Tutorial: Analysing iDynoMiCS Result Data using iDynoR Package

Table of Contents

1. Introduction

iDynoMiCS is a computer program, developed by an international team of researchers, whose purpose is to model and simulate microbial communities in an individual-based way. It is described in detail in the paper "iDynoMiCS: next-generation individual-based modelling of biofilms" by Lardon et al, published in Environmental Microbiology in 2011. The simulation produces results in XML file format, describing the state of each species in each timestep (agent_State), a summary of the species statistics for a timepoint (agent_Sum), the state of each solute grid in each timestep (env_State) and a summary of the solutes for a timestep (env_Sum). This R package provides a means of reading this XML data into R such that the simulation response can be statistically analysed.

Do note that this tutorial is specific to the iDynoR package, and not iDynoMiCS itself, and thus does not duplicate any of the information in the iDynoMiCS tutorial. Please ensure that you have studied the iDynoMiCS tutorial, and understand the simulation output format descriptors, before going through the tutorial for iDynoR. Also note that this tutorial assumes that you have some experience in using the R statistical analysis package.

2. Prerequisites

- The R statistical environment, version 2.13.1 or later.
- The iDyno R package, downloaded from the Comprehensive R Archive Network (CRAN) or from idynomics.org
- The XML and vegan R packages, available for download from CRAN.
- Tutorial example simulation results, available from the project website, or within the package

3. Installing iDynoR

There are two ways to install the iDynoR package into your R environment. For both methods, you should ensure that you have installed the `vegan` and `XML` R packages first:

3.1. Easiest Way: From the CRAN repository

Open the R environment (Linux/Mac: in the terminal, type R in most cases; Windows: Open from Programs menu). Enter the following at the prompt:

```
install.packages("iDynoR")
```

Or, to install to a specific directory, type the following:

```
install.packages("iDynoR",lib="/path/to/directory")
```

3.2. Download from the iDynoMiCS Website and Install From Source

Download the package from the iDynoMiCS website. Then:

Linux/Mac:

Open a terminal window and navigate to the directory where the iDynoR.tar.gz file has been saved. To install in the R default directory, type the following:

```
R CMD INSTALL iDynoR.tar.gz
```

To install to a specific directory, type the following:

```
R CMD INSTALL iDynoR.tar.gz -l /path/to/directory/
```

Windows:

Open the R environment, and type:

```
install.packages("[path to where you downloaded iDynoR]")
```

4. The Example Simulation Results

In this tutorial, we will work with simulation results generated from one of the example protocol files: multi_species_multi_substrate_nitification_2D.xml. You should have a read through the protocol file so you understand the simulation setup. You can either run this simulation yourself, download the example results set from the iDynoMiCS website, or unzip them from the inst/extdata folder of the iDynoR package itself. If you run the simulation yourself, you may get differing results to those that you see in this tutorial.

Whether you run the simulation yourself or download the example set, extract the agent_State.zip, agent_Sum.zip, env_State.zip, and env_Sum.zip files. There is no need to change the name of the extracted folders. Your results folder should look like that observed in Figure 1.

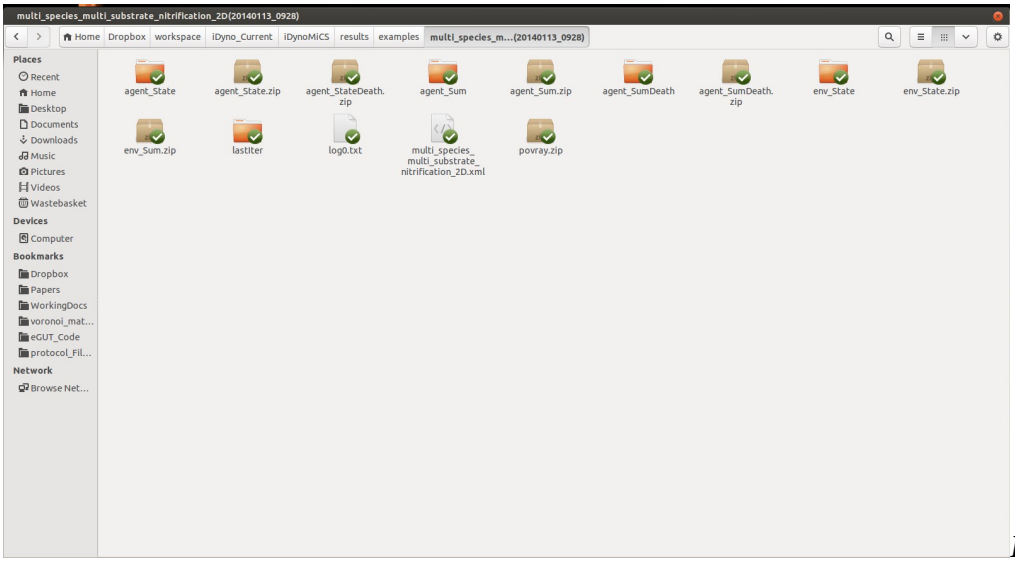


Figure 1: iDynoMiCS Results folder structure, with the relevant zip folders extracted

