PyNetMet: Python tools for efficient work with networks and metabolic models

Daniel Gamermann, Arnau Montagud, Ramon Jaime Infante, Julian Triana, Javier Urchueguía and Pedro Fernández de Córdoba

Abstract

The complexity of genome-scale metabolic models and networks associated to biological systems makes the use of computational tools an essential element in the field of systems biology. Here we present PyNetMet, a Python library of tools to work with networks and metabolic models. These are open-source free tools for use in a Python platform, which adds considerably versatility to them when compared to their desktop similar. On the other hand, these tools allow one to work with different standards of metabolic models (OptGene and SBML) and the fact that they are programmed in Python opens the possibility of efficient integration with any other existing Python package. In order to illustrate the most important features and some uses of our software, we show results obtained in the analysis of metabolic models taken from the literature. For this purpose, three different models (one in OptGene and two in SBML format) were downloaded and throughly analyzed with our software. Also, we performed a comparison of the underlying metabolic networks of these models with randomly generated networks, pointing out the main differences between them. The PyNetMet package is available from the python package index (`https://pypi.python.org/pypi/PyNetMet`) for different platforms and documentation and more extensive illustrative examples can be found in the webpage `pythonhosted.org/PyNetMet/`.

1 Introduction

Nowadays, the genome-scale reconstruction of metabolic models has become one of the corner stones of systems biology. Reconstructed metabolic models have been used in a wide range of applications, such as the study of metabolism regulation [2], [19], determination of the optimal conditions for growth or prediction of maximum yield of biomass in a determined organism [7], the search for potential sites for metabolic engineering [17], the production of biofuels [16] and even in the reconstruction of phylogenetic trees [9]. One of the most important computational tools for the analysis of metabolic models is the flux balance analysis (FBA) [25], which consists in the determination of a possible consistent solution for the fluxes in each one of the reactions present in a given model, that optimizes some given objective.

A particular way to study genome-scale metabolic models is to analyze their underling networks. The simplest example of such networks is to define each metabolite present in a metabolism as a network node, and assign connections in between the nodes based on the connection of the respective metabolites through chemical reactions. Such networks have been thoroughly studied in the literature [11], [21], [20].

Typical genome-scale metabolic models comprise around thousand different metabolites and chemical reactions and, correspondingly, the underlying metabolic networks are complex structures with around one thousand interconnected nodes. Other typical networks studied in systems biology, like protein protein interaction, can be even bigger. The analysis of these complex structures would be nearly unfeasible without the aid of modern computers. There are different available software for performing FBA on metabolic models like the COBRA toolbox, originally developed for MatLab [24], but now also available for Python, or the OptFlux software [22] and also several software for the analysis of networks. Unfortunately, many of the available software have drawbacks.

Concept Press Ltd.

For instance, there are two different standards for the storage of metabolic models: the SBML [10] and OptGene (also known as BioOpt) [6] formats, and the available software either use one or another, but not both. On the other hand, some software are not free (like MatLab) or are desktop software (like Cytoscape) which limits their uses and integrability with other bioinformatic tools. Also, in order to study different aspects of a given metabolic model, one has to use different software.

Software that perform FBA, which is the case of the COBRA toolbox or the PyCes [18] package, do not have the tools to analyze the underlying metabolic network represented by nodes and edges, while software like gephi [3], cytoscape [4] or pajek [12] that deal with complex networks, do not have the tools necessary to perform flux analysis over metabolic models. Integration between these softwares is extremely difficult when not unfeasible, moreover many of these softwares use different file standards in order to store the models.

In this article we present a series of tools, which have been developed in Python, for dealing with chemical reactions and analyzing networks and metabolic models. Python is a free, open-source, modular, object oriented programming language. Open-source libraries boost the development of bioinformatics by allowing researchers to develop new tools and applications over modules already existent. Moreover, modular programming languages like Python allow easy and efficient integration of its modules with other libraries and software (which is hardly done with desktop applications). In the last years, hundreds of bioinformatic related libraries have been written for Python, like the Biopython package [5] which contains various standards used in bioinformatics and allows the direct connection with different biological databases, the pysb [14], mstacommander [8] and many others.

The package presented here is called PyNetMet (from Python Network Metabolism), it comprises four classes called `Enzyme`, `Network`, `Metabolism` and `FBA`. PyNetMet can be downloaded from the Python Package Index (`pypi.python.org/pypi/PyNetMet`) where one can find installation files for two different operational systems: a windows installation file or a Linux source file which can be used in any UNIX based system (Linux or Mac). In the next section we describe the four classes contained in the PyNetMet package and in section 3 we comment on results obtained by our software in the analysis of three different published genome-scale metabolic models.

