implementations were available for limited dimensions only (Warnes et al., 2016; Eklund, 2017; Sievert, 2018). However, many problems are not limited in dimensionality, and require a flexible approach which can be used regardless of the number of variables. Therefore, we have developed an R package (R Development Core Team, 2017) which offers a density function for data sets of arbitrary dimension. The package is named MEPDF (Wiegand and Nadarajah, 2017).

In Section 2, we elaborate on the algorithm for computing the EPDF. Thereafter, we provide an overview of the functionality of the package and provide a number of examples for different possible configurations. In Section 4, we compare our method with previously existing ones, in terms of runtimes and accuracy, and discuss benefits and disadvantages of the methods. Section 5 discusses three dimensional examples. We close this note with conclusions on what we have accomplished.

2 Method

Let us assume we have a data set of dimension n and sample size N. The first step is to determine what the domain of the density function will look like. This means we assume an n dimensional hyperrectangle defined by a minimum and a maximum corner point:

$$R = \left\{ \mathbf{x} \in \mathcal{R} \,\middle|\, x_i^{min} \leq x_i \leq x_i^{max}, \forall i = 1, \ldots, n \right\}.$$

[1]Corresponding author, email: mbbsssn2@manchester.ac.uk

This domain is then subdivided into cells of dimensions $\mathbf{g} = (g_1, \ldots, g_n) \in \mathcal{R}^+$, so that we have a number of $\left(x_i^{max} - x_i^{min}\right)/g_i$ cells making up side i of the domain. Each of these cells can be once again defined by an upper and lower end point, p_j^{min} and p_j^{max} for all $j = 1, \ldots, g_1 g_2 \cdots g_n$. For any $\mathbf{x} \in \mathcal{R}^n$, we define $p^{min}(\mathbf{x})$ and $p^{max}(\mathbf{x})$ to be the corners of the smallest cell containing $\mathbf{x}$. We can then define the cell counts as follows:

$$c(\mathbf{x}) = \#\left\{\mathbf{y}\left| p_j^{min}(x)_i \leq y_i \leq p_j^{max}(x)_i, \forall i = 1, \ldots, n\right.\right\}.$$

With this number we can scale for the sample size and cell dimension, and are left with the density estimator:

$$\widehat{f}(\mathbf{x}) = \frac{c(\mathbf{x})}{n\left\{\prod_{i=1}^{n}\left[p^{max}(x)_i - p^{min}(x)_i\right]\right\}}.$$

Since the method is rather simplistic (essentially the multivariate generalisation of the well-known histogram), and heavily dependent on the distribution of the sample at hand, there may be grid entries for which there is no value observed (for example, no sample happens to fall within the respective grid), thus returning an estimate of $\mathbb{P}(x) = 0$. To offer a smoothing option to the users of this package, we introduce a technique borrowing heavily from kernel density estimators, yet modify the principle to suit our estimation approach. Essentially, we pick any given non-zero grid value, and distribute the value across adjacent cells. If $d \in \mathbb{N}$ is the dimension of the data set, p the initial estimate and $\#r \in \mathbb{N}$ the number of "spheres" around the central grid cell, then we can express the neighbouring values as follows:

$$p_r = \frac{\frac{p}{\#r}}{(2r+1)^d - (2r-1)^d}.$$

This means we divide the initial value evenly across each additional layer, and then across each cell within each layer around a central grid cell. This is then repeated for all non-zero grid cells, and the results are saved in a new grid of equal size. The function can be called via `pseudokernel`, which has the same input parameters as the standard grid method, with the optional parameter `rings`, indicating the amount of layers around each cell. This approach is akin to the averaged shifted histograms (ASH) introduced by Scott (1992) of which implementations are available for up to two dimensions (Scott, 2015). Once again we offer a multivariate version of a similar approach, to make the method more accessible to problems of arbitrary dimensions.

