Package 'iDynoR'

February 20, 2015

Type Package

Title R Analysis package for iDynoMiCS Simulation Results

Version 1.0

Date 2014-01-08

Author Kieran Alden, Jan-Ulrich Kreft

Maintainer Kieran Alden <kieran.alden@gmail.com>

Description 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. iDynoMiCS is available from the website iDynoMiCS.org, where a full tutorial on using both the simulation and this R package is provided.

Depends XML, vegan

License CeCILL

NeedsCompilation no

Repository CRAN

Date/Publication 2014-01-14 11:14:43

R topics documented:

Technique 1: Read in Simulation Results

Technique 1: Read in Simulation Results

Description

The iDynoMiCS simulation produces a set of XML files containing simulation output, either for each individual of each species, or each solute grid, or a summary of each species type and grid, for each timestep. 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 data frame 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.

Usage

```
readSimResultFile(resultFolder, resultFileType, timePoint)
```

Arguments

`resultFolder`	The directory containing all the simulation results. Note that this should not be the folder extracted from the zip file, but the folder where iDynoMiCS saves the simulation responses
`resultFileType`	The output response file type to read in. This should be either agent_State, agent_Sum, env_State, or env_Sum
`timePoint`	The simulation timepoint to process

References

Section 5 of the iDynoMiCS tutorial explains in detail how each output file is structured

Examples

```
## Not run:
# DONTRUN IS SET SO THIS IS NOT EXECUTED WHEN PACKAGE IS COMPILED - BUT THIS
# HAS BEEN TESTED THOROUGHLY BEFORE UPLOADING TO THE REPOSITORY

simResponse<-
readSimResultFile("/home/user/iDynoMiCS/results/","agent_State",40)

## End(Not run)
```

`Technique 2: Reading Agent State and Agent Sum Files`
Technique 2: Reading Agent State and Agent Sum Files

Description

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 simulation protocol file (see Protocol File section of iDynoMiCS tutorial for more information). In each file name, the number in brackets represents the simulation timestep at which the file was written. 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.

The previous technique read the file into a structure that can be processed in R. 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 the user with the basic functionality to extract the data they need, then write their analysis scripts accordingly. Some example analysis scripts that use the methods in this section have been provided, and are described in Technique 3.

The following methods are available for both agent_State and agent_Sum files:
agent_returnSimIteration: Returns the simulation iteration that produced this output file.
agent_returnSimTime: Returns the simulation time represented by that iteration number (hours).
agent_returnGridResolution: Returns the grid resolution of the simulation domain.
agent_returnIVoxels: Returns the number of voxels in the I direction of the domain.
agent_returnJVoxels: Returns the number of voxels in the J direction of the domain.
agent_returnKVoxels: Returns the number of voxels in the K direction of the domain.
agent_returnSpeciesResultData: Extracts all of the species information from the file, storing each species in a list. The method returns a list, containing each species list. Relevant simulation responses can then be extracted from the relevant list.
agent_returnNumSpecies: Returns the number of species in this results file.
agent_returnSpeciesColumnTotal: For a particular species and simulation response of interest (stored in columns in the result file), returns the total of that response. For example, if there were 100 individuals of a particular species, this totals all responses of a specified response.

Usage

```
agent_returnSimIteration(xmlResultData)
agent_returnSimTime(xmlResultData)
agent_returnGridResolution(xmlResultData)
agent_returnIVoxels(xmlResultData)
agent_returnJVoxels(xmlResultData)
agent_returnKVoxels(xmlResultData)
agent_returnSpeciesResultData(xmlResultData)
agent_returnNumSpecies(allSpecies)
agent_returnSpeciesColumnTotal(allSpecies, speciesReqd, columnName)
```

Arguments

xmlResultData	The structure created by Technique 1, containing the data in the agent_state or agent_sum file. Created using the method readSimResultFile
allSpecies	All of the species information in the agent_state or agent_sum file. Relevant species data from this list can then be extracted by other methods
speciesReqd	Where extracting data from the species information, this specifies the species of interest
columnName	Where extracting data from the species information, this specifies the column of interest (e.g. growthRate)

