BioHackrXiv

Unveiling ecological dynamics through simulation and visualization of biodiversity data cubes

Ward Langeraert[1], Wissam Barhdadi[2], Dimitri Brosens[1], Rocìo Cortès[3], Peter Desmet[1], Michele Di Musciano[4], Chandra Earl[5], Sanne Govaert[1], Pieter Huybrechts[1], Matilde Martini[3], Arthur V. Rodrigues[6], Annegreet Veeken[7], Mukhtar Muhammed Yahaya[8], and Toon Van Daele[1]

1 Research Institute for Nature and Forest (INBO), Havenlaan 88, 1000 Brussels, Belgium **2** Department of Data Analysis and Mathematical Modelling, Ghent University, Coupure links 653, 9000 Ghent, Belgium **3** Department of Biological, Geological and Environmental Sciences, University of Bologna, Via Irnerio 42, 40126 Bologna, Italy **4** Department of Life Health and Environmental Sciences, University of L'Aquila, Via Vetoio 1, 67100 Coppito, Italy **5** Biodiversity Knowledge Integration Center, School of Life Sciences, Arizona State University, Tempe, AZ 852281, USA **6** Research Centre for Ecological Change, Organismal and Evolutionary Biology Research Programme, University of Helsinki, Viikinkaari 1, 00790 Helsinki, Finland **7** Copernicus Institute of Sustainable Development, Utrecht University, Princetonlaan 8a, 3584 CB Utrecht, The Netherlands **8** Department of Mathematical Sciences, Stellenbosch University, Stellenbosch Central, Stellenbosch, South Africa

* Corresponding author: ward.langeraert@inbo.be

Keywords:

Simulation, Data cubes, Biodiversity, B-Cubed, Monte-Carlo, R package

BioHackathon series:
Hacking Biodiversity Data Cubes for Policy
Brussels, Belgium
Projects 2+8

Submitted: 08 Oct 2024

Published by BioHackrXiv.org

Abstract

The **gcube** R package, developed during the B-Cubed hackathon (Hacking Biodiversity Data Cubes for Policy), provides a flexible framework for generating biodiversity data cubes using minimal input. The package assumes three consecutive steps (1) the occurrence process, (2) the detection process, and (3) the grid designation process, accompanied by three main functions respectively: `simulate_occurrences()`, `sample_observations()`, and `grid_designation()`. It allows for customisable spatial and temporal patterns, detection probabilities, and sampling biases. During the hackathon, collaboration was highly efficient due to thorough preparation, task division, and the use of a scrum board. Fourteen participants contributed 209 commits, resulting in a functional package with a pkgdown website, 67 % code coverage, and successful CMD checks. However, certain limitations were identified, such as the lack of spatiotemporal autocorrelation in the occurrence simulations, which affects the model's realism. Future development will focus on improving spatiotemporal dynamics, adding comprehensive documentation and testing, and expanding functionality to support multi-species simulations. The package also aims to incorporate a virtual species workflow, linking the **virtualspecies** package to the **gcube** processes. Despite these challenges, **gcube** strikes a balance between usability and complexity, offering researchers a valuable tool for simulating biodiversity data cubes to assess research questions under different parameter settings, such as the effect of spatial clustering on the occurrence-to-grid designation and the effect of different patterns of missingness on data quality and robustness of derived biodiversity indicators.

Introduction

Simulation studies offer numerous benefits due to their ability to mimic real-world scenarios in controlled and customisable environments, offering insights that are often difficult to obtain through empirical methods alone (Christie et al., 2019; Zurell et al., 2010). Ecosystems and biodiversity data are very complex and involve a multitude of interacting factors. Some of those factors belong to the occurrence (biological) process, such as responses to environmental gradients and environmental change, while other factors belong to the detection (sampling) process, such as sampling effort and bias, positional uncertainty of observations and species detectability. Simulations allow researchers to model complex ecological processes, predict outcomes in various scenarios, and test hypotheses in a controlled, repeatable manner where the effects of biological and sampling processes can be disentangled (Münkemüller et al., 2012; Sokol et al., 2017; Zurell et al., 2010).

Species occurrences can be conceptualized as events in a three-dimensional space, where the dimensions represent taxonomic classification (what), time (when), and spatial (where). These data can be organized into "occurrence cubes" or "biodiversity data cubes", providing a standardized method for integrating species occurrence information from multiple sources. Along the taxonomic axis, occurrences are classified within a specific taxon level (e.g., species level). On the temporal axis, the data are grouped by time intervals (e.g., by year). For the spatial axis, occurrences are mapped to a reference grid, with a random point selected within the uncertainty radius and placed in the grid cell that contains it (Oldoni, Groom, Adriaens, et al., 2020; Oldoni, Groom, & Desmet, 2020) (see also https://b-cubed.eu/). Biodiversity data cubes improve the accessibility and usability of open-source biodiversity data, which often originate from various sources like citizen science, monitoring projects, and remote sensing (e.g., data hosted by the Global Biodiversity Information Facility, https://www.gbif.org/). They offer a standardised approach to integrating data from multiple sources, enabling applications such as species distribution modeling and biodiversity index calculations, while ensuring the data adheres to FAIR principles (Wilkinson et al., 2016). Additionally, data cubes streamline data cleaning and computation, significantly accelerating the transformation of biodiversity data into actionable insights.

There are R packages available for simulation of ecological communities (e.g., May et al., 2018; Sokol et al., 2015) and species distribution (Leroy et al., 2016). However, we lack tools that integrate the simulation of species distribution in space and time with respect to biodiversity data cubes. In this paper, we describe the development of a practical simulation framework for these cubes during the B-Cubed Hackathon (Hacking Biodiversity Data Cubes for Policy). This framework is composed of three steps (Fig. 1):

1. The occurrence process: Simulating occurrences of multiple species distributed in a landscape over a temporal scope. This depends on the **rarity**, which can differ between species and over time, and their **spatial clustering**, which can differ between species. A challenge for this part is to implement a consistent **spatial and temporal autocorrelation** for simulated species trends.
2. The detection process: Simulation of a variety of observation processes generates actual occurrence datasets. Each species has a different **detection probability**. The detection process also depends on the **sampling effort** which can be different among spatial and temporal dimensions. Spatial uncertainty can be assigned to each observation.
3. The grid designation process: Based on their spatial uncertainty, occurrences are designated to grid cells of a larger grid to form a data cube.

This simulation framework can be used to assess multiple research questions under different parameter settings, such as the effect of spatial clustering on the occurrence-to-grid designation and the effect of different patterns of missingness on data quality and robustness of derived indicators. Simulation studies can incorporate scenarios with missing data, allowing researchers to assess the impact of data gaps on analyses within the occurrence cube framework. Furthermore, the development of a visualisation tool for the simulated cubes can enhance

the understanding of data clustering and missingness within the simulated environment. By creating a visual representation, researchers can effectively help interpret patterns of clustered data and identify areas where data are missing. This visualisation capability contributes to a more comprehensive exploration of the simulated scenarios, allowing for deeper insights into the behaviour of aggregated heterogeneous biodiversity data.

The B-Cubed Hackathon took place from 2-5 April 2024. This paper describes the methods and results of projects 2 and 8 during this hackathon. The final commit hash of the GitHub repo is given at the end of this paper. In this paper, we use the function and argument names of the stable R package version developed after the hackathon (v0.4.0, Langeraert, 2024), because some names were changed shortly after the hackathon and will improve clarity regarding argument and function usage.

