R Python, and Ruby clients for GBIF species occurrence data

Scott Chamberlain[*,a], Carl Boettiger[b]

[a] *rOpenSci, Museum of Paleontology, University of California, Berkeley, CA, USA*

[b] *rOpenSci, Department of Enivornmental Science, Policy and Management, University of California, Berkeley, CA, USA*

Abstract

Corresponding Author:

Scott Chamberlain

rOpenSci, Museum of Paleontology, University of California, Berkeley, CA, USA

Email address: scott@ropensci.org

[*] Corresponding author

Email addresses: `scott(at)ropensci.org` (Scott Chamberlain), `carl(at)ropensci.org` (Carl Boettiger)

Background. The number of individuals of each species in a given location forms the basis for many sub-fields of ecology and evolution. Data on individuals, including which species, and where they're found can be used for a large number of research questions. Global Biodiversity Information Facility (hereafter, GBIF) is the largest of these. Programmatic clients for GBIF would make research dealing with GBIF data much easier and more reproducible.

Methods. We have developed clients to access GBIF data for each of the R, Python, and Ruby programming languages: `rgbif`, `pygbif`, `gbifrb`.

Results. For all clients we describe their design and utility, and demonstrate some use cases.

Discussion. Programmatic access to GBIF will facilitate more open and reproducible science - the three GBIF clients described herein are a significant contribution towards this goal.

Introduction

Perhaps the most fundamental element in many fields of ecology is the individual organism. The number of individuals of each species in a given location forms the basis for many sub-fields of ecology and evolution. Some research questions necessitate collecting new data, while others can easily take advantage of existing data. In fact, some ecology fields are built largely on existing data, e.g., macro-ecology (Brown, 1995; Beck et al., 2012).

Data on individuals, including which species, and where they're found, can be used for a large number of research questions. Biodiversity records have been used for a suite of other use cases: validating habitat suitability models with real occurrence data (Ficetola et al., 2014); ancestral range reconstruction (Ferretti et al., 2015; María Mendoza et al., 2015); development of invasive species watch lists (Faulkner et al., 2014); evaluating risk of invasive species spread (Febbraro et al., 2013); and effects of climate change on future biodiversity (Brown et al., 2015).

In addition to wide utility, this data is important for conservation. Biodiversity loss is one of the greatest challenges of our time (Pimm et al., 2014), and some have called this the sixth great mass extinction (Ceballos et al., 2015). Given this challenge there is a great need for data on specimen records, whether collected from live sightings in the field or specimens in museums.

Global Biodiversity Information Facility

There are many online services that collect and maintain specimen records. However, Global Biodiversity Information Facility (hereafter, GBIF, http://www.gbif.org) is the largest collection of biodiversity records globally, currently with 820 million records, roughly 5.9 million taxa, 36,000 datasets from 1,300 publishers (as of 2016-02-09). Many large biodiversity warehouses such as iNaturalist (http://www.inaturalist.org), VertNet (http://vertnet.org), and USGS's Biodiversity Information Serving Our Nation (BISON; http://bison.usgs.ornl.gov) all feed into GBIF.

The most important organizational level in GBIF occurrence data is the occurrence record. The fields in a record vary, but include information about taxonomy (kingdom, phylum, genus, species names) and their identifiers, dataset metadata, and locality information including geospatial position. Going upstream, each record is part of a dataset, where each dataset is submitted by an organization, organizations are organized into nodes, datasets are published through institutions (which may be hosted at another organization), and a network is a group of datasets (managed by GBIF).

Each occurrence record has some taxonomic name associated with it, which itself is linked to a lot of other taxonomic data - including a master taxonomic backbone that integrates taxonomies across many taxonomic authorities.

The organization of GBIF matters because you can navigate GBIF data through these hierarchical organizational levels - it helps to be familiar with the terminology and how each group relates to another.

The clients

Although we discuss libraries for R, Python, and Ruby here, we focus mostly on the R library `rgbif` as it has seen the most developer and user attention, and is the most mature.

rgbif

Herein, we describe the `rgbif` software package (Chamberlain et al.) for working with GBIF data in the R programming environment (R Core Team, 2014). R is a widely used language in academia, as well as non-profit and private sectors. Importantly, R makes it easy to execute all steps of the research process, including data management, data manipulation and cleaning, statistics, and visualization. Thus, an R client for getting GBIF data is a powerful tool to facilitate reproducible research.

The `rgbif` package is nearly completely written in R (a small Javascript library is included for reading well known text (Herring, 2011)), uses an MIT license to maximize use everywhere. `rgbif` is developed publicly on GitHub at https://github.com/ropensci/rgbif, where development versions of the package can be installed, and bugs and feature requests reported. Stable versions of `rgbif` can be installed from CRAN, the distribution network for R packages. `rgbif` is part of the rOpenSci project (http://ropensci.org), a developer network making R software to facilitate reproducible research.

pygbif

`pygbif` (Chamberlain) is a Python library for working with GBIF data in the Python programming environment. Python is a general purpose programming language used widely in all sectors, and for all parts of software development including server and client side use cases. Python is used exclusively in some scientific disciplines (e.g., astronomy), and has partial usage in other disciplines. A Python client for GBIF data is an important tool given the even wider usage of Python than R, though maybe slightly less than R for ecology/biology disciplines.

```
pip install pygbif
```

```
import pygbif
```

The `pygbif` library is less mature and complete than the R package. It also uses an MIT license to maximize use everywhere. `pygbif` is developed publicly on GitHub at https://github.com/sckott/pygbif, where development versions of the package can be installed, and bugs and feature requests reported. Stable versions of `pygbif` can be installed from pypi, the distribution network for Python libraries.

gbifrb

`gbifrb` (Chamberlain) is a library for working with GBIF data in the Ruby programming environment. Like Python, Ruby is a general purpose programming language used widely in all sectors. Unlike Python, Ruby is not used extensively in scientific disciplines. However, a Ruby client for GBIF data can be an important tool given how widely Ruby is used for web and web service development.

```
gem install gbifrb
```

```
require 'gbifrb'
```

The `gbifrb` library is less mature and complete than the R and Python libraries. It also uses an MIT license to maximize use everywhere. `gbifrb` is developed publicly on GitHub at https://github.com/sckott/gbifrb, where development versions of the package can be installed, and bugs and feature requests reported. Stable versions of `gbifrb` can be installed from [Rubygems][gemgbif], the distribution network for Ruby libraries.

Library interfaces

`rgbif`, `pygbif`, and `gbifrb` are designed following the GBIF Application Programming Interface, or API. The GBIF API has four major components: registry, taxonomic names, occurrences, and maps. We also include functions to interface with the OAI-PMH GBIF service; only dataset (registry) information is available via this service, however. An interface to the GBIF maps API is in development for `rgbif`, but is non-existent for both `pygbif` and `gbifrb`. All three libraries have a suite of functions dealing with each of registry, taxonomic, names, and occurrences - we'll go through each in turn describing design of the user interface and example usage.

