Optimizer for balanced growth models coded in SBML

—

By Anmol Reddy and Shiva Tarun, IIT Bombay
Supervised by Peter Schubert, HHU Düsseldorf.

Context

- Balanced growth describes growth of a unicellular organism, at a fixed rate.
- In “On balanced-growth states of autosynthetic cell models” (unpublished), Deniz Sezer defined three balanced growth models TR, TUR and TVR and derived formulae to derive optimum growth states and growth rates.
- These three models have been coded as standardized SBML models.
- We try to solve these in Python3, choosing it because of the extensive libraries available both for SBML and math and also its extensive popularity. We use the formulae from Deniz’s paper.

Motivation: Need to find metabolite concentrations for optimum cell growth, by extracting information directly from SBML models. This must also be able to work on large, complex models with many reactions.

What we have done:

For models that satisfy certain conditions, we have made a python package that can solve these models and optimize globally or under certain constraints on protein concentrations.

Assumptions

We are able to do this under some assumptions and constraints about the model so that it corresponds to the math in Deniz's paper.

- The splitting of cells after reaching a certain size has been modelled into another class of reactions taken as dilution reactions which operate continuously (balanced growth).
- We assume one of the enzymes is a ribosome and the others are metabolic proteins.
- Protein synthesis happens from metabolites which in turn are supplied from outside.

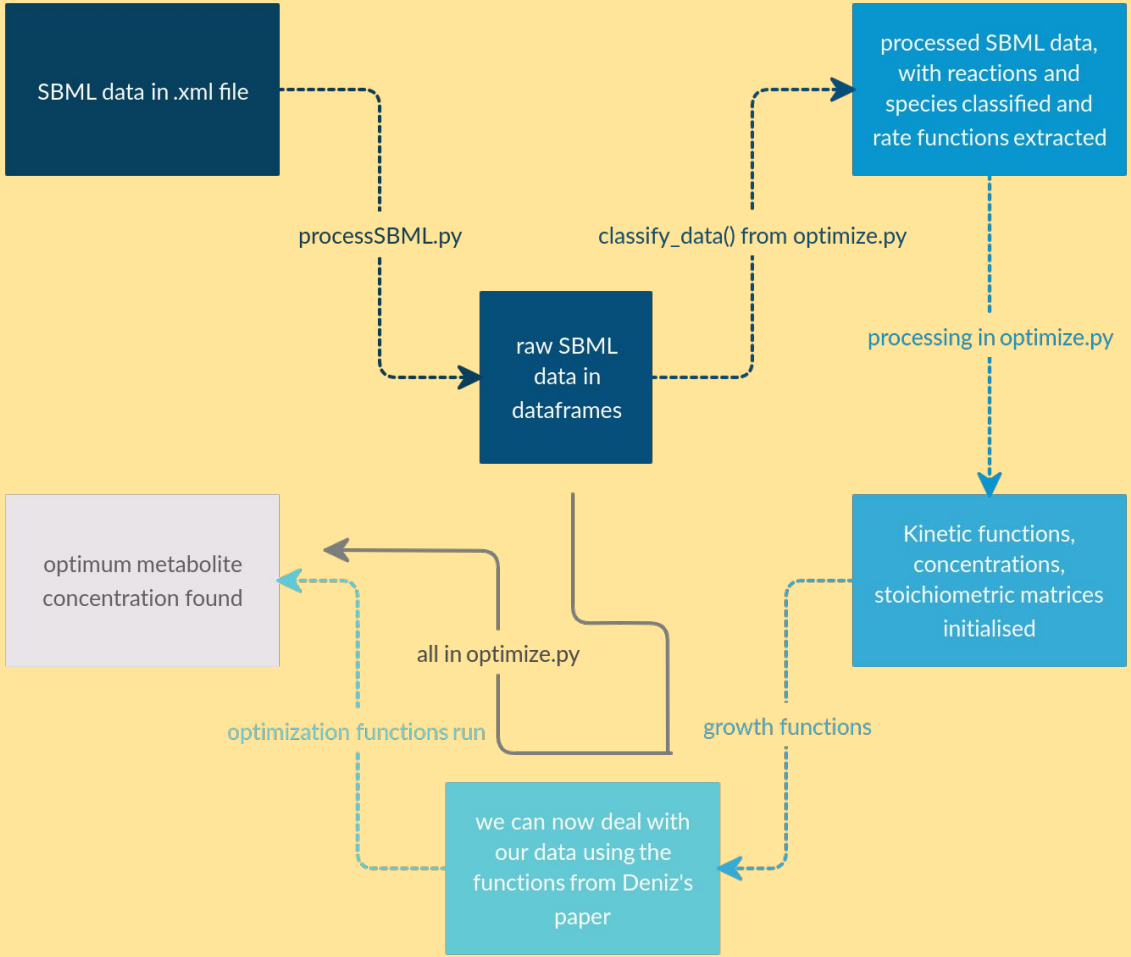
Technical requirements and capabilities

1. The package is `cell_growth_optimizer`, which runs on `python3` and uses basic `python3` math libraries like `numpy` and `pandas` and `libsbml`. SBML versions L3v1 and L2v4 are used.

