

How to use the **vegclust** package (ver. 1.6.5)

Miquel De Cáceres¹ and Susan K. Wiser²

¹Centre Tecnològic Forestal de Catalunya, Solsona, Catalonia, Spain

²Landcare Research, Lincoln, New Zealand

April 11, 2018

Contents

1	Introduction	2
1.1	What is this tutorial about?	2
1.2	Example vegetation data	2
2	Clustering methods in <code>vegclust</code>	3
2.1	Resemblance space	3
2.2	Prototype-based clustering	4
2.3	Clustering models	4
2.3.1	Hard (crisp) or fuzzy memberships	5
2.3.2	Centroids or medoids	5
2.3.3	Partitive clustering	6
2.3.4	Noise clustering	7
2.3.5	Possibilistic clustering	8
2.4	Dissimilarity-based duals	8
2.4.1	Medoid-based clustering and dissimilarity matrices . .	9
2.4.2	Centroid-based clustering and dissimilarity matrices .	9
3	Managing vegetation classifications	10
3.1	Creating classifications: <code>vegclust</code> and <code>vegclustdist</code>	10
3.1.1	The K-means model	10
3.1.2	The Fuzzy C-means model	12
3.1.3	The Noise clustering model	15
3.1.4	Medoid-based clustering	18
3.2	Supervised classification: <code>as.vegclust</code> and <code>vegclass</code>	19
3.3	Extending vegetation classifications	21
3.3.1	Conforming vegetation data sets	21
3.3.2	Re-calculating the centroids of the initial classification	22
3.3.3	Calling <code>vegclust</code> with fixed prototypes	22

3.3.4	Extending or refining classifications?	23
3.3.5	Using <code>vegclustdist</code> with fixed prototypes	23
4	Cluster characterization	24
4.1	Cluster prototypes: <code>clustcentroid</code> and <code>clustmedoid</code>	24
4.2	Cluster internal variability: <code>clustvar</code>	25
4.3	Distance between clusters: <code>interclustdist</code>	26
4.4	Constancy classes: <code>clustconst</code>	27

1 Introduction

1.1 What is this tutorial about?

Classification of vegetation plot records involves different activities, including the design of an appropriate vegetation survey, the use of a classification method to group vegetation observations and the characterization, validation and naming of the resulting vegetation groups. In this tutorial we focus on only one of this steps, namely to group vegetation observations, and we show how to conduct it with the help of the R package `vegclust`. Before starting our examples we need to load the `vegclust` package, which also loads the required R packages `vegan` and `permute`:

```
> library(vegclust)
```

1.2 Example vegetation data

In order to illustrate the functions in `vegclust` we will use a small wetland vegetation data set, consisting of 41 sites and 33 species and published by Bowman and Wilson [1986]. The data is included with the `vegclust` package:

```
> data(wetland)
> dim(wetland)
```

```
[1] 41 33
```

For a number of reasons that we will not detail here, the Euclidean distance is not an appropriate index to measure the resemblance in species composition between vegetation plot records. Therefore, we transform our community data using the chord transformation [Legendre and Gallagher, 2001], which divides each value by the norm of the row vector of the corresponding site:

```
> wetlandchord = decostand(wetland, "normalize")
```

Function `decostand` is provided within the `vegan` package. The Euclidean distance on the transformed data is equal to the chord distance [Orlóci, 1967] with respect to the raw community data:

```
> dchord = dist(wetlandchord)
```

In some of our examples we will use the R objects `wetlandchord` or `dchord` indistinctively, because package `vegclust` allows vegetation to be classified from either a site-by-species data table or from a site-by-site dissimilarity matrix. In the next section we briefly explain the bases of the classification methods that are provided in the `vegclust` package. We later show how to run those methods using functions in the package.

2 Clustering methods in `vegclust`

2.1 Resemblance space

Generally speaking, the goal of clustering is to derive c ‘natural’ classes or clusters from a set of n unlabelled objects. Those objects inside a ‘natural’ cluster show a certain degree of closeness or similarity and the cluster itself shows a certain degree of isolation from other clusters. In classification of vegetation the n ‘objects’ to be grouped are samples of plant communities (i.e. plot records or relevés) and the goal is to define vegetation types.

When speaking of ‘proximity’ or ‘similarity’, we implicitly assume there is a procedure to assess the degree of resemblance between the objects to be grouped. This procedure usually involves describing our objects using a set of p features (these are normally species in the case of vegetation) and specifying a resemblance measure (e.g. distance or dissimilarity). Let $\mathbf{X} = [x_{js}]$ be a site-by-species data table of dimensions $n \times p$, where x_{js} is the abundance of species s in site j , and let d be an appropriate dissimilarity or distance measure.

Another way to formalize the resemblance between objects is to directly provide the similarity or dissimilarity between pairs of objects in a symmetric resemblance matrix. Let $\mathbf{D} = [d_{ij}]$ be a symmetric dissimilarity matrix of dimensions $n \times n$, where $d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$ is the dissimilarity between objects i and j . In classification of vegetation $d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$ may represent the compositional dissimilarity between the two plant communities i and j , but it could be based on other vegetation attributes such as physiognomy.

Regardless of whether we use \mathbf{X} or \mathbf{D} , we will speak here of *resemblance space* and our objects (plant communities) will be formally represented as points in this space. Although we do not cover this topic in detail here, the reader should be aware that building an appropriate resemblance space is critical to achieving a vegetation classification that fits a given purpose. The results of the classification exercise may be completely different depending on the way the resemblance space is defined.

2.2 Prototype-based clustering

Prototype-based clustering methods assume that the properties of objects in a cluster can be represented using a cluster *prototype*, which is formalized as a point in the resemblance space. The problem is thus to find c prototypes and assign the n objects according to their proximity to those prototypes, such that the resulting clusters are compact and isolated one from another.

All the clustering methods discussed here follow an *alternate optimization* scheme, meaning that one group of parameters (e.g. the membership matrix) is optimized by holding the other group (e.g. the cluster prototypes) fixed and vice versa. Assuming an initial cluster configuration, this is achieved by iterating the following three steps:

- (S.1) Determine the prototype of each cluster i .
- (S.2) Calculate e_{ij} , the distance from each object j to the prototype of each cluster i .
- (S.3) Calculate u_{ij} , the (hard or fuzzy) membership of each object j to each cluster i (i.e. re-assign all objects into clusters).

Each iteration of the alternate optimization algorithm improves the *objective function* of the clustering method. The alternate optimization algorithm stops when there are no further changes in object memberships. More technically, it stops if the maximum difference in object membership values of the last two iterations does not exceed a user-specified threshold. The starting configuration of clusters is a critical issue in this kind of classification methods, because the iterations can result in the alternate optimization algorithm getting stuck in a suboptimal value of the objective function. For this reason, several executions of the algorithm are usually tried, each one using a different starting configuration.

2.3 Clustering models

There are several prototype-based clustering models in the **vegclust** package; all of which follow the iterative algorithm presented above. The differences between clustering models arise due to differences in the specific implementation of each step, resulting from different assumptions of how clusters should be defined. The clustering models can be divided according to their properties into:

- a) Whether object memberships are **fuzzy** or **hard** (crisp).
- b) Whether cluster prototypes are **centroids** or **medoids**.
- c) How outlier objects are handled, which gives three kinds of models: **partitive**, **noise clustering** or **possibilistic clustering**.

In the following subsections we describe the implications of each of these decisions.