3 Implementation

As with the univariate case, the multivariate EPDF rests on the organization of the data domain into sections and counting the contained data sample fraction. In higher dimensions these sections become cells of the respective dimension as described in Section 2. Each function therefore begins with setting up a grid, based on user specifications.

```
R> data <- mvrnorm(100000, mu = c(0, 0), Sigma = diag(2))
R> density <- epdf(data = data,
+         min.corner = c(-4, -4),
+          max.corner = c(4, 4),
+          main.gridsize = c(0.05, 0.05))
R> data2 <- exp(data)
R> density2 <- epdf(data = data2,
+          min.corner = c(0, 0),
+          max.corner = c(0.25, 0.25),
+          main.gridsize = c(0.025, 0.025))
```

In the code above, we describe the general usage of a single grid EPDF. The examples given use bivariate, normally distributed and log-normally distributed data.

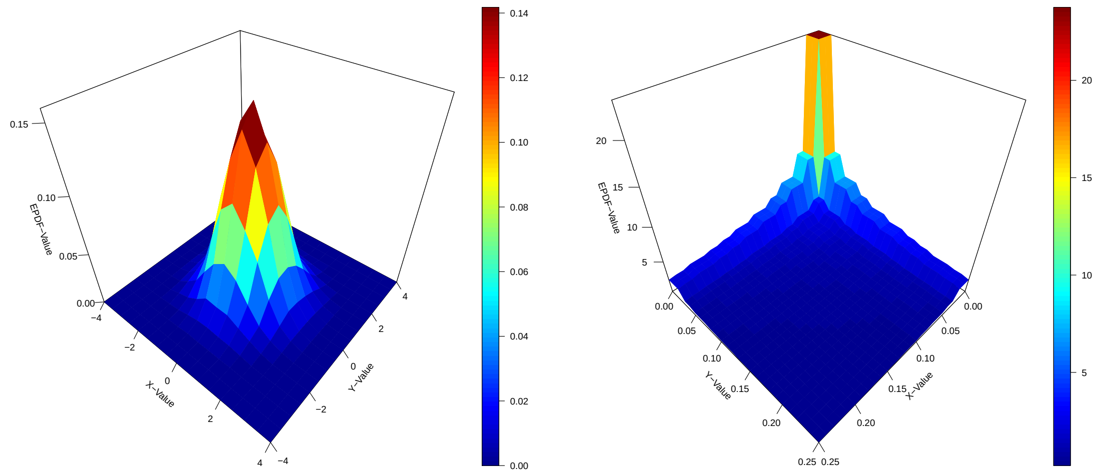

Figure 1: Single grid EPDFs: normal distribution (left) and log-normal distribution (right).

The code produces the EPDF density with a cropped domain between the points $(-4, -4)$ and $(4, 4)$ as well as $(0, 0)$ and $(0.25, 0.25)$. The cells are defined via the argument main.gridsize, giving the cell dimension, which are set to be squares with side lengths 0.05 and 0.025. The visualisation of the output in a three dimensional surface is shown in Figure 1.

For arbitrary data samples, not all regions of the plot require the same level of cell resolution. We have therefore added an option to superimpose regions of greater or lesser accuracy on top of one another. The EPDF function can therefore be called with an additional argument rescubes. This is a list of lower and upper corners of the additional grid as well as the respective grid sizes. Even though we have used cells of same side length in this example, rectangular cells are possible by specifying different side lengths.

In the code below, we describe how to call a grid with multiple resolutions on top of one another. The data set is once again a multivariate normally distributed one.

```
R> a <- list(mn = c(-1, -1),
+          mx = c(1, 1),
+          grid.size = c(0.05, 0.05))
R> b <- list(mn = c(-2, -2),
+         mx = c(2, 2),
+         grid.size = c(0.1, 0.1))
R> cubes <- list(a, b)
R>  pdf <- epdf(data = data,
+  max.corner = c(4, 4),
+  min.corner = c(-4, -4),
+  main.gridsize = c(0.2, 0.2),
+  rescubes = cubes)
```

While the main grid has the coarse resolution of 1×1, we add two more grids on top of the existing one, see Figure 2. The first grid stretches from $(-2, -2)$ to $(2, 2)$ with resolution 0.1×0.1 and the second from $(-1, -1)$ to $(1, 1)$ with resolution 0.05×0.05.

