Chapter 4
Functional Diversity

4.1 Objectives

The objectives of this chapter are to explore the variety of metrics and approaches for analyzing the functional composition and diversity of species assemblages. Important topics will include the consideration of how uni- and multivariate trait data are utilized in functional diversity analyses, the use of raw trait distance matrices versus trait dendrograms, and the degree of similarity between functional diversity metrics.

4.2 Background

The number of articles in ecology that are taking a "trait-based" approach is currently exploding with many of these articles seeking to quantify the functional diversity of the species in a community or assemblage (e.g., [73–92]). As we will see shortly, functional diversity can be quantified in a number of ways, but we can coarsely define it here as the diversity or dissimilarity of the ecological strategies or performance of species upon the basis of their morphological physiological traits. Traits directly or indirectly correlated with species performance (i.e., growth, mortality, reproduction) are increasingly termed "functional traits" and I will tend to use that nomenclature in the following text.

Despite the recent surge in interest, the measurement of functional diversity in communities or assemblages dates back at least 50 years with some of the most interesting early examples investigating the volume and packing of trait space of species in assemblages spanning an environmental or richness gradient [94–97]. For example, early work by Ricklefs and colleagues [93, 94] investigated whether the

The online version of this chapter (doi: 10.1007/978-1-4614-9542-0_4) contains supplementary material, which is available to authorized users

DOI 10.1007/978-1-4614-9542-0_4,

spacing of species in trait space was maintained across a species richness as expected by limiting similarity theory [39].

The recent explosion of functional diversity measurement in ecology can largely be traced to influential work linking functional diversity to ecosystem function (e.g., [11, 91, 98]) and detailed reviews formalizing the conceptual foundation for functional diversity research (e.g., [76]). This work also coincided with the focus of plant ecologists on identifying a series of plant traits that are believed to be related to plant performance across environmental gradients [99–102]. Thus, in the span of two decades plant ecologists, in particular, found themselves collecting large and detailed datasets of plant traits in assemblages and developing and implementing measures of functional diversity. In many instances this has unfortunately resulted in the "reinvention" of metrics that were originally developed decades earlier and presumably forgotten or ignored. The goal of the present chapter is to discuss the main approaches for measuring functional diversity in communities and assemblages and how to calculate these metrics using R. We will not cover every possible metric of functional diversity ever published. Such an approach would be difficult and would result in the covering of many redundant measures. I have therefore chosen to keep it simple by covering the main classes of functional diversity metrics that are flexible. From each of these classes I will provide one or a few example metrics that are likely monotonic with many of the metrics you will encounter in the literature. As in the other chapters we will not simply learn how to "plug and chug" using existing functions. Rather we will dissect the measures so we understand how they work and so you may learn how you may construct your own measures of functional diversity or adjust those measures covered presently.

4.3 Quantifying the Functional Composition of Communities Using the Moments of Trait Distributions

As we will see in the following subsections, a great number of metrics have been generated for quantifying the functional diversity of communities with, often redundant, metrics invented nearly monthly at this point in time. It can be difficult to decipher what the results of various metric actually mean and how they do or do not relate to results published using alternative metrics. Outside of this increasingly complex maze of functional diversity metrics are simple calculations of the four moments of the trait distribution within communities or assemblages. The four moments—mean, standard deviation, skew, and kurtosis—are easier to interpret for the average scientist and can therefore be the easiest way to begin understanding the functional composition of your study system. They may even be an effective method for detecting the imprint of deterministic community assembly processes (e.g., [77, 84, 103]). For example, the co-occurrence of functionally similar species will be reflected by a lower standard deviation and a higher kurtosis of trait values in an assemblage [77].

In this section we will compute the four moments of the trait distribution in assemblages weighting all species present equally. We will then compute the community-weighted mean trait value for communities. The community-weighted mean is simply the mean trait value weighted by the relative abundance of species and it is becoming frequently used in trait-based ecological analyses. We begin by reading in the example community dataset for this chapter, which is in the format of a community data matrix. Recall that the community data matrix is the general format for most of the ecology-specific functions you will encounter in R. The example community data matrix can be read in as a table.

```
> my.sample <- read.table("FD.example.sample.txt",
header = T, row.names = 1)
```

Ensure that your data has been read into R correctly with the community (or site) names as row names, the species (or taxa) names as column names, and number of individuals as the cell values. To do this, take a quick look at the matrix.

```
> my.sample
```

Next we can read in the example trait data for this chapter. The .txt file has species names in the first column followed by four columns of trait data. It is important that the species names become row names when reading in the example or your own trait data into R for the analyses that follow.

```
> traits <- read.table("FD.traits.txt", sep = "\t",
header = T, row.names = 1)
```

Look at the trait data to ensure that the data were read into R correctly and that species names are indeed represented as row names and there are four columns of trait data.

```
> traits
```

The community data and trait data matrices have now been loaded into R and we are now ready to calculate the moments of the trait distributions for communities. To begin we will calculate individual moments for individual communities in order to understand the code. The first step is to quickly review how to get the list of species present in a community or assemblage. For example, we would like to know all of the species in our first community or assemblage with an abundance greater than zero—the first row in our community data matrix.

```
> spp <- names(my.sample[1, my.sample[1,] > 0])
```

The result is an object called "spp" that contains the names of species present in our first community. The trait data for these species can now be extracted from our traits matrix by asking for only those rows with the row names matching the names in our species list.

```
> traits[spp, ]
```

The above allows us to extract the trait data for all species in a community or assemblage for subsequent analyses. The two lines of code can also be combined into one line.

```
> traits[names(my.sample[1, my.sample[1,] > 0]),]
```

We now know how to get the values for all traits in our matrix for all species present in our first community or assemblage. From this output we can calculate the moments of the trait distribution. We can first calculate the community mean for each trait by wrapping the above line of code with the function `mean()`.

```
> mean(traits[names(my.sample[1, my.sample[1,] >0), ],
na.rm = T)
```

The `na.rm = T` argument was utilized here in case your trait data matrix had missing trait values for some species. We can calculate the mean trait values for other communities or assemblages by altering what row is being selected from the community data matrix. For example, the next line of code calculates the mean trait values for community 3 instead of community 1.

```
> mean(traits[names(my.sample[3, my.sample[3,] >0), ],
na.rm = T)
```

At this point it is rather easy to calculate the remaining moments of the trait distribution. Here I calculate the standard deviation of the trait values in community 2.

```
> sd(traits[names(my.sample[2, my.sample[2,] >0), ],
na.rm = T)
```

High standard deviation values are indicative of more functional diversity in a community or assemblage, but they may be biased due to differences in the mean from community to community. To reduce this bias, a coefficient of variation in trait values can be calculated by dividing the standard deviations of the trait values in a community by the mean trait values in a community. This is easily done by dividing the last line of code by the line prior to it.