2.3.1 Hard (crisp) or fuzzy memberships

With the kind of classification methods described here, it is customary to represent the classification of objects into groups using a $c \times n$ membership matrix, $\mathbf{U} = [u_{ij}]$, where u_{ij} is the degree of membership of object j in cluster i . The classification is said to be *hard* when the u_{ij} values are either 0 (object j DOES NOT belong to cluster i) or 1 (object j DOES belong to cluster i). In contrast, if the classification is *fuzzy* the membership of its objects is expressed through a degree of membership bounded between 0 (i.e. the object does not belong to the set at all) and 1 (i.e. the object belongs completely to the set).

The advantages of using fuzzy set theory in vegetation classification are that it acknowledges the individualistic concept of species distribution across gradients of vegetation composition [Moravec, 1989], and therefore avoids assuming that vegetation observations (e.g. releves) must be unequivocal representatives of a type without no admixture of any other types [Dale, 1995].

2.3.2 Centroids or medoids

The *centroid* of a cluster i that contains n_i objects is a vector \mathbf{c}_i whose coordinates are the average, in each dimension s , of the coordinates of the n_i objects belonging to the cluster. In vector notation:

$$\mathbf{c}_i = \frac{\sum_{j=1}^{n_i} \mathbf{x}_j}{n_i} \quad (1)$$

The centroid has the property that minimizes the sum of squared Euclidean distances between itself and each point belonging to the cluster. Equation 1 can be generalized to the case of a fuzzy cluster by weighting the coordinates each object by its degree of membership to the cluster, u_{ij} :

$$\mathbf{c}_i = \frac{\sum_{j=1}^n u_{ij}^m \mathbf{x}_j}{\sum_{j=1}^n u_{ij}^m} \quad (2)$$

In equation 2, $m > 0$ is the *fuzziness exponent*, which is used to modulate the influence of fuzzy memberships in the calculation of the centroids. If m is very large only the objects whose membership is close to 1 will have an influence on the centroid. On the contrary, if m is small (i.e. close to 0) then all n objects will influence the centroid equally and it will approach the overall data center.

A *medoid* of a cluster i that contains n_i objects is defined as the object, chosen among the n_i objects, for which the sum of dissimilarities to all the

n_i objects is minimal i.e. it is a most centrally located point in the cluster. Formally, the medoid is the object k for which:

$$\sum_{j=1}^{n_i} d(\mathbf{x}_k, \mathbf{x}_j) \quad (3)$$

is minimal. When using fuzzy logic, the medoid of cluster i is defined as the object k (among all n objects) that minimizes:

$$\sum_{j=1}^n u_{ij}^m d(\mathbf{x}_k, \mathbf{x}_j) \quad (4)$$

Note that, because the medoid is a point chosen among the n input objects, we do not need to calculate its coordinates (although see explanation below for centroids), moreover, the distance between a medoid and the other objects (step S.2 in the alternate optimization algorithm) will be readily available from the beginning, so it does not need to be computed. All this makes dealing with medoids computationally much faster than dealing with centroids.

2.3.3 Partitive clustering

A clustering method is called *partitive* if object memberships (crisp or fuzzy) are constrained to sum to one for each object:

$$\sum_{i=1}^c u_{ij} = 1 \quad (5)$$

This constrain is usually referred to as the *partition restriction*. It ensures that all objects will be classified as either belonging to a single cluster or dividing their membership among different clusters. No objects will be left unclassified.

K-means (KM, also known as hard c-means) [MacQueen, 1967] and Fuzzy c-means (FCM) [Bezdek, 1981] are two, centroid-based, partitive clustering algorithms widely used in many unsupervised pattern recognition applications. The main difference between the two methods is that in KM every object belongs to a single cluster (i.e clusters are ‘hard’) whereas in FCM the memberships are fuzzy and a given object may have some degree of membership to more than one cluster.

Cluster memberships (step S.3) are determined in KM simply by *assigning each object to the cluster whose centroid is closest*. In the case of FCM, fuzzy memberships are calculated using the following formula:

$$u_{ij} = \frac{1}{\sum_{l=1}^c (e_{ij}/e_{lj})^{2/(m-1)}} \quad (6)$$

As said above, $m > 1$ is the fuzziness coefficient. The smaller the value of m , the closer to a hard partition will be the result. If m is set too high and the data is noisy the resulting partition may be completely fuzzy (i.e. where $u_{ij} = 1/c$ for all objects and clusters) and therefore uninformative.

As indicated above, KM and FCM are centroid-based, meaning they use equations 1 and 2, respectively, for step S.1. The corresponding medoid-based methods are Hard C-medoids and Fuzzy C-medoids, which instead use equations 3 and 4, respectively [Krishnapuram et al., 1999].

2.3.4 Noise clustering

The noise clustering (NC) method [Dave, 1991] is an attempt to make the FCM method more robust to the effect of outliers. The rationale underlying NC is the following: if an object is an outlier, this means that it lies far from all cluster prototypes and, therefore, it should have low membership values to all clusters. In order to achieve these low memberships, the NC considers an additional cluster, called *Noise*. This class is represented by an imaginary ‘prototype’ that lies at a constant distance δ from all the data points. Even if such a prototype does not really exist, the effect of including the Noise class is that it ‘captures’ objects that lie farther than δ from all the c ‘real’ prototypes. The NC membership function (step S.3) for the ‘real’ clusters is:

$$u_{ij} = \frac{1}{(e_{ij}/\delta)^{2/(m-1)} + \sum_{l=1}^c (e_{ij}/e_{lj})^{2/(m-1)}} \quad (7)$$

whereas the fuzzy membership to the Noise class, u_{Nj} , is one minus the sum of memberships to the real clusters.

$$u_{Nj} = 1 - \sum_{i=1}^c u_{ij} \quad (8)$$

Outlier objects have small membership values to the c real clusters because the first term in the denominator of equation 7 is large. The smaller the δ , the higher the memberships to the Noise class will be. In contrast, large values of δ make NC equivalent to FCM. In NC, the fuzziness exponent m has the same interpretation as in FCM. Including the Noise class has the effect of relaxing the partition restriction. In NC, the partition restriction is fulfilled when all c real clusters and the Noise class are considered.

Note that, like FCM and KM, we can define a ‘hard’ counterpart of the (fuzzy) noise clustering method. Indeed, the hard noise clustering (HNC) method differs from the fuzzy one in that memberships are not fuzzy. Like KM, its membership function can be described verbally. One assigns the object to the noise class if the distances to all cluster centroids is larger than δ . Otherwise, one assigns the object to the cluster whose centroid is closest, as in KM.

The noise clustering method was originally defined with centroids as prototypes. However, either hard or fuzzy noise clustering can be applied to medoids instead of centroids. Although we have found no references exploring this approach, the corresponding algorithms could be named ‘hard noise clustering with medoids’ (HNCdd) and ‘(fuzzy) noise clustering with medoids’ (NCdd).

2.3.5 Possibilistic clustering

Possibilistic C-means [Krishnapuram and Keller, 1993, 1996] is another modification of FCM seeking increased cluster robustness. The partition restriction is eliminated in PCM, which produces c independent fuzzy clusters, each corresponding to a dense region of points. Whereas the FCM and NC membership functions compare the distance from the object to the cluster of interest, e_{ij} , with the distances to the remaining prototypes (and to the noise class in the case of NC), in PCM the membership value for a given object to a cluster does not depend on the distances to the remaining cluster prototypes. Instead, the distance to the cluster of interest is compared to a reference distance (η_i):

$$u_{ij} = \frac{1}{1 + (e_{ij}^2 / \eta_i)^{1/(m-1)}} \quad (9)$$

The reference distance is a parameter that must be provided for each cluster. All objects whose distance to the cluster center is smaller than η_i will obtain a membership value higher than 0.5.