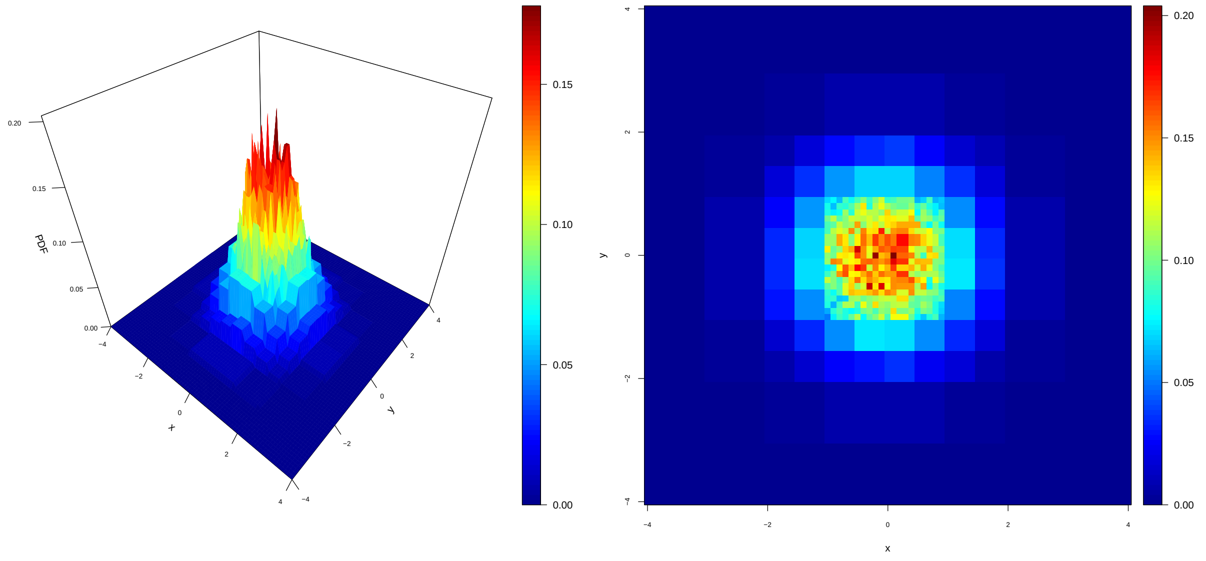

Figure 2: Normal distribution with two additional grids.

Notice that hyper rectangles, if added later to the argument, stand higher in the evaluation hierarchy. Thus if two additional grids overlap, the one to further down in the rescubes argument will be evaluated.

4 Comparison to other approximations

Lee and Joe (2017) introduced efficient sorting algorithms for bivariate and trivariate data samples, to compute the cumulative distribution function (CDF) at each sample point. We implemented the bivariate version of the modified sorting algorithm, and have tested the runtime and accuracy. We adapted the grid approach in Section 2 to compute the CDF, to retrieve comparable results (van der Vaart, 1998; Coles, 2001; Madsen et al., 2006).

We generated random samples of a bivariate standard normal distribution for a number of different grid sizes (with 20 repetitions for each sample size, and grid size). To compare the results we have computed the approximated CDF at the sample point, both via the quicksort algorithm and the grid method. The deviation is computed via the mean absolute error, as well as the error average error measured at every grid node (MGE):

$$\text{MAE} = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\left|f\left(x_{(i,j)}\right) - \hat{f}\left(x_{(i,j)}\right)\right|,$$
$$\text{MGE} = \frac{1}{N}\sum_{i=1}^{N}\left|f\left(x_i\right) - \hat{f}\left(x_i\right)\right|.$$

The points x_i for $i + 1, \ldots, N$ correspond to the sample points, whereas $x_{(i,j)}$ denote the grid points between $[-2.5, -2.5]$ and $[2.5, 2.5]$ equidistantly distributed for $i = 1, \ldots, n$ and $m = 1, \ldots, m$. Due to the square area of interest and the grid shape in this case $n = m$ holds. Table 1 provides a collection of the outcomes. While we did anticipate the grid algorithm to be inherently more dependent on the grid size, and potentially slower than the more refined and optimized quicksort algorithm, we call to mind one of the main benefits of the grid. The set up of the grid and computation of each value takes up more time than just sorting the data frame, but the evaluation for a single point is almost instant once the matrix of values has been set up. To make the package more user-friendly we have added Scott's optimal bandwidth approximation (Scott, 1992) as the default grid size rule:

$$h_k^* = 2 \cdot 3^{1/(2+d}\pi^{d/(4+2d)}\widehat{\sigma}_k n^{-1/(2+d)}. \tag{1}$$

For data of dimension d and sample size n as well as the empirical variance of the kth variable $\widehat{\sigma_k}^2$, the optimal grid size along the kth dimension can be described as in (1).