The functions `mean()` and `sd()` are in the *base* R package and are therefore available upon opening R. Functions to calculate skew and kurtosis, on the other hand, are not in the *base* package, but they are available in the *fBasics* package which can be installed and loaded as follows.

```
> install.packages("fBasics", dependencies = T)

> library(fBasics)
```

The skew of the trait distribution for community 1 can now be calculated in a similar way except using the function `skewness()`.

```
> skewness(traits[names(my.sample[1, my.sample[1,]
     >0), ], method = "moment", na.rm = T)
```

High values of skewness do not necessarily imply lower community functional diversity, but do indicate that most co-occurring species tend to have very similar trait values. The kurtosis of the traits in community 1 can be calculated using the `kurtosis()` function.

```
> kurtosis(traits[names(my.sample[1, my.sample[1,]
      > 0), ], method = "moment", na.rm = T)
```

Small kurtosis values indicate community trait distributions with "fatter" tails and therefore may indicate an increase in the average trait disparity between co-occurring species [77].

The above code calculates the moments of the trait distribution for one community, or row in the community data matrix, at a time. To automate this calculation across several communities (i.e., all rows in the community data matrix) we write a function where we calculate the moment for a single community and use an `apply()` function to apply the calculation to all rows (i.e., communities) in our system. In this example, we will write the moments for trait one in each community. We start by writing the kurtosis function for trait one.

```
> kurt.funk <- function(x){

      ## Calculate kurtosis for the first column in the
      ## trait matrix for species that are present in
      ## our community.
      kurtosis(traits[names(x[x > 0]), 1], method =
      "moment", na.rm = T)
}
```

The above code for calculating the kurtosis of the first trait in all communities can be easily changed to calculate the skewness and the mean and standard deviation as follows.

```
> skew.funk <- function(x){

      skewness(traits[names(x[x >0]), 1], method =
      "moment", na.rm = T)

}


> mean.funk <- function(x){

      mean(traits[names(x[x >0]), 1], na.rm = T)

}

> sd.funk <- function(x){

      sd(traits[names(x[x >0]), 1], na.rm = T)

}
```

These functions can now be applied to our community data matrix to calculate the mean, standard deviation, skew, and kurtosis of the trait values for the species present in each of our communities using the `apply()` function.

```
> apply(my.sample, MARGIN = 1, mean.funk)

> apply(my.sample, MARGIN = 1, sd.funk)

> apply(my.sample, MARGIN = 1, skew.funk)

> apply(my.sample, MARGIN = 1, kurt.funk)
```

In the above we have calculated the moments of the trait distributions in communities weighting all species or taxa present in the community equally. In many ecological analyses it is best not to treat all co-occurring species equally. Depending on the question of interest, it may be best to weight species by their abundances or some other measure of their dominance (e.g., percent canopy cover). One of the most common ways this is done when characterizing the functional composition of communities is to calculate what is has been termed the community-weighted mean (CWM) for a trait. The CWM is the mean trait value weighted by the relative abundance of each species. Here we will calculate the CWM for our communities using the same example datasets as before for comparison.

The first step in calculating the CWM is to transform our community data matrix from a tally of individuals of each species in a community into their relative abundances. These relative abundances can then be used to weight the mean trait value in a community. A rapid way to calculate the relative abundances of each species in each community is to divide the abundance of each species in a row (i.e., a community), by the total abundance in that row (i.e., a community). This can be accomplished using the function `rowSums()`.

```
> my.ra.sample <- my.sample / rowSums(my.sample)
```

All values in the cells of the matrix should now represent the fraction of the individuals in a community that for a particular species.

```
> my.ra.sample
```

It is possible that your data, for example, could contain the total canopy area or total biomass. In that instance the above code would provide you with percent cover or percent biomass. The R package *vegan* contains a function that can also be utilized to convert your community data matrix values into relative abundances, percent cover, or percent biomass. The function is called `decostand()` and can be used as follows.

```
> library(vegan)

> my.ra.sample2 <- decostand(my.sample, method =
"total", MARGIN = 1)
```

This function takes a community data matrix as input and can transform the matrix using several different methods applied to rows or columns. Here we have selected the "total" method and MARGIN = 1, indicating that we want the method applied to each row. The method "total" divides each value in a row by the row sum or row total. Although this is what we did previously simply using the `rowSums()` function, the `decostand()` function is useful to know as many methods can be invoked easily. These methods include transforming the matrix to presence/absence (method = "pa"), standardizing the values to a mean of zero and unit variance (method = "standardize"), standardizing the values to range between zero and one (method = "range"), and dividing the values by the maximum value in the row or column (method = "max"). Thus, it is a powerful tool for altering the format of your community data matrix in a variety of ways. Now let us quickly check to see that our results using `rowSums()` are similar to that we received using `decostand()`.

```
> my.ra.sample2
```

You will find that the results of this approach and the previous approach are identical. To check that our relative abundances or percent cover or biomass for each community does indeed sum to one we can check the sum of each row.

```
> rowSums(my.ra.sample2)
```

Now that we have successfully changed our community data matrix to represent the relative abundances of species we can easily calculate the CWM for a single trait for a single community using the `weighted.mean()` function. This function takes an input matrix of values and an input matrix of weights. The input matrix of values in this case will be the trait values for the first trait of all species in our study system sorted using the order of the column names in the community data matrix.

```
> weighted.mean(traits[colnames(my.ra.sample), 1] ,
my.ra.sample[1, ])
```

The above selected the first column of the `traits` object to give the CWM of the first trait. The weighted mean of the second trait could be calculated by changing the selected column to 2.

```
> weighted.mean(traits[colnames(my.ra.sample), 2] ,
my.ra.sample[1, ])
```

To calculate the CWM value for the second trait in community two, simply change the row selected from the `my.ra.sample` object to 2.

```
> weighted.mean(traits[colnames(my.ra.sample), 2] ,
my.ra.sample[2, ])
```

Because the above code calculates the weighted mean for one trait in one community at a time, it is best that we write some code that will automatically calculate the CWM for a trait in all communities. Thus, we would like to apply our code to all

communities simultaneously and will start by writing a function to calculate the weighted mean of trait one.

```
> weight.mean.funk <- function(x){
      weighted.mean(traits[names(x[x>0]), 1] , x[x > 0])
}
```

Now we can apply this function to our community data matrix to calculate the CWM for trait one in each community.

```
> apply(my.sample, MARGIN = 1, weight.mean.funk)
```

This approach can be extended to a weighted standard deviation as well using the `wt.sd()` function in the *SDMtools* package.

```
> library(SDMTools)
> weight.sd.funk <- function(x){
      wt.sd(traits[names(x[x>0]), 1] , x[x>0])
}
> apply(my.sample, MARGIN = 1, weight.sd.funk)
```

We have now see how to calculate the moments of the trait distribution for individual traits and individual communities. Some of these moments such as the mean cannot be deemed a measure of trait diversity. Indeed the mean is the exact opposite. Further, we may be interested in multivariate analyses of the function in a community. In the next sections we will discuss more detailed metrics designed to measure FD using one to many traits.

