Multivariate ECDF Based Models (Draft)

Charlotte Maia

January 11, 2010

Abstract

This vignette introduces the mecdf package (and the multivariate ECDF based models that it implements). Such models generalise so-called empirical cumulative distribution functions (ECDFs); firstly to the multivariate case (i.e. for multiple random variables, with a joint distribution); and secondly by allowing both step functions (true ECDFs) and continuous functions (via interpolation). The package also contains special purpose functions for plotting univariate and bivariate ECDFs.

Introduction

We will assume the reader has a basic understanding of both multivariate CDFs and univariate ECDFs. Noting that the mvtnorm package and the standard "ecdf" function in R, are both good references. Essentially, a cumulative distribution function defines the probably that a random variable takes a value less than or equal to some other value from the random variable's sample space. So for the univariate case:

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

For the bivariate case, we now have two random variables. We are interested in the probably that the first random variable takes some value less than or equal to some other value, and that the second random variable also takes some value less than or equal to some other value:

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

These expressions generalise to k random variables:

$$\mathrm{F}_{X_1,X_2,\ldots,X_k}(x_1, x_2, \ldots, x_k) = \mathbb{P}(X_1 \leq x_1, X_2 \leq x_2, \ldots, X_k \leq x_k)$$

Often we are interested in estimating the distribution from some data (most commonly the PDF, however we will focus on the CDF). The standard approach is to make some assumption about the distribution (i.e. that the actual distribution is one of a set of distributions, distinguished from each other by some parameters). The problem then simplifies to one of estimating the parameters only. An alternative approach, where we estimate the distribution directly from the data, what we will denote as $\hat{\mathrm{F}}$, without assumptions and without parameters, is to use an ECDF model. By definition a (univariate) ECDF is a step function. It's the proportion of realised (or "observed") values of a random variable, that are less than or equal to some value. We can write it as:

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

In the expression above, I've replaced x with u, and are using $\mathbf{x}$ (here a vector), to denote the realised values of the random variable X. The function I is an indicator function, which is equal one when the enclosed expression is true, and zero otherwise.

The Multivariate ECDF

Following the above expressions, we can formulate the bivariate and k-variate ECDFs as:

$$\hat{\mathrm{F}}_{X_1,X_2}(u_1,u_2) = \frac{\sum_{\forall i} \mathrm{I}(\mathbf{x}_{[i,1]} \leq u_1, \mathbf{x}_{[i,2]} \leq u_2)}{n}$$

$$\hat{\mathrm{F}}_{X_1,X_2,\ldots,X_k}(u_1,u_2,\ldots,u_k) = \frac{\sum_{\forall i} \mathrm{I}(\mathbf{x}_{[i,1]} \leq u_1, \mathbf{x}_{[i,2]} \leq u_2, \ldots, \mathbf{x}_{[i,k]} \leq u_k)}{n}$$

Note that now $\mathbf{x}$ is a matrix. Each row of the matrix represents one multivariate realisation. Each column represents realisations of one random variable. Let us consider an example of a bivariate ECDF.

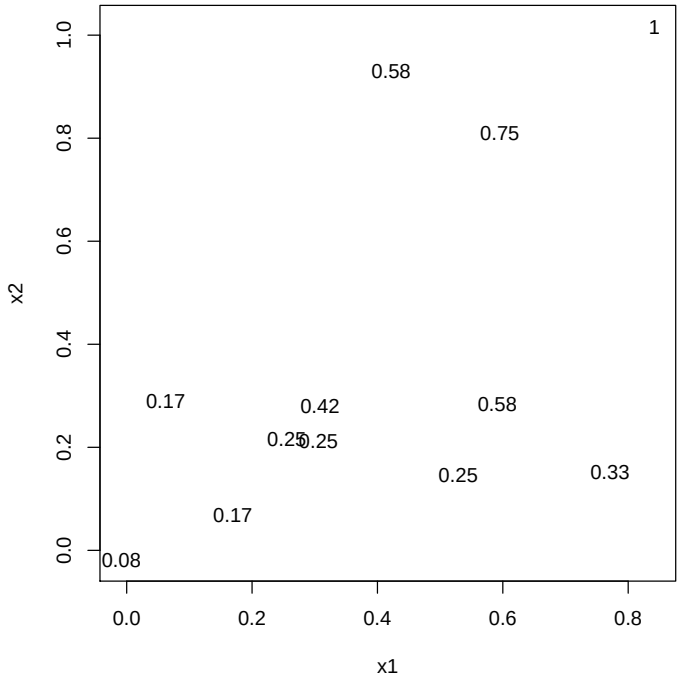
```
> #get warmed up
> library (mecdf, warn=FALSE)


> #simulated data
> x1 = runif (10)
> x2 = runif (10)
> x = cbind (x1, x2)


> #fit the model
> m = mecdf (x)
> m
mecdf: step function
       12 realisations of 2 random variables
```

