cytoskeleton-analyser

Release 1.6.6

Valerii Sukhorukov

Jun 19, 2021

CONTENTS:

Examines simulated cell microtubules

A postprocessor for extraction, analysis and human-friendly presentation of the raw binary results generated by Explicit-Microtubules.

Explicit-Microtubules is a C++ simulator of the whole-cell microtubule system. It reconstructs computationally the microtubule geometry and real-time dynamics in 3d space and time using stochastic algorithms. In the course of the simulation based on a multitude of bioogical information, it outputs detailed histories of microtubule dynamics along with three-dimensional snapshots of microtubules. However, designed to be focuced on versatile generation procedures, Explicit-Microtubules only includes basic data analysis.

Complementing the simulator, **cytoskeleton-analyser** is designed to extract, analyse and visualize various characteristics of the stochastically reconstructed microtubule system.

CHAPTER

ONE

INSTALLATION

1.1 Requirements

Cytoskeleton Analyser requires python3.9 or later.

When installing by pip, the following dependencies are installed *automatically*:

- NumPy
- SciPy
- Matplotlib
- meshio
- SQLAlchemy

1.2 Installation

Provided that python is present, in the simplest case the Cytoskeleton Analyser may be installed globallly using pip:

```
$ pip install cytoskeleton-analyser
```

However, the recommended way is installing the package into a virtual environment, which adds only a few commans:

1. Create a project directory (here named 'myproject') and cd to it.
2. Create a python virtual environment (here named 'venv3.9').
3. Activate the virtual environment.
4. Install as above.

```
$ mkdir myproject && cd myproject
$ python3 -m venv venv3.9
$ source venv3.9/bin/activate
$ pip install cytoskeleton-analyser
```

Alternatively, if you want to install the python package straight from the GitHub repository, replace the emphasized line above with

```
$ pip install git+git://github.com/vsukhor/cytoskeleton-analyser
```

1.3 The Source Code

Source code of Cytoskeleton Analyser is publicly available at GitHub. You can clone the whole repository with

```
$ git clone https://github.com/vsukhor/cytoskeleton-analyser.git
```

1.4 License

CHAPTER

TWO

GETTING STARTED

Simulation of the microtubule cytoskeleton with **Explicit Microtubules** produces output stored in text and binary files. The results carry information on spatial structure, composition and temporal evolution of the reconstructed system.

2.1 Organising configuration settings

It is often desirable to sample the parameter space by generating ensembles of cell reconstructions differing in one or many parameters of interest. Being able to automatically collect and categorise the applied configuration settings is often helpful as a starting point in examination of the microtubule bechavior.

The collected configurations may be stored in a database. Cytoskeleton Analyser uses *sqlite3* database. Everything necessary for hangling it is included in the package (For more convenience, additional open source GUI viewers, e.g. SQLiteStudio are also available for installation). The database itself is a regulary file on a disc and does not require a dedicated server.

Note: Use of the database is optional. Run settings are also serialised to a dictionary saved in a JSON file.

2.2 Features

Cytoskeleton Analyser can handle both the time-dependent and the geometric features of the reconstructed system of microtubules.

The software may be either configured to examine features of interest by specifying the specific names listed below or be set to process them all in batch mode. Depending on particular feature, Cytoskeleton-Analyser produces output saved on a disc as one or several files: whenever applicable, these are a JSON reporting the main summary, accompanied by the frequency distribution of the feature saved in CSV format and its (optianally gzipped) histogram as SVG image.

During the feature examination, the summary is also dumped to a log file.

2.2.1 Temporal

cytoskeleton-analyser examines time-dependent charactersitics of microtubule polymerisation kinetics using its 'history' subpackage.

One of the most prominent and biologically important aspects of microtubules is their dinamic instability, which induces ongoing switches between polymerisation and depolymerisation states referred to as recoveries and catastrophes respectively. Explicit-Microtubules simulator approximates the instability in real time and generates a history of microtubule states. Model of dymanic instability applied by Explicit-Microtubules utilizes extended dynamics whereby microtubule ends adopt stochastically one of three states: shrink, growth and pause. To represent the bechaviour of microtubules known from experiments in cell-specific environment (as opposed to in-vitro systems), probabilities of these states in the generated microtubules are position-dependent This accounts for the local modulation of the dynamic instability parameters by cell cortex and the plasma membrane.