4.4 Dendrogram-Based Versus Euclidean Distance-Based Measures of Functional Diversity

The metrics for FD that we will discuss in the following all rely on a branch length or Euclidean distance to be measured between species. The branch length information comes from a dendrogram generated with a method of hierarchical clustering with a Euclidean trait distance matrix as input. This involves the clustering of species in trait space and may remove fine-scale trait differentiation between species. An alternative approach to this is to simply use the original trait distance matrix in the FD calculations. The use of the original distance matrix is appealing since the data are not transformed, but dendrograms are still frequently used to calculate FD and are sometimes preferred because their data structure is similar to

that of a phylogeny which may be used in a concurrent analysis of phylogenetic diversity to which the FD analysis will be compared. Thus we will discuss both approaches presently.

4.4.1 Generating Trait Distance Matrices

The majority of the functional diversity metrics currently utilized in the literature are distance-based measures. One to many traits can be used in these metrics. The distances themselves are calculated either using the Euclidean distance between species in "trait space" or the branch lengths separating species on a dendrogram generated by clustering species based on their proximity in a trait distance matrix. In this subsection we will be focus on the calculation of the Euclidean distance between species in trait space.

First we will produce a distance matrix for the second trait in our trait matrix. To assure that we have species names on the output distance matrix we first generate a matrix containing the data for our second trait and with the row names from the original trait matrix.

```
> trait.2 <- as.matrix(traits[,2])
> rownames(trait.2) = rownames(traits)
> trait.2
```

Using this new object containing only the data for trait 2 we can calculate the Euclidean distance between all species upon the basis of this single trait. This is accomplished using the `dist()` function and the "euclidean" method.

```
> my.dist.mat.2 <- dist(trait.2, method = "euclidean")
```

As we will see below this distance matrix can be used in many of the existing metrics of functional diversity available in R. Although many studies will only analyze the functional diversity of an assemblage using a distance matrix constructed from multiple traits, it is generally useful to investigate the diversity of individual traits in an assemblage. Indeed the majority of the papers that have investigated the diversity of individual traits have found that not all traits behave similarly across assemblages (e.g., [64, 74, 75, 77, 79]). That is, one trait might increase in diversity along an environmental gradient while another may decrease. Thus, analyzing individual traits may help "unpack" the overall functional diversity calculated from multiple traits.

A distance matrix can be generated using all trait data simultaneously also using the `dist()` function. In this case, the `traits` input matrix already has species names as row names and therefore does not need any transformation.

```
> dist(traits, method = "euclidean")
```

While it is simple to compute the Euclidean distance between all pairs of species based on multiple traits, this approach is generally not advised. This is because it is likely that the measured traits may covary and may be measured on vastly different scales. For example, it is common in plant functional ecology to measure many covarying leaf traits and one or a few orthogonal stem or seed traits. If, for example, a distance matrix were computed on these raw data the leaf traits, which effectively only represent one axis of function, would dominate the structure of the distance matrix. Further, if a trait is measured, such as seed mass, that spans several orders of magnitude, that trait will dominate the structure of the distance matrix. Both of these scenarios are undesirable. Thus, it is often best to first transform your data to approximate a normal distribution for each trait. Then the data should be scaled and used in a principal components analysis to eliminate trait redundancy. We will make our data are approximately normal by using a `log()` in this example and we will scale the data using the `scale()` and `apply()` functions.

```
> traits.scaled <- apply(log(traits), MARGIN = 2,
scale)
```

The result is that all the trait values in each column are scaled to approximately a mean of zero and unit variance. These data can now be used in a principal components analysis to eliminate trait redundancy and to produce orthogonal axes of function that can be used in a distance matrix calculation. The scaled trait data can be input into the `princomp()` function as follows.

```
> pc = princomp(traits.scaled)
```

The principal components (PC) analysis will produce one PC axis for each input column (i.e., trait). If all PC axes were then used as input in a distance matrix calculation the result would be no different than if the PC analysis was not performed at all. Thus, we must select the few axes that explain the vast majority of the variance in the scaled trait data. We can examine the proportion of the total variance explained by each axis by examining a summary of the object output by the `princomp()` function.

```
> summary(pc)
```

A good rule of thumb is to include the PC axes that explain over 90 % and perhaps even 95 % of the variation in the scaled trait data. In this example, the first three PC axes explain 94.7 % of the variation. To determine what traits are most heavily weighted on these axes we can examine the trait loadings.

```
> print(pc$loadings, cutoff = 0.001)
```

From this we can see that trait 1 and trait 2 most heavily influence the first PC axes and traits 4 and 5 most heavily influence the second and third PC axes, respectively.

The next objective is to extract where each species lands on the first three PC axes and to use this information to generate a distance matrix. First we can simply look at these values by printing the PC scores to the screen.

```
> print(pc$scores, cutoff = 0.001)
```

We see one column for each PC axis and one row per species in the same order as the rows names (i.e., species names) in the input scaled distance matrix. As we are only using the first three PC axes we will extract the PC scores from those axes and put them in a new matrix assigning row names from the original `traits` matrix.

```
> pc.scores <- pc$scores[,1:3]
> rownames(pc.scores) <- rownames(traits)
> pc.scores
```

This data can now be used in the `dist()` function to calculate the multivariate Euclidean distance between all species in the study system.

```
> pc.dist.mat <- dist(pc.scores, method = "euclidean")
> pc.dist.mat
```

The result of this analysis is a distance matrix that is less likely to be biased by the co-variation of traits in the original dataset and differences in the scale of measurement between traits. It is therefore recommended that this approach be used in most cases for calculating a distance-based functional diversity metric. Of course, calculating the diversity of an actual trait and not PC scores is more intuitive to many. I do not completely discourage such an approach and I find it useful, but the individual trait diversities should not be assumed to be independent. Further, I do not recommend calculating a distance matrix using all raw trait data simultaneously given that many of the traits ecologists measure strongly covary. Similar to raw trait data it is often useful to investigate each individual axis of function. In the case of data that has been normalized, scaled, and run through a PC analysis, scores from individual PC axes can be used as individual "traits" for analysis.