GBIF headers

With each request `rgbif`, `pygbif`, `gbifrb` make to GBIF's API, we send request headers that tell GBIF what library the request is coming from, including what version of the library. This helps GBIF know what proportion of requests are coming from which library, and therefore from R vs. Python vs. Ruby; this information is helpful for GBIF in thinking about how people are using GBIF data.

Registry

The GBIF registry API services are spread across five sets of functions via the main GBIF API:

- Datasets
- Installations
- Networks
- Nodes
- Organizations

Dataset information in general is available via the OAI-PMH service, functions in `rgbif` prefixed with `gbif_oai_`, but not available in `pygbif` or `gbifrb` yet.

Datasets are owned by organizations. Organizations are endorsed by nodes to share datasets with GBIF. Datasets are published through institutions, which may be hosted at another organization. A network is a group of datasets (managed by GBIF). Datasets are the units that matter the most with respect to registry information, while installations, networks, nodes, and organizations are simply higher level organizational structure.

Datasets

Dataset functions include search, dataset metadata retrieval, and dataset metrics. Searching for datasets is an important part of the discovery process. One can search for datasets on the GBIF web portal. However, programmatic searching using any of these libraries is more powerful. Identifying datasets appropriate for a research question is helpful as you can get metadata for each dataset, and track down dataset specific problems, if any.

The `dataset_search()` function in `rgbif` is one way to search for datasets. Here, we search for the term "oregon", which finds any datasets that have words matching that term.

```r
res <- dataset_search(query = "oregon")
res$data$datasetTitle[1:10]
#>  [1] "Oregon State Ichthyology Collection"
#>  [2] "Oregon State University Herpetological Collection"
#>  [3] "Mygalomorph spiders from southwestern Oregon, USA, with descriptions of four new specie
#>  [4] "A new species of Helobdella (Hirudinida: Glossiphoniidae) from Oregon, USA"
#>  [5] "Annotated Checklist of the large branchiopod crustaceans of Idaho, Oregon and Washingto
#>  [6] "A new species of Chrysobothris Eschscholtz from Oregon and Washington, with notes on ot
#>  [7] "Three new species of Grylloblatta Walker (Insecta: Grylloblattodea: Grylloblattidae), f
#>  [8] "A new species of Cladotanytarsus (Lenziella) from Oregon supports the systematic concep
#>  [9] "A new monster from southwest Oregon forests: Cryptomasterbehemoth sp. n. (Opiliones, La
#> [10] "Two new species of Fluminicola (Caenogastropoda, Lithoglyphidae) from southwest Oregon,
```

See also `datasets()` and `dataset_suggest()` in `rgbif` for searching for datasets.

In Python, we can similarly search for datasets. Here, search for datasets of type `OCCURRENCE`:

```python
from pygbif import registry
registry.datasets(type="OCCURRENCE")
```

In Ruby, we can do the same. Here, search for datasets of type `OCCURRENCE`:

```ruby
require 'gbifrb'
registry = Gbif::Registry
registry.datasets(type: "OCCURRENCE")
```

Dataset metrics. Dataset metrics are another useful way of figuring out what datasets you may want to use. One drawback is that these metrics data are only available for datasets of type *checklist*, but there are quite a lot of them (21687).

Here, in R we search for dataset metrics for a single dataset, with uuid `ec93a739-1681-4b04-b62f-3a687127a17f`, a checklist of the ants (Hymenoptera: Formicidae) of the World.

```r
res <- dataset_metrics(uuid='ec93a739-1681-4b04-b62f-3a687127a17f')
data.frame(rank = names(res$countByRank),
           count = unname(unlist(res$countByRank)))
```

rank	count
SPECIES	13710
SUBSPECIES	3234
GENUS	726
TRIBE	53
SUBFAMILY	20
FAMILY	2
KINGDOM	1
PHYLUM	1
CLASS	1
ORDER	1

And in Python, get metrics for the same dataset as above:

```python
from pygbif import registry
registry.dataset_metrics(uuid='ec93a739-1681-4b04-b62f-3a687127a17f')
```

The same in Ruby:

```ruby
require 'gbifrb'
registry = Gbif::Registry
registry.dataset_metrics(uuid: 'ec93a739-1681-4b04-b62f-3a687127a17f')
```

Networks, nodes, and installations

Networks, nodes and installations are at a higher level of organization above datasets, but can be useful if you want to explore data from given organizations. Here, in R we search for the first 10 GBIF networks, returning just the title field.

```r
networks(limit = 10)$data$title
#>  [1] "GBIF Backbone Sources"
#>  [2] "Canadensys"
#>  [3] "Southwest Collections of Arthropods Network (SCAN)"
#>  [4] "VertNet"
#>  [5] "Dryad"
#>  [6] "GBIF Network"
#>  [7] "The Knowledge Network for Biocomplexity (KNB) "
#>  [8] "Online Zoological Collections of Australian Museums (OZCAM)"
#>  [9] "Catalogue of Life"
#> [10] "Ocean Biogeographic Information System (OBIS)"
```

And in Python:

```python
from pygbif import registry
registry.networks(limit = 10)
```

And in Ruby:

```ruby
require 'gbifrb'
registry = Gbif::Registry
registry.networks(limit: 10)
```

Taxonomic names

The GBIF taxonomic names API services are spread across five functions in `rgbif`:

- Search GBIF name backbone - `name_backbone()`
- Search across all checklists - `name_lookup()`
- Quick name lookup - `name_suggest()`
- Name usage of a name according to a checklist - `name_usage()`
- GBIF name parser - `parsenames()`

`pygbif` and `gbifrb` have all the same functions, except the name parser goes by `name_parser()` in `pygbif` and `gbifrb`.

The goal of these name functions is often to settle on a taxonomic name known to GBIF's database. This serves two purposes: 1) when referring to a taxonomic name, you can point to a URI on the Internet, and 2) you can search for metadata on a taxon, and occurrences of that taxon in GBIF.

Taxonomic names are particularly tricky. Many different organizations have their own unique codes for the same taxonomic names, and some taxonomic groups have preferred sources for the definitive names for that group. That's why it's best to determine what name GBIF uses, and its associated identifier, for the taxon of interest instead of simply searching for occurrences with a taxonomic name.

When searching for occurrences (see below) you can search by taxonomic name (and other filters, e.g., taxonomic rank), but you're probably better off figuring out the taxonomic key in the GBIF backbone taxonomy, and using that to search for occurrences. The `taxonkey` parameter in the GBIF occurrences API expects a GBIF backbone taxon key.

