# fundiversity: a modular R package to compute functional diversity indices

Matthias Grenié, Hugo Gruson

ECOGRAPHY

*Software note*

# `fundiversity`: a modular R package to compute functional diversity indices

**Matthias Grenié** [1,2] **and Hugo Gruson** [2,3]

[1]German Centre for Integrative Biodiversity Research (iDiv) Halle-Jena-Leipzig, Leipzig, Germany
[2]CEFE, Univ. de Montpellier, CNRS, EPHE, IRD, Univ. Paul Valéry Montpellier 3, Montpellier, France
[3]Centre for Mathematical Modelling of Infectious Diseases, London School of Hygiene and Tropical Medicine, London, UK

**Correspondence: Matthias Grenié (matthias.grenie@gmail.com)**

**Ecography**
**2023: e06585**
doi: 10.1111/ecog.06585

Subject Editor:
F. Guillaume Blanchet
Editor-in-Chief: Miguel Araújo
Accepted 28 December 2022

Functional diversity is widely used and widespread. However, the main packages used to compute functional diversity indices are not flexible and not adapted to the volume of data used in modern ecological analyses. We here present `fundiversity`, an R package that eases the computation of classical functional diversity indices. It leverages parallelization and memoization (caching results in memory) to maximize efficiency with data with thousands of columns and rows. We also did a performance comparison with packages that provide analog functions. In addition to being more flexible, `fundiversity` was always an order of magnitude quicker than alternatives. `fundiversity` aims to be a lightweight, efficient tool to compute functional diversity indices, which can be used in a variety of contexts. Because it has been designed following clear principles, it is easy to extend. We hope the wider community will adopt it and we welcome all contributions.

Keywords: biodiversity, community ecology, diversity facet, functional biogeography, functional ecology, R package

## Introduction

Functional diversity, the diversity of traits across scales, is a major facet of biodiversity (Pavoine and Bonsall 2011). It has been widely used across ecological contexts (Cadotte et al. 2011) and has been shown to relate to ecosystem functioning (Díaz and Cabido 2001; Leps et al. 2006). Many indices exist to characterize it across its three dimensions: richness (how much?), evenness (how regular?) and divergence (how different?) (Pavoine and Bonsall 2011). Ecologists rely on computational tools to compute these indices in a reproducible fashion, mainly in the R programming language (Lai et al. 2019; www.r-project.org). The `FD` package is the main tool available for functional diversity indices, having now accumulated more than 1200 citations (Laliberté et al. 2014). But `FD` was released in 2009 and received only minor

updates that stopped in 2015. During this time, software development practices have changed dramatically and new, higher-performance tools have emerged in the R ecosystem. Additionally, since 2009, the size of ecological datasets has grown exponentially (Farley et al. 2018, Wüest et al. 2020) and high-performance computing (HPC) environments have become standard. There is, therefore, a dire need for a modern alternative using state-of-the-art software development techniques and tools.

The main function of the `FD` package `dbFD()` lets users compute a dozen functional diversity indices in a single call from raw trait data (Laliberté et al. 2014). While great for exploratory analyses, this can increase computation time when only a single index is needed. Furthermore, it does not enforce the good practice of choosing beforehand the appropriate functional diversity index for the question(s) asked (Schleuter et al. 2010, Mason et al. 2013, Legras et al. 2018). It encourages the user to fish for the functional diversity index matching predicted relationships (a form of p-hacking). This can lead users to report all computed functional diversity indices even when there are no clear expectations about different functional diversity facets, or to report correlated indices (Schleuter et al. 2010, Mason et al. 2013, Legras et al. 2018, McPherson et al. 2018). Computing all indices in a single function also makes long-term maintenance and addition of new indices harder.

The average size of datasets analyzed in ecology increased severalfold in recent years (Wüest et al. 2020), calling for an increase in performance of computational tools. This increase is particularly urgent considering that many diversity analyses use null models that increase the data size by two or three orders of magnitude (Gotelli and Graves 1996). First, any improvement of the algorithmic efficiency to compute functional diversity indices could save substantial amounts of time as it is repeated many times. For example, we noted that many R packages (www.r-project.org) that compute functional diversity indices do not leverage the specifically optimized matrix algebra packages included in R. Their use can cut the number of operations dramatically compared to using a loop directly in R. Second, functional diversity indices are generally computed over many mathematically independent sites. With the rise of multicore computers, parallelization (i.e. splitting independent computations between different computing processor units, CPUs), is becoming the norm. Very few functional diversity R packages natively implement parallelization, leaving the burden of doing so to the user. There have been tremendous new developments in this area in R over the last few years with the release of the `future` framework (Bengtsson 2020) that allows computations to be seamlessly parallelized on multiple cores on a single machine, across several machines or even on a remote cluster without changing execution code. Third, computations on the same input can be cached through a process called memoization (Wickham et al. 2021). This avoids wasting computing power on previously seen inputs. For example, many functional diversity indices rely on the computation of convex hulls across a multi-dimensional space (Cornwell et al. 2006, Villéger et al. 2008). Caching the results of this costly computation could save time and computing power when measuring the diversity across similar sets, such as sites across a given region.

Discussions are increasingly being held regarding scientific software robustness and reliability in ecology (Poisot 2015, White 2015, Mislan et al. 2016, Wilson et al. 2017). Mainly because most ecologists are self-trained in programming (Farrell and Carey 2018), these virtuous practices are rarely applied in ecology (Barraquand et al. 2014). For example, unit tests use predefined inputs to compare outputs to expectations (Poisot 2015). Unit tests have also become standard in R packages since the release of packages streamlining this process, such as `testthat` and `tinytest` (Wickham 2011, van der Loo 2021). Very few R functional diversity packages provide unit tests to assess that the functions behave expectedly. Automatic tests of one's code are crucial when developing a tool for a wider audience, as it may be used across different contexts.