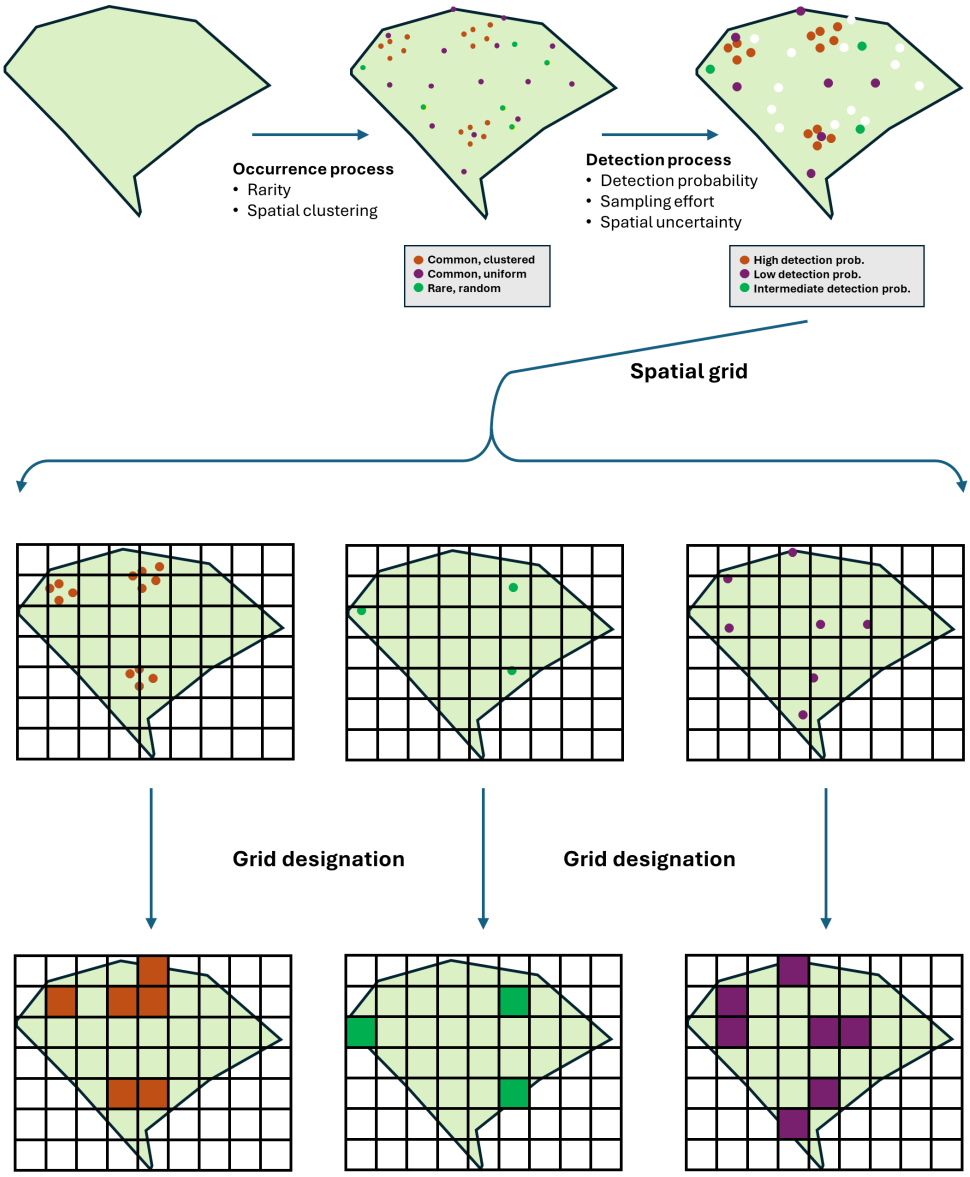


Figure 1: Simulation framework for biodiversity data cubes. An example of three species, represented by different colours, that differ in rarity, clustering, and detection probability/sampling effort. White dots are undetected occurrences.

Materials and Methods

Technical setup

The first author *a priori* decided to build the simulation framework using the R programming language as an R package (R Core Team, 2024), where participants could collaborate efficiently with each other via GitHub (https://github.com/). A repository for this package was prepared and a code structure was proposed for the framework (see next subsection).

Common guidelines for software development (e.g. related to coding style, function naming, and unit testing) were mentioned to ensure efficient collaboration as well as future maintenance and development (Huybrechts et al., 2024).

Code architecture

General code architecture of the package was proposed following preparation of the hackathon by the first author. The provided pseudocode and ideas for implementation are provided in Appendix 1. It is very similar to the developed and functional code described further in the results.

As indicated in the introduction, the simulation framework and thus the R package can be divided into three consecutive processes related to different variables that depend on *species*, *observation*, *space* and *time*.

1. occurrence process
2. detection process
3. grid designation process

For grid designation, R code was already available as the function `grid_designation()`. Thus, the focus of the hackathon was on the occurrence and detection processes.

Process	Variable	Dependency
occurrence	rarity	species, time
occurrence	spatial clustering	species
detection	detection probability	species
detection	sampling effort	space, time
detection	spatial uncertainty	observation

The three processes can be described in three main functions respectively `simulate_occurrences()`, `sample_observations()` and `grid_designation()`. Each main function consists of multiple supporting functions, for example, per variable mentioned above or for specific subprocesses (e.g. temporal autocorrelation).

Collaboration and division of tasks

Following the information provided in the previous subsections, four types of tasks were distinguished (Fig. 2).

1. **Low level tasks**: Tasks related to supporting and helper functions of the high level functions `simulate_occurrences()` and `sample_observations()`. They were the first priority of the development process during the hackathon.

Task	Description	High level relation
spatial autocorrelation	add spatial autocorrelation helper function	occurrence process function

Task	Description	High level relation
temporal autocorrelation	add temporal autocorrelation helper function	occurrence process function
detection probability	implement detection probability	detection process function
sampling bias	add sampling bias helper function	detection process function
coordinate uncertainty	add coordinate uncertainty helper function	detection process function

2. **High level tasks**: These tasks combine the low level functions into the main, high level functions `simulate_occurrences()` and `sample_observations()`. They were the final priority of the development process during the hackathon.

Task	Description
simulate occurrences	combine supporting and helper functions in `simulate_occurrences()` function
sample observations	combine supporting and helper functions in `sample_observations()` function

3. **Technical tasks**: These tasks cover general code development and testing. They were required throughout the development process of the hackathon.

Task	Description
unit tests	create unit tests and calculate code coverage
documentation	maintain function and package documentation
pkgdown website	maintain pkgdown website (Wickham et al., 2024)
GitHub repository	maintain GitHub repository

4. **Creative tasks**: Other tasks that require outside-the-box or creative thinking and which are (mainly) independent from other tasks. Participants were encouraged to come up with interesting applications and links to other frameworks/concepts/software/... These tasks could be done throughout the development process of the hackathon.

Task	Description
spatiotemporal autocorrelation	add spatiotemporal autocorrelation helper function
virtualspecies	link to **virtualspecies** package (Leroy et al., 2016)
vignettes	create vignettes
...	...