The above assumes that the trait data you are utilizing is continuous data with no values missing for any traits or any species in your system. This is not always the case in ecological studies where rare species can often not be located in the field or an available database and where we are interested in using categorical traits such as growth form along with continuous traits. In instances where we have mixed trait variables (i.e., continuous and categorical) and/or a few missing trait values for a few species an alternative approach is necessary as a `dist()` function will not accommodate this scenario. A Gower Distance, though, can be calculated using the `gowdis()` function in the *FD* package. This function calculates the overall similarity of species or taxa based on Gower [104] and converts this similarity to a

dissimilarity by subtracting it from one. If all traits are equally weighted, then the Gower similarity, for continuous variables, is the difference in values divided by the maximum observed difference for that variable in the dataset and this value is summed across all traits. For binary traits the contrast is either a zero or one for different or same. The Gower Distance for our dataset can be calculated as follows.

```
> library(FD)

> gowdis(traits)
```

Now that we have a trait distance matrix it can be used directly in FD analyses or as input into a hierarchical clustering analysis to generate a trait dendrogram. We will discuss trait dendrogram construction in the next subsection.

4.4.2 Generating Trait Dendrograms

The generation of trait dendrograms is quite simple. Perhaps it is so simple that the researcher often does not understand conceptually and mathematically what is being done in the background calculations. Thus, we will cover the few steps needed to calculate a trait dendrogram in R with some explanation of what is happening particularly with respect to the generation of the distance matrix and the clustering method.

The first step in generating a trait dendrogram is to produce a distance matrix representing the distance between all taxa or species in your system using one to many traits. In this example, we will first use multiple continuous traits stored in a matrix with species names as the row names. The most advisable and easily comprehended method for generating a trait distance matrix for construction of a dendrogram is to calculate the Euclidean distance between all species in trait space. Here we will assume that the traits are not correlated, but if your traits do covary please consult the code in the preceding section regarding reducing data redundancy prior to distance matrix generation.

```
> my.dist <- dist(traits, method = "euclidean")
```

As discussed above there may be instances where you are missing a trait value for a small number of species or you have mixed trait variables (i.e., continuous and categorical). In such instances the use of `dist()` is not possible and it is advisable to calculate a Gower distance using the `gowdis()` function in the *FD* package to produce a distance matrix for the next step. In this example, we will ignore such a situation.

We now use the trait distance matrix to generate a trait dendrogram using hierarchical clustering. The `hclust()` function in R performs hierarchical clustering using several different methods. In the majority of cases published in the literature

an Unweighted Pair Group Method with Arithmetic Mean, commonly referred to as UPGMA, is utilized to generate a trait dendrogram. The UPGMA begins by identifying the two closest species in the trait distance matrix (randomly if there are multiple pairs sharing the smallest distance). A new distance matrix is then calculated between the distance between that cluster and all other species. The two species in this new matrix that are closest to one another form the next cluster and so on until all species and clusters are clustered. The branch lengths in the resulting dendrogram between clusters are calculated using pairwise distances.

$$cluster\,dist = \frac{\sum_i^A \sum_j^B d_{ij}}{A \times B}$$

where there are A species in cluster 1 and B species in cluster 2 and d_{ij} is the distance between all members of cluster 1 and members of cluster 2. Thus, the distances (i.e., branch lengths) between species in a trait dendrogram are no longer the distances between species in trait space. Rather, they are the distances between all the species in the clusters the species belong to in the dendrogram. A UPGMA-based dendrogram can be calculated using the `hclust()` function in R and the "average" method (Fig. 4.1).

```
> my.dendro <- hclust(my.dist, method = "average")

> plot(my.dendro)
```

We can see that the plotted dendrogram has branch lengths measured on a continuous scale corresponding to multivariate distance in trait space. In many instances it might be desirable to investigate a single trait with dendrogram-based metrics. Generating a dendrogram from a single trait (i.e., a single column in your trait matrix) is simple, but it is important to assign species names as row names. If they are not assigned the resulting dendrogram will not have species names on it. Here we will make a UPGMA dendrogram for the trait found in the second column of our trait matrix (Fig. 4.2).

```
> trait.2 <- as.matrix(traits[,2])

> rownames(trait.2) <- rownames(traits)

> my.dendro.2 <- hclust(dist(trait.2, method =
"euclidean"), method = "average")

> plot(my.dendro.2)
```

The plotted dendrogram shows that species do not cluster in the same way as they did previously when using all traits simultaneously. This indicates that species do not rank similarly on all trait axes and it highlights why it is often important and interesting to perform all analyses on all traits at once and each trait individually.

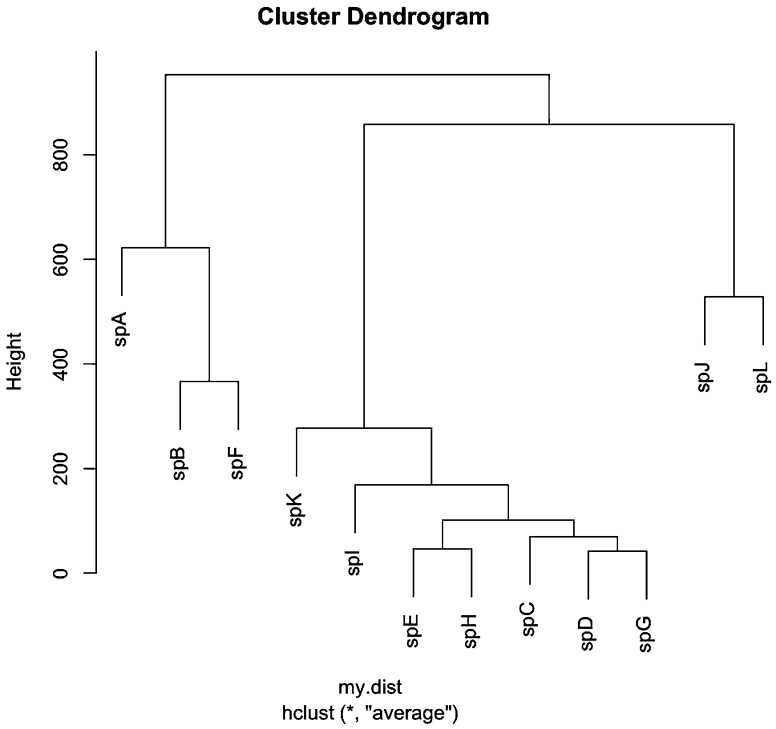

Fig. 4.1 A plot of our functional trait dendrogram constructed using UPGMA hierarchical clustering of a Euclidean distance matrix of all traits

