Optimizer for Balanced Growth Models in SBML

Anmol Reddy Mekala and Jatoth Shiva Tarun
Indian Institute of Technology Bombay

(Dated: June 9, 2020)

A description of the workings and various aspects of the cell-growth-optimizer package.

I. FUNCTIONALITIES

In *"On balanced-growth states of autosynthetic cell models"* (unpublished), Deniz Sezer defined balanced growth models and derived formulae to derive optimum growth states and growth rates. 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. We can also calculate the metabolite concentrations from a given metabolic state defined by the protein concentrations (`c_e`) in the method or protein ratios (`phi_m`).

II. REQUIREMENTS

The balanced growth models are coded in SBML in a .xml file. They contain information as that which is usually present in cell models on which time course analysis is done i.e. they have information about the reactions, species involved, their initial amounts and rate functions. Additionally they also need to have the SBO ID's coded for the species. Growth rates are calculated using the formula in Deniz's paper. Newton-Ralphson root finding method is used to solve stoichiometric constraints. Gradient descent is used for the optimization.

The cell system is assumed to be of the type as the simple models described in Deniz's paper:
- The reactions are of two types: metabolic synthesis (importing/exporting/converting metabolites) and protein synthesis (producing proteins).
- Each reaction (irreversible in the current scope) can consume and produce several metabolites
- Enzymes are of two types: metabolic proteins catalyzing metabolic synthesis reactions and ribosomes catalyzing protein synthesis reactions
- There is a one to one relationship between metabolic proteins and metabolic reactions, i.e. there are as many metabolic proteins as there are metabolic reactions.
- The ribosome catalyzes all protein synthesis reactions. In fact a specific fraction of total ribosome gets allocated for a specific protein synthesis reaction.
- The ribosome also catalyzes its own synthesis (autosynthetic)
- Cell size parameters and the alpha parameters are set to 1
- All metabolic synthesis reactions have modifiers

Runs on `python3`, uses basic math libraries like `numpy` and `pandas` and `libsbml` to deal with the models.

III. RUNNING THE CODE

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. 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"`. These concentration values or ratios are given in `initlist` as a python list. Use these definitions of `model`, `method` and `initlist` when calling the function.

And to run the code, by calling the function in the console:

- With the `cell_growth_optimizer` package folder in your directory import the definitions contained in `optimize.py`:

```
from cell_growth_optimizer import optimize
```

- Call the function to print the global optimum growth rate and the corresponding `c_s` and `phi_m`, (`initlist` can be anything):
`optimizer(model)` or
`optimizer(model, "optimum-growth")` or
`optimizer(model, "optimum-growth", initlist)`
- Call the function to print `c_e`, `phi_m`, the growth rate and the corresponding `c_s`, given `c_e` in `initlist`:
`optimizer(model,"optimize-for-protein-conc", initlist)`
- Call the function to print `phi_m`, the growth rate and the corresponding `c_s`, given `phi_m` in `initlist`:
`optimizer(model,"optimize-for-protein-ratios", initlist)`

IV. CODE DESCRIPTION

There are two files, `processSBML.py` which is a helper file and `optimize.py` the main file. The `processSBML.py` file gives the raw data from the SBML model and `optimize.py` processes this data. We describe only `optimize.py` here.

A. optimize.py

This file has been adapted from the code for time course anlaysis on SBML models (extracting info) part and the hard coded python model with Deniz's functions. What we have mainly done is build a pipeline between them and optimize for this specific purpose.

`processSBML.sbml2dfs(model)` takes all the info from the model and we get a dictionary of dataframes `dict_sbml` and these dataframes are then separately taken as `df_species`, `df_reactions`, `df_N`, `df_parameters` etc. The dictionary `model_params` has all the information about the current state of the model.