Here, we propose a modern alternative to `FD` called `fundiversity`, which benefits from modern development practices, necessary features for large-sized datasets (modularity, parallelization and memoization) and greater flexibility. The package can be easily extended to accommodate additional diversity indices not covered by following a clear design pattern detailed in the next section. We go through a use case to show how it can be used. We then compare the performance of `fundiversity` against similar packages.

## Main features of `fundiversity`

To ensure the consistency of its functions and to make it user-friendly, `fundiversity` follows clear design principles. In this section, we expose its distinctive features and principles.

To give maximum flexibility to the users, we tried to build `fundiversity` to be as modular as possible. Each function in `fundiversity` computes a single functional diversity index, so that it only returns a single index and nothing more. All functions in `fundiversity` are prefixed with `fd_` to avoid conflict with similarly named functions in other packages, as it is becoming standard practice in R packages (rOpenSci et al. 2021). In line with its modularity, we focused on making the inputs and outputs of functions coherent. The functions use two main inputs: a species by traits matrix and a site by species matrix; all functions accept them as first arguments. Across functions, the outputs are always structured similarly: one `site` column that contains the name of its sites and one column named as the computed index (such as `FRic` when computing functional richness). The shape of the output is predictable and easily combined with other data.

We designed `fundiversity` so it minimally modifies the input data before computing the indices (Fig. 1). When computing functional diversity indices, upstream choices regarding trait standardization and trait space construction

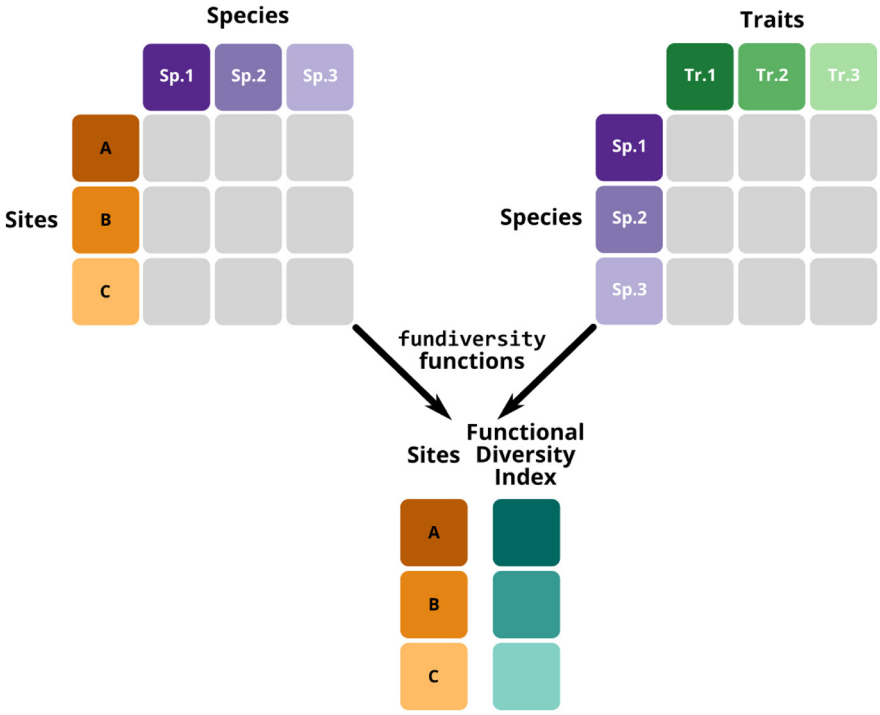


Figure 1. Conceptual diagram showing the input and typical ouput data from `fundiversity` functions. Input data are generally a site–species table and a species–traits table, and the output returns a table of functional diversity index per site.

are fully part of the scientific question (Leps et al. 2006, de Bello et al. 2013, Maire et al. 2015, Mammola et al. 2021). Several packages provide default options that automatically build multivariate spaces or dendrograms, and choose relevant axes of variation. While useful for naive users, these abstract away part of the scientific process that should be considered when using functional diversity indices. These choices have been shown to have strong consequences on the values of downstream indices (Leps et al. 2006, de Bello et al. 2013, Maire et al. 2015, Mammola et al. 2021). `fundiversity` chooses to enforce better practices by leaving trait space choices to the user.

Parallelization can vastly decrease computation speed by leveraging the architecture of modern computers. Most functions in `fundiversity` can be parallelized out of the box. `fundiversity` provides parallelization through the future backend (Bengtsson 2020). Parallelization is toggled through a single function call using `future::plan()` before using `fundiversity` functions. Thanks to the flexibility given by the `future` backend, the code to use will not change whether parallelizing across several cores on a single computer, across multiple computers or on a remote high-performance cluster. The user only needs to update the call to `future::plan()`. Furthermore, the `future` backend provides load balancing so that no cores/units stay idle for too long and the parallelized tasks are split evenly. The package contains a dedicated vignette to guide the users through transforming unparallelized to parallelized code (accessible through `vignette("fundiversity_1-parallel", package="fundiversity")`).