Examples

```
## Not run:
# DONTRUN IS SET SO THIS IS NOT EXECUTED WHEN PACKAGE IS COMPILED - BUT THIS
# HAS BEEN TESTED THOROUGHLY BEFORE UPLOADING TO THE REPOSITORY

# Read in the results of a particular agent state file, at timestep 40
simResponse<-
readSimResultFile("/home/user/iDynoMiCS/results/","agent_State",40)

# Get the simulation iteration
iteration<-agent_returnSimIteration(simResponse)

# Get the simulation time
time<-agent_returnSimTime(simResponse)

# Get the simulation domain information (sizes and resolution)
res<-agent_returnGridResolution(simResponse)
i<-agent_returnIVoxels(simResponse)
j<-agent_returnJVoxels(simResponse)
k<-agent_returnKVoxels(simResponse)

# Get all the species information from the file, and the number of species in the file
allSpecies<-agent_returnSpeciesResultData(simResponse)
numSpecies<-agent_returnNumSpecies(allSpecies)

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


## End(Not run)
```

Description

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 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 agent_state or agent_sum files, thus there is no need to independently run Technique 1 in this case.

The following methods are available:
agent_getMeasureOverTime: Processes all agent_State or agent_Sum files for a particular simulation run, storing the result for a specified simulation output (such as biomass or growth rate) for each timestep. A data frame of these responses is returned, of one column, with each timestep represented by one row. This information could then be plotted if desired.
plotAgents: This routine will plot all the agents for a particular timepoint using the 'agent_State' file. This is useful for examining the location of each species within the biofilm. Each agent is represented by a circle coloured for each species, but note that the sizes of the circles DO NOT correspond to the actual sizes of the agents. If you want a plot of the agents with the correct agent sizes, use POV-Ray to render the pov file output from iDynoMiCS (see iDynoMiCS tutorial).
getSpeciesSpecificAbundance: Returns a data frame containing the abundance of each species in the simulation at each time-point. Useful for monitoring growth of populations.
plotTimeCourseAgents: This will plot the number of each species at each iteration, as generated by the method above.
simpsonIndex: This will plot the diversity of the community at each iteration and return a data frame of the data comprising this graph. This utilises the data frame returned by the method getSpeciesSpecificAbundance.
getSpeciesAbundance: Returns a data frame containing the total abundance of individuals (of all species) in the simulation at each time-point.
plotTimeCourseAbund: This routine plots the sum of all the abundances of all species, as generated by the method above.

Usage

```
agent_getMeasureOverTime(resultFileFolder, resultFileType,
numTimepoints, outputPeriod,speciesReqd, columnName)

plotAgents(resultFileFolder, timePoint, folderForGraphOut)

getSpeciesSpecificAbundance(resultFileFolder, numTimepoints,
outputPeriod)

plotTimeCourseAgents(resultFileFolder, numTimepoints, outputPeriod,
folderForGraphOut)

simpsonIndex(resultFileFolder, numTimepoints, outputPeriod,
folderForGraphOut)

getSpeciesAbundance(resultFileFolder, numTimepoints, outputPeriod)
```

```
plotTimeCourseAbund(resultFileFolder, numTimepoints, outputPeriod,
folderForGraphOut)
```

Arguments

resultFileFolder	The directory where iDynoMiCS stored the simulation output files.
resultFileType	The output response file type to read in. This should be either agent_State, agent_Sum, env_State, or env_Sum
numTimepoints	The number of timepoints comprising this simulation run
outputPeriod	The number of timepoints between simulation output. See protocol file tutorial
timePoint	A particular timepoint from which the agent positions should be plotted
folderForGraphOut	Each graph is output to file. This should specify the folder where the graph should be written to
speciesReqd	The name of the species of interest, as a string
columnName	A string containing the column of the result file (or simulation response) of interest (such as biomass)