4.4.3 Pairwise and Nearest Neighbor Measures

We are now ready to calculate the main two classes of FD metrics. Exactly as is the case for phylogenetic diversity (PD) in the previous chapter, the two main classes are pairwise distance metrics and nearest neighbor metrics. Indeed, the calculations and code are generally identical in R. The main decision we have to make prior to calculating these metrics for FD is whether to use a dendrogram or a raw trait distance matrix as the following code in this section requires a distance matrix that can be generated from the dendrogram or is the raw distance matrix. We will not use the dendrogram for the following analyses, but if we did we would first convert the dendrogram to a distance matrix.

```
> dendro.dist.mat <- cophenetic(my.dendro)
```

Thus we have effectively input a trait distance matrix into a hierarchical clustering algorithm to generate a dendrogram only to convert that dendrogram into a new distance matrix that likely has less refined information. You can now perhaps see why many do not like using a dendrogram-based approach unless necessary.

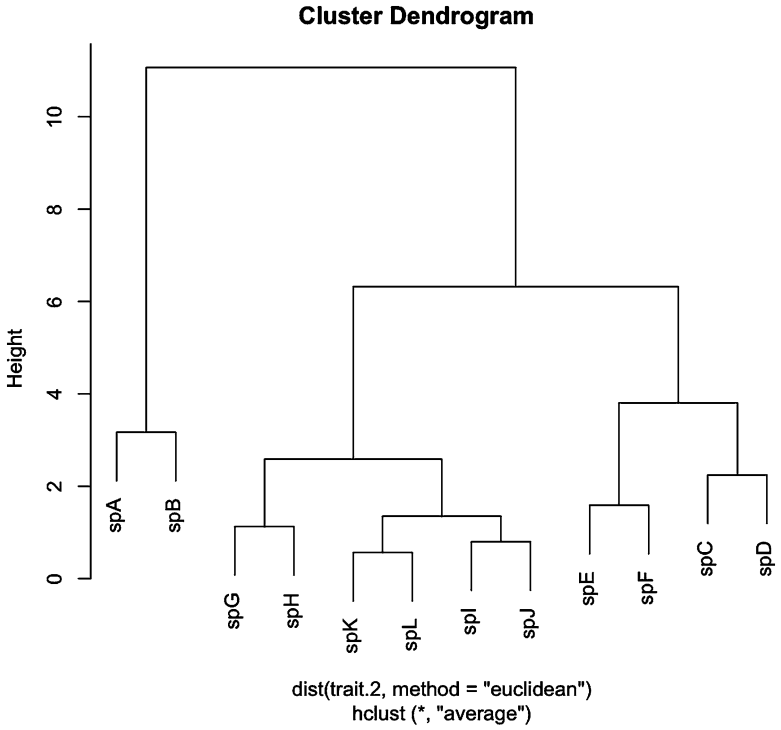

Fig. 4.2 A plot of our functional trait dendrogram for only our second trait constructed using UPGMA hierarchical clustering of a Euclidean distance matrix calculated for the second trait

For the remaining examples in this subsection we will use the raw trait distance matrix. If you recall above, we used `dist()` on our PC axes and we will now make this output a square matrix for the remaining functions.

```
> square.dist.mat <- as.matrix(pc.dist.mat)
```

The first FD measure we will compute is the unweighted pairwise functional distance between all present species in our second community. This can be calculated simply by first subsampling our distance matrix to only include present species and then calculating a mean value.

```
> mean(as.dist(square.dist.mat[names(my.sample[2,
my.sample[2,] > 0), names(my.sample[2, my.sample[2,] >
0)])]))
```

This can be calculated across all communities by first generating a small function to calculate the unweighted mean pairwise distance.

```
> trait.pw <- function(x){

     mean(as.dist(square.dist.mat[names(x[x > 0]),
     names(x[x > 0])]))

}
```

This function can now be applied to all rows (i.e., communities) in our community data matrix using the `apply()` function to compute the metric.

```
> apply(my.sample, MARGIN = 1, trait.pw)
```

This same calculation can be calculated using the `mpd()` function in the *picante* package which loops through communities rather than use an `apply()` making it computationally slower.

```
> mpd(my.sample, square.dist.mat, abundance.weighted =
F)
```

The abundance-weighted version of the pairwise trait distance metric can be calculated by again applying a function we will write. The function can be written as follows.

```
>trait.pw.prime <- function(x){

      ## Get the names of the species in the community.
      com.names <- names(x[x > 0])

      ## Make a matrix with one row containing
      ## abundances and names of all present species
      com <- t(as.matrix(x[x > 0]))

      ## Make functional distance matrix for taxa in
      ## community.
      com.dist <- square.dist.mat[com.names,com.names]

      ## Calculate the product of the abundances of all
      ## species in the community.
      abundance.products <- t(as.matrix(com[1,com[1, ] >
      0, drop = F]))%*% as.matrix(com[1,com[1, ] > 0,
      drop = F])

      ## Calculate a mean of the community functional
      ## distance matrix weighted by the products of
      ## all pairwise abundances.
      weighted.mean(com.dist, abundance.products)

}
```

The function is then applied to the rows of our community data matrix to calculate the abundance-weighted mean pairwise functional distance in each of our communities.

```
> apply(my.sample, MARGIN = 1, trait.pw.prime)
```

This metric can also be calculated using the `mpd()` function in the *picante* package.

```
> mpd(my.sample, square.dist.mat, abundance.weighted =
T)
```

The mean pairwise functional distances calculated above represent one major class of FD metric. Indeed, many of the "new" metrics that are published each year are monotonic with the mean pairwise functional distance.