	Sample size	Grid size	Setup time	Ev. time	MAE	MGE
Grid method	100	0.2	0.0000	0.0000	0.05494155	0.01363199
		0.1	0.0000	0.0000	0.02559191	0.01062833
		0.05	0.0000	0.0000	0.01630517	0.00814666
		0.01	0.0700	0.0100	0.03091841	0.01841441
	1000	0.2	0.0400	0.0300	0.02400647	0.00923922
		0.1	0.0300	0.0300	0.02103825	0.00587147
		0.05	0.0400	0.0300	0.01104718	0.00489833
		0.01			0.01021250	0.00250230
	10000	0.2	0.3300	0.3000	0.02837931	0.01012616
		0.1	0.4120	0.3460	0.01358187	0.00587147
		0.05	0.3710	0.3060	0.00677197	0.00312645
		0.01	0.5340	0.6080	0.00245682	0.00119964
	100000	0.2	3.4210	2.9190	0.02872715	0.00974854
		0.1	3.4110	3.1930	0.01381017	0.00511228
		0.05	3.5360	3.1770	0.00868906	0.00312839
		0.01	4.5940	3.6480	0.00146289	0.00093547
			Computation time			
QS method	100	-	0.0197		0.04008379	-
	1000	-	0.0686		0.01158748	-
	10000	-	1.1554		0.00606940	-
	100000	-	43.3694		0.00488396	-

Table 1: Runtime and error measure comparison for different methods and sample sizes.

Contrary to our anticipation, the grid method we have proposed performs mostly on par with, or better than the quicksort estimation. This is heavily dependent on the chosen gridsize, as large cells might squander the amount information provided by the sample, yet small cells might distort the true function we are estimating. Thus the results for the grid method are better than the quicksort algorithms performance, yet only when choosing the adequate cell size for the sample size at hand. The difference in computation time between the quicksort algorithm and the grid method seems to be marginal for most of the smaller sample sizes, as both computational efforts seem to increase linearly. However, for the highest sample size of $n = 100000$ the grid method performs very clearly faster, leading us to believe that the computational effort for the quicksort method exceeds linearity.

Another observation is that the mean grid error is considerably lower than the mean error at the sample points. We attribute this to the fact that the normal distribution does not change much outside the 3σ area around the mean value. Therefore, the CDF values will not change much from 0 or 1, depending on which corner of the observed area we are on.

More importantly, the grid gives opportunity to evaluate the CDF for arbitrary values, whereas sorting algorithms by nature can only make a statement on values at the individual sample points. The grid can therefore to some degree be used to interpolate between values, or to extrapolate beyond the samples, depending on the choice of grid sizes.

As the grid has to be set up only once, and the evaluation at arbitrary points is sufficiently fast, we believe this algorithm still has its place as benchmark or initial estimator when analyzing the distribution of data samples.

Lastly, we highlight the differences between the standard grid method, and the pseudo-kernel estimator we provide in this package. Due to the smoothing properties of this approach, the function can be particularly helpful for small sample sizes, which would otherwise give an incomplete picture of the underlying distribution.