Examples

```
## Not run:
# DONTRUN IS SET SO THIS IS NOT EXECUTED WHEN PACKAGE IS COMPILED - BUT THIS
# HAS BEEN TESTED THOROUGHLY BEFORE UPLOADING TO THE REPOSITORY

# Track the biomass of Pseudomonas over time, for 48 timepoints
# with an output period of 1
totalBiomass<-agent_getMeasureOverTime("/home/user/iDynoMiCS/results/",
"agent_State", 48, 1, "Pseudomonas", "biomass")

# Plot all the agents in the simulation at the 24th timestep
plotAgents("/home/user/iDynoMiCS/results/", 24, "/home/user/iDynoMiCS/results")

# Get the abundance of each species throughout a simulation of 48 timepoints
# with an output period of 1
speciesAbundance<-getSpeciesSpecificAbundance("/home/user/iDynoMiCS/results/",
48,1)

# Plot the abundance of each species, 48 timepoints, output period of 1
plotTimeCourseAgents("/home/user/iDynoMiCS/results/", 48, 1,
"/home/user/Desktop/iDynoMiCS/results/single_species/graphs/")

# Produce a diversity plot of this information, and store the data in the graph
# 48 timepoints, output period of 1
simpsonData<-simpsonIndex("/home/user/iDynoMiCS/results/", 48, 1,
"/home/user/Desktop/iDynoMiCS/results/single_species/graphs/")
```

```
# Get the total abundance of individuals throughout the simulation
# 48 timepoints, output period of 1
totalAbundance<-getSpeciesAbundance("/home/user/iDynoMiCS/results/", 48, 1)

# Plot the total abundance. 48 timepoints, output period of 1
plotTimeCourseAbund("/home/user/iDynoMiCS/results/", 48, 1,
"/home/user/iDynoMiCS/results/graphs/")


## End(Not run)
```

`Technique 4: Reading Solute Grid State (env_State) and Summary (env_Sum) Files`

Technique 4: Reading Solute Grid State (env_State) and Summary (env_Sum) Files

---

Description

During the course of a simulation, iDynoMiCS will save output files describing the current environment states. These output files will be written at the interval specified by the outputPeriod parameter specified in the simulation input (protocol) file (see Protocol File section of the iDynoMiCS tutorial). In each file name, the character in brackets represents the iteration number at which the file was written. The env_State and env_Sum files describe, respectively, the overall state of the solute fields and a more summarized version.

Technique 1 reads the file into a structure that can be processed in R. 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 the user with the basic functionality to extract the data they need, then write their analysis scripts accordingly. Some example analysis scripts that use the methods in this section have been provided, and are described in Technique 5.

The following methods are available for both env_State and env_Sum files:
env_returnSimIteration: Returns the simulation iteration at which this output file was produced.
env_returnSimTime: Returns the simulation time at which this output file was produced.
env_returnSoluteGridRes: Returns the grid resolution of a specified solute grid.
env_returnSoluteGridIVoxels: Returns the number of voxels in the I direction of a specified solute grid.
env_returnSoluteGridJVoxels: Returns the number of voxels in the J direction of of a specified solute grid
env_returnSoluteGridKVoxels: Returns the number of voxels in the K direction of of a specified solute grid.
env_returnMeanBiofilmThickness: Returns the mean biofilm thickness calculated at the timepoint of a particular env_State or env_Sum file.
env_returnMaxBiofilmThickness: Returns the maximum biofilm thickness calculated at the timepoint of a particular env_State or env_Sum file.
env_returnStdDevDBiofilmThickness: Returns the standard deviation calculated from biofilm thickness at the timepoint of a particular env_State or env_Sum file.

env_returnGlobalProductionRates: Extracts the global production rate of each solute from the result file. Each is stored in a list. An R list is returned which is a nested list, containing each of these lists.
env_returnConcentrationAndRateChange: Extracts the concentration and uptake rate of each solute from the result file. Each is stored in a list. An R list is returned which is a nested list, containing each of these lists.
env_returnSpecifiedSoluteData: Extracts the concentration grid for a specified solute from the result file. This information can be used to study how the concentration changes across the grid.