The fact that the membership to a given cluster does not depend on the distances to the remaining cluster prototypes entails that cluster repulsion is eliminated in PCM, with the consequence that samples can have high membership to different clusters. Good estimation of η_i is crucial for the success of the PCM method [De Cáceres et al., 2006]. Inadequate initialization of η_i can lead to a loss of cluster structures, even with the correct partition as initial starting configuration. A single PCM run can be regarded as c independent runs of NC, each looking for a single cluster and where $\delta_i^2 = \eta_i$ [Dave and Krishnapuram, 1997]. In vegetation data plant communities with intermediate characteristics are frequent. This fact makes the PCM method impractical for classification of vegetation, because without cluster repulsion PCM clusters are frequently highly mobile and converge to the same cluster, leaving large parts of the data unassigned [De Cáceres et al., 2010].

2.4 Dissimilarity-based duals

All the clustering methods presented above can be executed on a resemblance space described using either \mathbf{X} or \mathbf{D} . The latter case avoids explicitly dealing with coordinates when creating groups.

2.4.1 Medoid-based clustering and dissimilarity matrices

Because medoids are selected among the objects to be classified it is obvious that the distance to the cluster prototypes, e_{ij} , can be drawn from a symmetric matrix of pairwise distances between objects calculated before the alternate optimization algorithm is initiated. In other words, one can conduct medoid-based clustering on a site-by-site distance matrix instead of using a site-by-species rectangular matrix. Moreover, one can straightforwardly skip the use of Euclidean distance and use a dissimilarity measure more appropriate for ecological data.

2.4.2 Centroid-based clustering and dissimilarity matrices

When dealing with centroids, it may seem unavoidable to calculate centroid coordinates (step S.1) before calculating the (squared) Euclidean distances to cluster centers e_{ij} (step S.2):

$$e_{ij}^2 = \|\mathbf{c}_i - \mathbf{x}_j\|^2 \quad (10)$$

However, there is a mathematical trick that avoids the need to calculate the coordinates \mathbf{c}_i explicitly. Let \mathbf{D} be the matrix of Euclidean distances between pairs of objects. We can obtain e_{ij}^2 as follows:

$$e_{ij}^2 = \frac{1}{n_i} \sum_{l=1}^{n_i} d_{lj}^2 - \frac{1}{2n_i^2} \sum_{k=1}^{n_i} \sum_{l=1}^{n_i} d_{lk}^2 \quad (11)$$

The left part of this equation is a sum of squared distances from the target object to all the other objects in the cluster. The right part of the equation is an average of squared distances between objects in the cluster. Equation 11 can be generalized to the case of a fuzzy cluster:

$$e_{ij}^2 = \frac{1}{\sum_{k=1}^n u_{ik}^m} \sum_{l=1}^{n_i} d_{lj}^2 - \frac{1}{2 \left(\sum_{k=1}^n u_{ik}^m \right)^2} \sum_{k=1}^n \sum_{l=1}^n u_{ik}^m u_{il}^m d_{lk}^2 \quad (12)$$

Equations 11 and 12 allow the Euclidean distance to a centroid to be determined without calculating its coordinates. Therefore, they allow steps S.1 and S.2 to be combined into a single step. In other words, distance-based duals exist for centroid-based clustering methods when the resemblance space comprises Euclidean distances [Hathaway et al., 1989, 1996].

If we transform the original data in order to emulate a distance (like the chord), then the duality holds, although centroids are defined in the transformed space. What happens if the values in \mathbf{D} were not calculated using the Euclidean distance? Equations 11 and 12 are also valid for other dissimilarity measures, although there are important details to be remembered. These equations assume that the resemblance space is Euclidean (i.e. does not produce negative eigenvalues in principal coordinates analysis) and

that centroids are appropriate prototypes for clusters. If the dissimilarities do not have the Euclidean property some oddities may arise [Hathaway and Bezdek, 1994]. For example, it is possible to obtain negative e_{ij}^2 values, specially for groups of small size. In practice, however, when these negative distances occur they can be reset to zero [De Cáceres et al., 2006].

3 Managing vegetation classifications

3.1 Creating classifications: `vegclust` and `vegclustdist`

Functions `vegclust` and `vegclustdist` allow vegetation types to be defined from a set of unlabelled vegetation observations (i.e. relevés or plot records) using any of the clustering models explained in the previous section. `vegclust` requires a rectangular site-by-species matrix, whereas `vegclustdist` requires a symmetric site-by-site dissimilarity matrix.

3.1.1 The K-means model

The following piece of code produces a classification of our example data set into three groups using the K-means clustering model:

```
> wetland.km = vegclust(x = wetlandchord, mobileCenters=3,
+                       method="KM", nstart=20)
```

The result is an object of class ‘vegclust’, in fact a list with several components (method, parameters, prototypes, objective function, etc.):

```
> names(wetland.km)

[1] "mode"          "method"        "m"
[4] "dnoise"        "eta"           "memb"
[7] "mobileCenters" "fixedCenters"  "dist2clusters"
[10] "withinss"      "size"          "functional"
[13] "iter"
```

One of the most important components is the membership matrix **U**, which we show transposed here:

```
> t(wetland.km$memb)

      5  8 13  4 17  3  9 21 16 14  2 15  1  7 10 40 23 25 22 20  6 18 12 39
M1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0
M2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
M3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

      19 11 30 34 28 31 26 29 33 24 36 37 41 27 32 35 38
M1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1
M2  1  1  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0
M3  0  0  1  1  1  1  0  0  0  1  1  1  1  1  1  0  0
```

Another important component is the matrix containing the coordinates of cluster centroids (i.e. vectors \mathbf{c}_i for each cluster):

```
> round(wetland.km$mobileCenters, dig=3)
```

	Abefic	Merhed	Alyvag	Pancam	Abemos	Melcor	Ludoct	Eupvac	Echpas
M1	0.000	0.000	0.000	0.027	0.000	0.062	0.000	0.039	0.048
M2	0.000	0.000	0.047	0.076	0.016	0.153	0.000	0.050	0.066
M3	0.018	0.068	0.000	0.356	0.015	0.275	0.019	0.205	0.102

	Passcr	Poa2	Carhal	Dendio	Casobt	Aesind	Cyprot	Ipocop	Cynarc
M1	0.000	0.00	0.037	0.016	0.016	0.000	0.000	0.000	0.000
M2	0.000	0.01	0.069	0.000	0.000	0.075	0.081	0.013	0.013
M3	0.026	0.00	0.000	0.000	0.000	0.019	0.176	0.040	0.432

	Walind	Sessp.	Phynod	Echell	Helind	Ipoaqu	Orysp.	Elesp.	Psespi
M1	0.068	0.144	0.000	0.049	0.125	0.055	0.649	0.154	0.192
M2	0.064	0.104	0.715	0.119	0.172	0.048	0.172	0.138	0.027
M3	0.078	0.157	0.064	0.112	0.019	0.037	0.040	0.000	0.000

	Ludads	Polatt	Poa1	Helcri	Physp.	Goopur
M1	0.204	0.097	0.000	0.000	0.012	0.012
M2	0.000	0.000	0.013	0.013	0.000	0.000
M3	0.000	0.000	0.000	0.000	0.000	0.000

The same classification exercise can be conducted from the matrix of chord distances between objects if we use function `vegclustdist` instead of `vegclust`:

```
> wetland.kmdist = vegclustdist(x = dchord, mobileMemb=3,
+                               method="KM", nstart = 20)
> names(wetland.kmdist)
```

[1]	"mode"	"method"	"m"
[4]	"dnoise"	"eta"	"memb"
[7]	"mobileCenters"	"fixedCenters"	"dist2clusters"
[10]	"withinss"	"size"	"functional"

Note the different way to specify the number of clusters. In the case of `vegclustdist` we do not obtain cluster centroids, because they cannot be calculated explicitly:

```
> wetland.kmdist$mobileCenters
```

NULL

But we do obtain cluster memberships:

```
> t(wetland.kmdist$memb)
```

	5	8	13	4	17	3	9	21	16	14	2	15	1	7	10	40	23	25	22	20	6	18	12	39
M1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
M2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	19	11	30	34	28	31	26	29	33	24	36	37	41	27	32	35	38							
M1	1	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0							
M2	0	0	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0							
M3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1							

Because we used the same resemblance space (although in different form) in our examples, both `vegclust` and `vegclustdist` should give the same result provided the algorithm has not become stuck in a relative optimum of the objective function. Although both R functions return objects of class ‘vegclust’, we can identify whether calculations were done from dissimilarities or from rectangular matrices by inspecting the ‘mode’ element of the list:

```
> wetland.km$mode
```

```
[1] "raw"
```

```
> wetland.kmdist$mode
```

```
[1] "dist"
```

In the following subsections we run `vegclust` using other clustering models, but the same examples could be made using `vegclustdist`.