2. Models must contain rate laws, initial amounts, the SBO term ID's of all species, stoichiometries of the reactions etc. all in a .xml file.

3. The default capability of the program calculates the metabolite concentration `c_s` and corresponding protein to ribosome fraction, `phi_m` that correspond to max growth in "optimum-growth" method.

4. We can also calculate the metabolite concentrations from a given metabolic state defined by the protein concentrations (`c_e`) in the method "optimize-for-protein-conc" or protein ratios (`phi_m`) by the method "optimize-for-protein-ratios".


SBML data in .xml file
processSBML.py
raw SBML data in dataframes
classify_data() from optimize.py
processed SBML data, with reactions and species classified and rate functions extracted
processing in optimize.py
Kinetic functions, concentrations, stoichiometric matrices initialised
growth functions
all in optimize.py
optimum metabolite concentration found
optimization functions run
we can now deal with our data using the functions from Deniz's paper

We have a code to do time course analysis in python (and as a part of this, extract info from SBML) and one to optimize when this is hard-coded in python. We build a pipeline between these and so use a part of this time course analysis code and a part of this optimization code.

We take the relevant information from the first part, store it in a more optimized way and in a form which we can then deal with using Deniz's growth functions.

We get our rate law derivatives by numeric approximation. After the input for the growth functions is ready, we then use gradient descent to find the optimum concentrations.

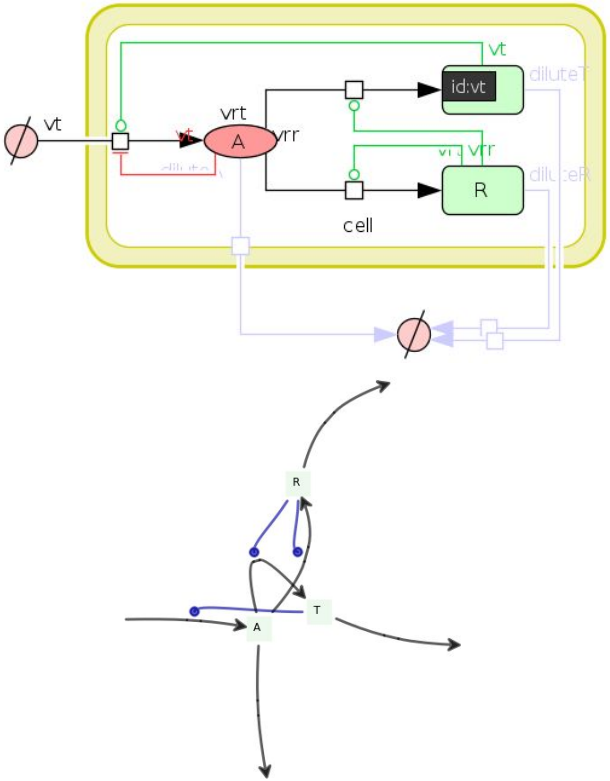
We need to store and process the data differently than in the time course analysis. For example, then we stored the rate functions into a 2D pandas data-frame but now directly dictionary for efficiency.

Also classify the species by SBO ID's and the reactions into metabolite synthesis, protein synthesis or dilutions reactions.

Several such methods are used to process various parts of the data and finally we have the kinetic functions of the reactions and their derivatives along with the stoichiometric matrices so that we can calculate growth rates on this.

Sample runs:

1) TR model:




```
optimize.optimizer(model="TR-model.xml", method="optimum-growth")
```

```
max growth rate: 2.151 1/h
phi_m:  [0.32365374] , c_s: [0.08164714]
```

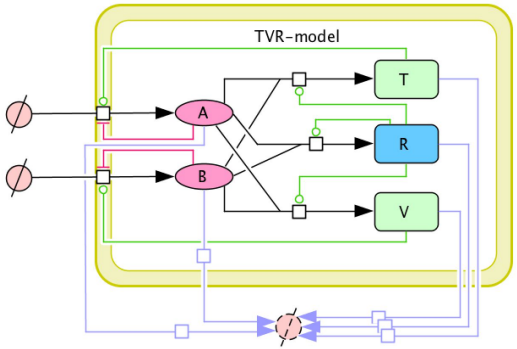
```
optimize.optimizer(model="TR-model.xml",
    method="optimize-for-protein-conc",
    initlist=[2.637e-4, 1e-3])
```

