Microxanox - an R package to simulate ...

Rainer M Krug Owen L. Petchey

last-modified

1. set the context for and purpose of the work;
2. indicate the approach and methods;
3. outline the main results;
4. identify the conclusions and the wider implications.

Table of contents

List of Figures

Introduction

{»Biological justification and background - one paragraph«} {»Rewrite the abstract for this«}

Abstract Romana of paper: The role of functional diversity in ecosystem regime shifts

Biodiversity can increase the resilience of ecosystems facing environmental change. However, we have limited understanding if this holds true for ecosystems that respond to gradual environmental change with abrupt regime shifts. Here we investigated if functional diversity influences the resilience of ecosystems that abruptly transition between alternative ecosystem states. As a case study, we used a mathematical model developed by Bush et al. (2017) which describes anoxic-oxic regime shifts of aquatic ecosystems mediated by cyanobacteria, sulfate-reducing bacteria, and phototrophic sulfur bacteria. We extended this model so as to include functional diversity in one, two, or all three functional groups. Specifically, we manipulated the amount of variation in environmental tolerance and in maximum growth rate, and explored how trait variation influenced anoxic-oxic regime shifts along a gradient of oxygen influx. We found that greater diversity of either cyanobacteria or sulfate-reducing bacteria increased the resilience of the ecosystem state that these groups dominated, while greater diversity of phototrophic sulfur bacteria reduced the resilience of the state they dominated. We also found that simultaneous variation in multiple functional groups reduced, reversed, or did not change the diversity effect of individual functional groups. Collectively, our findings highlight the importance of considering multiple interacting functional groups when predicting the effect of functional diversity on ecosystem resilience

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.

The Model

The model is an implementation of the model described by (Bush et al., 2017). It is an ordinary differential equation (ODE) based numerical non-stochastic model which was implemented in form of an R package.

We use the notation from the Table S1 of the Supplementary information of (Bush et al., 2017)), and not the notation in the main manuscript of (Bush et al., 2017). In the main manuscript the notation is simplified to `y_S_PB` ($y_{S,PB}$) (which is `y_SR_PB` ($y_{SR,PB}$) in Table S1) and `y_S_SB` ($y_{S,SB}$) (which is y_SO_SB ($y_{SO,SB}$) in Table S1). We also use `Pr_cB` (Pr_B) instead of `P_CB` (P_{CB}).

All parameter and variables follow these base dimensions:

- Time: `hours`
- Volume: `litres`
- Substrate quantity: `micromoles`
- Organism quantity: `cells`

{» Add a more detailed description of the model and the additions «} {»TODO«}

The ODEs are solved using the "General Solver for Ordinary Differential Equations" of the `deSolve` package (Soetaert, Petzoldt, & Setzer, 2010).

Availability and Accessability of the Package

The package is available in a github repo/. The newest version is also available under the DOI or via RUniverse (REF NEEDED, 2222). This manuscript describes version 1.0 which is available under DOI or installable within R by using `remotes::install_github("UZH-PEG", ref = "v1.0")`.

The package will be updated based on further research activities. It will be tested continuously for the last recent, current and devel version of R. {»If we want that, we have to include some, even rudimentary, GITHUB tests.«} {»RMK: Agreed - let's discuss this when we talk about the package paper.«}

Issues, questions and suggestions should be communicated through the issue tracker at https://github.com/UZH-PEG/microxanox/issues

The R Package

Documentation

The package contains the full documentetion of the functions, which includes

1. Standard R help for each function (`?COMMAND` in R)
2. Two vignettes acompanying the package:

- A **User Guide**, also available on the R-Universe Vignette 1. This gives an introduction of the package and its functions and provides a walk-through of the process of running, visualizing and extra and Vignette 2

3. This paper as a third vignette, also available under RUniverse Vignette 3

The modelling framework

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:

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


## Run the simulation and save the result
result <- run_simulatrion(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_simulatrion(parameter)
saveRDS(result, "sim2,rds")
```

Using the package

We will now discuss the general structure and functiuonality 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 visualisation.

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 parameter - 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 paramter 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).

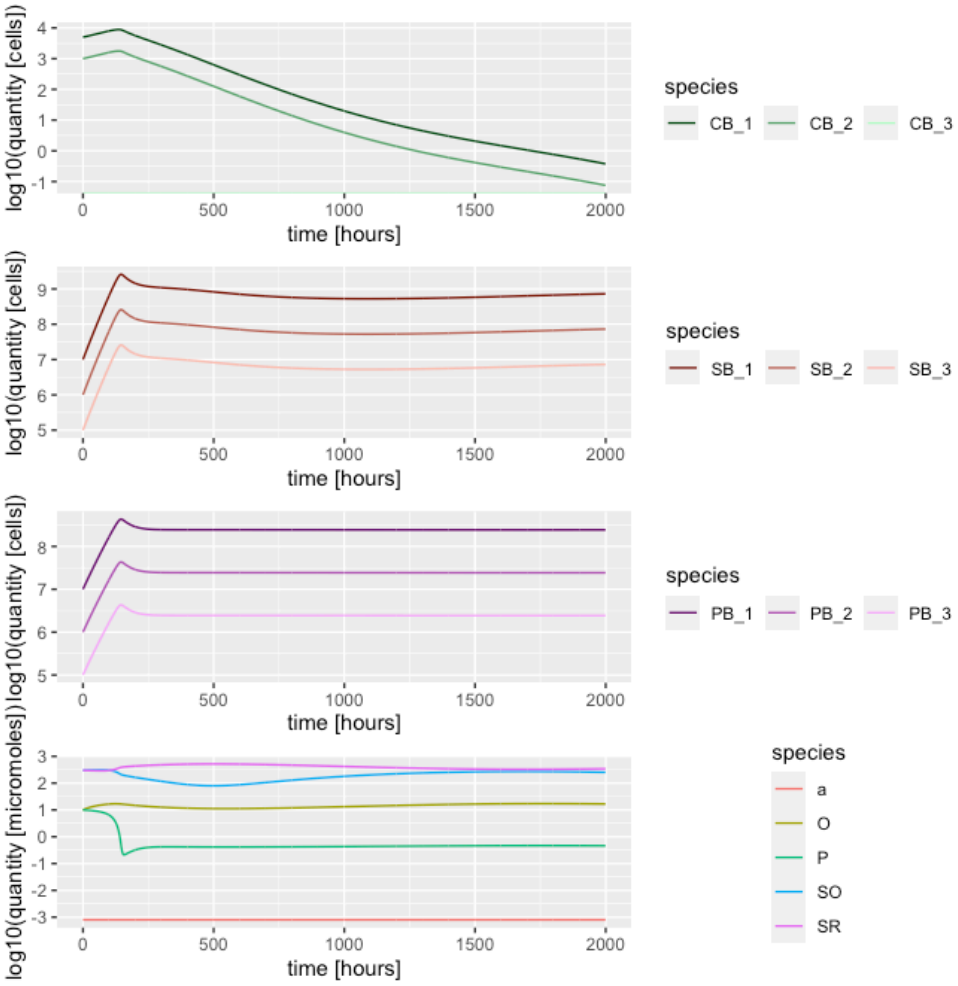

Figure 1: Plot of results of a simulation run using the function `plot_dynamics()`$‘. Details can be found in the “User Guide” section “Three strains per functional group”.

Find 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*).

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.

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.

Extract 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.

Use cases

The first two use cases come from the User Guide and the Partial Reproduction Vignettes. The third comes from a research article that relied on the package. All of these use cases can be expanded to larger numbers of strains per functional group.

Regime shifts during temporal environmental change

In the User Guide we ued a one strain system (section "1 strain per functional group") and three strain system (section "3 strains per functional group") to analyse as an example the determination of the stable states during temporal environmental chan ge (the oxygen diffusivity). From these simulatiuons, we extracted measures of nonlinearity and hysteresis.

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.

Effects of functional diversity on regime shifts

This model (as part of the package) has been used in the production of the paper (REF NEEDED, 2222). It can be used for similar studies in this study systems.

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

General usability and flexibility of the model

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.

Conclusions

{»Dependant on journal«}

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*(1), 789. doi:10.1038/s41467-017-00912-x

REF NEEDED, A. (2222). REFERNECE NEEDED. *Journal of Missing References*.

Soetaert, K., Petzoldt, T., & Setzer, R. W. (2010). Solving differential equations in R: Package deSolve. *Journal of Statistical Software*, *33*(9), 1–25. doi:10.18637/jss.v033.i09