3.1.2 The Fuzzy C-means model

Let us inspect the distance of each object to each cluster centroid:

```
> round(t(wetland.km$dist2clusters), dig=2)
```

	5	8	13	4	17	3	9	21	16	14	2	15
M1	0.54	0.68	0.44	0.45	0.64	0.76	0.95	0.50	0.64	0.46	0.38	0.57
M2	1.05	1.10	1.09	1.14	1.11	1.22	1.23	1.09	1.09	1.10	1.12	1.19
M3	1.12	1.12	1.14	1.17	1.07	1.22	1.22	1.20	1.17	1.16	1.17	1.21
	1	7	10	40	23	25	22	20	6	18	12	39
M1	0.47	0.72	0.87	1.11	0.92	0.86	0.73	1.11	1.15	1.19	1.16	1.12
M2	1.12	1.15	1.19	0.40	0.60	0.51	0.67	0.56	0.47	0.45	0.40	0.52
M3	1.21	1.18	1.23	1.09	1.07	1.12	1.03	1.15	1.06	1.12	1.12	1.06
	19	11	30	34	28	31	26	29	33	24	36	37
M1	0.97	1.13	1.20	1.22	1.19	1.20	1.22	1.17	1.22	1.20	1.18	1.17
M2	0.44	0.64	1.16	1.20	1.15	1.19	0.81	0.78	0.49	1.04	0.90	1.15
M3	1.07	1.07	0.86	0.66	0.77	0.69	1.00	0.92	1.00	0.55	0.62	0.55

	41	27	32	35	38
M1	0.99	1.22	1.17	0.80	0.68
M2	1.14	1.19	1.13	1.03	1.10
M3	0.56	0.78	0.73	0.95	0.99

For many objects the distance to the cluster where they have been assigned is much smaller than the distance to other clusters. However, for some objects (such as '22', '29' or '35') the distance to the closest cluster center does not differ much from the distance to second closest one. Are those latter objects well assigned? Should these objects have intermediate degrees of membership instead of picking one cluster arbitrarily? The Fuzzy C-means cluster model allows fuzzy memberships to be obtained as an alternative to crisp memberships:

```
> wetland.fcm = vegclust(x = wetlandchord, mobileCenters=3,
+                         method="FCM", m=1.2, nstart=20)
> round(t(wetland.fcm$memb), dig=3)
```

	5	8	13	4	17	3	9	21	16	14	2	15	1	7
M1	0.998	0.985	1	1	0.990	0.981	0.844	1	0.993	1	1	0.999	1	0.981
M2	0.001	0.007	0	0	0.007	0.011	0.086	0	0.003	0	0	0.001	0	0.009
M3	0.001	0.007	0	0	0.004	0.009	0.070	0	0.004	0	0	0.001	0	0.011

	10	40	23	25	22	20	6	18	12	39	19	11	30
M1	0.920	0	0.016	0.007	0.404	0.001	0	0	0	0.000	0	0.003	0.031
M2	0.036	0	0.004	0.001	0.014	0.001	0	0	0	0.001	0	0.005	0.928
M3	0.044	1	0.980	0.992	0.582	0.998	1	1	1	0.999	1	0.992	0.040

	34	28	31	26	29	33	24	36	37	41
M1	0.002	0.013	0.003	0.017	0.017	0.000	0.000	0.001	0.001	0.004
M2	0.996	0.971	0.993	0.141	0.241	0.001	0.998	0.978	0.999	0.995
M3	0.002	0.016	0.003	0.841	0.742	0.999	0.002	0.020	0.001	0.001

	27	32	35	38
M1	0.012	0.008	0.783	0.970
M2	0.973	0.980	0.160	0.024
M3	0.015	0.011	0.057	0.006

A comparison of these memberships with the distance to the clusters shown before will reveal that intermediate objects obtain fuzzier membership values than other objects.

Although FCM is theoretically a better model than KM for vegetation classification, vegetation scientists are normally interested in crisp assignments. The function `defuzzify` allows fuzzy membership matrix to be converted into a crisp one:

```
> groups = defuzzify(wetland.fcm)$cluster
> groups
```

```

      5      8     13      4     17      3      9     21     16     14      2     15      1
"M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1"
      7     10     40     23     25     22     20      6     18     12     39     19     11
"M1" "M1" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3" "M3"
     30     34     28     31     26     29     33     24     36     37     41     27     32
"M2" "M2" "M2" "M2" "M3" "M3" "M3" "M2" "M2" "M2" "M2" "M2" "M2"
     35     38
"M1" "M1"

```

```
> table(groups)
```

```

groups
M1 M2 M3
17 10 14

```

Another way of defuzzifying the membership matrix is by setting a threshold of minimum fuzzy membership:

```
> groups = defuzzify(wetland.fcm, method = "cut", alpha = 0.8)$cluster
> groups
```

```

      5      8     13      4     17      3      9     21     16     14      2     15      1
"M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1" "M1"
      7     10     40     23     25     22     20      6     18     12     39     19     11
"M1" "M1" "M3" "M3" "M3"  NA "M3" "M3" "M3" "M3" "M3" "M3" "M3"
     30     34     28     31     26     29     33     24     36     37     41     27     32
"M2" "M2" "M2" "M2" "M3"  NA "M3" "M2" "M2" "M2" "M2" "M2" "M2"
     35     38
     NA "M1"

```

```
> table(groups, useNA = "always")
```

```

groups
  M1  M2  M3 <NA>
  16  10  12    3

```

With this second defuzzification approach intermediate objects remain unclassified (indicated as NA's). It is important to recognise that FCM fuzzy membership values depend on the fuzziness exponent m . In fact, if we run FCM with a very large m value we will obtain uninformative results:

```
> wetland.fcm2 = vegclust(x = wetlandchord, mobileCenters=3,
+                          method="FCM", m=10, nstart=20)
> round(t(wetland.fcm2$memb), dig=3)
```

	5	8	13	4	17	3	9	21	16	14
M1	0.333	0.333	0.332	0.332	0.333	0.333	0.333	0.333	0.333	0.333
M2	0.334	0.334	0.334	0.334	0.334	0.334	0.334	0.334	0.334	0.334
M3	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333

	2	15	1	7	10	40	23	25	22	20
M1	0.332	0.333	0.333	0.333	0.333	0.334	0.334	0.334	0.334	0.334
M2	0.334	0.334	0.334	0.334	0.334	0.333	0.333	0.333	0.333	0.333
M3	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333

	6	18	12	39	19	11	30	34	28	31
M1	0.334	0.334	0.335	0.334	0.334	0.334	0.333	0.333	0.333	0.333
M2	0.332	0.333	0.332	0.333	0.333	0.333	0.333	0.333	0.333	0.333
M3	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.334	0.333	0.334