The computation of functional diversity indices often involves null models, which require repeated operations across the same data subsets. This results in computing the same indices over exactly the same assemblages over and over. Memoization can leverage the already computed indices and avoid duplicated work. For example, to compute functional richness (FRic) the convex hull of the input data has to first be identified, and then the volume of this convex hull is computed. The first step, identifying the convex hull, takes the most time and, as such, storing the results of each computed convex hull can vastly cut computation time for a little memory footprint. If the set of input points is encountered a second time, the results will be taken from memory instead of being recomputed. Memoization trades a little bit of computer memory (keeping the convex hulls stored) for more computation speed. For now, `fundiversity` leverages memoization only for computing convex hulls (as used when computing `FDiv`, `FRic` and `FRic_intersect`). It is activated by default. This behavior can be overridden by changing the `fundiversity.memoise` option before loading the package.

Packages depend on one another to avoid reinventing the wheel and thus reuse already developed functions. A higher number of dependencies means that a package requires more packages to be installed before its own installation. While having many dependencies minimizes code replication, it also comes with a high price, because if a single dependency breaks then the whole package cannot be installed anymore (Cox 2019). Inflated dependencies have been identified as a major risk in software, and

especially in scientific software development (Claes et al. 2014, Cox 2019). FD has only four dependencies but other functional diversity packages have many more dependencies (up to 100 dependencies for hypervolume, Supporting information). This renders them quite brittle for the users after years of not being actively developed. fundiversity has been designed to only have minimal external dependencies, it currently depends directly on four, carefully chosen, external packages (details on the criteria of choice in vignette("fundiversity_4-design-principles", package="fundiversity")): future.apply which itself depends on two other packages, Matrix (which is shipped with R), and geometry and vegan, on which FD also depends. Considering direct and indirect dependencies, fundiversity depends on a total of 21 packages while similar packages depend on 65 packages on average (Supporting information).

Because user flexibility is key, fundiversity has minimal assumptions on the input data structure. All its functions work with data frames, matrices or sparse matrices alike. Sparse matrices are a different formalization of matrices that do not store explicitly the cells that contain zero. They offer a reduced memory footprint and optimized algebra library for computation (Bates and Maechler 2021). These matrices are thus specifically relevant for occurrence/abundance matrices that contain many zeros. If the used data have a high proportion of zeros, using sparse matrices can vastly decrease computational time in fundiversity.

As we underlined in the introduction, automatic software testing, while not 100% flawless, is needed to increase the confidence in the behavior of functions. It is widespread in professional software engineering but much less so in scientific software development (Kanewala and Bieman 2014). This means that software behavior is seldom validated against known inputs to make sure that it behaves in expected ways. This fact does not mean that the software is of poor quality, but rather that some simple errors could introduce unnoticed changes in the behavior of functions. Most packages that compute functional diversity indices do not include any form of automatic testing (only 3 out of 11 do it following our assessment). We do want to point out that the lack of tests results from the lack of formal training in software development for ecologists (Farrell and Carey 2018). We designed fundiversity with many unit tests from the beginning, executing at least every single line of code once (i.e. achieving code coverage of 100%).

fundiversity mostly computes alpha functional diversity indices, because other recent packages exist to compute other types of functional diversity indices (Hill numbers, Li 2018; beta-diversity indices, Baselga and Orme 2012). We focused on indices available through the dbFD() function in the FD package and on indices that could benefit from faster implementations. The included indices cover the three dimensions of functional diversity: richness (how much total diversity within the set?), evenness (how regularly are species situated along the trait space?) and divergence (how different are species compared to an average position?) (Pavoine and Bonsall 2011). fundiversity contains the following alpha functional diversity indices: functional richness (FRic), functional dispersion (FDis), functional divergence (FDiv), functional evenness (FEve) and Rao's quadratic entropy (Q). We included Q as we identified the potential for important performance improvements relative to existing packages. fundiversity also contains a beta-diversity index (named FRic_intersect) as it can be useful to compare the overlap between convex hulls between sites. Thanks to its design, fundiversity can be easily extended to include more indices; we include below the list of available indices in the current version of fundiversity.

We made sure the indices were numerically exact by using the test dataset available in Villéger et al. (2008). The functions in fundiversity gave identical results to the one found in Fig. 2 of Villéger et al. (2008). We summarize our comparisons in the numerical correctness vignette accessible through vignette("fundiversity_3-correctness", package="fundiversity"). We furthermore compared the obtained results with functions in other packages and made sure that similar values were obtained.

## Case study

In this section, we show how to use fundiversity in practice. As an example dataset, we included in fundiversity site-species and trait data from Nowak et al. (2019). It is accessible through calls to site_sp_birds and traits_birds when fundiversity is loaded with library() or the use of the data("site_sp_birds", package="fundiversity") and data("traits_birds", package="fundiversity") functions when fundiversity is not loaded. This dataset describes the presence/absence of bird species in South America at different elevations and four of their morphological traits.

