PyNetMet: Description of the classes, attributes and methods

D. Gamermann*[a,b]

[a]Cátedra Energesis de Tecnología Interdisciplinar, Universidad Católica de Valencia San Vicente Mártir, Guillem de Castro 94, E-46003, Valencia, Spain.
[b]Instituto Universitario de Matemática Pura y Aplicada, Universidad Politécnica de Valencia, Camino de Vera 14, 46022 Valencia, Spain.

November 25, 2012

Abstract

This is a description of the PyNetMet package classes, their attributes and methods.

Keywords: Networks, Python, OptGene, SBML

1 Class Enzyme

The class enzyme defines a chemical reaction. The class should be called with at least one input which is a string containing a chemical reaction in OptGene format. The second input for this class is optional and can be used to indicate the pathway to which the reaction belongs. If none is given it is set to "GP" meaning general pathway.

The reaction should have the form:

$$\text{reaction name : } s_1\ \text{Sub}_1 + s_2\ \text{Sub}_2 + ... + s_n\ \text{Sub}_n \text{ -> } p_1\ \text{Prod}_1 + p_2\ \text{Prod}_2 + ... + p_n\ \text{Prod}_n$$

where the s_i are stoichiometric coefficients for the substrates Sub_i and p_i are stoichiometric coefficients for the products Prod_i. The symbol "->" indicates an irreversible reaction and can be replaced by "<->" if the reaction is reversible. It is important that the symbol ":" and "<->" or "->" are surrounded by spaces in the reaction string, otherwise and EnzError exception will be raised.

*daniel.gamermann@gmail.com

Enzyme objects can be negated, summed, compared with "==", subtracted and multiplied by numbers. The result of these mathematical operations is what one might expect intuitively from operations with chemical reactions. The negation of an Enzyme object will be the reversed version of the reaction. Addition of two reactions results in an effective reaction from the combined effect of both reactions. Subtraction is the addition of one reaction with the negated version of the second one. Multiply a reaction by a number and one obtains the reaction with every stoichiometric coefficient multiplied by this number. The comparison of reaction returns True or False if the reaction connects the same metabolites with the same stoichiometric coefficients. The following examples should illustrate this:

```
>>> from PyNetMet.enzyme import *
>>> enz1=Enzyme("re1 : A + 2 B -> C")
>>> enz2=Enzyme("re2 : D -> 2 C")
>>> enz3=0.5*enz2
>>> print enz3
re2*0.5 : 0.5 D  -> 1.0 C
>>> enz4=enz1+enz2
>>> print enz4
re1+re2 : 1.0 A + 2.0 B + 1.0 D  -> 3.0 C
>>> enz5=enz1-enz3
>>> print enz5
re1+re2*0.5_R : 1.0 A + 2.0 B  -> 0.5 D
>>> print enz5.issues_info
['Possibly eliminated: C']
>>> enz6=Enzyme("re3 : C -> E + A")
>>> enz7=enz1+enz6
>>> print enz7
re1+re3 : 2.0 B  -> 1.0 E
>>> print enz7.issues_info
['Possibly eliminated: C', 'Possible catalyzers: A']
```

A few things should be noted: these mathematical operations take little care whether the reactions are reversible or not, if one negates an irreversible reaction the result is the irreversible version of the reaction in the other direction. Moreover it eliminates superfluous metabolites: if one sums reactions where a metabolite appears in the products and substrates, it might eliminate this metabolite if the stoichimetric coefficients coincide. This elimination gets registered in the `issues_info` attribute of the reaction. In the same way a metabolite which is consumed by a reaction and latter produced by another is recognized as a possible catalyzer of the combined reactions.

The complete list of the enzyme object attributes is given below:

- name → a string indicating the name of the reaction.
- pathway → a string. The pathway to which the reaction belongs. If not given in the input, its value will be "GP".