	26	29	33	24	36	37	41	27	32	35
M1	0.334	0.334	0.334	0.333	0.334	0.333	0.333	0.333	0.333	0.333
M2	0.333	0.333	0.332	0.333	0.333	0.333	0.334	0.333	0.333	0.334
M3	0.333	0.333	0.333	0.334	0.333	0.334	0.334	0.333	0.334	0.333

	38
M1	0.333
M2	0.334
M3	0.333

These uninformative results become obvious after defuzzification of the membership matrix:

```
> groups2 = defuzzify(wetland.fcm2, method = "cut", alpha = 0.8)$cluster
> table(groups2, useNA = "always")
```

```
groups2
<NA>
 41
```

3.1.3 The Noise clustering model

In the previous two models, all objects were assigned, either completely to one cluster or with their membership divided among clusters (in other words, we accepted the partition restriction). This may be appropriate in many instances, but it may cause problems if some of the plant communities sampled include rare species assemblages. These plant communities would more appropriately be classified as ‘outliers’ and should not influence the cluster prototypes. In the noise clustering (NC) model we allow outlier objects to be excluded from the classification:

```
> wetland.nc = vegclust(x = wetlandchord, mobileCenters=3,
+                       method="NC", m=1.2, dnoise=0.8, nstart=20)
> round(t(wetland.nc$memb), dig=2)
```

```

      5      8 13 4   17      3      9 21   16   14 2    15 1    7   10
M1 0.00 0.00  0 0 0.00 0.01 0.01  0 0.00 0.00 0 0.00 0 0.01 0.01
M2 0.99 0.85  1 1 0.93 0.41 0.08  1 0.86 0.99 1 0.93 1 0.49 0.12
M3 0.00 0.00  0 0 0.00 0.01 0.01  0 0.00 0.00 0 0.00 0 0.01 0.01
N   0.01 0.14  0 0 0.07 0.58 0.90  0 0.13 0.01 0 0.07 0 0.49 0.86
      40   23   25   22   20 6 18 12   39 19   11   30   34   28   31
M1  1 0.93 0.98 0.39 0.98 1  1  1 0.99  1 0.93 0.01 0.00 0.02 0.00
M2  0 0.01 0.01 0.50 0.00 0  0  0 0.00  0 0.00 0.01 0.00 0.01 0.00
M3  0 0.00 0.00 0.00 0.00 0  0  0 0.00  0 0.00 0.07 0.98 0.08 0.98
N   0 0.06 0.01 0.11 0.02 0  0  0 0.01  0 0.06 0.90 0.02 0.89 0.02
      26   29   33 24 36 37   41   27   32   35   38
M1 0.28 0.34 0.99  0  0  0 0.00 0.01 0.02 0.03 0.00
M2 0.01 0.01 0.00  0  0  0 0.01 0.01 0.01 0.48 0.90
M3 0.03 0.08 0.00  1  1  1 0.93 0.05 0.11 0.02 0.00
N   0.68 0.58 0.01  0  0  0 0.06 0.92 0.85 0.47 0.09

```

As with FCM, some objects have intermediate memberships. In addition, there are some objects with high membership to the Noise class, which indicates that they are distant from all ‘true’ cluster centers. These objects can be considered ‘outliers’ and remain unclassified:

```

> groups = defuzzify(wetland.nc)$cluster
> groups

```

```

      5      8   13      4   17      3      9   21   16   14      2   15      1
"M2" "M2" "M2" "M2" "M2" "N"  "N" "M2" "M2" "M2" "M2" "M2" "M2"
      7   10   40   23   25   22   20      6   18   12   39   19   11
"N"  "N"  "M1" "M1" "M1" "M2" "M1" "M1" "M1" "M1" "M1" "M1" "M1"
      30   34   28   31   26   29   33   24   36   37   41   27   32
"N"  "M3"  "N"  "M3"  "N"  "N"  "M1" "M3" "M3" "M3" "M3"  "N"  "N"
      35   38
"M2" "M2"

```

```

> table(groups)

```

```

groups
M1 M2 M3  N
11 14  6 10

```

Note that we can defuzzify the membership matrix using a threshold, as before, and identify both intermediates (‘NA’) and outliers (members of the noise class: ‘N’):

```

> groups = defuzzify(wetland.nc, method="cut", alpha=0.8)$cluster
> groups

```



```

      5      8     13      4     17      3      9     21     16     14      2     15      1
"M2" "M2" "M2" "M2" "M2"  NA  "N" "M2" "M2" "M2" "M2" "M2" "M2"
      7     10     40     23     25     22     20      6     18     12     39     19     11
  NA  "N"  "M1" "M1" "M1"  NA  "M1" "M1" "M1" "M1" "M1" "M1" "M1"
     30     34     28     31     26     29     33     24     36     37     41     27     32
"N"  "M3"  "N"  "M3"   NA   NA  "M1" "M3" "M3" "M3" "M3" "M3"  "N"  "N"
     35     38
  NA  "M2"

```

```
> table(groups, useNA = "always")
```

```

groups
  M1  M2  M3   N <NA>
  11  12   6   6   6

```

In vegetation classification, distinguishing between an intermediate or an outlier will not always be clearcut. Nevertheless, the distinction may be useful in practice because outlier objects may relate to vegetation patterns that exist in the study area but happen to be underrepresented in the sample. That is, outlier plant communities may be rare for the given vegetation data set only, in the sense that if new data were added they would belong to a vegetation type. Alternatively, they may represent rare species assemblages for the study area. Distinguishing between one case or the other cannot be done without collecting more data [Wiser and De Cáceres, 2012].

An advantage of the NC model over FCM or KM is that ‘outliers’ do not influence the cluster centers. As a result, the cluster centers are more separated from each other than in the previous models. Compare the following distance matrices between cluster centroids:

```
> dist(wetland.km$mobileCenters)
```

```

      M1      M2
M2 0.9248370
M3 0.9567205 0.8997438

```

```
> dist(wetland.fcm$mobileCenters)
```

```

      M1      M2
M2 0.9387557
M3 0.9280444 0.8852202

```

```
> dist(wetland.nc$mobileCenters)
```

```

      M1      M2
M2 0.9735266
M3 1.0102707 1.0666958

```

However, this particular advantage can also be obtained (in partitive methods) if medoids are used as prototypes instead of centroids (see below).

3.1.4 Medoid-based clustering

All the examples that we have shown so far could be repeated using medoids as cluster prototypes instead of centroids. For example, with the K-medoids (the K-means analogue) would be:

```
> wetland.kmdd = vegclust(x = wetlandchord, mobileCenters=3,
+                          method="KMdd", nstart=20)
> t(wetland.kmdd$memb)
```

	5	8	13	4	17	3	9	21	16	14	2	15	1	7	10	40	23	25	22	20	6	18	12	39
M1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1	1
M3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0

	19	11	30	34	28	31	26	29	33	24	36	37	41	27	32	35	38
M1	0	0	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0
M2	1	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
M3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

When ran using a site-by-species matrix as input, `vegclust` returns the coordinates of medoids as the cluster centers:

```
> round(wetland.kmdd$mobileCenters, dig=3)
```

	Abefic	Merhed	Alyvag	Pancam	Abemos	Melcor	Ludoct	Eupvac	Echpas
M1	0	0	0	0.258	0	0.258	0	0.258	0
M2	0	0	0	0.000	0	0.177	0	0.000	0
M3	0	0	0	0.000	0	0.183	0	0.000	0

	Passcr	Poa2	Carhal	Dendio	Casobt	Aesind	Cyprot	Ipocop	Cynarc
M1	0.258	0	0.000	0	0	0	0	0	0.775
M2	0.000	0	0.177	0	0	0	0	0	0.000
M3	0.000	0	0.000	0	0	0	0	0	0.000

