CBM_tutorial

December 16, 2018

```
In [ ]: import cobra
        import cobra.core
        from cobra.core import Model, Reaction, Metabolite
        from IPython.display import Image
```

0.0.1 Full cobrapy documentation

1 Part 1

1.0.1 Objective:

To get familiar with cobra library by creating and manipulating the toy model (figure 1).

Figura 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_mode

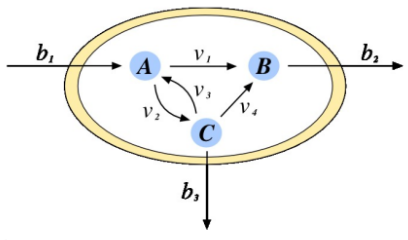
```
In [ ]: toy_model = Model('Toymodel')
```

```
In [ ]: A = Metabolite("A")
        A.compartment = 'cytosol'
        B = Metabolite("B")
        B.compartment = 'cytosol'
```

1.0.2 Excercise 1.1

Create Metabolite C

a)



Esquema

a)

$$\begin{bmatrix} \frac{dA}{dt} \\ \frac{dB}{dt} \\ \frac{dC}{dt} \end{bmatrix} = \underbrace{\begin{bmatrix} -1 & -1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 & -1 \end{bmatrix}}_{N} \underbrace{\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}}_{v}$$

Modelo

title

```
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 reactionsof a given metabolites
        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 reaction passing
        # a dictionary withs metabolites as the 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)
```

1.0.3 Excercice 1.2:

Create reactions v4 (and add its metabolites)

```
In [ ]: ########################
        # Create reactions v4
        ########################
```

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

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

1.0.4 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")
```

1.1 Part 1.2 Optimization

```
In [ ]: # Setting the limits on the inputs
        toy_model.reactions.b1.upper_bound = 1

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

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

        # the results is a solution object which containsthe 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 infeasible, 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 value 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")
        # instect the file
```

2 Part 2 - Genome-scale modeling

In this part we are gonna used a genome-scale metabolic models of E. coli named iJO1366 The files is already sored in the data/ folde and its path is data/iJO1366.xml

You can also access: - http://bigg.ucsd.edu/models/iJO1366
to download the model and to see other metada data (citation, description, etc)

2. Reading a 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

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

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

2.0.1 Inspecting the model

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: 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:
7. rxn.name
8. rxn.id
9. rxn.bounds
10. rxn.reaction
11. rxn.gene_reaction_rule

```
In [ ]: ## TODO
        ## Write your code below
```

2.0.2 Inspecting the genes

First print model description

1. Access a particular reaction:

- You can do it directly: gene = model.genes.b0720
- Or you can do use the function get_by_id: gene = model.genes.get_by_id('b0720')

6. Inspect the reaction by printing:
7. gene.name
8. gene.id
9. gene.reactions

```
In [ ]: ## TODO
        ## Write your code below
```

2.0.3 Inspecting the model (2)

- see the exchanges 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] * thre reaction id of the biomass is Ec_biomass_iJO1366_WT_53p95M and the exchanges fluxes can be accessed using: * model.boundary

In [ ]:
```
## TODO
## Write your code below
```

2.1 Part 2.2 Running Flux Balance Analysis (FBA)

Documentations: 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 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: ### Section **??**

(review this part again)

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")
```

2.1.1 Identificar en el listado generado anteriormente:

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: usar el objeto solución -> solution.fluxes.reaction_id

In [ ]:
```
## TODO
## Write your code below
```

2.2 Parte 3

2.2.1 3.1 – Knockout in silico

Documentations: 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

        # we pick a gene of interest
        gene =  model.genes.b0720

        # we caninspect 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 associate to this reaction
        reaction = model.reactions.CS
        print("GPR:",reaction.gene_reaction_rule)
```

To make out live easier, cobra can resolve the problem of findinding the correct reactions to disable when a gene is knocked as follows:

gene = model.genes.b0720

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

The give code knocks the gene b0720, recalculates the FBA and store the new solution in ko_solution

```
In [ ]: # We do the knockout in the "with" context and in this way we don't need to care
        # about restoring the kcnocked gene; it becomes automatically restore out of the with
        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_
        # What happened?

        ## write your code below
```

Got to the Ecocyc database and check the invivo experimatl result for the knockout of b0720 by accessing the following link: * https://ecocyc.org/gene?orgid=ECOLI&id=EG10402

Is b0720 essentail or no?

3 3.2 – Knockout in silico: Large Experiment

cobra has a special function to run single gene ko on a list of genes.
The function's name is single_gene_deletion
So, first should import the function

In [ ]:
```
# Import the funstion single_gene_deletion
from cobra.flux_analysis import single_gene_deletion
```

In [ ]:
```
# First 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 fixed 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 wheather the drop on the biomass flux is treated as
threshold = 0.01

# Use or threshold to find the set of genes whose ko reduce the predicted growth below
insilico_lethals = set(knockout.index[knockout.growth< threshold])
# The set of non-essential genes are the genes showing a growth value above the thresh
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 the experimentally verifeied essentail and non-essential genes

# read the set of essential genes
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 set 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 esscencials that are essential in-vivo (correctly pr
        TP =  insilico_lethals & invivo_lethals

        # True Negatives, genes predicted as NON-esscencials that are NON-essential in-vivo (c
        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
```

4 Evaluated Exercises

4.0.1 Excercise 1

1. Complete the following table

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

4.0.2 Excercise 2

Acces the following link:
https://en.wikipedia.org/wiki/Sensitivity_and_specificity
Get the formulas and calculate the for measures: * sensitivity * specificity * precision * accuracy

4.0.3 Excercise 3

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 value and dicuss their meaning
```