**The solutions to the exercises can be found in the tutorialX\_solutions.m files.**

# Exercise 0 - A basic example

This is not an exercise, but rather a short introduction that shows how to load a model, set some parameters and perform an optimization. The resulting fluxes are visualized and exported to a PDF file. The methods used here are explained in more detail in the coming exercises.

It is important to mention that the user must have an ability to open Excel format models successfully with *importExcelModel*. Users without such ability should skip to Exercise 4 directly as Exercise 1-3 involve working with models in Excel format.

# Exercise 1 - The basics

This exercise will deal with the RAVEN model format and the most basic aspects of GEM modeling. You will learn how to build a simple model from scratch, how to set parameters and how to perform simple simulations.

|  |
| --- |
| glycolysis.jpg |
| Figure 1. Glycolysis (some reactions are lumped together) |

Figure 1 shows a somewhat simplified version of glycolysis. Your task is to make a model of this system in order to answer how many units of ATP could be generated through glycolysis. If you don’t know this already it would take 15 seconds to count in the figure or 30 seconds to Google, but the goal here is to learn how to think about stoichiometric modeling.

* Open the file Excel file empty.xlsx, which contains the first reaction in the system expressed as a RAVEN model.
* Open the Matlab file tutorial1.m. This file contains the code for running the basic exercises. To run a section of code in Matlab, select it, right click, and choose Evaluate selection.
* Run the first line (*importExcelModel*…). This line converts the Excel representation to Matlab structure and performs a number of consistency checks at the same time. You will see that the program warns that there is no information about gene associations in the model, and that it warns that some metabolites are only used in one reaction. We will discuss the second warning later.
* Add the remaining 11 reactions to the Excel representation. You can name stuff whatever you like, so if you want to call “glucose 6-phospate” for “g6p” that’s fine. Save often and run the code to see if the model structure is correct.
* If you have entered all the reactions and run the first *importExcelModel* script you will see a warning similar to:

WARNING: The following internal metabolite(s) are only used in one reaction (zero flux is the only solution):

(m13 [c]) H2O

(m14 [e]) H2O

(m15 [c]) NAD+

(m16 [c]) NADH

(m18 [c]) phosphate

(m19 [c]) pyruvate

(m20 [e]) sucrose

* This is a very important warning so please be sure that you understand the following section before you move on. Since we’re modeling steady state, the production rate of a given metabolite must be the same as the consumption rate. But what if there are no reactions consuming or no reactions producing a given metabolite? Then the production/consumption rate must be zero, which is probably not what we want. The warning simply lists metabolites that only participate in one reaction, which means that it either cannot be consumed or that it cannot be produced. The fact that the warning says “internal metabolites” indicate that this is not the only kind of metabolite, and the solution lies in something we call “external metabolites”. These are metabolites which do not have to be mass balanced, but can be produced or consumed in any amount. Simulations are often centered on choosing which external metabolites are available and how fast they can be produced or consumed by the model. External metabolites are defined by having “true” in the UNCONSTRAINED field in the Excel sheet. They are also by convention placed in a compartment abbreviated “b”. These metabolites are also called “exchange metabolites” and the reactions involving them are called “exchange reactions”.
* In our case we need to allow for uptake of sucrose and production of pyruvate. At this point maybe we are not sure whether there will be net consumption or net synthesis of water, so make that exchange reaction reversible. We will later deal with NAD+/NADH and phosphate in some other way. The reaction for taking up sucrose would then look like “sucrose[b] => sucrose[e]”. Add the required reactions for sucrose, pyruvate and H2O. We normally only allow for uptake/excretion from the “extracellular” compartment so add transport reactions when needed. The warning should now read:

WARNING: The following internal metabolite(s) are only used in one reaction (zero flux is the only solution):

(m16 [c]) NAD+

(m17 [c]) NADH

(m19 [c]) phosphate

* We have now added all the “real” reactions involved in glycolysis. However, in order to be able to answer our question about the ATP production we have two remaining problems:
  1. How do we get the required NAD+ and how do we get rid of the produced NADH?
  2. How do we formulate the ATP production as a variable we can solve for?
* The first problem could be solved either by expanding the model to contain a larger proportion of metabolism so that the model has a way to regenerate NAD+ from NADH or by including a “fake” uptake reaction for NAD+ and a “fake” excretion reaction for NADH. I call these reactions fake because they are not how the system works in reality (the cell doesn’t take up extracellular NAD+ and so on…). The use of fake reactions is very common in this type of modeling, and cleverly designing fake reactions can help a lot when doing simulations. Choose either to deal with the problem by including ethanol production from pyruvate (via pyruvate decarboxylase + alcohol dehydrogenase) or by including fake exchange reactions.
* The second problem also requires fake reactions. Remember that the variables that we solve for are fluxes through reactions. One way to know how much ATP the system can generate is to maximize for the degradation of ATP (since the production and consumption has to match). Add a fake reaction for hydrolysis of ATP. Be careful about directionality so that you don’t add free ATP synthesis instead. When you have done this, you should see no warnings when running *importExcelModel*.
* The model is now done and we can finally try to answer our question about the ATP production. In GEM modeling we define the problem by:
  1. setting constraints on the fluxes, and
  2. defining an objective for the simulation
* We normally only set constraints on the exchange reactions; the things that the model can consume and produce. In our case it’s enough to constrain the uptake of sucrose. Set the UPPER BOUND for that reaction to 1.0 unit. We normally talk about the fluxes as being mmol/gDW/h, but when we are looking at yields, we might as well think of them as mol or “molecules”. Set the objective to maximize ATP degradation by putting 1 in the OBJECTIVE column for that reaction.
* Then import the model to Matlab by running *importExcelModel*. You will see that the model structure is printed. Select the name “smallModel”, right click, and choose Open selection (you could also write “open smallModel”). We will not go through this structure in detail now, but click around a little bit and try to figure out what the different fields stand for.
* Solve the optimization problem by running *solveLP*. We print the resulting exchange fluxes to be sure that everything worked like it should. Make sure that the carbon balance is correct.
* We then print all the fluxes. How much ATP could be generated from one unit of sucrose?

Answer: 4 mol/mol

## Exercise 2 - Performing simulations

This exercise will show you how to run FBA and MOMA simulations and how you can use GEMs as a scaffold for interpreting microarray data. We will be using a simplified model of yeast metabolism as an example. Figure 2 shows the system. Note that there are slight differences in naming of metabolites; particularly all metabolites have their compartment in their name.

|  |
| --- |
|  |
| Figure 2. Central carbon metabolism in yeast (adapted from Förster, 2003) |

* Open the file smallYeast.xlsx. You will hopefully recognize the model structure from Exercise 1, even though some more stuff is added this time. The most important difference is composition for the metabolites and gene associations for the reactions. Metabolite composition is useful for finding errors in the models and gene associations allow us to link the models to other types of data, such as from microarrays. By default production of acetate, biomass, glycerol, CO2, and ethanol is allowed and consumption of O2 and glucose is not allowed. Try to validate the model by confirming the results from Exercise 1 (the maximal ATP production in glycolysis). You can set simulation parameters in the Excel sheet like we did before, or you can try to use the *setParam* function (see tutorial2.m). Just remember that we used sucrose in the first example and that we use glucose here.
* Calculate the maximal yields of biomass, ethanol, glycerol, and acetate on glucose and fully aerobic (biomass is in 1/h and the other fluxes are in mmol/gDW/h).
* We want to predict how the flux distribution would change if we change conditions. You should now know how to import models and set parameters. Import the model twice and generate one model which has access to 1 mmol/gDW/h of glucose and unconstrained oxygen, and one which has access to 1 mmol/gDW/h of glucose and 0.5 mmol/gDW/h of O2. Optimize for growth under both conditions. What were the outputs? Take a look in the tutorial and try to run *drawMap* for the two models. Open the resulting pdf-file and see if you can understand (in broad terms) how metabolism changes between the two conditions.
* Say that you would like to increase glycerol production in your strain (maybe you want it to make sweeter wine). FBA is useful for suggesting gene deletions which can couple your goal with the overall objective for the cell, which is to grow as fast and efficient as possible. Set your parameters to maximize for growth under anaerobic conditions and run the code for making a single gene deletion using FBA. How large was the difference in glycerol production? Can you come up with a hypothesis as to how the deletion leads to higher glycerol production? You can also use the function *followChanged*, which compares two flux distributions and let you search between them (see tutorial2.m)
* FBA is based on the assumption that cells have the same objective even after a perturbation; to grow as efficiently as possible. This is probably true after long evolution times, but shortly after, say, a gene deletion the cell might have other objectives. MOMA is based on that the perturbed cell would like to change its metabolism as little as possible. MOMA is normally used when we have experimental data for the wild type strain and would like to guess how the phenotype would change after a perturbation. These are some batch data for growth on glucose
  + acOUT (Production of acetate):0
  + biomassOUT (Production of biomass):0.67706
  + co2OUT (Production of CO2):22.4122
  + ethOUT (Production of ethanol):19.0946
  + glyOUT (Production of glycerol):1.4717
  + glcIN (Uptake of glucose):15
  + o2IN (Uptake of O2):1.6
  + ethIN (Uptake of ethanol):0
* Load one model with these constraints and one without any constraints on the exchange reactions but with the ZWF reaction turned off. Run *qMOMA* and plot the flux distributions. Is the glycerol production larger?
* GEMs can also be used to aid in interpretation of other types of data. The last lines load microarray data from batch growth on ethanol compared to glucose and perform a Reporter metabolites test. These are metabolites around which significant transcriptional changes occur (for such a small model it is not super interesting though, because there are changes almost everywhere). The very last lines plot all reactions involving any of those metabolites. This can be very useful when working with large models.

## Exercise 3 – Finding problems and fixing them

The whole point of GEMs is that they are large. It is by incorporating the entire known metabolism of any given organism that complexity arises. However, this makes it almost certain that all models will contain errors. This is true regardless of whether you build the model yourself or if you use a model from someone else. One of the issues is that if one group publishes a model for some specific purpose it’s likely to function well in that specific part of metabolism, but it may not function at all for your problem. It is therefore a good idea to perform a round of error checking even if it’s a published model you are using.

Model validation is an iterative process because some errors might not have an effect until some other errors have been fixed. It is not uncommon that the model “works” well in the beginning of the reconstruction process because there are errors that let it cheat on things like redox or energy balance. The model then works worse and worse as the errors are dealt with until all or most errors are fixed, after which it will start to work again. Because of this I think it is much more important to try to make the model do something it shouldn’t be allowed to do rather than to test for the stuff it should do.

I have supplied a version of the small yeast model with errors inserted (smallYeastBad.xlsx). Your task is to find and fix them. Some errors will be obvious (it is rather difficult to introduce errors in such a small model, because there is very little redundancy in it), but please don’t fix them until you’ve “found” them, because otherwise you might get unpredictable results. Most of the stuff you do here can be done with *gapReport*, but I think it’s better to do them step by step.

1. The number one thing to check for is that your model cannot make something from nothing, i.e. no metabolites should be produced if you don’t give the model access to any carbon sources (this should be done for all elements, but carbon is the most important). A simple way to do this would be to optimize for the sum of all the producing exchange reactions, while keeping the consuming reactions closed. Any solution other than 0 would then be bad. Try that.
2. You didn’t get a solution, right? That’s good, but there could be other factors that prevent the error from showing its ugly face. Maybe it costs energy or redox power for example. Or maybe the necessary reactions are in different compartments. It’s generally a good idea to relax as many constraints as possible when searching for errors. I always include a temporary reaction like “ATP + H2O ⬄ ADP + PI” and similar reactions with NADH and NADPH. Remember that we’re trying to “provoke” the model to show the errors. Add these reactions and try again.
3. Did you get production of ethanol? If so, print the resulting fluxes and see if you can find the error. GEMs are normally very underdetermined, which means that there are infinite numbers of solutions to any given problem. When we solve using solveLP(model) we just get a random solution which meets our objective and satisfy the constraints. These solutions often contain loops and are therefore difficult to interpret. You can read more if you write “help solveLP”, but from here on we solve using “solveLP(model,1)”. This minimizes the sum of fluxes in order to have more easily interpreted results. Find and fix the error and rerun. You should not have any production of anything now.
4. In GEMs you normally have excretion of only a few metabolites while having very many internal metabolites. A common case is that you have an error that would like to produce something from nothing, but in order to do so it also has to produce some other metabolite which there is no exchange reaction for. A convenient way to test for this is to allow all metabolites to be excreted. We can do this by changing the model.b structure. Normally it’s always a vector of zeros, but if you add a second column RAVEN will interpret it as lower and upper bound on the equality constraints. So if we put model.b=[model.b inf(numel(model.b),1)]; we can now excrete anything we want. Do this and see if you can produce anything. I get ethanol, glycerol, and CO2. Look at the fluxes and find the error. You can get a clue by looking at the warnings from *SBMLFromExcel*. Since this is a problem that comes from reactions being unbalanced, the problematic ones have to be in one of the warnings. Which was the metabolite that had to be excreted in order for the error to appear? You need to do this step two times to find both errors.
5. The same thing we did in step 4 can be done with the function *canProduce*. There is a sister function called *canConsume*. It checks which metabolites can be consumed by the model. Change so that no production is allowed and run *canConsume*. You should see that 12 metabolites could be consumed even though the model is not allowed to produce anything. Pick one of them, force uptake of it by setting the lower bound to non-zero. If you do this you may not be able to get a feasible solution. That is because the problem solved by *canConsume* allows input of all metabolites, but our model only of O2 and glucose. Modify the model.b variable to allow for uptake of all metabolites. Study the fluxes and try to find the wrong one. Even if you fix the problem you will see that the model can still get rid of O2. This is because of the reactions that we included for testing (NAD ⬄ NADH is not elementally or redox balanced). We are done with this part of the exercise, so you can delete those reactions.
6. Unbalanced reactions are a relative small problem, since they are so easy to find. A much bigger problem is when metabolites are named differently even though we mean them to be the same. Use smallYeastBad2.xlsx from here on. A first check is to see which reactions can carry flux when we allow for all uptakes and outputs of exchange metabolites. There are several ways to check this, and here we’ll use *simplifyModel*. The primary purpose of this function is to remove unnecessary stuff from a model to make it smaller, but since it removes “bad” reactions we can use it for error identification as well. If you run it like it is in tutorial3.m you will see that there are about 20 metabolites and reactions that are dead ends. That is quite a lot, so take a look at the warnings from *importExcelModel* and see if you can catch the obvious spelling error.
7. That didn’t help very much. Sometimes it is very difficult to find out where the root of the problem is. This is particularly true if it is in a region with many interconversions between metabolites and no clear input/output (Figure 3).

|  |
| --- |
|  |
| Figure 3. If one reaction is wrong here it will be difficult to find since everything looks so connected because it’s produced and consumed in many reactions. |

A powerful but somewhat tricky function is *checkProduction*. It helps you answer the question “which metabolites do I need to synthesize if I want to have net synthesis of everything?”. Look at the suggestions from *checkProduction* if you run it like in the tutorial. minToConnect tells us that we need to synthesize 12 metabolites in order to have net synthesis of everything. However, 8 of them are co-factors or contain co-factors. Since we don’t have net synthesis of co-factors in this small model those are not very interesting (we don’t synthesize coenzyme A or ATP from glucose). I would take a look at the top one that is not a co-factor ☺. This one is a bit tricky and you might want to look it up in KEGG.

1. Still quite a bit of dead ends, and nothing that immediately looks like it would fix everything. It could be that we are missing some reactions. We could try to include reactions from a set of other models to fill the gaps. This is a computationally expensive task for a large network, but for our small it’s easy. We could use any model structure, but we take the small yeast model from exercise 2. Run the code and include the suggested reaction(s). Run the previous tests to make sure that everything works.
2. Tada! And don’t forget the *gapReport* does all these things.

ANSWERS:

3: Change ADH1 from acetaldehyde[c] + NADH[c] => 2 ethanol[c] + NAD(+)[c] to acetaldehyde[c] + NADH[c] => ethanol[c] + NAD(+)[c].

4: Change FBP from beta-D-fructofuranose 1,6-bisphosphate[c] => 2 beta-D-fructofuranose 6-phosphate[c] + phosphate[c] to beta-D-fructofuranose 1,6-bisphosphate[c] => beta-D-fructofuranose 6-phosphate[c] + phosphate[c] and change PFK from ATP[c] + beta-D-fructofuranose 6-phosphate[c] => ADP[c] + 2 beta-D-fructofuranose 1,6-bisphosphate[c] to ATP[c] + beta-D-fructofuranose 6-phosphate[c] => ADP[c] + beta-D-fructofuranose 1,6-bisphosphate[c].

5: Change PDC from pyruvate[c] => acetaldehyde[c] to pyruvate[c] => acetaldehyde[c] + CO2[c].

6: 6-O-phosphono-D-glucono-1,5-lactonec] in ZWF1 should be 6-O-phosphono-D-glucono-1,5-lactone[c].

7: Dihydroxyacetone phosphate (DHAP) and glycerone phosphate (GLYP) are the same metabolite.

# Exercise 4 – Reconstructing a model from KEGG

This exercise is about creating a model from KEGG, based on protein sequences in a FASTA file, and doing some functionality checks on the model. The example case is for the yeast *Saccharomyces cerevisiae*. This tutorial is more of a showcase than the other three, and its main purpose is to serve as a scaffold if you would like to reconstruct a GEM for your own organism. See the tutorial4.m file for details.

# Exercise 5 – Conduct *de novo* GEM reconstructions using both MetaCyc and KEGG databases, and refine the existing model by the combined draft model.

This exercise is to reconstruct draft GEMs for the antibiotic-producing bacterium *Streptomyces coelicolor* by using both KEGG and MetaCyc pathway databases. A combined model with comprehensive coverage of metabolic pathways will integrated from three different *de novo* reconstructions with input of a FASTA format file with all protein sequences of *S. coelicolor* A3(2). This combined model will be used for refinement of existing iMK1208 model and generation of a new version of GEM, the Sco4 model. The manual curation results are provided in the Excel file SupportingTables.xlsx. This tutorial is a showcase of the new features released in REVAN 2.0, and aim to demonstrate the utilization of the newly developed functions for reconstruction and curation. See the tutorial2018.m file for details.