GBIF Backbone

The GBIF backbone taxonomy is used in GBIF to have a consistent way to refer to taxonomic names throughout their services. The backbone has 5869207 unique names and 2818534 species names. The backbone taxonomy is also a dataset with key `d7dddbf4-2cf0-4f39-9b2a-bb099caae36c` (https://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c).

We can search the backbone taxonomy with the function `name_backbone()` in all thee clients. Here, we're searching for the name *Poa*, restricting to genera, and the family *Poaceae*, in R

```r
res <- name_backbone(name='Poa', rank='genus', family='Poaceae')
res[c('usageKey', 'kingdom')]
#> $usageKey
#> [1] 2704173
#>
#> $kingdom
#> [1] "Plantae"
```

And in Python

```python
from pygbif import species
res = species.name_backbone(name='Poa', rank='genus', family='Poaceae')
[ res[x] for x in ['usageKey', 'kingdom'] ]
```

And in Ruby

```ruby
require 'gbifrb'
species = Gbif::Species
res = species.name_backbone(name: 'Poa', rank: 'genus', family: 'Poaceae')
res.select { |k,v| k.match(/usageKey|kingdom/) }
```

Name searching

One of the quickest ways to search for names is using `name_suggest()`, which does a very quick search and returns minimal data. Here, we're searching for the query term *Pum*, and we get back many names:

```r
name_suggest(q='Pum', limit = 6)
```

key	canonicalName	rank
2142856	Althepus pum	SPECIES
8589398	Pumiliopimoidae	FAMILY
8783253	Pumililema	GENUS
4823360	Pumiliopareia	GENUS
4635949	Pumilina	GENUS
4648228	Pumilopaurus	GENUS

The same in Python

```python
from pygbif import species
species.name_suggest(q='Pum', limit = 6)
```

And in Ruby

```
require 'gbifrb'
species = Gbif::Species
species.name_suggest(q: 'Pum', limit: 6)
```

With these results, you can then proceed to search for occurrences with the taxon key(s), or drill down further with other name searching functions to get the exact taxon of interest.

Occurrences

GBIF provides two ways to get occurrence data: through the `/occurrence/search` route (see `occ_search` in `rgbif`, `occurrences.search` in `pygbif`, `Occurrences.search` in gbifrb), or via the `/occurrence/download` route (many functions, see below).

`occ_search()`/`occurrences.search`/`Occurrences.search` are the main functions for the search route, and are more appropriate when you want less data, while the download functions are more appropriate for larger data requests.

Small vs. large amounts of data of course is all relative. GBIF imposes for any given search a limit of 200,000 records in the search service, after which point you can't download any more records for that search. However, you can download more records for different searches.

We think the search service is still quite useful for many people even given the 200,000 limit. For those that need more data, we have created a similar interface in the download functions that should be easy to use with minimal work. Users should take note that using the download service has a few extra steps to get data into R, but is straight-forward.

The download service, like the occurrence search service, is rate-limited. That is, you can only have one to three downloads running simultaneously for your user credentials. However, simply check when a download job is complete, then you can start a new download request. See "Queuing Download Requests" below for help automating many download requests in R.

Download API

The download API syntax is similar to the occurrence search API in that the same parameters are used, but the way in which the query is defined is different. For example, in the download API you can do greater than searches (i.e., `latitude > 50`), whereas you cannot do that in the occurrence search

API. Thus, unfortunately, we couldn't make the query interface exactly the same for both search and download functions.

Using the download service can consist of as few as three steps: 1) Request data via a search; 2) Download data; 3) Import data into R.

Request data download given a query. Here, we search for the taxon key `3119195`, which is the key for *Helianthus annuus* (http://www.gbif.org/species/3119195).

```
occ_download('taxonKey = 3119195')
#> <<gbif download>>
#>   Username: xxxx
#>   E-mail: xxxx
#>   Download key: 0000840-150615163101818
```

You can check on when the download is ready using the functions `occ_download_list()` and `occ_download_meta()`. When it's ready use `occ_download_get()` to download the dataset to your computer.

```
(res <- occ_download_get("0000840-150615163101818", overwrite = TRUE))
#> <<gbif downloaded get>>
#>   Path: ./0000840-150615163101818.zip
#>   File size: 3.19 MB
```

What's printed out above is a very brief summary of what was downloaded, the path to the file, and its size (in human readable form).

Next, read the data in to R using the function `occ_download_import()`.

```
library("dplyr")
dat <- occ_download_import(res)
dat %>%
  select(gbifID, decimalLatitude, decimalLongitude)
#>       gbifID abstract accessRights accrualMethod accrualPeriodicity accrualPolicy alternative
#> 1  725767384       NA                            NA                 NA            NA          NA
```

```
#> 2  725767447       NA                          NA                  NA             NA            NA
#> 3  725767450       NA                          NA                  NA             NA            NA
#> 4  725767513       NA                          NA                  NA             NA            NA
#> 5  725767546       NA                          NA                  NA             NA            NA
#> 6  725767579       NA                          NA                  NA             NA            NA
#> 7  725767609       NA                          NA                  NA             NA            NA
#> 8  725767645       NA                          NA                  NA             NA            NA
#> 9  725767678       NA                          NA                  NA             NA            NA
#> 10 725767681       NA                          NA                  NA             NA            NA
#> ..       ...      ...          ...            ...                 ...            ...           ...
#> Variables not shown: available (lgl), bibliographicCitation (chr), conformsTo (lgl), contribu
#>      coverage (lgl), created (chr), creator (lgl), date (lgl), dateAccepted (lgl), dateCopyri
#>      (lgl), dateSubmitted (lgl), description (lgl), educationLevel (lgl), extent (lgl), forma
#>      hasFormat (lgl), hasPart (lgl), hasVersion (lgl), identifier (chr), instructionalMethod
```

In Python

```
from pygbif import occurrences as occ
occ.download('taxonKey = 3119195')
(res = occ.download_get("0000840-150615163101818", overwrite = True))
```

We don't have `pygbif` functionality at the moment for importing data, but it's coming soon.

The Ruby library `gbifrb` does not yet have occurrence download functionality.

Downloaded data format. The downloaded dataset from GBIF is a Darwin Core Archive (DwC-A), an internationally recognized biodiversity informatics standard (http://rs.tdwg.org/dwc/). The DwC-A downloaded is a compressed folder with a number of files, including metadata, citations for each of the datasets included in the download, and the data itself, in separate files for each dataset as well as one single `.txt` file. In `rgbif::occ_download_import()`, we simply fetch data from the `.txt` file. If you want to dig into the metadata, citations, etc., it is easily accessible from the folder on your computer.