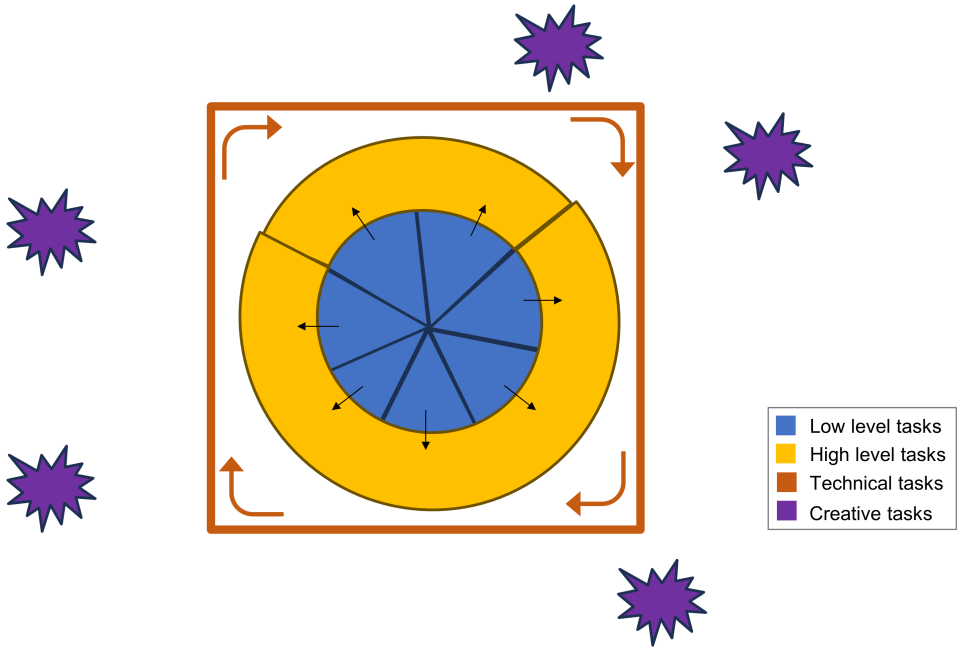


Figure 2: Schematic overview of the different types of tasks. See text for explanation.

Using a Google Form, we got an overview of participants' interest in the different tasks, and an idea where a potential shortage of coverage might occur. Tasks were divided and followed up via a simple scrum methodology by using sticky notes (coloured by task type) on a board. The board was divided into four parts (from left to right): 'Ice Box', 'In Progress', 'Review', and 'Complete'. The 'Ice Box' is where all the potential tasks and ideas were stored before they were prioritized and selected for development. The 'In Progress' category contains tasks that the team was actively working on during development. The 'Review' category is for tasks that have been completed but are awaiting review, testing, or approval. The 'Complete' category includes tasks that have been reviewed, approved, and finalised. The participants were free to choose and add tasks to the scrum board, indicating the task and their name on the sticky note.

Results

Collaboration

This was the general timing of activities during the hackathon. In total, we had about 6 half-days of coding time.

Day	Part	Activities
1	Morning	General introduction and presentations of the hackathon.
	Afternoon	Create project group. Acquaintance with participants and presentation of the objectives of the project.
2	Morning	Division of tasks and start of code development.
	Afternoon	Continued code development.

Day	Part	Activities
3	Morning	Further code development.
	Afternoon	Continued code development.
4	Morning	Code review and pull request merging.
	Afternoon	Final presentations of all hackathon projects.

After discussion with the participants, the R package was named **gcube**, which stands for 'generate cube' since it can be used to generate biodiversity data cubes from minimal input.

Tasks were efficiently distributed along the participants (Fig. 3). In total, we collaborated with fourteen people pushing 209 commits to the main branch and 300 commits to all branches. On main, 56 files were changed and there have been 2,856 additions and 373 deletions. By the end of the hackathon, we had a functional pkgdown website (Fig. 4), all CMD checks passed, and we had a code coverage of 67 %.

Figure 3: Scrum board progress during code development. Categories from left to right: 'Ice Box', 'In Progress', 'Review', and 'Complete'. Day 1 was mainly introduction and discussion. Day 2-3 mainly code development. Day 4 was primarily review and pull request merging. Coding ended before the final presentations on day 4 in the afternoon.

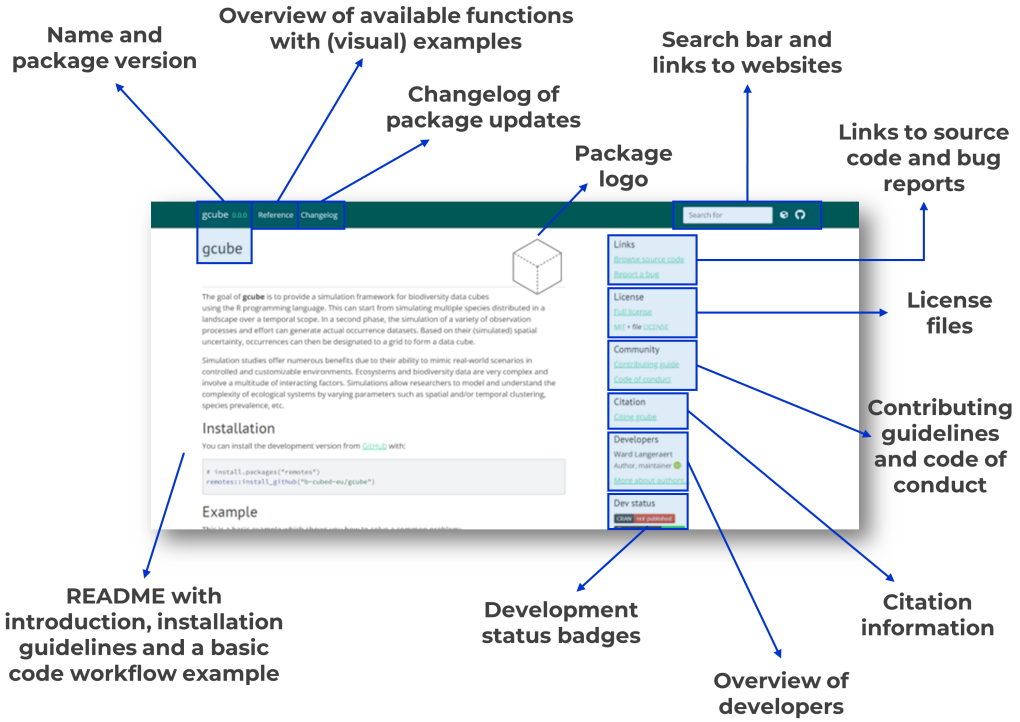


Figure 4: Overview of the **gcube** pkgdown website.

R package development

The biodiversity data cube simulation workflow of **gcube** is divided into three steps or processes:

1. Occurrence process
2. Detection process
3. Grid designation process

The three processes are executed by three main functions `simulate_occurrences()`, `sample_observations()`, and `grid_designation()`, respectively. The functions are designed such that a single polygon as input is enough to go through this workflow using default arguments. An example workflow is given in the next section. In this subsection, we give a more technical overview of the functions that were developed. The three functions all have a `seed` argument used to allow reproducible results. If `NA` (the default), no seed is used.

1. Occurrence process

The `simulate_occurrences()` function generates occurrences of a species within a given spatial and/or temporal span.

```
simulate_occurrences(
  species_range,
  initial_average_occurrences = 50,
  spatial_pattern = c("random", "clustered"),
  n_time_points = 1,
  temporal_function = NA,
  ...,
  seed = NA
)
```

The input (`species_range`) should be an sf object with POLYGON geometry indicating the spatial extension to simulate occurrences.