- reversible → Boolean, indicates if the reaction is reversible.
- Nsubstrates → the number of metabolites in the left-hand side of the reaction equation.
- Nproducts → the number of metabolites in the right-hand side.
- metabolites → a list of all metabolites in the reaction.
- substrates → a list of the metabolites in the left-hand side of the equation.
- products → a list of the metabolites appearing in the right-hand side of the equation.
- issues → is a Boolean which indicates if there is any issue with the enzyme like no substrates or products, or if the same metabolite appears in both sides of the reaction, or others derived from modifying the reaction.
- issues_info → a list with the issues recognized in the reaction equation.
- stoic → a list of two lists. Each one of the two lists has the stoichiometric coefficients of substrates and products respectively.
- tup → a tuple with the number of substrates and products. If the reaction is reversible, the smaller number will be on the left.

The enzyme objects have the standard methods `__repr__` and `__str__` which return an string in OptGene format for the reaction and also the `__eq__` which compares if two reactions are the same (have the same substrates and products with the right stoichiometric coefficients) and it takes into account reversibility of the reactions. All other methods of the Enzyme class are listed in table 1.

2 Class Network

This class defines a Network object, a set of N nodes and edges connecting them. The network is defined by its adjacency matrix: a $N \times N$ matrix where each element ij is either one or zero, one meaning that there is a directed connection from node i to node j. In the particular case where the adjacency matrix is symmetrical, the network is undirected. The adjacency matrix is the input that must be given when calling the class. It should be a list of N elements where each element is a list with N elements that are either 0 or 1. One can also give a second input which is a list with names for the nodes.

The class also has a function called `stats` that gets as input a list of numbers and returns two floats, the average of the numbers in the list and their standard deviation.

As an example, lets create a 30 nodes random network.

Table 1: Methods of the Enzyme class.

Method	Input(s)	Output(s)	Description
connects	List	Boolean	True if the two metabolites given in the input list are connected by the reaction as substrate and product and False otherwise.
copy	None	Enzyme	Return a copy of the reaction.
has_metabol	String	Boolean	True if the metabolite in the input appears in the reaction. False otherwise.
has_product	String	Boolean	True if the metabolite in the input appears in the products list of the reaction. False otherwise.
has_product_rev	String	Boolean	True if the metabolite in the input appears in the products list of the reaction or in the substrate list if the reaction is reversible. False otherwise.
has_substrate	String	Boolean	True if the metabolite in the input appears in the substrates list of the reaction. False otherwise.
has_substrate_rev	String	Boolean	True if the metabolite in the input appears in the substrates list of the reaction or in the products list if the reaction is reversible. False otherwise.
make_irr	None	Enzyme	Returns the irreversible version of the reaction.
make_rev	None	Enzyme	Returns the reversible version of the reaction.
pop	Sring	None	This method eliminates from the reaction the metabolite given as input.
rev_reac	String (optional)	Enzyme	Returns the reversed (negated) version of the reaction. The name of the new reaction is the optional string given as input or the old name with an "_R" added, otherwise.
stoic_n	String	Float	Returns the stoichiometric coefficient for the metabolite given as input. Raises EnzError exception if the metabolite is no in the reaction.

```
>>> from random import random as rand
>>> from PyNetMet.network import *
>>> M = [[float(rand()>0.5) for i in xrange(30)] for j in xrange(30)]
>>> net = Network(M)
>>> print net # The network is different in every realization
Directed Network with:
 30.000000 Nodes
 440.000000 Links
>>> print sum(net.Cis)/net.nnodes # average clustering for the network.
0.780297363447
>>> print 1.*sum(net.kis)/net.nnodes # average number of neighbors per node
21.7
>>> # Lets create an undirected network from M:
>>> for i in xrange(30):
...  M[i][i] = 0
...  for j in xrange(i+1,30):
...   M[j][i]=M[i][j]
...
>>> net2=Network(M)
>>> print net2
Undirected Network with:
 30.000000 Nodes
 202.000000 Links
```

In the following the attributes of a Network object are described. Note that when mentioning a matrix, we refer to a list of lists.