Search API

The search API follows the GBIF API and is broken down into the following functions:

- Get a single numeric count of occurrences - `rgbif`: `occ_count()` / `pygbif`: `occurrences.count` / `gbifrb`: `Occurrences.count`
- Search for occurrences - `rgbif`: `occ_search()` / `pygbif`: `occurrences.search` / `gbifrb`: `Occurrences.search`
- A simplified and optimized version of `rgbif`: `occ_search()` or `occ_data()` / none / none
- Get occurrences by occurrence identifier - `rgbif`: `occ_get()` / `pygbif`: `occurrences.get` / `gbifrb`: `Occurrences.get`
- Get occurrence metadata - `rgbif`: `occ_metadata()` / `pygbif`: various / `gbifrb`: various

Search for occurrences. The main search work-horse is `occ_search()`. This function allows very flexible search definitions. In addition, this function does paging internally, making it such that the user does not have worry about the 300 records per request limit - but of course we can't go over the 200,000 maximum limit.

The output of `occ_search()` presents a compact `data.frame` so that no matter how large the `data.frame`, the output is easily assessed because only a few of the records (rows) are shown, only a few columns are shown (with others shown in name only), and metadata is shown on top of the `data.frame` to indicate data found and returned, media records found, unique taxonomic hierarchies returned, and the query executed.

The output of these examples, except one, aren't shown.

Search by species name, using `name_backbone()` first to get key

R

```r
library(rgbif)
(key <- name_suggest(q = 'Helianthus annuus', rank = 'species')$key[1])
#> [1] 9206251
occ_search(taxonKey = key, limit = 2)
#> Records found [17858]
#> Records returned [2]
#> No. unique hierarchies [1]
#> No. media records [1]
```

```
#> No. facets [0]
#> Args [limit=2, offset=0, taxonKey=9206251, fields=all]
#> # A tibble: 2 x 75
#>                name        key decimalLatitude decimalLongitude
#>               <chr>      <int>           <dbl>            <dbl>
#> 1 Helianthus annuus 1433793045        59.66859         16.54257
#> 2 Helianthus annuus 1434024463        63.71622         20.31247
#> # ... with 71 more variables: issues <chr>, datasetKey <chr>,
#> #   publishingOrgKey <chr>, publishingCountry <chr>, protocol <chr>,
#> #   lastCrawled <chr>, lastParsed <chr>, crawlId <int>, extensions <chr>,
#> #   basisOfRecord <chr>, ...
```

Python

```python
from pygbif import species
from pygbif import occurrences as occ
key = species.name_suggest(q = 'Helianthus annuus', rank = 'species')['data'][0]['key']
occ.search(taxonKey = key, limit = 2)
```

Ruby

```ruby
require 'gbifrb'
species = Gbif::Species
occ = Gbif::Occurrences
key = species.name_suggest(q: 'Helianthus annuus', rank: 'species')['data'][0]['key']
occ.search(taxonKey: key, limit: 2)
```

Instead of getting a taxon key first, you can search for a name directly

R

```r
occ_search(scientificName = 'Ursus americanus')
```

Python

```
occ.search(scientificName = 'Ursus americanus')
```

Ruby

```
occ.search(scientificName: 'Ursus americanus')
```

Search for many species

R

```
splist <- c('Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa')
keys <- sapply(splist, function(x) name_suggest(x)$key[1], USE.NAMES = FALSE)
occ_search(taxonKey = keys, limit = 5, return = 'data')
```

Python

```
from pygbif import species
from pygbif import occurrences as occ
splist = ['Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa']
keys = [ species.name_suggest(x)['data'][0]['key'] for x in splist ]
occ.search(taxonKey = keys, limit = 5)
```

Ruby

```
species = Gbif::Species
occ = Gbif::Occurrences
splist = ['Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa']
keys = [ species.name_suggest(x)['data'][0]['key'] for x in splist ]
occ.search(taxonKey: keys, limit: 5)
```

Spatial search, based on well known text format (Herring, 2011), or a bounding box set of four co-ordinates. The well known text string and the bounding box in the below example specify the same rectangular area in California, centering approximately on Sacramento. Whereas the bounding box format requires `longitude SW corner, latitude SW corner, longitude NE corner, latitude NE corner`, the well known text string requires an extra long/lat pair to close the polygon.

R

```r
# well known text
wkt <- 'POLYGON((-122.6 39.9,-120.0 39.9,-120.0 37.9,-122.6 37.9,-122.6 39.9))'
occ_search(geometry = wkt, limit = 20)
# bounding box
occ_search(geometry = c(-122.6,37.9,-120.0,39.9), limit = 20)
```

Python

```python
from pygbif import occurrences as occ
# well known text
occ.search(geometry = 'POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 30.1 10.1))', limit = 20)
# bounding box
occ.search(geometry = '-125.0,38.4,-121.8,40.9', limit = 20)
```

Ruby

```ruby
occ = Gbif::Occurrences
# well known text
occ.search(geometry: 'POLYGON((30.1 10.1, 10 20, 20 40, 40 40, 30.1 10.1))', limit: 20)
# bounding box
occ.search(geometry: '-125.0,38.4,-121.8,40.9', limit: 20)
```

Get only occurrences with lat/long data using the `hasCoordinate` parameter

R

```r
occ_search(hasCoordinate = TRUE, limit = 5)
```

Python

```python
from pygbif import occurrences as occ
occ.search(hasCoordinate = True, limit = 5)
```

Ruby

```
occ = Gbif::Occurrences
occ.search(hasCoordinate: true, limit: 5)
```

Get only those occurrences with spatial issues. Spatial issues are a set of issues that are returned in the `issues` field. They each indicate something different about that record. For example, the issue `COUNTRY_COORDINATE_MISMATCH` indicates that the interpreted occurrence coordinates fall outside of the indicated country. You can see how that might be useful when it comes to cleaning your data prior to analysis/visualization.

R

```
occ_search(hasGeospatialIssue = TRUE, limit = 5)
```

Python

```
from pygbif import occurrences as occ
occ.search(hasGeospatialIssue = True, limit = 5)
```

Ruby

```
occ = Gbif::Occurrences
occ.search(hasGeospatialIssue: true, limit: 5)
```

Data cleaning. GBIF provides optional data issues with each occurrence record. These issues fall into many different pre-defined classes, covering issues with taxonomic names, geographic data, and more (see `rgbif::occ_issues_lookup()` to find out more information on GBIF issues; and the same data on GBIF's development site).

`rgbif::occ_issues()` provides a way to easily filter data downloaded via `rgbif::occ_search()` based on GBIF issues.