2 Software description

The package PyNetMet consists of four classes: `Enzyme`, `Network`, `Metabolism` and `FBA`, all fully programmed in Python 2.7 language. The class `Enzyme` has no dependencies, it defines a new type of object that represents a chemical reaction. Class `Network` has a single dependency (for two specific functions) which is the Python Imaging Package (PIL), for making plots representing the clustering of nodes. The Class `Metabolism` depends on the classes `Enzyme` and `Network`, and class `FBA` depends on the class `Metabolism` and on the Python library Pyglpk (which contains tools for solving the associated optimization linear problem).

For a complete list of all attributes and methods of the classes, more detailed examples of use and a short tutorial please refer to the manual that accompanies the PyNetMet distribution that can be downloaded from the python package index (`https://pypi.python.org/pypi/PyNetMet`), or the documentation webpage (`pythonhosted.org/PyNetMet/`).

In order to use each class, one just need to import it as a Python module. The examples commented in section 3 can be reproduced following the the commands given in the PyNetMet documentation webpage (`pythonhosted.org/PyNetMet/example.html`) or running the script from the supplementary materials. In our case they were executed in a computer running under a Linux operating system with a icore7 Intel processor and 6 Gb of RAM memory, but the python module is also compatible with windows and mac systems, given that a python interpreter is installed with the necessary above mentioned dependencies.

Next we briefly describe each class in the package and a few definitions and algorithms related to them.

2.1 Enzyme

The class `Enzyme` defines a new type of object that represents a chemical reaction[1]. `Enzymes` will be the main objects used to build the `Metabolism` object later on. Its obligatory input is a string containing a reaction written in OptGene format (ex: the string `"reac1 : A + 2 B -> C"` defines an irreversible reaction called reac1 where one molecule of metabolite A interacts with two molecules

[1] Although enzymes are not chemical reactions, enzymes catalyze the chemical reactions and it is a common practice in metabolic modeling to associate each chemical reaction to an enzyme and to name each chemical reaction after this enzyme using, for example, an EC number.

of metabolite B creating one molecule of metabolite C). When defining the object one can also give an optional input, also a string, which will be used to indicate the pathway name of a particular reaction. An `Enzyme` object has attributes that allow an easy verification of the reaction's substrates, products, reversibility, etc.

The class `Enzyme` has no counterpart in other programming languages or software. It allows the storage of chemical reactions as computer variables such that one is able to sum, subtract or multiply (by numbers) these objects in order to create new (lumped) reactions, or to run buckles over a list of these objects in order to filter them or perform complex analyses.

2.2 Network

The `Network` class defines a graph (collection of nodes and edges) and contains many classical graph theoretical algorithms for its analysis. It should be initiated with one obligatory input and an optional one. The input for this class is the $N \times N$ adjacency matrix which defines the network (N is the number of nodes in the network) and the optional one a list with the node's names. The adjacency matrix, M, is a list of N elements, where each element is again a list with N elements, each element of this latter list being 0 or 1. If $M[i][j]$ is 1, it means that node i has a directed connection to node j. If the matrix M is symmetric, the network is undirected, meaning that there is no distinction between a link from node i to j or from node j to i. Otherwise, the network is interpreted as a directed graph, where the connections have an incoming and outgoing node.

Every node in a network can be characterized by some parameters. First, the node's degree is the number of connections it has to other nodes. Another attribute of a node is its clustering coefficient. It is defined by:

$$C_i = \frac{2E_i}{k_i(k_i-1)} \tag{1}$$

where k_i is the degree of node i, and E_i is the number of connections between the neighbors of node i. The average clustering of a network can be calculated straightforward by averaging the values in the list containing the nodes' clustering.

Next, we define the topological overlap (O_{ij}) between two nodes according to [20]:

$$O_{ij} = \frac{V_{ij} + \begin{cases} 1 & \text{, if } i \text{ connected to } j \\ 0 & \text{, otherwise} \end{cases}}{\min(n_i, n_j)} \tag{2}$$

where V_{ij} is the number of common neighbors between nodes i and j and $\min(n_i, n_j)$ is the minimum between the number of neighbors of nodes i and j.

The network's average clustering is an important parameter in characterizing a Network. Different classes of networks have nodes with different tendencies to cluster together defining functional subnetworks [20]. Using the nodes clustering coefficients and their topological overlap, one is able to define algorithms that will organize the network's nodes according to their correlation.

In [20] a method for grouping the nodes in clusters is proposed basically by constructing a dendrogram (maximum spanning tree) with the values of the topological-overlap matrix (O_{ij}). This tree can be constructed with the Kruskal algorithm, which is implemented in the Network class. Another interesting method for ordering the nodes is proposed in [23]. Although this later method has many improvements with respect to the dendrogram one, it is based on a Monte-Carlo simulation and given the size of the networks and the amount of Monte-Carlo steps needed in order to perform a proper simulation, the computation is usually costly. Here we propose a yet different method which is computationally more efficient and returns results at least as good as the dendrogram method.