Note that there are two extra realisations (their kind of fake). We can prevent this from happening, however they are there for a reason. This is discussed later (in the section on model expansion). Our model, is also a regular R function, and we can use it to evaluate the estimated distribution function over some point.

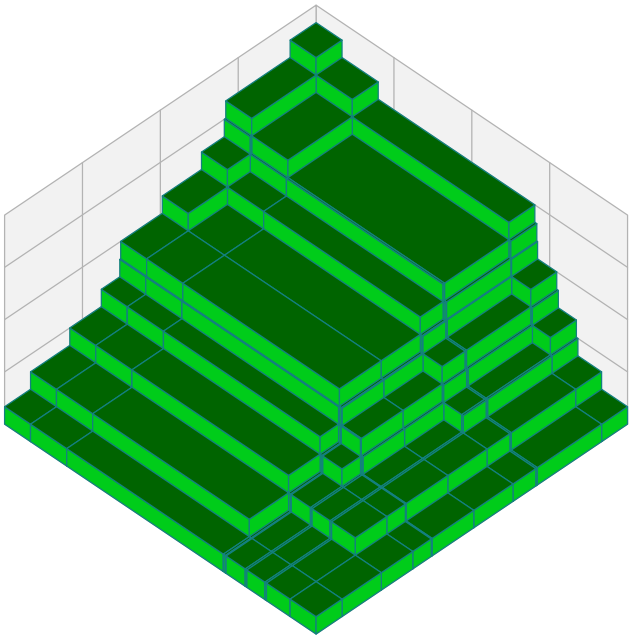
```
> m (c (0.5, 0.5) )
[1] 0.5
```

The mecdf package provides multiple ways to visualise bivariate ECDFs. That standard way, via plot, produces a plot of one variable against the other, with the values of the distribution function shown on the plot. There are also 3D plots using the function plotbcdf. By default, these produce surfaces evaluated over a regularly spaced grid. However, by including the argument simple=FALSE (not recommended for large samples), we can produce a surface over the actual realised values (which form an irregularly spaced grid). For step functions (which is what we have here), our surface is discontinuous. In the next section (with a continuous functions) our surface is continuous.

```
> plot (m)
```




```
> plotbcdf (m, simple=FALSE)
```



The Continuous ECDF

Not only can be generalise ECDFs to the multivariate case, we can also generalise them to the continuous case. In principle, continuous distribution functions should be better estimates than step distribution functions (for continuous random variables). However, estimating the continuous ECDF requires a high

computational cost (often with only a small improvement in the estimate), hence continuous ECDFs are the default for univariate models, however step ECDFs are the default for two or more variables. There is another problem with continuous ECDFs. The method presented here has the potential to produce negative values for the distribution function. The reasons for this and a solution are discussed later (also in the section on model expansion).