	Walind	Sessp.	Phynod	Echell	Helind	Ipoaqu	Orysp.	Elesp.	Psespi
M1	0	0.258	0.258	0.000	0.000	0.000	0.000	0.000	0.000
M2	0	0.000	0.884	0.177	0.177	0.177	0.000	0.177	0.177
M3	0	0.000	0.000	0.000	0.000	0.000	0.913	0.183	0.183

	Ludads	Polatt	Poa1	Helcri	Physp.	Goopur
M1	0.000	0.000	0	0	0	0
M2	0.000	0.000	0	0	0	0
M3	0.183	0.183	0	0	0	0

However, when using site-by-site dissimilarity matrices as input for `vegclustdist`, the indices of objects are returned instead:

```
> wetland.kmdd = vegclustdist(x = dchord, mobileMemb=3,
+                             method="KMdd", nstart=20)
> wetland.kmdd$mobileCenters
```

[1] 34 11 23

3.2 Supervised classification: `as.vegclust` and `vegclass`

Vegetation types are meant to be used. For example, a new area may be surveyed and a map of vegetation types may be needed. Here we simulate the process of assigning new observations to a vegetation classification created *a priori*. In order to simulate this two-step process, we split our example data set into two matrices, one with the 31 objects whose group classification will be known *a priori* and the other with the 10 objects whose classification is to be studied:

```
> wetland.31 = wetlandchord[1:31,]  
> wetland.31 = wetland.31[,colSums(wetland.31)>0]  
> dim(wetland.31)
```

```
[1] 31 27
```

```
> wetland.10 = wetlandchord[-(1:31),]  
> wetland.10 = wetland.10[,colSums(wetland.10)>0]  
> dim(wetland.10)
```

```
[1] 10 22
```

As initial classification, we simply take the two groups resulting from a K-means analysis (using function `kmeans` from the `stats` package) on the first data set:

```
> km = kmeans(wetland.31, 2)  
> groups = km$cluster  
> groups
```

```
 5  8 13  4 17  3  9 21 16 14  2 15  1  7 10 40 23 25 22 20  6 18  
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  
12 39 19 11 30 34 28 31 26  
 2  2  2  2  1  1  1  1  1
```

The idea is to know whether the ten objects of the second data set may be assigned to the vegetation types defined using the first data set. Because our initial classification was not obtained using `vegclust`, we need to transform the input classification of 31 objects into an object of class ‘vegclust’. This is done using function `as.vegclust`:

```
> wetland.31.km = as.vegclust(wetland.31, groups)
```

Note that we did not specify the clustering model for our ‘vegclust’ object. By default, the clustering method is K-means (KM):

```
> wetland.31.km$method
```

```
[1] "KM"
```

In this case, this matches the way we obtained our initial classification. In general, however, we may have a classification obtained following an informal (or unknown) procedure, and we will choose the clustering model according to our preferences for assignments. Once we have our object ‘vegclust’ we can use function `vegclass` to *assign* the second set of observations according to the membership rule of the k-means cluster model:

```
> wetland.10.km = vegclass(wetland.31.km, wetland.10)
> defuzzify(wetland.10.km)$cluster
```

```
29 33 24 36 37 41 27 32 35 38
"1" "2" "1" "1" "1" "1" "1" "1" "2" "2"
```

The same procedure can be repeated using dissimilarity matrices. First, we call `as.vegclust` to create a vegclust object:

```
> wetland.31.km.d = as.vegclust(dist(wetland.31), groups)
```

After that, we need a matrix of distances between the set of observations to be assigned and the original set of 31 observations. This can be obtained from the original distance matrix including all pairs of observations:

```
> wetland.d.10.31 = as.data.frame(as.matrix(dchord)[32:41,1:31])
```

Once we have these two objects we can use function `vegclass` to *assign* the second set of observations according to the membership rule of the k-means cluster model:

```
> wetland.d.11.km = vegclass(wetland.31.km.d, wetland.d.10.31)
> defuzzify(wetland.d.11.km)$cluster
```

```
29 33 24 36 37 41 27 32 35 38
"1" "2" "1" "1" "1" "1" "1" "1" "2" "2"
```

Note that in both cases all the objects of the second set were assigned to the nearest cluster. What if we want to determine whether it is more appropriate for any of the objects to define a new vegetation type? In that case, we may do better by using noise clustering. We can choose, for example, the hard noise clustering (HNC) model:

```
> wetland.31.nc = as.vegclust(wetland.31, groups, method="HNC",
+                             dnoise = 0.8)
> wetland.10.nc = vegclass(wetland.31.nc, wetland.10)
> defuzzify(wetland.10.nc)$cluster
```

```

29 33 24 36 37 41 27 32 35 38
"N" "N" "N" "N" "N" "N" "N" "N" "N" "2"

```

An additional parameter is needed: the distance to the noise class ‘dnoise’, δ . This can be set either conventionally (depending on the level of abstraction of vegetation types) or by relying on the variance of the original clusters [De Cáceres et al., 2006]. The results of the noise clustering model show that several of the ten objects are assigned to the noise class (‘N’), which indicates that some of them could be used to define a new cluster.

3.3 Extending vegetation classifications

Allowing vegetation classifications to be dynamic entities, in the sense that they may be modified or extended as new surveys are conducted (or in general, when new data becomes available) increases their usability [De Cáceres et al., 2010, Wiser and De Cáceres, 2012]. The aim here is to preserve the two prototypes of the initial classification and let `vegclust` to define a new vegetation type.

3.3.1 Conforming vegetation data sets

When new vegetation data becomes available, the set of species in the new data set will not normally be the same as the set of species used in the original classification exercise. In this situation, the practitioner will need to bring the two sets of samples to the same resemblance space. In our example we divided our initial data set into two subsets of 31 and 10 sites, respectively (i.e. `wetland.31` and `wetland.10`). Thus, our original data set had a common set of species. However, we will now pretend that `wetland.31` and `wetland.10` were obtained independently. A useful function to create a common space of species for two sets of vegetation observations is `conformveg`:

```

> cf = conformveg(wetland.31, wetland.10)
> wetland.31.cf<- cf$x
> wetland.10.cf<- cf$y
> dim(wetland.31.cf)

[1] 31 33

> dim(wetland.10.cf)

[1] 10 33

```

The only difference between these new objects and the preceding ones is that they include extra columns with zeros, so that the set of columns (i.e. species) match between the two data sets.

3.3.2 Re-calculating the centroids of the initial classification

Another step we need to conduct before calling `vegclust` is to re-calculate the centroids of the initial classification in the resemblance space of all species. Thus, we need to call `clustcentroid` using the conformed data set as input:

```
> fixed = clustcentroid(wetland.31.cf, groups)
```

Again, these centroids are exactly like the original ones except that they are expressed in a matrix that includes extra zeros for the new species.

3.3.3 Calling `vegclust` with fixed prototypes

At this point we are ready to call `vegclust` with the new data set (again, its conformed version):

```
> wetland.nc = vegclust(wetland.10.cf, mobileCenters=1,
+                       fixedCenters = fixed,
+                       method = wetland.31.nc$method,
+                       dnoise=wetland.31.nc$dnoise, nstart=10)
> defuzzify(wetland.nc)$cluster
```

```
29  33  24  36  37  41  27  32  35  38
"N"  "N" "M1" "M1" "M1" "M1"  "N"  "N"  "N" "F3"
```

Here, the function `vegclust` has renamed the original clusters as ‘F2’ and ‘F3’ (indicating that they are now fixed), while the new cluster is named ‘M1’.