The temporal component of this function is executed by the `simulate_timeseries()` supporting function.

```
simulate_timeseries(
  initial_average_occurrences = 50,
  n_time_points = 1,
  temporal_function = NA,
  ...,
  seed = NA
)
```

The initial number of occurrences (`initial_average_occurrences`) and the temporal trend function (`temporal_function`) generate a number of occurrences for each time point (the total number of time points is given by `n_time_points`). If the temporal function is `NA` (default), it samples `n_time_points` times from a Poisson distribution with an average equal to `initial_average_occurrences`. You can also specify a function which generates a trend in number of occurrences over time. This can be the internal function `simulate_random_walk()` or a custom function that takes `initial_average_occurrences` and `n_time_points` as arguments. Additional arguments for the temporal function can be passed to the ellipsis argument (`...`). The specified temporal function calculates a number of occurrences for each time point according to a certain function (e.g. a random walk in case of `temporal_function = simulate_random_walk`) and draws this from a Poisson distribution using `stats::rpois()` (R Core Team, 2024).

The spatial component of `simulate_occurrences()` is executed by the `create_spatial_pattern()` and `sample_occurrences_from_raster()` supporting functions.

```
create_spatial_pattern(
  polygon,
  resolution,
  spatial_pattern = c("random", "clustered"),
  seed = NA,
  n_sim = 1
)
```

`create_spatial_pattern()` creates a raster for the area of a polygon (`polygon`) with a resolution (`resolution`) according to a spatial patter `spatial_pattern`. `"random"` is the default pattern. The user is able to provide a numeric value $>= 1$ (1 is "random" and 10 is "clustered"). A larger number means a broader size of the clusters. This number changes the range parameter of the spherical variogram model. `spatial_pattern = 1` means the range has the same size of the grid cell, which is defined in resolution argument (calculated by `simulate_occurrences()` as one hundredth of the extend of the polygon). We use the function `gstat::vgm()` to implement the spherical variogram model (Gräler et al., 2016).

```
sample_occurrences_from_raster(
  raster,
  time_series,
  seed = NA
)
```

The raster output of `create_spatial_pattern()` is then used as input for `sample_occurrences_from_raster()` (argument `raster`). From this raster, it samples a number of occurrences (argument `time_series`) as provided by `simulate_timeseries()` for each time point using `terra::spatSample()` (Hijmans, 2024). The final result is thus a number of occurrences sampled from a spatial pattern for multiple time points that are passed as output for `simulate_occurrences()`.

2. Detection process

We have our occurrences, but not all occurrences are generally observed. The detection of occurrences depends on the detection probability of a species and the sampling bias (which includes both sampling bias and effort). This process is simulated using the `sample_observations()` function.

```
sample_observations(
  occurrences,
  detection_probability = 1,
  sampling_bias = c("no_bias", "polygon", "manual"),
  bias_area = NA,
  bias_strength = 1,
  bias_weights = NA,
  seed = NA
)
```

Detection probability (`detection_probability`) is passed as a numeric value between 0 and 1. For sampling bias, there are three options specified in `sampling_bias` (cf. Leroy et al. (2016)).

1. With `"no_bias"`, only the detection probability value decides whether an occurrence is observed or not. If `detection_probability = 1` and `sampling_bias = "no_bias"`, all occurrences are detected.
2. With `"polygon"`, bias weights depend on their location inside or outside a given polygon with a certain bias strength. This is achieved by the supporting function `apply_polygon_sampling_bias()`.

```
apply_polygon_sampling_bias(
  occurrences_sf,
  bias_area,
  bias_strength = 1
)
```

The function adds a sampling bias weight column to an sf object with POINT geometry containing the occurrences (`occurrences_sf`). This column contains the sample probability based on bias strength `bias_strength` within a given polygon `bias_area`. The bias strength is the strength of the bias to be applied in the biased area (as a multiplier). Above 1, the area will be oversampled. Below 1, the area will be undersampled. For example, a value of 50 will result in 50 times more samples within the `bias_area` than outside. Conversely, a value of 0.5 will result in half less samples.

3. With `"manual"`, bias weights depend on their location inside grid cells of a given grid where each cell has its own value. This is achieved by the supporting function `apply_manual_sampling_bias()`.

```
apply_manual_sampling_bias(
  occurrences_sf,
  bias_weight
)
```

The function adds a sampling bias weight column to an sf object with POINT geometry containing the occurrences (`occurrences_sf`). This column contains the sample probability based on bias weights within each cell of a given grid layer (`bias_weight`).

`sample_observations()` combines detection probability and sampling bias weight to a single value `p` as a product and uses this to draw for each occurrence from `stats::rbinom(1, 1, p)` (R Core Team, 2024) to decide whether an occurrence is observed or not.

To mimic real-life data collection, we can select observed occurrences and add coordinate uncertainty with the `add_coordinate_uncertainty()` function. This is optional.

```
add_coordinate_uncertainty(
  observations,
  coords_uncertainty_meters = 25
)
```

This is done by adding an additional column to the observed occurrences (`observations`). This column contains numeric values (passed to `coords_uncertainty_meters` as one value or a vector of values) that indicate the coordinate uncertainty in metres around each observation.

3. Grid designation process

Now that we have our observations, we designate them to a grid while taking into account the coordinate uncertainty in metres around the observation, if present. This function was already developed by the first author before the hackathon started, but is given here for the sake of completeness.

```
grid_designation(
  observations,
  grid,
  id_col = "row_names",
  seed = NA,
  aggregate = TRUE,
  randomisation = c("uniform", "normal"),
  p_norm = ifelse(tolower(randomisation[1]) == "uniform", NA, 0.95)
)
```

This function designates observations (`observations`) to cells of a given grid (`grid`) to create a data cube. `id_col` specifies the column name of the column with unique ids for each grid cell. If `id_col = "row_names"` (the default), a new column `cell_code` is created where the row names represent the unique ids. If `aggregate = TRUE` (default), the data cube is returned in aggregated form (grid with number of observations per grid cell). Otherwise, return the sampled points in their uncertainty circle. The randomisation method, specified with `randomisation`, is used for sampling within uncertainty circle around each observation. By default `"uniform"` which means each point uncertainty circle has an equal probability to be selected. If no coordinate uncertainty is present, the function takes the point itself for designation. The other option is `"normal"` where a point is sampled from a bivariate Normal distribution with means equal to the observation point and the variance equal to (-`coordinateUncertaintyInMeters`^2) / (2 * log(1 - `p_norm`)) such that `p_norm` % of all possible samples from this Normal distribution fall within the uncertainty circle. `p_norm` is only used if `randomisation = "normal"` and has the default value of 0.95. Uniform is the standard method to create biodiversity data cubes. The normal randomisation is an experimental feature.

The final output is (`aggregate = TRUE`) an sf object with POLYGON geometry containing the grid cells, an `n` column with the number of observations per grid cell, and a `min_coord_uncertainty` column with the minimum coordinate uncertainty per grid cell. If `aggregate = FALSE`, an sf object with POINT geometry containing the sampled observations within the uncertainty circles, and a column `coordinateUncertaintyInMeters` with the coordinate uncertainty for each observation.

The following imports and suggests were used. Packages listed under 'imports' are essential for the package to function and are automatically loaded when **gcube** is loaded. Packages listed under 'suggests' are not essential for the basic functionality of the package, but are useful for certain optional features, examples, or tests. These packages are not automatically loaded when **gcube** is loaded.