We can treat our realised values, as forming an irregularly spaced grid. This allows us to produce a continuous distribution function by linear interpolation over vertices of that grid. This is simple in the univariate case (where a one dimension grid, is quite trivial). For any vertex (here simply any realised value, $u \in \mathbf{x}$), we can compute the value of a distribution function $\mathrm{F}^*_X(u)$ over that vertex. We make a slight modification to our standard expression so that the values of our distribution function are between 0 and 1.

$$\mathrm{F}^*_X(u) = \frac{\sum_{\forall i} \mathrm{I}(\mathbf{x}_{[i]} \leq u) - 1}{n - 1}$$

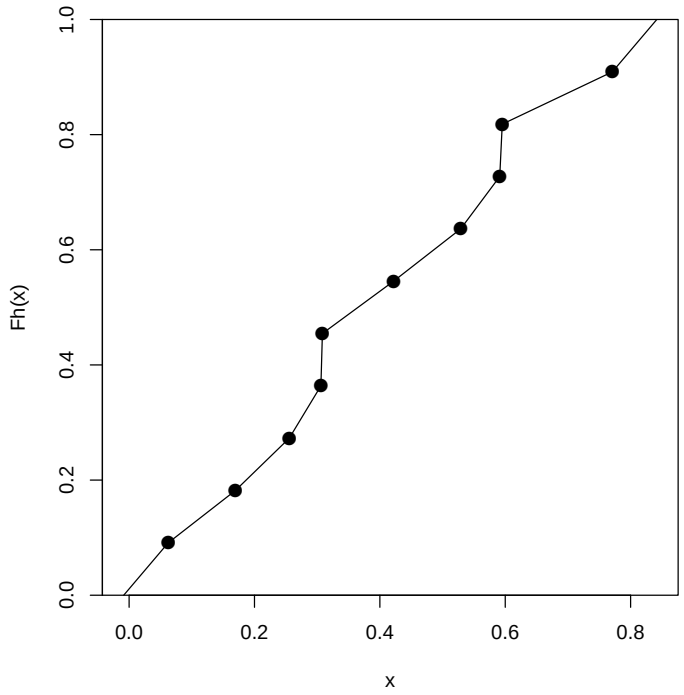
We have four possible cases:

1. If $u \leq \min(\mathbf{x})$, then $\hat{\mathrm{F}}_X = 0$.
2. Else if $u \geq \max(\mathbf{x})$, then $\hat{\mathrm{F}}_X = 1$.
3. Else if $u \in \mathbf{x}$, then $\hat{\mathrm{F}}_X(u) = \mathrm{F}^*_X(u)$
4. Otherwise we interpolate...

$$\hat{\mathrm{F}}_X(u) = q\mathrm{F}^*_X(a) + p\mathrm{F}^*_X(b)$$

Where: a, the greatest realised value less than u; b, the smallest realised value greater than u; p, the distance from a to u as a proportion of the distance between a and b; and q, which equals $1 - p$. Noting that $a < u < b$. Using the vector x1 created earlier:

```
> m = mecdf (x1)
> y = numeric (10)
> for (i in 1:10) y [i] = m (x1 [i])
> plot (m)
> points (x1, y, pch=16, cex=1.5)
```



To move from the univariate case to the bivariate case, we define a function $\mathrm{F}^*_{X_1,X_2}$, however our grid is no longer trivial. We have vertex whenever $u_1 \in \mathbf{x}_{[,1]}$ and $u_2 \in \mathbf{x}_{[,2]}$.

$$\mathrm{F}^*_{X_1,X_2}(u_1,u_2) = \frac{\sum_{\forall i} \mathrm{I}(\mathbf{x}_{[i,1]} \leq u_1, \mathbf{x}_{[i,2]} \leq u_2) - 1}{n-1}$$

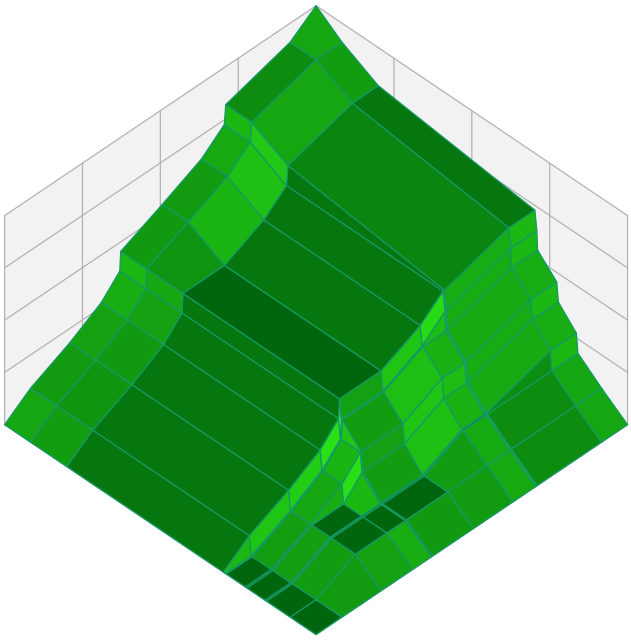
Now we have several cases:

1. If either $u_1 \leq \min(\mathbf{x}[,1])$ or $u_2 \leq \min(\mathbf{x}[,2])$, then $\hat{\mathrm{F}}_{X_1,X_2}(u_1,u_2) = 0$
2. Else if both $u_1 \geq \max(\mathbf{x}[,1])$ and $u_2 \geq \max(\mathbf{x}[,2])$, then $\hat{\mathrm{F}}_{X_1,X_2}(u_1,u_2) = 1$
3. Else if either $u_1 > \max(\mathbf{x}[,1])$ or $u_2 > \max(\mathbf{x}[,2])$, we substitute the corresponding u value with the corresponding maximum value, then start this if-else sequence again...
4. Else if both $u_1 \in \mathbf{x}_{[,1]}$ and $u_2 \in \mathbf{x}_{[,2]}$, then $\hat{\mathrm{F}}_{X_1,X_2}(u_1,u_2) = \mathrm{F}^*_{X_1,X_2}(u_1,u_2)$.
5. Else if one value is (an element) and the other isn't, we interpolate between two vertices (similar to the univariate case, except using two variables).
   e.g. $\hat{\mathrm{F}}_{X_1,X_2}(u_1,u_2) = q\mathrm{F}^*_{X_1,X_2}(u_1,a) + p\mathrm{F}^*_{X_1,X_2}(u_1,b)$.
6. Else (both values between pairs of realisations), then we compute values of a, b, p and q for each variable, and interpolate between four vertices:

$$\hat{\mathrm{F}}_{X_1,X_2}(u_1,u_2) = q_1q_2\mathrm{F}^*_{X_1,X_2}(a_1,a_2) + q_1p_2\mathrm{F}^*_{X_1,X_2}(a_1,b_2) + p_1q_2\mathrm{F}^*_{X_1,X_2}(b_1,a_2) + p_1p_2\mathrm{F}^*_{X_1,X_2}(b_1,b_2)$$

We can generalise this to k variables. For a grid with k variables, there will be up to 2^k vertices. Hence the large computational cost. Using the matrix x, created earlier:

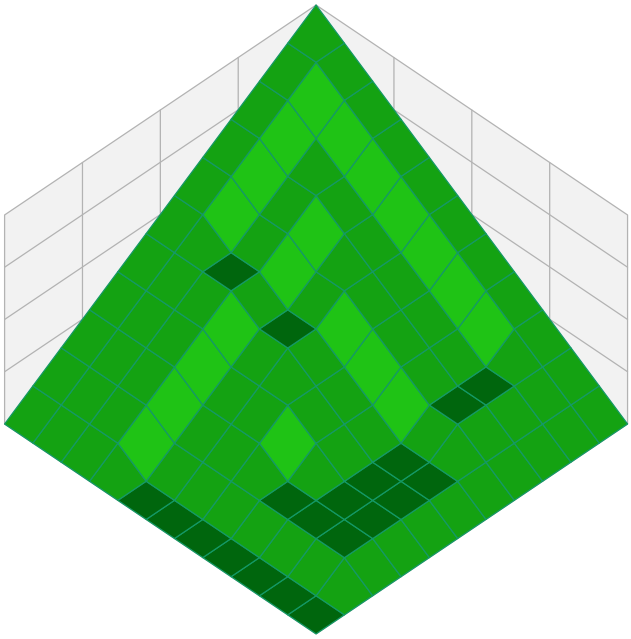
```
> m = mecdf (x, continuous=TRUE)
> plotbcdf (m, simple=FALSE)
```



