MMODES

Aug 22, 2020

Contents:

MMODES is a package built on top of COBRApy and SciPy to allow fast set up of dynamic simulations with GEM models (dFBA), isolated or in a consortium.

CHAPTER 1

Installing MMODES

A Python version >= 3.6 is required in order to install MMODES (install Python 3). To avoid messing up with versions, working on a virtualenv might be a good idea.

1.1 PIP Install

It's recommended to install MMODES via pip. This way, the latest stable version of the package is guaranteed. Naturally, in order to install MMODES through pip, pip tool is required (if it isn't installed, check the pip's installation instructions). Usually, the *–user* flag is required. On a bash shell:

```
pip3 install mmodes --user
```

1.2 Build from SOURCE

Otherwise, you can build from the GitHub repository. COBRApy version used is 15.2. Scipy, numpy, matplotlib and dill are also required. A cobra version >= 14 should also work, albeit not being guaranteed.

```
git clone https://github.com/carrascomj/mmodes.git # or ssh
cd path_to_mmodes/mmodes
sudo python3 setup.py install
```

1.3 DOCKER Install

A docker image is currently under development.

1.4 Uninstall

Uninstalling can be accomplished via *pip*:

```
pip3 uninstall mmodes --user # if user install
sudo pip3 uninstall mmodes # if superuser install (from source)
```

CHAPTER 2

Getting started

An example script is provided on the GitHub repository and will be here described here.

2.1 First steps: the Consortium object

You need a GEM model to run a consortium dynamic simulation. BiGG models is a good place to start. Conversely, a model example of *Bifidobacterium adolescentis* of the AGORA database is provided as example on the ModelsInput.

First, we instantiate a Consortium object, that will contain all the required parameters for the simulation.

```
from mmodes import Consortium, dMetabolite
cons = Consortium(mets_to_plot = ["thr_L[e]", "ac[e]"])
```

mets_to_plot parameter is supplied to later plot these metabolites. Now, we add the model to the Consortium object.

```
path_to_model = 'mmodes/ModelsInput/BGN4_eu.xml' # provided example GEM
# Additionally, we'll instantiate some metabolite
# to assign kinetic parameters
# for instance, https://www.ncbi.nlm.nih.gov/pubmed/18791026?dopt=Abstract
glc = dMetabolite(id = "glc_D[e]", Km = 14.8, Vmax = 0.13)
cons.add_model(path_to_model, 0.0003, dMets = {glc.id: glc})
```

More information of Consortium class can be found on *The Consortium Object*.

2.2 Instantiating the medium

The last step previous to running the simulation is assigning a medium to the Consortium. In the example, medium is read from a JSON file (which is a dictionary).

Here, we will instantiate a medium using all the extracellular metabolites of the model(s) added in the Consortium.

```
abs_media = {k: 1000 for k in cons.media}
cons.media = cons.set_media(abs_media)
print(cons)
```

