Microxanox - an R package for simulating an *MICR*obial ecosystem that can occupy *OX*ic or *ANOX*ic states.

Rainer M Krug[a,], Owen L. Petchey[a]

[a] *Department of Evolutionary Biology and Environmental Studies, Winterthurerstrasse 190, 8057 Zurich*

Abstract

Ca. 100 words.

Keywords: metadata quality; data curation; archival; long term storage; R package;

---

1. Required Metadata

1.1. *Current code version*

Ancillary data table required for subversion of the codebase.

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v0.9.0
C2	Permanent link to code/repository used for this code version	https://github.com/UZH-PEG/microxanox
C3	Code Ocean compute capsule	
C4	Legal Code License	CC BY 4.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	R
C7	Compilation requirements, operating environments	R (>= 4.1.0), ADD PACKAGES
C8	If available Link to developer documentation/manual	
C9	Support email for questions	Rainer.Krug@uzh.ch; Rainer@krugs.de

*Corresponding Author
**Equal contribution
Email addresses: `Rainer.Krug@uzh.ch \& Rainer@krugs.de` (Rainer M Krug), `Owen.Petchey@ieu.uzh.ch` (Owen L. Petchey)

2. Motivation and significance

Many ecosystems are exposed to gradual changes of environmental variables, to which the responses are not always as gradual as the change of the environmental variable. One example where the gradual change of a single environmental causing an abrupt change of the system is the switch from an aerobic to anaerobic system. This system hass been investigated by Bush et al. (2017). We wanted to take this investigation one step further, and look at the role of an increased biodiversity plays in these dynamics (REF NEEDED, 2222). For this purpose, we developed this package.

The microxanox package is a package for simulating a three functional group system (*CB*: cyanobacteria, *PB*: phototrophic sulfur bacteria, and *SB*: sulfate-reducing bacteria) with four chemical substrates (*P*: phosphorus, *O*: oxygen, *SR*: reduced sulfur, *SO*: oxidized sulfur). It includes feddback between biogeochemical processes and is based on (Bush et al., 2017) (See (Bush et al., 2017) for a detailed discussion of the model).

The aims of the `microxanox` package are twofold. Firstly, the package aims at reproducing the results shown by (Bush et al., 2017), which is acomplished in the vignette Partial reproduction of Bush et al. Secondly, to take these results one step further, it includes new functionality to adress our research question as presented in (REF NEEDED, 2222).

For this, we extended the model and added functionality for:

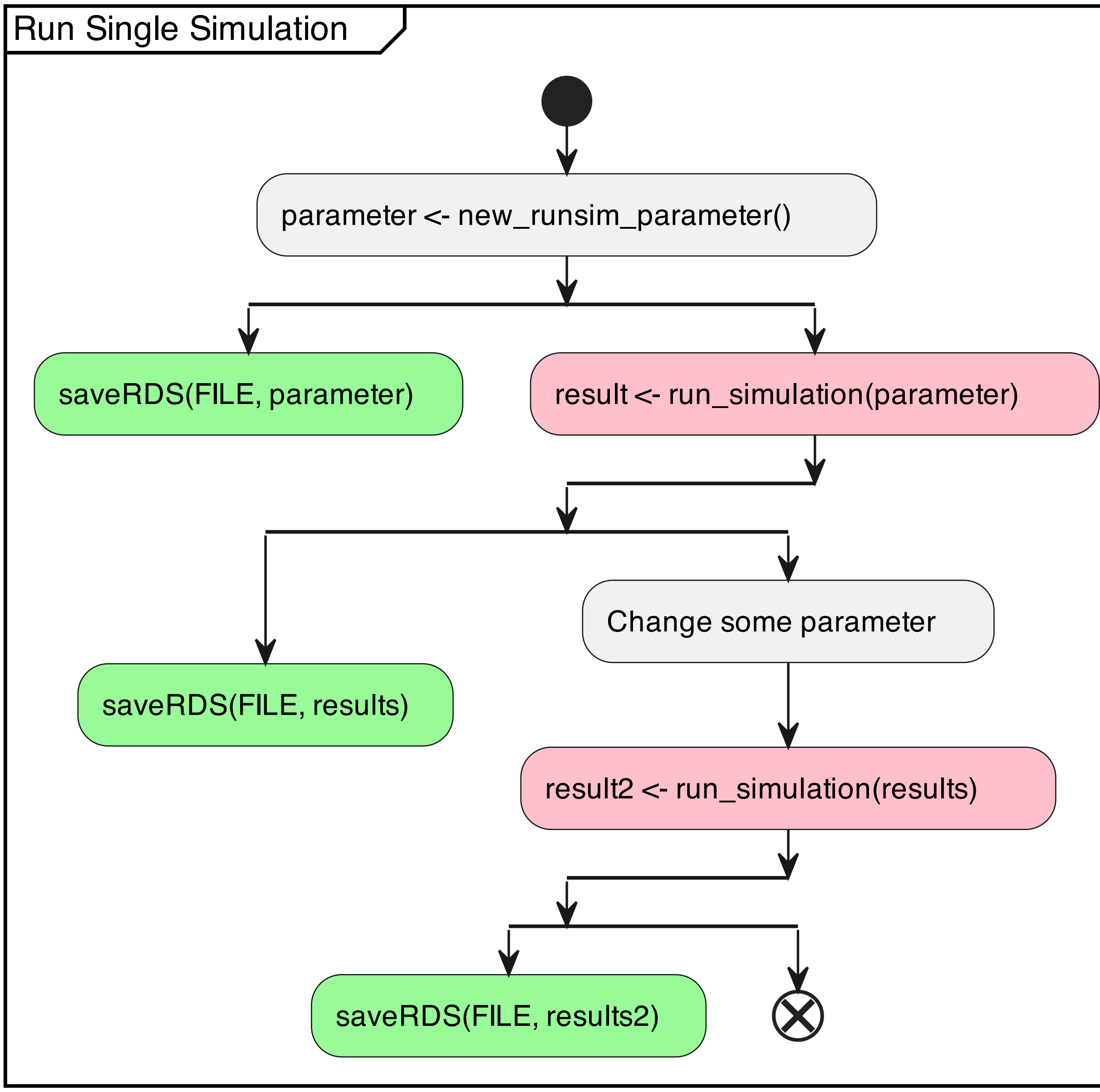
- Multiple strains (effectively unlimited) per functional group.
- Adding temporally varying oxygen diffusivity.
- Adding random noise in substrate concentrations.
- Including immigration.
- Setting minimum population abundances.

In addition to the model itself, the package includes some functions to analyse the results as well as visualise the results to provide a starting point for customised visualisations based on own requirements.

3. Software description

{»Describe the software in as much as is necessary to establish a vocabulary needed to explain its impact.«}

```
@startuml
partition "Run Single Simulation" {
   start
  :parameter <- new_runsim_parameter();
  split
    #palegreen:saveRDS(FILE, parameter);
    detach
  split again
    #pink:result <- run_simulation(parameter);
  end split
  split
    #palegreen:saveRDS(FILE, results);
    detach
  split again
    :Change some parameter;
    #pink:result2 <- run_simulation(results);
  end split
  split
    #palegreen:saveRDS(FILE, results2);
    detach
  split again
    end
  end split
}
@enduml
```



The package is not intended to provide a modelling framework which can be adjusted easily to all needs, but primarily a tool to implement the model used by (Bush et al., 2017) and to extend it to our needs (REF NEEDED, 2222). Consequently, any more substantial changes and adaptaitions, are likely to need a change in the source code.

Nevertheless, the model is structured in a way which builds on a modular structuere, so that e.g. the event definition can easily be changed. or other aspects can be adjusted. All values in the parameter object can be changed as needed and the general structure of the code should make it not to difficult to adapt the model to other similar systems.

3.1. Software Architecture

{»Give a short overview of the overall software architecture; provide a pictorial component overview or similar (if possible). If necessary provide implementation details.«}

The framework used when writing this package aims at reproducibility of the results. It builds on the folowing main considerations:

1. all parameter needed to run a simulation or find a stable state are contained in a single parameter object. This object is created by using the functions `new_..._parameter()`, `new_initial_state()` and `new_strain_parameter()`. Which one of the `new_..._parameter()` functions has to be used when, will be discussed in the section @ref(runsim) and in the User Guide.
2. The function call `run_...(parameter)` will run the simulation using the parameter as defined in the object `parameter`.
3. The return value of the `run_...(parameter)` function is identical to the parameter object plus an additionl slot named `results` which contains the results of the run
4. As this return value contains all parameter, it is possible to re-run the simulatuion by simply running `run_...(result)`.