Projecting ECDFs onto Unit Cubes

It is possible to project an ECDF model onto a unit (potentially hyper) cube, with marginal distributions on the interval (0, 1). I have no idea if this is useful or not. However, we can do it very easily (by replacing realised values with their index, if we were to sort them, one variable at a time), it's kind of interesting, and it does have a vague resemblance to a certain other multivariate approach, so why not...

```
> m = mecdf (x, continuous=TRUE, project=TRUE)
> plotbcdf (m, simple=FALSE)
```



Model Expansion

The mecdf function contains an argument expand (which represents what we will describe as an expansion factor). If this is NA it doesn't do anything. However if it is a numeric value (and it's 0.1 by default), two fake realisations are added to our sample. For each variable, the range is computed, multiplied by the expansion factor, to give an offset, then one value is computed for the minimum actual value less the offset, and another value for the maximum value plus the offset. There are two reasons for doing this.

Firstly, often the values in our sample, do not represent to smallest and largest possible values, and this method provides a simple heuristic-based allowance for this.

The second reason is a solution to a problem that can arise using the interpolation method described earlier (for two or more variables).

Obviously, when creating a continuous ECDF model we need a continuous distribution function. A distribution function should be monotonically non-decreasing in the direction of each variable, and only have values between 0 and 1. However, there is only a subtle difference between being monotonically non-decreasing and monotonically increasing.

ECDFs, even in the continuous case, have a step-like quality, whenever we move over the surface and pass a vertex, there is a tendency to increase the value of the function representing the surface. The method presented for interpolation is monotonically non-decreasing if we consider any value in the

domain (over even any vertex), however it is monotonically increasing if we only consider the expected surface at the actual realisations.

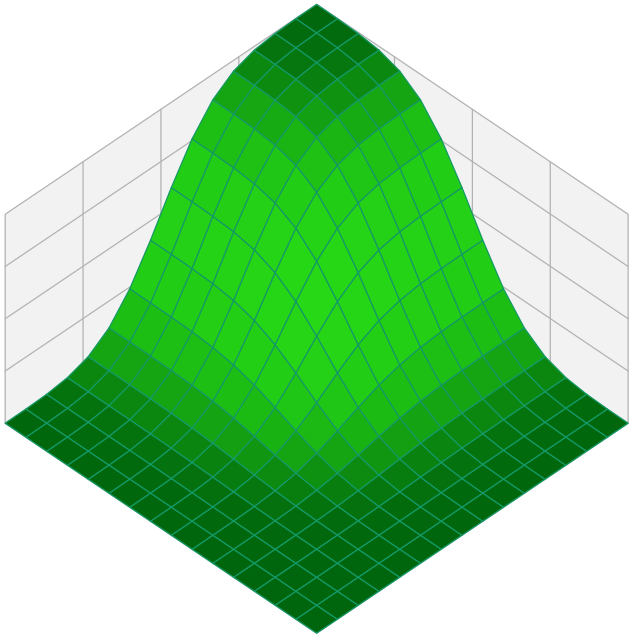
Which is where the problem arises. For two values u_1 and u_2, which define a vertex (on a bivariate continuous ECDF), if u_1 is equal to the smallest realised value of our first variable, the distribution function must be 0. However there are two possible cases here, u_2 could the smallest realised value of our second variable (in which case there's no problem), however if not, the vertex (u_1, u_2) defines a point where the distribution function needs to increase in the direction of the second variable, hence we have values where our distribution should be zero, however they also need to increase from a smaller value to a larger value.

The simple heuristic presented above, also fixes this problem, because the extra points, ensure that the above situation can never happen.

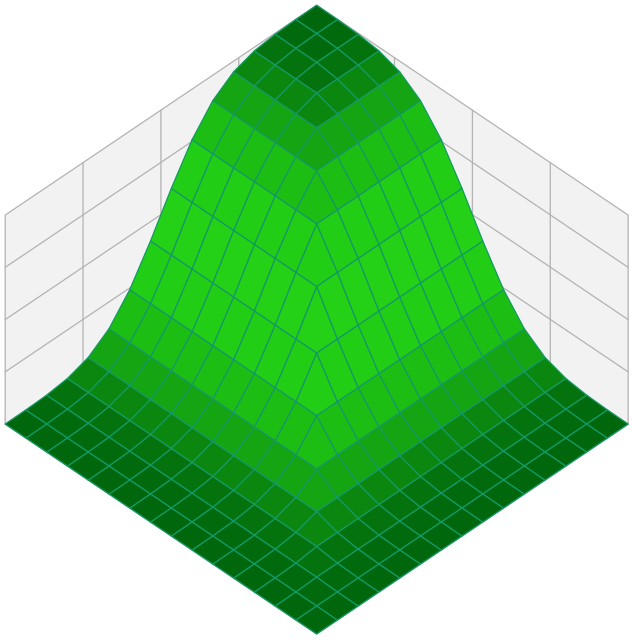
The Bivariate Normal vs The Bivariate ECDF

