Practical course "Constraint-based modelling: a primer on basics and examples"

Prepared by Miguel Ponce de León and Arnau Montagud from the Barcelona Supercomputing Center for the "Máster Universitario en Bioinformática" by the "Universitat de València".

10 December, 2019

In [ ]:

```
import cobra
import cobra.core
from cobra.core import Model, Reaction, Metabolite
from IPython.display import Image
```

[Full cobrapy documentation]

Part 1: Warming up.

Objective:

To get familiar with COBRA library by creating a toy model (figure 1) and manipulating it.

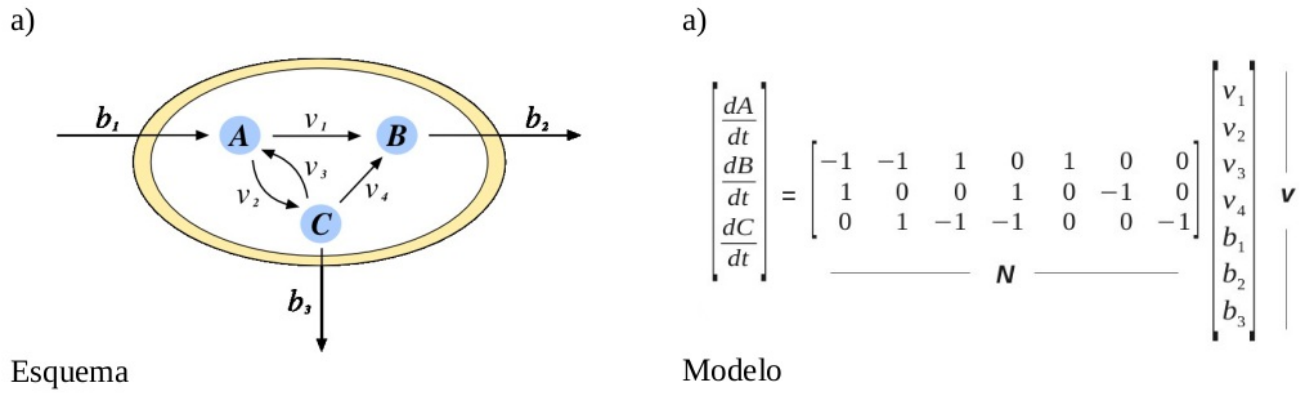

Figure 1. Toy model with three metabolites (A, B y C), four reactions (v1-v4) and three exchange fluxes (b1-b3). a) Model chart; b) Stoichiometric matrix

Documentation: https://cobrapy.readthedocs.io/en/latest/building_model.html/en/latest/building_model.html

In [ ]:

```
toy_model = Model('Toymodel')
```

In [ ]:

```
A = Metabolite("A")
A.compartment = 'cytosol'
B = Metabolite("B")
B.compartment = 'cytosol'
```

Excercise 1.1:

Create Metabolite C

In [ ]:

```
######################
# Create Metabolite C
######################

## TODO
## Write your code below
```

In [ ]:

```
## Add the metabolites to the model
toy_model.add_metabolites([A, B, C])
```

In [ ]:

```
# Print the reactions of a given metabolite
print(toy_model.metabolites.A.reactions)
# We get an empty set because we haven't created any reaction yet
```

In [ ]:

```
# Creating the reactions

# Create reaction with id b1
b1 = Reaction("b1")

# To add metabolites to the reactions we need to pass
# a dictionary with metabolites as keys
# and the stoichiometric coefficients as values
b1.add_metabolites({A: 1})

# The same is done for the other reactions
b2 = Reaction("b2")
b2.add_metabolites({B: -1})

b3 = Reaction("b3")
b3.add_metabolites({C: -1})

v1 = Reaction("v1")
v1.add_metabolites({A:-1, B:1})

v2 = Reaction("v2")
v2.add_metabolites({A:-1, C:1})

v3 = Reaction("v3")
v3.add_metabolites({A:1, C:-1})

# Create v4 (Excercise 1.2)
```

Excercice 1.2:

Create reactions v4 (and add its metabolites)

In [ ]:

```
########################
# Create reactions v4
########################

## TODO
## Write your code below
```

In [ ]:

```
# Adding the reactions to the toy model
toy_model.add_reactions([b1,b2,b3,v1,v2,v3,v4])
```

Write the model in SBML format

Write the model in SBML format

1. First import the corresponding function
2. Write the model
3. Optional: You can inspect the SBML using some plain text editor.

In [ ]:

```
import cobra.io
from cobra.io import write_sbml_model

# Saving the model to a file in SBML format
write_sbml_model(toy_model, "out/toymodel.sbml")
```

Excercice 1.3: Optimisation

In [ ]:

```
# Setting the limits on the inputs
toy_model.reactions.b1.upper_bound = 1

# Setting a reaction to be optimized as the objective or target
toy_model.reactions.b2.objective_coefficient = 1
```

In [ ]:

```
# To compute a FBA on the model we use the following function:
solution = toy_model.optimize()

# The result is a solution object which containsthe the following attributes:
# objective_value: the objective value of the optimized function (biomass!)
print("Objective value: %.2f\n" % solution.objective_value)
# solution.status: shows the status of the solution, it should be optimal
# if it is unfeasible, this means that there is no feasible solution
print("Status: %s\n" % solution.status)
# solution.fluxes: is a datagrame (pandas) storing the reactions (index) and
# their flux values found in the optimal solution.
print("Fluxes:\n")
print(solution.fluxes)

# Saving the solution into tab-separed-value (tsv) format (plain text)
solution.fluxes.to_csv("out/toymodel_fba.tsv", sep="\t")
# Inspect the file
```

Part 2: Genome-scale modelling.

In this part we are gonna use a genome-scale metabolic model of Escherichia coli named iJO1366 The file has already been stored in the data folder and its path is data/iJO1366.xml

Alternatively, you can also access it here:

- http://bigg.ucsd.edu/models/iJO1366

to download the model and to see other metadata (citation, description, etc)

Part 2.1: Studying the model.

Read the SBML model

First we need to import the function read_sbml_model from the cobra.io modules

In [ ]:

```
from cobra.io import read_sbml_model

# State the path to the file iJO1366.xml
sbml_fname = './data/iJO1366.xml'
```

```
# Read the model
model = read_sbml_model(sbml_fname)
```

Exercise 2.1: Inspecting the model's numbers

First print model description

1. print(model)
2. Print the total number of reactions: print len(model.reactions)
3. Print the total number of metabolties: print len(model.metabolites)
4. Print the total number of genes: print len(model.genes)
5. Access a particular reaction:
   - You can do it directly with: rxn = model.reactions.ENO
   - Or you can do use the function get_by_id: rxn = model.reactions.get_by_id('ENO')
6. Inspect the reaction by printing:
   A. rxn.name
   B. rxn.id
   C. rxn.bounds
   D. rxn.reaction
   E. rxn.gene_reaction_rule

In [ ]:

```
## TODO
## Write your code below
```

Exercise 2.2: Inspecting the genes

First print the model description

1. Access a particular reaction:
   - You can do it directly with: gene = model.genes.b0720
   - Or you can use the function get_by_id: gene = model.genes.get_by_id('b0720')
2. Inspect the reaction by printing:
   A. gene.name
   B. gene.id
   C. gene.reactions

In [ ]:

```
## TODO
## Write your code below
```

Exercise 2.3: Inspecting the systems' boundaries

- see the exchange fluxes
- see the objective function (the reaction set to be optimized)

Use print(model.summary())

You can also find the objective function using the following filtering technique:

- [r for r in model.reactions if r.objective_coefficient == 1]
- the reaction id of the biomass is Ec_biomass_iJO1366_WT_53p95M and the exchange fluxes can be accessed using:
- model.boundary

In [ ]:

```
## TODO
## Write your code below
```

Part 2.2 Running a Flux Balance Analysis (FBA).

Documentation: https://cobrapy.readthedocs.io/en/latest/simulating.html/simulating.html

By default, the model boundary condition (growth medium) is M9 aerobic (glucose minimal)

1. Check the medium by inspecting the lower_bound of the following reactions:
   - EX_glc_e_.lower_bound
   - EX_o2_e_.lower_bound
2. Optimize biomass using:
   - solution = model.optimize()
3. Inspect the solution as we did previously in Part 1.2 Optimization.

In [ ]:

```
solution = model.optimize()

print("Objective value: %.2f\n" % solution.objective_value)
print("Status: %s\n" % solution.status)

print("Fluxes:\n")
print(solution.fluxes)

# Saving the solution into tab-separed-value (tsv) format (plain text)
solution.fluxes.to_csv("out/iJO1366_fba.tsv", sep="\t")
```

Exercise 2.4: Identify reactions in the iJO1366 simulation:

Inspect the flux value of the following reactions

- The glucose consumption: EX_glc*e*
- The oxygen consumption: EX_o2*e*
- The biomass reaction: Ec_biomass_iJO1366_WT_53p95M

HINT: use the solution object -> solution.fluxes.reaction_id

In [ ]:

```
## TODO
## Write your code below
```

Part 3: Knock out studies.

Exercise 3.1: Single knock out study.

Documentation: https://cobrapy.readthedocs.io/en/latest/deletions.html#Knocking-out-single-genes-and-reactions

We will use gene b0720 as an example

In [ ]:

```
from cobra.manipulation import find_gene_knockout_reactions

# Pick a gene of interest
gene =  model.genes.b0720

# Inspect the reactions associated to b0720
print("id\treaction_name")
for r in gene.reactions:
    print("%s \t%s" % (r.id,r.name))

print()
# We can also check the genes associated to this reaction
reaction = model.reactions.CS
print("GPR:",reaction.gene_reaction_rule)
```

Alternatively, COBRA can find the proper reaction to be disabled when a gene is knocked out as follows:

gene = model.genes.b0720

```
with model:
    gene.knock_out()
    ko_solution = model.optimize()
```

This codes knocks out the gene b0720, recalculates the FBA and stores the new solution in ko_solution.

In [ ]:

```
# If we perform the knockout in the "with" block we don't need to care
# about restoring the knocked out gene afterwards; it is automatically restored out of the "with" block
.
with model:
    gene.knock_out()
    ko_solution = model.optimize()

#############################################################################################
# TODO
# Check the growth value (Hint: ko_solution.fluxes.Ec_biomass_iJO1366_WT_53p95M or ko_solution.objectiv
e_values)
# What happened?

## write your code below
```

Go to the Ecocyc database and check the in vivo experimental result for the knockout of b0720 by accessing the following link:

- https://ecocyc.org/gene?orgid=ECOLI&id=EG10402

Is b0720 essential or not?

Exercise 3.2: Systems-wide knock out study of *E. coli*.

COBRA has a special function to run the single gene knock outs of a list of genes.

The function's name is single_gene_deletion

First import the function

In [ ]:

```
# Import the function single_gene_deletion
from cobra.flux_analysis import single_gene_deletion
```

In [ ]:

```
# Then get the list of all the genes
all_genes = [g.id for g in model.genes]

# Running in silico (takes a while)
knockout = single_gene_deletion(model, gene_list=all_genes)

# This is a fix to get the gene's id as the index
index_mapper = {i:list(i)[0] for i in knockout.index}
knockout = knockout.rename(mapper=index_mapper, axis=0)

# The output of the function single_gene_deletion is a dataframe
print(knockout)
```

In [ ]:

```
# We define a threshold to define whether the reduction on the biomass flux is considered lethal.
threshold = 0.01

# Use this threshold to find which set of genes' knock out reduce the predicted growth below the thresh
old.
```

```
insilico_lethals = set(knockout.index[knockout.growth< threshold])
# The set of non-essential genes are the genes with a growth value above the threshold.
insilico_non_lethals = set(knockout.index[knockout.growth > threshold])

print("in silico lethals:", len(insilico_lethals))
print("in silico non lethals:", len(insilico_non_lethals))
```

In [ ]:

```
# Now we need to experimentally verify essential and non-essential gene sets.

# Read the set of essential genes in vivo
import json
fname = './data/m9_invivo_lethals.json'
with open(fname) as fh:
    invivo_lethals = json.load(fh)
    invivo_lethals = set(invivo_lethals)


# Convert the list of all model genes into a set.
all_genes = set([g.id for g in model.genes])

# We can use the difference to obtain the list of in vivo non-lethals
invivo_non_lethals = all_genes - invivo_lethals

# Print the size of both sets
print("in vivo lethals:", len(invivo_lethals))
print("in vivo non lethals:", len(invivo_non_lethals))
```

In [ ]:

```
# https://en.wikipedia.org/wiki/Receiver_operating_characteristic

# True Positives, genes predicted as essentials that are essentials in vivo (correctly predicted)
TP =  insilico_lethals & invivo_lethals

# True Negatives, genes predicted as NON-essentials that are NON-essential in vivo (correctly predicted
)
TN = insilico_non_lethals & invivo_non_lethals

# False Positives, wrongly predicted as NON-essential genes
FN = insilico_non_lethals & invivo_lethals

# False Positives, wrongly predicted as essential genes
FP = insilico_lethals & invivo_non_lethals

# True in vivo esssential genes
P = TP | FN
# True in vivo NON-esssential genes
N = TN | FP
```

Part 4: Metabolic engineering.

Exercise 4.1:

1. Get a model organism model and select a metabolite of interest. Hints:
   - Link to a models database.
   - Yeast consensus model
   - *E. coli* consensus model
   - Link to KEGG pathways, sections 1.9 thruogh 1.11
2. Identify which conditions are needed:
   - Find enzymes and genes
   - Find stoichiometry
   - Find proper growth media
3. Evaluate the production of the metabolite of interest under the wild type and the specific growth conditions.
4. Perform a theoretical productivity analysis. Hint: Perform several simulations changing the influxes values and studying the increase in production of you metbaolite of interest.

In [ ]:

```
```

Evaluated Exercises

Excercise E1:

Complete the following table

In vivo \ In silico	in silico lethal	in silico non-lethal
in vivo lethal	?	?
in vivo non-lehtal	?	?

Excercise E2:

Acces the following link:

https://en.wikipedia.org/wiki/Sensitivity_and_specificity

Get the formulas and calculate the following metrics:

- sensitivity
- specificity
- precision
- accuracy

Excercise E3:

In one paragraph, comment the predictive capacity of the model and briefly discuss the possible sources of errors.

In [ ]:

```
# Sensitivity, recall, hit rate, or true positive rate (TPR)
# We computed the sensitivity as follows
sensitivity = len(TP) / len(P)

# TODO
# complete the following code

# Specificity, selectivity or true negative rate (TNR)
specificity = ## COMPLETE HERE

# Precision or positive predictive value (PPV)
precision =  ## COMPLETE HERE

# Accuracy (ACC)
accuracy =  ## COMPLETE HERE

# Print the different values and discuss their meanings
```

Excercise E4:

In one paragraph, comment which organism and metabolite of interest you have chosen.

Detail the reactions that should be added to this organism's metabolism and which enzymes from which organism perform them.

Comment on what other modifications do you foresee will be needed in this modified organism (for instance, adjusting for codon usage bias, modifying some upstream reaction).