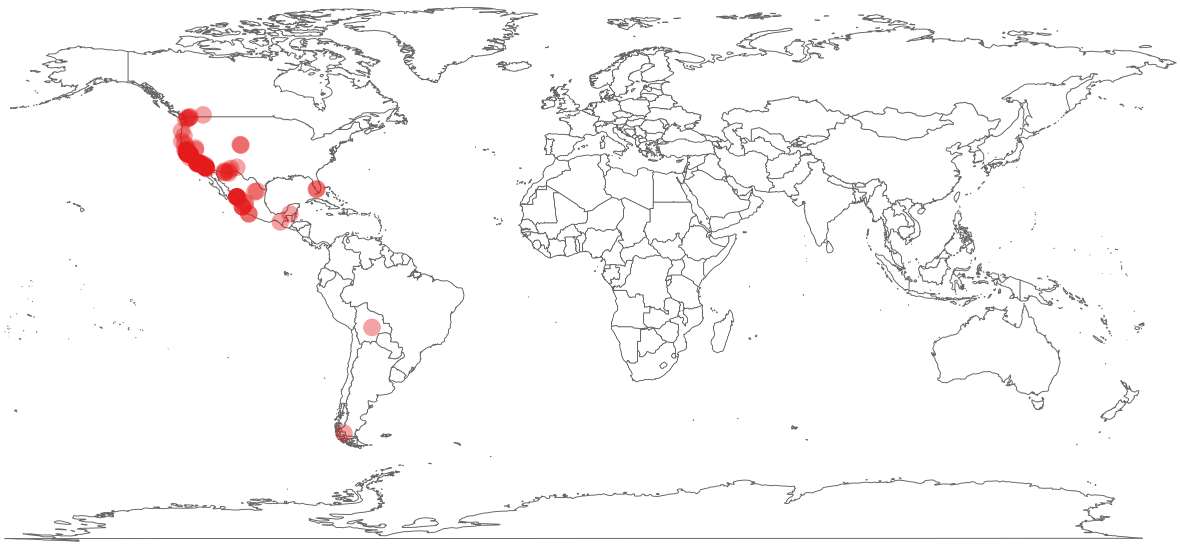
```
out <- occ_search(issue = 'DEPTH_UNLIKELY', limit = 500)
NROW(out)
#> [1] 5
out %>% occ_issues(-cudc) %>% .$data %>% NROW
#> [1] 0
```

There's no equivalent interface in `pygbif` or `gbifrb` yet.

Mapping

An obvious downstream use case for species occurrence data is to map the data. `rgbif` per se is largely not concerned with making this easier, although we do have a simple wrapper around `ggplot2` to make it easy to get a quick plot of occurrence data. For example, here we plot 100 occurrences for *Puma concolor*.

```r
key <- name_backbone(name='Puma concolor')$speciesKey
dat <- occ_search(taxonKey = key, limit = 100, hasCoordinate = TRUE)
gbifmap(dat$data)
```



Another package, mapr, is the perfect mapping companion to `rgbif`. It has convenient functions for handling input data from `rgbif`, `spocc`, or arbitrary data.frame's, and output plots for base plots, `ggplot2`, `ggmap` (`ggplot2` with map layers underneath), and interactive maps on GitHub gists or with Leaflet.js.

There's no equivalent interface in `pygbif` or `gbifrb`.

GBIF data in other R packages

We discuss usage of GBIF data in other R packages throughout the manuscript, but provide a synopsis here for clarity.

taxize

Some of the GBIF taxonomic services are also available in taxize, an R package that focuses on getting data from taxonomic data sources on the web. For example, with `get_gbifid()` one can get GBIF IDs used for a set of taxonomic names - then use those IDs in other functions in `taxize` to get additional information, like taxonomically downstream children.

spocc

GBIF occurrence data is available in the R package spocc via `rgbif`. `spocc` is a unified interface for fetching species occurrence data from many sources on the web. For example, a user can collect occurrence data from GBIF, iDigBio, and iNaturalist, and easily combine them, then use other packages to clean and visualize the data.

R vs. Python vs. Ruby

Both R and Python are commonly used in science, and can be used for similar tasks. Python, however, is a more general programming language, and can be used in more contexts than R can be used in. Ruby is used very little in science; but, like Python, Ruby is very widely used as a general purpose programming language, with heavy use in web development and web services.

The three clients can do a lot of the same tasks. We envision `rgbif` being more common in workflows of academics asking research questions, whereas `pygbif` and `gbifrb` can do that as well, but may be more easily used in a website.

The R client `rgbif` has had much more development time than `pygbif` and `gbifrb`, but with time `pygbif` and `gbifrb` will become equally mature.

Use cases

The following are three use cases for the R library `rgbif`: niche modeling, spatial change in biodiversity, and distribution mapping.

Ecological niche modeling

In this example, we plot actual occurrence data for *Bradypus* species against a single predictor variable, BIO1 (annual mean temperature). This is only one step in a species distribution modelling workflow.

This example can be done using BISON data as well with our rbison package.

Load libraries

```
library("sp")
library("rgbif")
library("dismo")
library("maptools")
library("dplyr")
```

Raster files

Make a list of files that are installed with the dismo package, then create a rasterStack from these

```
files <- list.files(paste(system.file(package = "dismo"), "/ex", sep = ""),
                    "grd", full.names = TRUE)
predictors <- stack(files)
```

Get world boundaries

```
data(wrld_simpl)
```

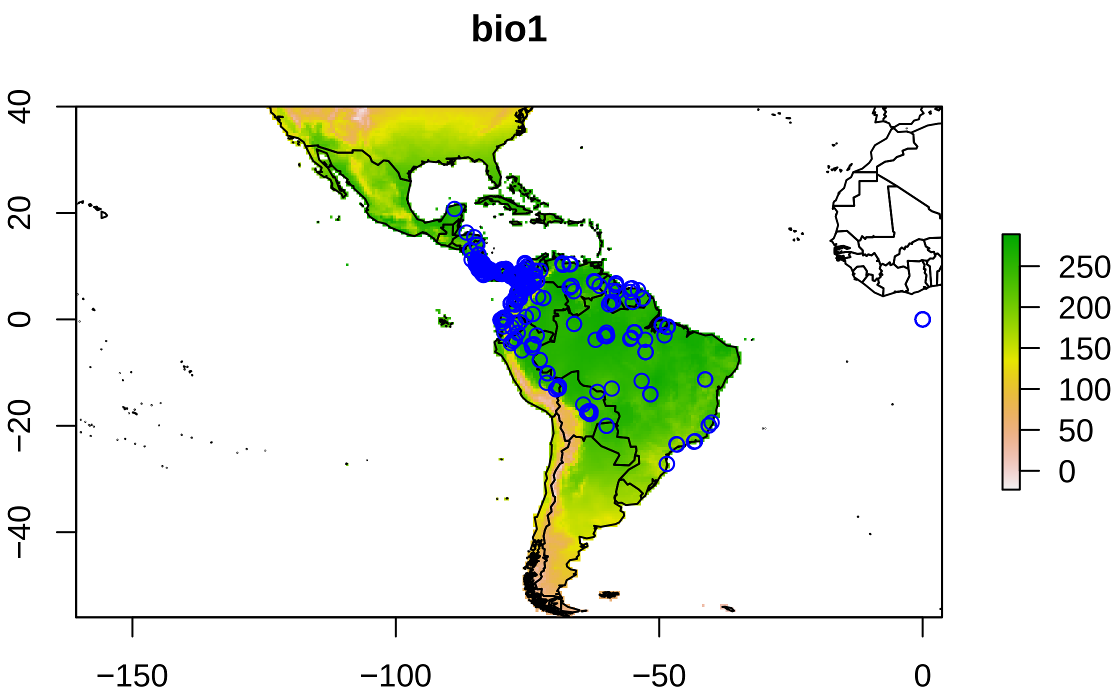
Get GBIF data using the rOpenSci package rgbif

```
nn <- name_lookup("bradypus*", rank = "species")
nn <- na.omit(unique(nn$data$nubKey))
df <- occ_search(taxonKey = nn, hasCoordinate = TRUE, limit = 500)
df_data <- df[ sapply(df, function(x) any(class(x$data) %in% "tbl_df")) ]
df_data <- dplyr::bind_rows(lapply(df_data, "[[", "data"))
df2 <- df_data %>% dplyr::select(decimalLongitude, decimalLatitude)
```