The trait values show species in rows (species are specified as row names) and traits in columns with trait names as column names (Fig. 1). Similarly, the site-species matrix contains sites as rows (site names are row names) and species as columns (species names are column names).

```
data("traits_birds",  package = "fundiversity")
data("site_sp_birds", package = "fundiversity")

head(traits_birds)

##                     Bill.width..mm. Bill.length..mm. Kipp.s.index Bodymass..g.
## Aburria_aburri                18.35            35.48         0.18       1407.5
## Amazona_farinosa              26.50            38.81         0.29        626.0
## Amazona_mercenaria            17.51            26.30         0.33        340.0
## Amazona_ochrocephala          20.17            31.40         0.26        440.0
## Ampelioides_tschudii          16.53            24.58         0.24         78.4
## Ampelion_rufaxilla            16.97            21.89         0.28         73.9


head(site_sp_birds)[, 1:3]

##           Aburria_aburri Amazona_farinosa Amazona_mercenaria
## elev_250               0                1                  0
## elev_500               0                1                  0
## elev_1000              1                1                  1
## elev_1500              1                0                  1
## elev_2000              0                0                  1
## elev_2500              0                0                  1
```

Now that we have obtained trait and occurrence data we need to compute the trait dissimilarity between each pair of species. As all traits are quantitative: we first scale them to zero mean and standard deviation of one (z-score), then we compute the Euclidean distance between pairs of species.

```
z_traits = scale(traits_birds, center = TRUE, scale = TRUE)

trait_distance = as.matrix(dist(z_traits))
```

We want to emphasize here that `fundiversity` does not assume anything in the upstream of the computation of functional diversity indices. Trait standardization and computation of a trait dissimilarity are left to the user's discretion. They are provided here as a full workflow example. The specific functions used in the previous chunk can vary depending on the scientific question, the nature of the traits or the transformation needed. `fundiversity` does not provide any functions to deal with these upstream choices as it is the user's responsibility to carefully examine them.

Then, we compute the functional richness at each location using the `fd_fric()` function. It expects quantitative trait values as the first argument and a site-species matrix as the second argument.

```
library("fundiversity")

birds_fric = fd_fric(z_traits, site_sp_birds)

head(birds_fric)

##        site      FRic
## 1  elev_250 66.048816
## 2  elev_500 71.465678
## 3 elev_1000 43.354008
## 4 elev_1500 25.466685
## 5 elev_2000  7.725843
## 6 elev_2500  7.046431
```

If the site–species matrix is not provided, `fundiversity` considers that all species present in the trait matrix are all present in a single site:

```
fd_fric(z_traits)

##   site    FRic
## 1   s1 88.9286
```

All other functions in `fundiversity` use a similar structure: the first input is trait data, the second is a site–species matrix (Fig. 1). For Rao's quadratic entropy, computed through `fd_raoq()`, functional dissimilarities can be specified as the third argument:

```
# With functional dissimilarity
birds_raoq = fd_raoq(traits = NULL, site_sp_birds, dist_matrix = trait_distance)

# With trait values
birds_raoq_2 = fd_raoq(z_traits, site_sp_birds)

# Both options give the same results
identical(birds_raoq, birds_raoq_2)

## [1] TRUE
```

If not all traits are quantitative, it is possible to transform them back into independent quantitative 'traits' through the use of Gower's distance (Gower 1971; and its extensions: Podani 1999, Pavoine et al. 2009) then applying multivariate analysis to obtain orthogonal dimensions (Maire et al. 2015). But there are many other ways to convert qualitative traits and, as such, this is out of the scope of `fundiversity`. We leave it to the user to decide how to proceed to obtain independent quantitative traits.

### Using parallel computation

As specified above, `fundiversity` allows for parallel computation of functional diversity metrics through the `future` framework. We here demonstrate how to use it in practice with the case study. A more detailed explanation is provided in the "Parallelization" vignette of `fundiversity` (available through `vignette("fundiversity_1-parallel", package="fundiversity")`).

We first have to check if the function in `fundiversity` is parallelizable: all functions except `fd_raoq()` are (Table 1). Then we define the parallel setting using the `future::plan()` function. This allows users to define how the parallel computation should be split: across cores, across computers, across jobs of a high-performing cluster, etc. Here, we split the computation locally across the 4 cores of the computer using the `future::multisession()` function. We specify the number of cores to use with the `workers` argument in the call to the `future::plan()` function.

```
# First: Setup a parallel plan
future::plan(future::multisession, workers = 4)

# Second: Perform the computation
bird_fric = fd_fric(traits_birds, site_sp_birds)

head(bird_fric)

##        site      FRic
## 1  elev_250 171543.73
## 2  elev_500 185612.55
## 3 elev_1000 112600.18
## 4 elev_1500  66142.75
## 5 elev_2000  20065.76
## 6 elev_2500  18301.18
```

To use a different backend, you can invoke a different argument in the `future::plan()` function. All possible arguments are detailed in the overview vignette of the `future` package (accessible through `vignette("future-1-overview", package="future")` once `future` has been installed).

The parallel computations are split across sites, so parallelization can drastically improve performance when having a large number of sites. However, given the efficiency of `fundiversity` functions, and the overhead costs of parallel computation, we recommend parallelizing only with matrices of at least 10 000 sites, or when hitting a performance limit of the default sequential execution.

Also, parallelization should never be used in conjunction with memoization because of the risk of cache corruption if several cores access the memoized cache simultaneously (make sure to use `options(fundiversity.memoise=FALSE)` before loading `fundiversity` when using parallel computations).