The second major class of functional diversity metric uses nearest functional neighbor distances. There are two main ways that nearest neighbor distances have been utilized in the trait literature. The first, and most common, method is to take the mean nearest neighbor distance. In this calculation, the distance to the nearest functional neighbor in the community that is not your own species is tallied for each species and a mean is taken. This is easy to calculate but first we will place an NA value in the diagonal of our functional distance matrix to ensure we are not counting conspecifics in our nearest neighbor calculations.

```
> new.square.dist <- square.dist.mat

> diag(new.square.dist) <- NA
```

Now we can calculate the mean nearest functional neighbor distance for the species in our third community by first extracting a community-level distance matrix.

```
> com.dist.mat <- new.square.dist[names(my.sample[3,
my.sample[3,] > 0), names(my.sample[3, my.sample[3,] >
0)])])
```

From this distance matrix we can use the `apply()` function to calculate the minimum value in each row (i.e., the nearest neighbor for each species) and take a mean. This is the mean nearest neighbor distance for our community.

```
> mean(apply(com.dist.mat, MARGIN = 1, min, na.rm = T),
na.rm = T)
```

The second way nearest neighbor distances could be utilized in a study of FD is to take a standard deviation of the nearest neighbor values. This gives the researcher an idea of the regularity of the spacing and not just the average distance. A low variance indicates that species are relatively evenly placed in functional space.

```
> sd(apply(com.dist.mat, MARGIN = 1, min, na.rm = T),
na.rm = T)
```

We can apply both of these metrics to all rows (i.e., communities) in our study system by first writing a function for each to calculate the value for a single community. First the mean nearest functional neighbor distance can be coded as follows:

```
> trait.mnn <- function(x){

      ## Get the names of the species present in a
      ## community.
      com.names <- names(x[x > 0])

      ## Make the community functional distance
      ## matrix by extracting those rows and columns
      ## that have species present in our community.
      my.com.dist <- square.dist.mat [com.names,com.names]

      ## Set all diagonal values to NA so that the
      ## zeros for conspecific comparisons do not
      ## interfere with our calculation of nearest
      ## neighbors.
      diag(my.com.dist) <- NA

      ## Use apply() to calculate the minimum value in
      ## each row of the community functional
      ## distance matrix and take a mean of those
      ## values.
      mean(apply(my.com.dist, MARGIN = 1, min),na.rm = T)

}
```

Next, the function for the standard deviation of nearest neighbor distances.

```
> trait.sdnn <- function(x){

      ## Get the names of the species present in a
      ## community.
      com.names <- names(x[x > 0])

      ## Make the community functional distance
      ## matrix by extracting those rows and columns
      ## that have species present in our community.
      my.com.dist <- square.dist.mat [com.names,
      com.names]

      ## Set all diagonal values to NA so that the
      ## zeros for conspecific comparisons do not
      ## interfere with our calculation of nearest
      ## neighbors.
      diag(my.com.dist) <- NA

      ## Use apply() to calculate the minimum value in
      ## each row of the community functional
      ## distance matrix and take a mean of those
      ## values.
      sd(apply(my.com.dist, MARGIN = 1, min),na.rm=T)

}
```

They both can be applied to our dataset as:

```
> apply(my.sample, MARGIN = 1, trait.mnn)

> apply(my.sample, MARGIN = 1, trait.sdnn)
```

Both functions can also be extended to weight the mean or standard deviation by the abundance of the focal species. The functions would be coded and applied as:

```
> trait.mnn.prime <- function(x){

     ## Get the names of the species present in a
     ## community.
     com.names <- names(x[x > 0])

     ## Make the community functional distance
     ## matrix by extracting those rows and columns
     ## that have species present in our community.
     my.com.dist <- square.dist.mat [com.names,com.names]

     ## Set all diagonal values to NA so that the
     ## zeros for conspecific comparisons do not
     ## interfere with our calculation of nearest
     ## neighbors.
     diag(my.com.dist) <- NA

     ## Use apply() to calculate the minimum value in
     ## each row of the community functional
     ## distance matrix and take a mean of those
     ## values.
     weighted.mean(apply(my.com.dist, MARGIN = 1, min),x[x >
     0], na.rm = T)

}
```

Next, the function for the standard deviation of nearest neighbor distances.

```
> trait.sdnn.prime <- function(x){

     ## Get the names of the species present in a
     ## community.
     com.names <- names(x[x > 0])

     ## Make the community functional distance
     ## matrix by extracting those rows and columns
     ## that have species present in our community.
     my.com.dist <- square.dist.mat [com.names,com.names]

     ## Set all diagonal values to NA so that the
     ## zeros for conspecific comparisons do not
     ## interfere with our calculation of nearest
     ## neighbors.
     diag(my.com.dist) <- NA

     ## Use apply() to calculate the minimum value in
     ## each row of the community functional
     ## distance matrix and take a mean of those
     ## values.
     wt.sd(apply(my.com.dist, MARGIN = 1, min), x[x >0],
     na.rm = T)

}

> apply(my.sample, MARGIN = 1, trait.mnn.prime)

> apply(my.sample, MARGIN = 1, trait.sdnn.prime)
```

We have now successfully calculated the unweighted and abundance weighted of the mean nearest functional neighbor distance and the standard deviation of the nearest functional neighbor distances for each community in our system. There is

currently no function in an R package, that I know of, that calculates the standard deviation of the nearest neighbor distance, but the unweighted and abundance-weighted mean can be calculated using the `mntd()` function in the *picante* package.

```
> mntd(my.sample, square.dist.mat, abundance.weighted = F)

> mntd(my.sample, square.dist.mat, abundance.weighted = T)
```

Again, low values of the mean nearest functional neighbor distance indicate functionally similar species co-occurring or low FD, whereas high values indicate functionally dissimilar species are co-occurring and therefore high FD. In most cases, the mean nearest neighbor distance is sensitive to the species richness gradient across the communities being compared in your system. In particular, the mean nearest neighbor distance will decrease with increasing species richness in most cases. We will discuss the implications of this and null models in Chap. 6.

4.4.4 Ranges and Convex Hulls

One of the oldest and simplest measurements of FD has been to quantify the range of functions in a sample or community. In the case of a single trait this measure of FD is simply the range of the trait values in the community. When there are two or three measured traits the FD then becomes the area or volume of the two- or three-dimensional shape, respectively, representing the community trait space with vertices defined by the maximum and minimum value for each trait [90, 92, 94]. Calculation of these areas or volumes for the purpose of investigating the FD in communities started in the 1960s if not earlier [94], though the approach has been "invented" again recently (e.g., [90]) spurring an increasing number of papers using this conceptual approach. The methodological advance made during these reinventions is the possibility of calculating the trait volume for a community in more than three dimensions (i.e., using more than three traits). In particular, "convex hulls" can now be calculated for high-dimensional data and the volumes of these hulls can be quantified. The convex hull volume for a community is now commonly referred to as FRic or Functional Richness, though the term Functional Richness can often be confused or conflated with Functional Group Richness, which is a distinct measure [92]. The calculation of a convex hull or FRic is conceptually appealing because it can help a researcher understand how species pack and fill trait space. For example, the original use of such metrics in the 1960s and 1970s was to ask whether communities with more species have a greater morphological trait volume [93, 94]. This research was often done in the context of limiting similarity theory where the expectation was that the only way one could "add" species to a community was to add them to the periphery of trait space because adding them somewhere within the existing trait distribution would mean they would be too similar to invade. Further, researchers predicted that environments that are more abiotically benign and/or that have stronger biotic interactions will allow the invasion and success of peripheral

phenotypes resulting in a large morphological volume whereas harsher abiotic environments will limit the invasion and success of peripheral phenotypes resulting in a small morphological volume.