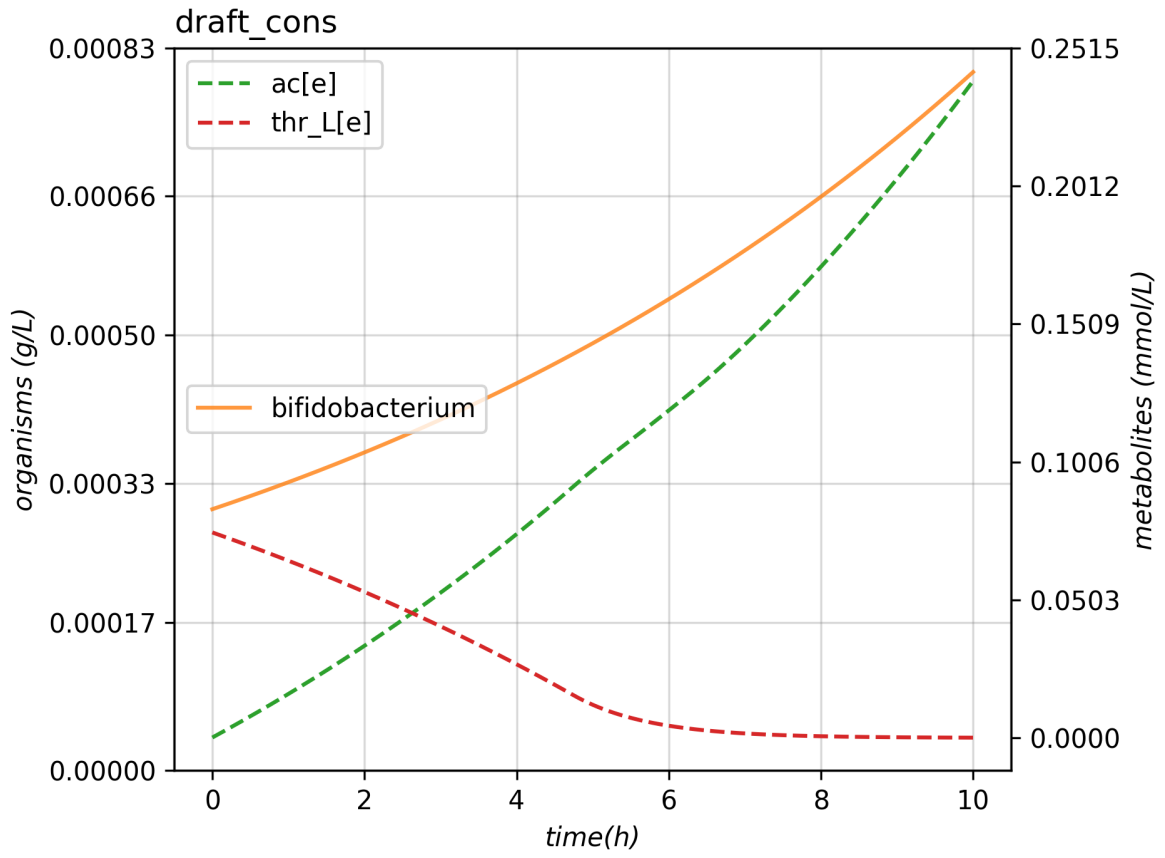
Make sure that all metabolites in the medium are named the same as in the added GEM models. MMODES supports working with metabolite identifiers or names, default is id. If some metabolites are misspelled, they won't be added. Conversely, metabolites that are in the extracellular compartment of the GEM members of the Consortium, that were not added to the medium, will be set to 0.

2.3 Run the simulation!

The last step is running the simulation.

```
cons.run(maxT = 10, outp = "plot_example.png", outf = "tab_example.tsv", verbose=True)
# print information pieces on screen
for mod in cons.models:
    print(mod, cons.models[mod].volume.q, sep = " -> ")
print("Glucose", str(cons.media["glc_D[e]"]), sep = " -> ")
print("Acetate", str(cons.media["ac[e]"]), sep = " -> ", end = "\n\n")
```

As demonstrated, MMODES allows running a dynamic (p)FBA simulation with a few lines of code. A TSV file and a plot should've been generated on the working directory.



The next steps into MMODES should be taking a glimpse at *The Consortium Object*.

Also, *The Experiment Object* demonstrates how the Consortium class can be extended to ease even more the configuration of microbial community simulations.

CHAPTER 3

The Consortium Object

The Consortium Class is the central axis of the MMODES package. It carries all the required parameters to run a dynamic simulation of a microbial community.

3.1 Most common parameters

class Consortium(*max_growth = 10*, *v = 1*, *stcut = 1e-8*, *title = "draft_cons"*, *mets_to_plot = []*, *work_based_on = "id"*, *manifest = ""*, *comets_output = False*)

Parameters

- **max_growth** (*if*) – is the maximum biomass that a GEM model is allowed to reach (*default = 10*)
- **v** (*float*) – is the volume of the modeled space. Amounts will be transformed to concentrations using this parameters (*default = 1*). Units are arbitrary, but L are used by convention.
- **stcut** (*float*) – is the limit of biomass flux (growth increment) where the simulation is considered to have reached a stable state and stops. Turn to a negative number to keep the simulation running (*default = 1e-8*).
- **title** (*str*) – of the generated plot of the simulation
- **mets_to_plot** (*list*) – are the metabolites to be later plotted.
- **work_based_on** (*str*) – *(="id" | "name")* is a REALLY important parameter. It indicates whether extracellular metabolite names or ids should be used to communicate models and understand the medium. One should use the attribute (id or name) that is consistent among all the GEM models (just consistency on the extracellular metabolites is required) (*default = "id"*)
- **manifest** (*str*) – if a non-empty string is provided, it will output a fluxes TSV file to this path (default="").
- **comets_output** (*bool*) – whether output (including fluxes) should be written in COMETS-like format.

COMETS-like output can be used to visualize reaction fluxes in VisAnt.