The point that the results object contains all parameter needed to run the simulation, promotes reproducibility and makes incremental changes of individual parameters and re-running the simulations much easier.

A typical simulation would look as followed:

```r
## Create the parameter
parameter <- new_runsim_parameter()
# manually setting certain parameter

## Run the simulation and save the result
result <- run_simulation(parameter)
saveRDS(result, "sim1.rds")

## Do other stuff, e.g. plotting

## Load results, change some parameter, and rerun the simulation and save the result
parameter <- loadRDS("sim1.rds")
# change some parameter
result <- run_simulation(parameter)
saveRDS(result, "sim2,rds")
```

3.2. Software Functionalities

{»Present the major functionalities of the software.«}

We will now discuss the general structure and functionality of the package without going into to much detail. A more detailed discussion can be found in the User Guide.

The ODEs for the rates of change are specified in the function `bushplus_dynamic_model()`. This augmented version of the model published in (Bush et al., 2017) can handle multiple strains within each of the three functional groups, temporal variation in oxygen diffusivity, and events.

In the following sections we describe the general usage of the package: running one simulation, finding steady states across an environmental gradient, calculating measures of stability, and visualization.

4. Illustrative Examples

{»Present the major functionalities of the software.«}

4.0.1. Running one simulation

The individual simulation (`run_simulation()` function) is the working horse in this package. In this function, the ODEs are solved. The function needs only one argument - an object as created by the function `new_runsim_parameter()`. One parameter of this object is the `strain_parameter` which can be created by the function `new_strain_parameter()`. For a detailed description of the parameter and how they are created please see the User Guide and which acomnpanies the package or is available at User Guide

After the parameter object has been defined, it can be used in the `run_simulation()` function. The function returns an object which is identical to the parameter, except of an additional slot containing the results. This design produces a fully reproducible object as it can be used instead of a parameter object to be fed back into the `run_simulation()` function to run the simulation again from the parameter used to generate the results from.

The function `plot_dynamics()` plots a single simulation run, as returned from the `run_simulation()` function. This functio is only provided as a convenience function to provide a way to easily see the results of a simulation run. An example plot resulting from this function is shown in @ref(fig:plot-dynamics).

4.0.2. Finding a Steady State of the model

There are two methods for finding steady states. The first runs a separate simulation for each combination of starting conditions and oxygen diffusivity (let us term this the *Replication method*). The second runs only two simulations, with step-wise and slowly temporally increasing or decreasing oxygen diffusivities and recorded of state just before change to a new oxygen diffusivity (let us term this the *Temporal method*).