The objective of the following algorithm is to reorder the nodes in the adjacency matrix (or the topological overlap one), such that nodes close to each other are correlated in the sense that they share neighbors which are also correlated among them, obtaining in this way an ordering where nodes belonging to common clusters are nearby to each other. So the output of this algorithm will be a list with a new ordering of the nodes. The algorithm follows the following steps:

(1) Choose any node i to start with. Add it to the ordering.

(2) From node i, find the node j for which χ^2_{ij} defined below is minimum:

$$\chi^2_{ij} = \sum_{k \in E'} \frac{1}{\max(\epsilon, C_k)} \left(\frac{O_{ik} - O_{jk}}{O_{ik} + O_{jk}} \right)^2 \tag{3}$$

where E' is the set of all nodes that have not been added to the ordering.

(3) Add node j to the ordering.

(4) Set node j as i and repeat the process from step (2) until the set E' is empty.

The use of ϵ and the function $\max(\epsilon, C_k)$ is to avoid a division by zero in the case that node k has clustering coefficient equal to zero and so to have a stable algorithm. The parameter ϵ should be very small (smaller than the value of the smallest non-zero C_k in the network). In performing the simulations the value is set to $\epsilon = 0.00001$, changing this value, as long as it is sufficiently small, has no effect in the obtained ordering. The algorithm is implemented in the `Network`'s class method `plot_nCCs`. Figure 1 shows examples of plots obtained for three metabolic models by three different ordering algorithms: without any ordering, with the Kruskal algorithm and with the algorithm described here.

2.3 Metabolism

This class defines an object with a full metabolic model. The metabolic model can be given as input in three different ways. By default one can use a single input which is a string containing the file name (with path) of a metabolic model in OptGene format. Alternatively, one can use a file in SBML format and finally one can define lists containing reactions, constraints, external metabolites and objective function directly from the command line and use them as input for the class. So, this class works either as a parser for OptGene or SBML file formats or as a platform to construct new metabolic models from scrap.

This class has also the `dump` method, that allows one to write an output file with the stored model either in OptGene or SBML file formats. This resource allows the class to be used as a translator between OptGene and SBML file formats, for one can load the model in one format and dump it in the other format. This feature is an advantage with respect to most software used in metabolic analysis which are usually compatible with only one format for the metabolic models.

When parsing a SBML file, the PyNetMet package will look for the SBML tags `<reaction>` and `<species>` and inside each element of `<reaction>` it will generate a list of substrates and products based on the elements defined by the tags `<listOfReactants>` and `<listOfProducts>`. As long as different versions of SBML maintain these tags as standards, and their attributes, the current version of PyNetMet will be able to read SBML. If these standards are changed in the future, a new version of PyNetMet will be needed in order to keep its functionality as a SBML parser.

The main attribute from this class is its `enzymes` list, which contains all chemical reactions in the model. This list can be altered either directly (which is not advisable since other attributes of the class will not be automatically updated unless one calls the `calcs` method afterward), or by making use of the `bad_reacs`, `add_reacs` and `pop` methods. The possibility to change the `enzymes` list, gives a new functionality to the class `Metabolism`, namely it can be used as a platform to produce *in silico* mutants and perform metabolic engineering studies and simulations.

The use of this class together with the `Network` and `FBA` classes offers rich resources for an extensive analysis of any metabolic model.

2.4 FBA

The `FBA` class offers tools for performing flux simulations and analysis of a metabolic model. It has methods defined which are based on the FBA for studying essential reactions, sensibility of the objective function with respect to any given reaction, comparison of different realizations of the FBA, among others.

To call this class one must give one obligatory input, which is a `Metabolism` object with a metabolic model. It can also receive two optional inputs which are the precision (`eps`, value under which a flux is considered zero which by default is set to 10^{-10}) and a choice of maximizing or minimizing the objective (the default choice is maximize).

3 Applications

In this section we exemplify some uses of our tools by analyzing real metabolic models taken from the literature. We chose three models to work with, the first is the iSyn811 model of *Synechocystis sp. PCC6803* [16]. The second is the metabolic model iCM925 for the organism *Clostridium beijerinckii* NCIMB 8052 [15] and last is the model iAK692 for *Spirulina platensis* C1 from [13]. All these models are available from the journals as supplementary materials, the first one in OptGene format and the other two in SBML format. These have been downloaded and saved in a working folder and can be directly accessed by PyNetMet tools.