Histories of all cell microtubules collected over a preset time interval are imported from Explicit-Microtubules output file 'history_eX_R.dat'. Letters 'X' and 'R' in the file name are placeholders indicating:

X : Microtubule end: *0* for minus-end or *1* for plus-end

R : Index of the simulation run: non-negative number

Experiments in living cells show that strucure and behavior of the cytoskeleton in the cell periphery differ strongly from those in the central regions for many cell types. Cytoskeleton Analyser accounts for cell compartmentation by examining peripheral and central cell regions separately, along with the analysis done for the whole-cell. The above discrimination is reflected in prefixes *soma*, *lamella* and *global* attached to output file names. Other components of the names is the feature description itself and index of the microtubule end. Below are the feature keywords.

Major temporal characteristics::

duration : Time of history recording.

length : Microtubule lengths.

comets : Radial distribution of growing plus-ends.

frequencies : Transition frequencies between the dynamic instability states.

spatial_maps : Spatial density maps of in cell xy projection for the transition types and their relations.

event_collections : Elongation, duration and velocity of microtubules in each of growth, shrink and pause states. Reorientation angle of the microtubule end over the states. Similar characteristics for two- and three-state sequencies where the consecutive states immediately follow each other.

2.2.2 Positional

cytoskeleton-analyser examines time-independent geometric features of the microtubule array using specific classes of its 'position' subpackage.

Explicit-Microtubules outputs spatially resolved data as a series of instantaneous cytoskeleton snapshots (see Note below), and these are used as input to Cytoskeleton-Analyser. Required files are 'positions_G_K_R' and 'csk_ages_G_K_R' containing 3d positions and time after polymerization of the nodes composing microtubules respectively. In the above names, the letters 'G', 'K' and 'R' are palceholders representing:

G : Granularity level: *fine* or *coarse*

K : Index of the snapshot: non-negative number

R : Index of the simulation run: non-negative number

While the imported replicas of the virtual cytoskeleton consists of a full 3d-resolved set of microtubules, experimental results in the literature are often derived from a flattened view based on optical microscopy images. To make the

recovered characteristics comparable with the empirical knowledge, one may choose to examine, in addition to the full representation, 2d projections of the reconstructed cytoskeleton filaments to xy plane as well as narrow slices cut parallel to xy plane, which approximate confocal or Total Internal Reflection Microscopy. Whole-cell 3d plot of the microtubules superimposed onto the plasma membrane may be also saved as an SVG image. Below are the class names of geometry-related features, which also serve as the key words used for configuring the cytoskeleton-analyser operation.

Characteristics relevant for both full-depth and sliced representations:

Lengths3d : Lengths of microtubules in 3d space.

Lengths2d : Lengths of filament projections to a xy plane.

RadialMass : Radial distribution of mass in filament projections to xy plane from cell geometrical center to periphery.

RadialEnds : Radial distribution of plus-ends in filament projections to xy plane from cell geometrical center to periphery.

Curvature3d : Curvature of microtubules in 3d space.

AnglesToRad : Angles between filament xy-projections and radial direction outwards in the cell marginal zone.

SegmentNumbers : Apparent total number of filaments in the system.

Characteristics relevant for the sliced representation only:

Curvature2dConv : Apparent curvature of filament xy-projections.

Curvature2dMboc17 : Apparent xy-curvature using a simplified approach developed for expeimental analysis of optical microscopy results[1].

Characteristics relevant for the full-depth representation only:

AgesByNode : Age distribution of all filament nodes (including internal ones).

AgesByFilament : Age distribution of filament '+'-end nodes.

Note: Stochastic models like that implemented within Explicit-Microtubules represent fluctuating environments. As a rule, one either collects several snapshots of the system separated by time an interval large enough for for temporal autocorrelations in the parameters of interest to decay or performs several independently seeded runs.

For the specific case of reconstructed microtubules, also single snapshot may be sufficient for the particular case when the simulation was configured with inter-microtubule interactions switched off. Then, the cytoskeleton filaments grow independent of each other and their whole-cell ensemble samples the geometric characteristics in a satisfactory way.

---

[1] Zhang Zh., Nishimura Y., and Kanchanawonga P. (2017) Extracting microtubule networks from superresolution single-molecule localization microscopy data MBoC, 28:2.

CHAPTER

THREE

EXAMPLES

Below are simple templeates cases of simulation postprocessing in a typical application.

Before proceeding to the proper analysis, feature-independent settings need to be sspecified.

- External input and output paths for the data. For the examples, data input files can be downloaded from data/in directory of the GitHub repository.
- Index of the simulation run.
- The cell type. It has three attributes packed inside a class: *typename* and *plmind* are used in constructing cell-specific path to input files and correspond to internal directory structure the Explicit Microtubules. *regions* cell field defines borders of cell internal subcompartments.