Type	Package	Source
imports	**cli**	(Csárdi, 2024)
imports	**dplyr**	(Wickham et al., 2023)
imports	**gstat**	(Gräler et al., 2016)
imports	**magrittr**	(Bache & Wickham, 2022)
imports	**methods**	(R Core Team, 2024)
imports	**mnormt**	(Azzalini & Genz, 2022)
imports	**rlang**	(Henry & Wickham, 2024)
imports	**sf**	(Pebesma, 2018; Pebesma & Bivand, 2023)
imports	**stats**	(R Core Team, 2024)
imports	**terra**	(Hijmans, 2024)
imports	**vegan**	(Oksanen et al., 2024)
imports	**withr**	(Hester et al., 2024)
suggests	**ggplot2**	(Wickham, 2016)
suggests	**testthat**	(Wickham, 2011)

Incorporation of virtual species to the simulation workflow

Project 8 originally aimed to address the challenges of incomplete and unreliable biodiversity data that hinder accurate species distribution models (SDMs). By creating virtual species with known ecological characteristics, researchers can simulate and analyse the effects of spatial, temporal, and taxonomic uncertainties. This "virtual ecologist" approach helps quantify sources of error and refine modelling techniques (Zurell et al., 2010). The goal is to improve conservation planning, especially for rare or endangered species, by providing more reliable predictions of species distributions under various environmental conditions, including climate change.

At an early stage of the hackathon, it was decided that the concepts and ideas developed by this group would be integrated into the cube simulation package of group 2. This decision was influenced by the existing proposal to incorporate a virtual species workflow using the **virtualspecies** package, as mentioned above. The focus was primarily on discussions, conceptualization, and experimentation with the code of both the **virtualspecies** package and the **gcube** package as it was being developed at the time.

The idea of working with a virtual species approach in **gcube** is that simulations can start from two points.

1. Original **gcube** workflow: Start from empty polygon and mathematical concepts.
2. **virtualspecies** workflow: Start from environmental data layers.

We identified the needs for future development of the virtual species approach. Link functions are required that accept output from the **virtualspecies** package and provide input for the three main simulation functions of **gcube**.

virtualspecies function(s)	**virtualspecies** output	Link function **gcube**	**gcube** function(s)
`generateSpFromFun()` or `generateSpFromPCA()`	environmental suitability map	`rescale_suitability_raster()`	`simulate_occurrences()`
`convertToPA()`	presence-absence map	`occurrences_from_raster()`	`sample_observations()`

virtualspecies function(s)	**virtualspecies** output	Link function **gcube**	**gcube** function(s)
`sampleOccurrences()`	sampled presence points	`virtual_occurrences_to_sf()`	`add_coordinate_uncertainty()` and/or `grid_designation()`

This was mainly conceptual and was not implemented in the package yet. The link functions are suggestions, which can differ from actual future implementation.

gcube workflow example

This is a basic example from the README which shows the workflow for simulating a biodiversity data cube using the **gcube** package. It is an example for one time point for a single species (the default). This is not the exact README example from the hackathon, but a cleaned version from the week after.

The functions are designed such that a single polygon as input is enough to go through this workflow using default arguments. The user can change these arguments to allow for more flexibility.

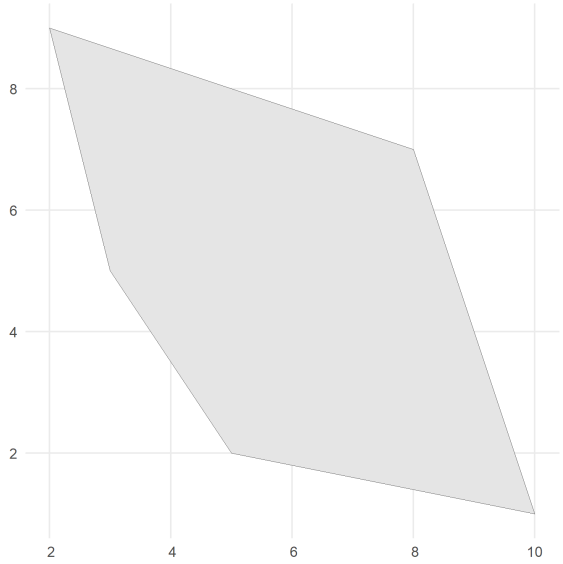
```
# Load packages
library(gcube)

library(sf)       # working with spatial objects
library(dplyr)    # data wrangling
library(ggplot2) # visualisation with ggplot
```

We create an arbitrary polygon as input.

```
# Create a polygon to simulate occurrences
polygon <- st_polygon(list(cbind(c(5, 10, 8, 2, 3, 5), c(2, 1, 7,9, 5, 2))))

# Visualise
ggplot() +
  geom_sf(data = polygon) +
  theme_minimal()
```

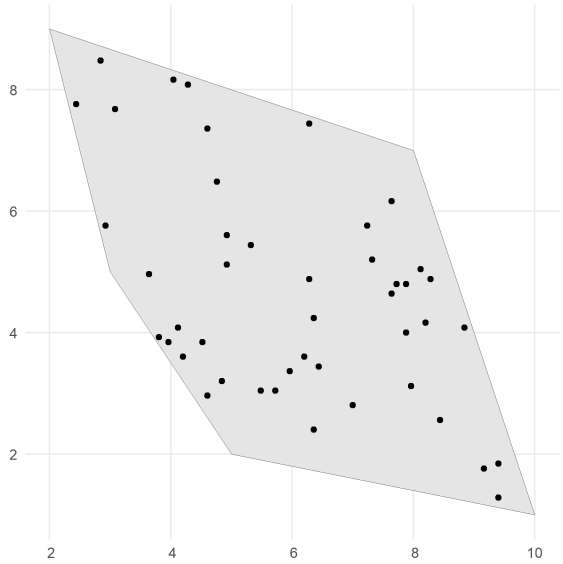


1. Occurrence process

We generate occurrence points within the polygon using the `simulate_occurrences()` function. These are the "real" occurrences of the species, whether we have observed them or not. In the `simulate_occurrences()` function, the user can specify different levels of spatial clustering, and can define the trend change of the species over time.

```
# Simulate occurrences within polygon
occurrences_df <- simulate_occurrences(
  species_range = polygon,
  seed = 123)
#> [using unconditional Gaussian simulation]

# Visualise
ggplot() +
  geom_sf(data = polygon) +
  geom_sf(data = occurrences_df) +
  theme_minimal()
```

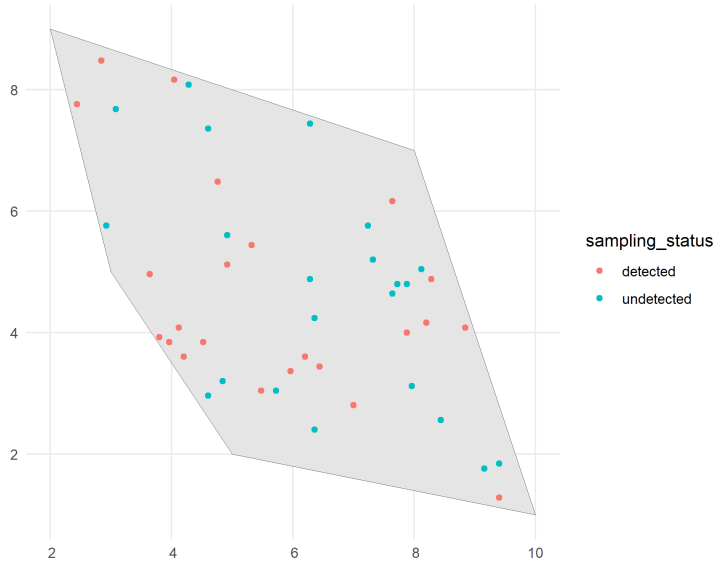


2. Detection process

In this step we define the sampling process, based on the detection probability of the species and the sampling bias. This is done using the `sample_observations()` function. The default sampling bias is `"no_bias"`, but bias can also be inserted using a polygon or a grid.

```
# Detect occurrences
detections_df_raw <- sample_observations(
  occurrences = occurrences_df,
  detection_probability = 0.5,
  seed = 123)

# Visualise
ggplot() +
  geom_sf(data = polygon) +
  geom_sf(data = detections_df_raw,
          aes(colour = sampling_status)) +
  theme_minimal()
```