The calculation of the convex hull volumes (i.e., FRic) in R is not difficult, but before we proceed to the measurement of FRic let us quickly determine how to calculate the univariate equivalent of calculating the range for each trait in a community. We begin by focusing on the first community in our community data matrix (i.e., row one) and extracting the names of all the species that are present in the community as determined by their positive values in the community data matrix.

```
> com.1.names <- names(my.sample[1, my.sample[1,] >
0])
```

We can now use the names of the species present in our first community to extract only those rows in the trait matrix containing the species in our fist community. This pruned trait matrix can then be used in an `apply()` function with a MARGIN of 2 to calculate the maximum trait value in each column (i.e., for each trait) and the minimum value. The difference between the outputs from these two `apply()` functions is the range for each trait in the first community.

```
> apply(traits[com.1.names, ], MARGIN = 2, max) -
apply(traits[com.1.names,], MARGIN = 2, min)
```

If we wanted to the above for our fourth community in a single line of code we would use the following.

```
> apply(traits[names(my.sample[4, my.sample[4,] > 0),
], MARGIN = 2, max) - apply(traits[names(my.sample[4,
my.sample[4,] > 0), ], MARGIN = 2, min)
```

Establishing how to calculate the ranges of all traits simultaneously for a single community now makes it clear how to scale the analysis up to analyze the range of all traits in all communities simultaneously. To accomplish this we first write a function that will calculate the range of all traits for a single community.

```
> range.function <- function(x){

      ## Get the names of the species present in our
      ## community.
      com.names <- names(x[x > 0])

      ## Calculate the range for each trait.
      apply(traits[com.names, ], MARGIN = 2, max) -
      apply(traits[com.names, ], MARGIN = 2, min)
}
```

We now apply this function to the rows in our community data matrix to calculate the range of all traits in all communities simultaneously.

```
> apply(my.sample, MARGIN = 1, range.function)
```

The calculation of trait ranges across communities can now be extended to calculate the overall convex hull volume (i.e., FRic). The calculations of convex hulls can now be done with multiple packages in R, but we will utilize the *geometry* package, which we can install and load using the following code.

```
> install.packages("geometry")

> library(geometry)
```

The function `convhulln()` in the *geometry* package can be used to perform all of the necessary calculations. We will begin by first quantifying the convex hull for only the first community in our community data matrix. The `convhulln()` function requires a matrix of continuous values that it will treat as the vertices where each column is a trait. Thus, we can use the following code to extract only those species present in our first community from the trait matrix and provide those values to the convex hull function.

```
> convhulln(traits[names(my.sample[1, my.sample[1, ]
      > 0), ])
```

We see that the function output a three-column matrix defining all of the vertices that constitute the convex hull, but no other information is provided. We must ask the function to output the volume of the convex hull choosing the "FA" option.

```
> hull.output <- convhulln(traits[names(my.sample[1,
my.sample[1,] > 0), ], options = "FA")

> names(hull.output)
```

A list has been output with three elements—hull, area, and vol. The hull element contains all of the vertices we saw above. The area element reports the area of the convex hull, but this is not of interest to us. The vol element contains the volume of the convex hull and therefore is our FRic value for this community.

```
> hull.output$vol
```

This approach for calculating the FRic for a single community can now be scaled up to calculate the FRic for all communities simultaneously using the following function.

```
> hull.function <- function(x){

      ## Get the names of the species present in our
      ## community.
      com.names <- names(x[x > 0])

      ## Calculate the hull using all traits.
      convhulln(traits[com.names, ], options =
      "FA")$vol

}
```

The hull function can now be applied to all rows in the community data matrix using an `apply()` function. This rapidly produces the convex hull volume or FRic for each community.

```
> apply(my.sample, MARGIN = 1, hull.function)
```

The FRic of the species in each community can also be performed by using the `dbFD()` function in the *FD* package. The *FD* package performs many additional useful analyses for trait-based ecology that we discuss in the next section. Here we will install and load the package.

```
> install.packages("FD")

> library(FD)
```

We will now use the `dbFD()` function to calculate FRic. This function calculates many metrics simultaneously, but for the moment we are only interested in outputting the FRic values for our communities.

```
> dbFD(traits, my.sample)$FRic
```

A valuable aspect of this function in the *FD* package is that it performs an initial test to determine whether the trait data require a reduction in dimensionality. This raises an important point that we have discussed above and that permeates through all functional diversity analyses. It is essential that redundant trait axes are reduced so as not to overly weight your metric of FD using several covarying traits. Fortunately, this function performs this for you, but you are advised that reducing dimensionality in other analyses is generally not done for you automatically.

4.4.5 *Other Measures*

The last three metrics that we will discuss are the Functional Evenness (FEve), Functional Divergence (FDiv), and Functional Dispersion (FDis). The FEve metric utilizes a minimum spanning tree (MST) to connect all species in functional trait space and measures the regularity of the species points along the branches of this tree and the regularity of their abundances. Thus, we may expect it to be very similar to a nearest neighbor metric. The FEve metric can be calculated as:

```
> dbFD(traits[colnames(my.sample), ], my.sample, w.abun = T,
stand.x = T )$FEve
```

The FDiv metric first measures the average distance of all species from the centroid of the trait space in a community and then sums the magnitude of the divergences from that mean. Thus higher values are supposed to indicate more dispersion towards the maximum and minimum of the range of traits. It can be calculated as

```
> dbFD(traits[colnames(my.sample), ], my.sample, w.abun = T,
stand.x = T )$FDiv
```

The FDis metric calculates the distance of each species from the centroid of the community traits. Thus, it is not quite the same calculation as the pairwise distance between species we can expect that it might be highly correlated. It is also similar to the FDiv metric, though conceptually perhaps easier to understand the biological meaning of FDis and it is likely a clearer indicator of trait dispersion in a community. The FDis can be calculated as:

```
> dbFD(traits[colnames(my.sample), ], my.sample, w.abun = T,
stand.x = T )$FDis
```

We have now calculated all of the major types and varieties of FD metrics that you will commonly see in the literature. In the next section we will quickly compare these metrics to evaluate their independence.