Below, we will process the same simulation, so it is convenient to store the settings in a separate python module for importing them into the process-specific scripts.

Listing 1: sim_specs.py

```
from pathlib import PurePath                    # OS-independedt paths
from numpy import Inf
from cytoskeleton_analyser.cells import CellType
from cytoskeleton_analyser.cells import Region, Regions

# Set data source and destination. !! CHANGE !!
data_in = PurePath('/Full/path/to/input/directory')
data_out = PurePath('Full/path/to/output/directory')

# Specify the simulation runs to analyse:
rinds = [1]

# The runs represent the same cell type.
# Derive the cell class from CellType to ensure
# that no attributes are missing.
cell = CellType(
   # Name of the data foder for this cell type.
   typename = 'irreg',
   # Index of the membrane mesh.
   plmind = 1,
   # Radial distance limits (in m) for the cell subcompartments.
   regions = Regions(
      cytosol=Region(0., Inf),
      soma=Region(0., 6.),
      lamella=Region(6., Inf),
```

(continues on next page)

(continued from previous page)

```
        edge=Region(10., Inf),
    )
)
```

Now let us see the use cases:

3.1 Example 1: Organizing Simulation Parameters

Two versions of configuration processing are shown. The first one merely collects the simulation parameters into a dictionary. The Second version stores them additionally in a database.

3.1.1 Without the use of a database:

```
from cytoskeleton_analyser.inout import Paths     # simulation-specific paths
from cytoskeleton_analyser.config.drivers import process  # driver fuction
from sim_specs import cell, data_in, data_out, rinds

# Decide if this is a new analysis, or we merely want to import one.
new = True

if __name__ == '__main__':

    for ci in rinds:
        paths = Paths(data_in, data_out, cell, ci)
        process(new, paths, ci, cell)
```

3.1.2 Accumulate the settings in a database:

Similar to the above, but the database module is included (line 3). The data are collected in 'cfs' (lines 11,12) list before sending to the database (line 13).

```
from cytoskeleton_analyser.inout import Paths     # simulation-specific paths
from cytoskeleton_analyser.config.drivers import process  # driver fuction
import cytoskeleton_analyser.database.sqlite_alchemy_orm.db as db  # database
from sim_specs import cell, data_in, data_out, rinds

# Decide if this is a new analysis, or we merely want to import one.
new = True

if __name__ == '__main__':

    cfs = [process(new, Paths(data_in, data_out, cell, ci), ci, cell)
           for ci in rinds]
    db.update(cfs, cell, Paths(data_in, data_out, cell).data_out.parent)
```

3.2 Example 2: Geometry-related features

The code snippet below shows a simple scenario for geometry analysis of the reconstructed cytoskelton.

```
from cytoskeleton_analyser.inout import Paths      # simulation-specific paths
from cytoskeleton_analyser.position.drivers import process  # driver fuction
from sim_specs import cell, data_in, data_out, rinds

# Set general parameters of the input simulation. (a shortlist
# of config settings).
params = {
   'edge_len_fine': 0.008,    # (m) length of filament edge.
   'iscoarse': True,          # use coarse-grained representation
   'coarsegraining': 25,      # coarse-graining factor
   'use_final': False,        # if True, limit to final snapshot only
   'cell': cell,              # cell object
   'slice_limits': {'bottom': 0., 'top': 0.8},  # z-pos. of slicing planes
}

# Specify features to process. If this is None,
# all features are included.
features = [
   'RadialMass',
]

# Decide if this is a new analysis, or we merely want to import one.
new = True

# This dict will become populated only if analysis
# is imported ('new' = False).
# Otherwise, the results are stored on a disc.
result = {}

if __name__ == '__main__':

   for ci in rinds:
      paths = Paths(data_in, data_out, params['cell'], ci)
      result[str(ci)] = process(new, paths, ci, params, True, features)
```

3.3 Example 3: Examination of microtubule dynamic instability

The code snippet below shows a scenario for analysis of the microtubule dynamics where only *duration*, *spatial_maps* and *event_collections* features are selected.

```
from cytoskeleton_analyser.inout import Paths      # simulation-specific paths
from cytoskeleton_analyser.history.drivers import process  # driver fuction
from sim_specs import cell, data_in, data_out, rinds

params = {
   'edge_len': 0.008,     # (m) length of filament edge.
   'cell': cell,          # cell object
```

(continues on next page)

(continued from previous page)

```
}

# Specify features to process. If this is None,
# all features are included.
features = [
   'duration',
   'spatial_maps',
   'event_collections',
]

if __name__ == '__main__':

   for ci in rinds:
      paths = Paths(data_in, data_out, params['cell'], ci)
      process(paths, ci, 1, params, True, features)
```

CHAPTER

FOUR

APPLICATION PROGRAMMING INTERFACE

cytoskeleton_analyser python package API.