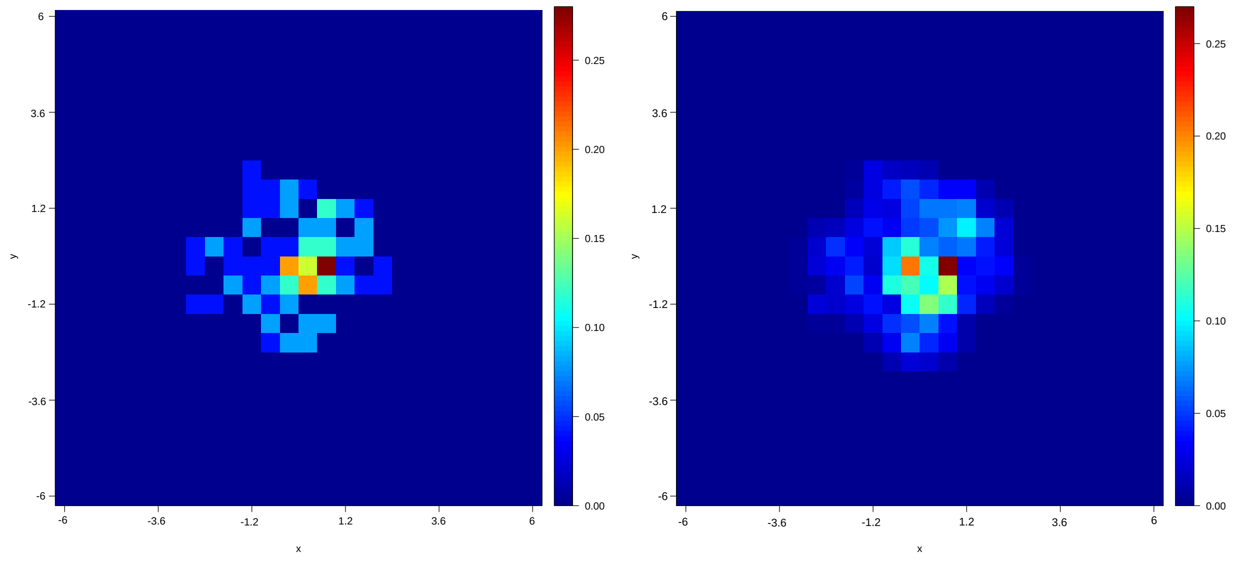

Figure 3: A two dimensional comparison between the standard, and pseudo-kernel options.

To visualise the differences between the grid method with and without the additional distribution of the cell values, we have provided a standard example with the two dimensional normal distribution of the comparatively low sample size $n = 100$. In Figure 3, we see a two dimensional colour plot of the results for both estimators side-by-side, as well as a three dimensional version of the same results in Figure 4. The functions are a simple result of calling the `pseudokernel` package.

```
R> data <- mvrnorm(n = 100, mean = rep(0,2), sigma = diag(2))
R> est <- pseudokernel(data = data, mn = c(-6,-6),mx = c(6,6),
+                      grid.sizes = c(0.5,0.5), rings = 1)
R> image2D(est$grid1)
R> image2D(est$grid2)
```

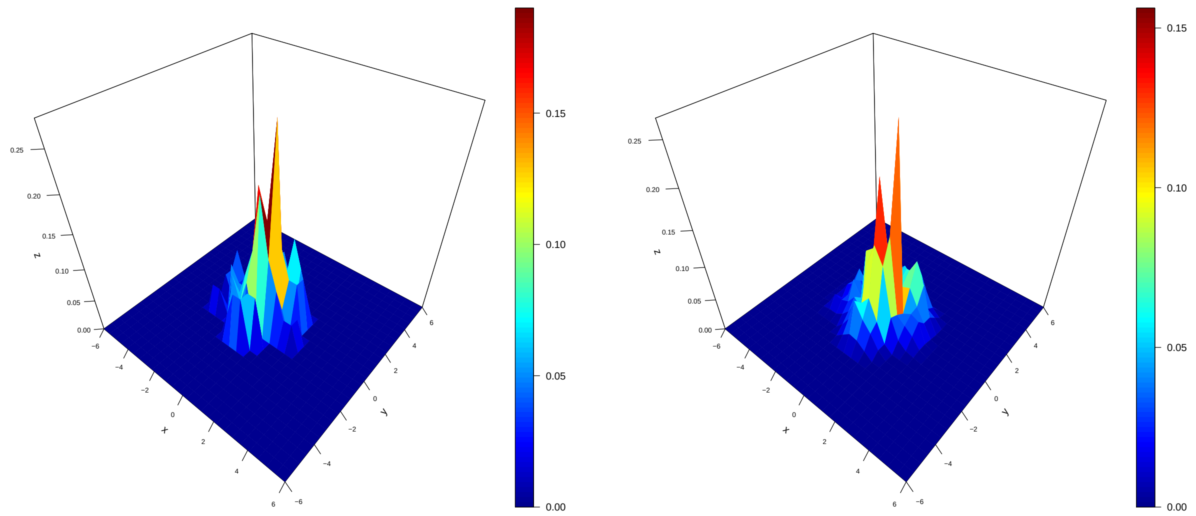

Figure 4: A three dimensional comparison between the standard, and pseudo-kernel options.