- nnodes → an integer with the number of nodes in the network.
- nodesnames → a list with the names of the nodes. If not given as input, it will be set to numbers between 0 and `nnodes-1`.
- directed → a Boolean, indicating if the network is directed or not.
- links → a list of tuples, each tuple with two numbers, indicating all edges in the graph.
- nlinks → an integer with the total number of connections in the network. For an undirected network this number will be twice the actual number.
- linksin → a list where each element is a list of all in-neighbors of a node (edges coming from a given node).
- linksout → a list where each element is a list of all out-neighbors of a node (edges going to a given node).
- neigbs → a list where element i is the list of neighbors (all nodes with connection to or from) node i.

- kis → a list with the degree (number of neighbors) for each node.
- Cis → a list with the clustering coefficients for each node.
- CCs → a matrix with a list in each element. The element ij of this matrix is a list with the common neighbors of nodes i and j (intersection).
- uCCs → a matrix with a list in each element. The element ij of this matrix is a list with all the neighbors of nodes i and j (union).
- nCCs → a matrix $N \times N$ where each element ij is the topological overlap between nodes i and j.
- weight → a list with the average of each row of matrix nCCs.
- sd_wei → a list with the standard deviation for each element of weight.

In table 2 we describe the methods of a Network object.

To illustrate the use of some of the methods of this class, lets consider the network illustrated in figure 1. The adjacency matrix for this undirected network is given by:

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \quad (1)$$

Lets create the Network object for this network.

```
>>> M=[[0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
... [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
... [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
... [0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0],
... [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
... [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
... [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
... [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
... [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
... [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1],
... [0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1],
```

Table 2: Methods of the Network class. A * beside an input indicates that the input is optional. The two method missing in this list (distance and dictance_wp are for internal use of the calc_dists methods, they are an implementation of Dijkstra's algorithm.)

Method	Input(s)	Output(s)	Description
plot_nCCs	-List * -Integer * -String *	-Image file -Two new attributes	Creates a JPG image file with name given by the input string and size given by the two numbers in the input list. The image is a graphical representation of the matrix nCCs with the nodes ordered according to their topological-overlaps. The float input set the spacing for drawing a grid in the image. Also creates the network attributes CCord with the ordered matrix nCCs and names_ord with the nodes names in the new order.
plot_matr	-Matrix -List -List * -Integer * -String *	-Image file -Matrix	The optional arguments are the same as in plot_nCCs. This function makes the same plot, but for the matrix given in the input with its rows and columns reordered in the order given in the input list. The plot is a $N \times N$ rectangle where each position is painted in a gray-scale tone based on the value of the ij element of the input matrix. 1 is black, 0 is white.
kruskal	-Matrix -Boolean *	-Four new attributes	Uses the kruskal algorithm to create a dendrogram (tree) connecting the closest related elements in the matrix given as input. By default the closest related ij elements are those for which the ij element of the matrix is the smallest. Change the input optional Boolean to False if the biggest is desired. Creates the attributes tree, dists, branches and krusk_ord with the resulting tree, distances between branches, branches and ordering.
calc_all_dists	List of lists	Matrix	The input is a list where each element is the list of neighbors for each node (use either linksin, linksout or neigbs). It returns a matrix where the element ig corresponds to the shortest distance between nodes i and g. If a node could not be reached a "X" indicates it.
calc_all_dists_wp	List of lists	-Matrix -Matrix	Same as calc_all_dists but returns first matrix where each element is a list with the shortest path between two nodes.
calc_dist	-List of lists -Integer	-List	Same as calc_all_dists but for a single node, whose index is the integer entered as input.
calc_dist_wp	-List of lists -Integer	-List of lists -List	Same as calc_all_dists_wp but for a single node, whose index is the integer entered as input. The list of lists returned is the list with the paths between the node and all others.
components	None	Three new attributes	This method creates a new attribute disc_comps where each element is a list with the indexes of nodes belonging to each connected component of the network. The other attributes created by this method, distances, paths, are the result of calling the method calc_all_dists_wp with neigbs as input.