The function `classify_data()` takes all the dataframes created and adds the info needed for optimization, like kinetic laws, various cell parameters like rate constants, cell compartment sizes, lambda functions, variables to store concentrations etc. to `model_params`.
- The various species from `df_species` are classified by their SBO ID's (247, 252 or 250) and stored into `model_params` as `model_params['enzymes']`, `model_params['metabolic_proteins']` and `model_params['ribosomes']`.
- Using `df_reactions`, rate law functions are created as `model_params[reaction_name]()` with no argument and with their scope as `model_params` so that they use the variables with same name in `model_params` as in the formula. These rate law formulae are there simply as the text without distinguishing between the various variables and the mathematical operators involved in the formula. For example a reaction called `vt` would be called by `model_params['vt']()` and it would return `kT/(1+A/KmTA)` by trying to to use the keys `'kT'`, `'A'` and `'KmTA'` from `model_params`. The dictionary `temp_d` which maps the formed products to the reaction names. At the same time, using `df_reactions`, we also classify and store the different types of reactions in `model_params['meta_syn_reactions']`, `model_params['prot_syn_reactions']` and `model_params['dil_reactions']`. And in the list `model_params['prot_syn_reactions']`, we choose the proteins to be in the same order as they are found in the rows of `df_species` as this list is used for making `Gr` the order of which must correspond to the columns of `df_P`, described later.
- The dictionaries `reactions2products`, `products2reactions` and `reactions2modifiers` are initialised while dealing with `df_reactions`. `products2reactions` is used to make the reaction list `model_params['prot_syn_reactions']` correspond to correspond to, in *order*, `model_params['enzymes']`. `reactions2modifiers` is used to assign the columns of `df_S`, which are made for particular reactions, to the modifier involved. And similarly, `reactions2products` is used to assign the columns of `df_P`, which are made for particular reactions, to the proteins produced.
- Variables are created and assigned by making keys and values for parameters in `df_parameters`, the helper lambda functions in `df_functions`, compartment sizes in `df_compartments` in `model_params`. - Data from the other dataframes like `df_initAss` (the created functions are called to overwrite compartment values in 2 runs), `df_rules` (functions are created for assigning various variablee), `df_N` (used for ODEs) is also handled, but this collected data is not useful for any future calculation.
- Two important stoichiometric dataframes, `df_S` and `df_P` are created, with `df_S` having rows indexed by various metabolites involved and columns indexed by the metabolic proteins which are the modifiers for the metabolite synthesis reactions, and the entries give the amount of each metabolite produced or consumed (negative) in each reaction. `df_P` has rows indexed by metabolites and columns indexed by the enzymes. It gives the contribution of each metabolite to each enzyme, so that each entry tells how many moles of the metabolite are needed as a part of making one mole of the enzyme. And so all its entries are negative. In our usage, we don't use the data frames but the values `Smat = df_S.values(); Pmat = df_P.values()`
Only the metabolite synthesis reactions and the proteins synthesis reactions contribute to the growth rate calculations. The rates from these reactions are taken as functions `Fm(c_s)`, `Gr(c_s)` and their derivatives in `Fmjac(c_s)`, `Grjac(c_s)`, `Grinvjac(c_s)` (`Grinvjac` is the derivative of `1/Grjac`).
- `Fm`, `Gr` contain all the rates of the metabolite synthesis and protein synthesis reactions respectively in a single `numpy` array.
- To find each individual rate for a particular `c_s`, since we cannot directly call the previously extracted rate functions with `c_s` (they have no arguments), we update the values of the species which are in `model_params` by calling `update_params(c_s)` then executing `model_params[reaction_name]()`.
- Since we do not have the analytical derivatives available, we find the derivative values by using the method of finite differences (i.e. $f'(x) = \frac{f(x+dx)-f(x-dx)}{2 \cdot dx}$). We choose our `dx` as $6 \cdot 10^{-6}$.

`c_s0` has the initial metabolite concentrations taken from the model. The various calculation functions like `tau()`, `mu()`, `Jphi()` etc. all use a parameter '`model`' which is given by us as an empty python dict, `d_model`. It is used for storing `model['caa_pr']` value in the mathematical functions. `opt_growth_rate()`, `find_nroots_cpr()` and `find_nroots_phi()` are the three functions for the three methods respectively which give the optimum `c_s`.