As we can clearly see, the addition of layers around the non-zero cells provides a closer approximation to the true distribution. Especially around the outer edges, we get a smooth gradient instead of the previous abrupt decline to zero values in the standard grid. Furthermore, we can detect zero-value grid points in the two dimensional plots, where the sample at hand did not happen to have any observations. This quite obviously stands in contrast to the real distribution, thus mandating a corrective measure. With a single additional distribution layer, these gaps can be corrected as we see in the second picture of Figure 3.

5 Three dimensional examples

To highlight the versatility of the proposed estimation, we introduce some example applications of three dimensions in this section. In Figure 5, we can see the empirical density for a three dimensional normal distribution on the left hand side, with a cube size of 0.2. Additionally, we like to emphasize the practical applications of the estimation technique. Therefore we have tested the empirical density on a real life data set. In the `MASS` library, we find the `gilgais` data set, listing parameters of soil properties in New South Wales, Australia.

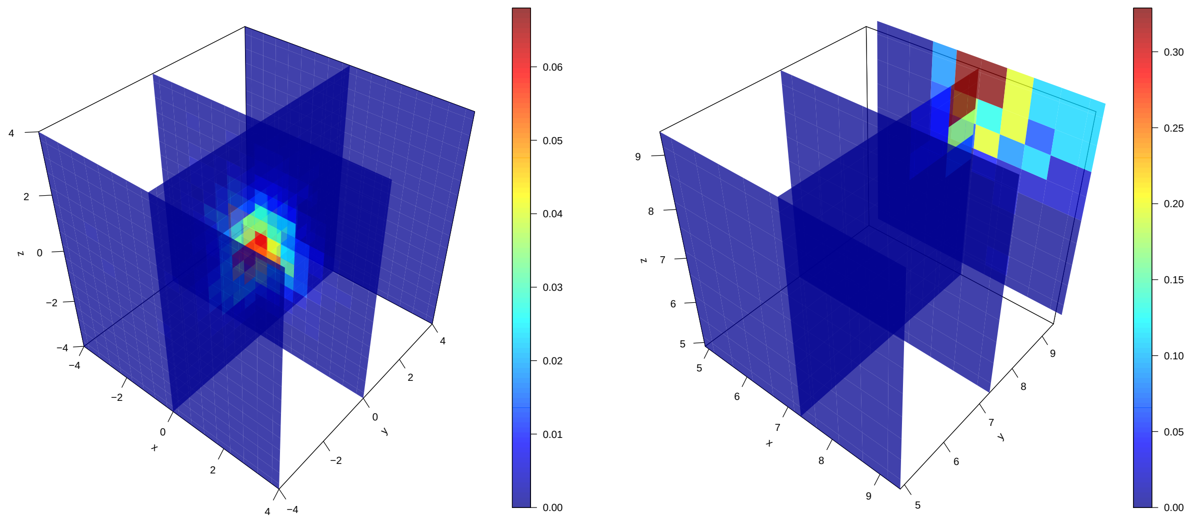

Figure 5: Crosssection of the three dimensional empirical PDF for a sampled three dimensional normal sample and data from the `gilgais` stock data set.

In Figure 5 on the right hand side, we have chosen the pH level of the soil in depths of 0-10cm, 30-40cm and 80-90cm. The data provides three dimensions for the empirical density example.

Lastly, for the three dimensional examples, we wish to give the reader an impression of the computational effort and performance behaviour of the grid method in higher dimensions. We have therefore repeated the experiment for both CDF and PDF of the three dimensional multivariate distribution in Table 2, and have recorded the mean absolute error of the estimation, along with the time required for the setup of the grid, and the evaluation for all the sample values. We see much of the same behaviour in terms of grid size and sample size, where we see the performance changing across grid sizes, if the mesh is chosen to fine. However, we can clearly see that the computation time remains not impacted by the grid size, as well as linear in relation to the input sample size.