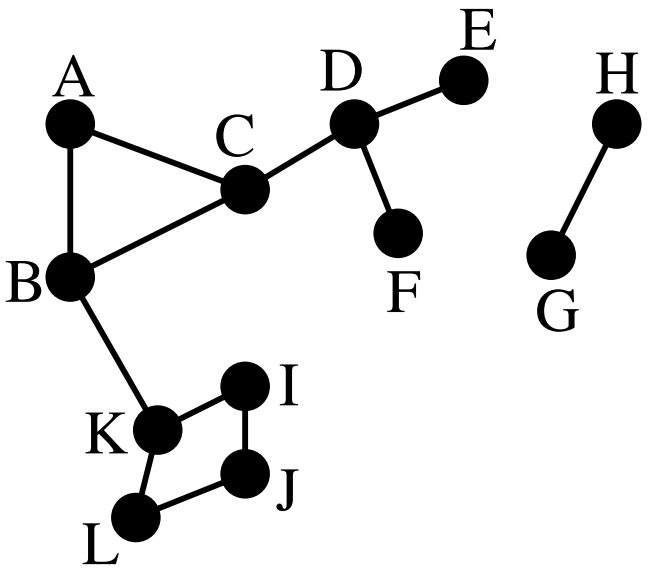

Figure 1: Simple network with two disconnected components.

```
... [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0]]
>>> nnames = [chr(65+i) for i in xrange(12)]
>>> net = Network(M,nnames)
>>> from random import shuffle
>>> nord = range(12)
>>> shuffle(nord)
>>> net.plot_matr(net.nCCs, nord, output="graf1.jpg")
>>> net.plot_nCCs(output="graf2.jpg")
>>> net.kruskal(net.nCCs, minimo=False)
>>> net.plot_matr(net.nCCs, net.krusk_ord, output="graf3.jpg")
>>> print [net.nodesnames[ele] for ele in nord]
['I', 'D', 'C', 'A', 'H', 'B', 'E', 'L', 'J', 'F', 'K', 'G']
>>> print net.names_ord
['G', 'H', 'A', 'B', 'J', 'K', 'I', 'L', 'C', 'D', 'E', 'F']
>>>> print [net.nodesnames[ele] for ele in net.krusk_ord]
['D', 'F', 'E', 'C', 'A', 'B', 'J', 'K', 'I', 'L', 'G', 'H']
```

In this example we produce tree different plots of the nCCs matrix. One with a random ordering of the nodes, another one with the algorithm implemented in the plot_nCCs method and a third with the ordering given by the dendrogram obtained with the kruskal algorithm. The three graphics generated can be seen in figure 2. The images that this class produces depend on the classes Image and ImageDraw from the Python Imaging Library (PIL) package.

From figure 1 it is clear that the network has two disconnected components.

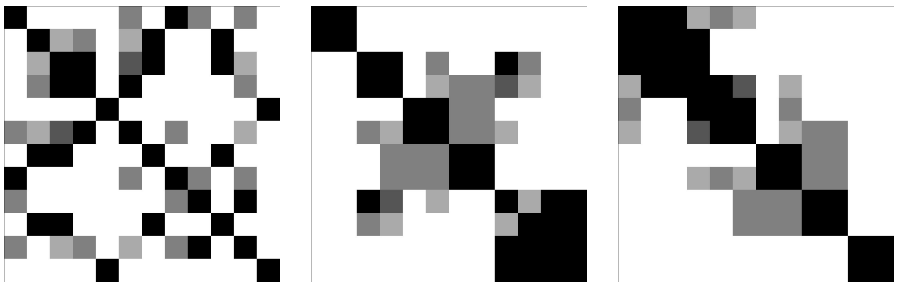

Figure 2: Image files graf1.jpg, graf2.jpg and graf3.jpg obtained with the methods plot_matr and plot_nCCs of the Network class.