First let us consider a (standard) bivariate normal distribution. We will create three examples with no correlation, perfect positive correlation and perfect negative correlation. We are going to use the mvtnorm package to help us.

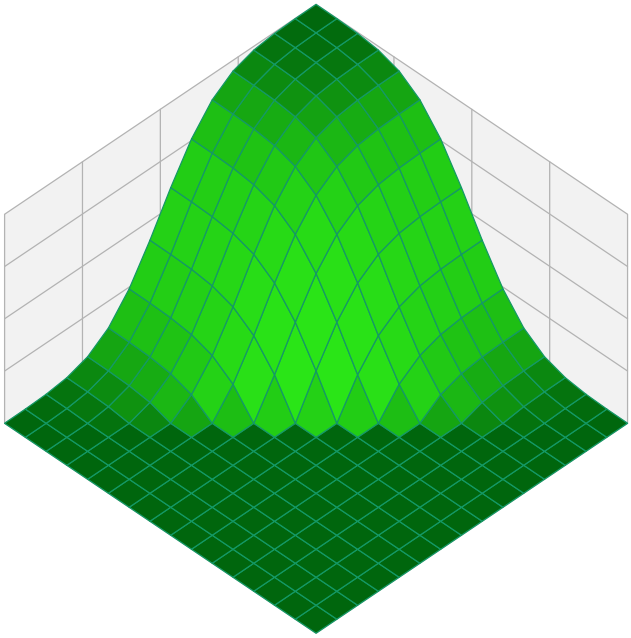
```
> #get warmed up...
> library (mvtnorm, warn=FALSE)


> #create a function, to plot the distributions
> normal.plot = function (cor=0, res=16, rng=c (-3.5, 3.5) )
 {        x1 = x2 = seq (rng [1], rng [2], length=res)
         m = matrix (numeric (), nr=res, nc=res)
         for (i in 1:res) for (j in 1:res)
                 m [i, j] = pmvnorm (c (-Inf, -Inf), c (x1 [i], x2 [j]),
                         sigma=matrix (c (1, cor, cor, 1), nr=2) )
         plotbcdf (m)
 }


> normal.plot(0)
```



```
> normal.plot (1)
```