## Performance comparison

To test the performance improvements realized by `fundiversity`, we compared computation time on standardized datasets across similar functions in other packages. We only compared packages that provide original functions, not wrappers that depend on other packages to compute functional diversity indices. Six packages computed similar indices to `fundiversity`. Most indices are also computed by the `FD::dbFD()` function, but the comparison would be unfair as it computes many indices in a single call, while functions in `fundiversity` only compute single indices. We considered functions from: `adiv` (Pavoine 2020), `BAT`

Table 1. List of functions available in `fundiversity` to compute functional diversity indices. The last two columns specify which functions are parallelizable and memoizable.

| Function name | Index name | Source | Parallelizable | Memoizable |
|---|---|---|---|---|
| `fd_fdis()` | Functional dispersion (FDis) | Laliberté and Legendre (2010) | Yes | No |
| `fd_fdiv()` | Functional divergence (FDiv) | Villéger et al. (2008) | Yes | Yes |
| `fd_feve()` | Functional evenness (FEve) | Villéger et al. (2008) | Yes | No |
| `fd_fric()` | Functional richness (FRic) | Villéger et al. (2008) | Yes | Yes |
| `fd_fric_intersect()` | Functional β-diversity | Villéger et al. (2013) | Yes | Yes |
| `fd_raoq()` | Rao's quadratic entropy (Q) | Rao (1982) | No | No |

(Cardoso et al. 2015), `betapart` (Baselga and Orme 2012), `hillR` (Li 2018), `mFD` (Magneville et al. 2022) and `FD` (Laliberté et al. 2014) (see Table 2 for the correspondence between packages). A continuously updated version of this section can be found in the performance comparison vignette within the `fundiversity` package with `vignette("fundiversity_2-performance", package="fundiversity")`.

For testing purposes, we used datasets of increasing size with the number of species being 200, 500 or 1000; the number of traits 2, 4 or 10; and the number of sites 50, 100 or 500. For each set of parameters, we generated a fictional site–species matrix and species–trait matrix containing only continuous traits. We used these simulated data to perform benchmarks across comparable functions (Table 2). The benchmark was run 30 times through the `bench` package (Hester and Vaughan 2021) (Fig. 2). The full results detailing the timings for each combination of parameters and functions are available in the Supporting information.

For all the indices and functions tested, `fundiversity` is at least one order of magnitude faster than alternative packages. For functional dispersion (`FDis`), `fundiversity` is two orders of magnitude faster compared to `BAT` and `mFD`. For functional divergence (FDiv), `fundiversity` is one order of magnitude faster than `mFD`. For functional evenness (FEve), `fundiversity` is two orders of magnitude faster than `mFD` with sequential and parallelized versions having similar performances. For Rao's quadratic entropy (Q), `fundiversity` is one order of magnitude faster than `hillR` and `mFD`, two orders faster than `adiv` and three orders of magnitude faster than `BAT`. For functional richness (FRic), `fundiversity` is half an order of magnitude faster than the hull version of `BAT`, as well as being one and a half orders of magnitude faster than its tree version and `mFD`. For functional richness intersection (beta functional diversity), `fundiversity` is two orders of magnitude faster than `betapart` and `hillR`.

The parallelized versions of `fundiversity` functions executed on average one order of magnitude faster than the sequential versions (Fig. 3). For functional richness we even observed a difference of two orders of magnitude. However, for functional dispersion, parallelization increased the overall computation time. This may be due to inherent parallelization issues: there is an overhead cost when splitting tasks across multiple cores of a computer. The efficiency of parallelization depends on the difficulty of the tasks that are split between cores. In the case of functional richness, the main task is computing the convex hull, which is computationally costly, which is why parallelization increases performance in this case. However, computing functional dispersion is simpler and, as such, does not benefit from being split across different cores. Different values for the number of cores, species, traits or sites produce qualitatively the same results (full results in Supporting information).

One important note regarding parallelization in `fundiversity` is that it is important to avoid doing both memoization and parallelization simultaneously. Memoization creates a cache to avoid recomputing results, and the cache may be corrupted if several cores access the same results at the same time. We noticed that toggling memoization while

Table 2. List of `fundiversity` functions with corresponding functions in other packages. The name of the package is indicated before the `::` while the name of the functions (including specified arguments) follows.

| Index name | `fundiversity` functions | Equivalent functions |
|---|---|---|
| Functional dispersion (FDis) | `fd_fdis()` | `FD::fdisp()`<br>`mFD::alpha.fd.multidim(..., ind_vect="fdis")` |
| Functional divergence (FDiv) | `fd_fdiv()` | `mFD::alpha.fd.multidim(..., ind_vect="fdiv")` |
| Functional evenness (FEve) | `fd_feve()` | `mFD::alpha.fd.multidim(..., ind_vect="feve")` |
| Functional richness (FRic) | `fd_fric()` | `BAT::alpha()` (tree)<br>`BAT::hull.alpha()` (hull)<br>`mFD::alpha.fd.multidim(..., ind_vect="fric")` |
| Rao's quadratic entropy (Q) | `fd_raoq()` | `adiv::QE()`<br>`BAT::rao()`<br>`hillR::hill_func()`<br>`mFD::alpha.fd.hill(..., q=2, tau="max")` |
| Functional β-diversity | `fd_fric_intersect()` | `betapart::functional.beta.pair()`<br>`hillR::hill_func_parti_pairwise()` |