These can be identified with the components method of the Network class.

```
>>> net.components()
>>> net.disc_comps
[[0, 1, 2, 3, 4, 5, 8, 9, 10, 11], [6, 7]]
>>> nodes_in_comps=[[net.nodesnames[ele] for ele in comp] for comp in net.disc_comps]
>>> print nodes_in_comps
[['A', 'B', 'C', 'D', 'E', 'F', 'I', 'J', 'K', 'L'], ['G', 'H']]
```

One can see that the components method generated a list with two elements, each element is a list with the indexes corresponding to the nodes in each disconnected component of the network. Moreover, this method generates two other attributes containing the distances and paths in between any two nodes of the network:

```
>>> #Distance of node J(9) to F(5):
>>> print net.distances[9][5]
6
>>> #Path of node J to F:
>>> print [net.nodesnames[ele] for ele in net.paths[9][5]]
['J', 'I', 'K', 'B', 'C', 'D', 'F']
```

3 Class Metabolism

This class reads, stores and analyze a metabolic model. There are three forms of input for this class: a string containing the name of the file with a model in OptGene, in SMBL or introducing lists with the metabolic model information directly.

By default the class expects a file in OptGene format, so if this is the case, only one input is needed:

```
>>> from PyNetMet.metabolism import *
>>> org = Metabolism("model.txt")
```

When reading the reaction list from a OptGene file, badly written reactions (reactions where the ":" or "->" symbol cannot be identified) will be ignored.

If one has a metabolic model in SBML format, this must be indicated by the optional input filetype:

```
>>> from PyNetMet.metabolism import *
>>> org = Metabolism("model.xml", filetype="sbml")
```

The third way to call this class is to give as input lists containing strings with the reactions, constraints, external metabolites and objective function, all in OptGene format, and set the optional input fromfile to False indicating that the model is not coming from a file:

```
>>> from PyNetMet.metabolism import *
>>> reac_list = ["reac1 : A_ext -> A", "reac2 : A -> B", "reac3 : A -> C",
... "reac4 : B + C -> D", "reac5 : D -> D_ext"]
>>> ext_list = ["A_ext", "D_ext"]
>>> constr_list = ["reac1 [0, 10]"]
>>> obj_list = ["reac5 1"]
>>> org = Metabolism("model_name", reactions=reac_list, constraints=constr_list,
... external=ext_list, objective=obj_list, fromfile=False)
>>> print org
# Reactions:7
# Metabolites:6
```

in this example we defined a very simple list with 5 reactions. The first string of the input can, in this case, be used to give a name to the model. Note that although we gave 5 reactions in the input, the class added two extra reactions, which are transport reactions. If one prints all reactions in the model:

```
>>> print "\n".join([str(ele) for ele in org.enzymes])
reac1 : 1.0 A_ext  -> 1.0 A
reac2 : 1.0 A  -> 1.0 B
reac3 : 1.0 A  -> 1.0 C
reac4 : 1.0 B + 1.0 C  -> 1.0 D
reac5 : 1.0 D  -> 1.0 D_ext
A_ext_transp :  <-> 1.0 A_ext
D_ext_transp :  <-> 1.0 D_ext
```

The class interprets reactions with blank products or substrates as transport reactions (blank means an actual space in place of a substrate or product, not the simple absence of it) and sets its metabolites to external metabolites directly. Alternatively the former example could be introduced as:

```
>>> from PyNetMet.metabolism import *
>>> reac_list = ["reac1 :  -> A", "reac2 : A -> B", "reac3 : A -> C",
```

```
... "reac4 : B + C -> D", "reac5 : D ->  "]
>>> ext_list = []
>>> constr_list = ["reac1 [0, 10]"]
>>> obj_list = ["reac5 1"]
>>> org = Metabolism("model_name", reactions=reac_list,
... constraints=constr_list, external=ext_list, objective=obj_list, fromfile=False)
>>> print org
# Reactions:5
# Metabolites:4

>>> print org.external
["A", "D"]
```

Below we give the complete list of attributes for an object of this class:

- file_name → the name of the input file (or model).
- enzymes → the list of all reactions (Enzyme objects) in the model.
- dic_enzs → a dictionary associating a reaction name to its index in the lists.
- nreacs → the total number of reactions.
- reac_irr → the list of indexes of the irreversible reactions.
- reac_rev → the list of indexes of the reversible reactions.
- nreac_irr → the number of irreversible reactions.
- nreac_rev → the number of reversible reactions.
- mets → the list of all metabolites in the model.
- dic_mets → a dictionary associating each metabolite name to its index in the lists.
- nmets → the total number of metabolites.
- pathnames → the list of comments[1] found in the model (usually comments are used to organize reactions into pathways).
- pathways → a list where each element is a list with the indexes of the reactions read after each comment.
- reac_per_met → a list where each element is the list of names for the reactions where the metabolite with the correspondent index in the list appears.