With a few lines of code, one is able to do complex analysis of any metabolic model (see supplementary ma-

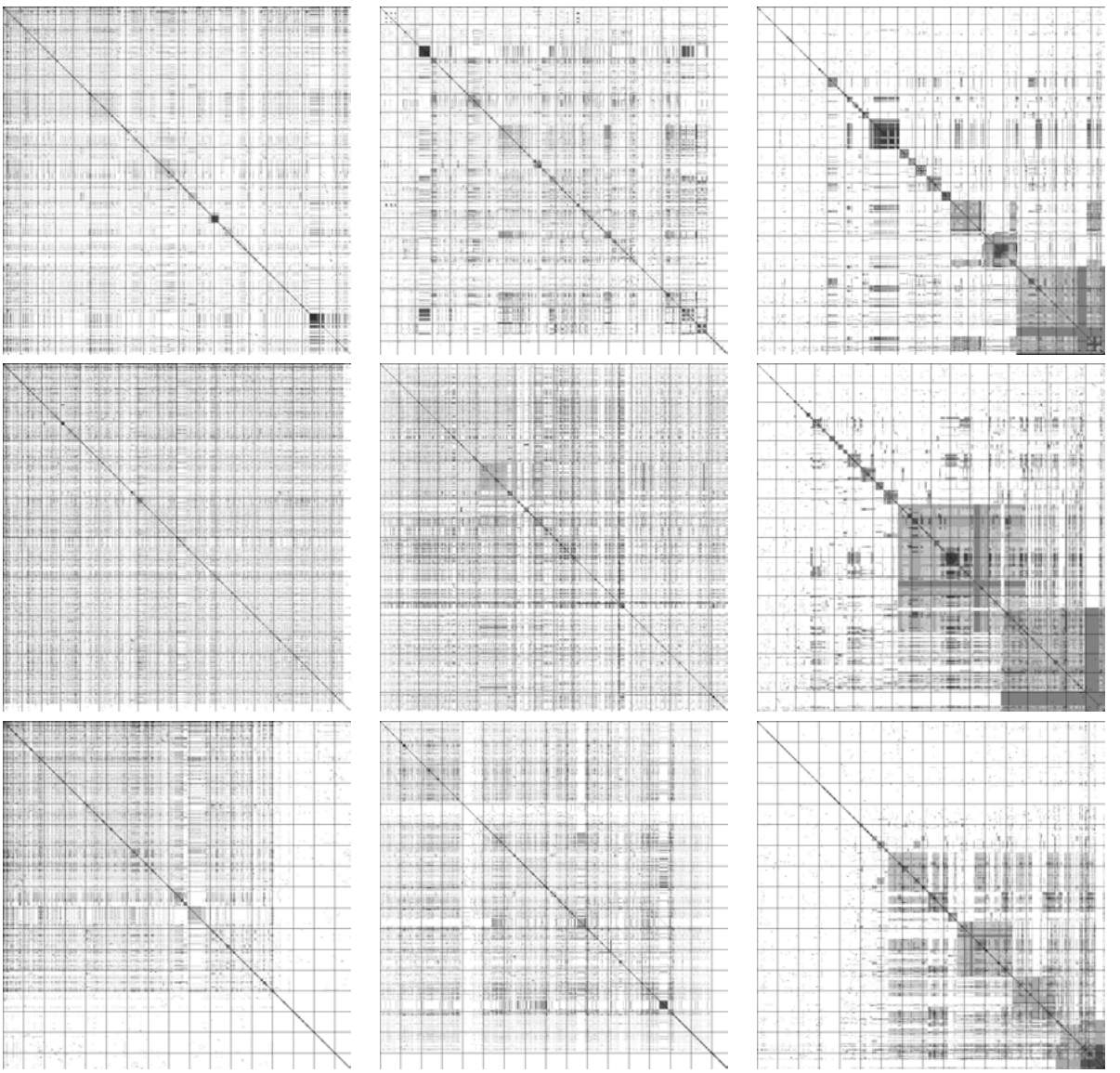

Figure 1: Plots for the topological overlap of metabolites. The first, second and third rows refer to the plots obtained from the three models analyzed: *i*Syn811, *i*CM925 and *i*AK692, respectively. The plots in the first column are for an arbitrary ordering of the metabolites, in the second column an ordering is obtained via the Kruskal algorithm and in the third column the ordering is obtained by the algorithm implemented in the `plot_nCCs` method of the `Network` class.

terials). We present in table 1 the results of calculating the average clustering of the networks representing each one of the metabolic models, the number of disconnected metabolites in each model and the average distance between any two metabolites in the network.

The two network averages shown in table 1 show two important features of biological networks that differentiate them from random networks and other kinds of networks, namely their relatively high (tough disperse) average clustering and the small world [1] property that refers to the fact that the average distance between two nodes scales very slowly with the size of the network. To

Model	$\bar{C}$	σ_C	#D. M.	$\bar{\ell}$	σ_ℓ	Essential
*i*Syn811	0.166	0.224	13	3.248	1.161	221
*i*CM925	0.245	0.244	2	2.790	0.703	166
*i*AK693	0.200	0.244	42	3.053	1.001	249

Table 1: Results from the models analysis. The network parameters were calculated for the undirected version of the networks obtained from connecting metabolites that appear as substrate and product in the model's reactions. The $\bar{C}$ column indicates the average clustering coefficient, the #D. M. column indicates the number of nodes disconnected from the main component of the network, the $\bar{\ell}$ column indicates the average distance between two nodes in the network and the column Essential has the number of essential reactions for producing the objective function. The σ's are the standard deviations for the two averages.

show this, we generated 100 random networks with the same number of nodes and links that the synechocystis metabolic network, and we show in table 2 a comparison between the random networks and the *i*Syn811 network. To create the random networks a set of nodes is generated. Then, two nodes are selected at random and connected. The random connecting process is repeated until the network has the desired number of links. From table 2 it is clear that the average clustering in random networks is basically zero and the average distance between two nodes is two times larger than in metabolic networks.

