MEPDF

DOI:
10.1080/03610918.2018.1554118

Document Version
Accepted author manuscript

Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

MEPDF: An R Package for Multivariate Empirical Density Functions

Martin Wiegand
University of Manchester

Saralees Nadarajah*
University of Manchester

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 distribution function, Grid size, Multivariate data.

1. Introduction

The empirical probability density function (EPDF) 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, Lee and Joe (2017) provided a kernel density estimator for higher dimensional data with the **ks** package.

Despite the wide spread use of approximations, implementations were only available for univariate data so far. Therefore, we have developed an R (R Development Core Team 2017) package 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 the 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

*Corresponding author, email: mbbsssn2@manchester.ac.uk

the methods. We close this note with a conclusion 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 maximum corner point:

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

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 $g_i / \left(x_i^{max} - x_i^{min}\right)$ 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 which contains $\mathbf{x}$. We can then define the cell counts as follows:

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

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

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

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.

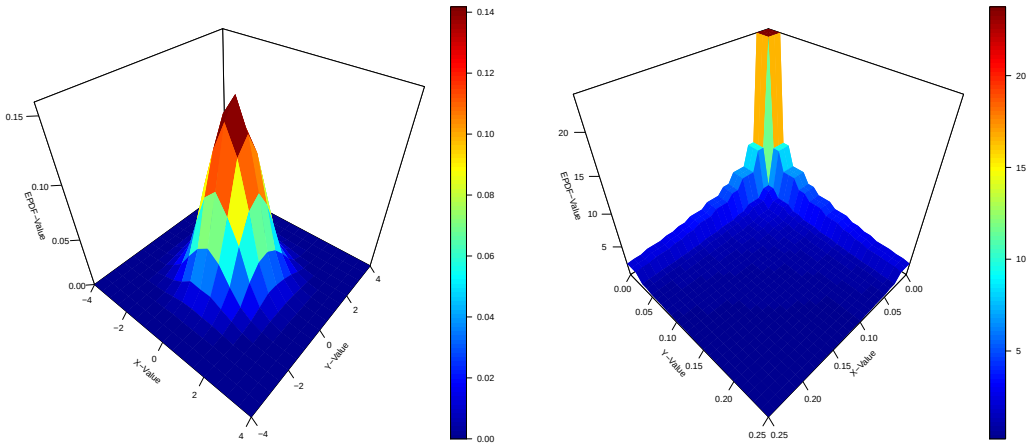

Figure 1: Single grid EPDFs: normal distribution (left) and log-normal distribution (right).

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 the 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 0.2×0.2 and the second from $(-1, -1)$ to $(1, 1)$ with resolution 0.1×0.1.

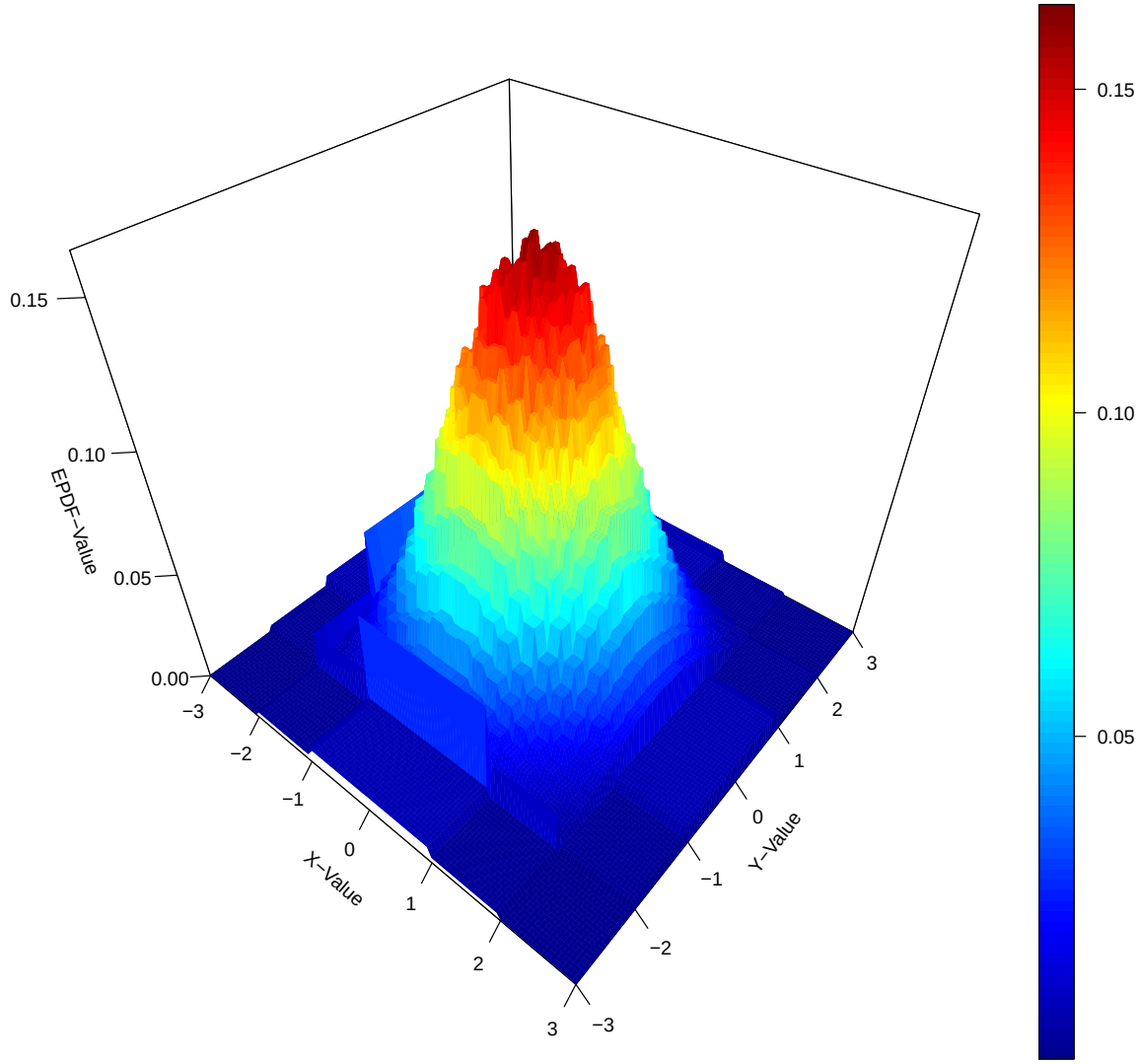

Figure 2: Normal distribution with two additional grids.

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 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 cumulative distribution function, to retrieve comparable results (Coles 2001; Madsen, Krenk, and Lind 2006; van der Vaart 1998).

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). In addition to the quick sort algorithm's performance and the evaluation of the grid at the sample values, we added the mean absolute error of the grid from $(-2.5, -2.5)$ to $(2.5, 2.5)$ in increments of 0.05 in both directions.

Table 1 provides a collection of the outcomes. While we did anticipate the grid algorithm to be inherently 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.

		Sample	Runtime	Mean abs. error	Mean grid error
Grid method	Cell size 0.2	100	0.21392936	0.04566793	0.04251113
		1000	0.24834236	0.02810882	0.01867514
		10000	1.6141590	0.02877145	0.0195883
		100000	206.25750	0.02762748	0.01884067
	Cell size 0.1	100	0.25150332	0.04788994	0.02828868
		1000	0.27418801	0.01752053	0.00931264
		10000	1.67261842	0.01419393	0.0058551
		100000	213.125000	0.01092680	0.0039695
	Cell size 0.05	100	0.24326998	0.04423368	0.02820412
		1000	0.27710048	0.01122147	0.00767438
		10000	1.63767783	0.00647778	0.00233407
		100000	209.730000	0.00646367	0.00092244
	Cell Size 0.01	100	0.25234084	0.04305427	0.0239167
		1000	0.31065388	0.01056779	0.00695079
		10000	1.70771075	0.00317742	0.00199478
		100000	236.791069	0.00159879	0.00067679
QS method		100	0.01965387	0.04008379	-
		1000	0.06860905	0.01158748	-
		10000	1.15542641	0.00606940	-
		100000	43.3694060	0.00488396	-

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

As expected, the quick sort algorithm performs faster for most sample sizes (especially smaller

ones, $n \leq 10000$), taking up roughly between a quarter and half the computation time of the grid. With increasing sample size, the factor between both implementations seems to be getting smaller.

Another observation we can make is that the mean grid error is considerably lower than the mean error at the sample points, and even below the error rate of the quicksort algorithm. With growing sample size, we observe a convergence of the mean grid error and mean error of the sorting algorithm. With the smallest cell size, namely squares of dimension 0.01×0.01, we are even able to achieve a smaller error rate at the sample points as well (about 30% smaller).

More importantly, the grid gives opportunity to evaluate the cumulative distribution function 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.

5. Conclusions

With the **MEPDF** (Wiegand and Nadarajah 2017) package, we have provided an implementation for empirical density functions of arbitrary dimension, which can be found on the **CRAN** repository. 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 still remains relevant.

References

Bentley J (1980). "Multidimensional Divide and Conquer." *Communications of the ACM*, **23**, 214–229.

Coles L (2001). *An Introduction to Statistical Modeling of Extreme Values*. Springer Verlag, London.

Lee D, Joe H (2017). "Efficient Computation of Multivariate Empirical Distribution Functions at the Observed Values." *Computational Statistics*. Doi: 10.1007/s00180-017-0771-x.

Madsen H, Krenk S, Lind S (2006). *Methods of Structural Safety*. Dover Publications, New York.

R Development Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

van der Vaart A (1998). *Asymptotic Statistics*. Cambridge University Press, Cambridge.

Wiegand M, Nadarajah S (2017). ***MEPDF**: Multivariate Empirical Density Function*. R package version 1.0, URL https://cran.r-project.org/web/packages/MEPDF/index.html.

Affiliation:

Martin Wiegand
School of Mathematics
University of Manchester
Manchester M13 9PL, UK
E-mail: martin.wiegand@manchester.ac.uk

Saralees Nadarajah
School of Mathematics
University of Manchester
Manchester M13 9PL, UK
E-mail: mbbsssn2@manchester.ac.uk

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd