# Lab 3 – Genome-scale metabolic modeling

Systems Biology 2019

Cheng Zhang – 4/12/2019

## About this lab

On this lab, we will learn how to perform genome-scale metabolic model (GEM) based computational analysis. Most of the analysis will be done in MATLAB (version 2017b or later), so you should have a MATLAB on your computer, or use one provided in the lab.

Similar to the previous lab, you have to answer all the questions to complete the lab, and you will be also provided with bonus questions which could award you points in the final examination if completed successfully.

## Preparation

To begin our lab, we need to install all the needed packages to prepare for the GEM based analysis. COBRA and RAVEN toolboxes are the two main toolboxes for GEM based analysis, and in this lab, we will be mainly focusing on RAVEN toolbox. If you are interested in COBRA toolbox, you could go to check the link 'https://opencobra.github.io/cobratoolbox/stable/index.html'.

For installation of RAVEN, you should first go to 'https://github.com/SysBioChalmers/RAVEN' and download the package as a zip file, and unzip it to the path you defined. After that, you should include this path in MATLAB to enable the usage of RAVEN functions.

The RAVEN toolbox is dependent on the Systems Biology Markup Language (SBML) library for importing/exporting GEMs in SBML format. The libSBML MATLAB API 5.17 is recommended, and you can go https://sourceforge.net/projects/sbml/files/libsbml/5.17.0/stable/MATLAB%20interface/ to download the library API for MATLAB. Again, you will need to unzip the downloaded package to a path you select, and include the path in MATLAB.

In addition, linear programming solver is needed for the model simulation for RAVEN toolbox. In this lab, we will use MOSEK for the modeling. You can download it (version 7.1.0.63) through the link https://www.mosek.com/downloads/7.1.0.63/, and install it follow the instruction (Note that if you are using a computer in the lab, it should have been already installed). You will also need to apply for an academic license from https://www.mosek.com/products/academic-licenses/, and you will get instant automatic response if you apply with your academic email. After installation and license the MOSEK solver following the instruction in the email, you will need to link the solver to MATLAB by including its path.

Finally, the course materials which include the model and data for the computer lab 3 should be downloaded through the link XXX to your working directory. Now, open MATLAB, and go to the working directory so you will have direct access to the course materials.

## Basic usage

By starting the GEM based analysis, you will need to import the model into MATLAB. Most of the model in the field are saved in SBML format (.xml file), and that's why we needed to install the SBML library before we start the actually analysis.

```
model = importModel('iAL1006 v1.00.xml',false)
```
'iAL1006 v1.00.xml' is a GEM of P. chrysogenum, and after imported with 'importModel' function, the fields in the model will be displayed in the Command Window (the function also reported 1027 errors, which is due to the update of SBML update and you can neglect).

Q1: How many reactions, metabolites and genes are there in this model? How can you know more about the reactions, metabolites and genes using the information included in the model?

Q2: Which fields in the model respectively represent the S matrix, objective function, upper/lower bounds of reactions and b?

Then, we could use the build in functions 'getExchangeRxns' and 'constructEquations' to find all the exchange reactions and show all their equations:

```
ExRxns = getExchangeRxns(model);
ExEqns = constructEquations(model,ExRxns)
```

Most modelling approaches using GEMs are based on the mass balancing around the internal metabolites in the system. However, in the exchange reactions, there are metabolites that only appear once in the model which would block the reaction. These metabolites are kept in the model to help the user to understand the biological meaning of the reaction. However, in order to simulate something, those metabolites have to be removed from the model to prevent the related reactions to be blocked. The function 'simplifyModel' is a general-purpose function for making models smaller. This includes the options such as grouping linear reactions and deleting reactions which cannot carry flux. Here it is chosen to delete the exchange metabolites, all reactions that are constrained to zero (mainly uptake of non-standard carbon sources), and all reactions that cannot carry flux (mainly reactions that were dependent on any of those non-standard carbons sources).

```
model = simplifyModel(model,true,false,true,true);
```

And how do the exchange reactions look like now? Let's look at them again.

```
ExRxns = getExchangeRxns(model);
ExEqns = constructEquations(model,ExRxns)
```

Now, the model is ready for simulation, and we will start with simple example. As discussed in the lecture, we need to constrain the exchange reaction of the model, and provide it with an objective function for simulation. As a first example of using the model, we will calculate the theoretical yield of carbon dioxide from glucose. The supplied model already allows for uptake of phosphate, sulfate, NH3, O2 and the co-factor precursors thiamin and pimelate.

Q3: How would you check the constraints for these uptake reactions? How much uptake are allowed for these metabolites in the model?

The 'setParam' function is used for setting constraints, reversibility and objective function coefficients.

Set the uptake of glucose to be no more than 1 mmol/gDW/h and no uptake of ethanol.

```
model = setParam(model,'ub',{'glcIN' 'etohIN'},[1 0]);
```

And also set the objective function to be maximize CO2 production.

```
model = setParam(model,'obj',{'co2OUT'},1);
```
The problem can now be solved using linear programming. The ‘solveLP’ function which uses the MOSEK solver will take a model and solves the linear programming problem defined by the constraints and the objective value coefficients.

```
sol = solveLP(model)
```

The obtained struct ‘sol’ now contains 4 fields, namely ‘x’, ‘f’, ‘stat’ and ‘msg’ which represent the full solution (flux distribution) to the model, the value of objective function (only it’s absolute value is important in this case), the status ID and the message of the status ID, respectively. ‘sol.x’ contains 1305 values which makes it rather difficult to interpret. A first approach is to look at only the reactions that transports metabolites in to or out from the system. The ‘printFluxes’ function prints (parts of) the flux distribution to the console.

```
printFluxes(model, sol.x, true, 10^-7);
```

One can see that the system took up one unit of glucose and 6 units of O2 while producing 6 units of CO2 and 6 units of H2O. To get more detailed insight how this happens, one can use the same function to plot all fluxes (above 10^-7).

```
printFluxes(model, sol.x, false, 10^-7);
```

The results show many reactions that have -1000 or 1000 flux. This is because there are loops in the solution. In order to clean up the solution one can minimize the sum of all the fluxes, which is alled parsimonious FBA (pFBA). This is done by setting the second flag to ‘solveLP’ to 1 (take a look at the documentation of ‘solveLP’, and you will find the option).

```
sol = solveLP(model,1);
printFluxes(model, sol.x, false, 10^-7);
```

Now there are much fewer reactions that are active. Some of them will probably appear to be a little "weird" but that is because the ATP and NAD(P)H produced has to go somewhere in order to be mass balanced.

If one wants to study the growth instead, change the objective to be the production of biomass instead of CO2.

```
model = setParam(model,'obj',{'bmOUT'},1);
sol = solveLP(model,1);
printFluxes(model, sol.x, true, 10^-7);
```

The results show that the growth rate is 0.084/h and that the system now also requires sulfate, phosphate, NH3, thiamin and pimelate.

Q4: How many reactions carried flux when optimize cellular growth?

## Flux Variability Analysis

Then, we will work on flux variability analysis (FVA). Essentially, flux variability analysis is looping FBA to explore the achievable upper and lower bounds of each reaction under specific constraints.

Firstly, we will use FVA to explore the upper and lower bounds for all reactions while allowing uptake of 1 mmol/gDW/h of glucose.

```
model = importModel('iAL1006 v1.00.xml',false);
model = simplifyModel(model,true,false,true,true);
model = setParam(model,'ub',{'glcIN' 'etohIN'},[1 0]);
for i = 1:length(model.rxns)
    tmodel = setParam(model,'obj',model.rxns(i),1);
    sol = solveLP(tmodel);
    UB(i,1) = sol.x(i);
    tmodel = setParam(model,'obj',model.rxns(i),-1);
    sol = solveLP(tmodel);
    LB(i,1) = sol.x(i);
end
```

Consequently, the obtained LB and UB will include the lower and upper bounds for each corresponding reactions, respectively.

Q5: How many essential and blocked reactions are there according to this FVA analysis result?

In addition, we will also try to set the growth of the model as the optimal value to keep the model under maximum growth rate, and then perform FVA to investigate again the upper and lower bounds for each reaction.

```
model = setParam(model,'obj',{'bmOUT'},1);
sol = solveLP(model);
vobj = sol.x(model.c==1);
model = setParam(model,'lb',{'bmOUT'},vobj);
for i = 1:length(model.rxns)
    tmodel = setParam(model,'obj',model.rxns(i),1);
    sol = solveLP(tmodel);
    UB_obj(i,1) = sol.x(i);
    tmodel = setParam(model,'obj',model.rxns(i),-1);
    sol = solveLP(tmodel);
    LB_obj(i,1) = sol.x(i);
end
```

The new upper and lower bounds are now stored in UB_obj and LB_obj, respectively.

Q6: How many essential and blocked reactions are there now with the new results? Is there more essential & blocked reactions compared to last result? Why?

Q7: Could you write your own code to find the upper and lower bound for CO2 secretion ('co2OUT') w/wo optimal growth rate, respectively?

## Essentiality analysis

Essentiality analysis is one of the most commonly used GEM based analysis, and it identifies the key reactions/genes/metabolites in the GEM that is required for the objective function. We will use the bacteria GEM again and perform essentiality analysis to identify what reactions and genes are essential for its growth.

```
model = importModel('iAL1006 v1.00.xml',false);
model = simplifyModel(model,true,false,true,true);
model = setParam(model,'ub',{'glcIN' 'etohIN'},[1 0]);
model = setParam(model,'obj',{'bmOUT'},1);
sol = solveLP(model);
vobj = sol.x(model.c==1);
```

```
cutoff = 0.1;
for i = 1:length(model.rxns)
    tmodel = removeReactions(model,model.rxns(i));
    sol = solveLP(tmodel);
    if isempty(sol.x) || sol.x(model.c==1)<=cutoff*vobj
        EssentialIndex(i,1) = true;
    else
        EssentialIndex(i,1) = false;
    end
end
EssentialReactions10 = model.rxns(EssentialIndex);
```

Here, we defined a reaction to be essential if its deletion will cause the growth decrease to less than 10% of its maximal value. Apparently, this is very strict criteria, and we can also try a different cutoff to see what will be changed.

```
cutoff = 0.5;
for i = 1:length(model.rxns)
    tmodel = removeReactions(model,model.rxns(i));
    sol = solveLP(tmodel);
    if isempty(sol.x) || sol.x(model.c==1)<=cutoff*vobj
        EssentialIndex(i,1) = true;
    else
        EssentialIndex(i,1) = false;
    end
end
EssentialReactions50 = model.rxns(EssentialIndex);
```

As we can see, there will be more essential reactions if we set a higher cutoff.

Q8: Which reaction(s) is additional in EssentialReactions50 compared to EssentialReactions10? Could you speculate the biological meaning of it?

Gene essentiality analysis is much more complicated compared to reactions essentiality, but luckily, there is some package available that could help us (https://doi.org/10.3389/fphys.2018.01355).

```
model = importModel('iAL1006 v1.00.xml',false);
model = simplifyModel(model,true,false,true,true);
model = setParam(model,'ub',{'glcIN' 'etohIN'},[1 0]);
model = setParam(model,'obj',{'bmOUT'},1);
sol = solveLP(model);
vobj = sol.x(model.c==1);
cutoff = 0.1;
EScores = ESS(model,1,cutoff,'g');
EssentialGenes10 = model.genes(EScores == 1);
```

Q8: If a gene is deleted from the model, how can you know which reactions are deleted based on the model itself?

## Bonus exercises

## Metabolite essentiality analysis

B1 Based on the reaction essentiality analysis script provided in this lab, write your own script to find the essential metabolites for the growth of the bacteria with 1 mmol/gDW/h of glucose uptake

(cutoff = 0.1). (Hint: you could look into RAVEN toolbox to find if there is any function that could be helpful.)

## Human cancer cell line model

## Download a HepG2 model, and set the constraint according to the given input file

B2.1 Find and download a human cancer GEM for liver cancer cell line named Hep-G2 from https://www.metabolicatlas.org/.

B2.2 Find the biomass reaction (representing the growth of the cancer cell line) in this model and set it as objective function.

B2.3 Set the constraint for the model based on the 'RPMI-1640 Media Model Input CB2030.xlsx' in the course material folder.

B2.4 Perform FVA analysis with the constraints and maximal growth rate and obtain the upper and lower bound for each reaction.

B2.5 Find all essential reactions and genes in this model with the constraints and 10% of growth as cutoff.