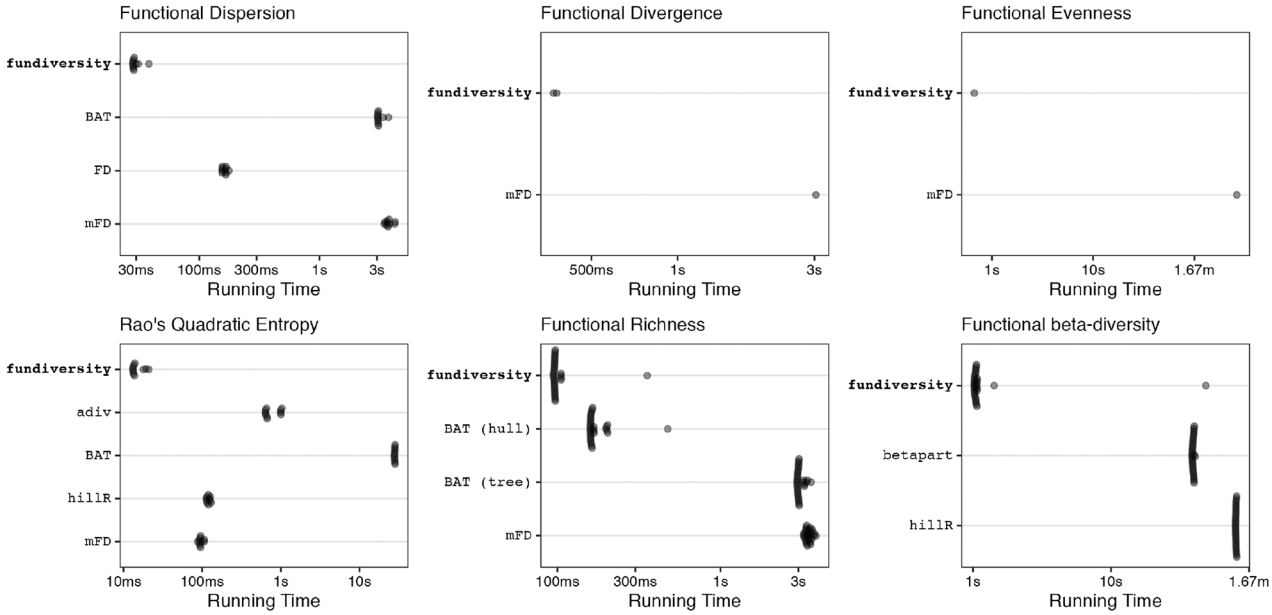


Figure 2. Timing comparison across functional diversity indices between packages. Each point represents the execution time of one run using a simulated dataset; the points are transparent and jittered to avoid overplotting. We show here the performance results considering only a single set of parameters with 4 traits, 500 species and 100 sites, repeated 30 times.

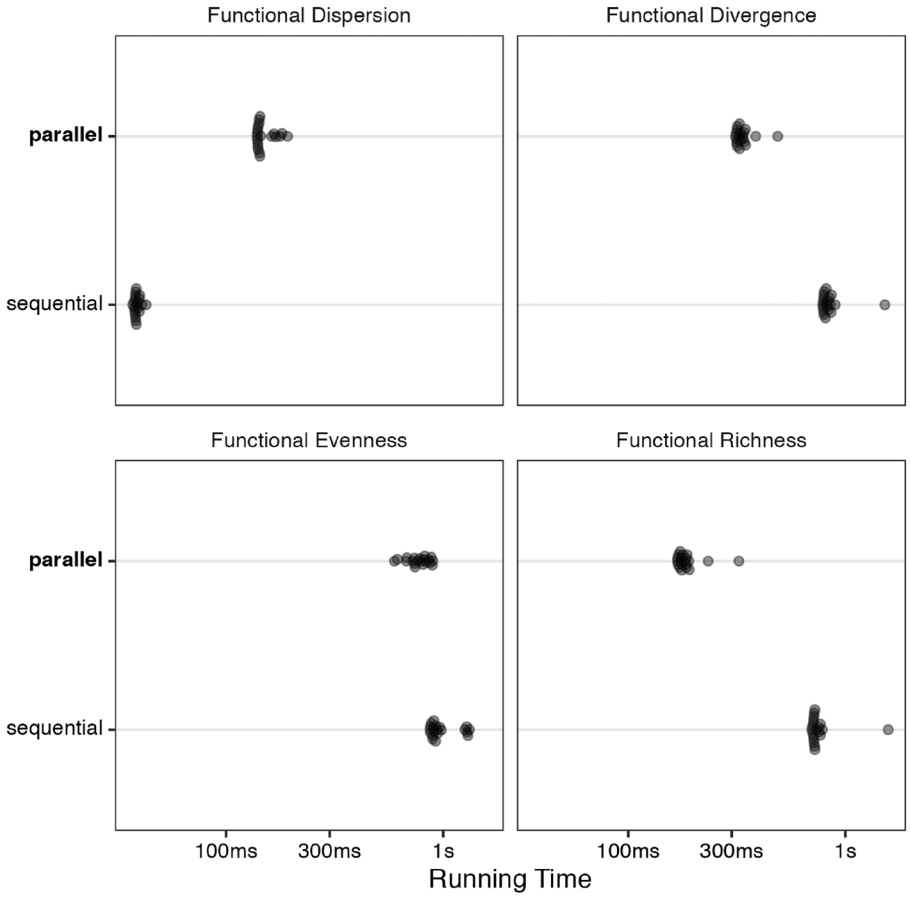


Figure 3. Timing comparison between parallel and sequential version of `fundiversity` functions across functional diversity indices. Each point represents the execution time of one run using simulated datasets with fixed properties (4 traits, 100 sites, 500 species); the points are transparent and jittered to avoid overplotting. The parallel version ran across 6 cores.

performing parallelization severely increases total computational time compared to sequential performance.

Note that these benchmarks only assess the packages' computation speed and in no way assess the intrinsic quality or usefulness of any package. We are comparing `fundiversity`, a package whose main goal is performance, with other packages that may have other primary goals and offer other benefits. For example, several packages offer nice default visualization functions to plot the different diversity indices, while we explicitly considered that visualization functions were not part of `fundiversity` and let the users decide how they want to plot their indices.