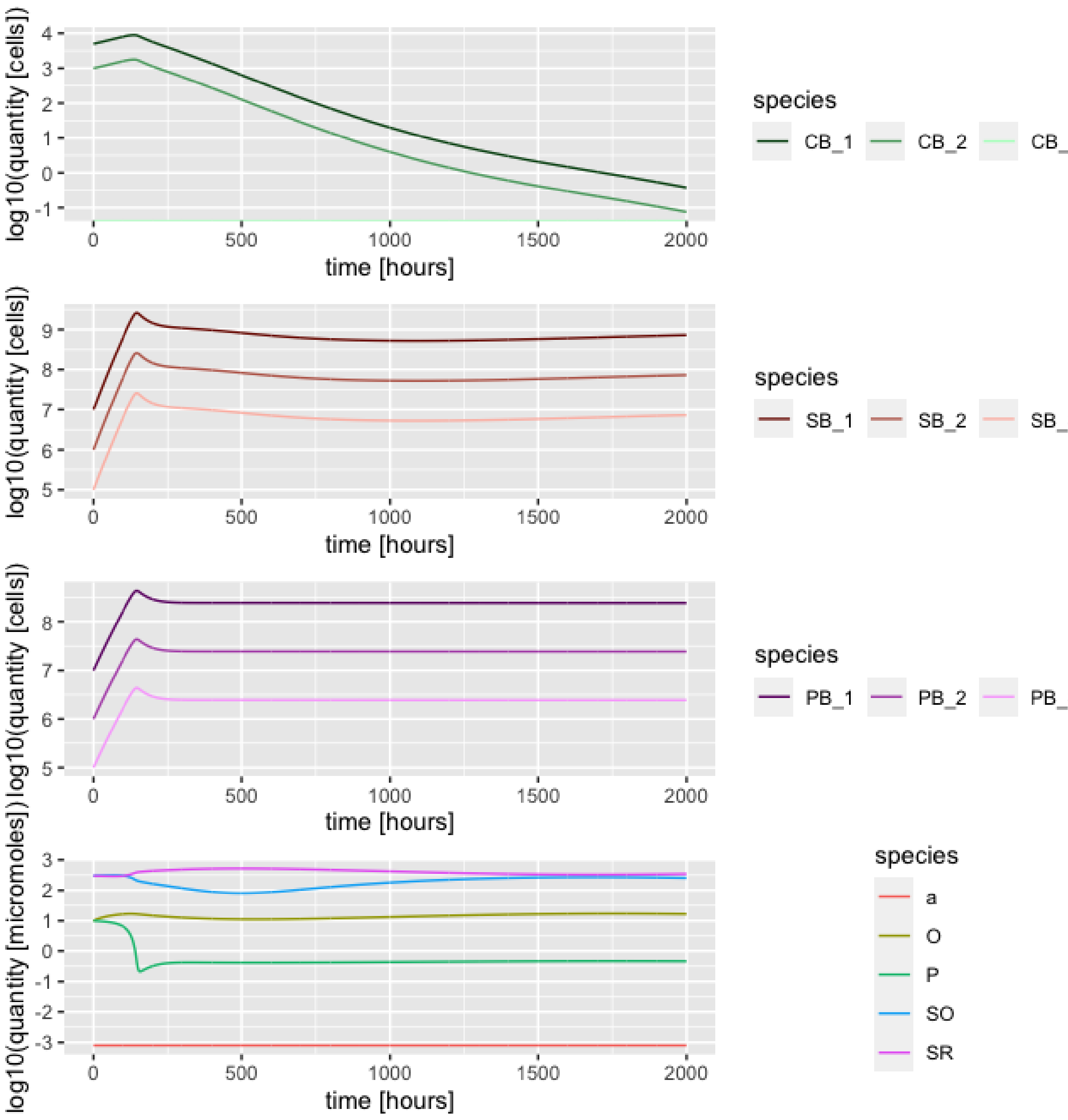


Figure 1: Plot of results of a simulation run using the function $plot_dynamics()$. Details can be found in the "User Guide" section "Three strains per functional group".

4.0.2.1. Replication Method. The replication method is implementad in the function `run_replication_ssfind()` which takes a parameter object as returned by the function `new_replication_ssfind_parameter()` and the number of cores for multithreading the simulation.

4.0.2.2. Temporal Method. The temporal method involves two simulations for a particular system configuration (parameter set). In one simulation the oxygen diffusivity is *increased* in a step-wise fashion. In the other it is *decreased* in a step-wise fashion. That is, oxygen diffusivity is held at a constant level for long enough for steady state to be reach, that state is recorded, and then a slightly higher (or lower) oxygen diffusivity value is set. Hence, at that time point, the system is effectively started with initial conditions that are the state of the system in the previous time step.

This is implemented in the function `run_temporal_ssfind()`, which takes a parameter object as created by the function `new_temporal_ssfind_parameter()` and a number indicating the .

For a more detailed walk-through of these two approaches and explanation please see the User Guide.

4.0.3. Extracting Stability Measures

From the raw results returned by these `run_...()` functions, the stability measures can be extracted by using the function `get_stability_measures()`. These measures include non-linearity and hysteresis measures, of the response of the simulated system to environmental change.

5. Impact

{»This is the main section of the article and the reviewers weight the description here appropriately Indicate in what way new research questions can be pursued as a result of the software (if any). Indicate in what way, and to what extent, the pursuit of existing research questions is improved (if so). Indicate in what way the software has changed the daily practice of its users (if so). Indicate how widespread the use of the software is within and outside the intended user group. Indicate in what way the software is used in commercial settings and/or how it led to the creation of spin-off companies (if so).«}

The first two use cases are described in detail in the User Guide and the Partial Reproduction Vignettes. The third is taken from the REF NEEDED (2222) for which this R package was designed. All of these use cases can be expanded to larger numbers of strains per functional group and variable values van be changed.

5.1. Regime shifts during temporal environmental change

In the User Guide we used a one strain system (section "1 strain per functional group") and three strain system (section "3 strains per functional group") to determine as an example the stable states during temporal environmental changes (the oxygen diffusivity). From these simulatiuons, we extracted measures of nonlinearity and hysteresis. See Fig @ref(fig:plot-dynamics) as an example plot of the simulations.