3.2 Setting models

dModel objects are containers of COBRA model objects, with some features to compute the multi-strain simulation. The method to add a GEM model to the *Consortium* is the following:

Consortium.**add_model**(*mod_path*, *volume_0*, *solver = "glpk"*, *method = "pfba"*, *dMets = {}*, *limit = False*)

Parameters

- **mod_path** (*string*) – path to the model in SBML, MAT or JSON format.
- **volume_0** (*float*) – initial concentration (usually, g/L) of biomass.
- **dMets** (*dictionary*) – of metabolite.id : dMetabolites.
- **limit** (*maximum*) – biomass values that is allowed for this particular dModel (*default = False*, no limitation).

More information about the *limit* parameter and the dModel in general can be found on *The dModel Object*.

3.3 Setting media

Consortium.media is a simple dictionary of metabolite ids/names : concentrations. It can be passed as a simple dictionary. Also, a handy method is provided to read from a JSON file object (with the structure of a dictionary).

Consortium.**set_media**(*media*, *concentration = False*)
Adds **media** as the Consortium medium object. If concentration is True, values of the dictionary will be converted to concentrations (using the volume parameter).

Consortium.**media_from_json**(*jfile*, *concentration = False*)
Uploads medium from a **jfile** path. This path corresponds to a JSON file which contains a dictionary.

Note: Concentration units are arbitrary, although the convention dictates *mmol/L* for metabolites and *g/L* for biomass. Take into account consistency among units when instantiating the medium. Also, time is assumed to be in hours.

3.4 Running the Community simulation

Consortium.**run**(*maxT=10*, *integrator='vode'*, *stepChoiceLevel=(0., 0.5, 1000.)*, *verbose = False*, *outf = "plot.tsv"*, *outp = "plot.png"*, *plot = True*, *actualize_every = float('-inf')*)
Starts the community simulation, solving the system of ODE's.

Parameters

- **maxT** (*float*) – in time simulation units (hours), simulation will stop when it reaches this parameter (*default = '10'*).
- **integrator** (*str*) – *('vode' 'dopri5' 'fea' 'rk4')* type of ODE integrator (*default = 'vode'*).
- **stepChoiceLevel** (*str*) – (0, max time step, max number of time-steps) for *vode* and (time-step, 0, max number of time-steps) for the rest of integrators (*default = 0., 0.5, 100*).

- **verbose** (*bool*) – a verbose simulation will show a progress bar and the reason of exiting the simulation (*default = False*).
- **outf** (*str*) – path where the output will be generated (*default= plot.tsv*).
- **outp** (*str*) – path where the plot will be generated (*default= "plot.png"*).
- **plot** (*bool*) – whether to generate the plot (*default= True*).

Assigning a *maxT* parameter doesn't guarantee to reach that time, since simulation could be terminated when it reaches the maximum number of steps (in *stepChoiceLevel*), when it's stabilized (*stcut* in the *Consortium*) or when some simulation exceeds that maximum growth (*max_growth* in the *Consortium*, different from *limit* in *The dModel Object*).

Once the simulation is finished, the output could be later generated:

```
from mmodes.vis import plot_comm
plot_comm(cons) # cons is a Consortium object which has already run
```

On this point, other metabolites could've been plotted changing *mets_to_plot* attribute of *Consortium*.

Warning: Please, take into account that the results will be *appended* to **outf** and the plot will be generated from this path. Thus, make sure a file with this name doesn't exist before a simulation is started.

3.5 Adding perturbations

Metabolites and perturbations are added with the following method:

Consortium.**add_mets**(*pert*, *concentration = False*)

Parameters

- **pert** (*dict*) – same format as media. Additionally, keys corresponding to model ID's can be used to add biomass to a model.
- **concentration** (*bool*) – whether amounts in pert should be transformed to concentration units.

Once the method has been called, the same Consortium can run again, simulating a perturbation.

Putting it all together:

```
from mmodes import Consortium
from mmodes.vis import plot_comm

cons = Consortium()
cons.add_models(mod_path = "path_to_some_model_file.mat", volume_0 = 0.001)
cons.add_models(mod_path = "path_to_some_other_model_file.xml", volume_0 = 0.0012)
cons.media = cons.media_from_json(jfile = 'some_dict_file.json')
cons.run(plot = False)
cons.add_mets({'glc[e] : 0.02'})
```

(continues on next page)

(continued from previous page)

```
cons.run(plot = False)
plot_comm(cons)
```

CHAPTER 4

The dModel Object

The dModel object expands the COBRApy Model object with some features. It acts as a container, not as a subclass of it.

4.1 Adding models to the Consortium

Usually, models are added with Consortium methods. Please refer to *The Consortium Object* to see how. It's important to note that when a dModel is added, it will be optimized to check if it's operative. The model.id cobra attribute will be used as dModel.id.