Plot

(1) Add raster data, (2) Add political boundaries, (3) Add the points (occurrences)

```r
plot(predictors, 1)
plot(wrld_simpl, add = TRUE)
points(df2, col = "blue")
```



Biodiversity in big cities

In this example, we collect specimen records across different cities using GBIF data from the `rgbif` package.

Load libraries

```r
devtools::install_github("ropensci/wicket", force = TRUE)
```

```r
library("rgbif")
library("ggplot2")
```

```r
library("plyr")
library("httr")
library("RColorBrewer")
library("wicket")
```

Get bounding boxes for some cites

Bounding lat/long data is from https://raw.github.com/amyxzhang/boundingbox-cities/master/boundbox.txt.

```r
url <- 'https://raw.githubusercontent.com/amyxzhang/
boundingbox-cities/master/boundbox.txt'
rawdat <- content(GET(sub("\n", "", url)), as = "text")
dat <- read.table(
  text = rawdat, header = FALSE,
  sep = "\t", col.names = c("city","minlat","maxlon","maxlat","minlon"),
  stringsAsFactors = FALSE)
dat <- data.frame(
  city = dat$city, minlon = dat$minlon,
  minlat = dat$minlat, maxlon = dat$maxlon,
  maxlat = dat$maxlat,
  stringsAsFactors = FALSE
)
```

A helper function to get count data. GBIF has a count API, but we can't use that with a geometry search as that API doesn't support geospatial search. We can however use the search API via `occ_search()` and set `limit = 1` so that we

```r
getdata <- function(x){
  coords <- as.numeric(x[c('minlon','minlat','maxlon','maxlat')])
  wkt <- wicket::wkt_correct(wicket::bounding_wkt(values = coords))
  num <- occ_search(geometry = wkt, limit = 1)$meta$count
  data.frame(
```

```
    city = x['city'],
    richness = num,
    stringsAsFactors = FALSE
  )
}
```

```
out <- apply(dat, 1, getdata)
```

Merge to original table

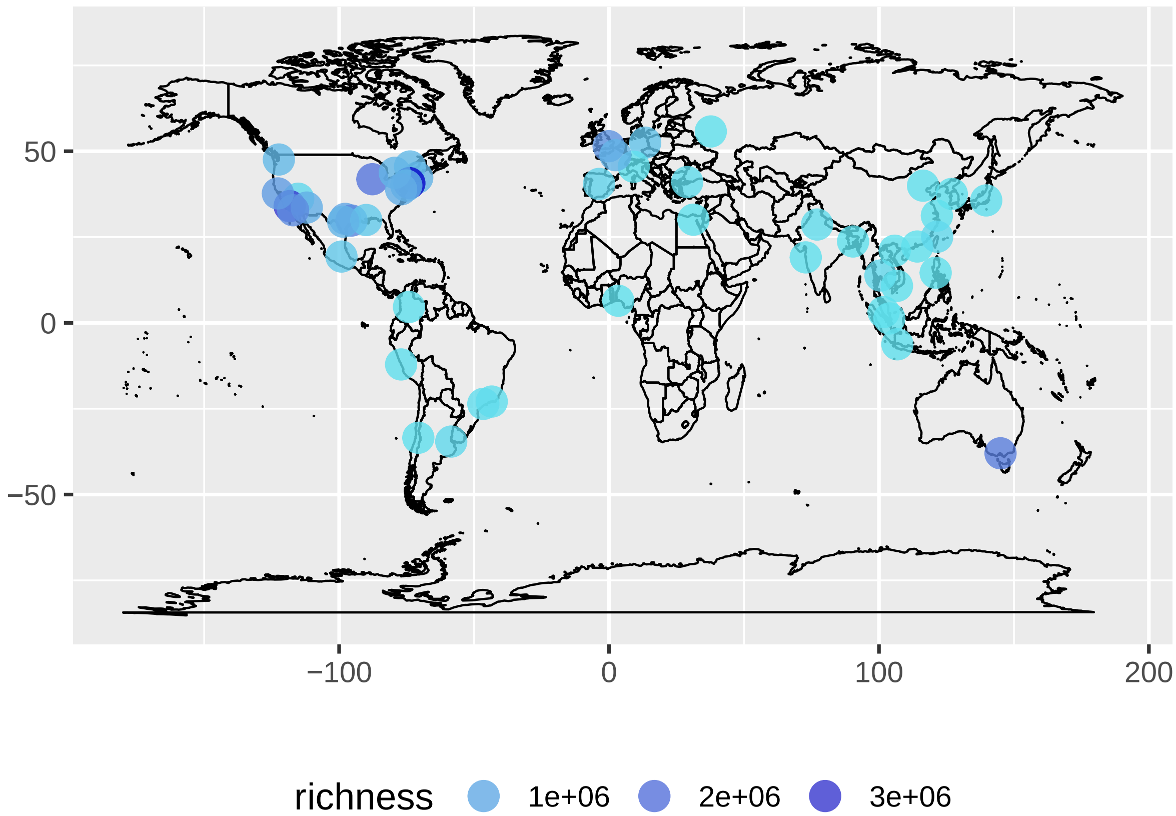
```
out <- merge(dat, ldply(out), by = "city")
```

Add centroids from bounding boxes

```
out <- transform(out, lat = (minlat + maxlat)/2, lon = (minlon + maxlon)/2)
```

Plot data

```
mapp <- map_data('world')
ggplot(mapp, aes(long, lat)) +
  geom_polygon(aes(group=group), fill="white", alpha=0, color="black", size=0.4) +
  geom_point(data=out, aes(lon, lat, color=richness), size=5, alpha=0.8) +
  scale_color_continuous(low = "#60E1EE", high = "#0404C8") +
  labs(x="", y="") +
  theme_grey(base_size=14) +
  theme(legend.position = "bottom", legend.key = element_blank()) +
  guides(color = guide_legend(keywidth = 2))
```



Valley oak occurrence data comparison

This example comes from Antonio J. Perez-Luque who shared his plot on Twitter. Antonio compared the occurrences of Valley Oak (*Quercus lobata*) from GBIF to the distribution of the same species from the Atlas of US Trees.