Instead of relying on the Noise clustering model, we could have chosen to use the K-means model to extend the classification:

```
> wetland.km = vegclust(wetland.10.cf, mobileCenters=1,
+                       fixedCenters = fixed,
+                       method = "KM",
+                       nstart=10)
> defuzzify(wetland.km)$cluster
```

```
29  33  24  36  37  41  27  32  35  38
"F2" "F3" "M1" "M1" "M1" "M1" "M1" "M1" "F3" "F3"
```

This avoids having objects in the Noise class. However, note that the noise clustering model allows objects in the Noise to be classified later on when new data becomes available, instead of forcing them to belong to one cluster or the other (i.e. the partition restriction).

3.3.4 Extending or refining classifications?

Instead of calling `vegclust` with the new data set only, we could have chosen to use the set of all vegetation observations:

```
> wetland.nc = vegclust(rbind(wetland.31.cf,wetland.10.cf), mobileCenters=1,
+                        fixedCenters = fixed,
+                        method = wetland.31.nc$method,
+                        dnoise=wetland.31.nc$dnoise, nstart=10)
> defuzzify(wetland.nc)$cluster

   5    8   13    4   17    3    9   21   16   14    2   15    1
"F3" "F3" "F3" "F3" "F3"  "N"  "N" "F3" "F3" "F3" "F3" "F3" "F3"
   7   10   40   23   25   22   20    6   18   12   39   19   11
"F3"  "N" "M1" "M1" "M1" "F3" "M1" "M1" "M1" "M1" "M1" "M1" "M1"
  30   34   28   31   26   29   33   24   36   37   41   27   32
"F2" "F2"  "N" "F2" "F2"  "N" "M1"  "N"  "N"  "N"  "N"  "N"  "N"
  35   38
"N"  "F3"
```

Note that some of the objects in the first data set may have been reassigned (to a different cluster or to the Noise class). While the centroids of the original classification are preserved, the membership of particular objects may change because the classification now includes new prototypes to which objects may be assigned. Using the complete data set instead of the new data implies that the used is ready to accept these reassignments. If the new memberships were strikingly different from the original ones, one might decide to start the three group classification from scratch.

3.3.5 Using `vegclustdist` with fixed prototypes

Extending vegetation classifications can also be done in the distance-based mode. In this case we do not have explicit coordinates for fixed prototypes, but we may have the distances to these centers. To begin our example we use the distances to the two clusters that we obtained at the stage of assigning new observations:

```
> fixedDist = wetland.d.11.km$dist2clusters
```

With this information, we can now call `vegclustdist` to define a new cluster while accounting for the previous ones:

```
> wetland.km.d = vegclustdist(dist(wetland.10), mobileMemb = 1,
+                             fixedDistToCenters=fixedDist,
+                             method = "KM",
+                             nstart=10)
> defuzzify(wetland.km.d)$cluster
```

```

29 33 24 36 37 41 27 32 35 38
"F2" "F3" "M1" "M1" "M1" "M1" "M1" "M1" "F3" "F3"

```

Analogously to `vegclust`, we could have chosen to use the set of all vegetation observations. In this case, we need to pool the distances to the centers:

```

> fixedDist = rbind(wetland.31.km.d$dist2clusters, wetland.d.11.km$dist2clusters)

```

And then we call `vegclustdist` using the full distance matrix:

```

> wetland.km.d = vegclustdist(dchord, mobileMemb = 1,
+                             fixedDistToCenters=fixedDist,
+                             method = "KM",
+                             nstart=10)
> defuzzify(wetland.km.d)$cluster

 5  8 13  4 17  3  9 21 16 14  2 15  1
"F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3" "F3"
 7 10 40 23 25 22 20  6 18 12 39 19 11
"F3" "F3" "M1" "F3" "F3" "F3" "M1" "M1" "M1" "M1" "M1" "M1" "M1"
30 34 28 31 26 29 33 24 36 37 41 27 32
"F2" "F2" "F2" "F2" "F2" "F2" "M1" "F2" "F2" "F2" "F2" "F2" "F2"
35 38
"F3" "F3"

```

As before, some objects may be reclassified in different clusters when using this option.

4 Cluster characterization

In this section we show how to use auxiliary functions that allow cluster properties to be extracted from an input classification. In all examples we will use this *a priori* classification of our data set:

```

> groups = c(rep(1, 17), rep(2, 14), rep(3,10))

```

4.1 Cluster prototypes: `clustcentroid` and `clustmedoid`

Functions `clustcentroid` and `clustmedoid` allow the cluster prototypes to be calculated according to an input classification structure, which can be specified using either a cluster vector or a membership matrix.

For example, with `clustcentroid` we can calculate the coordinates of the centroids of the initial groups using eq. 1:

```

> centroids = clustcentroid(wetlandchord, groups)
> round(centroids, dig=3)

```


	Abefic	Merhed	Alyvag	Pancam	Abemos	Melcor	Ludoct	Eupvac	Echpas
1	0.000	0.000	0.009	0.009	0.000	0.054	0.000	0.020	0.031
2	0.013	0.015	0.036	0.088	0.000	0.178	0.000	0.090	0.116
3	0.000	0.047	0.000	0.371	0.036	0.253	0.019	0.182	0.061
	Passcr	Poa2	Carhal	Dendio	Casobt	Aesind	Cyprot	Ipocop	Cynarc
1	0.000	0.00	0.064	0.000	0.016	0.009	0.000	0.000	0.000
2	0.000	0.01	0.036	0.000	0.000	0.064	0.179	0.028	0.106
3	0.026	0.00	0.000	0.027	0.000	0.019	0.039	0.018	0.301
	Walind	Sessp.	Phynod	Echell	Helind	Ipoaqu	Orysp.	Elesp.	Psespi
1	0.068	0.144	0.090	0.022	0.112	0.055	0.609	0.164	0.192
2	0.055	0.117	0.506	0.119	0.138	0.048	0.126	0.125	0.027
3	0.090	0.138	0.204	0.158	0.090	0.037	0.172	0.000	0.000
	Ludads	Polatt	Poa1	Helcri	Physp.	Goopur			
1	0.204	0.097	0.000	0.000	0.00	0.00			
2	0.000	0.000	0.013	0.013	0.00	0.00			
3	0.000	0.000	0.000	0.000	0.02	0.02			

As medoids are prototypes that are chosen among the objects to be classified, function `clustmedoid` does not return coordinates but the indices of objects. The following code uses eq. 3 to determine the medoids of each cluster:

```
> medoids = clustmedoid(wetlandchord, groups)
> medoids

2 12 41
11 23 37
```

The value returned by function `medoid` is a vector of indices with the corresponding object names (which are numbers in this particular case). If the classification structure is a fuzzy membership matrix, the cluster centroids or medoids are determined using eqs. 2 and 4, respectively.