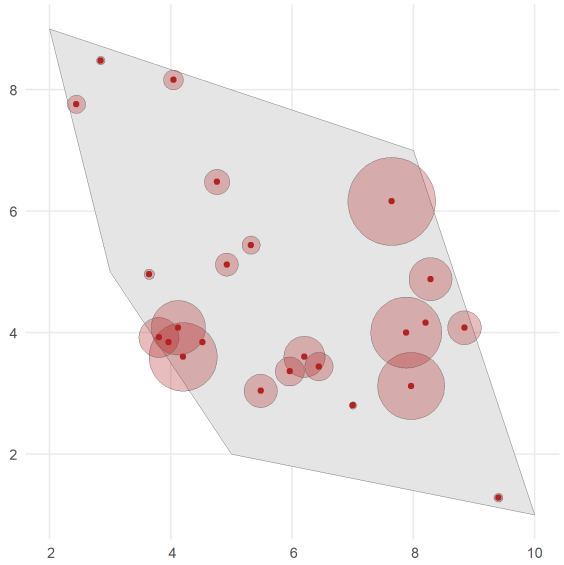
We select the detected occurrences and add an uncertainty to these observations, by using the `add_coordinate_uncertainty()` function.

```
# Select detected occurrences only
detections_df <- detections_df_raw %>%
  dplyr::filter(sampling_status == "detected")

# Add coordinate uncertainty
set.seed(123)
coord_uncertainty_vec <- rgamma(nrow(detections_df), shape = 2, rate = 6)
observations_df <- add_coordinate_uncertainty(
  observations = detections_df,
  coords_uncertainty_meters = coord_uncertainty_vec)

# Created and sf object with uncertainty circles to visualise
buffered_observations <- st_buffer(
  observations_df,
  observations_df$coordinateUncertaintyInMeters)
```

```
# Visualise
ggplot() +
  geom_sf(data = polygon) +
  geom_sf(data = buffered_observations,
          fill = alpha("firebrick", 0.3)) +
  geom_sf(data = observations_df, colour = "firebrick") +
  theme_minimal()
```



3. Grid designation process

Finally, observations are designated to a grid to create an occurrence cube. We create a grid over the spatial extend using `sf::st_make_grid()`.

```
# Define a grid over spatial extend
grid_df <- st_make_grid(
    buffered_observations,
    square = TRUE,
    cellsize = c(1.2, 1.2)
  ) %>%
  st_sf() %>%
  mutate(intersect = as.vector(st_intersects(geometry, polygon,
                                             sparse = FALSE))) %>%
  dplyr::filter(intersect == TRUE) %>%
  dplyr::select(-"intersect")
```

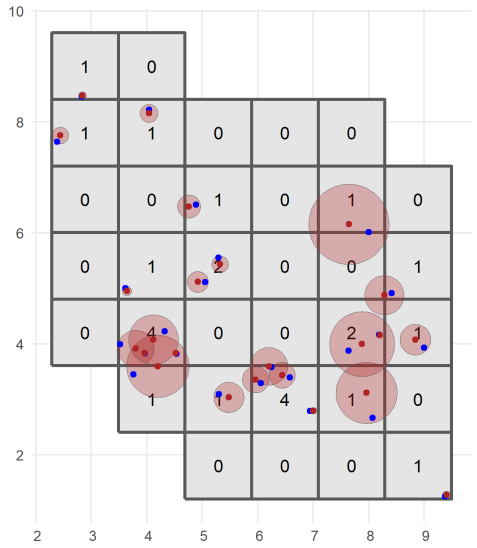
To create an occurrence cube, `grid_designation()` will randomly take a point within the uncertainty circle around the observations. These points can be extracted by setting the argument `aggregate = FALSE`.

```
# Create occurrence cube
occurrence_cube_df <- grid_designation(
  observations = observations_df,
  grid = grid_df,
  seed = 123)

# Get sampled points within uncertainty circle
sampled_points <- grid_designation(
  observations = observations_df,
  grid = grid_df,
```

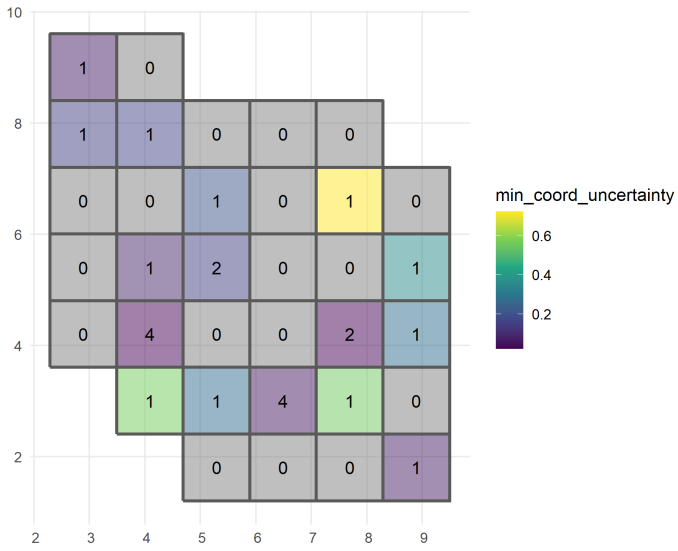
```
  aggregate = FALSE,
  seed = 123)

# Visualise grid designation
ggplot() +
  geom_sf(data = occurrence_cube_df, linewidth = 1) +
  geom_sf_text(data = occurrence_cube_df, aes(label = n)) +
  geom_sf(data = buffered_observations,
          fill = alpha("firebrick", 0.3)) +
  geom_sf(data = sampled_points, colour = "blue") +
  geom_sf(data = observations_df, colour = "firebrick") +
  labs(x = "", y = "", fill = "n") +
  theme_minimal()
```



The output gives the number of observations per grid cell and minimal coordinate uncertainty per grid cell.

```
# Visualise minimal coordinate uncertainty
ggplot() +
  geom_sf(data = occurrence_cube_df, aes(fill = min_coord_uncertainty),
          alpha = 0.5, linewidth = 1) +
  geom_sf_text(data = occurrence_cube_df, aes(label = n)) +
  scale_fill_continuous(type = "viridis") +
  labs(x = "", y = "") +
  theme_minimal()
```



Discussion and future work

Collaboration

The methods outlined in this paper proved to be very efficient for hackathon code collaboration. A thorough preparation turned out to be crucial for working together on a single project in a large group. We recommend to set up code repository structure and provide pseudocode (architecture) beforehand if possible. In this way, all participants can focus on the content of the project from the beginning. Defining modular tasks in advance and preparing an interactive follow-up schedule (the scrum board) also helped to ensure participant engagement and a quick and smooth start of code development.