4.1 Subpackages

4.1.1 config package

config module

drivers module

4.1.2 database package

sqlite_alchemy_orm package

containers package

cell_elements module

Subclasses of container base class for tailored to cell components.

class cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.cell_elements.**Centrosome**

Bases: *cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.Base*

Container class for configuration of centrosome-type MTOC.

model

alias of *cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_mtoc.ConfigMtocCentrosome*

class cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.cell_elements.**Golgi**

Bases: *cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.Base*

Container class for configuration of Golgi-type MTOC.

model

alias of *cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_mtoc.ConfigMtocGolgi*

class cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.cell_elements.**InSpace**
Bases: *cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.Base*

Container class for configuration of unanchored microtubule MTOC.

model
alias of *cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_mtoc.ConfigMtocInSpace*

class cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.cell_elements.**MembraneNucleus**
Bases: *cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.Optional*

Container class for configuration of cell nuclear membrane.

is_used: bool

model
alias of *cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_nucleus.ConfigNucleus*

class cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.cell_elements.**MembranePlasma**
Bases: *cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.Optional*

Container class for configuration of cell plasma membrane.

is_used: bool

model
alias of *cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_plasma.ConfigPlasma*

class cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.cell_elements.**Mtoc**(*InSpace*, *Golgi*, *Centrosome*, *Nucleus*)
Bases: tuple

Types of Microtubule Organizing Centers (MTOCs).

property Centrosome
Alias for field number 2

property Golgi
Alias for field number 1

property InSpace
Alias for field number 0

property Nucleus
Alias for field number 3

class cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.cell_elements.**Nucleus**
Bases: *cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.Base*

Container class for configuration of Nucleus-type MTOC.

model
alias of *cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_mtoc.ConfigMtocNucleus*

config module

container module

Base classes for setting containers.

The containers are responsible for interaction with the database models.

class cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.**Base**
Bases: object

Base class for implementing configuration setting containers.

It should be subclassed to implement cell-component specific interactors with the database tables.

classmethod add_to_db(*d: dict*, *session: sqlalchemy.orm.session.Session*) → *cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.Base*
Append dict items of dictionary d to the database.

classmethod assign(*d: dict*, *row:* cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.Base)

classmethod columns(*d: dict*)
Database columns corresponding to keys of dictionary d.

classmethod existing_db_rows(*d*, *session: sqlalchemy.orm.session.Session*) → list
SQL query to retrieve rows corresponding to dictionary d.

model: Any

class cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.**Optional**
Bases: *cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.Base*

Intermediate implementation for setting containers.

Tailored to optional settings.

classmethod add_to_db(*d: dict*, *session: sqlalchemy.orm.session.Session*) → *cytoskeleton_analyser.database.sqlite_alchemy_orm.containers.container.Base*
Append dict items of dictionary d to the database.

classmethod columns(*d: dict*)
Database columns corresponding to keys of dictionary d.

is_used: bool

models package

config module

SqlAlchemy database model of full configuration parameter set.

class cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config.**Config**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base

SqlAlchemy database model of full configuration parameter set.

cap_off

cap_on

cas_catastr_activation_extent

cas_catastr_activation_intensity

cas_catastr_inhibition_extent

cas_catastr_inhibition_intensity

cas_growth_slowdown_extent

cas_growth_slowdown_intensity

cas_orientation_extent

cas_orientation_intensity

cas_rescue_activation_extent

cas_rescue_activation_intensity

cas_rescue_inhibition_extent

cas_rescue_inhibition_intensity

cas_thickness

cut

cytosol_conc_coef

cytosol_tubulin_total

cytosol_volume

filament_persist_length

filament_step

id

mtoc_centrosome

mtoc_centrosome_id

mtoc_golgi

mtoc_golgi_id

mtoc_inspace

mtoc_inspace_id

mtoc_nucleus

mtoc_nucleus_id

nuc_membr_interact_thresh

nuc_membr_orient_strength

nucleus

nucleus_id

num_filaments

pl_membr_interact_thresh

pl_membr_orient_strength

plasma

plasma_id

proximity_omit_own_nodes

proximity_step

proximity_threshold

release

run_inds

state_change_gs

state_change_ps

state_change_sg

time_total

velocity_sh0

velocity_sh1

config_mtoc module

SqlAlchemy database models of MTOC configuration parameters.

Configuration parameters for Microtubule Organizing Centers (MTOCs) are arranged in type-specific database tables.

class cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_mtoc.**ConfigMtocCentrosome**(***kwargs*

Bases: sqlalchemy.orm.decl_api.Base

Database model of MTOC settings for centrosomal microtubules.

id

main_configs

minus_end_cargo_free

origin_0

origin_1

origin_2

size_0

size_1

size_2

total_frac

class cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_mtoc.**ConfigMtocGolgi**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

Database model of MTOC settings for Golgi-anchored microtubules.

angular_spread

id

main_configs

minus_end_cargo_free

origin_0

origin_1

origin_2

polar_bias

size_0

size_1

size_2

total_frac

class cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_mtoc.**ConfigMtocInSpace**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

Database model of microtubules having free minus-end.

id

main_configs

minus_end_cargo_free

par_int

par_real

total_frac

use_pbs

class cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_mtoc.**ConfigMtocNucleus**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

Database model of MTOC settings for Nucleus-anchored microtubules.

angular_spread

id

main_configs

minus_end_cargo_free

polar_bias

total_frac

config_nucleus module

SqlAlchemy database model of parameters for cell nucleus.

class cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_nucleus.**ConfigNucleus**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

SqlAlchemy database model of parameters for cell nucleus.

id

main_config

orientation_0

orientation_1

orientation_2

origin_0

origin_1

origin_2

size_0

size_1

size_2

config_plasma module

SqlAlchemy database model of parameters for cell plasma membrane.

class cytoskeleton_analyser.database.sqlite_alchemy_orm.models.config_plasma.**ConfigPlasma**(***kwargs*)
Bases: sqlalchemy.orm.decl_api.Base

SqlAlchemy database model of parameters for cell plasma membrane.

id

main_config

run

type

Module contents

db module

4.1.3 fitting package

base module

Exposes base class for subclassing in specific data models.

class cytoskeleton_analyser.fitting.base.**Base**(*x: Sequence*, *p0: Sequence[float]*, *x_units: Optional[str] = None*)

Bases: object

Base class for subclassing in specific data models.

class **Summary**(*name*, *p*, *chi2*, *mean*, *saturation*)

Bases: tuple

property **chi2**
Alias for field number 2

property **mean**
Alias for field number 3

property **name**
Alias for field number 0

property **p**
Alias for field number 1

property **saturation**
Alias for field number 4

aik
AIK indicator.

bounds
Bounds on the model parameters.

cc
Components of the model prediction.

chi2
Chi square.

cov
Covariamce matrix of the fitted parameters.

equilibrium_ind
Data index at equilibration.

fano
Fano factor

logger: logging.Logger = None

mean
Model mean.

name: Optional[str]
Model name.

p: numpy.ndarray
Optilized values of model parameters.

p0: numpy.ndarray
Initial values of model parameters.

par
value}

Type Model parameters as a dictionary {'name'

predict(*f: Callable*, *sl: slice*) → numpy.ndarray
Model prediction.

Parameters

- **f** – Model function.
- **sl** – Slice selecting model-specific parameters.

prediction
Model prediction.

report()
Dump a report summarizing the model to the logger.

residnorm
Norm of residuals.

saturation
Predicted saturation value.

set_equilibration(*y: numpy.ndarray*) → None
Determines predicted time to equilibration if such exists.

Is only applicable if the model achieves saturation. Assumes that equilibration point is reached whenever monotonously decreasing difference between saturation value and the model prediction becomes less than standard deviation. Sets `equilibrium_ind`: data array index at which equilibration is assumed to be achieved.

Parameters **y** – Fitted data array.

set_quality_indicators(*y: numpy.ndarray*) → None
Calculate quality indicators of the model after minimization.

Parameters **y** – Fitted data array.

summary() → dict
Create a dictionary sumarizing the model.

var
Model variance.

x: numpy.ndarray
Data x-values.

x_units: Optional[str]
Units for data x-values.

cytoskeleton_analyser.fitting.base.**set_logger**(*logger*)

const module

exponential module

gamma module

lognorm module

normal module

rayleigh module

4.1.4 history package

4.1.5 position package

spatial_systems module

4.2 cells module

4.3 histograms module

4.4 inout module

4.5 plasma_membrane module

4.6 report module

4.7 summary module

4.8 visualization module

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

E

F

G

I

L

M

N

O

P

R

S

T

U

V

X