Model	$\bar{C}$	σ_C	$\bar{\ell}$	σ_ℓ
iSyn811	0.166	0.224	3.248	1.161
Random	0.005	0.043	6.213	1.557

Table 2: Comparison between metabolic and random networks. Comparison between the clustering coefficient and average distance between two nodes in the metabolic network of the *Synechocystis sp. PCC6803* and the average of 100 random networks.

Apart from the network analysis of the models, one can use the methods in class FBA. Just by calling the class with a metabolic model as input one can directly obtain the FBA result by printing the class object. The result is a string listing the reaction names and their respective fluxes ordered, by default, according to these fluxes.

A straight forward method to analyze a FBA is the `essential` method, which checks if a reaction is essential for producing flux in the objective function. When called, it returns one Boolean value stating if the reaction is essential or not and a second value which informs of the relative change in the objective flux with the input reaction removed. In table 1 one can also find the number of essential reactions in each metabolic model.

Other methods to analyze a FBA are the `shadow` and `max_min` methods which work in a similar way. In each method one has to use as input an integer indicating a reaction number. The shadow method has two other optional inputs, the first one indicates the change in the original flux in order to calculate the derivative (the result should be independent of this choice, since the problem is linear) and the second indicates if one wishes the relative change or the absolute change.

The method `max_min` has the `fixobj` optional input. Its algorithm fixes the flux in the objective function to its original value times the value in `fixobj` (this should always be a value between 0 and 1). Then it sets the input reaction as objective and minimizes it, then maximizes it in its natural direction ($S \rightarrow P$) and then again minimizes it and maximizes it in the reversed direction ($S \leftarrow P$). It returns a two element list, each element is a tuple with the value of the flux in the reaction minimized and maximized in the direct direction and in the reversed direction, respectively. If the reaction is irreversible or there was no feasible solution, it returns the string "X". This method can be used in analysis of maximal and minimal throughput of a reaction. So, it determines limits in knockdowns or over-expression, since it shows the flux range at which a determined reaction can occur. Since a reaction rate is proportional to the expression of some enzyme, suppression or over-expression of the correspondent gene will have direct (proportional) effect over the reaction flux. Instead of using this method to determine limits, it is also possible to directly redefine the constraints of a given reaction (altering the values in `fba.constr`) in order to fix the flux of a given reaction.

Concept Press Ltd.

The implementation of these examples, which do not intend to exhaust the uses of the PyNetMet tools and their functionality, can be found in the `PyNetMet` documentation webpage (`pythonhosted.org/PyNetMet/example.html`).

4 Overview

We have presented the PyNetMet package which contains four classes (`Enzyme`, `Network`, `Metabolism` and `FBA`) intended to facilitate the analysis, work, curation and construction of networks and metabolic models. These tools allow one to work with metabolic models in either standard (OptGene and SBML) and to easily convert one to another. The `Metabolism` class can be used as a platform to produce variants of any model (*in silico* mutants) by producing knock-ins or knock-outs with the `add_reacs` and `pop` methods, respectively and studying its effects straightforwardly with the class `FBA`.

These tools are in the format of Python modules, which allow the researcher to integrate them with any other Python resource available. They are also open-source and free software which allows one to develop new tools using these as building blocks.

Author Biography

Daniel Gamermann (contact author)

Department of Physics, Universidade Federal do Rio Grande do Sul (UFRGS) - Instituto de Física.

danielg@if.ufrgs.br

D. Gamermann received his bachelor degree from Universidade Federal do Rio Grande do Sul in 2002. In 2005 he obtained the German Diplom degree from the University of Bonn and in 2010 his Ph.D degree from the University of Valencia.

Arnau Montagud

Instituto Universitario de Matemática Pura y Aplicada, Universidad Politécnica de Valencia. Currently at INSERM U900, Institut Curie, Paris, France

armontag@mat.upv.es

A. Montagud received his Grade in Biology in 2006, and his Molecular biology Master degree in 2007, from the Universitat de València (UV). He defended his PhD dissertation in 2012 at Universitat Politècnica de València

Ramon Jaime Infante

Universidad Pinar del Río "Hermanos Saíz Montes de Oca", Martí 270, 20100, Pinar del Río, Cuba

ramon@info.upr.edu.cu

R. J. Infante has his degree in Informatics from the Universidad de Pinar del Rio. He is currently making his Ph.D studies in the Universidad Politécnica de Valencia.

Julian Triana

Universidad Pinar del Río "Hermanos Saíz Montes de Oca", Martí 270, 20100, Pinar del Río, Cuba

Email: jtriana@vrect.upr.edu.cu

J. Triana has his degree in Biochemistry from the Universidad de La Habana. He is currently making his Ph.D studies in the Universidad Politécnica de Valencia.

Javier Urchueguía

Instituto Universitario de Matemática Pura y Aplicada, Universidad Politécnica de Valencia

jfurchueguia@fis.upv.es

J. Urchueguía received the Licenciado degree in Theoretical Physics from the Universitat de València (UV), Valencia, Spain in 1987, and the Ph.D. degree in Physics from the Universidad Politécnica de Valencia (UPV), Valencia, Spain, in 1993.

Pedro Fernández de Córdoba

Instituto Universitario de Matemática Pura y Aplicada, Universidad Politécnica de Valencia

pfernandez@mat.upv.es

P. F. de Córdoba received the Licenciado degree, and the M.S. and the Ph.D. degrees in Theoretical Physics from the Universitat de València (UV), Valencia, Spain in 1988, 1990, and 1992, respectively. He also received the Ph.D. degree in Mathematics from the Universidad Politécnica de Valencia (UPV), Valencia, Spain, in 1997.

Acknowledgement

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 308518 (CyanoFactory).

Appendix: Python Scripts

We provide two python scripts that exemplify some uses of PyNetMet.

In the first file (examples.py in Listing 1) the examples from the PyNetMet documentation page are found.

The second file (real_vs_rand.py in Listing 2) has the lines of code that were written in order to collect the data for tables 1 and 2.

```
1  ############################
2  #
3  # Enzyme examples
4  #
5  ############################
6  from PyNetMet.enzyme import *
7
8  enz1 = Enzyme("reac1␣:␣A␣+␣2␣B␣->␣C␣+␣D")
9  print enz1
10
11 enz2 = Enzyme("reac2␣:␣A␣+␣2␣B␣->␣C␣D")
12 print enz2
13
14 enz3 = Enzyme("reac3␣:␣A␣+␣2␣B␣->␣C+␣D")
15 print enz3
16
17 enz4 = Enzyme("reac4␣:␣A␣+␣2␣B␣->␣C+␣+␣
       ↪ D")
18 print enz4
19
20
21
22
23 ############################
24 #
25 # Metabolism & Network examples
26 #
27 ############################
28 from PyNetMet.metabolism import *
29
30 syn=Metabolism("iSyn811.txt")
31 cbe=Metabolism("iCM925.xml",
       ↪ filetype="sbml")
32 ak=Metabolism("iAK692.xml",
       ↪ filetype="sbml")
33
34 print cbe
35 print cbe.pathnames
36 print len(cbe.pathways[1])
37
38
39 #
40 # Generation of clustering plots
41 #
42 syn.net.plot_matr(syn.net.nCCs,
       ↪ range(syn.nmets),
       ↪ output="plot_syn1.jpg")
43 syn.net.kruskal(syn.net.nCCs,
       ↪ minimo=False)
44 syn.net.plot_matr(syn.net.nCCs,
       ↪ syn.net.krusk_ord,
       ↪ output="plot_syn2.jpg")
45 syn.net.plot_nCCs(output=
       ↪ "plot_syn3.jpg")
46 cbe.net.plot_matr(cbe.net.nCCs,
       ↪ range(cbe.nmets),
       ↪ output="plot_cbe1.jpg")
47 cbe.net.kruskal(cbe.net.nCCs,
       ↪ minimo=False)
48 cbe.net.plot_matr(cbe.net.nCCs,
       ↪ cbe.net.krusk_ord,
       ↪ output="plot_cbe2.jpg")
49 cbe.net.plot_nCCs(output=
       ↪ "plot_cbe3.jpg")
50 ak.net.plot_matr(ak.net.nCCs,
       ↪ range(ak.nmets),
       ↪ output="plot_ak1.jpg")
51 ak.net.kruskal(ak.net.nCCs, minimo=False)
52 ak.net.plot_matr(ak.net.nCCs,
       ↪ ak.net.krusk_ord,
       ↪ output="plot_ak2.jpg")
53 ak.net.plot_nCCs(output="plot_ak3.jpg")
54
55
56
57
58 #
59 # Average clustering
60 #
61 print sum(syn.net.Cis)/syn.net.nnodes
62 print sum(cbe.net.Cis)/cbe.net.nnodes
63 print sum(ak.net.Cis)/ak.net.nnodes
64
65
66
67 #
68 # Disconnected components
69 #
70 syn.net.components()
```