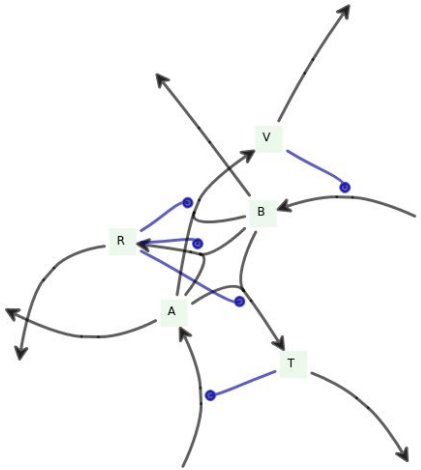
```
c_e:  [0.0002637 0.001    ]
phi_m:  [0.2637]    log10:  [-0.57888987]
c_s:  [0.05536532]    log10:  [-1.2567622]
growth rate: 5.8930e-04 1/s, 2.1215 1/h
```

```
optimize.optimizer(model="TR-model.xml",
    method="optimize-for-protein-ratios",
    initlist=[0.1])
```

```
phi_m:  [0.1]    log10: [-1.]
c_a:    [0.00854102]    log10: [-2.06849038]
growth rate: 3.5710e-04 1/s, 1.2855 1/h
```

2) TVR model:






```
optimize.optimizer(model="TVR-model.xml", method="optimum-growth")
```

```
max growth rate: 1.618 1/h
phi_m:  [0.0784061  0.21962039]
c_s: [0.09567716 0.04347405] , log10:  [-1.0191917  -1.36176988]
```

```
optimize.optimizer(model="TVR-model.xml",
                   method="optimize-for-protein-ratios",
                   initlist=[0.1, 0.1])
```

```
growth rate: 3.2729e-04 1/s, 1.1783 1/h
phi_m:  [0.1 0.1]
c_s:    [0.27720649 0.00561276]    log10: [-0.55719661 -2.25082387]
```

```
optimize.optimizer(model="TVR-model.xml",
                   method="optimize-for-protein-conc",
                   initlist=[9.55909e-7, 2.24928e-6, 1e-5])
```

```
growth rate: 4.2595e-04 1/s, 1.5334 1/h
phi_m:  [0.0955909 0.224928 ]
c_s:  [0.05400574 0.03461299]    log10:  [-1.26756006 -1.46076086]
```

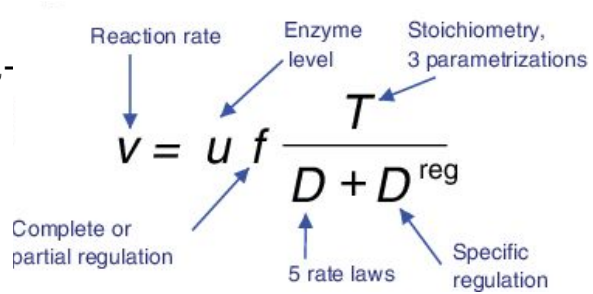
Next steps:

- If info is encoded (for example in the model annotations) about the type of reaction it can easily be mapped to the corresponding rate law, need to figure out where and in which format this info can be encoded.
- Applicable only for models of the kind described in the paper. This can adapt to newer models easily because we have processed the data already, we just need to supply to a new function.
- Rate derivative is not in the model, calculation is very inefficient and approximate. Need to have analytical derivative: using sympy or ready-made rate functions. Some calculation errors may add up. This is the major problem and a possible solution is described next.

Convenience rate laws

- Most reactions fall under certain categories in wrt their rate laws. We can take advantage of this and use ready-made rate laws.
- We have convenience rate laws of 8 different types of mechanics and kinetics like various Michaelis-Mentin reactions, transporter reactions, template driven polymerisations etc. (Liebermeister, Klipp, 2006 and Hofmeyer *et al.*, 2013).
- We have implemented these functions, and their derivatives for direct usage.

Modular rate laws



- Based on (Liebermeister et al., 2010) we have modular rate laws composed of several standardized components which can be selected by the reaction type.
- T gives the reversibility of the reaction (we use mass action kinetics); f the complete or partial regulation of the reaction; D the rate law of the reaction (direct/simultaneous binding, force dependent etc.) as DM, SM, FM, PM, or CM; D^{reg} as the specific regulation function.
- Using all these possibilities we've implemented a modular rate law (and derivative) that covers 20 types of reactions with all possible stoichiometries.

Using these rate laws

- The rate functions are much more efficient than those built into the model because of the vectorized implementation.
- The derivatives are even more efficient now, as they are both analytical and vectorized. They are also exact and not approximations.
- This solves the problem of finding efficient derivatives as now, we have ready-made laws which just need a mapping to the reaction.
- So we need a way to recognize the reactions as belonging to a type, by for example, using the SBO ID's of the reactions.