[1] In the OptGene file comments are indicated by the "#" character.

- reacs_per_met → list with the number of reactions in which each metabolite appears (length of each element in the reac_per_met list).
- M → the adjacency matrix for the metabolites.
- net → the metabolite's network (Network class object for the above adjacency matrix).
- transport → the list of all transport reactions indexes.
- external → the raw data in OptGene format for the external metabolites.
- external_in → the list of metabolites that can come inside the cell from the outside.
- external_out → the list of metabolites that the cell transports to the outside.
- constraints → the raw data in OptGene format for the constraints.
- constr → a list with constraints for each reaction.
- objective → the raw data in OptGene format for the objective (for FBA optimization).
- obj → list with the name of each reaction in the objective with its correspondent coefficient.

In table 3 we describe the methods of this class.

Note that the dump method can be used as a translator. If one has a model in OptGene format and wants to obtain the SBML version of it it is straightforward:

```
>>> from PyNetMet.metabolism import *
>>> org = Metabolism("model.txt")
>>> org.dump(fileout="model.xml",filetype="sbml")
```

Most attributes in this class come as lists. One should keep in mind that an index is associated to each metabolite and reaction corresponding to the position of the metabolite or reaction in the lists `mets` and `enzymes`, respectively. To find out the index of a particular metabolite or reaction one can use the index method of the list or use the `dic_mets` and `dic_enzs` attributes, respectively.

4 Class FBA

The FBA class uses the information from a Metabolism object to perform flux balance analysis (FBA) in the model. For performing the optimization, this class uses the python package PyGLPK. This class can be called with a single input which is a Metabolism object. A second optional input (eps) can be given, which is the minimum value a flux may have not to be considered zero[2],

[2]This has been introduced to avoid some numerical issues when calculating essential reactions ans sensibilities.

Table 3: Methods of the Metabolism class. A * beside an input indicates that the input is optional. The missing methods in this list (prepare_opt and prepare_sbml read the and gather information from the input files and are for internal use of the class.

Method	Input(s)	Output(s)	Description
calcs	None	None	Calculates all attributes of the object based in the enzyme list and raw data attributes. This should should only be used if one modifies the enzyme list or raw data attributes manually.
add_reacs	List	None	Adds to the model (modifying the needed attributes) the reactions whose strings are the elements of the input list.
pop	Integer	-New attribute	Removes the reaction whose index is the input integer from the model (modifying the needed attributes).
dump	-String * -String * -Boolean * -Dictionary * -Boolean * -Boolean *	File	Writes a file with the model. By default the written file is in OptGene format. The first optional string (fileout) is the name of the output file, the second optional string (filetype) can be set to "sbml" in order to change the default output OptGene to SBML. All other optional inputs are for the SBML file: The first Boolean (printgenes) should be set to True if one wishes that each reaction has a note indicating its correspondence with a gene, in this case the optional dictionary (dic_genes) should have this correspondence. The second Boolean (alllimits) should be set to True if one wishes the SBML to write constraints for all reactions and not only the ones in the constr list and the last Boolean (allobjs) must be set for True if one wishes a objective coefficient tag written for all reactions, as well.
M_matrix	-Boolean *	-Matrix	Returns the adjacency matrix for the metabolites (the ij element equal to 1 meaning that metabolite i is connected with j by some reaction. If one wishes the undirected version of the network, set the optional input to True.
M_matrix_reacs	-Boolean *	-Matrix -List	Returns the adjacency matrix for the bipartite network formed by metabolites and reactions. Also a list containing the name of each node in the network.
bad_reacs	None	None	Filters reactions belonging to disconnected components of the network and reactions where at least one substrate and one product appear only once in the model. It stores the poped reactions in the new attribute poped.
write_log	-Integer *	File	Writes a file with information on the model and connectivity of metabolites and reactions. The optional Integer input indicates how many metabolites and reactions should be listed in the file lists.

by default it is set to 10^{-10}.