Load libraries

```
library('rgbif')
library('raster')
library('sp')
library('sf')
library('rgeos')
library('scales')
library('rnaturalearth')
```

Get GBIF Data for Fraxinus excelsior

```r
keyFe <- name_backbone(name = 'Fraxinus excelsior', kingdom = 'plants')$speciesKey
dat.Fe <- occ_search(taxonKey = keyFe, return = 'data', limit = 10000L)
```

Get Distribution map of F. excelsior European Forest Genetic Resources Programme

From http://www.euforgen.org/species/fraxinus-excelsior/. And save shapefile in same directory

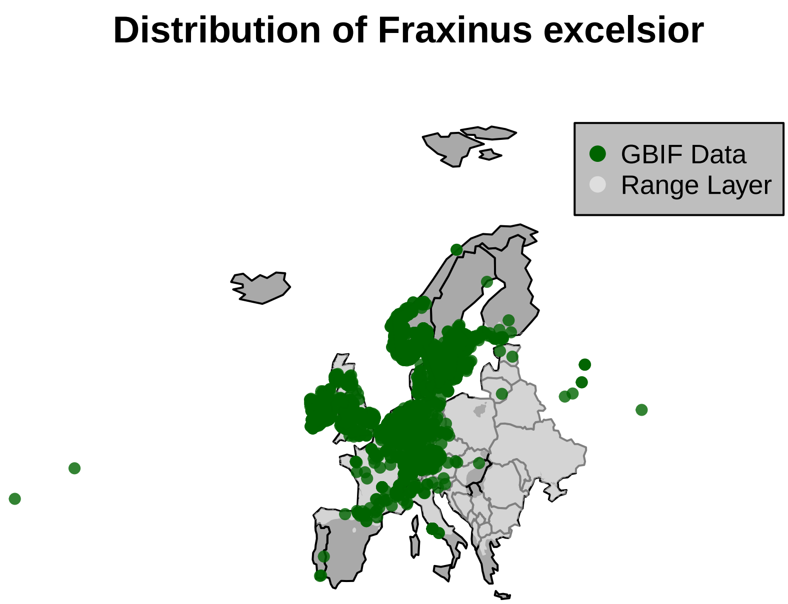
```r
url <- 'http://www.euforgen.org/fileadmin/templates/euforgen.org/upload/Documents/Maps/Shapefile
tmp <- tempdir()
download.file(url, destfile = "fraxinus_excelsior.zip")
unzip("fraxinus_excelsior.zip", exdir = tmp)
fe <- sf::read_sf(file.path(tmp, "Fraxinus_excelsior_EUFORGEN.shp"))
```

Get Elevation data of US

```r
eur <- rnaturalearth::ne_countries(continent = "europe", type = "map_units")
eur1 <- eur[eur$sovereignt != "Russia", ]
```

Plot map

```r
plot(eur1, col = "darkgrey", legend = FALSE,
     main = 'Distribution of Fraxinus excelsior')
# add distribution range layer
plot(fe, add = TRUE, col = alpha("white", 0.5), border = FALSE)
# add Gbif presence points
points(dat.Fe$decimalLongitude, dat.Fe$decimalLatitude,
       cex = .7, pch = 19, col = alpha("darkgreen", 0.8))
legend(x = 38, y = 81, c("GBIF Data", "Range Layer"), pch = 19, bg = "grey",
       col = c('darkgreen', alpha("white", 0.5)), pt.cex = 1, cex = .8)
```



Conclusions and future directions

The `rgbif`, `pygbif`, and `gbibrb` libraries provide programmatic interfaces to GBIF's application programming interface (API) - a powerful tool for working with species occurrence data, and facilitating reproducible research. In fact, the `rgbif` package has already been used in more than 20 scholarly publications (as of 2016-08-10), including (Amano, Lamming & Sutherland, 2016, Bartomeus et al. (2013), Barve (2014), Bone et al. (2015), Collins et al. (2015), Drozd & Šipoš (2013), Kong, Huang & Duan (2015), Richardson, Roux & Wilson (2015), Turner, Fréville & Rieseberg (2015), Verheijen et al. (2015), Zizka & Antonelli (2015), Butterfield et al. (2016), Dellinger et al. (2015), Feitosa et al. (2015), Malhado et al. (2015), Werner et al. (2015), Robertson, Visser & Hui (2016), Davison et al. (2015), Janssens et al. (2016)).

The `rgbif` package is relatively stable, and should not have many breaking changes unless necessitated due to changes in the GBIF API. However, it will gain function(s) to work with the maps API in the near future.

The `pygbif` and `gbifrb` libraries are in early development, and will greatly benefit from any feedback and use cases.

One area of focus in the future is to attempt to solve many use cases that have been brought up with respect to GBIF data. For example, some specimens are included in GBIF that are located in botanical gardens. For many research questions, researchers are interested in "wild" type occurrences, not those in human curated scenarios. Making removal of these occurrences easy would be very useful, but is actually quite a hard problem. There are many other problems like this, for which these three libraries will help in making more efficient and reproducible.

Acknowledgments

This project was supported in part by the Alfred P Sloan Foundation (Grant No. G-2014-13485), and in part by the Helmsley Foundation (Grant No. 2016PG-BRI004).

Data Accessibility

All scripts and data used in this paper can be found in the permanent data archive Zenodo under the digital object identifier (DOI). This DOI corresponds to a snapshot of the GitHub repository at https://github.com/sckott/gbifms. Software can be found at https://github.com/ropensci/rgbif, https://github.com/sckott/pygbif, and https://github.com/sckott/gibfrb, all under MIT licenses. We thank all the users that have used `rgbif`, `pygbif`, and `gbifrb` and have given feedback and reported bugs. In addition, we greatly appreciate all the contributors to the three libraries, found at https://github.com/ropensci/rgbif/graphs/contributors, https://github.com/sckott/pygbif/graphs/contributors, and https://github.com/sckott/gbifrb/graphs/contributors.

References

Amano T., Lamming JDL., Sutherland WJ. 2016. Spatial gaps in global biodiversity information and the role of citizen science. *BioScience* 66:393–400.

Bartomeus I., Park MG., Gibbs J., Danforth BN., Lakso AN., Winfree R. 2013. Biodiversity ensures plant-pollinator phenological synchrony against climate change. *Ecology Letters* 16:1331–1338.