## Conclusion

We proposed a modern alternative R package to compute functional diversity indices. This package follows current best development practices and leverages modern features like parallelization and memoization to increase its performance. This is only made possible by recent developments that were, for the most part, not available at the time when alternative packages came out. `fundiversity` does not propose to replace the entire toolkit for the researcher interested in functional diversity (including the upstream selection of the traits and the building of a functional space), but instead focuses on improving the most computationally costly step: computing functional diversity indices. We hope it will be a useful contribution to this toolkit. To ensure its long-term maintainability, we made the package available on GitHub; it is perennially archived on Zenodo; sits in an independent GitHub organization; and is written following clear design principles. This package aims to always be a work in progress, and as such we welcome contributions from interested users and developers.

To cite `fundiversity` or acknowledge its use, cite this Software note as follows, substituting the version of the application that you used for 'version 1.0':

Grenié, M. and Gruson, H. 2023. `fundiversity`: a modular R package to compute functional diversity indices. – Ecography 2023: e06585 (ver. 1.0).

*Acknowledgements* – Both authors thank Shan Kothari and two anonymous reviewers for their comments, which helped improve the manuscript.

*Funding* – M.G. gratefully acknowledges the support of iDiv funded by the German Research Foundation (DFG-FZT 118, 202548816). The authors acknowledge support from the iDiv Open Science Publication Fund and from the Open Access Publishing Fund of Leipzig University, which is supported by the German Research Foundation within the program Open Access Publication Funding.

### Author contributions

**Matthias Grenié**: Conceptualization (equal); Methodology (equal); Software (equal); Supervision (equal); Writing – original draft (equal); Writing – review and editing (equal). **Hugo Gruson**: Conceptualization (equal); Methodology (equal); Software (equal); Writing – review and editing (equal).

### Transparent peer review

The peer review history for this article is available at https://publons.com/publon/10.1111/ecog.06585.

### Data availability statement

`fundiversity` is available on CRAN through `install.packages("fundiversity")` as well as on GitHub at https://github.com/funecology/fundiversity, for archival all releases are available on Zenodo at https://doi.org/10.5281/zenodo.4761754. The data used in this article are available from the package, through `data(package="fundiversity")` call.

### Supporting information

The Supporting information associated with this article is available with the online version.

## References

Barraquand, F., Ezard, T. H. G., Jørgensen, P. S., Zimmerman, N., Chamberlain, S., Salguero-Gómez, R., Curran, T. J. and Poisot, T. 2014. Lack of quantitative training among early-career ecologists: a survey of the problem and potential solutions. – PeerJ 2: e285.

Baselga, A. and Orme, C. D. L. 2012. Betapart: an R package for the study of beta diversity. – Methods Ecol. Evol. 3: 808–812.

Bates, D. and Maechler, M. 2021. Matrix: sparse and dense matrix classes and methods (Manual). – https://cran.r-project.org/package=Matrix.

Bengtsson, H. 2020. A unifying framework for parallel and distributed processing in R using futures . – https://arxiv.org/abs/2008.00553.

Cadotte, M. W., Carscadden, K. and Mirotchnick, N. 2011. Beyond species: functional diversity and the maintenance of ecological processes and services. – J. Appl. Ecol. 48: 1079–1087.

Cardoso, P., Rigal, F. and Carvalho, J. C. 2015. BAT – biodiversity assessment tools, an R package for the measurement and estimation of alpha and beta taxon, phylogenetic and functional diversity. – Methods Ecol. Evol. 6: 232–236.

Claes, M., Mens, T. and Grosjean, P. 2014. On the maintainability of CRAN packages. – In: 2014 software evolution week – IEEE conference on software maintenance, reengineering and reverse engineering (CSMR-WCRE), pp. 308–312. IEEE.

Claes, M., Mens, T., & Grosjean, P. (2014, February). On the maintainability of CRAN packages. In 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE) (pp. 308-312). IEEE.

Cornwell, W. K., Schwilk, D. W. and Ackerly, D. D. 2006. A trait-based test for habitat filtering: convex hull volume. – Ecology 87: 1465–1471.

Cox, R. 2019. Surviving software dependencies. – Commun. ACM 62: 36–43.