Usage

```
env_returnSimIteration(xmlResultFile)

env_returnSimTime(xmlResultFile)

env_returnSoluteGridRes(xmlResultFile, soluteRequested)

env_returnSoluteGridIVoxels(xmlResultFile, soluteRequested)
env_returnSoluteGridJVoxels(xmlResultFile, soluteRequested)
env_returnSoluteGridKVoxels(xmlResultFile, soluteRequested)

env_returnMeanBiofilmThickness(xmlResultFile)
env_returnMaxBiofilmThickness(xmlResultFile)
env_returnStdDevBiofilmThickness(xmlResultFile)

env_returnGlobalProductionRates(xmlResultFile)

env_returnConcentrationAndRateChange(xmlResultFile)

env_returnSpecifiedSoluteData(xmlResultFile, soluteRequested)
```

Arguments

`xmlResultFile`	The structure created by Technique 1, containing the data in the agent_state or agent_sum file. Create using the method readSimResultFile
`soluteRequested`	The solute of interest, for which the results should be extracted. Note that this should be a number, not the name of the solute

References

The iDynoMiCS tutorial has a detailed description of the env_Sum and env_State files. Study this to ensure you understand what each part of the output response is

Examples

```
## Not run:
# DONTRUN IS SET SO THIS IS NOT EXECUTED WHEN PACKAGE IS COMPILED - BUT THIS
# HAS BEEN TESTED THOROUGHLY BEFORE UPLOADING TO THE REPOSITORY

# Read in the results of a particular env state file, in this case iteration 40
simResponse<-
readSimResultFile("/home/user/iDynoMiCS/results/","env_State",40)

# Get the simulation iteration
iteration<-env_returnSimIteration(simResponse)

# Get the simulation time
time<-env_returnSimTime(simResponse)

# Get the solute grid information, for a given solute. Let's say the first
res<-env_returnSoluteGridRes(simResponse,1)
i<-env_returnSoluteGridIVoxels(simResponse,1)
j<-env_returnSoluteGridJVoxels(simResponse,1)
k<-env_returnSoluteGridKVoxels(simResponse,1)

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

# Get the global production rates at this timepoint
gpr<-env_returnGlobalProductionRates(simResponse)

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

# Get the solute grid information for a particular solute, such as glucose.
#In this example, glucose is the first solute
glucoseGrid<-env_returnSpecifiedSoluteData(simResponse, 1)


## End(Not run)
```

Technique 5: Example Methods for Processing Solute Information

Technique 5: Example Methods for Processing Solute Information

---

Description

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 run Technique 1 in this case.

The following methods are available:
env_soluteProductionRateOverTime: Processes all env_State files for a particular simulation run, storing the global production rate of a particular solute at each timepoint. A data frame of these responses is returned. This information could then be plotted if desired.
plotContour: This routine draws a contour graph of the solute concentration field at a simultion timepoint.

Usage

```
env_soluteProductionRateOverTime(resultFileFolder, resultFileType, numTimepoints,
outputPeriod,soluteReqd)

plotContour(resultFileFolder, timepoint, soluteReqd, folderForGraphOut)
```

Arguments

`resultFileFolder`	The directory containing all the result files written during the simulation.
`resultFileType`	The output response file type to read in. This should be either agent_State, agent_Sum, env_State, or env_Sum
`numTimepoints`	The number of timepoints comprising this simulation run
`outputPeriod`	The number of iterations between simulation output. See protocol section of tutorial
`soluteReqd`	An integer value representing the solute that is of interest. This can be taken from the order in the output file.
`timepoint`	A particular timepoint from which the solute contour should be plotted
`folderForGraphOut`	Each graph is output to file (as a pdf). This should specify the folder where the graph should be written to

Examples

```
## Not run:
# DONTRUN IS SET SO THIS IS NOT EXECUTED WHEN PACKAGE IS COMPILED - BUT THIS
# HAS BEEN TESTED THOROUGHLY BEFORE UPLOADING TO THE REPOSITORY

# Track the production or consumption rate of glucose (first solute) over time,
# for 48 timepoints, where the output period was 1 (every step)
glucoseProduction<-
env_soluteProductionRateOverTime("/home/user/iDynoMiCS/results/",
"env_Sum", 48, 1,1)

# Plot the contour for glucose (1) at timepoint 40
plotContour("/home/user/Desktop/iDynoMiCS/results/single_species/", 40, 1,
```

```
"/home/user/Desktop/iDynoMiCS/results/single_species/graphs/")


## End(Not run)
```

Index