Multivariate ECDF Based Models (Rough Draft)

Charlotte Maia

December 9, 2009

Abstract

This vignette introduces the R package, mecdf, an experimental package for generalising the so-called "empirical distribution function" or "empirical cumulative distribution function" to the multivariate case, or equivalently, for vector random variables. Here we give special attention to the bivariate normal, we were compare the theoretical CDF, against an ECDF modelled using simulated data.

Introduction

We will assume the reader has a basic understanding of both multivariate CDFs, and univariate ECDFs. For those who are not, the pmvnorm function in the mvtnorm package is a good starting point for understanding multivariate distributions (noting we are going to make use of it in this vignette), and the ecdf function in the standard R distribution is a good starting point for understanding univariate ECDFs.

An ECDF is by definition a step function, however the author is interested in producing smooth(ish) functions and is currently considering this problem.

Essentially a cumulative distribution function defines the probably that a random variable takes a value less than or equal to some arbitrary value from the random variable's sample space. So for the univariate case:

$$\mathrm{m}(x) = \mathbb{P}(X \leq x)$$

Noting that I'm deviating from convention, and using $\mathrm{m}(x)$, to denote the CDF, usually it's F.

For the bivariate case, we have two random variables, or equivalently a vector random variable with two components. We are now interested in the probably that the first random variable is less than or equal to some value, and the second random variable is less than or equal to some other value:

$$\mathrm{m}(x_1, x_2) = \mathbb{P}(X_1 \leq x_1, X_2 \leq x_2)$$

Expanding on the point above, the comma inside the probability expression means "and". For p random variables, the expression generalises to:

$$\mathrm{m}(x_1, x_2, \ldots, x_p) = \mathbb{P}(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_p \leq x_p)$$

In practice we often have some data, and we want to compute the CDF (or perhaps more commonly the PDF, however we won't go there...) from the data. Generally, this is done by assuming some parametric distribution, then estimating the parameters of that distribution, that is, the parameters that are most likely, given the data (and more importantly given our assumption of the distribution). An alternative approach for univariate distributions, which is very simple, gives very good estimates for moderately sized datasets, and which has no assumptions per se, is to use an ECDF.

The standard ECDF can be defined as:

$$\hat{\mathrm{m}}(x) = \frac{\sum_{\forall i}(\mathbf{x}_{[i]} \leq x)}{n}$$

In the summation operation, summing over true, is equivalent to summing over one (similar to R). Subscripted-bracketed symbols refer to a realisation of the corresponding random variable, and n refers to the number of realisations.

Not surprising, we can compute a bivariate ECDF as:

$$\hat{\mathrm{m}}(x_1, x_2) = \frac{\sum_{\forall i}(\mathbf{x}_{1[i]} \leq x_1, \mathbf{x}_{2[i]} \leq x_2)}{n}$$

Plus the multivariate ECDF as:

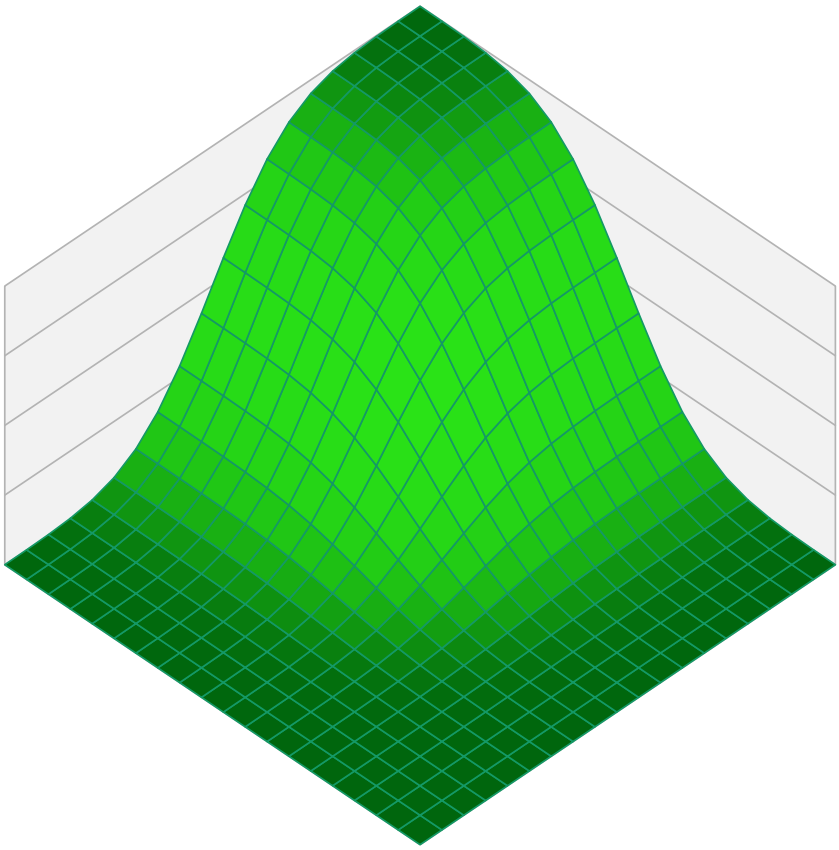
$$\hat{\mathrm{m}}(x_1, x_2, \ldots, x_p) = \frac{\sum_{\forall i}(\mathbf{x}_{1[i]} \leq x_1, \mathbf{x}_{2[i]} \leq x_2, \ldots, \mathbf{x}_{p[i]} \leq x_p)}{n}$$

Examples

Now for some examples. First let us consider the case of a bivariate normal, with no correlation. If there is no correlation then we could treat it as two independent distributions, however lets just treat as a single distribution, to give us a nice starting point.

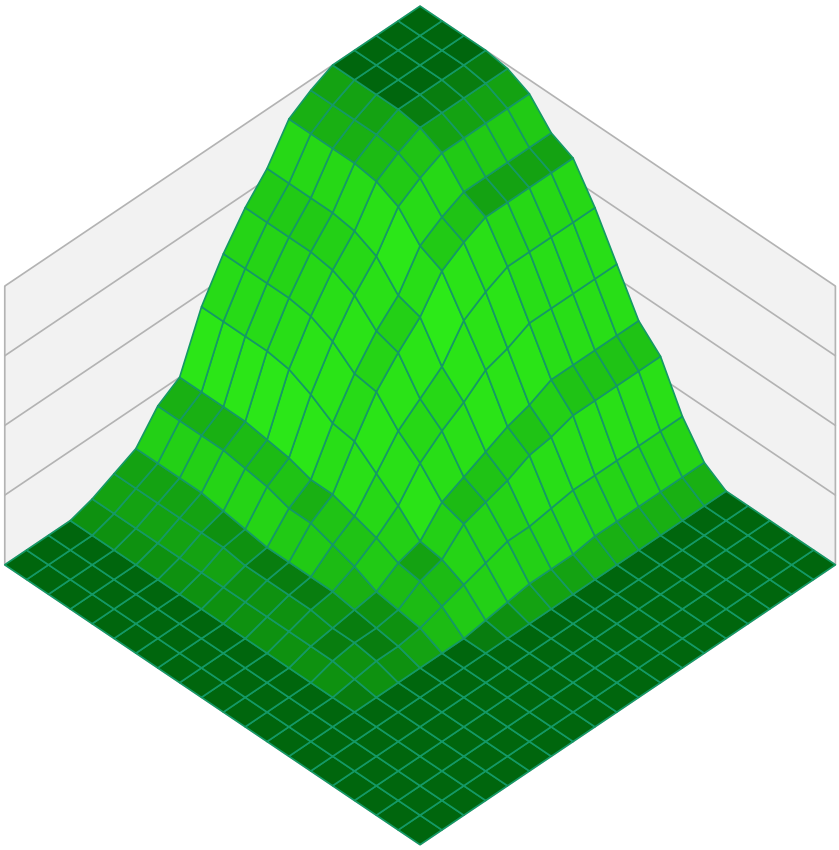
For visual reference, let us look at the theoretical version (with a little help from the mvtnorm package)

```
> #Get warmed up...
> library (mvtnorm, warn=FALSE)
> library (mecdf, warn=FALSE)

> #Create two vectors, representing points on a two-variable sample space
> #We will use this to create a square regularly-spaced grid
> res = 20
> x1 = x2 = seq (-3.5, 3.5, length=res)

> #Create a square matrix to store values of the distribution function
> #Evaluate the distribution function over our grid
> mt1 = matrix (numeric (), nr=res, nc=res)
> for (i in 1:res)
        for (j in 1:res)
                mt1 [i, j] = pmvnorm (c (-Inf, -Inf), c (x1 [i], x2 [j]) )


> #Create a pretty picture
> bcdf.plot (mt1)
```