Furthermore, the growth increment of the dModel will be taken from de biomass function. It doesn't need to be the objective funtion that will be optimized during the simulation, MMODES will take the first reaction that starts with "biomass". This allows some functionalities but can be troublesome when more than one biomass reaction is present in the model.

Once added to the Consortium, dModels are accesible from the attribute models. This attribute is a dictionary of dModel.id : dModel object.

4.2 Limiting growth of the model

At the moment of adding the model, **limit** is a parameter that can be tuned:

- if *False* (default), no limitations will be applied;
- if *True*, model won't grow, but the solution of fluxes will be used to update the medium and added to the output files;
- if a *numeric* value is passed, the model will be allowed to grow until this amount is reached, then it will behave as if *limit* = True.

4.3 Death rate

death_rate is an attribute that can be added once the dModel is instantiated (or added to the Consortium). This parameter is **incompatible** with *limit*.

CHAPTER 5

The Experiment Object

The Experiment class provides a handy wrapper around the Consortium object with two key objectives:

1. Store the state and perturbations occurred in a Consortium.
2. Run sequences of dFBA + perturbations directly from configuration files.
3. Automatic filtering of output files.

Using the package, we realized that a common operation was to initialize models from a directory with random initial biomasses, instantiate medium and run a loop that adds a perturbation and runs the simulation. That's exactly the objective of this class.

5.1 Instantiating the Experiment

Experiment is a subclass of Consortium and follows a similar initialization, adjusted to read directly from files at the time of instantiating:

class Experiment (*medium_path = "", models_dir = "", rand_biomasses = [0.0001,0.0005], perturbations = [], ..., lp = "fba", solver = "glpk"*)

Parameters

- **medium_path** (*string*) – path to a JSON file containing medium and perturbations
- **models_dir** (*string*) – path to the directory where the models are read from
- **rand_biomasses** (*list*) – of two floats, upper and lower constraints to randomized initial biomasses (*default = [0.0001,0.0005]*)
- **perturbations** (*list*) – of names of the perturbations (*default = list of generic names*)
- **lp** (*string*) – establishing the type of LP problem (*default = "fba"*) to solve by the solver (*default = "glpk"*)
- **..** – Rest of parameters of a *Consortium*.

The JSON file where *medium_path* points must have the structure of a list of dictionaries.

```
[
  {
    Metabolite_of_media1 : amount,
    Metabolite_of_media2 : amount
    ...
  },
  {
    Metabolite_of_pertubation1 : amount,
    Metabolite_of_pertubation2 : amount
    ...
  },
  {
    Metabolite_another_pertubation1 : amount,
    Metabolite_another_pertubation2 : amount
  },
  ...
]
```

The *perturbations* parameters must have the same length of this JSON list. Either way, a warning will be printed and, if needed, it will be filled with generic names. On the other hand, the models on *models_dir* directory should have an *xml*, *mat* or *json* extension. If a file in this directory isn't a model, a warning will be printed.

An example can be performed from the MMODES root directory in the GitHub repository.

```
from mmodes import Experiment

exp = Experiment(medium_path = "ModelsInput/media.json", models_dir = "ModelsInput",
                 perturbations = ['START', 'THR+TYR'],  mets_to_plot = ["ac[e]", "thr_
↪L[e]"],
                 title = "B. adolescentis with 1 perturbation")
```

Note: Don't get frightened by the walls of text in the screen. Each file is tried to be read by COBRAPy with different parsers and libSBML is really verbose right now. Sadly, it can't be filtered by now.

5.2 Running an Experiment

`Experiment.`**`run_experiment`**(*intervl = 10*, *integrator = 'vode'*, *stepChoiceLevel = ()*, *verbose = False*, *outp = "models_dir.png"*, *filter = False*, *equif = True*, *inplace_filter = False*, *plot = True*, *actualize_every = float(-inf)*)

Starts a loop of community simulations + perturbations

Parameters

- **intervl** (*float*) – time in simulation units (hours) between perturbations (*default = '10'*).
- **integrator** (*str*) – *('vode' 'dopri5' 'fea' 'rk4')* type of ODE integrator (*default = 'vode'*).
- **stepChoiceLevel** (*str*) – (0, max time step, max number of time-steps) for *vode* and (time-step, 0, max number of time-steps) for the rest of integrators (*default = 0., 0.5, 100*).