de Bello, F., Carmona, C. P., Mason, N. W. H., Sebastià, M.-T. and Lepš, J. 2013. Which trait dissimilarity for functional diversity: trait means or trait overlap? – J. Veg. Sci. 24: 807–819.
Díaz, S. and Cabido, M. 2001. Vive la différence: plant functional diversity matters to ecosystem processes. – Trends Ecol. Evol. 16: 646–655.
Farley, S. S., Dawson, A., Goring, S. J. and Williams, J. W. 2018. Situating ecology as a big-data science: current advances, challenges and solutions. – BioScience 68: 563–576.
Farrell, K. J. and Carey, C. C. 2018. Power, pitfalls and potential for integrating computational literacy into undergraduate ecology courses. – Ecol. Evol. 8: 7744–7751.
Gotelli, N. J. and Graves, G. R. 1996. Null models in ecology. – Smithsonian Institution Press.
Gower, J. C. 1971. A general coefficient of similarity and some of its properties. – Biometrics 27: 857–871.
Hester, J. and Vaughan, D. 2021. Bench: high precision timing of R expressions. – https://CRAN.R-project.org/package=bench.
Kanewala, U. and Bieman, J. M. 2014. Testing scientific software: a systematic literature review. – Inform. Softw. Technol. 56: 1219–1232.
Lai, J., Lortie, C. J., Muenchen, R. A., Yang, J. and Ma, K. 2019. Evaluating the popularity of R in ecology. – Ecosphere 10: e02567.
Laliberté, E. and Legendre, P. 2010. A distance-based framework for measuring functional diversity from multiple traits. – Ecology 91: 299–305.
Laliberté, E., Legendre, P. and Shipley, B. 2014. FD: measuring functional diversity from multiple traits, and other tools for functional ecology. – https://cran.r-project.org/package=FD.
Legras, G., Loiseau, N. and Gaertner, J.-C. 2018. Functional richness: overview of indices and underlying concepts. – Acta Oecol. 87: 34–44.
Leps, J., Bello, F., Lavorel, S. and Berman, S. 2006. Quantifying and interpreting functional diversity of natural communities: practical considerations matter. – Preslia 78: 481–501.
Li, D. 2018. hillR: taxonomic, functional and phylogenetic diversity and similarity through hill numbers. – J. Open Source Softw. 3: 1041.
Magneville, C., Loiseau, N., Albouy, C., Casajus, N., Claverie, T., Escalas, A., Leprieur, F., Maire, E., Mouillot, D. and Villéger, S. 2022. mFD: an R package to compute and illustrate the multiple facets of functional diversity. – Ecography 2022: e05904.
Maire, E., Grenouillet, G., Brosse, S. and Villéger, S. 2015. How many dimensions are needed to accurately assess functional diversity? A pragmatic approach for assessing the quality of functional spaces. – Global Ecol. Biogeogr. 24: 728–740.
Mammola, S., Carmona, C. P., Guillerme, T. and Cardoso, P. 2021. Concepts and applications in functional diversity. – Funct. Ecol. 35: 1869–1885.
Mason, N. W. H., de Bello, F., Mouillot, D., Pavoine, S. and Dray, S. 2013. A guide for using functional diversity indices to reveal changes in assembly processes along ecological gradients. – J. Veg. Sci. 24: 794–806.
McPherson, J. M., Yeager, L. A. and Baum, J. K. 2018. A simulation tool to scrutinise the behaviour of functional diversity metrics. – Methods Ecol. Evol. 9: 200–206.
Mislan, K. A. S., Heer, J. M. and White, E. P. 2016. Elevating the status of code in ecology. – Trends Ecol. Evol. 31: 4–7.
Nowak, L., Kissling, W. D., Bender, I. M. A., Dehling, D. M., Töpfer, T., Böhning-Gaese, K. and Schleuning, M. 2019. Data from: Projecting consequences of global warming for the functional diversity of fleshy-fruited plants and frugivorous birds along a tropical elevational gradient. – Data Dryad Digital Repository.
Pavoine, S. 2020. Adiv: an R package to analyse biodiversity in ecology . – Methods Ecol. Evol. 11: 1106–1112.
Pavoine, S. and Bonsall, M. B. 2011. Measuring biodiversity to explain community assembly: a unified approach. – Biol. Rev. 86: 792–812.
Pavoine, S., Vallet, J., Dufour, A.-B., Gachet, S. and Daniel, H. 2009. On the challenge of treating various types of variables: application for improving the measurement of functional diversity. – Oikos 118: 391–402.
Podani, J. 1999. Extending Gower's general coefficient of similarity to ordinal characters. – Taxon 48: 331–340.
Poisot, T. 2015. Best publishing practices to improve user confidence in scientific software. – Ideas Ecol. Evol. 8: 50–54.
Rao, C. R. 1982. Diversity and dissimilarity coefficients: a unified approach. – Theor. Popul. Biol. 21: 24–43.
rOpenSci, Anderson, B., Chamberlain, S., DeCicco, L., Gustavsen, J., Krystalli, A., Lepore, M., Mullen, L., Ram, K., Ross, N., Salmon, M. and Vidoni, M. 2021. rOpenSci packages: development, maintenance and peer review (ver. 0.6.0). – Zenodo, https://doi.org/10.5281/zenodo.4554776.
Schleuter, D., Daufresne, M., Massol, F. and Argillier, C. 2010. A user's guide to functional diversity indices. – Ecol. Monogr. 80: 469–484.
van der Loo, M. P. J. 2021. The R Journal: A method for deriving information from running R code. – R J. 13: 42–52.
Villéger, S., Mason, N. W. H. and Mouillot, D. 2008. New multidimensional functional diversity indices for a multifaceted framework in functional ecology. – Ecology 89: 2290–2301.
Villéger, S., Grenouillet, G. and Brosse, S. 2013. Decomposing functional β-diversity reveals that low functional β-diversity is driven by low functional turnover in European fish assemblages. – Global Ecol. Biogeogr. 22: 671–681.
White, E. 2015. Some thoughts on best publishing practices for scientific software. – Ideas Ecol. Evol. 8: 55–57.
Wickham, H. 2011. Testthat: get started with testing. – R J. 3: 5–10.
Wickham, H., Hester, J., Chang, W., Müller, K. and Cook, D. 2021. Memoise: memoisation of functions (Manual). – https://CRAN.R-project.org/package=memoi.
Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L. and Teal, T. K. 2017. Good enough practices in scientific computing. – PLoS Comput. Biol. 13: e1005510.
Wüest, R. O., Zimmermann, N. E., Zurell, D., Alexander, J. M., Fritz, S. A., Hof, C., Kreft, H., Normand, S., Cabral, J. S., Szekely, E., Thuiller, W., Wikelski, M. and Karger, D. N. 2020. Macroecology in the age of big data – where to go from here? – J. Biogeogr. 47: 1–12.