4.2 Cluster internal variability: `clustvar`

Vegetation types may differ in their internal variability. The function `clustvar` allows the amount of compositional variation (i.e. beta diversity) observed among the sites of sites belonging to each cluster to be determined. For clusters whose prototype is a centroid, this is calculated as the mean of squared distances from each object of the group to the group centroid. For example, the variability for a given group i would be:

$$Var(i) = \sum_{j=1}^{n_k} e_{ij}^2 / n_i \quad (13)$$

Note that division is by n_i and not by $(n_i - 1)$, which would give an unbiased sample estimate. Thus, the variances calculated in `clustvar` are population variances. For example, the variances of the three groups in our examples are:

```
> clustvar(wetlandchord, groups)
```

```
      1      2      3
0.4554668 0.5466579 0.5293836
```

The reason why population values are produced, instead of sample estimates, is because it allows the variance to be calculated using fuzzy membership values:

$$Var(i) = \frac{\sum_{j=1}^n u_{ij}^m e_{ij}^2}{\sum_{j=1}^n u_{ij}^m} \quad (14)$$

Cluster variances can also be obtained using distance or dissimilarity matrices. In this case, the variance for a given group is calculated as:

$$Var(i) = \frac{1}{n_i^2} \sum_{k=1}^{n_i} \sum_{l=1}^{n_i} d_{kl}^2 \quad (15)$$

Again, division by $n_i(n_i - 1)$ instead of n_i^2 would give an unbiased variance estimate. Because in our example the community data had been transformed using the chord transformation, the same variance values can be obtained using a distance matrix with chord distances:

```
> clustvar(dchord, groups)
```

```
      1      2      3
0.4554668 0.5466579 0.5293836
```

Finally, if no classification structure is provided function `clustvar` will return the overall variance (beta diversity) of the data table:

```
> clustvar(wetlandchord)
```

```
[1] 0.6751038
```

4.3 Distance between clusters: `interclustdist`

Calculating distance between pairs of cluster prototypes is useful to determine which vegetation types are more similar and which are more distinct. When prototypes of vegetation types are chosen to be cluster medoids, then the resemblance between vegetation types can be defined as the resemblance between the corresponding medoids:

```
> as.dist(as.matrix(dchord)[medoids,medoids])
```

```

      2      12
12 1.344006
41 1.093926 1.363297
```

In contrast, if prototypes of vegetation types are chosen to be cluster centroids, the distance between two vegetation types should be defined as the distance between the cluster centroids. Following our example, we can simply use function `dist` on the matrix of centroid coordinates:

```
> dist(centroids)
```

```

      1      2
2 0.7624211
3 0.8004329 0.5298008
```

Alternatively, the function `interclustdist` allows the distance between pairs of centroids to be calculated without the coordinates of centroids being supplied. Instead, the matrix of distances between objects is used. For example, if the distance between the centroids of groups i and j is desired, we can calculate the squared distance by:

$$d^2(i, j) = \frac{\sum_{k=1}^n \sum_{l=1}^n u_{ik}^m u_{jl}^m d_{kl}^2}{\sum_{k=1}^n u_{ik}^m \sum_{l=1}^n u_{jl}^m} - Var(i) - Var(j) \quad (16)$$

Using equation 16 in our example would be:

```
> interclustdist(dchord, groups)
```

```

      1      2
2 0.7624211
3 0.8004329 0.5298008
```

which returns the same values as before.

4.4 Constancy classes: `clustconst`

One way of characterizing a vegetation type is to examine how frequently each species occurs in vegetation observations belonging to the type. This frequency is often called *constancy* and the table that contains the constancy of all species in all vegetation types is called *constancy table*. The function `clustconst` allows this table to be calculated:

```
> c = clustconst(wetlandchord, memb = as.memb(groups))
```

The R object returned by `clustconst` can be examined in several ways. First, it is useful to print the constancy table ordering species from high to low constancy for each cluster:

```
> d=summary(c, mode="all")

----- 3 -----
          3      2      1
Pancam  1.000  0.357  0.059
Melcor   1.000  0.786  0.294
Eupvac   0.800  0.357  0.118
Sessp.   0.600  0.500  0.412
Echell   0.600  0.357  0.118
----- 2 -----
          3      2      1
Phynod   0.400  0.714  0.118
Helind   0.300  0.643  0.471
Elesp.   0.000  0.571  0.529
----- 1 -----
          3      2      1
Orysp.   0.300  0.286  1.000
Ludads   0.000  0.000  0.824
```

Alternatively, we can examine the constancy vector of each vegetation type:

```
> summary(c, mode="cluster", name=names(c)[1])

Orysp.  1.000
Ludads  0.824
Elesp.  0.529
```

References

- James C Bezdek. *Pattern recognition with fuzzy objective functions*. Plenum Press, New York, 1981.
- D.M.J.S. Bowman and B A Wilson. Wetland vegetation pattern on the Adelaide River flood plain, Northern Territory, Australia. *Proceedings of the Royal Society of Queensland*, 97:69–77, 1986.
- MB Dale. Evaluating classification strategies. *Journal of Vegetation Science*, 6(3):437–440, 1995. ISSN 1654-1103. URL <http://onlinelibrary.wiley.com/doi/10.2307/3236243/abstract>.
- R N Dave. Characterization and detection of noise in clustering. *Pattern Recognition Letters*, 12(11):657–664, 1991.

- R.N. Dave and R. Krishnapuram. Robust clustering methods: a unified view. *IEEE Transactions on Fuzzy Systems*, 5(2):270–293, May 1997. ISSN 10636706. doi: 10.1109/91.580801. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=580801>.
- Miquel De Cáceres, Francesc Oliva, and Xavier Font. On relational possibilistic clustering. *Pattern Recognition*, 39(11):2010–2024, November 2006. ISSN 00313203. doi: 10.1016/j.patcog.2006.04.008. URL <http://linkinghub.elsevier.com/retrieve/pii/S0031320306001634>.
- Miquel De Cáceres, Xavier Font, and Francesc Oliva. The management of vegetation classifications with fuzzy clustering. *Journal of Vegetation Science*, 21(6):1138–1151, December 2010. ISSN 11009233. doi: 10.1111/j.1654-1103.2010.01211.x. URL <http://doi.wiley.com/10.1111/j.1654-1103.2010.01211.x>.
- R Hathaway, J.C. Bezdek, and J Davenport. On relational data versions of c-means algorithms. *Pattern Recognition Letters*, 17(6):607–612, May 1996. ISSN 01678655. doi: 10.1016/0167-8655(96)00025-6. URL <http://linkinghub.elsevier.com/retrieve/pii/0167865596000256>.
- R J Hathaway and James C Bezdek. NERF c-means: Non-euclidean relational fuzzy clustering. *Pattern recognition*, 27(3):429–437, 1994.
- R J Hathaway, J W Davenport, and James C Bezdek. Relational duals of the c-means clustering algorithms. *Pattern recognition*, 22(2):205–212, 1989.
- R. Krishnapuram and J.M. Keller. A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, 1(2):98–110, May 1993. ISSN 10636706. doi: 10.1109/91.227387. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=227387>.
- R. Krishnapuram and J.M. Keller. The possibilistic C-means algorithm: insights and recommendations. *IEEE Transactions on Fuzzy Systems*, 4(3):385–393, 1996. ISSN 10636706. doi: 10.1109/91.531779. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=531779>.
- Raghu Krishnapuram, Anupam Joshi, and Liyu Yi. A Fuzzy relative of the k-medoids algorithm with application to web document and snippet clustering. In *IEEE International Fuzzy Systems*, pages 1281–1286, 1999. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=790086.
- Pierre Legendre and Eugene Gallagher. Ecologically meaningful transformations for ordination of species data. *Oecologia*, 129(2):271–280, October 2001. ISSN 0029-8549. doi: 10.1007/s004420100716. URL <http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s004420100716>

- J MacQueen. Some methods for classification and analysis of multivariate observation. Number 0, pages 281–297. University of California Press, Berkeley, 1967.
- J. Moravec. Influences of the individualistic concept of vegetation on syntaxonomy. *Vegetatio*, 81(1-2):29–39, July 1989. ISSN 0042-3106. doi: 10.1007/BF00045511. URL <http://www.springerlink.com/index/10.1007/BF00045511>.
- L Orlóci. An agglomerative method for classification of plant communities. *Journal of Ecology*, 55(0):193–206, 1967.
- Susan K. Wiser and Miquel De Cáceres. Updating vegetation classifications: an example with New Zealand’s woody vegetation. *Journal of Vegetation Science*, pages n/a–n/a, July 2012. ISSN 11009233. doi: 10.1111/j.1654-1103.2012.01450.x. URL <http://doi.wiley.com/10.1111/j.1654-1103.2012.01450.x>.