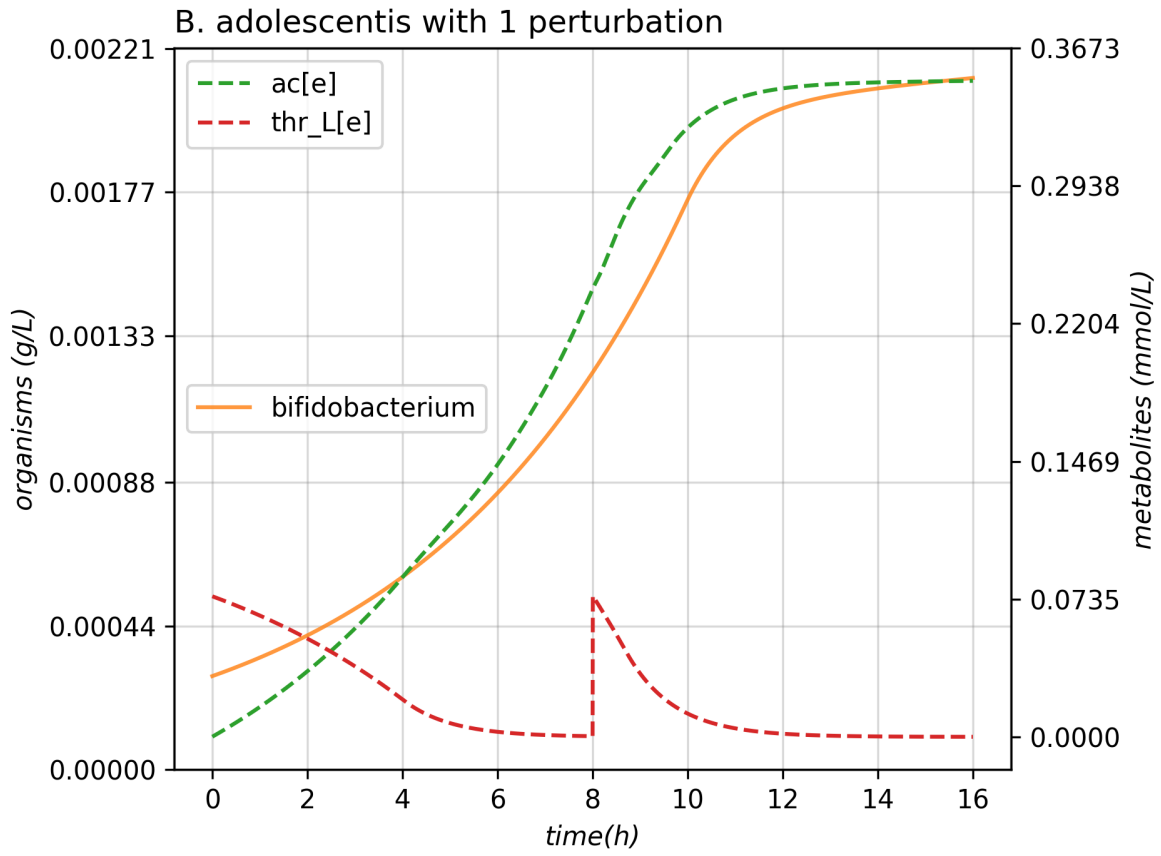
- **verbose** (*bool*) – a verbose simulation will show a progress bar, the reason of exiting the simulation and some messages of perturbations (*default = False*).
- **out** (*string*) – path where the output will be generated (*default= plot.tsv*).
- **outp** (*string*) – path where the plot will be generated (*default= "Some_Experiment.png"*).
- **plot** (*bool*) – whether to generate the plot (*default= True*).
- **actualize_every** (*float*) – time interval of writing to output files (*default = -inf*, write always)

The rest of parameters are discussed in the next section.

Following the above example:

```
#16 h of simulation
exp.run_experiment(intervl = 8, integrator = 'fea', stepChoiceLevel = (0.005,0.5,
↪10000))
```

It should have generated a tsv called *plot.tsv* and an image called *Some_Experiment.png* like the one presented:



5.3 Filtering the output

The 3rd objective of this class was to filter the output. But how exactly is filetered? Briefly, the points right before each Perturbation in Medium output and fluxes are kept. In addition, the fluxes can be filtered by keeping 100 equidistant points, generating other file. We found that those kind of files were really useful for some applications.

The parameters that control the filtering are in the `run_experiment()`.

param bool filter whether output should be filtered (*default = False*)

param bool equif whether the equidistant flux output should be generated. It only works when *filter* is True (*default = True*).

param bool inplace_filter whether the original output should be overwritten by the filtered one (*default = False*)

Note: The filtering functions currently support datatable. If datatable, which is not a requiring for installing MMODES, isn't available, pandas will be used. With large outputs, using pandas might be quite slow.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index