Sample size	Grid size	Setup time	Ev. time	MAE (PDF)	MAE (CDF)
1000	1	0.017	0.010	0.006853	0.142104
	0.5	0.019	0.010	0.011401	0.059951
	0.2	0.021	0.011	0.034676	0.032020
	0.1	0.020	0.009	0.038892	0.008663
10000	1	0.222	0.101	0.006039	0.134859
	0.5	0.185	0.095	0.004254	0.057359
	0.2	0.175	0.106	0.012245	0.020445
	0.1	0.182	0.089	0.032079	0.008012
100000	1	1.969	1.104	0.006011	0.130826
	0.5	1.718	0.955	0.003181	0.060196
	0.2	2.301	0.926	0.004022	0.021299
	0.1	1.901	0.932	0.010923	0.009881
1000000	1	19.640	10.801	0.005981	0.131438
	0.5	20.532	10.252	0.003036	0.059838
	0.2	19.604	10.469	0.001708	0.022123
	0.1	19.111	10.487	0.003593	0.010352

Table 2: Error values and computation times for the grid method in three dimensions.

6 Conclusions

With the MEPDF (Wiegand and Nadarajah, 2017) package, which can be found on the CRAN repository, we have provided an implementation for empirical density functions of arbitrary dimension. This gives a standardized tool with all necessary functions to work with higher dimensional data sets.

In Section 4, we have compared the grid algorithm to recent optimized sorting algorithms in terms of runtimes and accuracy. The results and wide range of applications for the grid algorithm have led us to believe that this very simple algorithm remains relevant, providing a fast and simple estimate with the right implementation.

While the quicksort algorithm serves as an example of a more sophisticated algorithm, it is by no means as versatile. Due to the nature of the quicksort algorithm, it can only give evaluations of the empirical CDF at the sample sizes, not allowing for results between the samples. Additionally, the algorithm needs readjusting, or even a completely new conceptualisation for higher dimensions. The grid approach on the other hand can easily be transferred to arbitrary dimensions, or be modified to compute CDFs instead of probabilities.

The grid method can be very easily parallelised as every cell is independent (in fact we will add the option to the package as a built-in feature), whereas the quick sort algorithm is intrinsically sequential (every new sorting decision is dependent on the previous one) and cannot be parallelised. This offers a decisive advantage to the grid method on a technical level.

Acknowledgments

The authors would like to thank the Editor and the referee for careful reading and comments which greatly improved the paper.

References

Bentley, J. 1980. Multidimensional divide and conquer. *Communications of the ACM* 23 : 214-229.

Coles, S. 2001. *An introduction to statistical modeling of extreme values.* London: Springer Verlag.

Duong, T. 2017. ks: Kernel smoothing. URL https://cran.r-project.org/web/packages/ks/index.html. R package version 1.10.7.

Eklund, A. C. 2017. squash: Color-based plots for multivariate visualization. R package version 1.0.8

Lee, D. and H. Joe. 2017. Efficient computation of multivariate empirical distribution functions at the observed values. *Computational Statistics*, doi: 10.1007/s00180-017-0771-x.

Madsen, H., S. Krenk, and S. Lind. 2006. *Methods of structural safety.* New York: Dover Publications.

R Development Core Team. 2017. *R: A language and environment for statistical computing.* Vienna, Austria: R Foundation for Statistical Computing.

Scott, D. W. 1992. *Multivariate density estimation: Theory, practice and visualisation.* New York: John Wiley and Sons.

Scott, D.W. 2015. ash: David Scott's ASH routines. https://cran.r-project.org/web/packages/ash/index.html. R package version 1.0-15

Sievert, C. 2018. plotly for R. R package version 4.8.0

van der Vaart, A. 1998. *Asymptotic statistics.* Cambridge: Cambridge University Press.

Warnes, G. R., B. Bolker, L. Bonebakker, R. Gentleman, W. H. A. Liaw, T. Lumley, M. Maechler, A. Magnusson, S. Moeller, M. Schwartz and B. Venables. 2016. gplots: Various R programming tools for plotting data. R package version 3.0.1

Wiegand, M. and S. Nadarajah. 2017. MEPDF: Multivariate empirical density function. URL https://cran.r-project.org/web/packages/MEPDF/index.html. R package version 3.0.