5. Reading in Result File Data

During the course of a simulation, iDynoMiCS will save output files describing the current agent states. These output files will be written at the interval specified by the outputPeriod parameter in the simulator mark-up of the protocol file. In each file name, the character in brackets represents the iteration number at which the file was written. These files are known as agent_state, agent_sum, env_state, and env_sum files: a detailed description of each can be found in the iDynoMiCS tutorial available from iDynoMiCS.org. This method reads an XML file into R, returning this as a structure that can then be accessed by the other methods within this package. Thus, this method is one of the key methods that you will use if you are utilising this package to analyse simulation results. This method works with all the four files specified above. Note that to use this method, you should extract the ZIP files produced by iDynoMiCS before running this method.

In this example, we are going to read in the agent_State file at iteration 40 of this simulation run. To do this, enter the following at the R prompt. The `[resultFolder]` is the full path to the simulation result folder structure, as seen in Figure 1. If you are using the example data downloaded from the iDynoMiCS website, this will be the full path where you saved that folder. **Note that the directory separator in R is always a forward slash "/", and not a backslash as used in Windows.**

```
library(iDynoR)
result40<-readSimResultFile("[resultFolder]", "agent_State", 40)
```

where "agent_State" is the type of file being analysed, and 40 is the timepoint.

This stores the result in the data frame `result40`. In the following methods, we will process data contained in this data frame.

When reading in other types of result file, simply replace "agent_State" with either "agent_Sum", "env_State", or "env_Sum".

6. Reading Data From Agent State and Agent Sum Files

The agent_State and agent_Sum files describe the state of the agents in the system; the agent_State file describes each agent in detail, while the agent_Sum file summarizes the agents on the species level. This section describes methods that can be utilised to extract data from an agent_state or agent_sum file for processing using statistical methods. This provides basic functionality to extract the data they need, then analysis scripts that utilise this data can be written accordingly.

In this example, we are going to extract information from the agent state file at iteration 40. Follow the steps below to read this result data into R. Note that lines that begin # are comments, and for explanation purposes, and thus do not need to be copied into the R terminal. Again, the `[resultFolder]` is the full path to the simulation result folder structure, as seen in Figure 1.

```
# Read in the simulation result file
result40<-readSimResultFile("[resultFolder]","agent_State",40)

# Get the simulation iteration that produced this result
iteration<-agent_returnSimIteration(result40)

# Get the simulation time represented by this result (i.e. hours)
time<-agent_returnSimTime(result40)

# Get the domain information (size and resolution)
res<-agent_returnGridResolution(result40)
i<-agent_returnIVoxels(result40)
j<-agent_returnJVoxels(result40)
k<-agent_returnKVoxels(result40)

# Get all the species information from the file
allSpecies<-agent_returnSpeciesResultData(result40)

# Get the number of species in this simulation
numSpecies<-agent_returnNumSpecies(allSpecies)

# Total the biomass column for each individual of a species
#(for example, MyAutotrophs)
biomassTotal<-agent_returnSpeciesColumnTotal(allSpecies, "MyAutotrophs",
"biomass")
```

In addition to agent_State files, the methods above also work for retrieving information from the agent_Sum result files.

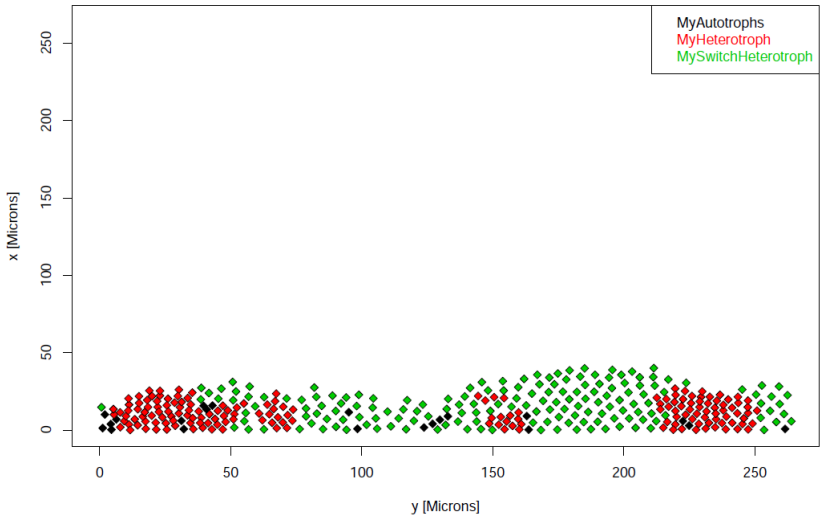
7. Example Methods for Processing Agent Information