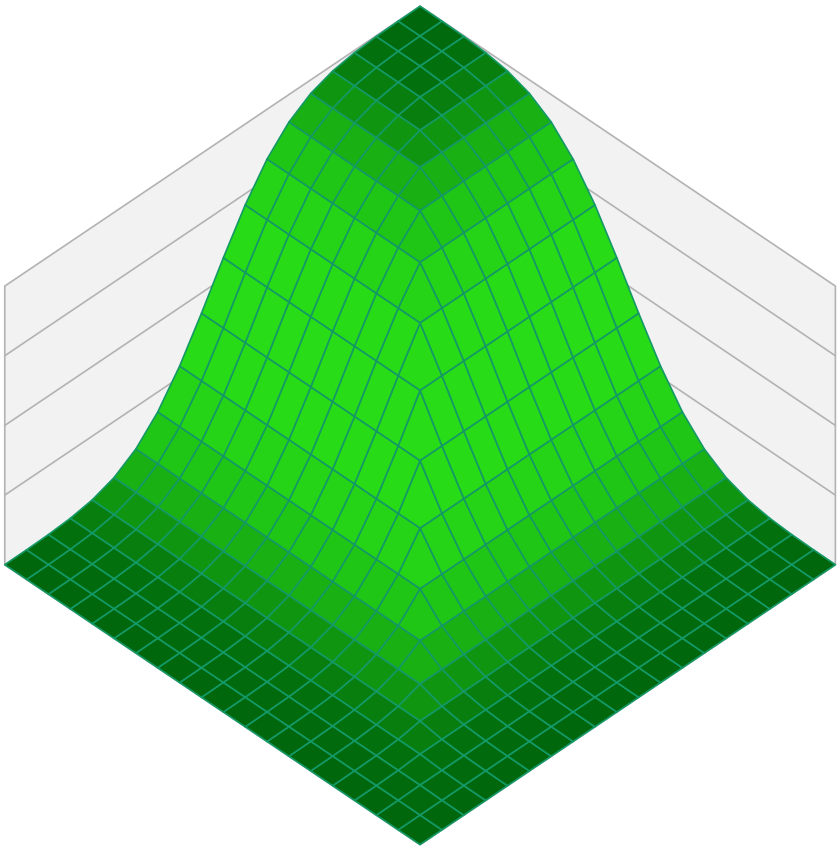
Well, that's kind of nice, in fact it's too nice. Let's create 80 "random variates" also with some help from the mvtnorm package, and then use the mecdf function, from the mecdf package, and see what we get...

```
> #80 of them, no correlation
> x = rmvnorm (80, c (0, 0), matrix (c (1, 0, 0, 1), nr=2) )

> #Create an MECDF model
> mf1 = mecdf (x)

> #Create another square matrix
> #However, using the MECDF for evaluation
> mh1 = matrix (numeric (), nr=res, nc=res)
> for (i in 1:res)
        for (j in 1:res)
                mh1 [i, j] = mf1 (c (x1 [i], x2 [j]) )


> #Create another pretty picture
> #Note that this plot is slightly misleading
> #We have a step function, not a smooth function
> #However, we are evaluating step and smooth functions the same way
> bcdf.plot (mh1)
```