Barve V. 2014. Discovering and developing primary biodiversity data from social networking sites: A novel approach. *Ecological Informatics* 24:194–199.

Beck J., Ballesteros-Mejia L., Buchmann CM., Dengler J., Fritz SA., Gruber B., Hof C., Jansen F., Knapp S., Kreft H., Schneider A-K., Winter M., Dormann CF. 2012. Whats on the horizon for macroecology? *Ecography* 35:673–683.

Bone RE., Smith JAC., Arrigo N., Buerki S. 2015. A macro-ecological perspective on crassulacean acid metabolism (CAM) photosynthesis evolution in afro-madagascan drylands: Eulophiinae orchids as a case study. *New Phytologist* 208:469–481.

Brown JH. 1995. *Macroecology*. University of Chicago Press.

Brown KA., Parks KE., Bethell CA., Johnson SE., Mulligan M. 2015. Predicting plant diversity patterns in madagascar: Understanding the effects of climate and land cover change in a biodiversity hotspot. *PLOS ONE* 10:e0122721.

Butterfield BJ., Copeland SM., Munson SM., Roybal CM., Wood TE. 2016. Prestoration: Using species in restoration that will persist now and into the future. *Restor Ecol*.

Ceballos G., Ehrlich PR., Barnosky AD., Garcia A., Pringle RM., Palmer TM. 2015. Accelerated modern human-induced species losses: Entering the sixth mass extinction. *Science Advances* 1:e1400253–e1400253.

Chamberlain S., Ram K., Barve V., Mcglinn D. *rgbif: An r interface to the global 'biodiversity' information facility API*.

Chamberlain S. *pygbif: A python interface to the global biodiversity information facility API*.

Chamberlain S. *gbifrb: A ruby interface to the global biodiversity information facility API*.

Collins R., Ribeiro ED., Machado VN., Hrbek T., Farias I. 2015. A preliminary inventory of the catfishes of the lower rio nhamundá, brazil (ostariophysi, siluriformes). *BDJ* 3:e4162.

Davison J., Moora M., Opik M., Adholeya A., Ainsaar L., Ba A., Burla S., Diedhiou AG., Hiiesalu I., Jairus T., Johnson NC., Kane A., Koorem K., Kochar M., Ndiaye C., Partel M., Reier U., Saks U., Singh R., Vasar M., Zobel M. 2015. Global assessment of arbuscular mycorrhizal fungus diversity reveals very low endemism. *Science* 349:970–973.

Dellinger AS., Essl F., Hojsgaard D., Kirchheimer B., Klatt S., Dawson W., Pergl J., Pyšek P., Kleunen M van., Weber E., Winter M., Hörandl E., Dullinger S. 2015. Niche dynamics of alien species do not differ among sexual and apomictic flowering plants. *New Phytologist* 209:1313–1323.

Drozd P., Šipoš J. 2013. R for all (i): Introduction to the new age of biological analyses. *Casopis slezskeho zemskeho muzea (A)* 62.

Faulkner KT., Robertson MP., Rouget M., Wilson JR. 2014. A simple, rapid methodology for developing invasive species watch lists. *Biological Conservation* 179:25–32.

Febbraro MD., Lurz PWW., Genovesi P., Maiorano L., Girardello M., Bertolino S. 2013. The use of climatic niches in screening procedures for introduced species to evaluate risk of spread: A case with the american eastern grey squirrel. *PLoS ONE* 8:e66559.

Feitosa YO., Absy ML., Latrubesse EM., Stevaux JC. 2015. Late quaternary vegetation dynamics from central parts of the madeira river in brazil. *Acta Bot. Bras.* 29:120–128.

Ferretti F., Verd GM., Seret B., Šprem JS., Micheli F. 2015. Falling through the cracks: The fading history of a large iconic predator. *Fish and Fisheries*:n/a–n/a.

Ficetola GF., Rondinini C., Bonardi A., Baisero D., Padoa-Schioppa E. 2014. Habitat availability for amphibians and extinction threat: A global analysis. *Diversity and Distributions* 21:302–311.

Herring J. 2011. OpenGIS implementation standard for geographic information-simple feature access-part 1: Common architecture. *OGC Document* 4:122–127.

Janssens SB., Vandelook F., Langhe ED., Verstraete B., Smets E., Vandenhouwe I., Swennen R. 2016. Evolutionary dynamics and biogeography of musaceae reveal a correlation between the diversification of the banana family and the geological and climatic history of southeast asia. *New Phytologist* 210:1453–1465.

Kong X., Huang M., Duan R. 2015. SDMdata: A web-based software tool for collecting species occurrence records. *PLOS ONE* 10:e0128295.

Malhado AC., Oliveira-Neto JA., Stropp J., Strona G., Dias LC., Pinto LB., Ladle RJ. 2015. Climatological correlates of seed size in amazonian forest trees. *J Veg Sci* 26:956–963.

María Mendoza., Ospina OE., Cárdenas-Henao H., García-R JC. 2015. A likelihood inference of historical biogeography in the world's most diverse terrestrial vertebrate genus: Diversification of direct-developing frogs (craugastoridae: Pristimantis) across the neotropics. *Molecular Phylogenetics and Evolution* 85:50–58.

Pimm SL., Jenkins CN., Abell R., Brooks TM., Gittleman JL., Joppa LN., Raven PH., Roberts CM., Sexton JO. 2014. The biodiversity of species and their rates of extinction, distribution, and protection. *Science* 344:1246752–1246752.

R Core Team. 2014. *R: A language and environment for statistical computing.* Vienna, Austria: R Foundation for Statistical Computing.

Richardson DM., Roux JJL., Wilson JR. 2015. Australian acacias as invasive species: Lessons to be learnt from regions with long planting histories. *Southern Forests: a Journal of Forest Science* 77:31–39.

Robertson MP., Visser V., Hui C. 2016. Biogeo: An r package for assessing and improving data quality of occurrence record datasets. *Ecography* 39:394–401.

Turner KG., Fréville H., Rieseberg LH. 2015. Adaptive plasticity and niche expansion in an invasive thistle. *Ecol Evol* 5:3183–3197.

Verheijen LM., Aerts R., Bönisch G., Kattge J., Bodegom PMV. 2015. Variation in trait trade-offs allows differentiation among predefined plant functional types: Implications for predictive ecology. *New Phytologist* 209:563–575.

Werner GDA., Cornwell WK., Cornelissen JHC., Kiers ET. 2015. Evolutionary signals of symbiotic persistence in the legumerhizobia mutualism. *Proceedings of the National Academy of Sciences* 112:10262–10269.

Zizka A., Antonelli A. 2015. *speciesgeocodeR: An r package for linking species occurrences, user-defined regions and phylogenetic trees for biogeography, ecology and evolution.* Cold Spring Harbor Laboratory Press.