Current shortcomings

It is impossible to list all potential shortcomings for a project developed within such a short time frame, but several conceptual issues emerged during discussions. A simulation framework is only valuable if it can realistically model biological and sampling processes. This is particularly true for the occurrence process, handled by the `simulate_occurrences()` function. For example, this function generates a new sample for each time point from the spatial pattern that remains unchanged over time, lacking spatiotemporal autocorrelation.

gcube is designed to be an accessible and easy-to-use package for exploring cube-related research questions within a controlled and customisable environment. However, there is an inherent trade-off between increasing the complexity of the package and maintaining its usability. Balancing these factors is crucial to ensure that the package remains practical for researchers while still offering enough flexibility for robust and realistic simulations. Future improvements should aim to introduce more sophisticated spatiotemporal dynamics without compromising ease of use.

We list some potentially useful references here without going too much into detail:

- Instead of sampling from a Poisson distribution, we can model the number of occurrences in space and time. For example, using point process models (Bachl et al., 2019) or spatiotemporal GLMMs (Generalized Linear Mixed Effects Models) (Anderson et al., 2024).
- Implement individual based models (on (meta)population level?), which allows for a high degree of complexity of individuals and of interactions among individuals (see e.g.,

Grimm & Railsback, 2013).
- Simulate random spatial point patterns (Baddeley et al., 2015), e.g. using a homogeneous or inhomogeneous Poisson process. This might be an alternative to the current method rather than a solution for spatiotemporal autocorrelation.

Future development

After working with many people on a single project, an obvious first step is the need for code cleanup and unifying coding style and documentation. Due to time constraints, some documentation needs to be added or corrected, and unit tests need to be added where necessary for higher code coverage. It would also be good to add vignettes that demonstrate the usage of the different functions' arguments throughout the cube simulation workflow.

Several (smaller) issues were posted on GitHub during the hackathon. These (and newly arising) issues will be monitored and fixed. Also, for future development, we will evidently aim to address the shortcomings listed in the subsection above. Other, major enhancements would be to (1) provide an efficient way, i.e. a set of functions, for creating cubes for multiple species at once, and (2) implement the virtual species approach as outlined earlier.

Finally, an interesting package that was not noticed during the preparation or the hackathon itself is the **mobsim** package (May et al., 2018). This package simulates the abundance and distribution of species in a spatial landscape, allowing users to set properties such as total individuals, species-abundance distribution, and spatial aggregation. It also provides functions to calculate biodiversity metrics, such as rarefaction curves and species–area relationships, and to simulate different sampling designs. It might be interesting to compare with and to look into the methods used by this package.

Links to software

The **gcube** code repository can be found here: https://github.com/b-cubed-eu/gcube. The pkgdown website here: https://b-cubed-eu.github.io/gcube. The final commit hash of the GitHub repo at the end of the hackathon:

https://github.com/b-cubed-eu/gcube/commit/6cceb2b229ac25d1df47a9c3a2e20b464f827e18

The current package version, at the time of publishing this paper, is 0.4.0 (Langeraert, 2024). It contains several vignettes on the pkgdown website that explain the simulation workflow in detail. Also, functions are added that provide calculation of data cubes for multiple species at once, unit tests are added, and documentation is completed.

The B-Cubed hackathon repository can be found here: https://github.com/b-cubed-eu/hackathon-projects-2024. It contains R Markdown scripts in preparation of the event.

Acknowledgements

We would like to express our gratitude to the Horizon Europe funded B-Cubed (Biodiversity Building Blocks for policy) project for the organisation of this hackathon. Special thanks go to Laura Abraham of Meise Botanic Garden for her outstanding efforts in coordinating the event. Additionally, we acknowledge the Research Foundation - Flanders (FWO) and KU Leuven for the support they gave us in making this event possible.

References

Anderson, S. C., Ward, E. J., English, P. A., Barnett, L. A. K., & Thorson, J. T. (2024). sdmTMB: An r package for fast, flexible, and user-friendly generalized linear mixed effects

models with spatial and spatiotemporal random fields. *bioRxiv*, *2022.03.24.485545*. https://doi.org/10.1101/2022.03.24.485545

Azzalini, A., & Genz, A. (2022). *The R package `mnormt`: The multivariate normal and t distributions (version 2.1.1)*. http://azzalini.stat.unipd.it/SW/Pkg-mnormt/

Bache, S. M., & Wickham, H. (2022). *magrittr: A Forward-Pipe Operator for R*. https://CRAN.R-project.org/package=magrittr

Bachl, F. E., Lindgren, F., Borchers, D. L., & Illian, J. B. (2019). inlabru: An R package for Bayesian spatial modelling from ecological survey data. *Methods in Ecology and Evolution*, *10*, 760–766. https://doi.org/10.1111/2041-210X.13168

Baddeley, A., Rubak, E., & Turner, R. (2015). *Spatial point patterns: Methodology and applications with R*. Chapman; Hall/CRC Press. https://doi.org/10.1201/b19708

Christie, A. P., Amano, T., Martin, P. A., Shackelford, G. E., Simmons, B. I., & Sutherland, W. J. (2019). Simple study designs in ecology produce inaccurate estimates of biodiversity responses. *Journal of Applied Ecology*. https://doi.org/10.1111/1365-2664.13499

Csárdi, G. (2024). *cli: Helpers for Developing Command Line Interfaces*. https://CRAN.R-project.org/package=cli

Gräler, B., Pebesma, E., & Heuvelink, G. (2016). Spatio-Temporal Interpolation using gstat. *The R Journal*, *8*, 204–218. https://journal.r-project.org/archive/2016/RJ-2016-014/index.html

Grimm, V., & Railsback, S. F. (2013). Individual-based modeling and ecology. In *Individual-based modeling and ecology*. Princeton university press. https://doi.org/10.1515/9781400850624

Henry, L., & Wickham, H. (2024). *rlang: Functions for Base Types and Core R and 'Tidyverse' Features*. https://CRAN.R-project.org/package=rlang

Hester, J., Henry, L., Müller, K., Ushey, K., Wickham, H., & Chang, W. (2024). *withr: Run Code 'With' Temporarily Modified Global State*. https://CRAN.R-project.org/package=withr

Hijmans, R. J. (2024). *terra: Spatial Data Analysis*. https://CRAN.R-project.org/package=terra

Huybrechts, P., Trekels, M., Abraham, L., & Desmet, P. (2024). *B-Cubed software development guide*. https://docs.b-cubed.eu/dev-guide/.