4.5 Comparing Metrics of Functional Diversity

As we consider diversity metrics in this book and elsewhere it is essential to consider their mathematical and statistical relationships in order to determine which are providing novel information and which are not and are simply monotonic with an existing metric. We have explored most of these metrics in the previous chapter on PD, but we will quickly plot them against each other and calculate their correlations (Fig. 4.3).

```
> richness <- rowSums(decostand(my.sample, method = "pa",
MARGIN = 1))

> pw <- mpd(my.sample, as.matrix(dist(traits)),
abundance.weighted = F)

> pw.prime <- mpd(my.sample, as.matrix(dist(traits)),
abundance.weighted = T)

> nn <- mntd(my.sample, as.matrix(dist(traits)),
abundance.weighted = F)

> nn.prime <- mntd(my.sample, as.matrix(dist(traits)),
abundance.weighted = T)

> fric <- dbFD(traits[colnames(my.sample), ],
my.sample, w.abun = T,  stand.x = T )$FRic

> feve <- dbFD(traits[colnames(my.sample), ],
my.sample, w.abun = T,  stand.x = T )$FEve

> fdiv <- dbFD(traits[colnames(my.sample), ],
my.sample, w.abun = T,  stand.x = T )$FDiv

> fdis <- dbFD(traits[colnames(my.sample), ],
my.sample, w.abun = T,  stand.x = T )$FDis

> outputs <- as.data.frame(cbind(richness, pw,
pw.prime, nn, nn.prime, fric, feve, fdiv, fdis))

> names(outputs) <- c("richness", "pw", "pw.prime",
"nn", "nn.prime", "FRic", "FEve", "FDiv", "Fdis" )

> plot(outputs, pch = 16)

> cor(outputs)
```

We see that the PW and FDis metrics are almost identical as has been previously pointed out by Laliberte and Legendre [92] and that FDis and FDiv, while being correlated, are not identical indices. Lastly, we see that the hull/FRic metric and the mean nearest neighbor metric are related to species richness, indicating that they should perhaps be considered in the context of a null model for comparative analyses (see Chap. 6).

4.6 Conclusions

This chapter has addressed a large number of metrics that can be used to characterize the functional structure and diversity of communities or assemblages. We began by simply characterizing the moments of the trait distribution in assemblages. While this is useful, these measures, particularly the mean, are not measures of functional diversity and should not be used as such. We then covered pairwise- and nearest

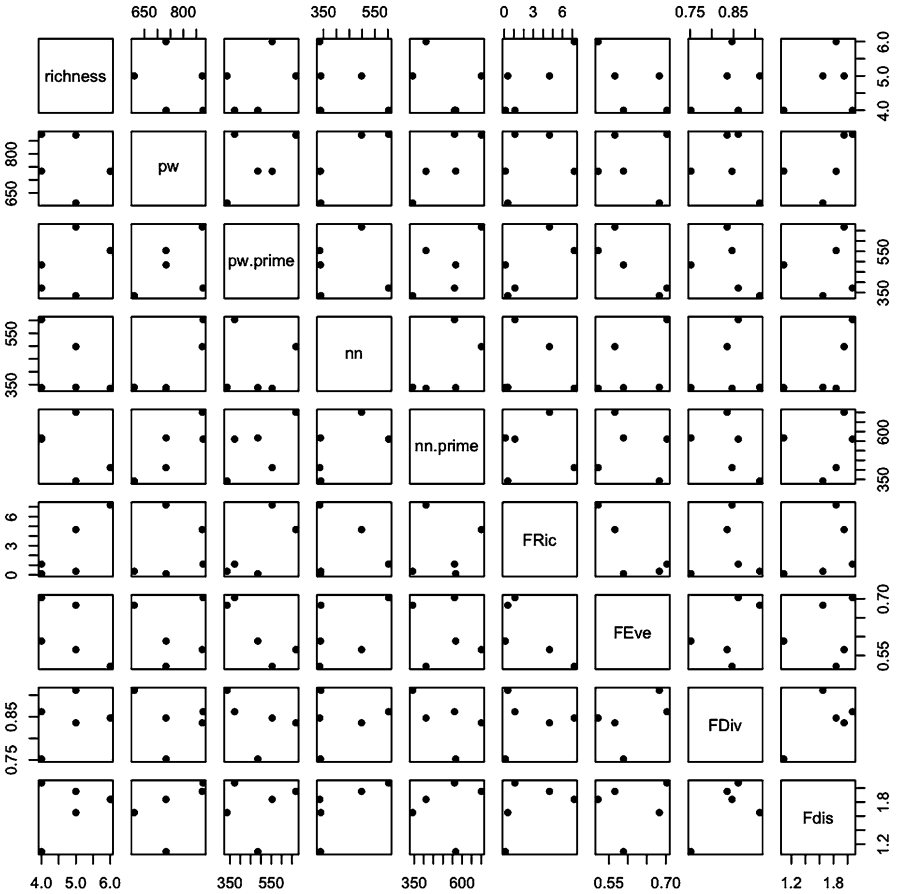

Fig. 4.3 A plot of each functional diversity metric against all other metrics

neighbor-based metrics of functional diversity which have their roots in the eco-morphology literature spanning back to at least the 1960s underscoring the long history of measuring trait similarity and functional diversity in species assemblages.

Almost all functional diversity metrics that have been published or that will be published will likely fall into either the pairwise or nearest neighbor classes. There are likely many metrics of functional diversity that I have not covered in this chapter, but it is likely that many of these are highly correlated with the measures we have covered. In deciding the "best" metric for your study I urge you to consider whether your chosen metric is actually that different from existing metrics. In many cases we may find that the metric is monotonic with an existing metric and may not provide much additional metric. It is also expected that the code above will provide you with enough details regarding how to handle trait and community data in R to formulate your own measures of functional diversity in R, but I again urge you to make sure your new metrics are indeed novel.

4.7 Exercises

1. Simulate two traits on your example phylogeny from Chap. 3 using Brownian Motion and the `fastBM()` function in the *phytools* package.
2. Quantify the functional diversity for each of your communities using the simulated trait data, the example community data matrix from Chap. 3, and the pairwise and nearest neighbor metrics weighted by abundance. Next do the same for phylogenetic diversity using the analogous pairwise and nearest neighbor metrics and the same phylogeny you used to simulate the trait data.
3. Perform a simple correlation between the phylogenetic and functional diversity measures. Do the functional and phylogenetic diversity measures correlate? Why or why not?
4. Repeat the above with other metrics of that are presence–absence weighted or weighted by abundance.