Concept Press Ltd.

```
print [len(ele) for ele in
    ↪ syn.net.disc_comps]
cbe.net.components()
print [len(ele) for ele in
    ↪ cbe.net.disc_comps]
ak.net.components()
print [len(ele) for ele in
    ↪ ak.net.disc_comps]


#
# Distance between nodes
#
import sys # For setting a new recursion
    ↪ limit for recursive functions
sys.setrecursionlimit(10000)
[Mreac,names] = syn.M_matrix_reacs()
net = Network(Mreac, names)
iglu = names.index("alpha-D-glucose")
ipyr = names.index("pyruvate")
[paths, dists] =
    ↪ net.calc_dist_wp(net.linksout,
    ↪ iglu)
print [names[ii] for ii in paths[ipyr]]
print paths[ipyr]


#
# Number of disconnected pairs and
    ↪ average distance
#
dists.count("X")
ndist = filter(lambda x:x != "X", dists)
print 1.*sum(ndist)/len(ndist)
print max(ndist)
dists.count(20)
names[dists.index(20)]


############################
#
# FBA Examples
#
############################

from PyNetMet.fba import *
fba_cbe=FBA(cbe)
print fba_cbe


#
# Optimal growth of syn and optimal
    ↪ production of H2
#
fba_syn1=FBA(syn)
print fba_syn1.Z
gro = fba_syn1.Z
syn.constr[syn.dic_enzs["_Growth"]] =
    ↪ (0.95*gro, 0.95*gro)
syn.obj = [("_H2","1")]
fba_syn2 = FBA(syn)
print >>open("diff.txt","w"),
    ↪ fba_syn1.__sub__(fba_syn2,
    ↪ keyz=lambda x:x[3])

# is this gene essential?
print syn.enzymes[926]
print fba_syn1.essential(926)

# Essential genes
print sum([fba_cbe.essential(ii)[0] for
    ↪ ii in xrange(cbe.nreacs)])
print sum([fba_syn1.essential(ii)[0] for
    ↪ ii in xrange(syn.nreacs)])
fba_ak = FBA(ak)
print ak.enzymes[746]
print ak.enzymes[748]
print sum([fba_ak.essential(ii)[0] for
    ↪ ii in xrange(ak.nreacs)])


# shadow flux and max_min methods
print fba_syn1.shadow(926)
print fba_syn1.shadow(926, relat=False)
print fba_syn1.max_min(926, fixobj=0.5)
print fba_syn1.max_min(926, fixobj=0.6)
```

Listing 1: examples.py

```
######
###
### Network analysis
###
######

```

```
if True:
    sys.setrecursionlimit(10000)
    syn = Metabolism("iSyn811.txt")
    cbe = Metabolism("iCM925.xml",
        ↪ filetype="sbml")
    ak = Metabolism("iAK692.xml",
        ↪ filetype="sbml")
    models = [syn, cbe, ak]
    for model in models:
        print "_______________"
        print model.file_name
        print "---"
        print ""
        print "Cbar = %f +- %f"%tuple(
            ↪ stats(model.net.Cis))
        model.net.components()
        print [len(ele) for ele in
            ↪ model.net.disc_comps]
        M = model.M_matrix(True)
        net = Network(M)
        dists = net.calc_all_dists(
            ↪ net.neigbs)
        NN = len(dists)
        meds1 = [dists[ii][jj] for ii in
            ↪ xrange(NN) for jj in
            ↪ xrange(ii+1, NN) if
            ↪ dists[ii][jj] != "X"]
        Xs = sum([ele.count("X") for ele
            ↪ in dists])
        print "lbar = %f +- %f"%tuple(stats(meds1))
        print "Xs = %i"%Xs
        fba = FBA(model)
        ess = sum([fba.essential(ii)[0]
            ↪ for ii in
            ↪ xrange(model.nreacs)])
        print "essentia = %i"%ess



######
###
### Random Networks
###
######
from random import random as rand

Nr = 100
Nnodes = 989
Nlinks = 2972

Cis = []
dev_C = []
lls = []
dev_ll = []

for ir in xrange(Nr):
    # generates random M matrix
    M = [[0 for ii in xrange(Nnodes)]
        ↪ for jj in xrange(Nnodes)]
    for il in xrange(Nlinks):
        # choses two nodes at random
        ii = int(Nnodes*rand())
        jj = int(Nnodes*rand())
        while ii == jj or M[ii][jj] == 1:
            ii = int(Nnodes*rand())
            jj = int(Nnodes*rand())
        # connects the two nodes
        M[ii][jj] = 1
        M[jj][ii] = 1
    print "Network %i created!"%ir
    # Creates the network
    net = Network(M)
    dists = net.calc_all_dists(
        ↪ net.neigbs)
    meds1 = [dists[ii][jj] for ii in
        ↪ xrange(Nnodes) for jj in
        ↪ xrange(ii+1, Nnodes) if
        ↪ dists[ii][jj] != "X"]
    [cb, sc] = stats(net.Cis)
    [lb, sl] = stats(meds1)
    Cis.append(cb)
    dev_C.append(sc)
    lls.append(lb)
    dev_ll.append(sl)
```

Listing 2: rand_vs_real.py

References

1. *L. A. Amaral, A. Scala, M. Barthelemy, and H. E. Stanley. Classes of small-world networks.* Proc. Natl. Acad. Sci. U.S.A., *97(21):11149–11152, Oct 2000.*

2. *C. L. Barrett, T. Y. Kim, and H. U. et al. Kim. Systems biology as a foundation for genome-scale synthetic biology.* Curr. Opin. Biotechnol., *17(5):488–492, Oct 2006.*

Concept Press Ltd.