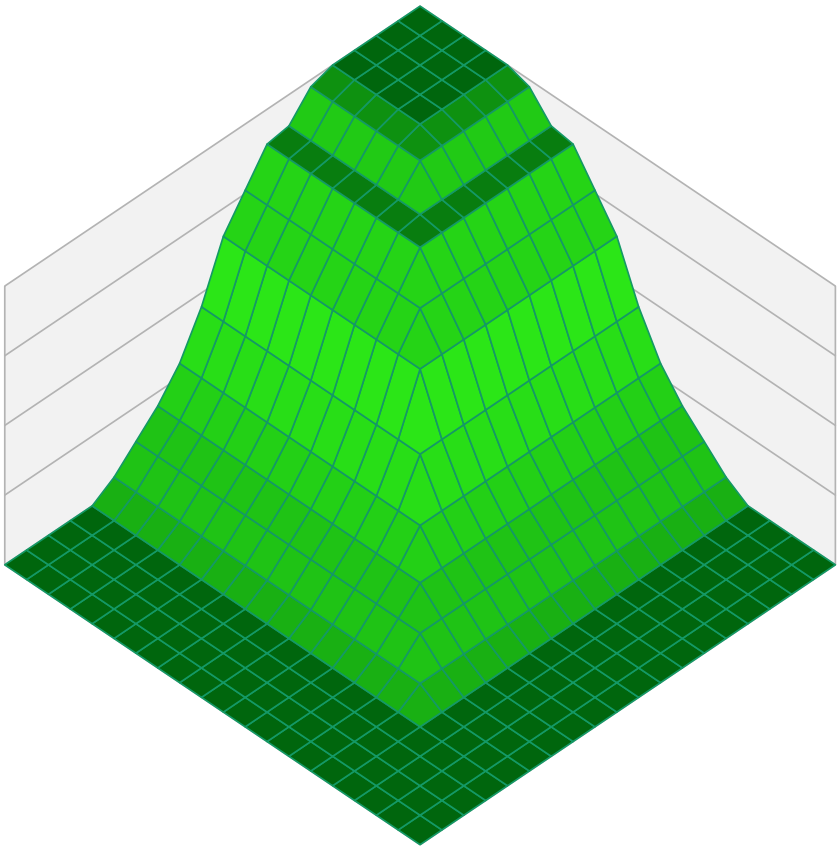


Let's repeat both the theoretical model and the mecdf model, using a correlation coefficient of 1. Just because it's fun...

```
> #The theoretical version
> mt2 = matrix (numeric (), nr=res, nc=res)
> for (i in 1:res)
         for (j in 1:res)
                 mt2 [i, j] = pmvnorm (c (-Inf, -Inf), c (x1 [i], x2 [j]),
                         sigma=matrix (c (1, 1, 1, 1), nr=2) )
> bcdf.plot (mt2)
```

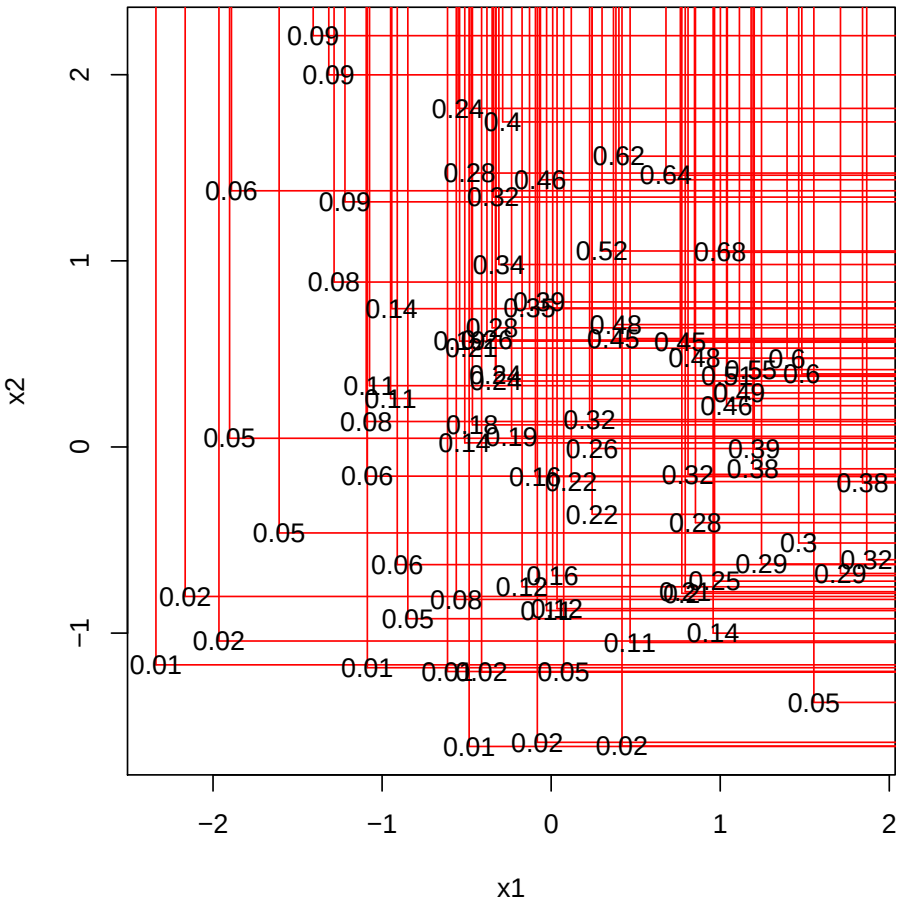


```
> #The MECDF version
> x = rmvnorm (80, c (0, 0), matrix (c (1, 1, 1, 1), nr=2) )
> mf2 = mecdf (x)
> mh2 = matrix (numeric (), nr=res, nc=res)
> for (i in 1:res)
        for (j in 1:res)
                mh2 [i, j] = mf2 (c (x1 [i], x2 [j]) )
> bcdf.plot (mh2)
```