5.2. The extent of hysteresis depends on community composition

One of the reasons to develop this package was to reproduce the results presented in Bush et al. (2017), this was achieved as demonstrated in the Partial Reproduction supplement. All aspects in the paper could be reproduced and are shown in the vignette.

5.3. Effects of functional diversity on regime shifts

As discussed in the paper (REF NEEDED, 2222), the role biodiversity plays in abrupt regime shifts based on gradual changing environmental parameter is not well understood. This model (as part of the package) has been used to investigate these dynamics and the results are available in REF NEEDED (2222).

{»More details from the paper.«} {»Romana will provide two or three sentences«}

6. Conclusions

Set out the conclusion of this original software publication.

7. Conflict of Interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

8. Acknowledgements

Optionally thank people and institutes you need to acknowledge.

References

Bush, T., Diao, M., Allen, R.J., Sinnige, R., Muyzer, G., Huisman, J., 2017. Oxic-anoxic regime shifts mediated by feedbacks between biogeochemical processes and microbial community dynamics. Nature Communications 8, 789. doi:10.1038/s41467-017-00912-x

REF NEEDED, A., 2222. REFERNECE NEEDED. Journal of missing references.

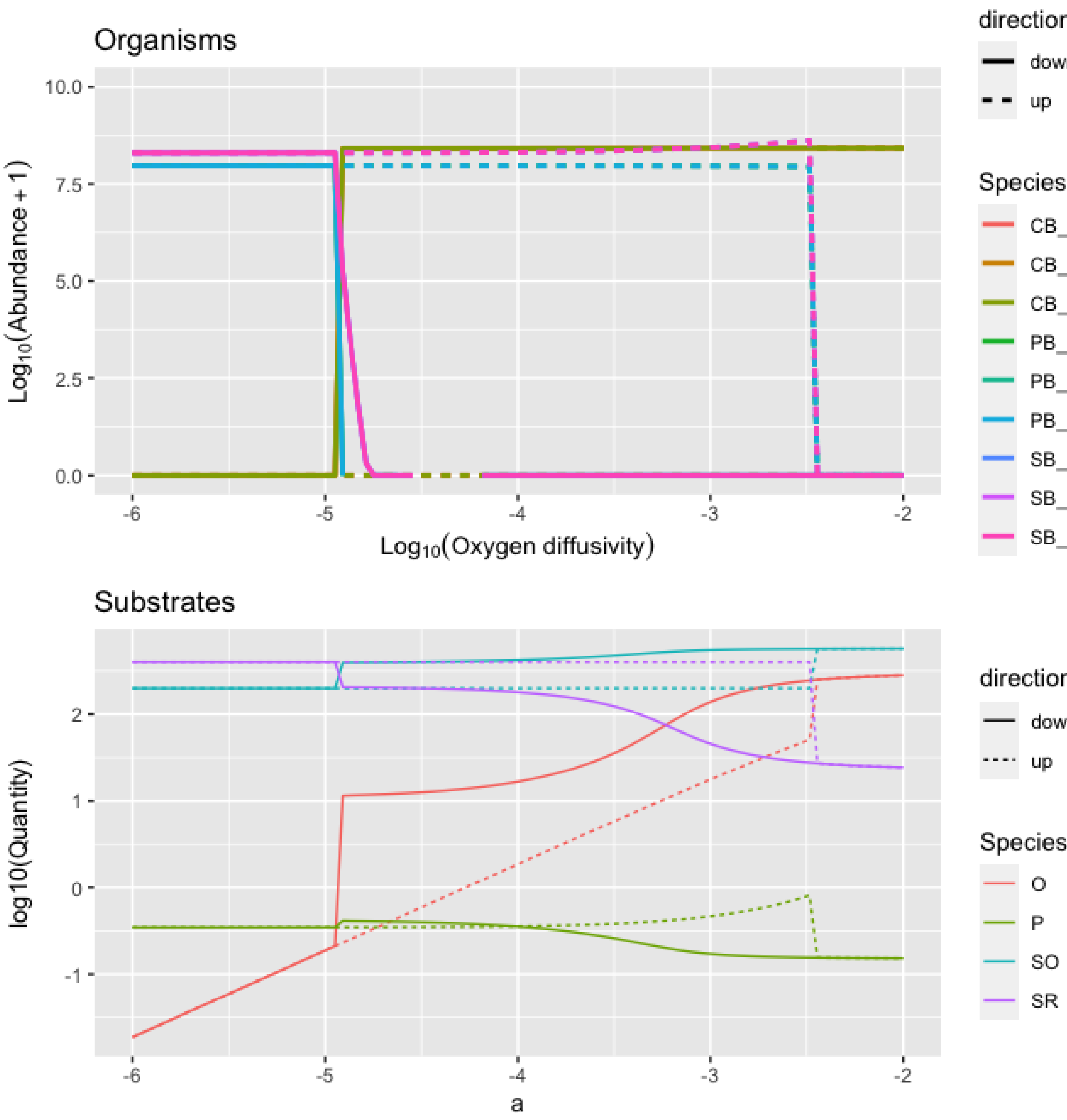


Figure 2: Plot of the stable states of the simulation runs under different oxygen diffusivity. The top graph are the Organisms (ech initially with three strains) while the lower graph is the substrate availability under the same oxygen diffusivities. Details can be found in the "User Guide" section "Three strains per functional group".

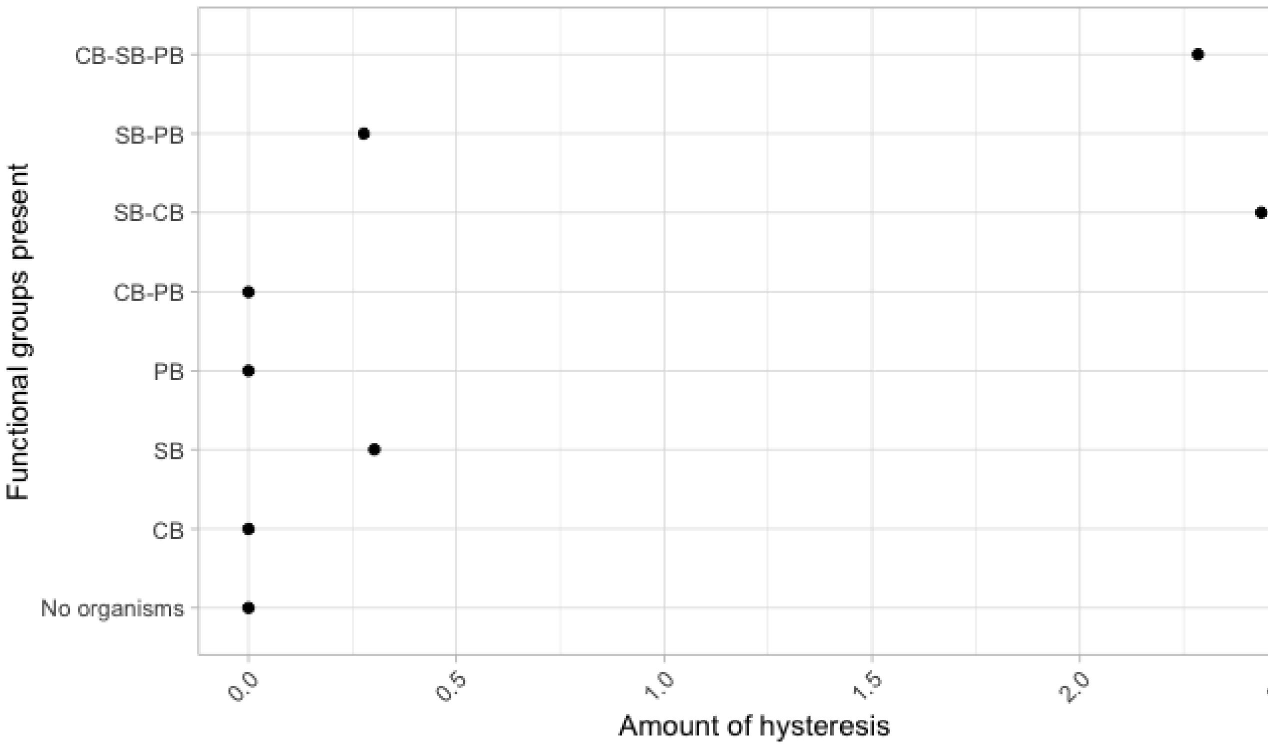


Figure 3: Hysteresis of all assessed combinations of variability.