Langeraert, W. (2024). *gcube: Simulating Biodiversity Data Cubes* (Version 0.4.0) [Computer software]. https://github.com/b-cubed-eu/gcube/

Leroy, B., Meynard, C. N., Bellard, C., & Courchamp, F. (2016). virtualspecies, an R package to generate virtual species distributions. *Ecography*, *39*(6), 599–607. https://doi.org/10.1111/ecog.01388

May, F., Gerstner, K., McGlinn, D. J., Xiao, X., & Chase, J. M. (2018). mobsim: An R package for the simulation and measurement of biodiversity across spatial scales. *Methods in Ecology and Evolution*, *9*(6), 1401–1408. https://doi.org/10.1111/2041-210X.12986

Münkemüller, T., Bello, F., Meynard, C., Gravel, D., Lavergne, S., Mouillot, D., Mouquet, N., & Thuiller, W. (2012). From diversity indices to community assembly processes: A test with simulated data. *Ecography*, *35*, 468–480. https://doi.org/10.1111/J.1600-0587.2011.07259.X

Oksanen, J., Simpson, G. L., Blanchet, F. G., Kindt, R., Legendre, P., Minchin, P. R., O'Hara, R. B., Solymos, P., Stevens, M. H. H., Szoecs, E., Wagner, H., Barbour, M., Bedward, M., Bolker, B., Borcard, D., Carvalho, G., Chirico, M., De Caceres, M., Durand, S., ... Weedon, J. (2024). *vegan: Community Ecology Package*. https://CRAN.R-project.org/package=vegan

Oldoni, D., Groom, Q., Adriaens, T., Davis, A. J. S., Reyserhove, L., Strubbe, D., Vanderhoeven, S., & Desmet, P. (2020). Occurrence cubes: A new paradigm for aggregating species occurrence data. *bioRxiv*. https://doi.org/10.1101/2020.03.23.983601

Oldoni, D., Groom, Q., & Desmet, P. (2020). Occurrence Cubes: A new way of aggregating heterogeneous species occurrence data. *Biodiversity Information Science and Standards*, *4*, e59154. https://doi.org/10.3897/biss.4.59154

Pebesma, E. (2018). Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, *10*(1), 439–446. https://doi.org/10.32614/RJ-2018-009

Pebesma, E., & Bivand, R. (2023). *Spatial Data Science: With applications in R*. Chapman and Hall/CRC. https://doi.org/10.1201/9780429459016

R Core Team. (2024). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Sokol, E. R., Brown, B. L., & Barrett, J. (2017). A simulation-based approach to understand how metacommunity characteristics influence emergent biodiversity patterns. *Oikos*, *126*, 723–737. https://doi.org/10.1111/OIK.03690

Sokol, E. R., Brown, B. L., Carey, C. C., Tornwall, B. M., Swan, C. M., & Barrett, J. (2015). Linking management to biodiversity in built ponds using metacommunity simulations. *Ecological Modelling*, *296*, 36–45. https://doi.org/10.1016/j.ecolmodel.2014.10.022

Wickham, H. (2011). Testthat: Get started with testing. *The R Journal*, *3*, 5–10. https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. https://doi.org/10.1007/978-3-319-24277-4

Wickham, H., François, R., Henry, L., Müller, K., & Vaughan, D. (2023). *dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr

Wickham, H., Hesselberth, J., Salmon, M., Roy, O., & Brüggemann, S. (2024). *pkgdown: Make Static HTML Documentation for a Package*. https://CRAN.R-project.org/package=pkgdown

Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., … Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, *3*(1), 1–9. https://doi.org/10.1038/sdata.2016.18

Zurell, D., Berger, U., Cabral, J. S., Jeltsch, F., Meynard, C. N., Münkemüller, T., Nehrbass, N., Pagel, J., Reineking, B., Schröder, B., & Grimm, V. (2010). The virtual ecologist approach: Simulating data and observers. *Oikos*, *119*(4), 622–635. https://doi.org/10.1111/j.1600-0706.2009.18284.x

Appendices

Appendix 1: Pseudocode provided in preparation of the hackathon

Some pseudocode and ideas for implementation were provided by the first author for `simulate_occurrences()` and `sample_observations()` on the first day:

1. Occurrence process

```
simulate_occurrences(
  polygon,
  initial_average_abundance = 50,
  spatial_autocorr = c("random", "clustered", "regular"),
```

```
  n_time_points = 10,
  temporal_autocorr = ifelse(time_points ==  1, NA, "random_walk"),
  spatiotemporal_autocorr = NA,
  seed = NA
)
```

- **polygon**:

An sf object with POLYGON geometry indicating the spatial extend to simulate occurrences.

- **initial_average_abundance**:

A positive integer value indicating the average number of occurrences to be simulated within the extend of `polygon` at time point 1. This value is used as mean of a Poisson distribution (λ parameter).

- **spatial_autocorr**:

`"random"`, `"clustered"`, `"regular"` or a numeric value between -1 and 1 representing Moran's I, indicating spatial autocorrelation. `"random"` corresponds to 0, `"clustered"` to 0.9 and `"regular"` to -0.9.

- **n_time_points**:

A positive integer value indicating the number of time points to simulate.

- **temporal_autocorr**:

`NA`, `"random_walk"` or a function which generates a trend in abundance over time, indicating temporal autocorrelation. Only used if `time_points > 1`. When there are multiple time points and `"random_walk"` is selected, an internal function can be used to create a random walk over time. The user is also free to specify its own function that depends on `initial_average_abundance` and `n_time_points`, e.g. a linearly decreasing trend over time.

- **spatiotemporal_autocorr**:

A numeric value between indicating the strength of spatiotemporal autocorrelation.

- **seed**:

A positive numeric value. The seed for random number generation to make results reproducible. If `NA` (the default), no seed is used.

2. Detection process

```
sample_observations(
  occurrences,
  detection_probability = 1,
  sampling_bias = c("no_bias", "polygon", "manual"),
  bias_area = NA,
  bias_strength = NA,
  bias_weights = NA,
  coordinate_uncertainty_meters = 25,
  seed = NA
)
```

- **occurrences**:

An sf object with POINT geometry.

- **detection_probability**:

A numeric value between 0 and 1, corresponding to the probability of detection of the species.

- **sampling_bias**:

"no_bias", "polygon" or "manual". The method used to generate a sampling bias (cf. the **virtualspecies** package by Leroy et al. (2016)). "polygon": bias the sampling in a polygon. Provide your polygon to bias_area. Provide bias strength to bias_strength. "manual": bias the sampling manually via a raster. Provide your raster layer in which each cell contains the probability to be sampled to bias_weights.

- **bias_area**:

NA or an sf object with POLYGON geometry. Only used if sampling_bias = "polygon". The area in which the sampling will be biased.

- **bias_strength**:

NA or a positive numeric value. Only used if sampling_bias = "polygon". The strength of the bias to be applied in the biased area (as a multiplier). Above 1, area will be oversampled. Below 1, area will be undersampled. For example, a value of 50 results in 50 times more samples within the bias_area than outside of it. Conversely, a value of 0.5 results in half less samples within the bias_area than outside of it.

- **bias_weights**:

NA or a raster layer (sf object with POLYGON geometry, or SpatRaster object). Only used if sampling_bias = "manual". The raster of bias weights to be applied to the sampling of occurrences. Higher weights mean a higher probability of sampling. Weights can be numeric values between 0 and 1 or positive integers that will be rescaled to values between 0 and 1.

- **coordinate_uncertainty_meters**:

A positive numeric value or vector with length nrow(occurrences) describing the uncertainty in meters around each observation.

- **seed**:

A positive numeric value. The seed for random number generation to make results reproducible. If NA (the default), no seed is used.

Finally, we also have the function add_coordinate_uncertainty().

```
add_coordinate_uncertainty(
  occurrences,
  coordinate_uncertainty_meters = 25
)
```

This function is a supporting function for sample_observations() to add a coordinateUncertaintyInMeters column that should also be exported in case users simulate observations based on virtual species distributions. This can for example be accomplished using the **virtualspecies** package (Leroy et al., 2016):

1. Species-environment relationship
   - generateSpFromFun()
   - generateSpFromPCA()
2. Conversion to presence-absence
   - convertToPA()
   - introduce distribution bias: limitDispersal()
3. Sample occurrences
   - sampleOccurrences()
4. Convert to sf object with POINT geometry
   - sf::st_as_sf()
   - create helper function virtualspecies_to_sf()?