In the following example, the file model.txt is the result of dumping (`org.dump()`) the 5 reaction model given as example in the former section.

```
>>> from PyNetMet.metabolism import *
>>> from PyNetMet.fba import *
>>> org = Metabolism("model.txt")
>>> fba = FBA(org)
>>> print fba
                          reac2 ---->  5.000000000
                          reac3 ---->  5.000000000
                          reac4 ---->  5.000000000
                          reac5 ---->  5.000000000
                          reac1 ---->  10.000000000

Flux on objective                    : 5.000000000
Reactions with flux    (flux>eps)    : 5
Reactions without flux (flux<eps)    : 0
Solution status: Optimal
```

The attributes of this class are:

- reacs → List with the reactions (Enzyme objects).
- nreacs → Total number of reactions.
- reac_names → List with the names of all reactions.
- mets → List with all metabolites.
- nmets → Total number of metabolites.
- ext_in → List of external metabolites that can enter the cell.
- ext_out → List of metabolites that can leave the cell.
- Mstoic → List of tuples with the non-zero elements of the stoichiometric matrix. Each tuple has first the index of the metabolite, secondly the index of the reaction and finally the stoichiometric coefficient.
- constr → List of tuples with the constraints to be applied. Each tuple indicates the lower and upper value for the flux. None indicates no limit.
- flux → List with the fluxes for all reaction from the last optimization.
- Z → Value of the flux in the objective.
- obj → List of tuples for the objective. Each tuple has the reaction name and objective coefficient.
- eps → Precision (value under which a flux is considered to be zero.

- lp → Linear problem. Pyglpk object with the FBA stored in it.

Table 4 contains the description of the methods in this class.

This class has one underscore method, the `__sub__` which defined the subtraction of two FBAs. This method actually compares two different FBA, returning a string with four columns, the first with the name of each reaction, the second and third with the flux of the reaction in each one of the FBAs and the third one with the realative difference in the flux ($100\frac{\nu_1-\nu_2}{\nu_1}$), in the case where the first flux is zero and the second is not, it return "NA" in this column.

5 Bugs and Feedback

These classes have been thoroughly tested in a debian linux system using python 2.7. Some functions have already been tested in a windows system running python 2.7 via idle with no conflicts found until now.

Please email any detected bugs, suggestions or your feedback to the author.

6 License

PyNetMet is released under the GNU GENERAL PUBLIC LICENSE. See COPYING and README files for further information.

Table 4: Methods of the FBA class. A * beside an input indicates that the input is optional. The missing methods in this list, check_flux and sum_flux_met should always return zero and have been created by debugging purposes.

Method	Input(s)	Output(s)	Description
print_flux	Integer	String	Will return a string with the reaction and flux correspondent to the input integer.
fba	None	None	Prepares the linear problem and performs the fba. It is automatically called when the object is created. Use it as a reset if the solution changes after several calls of essential or shadow.
shadow	-Integer -Float * -Boolean *	-Float	Calculates the derivative of the flux in the objective with respect to the flux in the reaction correspondent to the input integer. The optional float is the proportion it changes the original flux in order to calculate the derivative. If the original flux is zero, the method tries to force a flux of 0.0001 times the proportion. The boolean input is set to True by default. It means that the result is multiplied by the absolute value of the ratios of fluxes in the reaction and in the objective.
essential	Integer	List	Checks if a reaction is essential for producing flux in the objective by forcing its flux to be zero. Returns a two elements list the first is a Boolean indicating if the reaction is essential or not and the second is a number between zero and 1 indicating the proportion that the flux on the objective has been affected by the removal of the correspondent reaction.
max_min	-Integer -Float *	List	Calculates maximum and minimum of each reaction in each direction given a fixed value of the objective function. The optional float indicates the amount of objective to be fixed.