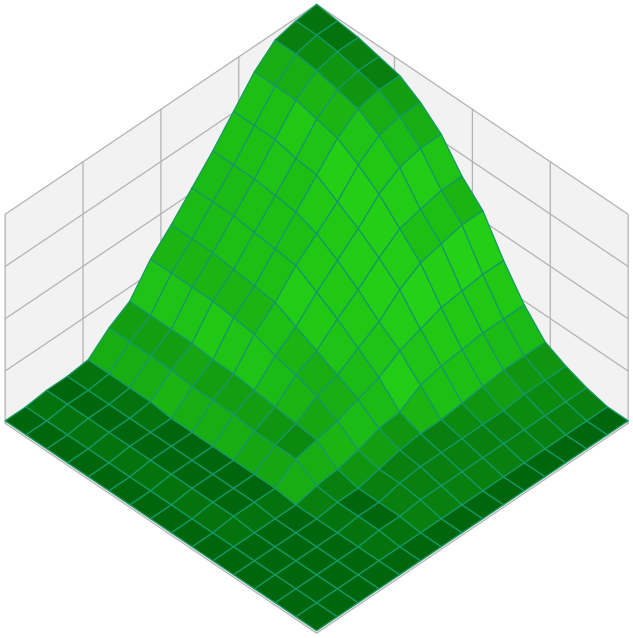


```
> normal.plot (-1)
```

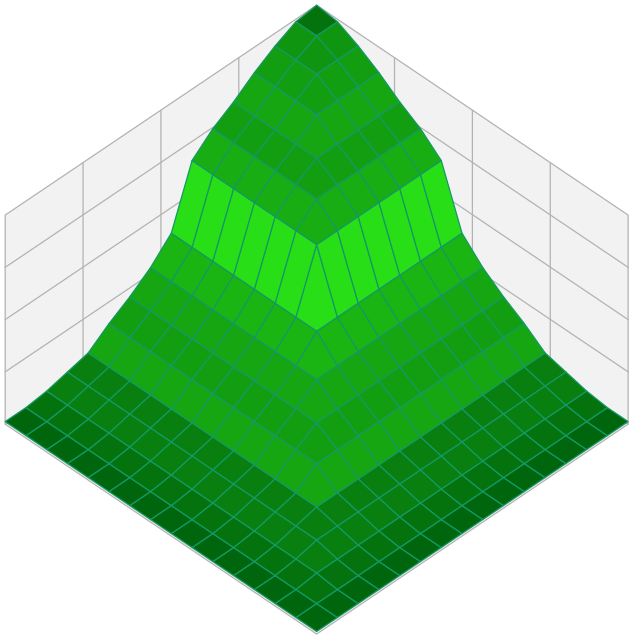


Now let's create three simulated datasets, from each of the distributions above (also using the mvtnorm package), and model the simulated data using bivariate ECDFs (from the mecdf package). We will use a sample size of 120. We could be pedantic and ensure the same ranges of the plots, however it makes little difference, so we will simply plot over the observed range.

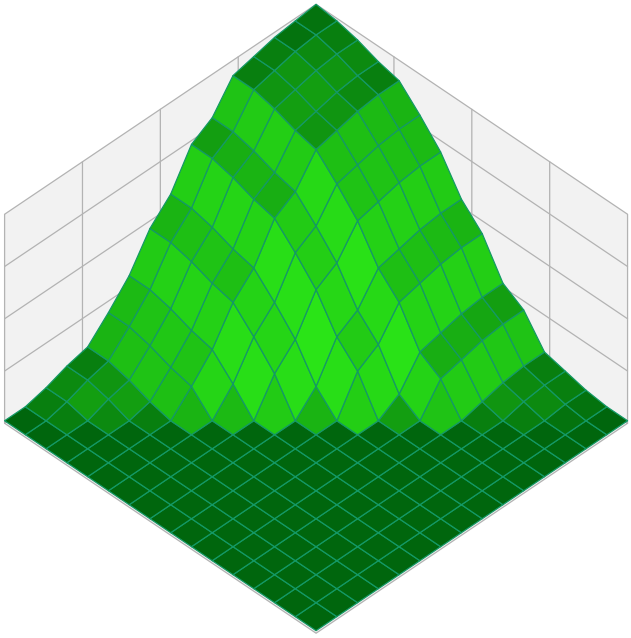
```
> #another convenience function
> normal.sim = function (n=1, cor=0)
         rmvnorm (80, c (0, 0), matrix (c (1, cor, cor, 1), nr=2) )
> m0 = mecdf (normal.sim (120, 0) )
> mpos = mecdf (normal.sim (120, 1) )
> mneg = mecdf (normal.sim (120, -1) )


> plotbcdf (m0)
```



```
> plotbcdf (mpos)
```



```
> plotbcdf (mneg)
```



Acknowledgments

Special thanks to:

- Martin Maechler, for the original "ecdf" function in R.
- Christophe Dutang, for providing some encouragement and feedback to some of my earlier ideas.
- The mvtnorm package authors, for creating a package that I've used to test my models with.

Todo

1. Refine the theoretical framework for the models, along with further research into associated multivariate models.
2. Derive (hopefully) conditional expectation and conditional ECDF models, directly from the multivariate ECDF models.
3. Polish the vignette.
4. Create some good examples, including examples with more than two variables.
5. Allow matrix arguments, for the distribution functions.
6. Give the plotting functions more options.
7. Optimise, and optimise some more...