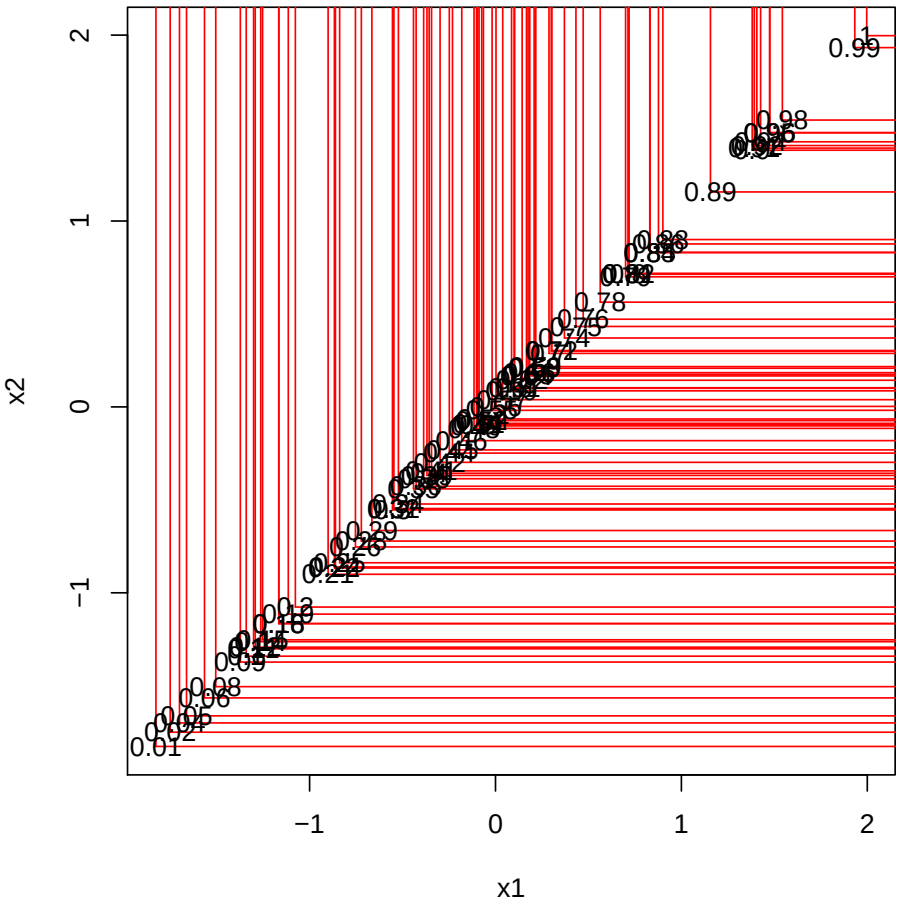


The above plots aren't very good at detecting dependence. Hence the standard plot for a bivariate ECDFs is slightly different. The logical arguments lower and upper draw lines on the plot. Lower is best for understanding how it's computed, however upper gives a better indication of the level of dependence. However, we could just use R's standard plotting commands here...

```
> plot (mf1, upper=TRUE)
```




```
> plot (mf2, upper=TRUE)
```


x2
x1
2
1
0
−1
−1
0
1
2
0.99
0.89
0.01