3. *M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks.* International AAAI Conference on Weblogs and Social Media, *2009.*

4. *A. Bauer-Mehren. Integration of genomic information with biological networks using Cytoscape.* Methods Mol. Biol., *1021:37–61, 2013.*

5. *P. J. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, et al. Biopython: freely available python tools for computational molecular biology and bioinformatics.* Bioinformatics, *25(11):1422–1423, 2009.*

6. *M. Cvijovic, R. Olivares-Hernandez, and R. et al. Agren. BioMet Toolbox: genome-wide analysis of metabolism.* Nucleic Acids Res., *38(Web Server issue):W144–149, Jul 2010.*

7. *J. S. Edwards, R. U. Ibarra, and B. O. Palsson. In silico predictions of Escherichia coli metabolic capabilities are consistent with experimental data.* Nat. Biotechnol., *19(2):125–130, Feb 2001.*

8. *B. C. Faircloth. msatcommander: detection of microsatellite repeat arrays and automated, locus-specific primer design.* Molecular Ecology Resources, *8(1):92–94, 2008.*

9. *D. Gamermann, A. Montagud, and A. et al. Conejero. Phylogenetic tree reconstruction from genome-scale metabolic models.* J. Comput. Biol., *21(0):1–12, Mar 2014.*

10. *M. Hucka, A. Finney, and H. M. et al. Sauro. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models.* Bioinformatics, *19(4):524–531, Mar 2003.*

11. *H. Jeong, B. Tombor, and R. et al. Albert. The large-scale organization of metabolic networks.* Nature, *407(6804):651–654, Oct 2000.*

12. *M. Jünger.* Graph drawing software. *Springer, Berlin New York, 2004.*

13. *A. Klanchui, C. Khannapho, and A. et al. Phodee. iAK692: a genome-scale metabolic model of Spirulina platensis C1.* BMC Syst Biol, *6:71, 2012.*

14. *C. F. Lopez, J. L. Muhlich, J. A. Bachman, and P. K. Sorger. Programming biological models in python using pysb.* Molecular systems biology, *9(1), 2013.*

15. *C. B. Milne, J. A. Eddy, and R. et al. Raju. Metabolic network reconstruction and genome-scale model of butanol-producing strain Clostridium beijerinckii NCIMB 8052.* BMC Syst Biol, *5:130, 2011.*

16. *A. Montagud, E. Navarro, and P. et al. Fernandez de Cordoba. Reconstruction and analysis of genome-scale metabolic model of a photosynthetic bacterium.* BMC Syst Biol, *4:156, 2010.*

17. *M. A. Oberhardt, B. O. Palsson, and J. A. Papin. Applications of genome-scale metabolic reconstructions.* Mol. Syst. Biol., *5:320, 2009.*

18. *B. G. Olivier, J. M. Rohwer, and J.-H. S. Hofmeyr. Modelling cellular systems with PySCeS.* Bioinformatics, *21(4):560–561, 2005.*

19. *K. R. Patil, M. Akesson, and J. Nielsen. Use of genome-scale microbial models for metabolic engineering.* Curr. Opin. Biotechnol., *15(1):64–69, Feb 2004.*

20. *E. Ravasz, A. L. Somera, and D. A. et al. Mongru. Hierarchical organization of modularity in metabolic networks.* Science, *297(5586):1551–1555, Aug 2002.*

21. *A. W. Rives and T. Galitski. Modular organization of cellular networks.* Proc. Natl. Acad. Sci. U.S.A., *100(3):1128–1133, Feb 2003.*

22. *I. Rocha, P. Maia, and P. et al. Evangelista. OptFlux: an open-source software platform for in silico metabolic engineering.* BMC Syst Biol, *4:45, 2010.*

23. *J. L. Rybarczyk-Filho, M. A. Castro, and R. J. et al. Dalmolin. Towards a genome-wide transcriptogram: the Saccharomyces cerevisiae case.* Nucleic Acids Res., *39(8):3005–3016, Apr 2011.*

24. *J. Schellenberger, R. Que, R. M. Fleming, I. Thiele, J. D. Orth, A. M. Feist, D. C. Zielinski, A. Bordbar, N. E. Lewis, S. Rahmanian, J. Kang, D. R. Hyduke, and B. O. Palsson. Quantitative prediction of cellular metabolism with constraint-based models: the COBRA Toolbox v2.0.* Nat Protoc, *6(9):1290–1307, Sep 2011.*

25. *A. Varma and B. Palsson. Metabolic capabilities of Escherichia coli: II. Optimal growth patterns.* J. Theor. Biol., *165:503–522, 1993.*

Concept Press Ltd.