The previous section detailed methods that have been provided for extracting information from an agent_state or agent_sum file. This section describes exemplar methods that have been provided to show how this data could be processed. Remember however that this package has been provided to enable you to access simulation data in R, and we hope that you will be able to build on these methods that we provide. Note that these methods read in the agent_state or agent_sum files, thus there is no need to independently read in the simulation file before running these methods.

Type the following commands into R (ignoring the comment lines that begin with the # character). Again, the `[resultFolder]` is the full path to the simulation result folder structure, as seen in Figure 1. In addition, [graphFolder] is the full path to a folder where any graph output should be saved.

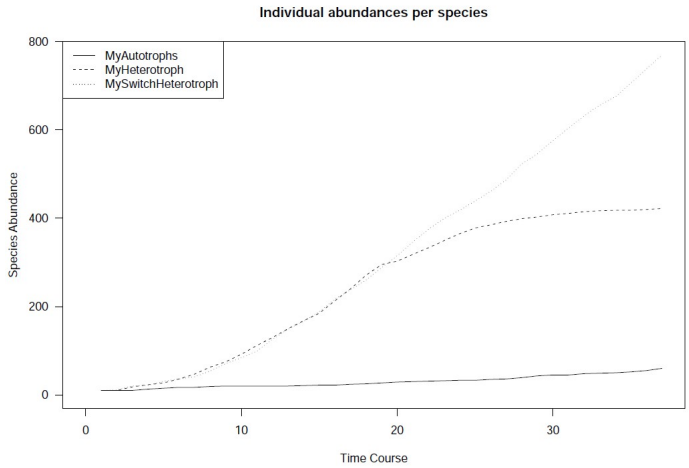
```
# Get a data frame containing the biomass of species MyAutotrophs over
# time, for all 720 timepoints, specifying that the output period was 20
totalBiomass<-agent_getMeasureOverTime("[resultFolder]","agent_State",
720, 20, "MyAutotrophs", "biomass")

# Plot all the agent positions in the simulation at the 260th timestep
# Graph saved as "agent_Plot_260.pdf" in the folder specified
plotAgents("[resultFolder]", 260, "[graphFolder]")
```

Example Output for plotAgents:



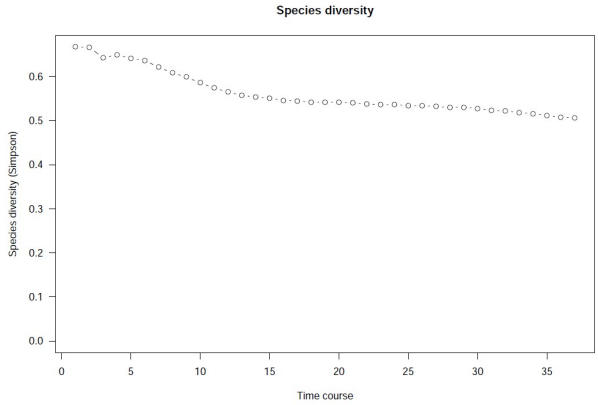

```
# Get the abundance of each species throughout the simulation
# (720 timepoints, output period of 20)
speciesAbundance<-getSpeciesSpecificAbundance("[resultFolder]", 720, 20)

# Plot the abundance of each species (720 timepoints, output period: 20)
# Graph saved as "By_Species_Abundance.pdf" in folder specified
plotTimeCourseAgents("[resultFolder]", 720, 20, "[graphFolder]")
```

Example Output for plotTimeCourseAgents:



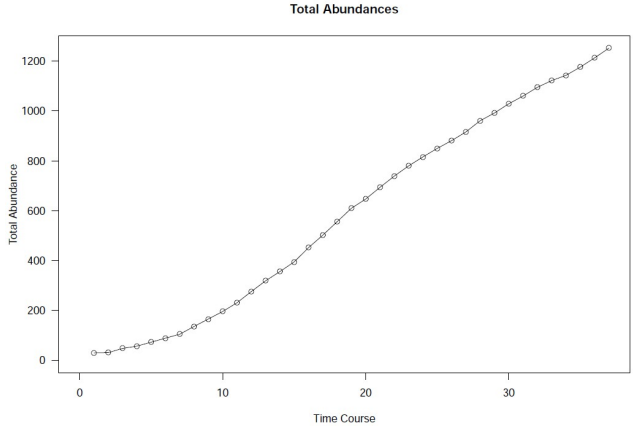

```
# Produce a diversity plot of this information, and store the data shown
# (720 timepoints, output period of 20)
# Graph saved as "Species_Diversity_Simpson.pdf" in the folder specified
simpsonData<-simpsonIndex("[resultFolder]", 720, 20, "[graphFolder]")
```

Example Output for simpsonData:




```
# Get the total abundance of individuals throughout the simulation
# (720 timepoints, output period of 20)
totalAbundance<-getSpeciesAbundance("[resultFolder]", 720, 20)

# Plot the total abundance (720 timepoints, output period of 20)
# Graph is saved as "Total_Abundance.pdf" in the folder specified
plotTimeCourseAbund("[resultFolder]", 720, 20, "[graphFolder]")
```

Example Output for plotTimeCourseAbund:



8. Reading Solute State (env_State) and Solute Sum (env_Sum) Files

The env_State and env_Sum files describe, respectively, the overall state of the solute fields and a more summarized version. This section describes methods that can be utilised to extract data from an env_state or env_sum file for processing using statistical methods. This provides basic functionality to extract the data, from which addititonal analysis scripts can be created.

In this example, we are going to extract information from the env_state file at iteration 40. Follow the steps below to read this result data into R. Again, lines that begin # are comments, and for explanation purposes, and thus do not need to be copied into the R terminal. The `[resultFolder]` is the full path to the simulation result folder structure, as seen in Figure 1.

```
# Read in the env_State simulation result file, at iteration 40
simResponse<-readSimResultFile("[resultFolder]","env_State",40)

# Get the simulation iteration that produced this file
iteration<-env_returnSimIteration(simResponse)

# Get the simulation time represented by this result (i.e. hours)
time<-env_returnSimTime(simResponse)

# Get the grid information for a given solute. In this example data,
# the first solute is o2d, represented by the number "1"
res<-env_returnSoluteGridRes(simResponse,1)
i<-env_returnSoluteGridIVoxels(simResponse,1)
j<-env_returnSoluteGridJVoxels(simResponse,1)
k<-env_returnSoluteGridKVoxels(simResponse,1)

# Get the biofilm thickness information from the file
meanThick<-env_returnMeanBiofilmThickness(simResponse)
maxThick<-env_returnMaxBiofilmThickness(simResponse)
stdDevThick<-env_returnStdDevBiofilmThickness(simResponse)

# Get the solute grid information for a particular solute, such as o2d.
# In this example data set, o2d is the first solute
o2dGrid<-env_returnSpecifiedSoluteData(simResponse, 1)
```

In addition to env_State files, the above methods above also work for retrieving information from the env_Sum result files.

The following are env_Sum file specific, and are used to obtain information from the file concerning production and consumption.

```
# Get the global production and consumption rates of solutes at this
# timepoint
gpr<-env_returnGlobalProductionRates(simResponse)

# Get the concentration and rate change of solutes, at this timepoint
c_rc<-env_returnConcentrationAndRateChange(simResponse)
```

9. Example Methods for Processing Solute Information

The previous section detailed methods that have been provided for extracting information from an env_State or env_Sum file. This section describes exemplar methods that have been provided to show how this data can be processed. Remember however that this package has been provided to enable you to access simulation data in R, and we hope that you will be able to build on these methods that we provide. Note that these methods read in the env_State or env_Sum files, thus there is no need to independently read in the simulation file before running these methods.

Type the following commands into R (ignoring the comment lines that begin with the # character). Again, the `[resultFolder]` is the full path to the simulation result folder structure, as seen in Figure 1. In addition, [graphFolder] is the full path to a folder where any graph output should be saved.

```
# Track the production rate of o2d (solute 1) over time,
# for 720 timepoints, with an output period of 20
o2dProduction<-
env_soluteProductionRateOverTime("[resultFolder]","env_Sum", 720, 20, 1)

# Plot the contour for o2d (1) at timepoint 40
# This will be saved as "Contour_Solute_1_Iteration_40.pdf" in folder
# specified
plotContour("[resultFolder]", 40, 1,"[graphFolder]")
```

Example Output for plotContour: